



開發人員指南

Amazon DynamoDB



API 版本 2012-08-10

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

Amazon DynamoDB: 開發人員指南

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

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

什麼是 Amazon DynamoDB ?	1
高度可用性與耐用性	1
DynamoDB 入門	1
DynamoDB 教學課程	2
運作方式	2
速查表	3
核心元件	7
DynamoDB API	16
支援的資料類型和命名規則	20
資料表類別	25
分割區與資料分佈	26
從 SQL 到 NoSQL	29
關聯式或 NoSQL ?	30
資料庫的特性	32
建立資料表	35
取得資料表的資訊	37
將資料寫入資料表	39
從資料表讀取資料	43
管理索引	51
修改資料表中的資料	56
從資料表刪除資料	60
移除資料表	62
其他 Amazon DynamoDB 資源	63
寫程式和視化工具	63
規範性指引	64
知識中心	65
部落格文章、儲存庫和指南	66
資料建模和設計模式	66
培訓課程	67
讀取和寫入	68
讀取一致性	68
讀取和寫入作業	69
讀取操作消耗	69
寫操作消耗	70

輸送量容量	72
DynamoDB 式概觀	72
隨需模式	72
佈建模式	73
隨需容量模式	73
讀取請求單位與寫入請求單位	74
初始輸送量和擴展屬性	74
隨選表格的最大輸送量	75
預熱的桌子	77
佈建容量模式	77
讀取和寫入容量單位	78
選擇初始輸送量設定	79
DynamoDB 自動擴展	80
使用自動擴展管理輸送容量	80
預留容量	101
突發和自適應能力	101
高載容量	102
調適型容量	102
設定 DynamoDB	104
設定 DynamoDB Local (可下載版本)	104
部署	105
使用須知	111
版本歷史記錄	115
DynamoDB 本機遙測功能	119
設定 DynamoDB (Web 服務)	121
註冊成為 AWS	121
授予程式設計存取權	122
設定您的憑證	123
與其他 DynamoDB 服務整合	123
存取 DynamoDB	124
使用主控台	124
使用 AWS CLI	125
下載和設定 AWS CLI	125
搭配 DynamoDB 使用 AWS CLI	126
AWS CLI 搭配 DynamoDB 機	127
使用 API	128

使用 NoSQL Workbench	128
IP 地址範圍	129
DynamoDB 入門	130
基本概念	130
必要條件	130
步驟 1：建立資料表	131
步驟 2：寫入資料	135
步驟 3：讀取資料	139
步驟 4：更新資料	142
步驟 5：查詢資料	145
步驟 6：建立全域次要索引	148
步驟 7：查詢全域次要索引	151
步驟 8：(選用) 清除	154
後續步驟	155
開始使用 DynamoDB 和開發套件 AWS	156
建立資料表	156
使用開發套件建立 DynamoDB 資料表 AWS	156
寫入項目	200
使用開發套件將項目寫入 DynamoDB 資料表 AWS	201
讀取項目	225
使用開發套件從 DynamoDB 資料表讀取項目 AWS	225
更新項目	248
使用 SDK 更新 DynamoDB 資料表中的項目 AWS	248
刪除項目	274
使用開發套件刪除 DynamoDB 資料表中的項目 AWS	274
查詢資料表	297
使用開發套件查詢 DynamoDB 資料表 AWS	297
掃描資料表	329
使用開發套件掃描 DynamoDB 資料表 AWS	297
使用 AWS 軟體開發套件	354
使用 DynamoDB 編寫程式	356
適用於 AWS SDK 的 SDK DynamoDB 援概觀	356
程式設計界面	359
低階 API	365
錯誤處理	370
更高階程式設計界面	377

Java 1.x : DynamoDBMapper	378
Java 2.x : DynamoDB 增強型用戶端	445
.NET : 文件模型	446
.NET : 物件持久性模型	477
執行程式碼範例	515
載入範例資料	516
Java 程式碼範例	517
.NET 程式碼範例	519
使用 Python 編程	522
關於博托	523
博托文檔	523
用戶端和資源層	524
使用批次寫入器	527
其他程式碼範例	527
會話和線程安全	528
Config	528
錯誤處理	532
日誌	534
事件掛鉤	535
分頁和分頁器	536
等待程式	538
編程與 JavaScript	539
關於 AWS SDK for JavaScript	539
AWS SDK for JavaScript V3	539
JavaScript 文件	540
抽象層	540
马歇尔函数	542
讀取項目	543
條件式寫入	545
分頁	545
Config	547
等待程式	550
錯誤處理	550
日誌	552
考量事項	553
使用 Java 2.x 進行編程	554

關於 AWS SDK for Java 2.x	554
開始使用	555
適用於 Java 2.x 文件的開發套件	564
支援的介面	564
其他程式碼範例	578
同步和異步編程	578
HTTP 用戶端	579
Config	580
錯誤處理	586
AWS 請求識別碼	587
日誌	587
分頁	589
資料類別註解	591
使用 DynamoDB	592
處理資料表	592
資料表上的基本操作	593
選擇資料表類別時的注意事項	601
項目大小和格式	602
標記 資源	603
使用資料表：Java	608
使用資料表：.NET	615
使用全域資料表	624
跨區域使用全域資料表無縫複寫資料	625
為您的全域資料表提供安全性和存取權 AWS KMS	626
運作方式	626
最佳實務和要求	630
教學課程：建立全域資料表	633
監控全域資料表	638
對全域資料表使用 IAM	639
判斷版本	642
升級全域資料表	644
使用讀取和寫入操作	652
DynamoDB API	652
PartiQL 查詢語言	839
使用索引	883
全域次要索引	887

本機次要索引	941
搭配交易使用	992
運作方式	992
將 IAM 搭配交易使用	1000
範例程式碼	1003
使用串流	1006
選項	1007
使用 Kinesis Data Streams	1009
使用 DynamoDB Streams	1025
使用隨需備份與還原功能	1081
使用 AWS 備份	1082
使用 DynamoDB 備份	1091
使用 point-in-time 恢復	1109
運作方式	1110
開始之前	1112
還原 資料表至某個時間點	1113
使用 DAX 的記憶體內加速	1119
DAX 使用案例	1120
DAX 使用須知	1121
運作方式	1121
DAX 如何處理請求	1123
項目快取	1124
查詢快取	1125
DAX 叢集元件	1126
節點	1126
叢集	1127
區域與可用區域	1128
參數群組	1128
安全群組	1128
叢集 ARN	1129
叢集端點	1129
節點端點	1129
子網路群組	1129
事件	1130
Maintenance window (維護時段)	1130
建立 DAX 叢集	1131

為 DAX 建立可存取 DynamoDB 的 IAM 服務角色	1132
使用 AWS CLI	1133
使用主控台	1139
一致性模式	1143
DAX 叢集節點之間的一致性	1144
DAX 項目快取行為	1144
DAX 查詢快取行為	1147
強烈一致和交易讀取	1148
負快取	1148
寫入策略	1148
使用 DAX 用戶端開發	1151
教學課程：執行範例應用程式	1152
修改現有應用程式以使用 DAX	1199
管理 DAX 叢集	1200
管理 DAX 叢集的 IAM 許可	1200
擴展 DAX 叢集	1203
自訂 DAX 叢集設定	1204
配置 TTL 設定	1205
DAX 的標記支援	1206
AWS CloudTrail 整合	1208
刪除 DAX 叢集	1208
監控 DAX	1208
監控工具	1209
使用監控 CloudWatch	1210
使用 AWS CloudTrail 記錄 DAX 操作	1230
DAX T3/T2 爆量執行個體	1230
DAX T2 執行個體系列	1230
DAX T3 執行個體系列	1231
DAX 存取控制	1231
DAX 的 IAM 服務角色	1232
允許 DAX 叢集存取的 IAM 政策	1234
使用案例：存取 DynamoDB 和 DAX	1235
存取 DynamoDB，但不可使用 DAX 存取	1236
存取 DynamoDB 和 DAX	1238
透過 DAX 存取 DynamoDB，但不直接存取 DynamoDB	1243
DAX 靜態加密	1245

使用 AWS Management Console 啟用靜態加密	1247
DAX 傳輸中加密	1248
使用 DAX 的服務連結角色	1249
DAX 的服務連結角色許可	1249
建立 DAX 的服務連結角色	1251
編輯 DAX 的服務連結角色	1251
刪除 DAX 的服務連結角色	1251
跨多 AWS 帳戶存取 DAX	1253
設定 IAM	1253
設定 VPC	1256
修改 DAX 用戶端以允許跨帳戶存取權	1258
DAX 叢集調整大小指南	1262
概觀	1263
預估流量	1263
負載測試	1264
最佳實務	1265
API 參考	1265
建立資料模型	1266
資料建模基礎	1267
單一資料表設計	1267
多資料表設計	1269
資料建模建置區塊	1271
複合排序索引鍵	1272
多租戶	1273
稀鬆索引	1274
生存時間	1275
封存生存時間	1276
垂直分割	1277
寫入碎片	1279
資料建模結構描述設計套件	1281
必要條件	1281
社交網路	1282
遊戲設定檔	1291
投訴管理系統	1299
週期性付款	1316
裝置狀態更新	1321

線上商店	1334
移轉至 DynamoDB	1358
遷移的原因	1358
移轉時的考量	1359
運作方式	1361
遷移工具	1361
選擇移轉策略	1362
離線遷移	1364
混合式移轉	1365
線上-移轉每個表格 1:1	1367
線上-使用自訂臨時資料表移轉	1368
NoSQL Workbench	1371
下載	1372
安裝	1373
資料模型建立工具	1377
建立新模型	1377
匯入現有的模型	1384
匯出模型	1387
編輯現有的模型	1389
資料視覺化工具	1393
新增範例資料	1393
從 CSV 匯入	1396
面向	1397
彙總檢視	1400
遞交資料模型	1401
操作建置器	1404
連線至資料集	1404
建置操作	1406
複製表格	1416
匯出至 CSV	1417
範例資料模型	1418
員工資料模型	1418
開發論壇資料模型	1419
音樂資料庫資料模型	1419
滑雪渡假村資料模型	1419
信用卡優惠資料模型	1420

書籤資料模型	1420
版本歷史記錄	1421
程式碼範例	1426
動作	1434
BatchExecuteStatement	1435
BatchGetItem	1461
BatchWriteItem	1484
CreateTable	1513
DeleteItem	1557
DeleteTable	1580
DescribeTable	1596
ExecuteStatement	1611
GetItem	1633
ListTables	1655
PutItem	1672
Query	1697
Scan	1729
UpdateItem	1755
UpdateTable	1781
案例	1791
使用 DAX 加速讀取	1792
開始使用資料表、項目和查詢	1800
使用多批 PartiQL 陳述式查詢資料表	1950
使用 PartiQL 查詢資料表	2009
使用文件模型	2062
使用高階物件持久性模型	2078
無伺服器範例	2087
從 DynamoDB 觸發程序叫用 Lambda 函數	2088
使用 DynamoDB 觸發程序報告 Lambda 函數的批次項目失敗	2096
跨服務範例	2107
建置應用程式以將資料提交至 DynamoDB 資料表	2108
建立 REST API 以追蹤 COVID-19 資料	2109
建立傳訊應用程式	2110
建立無伺服器應用程式來管理相片	2111
建立 Web 應用程式以追蹤 DynamoDB 資料	2114
建立 websocket 聊天應用程式	2117

偵測映像中的 PPE	2117
從瀏覽器調用 Lambda 函數	2118
監 DynamoDB 效能	2119
儲存 EXIF 和其他映像資訊	2120
使用 API Gateway 來調用 Lambda 函數	2120
使用 Step Functions 呼叫 Lambda 函數	2122
使用排程事件來呼叫 Lambda 函數	2123
安全	2125
AWS 受管政策	2125
AWS 受管政策	2126
AmazonDynamo資料庫 ReadOnlyAccess	2126
受管政策的 DynamoDB 更新 AWS	2127
資源型政策	2128
建立資料表	2129
附加資源型政策	2135
將原則附加至串流	2139
移除資源型政策	2142
跨帳戶存取權	2143
封鎖公開存取	2144
API 操作	2146
IAM 授權	2151
範例	2151
考量事項	2157
最佳實務	2158
資料保護	2159
靜態加密	2159
DAX 中的資料保護	2183
網際網路流量隱私權	2183
IAM	2185
身分和存取權管理	2185
使用條件	2214
DAX 中的身分和存取管理	2235
法規遵循驗證	2236
復原能力	2237
基礎架構安全	2237
使用 VPC 端點	2238

AWS PrivateLink 適用於動態支援	2247
Amazon VPC 端點的類型	2248
使用 Amazon DynamoDB 時 AWS PrivateLink 的注意事項	2248
建立 Amazon VPC 端點	2249
存取 Amazon DynamoDB 介面端點	2249
從 DynamoDB 介面端點存取表格並控制 API 作業	2249
更新內部部署 DNS 組態	2251
建立 Amazon VPC 端點政策	2253
組態與漏洞分析	2254
安全最佳實務	2255
預防性安全最佳實務	2255
偵測性安全最佳實務	2257
監控和記錄	2260
監控計畫	2260
效能基準	2260
整合服務	2261
自動化監控工具	2261
監控指標	2261
如何使用 DynamoDB 指標？	2262
在 CloudWatch 主控台中檢視指標	2263
檢視指標 AWS CLI	2263
指標與維度	2264
建立 CloudWatch 鬧鐘	2288
記錄作業	2291
DynamoDB 資訊 CloudTrail	2291
了解資料 DynamoDB 日誌檔案項目	2295
Contributor Insights	2314
運作方式	2314
開始使用	2320
使用 IAM	2325
最佳實務	2330
NoSQL 設計	2330
NoSQL 與 RDBMS 對比	2331
兩個主要概念	2331
一般方法	2331
NoSQL Workbench	2332

刪除保護	2333
DynamoDB Well-Architected Lens	2333
成本最佳化	2333
進行 Amazon DynamoDB Well-Architected Lens 檢閱	2377
Amazon DynamoDB Well-Architected Lens 的支柱	2377
分割區索引鍵設計	2379
分佈工作負載	2379
寫入碎片	2380
有效率地上傳資料	2382
排序索引鍵設計	2383
版本控制	2384
次要索引	2385
一般準則	2385
稀鬆索引	2387
聚合	2389
GSI 過載	2390
GSI 碎片	2392
建立一個複本	2393
大型項目	2393
壓縮	2394
垂直分割	2394
使用 Amazon S3	2394
時間序列資料	2395
時間序列資料的設計模式	2395
時間序列資料表範例	2396
M any-to-many 關係	2396
相鄰清單	2397
具體化圖形	2398
混合式 DynamoDB : RDBMS	2402
未遷移	2403
混合系統實作	2403
關聯式模型	2404
傳統關聯式資料庫模型	2404
DynamoDB 如何免除 JOIN 操作的需求	2406
DynamoDB 交易如何消除寫入程序的負荷	2406
首要步驟	2408

範例	2409
查詢及掃描	2412
掃描效能	2412
避免尖峰	2413
平行掃描	2415
資料表設計	2416
全域資料表設計	2416
全域資料表設計	2417
關鍵事實	2417
使用案例	2418
寫入模式	2419
要求路由	2426
疏散區域	2434
全域資料表的輸送容量	2436
全域資料表的檢查清單和常見問答集	2437
控制平台	2442
帳單和用量報告	2443
吞吐量容量	2446
串流	2449
儲存	2449
備份與恢復	2450
資料傳輸	2453
CloudWatch	2453
DAX	2454
切換容量模式	2455
佈建模式至隨選模式	2456
按需模式到佈建模式	2457
將 DynamoDB 與其他 AWS 服務搭配使用	2459
與 Amazon Cognito 整合	2459
與 Amazon Redshift 整合	2461
與 Amazon EMR 整合	2463
概要	2463
教學課程：使用 Amazon DynamoDB 和 Apache Hive	2464
在 Hive 中建立外部資料表	2472
處理 HiveQL 陳述式	2476
查詢 DynamoDB 中的資料	2477

在 Amazon DynamoDB 之中複製和貼入資料	2479
效能調校	2492
與 S3 整合	2497
從 Amazon S3 匯入	2497
匯出至 Amazon S3	2517
與 Amazon 服 OpenSearch 務集成	2540
運作方式	2540
建立整合	2541
後續步驟	2541
處理突破性變更	2542
整合最佳做法	2545
建立快照	2545
變更資料擷取	2545
零 ETL 與服務整合 OpenSearch	2546
配額和限制	2549
讀取/寫入容量模式與輸送量	2549
容量單位大小 (已佈建的資料表)	2550
請求單位大小 (隨需資料表)	2550
輸送量預設配額	2550
提高或降低輸送量 (已佈建的資料表)	2552
預留容量	101
匯入配額	2553
Contributor Insights	2553
資料表	2553
資料表大小	2553
每個帳戶每個區域的資料表數目上限	2553
全域資料表	2554
次要索引	2555
每份資料表的次要索引	2555
每份資料表的預估次要索引屬性	2555
分割區索引鍵和排序索引鍵	2555
分割區索引鍵長度	2555
分割區索引鍵值	2556
排序索引鍵長度	2556
排序索引鍵值	2556
命名規則	2556

資料表名稱和次要索引名稱	2556
屬性名稱	2556
資料類型	2557
字串	2557
Number	2557
二進位	2557
項目	2558
項目大小	2558
具有本機次要索引之資料表的項目大小	2558
Attributes	2558
每個項目的屬性名稱/值對	2558
清單、映射或集合的值數目	2558
屬性值	2558
巢狀屬性深度	2558
表達式參數	2559
長度	2559
運算子和運算元	2559
保留字	2559
DynamoDB 交易	2559
DynamoDB Streams	2560
DynamoDB Streams 中的碎片同時讀取	2560
啟用 DynamoDB Streams 之資料表的寫入容量上限	2560
DynamoDB Accelerator (DAX)	2561
AWS 區域可用性	2561
節點	2561
參數群組	2561
子網路群組	2561
API 特定限制	2561
DynamoDB 靜態加密	2564
資料表匯出至 Amazon S3	2564
備份和還原	2564
API 參考	2565
疑難排解	2566
Latency (延遲)	2566
限流	2567
針對限流問題進行疑難排解	2568

使用 CloudWatch 指標	2569
附錄	2571
對 SSL/TLS 連線建立問題進行疑難排解	2571
測試您的應用程式或服務	2571
測試您的用戶端瀏覽器	2572
更新您的軟體應用程式用戶端	2572
更新您的用戶端瀏覽器	2572
手動更新您的憑證套件	2573
監控工具	2573
自動化工具	2573
手動工具	2574
範例資料表和資料	2575
範例資料檔案	2576
建立範例資料表和上傳資料	2589
建立範例資料表和上傳資料 - Java	2589
建立範例資料表和上傳資料 - .NET	2599
使用 AWS SDK for Python (Boto3) 的範例應用程式	2611
步驟 1：在本機上部署及測試	2612
步驟 2：檢查資料模型和實作詳細資訊	2617
步驟 3：在生產環境中部署	2625
步驟 4：清除資源	2634
與整合 AWS Data Pipeline	2634
匯出和匯入資料的必要條件	2637
將資料從 DynamoDB 匯出至 Amazon S3	2645
從 Amazon S3 匯入資料至 DynamoDB	2646
故障診斷	2648
預先定義的 DynamoDB AWS Data Pipeline 和	2648
適用於 Titan 的 Amazon DynamoDB 儲存後端	2649
DynamoDB 中的保留字	2649
舊式條件式參數	2663
AttributesToGet	2664
AttributeUpdates	2665
ConditionalOperator	2667
預期	2668
KeyConditions	2673
QueryFilter	2676

ScanFilter	2678
使用舊式參數撰寫條件	2679
舊版低階 API (2011-12-05)	2687
BatchGetItem	2688
BatchWriteItem	2695
CreateTable	2701
DeleteItem	2708
DeleteTable	2714
DescribeTables	2718
GetItem	2722
ListTables	2726
PutItem	2728
Query	2734
Scan	2747
UpdateItem	2763
UpdateTable	2771
AWS 適用於 Java 1.x 的開發套件範例	2775
DAX 與 Java 開發套件第 1 版	2776
修改適用於 Java 1.x 的開發套件的現有應用程式來使用 DAX	2787
使用適用於 Java 1.x 的開發套件查詢全域次要索引	2792
文件歷史紀錄	2797
舊版更新	2813
舊版功能	2835
全域表格版本 (舊版)	2835
運作方式	2835
最佳實務和要求	2840
建立全域資料表	2843
監控全域資料表	2847
對全域資料表使用 IAM	2848
.....	mmdcccli

什麼是 Amazon DynamoDB ？

Amazon DynamoDB 是一項完全受管的 NoSQL 資料庫服務，可提供快速且可預期的效能及無縫的可擴展性。DynamoDB 是全受管的 NoSQL 資料庫服務，可讓您卸下操作及擴展分散式資料庫的管理負擔，不再需要煩惱硬體佈建、設定和組態、複寫、軟體修補或叢集擴展。DynamoDB 還提供靜態加密，解決了保護敏感資料所涉及的操作負擔和複雜性。如需詳細資訊，請參閱 [DynamoDB 靜態加密](#)。

您可以使用 DynamoDB 建立資料庫資料表，可存放和擷取任意數量的資料，並為任何層級的請求流量提供服務。您可以擴展或縮減資料表的輸送容量，且無須停機或降低效能。您可以使用 AWS Management Console 來監控資源使用率和效能指標。

DynamoDB 提供隨需備份功能。它可讓您針對法規合規需求，建立資料表的完整備份來長期保留與封存。如需詳細資訊，請參閱 [使用 DynamoDB 的隨需備份與還原](#)。

您可以建立隨需備份，並為 Amazon DynamoDB 表格啟用 point-in-time 復原功能。Point-in-time 復原可協助保護資料表免於意外寫入或刪除作業。透過 point-in-time 復原，您可以將資料表還原到過去 35 天內的任何時間點。如需詳細資訊，請參閱 [Point-in-time 恢復：它是如何工作的](#)。

DynamoDB 可讓您自動刪除資料表中的過期項目，協助您降低儲存體使用量以及存放不再相關的資料的成本。如需詳細資訊，請參閱 [生存時間 \(TTL\)](#)。

高度可用性與耐用性

DynamoDB 會自動將資料表的資料與傳輸流分散到足夠數量的伺服器上，以處理您的輸送量和儲存需求，同時保持快速且一致的效能。您的所有資料都儲存在固態硬碟 (SSD) 上，並自動複製到同一 AWS 區域中的多個可用區域，提供內建的高可用性和資料耐用性。您可以使用全域表，讓各區域的 DynamoDB 表保持同 AWS 步。如需詳細資訊，請參閱 [全域資料表：DynamoDB 的多區域複寫](#)。

DynamoDB 入門

建議您一開始先閱讀下列各節：

- [Amazon DynamoDB：運作方式](#)：學習基本的 DynamoDB 概念。
- [設定 DynamoDB](#)：了解如何設定 DynamoDB (可下載版本或 Web 服務)。
- [存取 DynamoDB](#)— 瞭解如何使用主控台或 API 存取 DynamoDB。AWS CLI

然後，您可選擇以兩種方式快速開始使用 DynamoDB：

- [DynamoDB 入門](#)
- [開始使用 DynamoDB 和開發套件 AWS](#)

若要進一步了解應用程式開發，請參閱以下內容：

- [使用 DynamoDB 和開發套件進程式設計 AWS](#)
- [使用資料表、項目、查詢、掃描和索引](#)

若要快速找到大幅提升效能並降低輸送量成本的建議，請參閱以下文件：[使用 DynamoDB 進行設計和架構的最佳實務](#)。若要了解如何標記 DynamoDB 資源，請參閱 [將標籤新增至資源](#)。

如需最佳實務、操作指南和工具，請參閱 [Amazon DynamoDB 資源](#)。

您可以使用 AWS Database Migration Service (AWS DMS) 將資料從關聯式資料庫或 MongoDB 移轉至 DynamoDB 表格。如需詳細資訊，請參閱《[AWS Database Migration Service 使用者指南](#)》。

若要了解如何使用 MongoDB 作為遷移來源，請參閱[使用 MongoDB 作為 AWS Database Migration Service 的來源](#)。若要了解如何使用 DynamoDB 做為移轉目標，請參閱[使用 Amazon DynamoDB 資料庫做為目標](#)。AWS Database Migration Service

DynamoDB 教學課程

下列教學課程提供完整的 end-to-end 程序，讓您熟悉 DynamoDB。這些教學課程可透過免費方案完成，AWS 並提供您使用 DynamoDB 的實務體驗。

- [使用 NoSQL 鍵 - 值資料存放區建置應用程式](#)
- [使用 Amazon DynamoDB 建立和查詢 NoSQL 資料表](#)

Amazon DynamoDB：運作方式

下列各節提供 Amazon DynamoDB 服務元件及其互動方式概觀。

閱讀此簡介之後，請嘗試進行「[在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#)」一節，該節將引導您完成建立範例資料表、上傳資料及執行一些基本資料庫操作的程序。

如需含有範本程式碼的特定語言教學，請參閱「[開始使用 DynamoDB 和開發套件 AWS](#)」。

主題

- [DynamoDB 速查表](#)
- [Amazon DynamoDB 的核心元件](#)
- [DynamoDB API](#)
- [Amazon DynamoDB 中支援的資料類型和命名規則](#)
- [資料表類別](#)
- [分割區與資料分佈](#)

DynamoDB 速查表

本備忘單提供使用 Amazon DynamoDB 及其各種 AWS 開發套件的快速參考資料。

初始設定

1. [註冊 AWS](#).
2. [取得 AWS 存取金鑰](#)，以程式設計方式存取 DynamoDB。
3. [設定您 DynamoDB 憑證](#)。

另請參閱：

- [設定 DynamoDB \(Web 服務\)](#)
- [DynamoDB 入門](#)
- [核心元件的基本概述](#)

SDK 或 CLI

選擇您偏好的 [SDK](#)，或設定 [AWS CLI](#)。

Note

當您 AWS CLI 在 Windows 上使用時，不在引號內的反斜線 (\) 會被視為歸位字元。此外，您必須逸出其他引號內的任何引號和大括號。如需範例，請參閱下一章節「建立資料表」的 Windows 標籤。

另請參閱：

- [AWS CLI 使用 DynamoDB](#)
- [DynamoDB 入門 - 步驟 2](#)

基本動作

本節提供基本 DynamoDB 任務的程式碼。如需這些工作的詳細資訊，請參閱 [DynamoDB 和開發套件入 AWS 門](#)。

建立資料表

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
```

將項目寫入資料表

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

從資料表讀取項目

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

從資料表刪除項目

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

查詢資料表

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

刪除資料表

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

列出資料表名稱

```
aws dynamodb list-tables
```

命名規則

- 所有名稱都必須使用 UTF-8 編碼並區分大小寫。
- 資料表名稱與索引名稱長度必須介於 3 到 255 個字元之間，而且只能包含下列字元：
 - a-z
 - A-Z
 - 0-9
 - _ (底線)
 - - (破折號)
 - . (點號)
- 屬性名稱至少必須為一個字元長，而且大小不能超過 64KB。

如需詳細資訊，請參閱[命名規則](#)。

服務配額基本概念

讀取和寫入單位

- 讀取容量單位 (RCU) – 每秒一個高度一致性讀取，或每秒兩個最終一致讀取，適用於大小上限為 4 KB 的項目。
- 寫入容量單位 (WCU) – 每秒一個寫入，適用於大小上限為 1 KB 的項目。

資料表限制

- 資料表大小 – 資料表大小沒有任何實際限制。就項目數或位元組數而言，資料表是沒有限制的。
- 表格數量 — 對於任何 AWS 帳戶，每個 AWS 區域都有 2,500 個表格的初始配額。
- 查詢和掃描的頁面大小限制 — 每個查詢或掃描的每頁限制為 1 MB。若您在資料表上的查詢參數或掃描操作產生超過 1 MB 的資料，DynamoDB 會傳回初始相符項目。同時也會傳回一個 LastEvaluatedKey 屬性，您可以在新的請求中使用該屬性來讀取下一頁。

索引

- 本機次要索引 (LSI) — 您可以定義最多五個本機次要索引。當索引必須與基礎資料表保持高度一致時，LSI 就特別有用。
- 全域次要索引 (GSIs) – 每個資料表有 20 個全域次要索引的預設配額。
- 每份資料表投影次要索引屬性 – 您最多可以投影 100 個屬性到資料表所有的區域和全域次要索引。這只適用於使用者指定的投影屬性。

分割區索引鍵

- 分割區索引鍵值的長度下限為 1 個位元組。長度上限為 2048 個位元組。
- 資料表或次要索引中，不同的分割區索引鍵值數目沒有實際限制。
- 排序索引鍵值的長度下限為 1 個位元組。長度上限為 1024 個位元組。
- 一般而言，每個分割區索引鍵值的相異排序索引鍵值數目沒有實際限制。例外狀況是有次要索引的資料表。

如需次要索引、分割區索引鍵設計和排序索引鍵設計的詳細資訊，請參閱[最佳實務](#)。

常用資料類型限制

- String (字串) – 字串長度受到項目大小上限 400 KB 的限制。字串是 UTF-8 二進位編碼的 Unicode。
- 數字 – 數字的精準度最多可達 38 位數，可為正數、負數或零。
- 二進位 – 二進位長度受到項目大小上限 400 KB 的限制。使用二進位屬性的應用程式必須先以 base64 編碼資料，再傳送到 DynamoDB。

如需支援的完整資料類型清單，請參閱[資料類型](#)。如需詳細資訊，請參閱[服務配額](#)。

項目、屬性和表達式參數

DynamoDB 的項目大小上限是 400 KB，包括屬性名稱二進位長度 (UTF-8 長度) 和屬性值二進位長度 (UTF-8 長度)。屬性名稱算作大小限制的一部分。

清單、映射或集合中值數目不限，只要含有值的項目符合 400 KB 項目大小限制。

針對表達式參數，任何表達式字串的長度上限為 4 KB。

如需項目大小、屬性和表達式參數的詳細資訊，請參閱[服務配額](#)。

其他資訊

- [安全性](#)
- [監控和記錄](#)
- [使用串流](#)
- [備份和 Point-in-time 恢復](#)
- [與其他 AWS 服務整合](#)
- [API 參考](#)
- [架構中心：資料庫最佳實務](#)
- [教學課程影片](#)
- [DynamoDB 論壇](#)

Amazon DynamoDB 的核心元件

在 DynamoDB 中，資料表、項目與屬性都是您會用到的核心元件。資料表是項目的集合，而每個項目則是屬性的集合。DynamoDB 使用主索引鍵來唯一識別資料表中的各個項目和次要索引，以便提供更多的查詢靈活性。您可以使用 DynamoDB Streams 來擷取 DynamoDB 資料表中的資料修改事件。

DynamoDB 中有其限制。如需詳細資訊，請參閱 [Amazon DynamoDB 中的服務、帳戶和資料表配額](#)。

以下影片將為您介紹資料表、項目與屬性。

[資料表、項目與屬性](#)

資料表、項目與屬性

以下是基本 DynamoDB 元件：

- **資料表**：與其他資料庫系統類似，DynamoDB 會將資料存放在資料表中。資料表是資料的集合。例如，您可以使用名為 People 的資料表範例，來存放朋友、家人或其他任何人的相關個人聯絡資訊。您也可以使用 Cars 資料表來存放各人駕駛之車輛的相關資訊。
- **項目**：每個資料表包含零或多個項目。項目是可從所有其他項目唯一識別的一組屬性。在 People 資料表中，每個項目代表一個人。在 Cars 資料表中，每個項目代表一輛車。DynamoDB 中的項目與其他資料庫系統中的資料列、紀錄或元組有許多相似之處。在 DynamoDB 中，可以存放在資料表中的項目數不限。
- **屬性**：每個項目是由一或多個屬性所組成。屬性是一種基本資料元素，不必再進一步細分。例如，「人員」表格中的項目包含稱為 PersonID、LastNameFirstName、等的屬性。在 Department 資料表中，項目可能會有 DepartmentID、Name、Manager 等屬性。DynamoDB 中的屬性與其他資料庫系統中的欄位或資料行有許多相似之處。

下圖顯示一個名為 People 的資料表，其中包含一些範例項目與屬性。

```
People

{
  "PersonID": 101,
  "LastName": "Smith",
  "FirstName": "Fred",
  "Phone": "555-4321"
}

{
  "PersonID": 102,
  "LastName": "Jones",
  "FirstName": "Mary",
  "Address": {
    "Street": "123 Main",
    "City": "Anytown",
    "State": "OH",
```

```
        "ZIPCode": 12345
    }
}

{
  "PersonID": 103,
  "LastName": "Stephens",
  "FirstName": "Howard",
  "Address": {
    "Street": "123 Main",
    "City": "London",
    "PostalCode": "ER3 5K8"
  },
  "FavoriteColor": "Blue"
}
```

People 資料表的注意事項如下：

- 資料表中的每個項目都有唯一識別符或主索引鍵，可區分該項目與資料表中的所有其他項目。在 People 資料表中，主索引鍵是由一個屬性 (PersonID) 所組成。
- 除了主索引鍵之外，People 資料表沒有結構描述，這表示您不需要事先定義屬性或其資料類型。每個項目可以有其專屬的不同屬性。
- 大多數屬性為純量，亦即只能有一個值。字串與數字是常見的純量範例。
- 有些項目有巢狀屬性 (Address)。DynamoDB 支援巢狀屬性，最多 32 層深。

以下是另一個名為 Music 的範例資料表，您可以用來追蹤音樂收藏。

```
Music

{
  "Artist": "No One You Know",
  "SongTitle": "My Dog Spot",
  "AlbumTitle": "Hey Now",
  "Price": 1.98,
  "Genre": "Country",
  "CriticRating": 8.4
}

{
  "Artist": "No One You Know",
```

```
"SongTitle": "Somewhere Down The Road",
"AlbumTitle": "Somewhat Famous",
"Genre": "Country",
"CriticRating": 8.4,
"Year": 1984
}

{
  "Artist": "The Acme Band",
  "SongTitle": "Still in Love",
  "AlbumTitle": "The Buck Starts Here",
  "Price": 2.47,
  "Genre": "Rock",
  "PromotionInfo": {
    "RadioStationsPlaying": [
      "KHCR",
      "KQBX",
      "WTNR",
      "WJJH"
    ],
    "TourDates": {
      "Seattle": "20150622",
      "Cleveland": "20150630"
    },
    "Rotation": "Heavy"
  }
}

{
  "Artist": "The Acme Band",
  "SongTitle": "Look Out, World",
  "AlbumTitle": "The Buck Starts Here",
  "Price": 0.99,
  "Genre": "Rock"
}
```

Music 資料表的注意事項如下：

- 音樂的主鍵包含兩個屬性 (藝術家和 SongTitle) 。資料表中的每個項目必須有這兩個屬性。藝術家的組合，並SongTitle區分表中的每個項目從所有其他項目。
- 除了主索引鍵之外，Music 資料表沒有結構描述，這表示您不需要事先定義屬性或其資料類型。每個項目可以有其專屬的不同屬性。

- 其中一個項目具有巢狀屬性 (PromotionInfo)，其中包含其他巢狀屬性。DynamoDB 支援巢狀屬性，最多 32 層深。

如需詳細資訊，請參閱 [在 DynamoDB 中使用資料表和資料](#)。

主索引鍵

當您建立資料表時，除了資料表名稱，您還必須指定資料表的主索引鍵。主索引鍵可唯一識別資料表中的每個項目，因此沒有兩個項目的索引鍵是相同的。

DynamoDB 支援兩種不同類型的主索引鍵：

- 分割區索引鍵：簡易主索引鍵，由一個屬性 (稱為分割區索引鍵) 所組成。

DynamoDB 使用分割區索引鍵值作為內部雜湊函數的輸入。雜湊函數的輸出決定要存放項目的分割區 (DynamoDB 的內部實體儲存體)。

在只有一個分割區索引鍵的資料表中，沒有兩個項目的分割區索引鍵值是相同的。

所以 [資料表、項目與屬性](#) 中描述的 People 資料表，是具有簡單主索引鍵 (PersonID) 的資料表範例。您可以透過提供該項目的 PersonID 值，直接存取「人員」(People) 表格中的任何項目。

- 分割區索引鍵與排序索引鍵：稱為複合主索引鍵，這種類型的索引鍵是由兩個屬性組成。第一個屬性是分割區索引鍵，第二個屬性是排序索引鍵。

DynamoDB 使用分割區索引鍵值作為內部雜湊函數的輸入。雜湊函數的輸出決定要存放項目的分割區 (DynamoDB 的內部實體儲存體)。具有相同分割區索引鍵值的所有項目會存放在一起，並依排序索引鍵值排序。

在具有一個分割區金鑰與一個排序金鑰的資料表中，兩個項目可能會有相同的分割區金鑰值。不過，這兩個項目必須具有不同的排序金鑰值。

中描述的「音樂」資料表 [資料表、項目與屬性](#) 是具有複合主索引鍵 (Artist 和 SongTitle) 的表格範例。如果您提供演出者和該項目的 SongTitle 值，則可以直接存取「音樂」表中的任何項目。

複合主索引鍵可讓您更有彈性地查詢資料。例如，如果您只提供 Artist 值，則 DynamoDB 會擷取該演出者的所有歌曲。若只要擷取特定演出者的一部分歌曲，您可以為演出者提供值以及的值範圍 SongTitle。

Note

項目的分割區索引鍵也稱為其雜湊屬性。雜湊屬性一詞衍生自 DynamoDB 中內部雜湊函數的用法，可將資料項目根據其分割區索引鍵值平均分佈到所有分割區。

項目的排序索引鍵也稱為其範圍屬性。範圍屬性一詞衍生自 DynamoDB 存放項目的方式，具有相同分割區索引鍵的項目會實際緊密相鄰，並依排序索引鍵值排序。

每個主索引鍵屬性必須是純量 (亦即只能保留一個值)。主索引鍵屬性允許的資料類型僅限於字串、數字或二進位。其他非索引鍵屬性則沒有此限制。

次要索引

您可以在資料表上建立一或多個次要索引。次要索引可讓您在除了使用主索引鍵查詢外，也可使用備用索引鍵查詢資料表中的資料。DynamoDB 不需要您使用索引，但可讓您的應用程式在查詢資料時更具靈活性。在資料表上建立次要索引之後，您可以從索引讀取資料，方法與從資料表讀取十分相似。

DynamoDB 支援兩種索引：

- 全域次要索引：一種含分割區索引鍵或排序索引鍵的索引，這些索引鍵可與資料表上的索引鍵不同。
- 本機次要索引：是一種與資料表擁有相同分區索引鍵但不同排序索引鍵的索引。

在 DynamoDB 中，全域次要索引 (GSI) 是跨越整個表格的索引，可讓您跨所有分區索引鍵進行查詢。本機次要索引 (LSI) 是與基底資料表具有相同分割索引鍵但排序索引鍵不同的索引鍵。

DynamoDB 中的每個資料表配額為 20 個全域次要索引 (預設配額) 與 5 個本機次要索引。

在先前顯示的「音樂」資料表範例中，您可以依照演出者 (分割區索引鍵) 或藝術家與 SongTitle(分割索引鍵和排序索引鍵) 來查詢資料項目。如果您還想按流派和查詢數據怎麼AlbumTitle辦？要做到這一點，你可以創建一個類型的索引 AlbumTitle，然後查詢索引的方式大致相同的方式，你會查詢音樂表。

下圖顯示了示例音樂表，名為一個新的索引GenreAlbumTitle。在索引中，「類型」是分割索引鍵，AlbumTitle是排序索引鍵。

音樂資料表	GenreAlbumTitle
<pre>{ "Artist": "No One You Know",</pre>	<pre>{ "Genre": "Country",</pre>

音樂資料表	GenreAlbumTitle
<pre>"SongTitle": "My Dog Spot", "AlbumTitle": "Hey Now", "Price": 1.98, "Genre": "Country", "CriticRating": 8.4 }</pre>	<pre>"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>

音樂資料表

```
{
  "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"
  }
}
```

GenreAlbumTitle

```
{
  "Genre": "Rock",
  "AlbumTitle": "The Buck Starts
Here",
  "Artist": "The Acme Band",
  "SongTitle": "Still In Love"
}
```

```
{
  "Artist": "The Acme Band",
  "SongTitle": "Look Out, World",
  "AlbumTitle": "The Buck Starts
Here",
  "Price": 0.99,
  "Genre": "Rock"
}
```

```
{
  "Genre": "Rock",
  "AlbumTitle": "The Buck Starts
Here",
  "Artist": "The Acme Band",
  "SongTitle": "Look Out, World"
}
```

請注意下列有關GenreAlbumTitle索引的事項：

- 每個索引都屬於一個資料表，稱為索引的基礎資料表。在上述範例中，Music 是索引GenreAlbumTitle引的基底資料表。
- DynamoDB 會自動維護索引。當您新增、更新或刪除基礎資料表中的項目時，DynamoDB 會在屬於該資料表的任何索引中新增、更新或刪除對應的項目。
- 建立索引時，您可以指定要從基礎資料表複製或投影到索引的屬性。DynamoDB 至少會將索引鍵屬性從基礎資料表投影到索引。GenreAlbumTitle 即為一例，其中只有索引鍵屬性會從 Music 資料表投影到索引。

您可以查詢索引GenreAlbumTitle引以尋找特定類型的所有相簿 (例如，所有 Rock 相簿)。您也可以查詢此索引，尋找特定內容類型中具有特定專輯標題的所有專輯 (例如標題開頭字母為 H 的所有 Country 專輯)。

如需詳細資訊，請參閱 [使用次要索引來改善資料存取](#)。

DynamoDB Streams

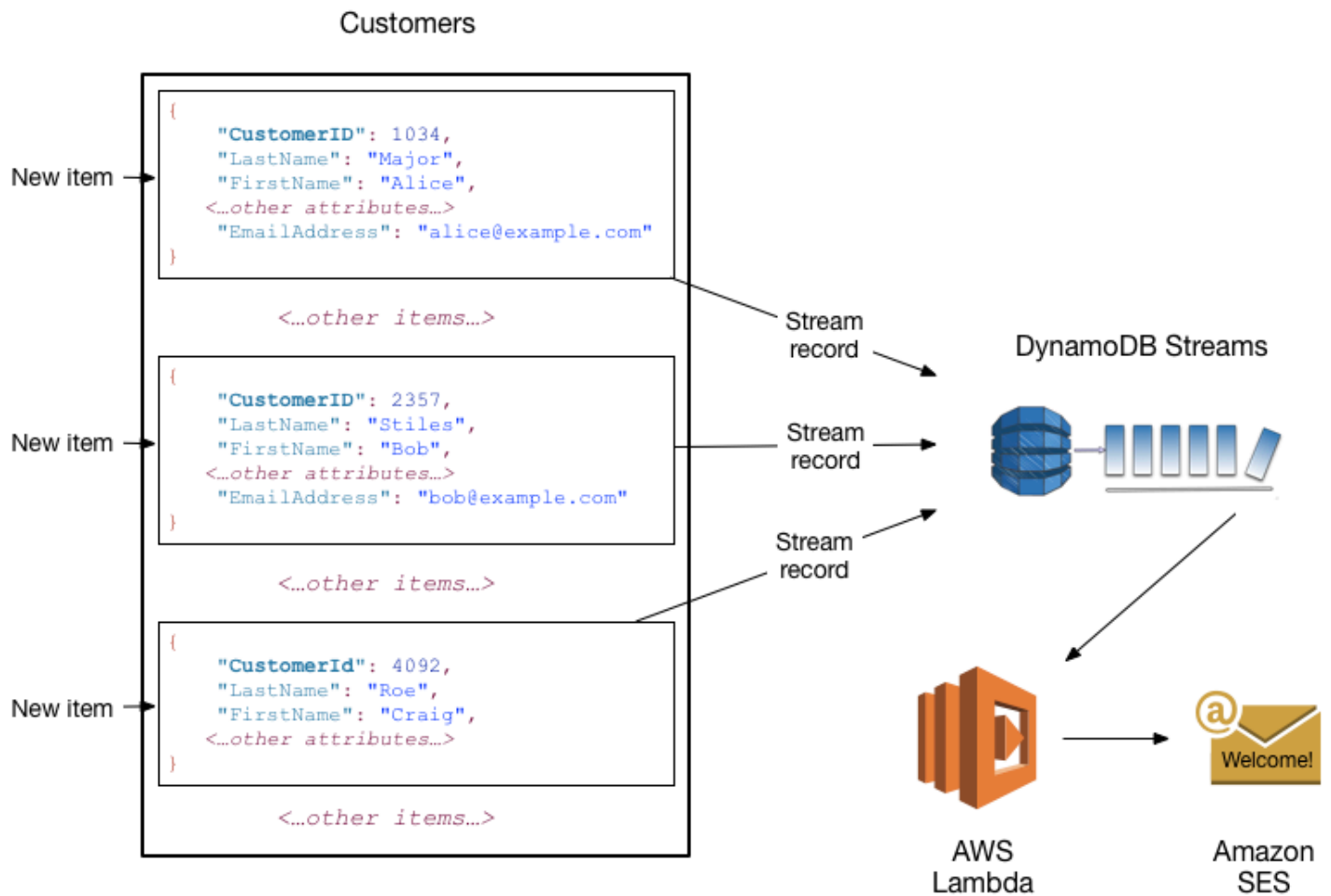
DynamoDB Streams 是選用功能，可擷取 DynamoDB 資料表中的資料修改事件。這些事件的相關資料會依事件出現的順序，近乎即時地出現在串流中。

每個事件是以串流紀錄表示。如果您在資料表上啟用串流，只要發生下列其中一個事件，DynamoDB Streams 就會寫入一個串流紀錄：

- 新增項目至資料表：串流會擷取整個項目的影像，包括其所有屬性。
- 項目已更新：串流會擷取項目中已修改之任何屬性的「之前」與「之後」影像。
- 從資料表刪除項目：串流會擷取整個項目的影像，再加以刪除。

每個串流紀錄也會包含資料表的名稱、事件時間戳記與其他中繼資料。串流紀錄的存留期為 24 小時，之後會自動從串流移除。

您可以搭配使用 DynamoDB Streams AWS Lambda 來建立觸發程式碼，每當串流中出現感興趣的事件時，就會自動執行。例如，以含有公司客戶資訊的 Customers 資料表為例。假設您想要傳送「歡迎」電子郵件給每個新客戶。您可以在該資料表上啟用串流，然後將串流與 Lambda 函數建立關聯。Lambda 函數會在每次出現新的串流紀錄時執行，但只會處理 Customers 資料表的新增項目。針對具有 EmailAddress 屬性的任何項目，Lambda 函數會呼叫 Amazon Simple Email Service (Amazon SES) 來將電子郵件傳送至該地址。



Note

在此範例中，最後一個客戶 Craig Roe 不會收到電子郵件，因為他沒有EmailAddress。

除了觸發器之外，DynamoDB Streams 還提供強大的解決方案，例如區域內部和 AWS 區域間的資料複製、DynamoDB 表中資料的具體化視圖、使用 Kinesis 具體化視圖的資料分析等等。

如需詳細資訊，請參閱 [DynamoDB Streams 的變更資料擷取](#)。

DynamoDB API

若要使用 Amazon DynamoDB，您的應用程式必須使用一些簡單的 API 操作。以下是這些操作的摘要，並依類別分組。

Note

如需 API 操作的完整清單，請參閱 [Amazon DynamoDB API 參考](#)。

主題

- [控制平台](#)
- [資料平面](#)
- [DynamoDB Streams](#)
- [交易](#)

控制平台

控制平面操作可讓您建立及管理 DynamoDB 資料表。它們也可讓您使用索引、串流，以及相依於資料表的其他物件。

- `CreateTable`：建立新的資料表。您可以選擇性地建立一或多個次要索引，並為資料表啟用 DynamoDB Streams。
- `DescribeTable`：傳回資料表的相關資訊，例如其主索引鍵結構描述、傳輸量設定、索引資訊。
- `ListTables`：傳回您所有的資料表名稱清單。
- `UpdateTable`：修改資料表或其索引的設定、在資料表上建立或移除新的索引，或修改資料表的 DynamoDB Streams 設定。
- `DeleteTable`：從 DynamoDB 移除資料表及其所有相依物件。

資料平面

資料平面操作可讓您對資料表中的資料執行建立、讀取、更新與刪除 (也稱為 CRUD) 動作。部分資料平面操作也可讓您從讀取資料。

您可以使用 [PartiQL：一種適用於 Amazon DynamoDB 的 SQL 相容查詢語言](#) 來執行這些 CRUD 操作，也可以使用 DynamoDB 的傳統 CRUD API 將每個操作分隔為不同的 API 呼叫。

PartiQL - 一種 SQL 相容查詢語言

- `ExecuteStatement`：讀取資料表中的多個項目。您也可以寫入或更新資料表中的單一項目。在寫入或更新單一項目時，您必須指定主索引鍵屬性。

- `BatchExecuteStatement`：從資料表寫入、更新或讀取多個項目。這比 `ExecuteStatement` 的效率高上好幾倍，因為您的應用程式只需要往返網路一次即可寫入或讀取項目。

傳統 API

建立資料

- `PutItem`：將單一項目寫入資料表。您必須指定主索引鍵屬性，但不必指定其他屬性。
- `BatchWriteItem`：最多可將 25 個項目寫入資料表。這比呼叫 `PutItem` 的效率高上好幾倍，因為您的應用程式只需要往返網路一次即可寫入項目。

讀取資料

- `GetItem`：從資料表擷取單一項目。您必須指定所需項目的主索引鍵。您可以擷取整個項目，或只擷取其屬性子集。
- `BatchGetItem`：最多可從一或多個資料表擷取 100 個項目。這比呼叫 `GetItem` 的效率高上好幾倍，因為您的應用程式只需要往返網路一次即可讀取項目。
- `Query`：擷取所有具有特定分割區索引鍵的項目。您必須指定分割區索引鍵值。您可以擷取整個項目，或只擷取其屬性子集。您可以選擇性地將條件套用至排序索引鍵值，只擷取具有相同分割區索引鍵的資料子集。您可以在資料表上使用此操作，只要該資料表同時具有分割區索引鍵與排序索引鍵。您也可以在此索引上使用此操作，只要該索引同時具有分割區索引鍵與排序索引鍵。
- `Scan`：擷取指定資料表或索引中的所有項目。您可以擷取整個項目，或只擷取其屬性子集。您可以選擇性地套用篩選條件，只傳回您感興趣的值並捨棄其餘值。

更新資料

- `UpdateItem`：修改項目中的一或多個屬性。您必須指定要修改之項目的主索引鍵。您可以新增屬性，以及修改或移除現有的屬性。您也可以執行條件式更新，只在符合使用者定義的條件時，更新才會成功。您可以選擇性地實作原子計數器，遞增或遞減數字屬性，而不會影響其他寫入請求。

刪除資料

- `DeleteItem`：從資料表刪除單一項目。您必須指定要刪除之項目的主索引鍵。
- `BatchWriteItem`：最多可從一或多個資料表刪除 25 個項目。這比呼叫 `DeleteItem` 的效率高上好幾倍，因為您的應用程式只需要往返網路一次即可刪除項目。

Note

您可以使用 `BatchWriteItem` 建立資料和刪除資料。

DynamoDB Streams

DynamoDB Streams 操作可讓您在資料表上啟用或停用串流，並允許存取串流中包含的資料修改紀錄。

- `ListStreams`：傳回您所有的串流清單，或只傳回特定資料表的串流。
- `DescribeStream`：傳回串流的相關資訊，例如其 Amazon Resource Name (ARN)，以及您的應用程式可開始讀取前幾個串流紀錄的位置。
- `GetShardIterator`：傳回碎片疊代運算，這是您的應用程式從串流擷取紀錄所使用的資料結構。
- `GetRecords`：使用指定的碎片疊代運算，擷取一或多個串流紀錄。

交易

交易提供了不可分割性、一致性、隔離性和持久性 (ACID)，讓您能夠輕鬆地維持應用程式的資料正確度。

您可以使用 [PartiQL：一種適用於 Amazon DynamoDB 的 SQL 相容查詢語言](#) 來執行交易操作，也可以使用 DynamoDB 的傳統 CRUD API 將每個操作分隔為不同的 API 呼叫。

PartiQL - 一種 SQL 相容查詢語言

- `ExecuteTransaction`-一種批處理操作，允許對表內和表之間的多個項目進行 CRUD 操作，並保證 all-or-nothing 結果。

傳統 API

- `TransactWriteItems`-一種批處理操作，允許 `PutUpdate`，和 `Delete` 操作在表內和跨表中對多個項目進行操作，並具有保證的 all-or-nothing 結果。
- `TransactGetItems`：允許 `Get` 操作來從一或多個資料表擷取多個項目的批次操作。

Amazon DynamoDB 中支援的資料類型和命名規則

本節說明 Amazon DynamoDB 命名規則與 DynamoDB 支援的各種資料類型。這些資料類型有所限制。如需詳細資訊，請參閱 [資料類型](#)。

主題

- [命名規則](#)
- [資料類型](#)
- [資料類型描述項](#)

命名規則

DynamoDB 中的資料表、屬性與其他物件必須具有名稱。名稱應該具有意義且簡潔，例如 Products、Books 與 Authors 等都是一目了然的名稱。

以下是 DynamoDB 的命名規則：

- 所有名稱都必須使用 UTF-8 編碼並區分大小寫。
- 資料表名稱與索引名稱長度必須介於 3 到 255 個字元之間，而且只能包含下列字元：
 - a-z
 - A-Z
 - 0-9
 - _ (底線)
 - - (破折號)
 - . (點號)
- 屬性名稱的長度至少須為一個字元，而且大小不能超過 64KB。屬性名稱盡可能簡潔是公認的最佳實務。這有助於減少使用的讀取請求單位，因為屬性名稱包含在儲存體和輸送量使用量的計量中。

以下為例外狀況。這些屬性名稱絕對不能超過 255 個字元。

- 次要索引分割區索引鍵名稱
- 次要索引排序索引鍵名稱
- 任何使用者指定的投影屬性名稱 (僅適用於本機次要索引)

所有數字都會以字串形式跨網路傳送到 DynamoDB，以提高語言與程式庫之間的相容性。不過，DynamoDB 會將其視為數學運算的數字類型屬性。

您可以使用數字資料類型來代表日期或時間戳記。一個做法是使用 epoch 時間，也就是自 1970 年 1 月 1 日 00:00:00 UTC 起經過的秒數。例如，epoch 時間 1437136300 代表 2015 年 7 月 17 日下午 12:31:40 UTC。

如需詳細資訊，請參閱 http://en.wikipedia.org/wiki/Unix_time。

字串

字串是 UTF-8 二進位編碼的 Unicode。如果屬性未用作為索引或資料表的索引鍵，則字串的最小長度可以是零，而且受到最大 DynamoDB 項目大小限制 400 KB 的限制。

下列其他限制適用於定義為類型字串的主索引鍵屬性：

- 針對簡易主索引鍵，第一個屬性值 (分割區索引鍵) 的長度上限為 2048 位元組。
- 針對複合主索引鍵，第二個屬性值 (排序索引鍵) 的長度上限為 1024 位元組。

DynamoDB 會收集並比較使用基本 UTF-8 字串編碼位元組的字串。例如，"a" (0x61) 大於 "A" (0x41)，而 "¿" (0xC2BF) 大於 "z" (0x7A)。

您可以使用字串資料類型來代表日期或時間戳記。一個做法是使用 ISO 8601 字串，如下列範例所示：

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

如需詳細資訊，請參閱 http://en.wikipedia.org/wiki/ISO_8601。

Note

與傳統的關聯式資料庫不同，DynamoDB 原生不支援日期和時間資料類型。使用 Unix Epoch 時間可能很有用，可用於將日期和時間資料儲存為數字資料類型。

二進位

二進位類型屬性可存放任何二進位資料，例如壓縮文字、加密資料或影像。每當 DynamoDB 比較二進位值時，都會將二進位資料的每個位元組視為不帶正負號。

如果屬性未用作為索引或資料表的索引鍵，則二進位屬性的長度可以是零，而且受到最大 DynamoDB 項目大小限制 400 KB 的限制。

如果您將主索引鍵屬性定義為二進位類型屬性，則會有以下其他限制：

- 針對簡易主索引鍵，第一個屬性值 (分割區索引鍵) 的長度上限為 2048 位元組。
- 針對複合主索引鍵，第二個屬性值 (排序索引鍵) 的長度上限為 1024 位元組。

您的應用程式必須將二進位值編碼為 base64 編碼格式，再傳送到 DynamoDB。收到這些值時，DynamoDB 會將資料解碼為不帶正負號的位元組陣列，並用作二進位屬性長度。

下列範例顯示使用 base64 編碼文字的二進位屬性。

```
dGhpcyB0ZXh0IGlzIGJhc2U2NC11bmNvZGVk
```

Boolean

布林值類型屬性可存放 true 或 false。

Null

Null 代表具有未知或未定義狀態的屬性。

文件類型

文件類型為清單與映射。這些資料類型可彼此互相巢狀來代表複雜的資料結構，最高可達 32 層深。

清單或映射中的值數目不限，只要含有值的項目符合 DynamoDB 項目大小限制 (400 KB)。

如果屬性不用於資料表或索引鍵，則屬性值可以是空字串或空的二進位值。屬性值不得為空的集合 (字串集合、數字集合或二進位集合)。然而，系統允許空的清單和映射。清單和映射中允許空字串和二進位值。如需詳細資訊，請參閱 [Attributes](#)。

清單

清單類型屬性可存放一組排序的值。清單會以方括號括住：`[...]`

清單類似於 JSON 陣列。清單元素中可存放的資料類型不限，而且清單元素中的元素不必屬於相同類型。

下列範例顯示一個清單，其中包含兩個字串與一個數字。

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

Note

DynamoDB 可讓您使用清單中的個別元素，即使這些元素的巢狀結構很深也一樣。如需詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。

Map

映射類型屬性可存放一組未排序的名稱/值對。映射會以大括弧括住：`{ ... }`

映射類似於 JSON 物件。映射元素中可存放的資料類型不限，而且映射中的元素不必屬於相同類型。

映射很適合用來將 JSON 文件存放到 DynamoDB 中。下列範例顯示一個映射，其中包含字串、數字與含有另一個映射的巢狀清單。

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

Note

DynamoDB 可讓您使用映射中的個別元素，即使這些元素的巢狀結構很深也一樣。如需詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。

集合

DynamoDB 支援代表數字集合、字串集合或二進位值集合的類型。集合內的所有元素必須屬於相同類型。例如，數字集合只能包含數字，字串集合只能包含字串。

集合中的值數目不限，只要含有值的項目符合 DynamoDB 項目大小限制 (400 KB)。

集合內的每個值必須是唯一的。集合內的值順序不會保留。因此，您的應用程式不得依賴集合內元素的任何特定順序。DynamoDB 不支援空集合，但集合中允許存在空字串和二進位值。

下列範例顯示字串集、數字集與二進位集：

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

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

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

資料類型描述項

低階 DynamoDB API 通訊協定使用資料類型描述項做為字符，告知 DynamoDB 如何解譯每項屬性。

下列為 DynamoDB 資料類型描述項的完整清單：

- **S**：字串
- **N**：數字
- **B**：二進位
- **BOOL**：布林值
- **NULL**：Null
- **M**：映射
- **L**：清單
- **SS**：字串集合
- **NS**：數字集合
- **BS**：二進位集合

資料表類別

DynamoDB 提供兩種資料表類別，旨在協助您最佳化成本。預設值為 DynamoDB 標準資料表類別，建議大多數工作負載使用。DynamoDB 標準-不常存取 (DynamoDB 標準-IA) 資料表類別針對以儲存為主要成本的資料表進行最佳化。例如，儲存不常存取資料的資料表 (例如應用程式日誌、舊社交媒體貼文、電子商務訂單歷史記錄以及過去遊戲成就) 都是適合標準-IA 資料表類別的選項。如需定價詳細資訊，請參閱 [Amazon DynamoDB 定價](#)。

每個 DynamoDB 資料表都與一個資料表類別相關聯 (依 DynamoDB Standard 預設)。與資料表相關聯的所有次要索引都使用相同的資料表類別。每個資料表類別針對資料儲存以及讀取和寫入要求提供不同的定價。您可以根據資料表的儲存體和輸送量使用模式，為資料表選取最具成本效益的資料表類別。

資料表類別的選擇不是永久性的 — 您可以使用 AWS Management Console、AWS CLI 或 SDK 變更此設定。AWS DynamoDB 也支援使用單一區域資料表和全域表來 AWS CloudFormation 管理您的資料表類別。若要進一步了解如何選取資料表類別，請參閱 [選擇資料表類別時的注意事項](#)。

分割區與資料分佈

Amazon DynamoDB 將資料存放在分割區中。分割區是資料表的儲存空間配置，由固態硬碟 (SSD) 支援，並自動複製到同一區域內的多個可用區域。AWS 分割區管理完全是由 DynamoDB 處理，您永遠不需要自行管理分割區。

當您建立資料表時，資料表的初始狀態為 CREATING。在此階段期間，DynamoDB 會配置足夠的分割區給資料表，讓它可以處理您的佈建輸送量需求。您可以在資料表狀態變更為 ACTIVE 之後，開始寫入及讀取資料表資料。

DynamoDB 會在下列情況下配置額外的分割區給資料表：

- 如果您將資料表的佈建輸送量設定，增加到超過現有分割區所能支援的設定。
- 如果現有的分割區容量將滿且需要更多儲存空間。

分割區管理會在背景自動進行，而且對您的應用程式是透明的。您的資料表會全程可供使用，並完整支援您的佈建輸送量需求。

如需詳細資訊，請參閱 [分割區索引鍵設計](#)。

DynamoDB 中的全域次要索引也是由分割區所組成。全域次要索引中的資料會與其基礎資料表中的資料分開存放，但索引分割區的運作方式與資料表分割區相同。

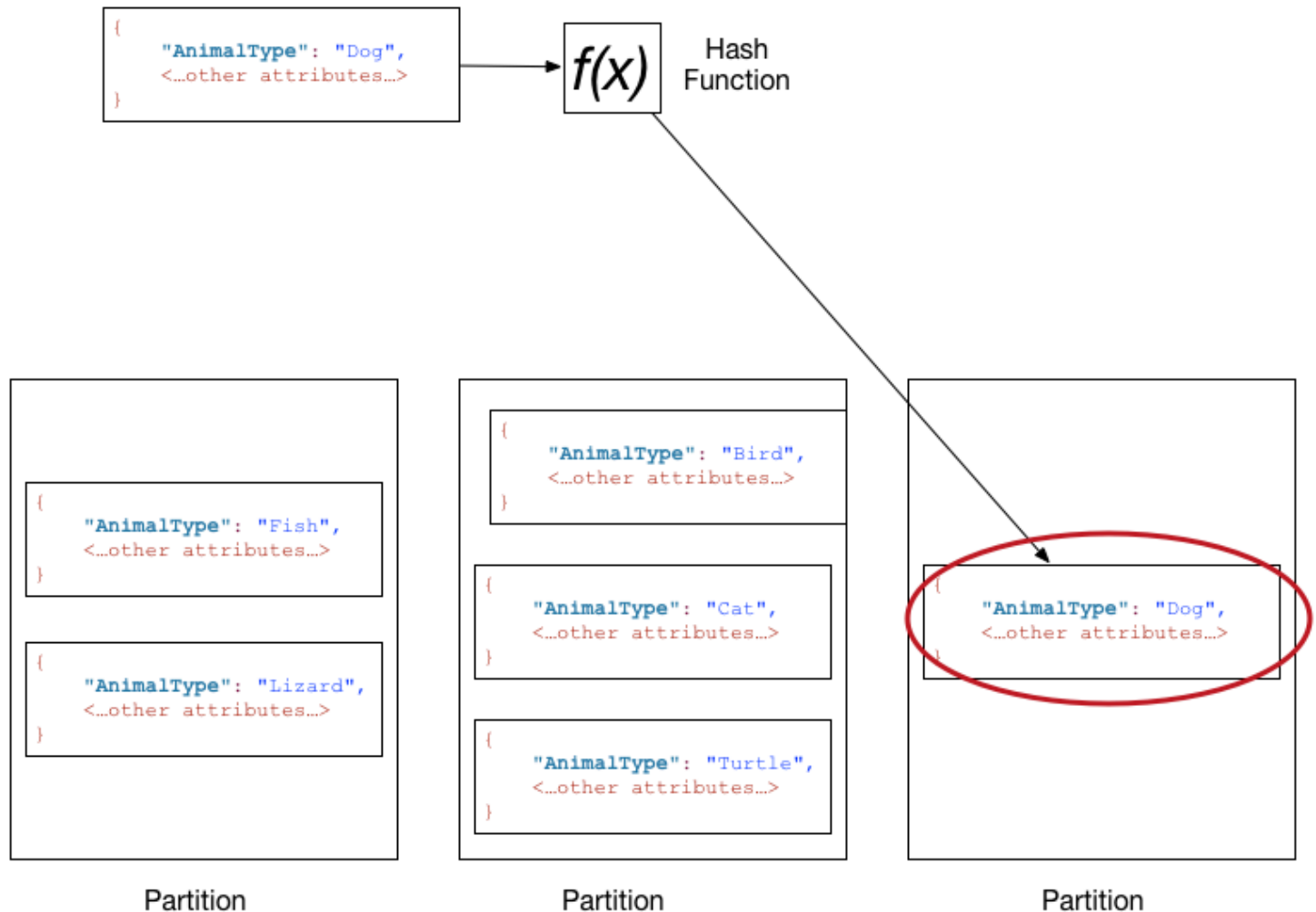
資料分佈：分割區索引鍵

如果您的資料表具有簡易主索引鍵 (僅限分割區索引鍵)，DynamoDB 會根據每個項目的分割區索引鍵值來存放與擷取項目。

為了將項目寫入資料表，DynamoDB 使用分割區索引鍵值作為內部雜湊函數的輸入。雜湊函數的輸出值決定要存放項目的分割區。

若要讀取資料表的項目，您必須指定項目的分割區索引鍵值。DynamoDB 會使用此值作為雜湊函數的輸入，產生能夠找到此項目的分割區。

下圖顯示一個名為 Pets 的資料表，其橫跨多個分割區。表的主鍵是 AnimalType (僅顯示此鍵屬性)。DynamoDB 使用其雜湊函數來判斷新項目的存放位置，在此例中是依據字串 Dog 的雜湊值。請注意，項目不會依序存放。每個項目的位置取決於其分割區索引鍵的雜湊值。



Note

DynamoDB 已經過最佳化，可將項目一致分佈到資料表的分割區，不論有多少分割區。建議您選擇相較於資料表中的項目數，可擁有大量相異值的分割區索引鍵。

資料分佈：分割區索引鍵與排序索引鍵

如果資料表具有複合主索引鍵 (分割區索引鍵和排序索引鍵)，則 DynamoDB 會以與 [資料分佈：分割區索引鍵](#) 中所述相同的方式計算分區索引鍵的雜湊值。但是，它傾向於將具有相同分區鍵值的項目保持在一起，並按排序鍵屬性的值排序順序保持在一起。具有分區鍵值相同的項目集合稱為項目集合。項目

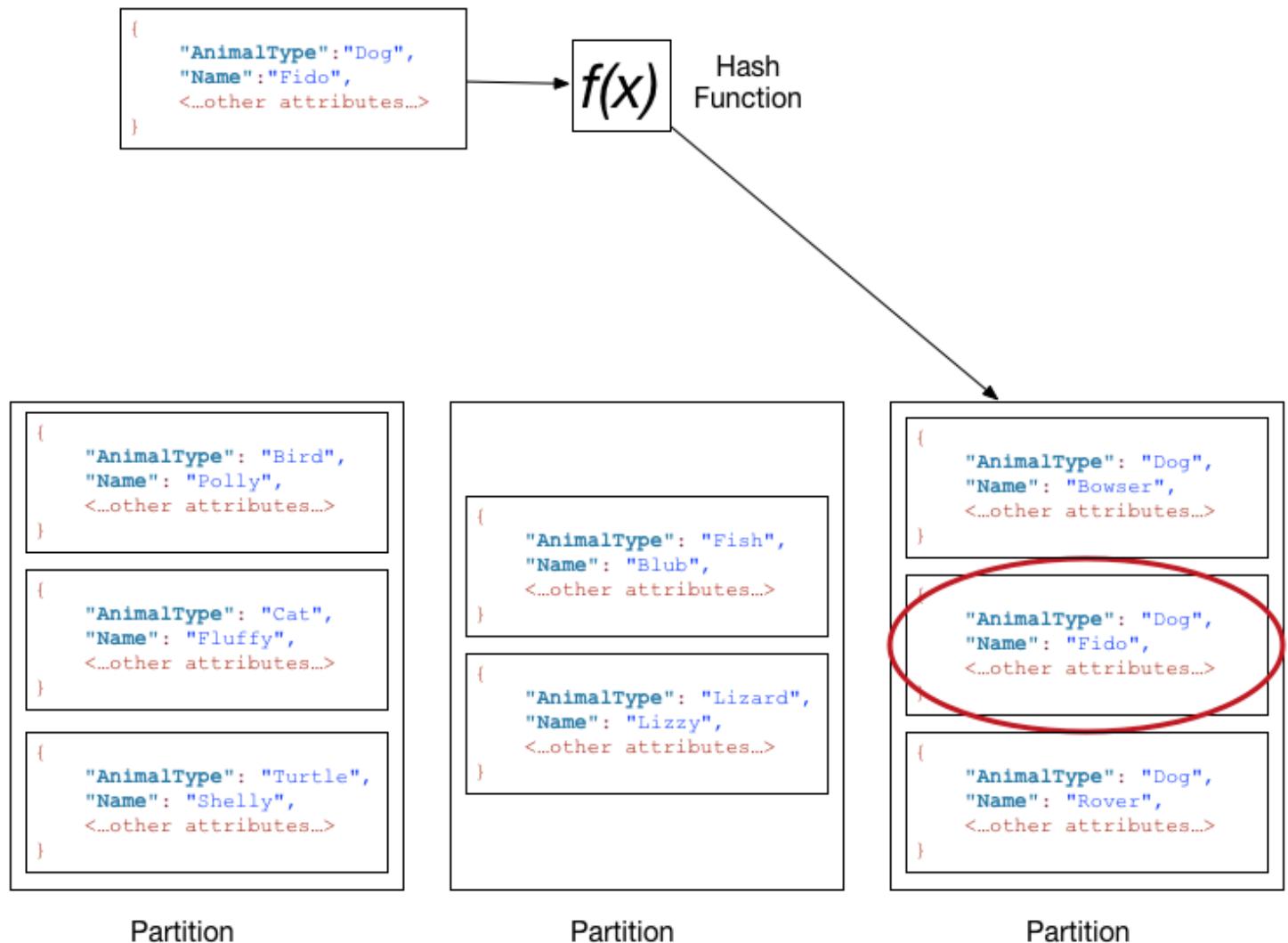
集合經過最佳化，可有效擷取集合中的項目範圍。如果您的資料表沒有本機次要索引，DynamoDB 會自動將項目集合分割為所需數量的分區，以儲存資料並提供讀取和寫入輸送量。

為了將項目寫入資料表，DynamoDB 會計算分割區索引鍵的雜湊值，來決定哪個分割區應該包含項目。在該分割區中，多個項目可以具有相同的分割區索引鍵值。因此，DynamoDB 使用相同的分割區索引鍵將項目儲存在其他項目中，按排序索引鍵遞增排列。

若要讀取資料表的項目，您必須指定項目的分割區索引鍵值和排序索引鍵值。DynamoDB 會計算分割區索引鍵的雜湊值，產生能夠找到此項目的分割區。

如果您想要的項目具有相同的分割區索引鍵值，您可以在單一操作 (Query) 中從資料表讀取多個項目。DynamoDB 會傳回具有該分割區索引鍵值的所有項目。您可以選擇性地將條件套用至排序索引鍵，讓它只傳回特定值範圍內的項目。

假設 `Pets` 表具有由 `AnimalType` (分區鍵) 和名稱 (排序鍵) 組成的複合主鍵。下圖顯示 DynamoDB 寫入一個項目，其分割區索引鍵值為 `Dog`，排序索引鍵值為 `Fido`。



若要從 Pets 資料表讀取同樣的項目，DynamoDB 會計算 Dog 的雜湊值，產生存放這些項目的分割區。然後，DynamoDB 會掃描排序鍵屬性值，直到找到 Fido。

要使用 Dog 讀取所有項目，您可以在不指定排序鍵條件 AnimalType 的情況下發出 Query 操作。根據預設，項目會依存放順序 (即依排序索引鍵的遞增順序) 傳回。您也可以改為請求遞減順序。

若只要查詢其中的一些 Dog 項目，您可以將條件套用至排序索引鍵 (例如僅限 Dog 項目，其中 Name 開始的字母介於 A 到 K 的範圍內)。

Note

在 DynamoDB 資料表中，每個分割區索引鍵值的相異排序索引鍵值數目沒有上限。如果您需要在 Pets 資料表中存放數十億個 Dog 項目，則 DynamoDB 會分配足夠的儲存空間來自動處理此需求。

從 SQL 到 NoSQL

若您是應用程式開發人員，您可能已有使用關聯式資料庫管理系統 (RDBMS) 和結構式查詢語言 (SQL) 的經驗。開始使用 Amazon DynamoDB 後，您可能會發現許多相似點，但也有許多相異點。NoSQL 一詞用來說明具高可用性、可擴展性且高效能最佳化的非關聯式資料庫系統。NoSQL 資料庫 (例如 DynamoDB) 與關聯式模型不同，它會使用另一種模型來管理資料，例如鍵/值對或文件儲存。如需詳細資訊，請參閱 [何謂 NoSQL ?](#)。

Amazon DynamoDB 支援 [PartiQL](#)，其為一種與 SQL 相容的開放原始碼查詢語言，讓您能以高效率的方式查詢資料，不論資料存放在何處或以何種格式存放。使用 PartiQL，您可以輕鬆處理來自關聯式資料庫的結構化資料、開放資料格式的半結構化和巢套資料，甚至是 NoSQL 或文件資料庫中允許不同行不同屬性的無固定結構資料。如需詳細資訊，請參閱 [PartiQL 查詢語言](#)。

以下章節說明常見的資料庫任務，並比較和比對 SQL 陳述式及其相對應的 DynamoDB 操作。

Note

本節中的 SQL 範例與 MySQL RDBMS 相容。

本節中的 DynamoDB 範例會顯示 DynamoDB 操作的名稱，並以 JSON 格式呈現該操作的參數。如需使用這些操作的程式碼範例，請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

主題

- [關聯式 \(SQL\) 或 NoSQL ?](#)
- [資料庫的特性](#)
- [建立資料表](#)
- [取得資料表的資訊](#)
- [將資料寫入資料表](#)
- [從資料表讀取資料時 SQL 與 DynamoDB 的主要差異](#)
- [管理索引](#)
- [修改資料表中的資料](#)
- [從資料表刪除資料](#)
- [移除資料表](#)

關聯式 (SQL) 或 NoSQL ?

當今應用程式的要求比以往更加嚴苛。例如，一款線上遊戲剛開始可能只有少數使用者和極少量的資料。但是，如果遊戲成功了，該遊戲可能容易超過基礎資料庫管理系統的資源。Web 應用程式常擁有數百、數千，甚至數百萬名並行使用者，並且每天產生數 TB 或以上的資料。這種應用程式的資料庫每秒必須處理成千上萬筆讀取和寫入。

Amazon DynamoDB 正適合這類工作負載。身為開發人員，您可以從小規模開始，並隨著您的應用程式變得愈來愈熱門而逐漸增加使用率。DynamoDB 可無縫擴展，能夠處理極大量的資料和使用者。

如需傳統關聯式資料庫建模以及如何針對 DynamoDB 進行調整的詳細資訊，請參閱 [在 DynamoDB 中製作關聯式資料模型的最佳實務](#)。

下表顯示關聯式資料庫管理系統 (RDBMS) 與 DynamoDB 之間的一些概要性差異。

特性	關聯式資料庫管理系統 (RDBMS)	Amazon DynamoDB
最佳工作負載	臨機操作查詢、資料倉儲、OLAP (線上分析處理)。	Web 規模應用程式，包括社群網路、遊戲、媒體共用及物聯網 (IoT)。
資料模型	關聯式模型需要定義良好的結構描述，所有資料皆會標準化成資料表、資料列及資料行。	DynamoDB 不具結構描述。每個資料表都必須具備一個主索引鍵，以唯一識別各個資料項。

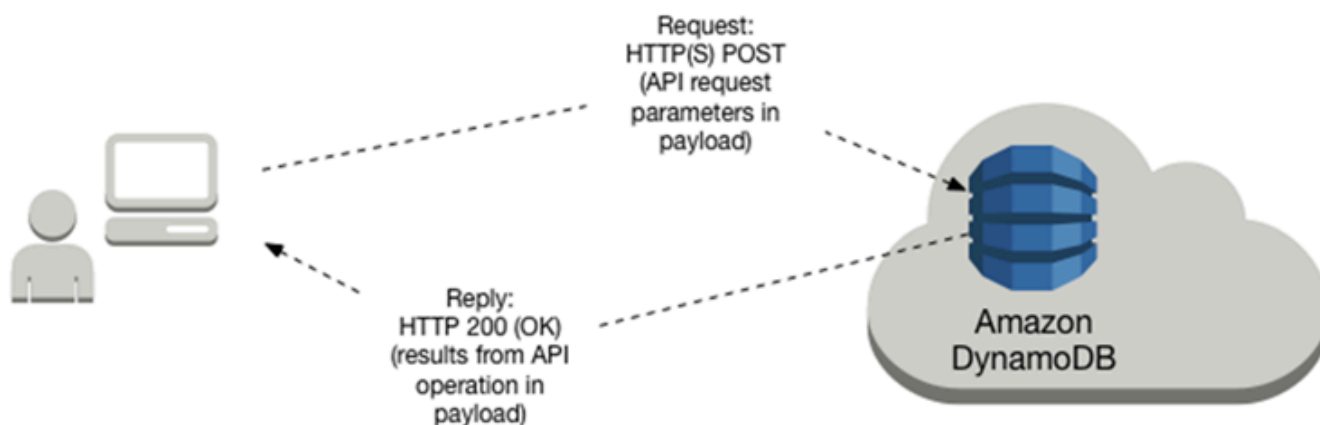
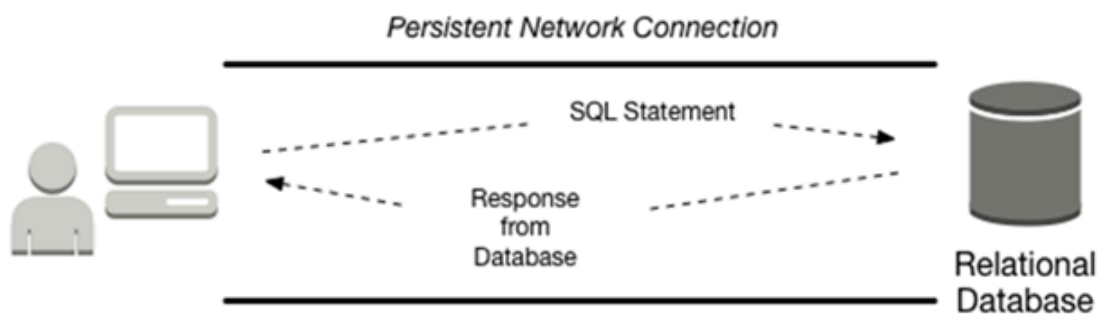
特性	關聯式資料庫管理系統 (RDBMS)	Amazon DynamoDB
	此外，資料表、資料行、索引和其他資料庫元素間也都會定義所有關聯性。	目，但其他非索引鍵屬性則沒有類似的限制條件。DynamoDB 可以管理結構化或半結構化資料，包括 JSON 文件。
資料存取	SQL 是存放和擷取資料的標準。關聯式資料庫提供豐富的工具組，可簡化資料庫驅動之應用程式的開發，但這些工具全都使用 SQL。	您可以使用 AWS Management Console、AWS CLI、或 NoSQL WorkBench 來處理 DynamoDB 並執行隨機操作任務。SQL 相容查詢語言 PartiQL 讓您能在 DynamoDB 中選取、插入、更新和刪除資料。應用程式可以使用 AWS 軟體開發套件 (SDK)，透過物件型、以文件為中心或低層級的介面與 DynamoDB 搭配使用。
效能	關聯式資料庫經儲存最佳化，因此效能一般取決於磁碟子系統。開發人員和資料庫管理員必須最佳化查詢、索引及資料表結構，才能達到尖峰效能。	DynamoDB 經運算最佳化，因此效能主要是基礎硬體和網路延遲的功能。作為受管服務，DynamoDB 會將您和您的應用程式與這些實作細節隔開，讓您可專心設計及建置穩固且高效能的應用程式。

特性	關聯式資料庫管理系統 (RDBMS)	Amazon DynamoDB
擴展	使用更快的硬體可以更容易地擴展。資料庫資料表也能在分散式系統中橫跨多部主機，但這需要額外投資。關聯式資料庫在檔案的數目和大小方面皆具有大小上限，而對最大擴展能力有所限制。	DynamoDB 專為使用分散式硬體叢集水平擴展而設計。這項設計可讓輸送量增加，卻不會增加延遲。客戶可指定其輸送量需求，DynamoDB 便會配置充足的資源以符合這些需求。每個資料表的項目數目沒有上限，該資料表的總大小也沒有上限。

資料庫的特性

在應用程式可以存取資料庫之前，必須先進行驗證，以確保允許應用程式使用資料庫。必須對其進行授權，以便應用程式執行其具有許可的動作。

下列圖表說明用戶端與關聯式資料庫及 Amazon DynamoDB 的互動。



下表提供用戶端互動任務的詳細資訊。

特性	關聯式資料庫管理系統 (RDBMS)	Amazon DynamoDB
存取資料庫的工具	多數關聯式資料庫皆提供命令列界面 (CLI)，讓您可輸入隨機操作 SQL 陳述式並立即查看結果。	在大多數的案例中，您會撰寫應用程式程式碼。您也可以使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或 NoSQL 工作台將隨機操作請求傳送至 DynamoDB 並檢視結果。SQL 相容查詢語言 PartiQL 讓您能在 DynamoDB 中選取、插入、更新和刪除資料。

特性	關聯式資料庫管理系統 (RDBMS)	Amazon DynamoDB
連線到資料庫	應用程式會建立並維持與資料庫的網路連線。當應用程式完成時，它會終止連線。	DynamoDB 是一項 Web 服務，與該服務之間的互動是無狀態的。應用程式不需要維持持久性網路連線。而是使用 HTTP(S) 請求及回應與 DynamoDB 互動。
身分驗證	應用程式在經過驗證前無法連線到資料庫。RDBMS 可自行執行身分驗證，也可將此任務交由主機作業系統或目錄服務執行。	每一個發送至 DynamoDB 的請求都必須附有密碼編譯簽章，此簽章會驗證該特定請求。AWS SDK 提供了創建簽名和簽名請求所需的所有邏輯。如需詳細資訊 AWS ，請參閱 AWS 一般參考 。
授權	應用程式只能執行獲得授權的動作。資料庫管理員或應用程式擁有者可使用 SQL GRANT 及 REVOKE 陳述式控制對資料庫物件 (例如資料表)、資料 (例如資料表中的資料列) 的存取，或控制發行特定 SQL 陳述式的能力。	在 DynamoDB 中，授權由 AWS Identity and Access Management (IAM) 處理。您可以撰寫 IAM 政策來授予 DynamoDB 資源 (例如資料表) 相關許可，然後允許使用者和角色使用該政策。IAM 也針對 DynamoDB 資料表中的個別資料項目提供精細的存取控制。如需詳細資訊，請參閱 Amazon DynamoDB 的 Identity and Access Management 。

特性	關聯式資料庫管理系統 (RDBMS)	Amazon DynamoDB
傳送請求	應用程式會針對想要執行的每項資料庫操作各發出一個 SQL 陳述式。在收到 SQL 陳述式後，RDBMS 會檢查其語法，建立執行操作的計畫，然後執行計畫。	應用程式會向 DynamoDB 傳送 HTTP(S) 請求。該請求包含要執行之 DynamoDB 操作的名稱及參數。DynamoDB 會立即執行請求。
接收回應	RDBMS 會傳回 SQL 陳述式的結果。若發生錯誤，RDBMS 會傳回錯誤狀態及訊息。	DynamoDB 會傳回含有操作結果的 HTTP(S) 回應。若發生錯誤，DynamoDB 會傳回 HTTP 錯誤狀態及訊息。

建立資料表

資料表是關聯式資料庫及 Amazon DynamoDB 中的基本資料結構。關聯式資料庫管理系統 (RDBMS) 需要您在建立資料表的同時定義其結構描述。反之，DynamoDB 資料表不具結構描述；除了主索引鍵之外，您不需要在建立資料表時定義任何額外屬性或資料類型。

下一節將比較使用 SQL 建立資料表的方式，以及使用 DynamoDB 建立資料表的方式。

主題

- [使用 SQL 建立資料表](#)
- [使用 DynamoDB 建立資料表](#)

使用 SQL 建立資料表

在 SQL 中，您會使用 CREATE TABLE 陳述式建立資料表，如下列範例所示。

```
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)
);
```

此表的主鍵由藝術家和 SongTitle。

您必須定義資料表所有資料行、資料類型及主索引鍵 (若有需要，您之後可以使用 ALTER TABLE 陳述式變更這些定義)。

許多 SQL 實作可讓您為資料表定義儲存規格，做為 CREATE TABLE 陳述式的一部分。除非您另外指示，否則會以預設儲存設定建立資料表。在生產環境中，資料庫管理員可協助判斷最佳儲存參數。

使用 DynamoDB 建立資料表

使用 CreateTable 操作建立佈建模式資料表，並依下方所示指定參數：

```
{
  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
  }
}
```

```
}
```

此表的主鍵由藝術家 (分區鍵) 和 SongTitle (排序鍵) 組成。

您必須提供下列參數給 CreateTable :

- TableName : 資料表的名稱。
- KeySchema : 用於主索引鍵的屬性。如需詳細資訊, 請參閱 [資料表、項目與屬性](#) 及 [主索引鍵](#)。
- AttributeDefinitions : 索引鍵結構描述屬性的資料類型。
- ProvisionedThroughput (for provisioned tables) : 此資料表所需的每秒讀取及寫入數。DynamoDB 會保留足夠的儲存和系統資源, 以便始終能滿足您的輸送量需求。若有需要, 您之後可以使用 UpdateTable 操作進行變更。因為儲存配置全由 DynamoDB 管理, 因此您不需要指定資料表的儲存需求。

Note

如需使用 CreateTable 的程式碼範例, 請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

取得資料表的資訊

您可以確認資料表是否已按照您的規格建立。關聯式資料庫中會顯示資料表的所有結構描述。Amazon DynamoDB 資料表不具結構描述, 因此只會顯示主索引鍵屬性。

主題

- [取得 SQL 資料表的相關資訊](#)
- [取得 DynamoDB 中資料表的相關資訊](#)

取得 SQL 資料表的相關資訊

多數關聯式資料庫管理系統 (RDBMS) 可讓您說明資料表的結構, 即欄、資料類型、主索引鍵定義等。SQL 沒有此操作的標準做法。但是, 許多資料庫系統皆會提供 DESCRIBE 命令。以下是 MySQL 的範例。

```
DESCRIBE Music;
```

這會傳回您資料表的結構, 內含所有資料行名稱、資料類型及大小。

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	

此表的主鍵由藝術家和 SongTitle。

取得 DynamoDB 中資料表的相關資訊

DynamoDB 具有類似的 DescribeTable 操作。唯一的參數是資料表名稱。

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

DescribeTable 的回覆如下所示。

```
{
  "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 也會傳回有關資料表索引、佈建輸送量設定、約略的項目計數及其他中繼資料的資訊。

將資料寫入資料表

關聯式資料庫資料表含有資料的資料列。資料列由欄所組成。Amazon DynamoDB 資料表包含項目。項目由屬性組成。

本節說明如何將一個資料列 (或項目) 寫入資料表。

主題

- [將資料寫入 SQL 資料表](#)
- [將資料寫入 DynamoDB 中的資料表](#)

將資料寫入 SQL 資料表

關聯式資料庫中的資料表為二維資料結構，由資料列和資料行組成。某些資料庫管理系統也支援半結構化資料，通常使用原生 JSON 或 XML 資料類型。但是，實作細節會因廠商而有所不同。

在 SQL 中，您會使用 INSERT 陳述式新增列至資料表。

```
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}'
);
```

此表的主鍵由藝術家和 SongTitle. 您必須指定這些資料行的值。

Note

此範例使用 Tags 資料行存放 Music 資料表中歌曲的半結構化資料。Tags 資料行已定義為 TEXT 類型，可在 MySQL 中存放多達 65535 個字元。

將資料寫入 DynamoDB 中的資料表

在 Amazon DynamoDB 中，您可以使用 DynamoDB API 或 [PartiQL](#) (SQL 相容查詢語言) 將項目新增至資料表。

DynamoDB API

借助 DynamoDB API，您可以使用 PutItem 操作將項目新增至資料表。


```
{
  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
    }
  }
}
```

此表的主鍵由藝術家和 SongTitle。您必須指定這些屬性的值。

以下為此 PutItem 範例的一些重要須知：

- DynamoDB 會使用 JSON 提供文件的原生支援。這讓 DynamoDB 適合存放半結構化資料，例如 Tags。您也可以從 JSON 文件擷取及運用資料。

- 音樂表沒有任何預先定義的屬性，除了主鍵 (演出者和 SongTitle)。
- 多數 SQL 資料庫都以交易為導向。當您發出 INSERT 陳述式時，資料都不會永久修改，直到您發出 COMMIT 陳述式為止。若使用 Amazon DynamoDB，當 DynamoDB 以 HTTP 200 狀態代碼 (OK) 回覆時，PutItem 操作會永久生效。

 Note

如需使用 PutItem 的程式碼範例，請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

下列是一些其他的 PutItem 範例。

```
{
  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

除了 PutItem 之外，DynamoDB 還支援 BatchWriteItem 操作，可讓您同時寫入多個項目。

PartiQL for DynamoDB

借助 PartiQL，您可以使用 PartiQL Insert 陳述式，使用 ExecuteStatement 操作將項目新增至資料表，

```

INSERT into Music value {
  'Artist': 'No One You Know',

```

```
'SongTitle': 'Call Me Today',  
'AlbumTitle': 'Somewhat Famous',  
'Year' : '2015',  
'Genre' : 'Acme'  
}
```

此表的主鍵由藝術家和 SongTitle. 您必須指定這些屬性的值。

Note

如需使用 Insert 和 ExecuteStatement 的程式碼範例，請參閱 [適用於 DynamoDB 的 PartiQL Insert 陳述式](#)。

從資料表讀取資料時 SQL 與 DynamoDB 的主要差異

使用 SQL，您可用 SELECT 陳述式從資料表擷取一或多個資料列。並可用 WHERE 子句決定傳回給您的資料。

這與使用 Amazon DynamoDB 不同，其提供下列操作來讀取資料：

- ExecuteStatement 會從資料表檢索單一或多個項目。BatchExecuteStatement 以單一操作從不同資料表檢索多個項目。這些操作都使用 [PartiQL](#)，這是一種與 SQL 相容的查詢語言。
- GetItem：從資料表擷取單一項目。因為可供直接存取項目的實體位置，所以此為讀取單一項目最有效率的方式。(DynamoDB 還會提供 BatchGetItem 操作，允許您在單一操作中執行多達 100 個 GetItem 呼叫。)
- Query：擷取具有特定分割區索引鍵的所有項目。在那些項目中，您可以將條件套用至排序索引鍵，並只擷取部分資料。Query 供您快速且有效率地存取存放資料的分割區。(如需詳細資訊，請參閱 [分割區與資料分佈](#)。)
- Scan：擷取指定資料表中的所有項目。(此操作不應用於大型資料表，因為會使用大量系統資源)。

Note

使用關聯式資料庫，您可用 SELECT 陳述式聯結多個資料表的資料，然後傳回結果。聯結是關聯式模型的基礎。若要確保聯結可有效率地執行，資料庫及其應用程式應持續調整其效能。DynamoDB 是不支援資料表聯結的非關聯式 NoSQL 資料庫。相反地，應用程式會一次從一個資料表讀取資料。

下列章節說明讀取資料的不同使用案例，以及如何使用關聯式資料庫和 DynamoDB 執行這些任務。

主題

- [使用項目的主索引鍵讀取項目](#)
- [查詢資料表](#)
- [掃描資料表](#)

使用項目的主索引鍵讀取項目

資料庫常見的一種存取模式是從資料表讀取單一項目。您必須指定您想要讀取之項目的主索引鍵。

主題

- [在 SQL 中使用項目的主索引鍵讀取項目](#)
- [在 DynamoDB 中使用項目的主索引鍵讀取項目](#)

在 SQL 中使用項目的主索引鍵讀取項目

在 SQL 中，您會使用 SELECT 陳述式從資料表擷取資料。您可以在結果中請求一或多個資料行 (或使用 * 運算子請求所有資料行)。WHERE 子句則可決定要傳回的資料列。

下列為從 Music 資料表擷取單一資料列的 SELECT 陳述式。WHERE 子句會指定主索引鍵值。

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

您可以修改此查詢，只擷取部分資料行。

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

請注意，此表格的主索引鍵由藝術家和 SongTitle。

在 DynamoDB 中使用項目的主索引鍵讀取項目

在 Amazon DynamoDB 中，您可以使用 DynamoDB API 或 [PartiQL](#) (SQL 相容查詢語言) 讀取資料表中的項目。

DynamoDB API

借助 DynamoDB API，您可以使用 PutItem 操作將項目新增至資料表。

DynamoDB 提供 GetItem 操作，可依主索引鍵擷取項目。GetItem 的效率極高，因為它可供您直接存取項目的實體位置。(如需詳細資訊，請參閱 [分割區與資料分佈](#)。)

根據預設，GetItem 會傳回整個項目及其所有屬性。

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

您可以新增 ProjectionExpression 參數，以只傳回某些屬性。

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

請注意，此表格的主索引鍵由藝術家和 SongTitle。

DynamoDB GetItem 操作非常有效率。它使用主索引鍵值判斷所指項目的確切儲存位置，然後直接從該位置擷取項目。在使用主索引鍵值擷取項目的案例中，SQL SELECT 陳述式一樣具有效率。

SQL SELECT 陳述式可支援多種查詢和資料表掃描。DynamoDB 使用其 Query 和 Scan 操作提供類似功能，這些操作會在 [查詢資料表](#) 和 [掃描資料表](#) 中說明。

SQL SELECT 陳述式可執行資料表聯結，可讓您同時從多個資料表擷取資料。在資料庫資料表標準化且資料表間的關聯性明確時，聯結最為有效。但是，若您在單一 SELECT 陳述式中聯結太多資料表，應用程式的效能可能會受到影響。您可以使用資料庫複寫、具體化檢視，或查詢重寫來因應這項問題。

DynamoDB 是非關聯式資料庫，不支援資料表聯結。若您正在將現有應用程式從關聯式資料庫遷移至 DynamoDB，必須將您的資料模型去標準化，才不需聯結。

Note

如需使用 GetItem 的程式碼範例，請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

PartiQL for DynamoDB

借助 PartiQL，您可以使用 PartiQL Select 陳述式，使用 ExecuteStatement 操作讀取資料表的項目。

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

請注意，此表格的主索引鍵由藝術家和 SongTitle。

Note

PartiQL Select 陳述式也可以用來查詢或掃描 DynamoDB 資料表

如需使用 Select 和 ExecuteStatement 的程式碼範例，請參閱 [適用於 DynamoDB 的 PartiQL Select 陳述式](#)。

查詢資料表

資料庫常見的另一種存取模式是根據您的查詢條件，從資料表讀取多個項目。

主題

- [使用 SQL 查詢資料表](#)
- [在 DynamoDB 中查詢資料表](#)

使用 SQL 查詢資料表

使用 SQL 時，SELECT 陳述式可讓您查詢索引鍵資料行、非索引鍵資料行或任何組合。WHERE 子句則可決定要傳回的資料列，如下列範例所示。


```
/* 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;
```

請注意，此表格的主索引鍵由藝術家和 SongTitle。

在 DynamoDB 中查詢資料表

在 Amazon DynamoDB 中，您可以使用 DynamoDB API 或 [PartiQL](#) (SQL 相容查詢語言) 查詢資料表中的項目。

DynamoDB API

若使用 Amazon DynamoDB，您可以使用 Query 操作以類似方式擷取資料。Query 操作可供您快速且有效率地存取存放資料的實體位置。如需詳細資訊，請參閱 [分割區與資料分佈](#)。

您可以對任何資料表或次要索引使用 Query。您必須為分割區索引鍵值指定相等條件，並可選擇性地為已定義的排序索引鍵屬性提供另一個條件。

KeyConditionExpression 參數指定您想要查詢的索引鍵值。您可以使用選用的 FilterExpression，在結果傳回之前從中移除特定的項目。

在 DynamoDB 中，您必須使用 `ExpressionAttributeValues` 作為表達式參數 (例如 `KeyConditionExpression` 及 `FilterExpression`) 中的預留位置。這與在關聯式資料庫中使用繫結變數類似，您會在執行時間將實際值代入 `SELECT` 陳述式。

請注意，此表格的主索引鍵由藝術家和 `SongTitle`。

下列是一些 DynamoDB Query 範例。

```
// 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

如需使用 Query 的程式碼範例，請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

PartiQL for DynamoDB

若使用 PartiQL，您可以藉由使用 ExecuteStatement 操作和 Select 陳述式對分割區索引鍵進行查詢。

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know'
```

以此方式使用 SELECT 陳述式會傳回所有與此特定 Artist 相關聯的歌曲。

如需使用 Select 和 ExecuteStatement 的程式碼範例，請參閱 [適用於 DynamoDB 的 PartiQL Select 陳述式](#)。

掃描資料表

在 SQL 中，沒有 SELECT 子句的 WHERE 陳述式會傳回資料表中的每個資料列。在 Amazon DynamoDB 中，Scan 操作會執行同樣的操作。在這兩種案例中，您可以擷取所有項目或部分項目。

無論是使用 SQL 或 NoSQL 資料庫，建議您謹慎使用掃描，因為掃描會使用大量系統資源。有時候適合掃描 (例如掃描小型資料表) 或必須掃描 (例如大量匯出資料)。但一般來說，您應將應用程式設計為避免執行掃描。如需詳細資訊，請參閱 [DynamoDB 中的查詢操作](#)。

Note

執行大量匯出也會為每個分割區建立至少 1 個檔案。每個檔案的所有項目都來自該特定分割區的雜湊金鑰空間。

主題

- [使用 SQL 掃描資料表](#)
- [掃描 DynamoDB 中的資料表](#)

使用 SQL 掃描資料表

使用 SQL 時，您可以使用不指定 WHERE 子句的 SELECT 陳述式來掃描資料表，並擷取其中的所有資料。您可以在結果中請求一或多個資料行。或者，您可以使用萬用字元 (*) 請求所有資料行。

以下為使用 SELECT 陳述式的範例。

```
/* 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;
```

掃描 DynamoDB 中的資料表

在 Amazon DynamoDB 中，您可以使用 DynamoDB API 或 [PartiQL](#) (SQL 相容查詢語言) 對資料表執行掃描。

DynamoDB API

借助 DynamoDB API，可使用 Scan 操作，存取資料表中的每個項目或次要索引，以傳回一或多個項目和項目屬性。

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

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

Scan 操作也提供了 FilterExpression 參數，您可以用它來捨棄不想要顯示在結果中的項目。FilterExpression 會在掃描完後並在結果傳回給您前套用。(此動作不建議與大型資料表一起使用。即使只傳回少量的符合項目，您仍須為整個 Scan 支付費用。)

Note

如需使用 Scan 的程式碼範例，請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

PartiQL for DynamoDB

借助 PartiQL，您可以藉由使用 ExecuteStatement 操作並使用 Select 陳述式傳回資料表的所有內容。

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

請注意，此陳述式會傳回「Music」資料表中的所有項目。

如需使用 Select 和 ExecuteStatement 的程式碼範例，請參閱 [適用於 DynamoDB 的 PartiQL Select 陳述式](#)。

管理索引

索引提供您替代的查詢模式，可加速查詢。本節會比較及比對 SQL 及 Amazon DynamoDB 中索引的建立及使用。

無論您是使用關聯式資料庫或 DynamoDB，都應明智地建立索引。每當寫入資料表時，資料表的所有索引都必須更新。在大型資料表的大量寫入環境中，這可能會使用大量系統資源。在唯讀或大部分讀取的環境中，這不算是問題。但是，您應確認應用程式確實使用了索引，而非只是占用空間。

主題

- [建立索引](#)
- [查詢及掃描索引](#)

建立索引

比較 SQL 中的 CREATE INDEX 陳述式及 Amazon DynamoDB 中的 UpdateTable 操作。

主題

- [使用 SQL 建立索引](#)

- [在 DynamoDB 中建立索引](#)

使用 SQL 建立索引

在關聯式資料庫中，索引是一種資料結構，可讓您快速查詢資料表中的不同資料行。您可以使用 CREATE INDEX SQL 陳述式對現有的資料表新增索引，並指定要編製索引的資料行。在索引建立後，您可以如常地查詢資料表中的資料，但資料庫現在已可使用索引在快速尋找資料表中的指定資料列，而不用掃描整個資料表。

在您建立索引後，資料庫會為您維護。每當您修改資料表中的資料時，索引就會自動修改，以反映資料表的變更。

在 MySQL 中，您會建立如下的索引。

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

在 DynamoDB 中建立索引

在 DynamoDB 中，您可以建立及使用次要索引來達到類似目的。

DynamoDB 中的索引與關聯式資料庫中的索引不同。在您建立次要索引時，必須指定其索引鍵屬性，即一個分割區索引鍵和一個排序索引鍵。在建立次要索引後，您便可以 Query 或 Scan 該索引，正如您對資料表所做的那樣。DynamoDB 沒有查詢最佳化程式，因此只有在您 Query 或 Scan 次要索引時才會使用此類索引。

DynamoDB 支援兩種不同的索引：

- 全域次要索引：索引的主索引鍵可以是資料表中任兩個屬性。
- 本機次要索引：索引的分割區索引鍵必須與資料表的分割區索引鍵相同。但是，排序索引鍵可以是其他任何屬性。

DynamoDB 會確認次要索引中的資料最終與其資料表一致。您可以對資料表或本機次要索引請求高度一致的 Query 或 Scan 操作。但是，全域次要索引只支援最終一致性。

您可以使用 UpdateTable 操作並指定 GlobalSecondaryIndexUpdates，對現有的資料表新增全域次要索引。

```
{
```

```

    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
          }
        }
      }
    ]
  }
}

```

您必須提供下列參數給 UpdateTable：

- **TableName**：會與索引相關聯的資料表。
- **AttributeDefinitions**：索引之索引鍵結構描述屬性的資料類型。
- **GlobalSecondaryIndexUpdates**：您想要建立之索引的相關詳細資訊：
 - **IndexName**：索引的名稱。
 - **KeySchema**：用於索引主索引鍵的屬性。
 - **Projection**：資料表中複製到索引的屬性。在此案例中，ALL 表示會複製所有屬性。
 - **ProvisionedThroughput (for provisioned tables)**：此索引所需的每秒讀取及寫入數。(這和資料表的佈建輸送量設定是分開的。)

部分此操作會涉及將資料表中的資料回填到新的索引。在回填過程中，資料表仍可使用。但是，在索引的 Backfilling 屬性從 true 變更為 false 之前，索引都尚未就緒。您可以使用 DescribeTable 操作來檢視此屬性。

Note

如需使用 UpdateTable 的程式碼範例，請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

查詢及掃描索引

比較在 SQL 中使用 SELECT 陳述式及在 Amazon DynamoDB 中使用 Query 及 Scan 操作查詢及掃描索引。

主題

- [在 SQL 中查詢及掃描索引](#)
- [在 DynamoDB 中查詢及掃描索引](#)

在 SQL 中查詢及掃描索引

在關聯式資料庫中，您不會直接使用索引。而是透過發出 SELECT 陳述式查詢資料表，然後查詢最佳化器就可使用任何索引。

查詢最佳化器是一項關聯式資料庫管理系統 (RDBMS) 元件，可評估可用的索引，並判斷是否可用其加速查詢。若索引可用來加速查詢，RDBMS 便會先存取索引，然後用它尋找資料表中的資料。

以下是一些可用 GenreAndPriceIndex 來改善效能的 SQL 陳述式。我們假設 Music 資料表具有足夠的資料，可讓查詢最佳化器決定使用此索引，而非單純地掃描整個資料表。

```
/* 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;
```


在 DynamoDB 中查詢及掃描索引

在 DynamoDB 中，您可以直接在索引上執行 Query 和 Scan 操作，與您在資料表上的操作方式相同。您可以使用 DynamoDB API 或 [PartiQL](#) (SQL 相容查詢語言) 來查詢或掃描索引。您必須同時指定 TableName 和 IndexName。

以下是關於 GenreAndPriceIndex 在 DynamoDB 中的一些查詢。(此索引的索引鍵結構描述包含 Genre 和 Price)。

DynamoDB API

```
// All of the rock songs

{
  TableName: "Music",
  IndexName: "GenreAndPriceIndex",
  KeyConditionExpression: "Genre = :genre",
  ExpressionAttributeValues: {
    ":genre": "Rock"
  },
};
```

此範例使用 ProjectionExpression 指出您只想要在結果中顯示部分屬性，而非所有屬性。

```
// 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"
};
```

以下是對的掃描 GenreAndPriceIndex。

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

{
```

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

PartiQL for DynamoDB

使用 PartiQL，您可以使用 PartiQL Select 陳述式對索引執行查詢和掃描。

```
// 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
```

以下是對的掃描GenreAndPriceIndex。

```
// Return all of the data in the index  
  
SELECT *  
FROM Music.GenreAndPriceIndex
```

Note

如需使用 Select 的程式碼範例，請參閱 [適用於 DynamoDB 的 PartiQL Select 陳述式](#)。

修改資料表中的資料

SQL 語言會提供 UPDATE 陳述式來修改資料。Amazon DynamoDB 使用 UpdateItem 操作來完成類似的任務。

主題

- [修改 SQL 資料表中的資料](#)

- [修改 DynamoDB 中資料表的資料](#)

修改 SQL 資料表中的資料

在 SQL 中，您會使用 UPDATE 陳述式修改一或多列。SET 子句可為一或多個資料行指定新的值，WHERE 子句則可決定要修改的資料列。以下是範例。

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

若沒有任何資料列符合 WHERE 子句，UPDATE 陳述式便沒有任何效果。

修改 DynamoDB 中資料表的資料

在 DynamoDB 中，您可以使用 DynamoDB API 或 [PartiQL](#) (SQL 相容查詢語言) 修改單一項目。如果您想要修改多個項目，則必須使用多個操作。

DynamoDB API

若使用 DynamoDB API，您可以使用 UpdateItem 操作修改單一項目。

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

您必須指定要修改之項目的 Key 屬性，以及 UpdateExpression 以指定屬性值。UpdateItem 的運作方式類似「upsert」操作。如果項目存在資料表中，則會進行更新，如果不存在，則會新增(插入)新項目。

UpdateItem 支援條件式寫入，即只有在特定 ConditionExpression 評估為 true 時，操作才會成功。例如，若歌曲的價格未大於或等於 2.00，下列 UpdateItem 操作就不會執行更新。

```
{
  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 也支援原子計數器，或可增加或減少的 Number 類型屬性。原子計數器與 SQL 資料庫中的順序產生器、身分欄位或自動遞增欄位在許多方面都非常類似。

下列是 UpdateItem 操作的範例，它會初始化新屬性 (Plays) 來追蹤歌曲的播放次數。

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

ReturnValues 參數會設為 UPDATED_NEW，這會傳回任何更新屬性的新值。在此案例中，它會傳回 0 (零)。

只要有人播放此歌曲，我們就可以使用下列 UpdateItem 操作將 Plays 增加 1。

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

```
    },
    UpdateExpression: "SET Plays = Plays + :incr",
    ExpressionAttributeValues: {
        ":incr": 1
    },
    ReturnValues: "UPDATED_NEW"
}
```

Note

如需使用 UpdateItem 的程式碼範例，請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

PartiQL for DynamoDB

借助 PartiQL，您可以使用 PartiQL Update 陳述式，使用 ExecuteStatement 操作修改資料表中的項目。

此表的主鍵由藝術家和 SongTitle。您必須指定這些屬性的值。

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

您也可以一次修改多個欄位，如以下範例所示。

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

Update 也支援原子計數器，或可增加或減少的 Number 類型屬性。原子計數器與 SQL 資料庫中的順序產生器、身分欄位或自動遞增欄位在許多方面都非常類似。

以下為 Update 陳述式的示例，用以初始化新的屬性 (Plays) 來追蹤歌曲的播放次數。

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

當某人播放此歌曲時，我們即可使用下列 Update 陳述式將 Plays 增加 1。

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

Note

如需使用 Update 和 ExecuteStatement 的程式碼範例，請參閱 [適用於 DynamoDB 的 PartiQL Update 陳述式](#)。

從資料表刪除資料

在 SQL 中，DELETE 陳述式會從資料表中移除一或多個資料列。Amazon DynamoDB 會使用 DeleteItem 操作一次刪除一個項目。

主題

- [從 SQL 資料表刪除資料](#)
- [從 DynamoDB 中的資料表刪除資料](#)

從 SQL 資料表刪除資料

在 SQL 中，您可使用 DELETE 陳述式刪除一或多個資料列。WHERE 子句則可決定您想要修改的資料列。以下是範例。

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

您可修改 WHERE 子句來刪除多個資料列。例如，您可以刪除特定演出者的所有歌曲，如下所示。

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

Note

若您省略 WHERE 子句，則資料庫會嘗試刪除資料表中的所有資料列。

從 DynamoDB 中的資料表刪除資料

在 DynamoDB 中，您可以使用 DynamoDB API 或 [PartiQL](#) (SQL 相容查詢語言) 刪除單一項目。如果您想要修改多個項目，則必須使用多個操作。

DynamoDB API

若使用 DynamoDB API，您會使用 `DeleteItem` 操作從資料表中刪除資料，一次刪除一個項目。您必須指定項目的主索引鍵值。

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

Note

除了 `DeleteItem` 之外，Amazon DynamoDB 還支援 `BatchWriteItem` 操作，可讓您同時刪除多個項目。

`DeleteItem` 支援條件式寫入，即只有在特定 `ConditionExpression` 評估為 `true` 時，操作才會成功。例如，下列 `DeleteItem` 作業只會在項目具有 `RecordLabel` 屬性時刪除該項目。

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

Note

如需使用 `DeleteItem` 的程式碼範例，請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

PartiQL for DynamoDB

借助 PartiQL，您可以使用 Delete 陳述式透過 ExecuteStatement 操作從資料表刪除資料，一次刪除一個項目。您必須指定項目的主索引鍵值。

此表的主鍵由藝術家和 SongTitle。您必須指定這些屬性的值。

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

您也可以為操作指定其他條件。以下 DELETE 操作僅刪除具有超過 11 個 Awards (獎項) 的項目。

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

Note

如需使用 DELETE 和 ExecuteStatement 的程式碼範例，請參閱 [適用於 DynamoDB 的 PartiQL Delete 陳述式](#)。

移除資料表

在 SQL 中，您可使用 DROP TABLE 陳述式移除資料表。在 Amazon DynamoDB 中，您可以使用 DeleteTable 操作。

主題

- [移除 SQL 資料表](#)
- [移除 DynamoDB 中的資料表](#)

移除 SQL 資料表

當您不再需要某個資料表，並想要永久將它捨棄時，您會在 SQL 中使用 DROP TABLE 陳述式。

```
DROP TABLE Music;
```

資料表捨棄後，便無法復原 (某些關聯式資料庫的確可讓您復原 DROP TABLE 操作，但這屬於廠商專屬的功能，並未廣泛實作)。

移除 DynamoDB 中的資料表

在 DynamoDB 中，DeleteTable 是類似的操作。在下列範例中，資料表會永久刪除。

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

Note

如需使用 DeleteTable 的程式碼範例，請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

其他 Amazon DynamoDB 資源

您可以使用下列其他資源，協助您了解及使用 DynamoDB。

主題

- [寫程式和視覺化工具](#)
- [規範性指引文章](#)
- [知識中心文章](#)
- [部落格文章、儲存庫和指南](#)
- [資料建模和設計模式簡報](#)
- [培訓課程](#)

寫程式和視覺化工具

您可以使用下列程式撰寫和視覺化工具來搭配 DynamoDB 使用：

- [適用於 Amazon DynamoDB 的 NoSQL Workbench](#)：一種統一的視覺化工具，可協助您設計、建立、查詢和管理 DynamoDB 資料表。其提供資料建模、資料視覺化和查詢開發功能。
- [Dynobase](#)：此桌面工具可讓您輕鬆查看和使用 DynamoDB 資料表、建立應用程式程式碼，以及透過即時驗證來編輯紀錄。
- [DynamoDB 工具箱](#) — 傑里米·達利 (Jeremy Daly) 提供有用的實用工具來處理數據建模和 Node.js 的項目。andJavaScript
- [DynamoDB Streams 處理器](#)：此簡易工具可以簡化使用 [DynamoDB Streams](#) 的作業。

規範性指引文章

AWS 規範性指引提供經過時間考驗的策略、指南和模式，協助您加速專案流程。這些資源是由 AWS 技術專家和全球合作夥伴社群 AWS，根據他們多年來協助客戶達成業務目標的經驗所開發。

資料建模與移轉

- [階層式資料模型](#)
- [使用 DynamoDB 建立資料模型](#)
- [使用 AWS DMS 將 Oracle 資料庫遷移到 DynamoDB。](#)

全域表

- [使用 Amazon DynamoDB 全域表](#)

無伺服器

- [使用 AWS Step Functions 實作無伺服器傳奇模式](#)

SaaS 架構

- [透過單一控制平面管理多個 SaaS 產品的租用戶](#)
- [在 SaaS 架構中使用 C# 和 AWS CDK 進行筒倉模型的租用戶上線](#)

資料保護與資料移動

- [設定對 Amazon DynamoDB 的跨帳戶存取權](#)
- [適用於 DynamoDB 的完整資料表複製選項](#)
- [AWS 的資料庫災難復原策略](#)

Miscellaneous (其他)

- [協助在 DynamoDB 中強制執行標記](#)

規範性指引影片逐步解說

- [使用無伺服器架構建立資料管道](#)
- [Novartis - Buying Engine : 人工智慧採購入口網站](#)
- [Veritiv : 啟用見解以預測 AWS 資料湖上的銷售需求](#)
- [mimik : 混合邊緣雲利用 AWS 來支援邊緣微服務網格](#)
- [使用 Amazon DynamoDB 變更資料擷取](#)

如需 DynamoDB 的其他規範性指引文章和影片，請參閱[規範性指引](#)。

知識中心文章

AWS 知識中心的文章和影片涵蓋了我們從 AWS 客戶收到的最常見問題和要求。以下是關於與 DynamoDB 相關之特定工作的最新知識中心文章：

成本最佳化

- [如何使用 Amazon DynamoDB 最佳化成本？](#)

限流和延遲

- [平均延遲正常時，為什麼我的 DynamoDB 最大延遲指標很高？](#)
- [為什麼我的 DynamoDB 資料表會被限流？](#)
- [為什麼我的隨選 DynamoDB 資料表會被限流？](#)

分頁

- [如何在 DynamoDB 中實作分頁](#)

交易

- [為什麼我的 TransactWriteItems API 調用在 DynamoDB 中失敗](#)

疑難排解

- [如何解決 DynamoDB 自動擴展的問題？](#)
- [如何疑難排解 DynamoDB 中的 HTTP 4XX 錯誤](#)

如需 DynamoDB 的其他文章和影片，請參閱[知識中心文章](#)。

部落格文章、儲存庫和指南

除了 [DynamoDB 開發人員指南](#) 之外，還有許多有用的資源可協助使用 DynamoDB。以下是使用 DynamoDB 的精選部落格文章、儲存庫和指南：

- AWS：[DynamoDB 程式碼範例](#) 在各種 AWS 開發套件語言：[Node.js](#)、[Java](#)、[Python](#)、[.NET](#)、[Go](#) 和 [Rust](#)。
- [DynamoDB 手冊](#) — [Alex](#) 提供的綜合指南，教導使 DeBrie 用 DynamoDB 進行資料建模的策略導向方法。
- [DynamoDB 指南](#) — [Alex](#) 提供的開放式指南 DeBrie，會逐步介紹 DynamoDB NoSQL 資料庫的基本概念和進階功能。
- [如何透過 20 個簡單步驟從 RDBMS 轉換至 DynamoDB](#)：[Jeremy Daly](#) 提供的學習資料建模實用步驟清單。
- [DynamoDB JavaScript DocumentClient 備忘單](#) — 可協助您開始在 Node.js 或環境中使用 DynamoDB 建置應用程式的備忘單。JavaScript
- [DynamoDB 核心概念影片](#)：此播放清單涵蓋許多 DynamoDB 的核心概念。

資料建模和設計模式簡報

您可以使用以下關於資料建模和設計模式的資源，以協助您充分利用 DynamoDB：

- [AWS re:Invent 2019：採用 DynamoDB 建立資料模型](#)
 - [Alex](#) 的演講 DeBrie 可協助您開始使用 DynamoDB 資料建模的原則。
- [AWS re:Invent 2020：採用 DynamoDB 的資料建模 \(第 1 部分\)](#)
- [AWS re:Invent 2020：採用 DynamoDB 的資料建模 \(第 2 部分\)](#)
- [AWS re:Invent 2017：進階的設計模式](#)
- [AWS re:Invent 2018：進階的設計模式](#)
- [AWS re:Invent 2019：進階的設計模式](#)
 - [Jeremy Daly](#) 分享了他在本次會議中提到的 [12 個重要結論](#)。
- [AWS re:Invent 2020：DynamoDB 進階設計模式 \(第 1 部分\)](#)
- [AWS re:Invent 2020：DynamoDB 進階設計模式 \(第 2 部分\)](#)
- [Twitch 上的 DynamoDB 辦公時間](#)

Note

每個會議涵蓋不同的使用案例和範例。

培訓課程

有許多不同的培訓課程和教育選項，可讓您進一步瞭解 DynamoDB。以下是一些目前的範例：

- [使用 Amazon DynamoDB 進行開發](#)：由 AWS 設計，讓您透過適用於 Amazon DynamoDB 的資料模型，從初學者成為開發真實世界應用程式的專家。
- [DynamoDB 深度課程](#)：Cloud Guru 課程。
- [Amazon DynamoDB：建置 NoSQL 資料庫驅動型應用程式](#)：AWS 培訓和認證團隊主持的 edX 線上課程。

讀取和寫入

DynamoDB 讀取和寫入是指從表格擷取資料 (讀取) 以及插入、更新或刪除資料表 (寫入) 中的資料的作業。當您使用 DynamoDB 時，務必瞭解讀取和寫入的概念，因為這些概念會直接影響應用程式的效能和成本。

本主題提供適用於 DynamoDB 之不同讀取一致性類型的詳細資料。本主題也說明您可能執行之不同讀取和寫入作業的單位耗用量。

主題

- [讀取一致性](#)
- [讀取和寫入作業](#)

讀取一致性

Amazon DynamoDB 會從資料表、本機次要索引 (LSI)、全域次要索引 (GSI) 和串流讀取資料。如需詳細資訊，請參閱 [Amazon DynamoDB 的核心元件](#)。資料表和 LSI 都提供兩個讀取一致性選項：最終一致 (預設) 和高度一致性讀取。GSI 和串流的所有讀取都是最終一致讀取。

當您的應用程式將資料寫入 DynamoDB 資料表並收到 HTTP 200 回應 (OK) 時，代表已成功寫入並會永久保存。DynamoDB 提供專供讀取隔離，可確保讀取操作一律傳回某個項目已確認的值。讀取將永遠不會從最終沒有成功的寫入中呈現對項目的視圖。專供讀取隔離不會防止在讀取操作後立即修改項目。

最終一致讀取

最終一致是所有讀取操作的預設讀取一致性模式。當向 DynamoDB 資料表或索引發出最終一致讀取時，回應可能無法反映出最近完成的寫入操作結果。如果您在短時間後重複讀取請求，應該最終會傳回最近的資料作為回應。最終一致讀取支援資料表、本機次要索引和全域次要索引。另外請注意，從 DynamoDB 串流的讀取也是最終一致讀取。

最終一致讀取的成本是高度一致性讀取的一半。如需詳細資訊，請參閱 [Amazon DynamoDB 定價](#)。

強烈一致讀取

讀取操作如 GetItem、Query 與 Scan 提供 ConsistentRead 參數。如果設定 ConsistentRead 為 true，DynamoDB 會傳回 up-to-date 資料最多的回應，反映所有先前成功寫入

作業的更新。高度一致性讀取僅支援資料表和本機次要索引。全域次要索引或 DynamoDB 串流不支援高度一致性讀取。

全域資料表讀取一致性

DynamoDB 也支援用於多重作用中和多區域複寫的[全域資料表](#)。全域表是由不同 AWS 區域中的多個複本表格所組成。對任何複本資料表中任何項目所做的任何變更，都會複寫到相同全域資料表中的所有其他複本，通常一秒內即可完成，並且是最終一致。如需詳細資訊，請參閱[一致性和衝突解決](#)。

讀取和寫入作業

DynamoDB 讀取作業可讓您透過指定分區索引鍵值和排序索引鍵值 (選擇性)，從資料表擷取一或多個項目。使用 DynamoDB 寫入作業，您可以插入、更新或刪除資料表中的項目。本主題說明這兩項作業的容量單位耗用量。

主題

- [讀取作業的容量單位消耗](#)
- [寫入作業的容量統一消耗](#)

讀取作業的容量單位消耗

DynamoDB 讀取請求可以是高度一致的、最終一致的，也可以是交易式。

- 高達 4 KB 的項目的強度一致讀取請求需要一個讀取單元。
- 項目的最終一致讀取請求最多 4 KB 需要一半的讀取單元。
- 最多 4 KB 的項目的交易讀取請求需要兩個讀取單元。

若要進一步了解 DynamoDB 讀取一致性模式，請參閱[讀取一致性](#)。

讀取的項目大小會向上四捨五入到下一個 4 KB 倍數。例如，讀取 3,500 個位元組的項目，將會使用與讀取 4 KB 項目相同的輸送量。

如果您需要讀取大於 4 KB 的項目，DynamoDB 需要額外的讀取單元。所需的讀取單元總數取決於項目大小，以及您是否需要最終一致性還是強烈一致的讀取。例如，如果您的項目大小為 8 KB，則需要 2 個讀取單元才能維持一個強烈一致的讀取。如果您選擇最終一致讀取或交易讀取請求選擇 4 個讀取單元，則需要 1 個讀取單元。

下列清單說明 DynamoDB 讀取作業如何使用讀取單位：

- [GetItem](#)：從表格中讀取單個項目。若要判斷GetItem將耗用的讀取單元數目，請取得項目大小並將其四捨五入至下一個 4 KB 邊界。這是指定強烈一致讀取時所需的讀取單元數。對於最終一致讀取(預設值)，請將此數字除以 2。

例如，如果您讀取的項目大小為 3.5 KB，則 DynamoDB 會將項目大小捨入為 4 KB。如果您讀取 10 KB 項目，則 DynamoDB 會將項目大小捨入為 12 KB。

- [BatchGetItem](#)：從一或多個表格讀取最多 100 個項目。DynamoDB 會以個別請求的形式處理批次中的每個GetItem項目。DynamoDB 會先將每個項目的大小四捨五入到下一個 4 KB 邊界，然後計算總大小。結果不一定與所有項目的總大小相同。例如，如果BatchGetItem讀取兩個大小為 1.5 KB 和 6.5 KB 的項目，則 DynamoDB 會將大小計算為 12 KB (4 KB + 8 KB)。DynamoDB 資料庫不會將大小計算為 8 KB (1.5 KB + 6.5 KB)。
- [查詢](#)：讀取具有相同分區索引鍵值的多個項目。傳回的所有項目都會視為單一讀取作業，其中 DynamoDB 會計算所有項目的總大小。然後，DynamoDB 會將大小四捨五入到下一個 4 KB 的邊界。例如，假設您的查詢傳回 10 個項目，其合併大小是 40.8 KB。DynamoDB 會將操作的項目大小四捨五入為 44 KB。如果查詢傳回各為 64 位元組的 1500 個項目，則累積大小為 96 KB。
- [掃描](#)：讀取表格中的所有項目。DynamoDB 會考量所評估項目的大小，而非掃描所傳回項目的大小。如需掃描作業的詳細資訊，請參閱[在 DynamoDB 中使用掃描](#)。

Important

如果您對不存在的項目執行讀取作業，DynamoDB 仍會如上所述消耗讀取輸送量。對於Query/Scan操作，即使沒有資料存在，仍會根據讀取一致性和搜尋要求的分割區數量向您收取額外的讀取輸送量費用。

針對傳回項目的任何操作，您可請求要擷取的部分屬性。不過，這樣做不會影響項目大小計算。此外，Query 和 Scan 也可以傳回項目計數，而非屬性值。獲取項目計數使用相同數量的讀取單位，並且受到相同的項目大小計算。這是因為 DynamoDB 必須讀取每個項目，才能遞增計數。

寫入作業的容量統一消耗

一個寫入單位代表大小不超過 1 KB 的項目的一次寫入。如果您需要寫入大於 1 KB 的項目，DynamoDB 需要使用額外的寫入單位。交易式寫入要求需要 2 個寫入單位，才能針對最大 1 KB 的項目執行一次寫入。所需的寫入請求單位總數取決於項目大小。例如，如果您的項目大小為 2 KB，則需要 2 個寫入單位來維持一個寫入請求，或者需要 4 個寫入單位來支持交易寫入請求。

寫入的項目大小會向上四捨五入到下一個 1 KB 倍數。例如，寫入 500 個位元組的項目，將會使用與寫入 1 KB 項目相同的輸送量。

下列清單說明 DynamoDB 寫入作業如何使用寫入單位：

- [PutItem](#)：將單一項目寫入資料表。如果資料表中已有具有相同主索引鍵的項目，則操作會取代該項目。若要計算佈建輸送量使用，則有關的項目大小是兩者中較大的一個。
- [UpdateItem](#)：修改表格中的單一項目。DynamoDB 會考量項目在更新前後所顯示的大小。使用的佈建輸送量會反映這些項目大小中較大的一個。即使您更新項目屬性的子集，仍UpdateItem會耗用完整數量的佈建輸送量（「之前」和「之後」項目大小中較大者）。
- [DeleteItem](#)：從表格中移除單一項目。佈建輸送量使用是根據所刪除項目的大小。
- [BatchWriteItem](#)：最多可將 25 個項目寫入一或多個表格。DynamoDB 會將批次中的每個項目作為個別 PutItem 或 DeleteItem 請求處理（不支援更新）。DynamoDB 會先將每個項目的大小四捨五入到下一個 1 KB 邊界，然後計算總大小。結果不一定與所有項目的總大小相同。例如，如果BatchWriteItem寫入兩個大小為 500 位元組和 3.5 KB 的項目，DynamoDB 會將大小計算為 5 KB (1 KB + 4 KB)。不會將大小計算為 4 KB (500 位元組 + 3.5 KB)。

針對 PutItem、UpdateItem 和 DeleteItem 操作，DynamoDB 會將項目大小捨入至下一個 1 KB。例如，如果您放置或刪除 1.6 KB 項目，則 DynamoDB 會將項目向上四捨五入到 2 KB。

PutItemUpdateItem、和DeleteItem運算允許條件式寫入，您可以在其中指定必須評估為 true 的運算式，才能成功執行作業。如果表達式計算結果為 false，則 DynamoDB 仍然會從資料表使用寫入容量單位：耗用的寫入容量單位數取決於項目的大小。這個物品可以是表格中的現有物品，也可以是您嘗試建立或更新的新物品。例如，假設現有項目為 300 KB。您嘗試建立或更新的新項目為 310 KB。對於新項目，使用的寫入容量單位將為 310 KB。

輸送量容量

資料表的輸送量容量模式會決定資料表容量的管理方式。輸送量容量也會決定資料表上讀取和寫入作業的費用。在 Amazon DynamoDB 中，您可以選擇表格的隨需模式和佈建模式，以符合不同的工作負載需求。

主題

- [DynamoDB 式概觀](#)
- [隨需容量模式](#)
- [佈建容量模式](#)
- [突發和自適應能力](#)

DynamoDB 式概觀

本節提供 DynamoDB 表格可用的兩種容量模式概觀，以及為應用程式選取適當容量模式時的考量事項。這些模式可讓您根據回應需求以及管理使用方式來滿足不同的需求。

隨需模式

Amazon DynamoDB 隨需是一種無伺服器計費選項，無需規劃容量即可處理每秒數百萬個請求。DynamoDB 隨需提供讀取和寫入請求的 pay-per-request 定價，因此您只需按使用量付費。若為隨需模式資料表，不需要指定您預期應用程式將進行的讀取和寫入輸送量。

使用隨需模式，DynamoDB 可處理輸送量管理的所有層面。您可以視需要進行 API 呼叫，而無需管理表格上的輸送量容量。

如果適用以下任何一種情況，隨需容量模式可能最適合您：

- 您才剛開始使用 Amazon DynamoDB。
- 您正在開發、測試、製作原型並在流量模式未知的生產新應用程式中執行。
- 您的應用程式有突發、間歇性或無法預測的流量，這些流量難以預測。
- 您偏好僅支付您實際用量的輕鬆付費方式。

如需詳細資訊，請參閱 [隨需容量模式](#)。

佈建模式

在佈建模式中，您可以指定應用程式所需的每秒讀取和寫入次數。即使您未完全利用佈建的容量，仍需支付輸送量容量的費用。我們會根據您佈建的每小時讀取和寫入容量向您收費。您可以使用 Auto Scaling 來自動調整表格的佈建容量，以回應流量變化。這有助於您控制 DynamoDB 的使用，以維持或低於已定義的請求率，藉此獲得可預測的成本。

如果適用以下任何一種情況，佈建的容量模式可能最適合您：

- 您擁有可預測或週期性的應用程式流量。
- 您可以在流量一致或逐漸增加的情況下執行應用程式。
- 您可以預測容量要求以控制成本。
- 您的短期流量爆發有限。

如需詳細資訊，請參閱 [佈建容量模式](#)。

下列影片提供表格輸送量容量的簡介。此影片也說明如何根據您的需求選擇容量模式。

隨需容量模式

Amazon DynamoDB 隨需是一種無伺服器計費選項，無需規劃容量即可處理每秒數百萬個請求。DynamoDB 隨需提供讀取和寫入請求的 pay-per-request 定價，因此您只需按使用量付費。

當您選擇隨需模式，DynamoDB 會在您的工作負載上升或下降到任何先前曾達到的流量程度時，立即因應您的工作負載。如果工作負載的流量層級達到新的峰值，DynamoDB 會快速調整來因應工作負載。如需隨選模式縮放屬性的詳細資訊，請參閱 [初始輸送量和擴展屬性](#)。

使用隨需模式的資料表提供相同的單位數毫秒延遲、服務水準協議 (SLA) 承諾，以及 DynamoDB 既有的安全性。您可以為新的與現有的資料表選擇隨需，並繼續使用現有的 DynamoDB API，無需變更程式碼。

隨需輸送量率受到表格層級輸送量配額的限制，該配額適用於具有該帳戶的所有表格。您可以要求提高此配額。如需詳細資訊，請參閱 [輸送量預設配額](#)。

或者，您也可以針對個別隨選表格和全域次要索引，設定每秒最大讀取或寫入 (或兩者) 輸送量。透過設定輸送量，您可以保持資料表層級使用量和成本限制，防止耗用資源的意外激增，並防止過度使用可預測的成本管理。超過最大表格輸送量的輸送量要求會受到限制。您可以根據應用程式需求隨時修改資料表特定的最大輸送量。如需詳細資訊，請參閱 [隨選表格的最大輸送量](#)。

若要開始使用，請建立或更新表格以使用隨選模式。如需詳細資訊，請參閱 [DynamoDB 資料表上的基本操作](#)。

您可以隨時將表格從隨選模式切換為佈建的容量模式。當您在容量模式之間進行多次切換時，適用下列條件：

- 您可以隨時在隨選模式下將新建立的表格切換為佈建的容量模式。不過，您只能在資料表建立時間戳記後 24 小時切換回隨選模式。
- 您可以隨時以隨需模式將現有表格切換為已佈建容量模式。但是，您只能在指示切換到隨選的最後一個時間戳記後 24 小時切換回隨選模式。

如需在讀取和寫入容量模式之間切換的詳細資訊，請參閱 [切換容量模式時的考量](#)。

主題

- [讀取請求單位與寫入請求單位](#)
- [初始輸送量和擴展屬性](#)
- [隨選表格的最大輸送量](#)
- [針對隨需容量模式預熱資料表](#)

讀取請求單位與寫入請求單位

DynamoDB 會針對應用程式在資料表上執行的讀取和寫入 (以讀取請求單位和寫入請求單位計算) 向您收取費用。

對於大小不超過 4 KB 的項目，一個讀取請求單元代表每秒一次強烈一致的讀取操作，或每秒兩個最終一致的讀取操作。如需 DynamoDB 讀取一致性模型的詳細資訊，請參閱 [讀取一致性](#)

對於大小不超過 1 KB 的項目，一個寫入請求單位代表每秒一次寫入操作。

若要取得有關如何使用讀取和寫入單位的更多資訊，請參閱 [〈〉 讀取和寫入作業](#)。

初始輸送量和擴展屬性

DynamoDB 資料表會自動使用隨需容量模式以因應您應用程式的流量。新的隨選資料表將能夠維持每秒高達 4,000 次寫入和每秒 12,000 次讀取。隨需容量模式會立即因應，最高達到資料表峰值流量的兩倍。例如，假設應用程式的流量模式在每秒 25,000 到 50,000 次強烈一致性讀取之間變化。每秒 50,000 次讀取是前一個流量峰值。隨需容量模式可立即容納每秒高達 100,000 次讀取的持續流量。如

如果您的應用程式維持每秒 100,000 次讀取的流量，那麼該峰值就會成為新的先前峰值。先前的峰值可讓後續流量達到每秒 200,000 次讀取。

如果您的工作負載在表格上產生的峰值超過兩倍，DynamoDB 會隨著流量增加而自動配置更多容量。此容量分配有助於確保您的工作負載不會受到限制。但是，如果在 30 分鐘之內超過先前峰值的兩倍以上，還是會發生調節降速。例如，假設您的應用程式的流量模式在每秒 25,000 到 50,000 次強烈一致性讀取之間變化。每秒 50,000 次讀取是先前達到的流量峰值。我們建議您在每秒驅動超過 100,000 次讀取之前，預先加熱桌面或將流量增長至少 30 分鐘的空間保持空間。如需預熱的更多資訊，請參閱[針對隨需容量模式預熱資料表](#)。

隨選表格的最大輸送量

對於隨選資料表，您可以選擇性地指定個別資料表和關聯全域次要索引 (GSI) 的每秒最大讀取或寫入 (或兩者) 輸送量。指定最大隨需輸送量有助於保持資料表層級使用量和成本限制。依預設，不會套用最大輸送量設定，而且您的隨選輸送率受表格中所有表格或 GSI 的[AWS 服務配額](#)限制。如有需要，您可以要求提高服務配額。

當您設定隨選資料表的最大輸送量時，超過指定之最大數量的輸送量要求將受到限制。您可以根據應用程式需求隨時修改表格層級輸送量設定。

以下是一些常見的使用案例，可從使用隨需資料表的最大輸送量中獲益：

- 輸送量成本最佳化 — 針對隨需資料表使用最大輸送量，可提供額外的成本預測性和管理性層。此外，它還提供了更大的靈活性，可以使用隨需模式來支援具有不同流量模式和預算的工作負載。
- 防止過度使用 — 透過設定最大輸送量，您可以防止因非最佳化程式碼或惡意程序而導致的讀取或寫入耗用意外激增，而不是隨選資料表。此表格層級設定可以保護組織，避免在特定時間範圍內耗用過多的資源。
- 保護下游服務 — 客戶應用程式可以包含無伺服器和非伺服器技術。無伺服器架構可快速擴充以符合需求。但具有固定容量的下游組件可能會不堪重負。針對隨選資料表實作最大輸送量設定，可防止大量事件傳播至具有非預期副作用的多個下游元件。

您可以為新的和現有的單一區域表、全域表和 GSI 設定隨需模式的最大輸送量。您也可以先在表格還原和從 Amazon S3 工作流程匯入資料期間設定最大輸送量。

您可以使用 [DynamoDB 主控台](#) 或 [DynamoDB API](#) 為隨需表格指定最大輸送量設定。AWS CLI [AWS CloudFormation](#)

Note

隨選表格的最大輸送量是以最大努力為基礎套用，應該被視為目標，而不是保證的請求上限。由於超載容量，您的工作負載可能會暫時超過指定的最大輸送量。在某些情況下，DynamoDB 會使用突發容量來容納超出表格最大輸送量設定的讀取或寫入。使用爆量容量，未預期的讀取或寫入請求在經過調節之後可能會成功。

主題

- [針對隨選模式使用最大輸送量時的考量](#)
- [請求節流和指標 CloudWatch](#)

針對隨選模式使用最大輸送量時的考量

當您在隨選模式中使用資料表的最大輸送量時，需要考量下列事項：

- 您可以針對任何隨選資料表或該表格內的個別全域次要索引獨立設定讀取和寫入的最大輸送量，以根據特定需求微調您的方法。
- 您可以使用 Amazon CloudWatch 監控和瞭解 DynamoDB 表格層級使用量指標，並為隨需模式決定適當的最大輸送量設定。如需詳細資訊，請參閱 [DynamoDB 指標和維度](#)。
- 當您在一個全域表格複本上指定最大讀取或寫入 (或兩者) 輸送量設定時，相同的最大輸送量設定會自動套用至所有複本表格。重要的是，全域表格中的複本表格和次要索引具有相同的寫入輸送量設定，以確保資料的正確複寫。如需詳細資訊，請參閱 [管理全域資料表的最佳實務和要求](#)。
- 您可以指定的最小讀取或寫入輸送量上限為每秒一個要求單位。
- 您指定的最大輸送量必須低於可用於任何隨選表格或該表格內個別全域次要索引的預設輸送量配額。

請求節流和指標 CloudWatch

如果您的應用程式超過您在隨選表格上設定的最大讀取或寫入輸送量，DynamoDB 會開始限制這些請求。當 DynamoDB 調節讀取或寫入時，會將 `ThrottlingException` 傳回至呼叫者。然後，您可以視需要採取適當的動作。例如，您可以增加或停用最大資料表輸送量設定，或是等待較短的時間後再重試要求。

若要簡化監視為表格或全域次要索引設定的最大傳輸量，請 CloudWatch 提供下列測量結果：[OnDemandMaxReadRequestUnits](#)和[OnDemandMaxWriteRequestUnits](#)。

針對隨需容量模式預熱資料表

對於隨需資料表，DynamoDB 會在流量增加時自動配置更多容量。新的隨選資料表將能夠維持每秒高達 4,000 次寫入和每秒 12,000 次讀取。如果資料表存取平均分佈在分割區之間，且資料表未超過先前尖峰流量的兩倍，則不會限制整個資料表。但是，如果輸送量在相同 30 分鐘內超過前一個峰值的兩倍，則仍然可能發生節流。

一種解決方案是將資料表預熱到預期的峰值容量。務必檢查您的帳戶限制，確認佈建模式下能夠達到所需的容量。[輸送量預設配額](#)如需帳戶層級和資料表層級限制的詳細資訊，請參閱。

Note

如果您要在隨選模式中預熱現有表格或新表格，請至少在預期尖峰前 24 小時啟動此程序。您可以在 24 小時內執行的開關數量有一定的條件。如需這些條件的資訊，請參閱 [切換容量模式時的考量](#)。

若要預熱表格，請執行下列步驟：

1. 根據表格的容量模式，執行下列其中一個步驟：
 - a. 若要預熱目前處於隨選模式的表格，請將其切換至佈建模式，然後等待 24 小時。
 - b. 若要預先加熱已佈建模式或已在佈建模式下 24 小時的新表格，請繼續執行下一個步驟，而不需等待。
2. 將資料表的寫入輸送量設定為所需的峰值，並將其維持數分鐘。在您切換回隨需之前，您將會因此大量輸送量產生成本。
3. 切換到隨需容量模式。這應該允許表格處理與佈建輸送量容量值類似數目的請求數目。

佈建容量模式

在 DynamoDB 中建立新的佈建表格時，必須指定其佈建的輸送量容量。這是資料表可支援的讀取和寫入輸送量。DynamoDB 會使用此資訊來確保有足夠的系統資源來滿足您的輸送量需求。

您可以選擇允許 DynamoDB Auto Scaling 功能來管理您資料表的輸送容量。若要使用 auto 調整比例，您必須在建立資料表時提供讀取和寫入容量的初始設定。DynamoDB auto 擴展使用這些初始設定做為起點，然後根據應用程式的需求動態調整它們。如需詳細資訊，請參閱 [使用 DynamoDB Auto Scaling 功能自動管理輸送容量](#)。

隨著應用程式的資料和存取需求變更，您可能需要調整表格的輸送量設定。如果使用了 DynamoDB Auto Scaling 功能，則會自動調整輸送量設定來回應實際工作負載。您也可以使用此[UpdateTable](#)作業手動調整表格的輸送量容量。如果您需要將現有資料存放區中的資料大量載入至新的 DynamoDB 資料表，則可以決定這麼做。您可以建立具有大型寫入輸送設定的資料表，然後在大量資料載入完成後減少此設定。

您可以隨時將表格從隨選模式切換為佈建的容量模式。當您在容量模式之間進行多次切換時，適用下列條件：

- 您可以隨時在隨選模式下將新建立的表格切換為佈建的容量模式。不過，您只能在資料表建立時間戳記後 24 小時切換回隨選模式。
- 您可以隨時以隨需模式將現有表格切換為已佈建容量模式。但是，您只能在指示切換到隨選的最後一個時間戳記後 24 小時切換回隨選模式。

如需在讀取和寫入容量模式之間切換的詳細資訊，請參閱[切換容量模式時的考量](#)。

主題

- [讀取容量單位和寫入容量單位](#)
- [選擇初始輸送量設定](#)
- [DynamoDB 自動擴展](#)
- [使用 DynamoDB Auto Scaling 功能自動管理輸送容量](#)
- [預留容量](#)

讀取容量單位和寫入容量單位

對於佈建模式表格，您可以根據容量單位指定輸送量需求。這些單位代表應用程式每秒需要讀取或寫入的資料量。您稍後可以視需要修改這些設定，或讓 DynamoDB Auto Scaling 功能自動進行修改。

對於最大 4 KB 的項目，一個讀取容量單位代表每秒一個強烈一致的讀取操作，或每秒兩個最終一致的讀取操作。如需 DynamoDB 讀取一致性模型的詳細資訊，請參閱[讀取一致性](#)

寫入容量單位代表一個項目每秒寫入一次，最多 1 KB。如需不同讀取和寫入作業的詳細資訊，請參閱[讀取和寫入作業](#)。

選擇初始輸送量設定

每個應用程序對讀取和寫入數據庫都有不同的要求。決定 DynamoDB 表格的初始輸送量設定時，請考慮下列事項：

- 預期的讀取和寫入要求率 — 您應該估算每秒需要執行的讀取和寫入次數。
- 項目大小 — 有些項目足夠小，可以使用單一容量單位讀取或寫入。較大的項目需要多個容量單位。藉由估計表格中項目的平均大小，您可以為表格的佈建輸送量指定準確的設定。
- 讀取一致性需求 — 讀取容量單位是以強式一致的讀取作業為基礎，這些作業消耗的資料庫資源是最終一致讀取的兩倍。您應該判斷應用程式需要強烈一致讀取，還是可以放寬這項需求並改為執行最終一致讀取。根據預設，DynamoDB 中的讀取作業最終是一致的。如有必要，您可以針對這些作業要求強烈一致的讀取。

例如，假設您想要從表格中每秒讀取 80 個項目。這些項目的大小為 3 KB，而且您需要強烈一致的讀取。在此情況下，每次讀取都需要一個佈建的讀取容量單位。若要判斷此數目，請將作業的項目大小除以 4 KB。然後，四捨五入至最接近的整數，如下列範例所示：

- $3 \text{ KB} / 4 \text{ KB} = 0.75$ 或 1 個讀取容量單位

因此，若要每秒從表格讀取 80 個項目，請將表格佈建的讀取輸送量設定為 80 個讀取容量單位，如下列範例所示：

- 每個項目 1 個讀取容量單位 × 每秒 80 個讀取 = 80 個讀取容量單位

現在假設您要每秒向表格寫入 100 個項目，並且每個項目的大小為 512 個字節。在這種情況下，每個寫入都需要一個佈建的寫入容量單位。若要判斷此數目，請將作業的項目大小除以 1 KB。然後，四捨五入至最接近的整數，如下列範例所示：

- $512 \text{ 位元組} / 1 \text{ KB} = 0.5$ 或 1 個寫入容量單位

若要每秒將 100 個項目寫入資料表，請將表格的佈建寫入輸送量設定為 100 個寫入容量單位：

- 每個項目 1 個寫入容量單位 × 每秒 100 個寫入 = 100 個寫入容量單位

DynamoDB 自動擴展

DynamoDB auto 擴展主動管理表格和全域次要索引的佈建輸送量容量。透過自動調整規模，您可以定義讀取與寫入容量單位的範圍 (上限與下限)。您也可以定義該範圍內的目標使用率百分比。DynamoDB Auto Scaling 功能會嘗試維持您的目標使用率，即使您的應用程式工作負載有所增減。

透過 DynamoDB Auto Scaling，資料表或全域次要索引可增加其佈建的讀取與寫入容量，來處理突然增加的流量，而不需要進行請求調節。當工作負載降低時，DynamoDB Auto Scaling 可降低輸送量，讓您無須支付未使用的佈建容量。

Note

如果您使用建 AWS Management Console 立表格或全域次要索引，預設會啟用 DynamoDB auto 調整規模。

您可以隨時使用主控台、或其中一個 AWS SDK 來管理 auto 調整設定。AWS CLI如需詳細資訊，請參閱 [使用 DynamoDB Auto Scaling 功能自動管理輸送容量](#)。

利用率

使用率可協助您判斷是否過度佈建容量，在這種情況下應該會降低資料表容量以節省成本。相反地，它也可以協助您判斷是否在佈建容量之下。在這種情況下，您應該增加表格容量，以防止在意外的高流量執行個體期間潛在的請求限制。如需詳細資訊，請參閱 [Amazon DynamoDB auto 擴展：任何規模的效能和成本優化](#)。

如果您使用 DynamoDB auto 擴展，則還需要設定目標使用率百分比。自動調整將使用此百分比做為向上或向下調整容量的目標。我們建議將目標使用率設定為 70%。如需詳細資訊，請參閱 [使用 DynamoDB Auto Scaling 功能自動管理輸送容量](#)。

使用 DynamoDB Auto Scaling 功能自動管理輸送容量

許多資料庫工作負載本來就具週期性且難以事先預測。例如，假設有一個社交聯網應用程式，其中大部分使用者會在日間活動。資料庫必須能夠處理日間活動，但夜間則不需要同樣多的輸送量。又例如，請設想一下正被快速採用的新行動遊戲應用程式。如果遊戲變得太熱門，可能會超過可用的資料庫資源，而導致效能變慢及客戶不滿意。這類工作負載通常需要手動介入來擴展或縮減資料庫資源，以回應不斷改變的用量。

Amazon DynamoDB auto 動擴展使用應用 Ap AWS plication Auto Scaling 服務代表您動態調整佈建的輸送量容量，以回應實際的流量模式。這可讓資料表或全域次要索引增加其佈建的讀取與寫入容量，不

需調節就以處理突然增加的流量。當工作負載降低時，Application Auto Scaling 可降低輸送量，讓您無須為未使用的佈建容量付費。

Note

如果您使用建 AWS Management Console 立表格或全域次要索引，預設會啟用 DynamoDB auto 調整規模。您可隨時修改自動調整規模設定。如需詳細資訊，請參閱 [AWS Management Console 搭配使用 auto 調整](#)。

當您刪除資料表或全域表格複本時，系統不會自動刪除任何相關聯的可調整目標、縮放原則或 CloudWatch 警示。

使用 Application Auto Scaling，您就可以為資料表或全域次要索引建立擴展政策。調整規模政策可指定您要擴展讀取容量或寫入容量 (或兩者)，也可為資料表或索引指定最大與最小佈建容量單位設定。

調整規模政策也包含目標使用率：即在某個時間點耗用的佈建輸送量百分比。Application Auto Scaling 使用目標追蹤演算法，向上或向下調整資料表 (或索引) 的佈建輸送量，以此回應實際的工作負載，讓實際容量使用率保持或接近目標使用率。

當兩個資料點在一分鐘範圍內違反設定的目標使用率值時，可觸發自動擴展。因此，由於消耗的容量在兩分鐘內高於目標使用率，因此可能會發生 auto 擴展。但是，如果尖峰相隔超過一分鐘，則可能不會觸發 auto 縮放。同樣地，當連續 15 個資料點低於目標使用率時，就會觸發縮減規模事件。在任何一種情況下，在觸發 auto 縮放之後，都會 [UpdateTable](#) 叫用呼叫。然後可能需要幾分鐘的時間來更新表格或索引的佈建容量。在此期間，任何超過表格先前佈建容量的要求都會受到限制。

Important

您不能調整要洩露的數據點數量以觸發基礎警報 (儘管當前數量 future 可能會更改)。

您可將自動調整規模目標使用率值設定在 20% 至 90% 之間，做為您的讀寫容量。

Note

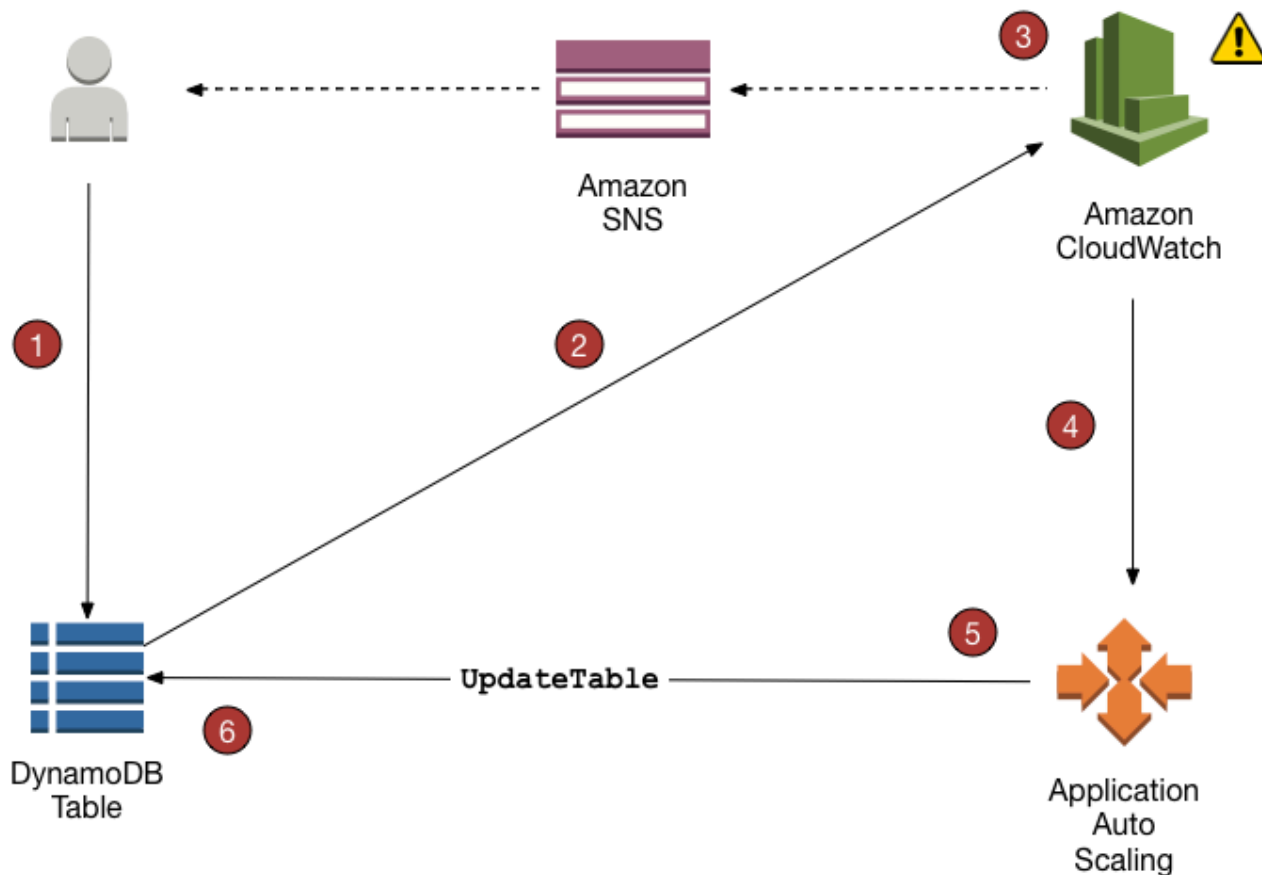
除了資料表外，DynamoDB Auto Scaling 功能也支援全域次要索引。每個全域次要索引都有自己的佈建輸送容量，與其基礎資料表的佈建輸送容量無關。當您為全域次要索引建立擴展政策時，Application Auto Scaling 會調整索引的佈建輸送量設定，以確保其實際使用率保持或接近您想要的使用比率。

DynamoDB Auto Scaling 功能的運作方式

Note

若要快速開始使用 DynamoDB Auto Scaling 功能，請參閱 [AWS Management Console 搭配使用 auto 調整](#)。

下圖提供 DynamoDB Auto Scaling 功能如何管理資料表輸送容量的高層級概觀。



下列步驟摘要說明上圖所示的自動調整規模程序：

1. 您可以為 DynamoDB 資料表建立 Application Auto Scaling 政策。
2. DynamoDB 會將使用的容量指標發佈到 Amazon CloudWatch
3. 如果表格的消耗容量在特定時間長度內超過目標使用率 (或低於目標)，Amazon CloudWatch 觸發警示。您可以在主控台檢視警示，並使用 Amazon Simple Notification Service (Amazon SNS) 接收通知。

4. CloudWatch 警示會叫 Application Auto Scaling 來評估您的資源調整政策。
5. Application Auto Scaling 發出 UpdateTable 請求來調整資料表的佈建輸送量。
6. DynamoDB 處理 UpdateTable 請求，並動態增加 (或減少) 資料表的佈建輸送容量，以便接近您的目標使用率。

若要了解 DynamoDB Auto Scaling 功能的運作方式，請假設您有一個名為 ProductCatalog 的資料表。該資料表不常大量載入資料，因此不會產生太多寫入活動。不過，它會發生大量讀取活動，並會隨著時間變化。透過監控的 Amazon CloudWatch 指標 ProductCatalog，您可以判斷表格需要 1,200 個讀取容量單位 (以避免 DynamoDB 在活動達到尖峰時節流讀取請求)。您也判斷出當讀取流量在最低點時，ProductCatalog 至少需要 150 個讀取容量單位。如需防止限流的詳細資訊，請參閱 [使用佈建容量模式的 DynamoDB 表格的節流問題](#)。

在 150 到 1,200 個讀取容量單位範圍內，您決定 ProductCatalog 資料表的適當目標使用率為 70%。目標使用率是以使用容量單位與佈建容量單位的比率，以百分比表示。Application Auto Scaling 使用其目標追蹤演算法，確保視需要調整 ProductCatalog 的佈建讀取容量，讓使用率維持在或接近 70%。

Note

只有在實際工作負載持續幾分鐘保持很高 (或很低) 的狀態時，DynamoDB Auto Scaling 功能才會修改佈建輸送量設定。Application Auto Scaling 目標追蹤演算法會設法長期將目標使用率保持在或接近您選擇的數值。

資料表的內建高載容量可應付短期遽增的活動。如需詳細資訊，請參閱 [高載容量](#)。

若要為 ProductCatalog 資料表啟用 DynamoDB Auto Scaling 功能，您可以建立擴展政策。此政策會指定以下內容：

- 要管理的資料表或全域次要索引
- 要管理的容量類型 (讀取容量或寫入容量)
- 佈建輸送量設定的上限與下限
- 您的目標使用率

當您建立擴展政策時，Application Auto Scaling 會代表您建立一對 Amazon CloudWatch 警示。每組警示皆代表您佈建輸送量設定的上限與下限。當表格的實際使用率在持續一段時間內偏離目標使用率時，就會觸發這些 CloudWatch 警示。

觸發其中一個 CloudWatch 警示時，Amazon SNS 會傳送通知給您 (如果您已啟用它)。然後 CloudWatch 警示會叫 Application Auto Scaling 整，進而通知 DynamoDB 適當地向上或向下調整 ProductCatalog 表格的佈建容量。

在擴展事件期間，AWS Config 會根據記錄的組態項目收費。發生縮放事件時，會針對每個讀取和寫入 auto-scaling 事件建立四 CloudWatch 個 ProvisionedCapacity 警報：警報: ProvisionedCapacityLow、ProvisionedCapacityHigh 和 ConsumedCapacity 警報: AlarmHigh、AlarmLow。這會導致總共八個警報。因此，每個擴展事件都會 AWS Config 記錄八個組態項目。

Note

您也可以排程 DynamoDB 擴展，使其在特定時間發生。[在這裡](#)了解基本步驟。

使用須知

開始使用 DynamoDB Auto Scaling 功能之前，您應該注意下列事項：

- DynamoDB Auto Scaling 功能可根據您的自動調整規模政策，視需要經常增加讀取容量或寫入容量。所有 DynamoDB 配額仍然有效，如 [Amazon DynamoDB 中的服務、帳戶和資料表配額](#) 所述。
- DynamoDB Auto Scaling 功能不會阻止您手動修改佈建輸送量設定。這些手動調整不會影響與 DynamoDB auto 調整相關的任何現有 CloudWatch 警示。
- 如果您為具有一或多個全域次要索引的資料表啟用 DynamoDB Auto Scaling，強烈建議您也將自動調整規模統一套用至這些索引。這將有助於確保改善資料表的寫入和讀取效能，並有助於避免限流。您可以在 AWS Management Console 中選取 Apply same settings to global secondary indexes (將相同的設定套用至全域次要索引) 來啟用自動調整規模的功能。如需詳細資訊，請參閱 [在現有資料表上啟用 DynamoDB Auto Scaling 功能](#)。
- 當您刪除表格或全域表格複本時，任何相關聯的可調整目標、縮放原則或 CloudWatch 警示都不會隨之自動刪除。
- 為現有資料表建立 GSI 時，不會為 GSI 啟用 Auto Scaling。建置 GSI 時，您必須手動管理容量。GSI 上的回填完成並達到活動狀態後，Auto Scaling 的操作將恢復正常。

AWS Management Console 搭配使用 auto 調整

當您使用建立新表格時，依預設會為該表啟用 Amazon DynamoDB auto 擴展功能。AWS Management Console 您也可以使用主控台，啟用現有資料表的自動調整規模、修改自動調整規模設定或停用自動調整規模。

Note

如需設定縮放和縮小冷卻時間等進階功能，請使用 AWS Command Line Interface (AWS CLI) 管理 DynamoDB auto 縮放。如需詳細資訊，請參閱 [使用 AWS CLI 管理 auto 調整](#)。

主題

- [開始之前：授予 DynamoDB Auto Scaling 功能的使用者許可](#)
- [建立啟用 Auto Scaling 的新資料表](#)
- [在現有資料表上啟用 DynamoDB Auto Scaling 功能](#)
- [在主控台上檢視自動調整規模活動](#)
- [修改或停用 DynamoDB Auto Scaling 設定](#)

開始之前：授予 DynamoDB Auto Scaling 功能的使用者許可

在 AWS Identity and Access Management (IAM) 中，AWS 受管政策 `DynamoDBFullAccess` 提供了使用 DynamoDB 主控台所需的許可。不過，如需 DynamoDB Auto Scaling 功能，則使用者需要額外的權限。

Important

須有 `application-autoscaling:*` 許可才能刪除已啟用自動擴展功能的資料表。受 AWS 管理的策略 `DynamoDBFullAccess` 包括此類權限。

若要為 DynamoDB 主控台存取和 DynamoDB auto 擴展設定使用者，請建立角色並將 `AmazonDynamoDBFullAccess` 原則新增至該角色。然後將角色指派給使用者。

建立啟用 Auto Scaling 的新資料表**Note**

DynamoDB Auto Scaling 需要有服務連結角色 (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`)，其會代表您執行自動調整規模動作。系統會自動建立此角色。如需詳細資訊，請參閱《Application Auto Scaling 使用者指南》中的 [適用於 Application Auto Scaling 的服務連結角色](#)。

在啟用 Auto Scaling 的情況下建立新資料表

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 選擇 Create Table (建立資料表)。
3. 在 Create table (建立資料表) 頁面上，輸入 Table name (資料表名稱) 和主索引鍵。
4. 如果選取 Default settings (預設設定)，則會在啟用自動擴展的情況下建立資料表。

否則，對於自訂設定：

- a. 選擇 Customize settings (自訂設定)。
- b. 在 Read/write capacity settings (讀取/寫入容量設定) 區段中，選取 Provisioned (佈建) 容量模式，並將 Read capacity (讀取容量)、Write capacity (寫入容量) 或兩者的 Auto scaling (自動擴展) 設定為 On (開啟)。針對這些項目，請為資料表設定所需的擴展政策，並選擇性地設定資料表的所有全域次要索引。
 - 容量單位下限：輸入自動調整規模範圍的下限。
 - 容量單位上限：輸入自動調整規模範圍的上限。
 - 目標使用率：輸入資料表的目標使用率百分比。

Note

如果您為新資料表建立全域次要索引，則建立時索引的容量將與基礎資料表的容量相同。建立資料表後，您可以在資料表的設定中變更索引的容量。

5. 當您滿意設定後，請選擇 Create table (建立資料表)。系統會使用自動調整規模參數建立資料表。

在現有資料表上啟用 DynamoDB Auto Scaling 功能

Note

DynamoDB Auto Scaling 需要有服務連結角色 (AWSServiceRoleForApplicationAutoScaling_DynamoDBTable)，其會代表您執行自動調整規模動作。系統會自動建立此角色。如需詳細資訊，請參閱 [Application Auto Scaling 的服務連結角色](#)。

啟用現有資料表的 DynamoDB Auto Scaling

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇您想要使用的資料表，然後選擇 Additional settings (其他設定) 索引標籤。
4. 在 Read/write capacity (讀取/寫入容量) 區段中，選擇 Edit (編輯)。
5. 在 Capacity mode (容量模式) 區段中，選擇 Provisioned (佈建)。
6. 在 Table capacity (資料表容量) 區段，將 Read capacity (讀取容量)、Write capacity (寫入容量)，或兩者的 Auto scaling (自動擴展) 設定為 On (開啟)。針對這些項目，請為資料表設定所需的擴展政策，並選擇性地設定資料表的所有全域次要索引。
 - 容量單位下限：輸入自動調整規模範圍的下限。
 - 容量單位上限：輸入自動調整規模範圍的上限。
 - 目標使用率：輸入資料表的目標使用率百分比。
 - 對所有全域次要索引使用相同的容量讀取/寫入容量設定-選擇全域次要索引是否應該使用與基礎資料表相同的自動擴展政策。

Note

為獲得最佳效能，我們建議您啟用 Apply same read/write capacity settings to global secondary indexes (將相同的讀取/寫入容量設定套用至全域次要索引)。此選項允許 DynamoDB Auto Scaling 功能以統一方式擴展基礎資料表上的所有全域次要索引。這包括現有的全域次要索引，以及您未來為此資料表建立的任何其他索引。啟用此選項後，您就無法在個別全域次要索引上設定擴展政策。

7. 當您滿意設定後，請選擇 Save (儲存)。

在主控台上檢視自動調整規模活動

當應用程式將讀取和寫入流量推送至資料表時，DynamoDB Auto Scaling 功能會動態修改資料表的輸送量設定。Amazon 會追 CloudWatch 蹤所有 DynamoDB 表格和次要索引的佈建和消耗容量、限制事件、延遲以及其他指標。

若要在 DynamoDB 主控台中檢視這些指標，請選擇要使用的資料表，然後選擇 Monitor (監控) 索引標籤。若要建立可自訂的表格指標檢視，請選取檢視全部於 CloudWatch。

修改或停用 DynamoDB Auto Scaling 設定

您可以使用 AWS Management Console 來修改您的 DynamoDB auto 縮放設定。若要這樣做，請前往 [Additional settings \(其他設定\) 索引標籤](#)，然後選擇 Edit (編輯) 中的 Read/write capacity (讀取/寫入容量) 一節。如需這些設定的詳細資訊，請參閱 [在現有資料表上啟用 DynamoDB Auto Scaling 功能](#)。

使用 AWS CLI 管理 auto 調整

您可以使用 AWS Command Line Interface (AWS CLI) 來管理 Amazon DynamoDB auto 擴展 AWS Management Console，而不是使用。本節中的教學課程會示範如何安裝和設定 AWS CLI 來管理 DynamoDB Auto Scaling 功能。在此教學課程中，您將執行下列操作：

- 建立名為 TestTable 的 DynamoDB 資料表。初始輸送量設定是 5 個讀取容量單位和 5 個寫入容量單位。
- 為 TestTable 建立 Application Auto Scaling 政策。該政策會設法將使用寫入容量與佈建寫入容量之間的目標比率保持在 50%。此指標範圍介於 5 到 10 個寫入容量單位之間。(Application Auto Scaling 無法調整超出此範圍的輸送量。)
- 運行 Python 程式來將寫入流量推送至 TestTable。當目標比率持續超過 50% 時，Application Auto Scaling 會通知 DynamoDB 提高 TestTable 的輸送量，以便維持 50% 的目標使用率。
- 確認 DynamoDB 已成功調整 TestTable 的佈建寫入容量。

Note

您也可以排程 DynamoDB 擴展，使其在特定時間發生。[在這裡](#)了解基本步驟。

主題

- [開始之前](#)
- [步驟 1：建立 DynamoDB 資料表](#)
- [步驟 2：註冊可擴展的目標](#)
- [步驟 3：建立擴展政策](#)
- [步驟 4：驅動寫入流量 TestTable](#)
- [步驟 5：檢視 Application Auto Scaling 動作](#)
- [\(選用\) 步驟 6：清除](#)

開始之前

開始這些教學課程之前，請完成以下任務。

安裝 AWS CLI

若您尚未執行此作業，您必須安裝及設定 AWS CLI。若要執行此作業，請遵循 AWS Command Line Interface 使用者指南中的這些說明：

- [安裝 AWS CLI](#)
- [設定 AWS CLI](#)

安裝 Python

本教學課程的一部分會要求您執行 Python 程式 (請參閱 [步驟 4：驅動寫入流量 TestTable](#))。如果您還沒有安裝 Python 程式，則可以 [下載 Python](#)。

步驟 1：建立 DynamoDB 資料表

在此步驟中，您可 AWS CLI 以使用建立 TestTable。主索引鍵包含 pk (分割區索引鍵) 和 sk (排序索引鍵)。這些屬性的類型皆為 Number。初始輸送量設定是 5 個讀取容量單位和 5 個寫入容量單位。

1. 使用下面的 AWS CLI 命令來創建表。

```
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. 若要檢查資料表的狀態，請使用以下命令。

```
aws dynamodb describe-table \  
  --table-name TestTable \  
  --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

當資料表狀態為 ACTIVE 時，即可供使用。

步驟 2：註冊可擴展的目標

接下來，您可以使用 Application Auto Scaling 將資料表寫入容量註冊為可擴展的目標。這可讓 Application Auto Scaling 調整佈建的寫入容量 TestTable，但只能在 5 到 10 個容量單位的範圍內調整。

Note

DynamoDB Auto Scaling 需要有服務連結角色 (AWSServiceRoleForApplicationAutoScaling_DynamoDBTable)，其會代表您執行自動調整規模動作。系統會自動建立此角色。如需詳細資訊，請參閱《Application Auto Scaling 使用者指南》中的[適用於 Application Auto Scaling 的服務連結角色](#)。

1. 輸入下列命令來註冊可擴展的目標。

```
aws application-autoscaling register-scalable-target \  
  --service-namespace dynamodb \  
  --resource-id "table/TestTable" \  
  --scalable-dimension "dynamodb:table:WriteCapacityUnits" \  
  --min-capacity 5 \  
  --max-capacity 10
```

2. 若要確認註冊，請使用下列命令。

```
aws application-autoscaling describe-scalable-targets \  
  --service-namespace dynamodb \  
  --resource-id "table/TestTable"
```

Note

您也可以針對全域次要索引註冊可擴展的目標。例如，對於全域次要索引 (test-index)，資源 ID 和可擴展的維度引數會適當地更新。

```
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
```

```
--max-capacity 10
```

步驟 3：建立擴展政策

在此步驟中，您會建立 TestTable 的擴展政策。此政策會定義 Application Auto Scaling 可在哪些詳細資訊下調整資料表的佈建輸送量，以及在執行此動作時要採取的動作。您可以將此政策與您在上一個步驟中定義的可擴展目標建立關聯 (TestTable 資料表的寫入容量單位)。

該政策包含下列元素：

- **PredefinedMetricSpecification**：允許 Application Auto Scaling 調整的指標。對於 DynamoDB，下列值是 **PredefinedMetricType** 的有效值：
 - **DynamoDBReadCapacityUtilization**
 - **DynamoDBWriteCapacityUtilization**
- **ScaleOutCooldown**：提高佈建輸送量的每個 Application Auto Scaling 事件之間的最短時間 (以秒為單位)。此參數可讓 Application Auto Scaling 持續 (但不積極) 增加輸送量，以便回應真實的工作負載。ScaleOutCooldown 的預設設定為 0。
- **ScaleInCooldown**：減少佈建輸送量的每個 Application Auto Scaling 事件之間的最短時間 (以秒為單位)。此參數可讓 Application Auto Scaling 以逐漸且可預測的方式降低輸送量。ScaleInCooldown 的預設設定為 0。
- **TargetValue**：Application Auto Scaling 可確保將耗用容量與佈建容量的比率保持在此值或接近此數值。您能以百分比的形式定義 TargetValue。

Note

為了進一步了解 TargetValue 如何運作，請假設您資料表的佈建輸送量設定為 200 個寫入容量單位。您決定為此資料表建立擴展政策，並將 TargetValue 設為 70%。現在，假設您開始將寫入流量導向該資料表，那實際的寫入輸送量就會是 150 個容量單位。現在的 consumed-to-provisioned 比率是 (一百五十/二百)，或百分之七十五。此比率超過您的目標，因此 Application Auto Scaling 會將佈建的寫入容量提高至 215，使得比率成為 (150/215) (也就是 69.77 百分比)；盡可能靠近但不超過 TargetValue。

對於 TestTable，您將 TargetValue 設為 50%。「Application Auto Scaling 整」會在 5 到 10 個容量單位的範圍內調整表格的佈建輸送量 (請參閱[步驟 2：註冊可擴展的目標](#))，使 consumed-to-

provisioned 比率維持在 50% 或接近 50%。您可以將 ScaleOutCooldown 和 ScaleInCooldown 的數值設為 60 秒。

1. 使用下列內容建立名為 scaling-policy.json 的檔案。

```
{
  "PredefinedMetricSpecification": {
    "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
  },
  "ScaleOutCooldown": 60,
  "ScaleInCooldown": 60,
  "TargetValue": 50.0
}
```

2. 使用下列 AWS CLI 命令建立原則。

```
aws application-autoscaling put-scaling-policy \
  --service-namespace dynamodb \
  --resource-id "table/TestTable" \
  --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
  --policy-name "MyScalingPolicy" \
  --policy-type "TargetTrackingScaling" \
  --target-tracking-scaling-policy-configuration file://scaling-policy.json
```

3. 請注意，在輸出中，Application Auto Scaling 建立了兩個 Amazon CloudWatch 警報，每個警報分別用於擴展目標範圍的上下邊界。
4. 使用下列 AWS CLI 命令可檢視有關資源調度政策的詳細資料。

```
aws application-autoscaling describe-scaling-policies \
  --service-namespace dynamodb \
  --resource-id "table/TestTable" \
  --policy-name "MyScalingPolicy"
```

5. 在輸出中，確認政策設定符合 [步驟 2：註冊可擴展的目標](#) 和 [步驟 3：建立擴展政策](#) 中的規格。

步驟 4：驅動寫入流量 TestTable

現在，您可以將資料寫入 TestTable 來測試擴展政策。為此，您須執行 Python 程式。

1. 使用下列內容建立名為 bulk-load-test-table.py 的檔案。

```
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. 請輸入下列命令來執程式。

```
python bulk-load-test-table.py
```

TestTable 的佈建寫入容量非常低 (5 個寫入容量單位)，因此程式偶爾會因寫入調節而停滯。這是預期的行為。

讓程式繼續執行，您則繼續下一個步驟。

步驟 5：檢視 Application Auto Scaling 動作

在此步驟中，您可以檢視代表您啟動的 Application Auto Scaling 動作。您也可以確認 Application Auto Scaling 已更新 TestTable 的佈建寫入容量。

1. 輸入下列命令來檢視 Application Auto Scaling 動作。

```
aws application-autoscaling describe-scaling-activities \
    --service-namespace dynamodb
```

在 Python 程式執行時，偶爾重新執行此命令。(叫用擴展政策需要幾分鐘的時間。) 您最終應該會看到下列輸出。

```
...
{
  "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"
},
...
```

這表示 Application Auto Scaling 已對 DynamoDB 發出 UpdateTable 請求。

2. 輸入下列命令，確認 DynamoDB 已增加資料表的寫入容量。

```
aws dynamodb describe-table \
  --table-name TestTable \
  --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

WriteCapacityUnits 應該從 5 擴展至 10。

(選用) 步驟 6：清除

在此教學課程中，您已建立數個資源。如果不再需要這些資源，就可以將其刪除。

1. 刪除 TestTable 的擴展政策。

```
aws application-autoscaling delete-scaling-policy \
  --service-namespace dynamodb \
  --resource-id "table/TestTable" \
  --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
```



```
--policy-name "MyScalingPolicy"
```

2. 取消註冊可擴展的目標。

```
aws application-autoscaling deregister-scalable-target \  
  --service-namespace dynamodb \  
  --resource-id "table/TestTable" \  
  --scalable-dimension "dynamodb:table:WriteCapacityUnits"
```

3. 刪除 TestTable 資料表。

```
aws dynamodb delete-table --table-name TestTable
```

使用開 AWS 發套件在 Amazon DynamoDB 表上設定 auto 擴展

除了使用 AWS Management Console 和 AWS Command Line Interface (AWS CLI) 之外，您還可以撰寫與 Amazon DynamoDB 自動擴展互動的應用程式。本節包含可用來測試此功能的兩個 Java 程式：

- EnableDynamoDBAutoscaling.java
- DisableDynamoDBAutoscaling.java

啟用資料表的 Application Auto Scaling

以下程式顯示 DynamoDB 資料表 (TestTable) 自動調整規模政策的設定範例。它會繼續如下：

- 此程式會將寫入容量單位註冊為 TestTable 的可擴展目標。此指標範圍介於 5 到 10 個寫入容量單位之間。
- 建立可擴展目標之後，程式會建置目標追蹤組態。該政策會設法將使用寫入容量與佈建寫入容量之間的目標比率保持在 50%。
- 程式接著會根據目標追蹤組態來建立擴展政策。

Note

手動移除表格或全域表格複本時，不會自動移除任何關聯的可調整目標、擴展政策或 CloudWatch 警示。

此程式需要您提供有效 Application Auto Scaling 服務連結角色的 Amazon Resource Name (ARN)。(例如 : `arn:aws:iam::122517410325:role/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`。) 在下列程式中，將 `SERVICE_ROLE_ARN_GOES_HERE` 取代為實際 ARN。

```
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());
}

}

}
```

停用資料表的 Application Auto Scaling

下列程式會反轉先前的程序。它會移除自動調整規模政策，然後取消註冊可擴展目標。

```
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());
}
```

```
}  
  
}
```

預留容量

對於使用標準表類別的**佈建容量表**，DynamoDB 提供針對讀取和寫入容量購買保留容量的功能。預留容量購買是一項協議，在合約期間內支付最低數量的佈建輸送容量，以換取折扣定價。

Note

您無法為複寫的寫入容量單位 (RWCU) 購買保留容量。保留容量僅適用於購買該容量的區域。使用 DynamoDB 標準 IA 資料表類別或隨需容量模式的資料表也無法使用預留容量。

預留容量是以 100 個 WCU 或 100 個 RCU 的配置方式購買。最小的保留容量供應項目為 100 個容量單位 (讀取或寫入)。DynamoDB 保留容量提供為期一年的承諾量，或在特定區域作為三年承諾提供。您可以在一年期內節省高達 54% 的標準費率折扣，三年期可節省 77% 的標準費率折扣。如需有關購買方式和時機的詳細資訊，請參閱 [Amazon DynamoDB 保留容量](#)。

購買 DynamoDB 保留容量時，您需要支付一次性部分預付款，並獲得已認可佈建用量的折扣小時費率。無論實際用量為何，您都需支付整個已認可佈建用量的費用，因此節省的成本與使用緊密相關。您佈建的任何超出已購買預留容量的容量，均以標準佈建容量費率計費。透過事先預留您的讀取與寫入容量單位，您就可以比佈建容量成本省下明顯更多費用。

您無法將預留容量出售、取消或轉移至其他區域或帳戶。

Note

保留容量不是組織專用的容量。這是帳單 discount，適用於使用佈建的容量進行讀取和/或寫入您帳戶。

突發和自適應能力

為了將輸送量例外導致的節流減到最低，DynamoDB 會使用突發容量來處理使用尖峰。DynamoDB 使用自適應容量來協助因應不均勻的存取模式。

高載容量

DynamoDB 透過高載容量，為您的輸送量佈建提供一些靈活性。每當您未完全使用可用輸送量時，DynamoDB 會保留未使用容量的一部分，以供日後突發輸送量以處理使用量尖峰。使用爆量容量，未預期的讀取或寫入請求在經過調節之後可能會成功。

DynamoDB 目前最多可保留五分鐘 (300 秒) 的未使用讀取和寫入容量。在偶爾爆發讀取或寫入活動期間，這些額外的容量單位可以快速消耗，甚至比您為表格定義的每秒佈建輸送量容量還要快。

DynamoDB 也可能在不事先通知的情況下為背景維護和其他任務使用高載容量。

請注意，高載容量的詳細資訊可能會在將來有所變更。

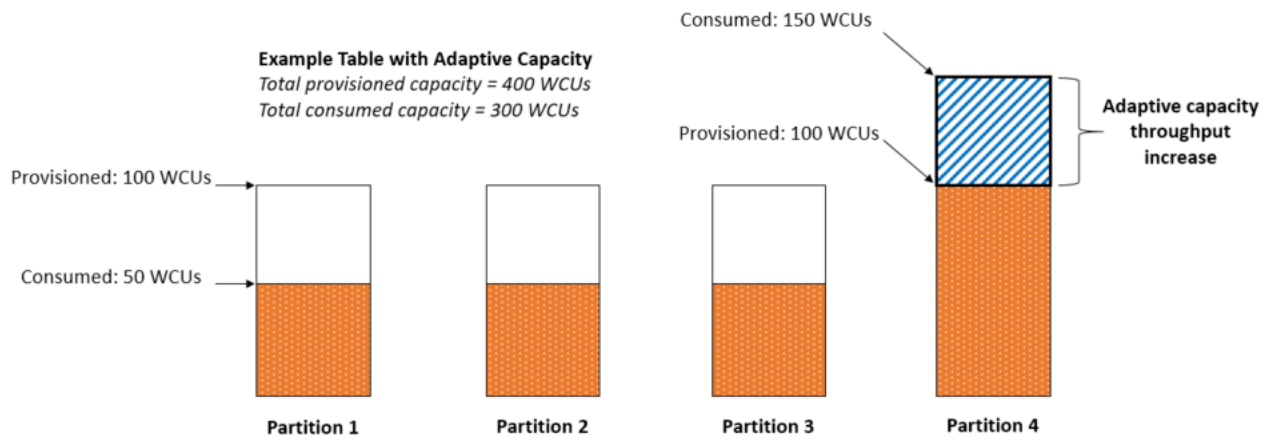
調適型容量

DynamoDB 會自動將您的資料分散到[分割區](#)，這些分割區會儲存在 AWS 雲端並非總是可以一直均勻分配讀取和寫入活動。當資料存取不平衡時，與其他分割區相比，「經常性」分割區可以接收更高量的讀取和寫入流量。由於分割區上的讀取和寫入作業是獨立管理的，因此，如果單一分割區接收超過 3000 個讀取作業或超過 1000 項寫入作業，就會發生節流。調適型容量用於自動增加能接收更多流量的分割區傳輸容量。

DynamoDB 調適型容量使您的應用程式能夠繼續讀取和寫入經常性分割區而不受限制，加強適應不均衡的存取模式，前提是流量不會超過您資料表的總佈建的容量或分割區最大容量。調適型容量用於自動並立即增加能接收更多流量的分割區傳輸容量。

下圖說明調適型容量的運作方式。此範例資料表已佈建 400 個單位 (WCU) 並在四個分割區之間平均共用，因此每個分割區每秒最多可承受 100 個 WCU。分割區 1、2 和 3 各接收每秒 50 個 WCU 的寫入流量。分割區 4 每秒接收 150 個 WCU。此經常性分割區可在仍有未使用的高載容量時接受寫入流量，但最終分割區會調節每秒超過 100 個 WCU 的流量。

DynamoDB 調適性容量透過增加磁碟分割 4 的容量進行回應，因此它可以在不受限制的情況下維持更高的工作負載，達到每秒 150 WCU。



所有 DynamoDB 資料表都會自動啟用調適型容量，不需額外費用。您不需要明確啟用或停用此功能。

隔離經常存取的项目

若您的應用程式導致一個或多個项目的流量過大，則調適型容量將重新平衡您的分區，以使經常存取的项目不會駐留在同一分區上。這種經常存取项目的隔離，可減少因您的工作負載於單一分割區超過輸送量配額而導致需要調節的可能性。只要项目集合不是由排序索引鍵單調增加或減少所追蹤的流量，您也可以依排序索引鍵將项目集合分成區段。

如果您的應用程式於單一项目經常有高流量，調適型容量可能重新平衡您的資料，讓分割區僅包含單一經常存取的项目。如果是這種情況，DynamoDB 可為分割區中該單一项目的主索引鍵帶來最多 3,000 個 RCU 和 1,000 個 WCU 的輸送量。資料表上存在[本機次要索引](#)時，調適型容量不會在多個資料表分割區中分割项目集合。

設定 DynamoDB

除了 Amazon DynamoDB 網路服務之外，還 AWS 提供您可以在電腦上執行的可下載版本的 DynamoDB。可下載的版本對於開發和測試您的程式碼來說很有幫助。它可讓您在本地機撰寫及測試應用程式，而不用存取 DynamoDB Web 服務。

本節的主題說明如何設定 DynamoDB (可下載版本) 和 DynamoDB Web 服務。

主題

- [設定 DynamoDB Local \(可下載版本\)](#)
- [設定 DynamoDB \(Web 服務\)](#)

設定 DynamoDB Local (可下載版本)

可下載版 Amazon DynamoDB 能讓您在本地機開發及測試應用程式，不用存取 DynamoDB Web 服務。而且資料庫在您的電腦上可獨自運作。當您準備好要在生產環境中部署您的應用程式時，您可移除程式碼中的本地端點，並將其指向 DynamoDB Web 服務。

擁有這個本地版本可讓您節省輸送量、資料儲存體和數據傳輸費用。此外，您在開發應用程式時，不需要網際網路連線。

DynamoDB 本地版可以作為 [Apache Maven 依存項目](#) (需要 JRE) 或 [Docker 映像檔下載](#)。

如果您偏好使用 Amazon DynamoDB Web 服務，請參閱 [設定 DynamoDB \(Web 服務\)](#)。

主題

- [在本機電腦上部署 DynamoDB](#)
- [DynamoDB Local 使用須知](#)
- [DynamoDB 本地版的版本歷史記錄](#)
- [DynamoDB 本地機中的遙測功能](#)

在本機電腦上部署 DynamoDB

Important

DynamoDB 本機 jar 可從這裡參考的 AWS CloudFront 發佈連結下載。從 2025 年 1 月 1 日開始，舊的 S3 發佈儲存貯體將不再處於作用中狀態，而 DynamoDB 本機將僅透過分 CloudFront 發連結進行分發。

有兩個主要版本的 DynamoDB 本機可供使用：本機 v2.x (目前) 和 DynamoDB 本機 v1.x (舊版)。客戶應盡可能使用 2.x 版 (當前)，因為它支持最新版本的 Java 運行時環境，並與 Maven 項目的 jakarta.* 命名空間兼容。DynamoDB 本機 v1.x 將於 2025 年 1 月 1 日起終止標準支援。在此日期之後，v1.x 將不再收到更新或錯誤修復。

Note

DynamoDB 本機版 AWS_ACCESS_KEY_ID 只能包含字母 (A-Z、a-z) 和數字 (0-9)。

下載本 DynamoDB

請依照這些步驟在您的電腦上安裝並執行 DynamoDB。

在電腦上設定 DynamoDB

1. 從下列其中一個位置免費下載本機 DynamoDB。

下載連結	檢查總和
.tar.gz .zip	.tar.gz.sha256 .zip.sha256

Important

若要在您的電腦上執行 DynamoDB v2.4.0 或更新版本，您必須擁有 Java 執行階段環境 (JRE) 17.x 或更新版本。應用程式無法在舊版的 JRE 上執行。

2. 在您下載封裝後，請解壓縮內容，並將解壓縮的目錄複製到您選擇的位置。
3. 若要在您的電腦上啟動 DynamoDB，請開啟命令提示視窗，導覽至您解壓縮 DynamoDBLocal.jar 的目錄，然後輸入下列命令。

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

Note

如果您使用的是 Windows PowerShell，請務必包含參數名稱或完整名稱和值，如下所示：

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

DynamoDB 會處理傳入的請求，直到您將其停止。若要停止 DynamoDB，請在命令提示字元中按 Ctrl+C。

DynamoDB 預設使用連接埠 8000。如果無法使用連接埠 8000，此命令就會擲出例外狀況。如需完整的 DynamoDB 執行時間選項清單 (包括 -port)，請輸入此命令。

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

4. 以程式設計方式或透過 AWS Command Line Interface (AWS CLI) 存取 DynamoDB 之前，您必須先設定您的登入資料，以便啟用您的應用程式授權。可下載版 DynamoDB 需要所有登入資料才能運作，如下列範例所示。

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

您可以使用 AWS CLI 的 `aws configure` 命令來設定登入資料。如需詳細資訊，請參閱 [使用 AWS CLI](#)。

5. 開始寫入應用程式。若要存取使用本機執行的 DynamoDB AWS CLI，請使用參數 `--endpoint-url`。例如，使用下列命令來列出 DynamoDB 資料表。

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

將 DynamoDB 當作泊塢視窗映像執行本機

Amazon DynamoDB 的可下載版本作為 Docker 影像提供。如需詳細資訊，請參閱 [dynamodb-local](#)。若要查看您目前的 DynamoDB 本機版本，請輸入下列命令：

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

如需使用本機 DynamoDB 做為 AWS Serverless Application Model (AWS SAM) 上建置的 REST 應用程式的一部分的範例，請參閱管理訂單的 [SAM DynamoDB 應用程式](#)。此範例應用程式示範如何使用 DynamoDB local 進行測試。

如果您想要執行同時使用 DynamoDB 本機容器的多容器應用程式，請使用 Docker Compose 定義並執行應用程式中的所有服務，包括 DynamoDB 本機。

使用 Docker Compose 安裝並執行 DynamoDB Local：

1. 下載並安裝 [Docker Desktop](#)。
2. 將以下程式碼複製到一個檔案中並儲存為 `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
```

如果您想讓應用程式和 DynamoDB local 位於不同的容器中，請使用以下 yml 檔案。

```
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
```

這個 docker-compose.yml 指令碼會建立一個 app-node 容器和一個 dynamodb-local 容器。指令碼會在 app-node 容器中執行命令，該命令使用 dynamodb-local 連線至 AWS CLI 容器，並說明帳戶和資料表限制。

若要搭配您自己的應用程式影像使用，請將下列範例中的 image 數值取代為您應用程式的數值：

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

```
environment:
  AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
  AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
  REGION: 'eu-west-1'
```

Note

YAML 指令碼需要您指定 AWS 存取金鑰和 AWS 秘密金鑰，但這些金鑰不需要是您存取 DynamoDB 本機的有效 AWS 金鑰。

3. 執行下列命令列命令：

```
docker-compose up
```

以阿帕奇 Maven 相依性的方式執行本機

請按照以下步驟操作，在您的應用程式中將 Amazon DynamoDB 用為依存項目。

部署 DynamoDB 為 Apache Maven 儲存庫

1. 下載並安裝 Apache Maven。如需詳細資訊，請參閱[下載 Apache Maven](#)和[安裝 Apache Maven](#)。
2. 將 DynamoDB Maven 儲存庫新增至您應用程式的專案物件模型 (POM) 檔案。

```
<!--Dependency-->
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>DynamoDBLocal</artifactId>
    <version>2.4.0</version>
  </dependency>
</dependencies>
```

與 Spring Boot 3 和/或 Spring Framework 6 搭配使用的範例範本：

```
<?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>2.4.0</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

您也可以使用 [Maven 中央存儲庫 URL](#)。

如需範例專案的範例專案範例，其中展示了多種設定和使用 DynamoDB 本機方法，包括下載 JAR 檔案、以 Docker 映像檔的形式執行，以及將其用作 Maven 相依性，請參閱 [DynamoDB 本機範例 Java 專案](#)。

DynamoDB Local 使用須知

除端點外，使用可下載版 Amazon DynamoDB 執行的應用程式，也應該能使用 DynamoDB Web 服務。但是在本機使用 DynamoDB 時，您應該清楚下列事項：

- 如果您使用 `-sharedDb` 此選項，DynamoDB 會建立名為 `shared-local-instance .db` 的單一資料庫檔案。連線至 DynamoDB 的每個程式都能存取此檔案。如果您刪除此檔案，就會遺失所有存放在其中的資料。
- 如果您省略 `-sharedDb`，資料庫檔案會名稱為 `myaccesskeyid_region.db`，而 AWS 存取金鑰 ID 和 [AWS 區域] 就如同它們出現在您的應用程式組態中一樣。如果您刪除此檔案，就會遺失所有存放在其中的資料。
- 如果您使用 `-inMemory` 選項，DynamoDB 完全不會寫入任何資料庫檔案。相反地，所有資料都會寫入記憶體，並且在您終止 DynamoDB 時不儲存資料。
- 如果您使用 `-inMemory` 選項，則也必須使用 `-sharedDb` 選項。

- 如果您使用 `-optimizeDbBeforeStartup` 選項，就必須也指定 `-dbPath` 參數，以便 DynamoDB 能找到它的資料庫檔案。
- DynamoDB 的 AWS 開發套件需要您的應用程式組態指定存取金鑰值和區域值。AWS 除非您使用的是 `-sharedDb` 或 `-inMemory` 選項，否則 DynamoDB 會使用這些數值來命名本機資料庫檔案。這些值不一定是在本機執行的有效 AWS 值。不過，您可能會發現使用有效的值很方便，因為稍後只要變更您使用的端點，就可以在雲端執行程式碼。
- DynamoDB 本機版一律對 `billingModeSummary` 傳回 Null。
- DynamoDB 本機版 `AWS_ACCESS_KEY_ID` 只能包含字母 (A-Z、a-z) 和數字 (0-9)。
- 本機不支援 [Point-in-time 復原 \(PITR\)](#)。

主題

- [命令列選項](#)
- [設定區域端點](#)
- [可下載版 DynamoDB 和 DynamoDB Web 服務之間的差異](#)

命令列選項

您可以使用下列命令列選項搭配可下載版 DynamoDB 使用：

- `-corsvalue`— 啟用對的跨來源資源共用 (CORS) 支援。JavaScript 您必須提供逗號分隔的特定網域「允許」清單。`-cors` 的預設設定是星號 (*)，意為允許公開存取。
- `-dbPath value` : DynamoDB 寫入其資料庫檔案的目錄。如果您不指定此選項，檔案會寫入目前的目錄。您不能同時指定 `-dbPath` 和 `-inMemory`。
- `-delayTransientStatuses` : 導致 DynamoDB 對部分操作造成延遲。DynamoDB (可下載版本) 幾乎可以立即執行部分任務，例如在資料表和索引上建立/更新/刪除操作。但是，DynamoDB 服務需要較多時間處理這些任務。設定此參數有助於在您電腦上執行的 DynamoDB 將 DynamoDB Web 服務的行為模擬得更逼真。(此參數目前只會造成 CREATING 或 DELETING 狀態的全域次要索引延遲)。
- `-help` : 列印用量摘要及選項。
- `-inMemory` : DynamoDB 在記憶體中執行，而非使用資料庫檔案。當您停止 DynamoDB 時，不會儲存任何資料。您不能同時指定 `-dbPath` 和 `-inMemory`。
- `-optimizeDbBeforeStartup` : 先最佳化基礎資料庫資料表，再啟動您電腦上的 DynamoDB。當您使用此參數時，也必須指定 `-dbPath`。

- `-port value` : DynamoDB 用來與您的應用程式進行通訊的連接埠號碼。如果您不指定此選項，預設連接埠為 8000。

Note

DynamoDB 預設使用連接埠 8000。如果無法使用連接埠 8000，此命令就會擲出例外狀況。您可以使用 `-port` 選項來指定不同的連接埠號碼。如需完整的 DynamoDB 執行時間選項清單 (包括 `-port`)，請輸入此命令：

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

- `-sharedDb` : 若您指定 `-sharedDb`，則 DynamoDB 會使用單一資料庫檔案，而非為每個登入資料和區域使用不同的檔案。
- `-disableTelemetry` : 指定時，DynamoDB 本機將不會傳送任何遙測資料。
- `-version`— 列印本 DynamoDB。

設定區域端點

依預設，AWS 開發套件和工具會使用適用於 Amazon DynamoDB 網路服務的端點。若要使用開發套件和工具搭配可下載版 DynamoDB，您必須指定區域端點：

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

AWS Command Line Interface

您可以使用 AWS Command Line Interface (AWS CLI) 與可下載的 DynamoDB 互動。例如，您可以使用它執行「[在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#)」中的所有步驟。

若要存取在本機執行的 DynamoDB，請使用 `--endpoint-url` 參數。以下是使用列出電腦 DynamoDB 中表格的範例。AWS CLI

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

Note

AWS CLI 無法使用 DynamoDB 的可下載版本做為預設端點。因此，您必須 `--endpoint-url` 使用每個指 AWS CLI 令來指定。

AWS 開發套件

您指定端點的方式，視您使用的程式設計語言和 AWS 開發套件而定。下列各節說明如何執行此作業：

- [Java：設定 AWS 區域與端點](#)(本機支援適用於 Java V1 和 V2 的 AWS 開發套件)
- [.NET：設定 AWS 區域與端點](#)

Note

如需其他程式設計語言的範例，請參閱「[開始使用 DynamoDB 和開發套件 AWS](#)」。

可下載版 DynamoDB 和 DynamoDB Web 服務之間的差異

可下載版 DynamoDB 僅用於開發和測試。相較之下，DynamoDB Web 服務則是受管服務，具可擴展性、可用性及耐用性的特色，因此適用於生產。

可下載版 DynamoDB 和 Web 服務之間的差異如下：

- AWS 區域 而且不同 AWS 帳戶 的用戶端層級不受支援。
- 可下載版 DynamoDB 會忽略佈建輸送量設定，即使 CreateTable 操作需要它們。您可為 CreateTable 指定任何您想要的佈建讀取和寫入輸送量數字，即使這些數字不予使用。您可以每天呼叫 UpdateTable，次數不限。但是會忽略任何佈建輸送量值的變更。
- Scan 操作以循序方式執行。不支援平行掃描。忽略 Segment 操作的 TotalSegments 與 Scan 參數。
- 資料表資料的讀取和寫入操作速度只受限於您電腦的速度。CreateTable、UpdateTable 和 DeleteTable 操作會立即發生，且資料表狀態一律為 ACTIVE。只變更資料表或全域次要索引之佈建輸送量設定的 UpdateTable 操作會立即發生。如果 UpdateTable 操作建立或刪除任何全域次要索引，則這些索引會先轉換到一般狀態 (例如分別為 CREATING 和 DELETING)，再變成 ACTIVE 狀態。資料表在這段時間內仍維持 ACTIVE。
- 讀取操做為最終一致。但是，因您電腦上執行的 DynamoDB 速度之故，大部分的讀取會是強烈一致。
- 不會追蹤項目集合指標和項目集合大小。操作回應中會傳回 Null，而不是項目集合指標。
- 在 DynamoDB 中，每個結果集傳回的資料有 1 MB 的限制。DynamoDB Web 服務和可下載版本都會執行此限制。但在查詢索引時，DynamoDB 服務只會計算投影索引鍵和屬性的大小。反之，可下載版 DynamoDB 會計算整個項目的大小。

- 如果您使用的是 DynamoDB Streams，建立碎片的速率可能不同。在 DynamoDB Web 服務中，碎片建立行為有部分會受到資料表分割區活動的影響。當您在本機執行 DynamoDB 時，不會分割資料表。無論哪一種情況，碎片都只是暫時存在，所以您的應用程式不應該依賴碎片行為。
- `TransactionConflictExceptions` 不會由可下載的 DynamoDB 針對交易 API 擲回。建議您使用 Java 模擬架構，在 DynamoDB 處理常式中模擬 `TransactionConflictExceptions`，以測試您的應用程式如何回應相衝突的交易。
- 在 DynamoDB Web 服務中，無論是透過主控台存取還是 AWS CLI，資料表名稱都區分大小寫。名為 `Authors` 和名為 `authors` 的資料表可同時存在，視為不同的資料表。在可下載的版本中，資料表名稱不區分大小寫，因為嘗試建立這樣的兩份資料表會造成錯誤。
- DynamoDB 的可下載版本不支援標記。
- DynamoDB 的可下載版本會忽略中的 [限制](#) 參數。 [ExecuteStatement](#)

DynamoDB 本機版的版本歷史記錄

下表說明 DynamoDB 本機版每個版本的重要變更。

版本	變更	描述	日期
2.4.0	Support <code>ReturnValuesOnConditionCheckFailure</code> - 嵌入式模式	<ul style="list-style-type: none"> • 用於在多個流上操作 <code>TrimmedDataAccessException</code> 的嵌入式模式修復 • 在嵌入式模式下修復 SDKv2 的異常轉譯 	2024年4月17日
2.3.0	碼頭和 JDK 升級	<ul style="list-style-type: none"> • 升級至碼頭 12.0.2 • 升級至 JDK • 將 ANTLR4 升級至 	2024年3月14日
2.2.0	增加了對表刪除保護和 <code>ReturnValuesOnConditionCheck</code>	<ul style="list-style-type: none"> • 增加了對表刪除保護的支持 • 增加了支持 <code>ReturnVal</code> 	2023年12月14日

版本	變更	描述	日期
	kFailure 參數的支持	<p>uesOnConditionCheckFailure</p> <ul style="list-style-type: none"> 增加了對-版本標誌的支持 	
2.1.0	SQLite Native Libraries for Maven 專案的支援和新增遙測功能	<ul style="list-style-type: none"> 在 DynamoDB 本機上新增遙測功能 動態複製 SQLite Native Libraries for Maven 專案 已移除 Maven 依存項目中的 io.github.ganadist.sqlite4java 程式庫 升級 GoogleGuava 至 3 月 1 日 	2023 年 10 月 23 日
2.0.0	從 javax 遷移到 jakarta 命名空間和 JDK11 支援	<ul style="list-style-type: none"> 從 javax 遷移到 jakarta 命名空間和 JDK11 支援 修復在伺服器啟動時處理無效存取和私密金鑰的問題 透過更新相依關係來修復 Maven 找到的漏洞 	2023 年 7 月 5 日

版本	變更	描述	日期
1.25.0	增加了對表刪除保護和ReturnValuesOnConditionCheckFailure 參數的支持	<ul style="list-style-type: none"> 增加了對表刪除保護的支持 增加了支持 ReturnValuesOnConditionCheckFailure 增加了對-版本標誌的支持 	2023 年 12 月 18 日
1.24.0	SQLite Native Libraries for Maven 專案的支援和新增遙測功能	<ul style="list-style-type: none"> 在 DynamoDB 本機上新增遙測功能 動態複製 SQLite Native Libraries for Maven 專案 已移除 Maven 依存項目中的 io.github.ganadist.sqlite4java 程式庫 升級 GoogleGuava 至 3 月 1 日 	2023 年 10 月 23 日
1.23.0	在伺服器啟動時處理無效存取和私密金鑰	<ul style="list-style-type: none"> 修復在伺服器啟動時處理無效存取和私密金鑰的問題 透過更新相依關係來修復 Maven 找到的漏洞 	2023 年 6 月 28 日
1.22.0	對於 PartiQL 限制操作的支援	<ul style="list-style-type: none"> 最佳化 PartiQL 的 IN 子句 限制操作支援 Maven 專案的 M1 支援 	2023 年 6 月 8 日

版本	變更	描述	日期
1.21.0	支援每筆交易 100 個動作	<ul style="list-style-type: none"> 將每筆交易的動作從 25 提升到 100 升級 Docker 映像檔 Open JDK 到 11 修復重複項目時拋出異常的奇偶校驗 BatchExecuteStatement 	2023 年 1 月 26 日
1.20.0	新增對 M1 Mac 的支援	<ul style="list-style-type: none"> 新增對 M1 Mac 的支援 將 Jetty 相依性升級至 9.4.48.v20220622 	2022 年 9 月 12 日
1.19.0	已升級 PartiQL 剖析器	已升級 PartiQL 剖析器和其他相關程式庫	2022 年 7 月 27 日
1.18.0	已升級 log4j-core 和 jackson-core	已升級 log4j-core 至 2.17.1 版和 jackson-core 2.10.x 到 2.12.0 版	2022 年 1 月 10 日
1.17.2	已升級 log4j-core	已升級 log4j-core 相依性到 2.16 版	2021 年 1 月 16 日
1.17.1	已升級 log4j-core	已更新 log4j-core 相依性以修補零時差漏洞，以防止遠端程式碼之執行 - Log4Shell	2021 年 1 月 10 日

版本	變更	描述	日期
1.17.0	Javascript Web Shell 已作廢	<ul style="list-style-type: none">更新了 AWS SDK 依賴關係到適用於 Java 1.12.x 的 AWS SDKJavascript Web Shell 已作廢	2021 年 1 月 8 日

DynamoDB 本機中的遙測功能

在 AWS，我們根據與客戶互動所學來開發和推出服務，並且採納客戶的意見回饋將我們的產品不斷推陳出新。遙測功能這項額外的資訊能幫助我們進一步了解客戶的需求、診斷問題，並且提供各項功能來改善客戶體驗。

DynamoDB 本機會收集遙測資訊，例如一般使用指標、系統和環境資訊以及錯誤。如需所收集遙測類型的詳細資訊，請參閱 [收集的資訊類型](#)。

DynamoDB 本機不會收集個人資訊，例如使用者姓名或電子郵件地址。同時也不會擷取敏感的專案層級資訊。

身為客戶，您可以全權掌控是否開啟遙測功能，並且可以隨時變更您的設定。如果遙測功能保持開啟，DynamoDB 本機會在背景傳送遙測資料，不需任何額外的客戶互動。

使用命令列選項關閉遙測功能

您可以在使用選項 `-disableTelemetry` 啟動 DynamoDB 本機時，使用命令列選項關閉遙測功能。如需詳細資訊，請參閱 [命令列選項](#)

關閉單一工作階段的遙測功能

在 macOS 和 Linux 作業系統中，您可以關閉單一工作階段的遙測功能。若要關閉目前工作階段的遙測功能，請執行下列命令，將環境變數 `DDB_LOCAL_TELEMETRY` 設定為 `false`。針對每個新的終端或工作階段重複此命令。

```
export DDB_LOCAL_TELEMETRY=0
```

在所有工作階段中關閉您的設定檔的遙測功能

當您在作業系統上執行 DynamoDB 本機時，執行下列命令即可關閉所有工作階段的遙測功能。

關閉 Linux 中的遙測功能

1. 執行：

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

2. 執行：

```
source ~/.profile
```

關閉 macOS 中的遙測功能

1. 執行：

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

2. 執行：

```
source ~/.profile
```

關閉 Windows 中的遙測功能

1. 執行：

```
setx DDB_LOCAL_TELEMETRY 0
```

2. 執行：

```
refreshenv
```

收集的資訊類型

- 使用情形資訊：一般遙測，例如同伺服器啟動/停止，以及呼叫的 API 或操作。

- 系統和環境資訊：Java 版本、作業系統 (Windows、Linux 或 macOS)、DynamoDB 本機執行所在的環境 (例如，單機版 JAR、Docker 容器或作為 Maven 依存項目)，以及用量屬性的雜湊值。

進一步了解

DynamoDB 本機收集的遙測資料遵循 AWS 資料隱私權政策。如需詳細資訊，請參閱下列內容：

- [AWS 服務條款](#)
- [資料隱私權常見問答集](#)

設定 DynamoDB (Web 服務)

使用 Amazon DynamoDB Web 服務：

1. [註冊 AWS](#)。
2. 取得 [AWS 存取金鑰](#) (用於以程式設計方式存取 DynamoDB)。

Note

如果您計劃僅透過與 DynamoDB 互動 AWS Management Console，則不需要 AWS 存取金鑰，而且可以略過。[使用主控台](#)

3. [設定您的登入資料](#) (用來以程式設計方式存取 DynamoDB)。

註冊成為 AWS

若要使用 DynamoDB 服務，您必須擁有一個 AWS 帳戶。如果您還沒有帳戶，系統會在您註冊時提示您建立帳戶。除非您使用它們，否則您註冊的任何 AWS 服務不會向您收取費用。

若要註冊 AWS

1. 開啟 <https://portal.aws.amazon.com/billing/signup>。
2. 請遵循線上指示進行。

部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

授予程式設計存取權

您必須擁有程式設計方式存取權，或透過 AWS Command Line Interface (AWS CLI) 存取 DynamoDB 之前。如果您打算只使用 DynamoDB 主控台，則不需要程式設計存取權。

如果使用者想要與 AWS 之外的 AWS Management Console. 授與程式設計存 AWS取權的方式取決於正在存取的使用者類型。

若要授與使用者程式設計存取權，請選擇下列其中一個選項。

哪個使用者需要程式設計存取權？	到	By
人力身分 (IAM Identity Center 中管理的使用者)	使用臨時登入資料來簽署對 AWS CLI、AWS SDK 或 AWS API 的程式設計要求。	請依照您要使用的介面所提供的指示操作。 <ul style="list-style-type: none"> 如需詳細資訊 AWS CLI，請參閱 《使 AWS CLI 用AWS Command Line Interface者指南》 AWS IAM Identity Center中的〈配置使用〉。 如需 AWS SDK、工具和 AWS API，請參閱 AWS SDK 和工具參考指南中的 IAM 身分中心身分驗證。
IAM	使用臨時登入資料來簽署對 AWS CLI、AWS SDK 或 AWS API 的程式設計要求。	遵循 《IAM 使用者指南》 中的〈將臨時登入資料搭配 AWS 資源使用〉中的指示
IAM	(不建議使用)	請依照您要使用的介面所提供的指示操作。

哪個使用者需要程式設計存取權？	到	By
	使用長期認證簽署對 AWS CLI、AWS SDK 或 AWS API 的程式設計要求。	<ul style="list-style-type: none">• 如需相關資訊 AWS CLI，請參閱使用指南中的使用 IAM 使用者登入資料進行驗證。AWS Command Line Interface• 對於 AWS SDK 和工具，請參閱 AWS SDK 和工具參考指南中的使用長期憑據進行身份驗證。• 如需 AWS API，請參閱 IAM 使用者指南中的管理 IAM 使用者的存取金鑰。

設定您的憑證

您必須先設定認證以程式設計方式或透過存取 DynamoDB AWS CLI，才能啟用應用程式的授權。

有幾種方式可以執行此作業。例如，您可以手動建立登入資料檔案，存放您的存取金鑰 ID 與私密存取金鑰。您也可以使用指 AWS CLI 令 `aws configure` 來自動建立檔案。或者，您也可以使用環境變數。如需有關設定認證的詳細資訊，請參閱程式設計專用 AWS SDK 開發人員指南。

若要安裝和配置 AWS CLI，請參閱[使用 AWS CLI](#)。

與其他 DynamoDB 服務整合

您可以將 DynamoDB 與許多其他 AWS 服務整合。如需詳細資訊，請參閱下列內容：

- [將 DynamoDB 與其他 AWS 服務搭配使用](#)
- [AWS CloudFormation 適用於動態支援](#)
- [搭配 DynamoDB 使用 AWS Backup](#)
- [AWS Identity and Access Management \(IAM\)](#)
- [使用 AWS Identity and Access Management 搭配 DynamoDB](#)

存取 DynamoDB

您可以使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或 DynamoDB API 存取 Amazon DynamoDB。

主題

- [使用主控台](#)
- [使用 AWS CLI](#)
- [使用 API](#)
- [使用適用於 DynamoDB 的 NoSQL Workbench](#)
- [IP 地址範圍](#)

使用主控台

您可以透過此連結存取 Amazon DynamoDB 的 AWS Management Console : <https://console.aws.amazon.com/dynamodb/home>。

您可以使用主控台在 DynamoDB 中執行下列作業：

- 在 DynamoDB 儀表板上監控最近的提醒、總容量、服務運作狀態，以及最新的 DynamoDB 新聞。
- 建立、更新和刪除資料表。容量計算器會根據您提供的用量資訊，提供要請求之容量單位的估計。
- 管理串流。
- 檢視、新增、更新及刪除存放在資料表中的項目。管理存留時間 (TTL)，定義資料表中的項目何時會過期，以從資料庫中自動將其刪除。
- 查詢及掃描資料表。
- 設定及檢視警示，以便監控您資料表的容量用量。從即時圖表檢視表格的主要監控指標 CloudWatch。
- 修改資料表的佈建容量。
- 修改資料表的資料表類別。
- 建立和刪除全域次要索引。
- 建立觸發條件，將 DynamoDB Streams 連線到 AWS Lambda 函數。
- 套用標籤至資源，協助整理和識別他們。
- 購買預留容量。

主控台會顯示簡介畫面，提示您建立第一個資料表。若要檢視您的資料表，請在主控台左側的導覽窗格中，選擇 Tables (資料表)。

以下是每個導覽標籤中，每個資料表可用動作的高層級概觀：

- 概觀：檢視資料表詳細資訊，包括項目計數和度量。
- 索引：管理全域和本機次要索引。
- 監控 — 檢視警示、CloudWatch 參與者洞察和 Cloudwatch 指標。
- 全域資料表：管理資料表複本。
- 備份 — 管理 point-in-time 復原和隨選備份。
- 匯出與串流：將資料表匯出到 Amazon S3，並管理 DynamoDB Streams 和 Kinesis Data Streams。
- 其他設定：管理讀取/寫入容量、存留時間設定、加密和標籤。

使用 AWS CLI

您可以使用 AWS Command Line Interface (AWS CLI) 從命令列控制多項 AWS 服務，並透過指令碼將服務自動化。您可以將 AWS CLI 應用於臨機操作，例如建立資料表。其也可以用於在公用程式指令碼中嵌入 Amazon DynamoDB 操作。

在您搭配 DynamoDB 使用 AWS CLI 之前，必須先取得存取金鑰 ID 及私密存取金鑰。如需詳細資訊，請參閱 [授予程式設計存取權](#)。

如需 AWS CLI 中 DynamoDB 所有可用命令的完整清單，請參閱 [《AWS CLI 命令參考》](#)。

主題

- [下載和設定 AWS CLI](#)
- [搭配 DynamoDB 使用 AWS CLI](#)
- [AWS CLI 搭配 DynamoDB 機](#)

下載和設定 AWS CLI

可透過此連結取得 AWS CLI：<http://aws.amazon.com/cli>。它可在 Windows、macOS，或 Linux 上執行。在您下載 AWS CLI 之後，請遵循這些步驟安裝及設定它：

1. 前往 [《AWS Command Line Interface 使用者指南》](#)。

2. 請遵循[安裝 AWS CLI](#) 及 [設定 AWS CLI](#) 的說明進行。

搭配 DynamoDB 使用 AWS CLI

命令列格式包含 DynamoDB 操作名稱，隨後接著該操作的參數。AWS CLI 支援適用於參數數值的速記語法以及 JSON。

例如，以下命令會建立名為 Music 的資料表。磁碟分割索引鍵是藝術家，排序索引鍵是 SongTitle。(為確保易讀性，本節的長命令以分行顯示。)

```
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
```

以下命令會為資料表新增新的項目。這些範例混合使用速記語法和 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
```

在命令列上難以編寫有效的 JSON。但是 AWS CLI 能夠讀取 JSON 檔案。例如，請試想下列 JSON 程式碼片段，其存放在名為 key-conditions.json 的檔案中。

```
{
```



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

您現在可以使用 AWS CLI 發行 Query 請求。在此範例中，key-conditions.json 檔案的內容會做為 --key-conditions 參數使用。

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

AWS CLI 搭配 DynamoDB 機

也 AWS CLI 可以與電腦上執行的 DynamoDB 本機 (可下載版本) 互動。若要啟用此功能，請為每個命令新增下列參數：

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

以下範例使用 AWS CLI 列出本機資料庫中的資料表。

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

若 DynamoDB 使用的連接埠號碼並非預設值 (8000)，請相應地修改 --endpoint-url 的數值。

Note

AWS CLI 無法使用 DynamoDB 本機 (可下載版本) 做為預設端點。因此，您必須為每個命令指定 --endpoint-url。

使用 API

您可以使用 AWS Management Console 和 AWS Command Line Interface 以互動方式使用 Amazon DynamoDB。但是，若要充分利用 DynamoDB，您可以使用 AWS 開發套件撰寫應用程式的程式碼。

[開AWS發套件在瀏覽器、網路、Node.js、PHP、Python、紅寶石、C++、圍棋、安卓和 iOS JavaScript 中為 DynamoDB 提供廣泛的支援。](#) 若要快速開始使用這些語言，請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

在您搭配 DynamoDB 使用 AWS 開發套件之前，必須先取得 AWS 存取金鑰 ID 及私密存取金鑰。如需詳細資訊，請參閱 [設定 DynamoDB \(Web 服務\)](#)。

如需使用 AWS 開發套件進行 DynamoDB 應用程式設計的高階概觀，請參閱 [使用 DynamoDB 和開發套件進程式設計 AWS](#)。

使用適用於 DynamoDB 的 NoSQL Workbench

您也可以透過下載並使用 [DynamoDB 專用 NoSQL Workbench](#) 來存取 DynamoDB。

Amazon DynamoDB 專用 NoSQL Workbench 是用於現代資料庫開發和操作的跨平台用戶端 GUI 應用程式。適用於 Windows、macOS 和 Linux。NoSQL Workbench 是視覺化開發工具，提供了資料模型建立、資料視覺化和查詢開發功能，協助您設計、建立、查詢及管理 DynamoDB 資料表。NoSQL Workbench 現在包含 DynamoDB 本機版做為安裝程序的選用部分，可讓您更輕鬆地在 DynamoDB 本機版中建立資料模型。若要深入了解 DynamoDB 本機版及其要求，請參閱 [設定 DynamoDB Local \(可下載版本\)](#)。

Note

適用於 DynamoDB 的 NoSQL 工作台目前不支援使用雙因素驗證 (2FA) 設定的AWS登入。

建立資料模型

借助 DynamoDB 專用 NoSQL Workbench，您可以使用滿足您應用程式資料存取模式的現有資料模型，來建置新的資料模型或設計模型。您也可以在程序結束時，匯入及匯出設計好的資料模型。如需詳細資訊，請參閱 [使用 NoSQL Workbench 建立資料模型](#)。

資料視覺化

資料模型視覺化工具提供畫布，您可在此映射查詢以及視覺化應用程式的存取模式 (面向)，不必編寫程式碼。每個面向都會對應 DynamoDB 中不同的存取模式。您可以自動產生範例資料，以便在資料模型中使用。如需詳細資訊，請參閱 [視覺化資料存取模式](#)。

建立操作

NoSQL Workbench 提供強大的圖形使用者界面供您開發及測試查詢。您可以使用 operation builder (操作建置器) 來檢視、探索及查詢即時資料集。您也可以使用結構式操作建置器來建立及執行資料平面操作。支援投射及條件表達式，並讓您使用多種語言產生範本程式碼。如需詳細資訊，請參閱 [使用 NoSQL Workbench 探索資料集與建立操作](#)。

IP 地址範圍

Amazon Web Services (AWS) 會以 JSON 格式發佈目前的 IP 地址範圍。若要檢視目前範圍，請下載 [ip-ranges.json](#)。如需詳細資訊，請參閱《AWS 一般參考》中的 [AWS IP 地址範圍](#)。

若要尋找您可以用來[存取 DynamoDB 資料表和索引](#)的 IP 地址範圍，請在 ip-ranges.json 檔案中搜尋下列字串："service": "DYNAMODB"。

Note

IP 地址範圍不適用於 DynamoDB Streams 或 DynamoDB Accelerator (DAX)。

DynamoDB 入門

使用本節中的實作教學課程，協助您開始使用並進一步了解 Amazon DynamoDB。

主題

- [DynamoDB 中的基本概念](#)
- [事前準備 - 入門教學](#)
- [步驟 1：建立資料表](#)
- [第 2 步：使用控制台或將數據寫入表 AWS CLI](#)
- [步驟 3：從資料表讀取資料](#)
- [步驟 4：更新資料表中的資料](#)
- [步驟 5：查詢資料表中的資料](#)
- [步驟 6：建立全域次要索引](#)
- [步驟 7：查詢全域次要索引](#)
- [步驟 8：\(選用\) 清除資源](#)
- [DynamoDB 入門：後續步驟](#)

DynamoDB 中的基本概念

開始之前，您應該熟悉 Amazon DynamoDB 中的基本概念。如需詳細資訊，請參閱 [DynamoDB 核心元件](#)。

接著您再了解[先決條件](#)，看看如何設定 DynamoDB。

事前準備 - 入門教學

開始 Amazon DynamoDB 教學課程之前，請遵循[設定 DynamoDB 中的步驟](#)。然後繼續[步驟 1：建立資料表](#)。

Note

- 如果您計劃只透過與 DynamoDB 互動 AWS Management Console，則不需要 AWS 存取金鑰。完成[註冊中的](#)步驟 AWS，然後繼續執行[步驟 1：建立資料表](#)。

- 如果您不想要註冊免費方案帳戶，則可以設定 [DynamoDB 本機版 \(可下載版本\)](#)。然後繼續 [步驟 1：建立資料表](#)。
- 在 Linux 和 Windows 的終端機中使用 CLI 命令時存在差異。以下指南介紹針對 Linux 終端機 (包括 macOS) 格式化的命令，以及針對 Windows CMD 格式化的命令。請選擇最適合您正在使用的終端機應用程式的命令。

步驟 1：建立資料表

在此步驟中，您可以在 Amazon DynamoDB 中建立 Music 資料表。此資料表具有下列詳細資訊：

- 分割區索引鍵：Artist
- 排序索引鍵：SongTitle

如需資料表操作的詳細資訊，請參閱 [在 DynamoDB 中使用資料表和資料](#)。

Note

開始之前，請確定您已遵循 [事前準備 - 入門教學](#) 中的步驟。

AWS Management Console

若要使用 DynamoDB 主控台建立新的 Music 資料表：

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 選擇 建立資料表。
4. 輸入表格詳細資訊，如下所示：
 - a. 對於 Table name (資料表名稱)，請輸入 **Music**。
 - b. 對於 Partition key (分區索引鍵)，請輸入 **Artist**。
 - c. 對於「排序」鍵，請輸入 **SongTitle**。
5. 對於「表格」設定，請保留預設設定的預設選項。
6. 選擇「建立表格」以建立表格。

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.

Table 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.

7. 當表格處於ACTIVE狀態時，建議您執行下列步驟在表格[適用於 DynamoDB 的 Point-in-time 復原](#)上啟用：
 - a. 選擇表格名稱以開啟表格。
 - b. 選擇備份。
 - c. 在 [Point-in-time 復原 (PITR)] 區段中選擇 [編輯]。
 - d. 在 [編輯 point-in-time 復原設定] 頁面上，選擇 [開啟 point-in-time 復原]。
 - e. 選擇儲存變更。

AWS CLI

下列 AWS CLI 範例會使用建立新Music資料表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
```

使用 `create-table` 傳回以下範例結果。

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

請注意，TableStatus 欄位的值會設定為 CREATING。

若要確認 DynamoDB 已完成建立 Music 資料表，請使用 describe-table 命令。

Linux

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

Windows CMD

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

此命令會傳回下列結果。當 DynamoDB 已完成建立資料表時，TableStatus 欄位的值會設定為 ACTIVE。

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

一旦資料表進入 ACTIVE 狀態，根據最佳實務，應執行以下命令，以在資料表上啟用 [適用於 DynamoDB 的 Point-in-time 復原](#)：

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
```

此命令會傳回下列結果。

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

Note

透過 point-in-time 復原啟用連續備份有成本影響。如需有關定價的詳細資訊，請參閱 [Amazon DynamoDB 定價](#)。

在建立新的資料表之後，請繼續 [第 2 步：使用控制台或將數據寫入表 AWS CLI](#)。

第 2 步：使用控制台或將數據寫入表 AWS CLI

在此步驟中，您會將數個項目插入至在 [步驟 1：建立資料表](#) 中建立的 Music 資料表。

如需寫入操作的詳細資訊，請參閱 [寫入項目](#)。

AWS Management Console

請遵循下列步驟，使用 DynamoDB 主控台將資料寫入至 Music 資料表。

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 在「表格」頁面上，選擇音樂表。
4. 選擇 探索資料表項目。
5. 在 [退回的項目] 區段中，選擇 [建立項目]。
6. 在「創建物件」頁中，執行下列操作以將物件添加到您的表格中：
 - a. 選擇 Add new attribute (新增屬性)，接著選擇 Number (數字)。
 - b. 對於「屬性名稱」，輸入 **Awards**。
 - c. 重複此程序，建立類型 String (字串) 的 **AlbumTitle**。
 - d. 為您的項目輸入下列值：
 - i. 針對 Artist (藝人)，輸入 **No One You Know**。
 - ii. 針對 SongTitle，請輸入 **Call Me Today**。
 - iii. 針對 AlbumTitle，請輸入 **Somewhat Famous**。
 - iv. 針對 Awards (獎項)，輸入 **1**。
7. 選擇 Create item (建立項目)。
8. 重複此程序並利用下列值建立另一個項目：
 - a. 針對 Artist (藝人)，輸入 **Acme Band**。
 - b. 對於 SongTitle 輸入 **Happy Day**。
 - c. 針對 AlbumTitle，請輸入 **Songs About Life**。
 - d. 針對 Awards (獎項)，輸入 **10**。
9. 再執行一次，建立另一個具有與上一步驟相同 Artist (藝術家) 但其他屬性值不同的項目：
 - a. 針對 Artist (藝人)，輸入 **Acme Band**。
 - b. 對於 SongTitle 輸入 **PartiQL Rocks**。
 - c. 針對 AlbumTitle，請輸入 **Another Album Title**。
 - d. 針對 Awards (獎項)，輸入 **8**。

AWS CLI

下列 AWS CLI 範例會在表格中建立數個新項 Music 目。您可以透過 DynamoDB API 或 [PartiQL](#) (一種適用於 DynamoDB 的 SQL 相容查詢語言) 進行此操作。

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'}"
```

如需有關使用 PartiQL 寫入資料的詳細資訊，請參閱 [PartiQL Insert 陳述式](#)。

如需有關 DynamoDB 中支援之資料類型的詳細資訊，請參閱 [資料類型](#)。

如需有關如何以 JSON 表示 DynamoDB 資料類型的詳細資訊，請參閱 [屬性值](#)。

將資料寫入至您的資料表之後，請繼續 [步驟 3：從資料表讀取資料](#)。

步驟 3：從資料表讀取資料

在此步驟中，您將讀回您在中建立的其中一個項目 [第 2 步：使用控制台或將數據寫入表 AWS CLI](#)。您可以使用 DynamoDB 主控台或透過指定 Artist 和 AWS CLI 來讀取 Music 表格中的項目。SongTitle

如需 DynamoDB 中讀取操作的詳細資訊，請參閱 [讀取項目](#)。

AWS Management Console

請遵循下列步驟，使用 DynamoDB 主控台從 Music 資料表讀取資料。

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 在「表格」頁面上，選擇音樂表。
4. 選擇 探索資料表項目。
5. 在 [傳回的項目] 區段中，檢視儲存在表格中的項目清單，依 Artist 和排序 SongTitle。列表中的第一個項目是藝術家名為 Acme 樂隊和 SongTitle PartiQL 岩石的項目。

AWS CLI

下列 AWS CLI 範例會從讀取項目 `Music`。您可以透過 DynamoDB API 或 [PartiQL](#) (一種適用於 DynamoDB 的 SQL 相容查詢語言) 進行此操作。

DynamoDB API

Note

DynamoDB 的預設行為是最終一致讀取。`consistent-read` 參數在下面用來展示強烈一致讀取。

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  
  \"/>
```

使用 `get-item` 傳回以下範例結果。

```
{  
  "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'"
```

使用 PartiQL Select 陳述式傳回以下範例結果。

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

如需有關使用 PartiQL 讀取資料的詳細資訊，請參閱 [PartiQL Select 陳述式](#)。

若要更新資料表中的資料，請繼續 [步驟 4：更新資料表中的資料](#)。

步驟 4：更新資料表中的資料

在此步驟中，您會更新已在 [第 2 步：使用控制台或將數據寫入表 AWS CLI](#) 建立的項目。您可以使用 DynamoDB 主控台或透過指定 Artist、SongTitle 和更新 AlbumTitle 的 AWS CLI 來更新資料 Music 表中項目的更新。AlbumTitle

如需寫入操作的詳細資訊，請參閱 [寫入項目](#)。

AWS Management Console

您可以使用 DynamoDB 主控台，更新 Music 資料表中的資料。

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 從資料表清單中，選擇 Music (音樂) 資料表。
4. 選擇 探索資料表項目。
5. 在「退回的物品」中，對於包含 Acme Band 藝術家和「快樂日」的物品行 SongTitle，執行以下操作：
 - a. 將光標放在 AlbumTitle 命名的歌曲關於生活。
 - b. 選擇「編輯」圖示。
 - c. 在「編輯字串」快顯視窗中，輸入 **Songs of Twilight**。
 - d. 選擇儲存。

Tip

或者，若要更新項目，請在 [退回的項目] 區段中執行下列動作：

1. 選擇藝術家名為 Acme Band 並 SongTitle 命名為「快樂日」的項目行。
2. 從 [動作] 下拉式清單中選擇 [編輯項目]。
3. 若要輸入 AlbumTitle，請輸入 **Songs of Twilight**。
4. 選擇儲存與關閉。

AWS CLI

下列 AWS CLI 範例會更新 Music 表格中的項目。您可以透過 DynamoDB API 或 [PartiQL](#) (一種適用於 DynamoDB 的 SQL 相容查詢語言) 進行此操作。

DynamoDB API

Linux

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

Windows CMD

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

使用 update-item 傳回以下範例結果，因為已指定 return-values ALL_NEW。

```
{  
  "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  
*"
```

使用 Update 陳述式傳回以下範例結果，因為已指定 RETURNING ALL NEW *。

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

如需有關使用 PartiQL 更新資料的詳細資訊，請參閱 [PartiQL Update 陳述式](#)。

若要查詢 Music 資料表中的資料，請繼續 [步驟 5：查詢資料表中的資料](#)。

步驟 5：查詢資料表中的資料

在此步驟中，您會指定 Music，來查詢您已在 [the section called “步驟 2：寫入資料”](#) 寫入至 Artist 資料表的資料。這將顯示與分割區索引鍵：Artist 相關聯的所有歌曲。

如需查詢操作的詳細資訊，請參閱 [DynamoDB 中的查詢操作](#)。

AWS Management Console

請遵循下列步驟，使用 DynamoDB 主控台來查詢 Music 資料表中的資料。

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 從資料表清單中，選擇 Music (音樂) 資料表。
4. 選擇 探索資料表項目。
5. 在 [掃描或查詢項目] 中，確定已選取 [查詢]。
6. 針對 Partition key (分區索引鍵)，輸入 **Acme Band**，然後選擇 Run (執行)。

AWS CLI

下列 AWS CLI 範例會查詢資料 Music 表中的項目。您可以透過 DynamoDB API 或 [PartiQL](#) (一種適用於 DynamoDB 的 SQL 相容查詢語言) 進行此操作。

DynamoDB API

您可以藉由使用 query 並提供分割區索引鍵，透過 DynamoDB API 查詢項目。

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\"}}"
```

使用 query 傳回所有與此特定 Artist 相關聯的歌曲。

```
{
  "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

您可以藉由使用 Select 陳述式並提供分割區索引鍵，透過 PartiQL API 查詢項目。

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'"
```

以此方式使用 Select 陳述式會傳回所有與此特定 Artist 相關聯的歌曲。

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

```
]
}
```

如需有關使用 PartiQL 查詢資料的詳細資訊，請參閱 [PartiQL Select 陳述式](#)。

若要為您的資料表建立全域次要索引，請繼續[步驟 6：建立全域次要索引](#)。

步驟 6：建立全域次要索引

在此步驟中，您會針對已在 Music 建立的 [步驟 1：建立資料表](#) 資料表建立全域次要索引。

如需全域次要索引的詳細資訊，請參閱 [在 DynamoDB 中使用全域次要索引](#)。

AWS Management Console

使用 Amazon DynamoDB 主控台，為 Music 資料表建立全域次要索引 AlbumTitle-index：

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 從資料表清單中，選擇 Music (音樂) 資料表。
4. 為 Music (音樂) 資料表選擇 Indexes (索引) 標籤。
5. 選擇 Create index (建立索引)。
6. 在 [建立全域次要索引] 頁面上，執行下列動作：
 - a. 針對 Partition key (分割區索引鍵)，輸入 **AlbumTitle**。
 - b. 對於 Index name (索引名稱)，輸入 **AlbumTitle-index**。
 - c. 保留頁面上其他設定的預設選項，然後選擇 [建立索引]。

AWS CLI

下列 AWS CLI 範例會使用建立 Music 資料表 AlbumTitle-index 的全域次要索引 update-table。

Linux

```
aws dynamodb update-table \  
  --table-name Music \  
  --index-name AlbumTitle-index
```

```
--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"}}}]"
```

使用 `update-table` 傳回以下範例結果。

```
{
  "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
}
}
```

請注意，IndexStatus 欄位的值會設定為 CREATING。

若要確認 DynamoDB 已完成建立 AlbumTitle-index 全域次要索引，請使用 describe-table 命令。

Linux

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

Windows CMD

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

此命令會傳回下列結果。當傳回的 IndexStatus 欄位值設定為 ACTIVE 時，表示索引已備妥可供使用。

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

接著，您可以查詢全域次要索引。如需詳細資訊，請參閱 [步驟 7：查詢全域次要索引](#)。

步驟 7：查詢全域次要索引

在此步驟中，您會使用 Amazon DynamoDB 主控台或 AWS CLI，查詢 Music 資料表上的全域次要索引。

如需全域次要索引的詳細資訊，請參閱 [在 DynamoDB 中使用全域次要索引](#)。

AWS Management Console

請遵循下列步驟來使用 DynamoDB 主控台，透過 AlbumTitle-index 全域次要索引查詢資料。

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 從資料表清單中，選擇 Music (音樂) 資料表。
4. 選擇 探索資料表項目。
5. 在 [掃描或查詢項目] 中，保留 [查詢] 的預設選項。
6. 對於選取表格或索引，請選擇 AlbumTitle-index。
7. 對於 AlbumTitle，請輸入 **Somewhat Famous**，然後選擇 [執行]。

AWS CLI

下列 AWS CLI 範例會查詢 Music 資料表 AlbumTitle-index 上的全域次要索引。您可以透過 DynamoDB API 或 [PartiQL](#) (一種適用於 DynamoDB 的 SQL 相容查詢語言) 進行此操作。

DynamoDB API

您可以藉由使用 `query` 並提供索引名稱，透過 DynamoDB API 查詢全域次要索引。

Linux

```
aws dynamodb query \  
  --table-name Music \  
  --index-name 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\"}}"
```

使用 `query` 傳回以下範例結果。

```
{  
  "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

您可以藉由使用 Select 陳述式並提供索引名稱，透過 PartiQL 查詢全域次要索引。

Note

您將需要逸出包圍 Music 和 AlbumTitle-index 的雙引號，因為您是透過 CLI 執行這項作業。

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'"
```

以此方式使用 Select 陳述式會傳回以下範例結果。

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

如需有關使用 PartiQL 查詢資料的詳細資訊，請參閱 [PartiQL Select 陳述式](#)。

步驟 8：(選用) 清除資源

如果您不再需要針對此教學課程建立的 Amazon DynamoDB 資料表，則可以將其刪除。此步驟協助確保您不會為了未使用的資源而付費。您可以使用 DynamoDB 主控台或刪除您在 AWS CLI 中建立的 Music 資料表。 [步驟 1：建立資料表](#)

如需 DynamoDB 中資料表操作的詳細資訊，請參閱 [在 DynamoDB 中使用資料表和資料](#)。

AWS Management Console

若要使用主控台刪除 Music 資料表：

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 Tables (資料表)。
3. 選擇表格清單中「音樂」表旁邊的核取方塊。
4. 選擇刪除。

AWS CLI

下列 AWS CLI 範例會使用刪除 Music 資料表 delete-table。

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

DynamoDB 入門：後續步驟

如需使用 Amazon DynamoDB 的詳細資訊，請參閱下列主題：

- [在 DynamoDB 中使用資料表和資料](#)
- [使用項目和屬性](#)
- [DynamoDB 中的查詢操作](#)
- [在 DynamoDB 中使用全域次要索引](#)
- [搭配交易使用](#)
- [使用 DynamoDB Accelerator \(DAX\) 的記憶體內加速](#)
- [開始使用 DynamoDB 和開發套件 AWS](#)
- [使用 DynamoDB 和開發套件進程式設計 AWS](#)

開始使用 DynamoDB 和開發套件 AWS

使用本節中的實作教學課程，開始使用 Amazon DynamoDB 和開發套件 AWS。您可於可下載版 DynamoDB 或 DynamoDB Web 服務執行該程式碼範例。

主題

- [建立 DynamoDB 資料表](#)
- [將項目寫入 DynamoDB 資料表](#)
- [從 DynamoDB 資料表讀取項目](#)
- [更新 DynamoDB 資料表中的項目](#)
- [從 DynamoDB 資料表中刪除一個項目](#)
- [查詢 DynamoDB 資料表](#)
- [掃描 DynamoDB 資料表](#)
- [搭配開發套件使用 DynamoDB AWS](#)

建立 DynamoDB 資料表

您可以使用 AWS Management Console、或 AWS SDK 建 AWS CLI 立資料表。如需資料表的詳細資訊，請參閱[Amazon DynamoDB 的核心元件](#)。

使用 AWS SDK 建立 DynamoDB 資料表

以下程式碼範例示範如何使用 AWS SDK 建立 DynamoDB 資料表。

.NET

AWS SDK for .NET

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執程式碼範例儲存庫](#)。

```
///<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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[CreateTable](#)中的。

Bash

AWS CLI 與 bash 腳本

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。


```
#####
# 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
    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
}

```

此範例中使用的公用程式函數。

```

#####
# 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
}

```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[CreateTable](#)中的。

C++

適用於 C++ 的 SDK

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
#!/ 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[CreateTable](#)中的。

CLI

AWS CLI

範例 1：若要建立含標籤的資料表

下列create-table範例會使用指定的屬性和索引鍵結構描述來建立名為的資料表MusicCollection。此表格使用佈建的輸送量，並使用預設 AWS 擁有的 CMK 進行靜態加密。此指令也會將標籤套用至資料表，鍵值為blueTeam。Owner

```
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
```

輸出：

```
{
  "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"
  }
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

範例 2：在隨選模式中建立表格

下列範例會建立名為MusicCollection使用隨選模式而非佈建輸送量模式的資料表。這對於工作負載不可預測的表格非常有用。

```
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
```

輸出：

```
{  
  "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"
    }
  }
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

範例 3：建立資料表並使用客戶管理的 CMK 加密資料表

下列範例會建立名為的資料表，MusicCollection並使用客戶管理的 CMK 加密資料表。

```

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

```

輸出：

```

{
  "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"
    }
  }
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

範例 4：若要建立具有本機次要索引的資料表

下列範例會使用指定的屬性和索引鍵結構描述，建立名為 MusicCollection 本機次要索引的資料表 AlbumTitleIndex。

```

aws dynamodb create-table \
  --table-name MusicCollection \
  --attribute-definitions AttributeName=Artist,AttributeType=S
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\"]
      }
    }
  ]"

```

輸出：

```

{
  "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"
    }
]
}

```

```
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

範例 5：若要建立具有全域次要索引的資料表

下列範例會建立名 GameScores 為全域次要索引的資料表 GameTitleIndex。基礎資料表具有分割區索引鍵 UserId 和排序索引鍵 GameTitle，可讓您有效率地找到個別使用者在特定遊戲中的最佳分數，而 GSI 具有分割區索引鍵 GameTitle 和排序索引鍵 TopScore，可讓您快速找到特定遊戲的整體最高分數。

```
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  
        }  
      }  
    ]"
```

輸出：

```
{  
  "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"
  }
]
}
}
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

範例 6：一次建立含有多個全域次要索引的表格

下列範例會建立一個以兩個全域次要索引命名 GameScores 的資料表。GSI 結構描述是透過檔案傳遞的，而不是透過命令列傳遞。

```

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

```

gsi.json 的內容：

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


輸出：

```
{
  "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"
    }
]
}
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

範例 7：建立啟用「串流」的表格

下列範例會建立一個名為 GameScores 用 DynamoDB Streams 的資料表。每個項目的新舊圖像都將被寫入流中。

```

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

```

輸出：

```

{
  "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"
}
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

範例 8：若要建立已啟用「僅限金鑰串流」的資料表

下列範例會建立一個名為 GameScores 用 DynamoDB Streams 的資料表。只有修改項目的索引鍵屬性會寫入串流。

```
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
```

輸出：

```
{
  "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
}
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的變更 DynamoDB [串流的資料擷取](#)。

範例 9：若要建立具有標準不常存取類別的資料表

下列範例會建立名為的資料表，GameScores並指派標準不常存取 (DynamoDB 標準 — IA) 表類別。此表類別針對主要成本的儲存進行了最佳化。

```

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

```

輸出：

```

{
  "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
}
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格類別](#)。

範例 10：建立啟用刪除保護的資料表

下列範例會建立名為的資料表，GameScores 並啟用刪除保護。

```
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
```

輸出：

```
{
  "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
  }
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[使用刪除保護](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考[CreateTable](#)中的。

Go

SDK for Go V2

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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
}

```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考 [CreateTable](#) 中的。

Java

適用於 Java 2.x 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```

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 "";  
  }  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[CreateTable](#)中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
};
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [CreateTable](#) 中的。

適用於 JavaScript (v2) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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);
  }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [CreateTable](#) 中的。

Kotlin

適用於 Kotlin 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [CreateTable](#) 中的 Kotlin API 參考。

PHP

適用於 PHP 的開發套件

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

建立 資料表。

```
$tableName = "ddb_demo_table_${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],
    ]);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考 [CreateTable](#) 中的。

PowerShell

用於的工具 PowerShell

範例 1：此範例會建立名為 Thread 的資料表，其主索引鍵由 'ForumName' (雜湊金鑰類型) 和 'Subject' (索引鍵類型範圍) 組成。用來建構資料表的結構描述可以傳送至每個指令程式，如所示或使用 -Schema 參數指定。

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

輸出：

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

範例 2：此範例會建立名為 Thread 的資料表，其主索引鍵由 'ForumName' (雜湊金鑰類型) 和 'Subject' (索引鍵類型範圍) 組成。也會定義本機次要索引。本地輔助索引的鍵將從表上的主哈希鍵自動設置 (ForumName)。用來建構資料表的結構描述可以傳送至每個指令程式，如所示或使用 -Schema 參數指定。

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

輸出：

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

範例 3：此範例顯示如何使用單一管線建立名為 Thread 的資料表，其主索引鍵由 'ForumName' (索引鍵類型 hash) 和 'Subject' (索引鍵類型範圍) 和本機次要索引組成。如果沒有從管線或-Schema 參數提供，Add-DDB KeySchema 和 Add-DDB 會為您IndexSchema 建立新 TableSchema 物件。

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

輸出：

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

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 指令程[CreateTable](#)式參考中的。

Python

適用於 Python (Boto3) 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

建立用於存放電影資料的資料表。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[CreateTable](#)中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
# 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
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考 [CreateTable](#) 中的。

Rust

適用於 Rust 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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))
        }
    }
}

```

- 如需 API 的詳細資訊，請參閱 AWS SDK [CreateTable](#) 中的 Rust API 參考資料。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```

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.

```

- 如需 API 詳細資訊，請參閱 AWS SDK [CreateTable](#) 中的 SAP ABAP API 參考資料。

Swift

適用於 Swift 的 SDK

Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

///

```
/// 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
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [CreateTable](#) 中的斯威夫特 API 參考。

如需更多的 DynamoDB 範例，請參閱 [使用開發套件的 DynamoDB 程式碼範例 AWS](#)。

將項目寫入 DynamoDB 資料表

您可以使用 AWS Management Console、或開發套件，將項目寫入 DynamoDB 資料表。AWS CLI AWS 如需項目的詳細資訊，請參閱 [Amazon DynamoDB 的核心元件](#)。

使用 AWS SDK 將項目寫入 DynamoDB 資料表

下列程式碼範例示範如何使用 AWS SDK 將項目寫入 DynamoDB 資料表。

.NET

AWS SDK for .NET

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// <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;
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考 [PutItem](#) 中的。

Bash

AWS CLI 使用 Bash 腳本

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
# 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

}

```

此範例中使用的公用程式函數。

```

#####
# 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
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[PutItem](#)中的。

C++

適用於 C++ 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考 [PutItem](#) 中的。

CLI

AWS CLI

範例 1：若要將項目新增至表格

下列 `put-item` 範例會將新項目新增至 `MusicCollection` 表格中。

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

`item.json` 的內容：

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

輸出：

```
{  
  "ConsumedCapacity": {  
    "TableName": "MusicCollection",
```

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

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[撰寫項目](#)。

範例 2：若要有條件地覆寫資料表中的項目

下列put-item範例只有在現有項目具有值為的AlbumTitle屬性時，才會覆寫MusicCollection表格中的Greatest Hits現有項目。該命令返回該項目的先前值。

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

item.json 的內容：

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

names.json 的內容：

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

```
}
```

values.json 的內容：

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

輸出：

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

如果密鑰已經存在，您應該看到以下輸出：

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

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[撰寫項目](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考 [PutItem](#) 中的。

Go

SDK for Go V2

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考 [PutItem](#) 中的。

Java

適用於 Java 2.x 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

使用將項目放入表格中 [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);
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考 [PutItem](#) 中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [PutItem](#) 中的。

適用於 JavaScript (v2) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

將項目放入資料表。

```
// 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);
    }
  });
});
```

使用 DynamoDB 文件用戶端將項目放入資料表。

```
// 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);
  }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [PutItem](#) 中的。

Kotlin

適用於 Kotlin 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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.")
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [PutItem](#) 中的 Kotlin API 參考。

PHP

適用於 PHP 的開發套件

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考[PutItem](#)中的。

PowerShell

適用的工具 PowerShell

範例 1：建立新料號，或以新料號取代現有料號。

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

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 指令程 [PutItem](#) 式參考中的。

Python

適用於 Python (Boto3) 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[PutItem](#)中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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

```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考 [PutItem](#) 中的。

Rust

適用於 Rust 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```

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,
})
}
```

- 如需 API 的詳細資訊，請參閱 AWS SDK [PutItem](#) 中的 Rust API 參考資料。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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.
```

- 如需 API 詳細資訊，請參閱 AWS SDK [PutItem](#) 中的 SAP ABAP API 參考資料。

Swift

適用於 Swift 的 SDK

Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [PutItem](#) 中的斯威夫特 API 參考。

如需更多的 DynamoDB 範例，請參閱 [使用開發套件的 DynamoDB 程式碼範例 AWS](#)。

從 DynamoDB 資料表讀取項目

您可以使用 AWS Management Console、或 SDK 讀取 DynamoDB 表格中的項目。AWS CLI AWS 如需項目的詳細資訊，請參閱 [Amazon DynamoDB 的核心元件](#)。

使用 AWS SDK 從 DynamoDB 資料表讀取項目

下列程式碼範例示範如何使用 AWS SDK 讀取 DynamoDB 資料表中的項目。

.NET

AWS SDK for .NET

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
    /// <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;
    }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考 [GetItem](#) 中的。

Bash

AWS CLI 與 Bash 腳本

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
# 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
}

```

此範例中使用的公用程式函數。

```

#####
# 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
}

```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[GetItem](#)中的。

C++

適用於 C++ 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。


```
//! 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();  
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[GetItem](#)中的。

CLI

AWS CLI

範例 1：若要讀取表格中的項目

下列 `get-item` 範例會從資料 `MusicCollection` 表擷取項目。資料表具有 `hash-and-range` 主索引鍵 (`Artist` 和 `SongTitle`)，因此您必須指定這兩個屬性。此命令也會要求作業所耗用之讀取容量的相關資訊。

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

`key.json` 的內容：

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

輸出：

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

```
    "Artist": {
      "S": "Acme Band"
    }
  },
  "ConsumedCapacity": {
    "TableName": "MusicCollection",
    "CapacityUnits": 0.5
  }
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的讀取項目](#)。

範例 2：若要使用一致讀取來讀取項目

下列範例會使用強烈一致的讀取，從資料 MusicCollection 表擷取項目。

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

key.json 的內容：

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

輸出：

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

```
    }
  },
  "ConsumedCapacity": {
    "TableName": "MusicCollection",
    "CapacityUnits": 1.0
  }
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的讀取項目](#)。

範例 3：擷取料號的特定屬性

下列範例會使用投影運算式，僅擷取所需項目的三個屬性。

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

names.json 的內容：

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

輸出：

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


```
}  
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的讀取項目](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考 [GetItem](#) 中的。

Go

SDK for Go V2

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考[GetItem](#)中的。

Java

適用於 Java 2.x 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用取得表格中的項目 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);
    }
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[GetItem](#)中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱[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;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [GetItem](#) 中的。

適用於 JavaScript (v2) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

從資料表取得項目。

```
// 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);
    }
  });
```

使用 DynamoDB 文件用戶端從資料表取得項目。

```
// 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);
  }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [GetItem](#) 中的。

Kotlin

適用於 Kotlin 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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)
        }
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [GetItem](#) 中的 Kotlin API 參考。

PHP

適用於 PHP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
$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,
    ]);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考[GetItem](#)中的。

PowerShell

適用的工具 PowerShell

範例 1：傳回含有分區索引鍵 SongTitle 和排序索引鍵演出者的 DynamoDB 項目。

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

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

輸出：

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

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 指令程[GetItem](#)式參考中的。

Python

適用於 Python (Boto3) 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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"]
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[GetItem](#)中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考[GetItem](#)中的。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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.
```

- 如需 API 詳細資訊，請參閱 AWS SDK [GetItem](#)中的 SAP ABAP API 參考資料。

Swift

適用於 Swift 的 SDK

Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [GetItem](#) 中的斯威夫特 API 參考。

如需更多的 DynamoDB 範例，請參閱 [使用開發套件的 DynamoDB 程式碼範例 AWS](#)。

更新 DynamoDB 資料表中的項目

您可以使用 AWS Management Console、或 SDK 更新 DynamoDB 資料表中的項目。AWS CLI AWS 如需項目的詳細資訊，請參閱 [Amazon DynamoDB 的核心元件](#)。

使用 AWS SDK 更新 DynamoDB 資料表中的項目

下列程式碼範例示範如何使用 AWS SDK 更新 DynamoDB 資料表中的項目。

.NET

AWS SDK for .NET

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
    /// <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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考 [UpdateItem](#) 中的。

Bash

AWS CLI 與 bash 腳本

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
# 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 getopt "n:k:e:v:h" option; do
        case "${option}" in
            n) table_name="${OPTARG}" ;;
            k) keys="${OPTARG}" ;;
            e) update_expression="${OPTARG}" ;;
            v) values="${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 "$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
}

```

此範例中使用的公用程式函數。

```

#####
# 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
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[UpdateItem](#)中的。

C++

適用於 C++ 的 SDK

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
#!/ 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考 [UpdateItem](#) 中的。

CLI

AWS CLI

範例 1：若要更新表格中的項目

下列 `update-item` 範例會更新 `MusicCollection` 資料表中的項目。它增加了一個新的屬性 (`Year`) 並修改 `AlbumTitle` 屬性。在回應中傳回項目中的所有屬性，就像它們在更新之後所顯示的那樣。

```
aws dynamodb update-item \  
  --table-name MusicCollection \  
  --key file://key.json \  
  --update-expression "SET #Y = :y, #AT = :t" \  
  --expression-attribute-names file://expression-attribute-names.json \  
  --expression-attribute-values file://expression-attribute-values.json \  
  --return-values ALL_NEW \  
  --return-consumed-capacity TOTAL \  
  --return-item-collection-metrics SIZE
```

`key.json` 的內容：

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

`expression-attribute-names.json` 的內容：

```
{  
  "#Y": "Year", "#AT": "AlbumTitle"  
}
```

`expression-attribute-values.json` 的內容：

```
{  
  ":y": {"N": "2015"},  
  ":t": {"S": "Louder Than Ever"}  
}
```

輸出：

```
{
  "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
  ]
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的撰寫項目](#)。

範例 2：若要有條件地更新料號

下列範例會更新 MusicCollection 表格中的項目，但前提是現有項目尚未具有 Year 屬性時。

```
aws dynamodb update-item \  
  --table-name MusicCollection \  
  --condition-expression "attribute_not_exists(year)"
```

```
--key file://key.json \  
--update-expression "SET #Y = :y, #AT = :t" \  
--expression-attribute-names file://expression-attribute-names.json \  
--expression-attribute-values file://expression-attribute-values.json \  
--condition-expression "attribute_not_exists(#Y)"
```

key.json 的內容：

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

expression-attribute-names.json 的內容：

```
{  
  "#Y": "Year",  
  "#AT": "AlbumTitle"  
}
```

expression-attribute-values.json 的內容：

```
{  
  ":y": {"N": "2015"},  
  ":t": {"S": "Louder Than Ever"}  
}
```

如果項目已具有Year屬性，DynamoDB 會傳回下列輸出。


```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem  
operation: The conditional request failed
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[撰寫項目](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考 [UpdateItem](#) 中的。

Go

SDK for Go V2

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考 [UpdateItem](#) 中的。

Java

適用於 Java 2.x 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

使用更新表格中的項目 [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!");
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[UpdateItem](#)中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [UpdateItem](#) 中的。

Kotlin

適用於 Kotlin 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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")
    }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [UpdateItem](#) 中的 Kotlin API 參考。

PHP

適用於 PHP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
    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
                ]
            ],
        ]);
    }
}

```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考 [UpdateItem](#) 中的。

PowerShell

用於的工具 PowerShell

範例 1：使用分區索引鍵 `SongTitle` 和排序索引鍵 `演出者`，將 DynamoDB 項目上的類型屬性設定為「Rap」。

```

$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
}

```

```

} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem

```

輸出：

Name	Value
----	-----
Genre	Rap

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 指令程[UpdateItem](#)式參考中的。

Python

適用於 Python (Boto3) 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用更新表達式更新項目。

```

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"]
```

使用包含算術運算的更新表達式更新項目。

```
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"]
```

僅在項目符合特定條件時更新項目。

```
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"]
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[UpdateItem](#)中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
  end
end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考[UpdateItem](#)中的。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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.
```

- 如需 API 詳細資訊，請參閱 AWS SDK [UpdateItem](#) 中的 SAP ABAP API 參考資料。

Swift

適用於 Swift 的 SDK

Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [UpdateItem](#) 中的斯威夫特 API 參考。

如需更多的 DynamoDB 範例，請參閱 [使用開發套件的 DynamoDB 程式碼範例 AWS](#)。

從 DynamoDB 資料表中刪除一個項目

您可以使用 AWS Management Console、或 SDK 從 DynamoDB 資料表中刪除項目。AWS CLI AWS 如需項目的詳細資訊，請參閱 [Amazon DynamoDB 的核心元件](#)。

使用 AWS SDK 刪除 DynamoDB 資料表中的項目

下列程式碼範例示範如何使用 AWS SDK 刪除 DynamoDB 資料表中的項目。

.NET

AWS SDK for .NET

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// <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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考 [DeleteItem](#) 中的。

Bash

AWS CLI 與 bash 腳本

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
# 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

}

```

此範例中使用的公用程式函數。

```

#####
# 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
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[DeleteItem](#)中的。

C++

適用於 C++ 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[DeleteItem](#)中的。

CLI

AWS CLI

範例 1：若要刪除項目

下列delete-item範例會從MusicCollection表格中刪除項目，並要求有關已刪除項目以及請求所使用容量的詳細資訊。

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

key.json 的內容：

```
{  
  "Artist": {"S": "No One You Know"},  
  "SongTitle": {"S": "Scared of My Shadow"}  
}
```

輸出：

```
{  
  "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
    ]
}
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[撰寫項目](#)。

範例 2：若要有條件地刪除項目

下列範例只會刪除 ProductCatalog 表格中的某 ProductCategory 個物件，Sporting Goods 或 Gardening Supplies 且其價格在 500 到 600 之間。它返回有關被刪除的項目的詳細信息。

```

aws dynamodb delete-item \
  --table-name ProductCatalog \
  --key '{"Id":{"N":"456"}}' \
  --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (#P
  between :lo and :hi)" \
  --expression-attribute-names file://names.json \
  --expression-attribute-values file://values.json \
  --return-values ALL_OLD

```

names.json 的內容：

```

{
  "#P": "Price"
}

```

values.json 的內容：

```

{
  ":cat1": {"S": "Sporting Goods"},
  ":cat2": {"S": "Gardening Supplies"},
  ":lo": {"N": "500"},
  ":hi": {"N": "600"}
}

```

輸出：

```

{
  "Attributes": {

```

```
    "Id": {
      "N": "456"
    },
    "Price": {
      "N": "550"
    },
    "ProductCategory": {
      "S": "Sporting Goods"
    }
  }
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[撰寫項目](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考 [DeleteItem](#) 中的。

Go

SDK for Go V2

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考 [DeleteItem](#) 中的。

Java

適用於 Java 2.x 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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);
    }
}
}
```


- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考 [DeleteItem](#) 中的。

JavaScript

適用於 JavaScript (v3) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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;
};
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [DeleteItem](#) 中的。

適用於 JavaScript (v2) 的開發套件

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

從資料表刪除項目。

```
// 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);
  }
});
```

使用 DynamoDB 文件用戶端從資料表刪除項目。

```
// 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);
  }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [DeleteItem](#) 中的。

Kotlin

適用於 Kotlin 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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")
}
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [DeleteItem](#) 中的 Kotlin API 參考。

PHP

適用於 PHP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
$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,
    ]);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考 [DeleteItem](#) 中的。

PowerShell

適用的工具 PowerShell

範例 1：移除符合所提供金鑰的 DynamoDB 項目。

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 指令程 [DeleteItem](#) 式參考中的。

Python

適用於 Python (Boto3) 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

您可以指定條件，僅在符合特定條件時刪除項目。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[DeleteItem](#)中的 Python (博托 3) API 參考。

Ruby

適用於 Ruby 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
end
```

```
# Deletes a movie from the table.
#
# @param title [String] The title of the movie to delete.
# @param year [Integer] The release year of the movie to delete.
def delete_item(title, year)
  @table.delete_item(key: {"year" => year, "title" => title})
rescue Aws::DynamoDB::Errors::ServiceError => e
  puts("Couldn't delete movie #{title}. Here's why:")
  puts("\t#{e.code}: #{e.message}")
  raise
end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考 [DeleteItem](#) 中的。

Rust

適用於 Rust 的 SDK

Note

還有更多關於 [GitHub](#)。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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)),
  }
}
```

- 如需 API 的詳細資訊，請參閱 AWS SDK [DeleteItem](#) 中的 Rust API 參考資料。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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.
```

- 如需 API 詳細資訊，請參閱 AWS SDK [DeleteItem](#) 中的 SAP ABAP API 參考資料。

Swift

適用於 Swift 的 SDK

Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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)
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [DeleteItem](#) 中的斯威夫特 API 參考。

如需更多的 DynamoDB 範例，請參閱 [使用開發套件的 DynamoDB 程式碼範例 AWS](#)。

查詢 DynamoDB 資料表

您可以使用 AWS Management Console、或 SDK 在 DynamoDB 資料表上執行 AWS CLI 查詢。AWS 如需查詢的詳細資訊，請參閱[DynamoDB 中的查詢操作](#)。

使用 AWS SDK 查詢 DynamoDB 資料表

以下程式碼範例示範如何使用 AWS SDK 查詢 DynamoDB 資料表。

.NET

AWS SDK for .NET

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// <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;
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for .NET API 參考》中的 [Query](#)。

Bash

AWS CLI 與 bash 腳本

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
# 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
}

```

此範例中使用的公用程式函數。

```

#####
# 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
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [Query](#)。

C++

適用於 C++ 的 SDK

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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;
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for C++ API 參考》中的 [Query](#)。

CLI

AWS CLI

範例 1：若要查詢資料表

下列query範例會查詢MusicCollection資料表中的項目。該表具有 hash-and-range 主鍵 (Artist和SongTitle)，但此查詢僅指定哈希鍵值。它返回名為「沒有人你認識」的藝術家的歌曲標題。

```
aws dynamodb query \
  --table-name MusicCollection \
  --projection-expression "SongTitle" \
  --key-condition-expression "Artist = :v1" \
  --expression-attribute-values file://expression-attributes.json \
  --return-consumed-capacity TOTAL
```

expression-attributes.json 的內容：

```
{
```

```
":v1": {"S": "No One You Know"}
}
```

輸出：

```
{
  "Items": [
    {
      "SongTitle": {
        "S": "Call Me Today"
      },
      "SongTitle": {
        "S": "Scared of My Shadow"
      }
    }
  ],
  "Count": 2,
  "ScannedCount": 2,
  "ConsumedCapacity": {
    "TableName": "MusicCollection",
    "CapacityUnits": 0.5
  }
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的使用 DynamoDB 中的查詢](#)。

範例 2：若要使用強烈一致的讀取來查詢資料表，並以遞減順序遍歷索引

下列範例會執行與第一個範例相同的查詢，但會以相反順序傳回結果，並使用強式一致的讀取。

```
aws dynamodb query \
  --table-name MusicCollection \
  --projection-expression "SongTitle" \
  --key-condition-expression "Artist = :v1" \
  --expression-attribute-values file://expression-attributes.json \
  --consistent-read \
  --no-scan-index-forward \
  --return-consumed-capacity TOTAL
```

expression-attributes.json 的內容：

```
{
```

```
":v1": {"S": "No One You Know"}
}
```

輸出：

```
{
  "Items": [
    {
      "SongTitle": {
        "S": "Scared of My Shadow"
      }
    },
    {
      "SongTitle": {
        "S": "Call Me Today"
      }
    }
  ],
  "Count": 2,
  "ScannedCount": 2,
  "ConsumedCapacity": {
    "TableName": "MusicCollection",
    "CapacityUnits": 1.0
  }
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的使用 DynamoDB 中的查詢](#)。

範例 3：篩選出特定結果

下列範例會查詢 MusicCollection 但排除 AlbumTitle 屬性中具有特定值的結果。請注意，這不會影響 ScannedCount 或 ConsumedCapacity，因為篩選器會在項目讀取之後套用。

```
aws dynamodb query \
  --table-name MusicCollection \
  --key-condition-expression "#n1 = :v1" \
  --filter-expression "NOT (#n2 IN (:v2, :v3))" \
  --expression-attribute-names file://names.json \
  --expression-attribute-values file://values.json \
  --return-consumed-capacity TOTAL
```

values.json 的內容：

```
{
  ":v1": {"S": "No One You Know"},
  ":v2": {"S": "Blue Sky Blues"},
  ":v3": {"S": "Greatest Hits"}
}
```

names.json 的內容：

```
{
  "#n1": "Artist",
  "#n2": "AlbumTitle"
}
```

輸出：

```
{
  "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
  }
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的使用 DynamoDB 中的查詢](#)。

範例 4：僅擷取料號計數

下列範例會擷取符合查詢的項目計數，但不會擷取任何項目本身。

```
aws dynamodb query \  
  --table-name MusicCollection \  
  --select COUNT \  
  --key-condition-expression "Artist = :v1" \  
  --expression-attribute-values file://expression-attributes.json
```

expression-attributes.json 的內容：

```
{  
  ":v1": {"S": "No One You Know"}  
}
```

輸出：

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

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的使用 DynamoDB 中的查詢](#)。

範例 5：若要查詢索引

下列範例會查詢本機次要索引 AlbumTitleIndex。查詢會傳回已投影至本機次要索引的基底資料表中的所有屬性。請注意，查詢本機次要索引或全域次要索引時，您也必須使用 table-name 參數提供基底資料表的名稱。

```
aws dynamodb query \  
  --table-name MusicCollection \  
  --index-name AlbumTitleIndex \  
  --key-condition-expression "Artist = :v1" \  
  --expression-attribute-values file://expression-attributes.json \  
  --select ALL_PROJECTED_ATTRIBUTES \  
  --return-consumed-capacity INDEXES
```

expression-attributes.json 的內容：

```
{  
  ":v1": {"S": "No One You Know"}  
}
```

```
}
```

輸出：

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



```
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南](#) 中的 [使用 DynamoDB 中的查詢](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [Query](#)。

Go

SDK for Go V2

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for Go API 參考》](#) 中的 Query。

Java

適用於 Java 2.x 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

使用查詢表格 [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;
}
```

```
}
```

使用 `DynamoDbClient` 和次要索引查詢資料表。

```
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);
    }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 [Query](#)。

JavaScript

適用於 JavaScript (v3) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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;
};
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 的詳細資訊，請參閱 [《AWS SDK for JavaScript API 參考》](#) 中的 Query。

適用於 JavaScript (v2) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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);
  }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 的詳細資訊，請參閱 [《AWS SDK for JavaScript API 參考》](#) 中的 Query。

Kotlin

適用於 Kotlin 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
    }
  }
}

```

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的 AWS SDK API 參考》中的 [Query](#)。

PHP

適用於 PHP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```

$birthKey = [
    'Key' => [
        'year' => [
            'N' => "$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);
}

```

- 如需 API 的詳細資訊，請參閱《AWS SDK for PHP API 參考》中的 [Query](#)。

PowerShell

用於的工具 PowerShell

範例 1：叫用傳回具有指定和演出者的 DynamoDB 項目 SongTitle 的查詢。

```

$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    }
}

```

```

    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem

```

輸出：

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

- 如需 API 詳細資訊，請參閱在AWS Tools for PowerShell 指令程式參考中[查詢](#)。

Python

適用於 Python (Boto3) 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用索引鍵條件表達式查詢項目。

```

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"]
```

查詢項目並投影以傳回資料子集。

```
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"]
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的 AWS SDK API 參考》中的 [Query](#)。

Ruby

適用於 Ruby 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for Ruby API 參考》](#) 中的 Query。

Rust

適用於 Rust 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

尋找在指定年份製作的電影。

```
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![])
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [Query](#)。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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_attributevalueulist = 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.

```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的 AWS SDK API 參考》中的 [Query](#)。

Swift

適用於 Swift 的 SDK

Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Swift 的 AWS SDK API 參考》中的 [Query](#)。

如需更多的 DynamoDB 範例，請參閱 [使用開發套件的 DynamoDB 程式碼範例 AWS](#)。

掃描 DynamoDB 資料表

您可以使用 AWS Management Console、或 SDK 在 DynamoDB 資料表上執行 AWS CLI 掃描。AWS 如需掃描的詳細資訊，請參閱 [在 DynamoDB 中使用掃描](#)。

使用 AWS SDK 掃描 DynamoDB 資料表

以下程式碼範例示範如何使用 AWS SDK 掃描 DynamoDB 資料表。

.NET

AWS SDK for .NET

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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;
}
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for .NET API 參考》](#) 中的 Scan。

Bash

AWS CLI 與 bash 腳本

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
# 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}" ;;
            h)
                usage
                return 0
                ;;
        esac
    done
}

```

```
\?)
    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
}

```

此範例中使用的公用程式函數。

```

#####
# 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
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [Scan](#)。

C++

適用於 C++ 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
//! Scan an Amazon DynamoDB table.
/*!
    \sa scanTable()
    \param tableName: Name for the DynamoDB table.
```



```
\param projectionExpression: An optional projection expression, ignored if
empty.
\param clientConfiguration: AWS client configuration.
\return bool: Function succeeded.
*/

bool AwsDoc::DynamoDB::scanTable(const Aws::String &tableName,
                                const Aws::String &projectionExpression,
                                const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::ScanRequest request;
    request.SetTableName(tableName);

    if (!projectionExpression.empty())
        request.SetProjectionExpression(projectionExpression);

    Aws::Vector<Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>>
all_items;
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
last_evaluated_key; // Used for pagination;
    do {
        if (!last_evaluated_key.empty()) {
            request.SetExclusiveStartKey(last_evaluated_key);
        }
        const Aws::DynamoDB::Model::ScanOutcome &outcome =
dynamoClient.Scan(request);
        if (outcome.IsSuccess()) {
            // Reference the retrieved items.
            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
            all_items.insert(all_items.end(), items.begin(), items.end());

            last_evaluated_key = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Scan items: " <<
outcome.GetError().GetMessage()
                << std::endl;
            return false;
        }
    } while (!last_evaluated_key.empty());
}
```

```
if (!all_items.empty()) {
    std::cout << "Number of items retrieved from scan: " << all_items.size()
              << std::endl;
    // Iterate each item and print.
    for (const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&itemMap: all_items) {
        std::cout << "*****"
                  << std::endl;
        // Output each retrieved field and its value.
        for (const auto &itemEntry: itemMap)
            std::cout << itemEntry.first << ": " << itemEntry.second.GetS()
                    << std::endl;
    }
}

else {
    std::cout << "No items found in table: " << tableName << std::endl;
}

return true;
}
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for C++ API 參考》](#) 中的 Scan。

CLI

AWS CLI

掃描表格

下列 scan 範例會掃描整個 MusicCollection 表格，然後將結果縮小為演出者「沒有您認識的人」的歌曲。對於每個項目，僅返回專輯標題和歌曲標題。

```
aws dynamodb scan \
  --table-name MusicCollection \
  --filter-expression "Artist = :a" \
  --projection-expression "#ST, #AT" \
  --expression-attribute-names file://expression-attribute-names.json \
  --expression-attribute-values file://expression-attribute-values.json
```

expression-attribute-names.json 的內容：

```
{
  "#ST": "SongTitle",
  "#AT": "AlbumTitle"
}
```

expression-attribute-values.json 的內容：

```
{
  ":a": {"S": "No One You Know"}
}
```

輸出：

```
{
  "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
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的在 Dynamo DB 中使用掃描](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [Scan](#)。

Go

SDK for Go V2

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for Go API 參考》](#) 中的 Scan。

Java

適用於 Java 2.x 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

使用掃描 Amazon DynamoDB 表。 [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.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);
    }
}
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for Java 2.x API 參考》](#) 中的 Scan。

JavaScript

適用於 JavaScript (v3) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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;
};
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for JavaScript API 參考》](#) 中的 Scan。

適用於 JavaScript (v2) 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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 + ")"
      );
    });
  }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 的詳細資訊，請參閱 [《AWS SDK for JavaScript API 參考》](#) 中的 Scan。

Kotlin

適用於 Kotlin 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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]}")
        }
    }
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的 AWS SDK API 參考》中的 [Scan](#)。

PHP

適用於 PHP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
$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);
}

```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for PHP API 參考》](#) 中的 Scan。

PowerShell

適用的工具 PowerShell

範例 1：傳回「音樂」資料表中的所有項目。

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

輸出：

Name	Value
----	-----
Genre	Country
Artist	No One You Know
Price	1.94
CriticRating	9
SongTitle	Somewhere Down The Road
AlbumTitle	Somewhat Famous
Genre	Country
Artist	No One You Know
Price	1.98
CriticRating	8.4
SongTitle	My Dog Spot
AlbumTitle	Hey Now

範例 2：傳回「音樂」資料表中 CriticRating 大於或等於 9 的項目。

```
$scanFilter = @{
    CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
        AttributeValueList = @(@{N = '9'})
        ComparisonOperator = 'GE'
    }
}
Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

輸出：

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

- 如需 API 詳細資訊，請參閱在 AWS Tools for PowerShell 指令程式參考中 [掃描](#)。

Python

適用於 Python (Boto3) 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的 AWS SDK API 參考》中的 [Scan](#)。

Ruby

適用於 Ruby 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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?
    end
  end
end
```

```
    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
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for Ruby API 參考》](#) 中的 Scan。

Rust

適用於 Rust 的 SDK

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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(())
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [Scan](#)。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的 AWS SDK API 參考》中的 [Scan](#)。

Swift

適用於 Swift 的 SDK

Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的 AWS SDK API 參考》中的 [Scan](#)。


如需更多的 DynamoDB 範例，請參閱 [使用開發套件的 DynamoDB 程式碼範例 AWS](#)。

搭配開發套件使用 DynamoDB AWS

AWS 軟件開發套件 (SDK) 可用於許多流行的編程語言。每個 SDK 都提供 API、程式碼範例和說明文件，讓開發人員能夠更輕鬆地以偏好的語言建置應用程式。

SDK 文件	代碼範例
AWS SDK for C++	AWS SDK for C++ 程式碼範例
AWS CLI	AWS CLI 程式碼範例
AWS SDK for Go	AWS SDK for Go 程式碼範例
AWS SDK for Java	AWS SDK for Java 程式碼範例
AWS SDK for JavaScript	AWS SDK for JavaScript 程式碼範例
適用於 Kotlin 的 AWS SDK	適用於 Kotlin 的 AWS SDK 程式碼範例
AWS SDK for .NET	AWS SDK for .NET 程式碼範例
AWS SDK for PHP	AWS SDK for PHP 程式碼範例
AWS Tools for PowerShell	PowerShell 程式碼範例工具
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 程式碼範例
AWS SDK for Ruby	AWS SDK for Ruby 程式碼範例
適用於 Rust 的 AWS SDK	適用於 Rust 的 AWS SDK 程式碼範例
適用於 SAP ABAP 的 AWS SDK	適用於 SAP ABAP 的 AWS SDK 程式碼範例
適用於 Swift 的 AWS SDK	適用於 Swift 的 AWS SDK 程式碼範例

如需 DynamoDB 專屬範例，請參閱 [使用開發套件的 DynamoDB 程式碼範例 AWS](#)。

 可用性範例

找不到所需的內容嗎？請使用本頁面底部的提供意見回饋連結申請程式碼範例。

使用 DynamoDB 和開發套件進程式設計 AWS

本節涵蓋開發人員相關的主題。如果您要改為執程式碼範例，請參閱[執行此開發人員指南中的程式碼範例](#)。

Note

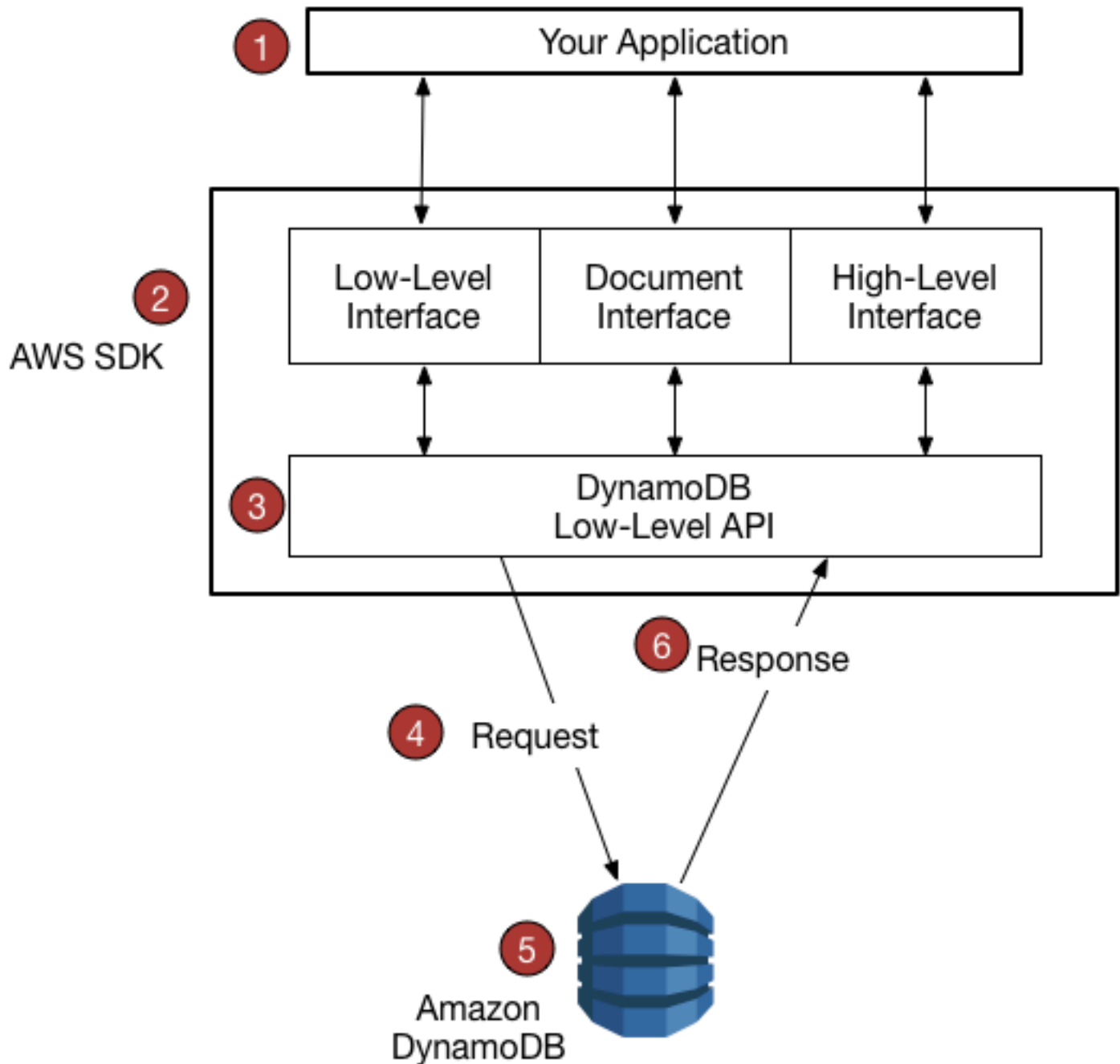
2017 年 12 月，AWS 開始遷移所有 Amazon DynamoDB 端點以使用 Amazon 信任服務 (ATS) 發行的安全憑證的程序。如需詳細資訊，請參閱[對 SSL/TLS 連線建立問題進行疑難排解](#)。

主題

- [適用於 AWS SDK 的 SDK DynamoDB 援概觀](#)
- [適用於 DynamoDB 的更高階程式設計界面](#)
- [執行此開發人員指南中的程式碼範例](#)
- [使用 Python 和肉毒桿菌 3 對 Amazon DynamoDB 進行編程](#)
- [使用 Amazon DynamoDB 進行編程 JavaScript](#)
- [使用 Amazon DynamoDB 進行編程 AWS SDK for Java 2.x](#)

適用於 AWS SDK 的 SDK DynamoDB 援概觀

下圖提供使用開發套件的 Amazon DynamoDB 應用程式程式設計的高階概觀。AWS



1. 您可以使用 AWS SDK 為您的程式設計語言撰寫應用程式。
2. 每個 AWS SDK 都提供一或多個程式設計介面，以便與 DynamoDB 搭配使用。可用的特定介面取決於您使用的程式設計語言和 AWS SDK。選項包括：
 - [低階界面](#)
 - [文件界面](#)
 - [物件持久性界面](#)
 - [高階介面](#)

3. 此開 AWS 發套件會建構 HTTP (S) 請求，以便與低階 DynamoDB API 搭配使用。
4. AWS SDK 會將請求傳送至 DynamoDB 端點。
5. DynamoDB 會執行請求。如果請求成功，DynamoDB 會傳回 HTTP 200 回應代碼 (OK)。如果請求不成功，DynamoDB 會傳回 HTTP 錯誤代碼和錯誤訊息。
6. AWS SDK 處理響應並將其傳播回您的應用程序。

每個 AWS SDK 都為您的應用程式提供重要的服務，包括下列項目：

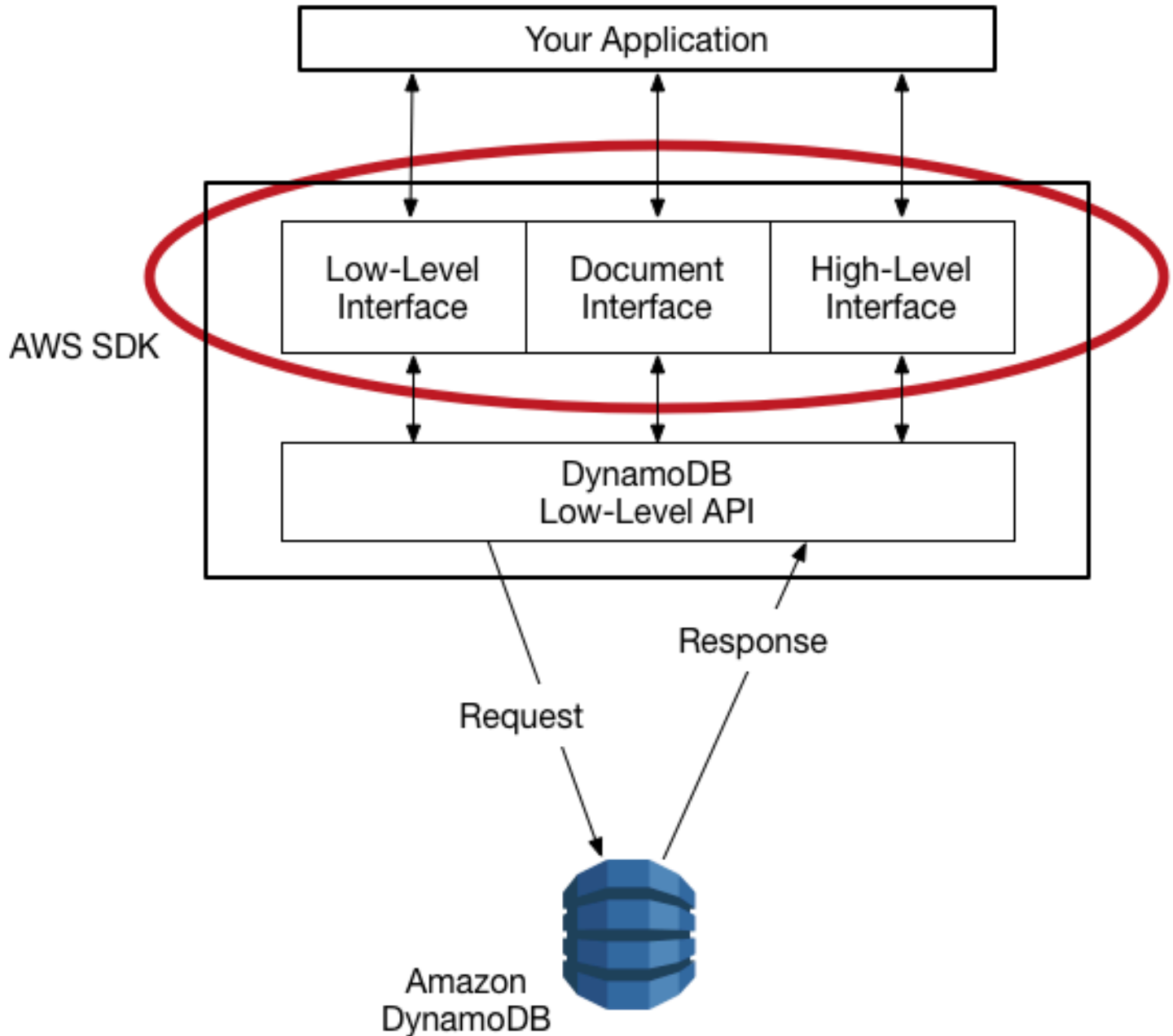
- 格式化 HTTP(S) 請求和序列化請求參數。
- 為每個請求產生密碼編譯簽章。
- 將請求轉寄至 DynamoDB 端點，並接收來自 DynamoDB 的回應。
- 從這些回應中提取結果。
- 在出現錯誤的情況下實作基本重試邏輯。

您不需要為上述任何任務編寫程式碼。

Note

如需有關 AWS SDK 的詳細資訊，包括安裝說明和文件，請參閱 [Amazon Web Services 的工具](#)。

程式設計界面



每個 [AWS 開發套件](#) 提供一或多個程式化界面，以便與 Amazon DynamoDB 搭配使用。這些界面範圍從簡單的低階 DynamoDB 包裝函式到物件導向的持久性層。可用的介面會根據您使用的 AWS SDK 和程式設計語言而有所不同。

下一節會以 AWS SDK for Java 為範例來重點介紹一些可用的界面。(並非全部界面都可用於所有 AWS 開發套件。)

主題

- [低階界面](#)
- [文件界面](#)
- [物件持久性界面](#)

低階界面

每個特定語言的 AWS 開發套件都為 Amazon DynamoDB 提供低階介面，其方法與低階 DynamoDB API 請求非常類似。

在某些情況下，您需要識別使用 [資料類型描述項](#) 的屬性資料類型，例如，適用於字串的 S 或適用於數字的 N。

Note

每種語言特定的 AWS 開發套件都提供低階界面。

下列 Java 程式使用 AWS SDK for Java 的低階界面。

```
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);
}
}
```

文件界面

許多 AWS SDK 提供了一個文檔界面，允許您對表和索引執行數據平面操作（創建，讀取，更新，刪除）。使用文件界面時，您不需要指定 [資料類型描述項](#)。資料類型是由資料本身的語義隱含的。這些 AWS 開發套件也提供方法，可輕鬆地將 JSON 文件轉換為原生 Amazon DynamoDB 資料類型，以及從原生資料類型轉換

Note

可在 [Java](#)、[.NET](#)、[Node.js](#) 和 [瀏覽器的 AWS 開發套件 JavaScript](#) 中使用文件介面。

下列 Java 程式使用 AWS SDK for Java 的文件界面。程式會建立代表 Music 資料表的 Table 物件，然後要求該物件使用 getItem 來擷取歌曲。然後程式會列印歌曲發行的年份。

com.amazonaws.services.dynamodbv2.document.DynamoDB 類別會實作 DynamoDB 文件界面。請注意 DynamoDB 作為低階用戶端 (AmazonDynamoDB) 包裝函式的方式。

```
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);

    }
}
```

物件持久性界面

某些 AWS SDK 提供了一個對象持久性接口，您不會直接執行數據平面操作。反之，您要建立代表 Amazon DynamoDB 資料表和索引中項目的物件，並且僅與這些物件互動。這允許您編寫以物件為中心的程式碼，而不是以資料庫為中心的程式碼。

Note

物件持續性介面可在 Java 和 .NET 的 AWS 開發套件中使用。如需詳細資訊，請參閱 [適用於 DynamoDB 的更高階程式設計界面](#) for 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();

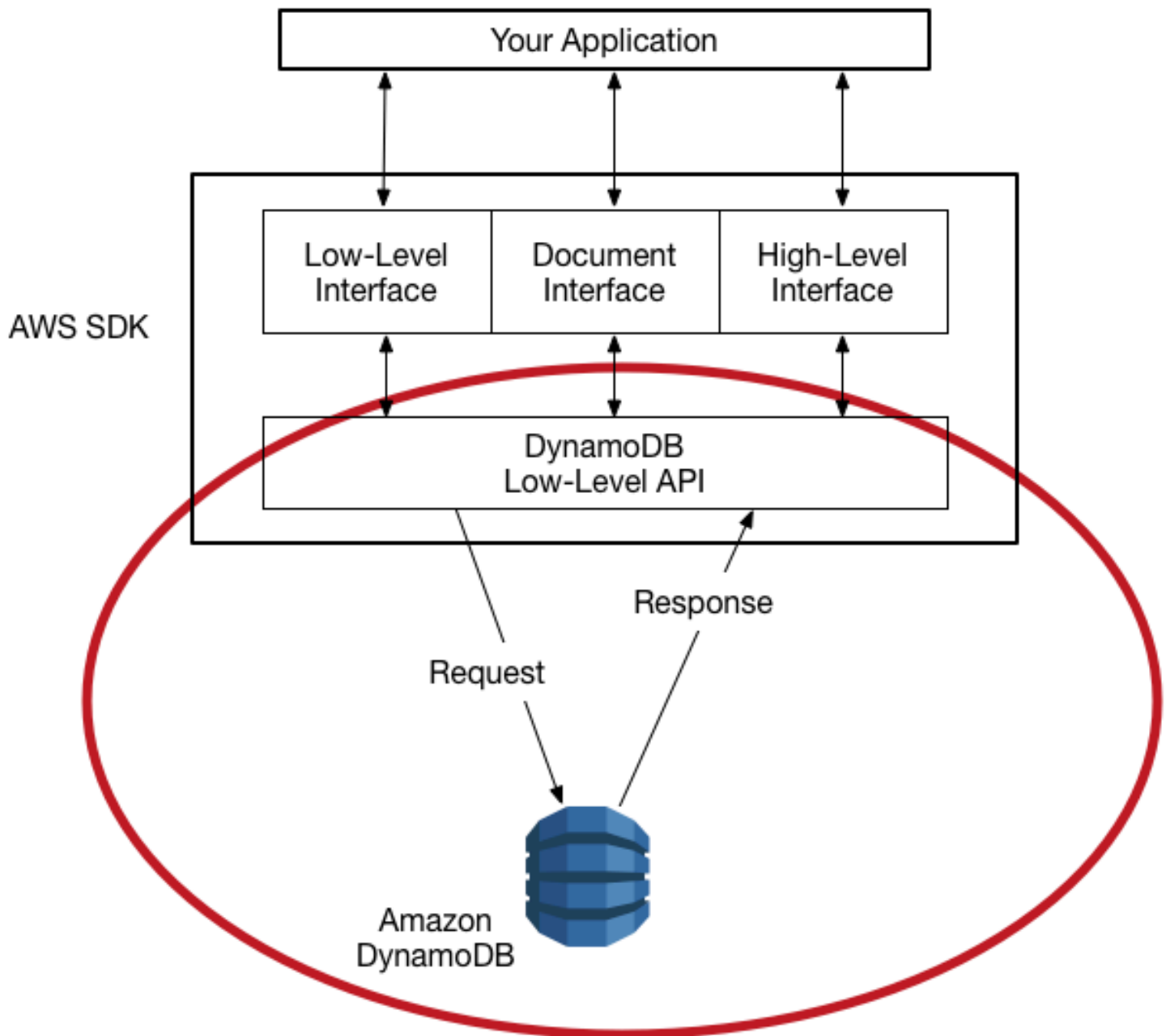
        // Get the item by using the key.
        result = table.getItem(
            (GetItemEnhancedRequest.Builder requestBuilder) ->
            requestBuilder.key(key));
        System.out.println("***** The description value is " +
            result.getCustName());

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

DynamoDB 低階 API

主題

- [要求格式](#)
- [回應格式](#)
- [資料類型描述項](#)
- [數值資料](#)
- [二進位資料](#)



Amazon DynamoDB 低階 API 為 DynamoDB 的協定層級界面。在此層級中，每個 HTTP(S) 要求都必須依照正確的格式，並附上有效的數位簽章。

AWS 開發套件會代表您建構低階 DynamoDB API 請求，並處理來自 DynamoDB 的回應。您如此即可專注於應用程式的邏輯，而非低階的結節。但您仍可從了解低階 DynamoDB API 運作方式的基本知識中獲益。

如需低階 DynamoDB API 的詳細資訊，請參閱 [《Amazon DynamoDB API 參考》](#)。

Note

DynamoDB Streams 具有自己的低階 API，此 API 與 DynamoDB 的 API 不同，且開發套件完全支援。AWS

如需詳細資訊，請參閱 [DynamoDB Streams 的變更資料擷取](#)。如需低階 DynamoDB Streams API，請參閱《[Amazon DynamoDB Streams API 參考](#)》。

低階 DynamoDB API 使用 JavaScript 物件標記法 (JSON) 做為有線通訊協定格式。JSON 以階層顯示資料，以便同時傳達資料值和資料結構。名稱/值對以 `name:value` 格式定義。資料階層由成對的巢狀括住之名稱與值加以定義。

DynamoDB 只會使用 JSON 作為傳輸協定，而非儲存格式。AWS 開發套件會使用 JSON 將資料傳送至動態資料 B，而 DynamoDB 則使用 JSON 回應。DynamoDB 不會以 JSON 格式永久存放資料。

Note

如需 JSON 的詳細資訊，請參閱 [JSON.org](#) 網站上的 [JSON 簡介](#)。

要求格式

DynamoDB 低階 API 接受 HTTP(S) POST 請求作為輸入。AWS 開發套件會為您建構這些請求。

假設您有一個名為 `Pets` 的資料表，其索引鍵結構描述由 `AnimalType` (分割區索引鍵) 及 `Name` (排序索引鍵) 所組成。這些屬性的類型皆為 `string`。若要從中擷取項目 `Pets`，AWS SDK 會建構下列要求。

```
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"}
}
```

請注意以下此要求的相關事宜：

- Authorization 標頭包含 DynamoDB 驗證請求所需之資訊。如需詳細資訊，請參閱 [《簽署 AWS API 要求》](#) 和 [《簽章版本 4》簽署程序 Amazon Web Services 一般參考](#)。
- X-Amz-Target 標頭包含 DynamoDB 操作的名稱：GetItem。(此也會同時附上低階 API 版本，在此案例中為 20120810。)
- 要求的承載 (主體) 包含操作的參數 (JSON 格式)。若是 GetItem 操作，參數為 TableName 與 Key。

回應格式

收到請求時，DynamoDB 會處理該請求並會傳回回應。針對前述的要求，HTTP(S) 回應承載會包含操作的結果，如下列範例中所示。

```
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"}
}
}
```

此時，AWS SDK 會將回應資料傳回您的應用程式，以供進一步處理。

Note

若 DynamoDB 無法處理請求，其會傳回 HTTP 錯誤碼及訊息。AWS 開發套件會以例外狀況的形式，將這些傳播到您的應用程式。如需詳細資訊，請參閱 [使用 DynamoDB 時發生錯誤](#)。

資料類型描述項

低階 DynamoDB API 協定需要每個屬性都要附有一個資料類型的描述項。資料類型描述項是告知 DynamoDB 如何解譯每項屬性的字符。

「[要求格式](#)」與「[回應格式](#)」中的範例，展示如何使用資料類型描述項的範例。GetItem 要求指定 Pets 索引鍵結構描述屬性 (AnimalType 與 Name) 為 S，即類型為 string。GetItem 回應包含具有 string (S)、number (N)、map (M) 以及 list (L) 類型屬性的 Pets 項目。

下列為 DynamoDB 資料類型描述項的完整清單：

- **S**：字串
- **N**：數字
- **B**：二進位
- **BOOL**：布林值
- **NULL**：Null
- **M**：映射
- **L**：清單

- **SS** : 字串集合
- **NS** : 數字集合
- **BS** : 二進位集合

Note

如需 DynamoDB 資料類型的詳細說明，請參閱 [資料類型](#)。

數值資料

不同的程式語言提供不同程度的 JSON 支援。在某些情況下，您可能會決定要使用第三方的程式庫進行 JSON 文件之驗證及剖析。

某些第三方程式庫會依據 JSON 數字類型建置，提供其本身的類型，例如 `int`、`long` 或 `double`。但 DynamoDB 中的原生數字資料類型，無法精確地映射到這些其他資料類型，所以這些類型的差異可能會導致衝突。此外，許多 JSON 程式庫不會處理固定精確度的數值，而是會自動針對包含小數點的數字，推斷為雙精確度資料類型。

為解決這些問題，DynamoDB 提供了不會造成資料遺失的單一數字類型。為避免不必要的隱含轉換為雙精確度值，DynamoDB 使用字串來進行數值資料傳輸。此方法提供了更新屬性值的彈性，同時還能維持適當的排序語意，例如將值 "01"、"2" 及 "03" 依適當的順序排列。

若數字精確度對您的應用程式來說很重要，則應在將其傳遞至 DynamoDB 之前，先將數值轉換為字串。

二進位資料

DynamoDB 支援二進位屬性。但 JSON 原本並不支援二進位資料的編碼。若要在要求中傳送二進位資料，您必須將其編碼為 Base64 格式。在接收到請求時，DynamoDB 會將 Base64 資料解碼回二進位。

DynamoDB 使用的 base64 編碼結構描述會在 Internet Engineering Task Force (IETF) 網站上的 [RFC 4648](#) 說明。

使用 DynamoDB 時發生錯誤

本節說明執行時間錯誤和其處理方式。其中也會說明 Amazon DynamoDB 特有的錯誤訊息和程式碼。如需適用於所有 AWS 服務的常見錯誤清單，請參閱 [存取管理](#)

主題

- [錯誤元件](#)
- [交易性錯誤](#)
- [錯誤訊息和錯誤碼](#)
- [應用程式中的錯誤處理](#)
- [錯誤重試和指數退避](#)
- [批次操作和錯誤處理](#)

錯誤元件

當您的程式傳送請求時，DynamoDB 會嘗試進行處理。如果請求成功，則 DynamoDB 會傳回 HTTP 成功狀態碼 (200 OK) 以及所請求操作的結果。

如果請求不成功，DynamoDB 會傳回錯誤。每個錯誤都會有三個元件：

- HTTP 狀態碼 (例如 400)。
- 例外狀況名稱 (例如 ResourceNotFoundException)。
- 錯誤訊息 (例如 Requested resource not found: Table: *tablename* not found)。

AWS SDK 負責將錯誤傳播到您的應用程式，以便您可以採取適當的措施。例如，在 Java 程式中，您可以撰寫 try-catch 邏輯來處理 ResourceNotFoundException。

如果您未使用 AWS SDK，則需要剖析 DynamoDB 中低階回應的內容。以下是這類回應的範例。

```
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"}
```

交易性錯誤

如需有關交易性錯誤的資訊，請參閱 [DynamoDB 中的交易衝突處理](#)

錯誤訊息和錯誤碼

以下是 DynamoDB 依 HTTP 狀態碼進行分組並傳回的例外狀況清單。確定要重試嗎？為是，則可以重新提交相同的請求。確定要重試嗎？為否，則需要先修正使用者端上的問題，再提交新的請求。

HTTP 狀態碼 400

HTTP 400 狀態碼指出請求發生問題，例如身分驗證失敗、遺失必要參數，或超過資料表的佈建輸送量。您必須先修正應用程式中的問題，再重新提交請求。

AccessDeniedException

訊息：Access denied. (存取遭拒。)

用戶端未正確地簽署請求。如果您使用的是 AWS 開發套件，系統會自動為您簽署請求；如果不是，則請移至 AWS 一般參考中的 [Signature 第 4 版簽署程序](#)。

OK to retry? (確定要重試嗎?) 否

ConditionalCheckFailedException

訊息：The conditional request failed. (條件式請求失敗。)

您已指定評估為 false 的條件。例如，您可能已嘗試對項目執行條件式更新，但屬性的實際值不符合條件中的預期值。

OK to retry? (確定要重試嗎?) 否

IncompleteSignatureException

訊息：The request signature does not conform to standards. (請求簽章不符合 AWS 標準。)

請求簽章未包含所有必要元件。如果您使用 AWS SDK，系統會自動為您簽署要求；否則，請移至 [簽名版本 4 簽署程序AWS 一般參考](#)。

OK to retry? (確定要重試嗎?) 否

ItemCollectionSizeLimitExceededException

訊息：Collection size exceeded. (已超過集合大小。)

針對具有本機次要索引的資料表，具有相同分割區索引鍵值的項目群組已超過 10 GB 的大小上限。如需項目集合的詳細資訊，請參閱 [本機次要索引中的項目集合](#)。

OK to retry? (確定要重試嗎?) 是

LimitExceededException

訊息：Too many operations for a given subscriber. (指定訂閱者的操作太多。)

並行控制平面操作太多。在 CREATING、DELETING 或 UPDATING 狀態中，資料表和索引的累積數量不能超過 500。

OK to retry? (確定要重試嗎?) 是

MissingAuthenticationTokenException

訊息：Request must contain a valid (registered) AWS Access Key ID. (請求必須包含有效 (已註冊) AWS 存取金鑰 ID。)

請求未包含必要授權標頭，或其格式不正確。請參閱 [DynamoDB 低階 API](#)。

OK to retry? (確定要重試嗎?) 否

ProvisionedThroughputExceededException

訊息：You exceeded your maximum allowed provisioned throughput for a table or for one or more global secondary indexes. 若要檢視佈建輸送量與消耗輸送量的效能指標，請開啟 [Amazon CloudWatch 主控台](#)。

範例：您的請求率太高。Dynamo AWS DB 的開發套件會自動重試收到此例外狀況的要求。除非您的重試佇列太大無法完成，否則您的請求最後會成功。請使用 [錯誤重試和指數退避](#) 來減少請求的頻率。

OK to retry? (確定要重試嗎?) 是

RequestLimitExceeded

訊息：輸送量超過帳戶的目前輸送量限制。若要要求提高限制，請透過 <https://aws.amazon.com/support> 聯絡 Sup AWS port 部門。

範例：隨選請求率超過允許的帳戶輸送量且資料表無法進一步擴展。

OK to retry? (確定要重試嗎?) 是

ResourceInUseException

訊息：The resource which you are attempting to change is in use. (正在使用您嘗試變更的資源。)

範例：您已嘗試重新建立現在資料表，或刪除目前處於 CREATING 狀態的資料表。

OK to retry? (確定要重試嗎?) 否

ResourceNotFoundException

訊息：Requested resource not found. (找不到已註冊的資源。)

範例：所請求的資料表不存在，或太早處於 CREATING 狀態。

OK to retry? (確定要重試嗎?) 否

ThrottlingException

訊息：Rate of requests exceeds the allowed throughput. (請求率超過允許的輸送。)

此例外狀況會傳回為含有節流 _ 例外狀態碼的回 AmazonServiceException 應。如果您執行[控制平台](#) API 操作太快，可能會傳回此例外狀況。

對於使用隨需的資料表，如果您的請求率太高，任何[資料平面](#) API 操作都可能會傳回此例外狀況。若要進一步了解隨需擴展，請參閱[初始輸送量和擴展屬性](#)。

OK to retry? (確定要重試嗎?) 是

UnrecognizedClientException

訊息：The Access Key ID or security token is invalid. (存取金鑰 ID 或安全字串無效。)

請求簽章不正確。最有可能的原因是無效的 AWS 訪問密鑰 ID 或密鑰。

OK to retry? (確定要重試嗎?) 是

ValidationException

訊息：Varies, depending upon the specific error(s) encountered (會根據發生的特定錯誤而不同)

可能會因數個原因而發生此錯誤，例如必要參數遺失、值超出範圍，或資料類型不符。錯誤訊息包含導致錯誤之請求特定部分的詳細資訊。

OK to retry? (確定要重試嗎?) 否

HTTP 狀態碼 5xx

HTTP 5xx 狀態碼指出 AWS 必須解決的問題。這可能是暫時性錯誤，在此情況下，您可以重試請求，直到成功為止。如果不是，請前往 [AWS Service Health Dashboard](#)，確認服務是否發生任何操作問題。

如需詳細資訊，請參閱[如何解決 Amazon DynamoDB 中的 HTTP 5xx 錯誤？](#)

InternalServerError (HTTP 500)

DynamoDB 無法處理您的請求。

OK to retry? (確定要重試嗎?) 是

Note

處理項目時，您可能會發生內部伺服器錯誤。在資料表生命週期期間，這些是預期錯誤。可以立即重試任何失敗的請求。

當您在寫入作業上收到狀態碼 500 時，作業可能已成功或失敗。如果寫入操作是 `TransactWriteItem` 請求，則可以重試操作。如果寫入作業是單一項目寫入要求，例如 `PutItem`、`UpdateItem`，或 `DeleteItem`，那麼您的應用程式應該在重試操作之前讀取項目的狀態，和/或使用 [條件表達式](#)，以確保項目在重試後保持正確的狀態，無論先前的作業是成功還是失敗。如果冪等性是寫入操作的要求，請使用 `TransactWriteItem`，它通過自動指定 `ClientRequestToken` 來消除多次嘗試執行相同動作的歧義。

ServiceUnavailable (HTTP 503)

DynamoDB 目前無法使用。(這應該是暫時狀態)。

OK to retry? (確定要重試嗎?) 是

應用程式中的錯誤處理

若要讓您的應用程式順暢執行，您需要新增邏輯來截獲並回應錯誤。一般方式包含使用 `try-catch` 區塊或 `if-then` 陳述式。

AWS SDK 執行自己的重試和錯誤檢查。如果您在使用其中一個 AWS SDK 時遇到錯誤，錯誤代碼和說明可以幫助您進行故障排除。

您應該也會在回應中看到 `Request ID`。如果您需要與 Sup AWS port 人員合作來診斷問題，`Request ID` 可能會有所幫助。

錯誤重試和指數退避

網路上有許多元件 (例如 DNS 伺服器、交換器、負載平衡器和其他項目) 可以在指定請求之生命週期中的任何階段產生錯誤。一般在網路環境中處理這些錯誤回應的技術，就是在用戶端應用程式中實作重試。這技術會提高應用程式的可靠性。

每個 AWS SDK 都會自動實作重試邏輯。您可以視需要修改重試參數。例如，請考慮使用需要快速失敗策略 (即發生錯誤時不允許重試) 的 Java 應用程式。使用時 AWS SDK for Java，您可以使用 `ClientConfiguration` 類別，並提供的 `maxErrorRetry` 值 0 來關閉重試。如需詳細資訊，請參閱您程式設計語言的 AWS SDK 文件。

如果您不使用 AWS SDK，則應重試收到服務器錯誤 (5xx) 的原始請求。不過，用戶端錯誤 (4xx，非 `ThrottlingException` 或 `ProvisionedThroughputExceededException`) 指出您需要先修訂請求本身更正問題，然後再試一次。

除了簡單的重試之外，每個 AWS SDK 都實現了指數輪詢算法，以實現更好的流量控制。指數退避的背後概念是，針對連續錯誤回應，讓重試之間的等待時間漸進拉長。例如，最多 50 毫秒再進行第一次重試、最多 100 毫秒再進行第二次重試、最多 200 毫秒再進行第三次重試，以此類推。不過，在一分鐘之後，如果請求未成功，則問題可能是請求大小超過佈建輸送量，而不是請求率。設定大約一分鐘停止的最大重試次數。如果請求未成功，請調查您的佈建輸送量選項。

Note

AWS SDK 實作自動重試邏輯和指數輪詢。

大多數指數退避演算法會使用抖動 (隨機延遲)，以防止連續衝突。因為您在這些情況下並未嘗試避免這種衝突，所以不需要使用此亂數。不過，如果您使用並行用戶端，抖動有助於讓請求更快取得成功。如需詳細資訊，請參閱關於 [指數退避和抖動](#) 的部落格文章。

批次操作和錯誤處理

DynamoDB 低階 API 支援讀取和寫入的批次操作。BatchGetItem 從一或多個資料表讀取項目，BatchWriteItem 在一或多個資料表中放置或刪除項目。在其他非批次 DynamoDB 操作中，這些批次操作會實作為包裝函式。換言之，BatchGetItem 會針對批次中的每個項目呼叫 GetItem 一次。同樣地，BatchWriteItem 會針對批次中的每個項目適當地呼叫 DeleteItem 或 PutItem。

批次操作可以容忍批次中個別請求的失敗。例如，請考慮使用 BatchGetItem 請求來讀取五個項目。即使部分基礎 GetItem 請求失敗，這也不會導致整個 BatchGetItem 操作失敗。但若全部五個讀取操作均失敗，則整個 BatchGetItem 便會失敗。

批次操作會傳回失敗個別請求的資訊，讓您可以診斷問題，並重試操作。針對 BatchGetItem，會在回應的 UnprocessedKeys 值中傳回有問題的資料表和主索引鍵。針對 BatchWriteItem，在 UnprocessedItems 中會傳回類似的資訊。

最可能的失敗讀取或失敗寫入原因是調節。針對 BatchGetItem，批次請求中的一或多個資料表沒有足夠的已佈建讀取容量可支援操作。針對 BatchWriteItem，一或多個資料表沒有足夠的已佈建寫入容量。

如果 DynamoDB 傳回任何未處理的項目，您應該對這些項目重試批次操作。不過，強烈建議您使用指數退避演算法。如果您立即重試批次操作，則基於個別資料表上的調節，基礎讀取或寫入請求仍然可能會失敗。如果您使用指數退避來延遲批次操作，則批次中的個別請求較可能會成功。

適用於 DynamoDB 的更高階程式設計界面

這些開 AWS 發套件為應用程式提供低階介面，以便與 Amazon DynamoDB 搭配使用。這些用戶端類別和方法直接對應至低階 DynamoDB API。不過，當需要將複雜的資料類型映射至資料庫資料表中的項目時，許多開發人員會遭遇到網路斷線的感覺，或阻抗不符。使用低階資料庫界面，開發人員必須撰寫讀取或寫入物件資料至資料庫資料表的方法，反之亦然。物件類型和資料庫表格每個組合所需的額外程式碼數量似乎非常大。

為了簡化開發，Java 和 .NET 的 AWS SDK 提供了具有更高層次抽象的其他接口。DynamoDB 的較高階界面可讓您定義程式中的物件與存放這些物件資料的資料庫表格之間的關係。定義此映射之後，您可以呼叫簡單的物件方法 (例如 save、load 或 delete)，也可以代表您自動叫用基礎低階 DynamoDB 操作。這允許您編寫以物件為中心的程式碼，而不是以資料庫為中心的程式碼。

適用於 Java AWS 和 .NET 的開發套件中提供了適用於 DynamoDB 的更高層級程式設計介面。

Java

- [Java 1.x : DynamoDBMapper](#)
- [Java 2.x : DynamoDB 增強型用戶端](#)

.NET

- [.NET : 文件模型](#)
- [.NET : 物件持久性模型](#)

Java 1.x : DynamoDBMapper

提 AWS SDK for Java 供一個DynamoDBMapper類別，可讓您將用戶端類別對應至 Amazon DynamoDB 表格。若要使用 DynamoDBMapper，請定義 DynamoDB 資料表中項目與程式碼中其對應物件執行個體之間的關係。DynamoDBMapper 類別也讓您執行各種建立、讀取、更新和對項目進行刪除 (CRUD) 操作，以及執行查詢和掃描資料表。

主題

- [適用於 Java 的 DynamoDB Mapper 支援的資料類型](#)
- [適用於 DynamoDB 的 Java 註釋](#)
- [DynamoDBMapper Class](#)
- [DynamoDBMapper 的選用組態設定](#)
- [含版本編號的樂觀鎖定](#)
- [映射任意資料](#)
- [DynamoDBMapper 範例](#)

Note

DynamoDBMapper 類別不允許您建立、更新或刪除資料表。若要執行這些任務，請改為使用低階適用於 Java 的開發套件界面。如需詳細資訊，請參閱 [在 Java 中使用 DynamoDB 資料表](#)。

適用於 Java 的開發套件提供一組註釋類型，讓您可以將類別映射至資料表。例如，考量其 Id 為分割區索引鍵的 ProductCatalog 資料表。

```
ProductCatalog(Id, ...)
```

您可以將用戶端應用程式中的類別映射至 ProductCatalog 資料表，如下列 Java 程式碼所示。此程式碼定義名為 CatalogItem 的純舊 Java 物件 (POJO)，而此物件使用註釋將物件欄位映射至 DynamoDB 屬性名稱。

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; }
}
```

在上述程式碼中，@DynamoDBTable 註釋會將 CatalogItem 類別映射至 ProductCatalog 資料表。您可以將個別類別執行個體存放為資料表中的項目。在類別定義中，@DynamoDBHashKey 註釋會將 Id 屬性映射至主索引鍵。

類別屬性預設會映射至資料表中的相同名稱屬性。Title 和 ISBN 屬性會映射至資料表中的相同名稱屬性。

DynamoDB 屬性的名稱符合類別中所宣告屬性的名稱時，@DynamoDBAttribute 註釋是選用項目。它們不同時，請搭配使用此註釋與 attributeName 參數，指定此屬性所對應的 DynamoDB 屬性。

在上述範例中，將 @DynamoDBAttribute 註釋新增至每個屬性，確保屬性名稱完全符合「[在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#)」中所建立的資料表，並與本指南的其他程式碼範例中所使用的屬性名稱一致。

類別定義可以有未映射至資料表中任何屬性的屬性。您可以新增 @DynamoDBIgnore 註釋來識別這些屬性。在上述範例中，SomeProp 屬性會標上 @DynamoDBIgnore 註釋。當您將 CatalogItem 執行個體上傳至資料表時，DynamoDBMapper 執行個體不會包含 SomeProp 屬性。此外，當您從資料表中擷取項目時，映射器不會傳回此屬性。

在您定義映射類別之後，可以使用 DynamoDBMapper 方法，將該類別的執行個體寫入至 Catalog 資料表中的對應項目。以下程式碼範例會示範此技術。

```
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);
```

下列程式碼範例示範如何擷取項目以及存取它的一些屬性：

```
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 提供直觀且自然的方式來處理 Java 內的 DynamoDB 資料。它也提供數項內建功能；例如，樂觀鎖定、ACID 交易、自動產生的分割區索引鍵和排序索引鍵值，以及物件版本控制。

適用於 Java 的 DynamoDB Mapper 支援的資料類型

本節說明 Amazon DynamoDB 中支援的基本 Java 資料類型、集合和任意資料類型。

Amazon DynamoDB 支援下列基本 Java 資料類型和基本包裝函式類別。

- String
- Boolean, boolean
- Byte, byte
- Date (作為 [ISO_8601](#) 毫秒精確性字串，已轉移為 UTC)
- Calendar (作為 [ISO_8601](#) 毫秒精確性字串，已轉移為 UTC)
- Long, long
- Integer, int
- Double, double
- Float, float
- BigDecimal
- BigInteger

Note

- 如需有關 DynamoDB 命名規則和各種支援之資料類型的詳細資訊，請參閱 [Amazon DynamoDB 中支援的資料類型和命名規則](#)。
- 空白的二進位值由 DynamoDBMapper 支援。
- AWS SDK for Java 2.x 支援空白字串值。

在適用於 Java 1.x 的 AWS SDK 中，DynamoDBMapper 支援讀取空字串屬性值，不過，它不會寫入空白字串屬性值，因為這些屬性是從要求中捨棄的。

DynamoDB 支援 Java [Set](#)、[List](#) 和 [Map](#) 集合類型。下表摘要說明如何將這些 Java 類型映射至 DynamoDB 類型。

Java 類型	DynamoDB 類型
所有數字類型	N (數字類型)
Strings	S (字串類型)
Boolean	BOOL (布林類型), 0 或 1。
ByteBuffer	B (二進位類型)
Date	S (字串類型)。Date 值會以 ISO-8601 格式字串存放。
Set 集合類型	SS (字串集) 類型、NS (數字集) 類型或 BS (二進位集) 類型。

DynamoDBTypeConverter 界面可讓您將自己的任意資料類型映射至 DynamoDB 原生支援的資料類型。如需詳細資訊，請參閱 [映射任意資料](#)。

適用於 DynamoDB 的 Java 註釋

本節說明可用於將類別和屬性映射至 Amazon DynamoDB 中資料表和屬性的註釋。

如需對應的 Javadoc 文件，請參閱 [《AWS SDK for Java API 參考》](#) 中的 [註釋類型摘要](#)。

Note

在下列註釋中，只需要 `DynamoDBTable` 和 `DynamoDBHashKey`。

主題

- [DynamoDBAttribute](#)
- [DynamoDB AutoGeneratedKey](#)
- [DynamoDB AutoGeneratedTimestamp](#)
- [DynamoDBDocument](#)
- [DynamoDB HashKey](#)
- [DynamoDBIgnore](#)
- [DynamoDB IndexHashKey](#)
- [DynamoDB IndexRangeKey](#)
- [DynamoDB RangeKey](#)
- [DynamoDBTable](#)
- [DynamoDB TypeConverted](#)
- [DynamoDBTyped](#)
- [DynamoDB VersionAttribute](#)

DynamoDBAttribute

將屬性映射至資料表屬性。每個類別屬性預設會映射至同名的項目屬性。不過，如果名稱不同，您可以使用此註釋將屬性 (property) 映射至屬性 (attribute)。在下列 Java 程式碼片段中，`DynamoDBAttribute` 會將 `BookAuthors` 屬性映射至資料表中的 `Authors` 屬性名稱。

```
@DynamoDBAttribute(attributeName = "Authors")
public List<String> getBookAuthors() { return BookAuthors; }
public void setBookAuthors(List<String> BookAuthors) { this.BookAuthors =
    BookAuthors; }
```

將物件儲存至資料表時，`DynamoDBMapper` 使用 `Authors` 做為屬性名稱。

DynamoDB AutoGeneratedKey

將分割區索引鍵或排序索引鍵屬性標示為自動產生。儲存這些屬性時，DynamoDBMapper 會產生隨機 [UUID](#)。只有字串屬性才能標示為自動產生的索引鍵。

以下是示範使用自動產生索引鍵參數的範例。

```
@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

自動產生時間戳記。

```
@DynamoDBAutoGeneratedTimestamp(strategy=DynamoDBAutoGenerateStrategy.ALWAYS)
public Date getLastUpdatedDate() { return lastUpdatedDate; }
public void setLastUpdatedDate(Date lastUpdatedDate) { this.lastUpdatedDate =
    lastUpdatedDate; }
```

或者，可以透過提供策略屬性來定義自動產生策略。預設值為 ALWAYS。

DynamoDBDocument

指出類別可以序列化為 Amazon DynamoDB 文件。

例如，假設您要將 JSON 文件映射至 Map (M) 類型的 DynamoDB 屬性。下列程式碼範例定義包含 Map 類型之巢狀屬性 (Pictures) 的項目。

```
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; }

    }
}
```

然後您就可儲存新的 ProductCatalog 項目，並具有 Pictures，如下列範例所示。

```
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);
```

產生的 ProductCatalog 項目看起來將會如下 (JSON 格式)。

```
{
  "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

將類別屬性映射至資料表的分割區索引鍵。此屬性必須是純量字串、數字或二進位類型的其中之一。此屬性不能是集合類型。

假設您的 ProductCatalog 資料表將 Id 作為主索引鍵。下列 Java 程式碼定義 CatalogItem 類別，並使用 @DynamoDBHashKey 標籤將其 Id 屬性映射至 ProductCatalog 資料表的主索引鍵。

```
@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

指出應該忽略相關聯屬性的 `DynamoDBMapper` 執行個體。將資料儲存至資料表時，`DynamoDBMapper` 不會將此屬性儲存至資料表。

套用至 `getter` 方法或非模組化屬性的類別欄位。若該註釋是直接套用至類別欄位，則必須在相同的類別宣告相對應的 `getter` 和 `setter`。

DynamoDB IndexHashKey

將類別屬性映射至全域次要索引的分割區索引鍵。此屬性必須是純量字串、數字或二進位類型的其中之一。此屬性不能是集合類型。

如果您需要對全域次要索引進行 Query，請使用此註釋。您必須指定索引名稱 (`globalSecondaryIndexName`)。如果類別屬性的名稱與索引的分割區索引鍵不同，您也必須指定該索引屬性的名稱 (`attributeName`)。

DynamoDB IndexRangeKey

將類別屬性映射至全域次要索引或本機次要索引的排序索引鍵。此屬性必須是純量字串、數字或二進位類型的其中之一。此屬性不能是集合類型。

如果您需要對本機次要索引或全域次要索引進行 Query，並且想要使用索引的排序索引鍵來縮小您結果的範圍，請使用此註釋。您必須指定索引名稱 (`globalSecondaryIndexName` 或 `localSecondaryIndexName`)。如果類別屬性的名稱與索引的排序索引鍵不同，您也必須指定該索引屬性的名稱 (`attributeName`)。

DynamoDB RangeKey

將類別屬性映射至資料表的排序索引鍵。此屬性必須是純量字串、數字或二進位類型的其中之一。它不能是集合類型。

如果主索引鍵是複合 (分割區索引鍵和排序索引鍵)，您可以使用此標籤，將類別欄位映射至排序索引鍵。例如，假設您的 `Reply` 資料表存放論壇主題回覆。每個對話都可以有許多回覆。因此，此資料表的主索引鍵是 `ThreadId` 和 `ReplyDateTime`。分割區索引鍵是 `ThreadId`，而排序索引鍵是 `ReplyDateTime`。

下列 Java 程式碼定義 `Reply` 類別，並將它映射至 `Reply` 資料表。它會使用 `@DynamoDBHashKey` 和 `@DynamoDBRangeKey` 標籤來識別映射至主索引鍵的類別屬性。

```
@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

識別 DynamoDB 中的目標資料表。下列 Java 程式碼定義 Developer 類別，並將它映射至 DynamoDB 中的 People 資料表。

```
@DynamoDBTable(tableName="People")
public class Developer { ...}
```

@DynamoDBTable 註釋可以予以繼承。任何繼承自 Developer 類別的新類別也會映射至 People 資料表。例如，假設您建立繼承自 Lead 類別的 Developer 類別。因為您已將 Developer 類別映射至 People 資料表，所以也會將 Lead 類別物件存放至相同的資料表。

@DynamoDBTable 也可以予以覆寫。任何繼承自 Developer 類別的新類別預設會映射至相同的 People 資料表。不過，您可以覆寫此預設映射。例如，如果您建立繼承自 Developer 類別的類別，則可以新增 @DynamoDBTable 註釋，明確地將它映射至另一個資料表，如下列 Java 程式碼範例所示。

```
@DynamoDBTable(tableName="Managers")
public class Manager extends Developer { ...}
```

DynamoDB TypeConverted

將屬性標示為使用自訂類型轉換器的註釋。可以標註於使用者定義的註釋，以將其他屬性傳遞給 DynamoDBTypeConverter。

DynamoDBTypeConverter 界面可讓您將自己的任意資料類型映射至 DynamoDB 原生支援的資料類型。如需詳細資訊，請參閱 [映射任意資料](#)。

DynamoDBTyped

覆寫標準屬性類型繫結的註釋。如果套用標準類型的預設屬性繫結，則該類型不需要註釋。

DynamoDB VersionAttribute

識別用於存放樂觀鎖定版本編號的類別屬性。DynamoDBMapper 在儲存新的項目時會將版本編號指派給此屬性，並在每次您更新項目時予以遞增。僅支援數字純量類型。如需資料類型的詳細資訊，請參閱 [資料類型](#)。如需版本控制的詳細資訊，請參閱「[含版本編號的樂觀鎖定](#)」。

DynamoDBMapper Class

DynamoDBMapper 類別是 Amazon DynamoDB 的進入點。它提供 DynamoDB 端點的存取，也讓您您可以存取多個資料表中的資料。它也讓您執行各種建立、讀取、更新和對項目進行刪除 (CRUD) 操作，以及執行查詢和掃描資料表。此類別提供下列使用 DynamoDB 的方法。

如需對應的 Javadoc 文件，請參閱《AWS SDK for Java API 參考》中的 [DynamoDBMapper](#)。

主題

- [save](#)
- [load](#)
- [刪除](#)
- [query](#)
- [queryPage](#)
- [scan](#)
- [scanPage](#)
- [parallelScan](#)
- [batchSave](#)
- [batchLoad](#)
- [batchDelete](#)
- [batchWrite](#)
- [transactionWrite](#)

- [transactionLoad](#)
- [count](#)
- [generateCreateTable請求](#)
- [createS3Link](#)
- [獲得 3 ClientCache](#)

save

將指定的物件儲存至資料表。您要儲存的物件是此方法的唯一必要參數。您可以使用 `DynamoDBMapperConfig` 物件來提供選用的組態參數。

如果具有相同主索引鍵的項目不存在，則此方法會在資料表中建立新的項目。如果具有相同主索引鍵的項目存在，則會更新現有項目。如果分割區索引鍵和排序索引鍵的類型是字串，並且標註 `@DynamoDBAutoGeneratedKey`，則會將隨機全域唯一識別符 (UUID) 授予它們 (若未初始化)。標註 `@DynamoDBVersionAttribute` 的版本欄位會遞增一。此外，如果更新版本欄位或產生索引鍵，則會因這項操作而更新傳入的物件。

根據預設，只會更新對應至已映射類別屬性的屬性。項目上的任何其他現有屬性則不受影響。不過，如果您指定 `SaveBehavior.CLOBBER`，則可以強制完全覆寫項目。

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER).build();

mapper.save(item, config);
```

如果您已啟用版本控制，則用戶端與伺服器端項目版本必須相符。不過，如果使用 `SaveBehavior.CLOBBER` 選項，則版本不需要相符。如需版本控制的詳細資訊，請參閱「[含版本編號的樂觀鎖定](#)」。

load

從資料表擷取項目。您必須提供要擷取之項目的主索引鍵。您可以使用 `DynamoDBMapperConfig` 物件來提供選用的組態參數。例如，您可以選擇性地請求強烈一致讀取，確保此方法只擷取最新的項目數值，如下列 Java 陳述式所示。

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT).build();
```



```
CatalogItem item = mapper.load(CatalogItem.class, item.getId(), config);
```

DynamoDB 預設會傳回數值最終一致的項目。如需 DynamoDB 最終一致性模型的資訊，請參閱 [讀取一致性](#)。

刪除

刪除資料表中的項目。您必須傳入已映射類別的物件執行個體。

如果您已啟用版本控制，則用戶端與伺服器端項目版本必須相符。不過，如果使用 `SaveBehavior.CLOBBER` 選項，則版本不需要相符。如需版本控制的詳細資訊，請參閱「[含版本編號的樂觀鎖定](#)」。

query

查詢資料表或次要索引。

假設您有一個存放論壇主題回覆的 `Reply` 資料表。每個對話主旨都可以有零個以上的回覆。`Reply` 資料表的主索引鍵包含 `Id` 和 `ReplyDateTime` 欄位；其中，`Id` 是分割區索引鍵，而 `ReplyDateTime` 是主索引鍵的排序索引鍵。

```
Reply ( Id, ReplyDateTime, ... )
```

假設您已建立 `Reply` 類別與 DynamoDB 中對應 `Reply` 資料表之間的映射。下列 Java 程式碼使用 `DynamoDBMapper` 來尋找特定對話主旨在過去兩週的所有回覆。

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);
```

查詢會傳回 Reply 物件的集合。

query 方法預設會傳回「延遲載入」集合。它一開始只會傳回一頁的結果，然後視需要進行服務呼叫來取得下一頁。若要取得所有相符的項目，請逐一查看 latestReplies 集合。

請注意，在集合上呼叫 size() 方法將會載入所有結果，以提供準確的計數。這可能會導致耗用大量已佈建的輸送量，而在非常大型的資料表上可能甚至會耗盡 JVM 中的所有記憶體。

若要查詢索引，您必須先將索引建模為映射器類別。假設該 Reply 表具有名為 PostedBy-消息索引的全局次要索引。此索引的分割區索引鍵是 PostedBy，而排序索引鍵是 Message。索引中項目的類別定義將會如下。

```
@DynamoDBTable(tableName="Reply")
public class PostedByMessage {
    private String postedBy;
    private String message;

    @DynamoDBIndexHashKey(globalSecondaryIndexName = "PostedBy-Message-Index",
        attributeName = "PostedBy")
    public String getPostedBy() { return postedBy; }
    public void setPostedBy(String postedBy) { this.postedBy = postedBy; }

    @DynamoDBIndexRangeKey(globalSecondaryIndexName = "PostedBy-Message-Index",
        attributeName = "Message")
    public String getMessage() { return message; }
    public void setMessage(String message) { this.message = message; }

    // Additional properties go here.
}
```

@DynamoDBTable 註釋指出此索引與 Reply 資料表建立關聯。此 @DynamoDBIndexHashKey 註釋代表索引的分割索引鍵 (PostedBy)，並 @DynamoDBIndexRangeKey 代表索引的排序索引鍵 (訊息)。

您現在可以使用 DynamoDBMapper 來查詢索引，並擷取特定使用者所張貼的訊息子集。如果資料表和索引之間沒有衝突的映射，且映射器中已建立映射，則不需要指定索引名稱。映射器將根據主索引鍵和

排序索引鍵推論。下列程式碼會查詢全域次要索引。因為全域次要索引支援最終一致讀取，但不支援強烈一致讀取，所以必須指定 `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);
```

查詢會傳回 `PostedByMessage` 物件的集合。

queryPage

查詢資料表或次要索引，並傳回單頁的相符結果。與使用 `query` 方法相同，您必須指定分割區索引鍵值以及套用至排序索引鍵屬性的查詢篩選條件。不過，`queryPage` 只會傳回第一「頁」的資料；亦即，符合 1 MB 的資料量

scan

掃描整個資料表或次要索引。您可以選擇性地指定 `FilterExpression` 來篩選結果集。

假設您有一個存放論壇主題回覆的 `Reply` 資料表。每個對話主旨都可以有零個以上的回覆。`Reply` 資料表的主索引鍵包含 `Id` 和 `ReplyDateTime` 欄位；其中，`Id` 是分割區索引鍵，而 `ReplyDateTime` 是主索引鍵的排序索引鍵。

```
Reply ( Id, ReplyDateTime, ... )
```

如果您已將 Java 類別映射至 `Reply` 資料表，則可以使用 `DynamoDBMapper` 來掃描資料表。例如，下列 Java 程式碼會掃描整個 `Reply` 資料表，而且只會傳回特定一年的回覆。

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);
```

`scan` 方法預設會傳回「延遲載入」集合。它一開始只會傳回一頁的結果，然後視需要進行服務呼叫來取得下一頁。若要取得所有相符的項目，請逐一查看 `replies` 集合。

請注意，在集合上呼叫 `size()` 方法將會載入所有結果，以提供準確的計數。這可能會導致耗用大量已佈建的輸送量，而在非常大型的資料表上可能甚至會耗盡 JVM 中的所有記憶體。

若要掃描索引，您必須先將索引建模為映射器類別。假設 `Reply` 資料表有一個名為 `PostedBy-Message-Index` 的全域次要索引。此索引的分割區索引鍵是 `PostedBy`，而排序索引鍵是 `Message`。此索引的映射器類別顯示在 [query](#) 區段。它使用 `@DynamoDBIndexHashKey` 和 `@DynamoDBIndexRangeKey` 註釋來指定索引的分割區索引鍵和排序索引鍵。

下列程式碼範例掃描 `PostedBy-Message-Index`。它不會使用掃描篩選條件，因此會將索引中的所有項目都傳回給您。

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false);

List<PostedByMessage> iList = mapper.scan(PostedByMessage.class, scanExpression);
Iterator<PostedByMessage> indexItems = iList.iterator();
```

scanPage

掃描資料表或次要索引，並傳回單頁的相符結果。與使用 `scan` 方法相同，您可以選擇性地指定 `FilterExpression` 來篩選結果集。不過，`scanPage` 只會傳回第一「頁」的資料；亦即，符合 1 MB 內的資料量。

parallelScan

執行整個資料表或次要索引的平行掃描。您可以指定資料表的一些邏輯區段，以及掃描表達式來篩選結果。`parallelScan` 會將掃描任務分到多個工作者（一個邏輯區段一個工作者）；工作者會平行處理資料，並傳回結果。

下列 Java 程式碼範例會對 `Product` 資料表執行平行掃描。

```
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);
```

針對說明如何使用 `parallelScan` 的 Java 程式碼範例，請參閱「[DynamoDBMapper 查詢和掃描操作](#)」。

batchSave

使用 `AmazonDynamoDB.batchWriteItem` 方法的一或多個呼叫，將物件儲存至一或多個資料表。此方法不提供交易保證。

下列 Java 程式碼會將兩個項目 (書籍) 儲存至 `ProductCatalog` 資料表。

```
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

使用一或多個資料表的主索引鍵，以從中擷取多個項目。

下列 Java 程式碼會從兩個不同的資料表擷取兩個項目。

```
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

使用 `AmazonDynamoDB.batchWriteItem` 方法的一或多個呼叫，刪除一或多個資料表中的物件。此方法不提供交易保證。

下列 Java 程式碼會刪除 `ProductCatalog` 資料表中的兩個項目 (書籍)。

```
Book book1 = mapper.load(Book.class, 901);
Book book2 = mapper.load(Book.class, 902);
mapper.batchDelete(Arrays.asList(book1, book2));
```

batchWrite

使用 `AmazonDynamoDB.batchWriteItem` 方法的一或多個呼叫，將物件儲存至一或多個資料表，並刪除物件。此方法未提供交易保證或支援版本控制 (條件式放置或刪除)。

下列 Java 程式碼會將新項目寫入至 `Forum` 資料表、將新項目寫入至 `Thread` 資料表，並刪除 `ProductCatalog` 資料表中的項目。

```
// 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

使用 `AmazonDynamoDB.transactWriteItems` 方法的一個呼叫，將物件儲存至一或多個資料表，並從中刪除物件。

[如需交易特定例外狀況的清單，請參閱 `TransactWriteItems` 錯誤。](#)

如需 DynamoDB 交易與提供之不可部分完成性、一致性、隔離性和持久性 (ACID) 保證的相關資訊，請參閱 [Amazon DynamoDB Transactions](#)。

Note

此方法不支援下列項目：

- [DynamoDB MapperConfig 支援 `SaveBehavior`。](#)

以下 Java 程式碼會以交易的方式，將新項目寫入每個 Forum 和 Thread 資料表。

```
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

使用 `AmazonDynamoDB.transactGetItems` 方法的一個呼叫，載入一或多個資料表中的物件。

[如需交易特定例外狀況的清單，請參閱TransactGetItems 錯誤。](#)

如需 DynamoDB 交易與提供之不可部分完成性、一致性、隔離性和耐久性 (ACID) 保證的相關資訊，請參閱 [Amazon DynamoDB Transactions](#)。

以下 Java 程式碼會以交易的方式，將從每個 Forum 和 Thread 資料表載入一個項目。

```
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

評估指定的掃描表達式，並傳回相符項目的計數。不傳回任何項目資料。

generateCreateTable請求

剖析代表 DynamoDB 資料表的 POJO 類別，並傳回該資料表的 CreateTableRequest。

createS3Link

建立 Amazon S3 中物件的連結。您必須指定儲存貯體名稱和索引鍵名稱，以唯一識別儲存貯體中的物件。

若要使用 createS3Link，映射器類別必須定義 getter 和 setter 方法。下列程式碼範例透過將新的屬性和 getter/setter 方法新增至 CatalogItem 類別來說明這種情況。

```
@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;
}

...
}
```

下列 Java 程式碼定義要寫入至 Product 資料表的新項目。此項目包含產品映像的連結；映像資料會上傳至 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);
```

S3Link 類別提供許多其他方法來操控 Amazon S3 中的物件。如需詳細資訊，請參閱 [Javadocs for S3Link](#)。

獲得 3 ClientCache

傳回基礎 S3ClientCache 來存取 Amazon S3。S3ClientCache 是 AmazonS3Client 物件的智慧映射。如果您有多個客戶，S3ClientCache 可以幫助您按 AWS 區域保持客戶組織，並可以根據需求創建新的 Amazon S3 客戶。

DynamoDBMapper 的選用組態設定

當您建立 DynamoDBMapper 執行個體時，會有特定預設行為；您可以使用 DynamoDBMapperConfig 類別來覆寫這些預設值。

下列程式碼片段會使用自訂設定來建立 `DynamoDBMapper`：

```
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);
```

如需詳細資訊，請參閱 [AWS SDK for Java API 參考資料MapperConfig](#) 中的 [DynamoDB](#)。

您可以針對 `DynamoDBMapperConfig` 執行個體使用下列引數：

- `DynamoDBMapperConfig.ConsistentReads` 列舉值：
 - `EVENTUAL`：映射器執行個體使用最終一致讀取請求。
 - `CONSISTENT`：映射器執行個體使用強烈一致讀取請求。您可以搭配使用這個選用設定與 `load`、`query` 或 `scan` 操作。強烈一致讀取具有效能和帳單隱憂；如需詳細資訊，請參閱 [DynamoDB 產品詳細資訊頁面](#)。

如果您未指定映射器執行個體的讀取一致性設定，則預設值為 `EVENTUAL`。

Note

此值僅適用於 `DynamoDBMapper` 的 `query`、`querypage`、`load` 和 `batch load` 操作。

- `DynamoDBMapperConfig.PaginationLoadingStrategy` 列舉值：控制映射器執行個體如何處理分頁資料清單，例如 `query` 或 `scan` 的結果：
 - `LAZY_LOADING`：映射器執行個體會可能在可能時載入資料，並將所有載入的結果保留在記憶體中。
 - `EAGER_LOADING`：映射器執行個體會初始化清單時立即載入資料。
 - `ITERATION_ONLY`：您只能使用從清單中讀取的迭代器。在反覆運算期間，清單會先清除所有先前的結果，再載入下一頁，因此清單最多會將一頁的已載入結果保留在記憶體中。這也表示清單只能重複使用一次。處理大型項目時，建議使用此策略，以減少記憶體負擔。

如果您未指定映射器執行個體的分頁載入策略，則預設值為 `LAZY_LOADING`。

- `DynamoDBMapperConfig.SaveBehavior` 列舉值：指定映射器執行個體在儲存操作期間應該如何處理屬性：
 - `UPDATE`：在儲存操作期間，會更新所有建模屬性，未建模屬性則不受影響。基本數字類型 (byte、int、long) 設定為 0。物件類型設定為 Null。
 - `CLOBBER`：在儲存操作期間，清除和取代所有屬性 (包含未建模屬性)。透過刪除並重新建立項目來完成這項操作。使用版本控制的欄位限制條件也會予以忽略。

如果您未指定映射器執行個體的儲存行為，則預設值為 `UPDATE`。

Note

`DynamoDBMapper` 交易操作不支援 `DynamoDBMapperConfig.SaveBehavior` 列舉。

- `DynamoDBMapperConfig.TableNameOverride` 物件：指示映射器執行個體忽略類別之 `DynamoDBTable` 註釋所指定的資料表名稱，並改為使用您提供的不同資料表名稱。在執行時間將資料分割為多個資料表時，這十分有用。

您可以視需要覆寫每個操作的 `DynamoDBMapper` 預設組態物件。

含版本編號的樂觀鎖定

樂觀鎖定是一種策略，確保您要更新 (或刪除) 的用戶端項目與 Amazon DynamoDB 中的項目相同。如果您使用此策略，則會保護其他項目的寫入不會覆寫資料庫寫入，反之亦然。

使用樂觀鎖定，每個項目都有做為版本編號的屬性。如果您從資料表中擷取項目，則應用程式會記錄該項目的版本編號。您可以更新項目，但只有在伺服器端的版本編號尚未變更時。若版本不相符，表示已有其他人在您之前進行修改。更新嘗試失敗，因為您有此項目的過期版本。如果發生這種情況，請擷取項目，然後嘗試更新項目，再試一次。樂觀鎖定預防您意外覆寫其他人所作的變更。同時也預防其他人意外覆寫您所作的變更。

雖然您可以實現自己的樂觀鎖定策略，但 AWS SDK for Java 提供

了 `@DynamoDBVersionAttribute` 註釋。在資料表的映射類別中，您指定一個屬性來存放版本編號，並使用此註釋進行標示。當您儲存物件時，DynamoDB 資料表中的對應項目會有屬性可存放版本編號。當您第一次儲存物件時，`DynamoDBMapper` 會指派版本編號，並在每次更新項目時自動遞增版本編號。只有在用戶端物件版本與 DynamoDB 資料表中項目的對應版本編號相符時，您的更新或刪除請求才會成功。

如果發生下列情況，則會擲回 `ConditionalCheckFailedException`：

- 您使用含 `@DynamoDBVersionAttribute` 的樂觀鎖定，而且伺服器上的版本值與用戶端上的值不同。
- 您可以搭配使用 `DynamoDBMapper` 與 `DynamoDBSaveExpression`，以在儲存資料時指定自己的條件式限制條件，而且這些限制條件會失敗。

Note

- DynamoDB 全域資料表會在並行更新間使用「最後寫入者獲勝」核對機制。如果您使用的是全域資料表，最後寫入者政策獲勝。因此，在此情況下，鎖定政策不會如預期般運作。
- `DynamoDBMapper` 交易寫入操作不支援同一物件的 `@DynamoDBVersionAttribute` 標註和條件表達式。如果一個事務性寫入中的對象被註釋，`@DynamoDBVersionAttribute` 並且還具有一個條件表達式，那麼一個 `SdkClientException` 將被拋出。

例如，下列 Java 程式碼定義有數個屬性的 `CatalogItem` 類別。`Version` 屬性會標上 `@DynamoDBVersionAttribute` 註釋。

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;}
}
```

您可以將 `@DynamoDBVersionAttribute` 註釋套用至基本包裝函式類別所提供的可為 Null 類型，而基本包裝函式類別提供可為 Null 類型 (例如 Long 和 Integer)。

樂觀鎖定對這些 `DynamoDBMapper` 方法的影響如下：

- `save`：針對新項目，`DynamoDBMapper` 會指派初始版本編號 1。如果您擷取項目，請更新其一或多個屬性並嘗試儲存變更，只有在用戶端的版本編號與伺服器端的相符時，儲存操作才會成功。`DynamoDBMapper` 會自動遞增版本編號。
- `delete`：`delete` 方法會採用物件作為參數，而 `DynamoDBMapper` 會在刪除項目之前執行版本檢查。如果請求中指定 `DynamoDBMapperConfig.SaveBehavior.CLOBBER`，則可以停用版本檢查。

`DynamoDBMapper` 內的樂觀鎖定內部實作，使用 `DynamoDB` 所提供的條件式更新和條件式刪除支援。

- `transactionWrite` —
 - `Put`：針對新項目，`DynamoDBMapper` 會指派初始版本編號 1。如果您擷取項目，請更新其一或多個屬性並嘗試儲存變更，只有在用戶端的版本編號與伺服器端的相符時，`Put` 操作才會成功。`DynamoDBMapper` 會自動遞增版本編號。
 - `Update`：針對新項目，`DynamoDBMapper` 會指派初始版本編號 1。如果您擷取項目，請更新其一或多個屬性並嘗試儲存變更，只有在用戶端的版本編號與伺服器端的相符時，`update` 操作才會成功。`DynamoDBMapper` 會自動遞增版本編號。
 - `Delete`：`DynamoDBMapper` 會在刪除項目之前執行版本檢查。只有用戶端與伺服器端的版本號碼相符時，`delete` 操作才會成功。

- `ConditionCheck` : 不支援 `ConditionCheck` 操作的 `@DynamoDBVersionAttribute` 註釋。當一個 `ConditionCheck` 項目被註釋與 `SdkClientException @DynamoDBVersionAttribute` 將被拋出。

停用樂觀鎖定

若要停用樂觀鎖定，您可以將 `DynamoDBMapperConfig.SaveBehavior` 列舉值從 `UPDATE` 變更為 `CLOBBER`。您可以建立跳過版本檢查的 `DynamoDBMapperConfig` 執行個體來執行這項操作，並將此執行個體用於所有請求。如需 `DynamoDBMapperConfig.SaveBehavior` 和其他選用 `DynamoDBMapper` 參數的資訊，請參閱「[DynamoDBMapper 的選用組態設定](#)」。

您也只能設定特定操作的鎖定行為。例如，下列 Java 程式碼片段使用 `DynamoDBMapper` 來儲存型錄項目。指定 `DynamoDBMapperConfig.SaveBehavior` 的方式是將選用 `DynamoDBMapperConfig` 參數新增至 `save` 方法。

Note

`transactionWrite` 方法不支援 DynamoDB 援。MapperConfig SaveBehavior 配置。不支援停用 `transactionWrite` 的樂觀鎖定。

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));
```

映射任意資料

除了支援的 Java 類型之外 (請參閱 [適用於 Java 的 DynamoDB Mapper 支援的資料類型](#))，您還可以使用應用程式中未直接映射至 Amazon DynamoDB 類型的類型。若要映射這些類型，您必須提供將複雜類型轉換為 DynamoDB 支援之類型的實作 (反之亦然)，並使用 `@DynamoDBTypeConverted` 註釋來

標註複雜類型存取子方法。轉換器程式碼會在儲存或載入物件時轉換資料。它也用於使用複雜類型的所有操作。請注意，在查詢和掃描操作期間比較資料時，會針對 DynamoDB 中所存放的資料進行比較。

例如，請考慮下列可定義本身為 `CatalogItem` 之 `Dimension` 屬性的 `DimensionType` 類別。此屬性會將項目維度存放為高度、寬度和厚度。假設您決定將這些項目維度存放為 DynamoDB 中的字串 (例如 8.5x11x.05)。下列範例提供轉換器程式碼，以將 `DimensionType` 物件轉換為字串以及將字串轉換為 `DimensionType`。

Note

此程式碼範例假設您已根據 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 一節的說明將資料載入 DynamoDB 的帳戶。

如需執行下列範例的 step-by-step 指示，請參閱 [Java 程式碼範例](#)。

Example

```
public class DynamoDBMapperExample {

    static AmazonDynamoDB client;

    public static void main(String[] args) throws IOException {

        // Set the AWS region you want to access.
        Regions usWest2 = Regions.US_WEST_2;
        client = AmazonDynamoDBClientBuilder.standard().withRegion(usWest2).build();

        DimensionType dimType = new DimensionType();
        dimType.setHeight("8.00");
        dimType.setLength("11.0");
        dimType.setThickness("1.0");

        Book book = new Book();
        book.setId(502);
        book.setTitle("Book 502");
        book.setISBN("555-5555555555");
        book.setBookAuthors(new HashSet<String>(Arrays.asList("Author1", "Author2")));
        book.setDimensions(dimType);

        DynamoDBMapper mapper = new DynamoDBMapper(client);
        mapper.save(book);
    }
}
```

```
Book bookRetrieved = mapper.load(Book.class, 502);
System.out.println("Book info: " + "\n" + bookRetrieved);

bookRetrieved.getDimensions().setHeight("9.0");
bookRetrieved.getDimensions().setLength("12.0");
bookRetrieved.getDimensions().setThickness("2.0");

mapper.save(bookRetrieved);

bookRetrieved = mapper.load(Book.class, 502);
System.out.println("Updated book info: " + "\n" + bookRetrieved);
}

@DynamoDBTable(tableName = "ProductCatalog")
public static class Book {
    private int id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private DimensionType dimensionType;

    // Partition key
    @DynamoDBHashKey(attributeName = "Id")
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    @DynamoDBAttribute(attributeName = "Title")
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    @DynamoDBAttribute(attributeName = "ISBN")
    public String getISBN() {
        return ISBN;
    }
}
```



```
    }

    public void setISBN(String ISBN) {
        this.ISBN = ISBN;
    }

    @DynamoDBAttribute(attributeName = "Authors")
    public Set<String> getBookAuthors() {
        return bookAuthors;
    }

    public void setBookAuthors(Set<String> bookAuthors) {
        this.bookAuthors = bookAuthors;
    }

    @DynamoDBTypeConverted(converter = DimensionTypeConverter.class)
    @DynamoDBAttribute(attributeName = "Dimensions")
    public DimensionType getDimensions() {
        return dimensionType;
    }

    @DynamoDBAttribute(attributeName = "Dimensions")
    public void setDimensions(DimensionType dimensionType) {
        this.dimensionType = dimensionType;
    }

    @Override
    public String toString() {
        return "Book [ISBN=" + ISBN + ", bookAuthors=" + bookAuthors + ",
dimensionType= "
                + dimensionType.getHeight() + " X " + dimensionType.getLength() + "
X "
                + dimensionType.getThickness()
                + ", Id=" + id + ", Title=" + title + "]";
    }
}

static public class DimensionType {

    private String length;
    private String height;
    private String thickness;

    public String getLength() {
```

```
        return length;
    }

    public void setLength(String length) {
        this.length = length;
    }

    public String getHeight() {
        return height;
    }

    public void setHeight(String height) {
        this.height = height;
    }

    public String getThickness() {
        return thickness;
    }

    public void setThickness(String thickness) {
        this.thickness = thickness;
    }
}

// Converts the complex type DimensionType to a string and vice-versa.
static public class DimensionTypeConverter implements DynamoDBTypeConverter<String,
DimensionType> {

    @Override
    public String convert(DimensionType object) {
        DimensionType itemDimensions = (DimensionType) object;
        String dimension = null;
        try {
            if (itemDimensions != null) {
                dimension = String.format("%s x %s x %s",
itemDimensions.getLength(), itemDimensions.getHeight(),
                itemDimensions.getThickness());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return dimension;
    }
}
```

```
@Override
public DimensionType unconvert(String s) {

    DimensionType itemDimension = new DimensionType();
    try {
        if (s != null && s.length() != 0) {
            String[] data = s.split("x");
            itemDimension.setLength(data[0].trim());
            itemDimension.setHeight(data[1].trim());
            itemDimension.setThickness(data[2].trim());
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return itemDimension;
}
}
```

DynamoDBMapper 範例

下列 Java 程式碼範例示範如何使用 DynamoDBMapper 類別來執行各種操作。您可以使用這些範例來執行 CRUD、查詢、掃描、批次和交易操作。

主題

- [DynamoDBMapper CRUD 操作](#)
- [DynamoDBMapper 查詢和掃描操作](#)
- [DynamoDBMapper 批次操作](#)
- [DynamoDBMapper 交易操作](#)

DynamoDBMapper CRUD 操作

下列 Java 程式碼範例會宣告具有 Id、Title、ISBN 和 Authors 屬性的 CatalogItem 類別。它使用註釋將這些屬性映射至 DynamoDB 中的 ProductCatalog 資料表。範例接著會使用 DynamoDBMapper 來儲存、擷取和更新書籍物件，然後刪除書籍項目。

Note

此程式碼範例假設您已根據 [在 DynamoDB 中建立資料表](#)，以及 [載入程式碼範例的資料](#) 一節的說明將資料載入 DynamoDB 的帳戶。

如需執行下列範例的 step-by-step 指示，請參閱 [Java 程式碼範例](#)。

匯入

```
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;
```

Code

```
public class DynamoDBMapperCRUDEExample {

    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.");
}
}
```

DynamoDBMapper 查詢和掃描操作

本節中的 Java 範例定義下列類別，並將它們映射至 Amazon DynamoDB 中的資料表。如需建立範例資料表的詳細資訊，請參閱「[在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#)」。

- Book 類別映射至 ProductCatalog 資料表。
- Forum、Thread 與 Reply 類別映射至同名的資料表。

範例接著會使用 DynamoDBMapper 執行個體來執行下列查詢和掃描操作。

- 依 Id 取得書籍。

ProductCatalog 資料表有 Id 作為其主索引鍵。它沒有排序索引鍵做為其主索引鍵的一部分。因此，您無法查詢資料表。您可以使用項目的 Id 值來取得項目。

- 針對 Reply 資料表執行下列查詢。

Reply 資料表的主索引鍵由 Id 和 ReplyDateTime 屬性組成。ReplyDateTime 是排序索引鍵。因此，您可以查詢此資料表。

- 尋找過去 15 天所張貼的論壇主題回覆。
- 尋找特定日期範圍內所張貼的論壇主題回覆。
- 掃描 ProductCatalog 資料表，尋找價格低於指定值的書籍。

基於效能考量，您應該使用查詢而不是掃描操作。不過，您有時可能需要掃描資料表。假設發生資料輸入錯誤，而其中一個書籍價格設定為低於 0。此範例會掃描 ProductCategory 資料表，尋找價格低於 0 的書籍項目 (ProductCategory 為 book)。

- 對 ProductCatalog 資料表執行平行掃描來尋找特定類型的自行車。

Note

此程式碼範例假設您已根據 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 一節的說明將資料載入 DynamoDB 的帳戶。

如需執行下列範例的 step-by-step 指示，請參閱[Java 程式碼範例](#)。

匯入

```
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;
```

Code

```
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;
}
}
```

DynamoDBMapper 批次操作

下列 Java 程式碼範例會宣告 Book、Forum、Thread 和 Reply 類別，並使用 DynamoDBMapper 類別將它們映射至 Amazon DynamoDB 資料表。

此程式碼會說明下列批次寫入操作：

- batchSave 將書籍項目放入 ProductCatalog 資料表之中。
- batchDelete 將項目從 ProductCatalog 資料表中刪除。
- batchWrite 將在 Forum 與 Thread 資料表中放入及刪除項目。

如需此範例中所使用資料表的詳細資訊，請參閱 [在 DynamoDB 中建立資料表](#)，以及載入程式碼範例的 [資料](#)。如需測試下列範例的 step-by-step 指示，請參閱 [Java 程式碼範例](#)。

匯入

```
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;
```

Code

```
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;
    }
}
}
```

DynamoDBMapper 交易操作

下列 Java 程式碼範例會宣告 Forum 和 Thread 類別，並使用 DynamoDBMapper 類別將它們映射至 DynamoDB 資料表。

此程式碼會說明下列交易操作：

- `transactionWrite`，可在單一交易中新增、更新和刪除一或多個資料表中的多個項目。
- `transactionLoad`，可在單一交易中從一或多個資料表擷取多個項目。

匯入

```
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;
```

Code

```
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 增強型用戶端

DynamoDB 增強型用戶端是AWS SDK for Java 2 (v2) 版中的高階程式庫。提供將客戶端類別映射到 DynamoDB 資料表的直接方式。您要在自己的程式碼中定義資料表及其對應模型類別之間的關係。

定義關係之後，您可以直觀地對 DynamoDB 中的資料表或項目執行各種建立、讀取、更新或刪除 (CRUD) 操作。

如需有關如何將增強型用戶端與 DynamoDB 搭配使用的詳細資訊，請參閱[AWS SDK for Java 2.x 中的使用 DynamoDB 增強型用戶端](#)。

.NET：文件模型

AWS SDK for .NET 提供文件模型類別，包裝一部分低階 Amazon DynamoDB 操作，可進一步簡化您的程式碼編寫。在文件模型中，主類別為 Table 和 Document。Table 類別提供資料操作方法，例如 PutItem、GetItem 和 DeleteItem。該類別也提供 Query 和 Scan 方法。Document 類別則代表資料表中的單一項目。

上述文件模型類別可在 Amazon.DynamoDBv2.DocumentModel 命名空間中取得。

Note

您無法使用文件模型建立、更新和刪除資料表。但文件模型支援大部分的常見資料操作。

主題

- [支援的資料類型](#)
- [利用 AWS SDK for .NET 文件模型使用 DynamoDB 中的項目](#)
- [範例：使用 AWS SDK for .NET 文件模型進行 CRUD 操作](#)
- [範例：使用 AWS SDK for .NET 文件模型 API 的批次操作](#)
- [利用 AWS SDK for .NET 文件模型使用 DynamoDB 中的資料表](#)

支援的資料類型

文件模型會支援一組原始的 .NET 資料類型和集合資料類型。模型目前支援下列基本資料類型。

- bool
- byte
- char
- DateTime

- decimal
- double
- float
- Guid
- Int16
- Int32
- Int64
- SByte
- string
- UInt16
- UInt32
- UInt64

下表摘要說明上述 .NET 類型與 DynamoDB 類型的映射。

.NET 基本類型	DynamoDB 類型
所有數字類型	N (數字類型)
所有字串類型	S (字串類型)
MemoryStream , 字節 []	B (二進位類型)
bool	N (數字類型)。0 代表 false , 1 代表 true。
DateTime	S (字串類型)。DateTime 值會以 ISO-8601 格式字串存放。
Guid	S (字串類型)。
集合類型 (列表 HashSet , 和數組)	BS (二進位集) 類型、SS (字串集) 類型和 NS (數字集) 類型。

AWS SDK for .NET 會定義將 DynamoDB 的布林值、null 值、清單和映射類型映射到 .NET 文件模型 API 的類型：

- 使用 `DynamoDBBool` 作為布林類型。
- 使用 `DynamoDBNull` 作為 `null` 類型。
- 使用 `DynamoDBList` 作為清單類型。
- 使用 `Document` 作為映射類型。

Note

- 支援空白的二進位值。
- 支援讀取空白字串值。寫入 DynamoDB 時，字串 `Set` 類型的屬性值會支援空白字串的屬性值。List 或 Map 類型中包含的字串類型的空白字串屬性值和空白字串值會從寫入請求中捨棄

利用 AWS SDK for .NET 文件模型使用 DynamoDB 中的項目

下列程式碼範例示範如何使用 AWS SDK for .NET 文件模型來執行各種操作。您可以使用這些範例來執行 CRUD、批次和交易操作。

主題

- [把一個項目-表。PutItem 方法](#)
- [指定選用參數](#)
- [獲取項目-表。GetItem](#)
- [刪除物件-表格。DeleteItem](#)
- [更新項目-表。UpdateItem](#)
- [批次寫入 - 放入和刪除多個項目](#)

若要使用文件模型執行資料操作，您必須先呼叫 `Table.LoadTable` 方法，該方法會建立代表特定資料表之 `Table` 類別的執行個體。以下 C# 程式碼範例會建立代表 Amazon DynamoDB 中 `ProductCatalog` 資料表的 `Table` 物件。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");
```

Note

一般來說，您會在應用程式的開頭使用 `LoadTable` 方法一次，因為它會呼叫 `DescribeTable`，並新增到 DynamoDB 的來回行程。

您接著便可使用 `Table` 物件執行不同資料操作。每一種資料操作都具有兩種類型的多載：一種只需接受最低需求的參數，另一種則接受選用的操作專屬組態資訊。例如，若要擷取項目，您必須提供資料表的主索引鍵值。在此案例中，您可以使用以下 `GetItem` 多載。

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);
```

您也可以傳遞選用的參數到這些方法。例如，上述 `GetItem` 會傳回整個項目，包含其全部的屬性。您可以選擇性的指定要擷取的屬性清單。在此案例中，您會使用以下 `GetItem` 多載，接受該操作專屬的組態物件參數。

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);
```

您可以使用組態物件指定數個選用參數，例如請求特定屬性清單，或指定頁面大小 (每一頁面的項目數)。每一種資料操作方法都具有自己的組態類別。舉例說明，您可以使用

GetItemOperationConfig 類別來為 GetItem 操作提供選項。舉例說明，您可以使用 PutItemOperationConfig 類別來為 PutItem 操作提供選用參數。

下節會討論 Table 類別支援的每一種資料操作。

把一個項目-表。 PutItem 方法

PutItem 方法會將輸入的 Document 執行個體上傳到資料表。若輸入 Document 中指定之主索引鍵的項目已存在於資料表中，則 PutItem 操作會取代整個現有項目。新項目和您為 PutItem 方法提供的 Document 物件完全相同。若您的原始項目具有任何額外的屬性，新的項目中便不再有這些屬性。

以下是使用 AWS SDK for .NET 文件模型將新項目放入資料表的步驟。

1. 執行 Table.LoadTable 方法，該方法會提供您希望放入項目之資料表的名稱。
2. 建立 Document 物件。該物件具有屬性名稱清單和其數值。
3. 藉由將 Document 執行個體提供為參數，執行 Table.PutItem。

下列 C# 程式碼範例示範上述工作。範例會將項目上傳至 ProductCatalog 資料表。

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);
```

在上述範例中，Document 實例會建立具有 Number、String、String Set、Boolean、和 Null 屬性的項目。（用 Null 於表示此產品 QuantityOnHand 的未知。）針對 Boolean 和 Null，請使用 DynamoDBBool 和 DynamoDBNull 建構函數方法。

在 DynamoDB 中，List 和 Map 資料類型可包含由其他資料類型組成的元素。下述為將這些資料類型映射到文件模型 API 的方式：

- List：使用 `DynamoDBList` 建構函數。
- Map：使用 `Document` 建構函數。

您可以修改上述範例，為項目新增 `List` 屬性。若要執行此作業，請使用 `DynamoDBList` 建構函數，如以下程式碼範例所示。

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);
```

若要為書籍新增 `Map` 屬性，您可以定義另一個 `Document`。以下程式碼範例示範如何執行此作業。

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);
```

這些範例是以「[使用表達式時指定項目屬性](#)」中顯示的項目為基礎的。文件模型可讓您建立複雜的巢狀屬性，例如在案例研究中顯示的 ProductReviews 屬性。

指定選用參數

您可以透過新增 PutItem 參數，設定 PutItemOperationConfig 操作的選用參數。如需取得選用參數的完整清單，請參閱「[PutItem](#)」。下列 C# 程式碼範例會將項目放於 ProductCatalog 資料表。它指定以下選用參數：

- 要執行此條件式放入請求的 ConditionalExpression 參數。範例會建立表達式，指定 ISBN 屬性必須具有您要取代之項目中存在的特定值。

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);
```

獲取項目-表。 GetItem

GetItem 操作會將項目擷取為 Document 執行個體。您必須提供要擷取之項目的主索引鍵，如以下 C# 程式碼範例中所示。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");
Document document = table.GetItem(101); // Primary key 101.
```

GetItem 操作會傳回項目所有的屬性，並根據預設執行最終一致讀取 (請參閱「[讀取一致性](#)」)。

指定選用參數

您可以透過新增 GetItem 參數，為 GetItemOperationConfig 操作設定額外的選項。如需取得選用參數的完整清單，請參閱「[GetItem](#)」。下列 C# 程式碼範例會從 ProductCatalog 資料表擷取項目。它指定 GetItemOperationConfig，提供下列選用參數：

- AttributesToGet 參數用於只擷取特定的屬性。
- ConsistentRead 參數用於請求所有指定之屬性的最新值。如要進一步了解資料一致性，請參閱「[讀取一致性](#)」。

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);
```

當您使用文件模型 API 擷取項目時，您可以存取傳回之 Document 物件中的個別元素，如下列範例所示。

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();
```

針對類型為 List 或 Map 的屬性，以下為將這些屬性映射到文件模型 API 的方式：

- List：使用 AsDynamoDBList 方法。
- Map：使用 AsDocument 方法。

下列程式碼範例會示範如何從 Document 物件擷取 List Map (RelatedItems) 和 a (Pictures)：

Example

```
DynamoDBList relatedItems = doc["RelatedItems"].AsDynamoDBList();

Document pictures = doc["Pictures"].AsDocument();
```

刪除物件-表格。 DeleteItem

DeleteItem 操作會從資料表刪除項目。您可以將項目的主索引鍵作為參數傳遞。或者，若您已讀取該項目，並具有相對應的 Document 物件，您可以將其做為參數傳遞給 DeleteItem 方法，如以下 C# 程式碼範例所示。

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)
```

指定選用參數

您可以透過新增 Delete 參數，為 DeleteItemOperationConfig 操作設定額外的選項。如需取得選用參數的完整清單，請參閱「[DeleteTable](#)」。以下 C# 程式碼範例在指定以下兩個選用參數：

- `ConditionalExpression` 參數用於確認要刪除之書籍項目的 ISBN 屬性具有指定的值。
- `ReturnValues` 參數用於請求 `Delete` 方法傳回已刪除的項目。

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);
```

更新項目-表。 UpdateItem

`UpdateItem` 操作會更新現有的項目 (若存在的話)。若找不到具有指定之主索引鍵的項目，`UpdateItem` 操作會新增新的項目。

您可以使用 `UpdateItem` 操作更新現有的屬性值、為現有的集合新增新的屬性，或從現有的集合刪除屬性。您可以藉由建立描述欲執行之更新的 `Document` 執行個體，提供這些更新。

`UpdateItem` 動作會使用以下準則：

- 若還沒有該項目，`UpdateItem` 會使用輸入中指定的主索引鍵新增項目。
- 若已有該項目，`UpdateItem` 會套用更新，如下所示：
 - 將現有的屬性值取代為更新的值。
 - 若您在輸入中提供的屬性不存在，它會為項目新增新的屬性。
 - 若輸入的屬性值為 `Null`，它會刪除該屬性 (若有的話)。

Note

此中層 UpdateItem 作業不支援基礎 DynamoDB 作業支援的 Add 動作 (請參閱 [UpdateItem](#))。

Note

PutItem 操作 ([把一個項目-表。PutItem 方法](#)) 也可以執行更新。若您呼叫 PutItem 上傳項目，且該項目的主索引鍵已存在，PutItem 操作會取代整個項目。若現有項目中具有並未在放入 Document 上指定的屬性，PutItem 操作會刪除那些屬性。但是，UpdateItem 只會更新指定的輸入屬性。任何其他該項目現有的屬性都不會變更。

以下是使用 AWS SDK for .NET 文件模型更新項目的步驟：

1. 藉由提供您希望執行更新操作之資料表的名稱，執行 Table.LoadTable 方法。
2. 藉由提供所有您要執行的更新，建立 Document 執行個體。

若要刪除現有的屬性，請將屬性值指定為 Null。

3. 呼叫 Table.UpdateItem 方法並提供 Document 執行個體做為輸入參數。

您必須在 Document 執行個體中或明確將其做為其中一個參數，提供主索引鍵。

下列 C# 程式碼範例示範上述工作。程式碼範例會更新 Book 資料表中的項目。UpdateItem 操作會更新現有的 Authors 屬性，刪除 PageCount 屬性，並新增新的 XYZ 屬性。Document 執行個體包含要更新之書籍的主索引鍵。

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);
```

指定選用參數

您可以透過新增 `UpdateItem` 參數，為 `UpdateItemOperationConfig` 操作設定額外的選項。如需取得選用參數的完整清單，請參閱「[UpdateItem](#)」。

以下 C# 程式碼範例會將書籍項目的價格更新為 25。它會指定以下兩個選用參數：

- `ConditionalExpression` 參數會識別數值為 20，並且您預期會存在的 `Price` 屬性。
- `ReturnValues` 參數會請求 `UpdateItem` 操作傳回已更新的項目。

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.ExpressionAttributeValueValues[":val"] = "20";

UpdateItemOperationConfig config = new UpdateItemOperationConfig()
{
    ConditionalExpression = expr,
    ReturnValues = ReturnValues.AllOldAttributes
};

Document d1 = table.Update(book, config);
```

批次寫入 - 放入和刪除多個項目

批次寫入表示在一個批次中放入和刪除多個項目。您可利用此操作，在單一呼叫中對來自一或多個資料表的多個項目，進行放入與刪除。以下是使用 AWS SDK for .NET 文件模型 API 將多個項目放入資料表或從資料表刪除多個項目的步驟。

1. 透過提供您希望執行批次操作之資料表的名稱，執行 `Table.LoadTable` 方法，建立 `Table` 物件。
2. 在您於上述步驟中建立的資料表執行個體上執行 `createBatchWrite` 方法，建立 `DocumentBatchWrite` 物件。
3. 使用 `DocumentBatchWrite` 物件方法指定您要上傳或刪除的文件。
4. 呼叫 `DocumentBatchWrite.Execute` 方法執行批次操作。

當使用文件模型 API 時，您可以在單一批次中指定任何數目的操作。但 DynamoDB 會限制批次中的操作數目，以及批次操作中的批次總大小。如需特定限制的詳細資訊，請參閱「[BatchWriteItem](#)」。若文件模型 API 偵測到您的批次寫入請求超過允許的寫入請求次數，或是批次的 HTTP 裝載大小超過 `BatchWriteItem` 所允許的限制，它會將批次分成數個較小的批次。此外，如果批次寫入的回應傳回未處理的項目，文件模型 API 會自動傳送含有這些未處理項目的另一個批次請求。

下列 C# 程式碼範例示範前述步驟。程式碼範例會使用批次寫入操作執行兩次寫入：上傳一個書籍項目，及刪除另一個書籍項目。

```
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();
```

如需運作範例，請參閱 [範例：使用 AWS SDK for .NET 文件模型 API 的批次操作](#)。

您可以使用 `batchWrite` 操作在多個資料表上執行放入和刪除操作。以下是 AWS SDK for .NET 文件模型放入多個資料表或從多個資料表刪除項目的步驟。

1. 您會為每個希望放入或刪除多個項目的資料表建立 `DocumentBatchWrite` 執行個體，如上述程序中所說明。
2. 建立 `MultiTableDocumentBatchWrite` 執行個體，然後將個別 `DocumentBatchWrite` 物件新增到該執行個體。
3. 執行 `MultiTableDocumentBatchWrite.Execute` 方法。

下列 C# 程式碼範例示範前述步驟。程式碼範例會使用批次寫入操作，執行以下寫入操作：

- 在 `Forum` 資料表項目中放入一個新的項目。
- 在 `Thread` 資料表放入一個項目，並從相同資料表中刪除一個項目。

```
// 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();
```

範例：使用 AWS SDK for .NET 文件模型進行 CRUD 操作

以下 C# 程式碼範例會執行下列動作：

- 在 ProductCatalog 資料表中建立書籍項目。
- 擷取書籍項目。
- 更新書籍項目。程式碼範例會示範新增新屬性和更新現有屬性的一般更新。在範例中也會示範條件式更新。只有在現有價格的數值為程式碼中指定的數值時，才更新書籍的價格。
- 刪除書籍項目。

如需測試下列範例的 step-by-step 指示，請參閱[.NET 程式碼範例](#)。

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);
    }
}
}
```

範例：使用 AWS SDK for .NET 文件模型 API 的批次操作

主題

- [範例：使用 AWS SDK for .NET 文件模型進行批次寫入](#)

範例：使用 AWS SDK for .NET 文件模型進行批次寫入

以下 C# 程式碼範例示範單一資料表和多個資料表的批次寫入操作。範例會執行下列任務：

- 說明單一資料表批次寫入。將兩個項目新增至 ProductCatalog 資料表。
- 說明多個資料表批次寫入。將一個項目同時新增至 Forum 和 Thread 資料表，並從 Thread 資料表中刪除一個項目。

如已完成「[在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#)」中的步驟，則您已建立 ProductCatalog、Forum 和 Thread 資料表。您也可以利用程式設計方式來建立這些範例資料表。如需詳細資訊，請參閱 [建立範例資料表並使用 AWS SDK for .NET](#)。如需測試下列範例的 step-by-step 指示，請參閱 [.NET 程式碼範例](#)。

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();
    }
}
}
```

利用 AWS SDK for .NET 文件模型使用 DynamoDB 中的資料表

主題

- [AWS SDK for .NET 中的 Table.Query 方法](#)
- [AWS SDK for .NET 中的 Table.Scan 方法](#)

AWS SDK for .NET 中的 Table.Query 方法

Query 方法可讓您查詢您的資料表。您只能查詢具有複合主索引鍵 (分割區索引鍵和排序索引鍵) 的資料表。若您資料表的主索引鍵僅由分割區索引鍵組成，將不支援 Query 操作。根據預設，Query 會於內部執行最終一致的查詢。若要進一步了解一致性模型，請參閱「[讀取一致性](#)」。

Query 方法提供兩種多載。Query 方法的最低需求參數為分割區索引鍵值和排序索引鍵篩選條件。您可以使用以下多載，提供這些最低需求參數。

Example

```
Query(Primitive partitionKey, RangeFilter Filter);
```

例如，以下 C# 程式碼會查詢所有在過去 15 天內張貼的論壇回覆。

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);
```

這會建立 Search 物件。您現在可以反覆呼叫 Search.GetNextSet 方法，一次擷取一頁結果，如以下 C# 程式碼範例所示。程式碼會印出每個查詢傳回項目的屬性值。

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);
    }
}
```

指定選用參數

您也可以為 Query 指定選用參數，例如指定要擷取的屬性清單、強烈一致讀取、頁面大小，以及每一頁傳回的項目數。如需有關參數的完整清單，請參閱 [Query](#)。若要指定選用參數，您必須使用以下多載，並在其中提供 QueryOperationConfig 物件。

Example

```
Query(QueryOperationConfig config);
```

假設您希望執行上述範例中的查詢 (擷取在過去 15 天內張貼的論壇回覆)。但是，假設您希望提供選用的查詢參數，只擷取指定屬性並請求強烈一致讀取。以下 C# 程式碼範例會使用 QueryOperationConfig 物件建構請求。

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);
```

範例：使用 Table.Query 方法的查詢

以下 C# 程式碼範例會使用 Table.Query 方法執行下列範例查詢。

- 以下查詢會針對 Reply 資料表執行。
 - 尋找在過去 15 天內張貼的論壇主題回覆。

此查詢會執行兩次。在第一次 Table.Query 呼叫中，範例只會提供必要的查詢參數。在第二次 Table.Query 呼叫中，您會提供選用查詢參數，請求強烈一致讀取和要擷取的屬性清單。

- 尋找在一段期間內張貼的論壇主題回覆。

此查詢會使用 Between 查詢運算子，尋找在兩個日期間張貼的回覆。

- 從 ProductCatalog 資料表中取得產品。

因為 ProductCatalog 資料表的主索引鍵僅由分割區索引鍵構成，所以您只能取得項目，而無法查詢資料表。範例會使用項目 Id 擷取特定產品項目。

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 { }
        }
    }
}
```

```
    }
    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())
        {
            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);
        }
    }
}
```

```
}
```

AWS SDK for .NET 中的 Table.Scan 方法

Scan 方法會執行完整資料表掃描。它提供兩種多載。Scan 方法的唯一必要參數為掃描篩選條件，您可以使用以下多載提供。

Example

```
Scan(ScanFilter filter);
```

例如，假設您維持一個論壇主題資料表的運作，追蹤像是對話標題 (主要)、相關訊息、對話歸屬的論壇 Id、Tags 及其他相關資訊。假設標題為主索引鍵。

Example

```
Thread(Subject, Message, ForumId, Tags, LastPostedDateTime, .... )
```

此為您在 AWS 論壇上看到內容的簡化版論壇及對話 (請參閱[開發論壇](#))。下列 C# 程式碼範例會查詢特定論壇 (ForumId = 101) 中標記為「sortkey」的所有執行緒。因 ForumId 不是主索引鍵，範例會掃描資料表。ScanFilter 包含兩個條件。該查詢會傳回所有同時符合這兩個條件的對話。

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);
```

指定選用參數

您也可以為 Scan 指定選用參數，例如要擷取的特定屬性清單，或是是否要執行強烈一致讀取。若要指定選用參數，您必須建立 ScanOperationConfig 物件，其中包含必要和選用參數，並使用以下多載。

Example

```
Scan(ScanOperationConfig config);
```

以下 C# 程式碼範例會執行的查詢與前述相同 (尋找 ForumId 為 101 且 Tag 屬性包含「sortkey」關鍵字的論壇主題)。假設您希望新增一個選用參數，只擷取特定屬性清單。在此案例中，您必須藉由提供所有參數 (必要及選用參數) 建立 ScanOperationConfig 物件，如以下程式碼範例所示。

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);
```

範例：使用 Table.Scan 方法的掃描

由於 Scan 操作會執行完整資料表掃描，因此屬於潛在上相當昂貴的操作。您應改為使用查詢。但是，有時候您可能需要對資料表執行掃描。例如，您的產品定價中可能含有資料項目錯誤，而必須掃描整個資料表，如以下 C# 程式碼範例所示。範例會掃描 ProductCatalog 資料表，尋找價格值低於 0 的產品。範例會示範使用兩個 Table.Scan 多載。

- Table.Scan 會接受 ScanFilter 物件做為參數。

您可以在只傳遞必要參數時，傳遞 ScanFilter 參數。

- Table.Scan 會接受 ScanOperationConfig 物件做為參數。

若您希望將任何選用參數傳遞到 ScanOperationConfig 方法，您必須使用 Scan 參數。

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：物件持久性模型

主題

- [DynamoDB 屬性](#)
- [DynamoDBContext 類別](#)
- [支援的資料類型](#)
- [透過 DynamoDB 使用版本號碼進行最佳鎖定，使用 AWS SDK for .NET 物件持續性模型](#)
- [使用 AWS SDK for .NET 物件持續性模型將任意資料與 DynamoDB 對應](#)
- [使用 AWS SDK for .NET 物件持久性模型的批次操作](#)
- [範例：使用 AWS SDK for .NET 物件持久性模型的 CRUD 操作](#)
- [範例：使用 AWS SDK for .NET 物件持續性模型進行 Batch 寫入作業](#)
- [範例：使用 AWS SDK for .NET 物件持久性模型在 DynamoDB 中查詢及掃描](#)

提 AWS SDK for .NET 供物件持續性模型，可讓您將用戶端類別對應至 Amazon DynamoDB 表格。然後每個物件執行個體會映射至相對應資料表中的某個項目。為了將用戶端物件儲存至資料表，物件持久性模型會提供 `DynamoDBContext` 類別 (即 DynamoDB 的進入點)。此類別可讓您連線至 DynamoDB，繼而存取資料表、執行各種 CRUD 操作以及執行查詢。

物件持久性模型提供了一組屬性，可將用戶端類別映射到資料表，並將屬性/欄位映射至資料表屬性。

Note

物件持久性模型不提供用於建立、更新或刪除資料表的 API。它只提供資料操作。您只能使用 AWS SDK for .NET 低階 API 來建立、更新和刪除資料表。如需詳細資訊，請參閱 [在 .NET 中處理 DynamoDB 資料表](#)。

以下範例會示範物件持久性模型的運作方式。它會從 `ProductCatalog` 資料表開始作業。它將 `Id` 作為主索引鍵。

```
ProductCatalog(Id, ...)
```

假設您的 Book 類別具有 Title、ISBN 以及 Authors 屬性。您可以透過新增物件持久性模型所定義的屬性，將 Book 類別映射至 ProductCatalog 資料表中，如下列 C# 程式碼範例所示。

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; }
}
```

在上述範例中，DynamoDBTable 屬性會將 Book 類別映射至 ProductCatalog 資料表。

物件持久性模型支援類別屬性與資料表屬性之間的明確映射和預設映射。

- **明確映射：**若要將屬性映射至主索引鍵，您必須使用 DynamoDBHashKey 和 DynamoDBRangeKey 物件持久性模型屬性。此外，對於非主索引鍵屬性，如果類別中的屬性名稱和要對應的資料表屬性不相同，則必須明確新增 DynamoDBProperty 屬性來定義映射。

在上述範例中，Id 屬性會映射至具有相同名稱的主索引鍵，BookAuthors 屬性則會映射至 ProductCatalog 資料表中的 Authors 屬性。

- **預設映射：**依預設，物件持久性模型會將類別屬性映射至資料表中具有相同名稱的屬性。

在上述範例中，屬性 Title 和 ISBN 會映射至 ProductCatalog 資料表中具有相同名稱的屬性。

您不必映射每個類別屬性。您可以新增 DynamoDBIgnore 屬性來識別這些屬性。當您將 Book 執行個體儲存至資料表時，DynamoDBContext 不會包含 CoverPage 屬性。在擷取書籍執行個體時，它也不會傳回此屬性。

您可以映射 .NET 基本類型的屬性，如 `int` 和字串。只要提供適當的轉換器將任意資料映射至其中一種 DynamoDB 類型，您也可以映射任意資料類型。若要進一步了解如何映射任意類型，請參閱 [使用 AWS SDK for .NET 物件持續性模型將任意資料與 DynamoDB 對應](#)。

物件持久性模型支援樂觀鎖定。在更新操作期間，這可確保您擁有即將更新項目的最新副本。如需詳細資訊，請參閱 [透過 DynamoDB 使用版本號碼進行最佳鎖定，使用 AWS SDK for .NET 物件持續性模型](#)。

DynamoDB 屬性

本節說明物件持久性模型所提供的屬性，讓您可以將類別和屬性映射至 DynamoDB 資料表和屬性。

Note

在下列屬性中，只有 `DynamoDBTable` 和 `DynamoDBHashKey` 是必要屬性。

DynamoDB GlobalSecondaryIndexHashKey

將類別屬性映射至全域次要索引的分割區索引鍵。如果您需要對全域次要索引進行 Query，請使用此屬性。

DynamoDB GlobalSecondaryIndexRangeKey

將類別屬性映射至全域次要索引的排序索引鍵。如果您需要對全域次要索引進行 Query，並且想要使用索引的排序索引鍵來縮小您結果的範圍，請使用此屬性。

DynamoDB HashKey

將類別屬性映射至資料表主索引鍵的分割區索引鍵。主索引鍵屬性不能是集合類型。

下列 C# 程式碼範例會映射 `Book` 類別至 `ProductCatalog` 資料表，並映射 `Id` 屬性至資料表主索引鍵的分割區索引鍵。

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }

    // Additional properties go here.
}
```

```
}
```

DynamoDBIgnore

指出應該忽略的相關聯屬性。如果您不想儲存任何類別屬性，可以新增此屬性來指示 `DynamoDBContext` 將物件儲存到資料表時不包含此屬性。

DynamoDB LocalSecondaryIndexRangeKey

將類別屬性映射至本機次要索引的排序索引鍵。如果您需要對本機次要索引進行 Query，並且想要使用索引的排序索引鍵來縮小您結果的範圍，請使用此屬性。

DynamoDBProperty

將類別屬性映射至資料表屬性。如果類別屬性映射至具有相同名稱的資料表屬性，則不需要指定此屬性。不過，如果名稱不同，您可以使用此標籤提供映射。在下列 C# 陳述式中，`DynamoDBProperty` 會將 `BookAuthors` 屬性映射至資料表中的 `Authors` 屬性。

```
[DynamoDBProperty("Authors")]  
public List<string> BookAuthors { get; set; }
```

當儲存物件資料至相應的資料表時，`DynamoDBContext` 會使用此映射資訊來建立 `Authors` 屬性。

可重命名動態

指定類別屬性的替代名稱。如果您正在撰寫自訂轉換器，以將任意資料映射至 `DynamoDB` 資料表 (其中類別屬性的名稱與資料表屬性不同)，則此功能非常有用。

DynamoDB RangeKey

將類別屬性映射至資料表主索引鍵的排序索引鍵。如果資料表具有複合主索引鍵 (分割區索引鍵和排序索引鍵)，則必須在類別映射中同時指定 `DynamoDBHashKey` 和 `DynamoDBRangeKey` 屬性。

例如，範例資料表 `Reply` 有一個由 `Id` 分割區索引鍵和 `Replenishment` 排序索引鍵組成的主索引鍵。下列 C# 程式碼範例將 `Reply` 類別映射至 `Reply` 資料表。類別定義也會指出其中兩個屬性映射至主索引鍵。

如需範例資料表的詳細資訊，請參閱 [在 DynamoDB 中建立資料表](#)，以及 [載入程式碼範例的資料](#)。

```
[DynamoDBTable("Reply")]  
public class Reply
```

```
{
  [DynamoDBHashKey]
  public int ThreadId { get; set; }
  [DynamoDBRangeKey]
  public string Replenishment { get; set; }

  // Additional properties go here.
}
```

DynamoDBTable

識別 DynamoDB 中類別要映射的目標資料表。例如，下列 C# 程式碼範例將 Developer 類別映射至 DynamoDB 中的 People 資料表。

```
[DynamoDBTable("People")]
public class Developer { ...}
```

此屬性可被繼承或覆寫。

- DynamoDBTable 屬性可被繼承。在上述範例中，如果新增新從 Developer 類別繼承的新類別 Lead，它也會映射至 People 資料表。Developer 和 Lead 物件都存放在 People 資料表中。
- DynamoDBTable 屬性也可被寫。在下列 C# 程式碼範例中，Manager 類別繼承自 Developer 類別。但是，明確新增 DynamoDBTable 屬性會將類別映射到另一個資料表 (Managers)。

```
[DynamoDBTable("Managers")]
public class Manager : Developer { ...}
```

您可以新增選用參數 LowerCamelCaseProperties，以便請求 DynamoDB 在儲存物件至資料表時，將屬性名稱的第一個字母變成小寫，如下列 C# 範例所示。

```
[DynamoDBTable("People", LowerCamelCaseProperties=true)]
public class Developer
{
  string DeveloperName;
  ...
}
```

在儲存 Developer 類別的執行個體時，DynamoDBContext 會儲存 DeveloperName 屬性作為 developerName。

動態版本

標識用於儲存項目版本編號的類別屬性。如需版本控制的詳細資訊，請參閱「[透過 DynamoDB 使用版本號碼進行最佳鎖定，使用 AWS SDK for .NET 物件持續性模型](#)」。

DynamoDBContext 類別

DynamoDBContext 類別是 Amazon DynamoDB 資料庫的進入點。它可讓您連線至 DynamoDB，繼而存取各種資料表中的資料、執行各種 CRUD 操作以及執行查詢。此 DynamoDBContext 類別提供下列方法。

創建 MultiTable BatchGet

建立由多個個別 BatchGet 物件組成的 MultiTableBatchGet 物件。其中的每個 BatchGet 物件都可用來從單一 DynamoDB 資料表中擷取項目。

若要從資料表擷取項目，請使用 ExecuteBatchGet 方法，將 MultiTableBatchGet 物件作為參數傳遞。

創建 MultiTable BatchWrite

建立由多個個別 BatchWrite 物件組成的 MultiTableBatchWrite 物件。其中的每個 BatchWrite 物件可用來寫入或刪除單一 DynamoDB 資料表中的項目。

若要寫入資料表，請使用 ExecuteBatchWrite 方法，將 MultiTableBatchWrite 物件作為參數傳遞。

CreateBatchGet

建立 BatchGet 物件，您可以使用該物件從資料表中擷取多個項目。如需詳細資訊，請參閱 [批次取得：取得多個項目](#)。

createBatchWrite

建立 BatchWrite 物件，您可以使用該物件將多個項目放入資料表中，或從資料表中刪除多個項目。如需詳細資訊，請參閱 [批次寫入：放入和刪除多個項目](#)。

Delete

刪除資料表中的項目。這個方法需要您希望刪除之項目的主索引鍵。您可以提供主索引鍵值或包含主索引鍵值的用戶端物件，作為此方法的參數。

- 如果指定用戶端物件作為參數，且已啟用樂觀鎖定，則只有在物件的用戶端和伺服器端版本相符時，才能成功刪除項目。
- 如果只指定主索引鍵值作為參數，無論您是否已啟用樂觀鎖定，刪除都會成功。

Note

若要在背景執行此操作，請改用 `DeleteAsync` 方法。

Dispose

處置所有受管和非受管的資源。

ExecuteBatchGet

從一或多個資料表讀取資料，處理 `MultiTableBatchGet` 中的所有 `BatchGet` 物件。

Note

若要在背景執行此操作，請改用 `ExecuteBatchGetAsync` 方法。

ExecuteBatchWrite

在一或多個資料表中寫入或刪除資料，處理 `MultiTableBatchWrite` 中的所有 `BatchWrite` 物件。

Note

若要在背景執行此操作，請改用 `ExecuteBatchWriteAsync` 方法。

FromDocument

若為 `Document` 執行個體，`FromDocument` 方法會傳回用戶端類別的執行個體。

如果您想使用文件模型類別以及物件持久性模型來執行任何資料操作，這會很有幫助。如需有關由提供的文件模型類別的詳細資訊 AWS SDK for .NET，請參閱 [.NET：文件模型](#)。

假設您有名為 doc 的 Document 物件，且其中包含 Forum 項目的表示法。(若要查看如何建構此物件，請參閱本主題後續內容中的 ToDocument 方法說明。) 您可以使用 FromDocument 從 Document 擷取 Forum 項目，如下列 C# 程式碼範例所示。

Example

```
forum101 = context.FromDocument<Forum>(101);
```

Note

如果您的 Document 物件會實作 IEnumerable 介面，您可以使用 FromDocuments 方法。這允許您逐一查看 Document 中的所有類別執行個體。

FromQuery

以 QueryOperationConfig 物件中定義的查詢參數執行 Query 操作。

Note

若要在背景執行此操作，請改用 FromQueryAsync 方法。

FromScan

以 ScanOperationConfig 物件中定義的掃描參數執行 Scan 操作。

Note

若要在背景執行此操作，請改用 FromScanAsync 方法。

Gettargettable

擷取指定類型的目標資料表。如果您正在撰寫自訂轉換器，將任意資料映射至 DynamoDB 資料表，而且需要判斷哪個資料表與自訂資料類型相關聯，則此功能非常有用。

載入

從資料表擷取項目。這個方法只需要您希望擷取之項目的主索引鍵。

DynamoDB 預設會傳回數值最終一致的項目。如需最終一致性模型的資訊，請參閱 [讀取一致性](#)。

Note

若要在背景執行此操作，請改用 LoadAsync 方法。

Query

根據您提供的查詢參數查詢資料表。

您只能查詢具有複合主索引鍵 (分割區索引鍵和排序索引鍵) 的資料表。查詢時，必須指定分割區索引鍵和套用至排序索引鍵的條件。

假設您有用戶端 Reply 類別映射至 DynamoDB 中的 Reply 資料表。下列 C# 程式碼範例會查詢 Reply 資料表，找出過去 15 天所張貼的論壇主題回覆。Reply 資料表的主索引鍵具有 Id 分割區索引鍵和 ReplyDateTime 排序索引鍵。如需 Reply 資料表的詳細資訊，請參閱 [在 DynamoDB 中建立資料表](#)，以及 [載入程式碼範例的資料](#)。

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);
```

這會傳回 Reply 物件的集合。

Query 方法會傳回「延遲載入」IEnumerable 集合。它一開始只會傳回一頁的結果，然後視需要進行服務呼叫來取得下一頁。若要取得所有相符的項目，您只需要逐一查看 IEnumerable。

如果資料表具有簡易主索引鍵 (分割區索引鍵)，您便不能使用 Query 方法。不過，您可以使用 Load 方法並提供分區索引鍵來擷取項目。

Note

若要在背景執行此操作，請改用 QueryAsync 方法。

Save

將指定的物件儲存至資料表。如果輸入物件中指定的主索引鍵不存在於資料表中，該方法會將新項目新增至資料表中。如果主索引鍵存在，該方法會更新現有項目。

如果您已設定樂觀鎖定，則只有在用戶端和項目的伺服器端版本相符時才能成功更新項目。如需詳細資訊，請參閱 [透過 DynamoDB 使用版本號碼進行最佳鎖定，使用 AWS SDK for .NET 物件持續性模型](#)。

Note

若要在背景執行此操作，請改用 `SaveAsync` 方法。

Scan

執行整個資料表掃描。

您可以指定掃描條件來篩選掃描結果。您可以根據資料表中的任何屬性來評估條件。假設您有用戶端 `Book` 類別映射至 DynamoDB 中的 `ProductCatalog` 資料表。下列 C# 範例會掃描資料表，並且只會傳回價格低於 0 的所有書籍項目。

Example

```
IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
    new ScanCondition("Price", ScanOperator.LessThan, price),
    new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
);
```

`Scan` 方法會傳回「延遲載入」`IEnumerable` 集合。它一開始只會傳回一頁的結果，然後視需要進行服務呼叫來取得下一頁。若要取得所有相符的項目，您只需要逐一查看 `IEnumerable`。

出於效能考量，您應查詢資料表並避免執行資料表掃描。

Note

若要在背景執行此操作，請改用 `ScanAsync` 方法。

ToDocument

從類別執行個體傳回 `Document` 文件模型類別的執行個體。

如果您想使用文件模型類別以及物件持久性模型來執行任何資料操作，這會很有幫助。如需有關由提供的文件模型類別的詳細資訊 AWS SDK for .NET，請參閱[.NET：文件模型](#)。

假設您有用戶端類別映射至範例 Forum 資料表。然後，您可以使用 `DynamoDBContext` 從 Forum 資料表中獲取項目作為 Document 物件，如下列 C# 程式碼範例所示。

Example

```
DynamoDBContext context = new DynamoDBContext(client);

Forum forum101 = context.Load<Forum>(101); // Retrieve a forum by primary key.
Document doc = context.ToDocument<Forum>(forum101);
```

指定 DynamoDBContext 的選用參數

使用物件持久性模型時，您可以指定 `DynamoDBContext` 的下列選用參數。

- **ConsistentRead**：當使用 `Load`、`Query` 或 `Scan` 操作來擷取資料時，您可以新增此選用參數來請求資料的最新數值。
- **IgnoreNullValues**：此參數會通知 `DynamoDBContext` 在 `Save` 操作期間忽略屬性上的 `Null` 數值。如果此參數為 `false` (或未設定)，則會將 `Null` 數值轉譯為刪除特定屬性的指令。
- **SkipVersionCheck**：此參數會通知 `DynamoDBContext` 不要在儲存或刪除項目時比較版本。如需版本控制的詳細資訊，請參閱「[透過 DynamoDB 使用版本號碼進行最佳鎖定，使用 AWS SDK for .NET 物件持續性模型](#)」。
- **TableNamePrefix**：使用特定字串作為所有資料表名稱的字首。如果此參數為 `null` (或未設定)，則不會使用字首。

下列 C# 範例會指定兩個上述選用參數來建立新的 `DynamoDBContext`。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context =
    new DynamoDBContext(client, new DynamoDBContextConfig { ConsistentRead = true,
        SkipVersionCheck = true});
```

`DynamoDBContext` 包含這些選用參數，以及您使用此內容傳送的每個請求。

您可以使用 `DynamoDBContext` 為個別操作指定參數，而不是在 `DynamoDBContext` 層級設定這些參數，如下列 C# 程式碼範例所示。此範例會載入特定的書籍項目。`DynamoDBContext` 的 `Load` 方法會指定上述選用參數。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context = new DynamoDBContext(client);
Book bookItem = context.Load<Book>(productId, new DynamoDBContextConfig{ ConsistentRead
    = true, SkipVersionCheck = true });
```

在此例中，`DynamoDBContext` 只有在傳送 `Get` 請求時才會包含這些參數。

支援的資料類型

物件持久性模型會支援一組基本的 .NET 資料類型、集合和任意資料類型。模型目前支援下列基本資料類型。

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

物件持久性模型也支援 .NET 集合類型。DynamoDBContext 能夠轉換具體的集合類型和簡單的純舊 CLR 物件 (POCO)。

下表摘要說明上述 .NET 類型與 DynamoDB 類型的映射。

.NET 基本類型	DynamoDB 類型
所有數字類型	N (數字類型)
所有字串類型	S (字串類型)
MemoryStream , 字節 []	B (二進位類型)
bool	N (數字類型)。0 代表 false , 1 代表 true。
集合類型	BS (二進位集) 類型、SS (字串集) 類型和 NS (數字集) 類型。
DateTime	S (字串類型)。DateTime 值會以 ISO-8601 格式字串存放。

物件持久性模型也支援任意資料類型。不過，您必須提供轉換器程式碼，才能將複雜類型映射至 DynamoDB 類型。

Note

- 支援空白的二進位值。
- 支援讀取空白字串值。寫入 DynamoDB 時，字串 Set 類型的屬性值會支援空白字串的屬性值。List 或 Map 類型中包含的字串類型的空白字串屬性值和空白字串值會從寫入請求中捨棄。

透過 DynamoDB 使用版本號碼進行最佳鎖定，使用 AWS SDK for .NET 物件持續性模型

物件持久性模型中的樂觀鎖定支援可確保應用程式的項目版本與伺服器端的項目版本相同，然後再更新或刪除項目。假設您要擷取待更新的項目。不過，在您傳回更新之前，某些其他應用程式會更新相同的項目。現在應用程式的項目有過時的副本。如果沒有樂觀鎖定，您執行的任何更新都會覆寫其他應用程式所做的更新。

物件持久性模型的樂觀鎖定功能提供了 `DynamoDBVersion` 標籤，可讓您用來啟用樂觀鎖定。若要使用此功能，請在類別中新增一項屬性，以便儲存版本編號。您可以將 `DynamoDBVersion` 屬性加入該屬性。當您第一次儲存物件時，`DynamoDBContext` 會指派版本編號，並在每次更新項目時遞增值。

只有在用戶端物件版本與伺服器端項目的對應版本編號相符時，您的更新或刪除請求才會成功。如果應用程式有過時的副本，它必須先從伺服器取得最新版本，才能更新或刪除該項目。

下列 C# 程式碼範例會定義 `Book` 類別，該類別具有將其映射至 `ProductCatalog` 資料表的物件持久性屬性。類別中的 `VersionNumber` 屬性裝飾了會儲存版本編號數值的 `DynamoDBVersion` 屬性。

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

您只可將 `DynamoDBVersion` 屬性套用至可為 null 的數字基本類型 (如 `int?`)。

樂觀鎖定對 `DynamoDBContext` 操作的影響如下：

- 對於新項目，`DynamoDBContext` 會指派初始版本編號 0。如果您擷取現有項目，請更新其一或多個屬性並嘗試儲存變更，只有在用戶端的版本編號與伺服器端的相符時，儲存操作才會成功。`DynamoDBContext` 會遞增版本號碼。您不需要設定版本編號。
- `Delete` 方法提供可將主索引鍵值或物件作為參數的多載，如下列 C# 程式碼範例所示。

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);
```

如果提供物件作為參數，則只有在物件版本符合對應的伺服器端項目版本時才能成功刪除。不過，如果提供主索引鍵值作為參數，DynamoDBContext 不知道任何版本編號，其會在沒有檢查是哪個版本的情況下刪除項目。

請注意，物件持久性模型程式碼中的樂觀鎖定內部實作會使用 DynamoDB 中的條件式更新和條件式刪除 API 動作。

停用樂觀鎖定

若要停用樂觀鎖定，您可以使用 SkipVersionCheck 組態屬性。您可以在建立 DynamoDBContext 時設定此屬性。在這種情況下，您使用內容發起的任何請求都會停用樂觀鎖定。如需詳細資訊，請參閱 [指定 DynamoDBContext 的選用參數](#)。

您可以停用特定操作的樂觀鎖定，而不是在內容層級設定屬性，如下列 C# 程式碼範例所示。此範例會使用內容來刪除書籍項目。Delete 方法可設定選用 SkipVersionCheck 屬性為 true，停用版本檢查。

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 });
```

使用 AWS SDK for .NET 物件持續性模型將任意資料與 DynamoDB 對應

除了支援的 .NET 類型之外 (請參閱 [支援的資料類型](#))，您還可以使用應用程式中未直接映射至 Amazon DynamoDB 類型的類型。只要您提供轉換器將資料從任意類型轉換為 DynamoDB 類型，物件持久性模型就支援儲存任意類型的資料，反之亦然。轉換器程式碼會在物件的儲存和載入期間轉換資料。

您可以在用戶端建立任何類型。不過，存放在資料表中的資料是 DynamoDB 類型之一；在查詢和掃描期間，所做的任何資料比較都會與存放在 DynamoDB 中的資料進行比較。

下列 C# 程式碼範例會定義具有 Id、Title、ISBN 以及 Dimension 屬性的 Book 類別。Dimension 屬性具有 DimensionType，可說明 Height、Width 和 Thickness 屬性。範例程式碼提供了轉換器方法 ToEntry 和 FromEntry 來轉換 DimensionType 與 DynamoDB 字串類型之間的資料。例如，在儲存 Book 執行個體時，轉換器會建立一個書籍 Dimension 字串，例如 "8.5x11x.05"。在擷取書籍時，它會將字串轉換為 DimensionType 執行個體。

此範例會將 Book 類型映射為 ProductCatalog 資料表。它儲存了一個範例 Book 執行個體、擷取、更新其維度並再次儲存更新的 Book。

如需測試下列範例的 step-by-step 指示，請參閱 [.NET 程式碼範例](#)。

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;
    }
}
```

使用 AWS SDK for .NET 物件持久性模型的批次操作

批次寫入：放入和刪除多個項目

若要在單一請求中放入或刪除資料表中的多個物件，請執行下列動作：

- 執行 `DynamoDBContext` 的 `createBatchWrite` 方法，然後建立 `BatchWrite` 類別的執行個體。
- 指定您要放入或刪除的項目。
 - 若要放入一或多個項目，請使用 `AddPutItem` 或 `AddPutItems` 方法。
 - 若要刪除一或多個項目，您可以指定項目的主索引鍵或映射至要所要刪除項目的用戶端物件。使用 `AddDeleteItem`、`AddDeleteItems` 和 `AddDeleteKey` 方法來指定要刪除的項目清單。
- 呼叫 `BatchWrite.Execute` 方法來放入和刪除資料表中的所有指定項目。

Note

在使用物件持久性模型時，您可以在單一批次中指定任何數目的操作。但請注意，Amazon DynamoDB 會限制批次中的操作數目，以及批次操作中的批次總大小。如需有關特定限制的更多資訊，請參閱 [BatchWriteItem](#)。若 API 偵測到您的批次寫入請求超過允許的寫入請求次數，或超過允許的 HTTP 承載大小上限，它會將批次分成數個較小的批次。此外，如果批次寫入的回應傳回未處理的項目，API 會自動傳送含有這些未處理項目的另一個批次請求。

假設您定義了一個 C# 類別 `Book`，該類別映射至 DynamoDB 中的 `ProductCatalog` 資料表。下列 C# 程式碼範例使用 `BatchWrite` 物件上傳兩個項目並從 `ProductCatalog` 資料表中刪除一個項目。

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();
```

若要放入或刪除多個資料表中的物件，請執行下列動作：

- 為每個類別建立一個 BatchWrite 類別執行個體，再指定要放入或刪除的項目，如前一節中所述。
- 使用下列其中一種方法來建立 MultiTableBatchWrite 的執行個體：
 - 對在前述步驟中建立的 BatchWrite 物件之一執行 Combine 方法。
 - 透過提供 BatchWrite 物件的清單來建立 MultiTableBatchWrite 類型的執行個體。
 - 執行 DynamoDBContext 的 CreateMultiTableBatchWrite 方法並傳入您的 BatchWrite 物件清單。
- 呼叫 MultiTableBatchWrite 的 Execute 方法，該方法會在各種資料表上執行指定的放入和刪除操作。

假設您定義了會映射至 DynamoDB 中 Forum 和 Thread 資料表的 Forum 和 Thread C# 類別。另外，假設 Thread 類別已啟用版本控制。由於使用批次操作時不支援版本控制，您必須明確停用版本控制，如下列 C# 程式碼範例所示。此範例使用 MultiTableBatchWrite 物件來執行多個資料表更新。

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();
```

如需運作範例，請參閱 [範例：使用 AWS SDK for .NET 物件持續性模型進行 Batch 寫入作業](#)。

Note

DynamoDB 批次 API 會限制批次中的寫入次數，也會限制批次大小。如需詳細資訊，請參閱 [BatchWriteItem](#)。使用 .NET 物件持久性模型 API 時，您可以指定任何次數的操作。不過，如果批次中的操作次數或大小超過限制，則 .NET API 會將批次寫入請求分成較小的批次，並將多個批次寫入請求傳送至 DynamoDB。

批次取得：取得多個項目

若要在單一請求中從資料表擷取多個項目，請執行下列動作：

- 建立 CreateBatchGet 類別的執行個體。
- 指定要擷取的主索引鍵清單。
- 呼叫 Execute 方法。回應會傳回 Results 屬性中的項目。

下列 C# 程式碼範例會從 ProductCatalog 資料表擷取三個項目。結果中項目列出的順序不一定與指定主索引鍵的順序相同。

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];
```

若要擷取多個資料表的物件，請執行下列動作：

- 針對每個類型，建立 CreateBatchGet 類型的執行個體並提供要從每個資料表擷取的主索引鍵值。
- 使用下列其中一種方法來建立 MultiTableBatchGet 類別的執行個體：
 - 對在上述步驟中建立的 BatchGet 物件之一執行 Combine 方法。
 - 透過提供 BatchGet 物件的清單來建立 MultiBatchGet 類型的執行個體。
 - 執行 DynamoDBContext 的 CreateMultiTableBatchGet 方法並傳入您的 BatchGet 物件清單。
- 呼叫 MultiTableBatchGet 的 Execute 方法，該方法會傳回單個 BatchGet 物件中的類型結果。

下列 C# 程式碼範例使用 CreateBatchGet 方法從 Order 和 OrderDetail 資料表中擷取多個項目。

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];
```

範例：使用 AWS SDK for .NET 物件持久性模型的 CRUD 操作

下列 C# 程式碼範例會宣告具有 Id、Title、ISBN 和 Authors 屬性的 Book 類別。範例會使用物件持久性屬性，將這些屬性映射至 Amazon DynamoDB 中的 ProductCatalog 資料表。該範例接著會使用 DynamoDBContext 來說明典型的建立、讀取、更新和刪除 (CRUD) 操作。該範例會建立範例 Book 執行個體，並將其儲存至 ProductCatalog 資料表。它會接著擷取書籍項目並更新其 ISBN 和 Authors 屬性。請注意，更新會取代現有的作者清單。最後，該範例會刪除書籍項目。

如需此範例中所使用 ProductCatalog 資料表的詳細資訊，請參閱 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#)。如需測試下列範例的 step-by-step 指示，請參閱 [.NET 程式碼範例](#)。

Note

下列範例無法搭配 .NET Core 使用，因為它不支援同步方法。如需詳細資訊，請參閱 [適用於 .NET 的 AWS 非同步 API](#)。

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;
    }
}
}
```

範例：使用 AWS SDK for .NET 物件持續性模型進行 Batch 寫入作業

下列 C# 程式碼範例會宣告 Book、Forum、Thread 和 Reply 類別，並使用物件持久性模型屬性將它們映射至 Amazon DynamoDB 資料表。

該範例接著會使用 `DynamoDBContext` 來說明下列批次寫入操作：

- 要從 `ProductCatalog` 資料表放入和刪除書籍項目的 `BatchWrite` 物件。
- 要從 `Forum` 和 `Thread` 資料表放入和刪除項目的 `MultiTableBatchWrite` 物件。

如需此範例中所使用資料表的詳細資訊，請參閱 [在 DynamoDB 中建立資料表](#)，以及載入程式碼範例的 [資料](#)。如需測試下列範例的 step-by-step 指示，請參閱 [.NET 程式碼範例](#)。

Note

下列範例無法搭配 .NET Core 使用，因為它不支援同步方法。如需詳細資訊，請參閱 [適用於 .NET 的 AWS 非同步 API](#)。

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;
    }
}
}
```

範例：使用 AWS SDK for .NET 物件持久性模型在 DynamoDB 中查詢及掃描

本節中的 C# 範例定義下列類別，並將它們映射至 DynamoDB 中的資料表。如需建立在此範例中所使用的資料表之詳細資訊，請參閱 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#)。

- Book 類別會映射至 ProductCatalog 資料表。
- Forum、Thread 與 Reply 類別映射至同名的資料表。

範例接著會使用 DynamoDBContext 來執行下列查詢和掃描操作。

- 依 Id 取得書籍。

ProductCatalog 資料表有 Id 作為其主索引鍵。它沒有排序索引鍵做為其主索引鍵的一部分。因此，您無法查詢資料表。您可以使用項目的 Id 值來取得項目。

- 針對 Reply 資料表執行下列查詢。(Reply 資料表的主索引鍵由 Id 和 ReplyDateTime 屬性組成。ReplyDateTime 是排序索引鍵。因此，您可以查詢此資料表。)
 - 尋找過去 15 天所張貼的論壇主題回覆。
 - 尋找特定日期範圍內所張貼的論壇主題回覆。
- 掃描 ProductCatalog 資料表，尋找價格低於零的書籍。

基於效能考量，您應該使用查詢操作，而不是掃描操作。不過，您有時可能需要掃描資料表。假設發生資料輸入錯誤，而其中一個書籍價格設定為低於 0。此範例會掃描 ProductCategory 資料表，尋找價格低於 0 的書籍項目 (ProductCategory 為 book)。

如需建立可行範例的說明，請參閱 [.NET 程式碼範例](#)。

Note

下列範例無法搭配 .NET Core 使用，因為它不支援同步方法。如需詳細資訊，請參閱 [適用於 .NET 的 AWS 非同步 API](#)。

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;  
    }  
  }  
}
```

執行此開發人員指南中的程式碼範例

AWS 開發套件提供以下語言的 Amazon DynamoDB 廣泛支援：

- [Java](#)

- [瀏覽器中的 JavaScript](#)
- [.NET](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Ruby](#)
- [C++](#)
- [Go](#)
- [Android](#)
- [iOS](#)

若要快速開始使用這些語言，請參閱 [開始使用 DynamoDB 和開發套件 AWS](#)。

此開發人員指南中的程式碼範例使用下列程式設計語言，提供 DynamoDB 操作的更深入說明：

- [Java 程式碼範例](#)
- [.NET 程式碼範例](#)

開始此練習之前，您需要建立 AWS 帳戶、取得存取金鑰與私密金鑰，並在您的電腦上設定 AWS Command Line Interface (AWS CLI)。如需詳細資訊，請參閱 [設定 DynamoDB \(Web 服務\)](#)。

Note

如果您使用可下載版 DynamoDB，您需要使用 AWS CLI 建立資料表與範例資料。您也需要指定每個 AWS CLI 命令的 `--endpoint-url` 參數。如需詳細資訊，請參閱 [設定區域端點](#)。

在 DynamoDB 中建立資料表，以及載入程式碼範例的資料

如需在 DynamoDB 中建立資料表、在範例資料集中載入、查詢資料以及更新資料的基礎知識，請參閱下文。

- [步驟 1：建立資料表](#)
- [第 2 步：使用控制台或將數據寫入表 AWS CLI](#)

- [步驟 3：從資料表讀取資料](#)
- [步驟 4：更新資料表中的資料](#)

Java 程式碼範例

主題

- [Java：設定您的 AWS 登入資料](#)
- [Java：設定 AWS 區域與端點](#)

此開發人員指南包含 Java 程式碼片段及可立即執行的程式。您可以在以下章節中找到這些程式碼範例：

- [使用項目和屬性](#)
- [在 DynamoDB 中使用資料表和資料](#)
- [DynamoDB 中的查詢操作](#)
- [在 DynamoDB 中使用掃描](#)
- [使用次要索引來改善資料存取](#)
- [Java 1.x：DynamoDBMapper](#)
- [DynamoDB Streams 的變更資料擷取](#)

您可以搭配使用 [AWS Toolkit for Eclipse](#) 與 Eclipse 來快速開始使用。除了功能完整的 IDE，您還會取得隨附自動更新的 AWS SDK for Java，以及用於建置 AWS 應用程式之預先設定的範本。

執行 Java 程式碼範例 (使用 Eclipse)

1. 下載並安裝 [Eclipse](#) IDE。
2. 下載並安裝 [AWS Toolkit for Eclipse](#)。
3. 啟動 Eclipse，然後在 Eclipse 選單上，選擇 File (檔案)、New (新建)，再選擇 Other (其他)。
4. 在 Select a wizard (選取協助程式) 中，選擇 AWS，選擇 AWS Java Project (AWS Java 專案)，再選擇 Next (下一步)。
5. 在 Create an AWS Java (建立 AWS Java 專案) 中，執行下列操作：
 - a. 在 Project name (專案名稱) 中，輸入您的專案名稱。

b. 在 Select Account (選取帳戶) 中，從清單選擇您的登入資料描述檔。

如果這是您第一次使用 [AWS Toolkit for Eclipse](#)，請選擇 Configure AWS Accounts (設定 AWS 帳戶) 來設定您的 AWS 登入資料。

6. 選擇 Finish (完成) 建立專案。
7. 從 Eclipse 選單，選擇 File (檔案)、New (新建)，再選擇 Class (類別)。
8. 在 Java Class (Java 類別) 的 Name (名稱) 中，輸入您的類別名稱 (使用與您要執行之程式碼範例相同的名稱)，然後選擇 Finish (完成) 建立類別。
9. 將程式碼範例從文件頁面複製到 Eclipse 編輯器。
10. 若要執行程式碼，請在 Eclipse 選單上選擇 Run (執行)。

Java 開發套件提供可與 DynamoDB 搭配使用的安全執行緒用戶端。根據最佳實務，您的應用程式應該建立一個用戶端，並在執行緒之間重複使用該用戶端。

如需詳細資訊，請參閱 [AWS SDK for Java](#)。

Note

此指南中的程式碼範例適用於最新版的 AWS SDK for Java。

如果您使用 AWS Toolkit for Eclipse，您可以設定 Java 開發套件的自動更新。若要在 Eclipse 中執行此作業，請前往 Preferences (偏好設定)，然後依次選擇 AWS 工具組、AWS SDK for Java、Download new SDKs automatically (自動下載新的開發套件)。

Java：設定您的 AWS 登入資料

適用於 Java 的開發套件需要您在執行時間將 AWS 登入資料提供給您的應用程式。本指南中的程式碼範例假設您正在使用 AWS 登入資料檔案，如《AWS SDK for Java 開發人員指南》中的 [設定您的 AWS 登入資料](#) 所述。

以下是名為 ~/.aws/credentials 的 AWS 登入資料檔案範例，其中波狀符號字元 (~) 代表您的主目錄。

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```


Java : 設定 AWS 區域與端點

依預設，程式碼範例會存取美國西部 (奧勒岡) 區域的 DynamoDB。您可以修改 AmazonDynamoDB 屬性來變更區域。

以下程式碼範例會執行個體化新的 AmazonDynamoDB。

```
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();
```

您可以使用 `withRegion` 方法，對任何可用區域中的 DynamoDB 執行程式碼。如需完整清單，請參閱 Amazon Web Services 一般參考中的 [AWS 區域與端點](#)。

如果您想要使用 DynamoDB 在電腦本機執行程式碼範例，請設定端點如下：

AWS SDK V1

```
AmazonDynamoDB client =
    AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration("http://localhost:8000", "us-west-2"))
    .build();
```

AWS SDK V2

```
DynamoDbClient client = DynamoDbClient.builder()
    .endpointOverride(URI.create("http://localhost:8000"))
    // The region is meaningless for local DynamoDb but required for client builder
    validation
    .region(Region.US_EAST_1)
    .credentialsProvider(StaticCredentialsProvider.create(
        AwsBasicCredentials.create("dummy-key", "dummy-secret")))
    .build();
```

.NET 程式碼範例

主題

- [.NET：設定您的 AWS 登入資料](#)
- [.NET：設定 AWS 區域與端點](#)

此指南包含 .NET 程式碼片段與可立即執行的程式。您可以在以下章節中找到這些程式碼範例：

- [使用項目和屬性](#)
- [在 DynamoDB 中使用資料表和資料](#)
- [DynamoDB 中的查詢操作](#)
- [在 DynamoDB 中使用掃描](#)
- [使用次要索引來改善資料存取](#)
- [.NET：文件模型](#)
- [.NET：物件持久性模型](#)
- [DynamoDB Streams 的變更資料擷取](#)

您可以將 AWS SDK for .NET 與 Toolkit for Visual Studio 搭配起來使用，以此實現快速入門。

執行 .NET 程式碼範例 (使用 Visual Studio)

1. 下載並安裝 [Microsoft Visual Studio](#)。
2. 下載並安裝 [Toolkit for Visual Studio](#)。
3. 啟動 Visual Studio。選擇 File (檔案)、New (新增)、Project (專案)。
4. 在 New Project (新專案) 中，選擇 AWS Empty Project (AWS 空白專案)，然後選擇 OK (確定)。
5. 在 AWS Access Credentials (AWS 存取登入資料) 中，選擇 Use existing profile (使用現有的描述檔)，然後從清單中選擇您的登入資料描述檔，再選擇 OK (確定)。

如果這是您第一次使用 Toolkit for Visual Studio，請選擇 Use a new profile (使用新的描述檔) 來設定您的 AWS 登入資料。

6. 在您的 Visual Studio 專案中，選擇您程式之來源碼 (Program.cs) 的標籤。將程式碼範例從文件頁面複製到 Visual Studio 編輯器，以取代您在編輯器中所看到的任何其他程式碼。
7. 如果看到 The type or namespace name...could not be found (找不到類型或命名空間名稱...) 格式的錯誤訊息，您需要為 DynamoDB 安裝 AWS 開發套件組件，如下所示：
 - a. 在方案總管中，開啟您專案的內容 (右鍵) 選單，然後選擇 Manage NuGet Packages (管理 NuGet 套件)。

- b. 在 NuGet 套件管理員中，選擇 Browse (瀏覽)。
 - c. 在搜尋方塊中輸入 **AWSSDK.DynamoDBv2**，並等候搜尋完成。
 - d. 選擇 AWSSDK.DynamoDBv2，然後選擇 Install (安裝)。
 - e. 安裝完成時，選擇 Program.cs 標籤返回您的程式。
8. 若要執行程式碼，請在 Visual Studio 工具列中選擇 Start (啟動)。

AWS SDK for .NET 提供可與 DynamoDB 搭配使用的安全執行緒用戶端。根據最佳實務，您的應用程式應該建立一個用戶端，並在執行緒之間重複使用該用戶端。

如需詳細資訊，請參閱[適用於 .NET 的 AWS 開發套件](#)。

Note

此指南中的程式碼範例適用於最新版的 AWS SDK for .NET。

.NET：設定您的 AWS 登入資料

AWS SDK for .NET 需要您在執行時間將 AWS 登入資料提供給您的應用程式。本指南中的程式碼範例假設您正在使用開發套件存放區管理您的 AWS 登入資料檔案，如《AWS SDK for .NET 開發人員指南》中[使用開發套件存放區](#)所述。

Toolkit for Visual Studio 支援來自任意數目帳戶的多組登入資料。每個集合都稱為描述檔。Visual Studio 會將項目新增至專案的 App.config 檔案，讓您的應用程式可以在執行時間尋找 AWS 登入資料。

下列範例顯示預設 App.config 檔案，當您使用 Toolkit for Visual Studio 建立新的專案時，就會生成此檔案。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="AWSProfileName" value="default"/>
    <add key="AWSRegion" value="us-west-2" />
  </appSettings>
</configuration>
```

在執行時間，此程式會使用 AWS 登入資料的 default 集合，如 AWSProfileName 項目所指定。AWS 登入資料本身會以加密形式保留在開發套件存放區中。Toolkit for Visual Studio 提供圖形化

使用者介面來管理所有來自於 Visual Studio 的登入資料。如需詳細資訊，請參閱《AWS Toolkit for Visual Studio 使用者指南》中的[指定登入資料](#)。

Note

依預設，程式碼範例會存取美國西部 (奧勒岡) 區域的 DynamoDB。您可以修改 App.config 檔案中的 AWSRegion 項目來變更區域。您可以將 AWSRegion 設定為 DynamoDB 可用的任何區域。如需完整清單，請參閱 Amazon Web Services 一般參考 中的 [AWS 區域與端點](#)。

.NET：設定 AWS 區域與端點

依預設，程式碼範例會存取美國西部 (奧勒岡) 區域的 DynamoDB。您可以修改 AWSRegion 檔案中的 App.config 項目來變更區域。或者，您亦可以修改 AmazonDynamoDBClient 屬性來變更區域。

以下程式碼範例會執行個體化新的 AmazonDynamoDBClient。這會修改用戶端，讓程式碼對不同區域中的 DynamoDB 執行。

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// This client will access the US East 1 region.
clientConfig.RegionEndpoint = RegionEndpoint.USEast1;
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

如需區域的完整清單，請參閱 Amazon Web Services 一般參考 中的 [AWS 區域與端點](#)。

如果您想要使用 DynamoDB 在電腦本機執行程式碼範例，請設定端點如下：

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// Set the endpoint URL
clientConfig.ServiceURL = "http://localhost:8000";
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

使用 Python 和肉毒桿菌 3 對 Amazon DynamoDB 進行編程

本指南為想要將 Amazon DynamoDB 與 Python 搭配使用的程式設計師提供了一個方向。瞭解不同的抽象層、組態管理、錯誤處理、控制重試原則、管理保持活動狀態等。

主題

- [關於博托](#)

- [使用博托文檔](#)
- [了解客戶端和資源抽象層](#)
- [使用表格資源批次寫入器](#)
- [探索客戶端和資源層的其他代碼示例](#)
- [了解用戶端和資源物件如何與工作階段和執行緒互動](#)
- [自定義 Config 對象](#)
- [錯誤處理](#)
- [日誌](#)
- [事件掛鉤](#)
- [分頁和分頁器](#)
- [等待程式](#)

關於博托

您可以使用適用於 Python 的官方 AWS 開發套件 (通常稱為 Boto3)，從 Python 存取 DynamoDB。Boto (發音為 boh-toh) 這個名字來自 Amazon 河原產的淡水海豚。Boto3 圖書館是圖書館的第三個主要版本，於 2015 年首次發布。Boto3 程式庫相當龐大，因為它支援所有 AWS 服務，而不僅僅是 DynamoDB。此方向僅針對與 DynamoDB 相關的肉毒桿 3 部分。

Boto 由 AWS 作為託管的開源項目進行維護和發布。GitHub [它被分成兩個軟件包：肉毒核和 Boto3。](#)

- 肉毒核提供了低級別的功能。在 Botocore 中，您將找到客戶端，會話，憑據，配置和異常類。
- 肉毒桿菌 3 建立在肉毒核之上。它提供了更高級別，更多 Pythonic 界面。具體而言，它將 DynamoDB 表公開為資源，並提供更簡單、更優雅的介面，相較於較低層級的服務導向用戶端介面。

由於這些專案是在託管 GitHub，因此您可以檢視原始程式碼、追蹤未解決的問題，或提交您自己的問題。

使用博托文檔

使用下列資源開始使用 Boto 文件：

- [從快速入門部分開始](#)，該部分為軟件包安裝提供了一個堅實的起點。如果尚未安裝 Boto3，請到那裡獲取有關安裝的說明 (Boto3 通常在 AWS 服務中自動提供，例如)。AWS Lambda

- 之後，請專注於文件的 [DynamoDB](#) 指南。它說明如何執行基本 DynamoDB 活動：建立和刪除資料表、操作項目、執行批次作業、執行查詢以及執行掃描。其示例使用資源接口。當你看到 `boto3.resource('dynamodb')` 明你正在使用更高級別的資源接口。
- 完成指南後，您可以檢閱 [DynamoDB](#) 參考資料。此登陸頁面提供您可用的類別和方法的詳盡清單。在頂部，你會看到 `DynamoDB.Client` 類。這提供了對所有控制面和數據平面操作的低級訪問。在底部，看看 `DynamoDB.ServiceResource` 班級。這是更高級別的 Pythonic 接口。有了它，您可以建立表格、跨表執行批次作業，或取得表格特定動作的 `DynamoDB.ServiceResource.Table` 執行個體。

了解客戶端和資源抽象層

您將使用的兩個介面是用戶端介面和資源介面。

- 低階用戶端介面提供與基礎服務 API 的 1 對 1 對 1 對應。DynamoDB 提供的每個 API 都可透過用戶端取得。這意味著客戶端界面可以提供完整的功能，但使用起來通常更加冗長和複雜。
- 較高層級的資源介面不會提供基礎服務 API 的 1 對 1 對 1 對應。但是，它提供了使您更方便地訪問該服務的方法，例如 `batch_writer`。

以下是使用用戶端介面插入項目的範例。請注意如何將所有值作為映射傳遞，其鍵指示其類型（'S' 表示字符串，'N' 表示數字）及其值作為字符串傳遞。這就是所 DynamoDB JSON 格式。

```
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 ...
    }
)
```

這是使用資源接口的相同 `PutItem` 操作。數據類型是隱含的：

```
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 ...
    }
)
```

如果需要，您可以使用 boto3 提供的和TypeDeserializer類別，在一般 JSON TypeSerializer 和 DynamoDB JSON 之間進行轉換：

```
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()
    }
```

以下是如何使用客戶端界面執行查詢。它將查詢表示為 JSON 結構。它使用需要變量替換來處理任何潛在的關鍵KeyConditionExpression字衝突的字符串：

```
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',
    }
)

```

使用資源接口的相同查詢操作可以縮短和簡化：

```

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')
)

```

作為最後一個例子，假設你想要獲得一個表的近似大小（這是保存在表上的元數據，大約每 6 小時更新一次）。使用客戶端界面，您可以執行 `describe_table()` 操作並從返回的 JSON 結構中提取答案：

```

import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.describe_table(TableName='YourTableName')
size = response['Table']['TableSizeBytes']

```

透過資源介面，表格會隱含地執行描述作業，並直接將資料顯示為屬性：

```

import boto3

```



```
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')
size = table.table_size_bytes
```

Note

考慮是否要使用用戶端或資源介面進行開發時，請注意，不會依據資源[說明文件將新功能新增至資源](#)介面：「AWS Python SDK 小組不打算將新功能新增至 boto3 中的資源介面。現有介面將在 boto3 的生命週期中繼續運作。客戶可以透過用戶端介面取得更新的服務功能。」

使用表格資源批次寫入器

只有更高層級的表格資源才能使用的便利之一是 `batch_writer`。DynamoDB 支援批次寫入操作，允許在一個網路請求中執行多達 25 個置入或刪除操作。像這樣的批處理通過最大限度地減少網路往返提高效率。

使用低階用戶端程式庫時，您可以使用此 `client.batch_write_item()` 作業執行批次。您必須手動將工作分割成 25 個批次。每次操作之後，您還必須請求接收未處理項目的列表（某些寫操作可能成功，而其他可能會失敗）。然後，您必須再次將這些未處理的項目傳遞到稍後的 `batch_write_item()` 操作中。沒有大量的樣板代碼。

該 [表格 `.batch_writer`](#) 方法創建一個前後關聯管理器，用於在批處理中寫入對象。它提供了一個界面，看起來好像你一次寫一個項目，但在內部它正在緩衝並批量發送項目。它還隱含處理未處理的項目重試。

```
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)
```

探索客戶端和資源層的其他代碼示例

您也可以參考下列程式碼範例儲存庫，以探索使用用戶端和資源的各種函式的使用方式：

- [官方的 AWS 單一動作程式碼範例。](#)
- [官方 AWS 面向場景的代碼示例。](#)
- [社群維護的單一動作程式碼範例。](#)

了解用戶端和資源物件如何與工作階段和執行緒互動

Resource 對象不是線程安全的，不應該跨線程或進程共享。有關更多詳細信息，[請參閱資源指南](#)。

相比之下，Client 物件通常是執行緒安全的，除了特定的進階功能之外。如需詳細資訊，[請參閱用戶端指南](#)。

會話對象不是線程安全的。因此，每次在多線程環境中創建客戶端或資源時，都應首先創建一個新的會話，然後從會話中創建客戶端或資源。如需詳細資訊，[請參閱工作階段指南](#)。

當您呼叫時 `boto3.resource()`，您隱含地使用預設工作階段。這對於編寫單線程代碼很方便。在編寫多線程代碼時，您需要首先為每個線程構建一個新的會話，然後從該會話中檢索資源：

```
# Explicitly create a new Session for this thread
session = boto3.Session()
dynamodb = session.resource('dynamodb')
```

自定義 Config 對象

當構造一個客戶端或資源對象，你可以傳遞可選的命名參數來自定義行為。名為的參數可 `config` 解鎖各種功能。這是一個實例 `botocore.client.Config`，[Config 的參考文檔](#) 顯示了它公開供您控制的所有內容。[配置指南](#) 提供了一個很好的概述。

Note

您可以在工作階段層級、AWS 組態檔案內或環境變數修改許多這些行為設定。

Config 逾時

自定義配置的一種用途是調整網絡行為：

- `connect_timeout` (浮點數或整數) — 嘗試建立連線時擲回逾時例外狀況的秒數。預設值為 60 秒。

- `read_timeout` (浮點數或整數) — 嘗試從連線讀取時擲回逾時例外狀況的秒數。預設值為 60 秒。

對於 DynamoDB 而言，60 秒的逾時時間過長。這意味著短暫的網絡故障將導致客戶端一分鐘的延遲，然後才能再試一次。下面的代碼將超時縮短到第二個：

```
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)
```

如需有關逾時的詳細討論，請參閱[針對延遲感知 DynamoDB 應用程式調整 AWS Java SDK HTTP 要求設定](#)。請注意，Java SDK 具有比 Python 更多的超時配置。

保持活著的 Config

如果您使用的是 botocore 1.27.84 或更高版本，您還可以控制 TCP 保持活動狀態：

- `tcp_keepalive` (布爾) - 如果設置為 (默認為)，則在創建新連接時啟用 TCP 保持活動通訊端選項。True False 這只能從肉毒核 1.27.84 開始使用。

將 TCP 保持活動狀態設置為 True 可以減少平均延遲。以下是示例代碼，當您擁有正確的 botocore 版本時，有條件地將 TCP 保持活動設置為 true：

```
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 保持活動與 HTTP 保持活動狀態不同。使用 TCP Keep-Alive 時，基礎作業系統會透過通訊端連線傳送小型封包，以保持連線作用中，並立即偵測任何中斷。使用 HTTP 保持活動，基於底層套接字構建的 Web 連接被重用。HTTP 保持活動始終與肉毒 3 啟用。

閒置連接可以保持活躍的時間是有限的。如果您有閒置連接，但希望下一個請求使用已建立的連接，請考慮發送定期請求（例如每分鐘）。

重試 Config

該配置還接受一個名為 `retries` 的字典，您可以在其中指定所需的重試行為。當 SDK 收到錯誤且錯誤為暫時性類型時，會在 SDK 中重試。如果在內部重試錯誤（並且重試最終會產生成功的響應），則從調用代碼的角度看不到錯誤，只是稍微提高了延遲。以下是您可以指定的值：

- `max_tries` — 整數，代表將在單一要求上進行的重試嘗試次數上限。例如，將此值設為 2 將導致在初始請求之後最多重試兩次請求。將此值設為 0 將導致在初始請求之後不會嘗試重試。
- `total_max_tries` — 整數，代表將在單一要求上進行的總嘗試次數上限。這包括初始請求，因此值 1 表示不會重試任何請求。如果 `total_max_attempts` 和兩者 `max_attempts` 都提供，`total_max_attempts` 優先。 `total_max_attempts` 優先選擇，`max_attempts` 因為它映射到 `AWS_MAX_ATTEMPTS` 環境變量和 `max_attempts` 配置文件值。
- `mode` — 應使用代表重試模式類型的字符串。有效的值如下：
 - 舊版 — 預設模式。在第一次重試時等待 50ms，然後使用基本因子為 2 的指數輪詢。對於 DynamoDB，它最多可執行總共 10 次嘗試次數（除非使用上述內容覆寫）。

Note

使用指數輪詢時，最後一次嘗試將等待將近 13 秒。

- 標準 — 命名為標準，因為它與其他 AWS SDK 更一致。等待從 0 毫秒到 1000 毫秒的隨機時間進行第一次重試。如果需要另一次重試，它會從 0ms 到 1,000ms 選擇另一個隨機時間，並將其乘以 2。如果需要額外的重試，它會執行相同的隨機選擇乘以 4，依此類推。每次等待時間上限為 20 秒。此模式會在偵測到的失敗情況下執行重試次數超過 legacy 模式。對於 DynamoDB，它最多可執行總共 3 次嘗試次數（除非使用上述內容覆寫）。

- **adaptive**—一種實驗重試模式，其中包括標準模式的所有功能，但增加了自動客戶端節流。使用自適應速率限制，SDK 可以減慢傳送要求的速率，以便更好地容納 AWS 服務的容量。這是一種臨時模式，其行為可能會改變。

您可以在[重試指南](#)以及 [SDK 參考的「重試行為」主題](#)中找到這些重試模式的擴充定義。

以下是一個明確使用 legacy 重試政策的示例，最多共有 3 個請求（2 次重試）：

```
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)
```

由於 DynamoDB 是高可用性、低延遲的系統，因此您可能想要比內建重試原則允許的速度更積極。您可以通過將最大嘗試次數設置為 0，自己捕獲異常並根據自己的代碼重試，而不是依賴 boto3 進行隱式重試來實現自己的重試策略。

如果您管理自己的重試原則，則需要區分節流和錯誤：

- 節流 (以 `ProvisionedThroughputExceededException` 或表示 `ThrottlingException`) 表示運作良好的服務，通知您已超過 DynamoDB 資料表或分區上的讀取或寫入容量。每隔一毫秒，就會提供更多的讀取或寫入容量，因此您可以快速重試 (例如每 50 毫秒)，以嘗試存取該新發佈的容量。使用節流時，您不需要特別需要指數輪詢，因為節流是輕量級的 DynamoDB 傳回，而且不會向您收取每個請求的費用。指數輪詢會將較長的延遲指派給已經等待最長時間的用戶端執行緒，從統計上將 p50 和 p99 向外延伸。
- 錯誤 (由 `InternalServerError` 或 a 表示 `ServiceUnavailable`，等等) 表示服務發生暫時性問題。這可以是整個表格，也可能只是您正在讀取或寫入的分區。如果發生錯誤，您可以在重試 (例如 250 毫秒或 500 毫秒) 之前暫停更長的時間，並使用抖動來錯開重試。

最大池連接的 Config

最後，配置允許您控制連接池的大小：

- 連線 (int) — 連線集區中要保留的最大連線數目。如果未設定此值，則會使用預設值 10。

此選項控制要保持集區以供重複使用的 HTTP 連線數目上限。每個工作階段會保留不同的池。如果您預期超過 10 個執行緒會針對用戶端或建立在相同工作階段的資源，您應該考慮提高這個問題，因此執行緒不必使用集區連線等待其他執行緒。

```
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
```

錯誤處理

AWS 服務異常並非所有在 Boto3 中靜態定義。這是因為 AWS 服務的錯誤和例外情況差異很大，並且可能會發生變化。Boto3 會將所有服務例外狀況包裝為 `a, ClientError` 並將詳細資訊公開為結構化 JSON。例如，錯誤回應的結構可能如下：

```
{
  '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
  }
}
```

```
}  
}
```

下列程式碼會擷取任何 `ClientError` 例外狀況，並查看 `Code` 中的字串值，以決定 `Error` 要採取的動作：

```
import botocore  
import boto3  
  
dynamodb = boto3.client('dynamodb')  
  
try:  
    response = dynamodb.put_item(...)  
  
except botocore.exceptions.ClientError as err:  
    print('Error Code: {}'.format(err.response['Error']['Code']))  
    print('Error Message: {}'.format(err.response['Error']['Message']))  
    print('Http Code: {}'.format(err.response['ResponseMetadata']['HTTPStatusCode']))  
    print('Request ID: {}'.format(err.response['ResponseMetadata']['RequestId']))  
  
    if err.response['Error']['Code'] in ('ProvisionedThroughputExceededException',  
    'ThrottlingException'):  
        print("Received a throttle")  
    elif err.response['Error']['Code'] == 'InternalServerError':  
        print("Received a server error")  
    else:  
        raise err
```

某些（但不是全部）異常代碼已被實現為頂級類。您可以選擇直接處理這些問題。使用用戶端介面時，會在用戶端上動態填入這些例外狀況，並且您可以使用用戶端執行個體來 `catch` 取這些例外狀況，如下所示：

```
except ddb_client.exceptions.ProvisionedThroughputExceededException:
```

使用資源接口時，您必須使用從資源遍歷 `.meta.client` 到底層客戶端以訪問異常，如下所示：

```
except ddb_resource.meta.client.exceptions.ProvisionedThroughputExceededException:
```

若要複查具體化例外類型清單，您可以動態產生清單：

```
ddb = boto3.client("dynamodb")
```

```
print([e for e in dir(ddb.exceptions) if e.endswith('Exception') or
      e.endswith('Error')])
```

使用條件運算式執行寫入作業時，您可以要求如果運算式失敗，應在錯誤回應中傳回項目的值。

```
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'])
```

有關錯誤處理和異常的進一步閱讀：

- 在 [boto3 指南錯誤處理有關於錯誤處理](#) 技術的更多信息。
- 有 [關於程式設計錯誤的 DynamoDB 開發人員指南一節](#) 會列出您可能遇到的錯誤。
- [API 參考資料中的常見錯誤區段](#)。
- 每個 API 操作的文檔列出了調用可能產生的錯誤（例如 [BatchWriteItem](#)）。

日誌

boto3 庫與 Python 的內置日誌記錄模塊集成在一起，用於跟踪會話期間發生的情況。若要控制記錄層級，您可以設定記錄模組：

```
import logging

logging.basicConfig(level=logging.INFO)
```

這將根記錄器配置為記錄 INFO 和高於級別的消息。記錄比級別不太嚴重的消息將被忽略。記錄層級包括 DEBUG、INFO、WARNING、ERROR、和 CRITICAL。預設值為 WARNING。

在 boto3 記錄器是分層的。該庫使用了幾個不同的記錄器，每個記錄器都對應於庫的不同部分。您可以單獨控制每個行為：

- boto3：boto3 模塊的主記錄器。
- 肉毒核心：肉毒核軟件包的主記錄器。

- `botocore.auth`：用於記錄 AWS 簽名創建請求。
- `botocore.credentials`：用於記錄憑據獲取和刷新的過程。
- `botocore.endpoint`：用於在通過網絡發送之前記錄請求創建。
- `botocore.hook`：用於在庫中觸發的日誌事件。
- `botocore.loaders`：用於加載服務模式的 AWS 部分時進行記錄。
- `botocore.parsers`：用於在解析之前記錄服 AWS 務響應。
- `botocore.retryhandler`：用於記錄 AWS 服務請求重試（傳統模式）的處理。
- `botocore.retries.standard`：用於記錄 AWS 服務請求重試的處理（標準或自適應模式）。
- `botocore.utils`：用於在圖書館記錄其他活動。
- `botocore.waiter`：用於記錄服務員的功能，該服務輪詢 AWS 服務，直到達到特定狀態。

其他程式庫也會記錄。在內部，`boto3` 使用第三方網址 `urllib3` 進行 HTTP 連接處理。當延遲很重要時，您可以通過查看 `urllib3` 何時建立新連接或關閉閒置連接來確保池正常利用。

- `urllib3`. 連線池：用於記錄連線集區處理事件。

下列程式碼片段會 INFO 透過端點和連線集區活動的 DEBUG 記錄設定大部分記錄：

```
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)
```

事件掛鉤

肉毒核在其執行的各個部分期間發出事件。您可以為這些事件註冊處理程序，以便無論何時發射事件，您的處理程序都會被調用。這使您可以擴展 `botocore` 的行為，而無需修改內部。

例如，假設您想要追蹤每次在應用程式中的任何 DynamoDB 表上呼叫 `PutItem` 作業時。您可以在 `'provide-client-params.dynamodb.PutItem'` 事件上註冊以 `catch` 和記錄每次在關聯的會話上調用 `PutItem` 操作時。範例如下：

```
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 ...
    }
)
```

在處理程序中，您甚至可以通過編程方式操作參數以更改行為：

```
params['TableName'] = "NewTableName"
```

如需有關事件的詳細資訊，請參閱有關事件的 [botocore 文件和事件](#) 的 [boto3](#) 文件。

分頁和分頁器

某些要求（例如「查詢和掃描」）會限制單一要求傳回的資料大小，並要求您重複提出要求以提取後續頁面。

您可以使用參數控制每個頁面要讀取的最大項目 `limit` 數。例如，如果您想要最後 10 個項目，則可以使用 `limit` 取最後 10 個項目。請注意，限制是在套用任何篩選之前，應從表格中讀取多少。沒有辦法在過濾後指定你想要的 10；你只能控制預先過濾的計數，並在你實際檢索到 10 時檢查客戶端。無論限制如何，每個響應始終具有 1 MB 的最大大小。

如果響應包含 `aLastEvaluatedKey`，則表示響應因為達到計數或大小限制而結束。該鍵是為響應評估的最後一個鍵。您可以檢索它 `LastEvaluatedKey` 並將其傳遞給後續調用，以 `ExclusiveStartKey` 便從該起點讀取下一個塊。如果沒有 `LastEvaluatedKey` 返回，則表示沒有更多符合「查詢」或「掃描」的項目。

下面是一個簡單的例子（使用 `Resource` 接口，但客戶端接口具有相同的模式），每頁最多讀取 100 個項目並循環，直到讀取所有項目。

```
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']
```

為了方便起見，`boto3` 可以使用分頁器為您做到這一點。但是，它僅適用於客戶端界面。以下是重寫以使用分頁器的代碼：

```
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)
```

如需詳細資訊，請參閱[分頁器指南](#)和[動態分頁器的 API 參考](#)。

Note

分頁器也有自己的組態設定 `MaxItems`，名為 `StartingToken`、`和` `PageSize` 若要使用 DynamoDB 進行分頁，您應該忽略這些設定。

等待程式

服務員提供了在繼續之前等待某些事情完成的能力。目前，它們僅支持等待創建或刪除表。在後台，服務員操作每 20 秒為您檢查一次，最多 25 次。您可以自己做，但是在編寫自動化時使用服務員很優雅。

此代碼顯示瞭如何等待特定表已創建：

```
# 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'])
```

有關更多信息，請參閱[服務員指南](#)和[服務員參考](#)。

使用 Amazon DynamoDB 進行編程 JavaScript

本指南為想要使用 Amazon DynamoDB 的程式設計人員提供了一個方向。JavaScript 瞭解可用的抽象層 AWS SDK for JavaScript、設定連線、處理錯誤、定義重試原則、管理保持活動狀態等。

主題

- [關於 AWS SDK for JavaScript](#)
- [使用 AWS SDK for JavaScript V3](#)
- [存取 JavaScript 文件](#)
- [抽象層](#)
- [使用馬歇爾實用程序函數](#)
- [讀取項目](#)
- [條件式寫入](#)
- [分頁](#)
- [指定組態](#)
- [等待程式](#)
- [錯誤處理](#)
- [日誌](#)
- [考量事項](#)

關於 AWS SDK for JavaScript

可讓您存取使 AWS 服務用瀏覽器指令碼或 Node.js。AWS SDK for JavaScript 本文件著重於 SDK (V3) 的最新版本。AWS SDK for JavaScript V3 作 AWS 為 [託管的開源項目進行維護](#) GitHub。問題和功能請求是公開的，您可以在存放庫的問題頁面上 GitHub 存取它們。

JavaScript V2 類似於 V3，但包含語法差異。V3 更加模塊化，可以更輕鬆地提供較小的依賴關係，並具有一流的 TypeScript 支持。我們建議使用最新版本的 SDK。

使用 AWS SDK for JavaScript V3

您可以使用節點 Package 管理員將 SDK 新增至 Node.js 應用程式。下列範例說明如何新增最常用的 SDK 套件，以便與 DynamoDB 搭配使用。

- `npm install @aws-sdk/client-dynamodb`

- `npm install @aws-sdk/lib-dynamodb`
- `npm install @aws-sdk/util-dynamodb`

安裝套件會將參考新增至 `package.json` 專案檔案的相依性區段。您可以選擇使用較新的 ECMAScript 模塊語法。如需這兩種方法的詳細資訊，請參閱 [< 考量 >](#) 一節。

存取 JavaScript 文件

使用下列資源開始使用 JavaScript 文件：

- 存取 [開發人員指南](#) 以取得核心 JavaScript 文件。安裝說明位於「設定」區段中。
- 存取 [API 參考](#) 文件以探索所有可用的類別和方法。
- 除了 DynamoDB 之外，適用的 SDK JavaScript 支援許多 AWS 服務 其他功能。使用下列程序來尋找 DynamoDB 的特定 API 涵蓋範圍：
 1. 在服務中，選擇 DynamoDB 和程式庫。這會記錄低階用戶端。
 2. 選擇庫動態。這會記錄高階用戶端。兩個用戶端代表您可以選擇使用的兩個不同的抽象層。有關抽象層的更多信息，請參見下面的部分。

抽象層

JavaScript V3 的 SDK 具有低級客戶端 (`DynamoDBClient`) 和高級客戶端 (`DynamoDBDocumentClient`)。

主題

- [低階用戶端 \(\) `DynamoDBClient`](#)
- [高階用戶端 \(`DynamoDBDocumentClient`\)](#)

低階用戶端 () `DynamoDBClient`

低級客戶端在基礎有線協議上不提供額外的抽象。它可讓您完全控制通訊的所有層面，但由於沒有抽象化，因此您必須執行諸如使用 DynamoDB JSON 格式提供項目定義之類的操作。

如下面的例子所示，用這種格式的數據類型必須明確說明。S 表示字串值，N 表示數值。配線上的號碼永遠做為標記為數字類型的字串傳送，以確保不會損失精確度。低階 API 呼叫具有命名模式，例如 `PutItemCommand` 和 `GetItemCommand`。

下列範例是使用使用 DynamoDB JSON Item 定義的低階用戶端：

```
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();
```

高階用戶端 (DynamoDBDocumentClient)

高階 DynamoDB 文件用戶端提供內建的便利功能，例如無需手動封送資料，並允許使用標準物件直接讀取和寫入。JavaScript 的[文件提 lib-dynamodb](#)供優點清單。

要實例化 `DynamoDBDocumentClient`，構建一個低級別，`DynamoDBClient` 然後用。

`DynamoDBDocumentClient` 這兩個套件之間的函數命名慣例略有不同。例如，低級別的使用，`PutItemCommand` 而高級使用 `PutCommand`。不同的名稱允許兩組函數在相同的上下文中共存，允許您在同一個腳本中混合兩個函數。

```
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();
```

當您使用 API 操作 (例如、或) 讀取項目時 `GetItem` , `Query` 使用模式是一致的 `Scan`。

使用馬歇爾實用程序函數

您可以使用低級客戶端和馬歇爾或解開自己的數據類型。公用程式套件公用 [程式 `Dynamodb`](#) 具有接受 JSON 並產生 DynamoDB JSON 的 `marshall()` 公用程式函數 , 以及可反向執行相反作業的 `unmarshall()` 函數。下列範例會使用具有呼叫所處理之資料封送的低階用戶端。 `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();
```

讀取項目

若要從 DynamoDB 讀取單一項目，請使用 `GetItem` API 作業。與 `PutItem` 命令類似，您可以選擇使用低階用戶端或高階 `Document` 用戶端。下面的例子演示了如何使用高級文檔客戶端檢索項目。

```
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();
```

使用 Query API 作業讀取多個項目。您可以使用低階用戶端或文件用戶端。以下範例使用高階文件用戶端。

```
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();
```

條件式寫入

DynamoDB 寫入作業可以指定邏輯條件運算式，必須評估為 true 才能繼續寫入。如果條件未評估為 true，寫入作業會產生例外狀況。條件運算式可以檢查項目是否已存在，或其屬性是否符合某些限制。

```
ConditionExpression = "version = :ver AND size(VideoClip) < :maxsize"
```

當條件運算式失敗時，您可以使用 `ReturnValuesOnConditionCheckFailure` 來要求錯誤回應包含不符合條件的項目，以推斷出問題所在。如需詳細資訊，請參閱 [使用 Amazon DynamoDB 在高並行案例中處理條件式寫入錯誤](#)。

```
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;
  }
}
```

其他程式碼範例顯示 JavaScript SDK V3 使用方式的其他層面，請參閱 [JavaScript SDK V3 文件和 DynamoDB- GitHub 軟體開發套件](#) 範例範例的範例儲存庫下。

分頁

主題

- [使用 `paginateScan` 便利的方法](#)

讀取要求 (例如 `Scan` 或可能 `Query` 會傳回資料集中的多個項目)。如果您 `Query` 使用 `Limit` 參數執行 `Scan` 或，則一旦系統讀取了許多項目，就會傳送部分回應，而且您需要分頁以擷取其他項目。

系統每個請求最多只能讀取 1 MB 的資料。如果您包含 `Filter` 運算式，系統仍會從磁碟讀取最多 MB 的資料，但會傳回符合篩選器的 MB 項目。篩選作業可能會針對頁面傳回 0 個項目，但在搜尋用盡之前仍需要進一步的分頁。

您應該在回應LastEvaluatedKey中尋找，並在後續要求中使用它作為ExclusiveStartKey參數，以繼續擷取資料。如下列範例中所述，這可做為書籤使用。

Note

樣本ExclusiveStartKey在第一次迭代時傳遞 nulllastEvaluatedKey，這是允許的。

使用以下示例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);
});
```

使用paginateScan便利的方法

SDK 提供了所謂的便利方法paginateQuery，paginateScan並為您完成此工作，並在幕後發出重複的請求。使用標準Limit參數指定每個請求要讀取的最大項目數。

```
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

除非資料表很小，否則不建議定期執行完整表格掃描。

指定組態

主題

- [Config 逾時](#)
- [保持活著的 Config](#)
- [重試 Config](#)

設定時 `DynamoDBClient`，您可以將組態物件傳遞給建構函式，以指定各種組態覆寫。例如，如果呼叫內容或要使用的端點 URL 尚未知道該區域，您可以指定要連接的區域。如果您想要針對開發目的將 DynamoDB 本機執行個體作為目標，此功能非常有用。

```
const client = new DynamoDBClient({
```

```
    region: "eu-west-1",
    endpoint: "http://localhost:8000",
  });
```

Config 逾時

DynamoDB 會使用 HTTPS 進行用戶端與伺服器的通訊。您可以透過提供物件來控制 HTTP 層的某些方面。例如，您可以調整金鑰逾時值 `connectionTimeout` 和 `requestTimeout`。這 `connectionTimeout` 是用戶端在放棄之前嘗試建立連線時等待的最長持續時間 (以毫秒為單位)。

定義 `requestTimeout` 義了多長時間客戶端將等待一個請求已發送後，也以毫秒為單位的響應。兩者的默認值都是零，這意味著超時被禁用，並且如果響應沒有到達，客戶端將等待多長時間沒有限制。您應該將超時設置為合理的值，以便在發生網絡問題時，請求將錯誤並啟動新請求。例如：

```
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

所提供的範例使用「[鐵匠鋪](#)」匯入。Smithy 是一種用於定義開放原始碼和維護的服務和 SDK 的語言。AWS

除了配置超時值之外，您還可以設置最大套接字數，這樣可以增加每個來源的並行連接數量。開發人員指南包含有關[配置 `maxSockets` 參數的詳細信息](#)。

保持活著的 Config

使用 HTTPS 時，第一個請求始終需要一些 back-and-forth 通信來建立安全連接。HTTP Keep-Alive 允許後續請求重複使用已經建立的連接，從而使請求更有效率並降低延遲。HTTP 保持活動狀態默認情況下與 JavaScript V3 啟用。

閒置連接可以保持活躍的時間是有限的。如果您有閒置連接，但希望下一個請求使用已建立的連接，請考慮發送定期請求，也許每分鐘發送一次請求。

Note

請注意，在 SDK 的較舊 V2 中，默認情況下保持活動處於關閉狀態，這意味著每個連接在使用後立即關閉。如果使用 V2，您可以覆寫此設定。

重試 Config

當 SDK 收到錯誤回應，而且由 SDK 決定可繼續發生錯誤時，例如節流例外或暫時性服務例外狀況，則會再次重試。作為調用者，您無形地發生這種情況，除了您可能會注意到請求花費更長的時間才能成功。

在放棄並將錯誤傳遞到調用上下文之前，JavaScript V3 的 SDK 將發出 3 個總請求，默認情況下。您可以調整這些重試的次數和頻率。

DynamoDBClient 構造函數接受一個限 `maxAttempts` 制會發生多少次嘗試的設置。下面的例子提出了從默認值 3 到總共 5 的值。如果將其設置為 0 或 1，則表示您不希望任何自動重試，並希望自己在 `catch` 塊中處理任何可恢復的錯誤。

```
const client = new DynamoDBClient({
  maxAttempts: 5,
});
```

您還可以使用自定義重試策略控制重試的時間。若要這麼做，請匯入 `util-retry` 公用程式套件並建立自訂輪詢函數，以根據目前的重試計數計算重試之間的等待時間。

下面的例子說，如果第一次嘗試失敗，最多可進行 5 次嘗試，延遲 15, 30, 90 和 360 毫秒。自訂輪詢函式會接受重試嘗試次數 (第一次重試時從 1 開始) 計算延遲 `calculateRetryBackoff`，並傳回等待該要求的毫秒數。

```
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.  
),  
});
```

等待程式

DynamoDB 用戶端包含兩個實用的[侍者函數](#)，當您希望程式碼等待資料表修改完成之前，建立、修改或刪除資料表時，可以使用這些函數。例如，您可以部署資料表、呼叫`waitUntilTableExists`函式，程式碼會封鎖，直到資料表變成 `ACTIVE` 為止。服務員會在內部輪詢 DynamoDB 服務，`describe-table`每 20 秒進行一次。

```
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}`);
```

只有當它可以執行顯示表狀態 `ACTIVE` 的`describe-table`命令時，該`waitUntilTableExists`功能才返回控制權。這樣可確保您可以用`waitUntilTableExists`來等待建立完成，以及新增 GSI 索引之類的修改，這可能需要一些時間才能套用資料表返回 `ACTIVE` 狀態。

錯誤處理

在這裡早期的例子中，我們已經廣泛地發現了所有錯誤。但是，在實際應用中，分辨各種錯誤類型並實現更精確的錯誤處理非常重要。

DynamoDB 錯誤回應包含中繼資料，包括錯誤名稱。您可以 `catch` 錯誤，然後匹配錯誤條件的可能字符串名稱，以確定如何繼續。對於伺服器端錯誤，您可以利用`instanceof`運算子搭配`@aws-sdk/client-dynamodb`封裝匯出的錯誤類型，以有效率地管理錯誤處理。

重要的是要注意，這些錯誤僅在所有重試用盡後才會顯示出來。如果重試錯誤並最終成功調用，則從代碼的角度來看，沒有錯誤只是稍微升高的延遲。重試將在 Amazon CloudWatch 圖表中顯示為不成功的請求，例如節流或錯誤請求。如果用戶端達到最大重試次數，就會放棄並產生例外狀況。這是用戶端說它不會重試的方式。

以下是 `catch` 獲錯誤並根據返回的錯誤類型採取行動的代碼片段。

```
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.
    }
  }
}
```

[the section called “錯誤處理”](#)如需 DynamoDB 開發人員指南中的常見錯誤字串，請參閱。可以在該 API 調用的文檔中找到任何特定 API 調用可能發生的確切錯誤，例如[查詢 API 文檔](#)。

錯誤的元數據包括其他屬性，具體取決於錯誤。對於 a `TimeoutError`，中繼資料包括已進行的嘗試次數和 `totalRetryDelay`，如下所示。

```
{
  "name": "TimeoutError",
  "$metadata": {
    "attempts": 3,
    "totalRetryDelay": 199
  }
}
```

如果您管理自己的重試原則，則需要區分節流和錯誤：

- 節流 (以 `ProvisionedThroughputExceededException` 或表示 `ThrottlingException`) 表示運作良好的服務，通知您已超過 DynamoDB 資料表或分區上的讀取或寫入容量。每隔一毫秒，就會提供更多的讀取或寫入容量，因此您可以快速重試 (例如每 50 毫秒)，以嘗試存取該新發行的容量。

使用節流時，您不需要特別需要指數輪詢，因為節流是輕量級的 DynamoDB 傳回，而且不會向您收取每個請求的費用。指數輪詢會將較長的延遲指派給已經等待最長時間的用戶端執行緒，從統計上將 p50 和 p99 向外延伸。

- 錯誤 (由 `InternalServerError` 或 a 表示 `ServiceUnavailable`，等等) 表示服務存在暫時性問題，可能是整個表格，或者只是您正在讀取或寫入的分區。如果發生錯誤，您可以在重試之前暫停更長的時間，例如 250 毫秒或 500 毫秒，並使用抖動來錯開重試。

日誌

打開日誌記錄以獲取有關 SDK 正在執行的更多詳細信息。您可以在上設定參數，如下列範例所示。DynamoDBClient 更多記錄檔資訊會顯示在主控台中，並包含中繼資料，例如狀態碼和使用的容量。如果您在終端機視窗中本機執行程式碼，記錄會出現在該處。如果您在中運行代碼 AWS Lambda，並且已經設置了 Amazon CloudWatch 日誌，那麼控制台輸出將寫入那裡。

```
const client = new DynamoDBClient({
  logger: console
});
```

您也可以掛接到內部 SDK 活動，並在某些事件發生時執行自訂記錄。下面的示例使用客戶端 `middlewareStack` 端攔截從 SDK 發送的每個請求，並在發生時對其進行記錄。

```
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提供了一個強大的鉤子來觀察和控制 SDK 行為。如需詳細資訊 AWS SDK for JavaScript，請參閱[模組化介紹中介軟體堆疊](#)的部落格。

考量事項

AWS SDK for JavaScript 在您的項目中實施時，以下是一些需要考慮的其他因素。

模塊系統

該 SDK 支持兩個模塊系統，共同 JS 和 ES (ECMAScript)。CommonJS 使用的require功能，而 ES 使用的import關鍵字。

1. 常見的 JS-const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");
2. ES (電子印刷稿 — import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

項目類型指定要使用的模塊系統，並在 package.json 文件的類型部分中進行指定。預設值為普通 JS。用"type": "module"於指示 ES 專案。如果您有使用 CommonJS 套件格式的現有 Node.JS 專案，您仍然可以使用 .mjs 副檔名命名您的函數檔案，來新增具有更現代 SDK V3 匯入語法的函數。這將允許代碼文件被視為 ES (ECMAScript)。

非同步作業

您會看到許多使用回呼和承諾來處理 DynamoDB 作業結果的程式碼範例。隨著現代 JavaScript 這種複雜性不再需要，開發人員可以利用更簡潔和可讀的異步操作異步/等待語法。

Web 瀏覽器運行時

Web 和移動開發人員建立與反應或反應原生可以 JavaScript 在他們的項目中使用 SDK。隨著 SDK 的早期 V2，網絡開發人員將不得不將完整的 SDK 加載到瀏覽器中，引用託管在 https://sdk.amazonaws.com/js/ 的 SDK 映像。

使用 V3 時，可以使用 Webpack 將所需的 V3 用戶端模組和所有必要的 JavaScript 函數捆綁到單個 JavaScript 檔案中，並將其新增到 HTML 頁面<head>的指令碼標記中，如 SDK 文件的[瀏覽器指令碼入門一節](#)所述。

DAX 資料平面作業

JavaScript V3 適用的開發套件目前不會提供 Amazon DynamoDB 串流加速器 (DAX) 資料平面操作的支援。如果您要求 DAX 支援，請考慮使用支援 DAX 資料平面作業的 JavaScript V2 SDK。

使用 Amazon DynamoDB 進行編程 AWS SDK for Java 2.x

本指南為想要將 Amazon DynamoDB 與 Java 搭配使用的程式設計人員提供方向。本指南涵蓋了不同的概念，例如抽象層，配置管理，錯誤處理，控制重試策略以及管理保持活動狀態。

主題

- [關於 AWS SDK for Java 2.x](#)
- [開始使用 AWS SDK for Java 2.x](#)
- [使用文 AWS SDK for Java 2.x 件](#)
- [支援的介面](#)
- [其他程式碼範例](#)
- [同步和異步編程](#)
- [HTTP 用戶端](#)
- [設定 HTTP 用戶端](#)
- [錯誤處理](#)
- [AWS 請求識別碼](#)
- [日誌](#)
- [分頁](#)
- [資料類別註解](#)

關於 AWS SDK for Java 2.x

您可以使用官方從 Java 存取 DynamoDB 資料庫。AWS SDK for Java SDK for Java 有兩個版本：1.x 和 2.x。對於 1.x，該 end-of-support 報告於二零二四年一月十二日[宣布](#)。它於 2024 年 7 月 31 日進入維護模式，截止日期 end-of-support 為 2025 年 12 月 31 日。對於新開發，我們強烈建議您使用 2.x，該 2.x 於 2018 年首次發布。本指南專門針對 2.x，並僅著重於與 DynamoDB 相關的 SDK 部分。

您可以在 SDK 和工具維護[政策以及AWS SDK 和工具版本支援矩陣主題中找到有關AWS SDK 和工具參考指南的維護和支援](#)的更多資訊。AWS

這 AWS SDK for Java 2.x 是 1.x 代碼庫的主要重寫，以支持現代 Java 功能，例如 Java 8 中引入的非阻塞 I/O。適用於 Java 2.x 的 SDK 還增加了對可插拔 HTTP 客戶端實現的支持，以為網絡連接提供更大的靈活性和可配置性。

適用於 Java 1.x 的 SDK 和適用於 Java 2.x 的 SDK 之間的一個明顯變化是使用新的軟件包名稱。Java 1.x 開發套件使用 `com.amazonaws` 套件名稱，而 Java 2.x SDK 使用 `software.amazon.awssdk` 同樣，對於 Java 1.x SDK 的 Maven 工件使用 `com.amazonaws groupId`，而 Java 2.x SDK 工件使用 `software.amazon.awssdk groupId`。

Important

AWS SDK for Java 1.x 有一個 DynamoDB 的套件。 `com.amazonaws.dynamodbv2` 套件名稱中的 `v2` 並不表示它是 Java `v2`。 `v2` 表示套件支援 DynamoDB 低階 API 的 [第二個版本](#)，而不是低階 API 的 [原始版本](#)。

Support Java 版本

提 AWS SDK for Java 2.x 供對長期支援 (LTS) [Java 版本](#) 的完整支援。

開始使用 AWS SDK for Java 2.x

下面的教程告訴你如何使用 [阿帕奇的 Maven](#) 定義的 Java 2.x 的 SDK 的依賴關係。本教學課程也會示範如何撰寫連線至 DynamoDB 的程式碼，以列出可用的 DynamoDB 表格。本教學課程是 [以開始使用 AWS SDK for Java 2.x](#)。我們已編輯此教學課程，以呼叫 DynamoDB 而非 Amazon S3。

請執行下列步驟來完成此教學課程：

- [步驟 1：設定此教學課程](#)
- [步驟 2：建立專案](#)
- [步驟 3：撰寫程式碼](#)
- [步驟 4：建置並執行應用程式](#)

步驟 1：設定此教學課程

在開始本自學課程之前，您需要下列項目：

- 存取 Amazon DynamoDB 權限。
- 設定為 AWS 服務 使用單一登入存取的 Java 開發環境 AWS IAM Identity Center

使用 [安裝程式概觀](#) 中的指示進行此教學課程的設定。在 [您為 Java SDK 設定開發環境的單一登入存取權](#)，並且擁有 [使用中 AWS 存取入口網站工作階段](#) 之後，請繼續執行本教學課程的 [步驟 2](#)。

步驟 2：建立專案

若要為本教學課程建立專案，請執行 Maven 命令，提示您輸入如何設定專案。輸入並確認所有輸入後，Maven 通過創建文件並創建存根 Java 文件來完成構建項目。pom.xml

1. 開啟終端機或命令提示字元視窗，然後導覽至您選擇的目錄，例如您的Desktop或Home資料夾。
2. 在終端機上輸入以下命令，然後按下Enter。

```
mvn archetype:generate \  
  -DarchetypeGroupId=software.amazon.awssdk \  
  -DarchetypeArtifactId=archetype-app-quickstart \  
  -DarchetypeVersion=2.22.0
```

3. 為每個提示輸入第二欄中列示的值。

提示	要輸入的值
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>

提示	要輸入的值
Define value for property 'package' org.example:	<Enter>

4. 輸入最後一個值後，Maven 會列出您所做的選擇。透過輸入Y或輸入重新輸入值來確認N。

Maven 創建getstarted根據您輸入的artifactId值命名的項目文件夾。在資料夾getstarted內，尋找README.md您可以檢閱的pom.xml檔案、檔案和src目錄。

Maven 的構建下面的目錄樹。

```
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

以下展示了pom.xml項目文件的內容。

pom.xml

本dependencyManagement節包含的相依性，AWS SDK for Java 2.x 而且該dependencies區段具有 Amazon DynamoDB 的相依性。指定這些依賴關係迫使 Maven 將相關的 jar 文件包含到您的 Java 類路徑中。依預設，AWS SDK 不會包含所有類別 AWS 服務。對於 DynamoDB，dynamodb-enhanced如果您使用低階介面或使用高階介面，您應該

對dynamodb成品具有相依性。如果您沒有包含相關的依賴關係，您的代碼將無法編譯。由於maven.compiler.source和maven.compiler.target屬性中的1.8值，專案會使用 Java 1.8。

```
<?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>
```

步驟 3：撰寫程式碼

下面的代碼顯示了由 Maven 創建的 App 類。該 main 方法是應用程序的進入點，它創建了該 Handler 類的實例，然後調用其 sendRequest 方法。

App 類別

```
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");
    }
}
```

由 Maven 創建的 `DependencyFactory` 類包含構建並返回 `DynamoDbClient` 實例的 `dynamoDbClient` 工廠方法。 `DynamoDbClient` 執行個體會使用以阿帕奇為基礎的 HTTP 用戶端的執行個體。這是因為您 `apache-client` 在 Maven 提示您要使用哪個 HTTP 客戶端時指定。

顯示 `DependencyFactory` 在下面的代碼。

DependencyFactory 類

```
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();
    }
}
```

```
}
```

該Handler類包含程序的主要邏輯。在App類別中建立Handler的執行個體時，會DependencyFactory提供DynamoDbClient服務用戶端。您的程式碼會使用DynamoDbClient執行個體來呼叫 DynamoDB 服務。

Maven 生成帶有*TODO*註釋以下Handler類。教學課程的下一個步驟會取代為程*TODO*式碼。

Handler類, 馬文生成

```
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.
    }
}
```

若要填入邏輯，請使用下列程式碼取代Handler類別的全部內容。該sendRequest方法被填入，並添加必要的導入。

處理常式類別, 實作

下列程式碼會使用[DynamoDbClient](#)執行個體擷取現有資料表的清單。如果指定帳戶和 Region 存在資料表，則程式碼會使用Logger執行個體來記錄這些資料表的名稱。

```
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));
        }
    }
}
```

步驟 4：建置並執行應用程式

創建項目並包含完整的Handler類後，構建並運行應用程式。

1. 確保您擁有有效的 IAM 身分中心工作階段。若要這麼做，請執行 AWS Command Line Interface 命令 `aws sts get-caller-identity` 並檢查回應。如果您沒有使用中的工作階段，請參閱 [使用登入以取 AWS CLI](#) 得指示。
2. 打開終端機或命令提示符窗口，然後導航到您的項目目錄 `getstarted`。
3. 使用以下命令來構建您的項目：

```
mvn clean package
```

4. 使用下面的命令來運行應用程式。

```
mvn exec:java -Dexec.mainClass="org.example.App"
```

檢視檔案後，請刪除物件，然後刪除值區。

Success (成功)

如果您的 Maven 項目構建並運行沒有錯誤，那麼恭喜！您已經使用適用於 Java 2.x 的 SDK 成功構建了第一個 Java 應用程序。

清除

若要清理您在本教學課程中建立的資源，請執行下列操作：

- 刪除專案資料夾 `getstarted`。

使用文 AWS SDK for Java 2.x 件

該[AWS SDK for Java 2.x 文檔](#)涵蓋了所有 SDK 的所有方面 AWS 服務。請使用下列主題作為起點：

- [從 1.x 版遷移至 2.x 版](#) — 包含 1.x 與 2.x 之間差異的詳細說明。本主題也包含如何使用這兩個主要版本的指示 side-by-side。
- [適用於 Java 2.x 開發套件的 DynamoDB 指南](#) — 向您展示如何執行基本 DynamoDB 作業：建立資料表、操作項目以及擷取項目。這些範例使用低階介面。Java 具有多個接口，如以下部分所述：[支援的介面](#)。

Tip

我們建議您在完成文件中的這些主題後，將 [Javadoc AWS SDK for Java 2.x](#) 文件加入書籤。它涵蓋了所有內容，AWS 服務 並將充當您的主要 API 參考。

支援的介面

根據您想要的抽象層級，AWS SDK for Java 2.x 支援以下介面。

本節主題

- [低階介面](#)
- [高階介面](#)
- [文件介面](#)
- [比較介面與查詢範例](#)

低階介面

低階介面提供基礎服務 API 的 one-to-one 對應。每個 DynamoDB API 都可透過此介面取得。這意味著低級界面可以提供完整的功能，但使用起來通常更加冗長和複雜。例如，您必須使用 `.s()` 函數來保存字串和 `.n()` 保存數字的功能。下面的例子中 [PutItem](#) 插入使用低級接口的項目。

```
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)");
    }
}
```

高階介面

中的高階介面稱 AWS SDK for Java 2.x 為 DynamoDB 增強型用戶端。此介面提供更慣用的程式碼撰寫體驗。

增強型用戶端提供了一種在用戶端資料類別和專為儲存該資料而設計的 DynamoDB 表之間進行對應的方法。您要在自己的程式碼中定義資料表及其對應模型類別之間的關係。然後，您可以依靠 SDK 來管理數據類型操作。如需增強型用戶端的詳細資訊，請參閱文件中的 [DynamoDB 增強型用戶端 API](#)。AWS SDK for Java 2.x

下面的示例 `PutItem` 使用高級接口。在此範例中，`DynamoDbBean` 具名 `YourItem` 會建立 `TableSchema` 允許其直接用作 `putItem()` 呼叫的輸入。

```
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 具有自己的高級接口，通常由其主類DynamoDBMapper引用。在一個單獨的包（和 Maven 工件）中發布名為software.amazon.awssdk.enhanced.dynamodb。AWS SDK for Java 2.x Java 2.x SDK 通常由其主類別DynamoDbEnhancedClient稱為。

使用不可變數據類的高級接口

DynamoDB 增強型用戶端 API 的對應功能也適用於不可變的資料類別。不可變類只有 getter，並且需要 SDK 用於創建類實例的構建器類。Java 中的不變性是一種常用的樣式，它允許開發人員創建無副作用的類，因此在複雜的多線程應用程序中的行為更可預測。而不是使用@DynamoDbBean註釋，如圖所示[High-level interface example](#)，不可變的類使用@DynamoDbImmutable註釋，它將構建器類作為其輸入。

下列範例會將建置器類別DynamoDbEnhancedClientImmutablePutItem當做輸入，以建立資料表結構定義。此範例接著會提供結構描述做為 [PutItem](#) API 呼叫的輸入。

```
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)");
    }
}
```

```
}  
}
```

下面的例子顯示了不可變的數據類。

```
@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); }
}
}
```

使用不可變數據類和第三方樣板生成庫的高級接口

上一節中提到的不可變數據類示例需要一些樣板代碼。例如，除了 Builder 類的數據類的吸氣和 setter 邏輯。協力廠商程式庫 (例如 [Project Lombok](#)) 可以協助您產生該樣板程式碼類型。

AWS SDK for Java 2.x 文件中的「[使用協力廠商程式庫](#)」(例如「[Lombok](#)」區段) 介紹了利用 Project Lombok 程式庫的概念。減少大部分樣板代碼可幫助您限制使用不可變數據類別和 SDK 所需的代碼量。AWS 這進一步導致提高了代碼的生產力和可讀性。

下列範例示範 Lombok 專案如何簡化使用 DynamoDB 增強型用戶端 API 所需的程式碼。

```
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)");
    }
}

```

下面的例子顯示了不可變數據類的不可變數據對象。

```

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;
}

```

該 `YourImmutableLombokItem` 類使用由項目龍目島和 AWS SDK 提供的以下註釋：

- [@Builder](#) - 為龍目島項目提供的數據類生成複雜的構建器 API。
- [@DynamoDbImmutable](#) — 將類別識 `DynamoDbImmutable` 別為 SDK 提供的 DynamoDB 可對應實體註釋。AWS
- [@Value](#) — 的不可變變體 `@Data`；預設情況下，所有欄位都是私有且最終的，並且不會產生 setter。此註釋由項目龍目島提供。

文件介面

Document 介面可避免指定資料類型描述元的需求。資料類型是由資料本身的語義隱含的。如果您熟悉 AWS SDK for Java 1.x Document 介面，中的「文件」介面會 AWS SDK for Java 2.x 提供類似但重新設計的介面。

以下[Document interface example](#)顯示使用「文件」介面表示的PutItem呼叫。這個例子也使用 EnhancedDocument。若要使用增強型文件 API 針對 DynamoDB 表執行命令，您必須先將表格與文件表格結構描述產生關聯，才能建立 DynamoDBTable 資源物件。Document 表格結構描述產生器需要主索引鍵和屬性轉換器提供者。

您可以使用AttributeConverterProvider.defaultProvider()來轉換預設類型的文件屬性。您可以使用自訂AttributeConverterProvider實作來變更整體預設行為。您也可以變更單一屬性的轉換器。[AWS SDK 文檔](#)提供了有關如何使用自定義轉換器的更多詳細信息和示例。它們的主要用途是用於沒有默認轉換器的域類的屬性。使用自訂轉換器，您可以為 SDK 提供寫入或讀取 DynamoDB 所需的資訊。

```
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)");
}
}
```

使用下列公用程式方法，您可以將 JSON 文件轉換為原生 Amazon DynamoDB 資料類型，以及從原生 Amazon DynamoDB 資料類型

- [EnhancedDocument.fromJson\(String json\)](#) 從 JSON 字串建立新 EnhancedDocument 執行個體。
- [EnhancedDocument.toJson\(\)](#)— 創建文檔的 JSON 字符串表示，它允許您像任何其他 JSON 對象一樣在應用程序中使用它。

比較介面與查詢範例

本節顯示使用各種接口表達的相同[查詢](#)調用。這些查詢使用幾個屬性來微調查詢結果：

- 您必須完全指定分割區索引鍵，因為 DynamoDB 會鎖定一個特定的分區索引鍵值。
- 排序索引鍵具有使用的索引鍵條件運算式，begins_with 以便只針對此查詢的購物車項目。
- 我們將查詢限制為使用返回最多 100 個項目 limit()。
- 我們設置 scanIndexForward 為假。結果會以 UTF-8 位元組的順序傳回，這通常表示會先傳回數目最低的購物車項目。通過將設置 scanIndexForward 為 false，我們反轉順序，並首先返回具有最高數量的購物車項目。

- 我們應用過濾器來刪除不匹配條件的任何結果。無論項目是否符合篩選器，所篩選的資料都會消耗讀取容量。

Example 使用低級接口進行查詢

下列範例會查詢名為YourTableName使用 a 的資料表keyConditionExpression，這會將查詢限制為特定的分區索引鍵值，並以特定前置碼值開頭的排序索引鍵值。這些關鍵條件會限制從 DynamoDB 讀取的資料量。最後，查詢會使用 a 對從 DynamoDB 擷取的資料套用篩選器。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 使用文件介面查詢

下列範例會查詢YourTableName使用文件介面命名的表格。

```
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 使用高階介面查詢

下列範例會查詢使用 DynamoDB 增強型YourTableName用戶端 API 命名的資料表。

```
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")))
            .build());
    }
}
```

```
        .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;
}
}
```

使用不可變數據類的高級接口

當您使用高階不可變資料類別執行 Query 時，除了實體類別 `YourItem` 或 `YourImmutableItem` 建構之外，程式碼與高階介面範例相同。如需詳細資訊，請參閱 [PutItem](#) 範例。

使用不可變數據類和第三方樣板生成庫的高級接口

當您使用高階不可變資料類別執行 Query 時，除了實體類別 `YourItem` 或 `YourImmutableLombokItem` 建構之外，程式碼與高階介面範例相同。如需詳細資訊，請參閱 [PutItem](#) 範例。

其他程式碼範例

您也可以參考下列程式碼範例儲存庫，其中提供了額外的範例，以便將 DynamoDB 與適用於 Java 2.x 的 SDK 搭配使用：

- [官方 AWS 單動作程式碼範例](#)
- [社群維護的單動作程式碼範例](#)
- [官方 AWS 場景導向的代碼示例](#)

同步和異步編程

提 AWS SDK for Java 2.x 供同步和非同步用戶端 AWS 服務，例如 DynamoDB。

DynamoDbClient和DynamoDbEnhancedClient類別會提供同步方法，這些方法會封鎖執行緒的執行，直到用戶端收到來自服務的回應為止。如果您不需要非同步作業，此用戶端是與 DynamoDB 互動的最直接方式。

DynamoDbAsyncClient和DynamoDbEnhancedAsyncClient類提供立即返回的異步方法，並將控制權交還給調用線程，而無需等待響應。非阻塞客戶端的優點是它允許在幾個線程之間進行高度並行，從而以最少的計算資源提供有效的 I/O 請求處理。這提高了吞吐量和響應能力。

AWS SDK for Java 2.x 使用原生支援非阻塞 I/O。AWS SDK for Java 1.x 必須模擬非阻塞 I/O。

同步方法在響應可用之前返回，因此您需要一種方法在準備就緒時獲取響應。在異步方法 AWS SDK for Java 返回一個包含 future 異步操作結果的 [CompletableFuture](#) 對象。當你調用 `get()` 或 `join()` 對這些 `CompletableFuture` 對象，你的代碼將阻塞，直到結果可用。如果您在發出請求的同時進行此調用，它的行為與普通同步調用類似。

[非同步程](#)式設計的 AWS SDK 文件提供了有關如何利用這種非同步樣式的更多詳細資訊。

HTTP 用戶端

為了支援每個用戶端，存在一個 HTTP 用戶端可以處理與 AWS 服務。您可以插入替代 HTTP 客戶端，選擇具有最適合您應用程序特性的客戶端。有些更輕量；有些則有更多的組態選項。

某些 HTTP 用戶端僅支援同步使用，而其他用戶端僅支援非同步使用。AWS SDK 文件提供 [流程圖](#)，可協助您為工作負載選取最佳的 HTTP 用戶端。

下面的列表顯示了一些可能的 HTTP 客戶端：

主題

- [基於阿帕奇的 HTTP 客戶端](#)
- [以網址連線為基礎的 HTTP 用戶端](#)
- [基於網絡的 HTTP 客戶端](#)
- [AWS 基於 CRT 的 HTTP 客戶端](#)

基於阿帕奇的 HTTP 客戶端

[ApacheHttpClient](#) 支持同步服務客戶端，並且是用於同步使用的默認 HTTP 客戶端。如需有關設定的資訊 `ApacheHttpClient`，請參閱文件中的 [設定以 Apache 為基礎的 HTTP 用戶端](#)。AWS SDK for Java 2.x

以網址連線為基礎的 HTTP 用戶端

[URLConnectionHttpClient](#)這是同步用戶端的另一個選項。它的載入速度比以 Apache 為基礎的 HTTP 用戶端更快，但功能較少。如需有關設定的資訊 [URLConnectionHttpClient](#)，請參閱[設定 URL 連線型 HTTP 用戶端](#)。

基於網絡的 HTTP 客戶端

[NettyNioAsyncHttpClient](#)支持異步客戶端，是異步使用的默認選擇。如需有關設定的資訊 [NettyNioAsyncHttpClient](#)，請參閱[設定以 NetTT 為基礎的 HTTP 用戶端](#)。

AWS 基於 CRT 的 HTTP 客戶端

較新 [AwsCrtHttpClient](#) 的 [AwsCrtAsyncHttpClient](#) AWS 通用運行時 (CRT) 庫是另一個支持同步和異步客戶端的選項。與其他 HTTP 客戶端相比，AWS CRT 提供：

- 更快的 SDK 啟動時間
- 更小的內存佔用
- 減少延遲時間
- 連線健康管理
- 負載平衡

如需有關設定 [AwsCrtHttpClient](#) 和 [的資訊AwsCrtAsyncHttpClient](#)，請參閱[設定 AWS 以 CRT 為基礎的 HTTP 用戶端](#)。

AWS CRT 型 HTTP 用戶端不是預設值，因為這會破壞現有應用程式的回溯相容性。但是，對於 DynamoDB，我們建議您使用 AWS 以 CRT 為基礎的 HTTP 用戶端進行同步和非同步用途。

如需有關 AWS 以 CRT 為基礎的 HTTP 用戶端的簡介，請參閱中 [的宣布 AWS CRT HTTP 用戶端的可用性](#)。AWS SDK for Java 2.x

設定 HTTP 用戶端

設定用戶端時，您可以提供各種組態選項，包括：

- 設置 API 調用的不同方面的超時
- 控制 TCP 保持活動狀態是否已啟用
- 發生錯誤時控制重試原則

- 指定執行[攔截器執行個體可以修改](#)的執行屬性。執行攔截器可以撰寫攔截 API 要求和回應執行的程式碼。這使您可以執行任務，例如發布指標和修改傳輸中的請求。
- 新增或操作 HTTP 標頭
- 啟用用[戶端效能指標](#)的追蹤。使用此功能可協助您收集有關應用程式中服務用戶端的指標，並分析 Amazon 中的輸出 CloudWatch。
- 指定用於排程工作的替代執行程式服務，例如非同步重試嘗試和逾時工作

您可以透過將[ClientOverrideConfiguration](#)物件提供給服務用戶端Builder類別來控制組態。您將在以下各節的一些代碼示例中看到這一點。

提ClientOverrideConfiguration供標準組態選擇。不同的可插拔 HTTP 客戶端也具有特定於實現的配置可能性。這些客戶端中的每一個也維護自己的文檔：

- [基於阿帕奇的 HTTP 客戶端](#)
- [以網址連線為基礎的 HTTP 用戶端](#)
- [基於網絡的 HTTP 客戶端](#)
- [AWS 基於 CRT 的 HTTP 客戶端](#)

本節主題

- [逾時組態](#)
- [RetryMode](#)
- [DefaultsMode](#)
- [保持活動配置](#)

逾時組態

您可以調整用戶端組態，以控制與維修呼叫相關的各種逾時。與其他產品相比，DynamoDB 提供的延遲時間較低。AWS 服務因此，您可能想要調整這些屬性以降低逾時值，以便在發生網路問題時快速失敗。

您可以使用 DynamoDB 用戶端ClientOverrideConfiguration上或變更基礎 HTTP 用戶端實作的詳細組態選項，來自訂延遲相關行為。

您可以使用以下方法配置以下具有影響力的屬性：ClientOverrideConfiguration

- apiCallAttemptTimeout放棄並逾時之前，等待 HTTP 要求完成單次嘗試的時間量。

- `apiCallTimeout` 允許用戶端完成 API 呼叫執行的時間量。這包括由所有 HTTP 要求 (包括重試) 組成的要求處理常式執行。

提 AWS SDK for Java 2.x 供某些逾時選項的預設值，例如連線逾時和通訊端逾時。SDK 不會為 API 呼叫逾時或個別 API 呼叫嘗試逾時提供預設值。如果未在中設定這些逾時 `ClientOverrideConfiguration`，SDK 將使用通訊端逾時值來取代整體 API 呼叫逾時。通訊端逾時的預設值為 30 秒。

RetryMode

與您應該考慮的超時配置相關的另一個配置是 `RetryMode` 配置對象。此配置物件包含重試行為的集合。

適用於 Java 2.x 的開發套件支援下列重試模式：

- `legacy`— 默認的重試模式，如果你沒有明確地改變它。此重試模式特定於 Java SDK，其特點是：
 - 最多可重試三次，或者針對服務進行更多次嘗試，例如最多可重試 8 次的 DynamoDB。
- `standard`— 命名為標準，因為它與其他 AWS SDK 更一致。此模式會等待第一次重試的隨機時間範圍從 0 毫秒到 1,000 毫秒。如果需要另一次重試，它會從 0 毫秒到 1,000 毫秒選擇另一個隨機時間，並將其乘以 2。如果需要額外的重試，它會執行相同的隨機選擇乘以 4，依此類推。每次等待時間上限為 20 秒。此模式會在偵測到的失敗情況下執行重試次數超過 `legacy` 模式。對於 DynamoDB，除非您使用覆寫，否則它可以執行最多三次總嘗試次數。[numRetries](#)
- `adaptive`— 建立在 `standard` 模式上並動態限制 AWS 請求的速率，以最大限度地提高成功率。這可能是以請求延遲為代價的。如果可預測的延遲很重要，我們不建議調適性重試模式。

您可以在 AWS SDK 和工具參考指南的「[重試行為](#)」主題中找到這些重試模式的擴展定義。

重試原則

所有 `RetryMode` 配置都有一個 [RetryPolicy](#)，它是基於一個或多個 [RetryCondition](#) 配置構建的。對 [TokenBucketRetryCondition](#) 於 DynamoDB SDK 用戶端實作的重試行為而言，這一點特別重要。此條件會限制 SDK 使用權杖值區演算法進行的重試次數。視選取的重試模式而定，節流例外可能會或可能不會從 `TokenBucket`

當用戶端遇到可重試的錯誤 (例如節流例外狀況或暫時性伺服器錯誤) 時，SDK 會自動重試要求。您可以控制這些重試發生的次數和速度。

設定用戶端時，您可以提供 `RetryPolicy` 支援下列參數的：

- `numRetries` 在請求被視為失敗之前，應套用的重試次數上限。無論您使用的重試模式為何，預設值都是 8。

Warning

請務必在適當考量之後變更此預設值。

- `backoffStrategy` 套用 [BackoffStrategy](#) 至重試，其 [FullJitterBackoffStrategy](#) 為預設策略。此策略會根據目前的次數或重試次數、基本延遲和最大輪詢時間，在其他重試之間執行指數延遲。然後，它增加了抖動以提供一些隨機性。無論重試模式如何，指數延遲中使用的基本延遲都是 25 ms。
- `retryCondition` [RetryCondition](#) 決定是否應重試請求。默認情況下，它將重試一組特定的 HTTP 狀態代碼和異常，它認為是可重試的。在大多數情況下，預設組態應該已足夠。

下列程式碼提供替代的重試原則。它指定允許總共五次重試（總共六個請求）。第一次重試應該在大約 100 毫秒的延遲之後進行，每次額外的重試會以指數方式加倍該時間，最多延遲一秒。

```
DynamoDbClient client = DynamoDbClient.builder()
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .retryPolicy(RetryPolicy.builder()
            .backoffStrategy(FullJitterBackoffStrategy.builder()
                .baseDelay(Duration.ofMillis(100))
                .maxBackoffTime(Duration.ofSeconds(1))
                .build())
            .numRetries(5)
            .build())
        .build())
    .build();
```

DefaultsMode

逾時屬性不受管理，`ClientOverrideConfiguration` 且 `RetryMode` 常未明確設定。而是透過指定 `DefaultsMode` 個隱含設定它們的組態。

在 AWS SDK for Java 2.x (版本 2.17.102 或更新版本) 中引入的 `Support`，可為常見的可設定設定提供一組預設值，例如 HTTP 通訊設定、重試行為、服務地區端點設定，以及可能的任何 SDK 相關組態。`DefaultsMode` 使用此功能時，您可以取得針對常見使用案例量身打造的新組態預設值。

默認模式在所有 AWS SDK 中都進行了標準化。適用於 Java 2.x 的開發套件支援下列預設模式：

- `legacy`— 提供依 SDK 而有所不同且在建立之前 `DefaultsMode` 已存在的預設設定。

- `standard`— 提供大多數案例的預設非最佳化設定。
- `in-region`— 基於標準模式構建，並包括針對 AWS 服務 從同一模式調用的應用程序量身定制的設置 AWS 區域。
- `cross-region`— 基於標準模式構建，並包括具有高超時的設置，適用於調用不同 AWS 服務 AWS 區域的應用程序。
- `mobile`— 基於標準模式構建，並包括針對延遲較高的移動應用程序量身定制的高超時設置。
- `auto`— 基於標準模式構建，並包括實驗功能。SDK 會嘗試探索執行階段環境，以自動決定適當的設定。自動檢測是基於啟發式的，不提供 100% 的準確性。如果無法確定運行時環境，則使用標準模式。自動偵測可能會查詢[執行個體中繼資料](#)和[使用者資料](#)，這可能會導致延遲。如果啟動延遲對您的應用程式很重要，我們建議您將 `DefaultsMode` 改為選擇明確的。

您可以透過下列方式設定預設模式：

- 直接在客戶端上通過 `AwsClientBuilder.Builder#defaultsMode(DefaultsMode)`。
- 在配置配置文件中通過 `defaults_mode` 配置文件屬性。
- 通過 `aws.defaultsMode` 系統屬性全局。
- 通過 `AWS_DEFAULTS_MODE` 環境變量全局。

Note

對於除此之外的任何模式 `legacy`，出售的默認值可能會隨著最佳實踐的發展而改變。因此，如果您使用的是以外的模式，建議您在升級 SDK 時執行測試 `legacy`。

AWS SDK 和工具參考指南中的 [智慧型組態預設值](#) 提供了不同預設模式下的組態屬性及其預設值的清單。

您可以根據應用程式的特性選擇預設模式值，並與應 AWS 服務 用程式互動。

這些值的配置時考慮到了廣泛 AWS 服務 的選擇。對於在一個區域中部署 DynamoDB 表和應用程式的典型 DynamoDB 部署，預設模式在 `in-region` 預設模式中最為相關。 `standard`

Example 針對低延遲呼叫進行調整的 DynamoDB SDK 用戶端組態

下列範例會將逾時調整為降低預期低延遲 DynamoDB 呼叫的值。

```
DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.builder()
```

```
.defaultsMode(DefaultsMode.IN_REGION)
.httpClientBuilder(AwsCrtAsyncHttpClient.builder())
.overrideConfiguration(ClientOverrideConfiguration.builder()
    .apiCallTimeout(Duration.ofSeconds(3))
    .apiCallAttemptTimeout(Duration.ofMillis(500))
    .build())
.build();
```

個別的 HTTP 用戶端實作可能會為您提供更精細的逾時和連線使用行為控制。例如，如果是以 AWS CRT 為基礎的用戶端，您可以啟用 `ConnectionHealthConfiguration` 它允許 AWS CRT 型用戶端主動監控已使用連線的健全狀況。如需詳細資訊，請參閱的 [文件](#) AWS SDK for Java 2.x。

保持活動配置

啟用保持活動狀態可以通過重複使用連接來減少延遲。有兩種不同類型的保持活動：HTTP 保持活動和 TCP 保持活動。

- **HTTP Keep-Alive** 嘗試維護用戶端和伺服器之間的 HTTPS 連線，以便稍後的要求可以重複使用該連線。這會在以後的請求中跳過重量級 HTTPS 身份驗證。HTTP 保持活動狀態預設會在所有用戶端上啟用。
- **TCP Keep-Alive** 請求底層操作系統通過套接字連接發送小數據包，以提供額外的保證套接字保持活動狀態並立即檢測到任何丟棄。這可確保以後的請求不會花費時間嘗試使用丟棄的套接字。TCP 保持活動狀態預設會在所有用戶端上停用。您可能想要啟用它。下列程式碼範例會示範如何使用每個 HTTP 用戶端執行此作業。針對所有非 CRT 型 HTTP 用戶端啟用時，實際的「持續作用」機制取決於作業系統。因此，您必須透過作業系統設定其他 TCP 保持連線值，例如逾時和封包數目。您可以使用 Linux 或 Mac 機器 `sysctl` 上的註冊表值來執行此操作，也可以使用 Windows 計算機上的註冊表值。

Example 在基於阿帕奇的 HTTP 客戶端上啟用 TCP 保持活動狀態

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(ApacheHttpClient.builder().tcpKeepAlive(true))
    .build();
```

以網址連線為基礎的 HTTP 用戶端

任何使用以 URL 連線為基礎的 HTTP 用戶端的同步用戶端都 [HttpURLConnection](#) 沒有啟用連線的 [機制](#)。

Example 在基於網絡的 HTTP 客戶端上啟用 TCP 保持活動狀態

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Example 在以 AWS CRT 為基礎的 HTTP 用戶端上啟用 TCP 保持活動狀態

使用 AWS CRT 型 HTTP 用戶端，您可以啟用 TCP 保持活動狀態並控制持續時間。

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(AwsCrtHttpClient.builder()
        .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
            .keepAliveInterval(Duration.ofSeconds(50))
            .keepAliveTimeout(Duration.ofSeconds(5))
            .build()))
    .build();
```

使用非同步 DynamoDB 用戶端時，您可以啟用 TCP 保持活動狀態，如下列程式碼所示。

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()
        .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
            .keepAliveInterval(Duration.ofSeconds(50))
            .keepAliveTimeout(Duration.ofSeconds(5))
            .build()))
    .build();
```

錯誤處理

當涉及到異常處理時，AWS SDK for Java 2.x 使用運行時（未選中）異常。

覆蓋所有 SDK 異常的基本異常是 [SdkServiceException](#)，它從未選中的 Java 擴展 `RuntimeException`。如果你發現這一點，你會 catch SDK 拋出的所有異常。

`SdkServiceException` 有一個名為的子類別 [AwsServiceException](#)。該子類別表示與「」通信中出現任何問題 AWS 服務。它有一個名為的子類別 [DynamoDbException](#)，表示與 DynamoDB 通訊時存在問題。如果您發現這一點，您將 catch 與 DynamoDB 相關的所有異常，但不會捕獲其他 SDK 例外。

下有更特定的[例外型態](#) `DynamoDbException`。其中一些例外狀況類型適用於控制平面操作，例如 [TableAlreadyExistsException](#) 其他則適用於資料平面作業。以下是一個常見的數據平面異常的例子：

- [ConditionalCheckFailedException](#) 您在請求中指定了評估為 `false` 的條件。例如，您可能已嘗試對項目執行條件式更新，但屬性的實際值不符合條件中的預期值。以這種方式失敗的請求將不會重試。

其他情況沒有定義特定的例外狀況。例如，當您的請求被限制時，特定的 `ProvisionedThroughputExceededException` 可能會被拋出，而在其他情況下，會拋出更通用 `DynamoDbException` 的。在任何一種情況下，您都可以通過檢查是否返回來確定異常是由節流引起的。 `isThrottlingException()` `true`

根據您的應用程式需求，您可以 `catch` 取所有 `AwsServiceException` 或 `DynamoDbException` 執行個體。但通常你需要在不同的情況下不同的行為。與處理節流相比，處理條件檢查失敗的邏輯會有所不同。定義您要處理的特殊路徑，並確保測試替代路徑。這可協助您確保您能夠處理所有相關案例。

會 [使用 DynamoDB 時發生錯誤](#) 列出您可能會遇到的一些常見錯誤。您也可以參考 [常見錯誤](#) 以查看常見錯誤清單。對於特定的 API 調用，您還可以找到可能的確切錯誤及其文檔，例如 [查詢 API](#)。如需有關處理例外狀況的詳細資訊，請參閱 [AWS SDK for Java 2.x](#)。

AWS 請求識別碼

每個請求都包含一個請求 ID，如果您正在使用診斷問題，則可 [AWS Support](#) 以提取該請求 ID。衍生自的每個例外狀況都 `SdkServiceException` 有 `getRequestId()` 可用來擷取 `requestId` 識別碼的方法。

日誌

使用 SDK 提供的記錄對於捕獲客戶端庫中的任何重要消息以及更深入的調試目的都很有用。記錄器是分層的，SDK 用 `software.amazon.awssdk` 作其根記錄器。圖層可以使用 `TRACE`、`DEBUG`、`INFO`、`WARN`、`ERROR`、`ALL`、或之一來設定 `OFF`。配置的級別將應用於該記錄器並向下到記錄器層次結構。

`AWS SDK for Java 2.x` 利用 `Java (SLF4J)` 的簡單日誌外觀進行日誌記錄，該外觀充當其他記錄器周圍的抽象層。這使您可以插入您喜歡的記錄器。如需有關插入記錄儀的指示，請參閱 [SLF4J 使用者手冊](#)。

每個記錄器都有特定的行為。`Log4j` 的 `2.x` 記錄器默認創建一個 `ConsoleAppender` 個將日誌事件附加到 `System.out` 和默認日誌級別。 `ERROR`

默認情況下，SLF4J 輸出中包含的 SimpleLogger 記錄器 `System.err`，默認為 INFO 日誌級別。

建議您針對任何生產部署將層級設定為 `WARN` `software.amazon.awssdk` 為，以便從 SDK 的用戶端程式庫 `catch` 取任何重要訊息，同時限制輸出數量。

如果 SLF4J 在類路徑上找不到受支持的記錄器（沒有 SLF4J 綁定），SLF4J 將默認為 [無操作實現](#)。此實現會記錄消息以 `System.err` 解釋 SLF4J 無法在類路徑上找到記錄器實現。為了防止這種情況，你需要添加一個記錄器實現。要做到這一點，你可以在你的 Apache Maven `pom.xml` 添加一個依賴關係的工件，如 `org.slf4j.slf4j-simple` 或 `org.apache.logging.log4j.log4j-slf4j2-imp`。

的文件 `AWS SDK for Java 2.x` 說明如何在 SDK 中設定記錄，包括將記錄相依性新增至您的應用程式設定。如需詳細資訊，請參閱 [使用適用於 Java 2.x 的 SDK 進行記錄](#)。

該 `Log4j2.xml` 文件中的以下配置顯示了如何調整日誌記錄行為，如果您使用的是 Apache Log4j 2 記錄器。此配置將根記錄器級別設置為 `WARN`。此日誌級別將由包括記錄器在內的層次結構中的所有 `software.amazon.awssdk` 記錄器繼承。

默認情況下，輸出將轉到 `System.out`。在下面的例子中，我們仍然覆蓋默認輸出 Log4j 的追加程序來應用量身定制的 Log4j。PatternLayout

Log4j2.xml 組態檔案的範例

下列組態會將 `ERROR` 和層 `WARN` 級的訊息記錄至所有記錄器階層的主控制台。

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
    <Root level="WARN">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

AWS 請求識別碼記錄

當出現問題時，您可以 RequestIds 在異常中找到。但是，如果您希望 RequestIds for 未生成異常的請求，則可以使用日誌記錄。

請求 ID 由記錄器在DEBUG水平輸出。software.amazon.awssdk.request下列範例會延伸前一[configuration example](#)個項目ERROR，以保持根記錄器層級WARN、software.amazon.awssdk at 層級和 software.amazon.awssdk.request at 層級DEBUG。設定這些層級有助於 catch 取要求 ID 和其他要求相關詳細資料，例如端點和狀態碼。

```
<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>
```

以下為日誌輸出的範例：

```
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
```

分頁

某些要求 (例如「[查詢](#)和[掃描](#)」) 會限制單一要求傳回的資料大小，並要求您重複提出要求以提取後續頁面。

您可以使用參數控制每個頁面要讀取的最大項目Limit數。例如，您可以使用Limit參數只擷取最後 10 個項目。此限制是在套用任何篩選之前，應從表格中讀取多少項目。沒有辦法在過濾後指定你想要的 10。您只能控制預先過濾的計數，並在實際檢索到 10 個項目時檢查客戶端。無論限制如何，每個響應始終具有 1 MB 的最大大小。

在 API 回應中LastEvaluatedKey可能包含 a，表示回應因為達到計數限制或大小限制而結束。此索引鍵是針對該回應評估的最後一個索引鍵。直接與 API 交互，您可以檢索它LastEvaluatedKey並將其傳遞給後續調用，以ExclusiveStartKey便從該起點讀取下一個塊。如果傳回 noLastEvaluatedKey，表示沒有其他符合Query或 Scan API 呼叫的項目。

下列範例使用低階介面，根據keyConditionExpression參數將項目限制為 100。

```
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());
}
```

透過提供 AWS SDK for Java 2.x 可進行多個服務呼叫的自動分頁方法，以自動為您取得下一頁結果，藉此簡化與 DynamoDB 的互動。這簡化了您的代碼，但確實消除了對資源使用情況的一些控制，您可以通過手動閱讀頁面來保持這些控制。

透過使用 DynamoDB 用戶端中可用的Iterable方法 (例如[QueryPaginator](#)和) [ScanPaginator](#)，SDK 會處理分頁。這些方法的返回類型是一個自定義迭代，您可以使用它來遍歷所有頁面。SDK 會在內部為您處理服務呼叫。使用 Java 流 API，你可以處理的結果顯QueryPaginator示在下面的例子。


```
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();
```

資料類別註解

Java SDK 提供了幾個註釋，你可以把你的數據類的屬性，這將影響 SDK 如何與他們交互。藉由新增註解，您可以讓屬性表現為隱含的原子計數器、維護自動產生的時間戳記值，或追蹤項目版本號碼。如需詳細資訊，請參閱 < [資料類別註解](#) >。

使用資料表、項目、查詢、掃描和索引

本節提供在 Amazon DynamoDB 中使用資料表、項目、查詢和更多內容的詳細資訊。

主題

- [在 DynamoDB 中使用資料表和資料](#)
- [全域資料表：DynamoDB 的多區域複寫](#)
- [使用讀取和寫入操作](#)
- [使用次要索引來改善資料存取](#)
- [管理 DynamoDB 交易的複雜工作流程](#)
- [使用 Amazon DynamoDB 變更資料擷取](#)
- [使用 DynamoDB 的隨需備份與還原](#)
- [適用於 DynamoDB 的 Point-in-time 復原](#)

在 DynamoDB 中使用資料表和資料

本節說明如何使用 AWS Command Line Interface (AWS CLI) 和 AWS 開發套件在 Amazon DynamoDB 中建立、更新和刪除表格。

Note

您也可以使用 AWS Management Console 來執行這些相同的任務。如需詳細資訊，請參閱 [使用主控台](#)。

本節也會使用 DynamoDB Auto Scaling 或手動設定佈建輸送量來提供輸送容量的詳細資訊。

主題

- [DynamoDB 資料表上的基本操作](#)
- [選擇資料表類別時的注意事項](#)
- [DynamoDB 項目大小和格式](#)
- [將標籤新增至資源](#)
- [在 Java 中使用 DynamoDB 資料表](#)
- [在 .NET 中處理 DynamoDB 資料表](#)

DynamoDB 資料表上的基本操作

與其他資料庫系統類似，Amazon DynamoDB 會將資料存放在資料表中。您可使用幾個基本操作來管理您的資料表。

主題

- [建立資料表](#)
- [說明資料表](#)
- [更新資料表](#)
- [刪除資料表](#)
- [使用刪除保護](#)
- [列出資料表名稱](#)
- [說明佈建輸送量配額](#)

建立資料表

使用 CreateTable 操作在 Amazon DynamoDB 中建立資料表。若要建立資料表，您必須提供以下資訊：

- 資料表名稱 名稱必須符合 DynamoDB 命名規則，且對於目前 AWS 帳戶和區域而言必須是唯一的。例如，您可以在美國東部 (維吉尼亞北部) 建立 People 資料表，在歐洲 (愛爾蘭) 建立另一個 People 資料表。不過，這兩個資料表彼此必須完全不同。如需詳細資訊，請參閱 [Amazon DynamoDB 中支援的資料類型和命名規則](#)。
- 主索引鍵。主索引鍵可以包含一個屬性 (分割區索引鍵) 或兩個屬性 (分割區索引鍵和排序索引鍵)。您需要提供屬性名稱、資料類型和每個屬性的角色：HASH (適用於分割區索引鍵) 和 RANGE (適用於排序索引鍵)。如需詳細資訊，請參閱 [主索引鍵](#)。
- 輸送量設定 (針對佈建的資料表)。如果使用佈建的模式，您必須指定資料表的讀取和寫入初始輸送量設定。您稍後可以修改這些設定，或讓 DynamoDB Auto Scaling 功能來管理設定。如需詳細資訊，請參閱 [佈建容量模式](#) 及 [使用 DynamoDB Auto Scaling 功能自動管理輸送容量](#)。

範例 1：建立佈建的資料表

下面的 AWS CLI 例子演示了如何創建一個 table (Music)。主索引鍵包含 Artist (分割區索引鍵) 和 SongTitle (排序索引鍵)，而且它們每一個的資料類型都是 String。此資料表的輸送上限是 10 個讀取容量單位和 5 個寫入容量單位。

```
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
```

CreateTable 操作會傳回資料表的中繼資料，如下所示。

```
{  
  "TableDescription": {  
    "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",  
    "AttributeDefinitions": [  
      {  
        "AttributeName": "Artist",  
        "AttributeType": "S"  
      },  
      {  
        "AttributeName": "SongTitle",  
        "AttributeType": "S"  
      }  
    ],  
    "ProvisionedThroughput": {  
      "NumberOfDecreasesToday": 0,  
      "WriteCapacityUnits": 5,  
      "ReadCapacityUnits": 10  
    },  
    "TableSizeBytes": 0,  
    "TableName": "Music",  
    "TableStatus": "CREATING",  
    "TableId": "12345678-0123-4567-a123-abcdefghijkl",  
    "KeySchema": [  
      {  
        "KeyType": "HASH",  
        "AttributeName": "Artist"  
      },  
      {  
        "KeyType": "RANGE",  
        "AttributeName": "SongTitle"  
      }  
    ]  
  }  
}
```

```
    }
  ],
  "ItemCount": 0,
  "CreationDateTime": 1542397215.37
}
}
```

TableStatus 元素指出資料表的目前狀態 (CREATING)。根據您指定的 ReadCapacityUnits 和 WriteCapacityUnits 值，可能需要一些時間才能建立資料表。這些項目的較大值需要 DynamoDB 為資料表配置更多資源。

範例 2：建立隨需資料表

使用隨需模式來建立相同的資料表 Music。

```
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
```

CreateTable 操作會傳回資料表的中繼資料，如下所示。

```
{
  "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

呼叫隨需資料表上的 DescribeTable 時，讀取容量單位與寫入容量單位設為 0。

範例 3：使用 DynamoDB 標準的不常存取資料表類別建立資料表

若要建立相同的 Music 資料表，使用 DynamoDB 標準–不常存取資料表類別。

```
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
```

```
ReadCapacityUnits=10,WriteCapacityUnits=5 \  
--table-class STANDARD_INFREQUENT_ACCESS
```

CreateTable 操作會傳回資料表的中繼資料，如下所示。

```
{  
  "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  
  }  
}
```

```
}  
}
```

說明資料表

若要檢視資料表的詳細資訊，請使用 `DescribeTable` 操作。您必須提供資料表名稱。 `DescribeTable` 中輸出的格式與 `CreateTable` 相同。它包含建立資料表的時間戳記、其索引鍵結構描述、其佈建的輸送量設定、其預估大小以及任何現有的次要索引。

Important

呼叫隨需資料表上的 `DescribeTable` 時，讀取容量單位與寫入容量單位設為 0。

Example

```
aws dynamodb describe-table --table-name Music
```

`TableStatus` 從 `CREATING` 變更為 `ACTIVE` 之後，就表示資料表準備就緒可供使用。

Note

如果您在 `CreateTable` 請求之後立即發出 `DescribeTable` 請求，則 DynamoDB 可能會傳回錯誤 (`ResourceNotFoundException`)。原因是 `DescribeTable` 使用最終一致查詢，而且您資料表的中繼資料目前可能無法使用。請等待幾秒，然後再次重試 `DescribeTable` 請求。

為計算費用，您的 DynamoDB 儲存費用包含 100 個位元組的個別項目成本。(如需詳細資訊，請前往 [DynamoDB 定價](#)。) 個別項目的這個額外 100 個位元組不會用於容量單位計算，或不供 `DescribeTable` 操作使用。

更新資料表

`UpdateTable` 操作可讓您執行下列其中一項：

- 修改資料表的佈建輸送量設定 (針對佈建模式的資料表)。
- 變更資料表的讀取/寫入容量模式。
- 在資料表中修改全域次要索引 (請參閱 [在 DynamoDB 中使用全域次要索引](#))。
- 啟用或停用資料表上的 DynamoDB Streams (請參閱 [DynamoDB Streams 的變更資料擷取](#))。

Example

下列 AWS CLI 範例顯示如何修改表格的佈建輸送量設定。

```
aws dynamodb update-table --table-name Music \  
  --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10
```

Note

當您發出 UpdateTable 請求時，資料表的狀態會從 AVAILABLE 變更為 UPDATING。資料表在處於 UPDATING 時仍然完全可供使用。此程序完成時，資料表狀態會從 UPDATING 變更為 AVAILABLE。

Example

下列 AWS CLI 範例顯示如何將資料表的讀取/寫入容量模式修改為隨選模式。

```
aws dynamodb update-table --table-name Music \  
  --billing-mode PAY_PER_REQUEST
```

刪除資料表

您可以使用 DeleteTable 操作來移除未使用的資料表。刪除資料表是無法復原的操作。

Example

下面的 AWS CLI 例子演示了如何刪除一個表。

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

當您發出 DeleteTable 請求時，資料表的狀態會從 ACTIVE 變更為 DELETING。根據使用的資源 (例如資料表中所存放的資料，以及資料表上的任何串流或索引)，可能需要一些時間才能刪除資料表。

DeleteTable 操作結束時，資料表就不再存在於 DynamoDB 中。

使用刪除保護

您可以使用刪除保護屬性，來保護表格免於遭到意外刪除。為資料表啟用此屬性有助於確保管理員在常態資料表管理操作期間，不會意外刪除資料表。這將有助於防止您的正常業務營運中斷。

資料表擁有者或授權管理員可控制每個資料表的刪除保護內容。每個資料表的刪除保護屬性預設是關閉的。這包括全域複本，以及從備份還原的資料表。停用資料表的刪除保護時，任何獲得 Identity and Access Management (IAM) 政策授權的使用者都可以刪除該資料表。資料表啟用刪除保護時，沒有人可將其刪除。

若要變更此設定，請移至資料表的其他設定，瀏覽至刪除保護面板，然後選取啟用刪除保護。

刪除保護屬性受到 DynamoDB 主控台、API、CLI/SDK 和 AWS CloudFormation 的支援。CreateTable API 支援資料表建立時的刪除保護屬性，而 UpdateTable API 支援變更現有資料表的刪除保護屬性。

Note

- 如果 AWS 帳戶遭到刪除，則該帳戶的所有資料 (包括表格) 仍會在 90 天內刪除。
- 如果 DynamoDB 喪失用來加密資料表的客戶受管金鑰存取權，它仍會封存資料表。存檔涉及製作資料表的備份和刪除原始資料表。

列出資料表名稱

此 ListTables 作業會傳回目前 AWS 帳戶和區域的 DynamoDB 表格名稱。

Example

下列 AWS CLI 範例顯示如何列出 DynamoDB 資料表名稱。

```
aws dynamodb list-tables
```

說明佈建輸送量配額

此 DescribeLimits 作業會傳回目前 AWS 帳戶和區域的目前讀取和寫入容量配額。

Example

下列 AWS CLI 範例顯示如何描述目前佈建的輸送量配額。

```
aws dynamodb describe-limits
```

輸出顯示目前 AWS 帳戶和區域的讀取和寫入容量單位的上限配額。

如需這些配額以及如何請求提高配額的詳細資訊，請參閱 [輸送量預設配額](#)。

選擇資料表類別時的注意事項

DynamoDB 提供兩種資料表類別，旨在協助您最佳化成本。預設值為 DynamoDB 標準資料表類別，建議大多數工作負載使用。DynamoDB 標準-不常存取 (DynamoDB 標準-IA) 資料表類別針對以儲存為主要成本的資料表進行最佳化。例如，儲存不常存取資料的資料表 (例如應用程式日誌、舊社交媒體貼文、電子商務訂單歷史記錄以及過去遊戲成就) 都是適合標準-IA 資料表類別的選項。

每個 DynamoDB 資料表都與資料表類別相關聯。與資料表相關聯的所有次要索引都使用相同的資料表類別。您可以在建立資料表時設定資料表類別 (依預設為 DynamoDB 標準)，並使用 AWS Management Console、AWS CLI 或 AWS SDK 更新現有資料表的資料表類別。DynamoDB 也支援使用單一區域表 (非全域資料表 AWS CloudFormation 的資料表) 來管理您的資料表類別。每個資料表類別針對資料儲存以及讀取和寫入要求提供不同的定價。當為您的表選擇一個表類時，請記住下列事項：

- DynamoDB 標準資料表類別提供比 DynamoDB 標準-IA 更低的輸送量成本，對於輸送量是主要成本的資料表來說，是最具成本效益的選項。
- DynamoDB 標準 IA 資料表類別提供比 DynamoDB 標準更低的儲存成本，對於儲存成為主要成本的資料表來說，是最具成本效益的選項。當儲存超過使用 DynamoDB 標準資料表類別的資料表輸送量 (讀取和寫入) 成本的 50% 時，DynamoDB 標準-IA 資料表類別可協助您降低資料表總成本。
- DynamoDB 標準 – IA 資料表提供與 DynamoDB 標準資料表相同的效能、耐用性和可用性。
- 在 DynamoDB 標準資料表與 DynamoDB 標準 – IA 資料表類之間切換不需要變更應用程式的程式碼。不論您的資料表使用何種資料表類型，都可以使用相同的 DynamoDB API 和服務端點。
- DynamoDB 標準 — IA 表與所有現有 DynamoDB 功能相容，例如 auto 擴展、隨需模式 time-to-live (TTL)、隨需備份、point-in-time 復原 (PITR) 和全域次要索引。

資料表最具成本效益的資料表類別取決於資料表預期的儲存體和輸送量使用模式。您可以使用成本和使用量報告和 Cost Explorer 查看表格的 AWS 歷史儲存體和輸送量 AWS 成本和使用量。使用此歷史資料來確定資料表最具成本效益的資料表類別。若要深入了解如何使用 AWS 成本和用量報告以及 AWS Cost Explorer，請參閱 [B AWS Billing and Cost Management 文件](#)。如需資料表類別定價詳細資訊，請參閱 [Amazon DynamoDB 定價](#)。

Note

資料表類別更新是一個背景流程。您仍然可以在資料表類別更新期間正常存取資料表。更新表類的時間取決於您的表流量，儲存大小和其他相關變量。在 30 天的追蹤期間內，資料表上不允許兩個以上的資料表類別更新。

DynamoDB 項目大小和格式

除了主索引鍵之外，DynamoDB 資料表不具結構描述，因此資料表中的項目全部會有不同的屬性、大小和資料類型。

項目的總大小是其屬性名稱和值長度的總和，加上下列任何適用的管理費用。您可以使用下列準則來預估屬性大小：

- 字串是 UTF-8 二進位編碼的 Unicode。字符串的大小是 (屬性名稱的 UTF-8 編碼字節數) + (UTF-8 編碼的字節數)。
- 數字的長度會不同，最多 38 個有意義位數。前後的零會截去。數字的大小約為 (屬性名稱的 UTF-8 編碼字節數) + (每兩個有效數字 1 個字節) + (1 個字節)。
- 必須先以 base64 格式編碼二進位值，才能將它傳送至 DynamoDB，但使用值的原始位元組長度來計算大小。二進制屬性的大小是 (屬性名稱的 UTF-8 編碼字節數) + (原始字節數)。
- null 屬性或布爾屬性的大小為 (屬性名稱的 UTF-8 編碼字節數) + (1 個字節)。
- 不論內容為何，類型為 List 或 Map 的屬性都需要 3 位元組的額外負荷。a List 或的大小 Map 為 (屬性名稱的 UTF-8 編碼字節數) + 總和 (嵌套元素的大小) + (3 個字節)。空 List 或的大小 Map 為 (屬性名稱的 UTF-8 編碼字節數) + (3 個字節)。
- 每個 List 或 Map 元素也需要 1 位元組的額外負荷。

Note

建議您選擇較短的屬性名稱，而不是較長的屬性名稱。如此有助您減少所需的儲存量，同時降低您的 RCU/WCU 用量。

基於儲存體計費目的，每個項目會包含每個項目的儲存體額外負荷，這取決於您啟用的功能。

- DynamoDB 中的所有項目都需要 100 個位元組的儲存額外負荷來進行索引。
- 某些 DynamoDB 功能 (使用 DynamoDB 進行 Kinesis Data Streams 的全域資料表、交易、變更資料擷取) 需要額外的儲存額外負荷來考量啟用這些功能所產生的系統建立屬性。例如，全域資料表需要額外 48 個位元組的儲存體額外負荷。

將標籤新增至資源

您可以使用 tags 來標示 Amazon DynamoDB 資源。標籤可讓您以不同的方式分類您的資源，例如依據目的、所有者、環境或其他條件。標籤可協助您執行以下作業：

- 根據您指派給資源的標籤來快速識別資源。
- 請查看依標籤細分的 AWS 帳單。

Note

任何與標記資料表相關的本機次要索引 (LSI) 和全域次要索引 (GSI) 都會自動標上相同的標籤。目前無法標記 DynamoDB Streams 使用量。

Amazon EC2、Amazon S3、DynamoDB 等 AWS 服務支援標記。有效標記可讓您跨帶有特定標籤的服務來建立報告，以提供成本深入資訊。

若要開始使用標記，請執行以下操作：

1. 了解 [DynamoDB 中的標記限制](#)。
2. 使用 [在 DynamoDB 中標記資源](#) 建立標籤。
3. 使用 [成本分配報告](#) 追蹤每個作用中標籤的 AWS 成本。

最後，最好遵循最佳標記策略。如需相關資訊，請參閱 [AWS 標記策略](#)。

DynamoDB 中的標記限制

每個標籤皆包含由您定義的索引鍵和值。將適用以下限制：

- 每個 DynamoDB 資料表都只能有一個具有相同索引鍵的標籤。若您嘗試新增現有的標籤 (相同索引鍵)，現有標籤的值會更新為新的值。
- 標籤鍵與值皆區分大小寫。
- 鍵長度上限為 128 個 Unicode 字元。
- 值長度上限為 256 個 Unicode 字元。
- 允許的字元為字母、空格和數字，以及下列特殊字元：`+ - = . _ : /`
- 每一資源標籤數最多為 50。

- AWS 指派的標籤名稱和值會自動指派 `aws:` 字首，您無法指派此值。AWS 指派的標籤名稱不會列入 50 個標籤限制。使用者指派的標籤名稱在成本分配報告中具有字首 `user:`。
- 標籤的套用不可回溯。

在 DynamoDB 中標記資源

若要新增、列出、編輯或刪除標籤，您可以使用 Amazon DynamoDB 主控台或 AWS Command Line Interface (AWS CLI)。然後，您可以啟動這些使用者定義的標籤，讓它們出現在 AWS Billing and Cost Management 主控台中，以便追蹤成本配置。如需詳細資訊，請參閱 [成本分配報告](#)。

您也可以使用 AWS Management Console 上的標籤編輯器進行大量編輯。如需詳細資訊，請參閱 [使用標籤編輯器](#)。

若要改用 DynamoDB API，請參閱 [《Amazon DynamoDB API 參考》](#) 中的下列操作：

- [TagResource](#)
- [UntagResource](#)
- [ListTagsOfResource](#)

主題

- [設定依標籤篩選的許可](#)
- [將標籤新增至新的或現有資料表 \(AWS Management Console\)](#)
- [將標籤新增至新的或現有資料表 \(AWS CLI\)](#)

設定依標籤篩選的許可

若要使用標籤篩選 DynamoDB 主控台內的資料表清單，請確定使用者的政策包含對下列操作的存取權：

- `tag:GetTagKeys`
- `tag:GetTagValues`

您可以依照下列步驟，將新的 IAM 政策連接至使用者，以便存取這些操作。

1. 使用管理員使用者前往 [IAM 主控台](#)。
2. 在左導覽選單中，選取 Policies (政策)。

3. 選取 Create Policy (建立政策)。
4. 將下列政策貼入至 JSON 編輯器。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "tag:GetTagKeys",
        "tag:GetTagValues"
      ],
      "Resource": "*"
    }
  ]
}
```

5. 完成協助程式並指派政策的名稱 (例如 TagKeysAndValuesReadAccess)。
6. 在左側導覽選單中，選擇 Users (使用者)。
7. 從清單中選取您通常用於存取 DynamoDB 主控台的使用者。
8. 選取 Add permissions (新增許可)。
9. 選取 Attach existing policies directly (直接連接現有政策)。
10. 從清單中選取您先前建立的政策。
11. 完成協助程式。

將標籤新增至新的或現有資料表 (AWS Management Console)

您可以使用 DynamoDB 主控台，在建立新資料表時在其中新增標籤，或為現有的資料表新增、編輯或刪除標籤。

在建立時為資源加上標籤 (主控台)

1. 登入 AWS Management Console，並在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在導覽窗格中，選擇 Tables (資料表)，然後選擇 Create table (建立資料表)。
3. 在 Create DynamoDB table (建立 DynamoDB 資料表) 頁面上，提供名稱和主索引鍵。在 Tags (標籤) 區段中，選擇 Add new tag (新增標籤)，然後輸入您要使用的標籤。

如需標籤結構的相關資訊，請參閱 [DynamoDB 中的標記限制](#)。

如需建立資料表的詳細資訊，請參閱 [DynamoDB 資料表上的基本操作](#)。

為現有的資源加上標籤 (主控台)

請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。

1. 在導覽窗格中，選擇 Tables (資料表)。
2. 在清單中選擇資料表，然後選擇 Additional settings (其他設定) 索引標籤。您可以新增、編輯或刪除 Tags (標籤) 頁面底部的區段。

將標籤新增至新的或現有資料表 (AWS CLI)

以下範例顯示如何在建立資料表和索引時，使用 AWS CLI 來指定標籤和為現有資源加上標籤。

在建立時為資源加上標籤 (AWS CLI)

- 以下範例會建立新 Movies 資料表並使用 blueTeam 值來新增 Owner 標籤：

```
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
```

為現有的資源加上標籤 (AWS CLI)

- 以下範例會為 Movies 資料表新增 blueTeam 值的 Owner 標籤：

```
aws dynamodb tag-resource \  
  --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies \  
  --tags Key=Owner,Value=blueTeam
```

列出資料表的所有標籤 (AWS CLI)

- 以下範例列出與 Movies 資料表相關聯的所有標籤：


```
aws dynamodb list-tags-of-resource \  
  --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies
```

成本分配報告

AWS 會使用標籤來組織成本分配報告上的資源成本。AWS 提供兩種成本分配標籤：

- AWS 產生的標籤。AWS 為您定義、建立和套用此標籤。
- 使用者定義的標籤。您可以定義、建立和套用這些標籤。

您必須分別啟用這兩種標籤，它們才會顯示在 Cost Explorer 或成本分配報告中。

啟用 AWS 產生的標籤：

1. 登入 AWS Management Console，並於 <https://console.aws.amazon.com/billing/home#/> 開啟「帳單和成本管理」主控台。
2. 在導覽窗格中，選擇 Cost Allocation Tags (成本分配標籤)。
3. 在 AWS-Generated Cost Allocation Tags (AWS 產生的成本分配標籤) 下，選擇 Activate (啟用)。

若要啟用使用者定義的標籤：

1. 登入 AWS Management Console，並於 <https://console.aws.amazon.com/billing/home#/> 開啟「帳單和成本管理」主控台。
2. 在導覽窗格中，選擇 Cost Allocation Tags (成本分配標籤)。
3. 在 User-Generated Cost Allocation Tags (使用者產生的成本分配標籤) 下，選擇 Activate (啟用)。

建立並啟用標籤後，AWS 會產生成本分配報告，內含按作用中標籤分組的用量與成本。成本分配報告包含每個計費期間的所有 AWS 成本。該報告同時包含有標籤和沒標籤的資源，以便您可清楚地整理資源的費用。

Note

目前，從 DynamoDB 傳出的任何資料都不會在成本分配報告上依標籤細分。

如需詳細資訊，請參閱[使用成本分配標籤](#)。

在 Java 中使用 DynamoDB 資料表

您可以使用 AWS SDK for Java 建立、更新及刪除 Amazon DynamoDB 資料表；列出您帳戶中的所有資料表；或是取得特定資料表的相關資訊。

主題

- [建立資料表](#)
- [更新資料表](#)
- [刪除資料表](#)
- [列出資料表](#)
- [範例：使用 AWS SDK for Java Document API 建立、更新、刪除及列出資料表](#)

建立資料表

若要建立資料表，您必須提供資料表名稱、其主索引鍵，以及佈建的輸送量數值。下列程式碼片段使用數值類型的屬性 ID 做為主索引鍵，建立一份範例資料表。

使用 AWS SDK for Java API 建立資料表

1. 建立 DynamoDB 類別的執行個體。
2. 將 `CreateTableRequest` 執行個體化，以提供要求資訊。

您必須提供資料表名稱、屬性定義、索引鍵結構描述，以及佈建的輸送量值。

3. 以參數形式提供請求物件，以便執行 `createTable` 方法。

下列程式碼範例示範前述步驟。

```
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();
```

資料表必須由 DynamoDB 建立，並將其狀態設定為 ACTIVE (作用中)，才可供使用。createTable 要求會傳回 Table 物件，您可用來取得資料表的詳細資訊。

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());
```

您可以呼叫用戶端的 describe 方法，隨時取得資料表資訊。

Example

```
TableDescription tableDescription = dynamoDB.getTable(tableName).describe();
```

更新資料表

您只可更新為現有資料表佈建的輸送量數值。視您的應用程式需求而定，您可能需要更新這些值。

Note

關於輸送量每日的增加和減少，詳細資訊請參閱[Amazon DynamoDB 中的服務、帳戶和資料表配額](#)。

使用AWS SDK for Java API 更新資料表

1. 建立 Table 類別的執行個體。
2. 建立 ProvisionedThroughput 類別的執行個體，提供新的輸送量值。
3. 以參數形式提供 ProvisionedThroughput 執行個體，以便執行 updateTable 方法。

下列程式碼範例示範前述步驟。

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();
```

刪除資料表

使用AWS SDK for Java API 刪除資料表

1. 建立 Table 類別的執行個體。
2. 建立 DeleteTableRequest 類別的執行個體，並提供您要刪除的資料表名稱。
3. 以參數形式提供 Table 執行個體，以便執行 deleteTable 方法。

下列程式碼範例示範前述步驟。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

table.delete();

table.waitForDelete();
```

列出資料表

若要列出帳戶中的資料表，請建立 DynamoDB 的執行個體，然後執行 `listTables` 方法。[ListTables](#) 操作不需要任何參數。

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());
}
```

範例：使用 AWS SDK for Java Document API 建立、更新、刪除及列出資料表

下列程式碼範例使用 AWS SDK for Java 文件 API 來建立、更新及刪除 Amazon DynamoDB 資料表 (ExampleTable)。資料表更新時，也會同時加大佈建的輸送量數值。此範例同時也會列出您帳戶中所有的資料表，以及取得特定資料表的說明。如需執行下列範例的 step-by-step 指示，請參閱[Java 程式碼範例](#)。

```
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());
        }
    }
}
```


在 .NET 中處理 DynamoDB 資料表

您可以使用 AWS SDK for .NET 建立、更新及刪除資料表；列出您帳戶中的所有資料表；或是取得特定資料表的相關資訊。

下列是使用 AWS SDK for .NET 執行 Amazon DynamoDB 資料表操作的一般步驟。

1. 建立 `AmazonDynamoDBClient` 類別的執行個體 (用戶端)。
2. 透過建立對應的請求物件，為操作提供必要及選用的參數。

例如，建立 `CreateTableRequest` 物件來建立資料表，以及建立 `UpdateTableRequest` 物件來更新現有的資料表。

3. 執行您在前一步驟中建立之用戶端所提供的適當方法。

Note

本節中的範例不使用 .NET Core，因為它不支援同步方法。如需詳細資訊，請參閱[適用於 .NET 的 AWS 非同步 API](#)。

主題

- [建立資料表](#)
- [更新資料表](#)
- [刪除資料表](#)
- [列出資料表](#)
- [範例：使用 AWS SDK for .NET 低階 API 建立、更新、刪除及列出資料表](#)

建立資料表

若要建立資料表，您必須提供資料表名稱、其主索引鍵，以及佈建的輸送量數值。

使用 AWS SDK for .NET 低階 API 建立資料表

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。
2. 建立 `CreateTableRequest` 類別的執行個體，以提供請求資訊。

您必須提供資料表名稱、主索引鍵，以及佈建的輸送量數值。

3. 以參數形式提供請求物件，以便執行 `AmazonDynamoDBClient.CreateTable` 方法。

下列 C# 範例示範前述步驟。此範例會建立使用 `Id` 做為主索引鍵的資料表 (`ProductCatalog`) 以及一組佈建輸送量值。您可以視您應用程式的需求，使用 `UpdateTable` API 更新佈建的輸送量數值。

```
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);
```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 `ACTIVE`。`CreateTable` 回應包含提供必要資料表資訊的 `TableDescription` 屬性。

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);
```

您也可以呼叫用戶端的 DescribeTable 方法，隨時取得資料表資訊。

Example

```
var res = client.DescribeTable(new DescribeTableRequest{TableName = "ProductCatalog"});
```

更新資料表

您只可更新為現有資料表佈建的輸送量數值。視您的應用程式需求而定，您可能需要更新這些值。

Note

您可以根據需要隨時增加傳輸量，並在一定的限制範圍內降低傳輸量。關於輸送量每日的增加和減少，詳細資訊請參閱[Amazon DynamoDB 中的服務、帳戶和資料表配額](#)。

使用AWS SDK for .NET低階 API 更新資料表

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 建立 UpdateTableRequest 類別的執行個體，以提供請求資訊。

您必須提供資料表名稱與新的佈建輸送量值。

3. 以參數形式提供請求物件，以便執行 AmazonDynamoDBClient.UpdateTable 方法。

下列 C# 範例示範前述步驟。

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);
```

刪除資料表

下列步驟會使用 .NET 低階 API 刪除資料表。

使用 AWS SDK for .NET 低階 API 刪除資料表

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。
2. 建立 `DeleteTableRequest` 類別的執行個體，並提供您要刪除的資料表名稱。
3. 以參數形式提供請求物件，以便執行 `AmazonDynamoDBClient.DeleteTable` 方法。

下列 C# 程式碼範例示範前述步驟。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ExampleTable";

var request = new DeleteTableRequest{ TableName = tableName };
var response = client.DeleteTable(request);
```

列出資料表

若要使用 AWS SDK for .NET 低階 API 列出您帳戶中的資料表，請建立 `AmazonDynamoDBClient` 的執行個體，然後執行 `ListTables` 方法。

[ListTables](#) 操作不需要任何參數。但是您可以指定選用參數。例如，如果您想要使用分頁來限制每頁的資料表名稱數目，您可以設定 `Limit` 參數。這需要您建立 `ListTablesRequest` 物件並提供選用參數，如下列 C# 程式碼範例所示。除了頁面大小外，請求還會設定 `ExclusiveStartTableName` 參

數。一開始，ExclusiveStartTableName 是 null。但是，在您擷取第一頁結果後，若要擷取下一頁結果，您必須將此參數值設定為目前結果的 LastEvaluatedTableName 屬性。

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);
```

範例：使用 AWS SDK for .NET 低階 API 建立、更新、刪除及列出資料表

下列 C# 範例會建立、更新及刪除資料表 (ExampleTable)。它也會列出您帳戶中的所有資料表，及取得特定資料表的描述。資料表更新會增加佈建的輸送量數值。如需測試下列範例的 step-by-step 指示，請參閱[.NET 程式碼範例](#)。

```
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");
}
}
```

全域資料表：DynamoDB 的多區域複寫

Amazon DynamoDB 全域資料表是全受管、多個區域及多個作用中的資料庫，為大幅擴展的全域應用程式提供快速且本地化的讀寫效能。

全域資料表提供全受管解決方案，用以部署多個區域和多個作用中資料庫，而不需要建置和維護您自己的複寫解決方案。您可以指定希望表格可供使用的 AWS 區域，DynamoDB 會將持續的資料變更傳播到所有這些區域。

使用全域資料表的特定優點包括：

- 在您選擇的區域中自動複製 DynamoDB 表 AWS
- 消除在區域之間複製資料的困難工作並解決更新衝突，讓您可以專注在應用程式的業務邏輯上。
- 即使在可能性較低整個區域皆隔離或降級的情況下，也能協助您的應用程式保持高可用性。

DynamoDB 全域資料表適用於使用者分散在全球的大規模應用程式。在這類環境中，使用者希望應用程式的效能可以很快。全域資料表為全球各 AWS 地區提供自動多重作用中複寫。他們可以讓您將低延遲的資料存取交付給使用者，無論他們身在何處。

以下影片將為您介紹全域資料表。

您可以在「AWS 管理主控台」或中設定全域表格 AWS CLI。全域資料表使用現有的 DynamoDB API，因此不需要變更應用程式。您只需為佈建的資源付費，無需前期成本或承諾。

[區域間複寫的全域資料表](#)

主題

- [跨區域使用全域資料表無縫複寫資料](#)
- [為您的全域資料表提供安全性和存取權 AWS KMS](#)
- [全域資料表：運作方式](#)
- [管理全域資料表的最佳實務和要求](#)
- [教學課程：建立全域資料表](#)
- [監控全域資料表](#)
- [對全域資料表使用 IAM](#)
- [判斷您正在使用的全域資料表版本](#)
- [將全域表格從舊版 \(2017.11.29\) 版本升級到目前版本](#)

跨區域使用全域資料表無縫複寫資料

假設您有分散在三個地理區域的龐大客戶群：美國東岸、美國西岸和西歐。這些客戶可以使用您的應用程式更新其描述檔資訊。在這個使用案例中，您必須在三個客戶所在的不同 AWS 區域中建立三個相同的 DynamoDB 資料表，且名為 CustomerProfiles。這三個資料表將相互完全分開，其中一個資料表的資料變更不會反映在其他資料表。若沒有受管的複寫解決方案，您可能必須撰寫程式碼以複寫資料變更。但是，執行這項作業耗時且費力。

您不需要自己撰寫程式碼，就能建立由三個區域特定的 CustomerProfiles 資料表組成的全域資料表。然後，DynamoDB 會自動在這些資料表之間複製資料變更，以便將對某個區域中的 CustomerProfiles 資料變更順暢地傳播到其他地區。此外，如果其中一個 AWS 區域暫時無法使用，您的客戶仍可在其他區域存取相同的 CustomerProfiles 資料。

Note

- 全域資料表 [全域表格版本 \(舊版\)](#) 的區域支援僅限於美國東部 (維吉尼亞北部)、美國東部 (俄亥俄)、美國西部 (加利佛尼亞北部)、美國西部 (奧勒岡)、歐洲 (愛爾蘭)、歐洲 (倫敦)、歐洲 (法蘭克福)、亞太區域 (新加坡)、亞太區域 (雪梨)、亞太區域 (東京) 和亞太區域 (首爾)。
- 交易操作僅在最初進行寫入的區域內提供原子性、一致性，隔離性和耐久性 (ACID) 保證。全域資料表不支援跨區域交易。例如，如果您在美國東部 (俄亥俄) 和美國西部 (奧勒岡) 區域有一個包含複本的全域表格，並在美國東部 (維吉尼亞北部) 區域執行 TransactWriteItems 作業，則在複寫變更時，您可能會在美國西部 (奧勒岡) 區域觀察部分完成的交易。只有當變更已在來源區域遞交的情況下，這些變更才會複寫至其他區域。
- 如果 [停用 AWS 區域](#)，DynamoDB 會在偵測到該區域無法存取的 20 小時後，從複寫群組中移除此複 AWS 本。此複本將不會被刪除，而且將會停止從該區域的複寫以及對該區域的複寫。
- 新增唯讀複本後，您必須等待 24 小時才能成功刪除來源資料表。若您在新增唯讀複本後的前 24 小時內嘗試刪除資料表，您將收到錯誤訊息：「無法刪除複本，因為它在過去 24 小時內做為新增到資料表中的新複本的來源區域」。
- 新增複本時，不會對來源區域造成效能影響。
- 當您變更複本的讀取和寫入容量時，新的寫入容量會反映到其他同步複本，但新的讀取容量則不會。

如需 AWS 區域可用性和定價的相關資訊，請參閱 [Amazon DynamoDB 定價](#)。

為您的全域資料表提供安全性和存取權 AWS KMS

- 您可以使[AWS 受管金鑰](#)用AWSServiceRoleForDynamoDBReplication服務連結角色對[客戶管理金鑰](#)或用於加密複本的服務連結角色，在全域表上執行 AWS KMS 作業。
- 如果用於加密複本的客戶受管金鑰無法存取，DynamoDB 會將此複本從複寫群組中移除。此複本將不會被刪除，而且將會在偵測到 KMS 金鑰為無法存取的 20 小時後，停止從該區域的複寫以及對該區域的複寫。
- 如果想要將用來加密複本列表的[客戶受管的金鑰](#)停用時，只有當金鑰不再用來加密複本列表時，您才必須執行這項操作。發出刪除複本列表的命令之後，您必須等待刪除操作完成，並讓全域資料表變成 Active，再停用金鑰。否則，可能會導致來往複本列表的資料複寫僅部分完成。
- 如果您要修改或刪除複本列表的 IAM 角色政策，則必須在複本列表處於 Active 狀態時這麼做。否則，建立、更新或刪除複本列表可能會失敗。
- 根據預設，系統會建立全域資料表，並停用刪除保護。即使全域資料表啟用刪除保護，該資料表的任何複本都會以刪除保護預設為停用開始。
- 當資料表停用刪除保護時，可能會意外刪除該資料表。資料表啟用刪除保護時，沒有人可將其刪除。
- 變更一個複本表的刪除保護設定，不會更新群組中的其他複本。

Note

[全域表格版本 \(舊版\)](#) 不支援客戶受管金鑰。如果您想要在 DynamoDB 全域表中使用客戶受管金鑰，則需要將表格升級至[全域表版本 2019.11.21 \(目前\)](#)，然後加以啟用。

全域資料表：運作方式

以下各節說明 Amazon DynamoDB 中全域資料表的概念和行為。

全域資料表概念

全域表是一或多個複本表格的集合，全部由單一 AWS 帳戶擁有。

複本列表 (簡稱複本) 是單一 DynamoDB 資料表，可作為全域資料表的一部分運作。每個複本會存放相同的資料項目集。任何特定全域資料表的每個 AWS 區域只能有一個複本列表。如需如何開始使用全域資料表的詳細資訊，請參閱 [教學課程：建立全域資料表](#)。

在建立 DynamoDB 全域資料表時，該資料表包含 DynamoDB 會將其視為單一單位的多個複本列表 (每個區域一個)。每個複本都有相同的資料表名稱和相同的主索引鍵結構描述。當應用程式將資料寫入一個區域中的複本表格時，DynamoDB 會自動將寫入傳播到其他區域中的其他複本表格。AWS

您可以將複本列表新增到全域資料表，以便在其他區域中使用該複本列表。

使用 2019.11.21 版 (目前)，當您在一個區域中建立全域次要索引時，它會自動複製到另一個區域並自動回填。

一般任務

全域資料表的一般任務如下。

您可以刪除全域資料表的複本資料表，方式與一般資料表相同。這將停止複寫到該區域，並刪除保留在該區域中的資料表複本。您不能斷開複製，並將表的副本作為獨立實體存在。您可以將全域表複製到該區域中的本機表格，然後刪除該區域的全域複本。

Note

在來源資料表用於啟動新區域後至少 24 小時之後，您才能刪除來源資料表。若您嘗試刪除，很快便會收到錯誤。

如果應用程式大約在同一時間更新不同區域中的相同項目，則可能會發生衝突。為了協助確保最終一致性，DynamoDB 全域資料表會在並行更新間使用「最後寫入者獲勝」方法，在並行更新間使用最後寫入者。所有的複本都會同意最新的更新，並朝它們都具有相同資料的狀態收斂。

Note

有幾種方式可以避免衝突，包括：

- 僅允許在一個區域中寫入資料表。
- 根據您的寫入策略將使用者流量路由到不同的區域，以確保沒有衝突。
- 避免使用非等冪更新，例如書籤 = 書籤 + 1，以支援靜態更新，例如書籤 = 25。
- 請記住，當您將寫入或讀取路由到一個 Region 時，這取決於您的應用程序，以確保強制執行流程。

監控全域資料表

您可以使用 CloudWatch 用觀察度量標準 `ReplicationLatency`。這會追蹤一個項目寫入至複本清單，到項目出現在全域資料表的另一個複本中的經過時間。延遲以毫秒為單位表示，並針對每個來源區域和目標區域成對發出。此指標會保留在來源區域中。這是全域表 v2 提供的唯一 CloudWatch 測量結果。

您將遇到的複寫延遲取決於您選擇 AWS 區域的距離以及其他變數。如果您的原始表格位於美國西部 (加利福尼亞北部) (us-west-1) 區域，則較近區域 (例如美國西部 (奧勒岡) (us-west-2) 區域) 中的複本相比，複寫延遲較遠，例如非洲 (開普敦) (af-south-1) 區域。

Note

複寫延遲不會影響 API 延遲。如果您在區域 A 中有用戶端和表格，並且在區域 B 中新增全域表格複本，則區域 A 中的用戶端和表格的延遲時間與新增區域 B 之前的延遲時間相同。如果您呼叫區域 B 中的 [PutItem](#) API 操作，最終可在區域 A 中讀取大約在 Amazon CloudWatch 中可用的 `ReplicationLatency` 統計資料。在複寫之前，您會收到一個空的回應，並在它被複製之後，您會收到該項目；這兩個呼叫都會有大致相同的 API 延遲。

存留時間 (TTL)

您可以使用存留時間 (TTL) 來指定屬性名稱，其值表示項目的到期時間。自 Unix epoch 開始以來，此值都以秒為單位。之後，DynamoDB 可刪除項目，而不會產生寫入成本。

有了全域資料表，您只要在一個區域設定 TTL，該設定便會自動複製到另一個區域。當透過 TTL 規則刪除項目時，該工作執行時不會耗用來源資料表上的寫入單位，但目標資料表會產生「複製寫入單位」成本。

請注意，如果來源和目標資料表的佈建寫入容量非常低，這可能會造成節流，因為 TTL 刪除需要寫入容量。

使用全域資料表的串流和交易

每個全域資料表都會根據其所有寫入作業產生獨立的串流，無論這些寫入的起始點為何。您可以選擇在一個區域或所有區域中獨立使用此 DynamoDB 串流。

如果您想要處理的本機寫入，但不要複寫寫，您可以將自己的 Region 屬性新增至每個項目。然後，您可以使用 Lambda 事件篩選條件，只叫用 Lambda 在本機區域中進行寫入。

事務操作僅在最初寫入的區域內提供 ACID (原子性，一致性，隔離和持久性) 保證。全域資料表中不支援跨區域的交易。

例如，如果您在美國東部 (俄亥俄) 和美國西部 (奧勒岡) 區域有一個包含複本的全域表格，並在美國東部 (俄亥俄) 區域執行 `TransactWriteItems` 作業，則可能會在美國西部 (奧勒岡) 區域觀察到部分完成的交易，當變更複寫時。只有在來源區域中提交變更後，才會將變更複製到其他區域。

Note

- 全域資料表透過直接更新 DynamoDB 來「寫入」DynamoDB 加速器。因此，DAX 不會意識到它持有陳舊的數據。只有在快取的 TTL 到期時，才會重新整理 DAX 快取。
- 全域資料表上的標籤不會自動傳播。

讀取和寫入輸送量

全域資料表會以下列方式管理讀取和寫入輸送量。

- 跨區域的所有表格執行個體的寫入容量必須相同。
- 對於版本 2019.11.21 (目前版本)，如果表格設定為支援 `auto` 動調整或處於隨選模式，則寫入容量會自動保持同步。這表示一個資料表的寫入容量變更會複寫到其他資料表。
- 區域之間的讀取容量可能會有所不同，因為讀取可能不相等。將全域複本新增至資料表時，會傳播來源區域的容量。建立之後，您可以調整一個複本的讀取容量，而這個新設定不會傳輸到另一端。

一致性和衝突解決

對任何複本列表中任何項目所做的任何變更都會複寫到相同全域資料表中的所有其他複本。在全域資料表中，新寫入的項目通常會在一秒內傳播到所有複本列表。

使用全域資料表，每個複本列表會存放相同的資料項目集。DynamoDB 不支援僅對某些項目進行部分複寫。

應用程式可以讀取資料和將資料寫入至任何複本列表。如果您的應用程式僅使用最終一致性讀取，並且僅針對一個 Region 讀取問題，則它將無需任何修改即可工作。不過，如果您的應用程式需要強烈一致讀取，則必須在相同區域中執行所有強烈一致讀取和寫入。DynamoDB 不支援跨區域的強式一致性讀取。因此，如果您對某個區域進行寫入並從另一個區域進行讀取，則讀取回應可能包含過時資料，這些資料不會反映最近在另一個區域中完成的寫入結果。

如果應用程式大約在同一時間更新不同區域中的相同項目，則可能會發生衝突。為了協助確保最終一致性，DynamoDB 全域資料表會在並行更新間使用最後寫入者獲勝核對機制，其中 DynamoDB 會在並

行更新間盡最大努力判斷最後寫入者。這會在項目層級執行。有了這個衝突解決機制，所有的複本都會同意最新的更新，並朝它們都具有相同資料的狀態收斂。

可用性與持久性

如果單一 AWS 區域變成隔離或降級，您的應用程式可以重新導向至不同的區域，並針對不同的複本表格執行讀取和寫入。您可以套用自訂商業邏輯來決定何時將請求重新導向至其他區域。

如果區域變成隔離或降級，DynamoDB 會追蹤已執行但尚未傳播到所有複本表格的任何寫入。當區域重新回到線上的狀態時，DynamoDB 會繼續將該區域的任何擱置寫入傳播到其他區域的複本列表中。它也會繼續將寫入從其他複本表格傳播到現在重新連線的區域。

管理全域資料表的最佳實務和要求

您可以使用 Amazon DynamoDB 全域表格，跨 AWS 區域複寫表格資料。全域資料表中的複本列表和次要索引必須具有相同的寫入容量設定，才能確保資料能正確複寫。

為了將來清楚起見，最好不要將 Region 放在任何有朝一日可能變成全域資料表的資料表的名稱中。

Warning

每個全域資料表的資料表名稱在您的 AWS 帳戶中必須是唯一的。

全域資料表版本

若要判斷您正在使用的全域資料表版本，請參閱[判斷您正在使用的全域資料表版本](#)。

管理容量的要求

全域資料表必須按以下兩種方式之一設定輸送容量：

1. 隨需容量模式，以複寫寫入請求單位 (rWRU) 測量
2. 具有自動擴展功能的佈建容量模式，以複寫寫入容量單位 (rWCU) 測量

使用具有自動擴展功能的佈建容量模式或隨需容量模式，可確保全域資料表擁有足夠的容量，以便對全域資料表的所有區域執行複寫寫入。

Note

在任何區域中從一種資料表容量模式切換到其他容量模式，會切換所有複本的模式。

部署全域資料表

在中 AWS CloudFormation，每個全域資料表都由單一區域中的單一堆疊控制。無論複本數量有多少。當您部署範本時，CloudFormation 會建立/更新所有複本，做為單一堆疊作業的一部分。因此，您不應該在多個區域中部署相同的 `AWS::DynamoDB::GlobalTable` 資源。否則會因不受支援而導致錯誤。

如果您在多個區域中部署應用程式範本，則可以使用條件僅在單一區域中建立資源。您也可以選擇在與應用程式分開的堆疊中定義 `AWS::DynamoDB::GlobalTable` 資源，並確保它僅部署到單一區域。如需詳細資訊，請參閱[全域表 CloudFormation](#)

DynamoDB 資料表由 `AWS::DynamoDB::Table` 參照，而全域資料表則為 `AWS::DynamoDB::GlobalTable`。就目前而言，這基本上使得它們有兩種不同的資源。CloudFormation 因此，一種方法是透過使用 `GlobalTable` 建構模組建立可能是全域的所有資料表。然後，您可以將它們保留為獨立資料表來啟動，並在稍後視需要將它們新增至區域。

如果你有一個常規表，並且你想在使用時轉換它 CloudFormation，推薦的方法是：

1. 設定要保留的刪除原則。
2. 從堆疊中移除資料表。
3. 在控制台中將資料表轉換為全域資料表。
4. 將全域資料表作為新資源導入堆疊。

Note

目前不支援跨帳戶複寫。

使用全域資料表來協助處理潛在的區域中斷

擁有或能在替代區域中快速建立執行堆疊的獨立副本，每個區域都會存取其本機 DynamoDB 端點。

使用 Route53 或前 AWS Global Accelerator 往最近的健康區域。或者，讓用戶端知道其可能使用的多個端點。

在每個區域中使用運作狀態檢查，可靠地判斷堆疊是否有任何問題，包括 DynamoDB 是否降級。例如，不要只偵測 DynamoDB 端點是否啟動。實際發出呼叫，確保完全成功的資料庫流量。

如果運作狀態檢查失敗，流量可以路由到其他區域 (透過使用 Route53 更新 DNS 項目、讓全域加速器執行不同的路由，或讓用戶端選擇不同的端點)。全域資料表具有良好的 RPO (復原點目標)，因為資料會持續同步，且具有良好的 RTO (復原時間目標)，因為這兩個區域都會隨時準備好資料表，以供讀取和寫入流量使用。

有關運作狀態檢查的詳細資訊，請參閱[運作狀態檢查](#)。

Note

DynamoDB 是其他服務經常在其上建立控制平面作業的核心服務，因此您不太可能會遇到 DynamoDB 在某個區域中降級服務，而其他服務未受到限制的情況。

備份全域資料表

備份全域資料表時，備份一個區域中的資料表應該已足夠，不需要備份所有區域中的所有資料表。如果目的是能恢復不小心刪除或修改的資料，那麼一個區域中的 PITR 便已足夠。同樣地，為了歷史目的 (例如法規要求) 而保留快照時，在一個區域中進行備份也應該足夠。備份資料可以透過 AWS Backup 複製到多個區域。

複本和計算寫入單位

為了進行規劃，您應該獲取區域將執行的寫入次數，並將其新增到其他區域的寫入次數中。這很重要，因為在一個區域中執行的每個寫入也必須在每個複本區域中執行。如果您沒有足夠的容量來處理所有寫入，就會發生容量例外狀況。此外，區域間複寫的等待時間也會增加。

例如，假設您預期在俄亥俄州對複本資料表進行每秒 5 次寫入，在維吉尼亞北部對複本資料表進行每秒 10 次寫入，在愛爾蘭對複本資料表進行每秒 5 次寫入。在這種情況下，您應預期在俄亥俄州、維吉尼亞北部和愛爾蘭區域使用 20 個 rWCU 或 rWRU。換句話說，您應預期在所有三個區域共使用 60 個 rWCU。

如需有關具有自動擴展功能的佈建容量和 DynamoDB 的詳細資訊，請參閱 [使用 DynamoDB Auto Scaling 功能自動管理輸送容量](#)。

Note

如果資料表在具有自動擴展功能的佈建容量模式下執行，則允許佈建的寫入容量在每個區域的這些自動擴展設定鐘中浮動。

教學課程：建立全域資料表

本節說明如何使用 Amazon DynamoDB 主控台或 AWS Command Line Interface (AWS CLI) 建立全域資料表。

主題

- [建立全域資料表 \(主控台\)](#)
- [建立全域資料表 \(AWS CLI\)](#)
- [建立全域資料表 \(Java\)](#)

建立全域資料表 (主控台)

請遵循下列步驟，使用主控台建立全域資料表。下列範例使用美國及歐洲的複本列表，建立全域資料表。

1. 在 <https://console.aws.amazon.com/dynamodb/home> 開啟 DynamoDB 主控台。針對此範例，請選擇 US East (Ohio) (美國東部俄亥俄州) 區域。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇 Create Table (建立資料表)。
 - a. 對於 Table name (資料表名稱)，請輸入 **Music**。
 - b. 在 Partition key (分區索引鍵)，輸入 **Artist**。選擇 Sort key (排序索引鍵)，然後輸入 **SongTitle**。(**Artist** 和 **SongTitle** 都必須是字串。)

選擇 Create table (建立資料表) 以建立資料表。此資料表會做為新全域資料表中的第一個複本資料表。這是您稍後新增其他複本資料表的原型。

4. 選擇 Global Tables (全域資料表) 標籤，然後選擇 Create replica (建立複本)。
5. 從 Available replication Regions (可用的複製區域) 下拉式清單，選擇 US West (Oregon) (美國西部 (奧勒岡))。

主控台會檢查，確認所選取的區域中沒有同名的資料表。若有同名的資料表，您必須先刪除現有的資料表，才可在該區域中建立新的複本資料表。

6. 選擇 Create replica (建立複本)。這會啟動在美國西部 (奧勒岡) 建立資料表的程序。

所選資料表的 Global Table (全域資料表) 標籤 (及所有其他複本資料表)，會顯示該資料表已複寫到多個區域。

7. 您現在可以新增其他區域，複寫及同步美國與歐洲的全域資料表。若要執行此作業，請重複步驟 5，但這次請改為指定 Europe (Frankfurt) (歐洲 (法蘭克福))，而非 US West (Oregon) (美國西部 (奧勒岡))。
8. 您應該仍會在美國東部 (俄亥俄) 區域使用 AWS Management Console。選取左側導覽功能表中的 Items (項目)，選取 Music (音樂) 資料表，然後選擇 Create Item (建立項目)。
 - a. 針對 Artist (藝人)，輸入 `item_1`。
 - b. 針對 SongTitle，請輸入 **Song Value 1**。
 - c. 若要寫入該項目，請選擇 Create item (建立項目)。
9. 稍待片刻之後，該項目將會複寫到您全域資料表中的所有三個區域。若要確認，請在主控台中移至右上角的區域選擇器，然後選擇 Europe (Frankfurt) (歐洲 (法蘭克福))。歐洲 (法蘭克福) 中的 Music 資料表此時應已包含新的項目。
10. 重複步驟 9，然後選擇 US West (Oregon) (美國西部 (奧勒岡)) 以驗證該區域中的複寫。

建立全域資料表 (AWS CLI)

請遵循下列步驟，使用 AWS CLI 建立全域資料表 Music。以下範例會建立全域資料表，並在美國及歐洲皆擁有複本資料表。

1. 在美國東部 (俄亥俄) 建立新的資料表 (Music)，並啟用 DynamoDB Streams (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. 在美國東部 (維吉尼亞北部) 建立相同的 Music 資料表。

```
aws dynamodb update-table --table-name Music --cli-input-json \  
'{  
  "ReplicaUpdates":  
  [  
    {  
      "Create": {  
        "RegionName": "us-east-1"  
      }  
    }  
  ]  
' \  
--region=us-east-2
```

3. 重複步驟 2 以在歐洲 (愛爾蘭) (eu-west-1) 中建立資料表。
4. 您可以使用 describe-table 檢視複本的清單。

```
aws dynamodb describe-table --table-name Music --region us-east-2
```

5. 若要確認複寫正常運作，請將項目新增到美國東部 (俄亥俄) 中的 Music 資料表。

```
aws dynamodb put-item \  
  --table-name Music \  
  --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \  
  --region us-east-2
```

6. 稍待幾秒鐘，然後檢查該項目是否已成功複寫到美國東部 (維吉尼亞北部) 與歐洲 (愛爾蘭)。

```
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. 刪除歐洲 (愛爾蘭) 區域中的複本列表。

```
aws dynamodb update-table --table-name Music --cli-input-json \  
'{  
  "ReplicaUpdates":  
  [  
    {  
      "Delete": {  
        "RegionName": "eu-west-1"  
      }  
    }  
  ]  
'
```

建立全域資料表 (Java)

下列 Java 程式碼範例在歐洲 (愛爾蘭) 區域中建立 Music 資料表，然後在亞太區域 (首爾) 中建立複本。

```
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))));

    }
}
```

監控全域資料表

您可以使 CloudWatch 用 Amazon 監控全域表的行為和效能。Amazon DynamoDB 為全域資料表中的每個複本發佈 ReplicationLatency 指標。

- **ReplicationLatency**：一個項目寫入至複本列表，以及當該項目於全域資料表中的另一個複本中顯示的所需時間。ReplicationLatency 會以毫秒表示，並會針對每對來源及目標區域配對發送。

ReplicationLatency 在正常操作期間應該很穩定。ReplicationLatency 值上升可能表示某個複本的更新未及時散佈到其他複本資料表。一段時間後，這會造成其他複本資料表落後，因為他們不再一致地收到更新。在此情況下，您應該確認每個複本資料表的讀取容量單位 (RCU) 和寫入容量單位 (WCU) 皆相同。此外，選擇 WCU 設定時應遵循 [全域資料表版本](#) 中的建議。

如果某個 AWS 區域遭到降級，而您在該區域有複本列表，ReplicationLatency 就會增加。這種情況下，您可以暫時將應用程式的讀取和寫入活動重新導向至不同的 AWS 區域。

如需更多詳細資訊，請參閱 [DynamoDB 指標和維度](#)。

對全域資料表使用 IAM

當您第一次建立全域資料表時，Amazon DynamoDB 會自動為您建立 AWS Identity and Access Management (IAM) 服務連結角色。此角色名為 [AWSServiceRoleForDynamoDBReplication](#)，可讓 DynamoDB 代您管理全域資料表的跨區域複寫。請勿刪除此服務連結角色。若您刪除，您所有全域資料表將無法再運作。

如需服務連結角色的詳細資訊，請參閱 IAM 使用者指南中的 [使用服務連結角色](#)。

若要在 DynamoDB 中建立複本列表，您必須在來源區域中有下列許可。

- dynamodb:UpdateTable

若要在 DynamoDB 中建立複本列表，您必須在目的地區域中有下列許可。

- dynamodb:CreateTable
- dynamodb:CreateTableReplica
- dynamodb:Scan
- dynamodb:Query
- dynamodb:UpdateItem
- dynamodb:PutItem
- dynamodb:GetItem
- dynamodb>DeleteItem

- `dynamodb:BatchWriteItem`

若要在 DynamoDB 中刪除複本列表，您必須在目的地區域中有下列許可。

- `dynamodb>DeleteTable`
- `dynamodb>DeleteTableReplica`

若要透過更新複本 auto 調整規模原則 `UpdateTableReplicaAutoScaling`，您必須在所有存在表格複本的區域中具有下列權限

- `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`

若要使用 `UpdateTimeToLive`，您必須在有資料表複本的所有區域中有 `dynamodb:UpdateTimeToLive` 的許可。

範例：新增複本

下列 IAM 政策授予許可，可讓您在全域資料表中新增複本。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
```

```

        "dynamodb:DescribeTable",
        "dynamodb:UpdateTable",
        "dynamodb:CreateTableReplica",
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "*"
}
]
}

```

範例：更新 AutoScaling 原則

以下 IAM 政策授予許可，以允許您更新複本 auto 擴展政策。

```

{
  "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": "*"
    }
  ]
}

```

範例：允許特定資料表名稱和區域的複本建立

下列 IAM 政策授予許可，允許三個區域中有複本的 Customers 資料表建立資料表和複本。

```

{

```

```
"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"
    ]
  }
]
```

判斷您正在使用的全域資料表版本

有兩個版本的 DynamoDB 全域資料表可供使用：[全域表版本 2019.11.21 \(目前版本\)](#) 和 [全域表格版本 \(舊版\)](#)。我們建議您使用「[全域表](#)」[版本 2019.11.21 \(目前版本\)](#)。它的效率更高，並且消耗的寫入容量比 [全域表格版本 \(舊版\)](#) 少。目前版本的優點包括：

- 來源和目標資料表會一起維護，並自動對齊輸送量、TTL 設定、auto 動縮放設定和其他有用的屬性。
- 全域次要索引也會保持對齊。
- 您可以從已填入資料的資料表中動態新增複本資料表
- 控制複寫所需的中繼資料屬性會隱藏起來，這有助於防止可能造成複寫問題的寫入。
- 目前版本支援的區域比舊版多，且可讓您在舊版不支援的情況下，在現有資料表中新增或移除區域。
- [全域表版本 2019.11.21 \(目前版本\)](#) 比較有效率，且耗用的寫入容量更少 [全域表格版本 \(舊版\)](#)，因此更具成本效益。具體來說：
 - 在一個區域中插入新項目，然後複製到其他區域，在 2017.11.29 版 (舊版) 每個區域需要 2 個 rWCU，但在 2019.11.21 版 (目前) 則只需要 1 個。
 - 在 2017.11.29 版 (舊版) 中更新項目，需要來源區域中的 2 個 rWCU，然後每個目標區域需要 1 個 rWCU，但在 2019.11.21 版 (目前) 中每個來源或目標都只需要 1 個 rWCU。

- 在 2017.11.29 版 (舊版) 中刪除項目，需要來源區域中的 1 個 rWCU，然後每個目標區域需要 2 個 rWCU，但在 2019.11.21 版 (目前) 中每個來源或目標都只需要 1 個 rWCU。

如需詳細資訊，請參閱 [Amazon DynamoDB 定價](#)。

透過 CLI 判斷版本

若要瞭解您正在使用哪個版本的全域表格 AWS CLI，請核取 `DescribeTable` 和 `DescribeGlobalTable`。如果是版本 2019.11.21 (目前版本)，則會顯示表格版本，如果是版本 2017.11.29 (舊版)，則 `DescribeGlobalTable` 屬性將顯示表格版本。

透過主控台判斷版本

透過主控台尋找版本

若要透過主控台了解您正使用的全域資料表版本，請執行以下操作：

1. 在 <https://console.aws.amazon.com/dynamodb/home> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇您希望重新使用的資料表。
4. 選擇 Global Tables (全域資料表) 標籤。
5. Global table version (全域資料表版本) 會顯示使用中的全域資料表版本：

 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.

若要將現有全域資料表從 2017.11.29 版 (舊版) 升級至 2019.11.21 版 (目前)，請遵照 [此處](#) 的這些步驟進行。整體升級程序可在不中斷即時資料表的情況下運作，且應該在一小時內完成。如需詳細資訊，請參閱 [更新至 2019.11.21 版 \(目前版本\)](#)

Note

- 如果全域資料表版本訊息未出現在主控台中，則表示在不同區域中有另一個具有相同名稱的資料表。在這種情況下，目前的資料表無法建立為全域資料表。必須將目前的資料表複製到具有唯一名稱的新資料表，或移除所有其他具有相同名稱的資料表。
- 如果您使用的是[全域表 2019.11.21 \(目前\) 版本](#)的全域表，而且您也使用存留[時間功能](#)，DynamoDB 會將 TTL 刪除複寫到所有複本表格。初始 TTL 刪除不會在 TTL 過期發生時消耗區域中的寫入容量。但是，在每個複本區域中，使用佈建容量時複製 TTL 刪除至複製的資料表會消耗一個複製的寫入容量單位，或在使用隨需容量模式時消耗一個複製的寫入容量單位，且您將支付適用的費用。
- 在[全域表版本 2019.11.21 \(目前版本\)](#) 中，當 TTL 刪除發生時，它會複寫到所有複本區域。這些複製的寫入不包含 type 或 principalID 屬性。這會導致難以區分 TTL 刪除與複寫資料表中的使用者刪除。

將全域表格從舊版 (2017.11.29) 版本升級到目前版本

有兩個版本的 DynamoDB 全域資料表可供使用：[全域表版本 2019.11.21 \(目前版本\)](#) 和 [全域表格版本 \(舊版\)](#)。客戶應盡可能使用 2019.11.21 版 (目前)，因為相較於 2017.11.29 (舊版)，其提供更大的彈性、更高的效率，且消耗的寫入容量更少。若要判斷您使用的版本，請參閱[判斷您正在使用的全域資料表版本](#)。

本節說明如何使用 DynamoDB 主控台將全域資料表升級至 2019.11.21 版 (目前版本)。從版本 2017.11.29 (舊版) 升級至版本 2019.11.21 (目前版本) 是一次性的動作，您無法將其還原。目前，您只能使用主控台升級全域表。

主題

- [舊版和目前版本之間的行為差異](#)
- [升級的必要](#)
- [全域表升級所需的權限](#)
- [升級期間的期望](#)
- [DynamoDB 在升級之前、期間和升級後串流行為](#)
- [升級至版本 \(目前版本\)](#)

舊版和目前版本之間的行為差異

下列清單說明全域資料表的舊版和目前版本之間的行為差異。

- 與版本 2017.11.29 (舊版) 相比，版本 2019.11.21 (目前版本) 在數個 DynamoDB 作業上消耗的寫入容量較少，因此對於大多數客戶而言，更具成本效益。這些 DynamoDB 作業的差異如下：
 - 在 2017.11.29 (舊版) 中呼叫 [PutItem](#) 區域中的 1 KB 項目並複製到其他區域需要 2 個 RWRU (舊版)，但 2019.11.21 (目前) 只需要 1 個 RWRU。
 - 呼叫 [UpdateItem](#) 1KB 料號在 2017.11.29 (舊版) 的來源區域中需要 2 個 RWRU，而在 2017.11.29 (舊版) 的每個目的地區域需要 1 個 RWRU，但 2019.11.21 (目前) 的來源區域與目的地區域只需要 1 個 RWRU。
 - 呼叫 [DeleteItem](#) 1KB 料號在 2017.11.29 (舊版) 的來源區域中需要 1 個 RWRU 和每個目的地區域 2 RWRU (舊版)，但 2019.11.21 (目前) 的來源或目的地區域只需要 1 個 RWRU。

下表顯示 2017 年 11 月 29 日 (舊版) 和 2019 年 11 月 21 日 (目前) 的註冊資料庫使用量。

在兩個區域中，每個 1 KB 項目的註冊資料耗用量分別為 2015 年 11 月 29 日 (舊版) 和 2019 年 11 月 21 日 (目前)

作業	2017 年 11 月 29 日 (舊版)	2019 年 11 月 21 日 (當前)	節省
PutItem	4 RRU	2 RRU	50%
UpdateItem	3 RRU	2 RRU	33%
DeleteItem	3 RRU	2 RRU	33%

- 版本 2017.11.29 (舊版) 僅提供 11 版。AWS 區域但是，版本 2019.11.21 (目前版本) 在所有版本中都可以使用。AWS 區域
- 您可以建立版本 2017.11.29 (舊版) 全域資料表，方法是先建立一組空白區域資料表，然後叫用 [CreateGlobalTable](#) API 來形成全域資料表。您可以透過呼叫 [UpdateTable](#) API 將複本新增至現有地區資料表，來建立版本 2019.11.21 (目前) 全域表格。
- 版本 2017.11.29 (舊版) 要求您先清空表格中的所有複本，然後才能在新區域中新增複本 (包括建立期間)。版本 2019.11.21 (目前版本) 支援您在已包含資料的資料表上新增和移除區域的複本。
- 版本 2017.11.29 (舊版) 使用以下專用的控制平面 API 集來管理複本：
 - [CreateGlobalTable](#)
 - [DescribeGlobalTable](#)

- [DescribeGlobalTableSettings](#)
- [ListGlobalTables](#)
- [UpdateGlobalTable](#)
- [UpdateGlobalTableSettings](#)

版本 2019.11.21 (目前版本) 使用 [DescribeTable](#) 和 [UpdateTable](#) API 來管理複本。

- 版本 2017.11.29 (舊版) 會針對每次寫入發佈兩筆 DynamoDB Streams 記錄。版本 2019.11.21 (目前版本) 只會針對每次寫入發佈一筆 DynamoDB Streams 記錄。
- 版本 2017.11.29 (舊版) 會填入並更新 `aws:rep:deleting`、`aws:rep:updateregion` 和屬性。 `aws:rep:updatetime` 版本 2019.11.21 (目前版本) 不會填入或更新這些屬性。
- 版本 2017.11.29 (舊版) 不會同步處理所有複本的 [生存時間 \(TTL\)](#) 設定。版本 2019.11.21 (目前版本) 會在各個複本之間同步 TTL 設定。
- 版本 2017.11.29 (舊版) 不會將 TTL 刪除複寫到其他複本。版本 2019.11.21 (目前版本) 會將 TTL 刪除項目複製到所有複本。
- 版本 2017.11.29 (舊版) 不會同步處理跨複本的 [auto 調整](#) 設定。版本 2019.11.21 (目前版本) 會在各個複本之間同步 auto 調整設定。
- 版本 2017.11.29 (舊版) 不會同步處理所有複本之間的 [全域次要索引 \(GSI\)](#) 設定。版本 2019.11.21 (目前版本) 會在多個複本間同步 GSI 設定。
- 版本 2017.11.29 (舊版) 不會 [在複本之間同步處理靜態加密](#) 設定。版本 2019.11.21 (目前版本) 在複本之間同步處理靜態加密設定。
- 版本 2017.11.29 (舊版) 會發佈量度。 `PendingReplicationCount` 版本 2019.11.21 (目前版本) 不會發佈此測量結果。

升級的必要

開始升級至版本 2019.11.21 (目前) 全域表格之前，您必須符合下列先決條件：

- [生存時間 \(TTL\)](#) 複本上的設定在各個區域之間是一致的。
- 複本上的 [全域次要索引 \(GSI\)](#) 定義在各個區域之間是一致的。
- 複本上的 [靜態加密](#) 設定在各個區域之間是一致的。
- DynamoDB auto 調整功能已針對所有複本啟用 WCU，或針對所有複本啟用 [隨需](#) 容量模式。
- 應用程式不需要表格項目中的 `aws:rep:deleting`、`aws:rep:updateregion`、和 `aws:rep:updatetime` 屬性。

全域表升級所需的權限

若要升級至版本 2019.11.21 (目前版本)，您必須在所有具有複本的區域中擁有 `dynamodb:UpdateGlobalTableVersion` 權限。除了存取 DynamoDB 主控台和檢視表格所需的權限外，還需要這些權限。

下列 IAM 政策授予將任何全域資料表升級至 2019.11.21 版 (目前版本) 的許可。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:UpdateGlobalTableVersion",
      "Resource": "*"
    }
  ]
}
```

下列 IAM 政策授予許可，只將兩個區域中具有複本的 Music 全域資料表升級為 2019.11.21 版 (目前版本)。

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

升級期間的期望

- 升級時，所有全域表格複本都會繼續處理讀取和寫入流量。
- 升級程序需要幾分鐘到數小時的時間，具體取決於資料表大小和複本數目。

- 在升級程序期間，的值 `TableStatus` 會從變更 ACTIVE 為 UPDATING。您可以透過叫用 [DescribeTable](#) API 或 [DynamoDB](#) 主控台中的「表格」檢視來檢視表格的狀態。
- 在升級表格時，Auto Scaling 不會調整全域表的佈建容量設定。強烈建議您在升級期間將表格設定為 [隨需](#) 容量模式。
- 如果您選擇在升級期間使用 [佈建](#) 的容量模式搭配 auto 擴展，則必須增加原則的最小讀取和寫入輸送量，以適應任何預期的流量增加，以避免升級期間的限制。
- 升級程序完成後，您的表格狀態將變更為 ACTIVE。

DynamoDB 在升級之前、期間和升級後串流行為

作業	複本區域	升級前的行為	升級期間的行為	升級後的行為
放置或更新	來源	時間戳記人口發生使用 UpdateItem .	時間戳記人口發生使用 PutItem .	不會產生客戶可見的時間戳記。
		會產生兩個串流記錄。第一條記錄包含客戶寫入的屬性。第二條記錄包含 <code>aws:rep:*</code> 屬性。	會產生兩個串流記錄。第一條記錄包含客戶寫入的屬性。第二條記錄包含 <code>aws:rep:*</code> 屬性。	系統會產生包含客戶寫入屬性的單一「串流」記錄。
		每個客戶寫入時會使用兩個 RWCU。	每個客戶寫入時會使用兩個 RWCU。	每個客戶寫入時會使用一個 RWCU。
		ReplicationLatency 和 PendingReplicationCount 量度在中發佈 CloudWatch。	ReplicationLatency 和 PendingReplicationCount 量度在中發佈 CloudWatch。	ReplicationLatency 量度發佈於 CloudWatch。

作業	複本區域	升級前的行為	升級期間的行為	升級後的行為
	目的地	<p>複寫發生使用 PutItem.</p> <p>系統會產生單一「串流」記錄，其中包含客戶撰寫的屬性和aws:rep:* 屬性。</p> <p>如果料號存在於目的地區域中，則會耗用一個 RWCU。如果目的地區域中不存在該項目，則會使用兩個 RWCU。</p> <p>ReplicationLatency 和PendingReplication Count 量度在中發佈 CloudWatch。</p>	<p>複寫發生使用 PutItem.</p> <p>系統會產生單一「串流」記錄，其中包含客戶撰寫的屬性和aws:rep:* 屬性。</p> <p>如果料號存在於目的地區域中，則會耗用一個 RWCU。如果目的地區域中不存在該項目，則會使用兩個 RWCU。</p> <p>ReplicationLatency 和PendingReplication Count 量度在中發佈 CloudWatch。</p>	<p>複寫發生使用 PutItem.</p> <p>系統會產生單一「串流」記錄，其中僅包含客戶撰寫的屬性，且不包含複製屬性。</p> <p>每個客戶寫入時會使用一個 RWCU。</p> <p>ReplicationLatency 量度發佈於 CloudWatch。</p>
刪除	來源	<p>使用刪除具有較小時間戳的任何項目 DeleteItem。</p>	<p>使用刪除具有較小時間戳的任何項目 DeleteItem。</p>	<p>使用刪除具有較小時間戳的任何項目 DeleteItem。</p>

作業	複本區域	升級前的行為	升級期間的行為	升級後的行為
		<p>系統會產生單一「串流」記錄，其中包含客戶撰寫的屬性和aws:rep:* 屬性。</p> <p>每個客戶刪除都會使用一個 RWCU。</p>	<p>系統會產生單一「串流」記錄，其中包含客戶撰寫的屬性和aws:rep:* 屬性。</p> <p>每個客戶刪除都會使用一個 RWCU。</p>	<p>系統會產生單一「串流」記錄，其中包含客戶撰寫的屬性。</p> <p>每個客戶刪除都會使用一個 RWCU。</p>
		<p>ReplicationLatency 和PendingReplicationCount 量度在中發佈 CloudWatch。</p>	<p>ReplicationLatency 和PendingReplicationCount 量度在中發佈 CloudWatch。</p>	<p>ReplicationLatency 量度發佈於 CloudWatch。</p>
	目的地	<p>兩階段刪除發生：</p> <ul style="list-style-type: none"> 在階段 1 中，UpdateItem 設定刪除旗標。 在階段 2 中，DeleteItem 刪除項目。 	<p>使用刪除項目 DeleteItem。</p>	<p>使用刪除項目 DeleteItem。</p>

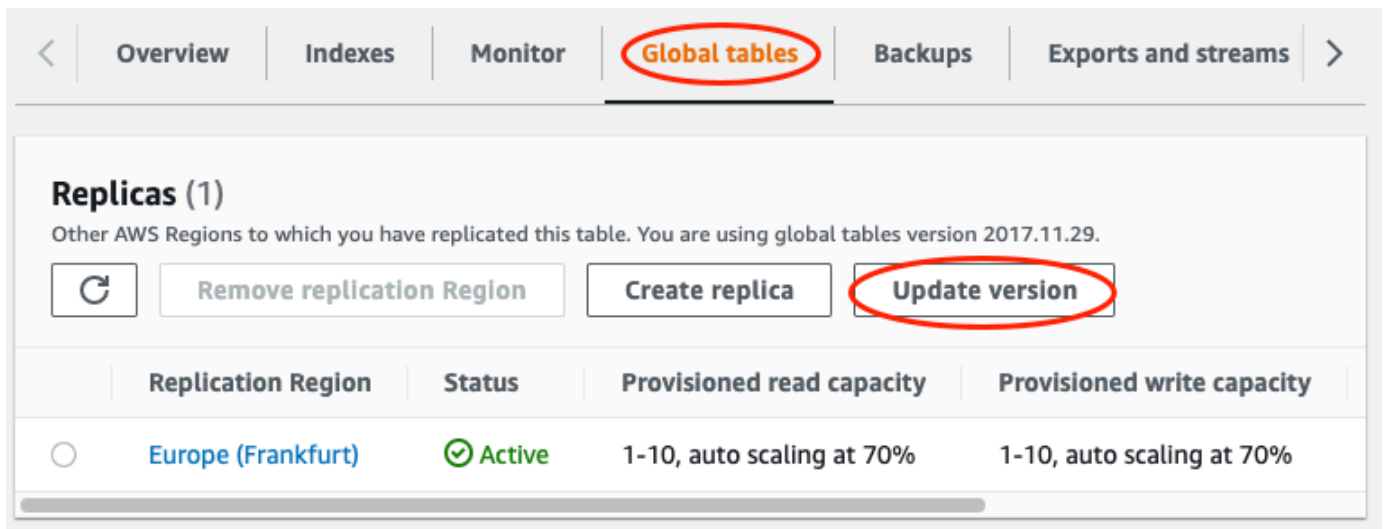
作業	複本區域	升級前的行為	升級期間的行為	升級後的行為
		會產生兩個串流記錄。第一筆記錄包含對 <code>aws:rep:deleting</code> 欄位的變更。第二個記錄包含客戶寫入的屬性和 <code>aws:rep:*</code> 屬性。	系統會產生單一資料流記錄，其中包含客戶撰寫的屬性。	系統會產生單一資料流記錄，其中包含客戶撰寫的屬性。
		每個客戶刪除時會使用兩個 RWCU。	每個客戶刪除都會使用一個 RWCU。	每個客戶刪除都會使用一個 RWCU。
		ReplicationLatency 和 PendingReplicationCount 量度在中發佈 CloudWatch。	ReplicationLatency 量度發佈於 CloudWatch。	ReplicationLatency 量度發佈於 CloudWatch。

升級至版本 (目前版本)

執行下列步驟，以使用升級您的 DynamoDB 全域表版本。AWS Management Console

若要將全域表格升級至版本 2019.11.21 (目前版本)

1. 在 <https://console.aws.amazon.com/dynamodb/home> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 [表格]，然後選取要升級至版本 2019.11.21 (目前) 的全域資料表。
3. 選擇 Global Tables (全域資料表) 標籤。
4. 選擇 Update version (更新版本)。



5. 閱讀並同意新的要求，然後選擇 Continue (繼續)。
6. 升級程序完成後，顯示在主控台上的全域表格版本會變更為 2019.11.21。

使用讀取和寫入操作

您可以使用 DynamoDB API 或用於 DynamoDB 的 PartiQL 來執行讀取和寫入操作。這些操作將允許您與資料表中的項目互動，以執行基本的建立、讀取、更新和刪除 (CRUD) 功能。

以下各節為本主題的詳細說明。

主題

- [DynamoDB API](#)
- [PartiQL：一種適用於 Amazon DynamoDB 的 SQL 相容查詢語言](#)

DynamoDB API

主題

- [使用項目和屬性](#)
- [項目集合-如何在 DynamoDB 中 one-to-many 建立關係的模型](#)
- [在 DynamoDB 中使用掃描](#)

使用項目和屬性

在 Amazon DynamoDB 中，項目是屬性的集合。每個屬性都有名稱和數值。屬性值可以是純量、集合，或文件類型。如需詳細資訊，請參閱 [Amazon DynamoDB：運作方式](#)。

DynamoDB 提供四種用於基本建立、讀取、更新和刪除 (CRUD) 功能的操作：所有這些操作都是不能中斷的。

- PutItem：建立項目。
- GetItem：閱讀項目。
- UpdateItem：更新項目。
- DeleteItem：刪除項目。

這些操作每一項都需要您指定希望處理之項目的主索引鍵。例如，若要使用 GetItem 讀取項目，您必須指定該項目的分割區索引鍵和排序索引鍵 (若適用的話)。

此外，除了四種基本 CRUD 操作之外，DynamoDB 也提供了下列項目：

- BatchGetItem：最多可從一或多個資料表讀取 100 個項目。
- BatchWriteItem：在一或多個資料表中最多可建立或刪除 25 個項目。

這些批次操作可將多個 CRUD 操作合併為單一請求。此外，批次操作會平行讀取和寫入項目，將回應延遲減至最低。

本節說明如何使用這些操作，並包含相關主題，例如條件式更新和原子計數器。本節也包含使用 AWS SDK 的範例程式碼。

主題

- [讀取項目](#)
- [寫入項目](#)
- [傳回值](#)
- [批次操作](#)
- [原子計數器](#)
- [條件式寫入](#)
- [在 DynamoDB 中使用表達式](#)

- [生存時間 \(TTL\)](#)
- [使用項目 : Java](#)
- [處理項目 : .NET](#)

讀取項目

若要從 DynamoDB 資料表讀取項目，請使用 `GetItem` 操作。您必須提供資料表的名稱，以及您希望取得之項目的主索引鍵。

Example

下列 AWS CLI 範例說明如何從 `ProductCatalog` 表格中讀取項目。

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"1"}}'
```

Note

使用 `GetItem`，您必須指定整個主索引鍵，而非其中一部分。例如，若資料表有複合主索引鍵 (分割區索引鍵和排序索引鍵)，您必須提供分割區索引鍵的數值和排序索引鍵的數值。

根據預設，`GetItem` 請求會執行最終一致讀取。您可以使用 `ConsistentRead` 參數改為請求強烈一致讀取 (這會消耗額外的讀取容量單位，但會傳回最多 up-to-date 版本的項目。)

`GetItem` 會傳回項目所有的屬性。您可以使用投影表達式只傳回一部分的屬性。如需詳細資訊，請參閱 [投射表達式](#)。

若要傳回 `GetItem` 使用的讀取容量單位總數，請將 `ReturnConsumedCapacity` 參數設為 `TOTAL`。

Example

以下 AWS Command Line Interface (AWS CLI) 示例顯示了一些可選 `GetItem` 參數。

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"1"}}' \  
  --return-consumed-capacity TOTAL
```



```
--consistent-read \  
--projection-expression "Description, Price, RelatedItems" \  
--return-consumed-capacity TOTAL
```

寫入項目

若要在 DynamoDB 資料表中建立、更新或刪除項目，請使用下列其中一項操作：

- PutItem
- UpdateItem
- DeleteItem

針對每一項操作，您都必須指定整個主索引鍵，非僅指定一部分。例如，若資料表有複合主索引鍵 (分割區索引鍵和排序索引鍵)，您必須提供分割區索引鍵的值和排序索引鍵的值。

若要傳回這些操作使用的寫入容量單位，請將 ReturnConsumedCapacity 參數設為下列其中一項：

- TOTAL：傳回所耗用的寫入容量單位總數。
- INDEXES：傳回所耗用的寫入容量單位總數，以及資料表的小計和受操作影響的任何次要索引。
- NONE：不傳回寫入容量的詳細資訊。(此為預設值)。

PutItem

PutItem 會建立新的項目。若資料表中已存在具有相同索引鍵的項目，該項目會取代為新的項目。

Example

將新的項目寫入 Thread 表。Thread 的主索引鍵包含 ForumName (分割區索引鍵) 和 Subject (排序索引鍵)。

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

--item 的引數會存放在 item.json 檔案中。

```
{  
  "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

若不存在具有指定之索引鍵的項目，UpdateItem 會建立新的項目。否則，它會修改現有項目的屬性。

您可以使用更新表達式指定您希望修改的屬性及其新值。如需詳細資訊，請參閱 [更新表達式](#)。

在更新表達式中，您會使用表達式屬性值做為實際數值的預留位置。如需詳細資訊，請參閱 [表達式屬性值](#)。

Example

修改 Thread 項目中的各種屬性。選用的 ReturnValues 參數會在項目更新之後顯示更新後的項目。如需詳細資訊，請參閱 [傳回值](#)。

```
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
```

--key 的引數會存放在 key.json 檔案中。

```
{
  "ForumName": {"S": "Amazon DynamoDB"},
  "Subject": {"S": "New discussion thread"}
}
```

--expression-attribute-values 的引數會存放在 expression-attribute-values.json 檔案中。

```
{
```

```
  "zero": {"N": "0"},
  "lastpostedby": {"S": "barney@example.com"}
}
```

DeleteItem

DeleteItem 會刪除具有指定之索引鍵的項目。

Example

下列 AWS CLI 範例顯示如何刪除 Thread 項目。

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

傳回值

在某些案例中，您可能會希望 DynamoDB 傳回修改前或修改後的特定屬性值。PutItem、UpdateItem 和 DeleteItem 操作具有 ReturnValues 參數，您可以使用此參數傳回修改前或修改後的屬性值。

ReturnValues 的預設值為 NONE，表示 DynamoDB 不會傳回任何已修改的屬性資訊。

以下為 ReturnValues 的其他有效設定，依 DynamoDB API 操作整理。

PutItem

- ReturnValues: ALL_OLD
 - 若您覆寫現有項目，ALL_OLD 會傳回覆寫前的整個項目。
 - 若您寫入原先不存在的項目，ALL_OLD 將不具任何效果。

UpdateItem

UpdateItem 最常見的用法便是更新現有的項目。但是，UpdateItem 實際上執行的是 upsert，表示若項目尚未存在，它會自動建立項目。

- ReturnValues: ALL_OLD
 - 若您更新現有項目，ALL_OLD 會傳回更新前的整個項目。

- 若您更新原先不存在的項目 (upsert)，ALL_OLD 將不具任何效果。
- ReturnValues: ALL_NEW
 - 若您更新現有項目，ALL_NEW 會傳回更新後的整個項目。
 - 若您更新原先不存在的項目 (upsert)，ALL_NEW 會傳回整個項目。
- ReturnValues: UPDATED_OLD
 - 若您更新現有項目，UPDATED_OLD 只會傳回更新後的屬性在更新前的樣子。
 - 若您更新原先不存在的項目 (upsert)，UPDATED_OLD 將不具任何效果。
- ReturnValues: UPDATED_NEW
 - 若您更新現有項目，UPDATED_NEW 只會傳回受影響的屬性在更新後的樣子。
 - 若您更新原先不存在的項目 (upsert)，UPDATED_NEW 只會傳回更新後的屬性在更新後的樣子。

DeleteItem

- ReturnValues: ALL_OLD
 - 若您刪除現有項目，ALL_OLD 會傳回您刪除該項目前的整個項目。
 - 若您刪除原先不存在的項目，ALL_OLD 不會傳回任何資料。

批次操作

針對需要讀取或寫入多個項目的應用程式，DynamoDB 提供 BatchGetItem 和 BatchWriteItem 操作。使用這些操作可減少您應用程式與 DynamoDB 之間網路來回行程的次數。此外，DynamoDB 會平行執行個別讀取或寫入操作。您的應用程式可從此平行處理原則中獲益，而無須管理並行或執行緒。

批次操作本質上是多個讀取或寫入請求的包裝函式。例如，若 BatchGetItem 請求包含五個項目，DynamoDB 就會代您執行五個 GetItem 操作。同樣的，若 BatchWriteItem 請求包含兩個 PUT 請求和四個刪除請求，則 DynamoDB 就會執行兩個 PutItem 請求和四個 DeleteItem 請求。

一般而言，除非批次中所有的請求皆失敗，否則批次操作不會失敗。例如，假設您執行 BatchGetItem 操作，但批次中一個個別的 GetItem 請求失敗。在此案例中，BatchGetItem 會傳回失敗 GetItem 請求的索引鍵和資料。其他批次中的 GetItem 請求則不會受到影響。

BatchGetItem

單一 BatchGetItem 操作可包含最多 100 個個別 GetItem 請求，並可擷取最多 16 MB 的資料。此外，BatchGetItem 操作可從多個資料表擷取項目。

Example

從 Thread 表擷取兩個項目，使用投影表達式卻只傳回一部分的屬性。

```
aws dynamodb batch-get-item \  
  --request-items file://request-items.json
```

--request-items 的引數會存放在 request-items.json 檔案中。

```
{  
  "Thread": {  
    "Keys": [  
      {  
        "ForumName":{"S": "Amazon DynamoDB"},  
        "Subject":{"S": "DynamoDB Thread 1"}  
      },  
      {  
        "ForumName":{"S": "Amazon S3"},  
        "Subject":{"S": "S3 Thread 1"}  
      }  
    ],  
    "ProjectionExpression":"ForumName, Subject, LastPostedDateTime, Replies"  
  }  
}
```

BatchWriteItem

BatchWriteItem 操作可包含最多 25 個個別 PutItem 和 DeleteItem 請求，並可寫入最多 16 MB 的資料。(個別項目的大小上限為 400 KB。)此外，BatchWriteItem 操作可在多個資料表中寫入或刪除項目。

Note

BatchWriteItem 不支援 UpdateItem 請求。

Example

將兩個項目寫入 ProductCatalog 表。

```
aws dynamodb batch-write-item \  
  --request-items file://request-items.json
```

```
--request-items file://request-items.json
```

--request-items 的引數會存放在 request-items.json 檔案中。

```
{
  "ProductCatalog": [
    {
      "PutRequest": {
        "Item": {
          "Id": { "N": "601" },
          "Description": { "S": "Snowboard" },
          "QuantityOnHand": { "N": "5" },
          "Price": { "N": "100" }
        }
      }
    },
    {
      "PutRequest": {
        "Item": {
          "Id": { "N": "602" },
          "Description": { "S": "Snow shovel" }
        }
      }
    }
  ]
}
```

原子計數器

您可以使用 UpdateItem 操作實作原子計數器：原子計數器為在不影響其他寫入請求的情況下，無條件遞增的數字屬性。(所有寫入請求都會按照接收的順序套用)。使用原子計數器，更新便不是等冪的。換言之，每次呼叫 UpdateItem 時，數值就會增加或減少。如果用於更新原子計數器的增量值是正的，那麼它可能會導致多計。如果增量值為負數，則可能會導致少計。

您可以使用原子計數器追蹤網站的訪客數。在此案例中，您的應用程式會增加數值，無論目前的數值為何。若 UpdateItem 操作失敗，應用程式可能只會重試操作。這可能會導致計數器更新兩次，但您或許可以容忍計數器些微多計或少計網站的訪客數。

原子計數器不適用於無法容忍多計或少計的情況 (例如銀行的應用程式)。在此案例中，使用條件式的更新而非原子計數器會更安全。

如需詳細資訊，請參閱 [增加和減少數值屬性](#)。

Example

下列 AWS CLI 範例會將 Price 產品遞增 5。在此範例中，已知項目在計數器更新之前存在。因為 UpdateItem 並非等冪，Price 會在每次執行此程式碼時增加。

```
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
```

條件式寫入

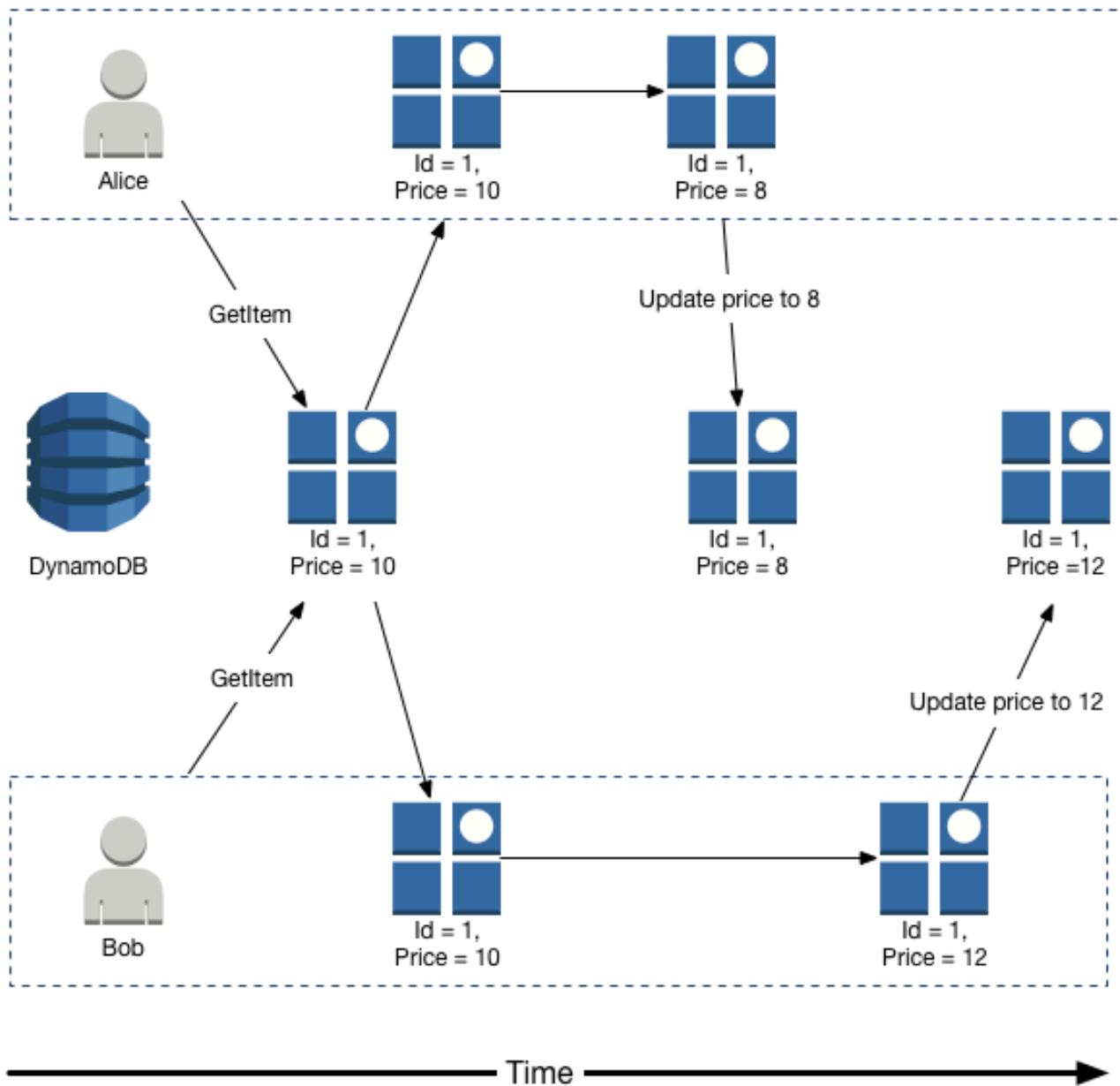
根據預設，DynamoDB 的寫入操作 (PutItem、UpdateItem、DeleteItem) 為非條件式操作：每項操作都會覆寫具有指定主索引鍵的現有項目。

DynamoDB 可選擇性的支援這些操作的條件式寫入。條件式寫入只有在項目屬性滿足一或多個預期條件時才會成功。否則會傳回錯誤。

條件式寫入會根據項目的最新更新版本檢查其條件。請注意，如果該項目以前不存在，或者對該項目的最新成功操作是刪除，則條件寫入將找不到以前的項目。

條件式寫入在許多情況下都很有幫助。例如，您可能會希望 PutItem 操作僅在沒有具有相同主索引鍵的項目時才成功。或者，您可以防止 UpdateItem 操作修改其中一個屬性為特定數值的項目。

條件式寫入在多個使用者嘗試修改相同項目的案例中會很有幫助。考慮下圖，其中兩名使用者 (Alice 和 Bob) 正在處理 DynamoDB 表中同一個項目。



假設愛麗絲使用 AWS CLI 將 Price 屬性更新為 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
```

--expression-attribute-values 的引數會存放在 expression-attribute-values.json 檔案中：

```
{
  ":newval":{"N":"8"}
}
```

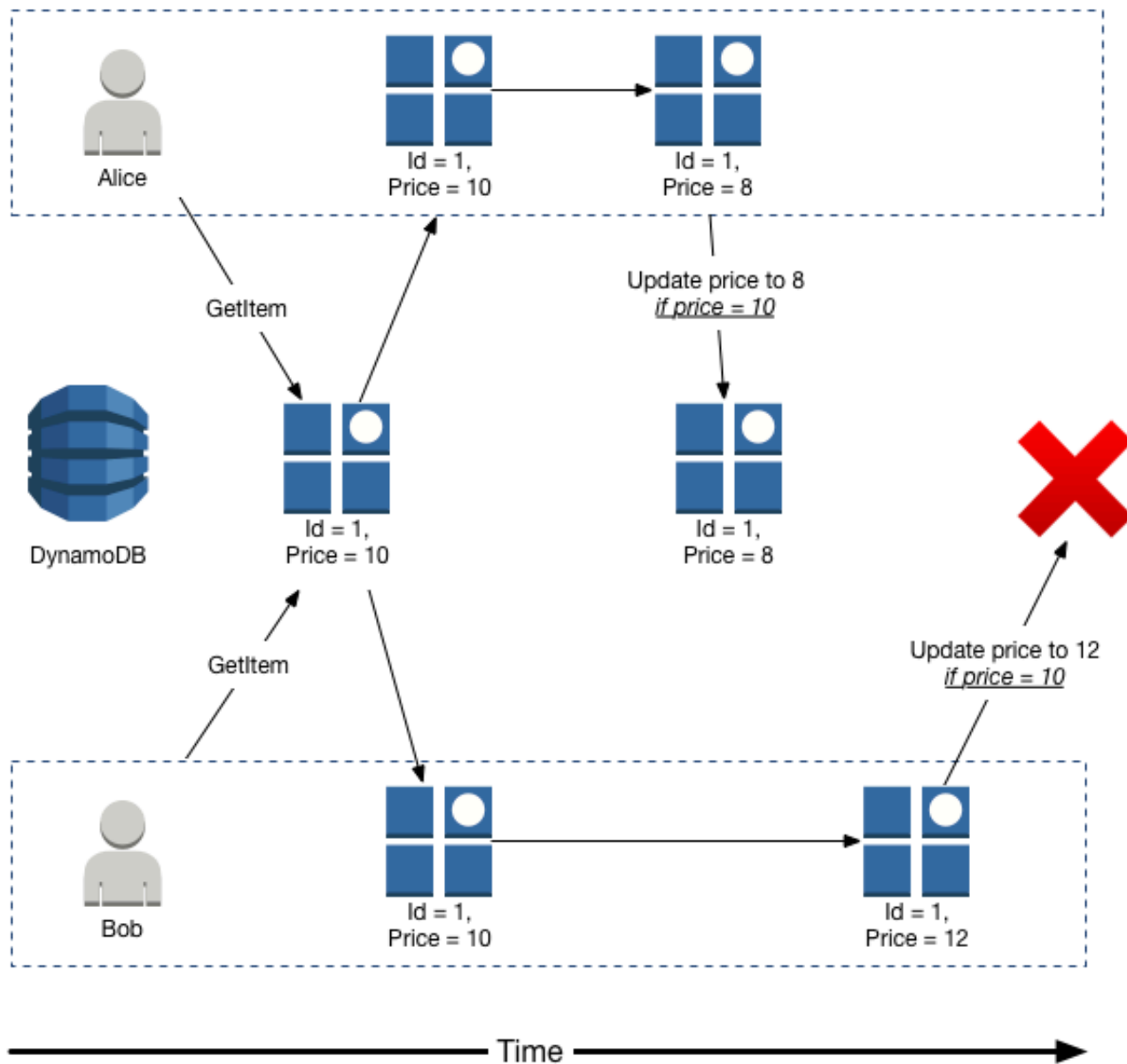
現在假設 Bob 稍後發行一個相似的 UpdateItem 請求，但將 Price 變更為 12。對 Bob 而言，--expression-attribute-values 參數看起來如下。

```
{
  ":newval":{"N":"12"}
}
```

Bob 的請求會成功，但 Alice 先前做出的更新便會遺失。

若要請求條件式 PutItem、DeleteItem 或 UpdateItem，您可以指定條件表達式。條件表達式為包含屬性名稱、條件運算子和內建函數的字串。整個表達式都必須評估為 true。否則，操作會失敗。

現在考慮下圖，示範條件式寫入如何阻擋覆寫 Alice 的更新。



Alice 首先會嘗試將 Price 更新為 8，但只有在目前的 Price 為 10 時才可以。

```
aws dynamodb update-item \
  --table-name ProductCatalog \
  --key '{"Id":{"N":"1"}}' \
  --update-expression "SET Price = :newval" \
  --condition-expression "Price = :currval" \
```

```
--expression-attribute-values file://expression-attribute-values.json
```

--expression-attribute-values 的引數會存放在 expression-attribute-values.json 檔案中。

```
{
  "newval":{"N":"8"},
  "currval":{"N":"10"}
}
```

Alice 的更新會成功，因為條件評估的結果為 true。

接下來，Bob 會嘗試將 Price 更新為 12，但目前的 Price 必須為 10 才能成功。對 Bob 而言，--expression-attribute-values 參數看起來如下。

```
{
  "newval":{"N":"12"},
  "currval":{"N":"10"}
}
```

因為 Alice 先前已將 Price 更新為 8，條件表達式評估的結果為 false，因此 Bob 的更新會失敗。

如需詳細資訊，請參閱 [條件表達式](#)。

條件式寫入等冪性

如果針對正受到更新的相同屬性進行條件式檢查，則條件式寫入可能為等冪。此表示僅在項目中的某個屬性值符合您在要求時所預期的值，DynamoDB 才會執行特定的寫入要求。

例如，假設您發出 UpdateItem 請求將項目的 Price 增加 3，但只有在目前的 Price 為 20 時才可以。在傳送請求後、取得結果前，發生網路錯誤，因此您不知道請求是否成功。因為此條件式寫入是等冪的，所以您可以重試相同的 UpdateItem 請求，且 DynamoDB 只有在目前的 Price 為 20 時才會更新項目。

條件式寫入使用的容量單位

若在條件式寫入期間，ConditionExpression 評估的結果為 false，DynamoDB 仍然會使用資料表的寫入容量：使用的量取決於現有項目的大小 (或最少為 1)。例如，如果現有項目為 300kb，而您嘗試建立或更新的新項目為 310kb，那麼條件失敗時，使用的寫入容量單位將是 300，條件成功時則會是 310。如果這是新項目 (沒有現有項目)，那麼條件失敗時，使用的寫入容量單位將是 1，條件成功時則會是 310。

Note

寫入操作只會使用寫入容量單位。它們永遠不會使用讀取容量單位。

失敗的條件式寫入會傳回 `ConditionalCheckFailedException`。發生這種情況時，您不會在回應中收到有關已使用之寫入容量的任何資訊。

若要傳回條件式寫入期間使用的寫入容量單位數，請使用 `ReturnConsumedCapacity` 參數：

- **TOTAL**：傳回所耗用的寫入容量單位總數。
- **INDEXES**：傳回所耗用的寫入容量單位總數，以及資料表的小計和受操作影響的任何次要索引。
- **NONE**：不傳回寫入容量的詳細資訊。(此為預設值)。

Note

與全域次要索引不同，本機次要索引會與其資料表共用佈建的輸送容量。本機次要索引上的讀取和寫入活動會使用其資料表佈建的輸送容量。

在 DynamoDB 中使用表達式

在 Amazon DynamoDB 中，您可以使用表達式來表示要從項目讀取的屬性。您也可以寫入項目時使用表達式來表示必須符合的任何條件 (也稱為條件更新) 及屬性的更新方式。本節說明基本表達式語法及可用的表達式類型。

Note

為了與舊版相容，DynamoDB 還支援不使用表達式的條件式參數。如需詳細資訊，請參閱 [舊式條件式參數](#)。
新的應用程式應該使用表達式，而不是舊版參數。

主題

- [使用表達式時指定項目屬性](#)
- [投射表達式](#)
- [DynamoDB 中的表達式屬性名稱](#)

- [表達式屬性值](#)
- [條件表達式](#)
- [更新表達式](#)

使用表達式時指定項目屬性

本節說明如何在 Amazon DynamoDB 中的表達式內參考項目屬性。您可以使用任何屬性，即使該位於多個清單和映射的深層巢狀結構中也一樣。

主題

- [最上層屬性](#)
- [巢狀屬性](#)
- [文件路徑](#)

樣品項目：ProductCatalog

以下是 ProductCatalog 資料表中項目的表示。(此資料表會在 [範例資料表和資料](#) 中說明。)

```
{
  "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"
}
```

注意下列事項：

- 分割區索引鍵值 (Id) 為 123。沒有排序索引鍵。
- 大多數的屬性都有純量資料類型，例如 String、Number、Boolean 和 Null。
- 一個屬性 (Color) 為 String Set。
- 下列屬性為文件資料類型：
 - RelatedItems 的清單。每個元素是相關產品的 Id。
 - Pictures 的映射。每個元素是圖片的簡短說明，以及對應影像檔的 URL。
 - ProductReviews 的映射。每個元素都代表一個評分及對應至該評分的評論清單。一開始，此映射會填入五星與一星評論。

最上層屬性

如果屬性未內嵌於另一個屬性，則為最上層。針對 ProductCatalog 項目，最上層屬性包括：

- Id
- Title
- Description
- BicycleType
- Brand
- Price
- Color
- ProductCategory
- InStock

- QuantityOnHand
- RelatedItems
- Pictures
- ProductReviews
- Comment
- Safety.Warning

除了 Color (清單)、RelatedItems (清單)、Pictures (映射) 與 ProductReviews (映射) 之外，上述所有最上層屬性都是純量。

巢狀屬性

如果屬性內嵌於另一個屬性，則為巢狀。若要存取巢狀屬性，您可以使用取值運算子：

- [n]：用於清單元素
- . (點)：用於映射元素

存取清單元素

清單元素的取消參考運算子為 [N]，其中 n 是元素編號。清單元素的開頭為零，所以 [0] 代表清單中的第一個元素，[1] 代表第二個，以此類推。以下是一些範例：

- MyList[0]
- AnotherList[12]
- ThisList[5][11]

元素 ThisList[5] 本身是巢狀清單。因此，ThisList[5][11] 是指該清單中的第 12 個元素。

方括號內的數字必須是非負整數。因此，下列表達式無效：

- MyList[-1]
- MyList[0.4]

存取映射元素

映射元素的取消參考運算子為 . (點號)。請在映射中的元素之間使用點號做為分隔符號：

- `MyMap.nestedField`
- `MyMap.nestedField.deeplyNestedField`

文件路徑

在表達式中，您可以使用文件路徑指示 DynamoDB 要在何處尋找屬性。針對最上層屬性，文件路徑就是屬性名稱。針對巢狀屬性，您可以使用取消參考運算子來建構文件路徑。

以下是文件路徑的一些範例 (請參閱 [使用表達式時指定項目屬性](#) 中所示的項目。)

- 最上層純量屬性。

Description

- 最上層清單屬性。(這會傳回整個清單，而不只是部分元素。)

RelatedItems

- RelatedItems 清單中的第三個元素 (記得清單元素的開頭為零)。

RelatedItems[2]

- 產品的正視圖。

Pictures.FrontView

- 所有五星評論。

ProductReviews.FiveStar

- 第一個五星評論。

ProductReviews.FiveStar[0]

Note

文件路徑的最大深度為 32。因此，路徑中的取消參考運算子數目不得超過此限制。

您可以在文件路徑中使用任何屬性名稱，只要名稱符合下列要求：

- 屬性名稱必須以井字號 (#) 開頭
- 第一個字元為 a-z、A-Z 或 0-9

- 第二個字符 (如果有) 為 a-z 或 A-Z

Note

若屬性名稱未符合此需求，您必須將表達式屬性名稱定義為預留位置。

如需詳細資訊，請參閱 [DynamoDB 中的表達式屬性名稱](#)。

投射表達式

若要從資料表讀取資料，您可以使用 `GetItem`、`Query` 或 `Scan` 等操作。Amazon DynamoDB 會依預設傳回所有項目屬性。若只要取得部分而非全部屬性，請使用投射表達式。

投射表達式是識別所需屬性的字串。若要擷取單一屬性，請指定其名稱。若為多個屬性，則必須以逗號分隔名稱。

以下是根據 [使用表達式時指定項目屬性](#) 中的 `ProductCatalog` 項目而來的一些投射表達式範例：

- 單一最上層屬性。

`Title`

- 三個最上層屬性。DynamoDB 會擷取整個 `Color` 設定。

`Title, Price, Color`

- 四個最上層屬性。DynamoDB 會傳回 `RelatedItems` 和 `ProductReviews` 的整個內容。

`Title, Description, RelatedItems, ProductReviews`

DynamoDB 有一份保留字與特殊字元清單。您可以在投射表達式中使用任何屬性名稱，只要第一個字元為 a-z 或 A-Z，且第二個字元 (若有的話) 為 a-z、A-Z 或 0-9 即可。如果屬性名稱不符合此需求，您必須將運算式屬性名稱定義為預留位置。如需完整清單，請參閱 [DynamoDB 中的保留字](#)。此外，下列字元在 DynamoDB 中具有特殊意義：`#` (井字號) 與 `:` (冒號)。

雖然 DynamoDB 可讓您將這些保留字與特殊字元用於名稱，但建議您避免這樣做，因為每次在表達式中使用這些名稱，都必須定義預留位置變數。如需詳細資訊，請參閱 [DynamoDB 中的表達式屬性名稱](#)。

下列 AWS CLI 範例顯示如何搭配作業使用投影 GetItem 運算式。此投射表達式會擷取最上層純量屬性 (Description)、清單中的第一個元素 (RelatedItems[0])，以及位於映射巢狀結構內的清單 (ProductReviews.FiveStar)。

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key file://key.json \  
  --projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"
```

此範例會傳回下列 JSON。

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

--key 的引數會存放在 key.json 檔案中。

```
{
  "Id": { "N": "123" }
}
```

如需程式設計語言特定的程式碼範例，請參閱[開始使用 DynamoDB 和開發套件 AWS](#)。

DynamoDB 中的表達式屬性名稱

表達式屬性名稱是您在 Amazon DynamoDB 表達式中所使用的預留位置，其為實際屬性名稱的替代項目。表達式屬性名稱的開頭必須是井字號 (#)，其後接著一或多個英數字元和底線 (_) 字元。

本節說明必須使用表達式屬性名稱的數種狀況。

Note

本節中的範例使用 AWS Command Line Interface (AWS CLI)。如需程式設計語言特定的程式碼範例，請參閱[開始使用 DynamoDB 和開發套件 AWS](#)。

主題

- [保留字](#)
- [包含特殊字元的屬性名稱](#)
- [巢狀屬性](#)
- [重複屬性名稱](#)

保留字

有時候您需要撰寫的表達式可能會包含與 DynamoDB 保留字衝突的屬性名稱。(如需完整的保留字清單，請參閱[DynamoDB 中的保留字](#)。)

例如，下列 AWS CLI 範例會失敗，因為 COMMENT 是保留字。

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "Comment"
```

若要解決此情況，您可以將 Comment 取代為表達式屬性名稱 (例如 #c)。# (井字號) 是必要項目，指出這是屬性名稱的預留位置。此 AWS CLI 範例現在如下所示。

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "#c" \  
  --expression-attribute-names '{"#c":"Comment"}'
```

Note

若屬性名稱的開頭是數字、包含空格或包含保留字，您必須使用表達式屬性名稱來取代表達式中的該屬性名稱。

包含特殊字元的屬性名稱

在表達式中，點號(".") 會被解譯為文件路徑中的分隔符號字元。不過，DynamoDB 允許您使用點字元和其他特殊字元，例如連字號 (" - ") 作為屬性名稱的一部分。這在某些情況下可能會模稜兩可。為了示範，假設您想要從 ProductCatalog 項目擷取 Safety.Warning 屬性 (請參閱[使用表達式時指定項目屬性](#))。

假設您想要使用投射表達式來存取 Safety.Warning。

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "Safety.Warning"
```

DynamoDB 將會傳回空的結果，而不是預期字串 (Always wear a helmet)。原因是 DynamoDB 會將表達式中的點解譯為文件路徑分隔符號。在此情況下，您必須定義表達式屬性名稱 (例如 #sw) 做為 Safety.Warning 的替代項目。您接著可以使用下列投射表達式。

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "#sw" \  
  --expression-attribute-names '{"#sw":"Safety.Warning"}'
```

DynamoDB 接著會傳回正確結果。

Note

若屬性名稱包含點 (「.」) 或連字號 (「-」)，您必須使用表達式屬性名稱來取代表達式中的該屬性名稱。

巢狀屬性

假設您想要使用以下投射表達式來存取巢狀屬性 `ProductReviews.OneStar`。

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "ProductReviews.OneStar"
```

結果會包含所有一星產品檢閱，這是預期結果。

但若決定改為使用表達式屬性名稱時可以怎麼做？例如，您要將 `#pr1star` 定義為 `ProductReviews.OneStar` 的替代項目時會發生什麼事？

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "#pr1star" \  
  --expression-attribute-names '{"#pr1star":"ProductReviews.OneStar"}'
```

DynamoDB 會傳回空結果，而不是原先預期的一星檢閱映射。原因是 DynamoDB 會將表達式屬性名稱中的點解譯為屬性名稱內的字元。DynamoDB 評估表達式屬性名稱 `#pr1star` 時，會判斷 `ProductReviews.OneStar` 指的是純量屬性 (但這不是所需的屬性)。

正確的作法為定義文件路徑中每個元素的表達式屬性名稱：

- `#pr` – `ProductReviews`
- `#1star` – `OneStar`

您接著可以使用 `#pr.#1star` 做為投射表達式。

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "#pr.#1star"
```

```
--projection-expression "#pr.#1star" \  
--expression-attribute-names '{"#pr":"ProductReviews", "#1star":"OneStar"}'
```

DynamoDB 接著會傳回正確結果。

重複屬性名稱

當您需要反覆參考相同的屬性名稱時，表達式屬性名稱十分有用。例如，請考慮以下表達式，從 ProductCatalog 項目擷取部分檢閱。

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "ProductReviews.FiveStar, ProductReviews.ThreeStar,  
ProductReviews.OneStar"
```

更精確地來說，您可以將 ProductReviews 取代為表達式屬性名稱 (例如 #pr)。修訂過的表達式如下所示。

- #pr.FiveStar, #pr.ThreeStar, #pr.OneStar

```
aws dynamodb get-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"123"}}' \  
  --projection-expression "#pr.FiveStar, #pr.ThreeStar, #pr.OneStar" \  
  --expression-attribute-names '{"#pr":"ProductReviews"}'
```

如果您定義表達式屬性名稱，則必須在整個表達式中一致地使用它。您也無法省略 # 符號。

表達式屬性值

如果您需要比較屬性與值，請將表達式屬性值定義為預留位置。Amazon DynamoDB 中的表達式屬性值是您要比較的實際值的替代項目 (在執行時間之前您可能不知道這些值為何)。表達式屬性值的開頭必須是冒號 (:)，且後面跟隨一或多個英數字元。

例如，假設您想要傳回提供 Black 顏色且價格不超過 500 (含) 的所有可用 ProductCatalog 項目。您可以搭配篩選條件表達式使用 Scan 操作，如此 AWS Command Line Interface (AWS CLI) 範例所示。

```
aws dynamodb scan \  
  --table-name ProductCatalog \  
  --filter-expression "#pr.Color = :Black AND #pr.Price <= :500"
```

```
--filter-expression "contains(Color, :c) and Price <= :p" \  
--expression-attribute-values file://values.json
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{  
  ":c": { "S": "Black" },  
  ":p": { "N": "500" }  
}
```

Note

Scan 操作會讀取資料表或中的每個項目。因此您應該避免針對大型資料表使用 Scan。篩選條件表達式會套用至 Scan 結果，且不符合篩選條件表達式的項目會遭到捨棄。

如果您定義表達式屬性值，則必須在整個表達式中一致地使用它。您也無法省略：符號。

表達式屬性值可搭配索引鍵條件表達式、條件表達式、更新表達式與篩選條件表達式使用。

Note

如需程式設計語言特定的程式碼範例，請參閱[開始使用 DynamoDB 和開發套件 AWS](#)。

條件表達式

若要操控 Amazon DynamoDB 資料表中的資料，您可以使用 PutItem、UpdateItem 和 DeleteItem 操作。(您也可以使用 BatchWriteItem，透過單一呼叫執行多個 PutItem 或 DeleteItem 操作)。

針對這些資料操控操作，您可以指定條件表達式來判斷應該修改的項目。如果條件表達式評估為 true，則操作會成功；否則，操作會失敗。

PutItemUpdateItem、和DeleteItem作業有一個ReturnValues參數，可用來傳回屬性值在您修改之前或之後所顯示的屬性值。如需詳細資訊，請參閱[ReturnValues](#)。

以下是一些 AWS Command Line Interface (AWS CLI) 使用條件運算式的範例。這些範例是以 [使用表達式時指定項目屬性](#) 中引進的 ProductCatalog 資料表為基礎。此資料表的分割區索引鍵是 Id；沒有排序索引鍵。以下 PutItem 操作會建立將在範例中參考的範例 ProductCatalog 項目。

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

--item 的引數會存放在 item.json 檔案中。(為求簡化，只會使用一些項目屬性)。

```
{  
  "Id": {"N": "456" },  
  "ProductCategory": {"S": "Sporting Goods" },  
  "Price": {"N": "650" }  
}
```

主題

- [條件式放置](#)
- [條件式刪除](#)
- [條件式更新](#)
- [條件式表達式範例](#)
- [比較運算子與函數參考](#)

條件式放置

PutItem 操作將會覆寫具有相同主索引鍵 (如有) 的項目。若您想要避免這種情況，請使用條件表達式。只有在相關項目還沒有相同的主索引鍵時，才會允許繼續寫入。

下列範例會在嘗試寫入操作之前，使用 attribute_not_exists() 檢查資料表中是否存在主索引鍵。

Note

如果您的主索引鍵同時包含分割區索引鍵 (pk) 和排序索引鍵 (sk)，則在嘗試寫入操作之前，參數會檢查 attribute_not_exists(pk) 和 attribute_not_exists(sk) 評估為 true 或 false。

```
aws dynamodb put-item \  
  --table-name ProductCatalog \  
  --item file://item.json \  
  --condition-expression attribute_not_exists(pk)
```



```
--condition-expression "attribute_not_exists(Id)"
```

如果條件表達式評估為 `false`，則 DynamoDB 會傳回以下錯誤訊息：The conditional request failed (條件式請求失敗)。

Note

如需 `attribute_not_exists` 和其他函數的詳細資訊，請參閱 [比較運算子與函數參考](#)。

條件式刪除

若要執行條件式刪除，您可以搭配使用 `DeleteItem` 操作與條件表達式。條件表達式必須評估為 `true`，操作才會成功；否則，操作會失敗。

從 [條件表達式](#) 考慮項目。

```
{
  "Id": {
    "N": "456"
  },
  "Price": {
    "N": "650"
  },
  "ProductCategory": {
    "S": "Sporting Goods"
  }
}
```

假設您想要刪除項目，但只限在下列條件下：

- `ProductCategory` 是 "Sporting Goods" 或 "Gardening Supplies"。
- `Price` 介於 500 與 600 之間。

以下範例會嘗試刪除項目。

```
aws dynamodb delete-item \
  --table-name ProductCatalog \
  --key '{"Id":{"N":"456"}}' \
  --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (Price between :lo
and :hi)" \
```

```
--expression-attribute-values file://values.json
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{
  ":cat1": {"S": "Sporting Goods"},
  ":cat2": {"S": "Gardening Supplies"},
  ":lo": {"N": "500"},
  ":hi": {"N": "600"}
}
```

Note

在條件表達式中，: (冒號字元) 表示表達式屬性值 (即實際值的預留位置)。如需詳細資訊，請參閱 [表達式屬性值](#)。

如需 IN、AND 和其他關鍵字的詳細資訊，請參閱 [比較運算子與函數參考](#)。

在此範例中，ProductCategory 比較會評估為 true，但 Price 比較會評估為 false。這會導致條件表達式評估為 false，使 DeleteItem 操作失敗。

條件式更新

若要執行條件式更新，您可以搭配使用 UpdateItem 操作與條件表達式。條件表達式必須評估為 true，操作才會成功；否則，操作會失敗。

Note

UpdateItem 也支援更新表達式；其中，您可以指定想要對項目進行的修改。如需詳細資訊，請參閱 [更新表達式](#)。

假設您已開始使用 [條件表達式](#) 中所顯示的項目。

```
{
  "Id": { "N": "456"},
  "Price": {"N": "650"},
  "ProductCategory": {"S": "Sporting Goods"}
}
```

以下範例會執行 UpdateItem 操作。其試圖將產品的 Price 減少 75，但如果目前的 Price 小於或等於 500，條件表達式會阻止更新。

```
aws dynamodb update-item \  
  --table-name ProductCatalog \  
  --key '{"Id": {"N": "456"}}' \  
  --update-expression "SET Price = Price - :discount" \  
  --condition-expression "Price > :limit" \  
  --expression-attribute-values file://values.json
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{  
  ":discount": { "N": "75"},  
  ":limit": { "N": "500"}  
}
```

如果起始 Price 是 650，則 UpdateItem 操作會將 Price 減少為 575。如果您再次執行 UpdateItem 操作，則 Price 會減少為 500。如果您執行它第三次，則條件表達式會評估為 false，而更新會失敗。

Note

在條件表達式中，: (冒號字元) 表示表達式屬性值 (即實際值的預留位置)。如需詳細資訊，請參閱 [表達式屬性值](#)。

如需 ">" 和其他運算子的詳細資訊，請參閱 [比較運算子與函數參考](#)。

條件式表達式範例

如需下列範例中所用函數的詳細資訊，請參閱 [比較運算子與函數參考](#)。若要進一步了解如何指定表達式中的不同屬性類型，請參閱 [使用表達式時指定項目屬性](#)。

檢查項目中的屬性

您可以檢查任何屬性是否存在。如果條件表達式評估為 true，則操作會成功；否則，操作會失敗。

只有在產品沒有 Price 屬性時，以下範例才會使用 attribute_not_exists 來刪除產品。

```
aws dynamodb delete-item \  
  --table-name ProductCatalog \  
  --key '{"Id": {"N": "456"}}'
```

```
--key '{"Id": {"N": "456"}}' \  
--condition-expression "attribute_not_exists(Price)"
```

DynamoDB 也提供 `attribute_exists` 函數。只有在產品收到不佳的檢閱時，以下範例才會刪除產品。

```
aws dynamodb delete-item \  
  --table-name ProductCatalog \  
  --key '{"Id": {"N": "456"}}' \  
  --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

檢查屬性類型

您可以使用 `attribute_type` 函數，以檢查屬性值的資料類型。如果條件表達式評估為 `true`，則操作會成功；否則，操作會失敗。

下列範例只有在具有字串集合類型的 `Color` 屬性的情況下，才會使用 `attribute_type` 刪除產品。

```
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
```

的引數 `--expression-attribute-values` 會儲存在 `expression-attribute-values.json` 檔案中。

```
{  
  ":v_sub":{"S":"SS"}  
}
```

檢查字串起始值

您可以使用 `begins_with` 函數，來檢查字串屬性值是否以特定子字串做為開頭。如果條件表達式評估為 `true`，則操作會成功；否則，操作會失敗。

只有在 `begins_with` 映射的 `FrontView` 元素以特定值作為開頭時，下列範例才會使用 `Pictures` 刪除產品。

```
aws dynamodb delete-item \  
  --table-name ProductCatalog \  
  --key '{"Id": {"N": "456"}}' \  
  --condition-expression "begins_with(Pictures.FrontView, :v_sub)" \  
  --expression-attribute-values file://expression-attribute-values.json
```

```
--expression-attribute-values file://expression-attribute-values.json
```

的引數 `--expression-attribute-values` 會儲存在 `expression-attribute-values.json` 檔案中。

```
{
  ":v_sub":{"S":"http://"}
}
```

檢查集合中的元素

您可以使用 `contains` 函數，檢查集合中的元素或尋找字串內的子字串。如果條件表達式評估為 `true`，則操作會成功；否則，操作會失敗。

只有在 `Color` 字串集合具有含特定值的元素時，下列範例才會使用 `contains` 刪除產品。

```
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
```

的引數 `--expression-attribute-values` 會儲存在 `expression-attribute-values.json` 檔案中。

```
{
  ":v_sub":{"S":"Red"}
}
```

檢查屬性值的大小

您可以使用 `size` 函數，來檢查屬性值的大小。如果條件表達式評估為 `true`，則操作會成功；否則，操作會失敗。

只有在 `VideoClip` 二進位屬性大於 64000 位元組時，下列範例才會使用 `size` 刪除產品。

```
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
```

的引數 `--expression-attribute-values` 會儲存在 `expression-attribute-values.json` 檔案中。

```
{
  "v_sub":{"N":"64000"}
}
```

比較運算子與函數參考

本節說明在 Amazon DynamoDB 中撰寫篩選條件表達式和條件表達式所使用的內建函數與關鍵字。如需有關 DynamoDB 的函數和進程式設計的詳細資訊，請參閱 [使用 DynamoDB 和開發套件進程式設計 AWS](#) 和 [DynamoDB API 參考](#)。

主題

- [篩選條件和條件表達式的語法](#)
- [進行比較](#)
- [函數](#)
- [邏輯評估](#)
- [括號](#)
- [條件的優先順序](#)

篩選條件和條件表達式的語法

在以下語法摘要中，*operand* 可以是以下項目：

- 最上層屬性名稱，例如 Id、Title、Description 或 ProductCategory
- 參考巢狀屬性的文件路徑

```
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)
```

進行比較

使用這些比較子可比較運算元與某範圍的值或列舉值清單：

- $a = b$ ：如果 a 等於 b ，則為 true。
- $a <> b$ ：如果 a 不等於 b ，則為 true。
- $a < b$ ：如果 a 小於 b ，則為 true。
- $a <= b$ ：如果 a 小於或等於 b ，則為 true。
- $a > b$ ：如果 a 大於 b ，則為 true。
- $a >= b$ ：如果 a 大於或等於 b ，則為 true。

使用 BETWEEN 與 IN 關鍵字可比較運算元與某範圍的值或列舉值清單：

- a BETWEEN b AND c ：如果 a 大於或等於 b 且小於或等於 c ，則為 true。
- a IN (b , c , d)：如果 a 等於清單中的任何值 (例如， b 、 c 或 d 的任何一個)，則為 true。此清單最多可包含 100 個值，並以逗號分隔。

函數

使用下列函數可判斷項目中是否具有某屬性，或評估某屬性的值。這些函數名稱區分大小寫。針對巢狀屬性，您必須提供其完整文件路徑。

函式	描述
attribute_exists (<i>path</i>)	如果項目包含 <i>path</i> 指定的屬性，則為 true。

函式	描述
	<p>範例：檢查 Product 資料表中的項目是否具有側視圖。</p> <ul style="list-style-type: none">• <code>attribute_exists (#Pictures.#SideView)</code>
<code>attribute_not_exists (<i>path</i>)</code>	<p>如果項目中沒有 <code>path</code> 指定的屬性，則為 true。</p> <p>範例：檢查項目是否具有 <code>Manufacturer</code> 屬性。</p> <ul style="list-style-type: none">• <code>attribute_not_exists (Manufacturer)</code>

函式	描述
<code>attribute_type (<i>path</i>, <i>type</i>)</code>	<p>如果指定路徑的屬性是特定資料類型，則為 true。type 參數必須是下列其中一種：</p> <ul style="list-style-type: none">• S：字串• SS：字串集合• N：數字• NS：數字集合• B：二進位• BS：二進位集合• BOOL：布林值• NULL：Null• L：清單• M：映射 <p>您必須為 type 參數使用表達式屬性值。</p> <p>範例：檢查 <code>QuantityOnHand</code> 屬性的類型是否為清單。在此範例中，<code>:v_sub</code> 是字串 L 的預留位置。</p> <ul style="list-style-type: none">• <code>attribute_type (ProductReviews.FiveStar, :v_sub)</code> <p>您必須為 type 參數使用表達式屬性值。</p>

函式	描述
<code>begins_with (<i>path</i>, <i>substr</i>)</code>	<p>如果 <code>path</code> 指定的屬性以特定子字串開頭，則為 true。</p> <p>範例：檢查正視圖 URL 的前幾個字元是否為 <code>http://</code>。</p> <ul style="list-style-type: none"><code>begins_with (Pictures.FrontView, :v_sub)</code> <p>表達式屬性值 <code>:v_sub</code> 是 <code>http://</code> 的預留位置。</p>

函式	描述
<code>contains (path, operand)</code>	<p>如果 path 指定的屬性為下列其中一個項目，則為 true：</p> <ul style="list-style-type: none">• 內含特定子字串的 String。• 在組合中內含特定元素的 Set。• 在清單內含特定元素的 List。 <p>如果 path 指定的屬性是 String，則 operand 必須是 String。如果 path 指定的屬性是 Set，則 operand 必須是集合的元素類型。</p> <p>路徑和運算元必須不同。也就是說，contains (a, a) 會傳回錯誤。</p> <p>範例：檢查 Brand 屬性是否包含子字串 Company。</p> <ul style="list-style-type: none">• <code>contains (Brand, :v_sub)</code> <p>表達式屬性值 :v_sub 是 Company 的預留位置。</p> <p>範例：檢查產品是否提供紅色款式。</p> <ul style="list-style-type: none">• <code>contains (Color, :v_sub)</code> <p>表達式屬性值 :v_sub 是 Red 的預留位置。</p>

函式	描述
<code>size (<i>path</i>)</code>	<p>傳回代表屬性大小的數字。以下是可搭配 <code>size</code> 使用的有效資料類型。</p> <p>如果屬性的類型為 <code>String</code> , <code>size</code> 會傳回字串長度。</p> <p>範例：檢查字串 <code>Brand</code> 是否小於或等於 20 個字元。表達式屬性值 <code>:v_sub</code> 是 20 的預留位置。</p> <ul style="list-style-type: none"><code>size (Brand) <= :v_sub</code> <p>如果屬性的類型為 <code>Binary</code> , <code>size</code> 會傳回屬性值中的位元組數目。</p> <p>範例：假設 <code>ProductCatalog</code> 項目具有名為 <code>VideoClip</code> 的二進制屬性，其中包含使用中產品的短片。以下表達式會檢查 <code>VideoClip</code> 是否超過 64,000 個位元組。表達式屬性值 <code>:v_sub</code> 是 64000 的預留位置。</p> <ul style="list-style-type: none"><code>size(VideoClip) > :v_sub</code> <p>如果屬性是 <code>Set</code> 資料類型，<code>size</code> 會傳回集合中的元素數目。</p> <p>範例：檢查產品是否提供多色款式。表達式屬性值 <code>:v_sub</code> 是 1 的預留位置。</p> <ul style="list-style-type: none"><code>size (Color) < :v_sub</code>

函式	描述
	<p>如果屬性的類型是 List 或 Map，size 會傳回子元素數目。</p> <p>範例：檢查 OneStar 評論數目是否超過特定閾值。表達式屬性值 :v_sub 是 3 的預留位置。</p> <ul style="list-style-type: none"> <pre>size(ProductReviews.OneStar) > :v_sub</pre>

邏輯評估

使用 AND、OR 與 NOT 關鍵字可執行邏輯評估。在下列清單中，*a* 與 *b* 代表要評估的條件。

- *a* AND *b*：如果 *a* 與 *b* 皆為 true，則為 true。
- *a* OR *b*：如果 *a* 或 *b* (或兩者) 為 true，則為 true。
- NOT *a*：如果 *a* 為 false，則為 true。如果 *a* 為 true，則為 false。

以下是運算中 AND 的程式碼範例。

```
dynamodb-local (*)> select * from exprtest where a > 3 and a < 5;
```

括號

使用括號可變更邏輯評估的優先順序。例如，假設條件 *a* 與 *b* 皆為 true，且條件 *c* 為 false。下列表達式會評估為 true：

- *a* OR *b* AND *c*

不過，如果您以括號括住條件，則會先評估該條件。例如，下列表達式會評估為 false：

- (*a* OR *b*) AND *c*

Note

您可以在表達式中巢狀使用括號。最內部的條件最先評估。

以下是邏輯評估中帶括號的程式碼範例。

```
dynamodb-local (*)> select * from exprtest where attribute_type(b, string)
or ( a = 5 and c = "coffee");
```

條件的優先順序

DynamoDB 使用下列優先順序規則，從左到右評估條件：

- = <> < <= > >=
- IN
- BETWEEN
- attribute_exists attribute_not_exists begins_with contains
- 括號
- NOT
- AND
- OR

更新表達式

UpdateItem 操作會更新現有項目，如尚不存在，則會將新項目新增到資料表中。您必須提供要更新項目的索引鍵。您也必須提供更新表達式，指出您想要修改的屬性以及您想要指派給它們的值。

更新表達式會指定 UpdateItem 如何修改項目的屬性，例如設定純量值或將元素從清單或映射中刪除。

以下是更新表達式的語法摘要。

```
update-expression ::=
  [ SET action [, action] ... ]
  [ REMOVE action [, action] ... ]
  [ ADD action [, action] ... ]
```

```
[ DELETE action [, action] ...]
```

更新表達式包含一或多個子句。每個子句的開頭是 SET、REMOVE、ADD 或 DELETE 關鍵字。您可以依任意順序在更新表達式中包含其中任何子句。不過，每個動作關鍵字都只能出現一次。

在每個子句內，會有以逗號分隔的一或多個動作。每個動作都會代表資料修改。

本節中的範例是以[投射表達式](#)中所顯示的 ProductCatalog 項目為基礎。

以下主題涵蓋了 SET 動作的一些不同使用案例。

主題

- [SET — 修改或新增項目屬性](#)
- [REMOVE — 刪除項目中的屬性](#)
- [ADD — 更新數字與集合](#)
- [DELETE — 移除集合中的元素](#)
- [使用多個更新表達式](#)

SET — 修改或新增項目屬性

在更新表達式中使用 SET 動作，將一或多個屬性新增至項目。這些屬性如已存在，每一個都會為新值所覆寫。

您也可以使用 SET 來加減類型為 Number 的屬性。若要執行多個 SET 動作，請使用逗號分隔它們。

在下列語法摘要中：

- *path* 元素是項目的文件路徑。
- *operand* 元素可以是項目或函數的文件路徑。

```
set-action ::=
    path = value

value ::=
    operand
    | operand '+' operand
    | operand '-' operand

operand ::=
```

path | function

下列 PutItem 操作會建立範例參考的範例項目。

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

--item 的引數會存放在 item.json 檔案中。(為求簡化，只會使用一些項目屬性)。

```
{  
  "Id": {"N": "789"},  
  "ProductCategory": {"S": "Home Improvement"},  
  "Price": {"N": "52"},  
  "InStock": {"BOOL": true},  
  "Brand": {"S": "Acme"}  
}
```

主題

- [修改屬性](#)
- [新增清單和映射](#)
- [將元素新增至清單](#)
- [新增巢狀映射屬性](#)
- [增加和減少數值屬性](#)
- [將元素附加至清單](#)
- [防止覆寫現有屬性](#)

修改屬性

Example

更新 ProductCategory 和 Price 屬性。

```
aws dynamodb update-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"789"}}' \  
  --update-expression "SET ProductCategory = :c, Price = :p" \  
  --expression-attribute-values file://values.json \  
  --expression-attribute-names file://names.json
```



```
--return-values ALL_NEW
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{
  "c": { "S": "Hardware" },
  "p": { "N": "60" }
}
```

Note

在 UpdateItem 操作中，--return-values ALL_NEW 會讓 DynamoDB 傳回在更新後所顯示的項目。

新增清單和映射

Example

新增清單和映射。

```
aws dynamodb update-item \
  --table-name ProductCatalog \
  --key '{"Id":{"N":"789"}}' \
  --update-expression "SET RelatedItems = :ri, ProductReviews = :pr" \
  --expression-attribute-values file://values.json \
  --return-values ALL_NEW
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{
  "ri": {
    "L": [
      { "S": "Hammer" }
    ]
  },
  "pr": {
    "M": {
      "FiveStar": {
        "L": [
          { "S": "Best product ever!" }
        ]
      }
    }
  }
}
```

```
    ]
  }
}
}
```

將元素新增至清單

Example

將新的屬性新增至 RelatedItems 清單。(請記住，清單元素的開頭為零，因此 [0] 代表清單中的第一個元素、[1] 代表第二個元素，以此類推)。

```
aws dynamodb update-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"789"}}' \  
  --update-expression "SET RelatedItems[1] = :ri" \  
  --expression-attribute-values file://values.json \  
  --return-values ALL_NEW
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{  
  ":ri": { "S": "Nails" }  
}
```

Note

當您使用 SET 更新清單元素時，會將該元素的內容取代為您所指定的新資料。如果元素尚未存在，則 SET 會在清單結尾附加新的元素。

如果您在單一 SET 操作中新增多個元素，則會依元素號碼依序排序元素。

新增巢狀映射屬性

Example

新增一些巢狀映射屬性。

```
aws dynamodb update-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"789"}}' \  
  --update-expression "SET RelatedItems[1] = :ri" \  
  --expression-attribute-values file://values.json \  
  --return-values ALL_NEW
```

```
--table-name ProductCatalog \  
--key '{"Id":{"N":"789"}}' \  
--update-expression "SET #pr.#5star[1] = :r5, #pr.#3star = :r3" \  
--expression-attribute-names file://names.json \  
--expression-attribute-values file://values.json \  
--return-values ALL_NEW
```

--expression-attribute-names 的引數會存放在 names.json 檔案中。

```
{  
  "#pr": "ProductReviews",  
  "#5star": "FiveStar",  
  "#3star": "ThreeStar"  
}
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{  
  ":r5": { "S": "Very happy with my purchase" },  
  ":r3": {  
    "L": [  
      { "S": "Just OK - not that great" }  
    ]  
  }  
}
```

增加和減少數值屬性

您可以新增或扣除現有數值屬性。若要執行此作業，請使用 + (加號) 和 - (減號) 運算子。

Example

減少項目的 Price。

```
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
```

若要增加 Price，您要在更新表達式中使用 + 運算子。

將元素附加至清單

您可以將元素新增至清單結尾。若要執行此作業，請搭配使用 SET 與 `list_append` 函數 (函數名稱區分大小寫。) `list_append` 函數為 SET 動作專有，而且只能用於更新表達式。語法如下。

- `list_append (list1, list2)`

此函數需要兩個列表作為輸入，並將所有元素從 `list2` 附加到 `list1`。

Example

在[將元素新增至清單](#)中，您會建立 `RelatedItems` 清單，並使用兩個元素填入它：Hammer 和 Nails。現在，您可在 `RelatedItems` 結尾多附加兩個元素。

```
aws dynamodb update-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"789"}}' \  
  --update-expression "SET #ri = list_append(#ri, :vals)" \  
  --expression-attribute-names '{"#ri": "RelatedItems"}' \  
  --expression-attribute-values file://values.json \  
  --return-values ALL_NEW
```

`--expression-attribute-values` 的引數會存放在 `values.json` 檔案中。

```
{  
  ":vals": {  
    "L": [  
      { "S": "Screwdriver" },  
      { "S": "Hacksaw" }  
    ]  
  }  
}
```

最後，您將另一個元素附加到 `RelatedItems` 的開頭。若要執行此作業，請切換 `list_append` 元素的順序。(請記住，`list_append` 採用兩份清單做為輸入，並將第二份清單附加至第一份清單。)

```
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
```

```
--return-values ALL_NEW
```

產生的 `RelatedItems` 屬性現在會包含五個元素，順序如下：`Chisel`、`Hammer`、`Nails`、`Screwdriver`、`Hacksaw`。

防止覆寫現有屬性

如果您想要避免覆寫現有屬性，則可以搭配使用 `SET` 與 `if_not_exists` 函數。(函數名稱區分大小寫。) `if_not_exists` 函數為 `SET` 動作專有，而且只能用於更新表達式。語法如下。

- `if_not_exists (path, value)`

如果項目不包含指定 `path` 中的屬性，則 `if_not_exists` 會評估為 `value`；否則，它會評估為 `path`。

Example

設定項目的 `Price`，但只有在項目還沒有 `Price` 屬性時。(如果 `Price` 已存在，則不會發生任何事。)

```
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 — 刪除項目中的屬性

在更新表達式中使用 `REMOVE` 動作，從而在 Amazon DynamoDB 中移除項目的一或多個屬性。若要執行多個 `REMOVE` 動作，請使用逗號分隔它們。

下列是更新表達式中 `REMOVE` 的語法摘要。唯一的運算元是您想要移除的屬性文件路徑。

```
remove-action ::=  
path
```

Example

移除項目中的一些屬性 (如果屬性不存在，則不會發生任何事。)

```
aws dynamodb update-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"789"}}' \  
  --update-expression "REMOVE Price" \  
  --return-values ALL_NEW
```

```
--table-name ProductCatalog \  
--key '{"Id":{"N":"789"}}' \  
--update-expression "REMOVE Brand, InStock, QuantityOnHand" \  
--return-values ALL_NEW
```

移除清單中的元素

您可以使用 REMOVE 刪除清單中的個別元素。

Example

在 [將元素附加至清單](#) 中，您修改清單屬性 (RelatedItems)，讓它包含五個元素：

- [0]—Chisel
- [1]—Hammer
- [2]—Nails
- [3]—Screwdriver
- [4]—Hacksaw

下列 AWS Command Line Interface (AWS CLI) 範例會從清單 Nails 中刪除 Hammer 和

```
aws dynamodb update-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"789"}}' \  
  --update-expression "REMOVE RelatedItems[1], RelatedItems[2]" \  
  --return-values ALL_NEW
```

移除 Hammer 和 Nails 之後，會轉移其餘的元素。此清單現在包含下列項目：

- [0]—Chisel
- [1]—Screwdriver
- [2]—Hacksaw

ADD — 更新數字與集合


Note

一般而言，建議使用 SET，而非 ADD。

在更新表達式中使用 ADD 動作，將新的屬性及其值新增至項目。

如果屬性已存在，則 ADD 的行為取決於屬性的資料類型：

- 如果屬性是數字，而您要新增的值也是數字，則此值會以數學方式新增到現有屬性。(如果值是負數，則會從現有屬性減去。)
- 如果屬性是集合，而您要新增的值也是集合，則此值會附加至現有集合。

 Note

ADD 動作只支援數字和集合資料類型。

若要執行多個 ADD 動作，請使用逗號分隔它們。

在下列語法摘要中：

- *path* 元素是屬性的文件路徑。屬性必須是 Number 或集合資料類型。
- *value* 元素是您要新增至屬性的數字 (針對 Number 資料類型)，或要附加至屬性的集合 (針對集合類型)。

```
add-action ::=  
  path value
```

以下主題涵蓋了 ADD 動作的一些不同使用案例。

主題

- [新增數字](#)
- [將元素新增至集合](#)

新增數字

假設 QuantityOnHand 屬性不存在。下列 AWS CLI 範例會設 QuantityOnHand 定為 5。

```
aws dynamodb update-item \  
  --table-name ProductCatalog \  
  --update-expression 'ADD QuantityOnHand 5'
```

```
--key '{"Id":{"N":"789"}}' \  
--update-expression "ADD QuantityOnHand :q" \  
--expression-attribute-values '{"q": {"N": "5"}}' \  
--return-values ALL_NEW
```

現在已有 `QuantityOnHand`，您可以重新執行範例，`QuantityOnHand` 每次的增量為 5。

將元素新增至集合

假設 `Color` 屬性不存在。下列 AWS CLI 範例會將 `Color` 設定為具有兩個元素的字串集。

```
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
```

現在已有 `Color`，您可以在其中新增更多元素。

```
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 — 移除集合中的元素

Important

DELETE 動作僅支援 Set 資料類型。

在更新表達式中使用 DELETE 動作，移除集合中的一或多個元素。若要執行多個 DELETE 動作，請使用逗號分隔它們。

在下列語法摘要中：

- *path* 元素是屬性的文件路徑。屬性必須是集合資料類型。

- *subset* 是您要從 *path* 刪除的一或多個元素。您必須指定 *subset* 做為集合類型。

```
delete-action ::=
  path subset
```

Example

在 [將元素新增至集合](#) 中，您會建立 Color 字串集。本範例會從該集合中移除一些元素。

```
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
```

使用多個更新表達式

您可以在單一陳述式中使用多個更新表達式。

Example

如果您想要修改屬性的值並完全移除其他屬性，您可以在單一陳述式中使用 SET 和 REMOVE 動作。此操作會將 Price 值減少為 15，同時也會從項目中移除 InStock 屬性。

```
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

如果您想要新增至清單，同時變更其他屬性的值，您可以在單一陳述式中使用兩個 SET 動作。此操作會將「釘子」新增到 RelatedItems 清單屬性中，並將 Price 值設定為 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
```

生存時間 (TTL)

DynamoDB 的存留時間 (TTL) 是一種符合成本效益的方法，可用來刪除不再相關的項目。TTL 可讓您定義每個項目的到期時間戳記，以指出不再需要某個項目的時間。DynamoDB 會在過期時間後的幾天內自動刪除過期的項目，而不會耗用寫入輸送量。

若要使用 TTL，請先在資料表上啟用它，然後定義特定屬性來儲存 TTL 到期時間戳記。時間戳記必須以秒粒度儲存在 [Unix 紀元時間格式](#)。每次建立或更新項目時，您都可以計算到期時間，並將其儲存在 TTL 屬性中。

具有有效、過期 TTL 屬性的料件可隨時被系統刪除，通常會在其到期後的幾天內刪除。您仍然可以更新擱置刪除的過期項目，包括變更或移除其 TTL 屬性。更新過期的項目時，我們建議您使用條件運算式，以確定之後未刪除該項目。使用篩選器運算式從[掃描](#)和[查詢](#)結果中移除過期的項目。

刪除項目的運作方式與透過一般刪除作業刪除的項目類似。刪除後，項目會以服務刪除 (而非使用者刪除) 的形式進入 DynamoDB Streams，並且會像其他刪除作業一樣，從本機次要索引和全域次要索引中移除。

如果您使用的是[全域表 2019.11.21 \(目前\) 版本](#)的全域表，而且您也使用 TTL 功能，DynamoDB 會將 TTL 刪除項目複寫到所有複本表格。初始 TTL 刪除不會耗用 TTL 到期發生之區域中的寫入容量單位 (WCU)。不過，複寫至複本表格的 TTL 刪除會在使用佈建容量時耗用複寫的寫入容量單位，或在使用隨需容量模式時，在每個複本區域中使用複寫寫入單位，且會收取適用的費用。

如需 TTL 的詳細資訊，請參閱下列主題：

主題

- [啟用存留時間 \(TTL\)](#)
- [計算存活時間 \(TTL\)](#)
- [使用過期的物品](#)

啟用存留時間 (TTL)

您可以在 Amazon DynamoDB 主控台中啟用 TTL、AWS Command Line Interface (AWS CLI)，或搭配任何假設的開發套件使用 [Amazon DynamoDB API 參考](#)。AWS 在所有分割區上啟用 TTL 大約需要一個小時。

使用主控台啟用 DynamoDB 支援 AWS

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 選擇 Tables (資料表)，然後選擇您想要修改的資料表。
3. 在 [其他設定] 索引標籤的 [存留時間 (TTL)] 區段中，選擇 [開啟] 以啟用 TTL。

The screenshot displays the 'Additional settings' tab for a DynamoDB table. It is divided into several sections:

- Read/write capacity:** Includes an 'Edit' button and a description: 'The read/write capacity mode controls how you are charged for read and write throughput and how you manage capacity.' The 'Capacity mode' is set to 'Provisioned'.
- Table capacity:** A table comparing 'Read capacity auto scaling' and 'Write capacity auto scaling'. Both are set to 'On'. It also lists 'Provisioned read/write capacity units' (5), 'Provisioned range for reads/writes' (1 - 10), and 'Target read/write capacity utilization' (70%).
- Estimated read/write capacity cost:** A section with a right-pointing arrow.
- Auto scaling activities (0):** Includes a search bar for 'Find events', a refresh button, and a table with columns: Start time, End time, Target, Capacity unit, Description, and Status. The table is empty, with a message: 'No auto scaling activities found. There are no auto scaling activities for the table or its global secondary indexes.'
- Time to Live (TTL):** Includes an 'Info' link, a description: 'Automatically delete expired items from a table.', a 'Run preview' button, and a 'Turn on' button (highlighted with a red box). Below this, the 'TTL status' is shown as 'Off'.

4. 在資料表上啟用 TTL 時，DynamoDB 會要求您識別服務在判斷項目是否符合過期資格時將尋找的特定屬性名稱。TTL 屬性名稱 (如下所示) 區分大小寫，且必須與讀取和寫入作業中定義的屬性相符。不匹配將導致過期的項目被取消刪除。若要重新命名 TTL 屬性，您必須先停用 TTL，然後再重新啟用該屬性，然後再重新啟用該屬性。一旦禁用，TTL 將繼續處理大約 30 分鐘的刪除。必須在還原的資料表上重新設定 TTL。

[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.

September 13, 2023, 15:28:52 (UTC-06:00)

 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.

5. (可選) 您可以通過模擬到期日期和時間並匹配一些項目來執行測試。這會為您提供項目的範例清單，並確認有項目包含所提供的 TTL 屬性名稱以及到期時間。

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.

December 11, 2023, 16:58:01 (UTC-07:00)

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>

啟用 TTL 之後，當您在 DynamoDB 主控台上檢視項目時，TTL 屬性會標記為 TTL。您可以將滑鼠游標懸停到屬性上，來檢視項目過期的日期和時間。

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...	1694559479	1702339079
<input type="checkbox"/>	6734d779-5d71-47f3-ae4a-4b617...	52d667bd-cd9d-48a4-9a66-3bf...	1694559479	1702339079
<input type="checkbox"/>	cdb74466-0b36-41cd-9b39-cbe41...	52965e04-cb1a-4089-b891-9a1...	1694559479	1702339079
<input type="checkbox"/>	70aba065-a9d3-40f3-bd64-0d34c...	3272c168-4de2-4edf-a253-e02...	1694559479	1702339079
<input type="checkbox"/>	54caf925-843f-4966-b1e3-95530...	5e723d06-877d-4572-808b-e8d...	1694559479	1702339079
<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

使用 API 啟用 DynamoDB 支援 TTL

Python

您可以使用該[UpdateTimeToLive](#)操作啟用帶有代碼的 TTL。

```
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')

```

您可以使用描述表格 TTL 狀態的 [DescribeTimeToLive](#) 作業來確認 TTL 是否已啟用。狀TimeToLive態為ENABLED或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

您可以使用該 [UpdateTimeToLiveCommand](#) 操作啟用帶有代碼的 TTL。

```

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');
```

使用「啟用存留時間」 AWS CLI

1. 啟用 TTLExample 表中的 TTL。

```
aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
specification "Enabled=true, AttributeName=ttl"
```

2. 描述 TTLExample 表中的 TTL。

```
aws dynamodb describe-time-to-live --table-name TTLExample
{
  "TimeToLiveDescription": {
    "AttributeName": "ttl",
    "TimeToLiveStatus": "ENABLED"
  }
}
```

3. 使用 BASH shell 及 AWS CLI，將項目新增至已設定存留時間屬性的 TTLExample 資料表。

```
EXP=`date -d '+5 days' +%s`
```



```
aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl": {"N": "'$EXP'}}'
```

此範例建立的過期時間為從目前的日期開始加 5 天。然後，將過期時間轉換成 Epoch 時間格式，最終將項目新增到 "TTLExample" 表。

Note

設定存留時間過期數值的其中一種方式，是計算要新增到過期時間的秒數。例如，5 天等於 432,000 秒。但是通常建議從日期開始操作。

取得目前時間的 Epoch 時間格式非常容易，如以下範例所示。

- Linux 終端機 : `date +%s`
- Python : `import time; int(time.time())`
- Java : `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

使用以下方式啟用動態 B TTL AWS CloudFormation

1. 啟用 TTLExample 表中的 TTL。

```
aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-specification "Enabled=true, AttributeName=ttl"
```

2. 描述 TTLExample 表中的 TTL。

```
aws dynamodb describe-time-to-live --table-name TTLExample
{
  "TimeToLiveDescription": {
    "AttributeName": "ttl",
    "TimeToLiveStatus": "ENABLED"
  }
}
```

3. 使用 BASH shell 及 AWS CLI，將項目新增至已設定存留時間屬性的 TTLExample 資料表。

```
EXP=`date -d '+5 days' +%s`
```

```
aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl": {"N": "'$EXP'}}'
```

此範例建立的過期時間為從目前的日期開始加 5 天。然後，將過期時間轉換成 Epoch 時間格式，最終將項目新增到 "TTLExample" 表。

Note

設定存留時間過期數值的其中一種方式，是計算要新增到過期時間的秒數。例如，5 天等於 432,000 秒。但是通常建議從日期開始操作。

取得目前時間的 Epoch 時間格式非常容易，如以下範例所示。

- Linux 終端機 : `date +%s`
- Python : `import time; int(time.time())`
- Java : `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

計算存活時間 (TTL)

實作 TTL 的常用方法是根據項目的建立或上次更新的時間來設定項目的到期時間。這可以通過在 `createdAt` 和時間 `updatedAt` 戳中增加時間來完成。例如，新創建的項目的 TTL 可以設置為 `createdAt + 90` 天。更新項目時，TTL 可以重新計算為 `updatedAt + 90` 天。

計算的到期時間必須是 epoch 格式，以秒為單位。要考慮到期和刪除，TTL 過去不能超過五年。如果您使用任何其他格式，TTL 處理程序會忽略該項目。如果您將到期日設定為 future 某個時間，當您希望項目過期時，該項目將在該時間之後過期。例如，假設您將到期日設定為 1724241326 (即 2024 年 8 月 21 日星期一，11:55 : 26 (格林威治標準時間))。該項目將在指定時間後過期。

主題

- [建立項目並設定存留時間](#)
- [更新項目並重新整理存留時間](#)

建立項目並設定存留時間

下列範例會示範如何在建立新項目時計算到期時間，'expireAt' 以 JavaScript 及 'expirationDate' Python 的 TTL 屬性名稱。賦值語句獲得當前時間作為一個變量。在此範例中，到期時間的計算方式為目前時間 90 天。然後，時間會轉換為 epoch 格式，並在 TTL 屬性中儲存為整數資料類型。

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,
            'creationDate': current_time,
            'expirationDate': 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

在此請求中，我們添加了邏輯來計算新創建的項目的到期時間：

```
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');
```

更新項目並重新整理存留時間

這個例子是上一節中的一個延續。如果項目已更新，則可以重新計算到期時間。下列範例會將`expireAt`時間戳記重新計算為目前時間 90 天。

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')

```

更新操作的輸出顯示，雖然createdAt時間不變，但updatedAt和expireAt時間已更新。現在expireAt時間設定為上次更新時間起 90 天，即 2023 年 10 月 19 日 (星期四) 下午 1:27 : 15。

partition_key	createdAt	updatedAt	到期時間	屬性_1	屬性_2
某些值	2023-07-17 14:11:05.322 323	2023-07-19 13:27:15.213 423	1697722035	new_value	某些值

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');
```

本簡介中討論的 TTL 範例示範了一種方法，以確保僅將最近更新的項目保存在表格中。更新項目的壽命會延長，而未更新的項目會在建立後過期並免費刪除，從而減少儲存空間並維護乾淨的表格。

使用過期的物品

可以從讀取和寫入作業中篩選擱置刪除的過期項目。這在過期的數據不再有效且不應該被使用的情況下很有用。如果未篩選它們，它們將繼續顯示在讀取和寫入作業中，直到背景處理程序將它們刪除為止。

Note

這些項目仍會計入儲存和讀取成本，直到它們被刪除為止。

TTL 刪除可以在 DynamoDB Streams 中識別，但只能在發生刪除的區域中識別。複寫至全域資料表區域的 TTL 刪除作業在刪除複寫目標區域的 DynamoDB 串流中無法識別。

從讀取作業中篩選過期的項目

對於[掃描](#)和[查詢](#)等讀取作業，篩選器運算式可以篩選出擱置刪除的過期項目。如下列程式碼片段所示，篩選器運算式可以篩選出 TTL 時間等於或小於目前時間的項目。這是通過一個賦值語句來完成的，該語句將當前時間作為一個變量 (now)，該變量被轉換int為 epoch 時間格式。

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')
```

更新操作的輸出顯示，雖然createdAt時間不變，但updatedAt和expireAt時間已更新。現在expireAt時間設定為上次更新時間起 90 天，即 2023 年 10 月 19 日 (星期四) 下午 1:27 : 15。

partition_key	createdAt	updatedAt	到期時間	屬性_1	屬性_2
某些值	2023-07-17 14:11:05.322 323	2023-07-19 13:27:15.213 423	1697722035	new_value	某些值

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');
```

有條件地寫入過期的項目

條件運算式可用來避免對過期項目進行寫入。下面的代碼片段是條件式更新，用於檢查到期時間是否大於當前時間。如果為 true，寫入操作將繼續。

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')

```

更新操作的輸出顯示，雖然createdAt時間不變，但updatedAt和expireAt時間已更新。現在expireAt時間設定為上次更新時間起 90 天，即 2023 年 10 月 19 日 (星期四) 下午 1:27 : 15。

partition_key	createdAt	updatedAt	到期時間	屬性 _1	屬性 _2
某些值	2023-07-17 14:11:05.322 323	2023-07-19 13:27:15.213 423	1697722035	new_value	某些值

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');
```

在 DynamoDB Streams 中識別已刪除的項目

串流紀錄包含使用者身分欄位 `Records[<index>].userIdentity`。由 TTL 程序刪除的項目具有下列欄位：

```
Records[<index>].userIdentity.type  
"Service"  
  
Records[<index>].userIdentity.principalId  
"dynamodb.amazonaws.com"
```

下列 JSON 會顯示單一串流記錄的相關部分：

```
"Records": [  
  {  
    ...  
    "userIdentity": {  
      "type": "Service",  
      "principalId": "dynamodb.amazonaws.com"  
    }  
    ...  
  }  
]
```

使用項目：Java

您可以使用 AWS SDK for Java 文件 API 對資料表中的 Amazon DynamoDB 項目，執行一般的建立、讀取、更新與刪除 (CRUD) 操作。

Note

適用於 Java 的開發套件也提供物件持久性模型，讓您將用戶端類別映射至 DynamoDB 資料表。此方法可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱 [Java 1.x : DynamoDBMapper](#)。

本節包含 Java 範例，執行數個 Java 文件 API 項目動作及數個完整的工作範例。

主題

- [放入項目](#)
- [取得項目](#)
- [批次寫入：放入和刪除多個項目](#)
- [批次取得：取得多個項目](#)
- [更新項目](#)
- [刪除項目](#)
- [範例：使用 AWS SDK for Java Document API 進行 CRUD 操作](#)
- [範例：使用 AWS SDK for Java Document API 進行批次操作](#)
- [範例：使用 AWS SDK for Java Document API 處理二進位類型屬性](#)

放入項目

`putItem` 方法會將項目存放在資料表中。若已有該項目，其會取代整個項目。若不希望取代整個項目，而是只想要更新特定的屬性，可以使用 `updateItem` 方法。如需詳細資訊，請參閱 [更新項目](#)。

請遵循下列步驟：

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體，代表您要進行作業的資料表。
3. 建立 Item 類別的執行個體，代表新的項目。您必須指定新項目的主要索引鍵及其屬性。
4. 使用您在前述步驟中所建立的 `putItem`，呼叫 Table 物件的 `Item` 方法。

下列 Java 程式碼範例示範上述工作。程式碼會將新的項目寫入 ProductCatalog 表。

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);
```

在前述範例中，項目具有的屬性為純量 (String、Number、Boolean、Null)、集合 (String Set) 及文件類型 (List、Map)。

指定選用參數

除了必要的參數之外，您也可為 `putItem` 方法指定選用的參數。例如，下列 Java 程式碼範例會使用選用參數來指定上傳項目的條件。若無法滿足指定的條件，AWS SDK for Java 會擲回 `ConditionalCheckFailedException`。程式碼範例在 `putItem` 方法中指定了下列選用參數：

- `ConditionExpression` 會定義要求的條件。程式碼定義的條件是，具有相同主索引鍵的現有項目，只有在其 ISBN 屬性等於特定值時，才會被取代。
- 用於條件中的 `ExpressionAttributeValues` 映射。在本案例中，只需要一次替換：條件表達式中的預留位置 `:val`，會在執行時間更換為實際要查看的 ISBN 值。

下列範例會使用這些選用參數，新增新的書籍項目。

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 和 JSON 文件

您可以將 JSON 文件以屬性形式存放在 DynamoDB 資料表中。若要執行此作業，請使用 `Item` 的 `withJSON` 方法。此方法會剖析 JSON 文件，並將每個元素映射到原生的 DynamoDB 資料類型。

假設您希望存放下列 JSON 文件，其包含可完成特定產品訂單的廠商。

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"
  ]
  }
}
```

您可以使用 `withJSON` 方法，將此存放在 `ProductCatalog` 表，放在名為 `VendorInfo` 的 Map 屬性中。下列 Java 程式碼範例會示範如何執行此作業。

```
// 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);
```

取得項目

若要擷取單一項目，請使用 `getItem` 物件的 `Table` 方法。請遵循下列步驟：

1. 建立 DynamoDB 類別的執行個體。
2. 建立 `Table` 類別的執行個體，代表您要進行作業的資料表。
3. 呼叫 `getItem` 執行個體的 `Table` 方法。您必須指定希望擷取之項目的主要索引鍵。

下列 Java 程式碼範例示範上述步驟。程式碼會取得具有指定分割區索引鍵的項目。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Item item = table.getItem("Id", 210);
```

指定選用參數

除了必要的參數之外，您也可為 `getItem` 方法指定選用的參數。例如，下列 Java 程式碼範例使用選用方法，只擷取特定的屬性清單，以及指定強烈一致讀取。(若要進一步了解讀取一致性，請參閱「[讀取一致性](#)」。)

您可以使用 `ProjectionExpression`，只擷取特定的屬性或元素，而非整個項目。`ProjectionExpression` 可以使用文件路徑，指定最上層或巢狀屬性。如需詳細資訊，請參閱[投射表達式](#)。

`getItem` 方法的參數不會讓您指定讀取一致性。但您可以建立 `GetItemSpec`，它會提供低階 `getItem` 操作輸入的完整存取。以下程式碼範例會建立 `GetItemSpec`，然後使用該規格做為 `getItem` 方法的輸入。

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());
```

若要將 Item 以可供人閱讀的格式印出，請使用 `toJSONPretty` 方法。上個範例的輸出類似這樣。

```
{
  "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 和 JSON 文件

在 [PutItem 和 JSON 文件](#) 一節中，您將 JSON 文件存放在名為 `VendorInfo` 的 Map 屬性中。您可以使用 `getItem` 方法擷取 JSON 格式的整份文件。或者可以使用文件路徑標記法擷取文件中的部分元素。下列 Java 程式碼範例會示範這些技術。

```
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());
```

上個範例的輸出類似這樣。

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

您可以使用 `toJSON` 方法，將任一項目 (或其屬性) 轉換為格式化為 JSON 格式的字串。下列程式碼會擷取數個最上層及巢狀屬性，然後將結果以 JSON 印出。

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210)
    .withProjectionExpression("VendorInfo.V01, Title, Price");

Item item = table.getItem(spec);
System.out.println(item.toJSON());
```

輸出看起來如下。

```
{"VendorInfo":{"V01":{"Name":"Acme Books","Offices":["Seattle"]},"Price":30,"Title":"Book 210 Title"}
```

批次寫入：放入和刪除多個項目

批次寫入表示在一個批次中放入和刪除多個項目。您可利用 `batchWriteItem` 方法，在單一呼叫中對來自一或多個資料表的多個項目執行放入與刪除操作。以下為使用 AWS SDK for Java Document API 寫入或刪除多重項目的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 `TableWriteItems` 類別的執行個體，其描述針對資料表所有的寫入與刪除操作。若希望用單一批次寫入操作寫入多份資料表，您必須每份資料表都建立一個 `TableWriteItems` 執行個體。
3. 提供您在前述步驟中建立的 `batchWriteItem` 物件，藉以呼叫 `TableWriteItems` 方法。

4. 處理回應。您應該檢查回應中是否傳回任何未經處理的請求項目。如果您達到佈建輸送量配額或遇到一些其他暫時性錯誤，就可能發生這個狀況。此外，DynamoDB 會限制您在請求中指定的請求大小及操作次數。如果您超出這些限制，DynamoDB 會拒絕此請求。如需詳細資訊，請參閱 [Amazon DynamoDB 中的服務、帳戶和資料表配額](#)。

下列 Java 程式碼範例示範上述步驟。範例會在兩份資料表上執行 `batchWriteItem` 操作：Forum 和 Thread。對應的 `TableWriteItems` 物件會定義下列動作：

- 在 Forum 表中放入一個項目。
- 在 Thread 表中寫入及刪除一個項目。

該程式碼接著會呼叫 `batchWriteItem` 來執行操作。

```
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
```

如需運作範例，請參閱 [範例：使用 AWS SDK for Java Document API 進行批次寫入操作](#)。

批次取得：取得多個項目

您可利用 `batchGetItem` 方法，從一或多個資料表擷取多個項目。若要擷取單一項目，可以使用 `getItem` 方法。

請遵循下列步驟：

1. 建立 DynamoDB 類別的執行個體。
2. 建立 TableKeysAndAttributes 類別的執行個體，其描述要從資料表擷取的主要索引鍵值清單。若希望用單一批次擷取操作讀取多份資料表，您必須每份資料表都建立一個 TableKeysAndAttributes 執行個體。
3. 提供您在前述步驟中建立的 batchGetItem 物件，藉以呼叫 TableKeysAndAttributes 方法。

下列 Java 程式碼範例示範上述步驟。本範例會從 Forum 表擷取兩個項目，從 Thread 表擷取三個項目。

```
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);
    }
}
```

指定選用參數

除了必要的參數之外，您也可在使用 `batchGetItem` 時指定選用的參數。例如，您可為每個定義的 `ProjectionExpression` 提供 `TableKeysAndAttributes`。您如此即可指定要從資料表擷取的屬性。

下列程式碼範例會從 `Forum` 表擷取兩個項目。`withProjectionExpression` 參數會指定只擷取 `Threads` 屬性。

Example

```
TableKeysAndAttributes forumTableKeysAndAttributes = new
    TableKeysAndAttributes("Forum")
        .withProjectionExpression("Threads");

forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(forumTableKeysAndAttributes);
```

更新項目

`updateItem` 物件的 `Table` 方法可更新現有的屬性值、新增新的屬性，或是從現有項目中刪除屬性。

`updateItem` 方法的行為如下：

- 若某項目不存在 (資料表中沒有具備指定主索引鍵的項目)，則 `updateItem` 會在資料表中新增一個新項目。
- 若項目存在，`updateItem` 會依 `UpdateExpression` 參數的指定，執行更新。

Note

您也可以使用 `putItem` 來「更新」項目。例如，若呼叫 `putItem` 將項目新增至資料表，但具備指定主索引鍵的項目已存在，則 `putItem` 會取代整個項目。如果輸入內並未指定現有項目中的屬性，`putItem` 就會從項目中移除這些屬性。

一般而言，建議您在想要修改任何項目屬性時，使用 `updateItem`。`updateItem` 方法只會修改您在輸入內指定的項目屬性，而項目內的其他屬性則保持不變。

請遵循下列步驟：

1. 建立 Table 類別的執行個體，代表您要使用的資料表。
2. 呼叫 updateTable 執行個體的 Table 方法。您必須指定要擷取之項目的主要索引鍵，以及描述要修改之屬性及修改方式的 UpdateExpression。

下列 Java 程式碼範例示範上述工作。程式碼會更新 ProductCatalog 表中的書籍項目。其會將新的作者新增到 Authors 集合中，並刪除現有的 ISBN 屬性。它也會將價格減一。

ExpressionAttributeValue 映射內容會用於 UpdateExpression 中。預留位置 :val1 與 :val2，會於執行時間由 Authors 和 Price 的實際值取代。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#A", "Authors");
expressionAttributeNames.put("#P", "Price");
expressionAttributeNames.put("#I", "ISBN");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1",
    new HashSet<String>(Arrays.asList("Author YY", "Author ZZ")));
expressionAttributeValues.put(":val2", 1); //Price

UpdateItemOutcome outcome = table.updateItem(
    "Id", // key attribute name
    101, // key attribute value
    "add #A :val1 set #P = #P - :val2 remove #I", // UpdateExpression
    expressionAttributeNames,
    expressionAttributeValues);
```

指定選用參數

除了必要的參數之外，您也可以為 updateItem 方法指定選用參數，包括必須滿足才會發生更新的條件。若無法滿足指定的條件，AWS SDK for Java 會擲回 ConditionalCheckFailedException。例如，下列 Java 程式碼範例會依據條件，將書籍項目的價格更新為 25。其會指定 ConditionExpression，指出只有在現有價格為 20 時才更新價格。

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);
```

原子計數器

您可以使用 `updateItem` 來實作原子計數器以增加或減少現有屬性的值，卻不干擾其他寫入請求。若要增加原子計數器，請於使用 `UpdateExpression` 時搭配 `set` 動作，將數值新增到類型為 `Number` 的現有屬性。

下列範例示範這項作業，將 `Quantity` 屬性加 1。其也同時示範在 `ExpressionAttributeNames` 中使用 `UpdateExpression` 參數。

```
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);
```

刪除項目

`deleteItem` 方法會從資料表刪除項目。您必須提供要刪除項目的主索引鍵。

請遵循下列步驟：

1. 建立 DynamoDB 用戶端執行個體。
2. 提供希望刪除之項目的索引鍵，可呼叫 `deleteItem` 方法。

下列 Java 範例示範這些任務。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

DeleteItemOutcome outcome = table.deleteItem("Id", 101);
```

指定選用參數

您可以為 `deleteItem` 指定選用參數。例如，下列 Java 程式碼範例指定 `ConditionExpression`，指出只有當書籍不再出版時 (`InPublication` 屬性為 `false`)，才能刪除 `ProductCatalog` 中的書籍項目。

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);
```

範例：使用 AWS SDK for Java Document API 進行 CRUD 操作

下列程式碼範例示範對 Amazon DynamoDB 項目執行 CRUD 操作。此範例會建立項目、擷取該項目、執行數次更新，最後刪除該項目。

Note

適用於 Java 的開發套件也提供物件持久性模型，讓您將用戶端類別映射至 DynamoDB 資料表。此方法可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱 [Java 1.x : DynamoDBMapper](#)。

Note

此程式碼範例假設您已根據 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 一節的說明將資料載入 DynamoDB 的帳戶。

如需執行下列範例的 step-by-step 指示，請參閱 [Java 程式碼範例](#)。

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DeleteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.UpdateItemOutcome;
import com.amazonaws.services.dynamodbv2.document.spec.DeleteItemSpec;
import com.amazonaws.services.dynamodbv2.document.spec.UpdateItemSpec;
import com.amazonaws.services.dynamodbv2.document.utils.NameMap;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.ReturnValue;

public class DocumentAPIItemCRUDExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);
```

```
static String tableName = "ProductCatalog";

public static void main(String[] args) throws IOException {

    createItems();

    retrieveItem();

    // Perform various updates.
    updateMultipleAttributes();
    updateAddNewAttribute();
    updateExistingAttributeConditionally();

    // Delete the item.
    deleteItem();

}

private static void createItems() {

    Table table = dynamoDB.getTable(tableName);
    try {

        Item item = new Item().withPrimaryKey("Id", 120).withString("Title", "Book
120 Title")
            .withString("ISBN", "120-1111111111")
            .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author12", "Author22")))
            .withNumber("Price", 20).withString("Dimensions",
"8.5x11.0x.75").withNumber("PageCount", 500)
            .withBoolean("InPublication", false).withString("ProductCategory",
"Book");
        table.putItem(item);

        item = new Item().withPrimaryKey("Id", 121).withString("Title", "Book 121
Title")
            .withString("ISBN", "121-1111111111")
            .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author21", "Author 22")))
            .withNumber("Price", 20).withString("Dimensions",
"8.5x11.0x.75").withNumber("PageCount", 500)
            .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
        table.putItem(item);

    } catch (Exception e) {
        // Handle exception
    }
}
```

```
    } 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());
    }
}
```



```
    }  
  }  
}
```

範例：使用 AWS SDK for Java Document API 進行批次操作

本節提供在 Amazon DynamoDB 中使用 AWS SDK for Java 文件 API，執行批次寫入及批次擷取操作的範例。

Note

適用於 Java 的開發套件也提供物件持久性模型，讓您將用戶端類別映射至 DynamoDB 資料表。此方法可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱 [Java 1.x : DynamoDBMapper](#)。

主題

- [範例：使用 AWS SDK for Java Document API 進行批次寫入操作](#)
- [範例：使用 AWS SDK for Java Document API 進行批次擷取操作](#)

範例：使用 AWS SDK for Java Document API 進行批次寫入操作

下列 Java 程式碼範例使用 `batchWriteItem` 方法，執行下列寫入及刪除操作：

- 在 Forum 表中放入一個項目。
- 在 Thread 表中放入一個項目並刪除一個項目。

建立您的批次寫入請求時，您可以對一或多個資料表指定任何次數的放入和刪除請求。但 `batchWriteItem` 對於批次寫入要求的大小，以及單一批次寫入操作中的寫入及刪除操作次數有所限制。如果您的請求超出這些限制，您的請求會被拒絕。如果您的資料表因佈建輸送量不足而無法處理此請求，回應就會傳回未經處理的請求項目。

以下範例會檢查回應，查看它是否有任何未經處理的請求項目。若的確有所限制，其會重頭迴圈並會重新傳送 `batchWriteItem` 請求，同時附上請求中未經處理的項目。如已完成 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 一節，應已建立 Forum 和 Thread 表。您也可利用程式設計方式，建立這些資料表並上傳範例資料。如需詳細資訊，請參閱 [建立範例資料表並使用 AWS SDK for Java](#)。

如需測試下列範例的 step-by-step 指示，請參閱 [Java 程式碼範例](#)。

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);
    }

}

}
```

範例：使用 AWS SDK for Java Document API 進行批次擷取操作

下列 Java 程式碼範例使用 `batchGetItem` 方法，從 `Forum` 和 `Thread` 表擷取多個項目。 `BatchGetItemRequest` 指定每個要擷取之項目的資料表名稱與索引鍵清單。此範例處理回應的方式是列印已擷取的項目。

Note

此程式碼範例假設您已根據 [在 DynamoDB 中建立資料表](#)，以及 [載入程式碼範例的資料](#) 一節的說明將資料載入 DynamoDB 的帳戶。

如需執行下列範例的 step-by-step 指示，請參閱 [Java 程式碼範例](#)。

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());  
}  
}  
}
```

範例：使用 AWS SDK for Java Document API 處理二進位類型屬性

下列 Java 程式碼範例示範二進位類型屬性的處理。範例會將項目新增至 Reply 表。該項目包含存放壓縮資料的二進位類型屬性 (ExtendedMessage)。範例接著會擷取項目，並列印所有屬性值。為了進行示範，該範例使用 GZIPOutputStream 類別來壓縮範例串流，並將其指派給 ExtendedMessage 屬性。擷取二進位屬性時，其便會使用 GZIPInputStream 類別進行解壓縮。

Note

適用於 Java 的開發套件也提供物件持久性模型，讓您將用戶端類別映射至 DynamoDB 資料表。此方法可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱 [Java 1.x : DynamoDBMapper](#)。

如已完成在 [DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 一節，應已建立 Reply 表。您也可以透過編寫程式的方式建立此資料表。如需詳細資訊，請參閱 [建立範例資料表並使用 AWS SDK for Java](#)。

如需測試下列範例的 step-by-step 指示，請參閱 [Java 程式碼範例](#)。

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;  
  }  
}
```

處理項目：.NET

您可以使用 AWS SDK for .NET 低階 API 對資料表項目執行一般的建立、讀取、更新與刪除 (CRUD) 操作。以下是使用 .NET 低階 API 執行資料 CRUD 操作所遵循的一般步驟：

1. 建立 `AmazonDynamoDBClient` 類別的執行個體 (用戶端)。
2. 在對應的請求物件中，提供操作專屬的必要參數。

例如，上傳項目時使用 `PutItemRequest` 請求物件，擷取現有的項目時使用 `GetItemRequest` 請求物件。

您可以使用請求物件來提供必要和選用的參數。

3. 透過傳遞您在前一步驟中所建立的請求物件，執行用戶端提供的適當方法。

`AmazonDynamoDBClient` 用戶端提供 CRUD 操作的 `PutItem`、`GetItem`、`UpdateItem` 和 `DeleteItem` 方法。

主題

- [放入項目](#)
- [取得項目](#)
- [更新項目](#)
- [原子計數器](#)
- [刪除項目](#)
- [批次寫入：放入和刪除多個項目](#)
- [批次取得：取得多個項目](#)

- [範例：使用 AWS SDK for .NET 低階 API 的 CRUD 操作](#)
- [範例：使用 AWS SDK for .NET 低階 API 的批次操作](#)
- [範例：使用 AWS SDK for .NET 低階 API 處理二進位類型屬性](#)

放入項目

PutItem 方法會將項目上傳至資料表。若已有該項目，其會取代整個項目。

Note

若不希望取代整個項目，而是只想要更新特定的屬性，可以使用 UpdateItem 方法。如需詳細資訊，請參閱 [更新項目](#)。

以下是使用低階 .NET 開發套件 API 上傳項目的步驟：

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 您可以透過建立 PutItemRequest 類別的執行個體來提供必要的參數。

若要放入項目，您必須提供資料表名稱及項目。

3. 提供您在第一個步驟中建立的 PutItem 物件來執行 PutItemRequest 方法。

下列 C# 範例示範前述步驟。範例會將項目上傳至 ProductCatalog 資料表。

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);
```

在上述範例中，您要上傳具有 Id、Title、ISBN 和 Authors 屬性的書籍項目。請注意，Id 是數值類型的屬性，而所有其他屬性則是字串類型。作者是 String 集合。

指定選用參數

您也可以使用 PutItemRequest 物件提供選用參數，如以下 C# 範例所示。此範例會指定下列選用參數：

- ExpressionAttributeNames、ExpressionAttributeValues 和 ConditionExpression 指定，只有當現有項目的 ISBN 屬性為特定值時，才能取代該項目。
- ReturnValues 參數請求回應中為舊項目。

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 = "444-444444444444" } },
            { "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 = "444-444444444444"}}
        },
};
```

```
    ConditionExpression = "#I = :isbn"

};
var response = client.PutItem(request);
```

如需詳細資訊，請參閱[PutItem](#)。

取得項目

GetItem 方法會擷取項目。

Note

若要擷取多個項目，您可以使用 BatchGetItem 方法。如需詳細資訊，請參閱 [批次取得：取得多個項目](#)。

以下是使用低階AWS SDK for .NET API 來擷取現有項目的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 您可以透過建立 GetItemRequest 類別的執行個體來提供必要的參數。

若要取得項目，您必須提供資料表名稱及項目的主索引鍵。
3. 提供您在前一個步驟中建立的 GetItem 物件來執行 GetItemRequest 方法。

下列 C# 範例示範前述步驟。下列範例會從 ProductCatalog 表擷取項目。

```
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.
```

指定選用參數

您也可以使用 `GetItemRequest` 物件提供選用參數，如以下 C# 範例所示。範例會指定下列選用參數：

- `ProjectionExpression` 參數會指定要擷取的屬性。
- `ConsistentRead` 參數，用來執行強烈一致讀取。若要進一步了解讀取一致性，請參閱 [讀取一致性](#)。

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;
```

如需詳細資訊，請參閱 [GetItem](#)。

更新項目

`UpdateItem` 方法會更新現有的項目 (若存在的話)。您可以使用 `UpdateItem` 操作來更新現有的屬性值、新增新的屬性，或從現有的集合刪除屬性。若找不到具有指定主索引鍵的項目，則其會新增新的項目。

`UpdateItem` 操作使用以下準則：

- 若還沒有該項目，UpdateItem 會使用輸入中指定的主索引鍵新增項目。
- 若已有該項目，UpdateItem 會套用更新，如下所示：
 - 使用更新中的值取代現有的屬性值。
 - 若您在輸入中提供的屬性不存在，則其會為項目新增新的屬性。
 - 若輸入屬性為 Null，則其會刪除該屬性 (若有的話)。
 - 如果您針對 Action 使用 ADD，就可以將值新增至現有的集合 (字串集或數字集)，或以數學方法加 (使用正數) 減 (使用負數) 現有的數值屬性值。

Note

PutItem 操作也可以執行更新。如需詳細資訊，請參閱 [放入項目](#)。例如，如果您呼叫 PutItem 上傳項目，且主索引鍵已存在，則 PutItem 操作會取代整個項目。如果現有的項目中有屬性，但輸入中並未指定這些屬性，則 PutItem 操作會刪除這些屬性。但是，UpdateItem 只會更新指定的輸入屬性。任何其他該項目現有的屬性都不會變更。

以下是使用低階 .NET 開發套件 API 來更新現有項目的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 您可以透過建立 UpdateItemRequest 類別的執行個體來提供必要的參數。

這是您說明所有更新的請求物件，例如新增屬性、更新現有的屬性或刪除屬性。若要刪除現有的屬性，請以 Null 值指定屬性名稱。

3. 提供您在前一個步驟中建立的 UpdateItem 物件來執行 UpdateItemRequest 方法。

下列 C# 程式碼範例示範前述步驟。範例會更新 ProductCatalog 表中的項目。它會將新的作者新增到 Authors 集合，並刪除現有的 ISBN 屬性。它也會將價格減一。

```
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);
```

指定選用參數

您也可以使用 `UpdateItemRequest` 物件提供選用參數，如以下 C# 範例所示。它會指定以下選用參數：

- `ExpressionAttributeValues` 和 `ConditionExpression` 指定只有當現有價格為 20.00 時才更新價格。
- `ReturnValues` 參數，其會請求回應中的更新項目。

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);
```

如需詳細資訊，請參閱[UpdateItem](#)。

原子計數器

您可以使用 `updateItem` 來實作原子計數器以增加或減少現有屬性的值，卻不干擾其他寫入請求。若要更新原子計數器，`UpdateExpression` 參數請使用屬性類型為 `Number` 的 `updateItem`，`Action` 使用 `ADD`。

下列範例示範這項作業，將 `Quantity` 屬性加 1。

```
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);
```

刪除項目

DeleteItem 方法會從資料表刪除項目。

以下是使用低階 .NET 開發套件 API 來刪除項目的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 您可以透過建立 DeleteItemRequest 類別的執行個體來提供必要的參數。

若要刪除項目，需要有資料表名稱及項目的主索引鍵。

3. 提供您在第一個步驟中建立的 DeleteItem 物件來執行 DeleteItemRequest 方法。

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);
```

指定選用參數

您也可以使用 DeleteItemRequest 物件提供選用參數，如以下 C# 程式碼範例所示。它會指定以下選用參數：

- ExpressionAttributeValues 並指 ConditionExpression 定只有在書籍項目不再出版時才能刪除 (InPublication 屬性值為 false)。
- ReturnValues 參數，其會請求回應中的刪除項目。

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);
```

如需詳細資訊，請參閱[DeleteItem](#)。

批次寫入：放入和刪除多個項目

批次寫入表示在一個批次中放入和刪除多個項目。您可利用 `BatchWriteItem` 方法，在單一呼叫中對來自一或多個資料表的多個項目執行放入與刪除操作。以下是使用低階 .NET 開發套件 API 來擷取多個項目的步驟。

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。
2. 透過建立 `BatchWriteItemRequest` 類別的執行個體來說明所有的放入與刪除操作。
3. 提供您在前一個步驟中建立的 `BatchWriteItem` 物件來執行 `BatchWriteItemRequest` 方法。
4. 處理回應。您應該檢查回應中是否傳回任何未經處理的請求項目。如果您達到佈建輸送量配額或遇到一些其他暫時性錯誤，就可能發生這個狀況。此外，DynamoDB 會限制您在請求中指定的請求大小及操作次數。如果您超出這些限制，DynamoDB 會拒絕此請求。如需詳細資訊，請參閱[BatchWriteItem](#)。

下列 C# 程式碼範例示範前述步驟。範例會建立 `BatchWriteItemRequest` 來執行下列寫入操作：

- 在 `Forum` 表中放入一個項目。
- 在 `Thread` 表中放入及刪除一個項目。

此程式碼會執行 `BatchWriteItem` 來執行批次操作。

```
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);
```

如需運作範例，請參閱 [範例：使用 AWS SDK for .NET 低階 API 的批次操作](#)。

批次取得：取得多個項目

您可利用 `BatchGetItem` 方法，從一或多個資料表擷取多個項目。

Note

若要擷取單一項目，可以使用 `GetItem` 方法。

以下是使用低階 AWS SDK for .NET API 來擷取多個項目的步驟。

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。
2. 您可以透過建立 `BatchGetItemRequest` 類別的執行個體來提供必要的參數。

若要擷取多個項目，需要有資料表名稱及主索引鍵值的清單。

3. 提供您在上一個步驟中建立的 `BatchGetItem` 物件來執行 `BatchGetItemRequest` 方法。

4. 處理回應。您應該檢查是否有任何未經處理的索引鍵，如果您達到佈建輸送量配額或遇到一些其他暫時性錯誤，就可能發生這個狀況。

下列 C# 程式碼範例示範前述步驟。範例會從 Forum 和 Thread 這兩份資料表擷取項目。請求會指定兩個 Forum 表的項目及三個 Thread 表的項目。回應會包含這兩份資料表的項目。程式碼會顯示您處理回應的方法。

```
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);
}
```

指定選用參數

您也可以使用 `BatchGetItemRequest` 物件提供選用參數，如以下 C# 程式碼範例所示。範例會從 `Forum` 表擷取兩個項目。它指定以下選用參數：

- `ProjectionExpression` 參數會指定要擷取的屬性。

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);
```

如需詳細資訊，請參閱[BatchGetItem](#)。

範例：使用 AWS SDK for .NET 低階 API 的 CRUD 操作

下列 C# 程式碼範例會示範對 Amazon DynamoDB 項目執行 CRUD 操作。此範例會在 ProductCatalog 表中新增項目、擷取它、執行各種更新，最後刪除該項目。如完成在 [DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 中的步驟，即已建立 ProductCatalog 表。您也可以利用程式設計方式來建立這些範例資料表。如需詳細資訊，請參閱 [建立範例資料表並使用 AWS SDK for .NET](#)。

如需測試下列範例的 step-by-step 指示，請參閱 [.NET 程式碼範例](#)。

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelItemCRUDExample
    {
        private static string tableName = "ProductCatalog";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                CreateItem();
                RetrieveItem();

                // Perform various updates.
                UpdateMultipleAttributes();
                UpdateExistingAttributeConditionally();

                // Delete item.
                DeleteItem();
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (Exception e)
            {
            }
        }
    }
}
```

```
        {
            Console.WriteLine(e.Message);
            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }
    }

private static void CreateItem()
{
    var request = new PutItemRequest
    {
        TableName = tableName,
        Item = new Dictionary<string, AttributeValue>()
        {
            { "Id", new AttributeValue {
                N = "1000"
            } },
            { "Title", new AttributeValue {
                S = "Book 201 Title"
            } },
            { "ISBN", new AttributeValue {
                S = "11-11-11-11"
            } },
            { "Authors", new AttributeValue {
                SS = new List<string>{"Author1", "Author2" }
            } },
            { "Price", new AttributeValue {
                N = "20.00"
            } },
            { "Dimensions", new AttributeValue {
                S = "8.5x11.0x.75"
            } },
            { "InPublication", new AttributeValue {
                BOOL = false
            } }
        }
    };
    client.PutItem(request);
}

private static void RetrieveItem()
{
    var request = new GetItemRequest
    {
```

```
        TableName = tableName,
        Key = new Dictionary<string, AttributeValue>()
    {
        { "Id", new AttributeValue {
            N = "1000"
        } }
    },
    ProjectionExpression = "Id, ISBN, Title, Authors",
    ConsistentRead = true
};
var response = client.GetItem(request);

// Check the response.
var attributeList = response.Item; // attribute list in the response.
Console.WriteLine("\nPrinting item after retrieving it .....");
PrintItem(attributeList);
}

private static void UpdateMultipleAttributes()
{
    var request = new UpdateItemRequest
    {
        Key = new Dictionary<string, AttributeValue>()
        {
            { "Id", new AttributeValue {
                N = "1000"
            } }
        },
        // Perform the following updates:
        // 1) Add two new authors to the list
        // 1) Set a new attribute
        // 2) Remove the ISBN attribute
        ExpressionAttributeNames = new Dictionary<string, string>()
        {
            {"#A", "Authors"},
            {"#NA", "NewAttribute"},
            {"#I", "ISBN"}
        },
        ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
        {
            {":auth", new AttributeValue {
                SS = {"Author YY", "Author ZZ"}
            }},
            {":new", new AttributeValue {
```

```
        S = "New Value"
    }}
},

UpdateExpression = "ADD #A :auth SET #NA = :new REMOVE #I",

TableName = tableName,
ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
};
var response = client.UpdateItem(request);

// Check the response.
var attributeList = response.Attributes; // attribute list in the response.
                                         // print attributeList.
Console.WriteLine("\nPrinting item after multiple attribute
update .....");
PrintItem(attributeList);
}

private static void UpdateExistingAttributeConditionally()
{
    var request = new UpdateItemRequest
    {
        Key = new Dictionary<string, AttributeValue>()
        {
            { "Id", new AttributeValue {
                N = "1000"
            } }
        },
        ExpressionAttributeNames = new Dictionary<string, string>()
        {
            {"#P", "Price"}
        },
        ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
        {
            {":newprice", new AttributeValue {
                N = "22.00"
            }},
            {":currprice", new AttributeValue {
                N = "20.00"
            } }
        },
        // This updates price only if current price is 20.00.
        UpdateExpression = "SET #P = :newprice",
```

```
        ConditionExpression = "#P = :currprice",

        TableName = tableName,
        ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
    };
    var response = client.UpdateItem(request);

    // Check the response.
    var attributeList = response.Attributes; // attribute list in the response.
    Console.WriteLine("\nPrinting item after updating price value
conditionally .....");
    PrintItem(attributeList);
}

private static void DeleteItem()
{
    var request = new DeleteItemRequest
    {
        TableName = tableName,
        Key = new Dictionary<string, AttributeValue>()
        {
            { "Id", new AttributeValue {
                N = "1000"
            } }
        },

        // Return the entire item as it appeared before the update.
        ReturnValues = "ALL_OLD",
        ExpressionAttributeNames = new Dictionary<string, string>()
        {
            {"#IP", "InPublication"}
        },
        ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
        {
            {":inpub", new AttributeValue {
                BOOL = false
            }}
        },
        ConditionExpression = "#IP = :inpub"
    };

    var response = client.DeleteItem(request);

    // Check the response.
```

```
        var attributeList = response.Attributes; // Attribute list in the response.
                                                // Print item.
        Console.WriteLine("\nPrinting item that was just deleted .....");
        PrintItem(attributeList);
    }

    private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
    {
        foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
        {
            string attributeName = kvp.Key;
            AttributeValue value = kvp.Value;

            Console.WriteLine(
                attributeName + " " +
                (value.S == null ? "" : "S=[" + value.S + "]") +
                (value.N == null ? "" : "N=[" + value.N + "]") +
                (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +
                (value.NS == null ? "" : "NS=[" + string.Join(",",
value.NS.ToArray()) + "]")
                );
        }
        Console.WriteLine("*****");
    }
}
```

範例：使用 AWS SDK for .NET 低階 API 的批次操作

主題

- [範例：使用 AWS SDK for .NET 低階 API 的批次寫入操作](#)
- [範例：使用 AWS SDK for .NET 低階 API 的批次取得操作](#)

本節提供 Amazon DynamoDB 支援的批次操作、批次寫入和批次取得的範例。

範例：使用 AWS SDK for .NET 低階 API 的批次寫入操作

下列 C# 程式碼範例使用 BatchWriteItem 方法來執行下列放入及刪除操作：

- 在 Forum 表中放入一個項目。
- 在 Thread 表中放入一個項目並刪除一個項目。

建立您的批次寫入請求時，您可以對一或多個資料表指定任何次數的放入和刪除請求。但 DynamoDB BatchWriteItem 對於批次寫入要求的大小，以及單一批次寫入操作中的寫入及刪除操作次數有所限制。如需詳細資訊，請參閱[BatchWriteItem](#)。如果您的請求超出這些限制，您的請求會被拒絕。如果您的資料表因佈建輸送量不足而無法處理此請求，回應就會傳回未經處理的請求項目。

以下範例會檢查回應，查看它是否有任何未經處理的請求項目。若的確有所限制，其會重頭迴圈並會重新傳送 BatchWriteItem 請求，同時附上請求中未經處理的項目。如完成在 [DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 中的步驟，即已建立 Forum 和 Thread 表。您也可利用程式設計方式來建立這些範例資料表並上傳範例資料。如需詳細資訊，請參閱 [建立範例資料表並使用 AWS SDK for .NET](#)。

如需測試下列範例的 step-by-step 指示，請參閱[.NET 程式碼範例](#)。

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);
}
}
}
```

範例：使用 AWS SDK for .NET 低階 API 的批次取得操作

下列 C# 程式碼範例使用 `BatchGetItem` 方法，從 Amazon DynamoDB 的 `Forum` 和 `Thread` 表中擷取多個項目。`BatchGetItemRequest` 指定每份資料表的資料表名稱及主索引鍵清單。此範例處理回應的方式是列印已擷取的項目。

如已完成 [在 DynamoDB 中建立資料表](#)，以及載入程式碼範例的資料中的步驟，則您即已使用範例資料來建立這些資料表。您也可利用程式設計方式來建立這些範例資料表並上傳範例資料。如需詳細資訊，請參閱 [建立範例資料表並使用 AWS SDK for .NET](#)。

如需測試下列範例的 step-by-step 指示，請參閱 [.NET 程式碼範例](#)。

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("*****");
}
}
}

```

範例：使用 AWS SDK for .NET 低階 API 處理二進位類型屬性

下列 C# 程式碼範例示範二進位類型屬性的處理方式。範例會將項目新增至 Reply 表。該項目包含存放壓縮資料的二進位類型屬性 (ExtendedMessage)。範例接著會擷取項目，並列印所有屬性值。為了進行說明，該範例使用 GZipStream 類別來壓縮範例串流，將其指派給 ExtendedMessage 屬性，並在列印屬性值時解壓縮它。

如完成在 [DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 中的步驟，即已建立 Reply 表。您也可以利用程式設計方式來建立這些範例資料表。如需詳細資訊，請參閱 [建立範例資料表並使用 AWS SDK for .NET](#)。

如需測試下列範例的 step-by-step 指示，請參閱 [.NET 程式碼範例](#)。

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();
        }
    }
}
```

項目集合-如何在 DynamoDB 中 one-to-many 建立關係的模型

在 DynamoDB 中，物品集合是共享相同分割區索引鍵值的物品群組，這表示這些物品是相關聯的。項目集合是在 DynamoDB 中 one-to-many 建立關係模型的主要機制。物品集合只能存在於配置為使用[複合主鍵](#)的資料表或索引。

Note

物品集合可以存在於基底資料表中，也可以存在於次要索引中。有關物品集合如何與索引互動的詳細資訊，請參閱[本機次要索引中的項目集合](#)。

請考慮下表，其中顯示三位不同的使用者及其遊戲內物品清單：

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	d1e8a70b5ccab1dc2f56bbf7e99f064a660c08e361a35751b9c483c88943d082	state	last-login
			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	k2g8jk0m5ppab1dc2f56bbf7e99f064a660c08e361a35751b9c464r23943i082	state	last-login
			Banned	1649456737
account1138	info	data		
		{ "email": "luh-3417@gmail.com" }		
login-data	pw	88A41A9A62B11CC8C120861928765A3EA41DEB9EAFE261D90F619473B89A2D4	state	last-login
			Active	14275516966

對於每個集合中的特定物品，排序索引鍵是由用於將資料分組的資訊 (例如 `inventory::armor`、`inventory::weapon` 或 `info`) 組成的串連。每個物品集合可以具有不同組合的屬性作為排序索引鍵。使用者 `account1234` 具有 `inventory::weapons` 物品，而使用者 `account1387` 沒有 (因為他們還沒有找到)。使用 `account1138` 只使用兩個物品作為排序索引鍵 (因為他們還沒有物品清單)，而其他使用者使用三個物品。

DynamoDB 允許您選擇性地從這些物品集中擷取物品，以執行以下操作：

- 從特定使用者擷取所有物品
- 從特定使用者擷取一個物品
- 擷取屬於特定使用者的特定類型的所有物品

使用物品集合組織資料來加快查詢

在此範例中，這三個物品集合中的每個物品都代表一個玩家，和我們根據遊戲和玩家的存取模式選擇的資料模型。遊戲需要什麼資料？什麼時候需要？需要的頻率為何？這樣做的成本是多少？這些資料建模決策是根據這些問題的答案做出。

在這個遊戲中，有一個頁面為玩家呈現其武器庫存清單，還有另一個頁面呈現盔甲。玩家開啟他們的庫存清單時，會先顯示武器，因為我們希望該頁面能夠以極快速度載入，之後才會載入後續的庫存清單頁

面。隨著玩家獲得更多的遊戲內物品，這些物品類型可能會相當大，因此我們決定每個庫存清單頁面在資料庫的玩家物品集合中，都獨立自成一個項目。

以下區段詳細介紹如何透過 Query 操作與物品集合互動。

主題

- [DynamoDB 中的查詢操作](#)

DynamoDB 中的查詢操作

您可以使用 Amazon DynamoDB 中的 Query API 操作，以根據主索引鍵值尋找項目。

您必須提供分割區索引鍵屬性的名稱，以及該屬性的單一值。Query 會傳回所有具有該分割區索引鍵值的項目。您可以選擇是否提供排序索引鍵屬性，使用比較運算子縮小搜尋結果。

如需如何使用 Query (例如請求語法、回應參數和其他範例)，請參閱[查詢](#) (在 Amazon DynamoDB API 參考中)。

主題

- [查詢操作的索引鍵條件表達式](#)
- [查詢操作的篩選條件表達式](#)
- [分頁表查詢結果](#)
- [使用查詢操作的其他方面](#)
- [查詢資料表及索引：Java](#)
- [查詢資料表與索引：.NET](#)

查詢操作的索引鍵條件表達式

若要指定搜尋條件，請使用索引鍵條件表達式 (判斷要從資料表或索引讀取之項目的字串)。

您必須將分割區索引鍵名稱及數值指定為相等條件。您無法在索引鍵條件表達式中使用非索引鍵屬性。

您可以選擇性為排序索引鍵提供第二個條件 (若有的話)。排序索引鍵條件必須使用下列其中一個比較運算子：

- $a = b$ ：如果屬性 a 等於數值 b ，則為 true
- $a < b$ ：如果 a 小於 b ，則為 true

- $a \leq b$: 如果 a 小於或等於 b , 則為 true
- $a > b$: 如果 a 大於 b , 則為 true
- $a \geq b$: 如果 a 大於或等於 b , 則為 true
- a BETWEEN b AND c : 如果 a 大於或等於 b 且小於或等於 c , 則為 true。

同樣支援下列函數 :

- `begins_with (a, substr)` : 如果屬性 a 的值開頭為特定子字串 , 則為 true。

下面的 AWS Command Line Interface (AWS CLI) 實例演示了鍵條件表達式的用法。這些表達式會使用預留位置 (例如 `:name` 和 `:sub`) , 而非實際的值。如需更多詳細資訊 , 請參閱 [DynamoDB 中的表達式屬性名稱](#) 及 [表達式屬性值](#)。

Example

查詢 Thread 表是否有特定的 ForumName (分割區索引鍵)。查詢會讀取所有具有該 ForumName 值的項目 , 因為 KeyConditionExpression 中不包含排序索引鍵 (Subject)。

```
aws dynamodb query \  
  --table-name Thread \  
  --key-condition-expression "ForumName = :name" \  
  --expression-attribute-values '{":name":{"S":"Amazon DynamoDB"}}'
```

Example

查詢 Thread 表是否有特定的 ForumName (分割區索引鍵) , 但這次僅傳回具有指定 Subject (排序索引鍵) 的項目。

```
aws dynamodb query \  
  --table-name Thread \  
  --key-condition-expression "ForumName = :name and Subject = :sub" \  
  --expression-attribute-values file://values.json
```

`--expression-attribute-values` 的引數會存放在 `values.json` 檔案中。

```
{  
  ":name":{"S":"Amazon DynamoDB"},
```

```
":sub":{"S":"DynamoDB Thread 1"}
}
```

Example

查詢 Reply 表是否有特定的 Id (分割區索引鍵)，但只傳回以特定字元開頭的 ReplyDateTime (排序索引鍵) 項目。

```
aws dynamodb query \  
  --table-name Reply \  
  --key-condition-expression "Id = :id and begins_with(ReplyDateTime, :dt)" \  
  --expression-attribute-values file://values.json
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{  
  ":id":{"S":"Amazon DynamoDB#DynamoDB Thread 1"},  
  ":dt":{"S":"2015-09"}  
}
```

您可以在索引鍵條件表達式中使用任何屬性名稱，只要第一個字元為 a-z 或 A-Z，且其餘字元 (若有的話，從第二個字元開始) 為 a-z、A-Z，或 0-9。此外，屬性名稱不得為 DynamoDB 的保留字。(如需這些保留字的完整清單，請參閱「[DynamoDB 中的保留字](#)」)。若屬性名稱不符合這些需求，您必須將表達式屬性名稱定義為預留位置。如需詳細資訊，請參閱 [DynamoDB 中的表達式屬性名稱](#)。

針對具有給定分割區索引鍵值的項目，DynamoDB 會依照排序索引鍵值，將這些項目以排序後的順序存放在緊鄰位置。在 Query 操作中，DynamoDB 會以排序後的順序擷取項目，並使用 KeyConditionExpression 及任何存在的 FilterExpression 處理項目。只有在這個時候，Query 的結果才會傳回用戶端。

Query 操作永遠都會傳回一個結果集。如果找不到相符項目，表示結果集是空的。

Query 結果永遠都會根據排序索引鍵值進行排序。如果排序索引鍵的資料類型是 Number，即依數值順序傳回結果。否則，按 UTF-8 位元組順序傳回結果。根據預設，排序順序為遞增排序。若要反轉順序，請將 ScanIndexForward 參數設為 false。

單一 Query 操作最多可擷取 1 MB 的資料。這項限制會在任何 FilterExpression 或 ProjectionExpression 套用到結果之前套用。若回應中有 LastEvaluatedKey 且為非 Null，您即必須為結果集編製分頁 (請參閱[分頁表查詢結果](#))。

查詢操作的篩選條件表達式

若您需要更精確的 Query 結果，您可以選擇性的提供篩選條件表達式。篩選條件表達式會判斷要傳回 Query 結果中的哪些項目。所有其他結果皆會捨棄。

篩選條件表達式會在 Query 完成之後，並在傳回結果之前套用。因此，無論是否有篩選條件表達式，Query 都會使用相同數量的讀取容量。

Query 操作最多可擷取 1 MB 的資料。系統會先套用這項限制，再評估篩選條件表達式。

篩選條件表達式無法包含分割區索引鍵或排序索引鍵屬性。您必須在索引鍵條件表達式中指定這些屬性，而非篩選條件表達式。

篩選條件表達式的語法和索引鍵條件表達式的語法類似。篩選條件表達式可以使用與索引鍵條件表達式相同的比較子、函數和邏輯運算子。此外，篩選條件表達式可以使用不等於運算子 (<>)、OR 運算子、CONTAINS 運算子、IN 運算子、BEGINS_WITH 運算子、BETWEEN 運算子、EXISTS 運算子和 SIZE 運算子。如需詳細資訊，請參閱 [查詢操作的索引鍵條件表達式](#) 及 [篩選條件和條件表達式的語法](#)。

Example

下列 AWS CLI 範例會查詢特定 ForumName (分割索引鍵) 和 Subject (排序索引鍵) 的 Thread 資料表。在找到的項目中，只會傳回最受歡迎的討論主題：換句話說，只會傳回具有超過一定 Views 數量的主題。

```
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
```

--expression-attribute-values 的引數會存放在 values.json 檔案中。

```
{  
  ":fn":{"S":"Amazon DynamoDB"},  
  ":sub":{"S":"DynamoDB Thread 1"},  
  ":num":{"N":"3"}  
}
```

請注意，Views 為 DynamoDB 中的保留字 (請參閱 [DynamoDB 中的保留字](#))，所以此範例會使用 #v 作為預留位置。如需詳細資訊，請參閱 [DynamoDB 中的表達式屬性名稱](#)。

Note

篩選條件表達式會從 Query 結果集中移除項目。若可能的話，請避免在預期會擷取大量項目，但又需要捨棄它們大部分的情況下，使用 Query。

分頁表查詢結果

DynamoDB 會對 Query 操作的結果進行分頁。透過編製分頁，Query 結果會分成數個大小為 1 MB (或更小) 的資料「頁」。應用程式可以處理結果的第一頁、第二頁，以此類推。

單一 Query 只會傳回符合 1 MB 大小限制的結果集。若要判斷是否有更多結果，且要一次擷取一頁結果，應用程式需執行下列作業：

1. 檢查低層級 Query 結果：
 - 若結果包含 LastEvaluatedKey 元素且為非空值，請接著進行步驟 2。
 - 若結果中「沒有」LastEvaluatedKey，就表示再也沒有要擷取的項目。
2. 建構新的 Query 請求，和前一個請求使用相同的參數。但這一次採用步驟 1 的 LastEvaluatedKey 值，並用它做為新 Query 要求的 ExclusiveStartKey 參數。
3. 執行新的 Query 請求。
4. 前往步驟 1。

換句話說，LastEvaluatedKey 回應的 Query 應做為下一個 ExclusiveStartKey 請求的 Query 使用。若 LastEvaluatedKey 回應中沒有 Query 元素，表示您已擷取到結果的最終頁。如果 LastEvaluatedKey 不是空的，則不一定意味著結果集中有更多資料。檢查 LastEvaluatedKey 是否為空，是確定您是否已到達結果集末頁的唯一方式。

您可以使用 AWS CLI 來檢視此行為。會重複 AWS CLI 傳送低層級 Query 請求至 DynamoDB，直到結果中 LastEvaluatedKey 不再出現為止。請考慮下列 AWS CLI 範例，從特定年份擷取電影標題。

```
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
```


通常情況下，AWS CLI 句柄自動分頁。但是，在此範例中，AWS CLI `--page-size` 參數會限制每頁的項目數。`--debug` 參數會列印請求及回應的下層資訊。

如果您執行此範例，DynamoDB 的第一個回應會類似以下內容。

```
2017-07-07 11:13:15,603 - MainThread - boto.core.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}'
```

回應中的 `LastEvaluatedKey` 會指出並未擷取所有項目。AWS CLI 然後會向 DynamoDB 發出另一個 `Query` 請求。此請求和回應模式會持續到最終回應出現為止。

```
2017-07-07 11:13:16,291 - MainThread - boto.core.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"What\'s Eating Gilbert
Grape"}}],"ScannedCount":1}'
```

若沒有 `LastEvaluatedKey`，就表示已不再有要擷取的項目。

Note

AWS SDK 會處理低階 DynamoDB 回應 (包括是否存在 `LastEvaluatedKey`)，並為分頁結果提供各種抽象。`Query` 例如，適用於 Java 的開發套件文件界面會提供 `java.util.Iterator` 支援，讓您可一次處理一個結果。

如需各種程式設計語言的程式碼範例，請參閱 [《Amazon DynamoDB 入門指南》](#) 和所需語言的 AWS 開發套件文件。

您還可以透過限制結果集的項目數 (使用 `Query` 操作的 `Limit` 參數) 來減少頁面大小。

如需使用 DynamoDB 進行查詢的詳細資訊，請參閱 [DynamoDB 中的查詢操作](#)。

使用查詢操作的其他方面

限制結果集的項目數

您可以使用 `Query` 操作來限制其讀取的項目數。若要執行此作業，請將 `Limit` 參數設為您希望的最大項目數。

例如，假設您 Query 一份資料表，將 Limit 值設為 6 且不使用篩選條件表達式。Query 結果會包含資料表中符合請求索引鍵條件表達式的前六個項目。

現在假設您在 Query 中新增一個篩選條件表達式。在此情況下，DynamoDB 最多可讀取六個項目，然後只傳回符合篩選條件表達式的項目。最後的 Query 結果包含六個或更少的項目，即使有更多項目符合篩選條件表達式 (如果 DynamoDB 持續讀取更多項目)。

計算結果中的項目

除了符合您條件的項目之外，Query 回應還包含了下列元素：

- ScannedCount：套用篩選條件表達式 (若有) 前符合索引鍵條件表達式的項目數。
- Count：套用篩選條件表達式 (若有) 後剩餘的項目數。

Note

若不使用篩選條件表達式，ScannedCount 和 Count 就會有相同的值。

若 Query 結果集的大小大於 1 MB，則 ScannedCount 和 Count 僅代表總項目的部分計數。您需要執行多項 Query 操作，才能擷取所有的結果 (請參閱[分頁表查詢結果](#))。

每個 Query 回應都包含經該特定 Query 請求處理過的項目 ScannedCount 和 Count。若要取得所有 Query 請求的總計，您可以為 ScannedCount 及 Count 記錄流水帳。

查詢使用的容量單位

您可以 Query 任何資料表或次要索引，只要您提供分割索引鍵屬性的名稱以及該屬性的單一值即可。Query 會傳回具有該分割區索引鍵值的所有項目。您可以選擇是否提供排序索引鍵屬性，並使用比較運算子縮小搜尋結果。QueryAPI 操作會使用讀取容量單位，如下所示。

若您對下列進行 Query	DynamoDB 使用的讀取容量單位就會來自...
資料表	該資料表的佈建讀取容量。
全域次要索引	該索引的佈建讀取容量。
本機次要索引	該基礎資料表的佈建讀取容量。

根據預設，Query 操作不會傳回任何使用之讀取容量的相關資料。但您可以在 ReturnConsumedCapacity 請求中指定 Query 參數，來取得這項資訊。下列為 ReturnConsumedCapacity 的有效設定：

- NONE：不會傳回耗用的容量資料。(此為預設值)。
- TOTAL：回應包括耗用的讀取容量單位總數。
- INDEXES：回應顯示耗用的讀取容量單位總數，以及每個資料表和存取之索引的耗用容量。

DynamoDB 會根據項目數量和這些項目的大小 (而不是傳回給應用程式的資料量) 來計算使用的讀取容量單位數量。因此，無論您請求所有屬性 (預設行為) 或只請求部分屬性 (使用投影表達式)，使用的容量單位數都相同。無論您是否使用篩選器運算式，數字也相同。Query 消耗最小讀取容量單位以每秒執行一次強烈一致讀取，或每秒針對 4 KB 以下的項目執行兩次最終一致讀取。如果您需要讀取大於 4KB 的項目，DynamoDB 需要額外的讀取請求單位。空白資料表和具有稀疏數量分割索引鍵的非常大資料表，可能會看到額外的 RCU 計費超出查詢的資料量。這涵蓋了服務 Query 請求的成本，即使沒有數據存在。

查詢的讀取一致性

根據預設，Query 操作會執行最終一致讀取。這表示 Query 的結果可能不會反映最近完成之 PutItem 或 UpdateItem 操作所造成的變更。如需詳細資訊，請參閱 [讀取一致性](#)。

若您需要強烈一致讀取，請在 ConsistentRead 請求中將 true 參數設為 Query。

查詢資料表及索引：Java

Query 操作可讓您查詢 Amazon DynamoDB 中的資料表或次要索引。您必須提供分割區索引鍵值與相等條件。如果資料表或索引具有排序索引鍵，您可以提供排序索引鍵值與條件來縮小結果。

Note

AWS SDK for Java 也提供物件持續性模型，可讓您將用戶端類別對應至 DynamoDB 表格。此方式可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱 [Java 1.x : DynamoDBMapper](#)。

以下是使用 AWS SDK for Java 文件 API 擷取項目的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體，代表您要進行作業的資料表。

3. 呼叫 `query` 執行個體的 `Table` 方法。您必須指定要擷取的項目分割區索引鍵值，以及任何選用的查詢參數。

回應包含的 `ItemCollection` 物件，提供查詢傳回的所有項目。

下列 Java 程式碼範例示範上述工作。此範例假設您的 `Reply` 表存放論壇主題的回覆。如需詳細資訊，請參閱 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#)。

```
Reply ( Id, ReplyDateTime, ... )
```

每個論壇主題具有唯一的 ID，並可擁有零或多個回覆。因此，`Reply` 表的 `Id` 屬性同時包含論壇名稱和論壇主旨。`Id` (分割區索引鍵) 和 `ReplyDateTime` (排序索引鍵) 構成資料表的複合主索引鍵。

下列查詢會擷取特定主題主旨的所有回覆。查詢需要資料表名稱與 `Subject` 值。

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());
}
```

指定選用參數

`query` 方法支援數個選用的參數。例如，您可以指定條件，選擇性縮減上個查詢的結果，傳回過去兩週的回覆。此條件稱為排序索引鍵條件，因為 `DynamoDB` 會根據主索引鍵的排序索引鍵評估您指定的查詢條件。您可以指定其他選用參數，只在查詢結果的項目中擷取特定的屬性清單。

以下 Java 程式碼範例會擷取過去 15 天所張貼的論壇主題回覆。此範例會使用下列項目指定選用參數：

- `KeyConditionExpression`，會擷取特定開發論壇 (分割區索引鍵) 的回覆，且為該項目集合過去 15 天內 (排序索引鍵) 張貼的回覆。
- `FilterExpression`，只傳回特定使用者的回覆。篩選條件會在查詢完成之後、使用者傳回結果之前套用。
- `ValueMap`，定義 `KeyConditionExpression` 預留位置的實際值。
- `ConsistentRead` 設為 `true`，請求強烈一致讀取。

本範例使用的 `QuerySpec` 物件允許存取所有低階 `Query` 輸入參數。

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());
}
```

您也可以使用 `withMaxPageSize` 方法，選擇性地限制每一頁面的項目數目。當您呼叫 `query` 方法時，您會取得包含所產生項目的 `ItemCollection`。然後，您可以瀏覽結果，一次處理一頁，直到完成所有頁面。

下列 Java 程式碼範例會修改前文所示的查詢規格。這次，查詢規格會使用 `withMaxPageSize` 方法。Page 類別提供允許程式碼處理各頁項目的迭代器。

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());
    }
}
```

範例 - 使用 Java 查詢

下列資料表會存放論壇集合的相關資訊。如需詳細資訊，請參閱 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#)。

Note

適用於 Java 的開發套件也提供物件持久性模型，讓您將用戶端類別映射至 DynamoDB 資料表。此方式可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱 [Java 1.x : DynamoDBMapper](#)。

Example

```
Forum ( Name, ... )
Thread ( ForumName, Subject, Message, LastPostedBy, LastPostDateTime, ... )
Reply ( Id, ReplyDateTime, Message, PostedBy, ... )
```

在這個 Java 程式碼範例中，您要在論壇「DynamoDB」中為主題「DynamoDB Thread 1」執行尋找回覆的變數。

- 尋找主題的回覆。
- 尋找某個主題的回覆，指定每頁產生的項目數限制。如果結果集的項目數超過頁面大小，您只會收到第一頁的結果。此編碼模式可確保您的程式碼會處理查詢結果中的所有頁面。
- 尋找過去 15 天的回覆。
- 尋找特定日期範圍內的重覆。

前面兩項查詢示範如何指定排序索引鍵條件以縮小查詢結果，以及使用其他選用查詢參數。

Note

此程式碼範例假設您已根據 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 一節的說明將資料載入 DynamoDB 的帳戶。

如需執行下列範例的 step-by-step 指示，請參閱 [Java 程式碼範例](#)。

```
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());
        }
    }
}
```

查詢資料表與索引：.NET

Query 操作可讓您查詢 Amazon DynamoDB 中的資料表或次要索引。您必須提供分割區索引鍵值與相等條件。如果資料表或索引具有排序索引鍵，您可以提供排序索引鍵值與條件來縮小結果。

以下是使用低級 AWS SDK for .NET API 查詢表的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 建立 QueryRequest 類別的執行個體，並提供查詢操作參數。
3. 執行 Query 方法，並提供您在上一個步驟中建立的 QueryRequest 物件。

回應包含的 QueryResult 物件，提供查詢傳回的所有項目。

下列 C# 程式碼範例示範上述工作。此程式碼假設您的 Reply 表存放論壇主題的回覆。如需詳細資訊，請參閱 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#)。

Example

```
Reply Id, ReplyDateTime, ... )
```

每個論壇主題具有唯一的 ID，並可擁有零或多個回覆。因此，主索引鍵同時包含 Id (分割區索引鍵) 和 ReplyDateTime (排序索引鍵)。

下列查詢會擷取特定主題主旨的所有回覆。查詢需要資料表名稱與 Subject 值。

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);
}
```

指定選用參數

Query 方法支援數個選用的參數。例如，您可以指定條件，選擇性縮減上個查詢的查詢結果，傳回過去兩週的回覆。此條件稱為排序索引鍵條件，因為 DynamoDB 會根據主索引鍵的排序索引鍵評估您指定的查詢條件。您可以指定其他選用參數，只在查詢結果的項目中擷取特定的屬性清單。如需詳細資訊，請參閱[查詢](#)。

下列 C# 程式碼範例會擷取過去 15 天所張貼的論壇主題回覆。此範例會指定下列選用參數：

- KeyConditionExpression，只擷取過去 15 天的回覆。

- `ProjectionExpression` 參數，指定在查詢結果中要擷取的項目屬性清單。
- `ConsistentRead` 參數，執行強烈一致讀取。

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);
}
```

您也可以透過新增選用的 `Limit` 參數，選擇性地限制頁面大小或是每一頁面的項目數目。每次您執行 `Query` 方法時，就會擷取一頁具有指定項目數的結果。若要擷取下一頁面，您可以再次執行 `Query` 方法，並提供先前頁面最後一個項目的主索引鍵值，讓方法傳回下一組項目。您可以透過設定 `ExclusiveStartKey` 屬性，在請求中提供此資訊。一開始，此屬性可為 `Null`。若要擷取後續頁面，您必須將此屬性值更新為先前頁面最後一個項目的主索引鍵。

下列 C# 範例會查詢 `Reply` 表。在請求中，它會指定 `Limit` 和 `ExclusiveStartKey` 選用參數。`do/while` 迴圈會繼續一次掃描一個頁面，直到 `LastEvaluatedKey` 傳回 `Null` 數值為止。

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);
```

範例-使用 AWS SDK for .NET

下列資料表會存放論壇集合的相關資訊。如需詳細資訊，請參閱 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#)。

Example

```
Forum ( Name, ... )
Thread ( ForumName, Subject, Message, LastPostedBy, LastPostDateTime, ... )
Reply ( Id, ReplyDateTime, Message, PostedBy, ... )
```

在本範例中，您要在論壇「DynamoDB」中為主題「DynamoDB Thread 1」執行「Find replies」的變數。

- 尋找主題的回覆。
- 尋找主題的回覆。指定 Limit 查詢參數來設定頁面大小。

此功能說明如何使用分頁來處理多頁結果。DynamoDB 有頁面大小限制，如果結果超出頁面大小，您只會收到第一頁的結果。此編碼模式可確保您的程式碼會處理查詢結果中的所有頁面。

- 尋找過去 15 天的回覆。
- 尋找特定日期範圍內的重覆。

前面兩項查詢示範如何指定排序索引鍵條件以縮小查詢結果，以及使用其他選用查詢參數。

如需測試下列範例的 step-by-step 指示，請參閱[.NET 程式碼範例](#)。

```
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("*****");
}
```

在 DynamoDB 中使用掃描

Amazon DynamoDB 中的 Scan 操作會讀取資料表或次要索引中的每個項目。根據預設，Scan 操作會傳回資料表或索引中每個項目的所有資料屬性。您可以使用 ProjectionExpression 參數，以便 Scan 只傳回部分屬性，而不會全部傳回。

Scan 一律會傳回結果集。如果找不到相符項目，表示結果集是空的。

單一 Scan 請求最多可擷取 1 MB 的資料。或者，DynamoDB 可將篩選條件表達式套用至此資料，縮減結果後再傳回給使用者。

主題

- [掃描的篩選條件表達式](#)
- [限制結果集的項目數](#)
- [為結果編製分頁](#)
- [計算結果中的項目](#)
- [掃描使用的容量單位](#)
- [掃描的讀取一致性](#)
- [平行掃描](#)

- [掃描資料表及索引：Java](#)
- [掃描資料表及索引：.NET](#)

掃描的篩選條件表達式

若您需要更精確的 Scan 結果，您可以選擇性的提供篩選條件表達式。篩選條件表達式會判斷要傳回 Scan 結果中的哪些項目。所有其他結果皆會捨棄。

篩選條件表達式會在 Scan 完成後、結果傳回前套用。因此，無論是否有篩選條件表達式，Scan 都會使用相同數量的讀取容量。

Scan 操作最多可擷取 1 MB 的資料。系統會先套用這項限制，再評估篩選條件表達式。

使用 Scan，您可在篩選條件表達式中指定任何屬性，包括分割區索引鍵和排序索引鍵屬性。

篩選條件表達式的語法和條件表達式的語法相同。篩選條件表達式可以使用與條件表達式相同的比較子、函數和邏輯運算子。如需邏輯運算子的詳細資訊，請參閱 [比較運算子與函數參考](#)。

Example

下列 AWS Command Line Interface (AWS CLI) 範例會掃描資料表，並只傳回特定使用者上次張貼至的項目。

```
aws dynamodb scan \  
  --table-name Thread \  
  --filter-expression "LastPostedBy = :name" \  
  --expression-attribute-values '{":name":{"S":"User A"}}'
```

限制結果集的項目數

Scan 操作可讓您限制在結果中傳回的項目數目。若要執行此作業，請將 Limit 參數設定為您希望 Scan 操作在篩選表達式評估前回傳的最大項目數。

例如，假設您 Scan 一份資料表，將 Limit 值設為 6 且不使用篩選條件表達式。此 Scan 結果會包含資料表中的前六個項目。

現在假設您在 Scan 中新增一個篩選條件表達式。在本案例中，DynamoDB 會將篩選條件表達式套用於傳回的六個項目，捨棄那些不符的項目。最終的 Scan 結果會包含 6 個或以下項目，視篩選的項目數而定。

為結果編製分頁

DynamoDB 會對 Scan 操作的結果進行分頁。透過編製分頁，Scan 結果會分成數個大小為 1 MB (或更小) 的資料「頁」。應用程式可以處理結果的第一頁、第二頁，以此類推。

單一 Scan 只會傳回符合 1 MB 大小限制的結果集。

為判斷是否有更多結果，並且一次擷取一頁結果，應用程式應執行下列作業：

1. 檢查低層級 Scan 結果：
 - 若結果包含 LastEvaluatedKey 元素，請接著進行步驟 2。
 - 若結果中沒有 LastEvaluatedKey，就表示再也沒有要擷取的項目。
2. 建構新的 Scan 請求，和前一個請求使用相同的參數。但這一次採用步驟 1 的 LastEvaluatedKey 值，並用它做為新 Scan 要求的 ExclusiveStartKey 參數。
3. 執行新的 Scan 請求。
4. 前往步驟 1。

換句話說，LastEvaluatedKey 回應的 Scan 應做為下一個 ExclusiveStartKey 請求的 Scan 使用。若 Scan 回應中沒有 LastEvaluatedKey 元素，表示您已擷取到結果的最終頁。(檢查是否沒有 LastEvaluatedKey 是確定您是否已到達結果集末頁的唯一方式)。

您可以使用 AWS CLI 來檢視此行為。會重複 AWS CLI 傳送低層級 Scan 請求至 DynamoDB，直到結果中 LastEvaluatedKey 不再出現為止。請考慮以下 AWS CLI 示例，該示例掃描整個 Movies 表，但僅返回特定類型的電影。

```
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
```

通常情況下，AWS CLI 句柄自動分頁。但是，在此範例中，AWS CLI --page-size 參數會限制每頁的項目數。--debug 參數會列印請求及回應的下層資訊。

Note

您的分頁結果也會根據您傳遞的輸入參數而有所不同。

- 使用 `aws dynamodb scan --table-name Prices --max-items 1` 會傳回 `NextToken`
- 使用 `aws dynamodb scan --table-name Prices --limit 1` 會傳回 `LastEvaluatedKey`。

另請注意，使用特定的 `--starting-token` 需要 `NextToken` 值。

如果您執行此範例，DynamoDB 的第一個回應會類似以下內容。

```
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}'
```

回應中的 `LastEvaluatedKey` 會指出並未擷取所有項目。AWS CLI 然後會向 DynamoDB 發出另一個 `Scan` 請求。此請求和回應模式會持續到最終回應出現為止。

```
2017-07-07 12:19:17,830 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"WarGames"}}], "ScannedCount":6}'
```

若沒有 `LastEvaluatedKey`，就表示已不再有要擷取的項目。

Note

AWS SDK 會處理低階 DynamoDB 回應 (包括是否存在 `LastEvaluatedKey`)，並為分頁結果提供各種抽象。Scan 例如，適用於 Java 的開發套件文件界面會提供 `java.util.Iterator` 支援，讓您可一次處理一個結果。

如需各種程式設計語言的程式碼範例，請參閱 [《Amazon DynamoDB 入門指南》](#) 和所需語言的 AWS 開發套件文件。

計算結果中的項目

除了符合您條件的項目之外，Scan 回應還包含了下列元素：

- **ScannedCount**：在套用任何 ScanFilter 前評估的項目數。ScannedCount 值很高，但 Count 結果很少或沒有，表示 Scan 操作不足。如未在請求中使用篩選條件，則 ScannedCount 與 Count 相同。
- **Count**：套用篩選條件表達式 (若有的話) 之後剩下的項目數。

Note

若不使用篩選條件表達式，則 ScannedCount 和 Count 會有相同的值。

若 Scan 結果集的大小大於 1 MB，則 ScannedCount 和 Count 僅代表總項目的部分計數。您需要執行多項 Scan 操作，才能擷取所有的結果 (請參閱 [為結果編製分頁](#))。

每個 Scan 回應都包含經該特定 Scan 請求處理過的項目 ScannedCount 和 Count。若要取得所有 Scan 請求的總計，您可以為 ScannedCount 及 Count 記錄流水帳。

掃描使用的容量單位

您可以 Scan 任何資料表或次要索引。Scan 操作會使用讀取容量單位，如下所示。

若您對下列進行 Scan	DynamoDB 使用的讀取容量單位就會來自...
資料表	該資料表的佈建讀取容量。
全域次要索引	該索引的佈建讀取容量。
本機次要索引	該基礎資料表的佈建讀取容量。

根據預設，Scan 操作不會傳回任何使用之讀取容量的相關資料。但您可以在 ReturnConsumedCapacity 請求中指定 Scan 參數，來取得這項資訊。下列為 ReturnConsumedCapacity 的有效設定：

- **NONE**：不會傳回耗用的容量資料。(此為預設值)。
- **TOTAL**：回應包括耗用的讀取容量單位總數。

- INDEXES：回應顯示耗用的讀取容量單位總數，以及每個資料表和存取之索引的耗用容量。

DynamoDB 會根據項目數量和這些項目的大小 (而不是傳回給應用程式的資料量) 來計算使用的讀取容量單位數量。因此，無論您請求所有屬性 (預設行為) 或只請求部分屬性 (使用投影表達式)，使用的容量單位數都相同。無論您是否使用篩選器運算式，數字也相同。Scan 消耗最小讀取容量單位以每秒執行一次強烈一致讀取，或每秒針對 4 KB 以下的項目執行兩次最終一致讀取。如果您需要讀取大於 4KB 的項目，DynamoDB 需要額外的讀取請求單位。空白資料表和非常大型的資料表若有稀疏數量的分割索引鍵，可能會看到額外的 RCU 計費超過掃描的資料量。這涵蓋了服務 Scan 請求的成本，即使沒有數據存在。

掃描的讀取一致性

根據預設，Scan 操作會執行最終一致讀取。這表示 Scan 的結果可能不會反映最近完成之 PutItem 或 UpdateItem 操作所造成的變更。如需詳細資訊，請參閱 [讀取一致性](#)。

若您要進行強烈一致讀取，請在 Scan 開始時，將 ConsistentRead 請求中的 true 參數設為 Scan。這可確保 Scan 開始前即已完成的所有寫入操作，都會包含在 Scan 回應中。

將 ConsistentRead 設為 true，再結合 [DynamoDB Streams](#)，在資料表備份或複寫案例中很實用。首先使用 ConsistentRead 設為 true 的 Scan，取得和資料表資料一致的副本。在 Scan 期間，DynamoDB Streams 會紀錄資料表發生的所有額外寫入活動。Scan 完成後，您可將串流的寫入活動套用至資料表。

Note

與將 ConsistentRead 保留預設值 (false) 相較，將 ConsistentRead 設成 true 的 Scan 操作會使用兩倍的讀取容量單位。

平行掃描

根據預設，Scan 操作會依序處理資料。Amazon DynamoDB 會以 1 MB 的增量將資料傳回應用程式，而應用程式會執行額外的 Scan 操作來擷取下一個 1 MB 的資料。

掃描的資料表或索引越大，Scan 完成所需的時間就更多。此外，循序 Scan 可能並不總是能夠完全使用佈建的讀取輸送容量：即使 DynamoDB 將大型資料表的資料分配到多個實體分割區，Scan 操作一次只能讀取一個分割區。因此，Scan 的輸送量受到單一分割區的最大輸送量限制。

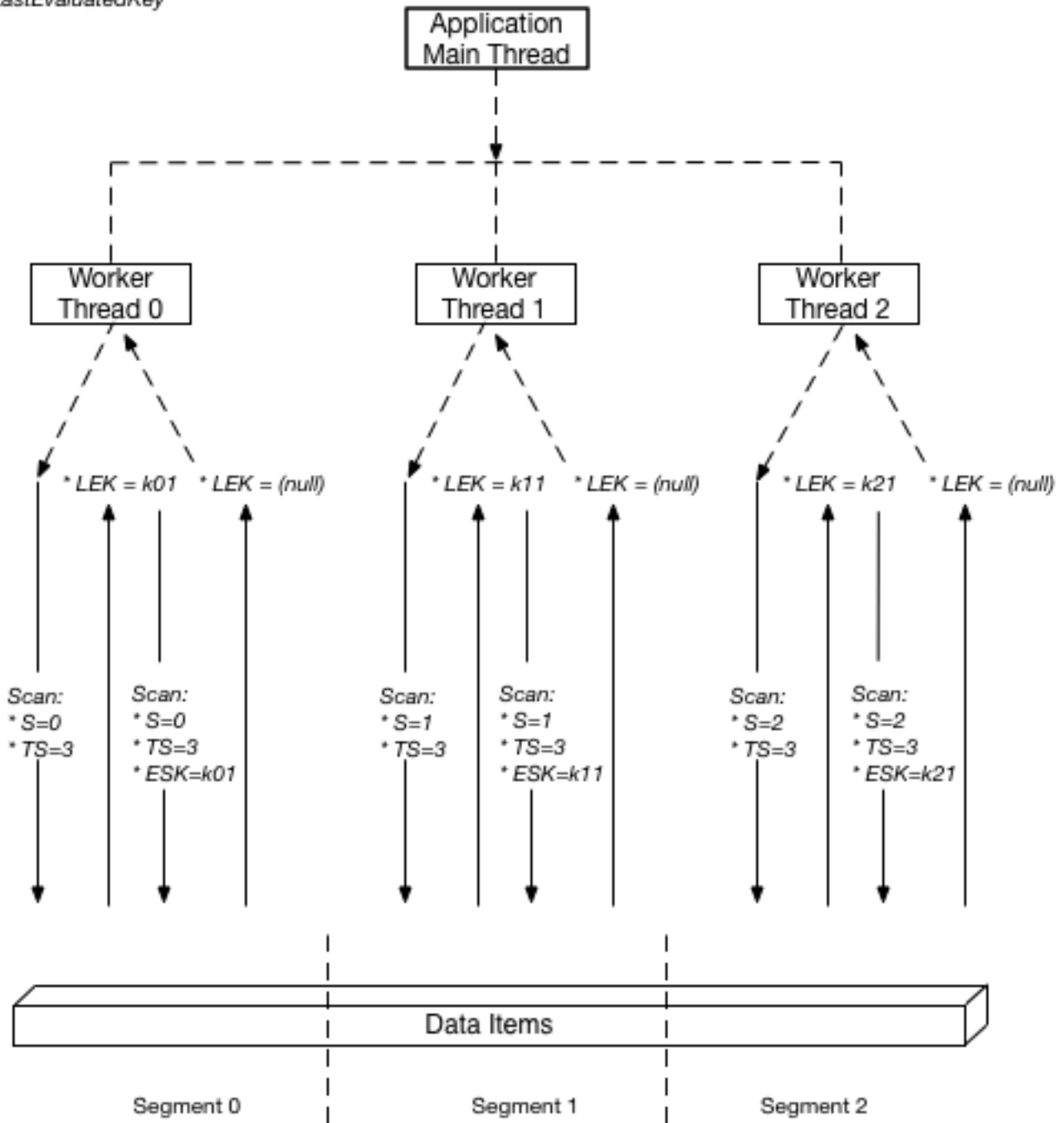
若要解決這些問題，Scan 操作在邏輯上可能會將資料表或次要索引分為多個區段，其中有多個應用程式工作者平行掃描區段。每個工作者都可以是執行緒 (在支援多執行緒的程式設計語言中) 或作業系統程序。若要執行平行掃描，每個工作者都會使用下列參數發出自己的 Scan 請求：

- Segment：特定工作者要掃描的區段。每個工作者應該使用不同的 Segment 值。
- TotalSegments：平行掃描區段的總數。此值必須與應用程式要使用的工作者數目相同。

下圖顯示多執行緒應用程式如何以三度平行處理執行平行 Scan。

S: Segment
TS: TotalSegments

ESK: ExclusiveStartKey
LEK: LastEvaluatedKey



在此圖中，應用程式產生三個執行緒，並為每個執行緒指派一個數字。(區段從零開始，所以第一個數字永遠為 0。) 每個執行緒都會發出 Scan 請求、將 Segment 設定為指定號碼，並將

TotalSegments 設定為 3。每個執行緒都會掃描其指定的區段，一次擷取 1 MB 的資料，並將資料傳回至應用程式的主執行緒。

Segment 和 TotalSegments 的值會套用至個別 Scan 請求，並且您可以隨時使用不同的值。您可能需要對這些值以及使用的工作者數量進行試驗，直到應用程式達到最佳效能為止。

Note

具有大量工作者的平行掃描可以輕易耗用所掃描資料表或索引的所有佈建輸送量。如果資料表或索引也會造成來自其他應用程式的大量讀取或寫入活動，就最好避免這類掃描。若要控制每個請求傳回的資料量，請使用 Limit 參數。這有助於預防某個工作者耗用所有佈建輸送量的情況，避免犧牲所有其他工作者。

掃描資料表及索引：Java

此 Scan 操作會讀取 Amazon DynamoDB 中資料表或索引中的所有項目。

以下是使用 AWS SDK for Java 文件 API 掃描表格的步驟。

1. 建立 AmazonDynamoDB 類別的執行個體。
2. 建立 ScanRequest 類別的執行個體，並提供掃描參數。

唯一所需的參數是資料表名稱。

3. 執行 scan 方法，並提供您在上一個步驟中建立的 ScanRequest 物件。

以下 Reply 資料表會存放論壇主題的回覆。

Example

```
Reply ( Id, ReplyDateTime, Message, PostedBy )
```

此資料表會維護各個論壇主題的所有回覆。因此，主索引鍵同時包含 Id (分割區索引鍵) 和 ReplyDateTime (排序索引鍵)。下列 Java 程式碼範例會掃描整個資料表。此 ScanRequest 執行個體會指定要掃描的資料表名稱。

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);
}
```

指定選用參數

scan 方法支援數個選用的參數。例如，您可以選擇使用篩選表達式來篩選掃描結果。在篩選表達式中，您可以指定條件以及要據以評估條件的屬性名稱和值。如需詳細資訊，請參閱[掃描](#)。

下列 Java 範例會掃描 ProductCatalog 資料表，從中尋找價格低於 0 的項目。此範例會指定下列選用參數：

- 只擷取價格低於 0 的項目的篩選表達式 (錯誤條件)。
- 在查詢結果中要擷取的項目屬性清單。

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);
}
```

您也可以使用掃描請求的 withLimit 方法，選擇性地限制頁面的大小或每個頁面的項目數。每次您執行 scan 方法時，就會擷取一頁具有指定項目數的結果。若要擷取下一頁面，您可以再次執行 scan 方法，並提供先前頁面最後一個項目的主索引鍵值，讓 scan 方法傳回下一組項目。您可以透過使用

`withExclusiveStartKey` 方法，在請求中提供此資訊。最初，此方法的參數可以為 `Null`。若要擷取後續頁面，您必須將此屬性值更新為先前頁面最後一個項目的主索引鍵。

下列 Java 程式碼範例會掃描 `ProductCatalog` 資料表。此請求中會用到 `withLimit` 和 `withExclusiveStartKey` 方法。do/while 迴圈會繼續一次掃描一個頁面，直到結果的 `getLastEvaluatedKey` 方法傳回 `Null` 數值為止。

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);
```

範例 - 使用 Java 掃描

下列 Java 程式碼範例提供的工作範例會掃描 `ProductCatalog` 資料表，從中尋找價格低於 100 的項目。

Note

適用於 Java 的開發套件也提供物件持久性模型，讓您將用戶端類別映射至 DynamoDB 資料表。此方法可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱 [Java 1.x : DynamoDBMapper](#)。

Note

此程式碼範例假設您已根據 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 一節的說明將資料載入 DynamoDB 的帳戶。
如需執行下列範例的 step-by-step 指示，請參閱 [Java 程式碼範例](#)。

```
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());
    }
}
}
```

範例 - 使用 Java 的平行掃描

下列 Java 程式碼範例會示範平行掃描。該程式會刪除並重新建立名為 `ParallelScanTest` 的資料表，然後載入包含資料的資料表。當資料載入完成時，此程式會產生多個執行緒並發出平行 Scan 請求。此程式會為每個平行請求列印執行時間統計數字。

Note

適用於 Java 的開發套件也提供物件持久性模型，讓您將用戶端類別映射至 DynamoDB 資料表。此方法可以減少您必須撰寫的程式碼數量。如需詳細資訊，請參閱 [Java 1.x : DynamoDBMapper](#)。

Note

此程式碼範例假設您已根據 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 一節的說明將資料載入 DynamoDB 的帳戶。
如需執行下列範例的 step-by-step 指示，請參閱 [Java 程式碼範例](#)。

```
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();
    }
}
}
```

掃描資料表及索引：.NET

此 Scan 操作會讀取 Amazon DynamoDB 中資料表或索引中的所有項目。

以下是使用 AWS SDK for .NET 低級 API 掃描表的步驟：

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。
2. 建立 `ScanRequest` 類別的執行個體，並提供掃描操作參數。

唯一所需的參數是資料表名稱。

3. 執行 `Scan` 方法，並提供您在上一個步驟中建立的 `ScanRequest` 物件。

以下 Reply 資料表會存放論壇主題的回覆。

Example

```
>Reply ( Id, ReplyDateTime, Message, PostedBy )
```

此資料表會維護各個論壇主題的所有回覆。因此，主索引鍵同時包含 `Id` (分割區索引鍵) 和 `ReplyDateTime` (排序索引鍵)。下列 C# 程式碼範例會掃描整個資料表。此 `ScanRequest` 執行個體會指定要掃描的資料表名稱。

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);
}
```

指定選用參數

`Scan` 方法支援數個選用的參數。例如，您可以選擇使用掃描篩選條件來篩選掃描結果。在掃描篩選條件中，您可以指定條件以及要據以評估條件的屬性名稱。如需詳細資訊，請參閱[掃描](#)。

以下 C# 程式碼會掃描 `ProductCatalog` 資料表，從中尋找價格低於 0 的項目。範例會指定下列選用參數：

- 只擷取價格低於 0 的項目的 `FilterExpression` 參數 (錯誤條件)。
- 指定在查詢結果中要擷取的項目 `ProjectionExpression` 參數。

下列 C# 範例會掃描 `ProductCatalog` 資料表，從中尋找價格低於 0 的所有項目。

Example

```
var forumScanRequest = new ScanRequest
{
    TableName = "ProductCatalog",
    // Optional parameters.
    ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
        {":val", new AttributeValue { N = "0" }}
    },
    FilterExpression = "Price < :val",
    ProjectionExpression = "Id"
};
```

您也可以透過新增選用的 `Limit` 參數，選擇性地限制頁面大小或是每一頁面的項目數目。每次您執行 `Scan` 方法時，就會擷取一頁具有指定項目數的結果。若要擷取下一頁面，您可以再次執行 `Scan` 方法，並提供先前頁面最後一個項目的主索引鍵值，讓 `Scan` 方法傳回下一組項目。您可以透過設定 `ExclusiveStartKey` 屬性，在請求中提供此資訊。一開始，此屬性可為 `Null`。若要擷取後續頁面，您必須將此屬性值更新為先前頁面最後一個項目的主索引鍵。

下列 C# 程式碼範例會掃描 `ProductCatalog` 資料表。在請求中，它會指定 `Limit` 和 `ExclusiveStartKey` 選用參數。do/while 迴圈會繼續一次掃描一個頁面，直到 `LastEvaluatedKey` 傳回 `Null` 數值為止。

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);
```

範例 - 使用 .NET 進行掃描

下列 C# 程式碼提供的工作範例會掃描 ProductCatalog 資料表，從中尋找價格低於 0 的項目。

如需測試下列範例的 step-by-step 指示，請參閱[.NET 程式碼範例](#)。

```
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("*****");
    }
}
```

範例 - 使用 .NET 進行平行掃描

下列 C# 程式碼範例示範平行掃描。此程式會刪除再重新建立 ProductCatalog 資料表，然後載入含有資料的資料表。當資料載入完成時，此程式會產生多個執行緒並發出平行 Scan 請求。最後，此程式會列印執行時間統計數字的摘要。

如需測試下列範例的 step-by-step 指示，請參閱[.NET 程式碼範例](#)。

```
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.WriteLine("Uploading Items: ");
    for (int itemIndex = 0; itemIndex < exampleItemCount; itemIndex++)
    {
        Console.WriteLine("{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：一種適用於 Amazon DynamoDB 的 SQL 相容查詢語言

Amazon DynamoDB 支援 [PartiQL](#) (SQL 相容查詢語言)，可在 Amazon DynamoDB 中選取、插入、更新和刪除資料。使用 PartiQL，您可以輕鬆地與 DynamoDB 資料表互動，並使用 AWS Management Console、NoSQL Workbench、AWS Command Line Interface 和適用於 PartiQL 的 DynamoDB API 執行隨機查詢。

PartiQL 操作提供與其他 DynamoDB 資料平面操作相同的可用性、延遲和效能。

下列各節將介紹 PartiQL 的 DynamoDB 實作。

主題

- [什麼是 PartiQL？](#)
- [Amazon DynamoDB 中的 PartiQL](#)
- [DynamoDB 專用 PartiQL 入門](#)
- [適用於 DynamoDB 的 PartiQL 資料類型](#)
- [適用於 DynamoDB 的 PartiQL 陳述式](#)
- [搭配 Amazon DynamoDB 使用 PartiQL 函數](#)
- [適用於 DynamoDB 的 PartiQL 算術、比較和邏輯運算子](#)
- [使用 DynamoDB 專用 PartiQL 執行交易](#)

- [使用 DynamoDB 專用 PartiQL 執行批次操作](#)
- [DynamoDB 專用 PartiQL 的 IAM 安全政策](#)

什麼是 PartiQL ?

PartiQL 在包含結構化資料、半結構化資料和巢狀資料的多個資料存放區提供與 SQL 相容的查詢存取。此語言在 Amazon 內廣泛使用，現在作為許多 AWS 服務 (包括 DynamoDB) 的一部分提供。

如需 PartiQL 規範和核心查詢語言的教學課程，請參閱 [PartiQL 文件](#)。

Note

- Amazon DynamoDB 支援 [PartiQL](#) 查詢語言的子集。
- Amazon DynamoDB 不支援 [Amazon Ion](#) 資料格式或 Amazon Ion 文字。

Amazon DynamoDB 中的 PartiQL

若要在 DynamoDB 中執行 PartiQL 查詢，您可以使用：

- DynamoDB 主控台
- NoSQL Workbench
- AWS Command Line Interface (AWS CLI)
- DynamoDB API

如需使用這些方法存取 DynamoDB 的相關資訊，請參閱[存取 DynamoDB](#)。

DynamoDB 專用 PartiQL 入門

本節說明如何從 Amazon DynamoDB 主控台、AWS Command Line Interface (AWS CLI) 和 DynamoDB API 使用 DynamoDB 專用 PartiQL。

在下列範例中，[DynamoDB 入門](#)教學課程中所定義的 DynamoDB 資料表是先決條件。

如需使用 DynamoDB 主控台、AWS Command Line Interface 或 DynamoDB API 來存取 DynamoDB 的相關資訊，請參閱[存取 DynamoDB](#)。

若要下載並使用 [NoSQL Workbench](#) 來建置 [DynamoDB 專用 PartiQL](#) 陳述式，請選擇位於 NoSQL Workbench for DynamoDB [Operation Builder](#) 右上角的 PartiQL operations (PartiQL 操作)。

Console

The screenshot shows the AWS Management Console interface for DynamoDB. On the left, the navigation sidebar includes 'PartiQL editor' (2). The main area shows the 'Music' table (3) with a PartiQL query (5) in the editor. A context menu (4) is open over the query. Below the query, there are 'Table view' and 'JSON view' options (6). The query execution status is 'Completed' (7). The results table shows columns for AlbumTitle, Awards, Artist, and SongTitle, with data rows for 'Somewhat ...', 'Songs Abou...', and 'Happy Day'.

Note

只有新版 DynamoDB 主控台中才能使用 DynamoDB 專用 PartiQL。若要使用新版 DynamoDB 主控台，請選擇主控台左側導覽窗格中的 Try the Preview of the new console (試用預覽版新主控台)。

1. 登入 AWS Management Console，並在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 PartiQL editor (PartiQL 編輯器)。
3. 選擇 Music (音樂) 資料表。
4. 選擇 Query table (查詢資料表)。此動作會產生不會導致完整資料表掃描的查詢。
5. 使用字串值 Acme Band 取代 partitionKeyValue。使用字串值 Happy Day 取代 sortKeyValue。
6. 選擇 Run (執行) 按鈕。
7. 您可以選擇 Table view (資料表檢視) 或 JSON view (JSON 檢視) 按鈕來檢視查詢的結果。

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
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. 選擇 PartiQL statement (PartiQL 陳述式)。
 2. 輸入下列 PartiQL [SELECT 陳述式](#)
- ```
SELECT *
FROM Music
WHERE Artist=? and SongTitle=?
```
3. 若要指定 Artist 和 SongTitle 參數的值：
    - a. 選擇 Optional request parameters (選用的請求參數)。
    - b. 選擇 Add new parameters (新增新參數)。
    - c. 選擇屬性類型 string 和數值 Acme Band。

- d. 重複步驟 b 和 c，然後選擇類型 string 和數值 PartiQL Rocks。
4. 若要產生程式碼，請選擇 Generate code (產生程式碼)。從顯示的標籤中選取所需的語言。您現在可以複製此程式碼，並使用在您的應用程式中。
5. 若希望立即執行此操作，請選擇 Run (執行)。

## AWS CLI

1. 使用 INSERT PartiQL 陳述式在 Music 資料表中建立項目。

```
aws dynamodb execute-statement --statement "INSERT INTO Music \
 VALUE \
 {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

2. 使用 SELECT PartiQL 陳述式從 Music 資料表中檢索項目。

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

3. 使用 UPDATE PartiQL 陳述式在 Music 資料表中更新項目。

```
aws dynamodb execute-statement --statement "UPDATE Music \
 SET AwardsWon=1 \
 SET AwardDetail={'Grammys':[2020,
 2018]} \
 WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

在 Music 資料表中新增項目的清單值。

```
aws dynamodb execute-statement --statement "UPDATE Music \
 SET AwardDetail.Grammys
 =list_append(AwardDetail.Grammys,[2016]) \
 WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

在 Music 資料表中移除項目的清單值。

```
aws dynamodb execute-statement --statement "UPDATE Music \
```

```
REMOVE AwardDetail.Grammys[2] \
WHERE Artist='Acme Band' AND
SongTitle='PartiQL Rocks''
```

在 Music 資料表中新增項目的新映射成員。

```
aws dynamodb execute-statement --statement "UPDATE Music \
SET AwardDetail.BillBoard=[2020] \
WHERE Artist='Acme Band' AND
SongTitle='PartiQL Rocks''
```

在 Music 資料表中新增項目的新字串集屬性。

```
aws dynamodb execute-statement --statement "UPDATE Music \
SET BandMembers =<<'member1',
'member2'>> \
WHERE Artist='Acme Band' AND
SongTitle='PartiQL Rocks''
```

在 Music 資料表中更新項目的字串集屬性。

```
aws dynamodb execute-statement --statement "UPDATE Music \
SET BandMembers
=set_add(BandMembers, <<'newmember'>>) \
WHERE Artist='Acme Band' AND
SongTitle='PartiQL Rocks''
```

4. 使用刪除 PartiQL 陳述式從 Music 資料表中刪除項目。

```
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 DynamoDBPartiQGettingStarted {

 public static void main(String[] args) {
 // Create the DynamoDB Client with the region you want
 AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-1");

 try {
 // Create ExecuteStatementRequest
 ExecuteStatementRequest executeStatementRequest = new
ExecuteStatementRequest();
 List<AttributeValue> parameters= getPartiQLParameters();

 //Create an item in the Music table using the INSERT PartiQL statement
 processResults(executeStatementRequest(dynamoDB, "INSERT INTO Music
value {'Artist':?, 'SongTitle':?}" , parameters));

 //Retrieve an item from the Music table using the SELECT PartiQL
statement.
 processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
where Artist=? and SongTitle=?", parameters));

 //Update an item in the Music table using the UPDATE PartiQL statement.
 processResults(executeStatementRequest(dynamoDB, "UPDATE Music
SET AwardsWon=1 SET AwardDetail={'Grammys':[2020, 2018]} where Artist=? and
SongTitle=?", parameters));

 //Add a list value for an item in the Music table.
 processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016]) where Artist=? and
SongTitle=?", parameters));
```

```
 //Remove a list value for an item in the Music table.
 processResults(executeStatementRequest(dynamoDB, "UPDATE Music REMOVE
AwardDetail.Grammys[2] where Artist=? and SongTitle=?", parameters));

 //Add a new map member for an item in the Music table.
 processResults(executeStatementRequest(dynamoDB, "UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist=? and SongTitle=?", parameters));

 //Add a new string set attribute for an item in the Music table.
 processResults(executeStatementRequest(dynamoDB, "UPDATE Music
SET BandMembers =<<'member1', 'member2'>> where Artist=? and SongTitle=?",
parameters));

 //update a string set attribute for an item in the Music table.
 processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
BandMembers =set_add(BandMembers, <<'newmember'>>) where Artist=? and SongTitle=?",
parameters));

 //Retrieve an item from the Music table using the SELECT PartiQL
statement.
 processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
where Artist=? and SongTitle=?", parameters));

 //delete an item from the Music Table
 processResults(executeStatementRequest(dynamoDB, "DELETE FROM Music
where Artist=? and SongTitle=?", parameters));
 } catch (Exception e) {
 handleExecuteStatementErrors(e);
 }
}

private static AmazonDynamoDB createDynamoDbClient(String region) {
 return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
}

private static List<AttributeValue> getPartiQLParameters() {
 List<AttributeValue> parameters = new ArrayList<AttributeValue>();
 parameters.add(new AttributeValue("Acme Band"));
 parameters.add(new AttributeValue("PartiQL Rocks"));
 return parameters;
}
```



```
private static ExecuteStatementResult executeStatementRequest(AmazonDynamoDB
client, String statement, List<AttributeValue> parameters) {
 ExecuteStatementRequest request = new ExecuteStatementRequest();
 request.setStatement(statement);
 request.setParameters(parameters);
 return client.executeStatement(request);
}

private static void processResults(ExecuteStatementResult
executeStatementResult) {
 System.out.println("ExecuteStatement successful: "+
executeStatementResult.toString());
}

// Handles errors during ExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
// your application use-case.
private static void handleExecuteStatementErrors(Exception exception) {
 try {
 throw exception;
 } catch (ConditionalCheckFailedException ccfe) {
 System.out.println("Condition check specified in the operation failed,
review and update the condition " +
 "check before retrying. Error: " +
ccfe.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.getMessage());
 } 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.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());
 }
}
}
```

## 適用於 DynamoDB 的 PartiQL 資料類型

下表列出您可搭配 DynamoDB 專用 PartiQL 使用的資料類型。

| DynamoDB 資料類型 | PartiQL 表示法              | 備註                                |
|---------------|--------------------------|-----------------------------------|
| Boolean       | TRUE   FALSE             | 不區分大小寫。                           |
| Binary        | N/A                      | 僅透過程式碼支援。                         |
| List          | [ value1, value2,...]    | 清單類型中可存放的資料類型不限，而且清單中的元素不必屬於相同類型。 |
| Map           | { 'name' : value }       | 映射類型中可存放的資料類型不限，而且映射中的元素不必屬於相同類型。 |
| Null          | NULL                     | 不區分大小寫。                           |
| Number        | 1、1.0、1 e0               | 數字可以是正數、負數或零。數字精確度最高可達 38 位數。     |
| Number Set    | <<number1, number2>>     | 數字集中的元素必須是 Number (數字) 類型。        |
| String Set    | <<'string1', 'string2'>> | 字串集中的元素必須是 String (字串) 類型。        |
| String        | 'string value'           | 必須使用單引號來指定 String (字串) 值。         |

### 範例

下述陳述式示範如何插入以下資料類型：String、Number、Map、List、Number Set 和 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'>>
}
```

下述陳述式示範如何將新的元素插入 Map、List、Number Set 和 String Set 類型，並變更 Number 類型的數值。

```
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'
```

下述陳述式示範如何從 Map、List、Number Set 和 String Set 類型移除元素，並變更 Number 類型的數值。

```
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'
```

如需詳細資訊，請參閱 [DynamoDB 資料類型](#)。

## 適用於 DynamoDB 的 PartiQL 陳述式

Amazon DynamoDB 支援下列 PartiQL 陳述式。

### Note

DynamoDB 不支援所有 PartiQL 陳述式。  
此參考提供您使用 AWS CLI 或 API 手動執行的 PartiQL 陳述式基本語法和使用範例。

資料操作語言 (DML) 是一組用來管理 DynamoDB 資料表中資料的 PartiQL 陳述式。您可使用 DML 陳述式新增、修改或刪除資料表中的資料。

支援下列 DML 和查詢語言陳述式：

- [適用於 DynamoDB 的 PartiQL Select 陳述式](#)
- [適用於 DynamoDB 的 PartiQL Update 陳述式](#)
- [適用於 DynamoDB 的 PartiQL Insert 陳述式](#)
- [適用於 DynamoDB 的 PartiQL Delete 陳述式](#)

DynamoDB 專用 PartiQL 亦支援 [使用 DynamoDB 專用 PartiQL 執行交易](#) 和 [使用 DynamoDB 專用 PartiQL 執行批次操作](#)。

適用於 DynamoDB 的 PartiQL Select 陳述式

在 Amazon DynamoDB 中，使用 SELECT 陳述式從資料表檢索資料。

如果 WHERE 子句中未提供具有分區索引鍵的相等或 IN 條件，使用 SELECT 陳述式可能會導致完整資料表掃描。掃描操作會檢查每個項目的要求值，並可能會在單一操作中耗用大型資料表或索引的佈建輸送量。

如果您想避免在 PartiQL 中進行完整的資料表掃描，則可以：

- 確保 [WHERE 子句條件](#) 會據此進行相應設定，撰寫您的 SELECT 陳述式便不會導致完整的資料表掃描。
- 請使用《DynamoDB 開發人員指南》中 [範例：允許在 DynamoDB 專用 PartiQL 中執行選取陳述式並拒絕完整的資料表掃描陳述式](#) 所述的 IAM 政策來停用完整的資料表掃描。

如需詳細資訊，請參閱《DynamoDB 開發人員指南》中的 [查詢和掃描資料的最佳實務](#)。

主題

- [語法](#)
- [參數](#)
- [範例](#)

語法

```
SELECT expression [, ...]
FROM table[.index]
```

```
[WHERE condition] [[ORDER BY key [DESC|ASC] , ...]
```

## 參數

### ###

(必要) 從 \* 萬用字元形成的投影，或來自結果集中一或多個屬性名稱或文件路徑的投影清單。一個表達式可以包含對 [搭配 Amazon DynamoDB 使用 PartiQL 函數](#) 的呼叫或由 [適用於 DynamoDB 的 PartiQL 算術、比較和邏輯運算子](#) 修改的欄位。

### ##

(必要) 要查詢的資料表名稱。

### ##

(選用) 要查詢的索引名稱。

#### Note

查詢索引時，必須在資料表名稱和索引名稱上加上雙引號。

```
SELECT *
FROM "TableName"."IndexName"
```

## *condition*

(選用) 查詢的選取條件。

#### Important

若要確保 SELECT 陳述式不會導致完整的資料表掃描，WHERE 子句條件必須指定分割區索引鍵。使用等於或 IN 運算子。

例如，如果一個 Orders 資料表包含 OrderID 分割區索引鍵和其他非索引鍵屬性 (包括 Address)，則下列陳述式不會導致完整的資料表掃描：

```
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]
```

不過，下列 SELECT 陳述式會導致完整的資料表掃描：

```
SELECT *
FROM "Orders"
WHERE OrderID > 1

SELECT *
FROM "Orders"
WHERE Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 OR Address='some address'
```

## ##

(選用) 用來排序傳回結果的雜湊索引鍵或排序索引鍵。預設順序是升序 (ASC)；如果您希望依降序傳回結果，請指定 DESC。

### Note

如果您省略 WHERE 子句，則會檢索資料表中的所有項目。

## 範例

如果存在一個項目，則下列查詢會指定分割區索引鍵和 OrderID 並使用等於運算子，從 Orders 資料表傳回一個項目。

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1
```

下列查詢會針對具有特定分割區索引鍵、OrderID 和值的 Orders 資料表，使用 OR 運算子傳回其中的所有項目。

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1 OR OrderID = 2
```

下列查詢會針對具有特定分割區索引鍵、OrderID 和值的 Orders 資料表，使用 IN 運算子傳回其中的所有項目。傳回的結果會根據 OrderID 索引鍵屬性值以遞減順序顯示。

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID IN [1, 2, 3] ORDER BY OrderID DESC
```

下列查詢顯示完整的資料表掃描，該掃描會傳回 Orders 資料表中 Total 大於 500 的所有項目，其中 Total 是非索引鍵屬性。

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total > 500
```

下列查詢顯示完整的資料表掃描，該掃描會傳回使用 IN 運算子和非索引鍵屬性 Total 之特定 Total 順序範圍內 Orders 資料表中的所有項目。

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total IN [500, 600]
```

下列查詢顯示完整的資料表掃描，該掃描會傳回使用 BETWEEN 運算子和非索引鍵屬性 Total 之特定 Total 順序範圍內 Orders 資料表中的所有項目。

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total BETWEEN 500 AND 600
```



下列查詢會在 WHERE 子句條件中指定分割區索引鍵 CustomerID 和排序索引鍵 MovieID，並在 SELECT 子句中使用文件路徑，以此傳回 FireStick 裝置用於觀看的第一個日期。

```
SELECT Devices.FireStick.DateWatched[0]
FROM WatchList
WHERE CustomerID= 'C1' AND MovieID= 'M1'
```

下列查詢會顯示完整的資料表掃描，該掃描會使用 WHERE 子句條件中的文件路徑傳回在 2019 年 12 月 24 日之後首次使用 FireStick 裝置的項目清單。

```
SELECT Devices
FROM WatchList
WHERE Devices.FireStick.DateWatched[0] >= '12/24/19'
```

適用於 DynamoDB 的 PartiQL Update 陳述式

使用 UPDATE 陳述式來修改 Amazon DynamoDB 資料表中項目內一或多個屬性的值。

#### Note

您一次只能更新一個項目，因為您無法發出可刪除多個項目的單個 DynamoDB PartiQL 陳述式。如需更新多個項目的相關資訊，請參閱 [使用 DynamoDB 專用 PartiQL 執行交易](#) 或 [使用 DynamoDB 專用 PartiQL 執行批次操作](#)。

## 主題

- [語法](#)
- [參數](#)
- [傳回值](#)
- [範例](#)

## 語法

```
UPDATE table
[SET | REMOVE] path [= data] [...]
WHERE condition [RETURNING returnvalues]
<returnvalues> ::= [ALL OLD | MODIFIED OLD | ALL NEW | MODIFIED NEW] *
```

## 參數

### ##

(必要) 包含要修改之資料的資料表。

### ##

(必要) 要建立或修改的屬性名稱或文件路徑。

### *data*

(必要) 屬性值或操作的結果。

與 SET 搭配使用的支援操作：

- LIST\_APPEND：將數值新增至清單類型。
- SET\_ADD：將數值新增至數字或字串集。
- SET\_DELETE：從數字或字串集中刪除數值。

### *condition*

(必要) 要修改之項目的選取條件。此條件必須解析為單一主索引鍵值。

### *returnvalues*

(選用) 若想取得更新之前或之後出現的項目屬性，則請使用 `returnvalues`。有效值為：

- ALL OLD \*：傳回更新操作之前出現的所有項目屬性。
- MODIFIED OLD \*：僅傳回新操作之前出現的更新屬性。
- ALL NEW \*：傳回更新操作之後出現的所有項目屬性。
- MODIFIED NEW \*：僅傳回 UpdateItem 操作之後出現的更新屬性。

## 傳回值

此陳述式不會傳回值，除非指定 `returnvalues` 參數。

### Note

如果 DynamoDB 資料表中任何項目的 UPDATE 陳述式的 WHERE 子句未評估為 true，則會傳回 ConditionalCheckFailedException。

## 範例

更新現有項目中的屬性值。如果屬性不存在，則會建立此屬性。

下列查詢會透過新增數字類型的屬性 (AwardsWon) 和映射類型的屬性 (AwardDetail) 來更新 "Music" 資料表中的項目。

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

您可以新增 RETURNING ALL OLD \* 以傳回 Update 操作前顯示的屬性。

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL OLD *
```

這會傳回下列內容：

```
{
 "Items": [
 {
 "Artist": {
 "S": "Acme Band"
 },
 "SongTitle": {
 "S": "PartiQL Rocks"
 }
 }
]
}
```

您可以新增 RETURNING ALL NEW \* 以傳回 Update 操作後顯示的屬性。

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL NEW *
```

這會傳回下列內容：

```
{
 "Items": [
 {
 "AwardDetail": {
 "M": {
 "Grammys": {
 "L": [
 {
 "N": "2020"
 },
 {
 "N": "2018"
 }
]
 }
 }
 },
 "AwardsWon": {
 "N": "1"
 }
 }
]
}
```

下列查詢會透過附加至清單 `AwardDetail.Grammys` 來更新 "Music" 中的項目。

```
UPDATE "Music"
SET AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

下列查詢會透過從清單 `AwardDetail.Grammys` 中移除來更新 "Music" 資料表中的項目。

```
UPDATE "Music"
REMOVE AwardDetail.Grammys[2]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

下列查詢會透過將 `BillBoard` 新增至映射 `AwardDetail` 來更新 "Music" 資料表中的項目。

```
UPDATE "Music"
SET AwardDetail.BillBoard=[2020]
```

```
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

下列查詢會透過新增字串集屬性 BandMembers 來更新 "Music" 資料表中的項目。

```
UPDATE "Music"
SET BandMembers =<<'member1', 'member2'>>
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

下列查詢會透過將 newbandmember 新增至字串集 BandMembers 來更新 "Music" 資料表中的項目。

```
UPDATE "Music"
SET BandMembers =set_add(BandMembers, <<'newbandmember'>>)
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

適用於 DynamoDB 的 PartiQL Delete 陳述式

使用 DELETE 陳述式從您的 Amazon DynamoDB 資料表中刪除現有項目。

#### Note

您一次只能刪除一個項目。您無法發出可刪除多個項目的單個 DynamoDB PartiQL 陳述式。如需刪除多個項目的相關資訊，請參閱 [使用 DynamoDB 專用 PartiQL 執行交易](#) 或 [使用 DynamoDB 專用 PartiQL 執行批次操作](#)。

## 主題

- [語法](#)
- [參數](#)
- [傳回值](#)
- [範例](#)

## 語法

```
DELETE FROM table
WHERE condition [RETURNING returnvalues]
<returnvalues> ::= ALL OLD *
```

## 參數

### ##

(必要) 包含要刪除之項目的 DynamoDB 資料表。

### **condition**

(必要) 要刪除之項目的選取條件；此條件必須解析為單一主索引鍵值。

### **returnvalues**

(選用) 若想取得在刪除之前出現的項目屬性，則請使用 `returnvalues`。有效值為：

- `ALL OLD *`：傳回舊項目的內容。

## 傳回值

此陳述式不會傳回值，除非指定 `returnvalues` 參數。

### Note

如果 DynamoDB 資料表沒有任何與發出 `DELETE` 的項目具有相同主索引鍵的項目，則會傳回 `SUCCESS`，並刪除 0 個項目。如果資料表具有相同主索引鍵的項目，但 `DELETE` 陳述式的 `WHERE` 子句中的條件評估結果為 `false`，則會傳回 `ConditionalCheckFailedException`。

## 範例

下列查詢會刪除 "Music" 資料表中的一個項目。

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

您可以新增參數 `RETURNING ALL OLD *` 以傳回已刪除的資料。

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
RETURNING ALL OLD *
```

Delete 陳述式現在傳回下列內容：

```
{
```

```
"Items": [
 {
 "Artist": {
 "S": "Acme Band"
 },
 "SongTitle": {
 "S": "PartiQL Rocks"
 }
 }
]
```

適用於 DynamoDB 的 PartiQL Insert 陳述式

使用 INSERT 陳述式在 Amazon DynamoDB 中將項目新增至資料表。

#### Note

您一次只能插入一個項目，因為您無法發出可插入多個項目的單個 DynamoDB PartiQL 陳述式。如需插入多個項目的相關資訊，請參閱 [使用 DynamoDB 專用 PartiQL 執行交易](#) 或 [使用 DynamoDB 專用 PartiQL 執行批次操作](#)。

## 主題

- [語法](#)
- [參數](#)
- [傳回值](#)
- [範例](#)

## 語法

插入單一項目。

```
INSERT INTO table VALUE item
```

## 參數

### ##

(必要) 您要插入資料的資料表。索引資料表必須已存在。

## ##

(必要) 有效的 DynamoDB 項目表示為 [PartiQL 元組](#)。您必須在 PartiQL 中僅指定一個項目，項目中的每個屬性名稱均區分大小寫，且可以使用單引號 ('...') 表示。

PartiQL 中的字串值也使用單引號 ('...') 表示。

### 傳回值

此陳述式不會傳回任何值。

#### Note

如果 DynamoDB 資料表已經有一個項目的主索引鍵與插入的項目的主索引鍵相同，則會傳回 `DuplicateItemException`。

### 範例

```
INSERT INTO "Music" value {'Artist' : 'Acme Band', 'SongTitle' : 'PartiQL Rocks'}
```

### 搭配 Amazon DynamoDB 使用 PartiQL 函數

Amazon DynamoDB 中的 PartiQL 支援下列 SQL 標準函數的內建變體。

#### Note

DynamoDB 目前不支援任何未包含在此清單中的 SQL 函數。

### 彙總函數

- [搭配 Amazon DynamoDB 專用 PartiQL 使用 SIZE 函數](#)

### 條件函數

- [搭配 DynamoDB 專用 PartiQL 使用 EXISTS 函數](#)
- [搭配 DynamoDB 專用 PartiQL 使用 ATTRIBUTE\\_TYPE 函數](#)



- [搭配 DynamoDB 專用 PartiQL 使用 BEGINS\\_WITH 函數](#)
- [搭配 DynamoDB 專用 PartiQL 使用 CONTAINS 函數](#)
- [搭配 DynamoDB 專用 PartiQL 使用 MISSING 函數](#)

## 搭配 DynamoDB 專用 PartiQL 使用 EXISTS 函數

您可以使用 EXISTS 來執行與 ConditionCheck 會在 [TransactWriteItems](#) API 中執行的相同函數。EXISTS 函數只能在交易中使用。

給定一個值，如果該值是一個非空集合，則傳回 TRUE。如果不是，則傳回 FALSE。

### Note

此函數只能在交易操作中使用。

## 語法

```
EXISTS (statement)
```

## 引數

###

(必要) 函數評估的 SELECT 陳述式。

### Note

SELECT 陳述式必須指定完整的主索引鍵和其他一個條件。

## 傳回類型

bool

## 範例

```
EXISTS(
 SELECT * FROM "Music"
```

```
WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks')
```

搭配 DynamoDB 專用 PartiQL 使用 `BEGINS_WITH` 函數

如果指定的屬性以特定子字串開頭，則傳回 TRUE。

語法

```
begins_with(path, value)
```

引數

**##**

(必要) 要使用的屬性名稱或文件路徑。

**#**

(必要) 要搜尋的字串。

傳回類型

bool

範例

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND begins_with("Address", '7834 24th')
```

搭配 DynamoDB 專用 PartiQL 使用 `MISSING` 函數

如果項目不包含指定的屬性，即傳回 TRUE。只有等於和不等於運算子可以與此函數一起使用。

語法

```
attributename IS | IS NOT MISSING
```

引數

***attributename***

(必要) 要尋找的屬性名稱。

## 傳回類型

bool

## 範例

```
SELECT * FROM Music WHERE "Awards" is MISSING
```

搭配 DynamoDB 專用 PartiQL 使用 ATTRIBUTE\_TYPE 函數

如果指定路徑的屬性是特定資料類型，則傳回 TRUE。

## 語法

```
attribute_type(attributename, type)
```

## 引數

*attributename*

(必要) 要使用的屬性名稱。

##

(必要) 要檢查的屬性類型。如需有效值的清單，請參閱 DynamoDB [attribute\\_type](#)。

## 傳回類型

bool

## 範例

```
SELECT * FROM "Music" WHERE attribute_type("Artist", 'S')
```

搭配 DynamoDB 專用 PartiQL 使用 CONTAINS 函數

如果路徑指定的屬性為下列其中一個項目，則傳回 TRUE：

- 包含特定子字串的字串。
- 在組合中內含特定元素的組合。

如需詳細資訊，請參閱 DynamoDB 的 [contains](#) 函數。

## 語法

```
contains(path, substring)
```

## 引數

##

(必要) 要使用的屬性名稱或文件路徑。

###

(必要) 要檢查的屬性子字串或集合成員。如需詳細資訊，請參閱 DynamoDB 的 [contains](#) 函數。

## 傳回類型

bool

## 範例

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND contains("Address", 'Kirkland')
```

## 搭配 Amazon DynamoDB 專用 PartiQL 使用 SIZE 函數

傳回代表屬性大小的數字 (以位元組為單位)。以下是可搭配 size 使用的有效資料類型。如需詳細資訊，請參閱 DynamoDB 的 [size](#) 函數。

## 語法

```
size(path)
```

## 引數

##

(必要) 屬性名稱或文件路徑。

如需了解支援的類型，請參閱 DynamoDB [size](#) 函數。

## 傳回類型

int

## 範例

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND size("Image") >300
```

## 適用於 DynamoDB 的 PartiQL 算術、比較和邏輯運算子

Amazon DynamoDB 中的 PartiQL 支援以下 [SQL 標準運算子](#)。

### Note

DynamoDB 目前不支援任何未包含在此清單中的 SQL 運算子。

## 算術運算子

| 運算子 | 描述 |
|-----|----|
| +   | 加  |
| -   | 減  |

## 比較運算子

| 運算子 | 描述  |
|-----|-----|
| =   | 等於  |
| <>  | 不等於 |
| !=  | 不等於 |
| >   | 大於  |
| <   | 小於  |

| 運算子 | 描述    |
|-----|-------|
| >=  | 大於或等於 |
| <=  | 小於或等於 |

## 邏輯運算子

| 運算子     | 描述                                                      |
|---------|---------------------------------------------------------|
| AND     | 如果以 AND 分隔的所有的條件為 TRUE，則為 TRUE                          |
| BETWEEN | 如果運算元在比較範圍內，則為 TRUE                                     |
| IN      | TRUE 如果運算元等於表達式清單中的一個表達式 (最多為 50 個雜湊屬性值或最多 100 個非金鑰屬性值) |
| IS      | 如果運算元是給定的 PartiQL 資料類型 (包括 NULL 或 MISSING)，則為 TRUE      |
| NOT     | 反轉指定布林表達式的值                                             |
| OR      | 如果以 OR 分隔任何的條件為 TRUE，則為 TRUE                            |

## 使用 DynamoDB 專用 PartiQL 執行交易

本節說明如何搭配 DynamoDB 專用 PartiQL 使用交易。PartiQL 交易限制為全部 100 個陳述式 (動作)。

如需 DynamoDB 交易的詳細資訊，請參閱[管理包含 DynamoDB 交易的複雜工作流程](#)。

### Note

整個交易必須由讀取陳述式或寫入陳述式所組成。你不能在一個交易中混合兩者。EXISTS 函數為例外。您可以使用它，以類似於在 [TransactWriteItems](#) API 操作中的方式檢查項目的特定屬性的條件。ConditionCheck

## 主題

- [語法](#)
- [參數](#)
- [傳回值](#)
- [範例](#)

## 語法

```
[
 {
 "Statement": " statement ",
 "Parameters": [
 {
 " parametertype " : " parametervalue "
 }, ...]
 }, ...
]
```

## 參數

### ###

(必要) DynamoDB 專用 PartiQL 支援的陳述式。

#### Note

整個交易必須由讀取陳述式或寫入陳述式所組成。你不能在一個交易中混合兩者。

### *parametertype*

(選用) DynamoDB 類型，如果在指定 PartiQL 陳述式時使用了參數。

### *parametervalue*

(選用) 參數值，如果在指定 PartiQL 陳述式時使用了參數。

## 傳回值

此陳述式不會傳回寫入作業 (插入、更新或刪除) 的任何值。但是，它根據 WHERE 子句中指定的條件回傳讀取操作 (SELECT) 的不同值。

### Note

如果任何單一 INSERT、UPDATE 或 DELETE 操作傳回錯誤，交易會取消並出現 `TransactionCanceledException` 例外狀況，且取消原因程式碼會包含個別單一操作的錯誤。

## 範例

以下範例會以交易形式執行多個陳述式。

## AWS CLI

1. 將以下 JSON 程式碼儲存至名為 `partiql.json` 的檔案

```
[
 {
 "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. 在命令提示中執行下列命令。

```
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 isee) {
 System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + isee.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());
 }
}
}
```

下列範例顯示當 DynamoDB 讀取 WHERE 子句中指定之不同條件的項目時，不同的傳回值。

## AWS CLI

1. 將以下 JSON 程式碼儲存至名為 partiql.json 的檔案

```
[
 // 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. 在命令提示中執行命令。

```
aws dynamodb execute-transaction --transact-statements file://partiql.json
```

3. 會傳回下列回應：

```

{
 "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
 {}
]
}

```

## 使用 DynamoDB 專用 PartiQL 執行批次操作

本節說明如何搭配 DynamoDB 專用 PartiQL 使用批次陳述式。

### Note

- 整個批次必須由讀取陳述式或寫入陳述式所組成；一個批次中不能同時出現這兩種陳述式樣。
- BatchExecuteStatement 和 BatchWriteItem 每個批次不可以執行超過 25 條陳述式。

### 主題

- [語法](#)
- [參數](#)
- [範例](#)

### 語法

```
[
 {
 "Statement": " statement ",
 "Parameters": [
 {
 " parametertype " : " parametervalue "
 }, ...]
 } , ...
]
```

### 參數

#### ###

(必要) DynamoDB 專用 PartiQL 支援的陳述式。

### Note

- 整個批次必須由讀取陳述式或寫入陳述式所組成；一個批次中不能同時出現這兩種陳述式樣。

- `BatchExecuteStatement` 和 `BatchWriteItem` 每個批次不可以執行超過 25 條陳述式。

### *parametertype*

(選用) DynamoDB 類型，如果在指定 PartiQL 陳述式時使用了參數。

### *parametervalue*

(選用) 參數值，如果在指定 PartiQL 陳述式時使用了參數。

## 範例

### AWS CLI

1. 將以下 json 儲存至名為 `partiql.json` 的檔案

```
[
 {
 "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. 在命令提示中執行下列命令。

```
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());
 }
}
}
```

## DynamoDB 專用 PartiQL 的 IAM 安全政策

需要以下許可：

- 若要使用 DynamoDB 專用 PartiQL 讀取項目，您必須擁有資料表或索引的 `dynamodb:PartiQLSelect` 許可。
- 若要使用 DynamoDB 專用 PartiQL 插入項目，您必須擁有資料表或索引的 `dynamodb:PartiQLInsert` 許可。
- 若要使用 DynamoDB 專用 PartiQL 更新項目，您必須擁有資料表或索引的 `dynamodb:PartiQLUpdate` 許可。
- 若要使用 DynamoDB 專用 PartiQL 刪除項目，您必須擁有資料表或索引的 `dynamodb:PartiQLDelete` 許可。

範例：允許在資料表上執行所有 DynamoDB 專用 PartiQL 陳述式 (選取/插入/更新/刪除)

下列 IAM 政策會授予許可，允許在資料表上執行所有 DynamoDB 專用 PartiQL 陳述式。

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

範例：允許在資料表上執行 DynamoDB 專用 PartiQL 選取陳述式

下列 IAM 政策會授予許可，允許在特定資料表上執行 select 陳述式。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "dynamodb:PartiQLSelect"
],
 "Resource": [
 "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
]
 }
]
}
```

範例：允許在索引上執行 DynamoDB 專用 PartiQL 插入陳述式

下列 IAM 政策會授予許可，允許在特定索引上執行 insert 陳述式。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "dynamodb:PartiQLInsert"
],
 "Resource": [
 "arn:aws:dynamodb:us-west-2:123456789012:table/Music/index/index1"
]
 }
]
}
```

```
]
}
```

範例：允許在資料表上僅執行 DynamoDB 專用 PartiQL 交易陳述式

下列 IAM 政策會授予許可，允許僅在特定資料表上執行交易陳述式。

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

範例：允許在資料表上執行 DynamoDB 專用 PartiQL 非交易式讀取和寫入，並封鎖 PARTSQL 交易式讀取和寫入交易陳述式。

下列 IAM 政策會授予許可，允許執行 DynamoDB 專用 PartiQL 非交易式讀取和寫入，同時封鎖 DynamoDB 專用 PartiQL 交易式讀取和寫入。

```
{
 "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"
]
 }
]
}

```

範例：允許在 DynamoDB 專用 PartiQL 中執行選取陳述式並拒絕完整的資料表掃描陳述式

下列 IAM 政策會授予許可，允許在特定資料表上執行 select 陳述式，同時封鎖會導致完整資料表掃描的 select 陳述式。

```

{
 "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"
]
 }
]
```

## 使用次要索引來改善資料存取

### 主題

- [在 DynamoDB 中使用全域次要索引](#)
- [本機次要索引](#)

Amazon DynamoDB 透過指定主索引鍵值來提供資料表中項目的快速存取。不過，許多應用程式都會受益於下列情況：有一或多個次要 (或替代) 索引鍵可用，允許使用主索引鍵以外的屬性來有效存取資料。若要解決此問題，您可以在資料表上建立一或多個次要索引，並針對這些索引發出 Query 或 Scan 請求。

次要索引是一種資料結構，包含資料表中的屬性子集，以及可支援 Query 操作的替代索引鍵。您可以使用 Query 從索引擷取資料，方式與您搭配使用 Query 與資料表相同。一份資料表可以有許多次要索引，讓您的應用程式能夠存取許多不同的查詢模式。

**Note**

您也可以對索引進行 Scan，方式與您對資料表進行 Scan 相同。

每個次要索引都只能與一個資料表建立關聯，次要索引可從資料表中取得其資料。這稱為索引的基礎資料表。當您建立索引時，請定義索引的替代索引鍵 (分割區索引鍵和排序索引鍵)。您還可以定義要從基礎資料表投影或複製到索引中的屬性。DynamoDB 會將這些屬性以及基礎資料表中的主索引鍵屬性複製到索引。您接著可以查詢或掃描索引，就像查詢或掃描資料表一樣。

DynamoDB 會自動維護每個次要索引。當您新增、修改或刪除基礎資料表中的項目時，也會更新該資料表上的任何索引，以反映這些變更。

DynamoDB 支援兩種次要索引：

- **全域次要索引**：一種含分割區索引鍵或排序索引鍵的索引，這些索引鍵可與基礎資料表上的索引鍵不同。全域次要索引之所以視為全域，是因為索引上的查詢可以跨越所有分割區之間基礎資料表中的所有資料。全域次要索引會儲存在基本資料表以外的專屬分割區空間，且會和基本資料表分開調整。
- **本機次要索引**：是一種與基礎資料表擁有相同分割區索引鍵但不同排序索引鍵的索引。本機次要索引的「本機」概念是指，本機次要索引的每個分割區都會列入基礎資料表分割區的範圍，其中分割區索引鍵的值皆相同。

如需全域次要索引與本機次要索引的比較，請觀看此影片。

### [在 GSI 和 LSI 之間做出正確的選擇](#)

當您判斷要使用的索引類型時，應該考量應用程式需求。下列資料表會顯示全域次要索引和本機次要索引之間的主要差異。

| 特性      | 全域次要索引                                        | 本機次要索引                                  |
|---------|-----------------------------------------------|-----------------------------------------|
| 索引鍵結構描述 | 全域次要索引的主索引鍵可以是簡單 (分割區索引鍵) 或複合 (分割區索引鍵和排序索引鍵)。 | 本機次要索引的主索引鍵必須是複合形式 (分割區索引鍵和排序索引鍵)。      |
| 索引鍵屬性   | 索引的分割區索引鍵和排序索引鍵 (若存在) 可以是字串、數                 | 索引的分割區索引鍵與基礎資料表的分割區索引鍵具有相同的屬性。排序索引鍵可以是字 |

| 特性             | 全域次要索引                                                                                                              | 本機次要索引                                                                                                      |
|----------------|---------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
|                | 字或二進位類型的任何基礎資料表屬性。                                                                                                  | 串、數字或二進位類型的任何基礎資料表屬性。                                                                                       |
| 每個分割區索引鍵值的大小限制 | 全域次要索引沒有大小限制。                                                                                                       | 針對每個分割區索引鍵值，所有已索引項目的大小總計必須是 10 GB 或以下。                                                                      |
| 線上索引操作         | 您可在建立資料表的同時建立全域次要索引。您也可以將新的全域次要索引新增至現有的資料表，或刪除現有的全域次要索引。如需詳細資訊，請參閱 <a href="#">管理全域次要索引</a> 。                       | 本機次要索引是在建立資料表的同時建立的。您無法對現有資料表新增本機次要索引，也無法刪除現有的任何本機次要索引。                                                     |
| 查詢和分割區         | 全域次要索引可讓您跨所有分割區查詢整個資料表。                                                                                             | 本機次要索引可讓您查詢單一分割區 (由查詢中的分割區索引鍵值所指定)。                                                                         |
| 讀取一致性          | 全域次要索引上的查詢僅支援最終一致性。                                                                                                 | 當您查詢本機次要索引時，可以選擇最終一致性或強烈一致性。                                                                                |
| 佈建輸送量耗用        | 針對讀取和寫入活動，每個全域次要索引都有自己的佈建輸送量設定。對全域次要索引的查詢或掃描會使用來自索引的容量單位，而不是來自基礎資料表。這適用於資料表寫入引發的全域次要索引更新。與全域資料表相關聯的全域次要索引會耗用寫入容量單位。 | 對本機次要索引的查詢或掃描會使用來自基礎資料表的讀取容量單位。當您寫入至資料表時，也會一併更新其本機次要索引，且這些更新會使用來自基礎資料表的寫入容量單位。與全域資料表相關聯的本機次要索引會耗用複寫的寫入容量單位。 |
| 投影屬性           | 您可以使用全域次要索引查詢或掃描來僅請求投影至索引的屬性。DynamoDB 不會從資料表中擷取任何屬性。                                                                | 若要查詢或掃描本機次要索引，您可以請求未投影至索引的屬性。DynamoDB 會自動從資料表中擷取這些屬性。                                                       |

如果您想要建立具有次要索引的多個資料表，則必須循序進行這項操作。例如，您建立第一個資料表並等待它變成 ACTIVE、建立下一個資料表並等待其變成 ACTIVE，以此類推。如果您嘗試並行建立具有次要索引的多份資料表，則 DynamoDB 會傳回 `LimitExceededException`。

每個次要索引使用的[資料表類別](#)和[容量模式](#)皆與其相關聯的基本資料表相同。針對每個次要索引，您必須指定下列項目：

- 要建立的索引類型：全域次要索引或本機次要索引。
- 索引的名稱。索引的命名規則與資料表的命名規則相同，如「[Amazon DynamoDB 中的服務、帳戶和資料表配額](#)」中所列。名稱對與其建立關聯的基礎資料表而言必須是唯一的，但您可以將相同的名稱用於與其不同基礎資料表建立關聯的索引。
- 索引的索引鍵結構描述。索引鍵結構描述中的每個屬性都必須是 String、Number 或 Binary 類型的最上層屬性。不允許其他資料類型 (包含文件和集合)。索引鍵結構描述的其他需求取決於索引類型：
  - 針對全域次要索引，分割區索引鍵可以是基礎資料表的任何純量屬性。排序索引鍵是選用項目，它也可以是基礎資料表的任何純量屬性。
  - 針對本機次要索引，分割區索引鍵必須與基礎資料表的分割區索引鍵相同，而排序索引鍵必須是非索引鍵基礎資料表屬性。
- 從基礎資料表投影至索引的其他屬性 (如果有的話)。這些是自動投影至每個索引之資料表索引鍵屬性以外的屬性。您可以投影任何資料類型的屬性 (包含純量、文件和集合)。
- 索引的佈建輸送量設定 (必要時)：
  - 針對全域次要索引，您必須指定讀取和寫入容量單位設定。這些佈建輸送量設定與基礎資料表設定無關。
  - 針對本機次要索引，您不需要指定讀取和寫入容量單位設定。本機次要索引的任何讀取和寫入操作都是取自其基礎資料表的佈建輸送量設定。

為獲得最大的查詢靈活性，您最多可以為每份資料表建立 20 個全域次要索引 (預設配額) 和 5 個本機次要索引。

下列 AWS 區域每份資料表的全域次要索引配額為 5 個：

- AWS GovCloud (美國東部)
- AWS GovCloud (美國西部)
- 歐洲 (斯德哥爾摩)



若要取得資料表次要索引的詳細清單，請使用 DescribeTable 操作。DescribeTable 會傳回資料表中每個次要索引的名稱、儲存大小和項目計數。這些值不會即時更新，但大約每六小時會重新整理一次。

您可以使用 Query 或 Scan 操作來存取次要索引中的資料。您必須指定基礎資料表名稱以及您要使用的索引名稱、要在結果中傳回的屬性，以及您要套用的任何條件表達式或篩選條件。DynamoDB 可以依遞增或遞減順序傳回結果。

當您刪除資料表時，也會刪除與該資料表建立關聯的所有索引。

如需最佳實務做法，請參閱「[使用 DynamoDB 中次要索引的最佳實務](#)」。

## 在 DynamoDB 中使用全域次要索引

某些應用程式可能需要執行多種查詢，並使用各種不同的屬性做為查詢條件。若要支援這些需求，您可以建立一或多個全域次要索引，並在 Amazon DynamoDB 中對這些索引發出 Query 請求。

### 主題

- [藍本：使用全域次要索引](#)
- [屬性投影](#)
- [從全域次要索引讀取資料](#)
- [資料表與全域次要索引之間的資料同步](#)
- [具有全域次要索引的資料表類別](#)
- [全域次要索引的佈建輸送量考量](#)
- [全域次要索引的儲存考量](#)
- [管理全域次要索引](#)
- [使用全域次要索引：Java](#)
- [使用全域次要索引：.NET](#)
- [使用全域次要索引：AWS CLI](#)

### 藍本：使用全域次要索引

為了示範，假設有一份名為 GameScores 的資料表，會追蹤手機遊戲應用程式的使用者與分數。GameScores 中的每個項目都是按分割區索引鍵 (UserId) 和排序索引鍵 (GameTitle) 識別。下圖顯示這些項目在資料表中的組織方式。(並未顯示所有屬性。)

## 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     | ... |
| ...    | ...               | ...      | ...                   | ...  | ...    | ... |

現在假設您想要撰寫排行榜應用程式來顯示每個遊戲的最高分。已指定索引鍵屬性 (UserId 和 GameTitle) 的查詢會很有效率。但若應用程式僅能根據 GameTitle 從 GameScores 擷取資料，就需要使用 Scan 操作。當愈來愈多項目新增至資料表時，掃描所有資料會變得很慢且效率不彰。這會讓回答問題變得很困難，例如下列問題：

- Meteor Blasters 遊戲曾記錄到的最高分為何？
- 哪位使用者在 Galaxy Invaders 中得到最高分？
- 最高的勝負比為何？

若要加速非索引鍵屬性的查詢，您可以建立全域次要索引。全域次要索引包含基礎資料表中的一系列屬性，但這些屬性會依與該資料表不同的主索引鍵組織。索引鍵不需要有任何來自資料表的索引鍵屬性。甚至不需要有和資料表相同的索引鍵結構描述。

例如，您可以建立名為 GameTitleIndex 的全域次要索引，其中分割區索引鍵為 GameTitle，排序索引鍵為 TopScore。因為基礎資料表的主索引鍵屬性一律會投影到索引，所以也會顯示 UserId 屬性。下圖說明 GameTitleIndex 索引可能的樣子。

## GameTitleIndex

| GameTitle         | TopScore | UserId |
|-------------------|----------|--------|
| "Alien Adventure" | 192      | "102"  |
| "Attack Ships"    | 3        | "103"  |
| "Galaxy Invaders" | 0        | "102"  |
| "Galaxy Invaders" | 2317     | "103"  |
| "Galaxy Invaders" | 5842     | "101"  |
| "Meteor Blasters" | 723      | "103"  |
| "Meteor Blasters" | 1000     | "101"  |
| "Starship X"      | 24       | "101"  |
| "Starship X"      | 42       | "103"  |
| ...               | ...      | ...    |

您現在可以查詢 GameTitleIndex 並輕鬆取得 Meteor Blasters 的分數。結果會依排序索引鍵值 TopScore 排序。如果您將 ScanIndexForward 參數設定為 false，結果會依遞減順序傳回，因此最高分優先傳回。

每個全域次要索引都必須有分割區索引鍵，並可以有一個選用的排序索引鍵。索引鍵結構描述可以和基礎資料表結構描述不同。您可以擁有一個具有簡易主索引鍵 (分割區索引鍵) 的資料表，並使用複合主索引鍵 (分割區索引鍵和排序索引鍵) 建立全域次要索引，反之亦然。索引鍵屬性可包含來自基礎資料表的任何最上層 String、Number 或 Binary 屬性。不允許其他純量類型、文件類型和集合類型。

您可以視需要將其他基礎資料表屬性投影到索引。當您查詢索引時，DynamoDB 可有效率地擷取這些投影屬性。但是，全域次要索引查詢無法自基礎資料表擷取屬性。例如，如果您依上圖所示查詢 GameTitleIndex，則除 TopScore 外，此查詢不能存取任何非索引鍵屬性 (雖然會自動投影索引鍵屬性 GameTitle 和 UserId)。

在 DynamoDB 資料表中，每個索引鍵值必須是唯一的。不過，全域次要索引中的索引鍵值不需要是唯一的。為了示範，假設有一款名為 Comet Quest 的遊戲特別難，許多新的使用者試過後均無法突破零分的成績。以下為可代表此遊戲的一些資料。

| UserId | GameTitle   | TopScore |
|--------|-------------|----------|
| 123    | Comet Quest | 0        |
| 201    | Comet Quest | 0        |
| 301    | Comet Quest | 0        |

當此資料新增至 GameScores 資料表時，DynamoDB 會將其傳播至 GameTitleIndex。如果我們接著使用 GameTitle Comet Quest 與 TopScore 0 來查詢索引，即會傳回下列資料。

| GameTitle     | TopScore | UserId |
|---------------|----------|--------|
| "Comet Quest" | 0        | "123"  |
| "Comet Quest" | 0        | "201"  |
| "Comet Quest" | 0        | "301"  |

回應中只會顯示具有指定索引鍵值的項目。在此資料集中，項目未依特定順序排列。

全域次要索引只會追蹤索引鍵屬性確實存在的資料項目。例如，假設您在 GameScores 表中新增另一個新項目，但只提供必要的主索引鍵屬性。

| UserId | GameTitle   |
|--------|-------------|
| 400    | Comet Quest |

因為您未指定 TopScore 屬性，所以 DynamoDB 不會將此項目傳播至 GameTitleIndex。因此，如已針對 GameScores 查詢所有 Comet Quest 項目，可能會取得下列四個項目。

| UserId | GameTitle     | TopScore |
|--------|---------------|----------|
| "123"  | "Comet Quest" | 0        |
| "201"  | "Comet Quest" | 0        |
| "301"  | "Comet Quest" | 0        |
| "400"  | "Comet Quest" |          |

針對 `GameTitleIndex` 的類似查詢仍只會傳回三個項目，而不是四個。這是因為具有不存在 `TopScore` 的項目不會傳播至索引。

| <i>GameTitle</i> | <i>TopScore</i> | <i>UserId</i> |
|------------------|-----------------|---------------|
| "Comet Quest"    | 0               | "123"         |
| "Comet Quest"    | 0               | "201"         |
| "Comet Quest"    | 0               | "301"         |

## 屬性投影

投影是指從資料表複製到次要索引的屬性集合。資料表的分割區索引鍵和排序索引鍵一律會投影到索引中；您可以投影其他屬性來支援應用程式的查詢需求。查詢索引時，Amazon DynamoDB 可以存取投影中的任何屬性，就好像這些屬性在它們自己的資料表中一樣。

在建立次要索引時，您需要指定要投影到索引中的屬性。DynamoDB 為此提供三種不同的選項：

- `KEYS_ONLY`：索引中的每個項目都只包含資料表分割索引鍵和排序索引鍵值，以及索引鍵值。`KEYS_ONLY` 選項會產生最小的可行次要索引。
- `INCLUDE`：除了 `KEYS_ONLY` 中描述的屬性外，次要索引還包含您指定的其他非索引鍵屬性。
- `ALL`：次要索引包含來源資料表中的所有屬性。因為索引中的所有資料表資料都會重複，所以 `ALL` 投影會產生最大的可行次要索引。

在上圖中，`GameTitleIndex` 只有一個投影的屬性：`UserId`。因此，雖然應用程式可以在查詢中使用 `GameTitle` 和 `TopScore` 有效率地判斷每個遊戲最高分得主的 `UserId`，卻無法有效率地判斷最高分得主的最高勝負比。若要做到這一點，應用程式必須在基礎資料表執行額外的查詢，以擷取每個最高分得主的勝負比。支援查詢此資料更有效率的方法，便是將這些屬性從基礎資料表投影到全域次要索引，如下圖所示。

## 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     |
| ...               | ...      | ...    | ...  | ...    |

由於非索引鍵屬性 Wins 與 Losses 會投影到索引，因此應用程式可以判斷任何遊戲或任何遊戲與使用者 ID 組合的勝負比。

在選擇要投影到全域次要索引的屬性時，您必須權衡佈建輸送量成本和儲存成本：

- 若您只需要存取少量的屬性，並且希望盡可能地降低延遲，建議您考慮只將那些屬性投影到全域次要索引。索引愈小，存放成本就愈低，您的寫入成本也愈低。
- 如果應用程式會頻繁存取某些非索引鍵屬性，您應該考慮將這些屬性投影到全域次要索引。全域次要索引的額外儲存成本會抵銷頻繁執行資料表掃描的成本。
- 如果需要頻繁存取大多數的非索引鍵屬性，您可以將這些屬性 (或整個基礎資料表) 投影到全域次要索引。這能讓您掌握最大的靈活性。但儲存成本可能會增加，甚至翻倍。
- 如果您的應用程式不需頻繁查詢資料表，但必須對資料表中的資料執行許多寫入或更新，請考慮投影 KEYS\_ONLY。全域次要索引的大小將會最小，但仍能在需要的時候進行查詢活動。

## 從全域次要索引讀取資料

您可以使用 Query 和 Scan 操作從全域次要索引擷取項目。GetItem 和 BatchGetItem 操作不能用於全域次要索引。

### 查詢全域次要索引

您可以使用 Query 操作存取全域次要索引中一或多個項目。詢必須指定基礎資料表名稱以及您要使用的索引名稱、要在查詢結果中傳回的屬性，以及您要套用的任何查詢條件。DynamoDB 可以依遞增或遞減順序傳回結果。

假設下列資料從請求排行榜應用程式遊戲資料的 Query 傳回。

```
{
 "TableName": "GameScores",
 "IndexName": "GameTitleIndex",
 "KeyConditionExpression": "GameTitle = :v_title",
 "ExpressionAttributeValues": {
 ":v_title": {"S": "Meteor Blasters"}
 },
 "ProjectionExpression": "UserId, TopScore",
 "ScanIndexForward": false
}
```

在此查詢中：

- DynamoDB 存取 GameTitleIndex，使用 GameTitle 分區索引鍵來尋找流星爆破器的索引項目。所有具有此索引鍵的索引項目都會相鄰存放，以利快速擷取。
- 在此遊戲中，DynamoDB 使用索引存取此遊戲的所有使用者 ID 與最高分。
- 結果會傳回並遞減排序，因為 ScanIndexForward 參數設定為 false。

### 掃描全域次要索引

您可以使用 Scan 操作從全域次要索引檢索所有資料。您必須在請求中提供基礎資料表名稱及索引名稱。使用 Scan，DynamoDB 會讀取索引中的所有資料，並將其傳回應用程式。您也可以請求只傳回一部分的資料，並捨棄其他剩餘的資料。若要執行此作業，請使用 FilterExpression 操作的 Scan 參數。如需詳細資訊，請參閱 [掃描的篩選條件表達式](#)。



## 資料表與全域次要索引之間的資料同步

DynamoDB 會自動將每個全域次要索引與其基礎資料表同步。當應用程式寫入或刪除資料表中的項目時，該資料表的任何全域次要索引都會使用最終一致模型非同步更新。應用程式永遠不會直接寫入索引。然而，了解 DynamoDB 維持這些索引之方式的含義很重要。

全域次要索引會從基礎資料表繼承讀取/寫入容量模式。如需詳細資訊，請參閱 [切換容量模式時的考量](#)。

在建立全域次要索引時，您要指定一或多個索引鍵屬性及其資料類型。這表示每次您將項目寫入基礎資料表時，這些屬性的資料類型都必須符合索引鍵結構描述的資料類型。在 GameTitleIndex 案例中，索引的 GameTitle 分割區索引鍵是定義為 String 資料類型。索引的 TopScore 排序索引鍵類型為 Number。如果您嘗試在 GameScores 表中新增項目，並為 GameTitle 或 TopScore 指定不同的資料類型，DynamoDB 會因資料類型不符而傳回 ValidationException。

當您在資料表中放置或刪除項目時，該資料表的全域次要索引會以最終一致的方式更新。在正常的情況下，資料表的資料變更會在瞬間傳播至全域次要索引。不過，在某些不太可能發生的失敗情況下，可能發生較久的傳播延遲。因此，您的應用程式必須能夠預期及處理針對全域次要索引查詢，卻傳回非最新結果的情況。

如果將項目寫入資料表，您不需指定任何全域次要索引排序索引鍵的屬性。以 GameTitleIndex 為例，您不需指定 TopScore 屬性的值，也能將新的項目寫入 GameScores 表。在此案例中，DynamoDB 不會將任何資料寫入此特定項目的索引。

相較於索引較少的資料表，全域次要索引較多的資料表會有較高的寫入活動成本。如需詳細資訊，請參閱 [全域次要索引的佈建輸送量考量](#)。

### 具有全域次要索引的資料表類別

全域次要索引永遠會使用相同的資料表類別做為其基礎資料表。每當新增資料表的全域次要索引時，新索引都會使用與其基礎資料表相同的資料表類別。更新資料表的資料表類別時，也會更新所有相關聯的全域次要索引。

### 全域次要索引的佈建輸送量考量

當您對佈建模式資料表建立全域次要索引時，必須為該索引的預期工作負載指定讀取與寫入容量單位。全域次要索引的佈建輸送量設定與其基礎資料表的佈建輸送量設定無關。對全域次要索引執行 Query 操作會使用索引 (而不是基礎資料表) 的讀取容量單位。當您在資料表中放置、更新或刪除項目時，也會更新該資料表中的全域次要索引。這些索引更新會使用來自索引的寫入容量單位，而不是基礎資料表的單位。



例如，如果對全域次要索引執行 Query 操作，並超出其佈建讀取容量，您的請求就會經過調節。如果您對資料表執行大量寫入活動，但該資料表全域次要索引的寫入容量不足，即會調節該資料表的寫入活動。

### Important

若要避免調節問題產生，全域次要索引的佈建寫入容量必須大於等於基礎資料表的寫入容量，因為新的更新會同時寫入基礎資料表及全域次要索引。

若要檢視全域次要索引的佈建輸送量設定，請使用 DescribeTable 操作。會傳回所有資料表全域次要索引的詳細資訊。

### 讀取容量單位

全域次要索引支援最終一致讀取，每個讀取均使用一半的讀取容量單位。這表示單一全域次要索引查詢每個讀取容量單位最多可擷取  $2 \times 4 \text{ KB} = 8 \text{ KB}$ 。

在全域次要索引查詢方面，DynamoDB 會使用為資料表查詢計算佈建讀取活動相同的方式，計算佈建讀取活動。唯一的差別在於計算方式是以索引項目的大小為基礎，而非基礎資料表中項目的大小。讀取容量單位數為所有傳回項目中，所有投影屬性大小的總和。然後，結果四捨五入至下一個 4 KB 界限。如需 DynamoDB 如何計算佈建輸送用量的詳細資訊，請參閱 [佈建容量模式](#)。

Query 操作傳回的結果大小上限是 1 MB。這包含所有傳回項目之所有屬性名稱與值的大小。

例如，假設有一個全域次要索引，其每個項目均包含 2,000 個位元組的資料。現在假設您 Query 此索引，而此查詢的 KeyConditionExpression 會傳回 8 個項目。相符項目的總大小為 2,000 位元組  $\times$  8 個項目 = 16,000 位元組。然後，此結果四捨五入至最近的 4 KB 界限。因為全域次要索引查詢為最終一致，所以總成本為  $0.5 \times (16 \text{ KB}/4 \text{ KB})$  或 2 個讀取容量單位。

### 寫入容量單位

當新增、更新或刪除資料表項目，並影響到全域次要索引時，全域次要索引會使用該操作的佈建寫入容量單位。寫入的總佈建輸送量成本包含寫入基礎資料表使用的寫入容量單位加上更新全域次要索引使用的寫入容量單位。如果資料表的寫入作業不需更新全域次要索引，就不會使用該索引的寫入容量。

為成功寫入資料表，資料表及其所有全域次要索引的佈建輸送量設定皆必須具有足以容納寫入的寫入容量。否則資料表的寫入會受到調節。

將項目寫入全域次要索引的成本取決於幾個因素：

- 若您將項目寫入定義索引屬性的資料表，或是您將現有的項目更新為先前未定義的索引屬性，則將該項目寫入索引需要進行一次寫入操作。
- 若資料表的更新會變更索引鍵屬性的值 (從 A 到 B)，則需要兩次寫入：一次是從索引刪除先前的項目，第二次則是將新的項目寫入索引。
- 若項目存在於索引中，但寫入資料表致使索引屬性遭到刪除，則需要進行一次寫入，從索引刪除舊項目的投影。
- 若項目在更新之前或之後並不存在於索引中，則該索引將不會有任何額外的寫入成本。
- 如果資料表的更新只變更索引鍵結構描述中投影屬性的值，但並不變更任何索引鍵屬性的值，則需執行一個寫入將投影屬性的值更新至索引中。

所有這些因素都假設索引中每個項目的大小都小於或等於 1 KB 項目大小 (用於計算寫入容量單位)。較大的索引項目需要額外的寫入容量單位。您可以考慮您的查詢所需要傳回的屬性，並只將那些屬性投影到索引中，來將寫入成本降至最小。

## 全域次要索引的儲存考量

當應用程式將項目寫入資料表時，DynamoDB 會自動將正確的部分屬性複製到應顯示這些屬性的任何全域次要索引中。您的 AWS 帳戶需支付基底表格中項目的儲存費用，以及該表格上任何全域次要索引中的屬性儲存費用。

索引項目使用的空間數為下列項目的總和：

- 基礎資料表主索引鍵 (分割區索引鍵和排序索引鍵) 的大小 (位元組)
- 索引鍵屬性的大小 (位元組)
- 投影屬性 (若有的話) 的大小 (位元組)
- 每個索引項目 100 位元組的額外負荷

若要估算全域次要索引的儲存需求，您可以估算索引中項目的平均大小，再乘以基礎資料表中具有全域次要索引鍵屬性的項目數。

如果資料表包含特定屬性未經定義的項目，但該屬性卻已定義為索引的分割區索引鍵或排序索引鍵時，DynamoDB 不會將該項目的任何資料寫入索引。

## 管理全域次要索引

本節說明如何在 Amazon DynamoDB 中建立、修改和刪除全域次要索引。

## 主題

- [建立具有全域次要索引的資料表](#)
- [描述資料表上的全域次要索引](#)
- [將全域次要索引新增至現有資料表](#)
- [刪除全域次要索引](#)
- [建立期間修改全域次要索引](#)
- [偵測和修正索引鍵違規情況](#)

### 建立具有全域次要索引的資料表

若要建立具有一或多個全域次要索引的資料表，請搭配 `CreateTable` 參數使用 `GlobalSecondaryIndexes` 操作。為達最大的查詢靈活性，您最多可以為每個資料表建立 20 個全域次要索引 (預設配額)。

您必須指定一個屬性做為分割區索引鍵。您可以選擇為排序索引鍵指定另一個屬性。上述任一索引鍵屬性皆不需與資料表的索引鍵屬性相同。例如，在 `GameScores` 表格中 (請參閱 [在 DynamoDB 中使用全域次要索引](#))，`TopScore` 也不 `TopScoreDateTime` 是關鍵屬性。您可以建立分割區索引鍵為 `TopScore`、排序索引鍵為 `TopScoreDateTime` 的全域次要索引。您可能會使用這類索引來判斷高分與一天中玩遊戲的時間之間是否有相互關聯性。

每個索引鍵屬性都必須是類型為 `String`、`Number` 或 `Binary` 的純量。(而不可以是文件或集合)。您可以將任何資料類型的屬性投影到全域次要索引。這包括純量、文件和集合。如需資料類型的完整清單，請參閱 [資料類型](#)。

如果使用的是佈建模式，您必須為索引提供 `ProvisionedThroughput` 設定，包含 `ReadCapacityUnits` 和 `WriteCapacityUnits`。這些佈建輸送量設定與資料表的佈建輸送量設定無關，但運作方式類似。如需詳細資訊，請參閱 [全域次要索引的佈建輸送量考量](#)。

全域次要索引會從基礎資料表繼承讀取/寫入容量模式。如需詳細資訊，請參閱 [切換容量模式時的考量](#)。

#### Note

回填操作和正在進行的寫入操作在全域次要索引中共享寫入輸送量。建立新 GSI 時，檢查您選擇的分割區索引鍵是否會在新索引的分割區索引鍵值之間產生資料或流量的不均勻分佈或小範圍分佈，這一點非常重要。如果發生這種情況，您可能會看到回填和寫入操作同時發生，並調

節對基礎資料表的寫入。該服務會採取措施以將此情況的可能性降至最低，但沒有與索引分割區索引鍵、所選擇的投射或索引主鍵稀疏程度有關的客戶資料形狀的洞察。

如果您懷疑新的全域次要索引在分割區索引鍵值上可能具有小範圍或偏斜的資料或流量分佈，請在將新索引新增到對操作而言重要的資料表之前，考慮以下項目。

- 在應用程式驅動的流量為最少時新增索引可能是最安全的做法。
- 請考慮在基底資料表和索引上啟用 CloudWatch 參與者深入解析。這將為您的流量分佈提供有價值的洞察。
- 對於佈建的容量模式基礎資料表和索引，請將新索引的佈建寫入容量設為至少是基礎資料表的兩倍。在整個過程中觀看 `WriteThrottleEventsOnlineIndexPercentageProgress`、`OnlineIndexConsumedWriteCapacityUnits` 和 `OnlineIndexConsumedWriteCapacityUnits` CloudWatch 指標。 `ThrottledRequests` 根據需要調整佈建的寫入容量，以便在合理的時間內完成回填，而不會對正在進行的操作產生任何重大的調節作用。
- 如果您面臨因為寫入調節而影響操作且提高新 GSI 中的佈建寫入容量也無法解決的問題，應做好取消索引建立的準備。

## 描述資料表上的全域次要索引

若要檢視資料表上所有全域次要索引的狀態，請使用 `DescribeTable` 操作。回應的 `GlobalSecondaryIndexes` 部分會顯示資料表上的所有索引，以及每個索引的目前狀態 (`IndexStatus`)。

全域次要索引的 `IndexStatus` 會是下列其中一項：

- `CREATING`：正在建立索引，尚無法使用。
- `ACTIVE`：索引已可供使用，應用程式可以在索引上執行 `Query` 操作。
- `UPDATING`：正在變更索引的佈建輸送量設定。
- `DELETING`：正在刪除索引，無法再使用。

當 DynamoDB 建置完全域次要索引時，索引狀態會從 `CREATING` 變更為 `ACTIVE`。

## 將全域次要索引新增至現有資料表

若要將全域次要索引新增至現有資料表，請搭配 `GlobalSecondaryIndexUpdates` 參數使用 `UpdateTable` 操作。您必須提供下列項目：

- 索引名稱。該名稱在資料表的所有索引中必須是唯一的。

- 索引的索引鍵結構描述。您必須指定一個屬性做為索引的分割區索引鍵，並可以選擇性地指定另一個屬性做為索引的排序索引鍵。上述任一索引鍵屬性皆不需與資料表的索引鍵屬性相同。每個結構描述屬性的資料類型都必須是純量：String、Number 或 Binary。
- 要從資料表投影到索引的屬性：
  - KEYS\_ONLY：索引中的每個項目都只包含資料表分割索引鍵和排序索引鍵值，以及索引鍵值。
  - INCLUDE：除了 KEYS\_ONLY 中描述的屬性外，次要索引會包含您指定的其他非索引鍵屬性。
  - ALL：索引包含來源資料表中的所有屬性。
- 索引的佈建輸送量設定，包含 ReadCapacityUnits 和 WriteCapacityUnits。這些佈建輸送量設定與資料表的佈建輸送量設定無關。

每個 UpdateTable 操作只能建立一個全域次要索引。

### 索引建立階段

當您將新的全域次要索引新增至現有資料表時，資料表在索引建置時仍可繼續使用。但新的索引在其狀態從 CREATING 變更為 ACTIVE 前，都不可進行 Query 操作。

#### Note

全域次要索引建立不使用 Application Auto Scaling。提高 MIN Application Auto Scaling 容量不會減少全域次要索引的建立時間。

DynamoDB 幕後的索引建置分為兩個階段：

### 資源配置

DynamoDB 會配置建立索引所需的運算和儲存資源。

在資源配置階段，IndexStatus 屬性是 CREATING，而 Backfilling 屬性是 false。您可使用 DescribeTable 操作擷取資料表的狀態和其所有次要索引。

當索引處於資源配置階段時，您無法刪除索引或刪除其父資料表。您也無法修改索引或資料表的佈建輸送量。您無法新增或刪除資料表上的其他索引。不過，您可以修改這些其他索引的佈建輸送量。

### 回填

對於資料表中的每個項目，DynamoDB 會根據屬性的投影，判斷將哪一組屬性寫入索引 (KEYS\_ONLY、INCLUDE 或 ALL)。然後它會將這些屬性寫入索引。在回填階段期間，DynamoDB

會追蹤正在資料表中新增、刪除或更新的項目。也會適當地在索引中新增、刪除或更新這些項目的屬性。

在回填階段，`IndexStatus` 屬性會設為 `CREATING`，而 `Backfilling` 屬性是 `true`。您可使用 `DescribeTable` 操作擷取資料表的狀態和其所有次要索引。

當索引正在回填時，您無法刪除其父資料表。但您仍然可以刪除索引，或修改資料表及其任何全域次要索引的佈建輸送量。

#### Note

在回填階段，違規項目的寫入可能一部分成功，而一部分則遭到拒絕。回填後，所有違反新索引鍵結構描述的項目寫入都會遭到拒絕。建議您在回填階段完成後執行 `Violation Detector` 工具，以偵測和解決任何可能已發生的索引鍵違規情況。如需詳細資訊，請參閱 [偵測和修正索引鍵違規情況](#)。

進行資源配置和回填階段時，索引的狀態為 `CREATING`。在這期間，DynamoDB 會對資料表執行讀取操作。從基礎資料表填入全域次要索引的讀取操作不需支付費用。不過，您需要為填入新建立全域次要索引的寫入操作支付費用。

當索引建置完成時，其狀態會變更為 `ACTIVE`。您無法 `Query` 或 `Scan` 索引，直到其 `ACTIVE` 為止。

#### Note

在部分情況下，DynamoDB 會因索引鍵違規情況而無法將資料表中的資料寫入索引。如果發生下列情況，就會發生此情形：

- 屬性值的資料類型與索引鍵結構描述資料類型的資料類型不相符。
- 屬性的大小超過索引鍵屬性的最大長度。
- 索引鍵屬性具有空字串或空的二進位屬性值。

索引鍵違規情況不會影響全域次要索引建立。不過，當索引變成 `ACTIVE` 時，索引中不會出現違規索引鍵。

DynamoDB 提供獨立工具來尋找和解決這些問題。如需詳細資訊，請參閱 [偵測和修正索引鍵違規情況](#)。

## 將全域次要索引新增至大型資料表

建置全域次要索引所需的時間取決於幾個因素，例如：

- 資料表的大小
- 資料表中符合納入索引資格的項目數目
- 投影到索引的屬性數目
- 索引的佈建寫入容量
- 索引建置期間主要資料表上的寫入活動

如果您要將全域次要索引新增至極大的資料表，建立程序可能需要很久才能完成。若要監控進度並判斷索引是否具有足夠的寫入容量，請參閱下列 Amazon 指 CloudWatch 標：

- `OnlineIndexPercentageProgress`
- `OnlineIndexConsumedWriteCapacity`
- `OnlineIndexThrottleEvents`

### Note

如需 DynamoDB 相關 CloudWatch 度量的詳細資訊，請參閱。[DynamoDB 指標](#)

如果索引上的佈建寫入輸送量設定太低，索引建置就需要較久才能完成。若要縮短新全域次要索引的建置時間，您可以暫時增加其佈建寫入容量。

### Note

通常建議您將索引的佈建寫入容量設定為資料表寫入容量的 1.5 倍。這是適合許多使用案例的良好設定。但您的實際需求可能或高或低。

回填索引時，DynamoDB 會使用內部系統容量從資料表讀取。這是要將索引建立的影響降到最小，並確保資料表的讀取容量不會用盡。

不過，傳入的寫入活動量可能會超過索引的佈建寫入容量。這種情況會陷入瓶頸，因為索引的寫入活動會受到調節，所以建立索引需要更久。建立索引期間，建議您監控索引的 Amazon 指 CloudWatch 標，



以判斷其使用的寫入容量是否超過其佈建容量。在瓶頸情況中，您應該增加索引上的佈建寫入容量，以免在回填階段期間調節寫入。

索引建立之後，您應該設定其佈建寫入容量，以反映出您應用程式的正常用量。

## 刪除全域次要索引

如果您不再需要全域次要索引，可以使用 `UpdateTable` 操作將其刪除。

每個 `UpdateTable` 操作只能刪除一個全域次要索引。

刪除全域次要索引時，父資料表中所有讀取和寫入活動皆不受影響。當刪除正在進行時，仍可修改其他索引上的佈建輸送量。

### Note

- 當您使用 `DeleteTable` 動作刪除資料表時，也會刪除該資料表中所有全域次要索引。
- 您的帳戶將不會因全域次要索引的刪除操作而被收取費用。

## 建立期間修改全域次要索引

當索引建置時，您可以使用 `DescribeTable` 操作來判斷它所處的階段。索引的描述包含布林值屬性 `Backfilling`，指出 DynamoDB 目前是否正在將資料表中的項目載入索引。如果 `Backfilling` 為 `true`，則資源配置階段已完成，且正在回填索引。

當回填正在進行時，您可以更新索引的佈建輸送量參數。您可能會為了加速索引建置而決定進行此動作：您可以在索引建置時，增加索引的寫入容量，之後再減少。若要修改索引的佈建輸送量設定，請使用 `UpdateTable` 操作。索引狀態會變更為 `UPDATING`，而 `Backfilling` 則會是 `true`，直到索引可供使用為止。

在回填階段，您可以刪除正在建立的索引。在這段階段，您無法新增或刪除資料表上的其他索引。

### Note

針對已在 `CreateTable` 操作中建立的索引，`Backfilling` 屬性不會出現在 `DescribeTable` 輸出中。如需更多詳細資訊，請參閱 [索引建立階段](#)。



## 偵測和修正索引鍵違規情況

在建立全域次要索引的回填階段，Amazon DynamoDB 會檢查資料表中的每個項目，判斷是否符合納入索引中的資格。某些項目可能不符合資格，因為這些項目會導致索引鍵違規情況。在這些情況下，項目會保留在資料表中，但索引沒有該項目的對應項目。

出現以下狀況時會發生索引鍵違規情況：

- 屬性值與索引鍵結構描述資料類型之間存在資料類型不相符的問題。例如，假設其中一個項目在 GameScores 資料表中有一個類型為 String 的值 TopScore。如果新增分割區索引鍵為 TopScore、類型為 Number 的全域次要索引，資料表中的項目會違反索引鍵。
- 來自資料表的屬性數值超過索引鍵屬性的最大長度。分割區索引鍵的最大長度是 2048 個位元組，而排序索引鍵的最大長度是 1024 個位元組。如果資料表中的任何對應屬性值超過這些限制，資料表中的項目將違反索引鍵。

### Note

如果為用作索引鍵的屬性設定了「字串」或「二進位」屬性值，則屬性值的長度必須大於零；否則，資料表中的項目將違反索引鍵。  
這個工具此時不會標記此索引鍵違規情況。

如果發生索引鍵違規情況，回填階段會繼續，而不會中斷。但是，索引中不會包含任何違規項目。回填階段完成後，所有違反新索引鍵結構描述的項目寫入都會遭到拒絕。

若要識別並修正資料表中違反索引鍵的屬性值，請使用 Violation Detector 工具。若要執行 Violation Detector，請建立組態檔案，指定要掃描的資料表名稱、全域次要索引分割區索引鍵及排序索引鍵的名稱和資料類型，以及發現任何索引鍵違規情況時要採取的動作。Violation Detector 可以在以下兩種不同模式之一執行：

- 偵測模式：偵測索引鍵違規情況。使用偵測模式報告資料表中會導致全域次要索引中發生索引鍵違規情況的項目。(您可以選擇請求在找到這些違規的資料表項目時立即將其刪除。) 偵測模式的輸出會寫入檔案，可供您用其進行進一步分析。
- 校正模式：校正索引鍵違規情況。在校正模式下，Violation Detector 會從偵測模式讀取與輸出檔案格式相同的輸入檔案。校正模式會從輸入檔案讀取紀錄，並且會針對每個紀錄刪除或更新資料表中的對應項目。(請注意，如果您選擇更新項目，則必須編輯輸入檔案並為這些更新設定適當的值。)

## 下載並執行 Violation Detector

Violation Detector 可以作為可執行的 Java 封存檔 (.jar 檔案)，並在 Windows、macOS 或 Linux 電腦上執行。Violation Detector 需要 Java 1.7 (或更新版本) 和 Apache Maven。

- [從 GitHub 下載 Violation Detector](#)

請遵循 README.md 檔案中的說明使用 Maven 下載並安裝 Violation Detector。

若要啟動 Violation Detector，請移至您已建立 ViolationDetector.java 的目錄，然後輸入下列命令。

```
java -jar ViolationDetector.jar [options]
```

Violation Detector 命令列接受下列選項：

- -h | --help：列印 Violation Detector 的使用摘要和選項。
- -p | --configFilePath value：Violation Detector 組態檔案的完全合格名稱。如需詳細資訊，請參閱 [Violation Detector 組態檔案](#)。
- -t | --detect value：偵測資料表中的索引鍵違規情況，並將其寫入 Violation Detector 輸出檔案。如果此參數的值設定為 keep，則不會修改具有索引鍵違規情況的項目。如果值設定為 delete，則會從資料表中刪除具有索引鍵違規情況的項目。
- -c | --correct value：從輸入檔案讀取索引鍵違規情況，並對資料表中的項目採取更正動作。如果此參數的值設定為 update，則會使用新的非違規值來更新具有索引鍵違規情況的項目。如果值設定為 delete，則會從資料表中刪除具有索引鍵違規情況的項目。

## Violation Detector 組態檔案

在執行時間，Violation Detector 工具需要組態檔案。此檔案中的參數決定 Violation Detector 可以存取哪些 DynamoDB 資源，以及其可以消耗多少佈建的輸送量。下表描述了這些參數。

| 參數名稱               | 描述                                     | 是否為必要？ |
|--------------------|----------------------------------------|--------|
| awsCredentialsFile | 包含您 AWS 登入資料的檔案的完全合格名稱。登入資料檔案必須採用下列格式： | 是      |

| 參數名稱            | 描述                                                                                            | 是否為必要？ |
|-----------------|-----------------------------------------------------------------------------------------------|--------|
|                 | <pre>accessKey = <i>access_key_id_goes_here</i> secretKey = <i>secret_key_goes_here</i></pre> |        |
| dynamoDBRegion  | 資料表所在的 AWS 區域。例如：us-west-2。                                                                   | 是      |
| tableName       | 所要掃描 DynamoDB 資料表的名稱。                                                                         | 是      |
| gsiHashKeyName  | 索引分割區索引鍵的名稱。                                                                                  | 是      |
| gsiHashKeyType  | 索引分割區索引鍵的資料類型：String、Number 或 Binary：<br><br>S   N   B                                        | 是      |
| gsiRangeKeyName | 索引排序索引鍵的名稱。如果索引只有簡易的主索引鍵 (分割區索引鍵)，請勿指定此參數。                                                    | 否      |
| gsiRangeKeyType | 索引排序索引鍵的資料類型：String、Number 或 Binary：<br><br>S   N   B<br><br>如果索引只有簡易的主索引鍵 (分割區索引鍵)，請勿指定此參數。  | 否      |

| 參數名稱                            | 描述                                                                                                                                                                                                                                                                                                                               | 是否為必要？ |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| recordDetails                   | 是否將索引鍵違規情況的完整詳細資訊寫入輸出檔案。如果設定為 true (預設值)，則會報告違規項目的完整資訊。如果設定為 false，則只會報告違規情況的數量。                                                                                                                                                                                                                                                 | 否      |
| recordGsiValueInViolationRecord | 是否將違反索引鍵的數值寫入輸出檔案。如果設定為 true (預設值)，則會報告索引鍵值。如果設定為 false，則不會報告索引鍵值。                                                                                                                                                                                                                                                               | 否      |
| detectionOutputPath             | <p>Violation Detector 輸出檔案的完整路徑。此參數支援寫入本機目錄或 Amazon Simple Storage Service (Amazon S3)。範例如下：</p> <pre>detectionOutputPath = //local/path/filename.csv</pre> <pre>detectionOutputPath = s3://bucket/filename.csv</pre> <p>輸出檔案中的資訊會以逗號分隔值 (CSV) 格式顯示。如果您未設定 detectionOutputPath，則輸出檔案會命名為 violation_detection.csv 並寫入目前的工作目錄。</p> | 否      |

| 參數名稱            | 描述                                                                                                                                                                                                                                                                                             | 是否為必要？ |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| numOfSegments   | <p>當 Violation Detector 掃描資料表時，要使用的平行掃描區段數目。預設值為 1，表示會以連續方式掃描資料表。如果數值為 2 或更高，則 Violation Detector 會將資料表分割為許多邏輯區段和相同數量的掃描執行緒。</p> <p>numOfSegments 的最高設定為 4096。</p> <p>對於較大的資料表，平行掃描通常比循序掃描更快。此外，如果資料表大小足以跨越多個分割區，則平行掃描會將讀取活動平均分配到多個分割區。如需在 DynamoDB 中執行平行掃描的詳細資訊，請參閱 <a href="#">平行掃描</a>。</p> | 否      |
| numOfViolations | <p>索引鍵違規情況寫入輸出檔案的上限。如果設定為 -1 (預設值)，則會掃描整個資料表。如果設定為正整數，則 Violation Detector 會在偵測到該數量的違規情況後停止。</p>                                                                                                                                                                                               | 否      |
| numOfRecords    | <p>要掃描的資料表中的項目數。如果設定為 -1 (預設值)，則會掃描整個資料表。如果設定為正整數，則 Violation Detector 會在掃描到資料表中該數量的項目後停止。</p>                                                                                                                                                                                                 | 否      |

| 參數名稱                              | 描述                                                                                                                                                                                                                        | 是否為必要？ |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| <code>readWriteIOPSPercent</code> | 規範資料表掃描期間耗用的已佈建讀取容量單位百分比。有效值範圍從 1 到 100。預設值 (25) 表示 Violation Detector 將消耗不超過資料表佈建的讀取輸送量的 25%。                                                                                                                            | 否      |
| <code>correctionInputPath</code>  | <p>Violation Detector 修正輸入檔案的完整路徑。如果您在校正模式下執行 Violation Detector，則會使用此檔案的內容來修改或刪除資料表中違反全域次要索引的資料項目。</p> <p><code>correctionInputPath</code> 檔案的格式與 <code>detectionOutputPath</code> 檔案的格式相同。這可讓您在校正模式下將偵測模式的輸出作為輸入處理。</p> | 否      |

| 參數名稱                 | 描述                                                                                                                                                                                                                                                                                                                              | 是否為必要？ |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| correctionOutputPath | <p>Violation Detector 修正輸出檔案的完整路徑。只有在發生更新錯誤時，才會建立此檔案。</p> <p>此參數支援寫入本機目錄或 Amazon S3。範例如下：</p> <pre>correctionOutputPath = //local/path/ filename.csv</pre> <pre>correctionOutputPath = s3://bucket/filename. csv</pre> <p>輸出檔案中的資訊會以 CSV 格式顯示。如果您未設定 correctionOutputPath，則輸出檔案會命名為 violation_update_errors.csv 並寫入目前的工作目錄。</p> | 否      |

## 偵測

若要偵測索引鍵違規情況，請將 Violation Detector 與 `--detect` 命令列選項搭配使用。若要顯示此選項的運作方式，請考慮顯示於 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 的 ProductCatalog 資料表。以下是資料表中的項目清單。僅顯示主索引鍵 (Id) 以及 Price 屬性。

| ID (主索引鍵) | 價格 |
|-----------|----|
| 101       | 5  |
| 102       | 20 |

| ID (主索引鍵) | 價格  |
|-----------|-----|
| 103       | 200 |
| 201       | 100 |
| 202       | 200 |
| 203       | 300 |
| 204       | 400 |
| 205       | 500 |

所有 Price 的數值均為類型 Number。不過，因為 DynamoDB 不具結構描述，所以可以新增具有非數字 Price 的項目。例如，假設您將另一個項目新增至 ProductCatalog 資料表。

| ID (主索引鍵) | 價格      |
|-----------|---------|
| 999       | "Hello" |

資料表現在共有九個項目。

現在，您將新的全域次要索引新增至資料表 PriceIndex。此索引的主索引鍵是分割區索引鍵 Price，其類型是 Number。建置索引後，將會包含八個項目，但 ProductCatalog 資料表含有九個項目。造成此差異的原因是數值 "Hello" 的類型為 String，但 PriceIndex 的主索引鍵為類型 Number。此 String 值違反全域次要索引鍵，因此不會出現在索引中。

若要在此情況中使用 Violation Detector，請先建立如下的組態檔案。

```
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
```

接著，您可以執行 Violation Detector，如下列範例所示。

```
$ 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
```

如果 `recordDetails` 組態參數設為 `true`，Violation Detector 會將每個違規情況的詳細資訊寫入輸出檔案，如下列範例所示。

```
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,,
```

輸出檔案採用 CSV 格式。檔案中的第一行是標頭，後接每個違反索引鍵的項目的一筆紀錄。這些違規情況紀錄的欄位如下所示：

- 資料表雜湊索引鍵 - 資料表中項目的分割區索引鍵值。
- 資料表範圍索引鍵 - 資料表中項目的排序索引鍵值。
- GSI 雜湊索引鍵值 - 全域次要索引的分割區索引鍵值。
- GSI 雜湊索引鍵違規情況類型 - Type Violation 或 Size Violation。
- GSI 雜湊索引鍵違規情況說明 - 違反的原因。
- GSI 雜湊索引鍵更新值 (針對使用者) - 在校正模式下，屬性的新使用者提供的數值。
- GSI 範圍索引鍵值 - 全域次要索引的排序索引鍵值。
- GSI 範圍索引鍵違規情況類型 - Type Violation 或 Size Violation。

- GSI 範圍索引鍵違規情況說明 - 違反的原因。
- GSI 範圍索引鍵更新值 (針對使用者) - 在校正模式下，屬性的新使用者提供的數值。
- 更新時刪除空白屬性 (是/否) - 在校正模式下，決定要刪除 (是) 或保留 (否) 資料表中的違規情況，但只有在下列其中一個欄位為空白時：
  - GSI Hash Key Update Value(FOR USER)
  - GSI Range Key Update Value(FOR USER)

如果這些欄位中的任何一個並非空白，則 Delete Blank Attribute When Updating(Y/N) 不起作用。

#### Note

輸出格式可能會因組態檔案和命令列選項而有所不同。例如，如果資料表有簡單的主索引鍵 (沒有排序索引鍵)，則輸出中不會出現任何排序索引鍵欄位。檔案中的違規情況紀錄可能不會以排序順序顯示。

## 更正

若要校正索引鍵違規情況，請將 Violation Detector 與 `--correct` 命令列選項搭配使用。在校正模式下，Violation Detector 會讀取 `correctionInputPath` 參數指定的輸入檔案。此檔案具有與 `detectionOutputPath` 檔案相同的格式，以便您可以使用來自偵測的輸出作為校正的輸入。

Violation Detector 提供兩種不同的方法來校正索引鍵違規情況：

- 刪除違規情況：刪除具有違反屬性值的資料表項目。
- 更新違規情況：更新資料表項目，以非違規值取代違規屬性。

在任何一種情況下，您都可以使用偵測模式的輸出檔案作為校正模式的輸入。

繼續進行 ProductCatalog 範例，假設您想要從資料表刪除違規情況。若要執行此操作，請使用以下命令列。

```
$ java -jar ViolationDetector.jar --configFilePath config.txt --correct delete
```

此時，系統會要求您確認是否要刪除違規項目。

```
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
```

現在 ProductCatalog 和 PriceIndex 都具有相同數量的項目。

## 使用全域次要索引：Java

您可以使用 AWS SDK for Java Document API 建立含有一或多個全域次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，以及使用索引執行查詢。

下列是資料表操作的常用步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 透過建立對應的請求物件，為操作提供必要及選用的參數。
3. 呼叫您在前一步驟中建立之用戶端所提供的適當方法。

### 主題

- [建立具有全域次要索引的資料表](#)
- [使用全域次要索引描述資料表](#)
- [查詢全域次要索引](#)
- [範例：使用 AWS SDK for Java Document API 的全域次要索引](#)

### 建立具有全域次要索引的資料表

您可在建立資料表的同時建立全域次要索引。若要執行這項操作，請使用 CreateTable，並提供一或多個全域次要索引的規格。以下 Java 程式碼範例會建立資料表來保存天氣資料的相關資訊。分割區索引鍵為 Location，而排序索引鍵為 Date。名為 PrecipIndex 的全域次要索引允許快速存取各個地點的降水資料。

以下是使用 DynamoDB Document API 建立具有全域次要索引的資料表的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 CreateTableRequest 類別的執行個體，以提供請求資訊。

您必須提供資料表名稱、其主索引鍵，以及佈建的輸送量數值。對於全域次要索引，您必須提供索引名稱、其佈建的輸送量設定值、索引排序索引鍵的屬性定義、索引的索引鍵結構描述以及屬性投影。

3. 以參數形式提供請求物件，以便呼叫 `createTable` 方法。

下列 Java 程式碼範例示範上述步驟。程式碼會建立具有全域次要索引 (PrecipIndex) 的資料表 (WeatherData)。索引分割區索引鍵是 `Date`，而其排序索引鍵是 `Precipitation`。所有的資料表屬性都會投影到索引。使用者可以查詢此索引以取得特定日期的天氣資料，可選擇依降水量排序資料。

因為 `Precipitation` 不是資料表的索引鍵屬性，所以其並非必要項目。不過，沒有 `Precipitation` 的 `WeatherData` 項目不會在 `PrecipIndex` 中顯示。

```
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());
```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。

### 使用全域次要索引描述資料表

若要取得資料表上全域次要索引的資訊，請使用 DescribeTable。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

以下是存取資料表的全域次要索引資訊的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體，代表您要進行作業的索引。
3. 在 Table 物件上呼叫 describe 方法。

下列 Java 程式碼範例示範上述步驟。

### 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());
 }
}
}
```

### 查詢全域次要索引

您可以在全域次要索引上使用 Query，與 Query 資料表的方式相同。您需要指定索引名稱、索引分割區索引鍵和排序索引鍵的查詢條件 (如存在)，以及要傳回的屬性。在本例中，索引為 PrecipIndex，其分割區索引鍵為 Date，排序索引鍵為 Precipitation。索引查詢會傳回特定日期的所有天氣資料，其中降水量大於零。

以下是使用 AWS SDK for Java Document API 查詢全域次要索引的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體，代表您要進行作業的索引。

3. 針對您想要查詢的索引建立 Index 類別的執行個體。
4. 在 Index 物件上呼叫 query 方法。

屬性名稱 Date 是 DynamoDB 保留字。因此，您必須使用表達式屬性名稱作為 KeyConditionExpression 中的預留位置。

下列 Java 程式碼範例示範上述步驟。

### 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());
}
```

### 範例：使用 AWS SDK for Java Document API 的全域次要索引

下列 Java 程式碼範例示範如何使用全域次要索引。此範例會建立名為 Issues 的資料表，可用於軟體開發的簡單錯誤追蹤系統。分割區索引鍵為 IssueId，而排序索引鍵為 Title。此資料表上有三個全域次要索引：

- CreateDateIndex：分割區索引鍵為 CreateDate，而排序索引鍵為 IssueId。除了資料表索引鍵之外，屬性 Description 和 Status 都會投影到索引中。
- TitleIndex：分割區索引鍵為 Title，而排序索引鍵為 IssueId。除了資料表索引鍵之外，其他屬性均不會投影到索引中。

- `DueDateIndex` : 分割區索引鍵為 `DueDate` , 沒有排序索引鍵。所有的資料表屬性都會投影到索引。

建立 `Issues` 資料表後，程式會載入所含資料表示軟體錯誤報告的資料表。然後，便會使用全域次要索引查詢資料。最後，程式會刪除 `Issues` 資料表。

如需測試下列範例的 step-by-step 指示，請參閱[Java 程式碼範例](#)。

## 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);
 }
}
```

## 使用全域次要索引：.NET

您可以使用 AWS SDK for .NET 低階 API 建立含有一或多個全域次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，以及使用索引執行查詢。這些操作會映射至對應的 DynamoDB 操作。如需詳細資訊，請參閱《[Amazon DynamoDB API 參考](#)》。

下列是使用 .NET 低階 API 執行資料表操作的一般步驟。

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。
2. 透過建立對應的請求物件，為操作提供必要及選用的參數。

例如，建立 `CreateTableRequest` 物件來建立資料表，以及建立 `QueryRequest` 物件來查詢資料表或索引。

3. 執行您在前一步驟中建立之用戶端所提供的適當方法。

### 主題

- [建立具有全域次要索引的資料表](#)
- [使用全域次要索引描述資料表](#)
- [查詢全域次要索引](#)
- [範例：使用 AWS SDK for .NET 低階 API 的全域次要索引](#)

### 建立具有全域次要索引的資料表

您可在建立資料表的同時建立全域次要索引。若要執行這項操作，請使用 `CreateTable`，並提供一或多個全域次要索引的規格。以下 C# 程式碼範例會建立資料表來保存天氣資料的相關資訊。分割區索引鍵為 `Location`，而排序索引鍵為 `Date`。名為 `PrecipIndex` 的全域次要索引允許快速存取各個地點的降水資料。

以下是使用 .NET 低階 API 建立具有全域次要索引之資料表的步驟。

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。

## 2. 建立 CreateTableRequest 類別的執行個體，以提供請求資訊。

您必須提供資料表名稱、其主索引鍵，以及佈建的輸送量數值。對於全域次要索引，您必須提供索引名稱、其佈建的輸送量設定值、索引排序索引鍵的屬性定義、索引的索引鍵結構描述以及屬性投影。

## 3. 以參數形式提供請求物件，以便執行 CreateTable 方法。

下列 C# 程式碼範例示範前述步驟。程式碼會建立具有全域次要索引 (PrecipIndex) 的資料表 (WeatherData)。索引分割區索引鍵是 Date，而其排序索引鍵是 Precipitation。所有的資料表屬性都會投影到索引。使用者可以查詢此索引以取得特定日期的天氣資料，可選擇依降水量排序資料。

因為 Precipitation 不是資料表的索引鍵屬性，所以其並非必要項目。不過，沒有 Precipitation 的 WeatherData 項目不會在 PrecipIndex 中顯示。

```
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);
```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。



## 使用全域次要索引描述資料表

若要取得資料表上全域次要索引的資訊，請使用 `DescribeTable`。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

以下是使用 .NET 低階 API 存取資料表的全域次要索引資訊的步驟。

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。
2. 以參數形式提供請求物件，以便執行 `describeTable` 方法。

建立 `DescribeTableRequest` 類別的執行個體，以提供請求資訊。您必須提供資料表名稱。

3.

下列 C# 程式碼範例示範前述步驟。

### 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);
 }
}
```

```
}
}
```

## 查詢全域次要索引

您可以在全域次要索引上使用 Query，與 Query 資料表的方式相同。您需要指定索引名稱、索引分割區索引鍵和排序索引鍵的查詢條件 (如存在)，以及要傳回的屬性。在本例中，索引為 PrecipIndex，其分割區索引鍵為 Date，排序索引鍵為 Precipitation。索引查詢會傳回特定日期的所有天氣資料，其中降水量大於零。

以下是使用 .NET 低階 API 查詢全域次要索引的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 建立 QueryRequest 類別的執行個體，以提供請求資訊。
3. 以參數形式提供請求物件，以便執行 query 方法。

屬性名稱 Date 是 DynamoDB 保留字。因此，您必須使用表達式屬性名稱作為 KeyConditionExpression 中的預留位置。

下列 C# 程式碼範例示範前述步驟。

### 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();
}
```

### 範例：使用 AWS SDK for .NET 低階 API 的全域次要索引

下列 C# 程式碼範例示範如何使用全域次要索引。此範例會建立名為 Issues 的資料表，可用於軟體開發的簡單錯誤追蹤系統。分割區索引鍵為 IssueId，而排序索引鍵為 Title。此資料表上有三個全域次要索引：

- **CreateDateIndex**：分割區索引鍵為 CreateDate，而排序索引鍵為 IssueId。除了資料表索引鍵之外，屬性 Description 和 Status 都會投影到索引中。
- **TitleIndex**：分割區索引鍵為 Title，而排序索引鍵為 IssueId。除了資料表索引鍵之外，其他屬性均不會投影到索引中。
- **DueDateIndex**：分割區索引鍵為 DueDate，沒有排序索引鍵。所有的資料表屬性都會投影到索引。

建立 Issues 資料表後，程式會載入所含資料表示軟體錯誤報告的資料表。然後，便會使用全域次要索引查詢資料。最後，程式會刪除 Issues 資料表。

如需測試下列範例的 step-by-step 指示，請參閱[.NET 程式碼範例](#)。

### 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;
 }
 }
}
```

```
 }
 }
}
```

## 使用全域次要索引：AWS CLI

您可以使用 AWS CLI 建立含有一或多個全域次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，以及使用索引執行查詢。

### 主題

- [建立具有全域次要索引的資料表](#)
- [將全域次要索引新增至現有資料表](#)
- [使用全域次要索引描述資料表](#)
- [查詢全域次要索引](#)

### 建立具有全域次要索引的資料表

您可在建立資料表的同時建立全域次要索引。若要執行這項操作，請使用 `create-table` 參數，並提供一或多個全域次要索引的規格。以下範例會建立名為 `GameScores` 的資料表，以及名為 `GameTitleIndex` 的全域次要索引。基礎資料表具有分割區索引鍵 `UserId` 和排序索引鍵 `GameTitle`，可讓您有效率地找到個別使用者在特定遊戲中的最佳分數，而 GSI 具有分割區索引鍵 `GameTitle` 和排序索引鍵 `TopScore`，可讓您快速找到特定遊戲的整體最高分數。

```
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
 }
 }
] "

```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。您可以使用 [describe-table](#) 來判斷資料表建立的狀態。

### 將全域次要索引新增至現有資料表

全域次要索引也可以在建立資料表之後新增或修改。若要執行這項操作，請使用 `update-table` 參數，並提供一或多個全域次要索引的規格。下列範例使用與上一個範例相同的結構描述，但假設資料表已經建立，我們稍後會新增 GSI。

```

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\ "]
 }
 }
] "

```

## 使用全域次要索引描述資料表

若要取得資料表上全域次要索引的資訊，請使用 `describe-table` 參數。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

```
aws dynamodb describe-table --table-name GameScores
```

## 查詢全域次要索引

您可以依照與 `query` 資料表大致相同的方式在全域次要索引上使用 `query` 操作。您必須指定索引名稱、索引排序索引鍵的查詢準則，以及您要傳回的屬性。在本例中，索引是 `GameTitleIndex`，而索引排序索引鍵為 `GameTitle`。

傳回的唯一屬性是已投影到索引的屬性。您也可以修改此查詢來選擇非索引鍵屬性，但這需要相對昂貴的資料表擷取活動。如需資料表擷取的詳細資訊，請參閱 [屬性投影](#)。

```
aws dynamodb query --table-name GameScores\
 --index-name GameTitleIndex \
 --key-condition-expression "GameTitle = :v_game" \
 --expression-attribute-values '{":v_game":{"S":"Alien Adventure"}}'
```

## 本機次要索引

某些應用程式只需要使用基礎資料表的主索引鍵查詢資料。但是，在某些情況下，替代排序索引鍵會有所幫助。為了給應用程式提供排序索引鍵選擇，您可以在 Amazon DynamoDB 資料表上建立一或多個本機次要索引，並針對這些索引發出 `Query` 或 `Scan` 請求。

### 主題

- [案例：使用本機次要索引](#)
- [屬性投影](#)
- [建立本機次要索引](#)
- [從本機次要索引讀取資料](#)
- [項目寫入和本機次要索引](#)
- [本機次要索引的佈建輸送量考量](#)
- [本機次要索引的儲存考量](#)
- [本機次要索引中的項目集合](#)

- [使用本機次要索引：Java](#)
- [使用本機次要索引：.NET](#)
- [使用本機次要索引：AWS CLI](#)

## 案例：使用本機次要索引

例如，考量 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 中定義的 Thread 資料表。此資料表適用於應用程式，例如 [AWS 開發論壇](#)。下圖顯示這些項目在資料表中的組織方式。(並未顯示所有屬性。)

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 會連續儲存具有相同分割區索引鍵值的所有項目。在本例中，給定一個特定的 ForumName，Query 操作就可以立即找到該論壇的所有主題。在具有相同分割區索引鍵值的項目群組中，項目會依排序索引鍵值排序。如果查詢中還提供了排序索引鍵 (Subject)，DynamoDB 可以縮小傳回的結果範圍，例如傳回「S3」論壇中以字母「a」開頭的 Subject 的所有主題。

某些請求可能需要更複雜的資料存取模式。例如：

- 哪些論壇主題獲得最多的觀看次數和回覆？
- 特定論壇中哪個主題的訊息數量最多？
- 在特定時段內，有多少個主題發佈在特定論壇？

若要回答這些問題，Query 動作並不足夠。您還必須 Scan 整個資料表。對於包含數百萬個項目的資料表，這會消耗大量佈建的讀取輸送量，而且需要很長的時間才能完成。



不過，您可以在非索引鍵屬性上指定一或多個本機次要索引，例如 Replies 或 LastPostDateTime。

本機次要索引會針對指定分割區索引鍵值維護替代排序索引鍵。本機次要索引也包含基礎資料表中部分或全部屬性的複本。您可以指定在建立資料表時要投影到本機次要索引中的屬性。本機次要索引中的資料由與基礎資料表相同的分割區索引鍵組織，但具有不同的排序索引鍵。這可讓您在此不同維度之間有效地存取資料項目。為獲得更大的查詢或掃描靈活性，您最多可以為每個資料表建立五個本機次要索引。

假設應用程式需要尋找在最近三個月內張貼於特定論壇中的所有主題。如果沒有本機次要索引，應用程式必須 Scan 整個 Thread 資料表，並捨棄任何不在指定時間範圍內的貼文。使用本機次要索引，Query 操作可以使用 LastPostDateTime 作為排序索引鍵，並快速找到資料。

下圖顯示名為 LastPostIndex 的本機次要索引。請注意，分割區索引鍵與 Thread 資料表的分割區索引鍵相同，但排序索引鍵是 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"          |
| ...              | ...                     | ...            |

每個本機次要索引均須符合下列條件：

- 分割區索引鍵與其基礎資料表的分割區索引鍵相同。

- 排序索引鍵只包含一個純量屬性。
- 基礎資料表的排序索引鍵會投影到索引中，在索引中充當非索引鍵屬性。

在本例中，分割區索引鍵是 ForumName，而本機次要索引的排序索引鍵是 LastPostDateTime。此外，基礎資料表中的排序索引鍵值 (在本例中為 Subject) 會投影到索引中，但不是索引鍵的一部分。如果應用程式需要以 ForumName 和 LastPostDateTime 為基礎的清單，則可針對 LastPostIndex 發出 Query 請求。查詢結果會依 LastPostDateTime 排序，並且可以依升序或降序排列傳回。查詢也可以套用索引鍵條件，例如僅傳回在特定的時間範圍內具有 LastPostDateTime 的項目。

每個本機次要索引會自動包含來自其基礎資料表的分割區索引鍵和排序索引鍵；您可以選擇將非索引鍵屬性投影到索引中。當您查詢索引時，DynamoDB 可有效率地擷取這些投影屬性。在查詢本機次要索引時，查詢還可以檢索未投影到索引中的屬性。DynamoDB 會自動從基礎資料表擷取這些屬性，但延遲較大，且佈建的輸送量成本也較高。

對於任何本機次要索引，每個不同的分割區索引鍵值最多可存放 10 GB 的資料。此圖包含基礎資料表中的所有項目，以及索引中具有相同分割區索引鍵值的所有項目。如需詳細資訊，請參閱 [本機次要索引中的項目集合](#)。

## 屬性投影

應用程式可以借助 LastPostIndex 將 ForumName 和 LastPostDateTime 用作查詢條件。不過，若要擷取任何其他屬性，DynamoDB 必須對 Thread 資料表執行額外的讀取操作。這些額外讀取稱為擷取，可以增加查詢所需的佈建輸送量總量。

假設您想要填入一個網頁，其中包含「S3」中所有主題的清單，以及每個主題的回覆次數 (依上次回覆日期/時間，從最近回覆開始排序)。若要填入此清單，您需要下列屬性：

- Subject
- Replies
- LastPostDateTime

查詢這些資料並避免擷取操作的最有效方法是將 Replies 屬性從資料表投影至本機次要索引，如此圖所示。

## 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      |
| ...       | ...                   | ...     | ...     |

投影是指從資料表複製到次要索引的屬性集合。資料表的分割區索引鍵和排序索引鍵一律會投影到索引中；您可以投影其他屬性來支援應用程式的查詢需求。查詢索引時，Amazon DynamoDB 可以存取投影中的任何屬性，就好像這些屬性在它們自己的資料表中一樣。

在建立次要索引時，您需要指定要投影到索引中的屬性。DynamoDB 為此提供三種不同的選項：

- KEYS\_ONLY：索引中的每個項目都只包含資料表分割索引鍵和排序索引鍵值，以及索引鍵值。KEYS\_ONLY 選項會產生最小的可行次要索引。
- INCLUDE：除了 KEYS\_ONLY 中描述的屬性外，次要索引還包含您指定的其他非索引鍵屬性。
- ALL：次要索引包含來源資料表中的所有屬性。因為索引中的所有資料表資料都會重複，所以 ALL 投影會產生最大的可行次要索引。

在上圖中，非索引鍵屬性 Replies 投影到 LastPostIndex。應用程式可以查詢 LastPostIndex，而不是完整的 Thread 資料表，以便使用 Subject、Replies 和 LastPostDateTime 來填充網頁。如果請求任何其他非索引鍵屬性，DynamoDB 需要從 Thread 資料表擷取這些屬性。

從應用程式的角度來看，從基礎資料表擷取其他屬性是自動且透明的程序，因此不需要重寫任何應用程式邏輯。不過，此類擷取會大幅降低使用本機次要索引的效能優勢。

在選擇要投影到本機次要索引的屬性時，您必須權衡佈建輸送量成本和儲存成本：

- 若您只需要存取少量的屬性，並且希望盡可能的降低延遲，建議您考慮只將那些屬性投影到本機次要索引。索引愈小，存放成本就愈低，您的寫入成本也愈低。如果您偶爾需要擷取屬性，佈建輸送量的成本可能會超過儲存這些屬性的長期成本。
- 如果應用程式會頻繁存取某些非索引鍵屬性，您應該考慮將這些屬性投影到本機次要索引。本機次要索引的額外儲存成本會抵銷頻繁執行資料表掃描的成本。
- 如果需要頻繁存取大多數的非索引鍵屬性，您可以將這些屬性 (或整個基礎資料表) 投影到本機次要索引。這可為您提供最大的靈活性和最低的佈建輸送量耗用量，因為不需要擷取。但是，如果投影所有屬性，您的儲存成本將會增加，甚至翻倍。
- 如果您的應用程式不需頻繁查詢資料表，但必須對資料表中的資料執行許多寫入或更新，請考慮投影 KEYS\_ONLY。本機次要索引的大小將會最小，但仍能在需要的時候進行查詢活動。

## 建立本機次要索引

若要在資料表上建立具有一或多個本機次要索引，請使用 CreateTable 操作的 LocalSecondaryIndexes 參數。建立資料表時，會建立資料表上的本機次要索引。當您使用刪除資料表時，也會刪除該資料表上的任何本機次要索引。

您必須指定一個非索引鍵屬性作為本機次要索引的排序索引鍵。您選擇的屬性必須是純量 String、Number 或 Binary。不允許其他純量類型、文件類型和集合類型。如需資料類型的完整清單，請參閱 [資料類型](#)。

### Important

對於具有本機次要索引的資料表，每個分割區索引鍵值都有 10 GB 的大小限制。具有本機次要索引的資料表可以存放任何數量的項目，只要任何一個分割區索引鍵值的總大小不超過 10 GB 即可。如需詳細資訊，請參閱 [項目集合大小限制](#)。

您可以將任何資料類型的屬性投影到本機次要索引。這包括純量、文件和集合。如需資料類型的完整清單，請參閱 [資料類型](#)。

## 從本機次要索引讀取資料

您可以使用 Query 和 Scan 操作從本機次要索引擷取項目。GetItem 和 BatchGetItem 操作不能用於本機次要索引。

### 查詢本機次要索引

在 DynamoDB 資料表中，每個項目的組合分割區索引鍵值和排序索引鍵值必須是唯一的值。不過，在本機次要索引中，排序索引鍵值對於指定的分割區索引鍵值不一定是唯一的值。如果本機次要索引中有許多項目具有相同的排序索引鍵值，Query 操作會傳回具有相同分割區索引鍵值的所有項目。在回應中，符合的項目不會以任何特定的順序傳回。

您可以使用最終一致或強烈一致讀取來查詢本機次要索引。若要指定您想要的一致性類型，請使用 Query 操作的 ConsistentRead 參數。來自本機次要索引的強烈一致讀取一律會傳回最新的更新值。如果查詢需要從基礎資料表中取得其他屬性，這些屬性將與索引保持一致。

### Example

請考慮從 Query (請求來自特定論壇主題的資料) 傳回的下列資料。

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

在此查詢中：

- DynamoDB 存取 LastPostIndex，使用 ForumName 分割區索引鍵尋找「EC2」的索引項目。所有具有此索引鍵的索引項目都會相鄰存放，以利快速擷取。
- 在此論壇中，DynamoDB 使用索引來尋找符合指定 LastPostDateTime 條件的索引鍵。
- 由於 Replies 屬性投影到索引中時，DynamoDB 可以檢索此屬性，而不會耗用任何額外的佈建輸送量。

- 此 Tags 屬性不會投影到索引中，因此 DynamoDB 必須存取 Thread 資料表並擷取此屬性。
- 結果會傳回，並依 LastPostDateTime 排序。索引項目依分割區索引鍵值排序，再依排序索引鍵值排序，然後由 Query 依儲存的順序將其傳回。(您可以使用 ScanIndexForward 參數，以遞減順序傳回結果。)

由於 Tags 屬性未投影到本機次要索引中，DynamoDB 必須消耗額外的讀取容量單位，才能從基礎資料表擷取此屬性。如果需要經常執行此查詢，您應將 Tags 投影至 LastPostIndex，避免從基礎資料表擷取。不過，如果您僅偶爾需要存取 Tags，則將 Tags 投影至索引中的額外儲存成本便可能不值得。

### 掃描本機次要索引

您可以使用 Scan 從本機次要索引檢索所有資料。您必須在請求中提供基礎資料表名稱及索引名稱。使用 Scan，DynamoDB 會讀取索引中的所有資料，並將其傳回應用程式。您也可以請求只傳回一部分的資料，並捨棄其他剩餘的資料。若要執行此作業，請使用 Scan API 的 FilterExpression 參數。如需詳細資訊，請參閱 [掃描的篩選條件表達式](#)。

### 項目寫入和本機次要索引

DynamoDB 會自動讓所有本機次要索引與各自的基本資料表同步。應用程式永遠不會直接寫入索引。然而，了解 DynamoDB 維持這些索引之方式的含義很重要。

在建立本機次要索引時，您可以指定一個屬性作為索引的排序索引鍵。您也可以為該屬性指定資料類型。這表示每次您將項目寫入基礎資料表時，如果項目定義了索引鍵屬性，則其類型必須與索引鍵結構描述的資料類型相符。在 LastPostIndex 案例中，索引的 LastPostDateTime 排序索引鍵是定義為 String 資料類型。如果您嘗試在 Thread 資料表中新增項目，並為 LastPostDateTime (如 Number) 指定不同的資料類型，DynamoDB 會因資料類型不符而傳回 ValidationException。

基底資料表中的項目與本機次要索引中的項目之間不需要 one-to-one 建立關係。事實上，這種行為對許多應用程式來說可能是有利的。

相較於索引較少的資料表，本機次要索引較多的資料表會有較高的寫入活動成本。如需詳細資訊，請參閱 [本機次要索引的佈建輸送量考量](#)。

#### Important

對於具有本機次要索引的資料表，每個分割區索引鍵值都有 10 GB 的大小限制。具有本機次要索引的資料表可以存放任何數量的項目，只要任何一個分割區索引鍵值的總大小不超過 10 GB 即可。如需詳細資訊，請參閱 [項目集合大小限制](#)。



## 本機次要索引的佈建輸送量考量

在 DynamoDB 中建立資料表時，您可以為資料表的預期工作負載佈建讀取和寫入容量單位。該工作負載包括在資料表本機次要索引上的讀取和寫入活動。

若要檢視佈建的輸送容量的目前費率，請參閱 [Amazon DynamoDB 定價](#)。

### 讀取容量單位

在查詢本機次要索引時，消耗的讀取容量單位數目取決於資料的存取方式。

與資料表查詢一樣，索引查詢可以使用最終一致性或強烈一致讀取，取決於 `ConsistentRead` 的值。一個強烈一致讀取會消耗一個讀取容量單位；最終一致讀取則只會消耗一半的讀取容量單位。因此，透過選擇最終一致讀取，您可以減少讀取容量單位的費用。

對於僅請求索引鍵和投影屬性的索引查詢，DynamoDB 會使用為資料表查詢計算佈建讀取活動相同的方式，計算佈建讀取活動。唯一的差別在於計算方式是以索引項目的大小為基礎，而非基礎資料表中項目的大小。讀取容量單位數為所有傳回項目中，所有投影屬性大小的總和；結果會四捨五入至下一個 4 KB 界限。如需 DynamoDB 如何計算佈建輸送用量的詳細資訊，請參閱 [佈建容量模式](#)。

對於讀取未投影到本機次要索引的屬性的索引查詢，除了從索引讀取投影的屬性之外，DynamoDB 還需要從基本資料表擷取這些屬性。當您在 Query 操作的 `Select` 或 `ProjectionExpression` 參數中包含任何非投影屬性，即會發生這些擷取。擷取會在查詢回應中造成額外的延遲，也會產生較高的佈建輸送量成本：除了從先前描述的本機次要索引讀取之外，您還需要針對每個擷取的基礎資料表項目支付讀取容量單位費用。此費用是用於讀取資料表中的每個項目，而不僅僅是請求的屬性。

Query 操作傳回的結果大小上限是 1 MB。這包含所有傳回項目之所有屬性名稱與值的大小。不過，如果針對本機次要索引進行查詢導致 DynamoDB 從基礎資料表擷取項目屬性，則結果中資料的大小上限可能會降低。在這種情況下，結果大小是以下各項的總和：

- 索引中相符項目的大小，且項目大小會四捨五入至下一個 4 KB。
- 基礎資料表中每個相符項目的大小，且每個項目的大小會四捨五入至下一個 4 KB。

若使用此公式，查詢操作傳回的結果的大小上限仍為 1 MB。

例如，假設有一個資料表，其中每個項目的大小均為 300 個位元組。該資料表上有本機次要索引，但每個項目只有 200 個位元組投影到索引中。現在假設您 Query 此索引，查詢將需為每個項目擷取資料表，且查詢會傳回 4 個項目。DynamoDB 會計算以下總數：

- 索引中相符項目的大小：200 個位元組 × 4 個項目 = 800 個位元組；接著會四捨五入至 4 KB。

- 基礎資料表中每個相符項目的大小： $(300 \text{ 個位元組}, \text{四捨五入至 } 4 \text{ KB}) \times 4 \text{ 個項目} = 16 \text{ KB}$ 。

因此，結果中的資料總大小為 20 KB。

### 寫入容量單位

當新增、更新或刪除資料表中的項目時，更新本機次要索引會耗用資料表的佈建寫入容量單位。寫入的總佈建輸送量成本為寫入資料表使用的寫入容量單位，加上更新本機次要索引使用的寫入容量單位。

將項目寫入本機次要索引的成本取決於幾個因素：

- 若您將項目寫入定義索引屬性的資料表，或是您將現有的項目更新為先前未定義的索引屬性，則將該項目寫入索引需要進行一次寫入操作。
- 若資料表的更新會變更索引鍵屬性的值 (從 A 到 B)，則需要兩次寫入：一次是從索引刪除先前的項目，第二次則是將新的項目寫入索引。
- 若項目存在於索引中，但寫入資料表致使索引屬性遭到刪除，則需要進行一次寫入，從索引刪除舊項目的投影。
- 若項目在更新之前或之後並不存在於索引中，則該索引將不會有任何額外的寫入成本。

所有這些因素都假設索引中每個項目的大小都小於或等於 1 KB 項目大小 (用於計算寫入容量單位)。較大的索引項目需要額外的寫入容量單位。您可以考慮查詢需要傳回的屬性，並只將那些屬性投影到索引中，從而將寫入成本降至最小。

### 本機次要索引的儲存考量

當應用程式將項目寫入資料表時，DynamoDB 會自動將正確的部分屬性複製到應顯示這些屬性的任何本機次要索引中。您的 AWS 帳戶需支付儲存基底資料表中項目的費用，以及該資料表上任何本機次要索引中的屬性儲存費用。

索引項目使用的空間數為下列項目的總和：

- 基礎資料表主索引鍵 (分割區索引鍵和排序索引鍵) 的大小 (位元組)
- 索引鍵屬性的大小 (位元組)
- 投影屬性 (若有的話) 的大小 (位元組)
- 每個索引項目 100 位元組的額外負荷

若要估算本機次要索引的儲存需求，您可以估算索引中項目的平均大小，再乘以索引中的項目數量。



如果資料表包含特定屬性未經定義的項目，但該屬性卻已定義為索引的排序索引鍵時，DynamoDB 不會將該項目的任何資料寫入索引。

## 本機次要索引中的項目集合

### Note

此節僅適用於具有本機次要索引的資料表。

在 DynamoDB 中，項目集合是資料表及其所有本機次要索引中具有相同分割區索引鍵值的任何項目群組。在本節所使用的範例中，Thread 資料表的分割區索引鍵為 ForumName，LastPostIndex 的分割區索引鍵也為 ForumName。所有具有相同 ForumName 的資料表和索引項目均屬於相同的項目集合。例如，在 Thread 資料表與 LastPostIndex 本機次要索引中，論壇 EC2 有一個項目集合，論壇 RDS 則有一個不同的項目集合。

下圖顯示論壇 S3 的項目集合。

### 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      |
| ...       | ...                   | ...     | ...     |

在此圖表中，項目集合由 Thread 和 LastPostIndex 中的所有項目組成，其中 ForumName 分割區索引鍵值為「S3」。如果資料表上有其他本機次要索引，則這些索引中 ForumName 等於「S3」的任何項目也將成為項目集合的一部分。

您可以在 DynamoDB 中使用下列任何操作來傳回項目集合的相關資訊：

- BatchWriteItem
- DeleteItem
- PutItem
- UpdateItem
- TransactWriteItems

這些操作中的每一個都支援 ReturnItemCollectionMetrics 參數。在將此參數設定為 SIZE，您可以檢視索引中每個項目集合大小的相關資訊。

### Example

以下是 Thread 資料表上 UpdateItem 操作的輸出範例，其中 ReturnItemCollectionMetrics 設定為 SIZE。更新的項目的 ForumName 值為「EC2」，因此輸出包含有關該項目集合的資訊。

```
{
 ItemCollectionMetrics: {
 ItemCollectionKey: {
 ForumName: "EC2"
 },
 SizeEstimateRangeGB: [0.0, 1.0]
 }
}
```

SizeEstimateRangeGB 物件會顯示此項目集合的大小介於 0 到 1 GB 之間。DynamoDB 會定期更新此大小估計值，因此下次修改項目時，數字可能會有所不同。

### 項目集合大小限制

包含一或多個本機次要索引之資料表，其任何項目集合的大小上限為 10GB。這不適用於沒有本機次要索引之資料表的項目集合，也不適用於全域次要索引中的項目集合。僅具有一或多個本機次要索引的資料表才會受到影響。

如果項目集合超過 10 GB 的限制，DynamoDB 會傳回

ItemCollectionSizeLimitExceededException，而且您將無法將更多項目新增至項目集合，或

增加項目集合中項目的大小。(仍允許會縮小項目集合大小的讀取和寫入操作。)您仍可將項目新增至其他項目集合。

若要減少項目集合的大小，您可以執行下列操作之一：

- 刪除有問題的分割區索引鍵值的任何不必要項目。在從基礎資料表中刪除這些項目時，DynamoDB 也會移除具有相同分割區索引鍵值的任何索引項目。
- 透過移除屬性或減少屬性大小來更新項目。如果這些屬性投影到任何本機次要索引中，DynamoDB 也會減少對應索引項目的大小。
- 使用相同的分割區索引鍵和排序索引鍵來建立新資料表，然後將項目從舊資料表移至新資料表。如果資料表具有不常存取的歷史資料，這可能是一個很好的方法。您也可以考慮將此歷史資料封存至 Amazon Simple Storage Service (Amazon S3)。

當項目集合的總大小降到 10 GB 以下時，您可以再次新增具有相同分割區索引鍵值的項目。

作為最佳實務，我們建議您檢測應用程式以監控項目集合的大小。方法之一是使用 `BatchWriteItem`、`DeleteItem`、`PutItem` 或 `UpdateItem` 時將 `ReturnItemCollectionMetrics` 參數設定為 `SIZE`。您的應用程式應該檢查輸出中的 `ReturnItemCollectionMetrics` 物件，並在項目集合超過使用者定義的限制 (例如 8 GB) 時記錄錯誤訊息。設定小於 10 GB 的限制會提供一個預警系統，讓您及時掌握項目集合正接近限制，以便採取相關措施。

## 項目集合和分割區

在包含一或多個本機次要索引的資料表中，每個項目集合會儲存在一個分割區中。此項目集合的總大小僅限於該分割區的容量：10GB。應用程式中，若資料模型包含大小無上限的項目集合，或您可能合理預期某些項目集合日後會超過 10GB，您應考慮改用全域次要索引。

您應將應用程式設計為讓資料表資料均勻分佈在不同的分割區索引鍵值之間。對於具有本機次要索引的資料表，您的應用程式不應在單一分割區上的單一項目集合內建立讀取和寫入活動的「熱點」。

## 使用本機次要索引：Java

您可以使用 AWS SDK for Java Document API 建立含有一或多個本機次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，以及使用索引執行查詢。

下列是使用 AWS SDK for Java Document API 執行資料表操作的一般步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 透過建立對應的請求物件，為操作提供必要及選用的參數。

3. 呼叫您在前一步驟中建立之用戶端所提供的適當方法。

## 主題

- [使用本機次要索引建立資料表](#)
- [使用本機次要索引描述資料表](#)
- [查詢本機次要索引](#)
- [範例：使用 Java Document API 的本機次要索引](#)

## 使用本機次要索引建立資料表

您必須同時建立資料表和本機次要索引。若要執行這項操作，請使用 `createTable` 方法，並提供一或多個本機次要索引的規格。以下 Java 程式碼範例會建立資料表來保存音樂收藏中歌曲的相關資訊。分割區索引鍵為 `Artist`，而排序索引鍵為 `SongTitle`。次要索引 `AlbumTitleIndex` 有助於依照專輯標題查詢。

以下是使用 DynamoDB Document API 建立具有本機次要索引的資料表的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 `CreateTableRequest` 類別的執行個體，以提供請求資訊。

您必須提供資料表名稱、其主索引鍵，以及佈建的輸送量數值。對於本機次要索引，您必須提供索引名稱、索引排序索引鍵的名稱和資料類型、索引的索引鍵結構描述以及屬性投影。

3. 以參數形式提供請求物件，以便呼叫 `createTable` 方法。

下列 Java 程式碼範例示範上述步驟。程式碼會建立一個資料表 (`Music`) 與次要索引 `AlbumTitle` 屬性。資料表分割區索引鍵和排序索引鍵以及索引排序索引鍵，是唯一投影到索引的屬性。

```
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());
```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。

### 使用本機次要索引描述資料表

若要取得資料表上本機次要索引的資訊，請使用 `describeTable` 方法。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

以下是使用 AWS SDK for Java Document API 存取資料表的本機次要索引資訊的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體。您必須提供資料表名稱。
3. 在 Table 物件上呼叫 `describeTable` 方法。

下列 Java 程式碼範例示範上述步驟。

### 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());
}
}
```

## 查詢本機次要索引

您可以依照與 Query 資料表大致相同的方式在本機次要索引上使用 Query 操作。您必須指定索引名稱、索引排序索引鍵的查詢準則，以及您要傳回的屬性。在本例中，索引是 AlbumTitleIndex，而索引排序索引鍵為 AlbumTitle。

傳回的唯一屬性是已投影到索引的屬性。您也可以修改此查詢來選擇非索引鍵屬性，但這需要相對昂貴的資料表擷取活動。如需資料表擷取的詳細資訊，請參閱 [屬性投影](#)。

以下是使用 AWS SDK for Java Document API 查詢本機次要索引的步驟。

1. 建立 DynamoDB 類別的執行個體。
2. 建立 Table 類別的執行個體。您必須提供資料表名稱。
3. 建立 Index 類別的執行個體。您必須提供索引名稱。
4. 呼叫 Index 類別的 query 方法。

下列 Java 程式碼範例示範上述步驟。

### 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());
}
```

### 範例：使用 Java Document API 的本機次要索引

下列 Java 程式碼範例示範如何使用 Amazon DynamoDB 中的本機次要索引。例如，您可以建立名為 CustomerOrders 的資料表，其中分割區索引鍵為 CustomerId，排序索引鍵為 OrderId。此資料表上有兩個本機次要索引：

- OrderCreationDateIndex：排序索引鍵為 OrderCreationDate，並且以下屬性會投影到索引：
  - ProductCategory
  - ProductName
  - OrderStatus
  - ShipmentTrackingId
- IsOpenIndex：排序索引鍵為 IsOpen，並且所有的資料表屬性都會投影到索引。

建立 CustomerOrders 資料表後，程式會載入所含資料表示客戶訂單的資料表。然後，其便會使用本機次要索引查詢資料。最後，程式會刪除 CustomerOrders 資料表。

如需測試下列範例的 step-by-step 指示，請參閱[Java 程式碼範例](#)。

### 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;
 }
}
```

## 使用本機次要索引：.NET

### 主題

- [使用本機次要索引建立資料表](#)
- [使用本機次要索引描述資料表](#)
- [查詢本機次要索引](#)
- [範例：使用 AWS SDK for .NET 低階 API 的本機次要索引](#)

您可以使用 AWS SDK for .NET 低階 API 建立含有一或多個本機次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，以及使用索引執行查詢。這些操作會映射至對應的低階 DynamoDB API 動作。如需詳細資訊，請參閱 [.NET 程式碼範例](#)。

下列是使用 .NET 低階 API 執行資料表操作的一般步驟。

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。
2. 透過建立對應的請求物件，為操作提供必要及選用的參數。

例如，建立 `CreateTableRequest` 物件來建立資料表，以及建立 `QueryRequest` 物件來查詢資料表或索引。

3. 執行您在前一步驟中建立之用戶端所提供的適當方法。

### 使用本機次要索引建立資料表

您必須同時建立資料表和本機次要索引。若要執行這項操作，請使用 `CreateTable`，並提供一或多個本機次要索引的規格。以下 C# 程式碼範例會建立資料表來保存音樂收藏中歌曲的相關資訊。分割區索引鍵為 `Artist`，而排序索引鍵為 `SongTitle`。次要索引 `AlbumTitleIndex` 有助於依照專輯標題查詢。

以下是使用 .NET 低階 API 建立具有本機次要索引之資料表的步驟。

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。
2. 建立 `CreateTableRequest` 類別的執行個體，以提供請求資訊。

您必須提供資料表名稱、其主索引鍵，以及佈建的輸送量數值。對於本機次要索引，您必須提供索引名稱、索引排序索引鍵的名稱和資料類型、索引的索引鍵結構描述以及屬性投影。

3. 以參數形式提供請求物件，以便執行 `CreateTable` 方法。

下列 C# 程式碼範例示範前述步驟。程式碼會建立一個資料表 (Music) 與次要索引 AlbumTitle 屬性。資料表分割區索引鍵和排序索引鍵以及索引排序索引鍵，是唯一投影到索引的屬性。

```
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);
```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。

使用本機次要索引描述資料表

若要取得資料表上本機次要索引的資訊，請使用 DescribeTable API。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

以下是使用 .NET 低階 API 存取資料表的本機次要索引資訊的步驟。

1. 建立 `AmazonDynamoDBClient` 類別的執行個體。
2. 建立 `DescribeTableRequest` 類別的執行個體，以提供請求資訊。您必須提供資料表名稱。
3. 以參數形式提供請求物件，以便執行 `describeTable` 方法。
- 4.

下列 C# 程式碼範例示範前述步驟。

### 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);
 }
 }
}
```

```
 }
}
```

## 查詢本機次要索引

您可以依照與 Query 資料表大致相同的方式在本機次要索引上使用 Query。您必須指定索引名稱、索引排序索引鍵的查詢準則，以及您要傳回的屬性。在本例中，索引是 AlbumTitleIndex，而索引排序索引鍵為 AlbumTitle。

傳回的唯一屬性是已投影到索引的屬性。您也可以修改此查詢來選擇非索引鍵屬性，但這需要相對昂貴的資料表擷取活動。如需資料表擷取的詳細資訊，請參閱 [屬性投影](#)

以下是使用 .NET 低階 API 查詢本機次要索引的步驟。

1. 建立 AmazonDynamoDBClient 類別的執行個體。
2. 建立 QueryRequest 類別的執行個體，以提供請求資訊。
3. 以參數形式提供請求物件，以便執行 query 方法。

下列 C# 程式碼範例示範前述步驟。

## 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();
}
```

範例：使用 AWS SDK for .NET 低階 API 的本機次要索引

下列 C# 程式碼範例示範如何在 Amazon DynamoDB 中使用本機次要索引。例如，您可以建立名為 CustomerOrders 的資料表，其中分割區索引鍵為 CustomerId，排序索引鍵為 OrderId。此資料表上有兩個本機次要索引：

- OrderCreationDateIndex：排序索引鍵為 OrderCreationDate，並且以下屬性會投影到索引：
  - ProductCategory
  - ProductName
  - OrderStatus
  - ShipmentTrackingId
- IsOpenIndex：排序索引鍵為 IsOpen，並且所有的資料表屬性都會投影到索引。

建立 CustomerOrders 資料表後，程式會載入所含資料表示客戶訂單的資料表。然後，其便會使用本機次要索引查詢資料。最後，程式會刪除 CustomerOrders 資料表。

如需測試下列範例的 step-by-step 指示，請參閱[.NET 程式碼範例](#)。

## 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"
 } }
};

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;
 }
 }
}
}
```

## 使用本機次要索引：AWS CLI

您可以使用 AWS CLI 建立含有一或多個本機次要索引的 Amazon DynamoDB 資料表、描述資料表上的索引，以及使用索引執行查詢。

### 主題

- [使用本機次要索引建立資料表](#)
- [使用本機次要索引描述資料表](#)
- [查詢本機次要索引](#)

### 使用本機次要索引建立資料表

您必須同時建立資料表和本機次要索引。若要執行這項操作，請使用 `create-table` 參數，並提供一或多個本機次要索引的規格。以下範例會建立資料表 (Music) 來保存音樂收藏中歌曲的相關



資訊。分割區索引鍵為 Artist，而排序索引鍵為 SongTitle。AlbumTitle 屬性上的次要索引 AlbumTitleIndex 有助於依照專輯標題查詢。

```
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\\"]}}]"
```

您必須等到 DynamoDB 建立資料表，並將資料表狀態設定為 ACTIVE。之後，您可以開始將資料項目放入資料表中。您可以使用 [describe-table](#) 來判斷資料表建立的狀態。

使用本機次要索引描述資料表

若要取得資料表上本機次要索引的資訊，請使用 describe-table 參數。對於每個索引，您可以存取其名稱、索引鍵結構描述和投影屬性。

```
aws dynamodb describe-table --table-name Music
```

查詢本機次要索引

您可以依照與 query 資料表大致相同的方式在本機次要索引上使用 query 操作。您必須指定索引名稱、索引排序索引鍵的查詢準則，以及您要傳回的屬性。在本例中，索引是 AlbumTitleIndex，而索引排序索引鍵為 AlbumTitle。

傳回的唯一屬性是已投影到索引的屬性。您也可以修改此查詢來選擇非索引鍵屬性，但這需要相對昂貴的資料表擷取活動。如需資料表擷取的詳細資訊，請參閱 [屬性投影](#)。

```
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"} }'
```

## 管理 DynamoDB 交易的複雜工作流程

Amazon DynamoDB 交易可簡化開發人員在表格內和跨表格對多個項目進行協調 all-or-nothing 變更的體驗。交易在 DynamoDB 中提供了不可分割性、一致性、隔離性和持久性 (ACID)，讓您能夠輕鬆地維持應用程式的資料正確性。

您可以使用 DynamoDB 交易式讀寫 API 來管理複雜的業務工作流程，這些工作流程需要以單一作業的形式新增、更新或刪除多個項目。all-or-nothing 例如，當玩家在電玩遊戲中交換物品或在遊戲中購買物品時，遊戲開發人員可以確保玩家的資料正確更新。

使用交易寫入 API，您可分組多個 Put、Update、Delete 和 ConditionCheck 動作。然後，將動作提交為單一 TransactWriteItems 操作，以單位形式成功或失敗。多項 Get 動作也可以相同的方式處理，在將這些動作分組後，當做單一 TransactGetItems 操作提交。

在您的 DynamoDB 資料表中啟用交易功能，不需額外付費。您只需針對交易中所進行的讀取和寫入付費即可。DynamoDB 會對交易中的每個項目進行兩項基本的讀取和寫入動作：一項是用來準備交易，一項是用來遞交交易。這兩個基礎讀取/寫入操作在您的 Amazon CloudWatch 指標中可見。

若要開始使用 DynamoDB 交易，請下載最新的 AWS 開發套件或 AWS Command Line Interface (AWS CLI)。然後遵循 [DynamoDB 交易範例](#)。

下列各節提供交易 API 的詳細概觀，並說明如何在 DynamoDB 中使用這些 API。

### 主題

- [Amazon DynamoDB Transactions : 運作方式](#)
- [將 IAM 與 DynamoDB 交易搭配使用](#)
- [DynamoDB 交易範例](#)

## Amazon DynamoDB Transactions : 運作方式

使用 Amazon DynamoDB 交易，您可以將多個動作分組在一起，然後將它們作為單一 all-or-nothing TransactWriteItems 或 TransactGetItems 作業提交。下列各節說明 API 操作、容量管理、最佳實務，以及關於在 DynamoDB 中使用交易操作的其他詳細資訊。

### 主題

- [TransactWriteItems API](#)
- [TransactGetItems API](#)
- [DynamoDB 交易的隔離層級](#)
- [DynamoDB 中的交易衝突處理](#)
- [使用 DynamoDB Accelerator \(DAX\) 中的交易 API](#)
- [交易的容量管理](#)
- [交易的最佳實務](#)
- [將交易 API 與全域資料表搭配使用](#)
- [交易與交易用戶端程式 AWSLabs 庫](#)

## TransactWriteItems API

TransactWriteItems 是同步且冪等的寫入作業，可在單 all-or-nothing 一作業中將多達 100 個寫入動作分組。這些動作可以將目標鎖定為一或多個 DynamoDB 資料表 (在相同的 AWS 帳戶和區域) 中的 100 個不同項目。交易中項目的彙總大小不能超過 4 MB。這些動作的完成具有不可分割性，也就是全部成功或全部失敗。

### Note

- TransactWriteItems 操作與 BatchWriteItem 操作的不同之處，在於前者所包含的全部動作，都必須順利地完成，否則就完全不會進行變更。如果是使用 BatchWriteItem 操作，即可允許批次中只有某些動作順利完成 (其他的動作失敗)。
- 無法使用索引執行交易。

在同一筆交易中，您無法針對同一個項目進行多項操作。例如，您不能對同一筆交易中的同一個項目既執行 ConditionCheck 又執行 Update 動作。

您可以將下列類型的動作新增到交易中：

- Put：發起 PutItem 操作，來建立新的項目，或是用新的項目取代舊的項目 (具有條件或不指定任何條件)。
- Update：發起 UpdateItem 操作，來編輯現有項目的屬性，或是在新項目尚未存在時，將新項目加入到資料表。利用此項動作，來新增、刪除或更新現有項目的屬性 (具有條件或無條件)。
- Delete：發起 DeleteItem 操作，透過指定資料表中某項目的主索引鍵，來刪除該單一項目。

- `ConditionCheck`：查看某項目是否存在，或是查看某項目特定屬性的條件。

交易順利完成後，該交易內完成的變更會傳播到全域次要索引 (GSI)、串流和備份。由於傳輸不是立即或即時的，所以如果資料表是從 `backup` ([RestoreTableFromBackup](#)) 還原或匯出到時間點 ([ExportTableToPointInTime](#)) 中間傳輸，它可能會包含最近交易期間所做的一些變更，但不是所有變更。

## 冪等性

在進行 `TransactWriteItems` 呼叫時，您可以選擇性地加入用戶端符記，以確保請求具有冪等性。如果讓交易具有冪等性，則在因為連線逾時或其他連線問題，而多次提交同一項操作時，有助於防止應用程式發生錯誤。

如果原始的 `TransactWriteItems` 呼叫順利完成，便會成功傳回後續具有相同用戶端字符的 `TransactWriteItems` 呼叫，不建立任何變更。如果設定了 `ReturnConsumedCapacity` 參數，則初始的 `TransactWriteItems` 呼叫會傳回在建立變更時所使用的寫入容量單位數量。後續具有相同用戶端符記的 `TransactWriteItems` 呼叫，會傳回在讀取項目時所用掉的讀取容量單位數量。

## 關於等冪性的重點

- 用戶端符記在使用該符記的請求完成後 10 分鐘內為有效。10 分鐘過後，使用同一個用戶端符記的任何請求，會被視為新的請求。在經過 10 分鐘之後，您就不應該針對相同的請求，重複使用同一個用戶端符記。
- 如果您在 10 分鐘的冪等性有效期間內，使用相同的用戶端符記來重複請求，但變更了其他的某些請求參數，則 DynamoDB 會傳回 `IdempotentParameterMismatch` 例外。

## 寫入的錯誤處理

在下列的情況中，寫入交易不會成功：

- 當其中一個條件表達式中的條件不符時。
- 因為在同一個 `TransactWriteItems` 操作中，有超過一個以上的動作鎖定相同的項目，而造成交易驗證錯誤時。
- 如果 `TransactWriteItems` 請求與 `TransactWriteItems` 請求中一或多個項目持續執行的 `TransactWriteItems` 操作相互衝突時。在這種情況中，請求會失敗，並且丟出 `TransactionCanceledException` 例外。
- 佈建的容量不足，而使交易無法完成時。

- 當項目的大小過大 (超過 400 KB)、本機次要索引 (LSI) 變得過大，或是因為交易所進行的變更造成類似的驗證錯誤時。
- 發生使用者錯誤時，例如無效的資料格式。

如需如何處理 `TransactWriteItems` 操作衝突的詳細資訊，請參閱 [DynamoDB 中的交易衝突處理](#)。

## TransactGetItems API

`TransactGetItems` 是一項同步讀取操作，可將最多 100 個 `Get` 動作歸成一組。這些動作可以將目標鎖定為一或多個 DynamoDB 資料表 (在相同的 AWS 帳戶和區域) 中的 100 個不同項目。交易中項目的彙總大小不能超過 4 MB。

`Get` 動作的執行具有不可分割性，也就是全部成功或全部失敗：

- `Get`：發起 `GetItem` 操作，針對具有指定主索引鍵的項目，擷取該項目的一組屬性。如果找不到符合的項目，則 `Get` 不會傳回任何資料。

### 讀取的錯誤處理

在下列的情況中，讀取交易不會成功：

- 如果 `TransactGetItems` 請求與 `TransactGetItems` 請求中一或多個項目持續執行的 `TransactWriteItems` 操作相互衝突時。在這種情況中，請求會失敗，並且丟出 `TransactionCanceledException` 例外。
- 佈建的容量不足，而使交易無法完成時。
- 發生使用者錯誤時，例如無效的資料格式。

如需如何處理 `TransactGetItems` 操作衝突的詳細資訊，請參閱 [DynamoDB 中的交易衝突處理](#)。

## DynamoDB 交易的隔離層級

交易操作 (`TransactWriteItems` 或 `TransactGetItems`) 和其他操作的隔離層級如下。

### 可序列化

如果在前一項操作完成之前，沒有任何操作開始執行，則可序列化隔離可確保多個同時並行操作的結果會相同。

在下列的操作類型之間，具有可序列化的隔離層級：

- 在任何交易操作和任何標準寫入操作 (PutItem、UpdateItem 或 DeleteItem) 之間。
- 在任何交易操作和任何標準讀取操作 (GetItem) 之間。
- 在 TransactWriteItems 操作和 TransactGetItems 操作之間。

雖然事務性操作之間存在可序列化隔離，以及操作中的每個單獨的標準寫入，但是事務與作為一個單元的 BatchWriteItem 操作之間沒有可序列化的隔離。

同樣地，交易操作與 BatchGetItem 操作中個別 GetItems 之間的隔離層是可以序列化。但是交易與當作一個單位的 BatchGetItem 操作之間的隔離層是專供讀取。

單一 GetItem 請求是兩種可序列化的 TransactWriteItems 請求方式之一，可以發生在 TransactWriteItems 請求之前或之後。與 TransactWriteItems 請求同時發出的多個 GetItem 請求，能夠以任何順序執行，因此結果會是專供讀取。

例如，如果項目 A 和項目 B 的 GetItem 請求與修改項目 A 和項目 B 的 TransactWriteItems 請求同時執行，則會有四種可能性：

- 兩個 GetItem 請求皆會在 TransactWriteItems 請求之前執行。
- 兩個 GetItem 請求皆會在 TransactWriteItems 請求之後執行。
- 項目 A 的 GetItem 請求會在 TransactWriteItems 請求之前執行。針對項目 B，GetItem 會在 TransactWriteItems 之後執行。
- 項目 B 的 GetItem 請求會在 TransactWriteItems 請求之前執行。針對項目 A，GetItem 會在 TransactWriteItems 之後執行。

TransactGetItems 如果您更喜歡多個 GetItem 請求的可序列化隔離級別，則應該使用。

如果在執行中屬於相同交易寫入要求的多個項目上進行非交易讀取，您可能能夠讀取某些項目的新狀態和其他項目的舊狀態。只有在收到交易寫入的成功回應時，您才能讀取屬於交易寫入要求一部分的所有項目的新狀態。

### 專供讀取

專供讀取隔離可確保讀取操作對特定項目一律傳回已確認的值 - 如果交易寫入最終未成功，則讀取操作一律不向該項目呈現此寫入狀態的檢視。專供讀取隔離不會防止在讀取操作後立即修改項目。

在任何交易操作和牽涉到多次標準讀取 (BatchGetItem、Query 或 Scan) 的任何讀取操作之間，其隔離層級為專供讀取。如果交易寫入在 BatchGetItem、Query 或 Scan 操作期間更新項目，則讀

取操作的後續部分會傳回新確認的值 (含有 ConsistentRead) 或可能是先前已確認的值(最終一致讀取)。

## 操作摘要

做為總結，下表顯示了交易操作 (TransactWriteItems 或 TransactGetItems) 和其他操作之間的隔離層級。

| 操作             | 隔離層級   |
|----------------|--------|
| DeleteItem     | 可序列化   |
| PutItem        | 可序列化   |
| UpdateItem     | 可序列化   |
| GetItem        | 可序列化   |
| BatchGetItem   | 專供讀取*  |
| BatchWriteItem | 不可序列化* |
| Query          | 專供讀取   |
| Scan           | 專供讀取   |
| 其他交易操作         | 可序列化   |

標有星號的層級 (\*) 適用於單位式操作。不過，這些操作內的個別動作具有可序列化的隔離層級。

## DynamoDB 中的交易衝突處理

在交易內的項目上進行並行項目層級請求期間，可能會發生交易衝突。下列情境可能發生交易衝突：

- 項目的 PutItem、UpdateItem 或 DeleteItem 請求與包括相同項目的持續 TransactWriteItems 請求發生衝突。
- TransactWriteItems 請求內的項目是另一個持續 TransactWriteItems 請求的一部分。
- TransactGetItems 請求的項目是持續 TransactWriteItems、BatchWriteItem、PutItem、UpdateItem 或 DeleteItem 請求的一部分。



**Note**

- 當 PutItem、UpdateItem 或 DeleteItem 請求遭到拒絕時，請求失敗並顯示 TransactionConflictException。
- 如果 TransactWriteItems 或 TransactGetItems 內的項目層級請求遭到拒絕，則請求失敗並顯示 TransactionCanceledException。如果該請求失敗，AWS 開發套件不會重試請求。

如果您使用的是 AWS SDK for Java，例外狀況會包含的清單 [CancellationReasons](#)，根據 TransactItems request 參數中的項目清單進行排序。對於其他語言，清單的字串表示法會併入異常的錯誤訊息中。

- 不過，如果有持續執行的 TransactWriteItems 和 TransactGetItems 操作，與同時並行的 GetItem 請求相互衝突時，這兩項操作都可以順利完成。

每個失敗的項目層級要求都會增加 [TransactionConflict CloudWatch 量度量](#)。

## 使用 DynamoDB Accelerator (DAX) 中的交易 API

DynamoDB Accelerator (DAX) 中支援 TransactWriteItems 和 TransactGetItems，且隔離層級與在 DynamoDB 中相同。

TransactWriteItems 會透過 DAX 寫入。DAX 會將 TransactWriteItems 呼叫傳送給 DynamoDB，然後傳回回應。為在寫入後填入快取，對於 TransactWriteItems 操作中的每個項目，DAX 會在背景呼叫 TransactGetItems，並耗用更多讀取容量單位。(如需詳細資訊，請參閱 [交易的容量管理](#)。) 此功能可讓您保持簡單的應用程式邏輯，並使用 DAX 執行交易及非交易操作。

TransactGetItems 呼叫透過 DAX 傳遞，項目不在本機進行快取。這個行為與 DAX 中的強烈一致讀取 API 相同。

## 交易的容量管理

在您的 DynamoDB 資料表中啟用交易功能，不需額外付費。您只需針對交易中所進行的讀取和寫入付費即可。DynamoDB 會對交易中的每個項目進行兩項基本的讀取和寫入動作：一項是用來準備交易，一項是用來遞交交易。您的 Amazon CloudWatch 指標中會顯示兩個基礎讀取/寫入操作。

在為資料表佈建容量時，請規劃交易 API 所需的額外讀取與寫入容量。例如，假設您的應用程式每秒執行一項交易，而每項交易會在您的資料表中寫入三個 500 位元組的項目。每個項目需要兩個寫入容



量單位 (WCU)：一個單位用來準備交易，另一個單位用來遞交交易。因此，您會需要佈建六個 WCU 給該資料表。

如果您在前一個範例中使用 DynamoDB Accelerator (DAX)，則也會針對 `TransactWriteItems` 呼叫中的每個項目使用兩個讀取容量單位 (RCU)。因此，您會需要佈建六個額外 RCU 給資料表。

同樣地，如果您的應用程式每秒執行一次讀取交易，而每項交易會在您的資料表中讀取三個 500 位元組的項目，則您會需要佈建六個讀取容量單位 (RCU) 給該資料表。讀取每個項目需要兩個 RCU：一個用來準備交易，另一個用來遞交交易。

此外，在出現 `TransactionInProgressException` 例外時，預設的 SDK 動作是重試交易。請規劃這些重試動作會使用的額外讀取容量單位 (RCU)。如果用您自己的程式碼，使用 `ClientRequestToken` 來重試交易時，也請進行同樣的規劃。

## 交易的最佳實務

在使用 DynamoDB 交易時，請考慮採用下列建議的做法。

- 啟用資料表的自動調整規模功能，或是確定您已佈建足夠的輸送容量，來針對您交易中的每個項目，進行兩項讀取或寫入操作。
- 如果您未使用 AWS 所提供的開發套件，請在發出 `TransactWriteItems` 呼叫時加入 `ClientRequestToken` 屬性，以確保請求具有冪等性。
- 如非必要，請勿在交易中將操作歸為一組。例如，如果包含 10 項操作的單一交易，可以分成多項交易執行，而不會影響到應用程式的正確度，則我們建議拆分該項交易。簡化的交易可提升輸送量，而且成功的機率更高。
- 更新同一個項目的多項交易如果同時執行，可能會造成衝突而導致交易取消。我們建議採用下列的 DynamoDB 資料建模最佳實務，來將此等衝突減到最少。
- 如果在單一交易中，經常更新跨多個項目的一組屬性，請考慮將這些屬性分組成為單一項目，來縮小交易的範圍。
- 避免使用大量擷取資料的交易。如果是大量寫入作業，較理想的做法是使用 `BatchWriteItem`。

## 將交易 API 與全域資料表搭配使用

DynamoDB 交易中包含的作業只能在最初執行交易的區域中保證交易。當在交易中套用的變更跨區域複製到全域表格複本時，不會保留交易性。

## 交易與交易用戶端程式 AWS Labs 庫

DynamoDB 交易可為交易用戶端程式庫提供更符合成本效益、穩健且高效能的 [AWS Labs](#) 替代品。我們建議您更新應用程式，以使用原生的伺服器端交易 API。

## 將 IAM 與 DynamoDB 交易搭配使用

您可以利用 AWS Identity and Access Management (IAM) 來限制交易操作可在 Amazon DynamoDB 中執行的動作。如需在 DynamoDB 中使用 IAM 政策的詳細資訊，請參閱 [適用於 DynamoDB 的以身分為基礎的政策](#)。

Put、Update、Delete 和 Get 動作的許可，受到底層 PutItem、UpdateItem、DeleteItem 和 GetItem 操作使用的許可所管理。如果是 ConditionCheck 動作，您可以使用 IAM 政策中的 dynamodb:ConditionCheck 許可。

您可以使用下列的 IAM 政策範例，來設定 DynamoDB 交易。

### 範例 1：允許交易操作

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "dynamodb:ConditionCheckItem",
 "dynamodb:PutItem",
 "dynamodb:UpdateItem",
 "dynamodb>DeleteItem",
 "dynamodb:GetItem"
],
 "Resource": [
 "arn:aws:dynamodb:*:*:table/table04"
]
 }
]
}
```

### 範例 2：只允許交易操作

```
{
```

```

"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"
]
 }
 }
 }
]
}

```

### 範例 3：允許非交易讀取與寫入，並且封鎖交易讀取與寫入

```

{
 "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"
]
 }
]
}

```

#### 範例 4：防止 ConditionCheck 失敗時傳回資訊

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "dynamodb:ConditionCheckItem",
 "dynamodb:PutItem",
 "dynamodb:UpdateItem",
 "dynamodb>DeleteItem",
 "dynamodb:GetItem"
],
 "Resource": "arn:aws:dynamodb:*:*:table/table01",
 "Condition": {
 "StringEqualsIfExists": {
 "dynamodb:ReturnValues": "NONE"
 }
 }
 }
]
}

```

```
 }
 }
]
}
```

## DynamoDB 交易範例

以 Amazon DynamoDB Transactions 十分實用的情況作為範例，請考慮此線上市場的 Java 應用程式範例。

應用程式在其後端有三個 DynamoDB 資料表：

- **Customers**：此資料表儲存有關市場客戶的詳細資訊。其主索引鍵為 `CustomerId` 唯一識別符。
- **ProductCatalog**：此資料表儲存的詳細資訊包括關於市場上銷售之產品的價格和供貨情況。其主索引鍵為 `ProductId` 唯一識別符。
- **Orders**：此資料表儲存有關市場訂單的詳細資訊。其主索引鍵為 `OrderId` 唯一識別符。

### 下訂單

下列程式碼片段說明如何使用 DynamoDB 交易來協調建立和處理訂單所需的多個步驟。使用單一「全有或全無」操作，可確保如果交易的任何部分失敗，便不會執行交易中的任何動作，也不會進行任何變更。

在此範例中，您可以為 `customerId` 為 `09e8e9c8-ec48` 的客戶設定訂單。然後，您可以使用下列簡單的訂單處理工作流程，將其當作單一交易執行：

1. 判斷客戶 ID 有效。
2. 確定產品為 `IN_STOCK`，並將產品狀態更新為 `SOLD`。
3. 請確定訂單不存在，並建立訂單。

### 驗證客戶

首先，請定義一個動作，以便驗證客戶資料表中是否存在 `customerId` 等於 `09e8e9c8-ec48` 的客戶。

```
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 + ")");
```

## 更新產品狀態

接下來，如果產品狀態目前是將 `IN_STOCK` 設定為 `true`，請定義要將產品狀態更新至 `SOLD` 的動作。如果項目的產品狀態屬性不等於 `IN_STOCK`，則設定 `ReturnValuesOnConditionCheckFailure` 參數會傳回項目。

```
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);
```

## 建立訂單

最後，只要具有 `OrderId` 的訂單尚不存在，便可建立訂單。

```
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 + ")");
```

## 執行交易

下列範例說明如何執行先前定義為「全有或全無」的單一操作動作。

```
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());
}
```

## 讀取訂單詳細資訊

下列範例示範如何以交易方式讀取 Orders 和 ProductCatalog 資料表中的已完成訂單。

```
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());
}
```

## 其他範例

- [使用來自 DynamoDBMapper 的交易](#)

## 使用 Amazon DynamoDB 變更資料擷取

當存放在 DynamoDB 資料表中的項目發生變更時，擷取這類變更的功能可為許多應用程式提供好處。下列是一些範例使用案例：

- 熱門行動應用程式以每秒數千筆更新的速率修改 DynamoDB 資料表中的資料。另一個應用程式會擷取並儲存這些更新的相關資料，為行動應 near-real-time 用程式提供使用量度。



- 財務應用程式會修改 DynamoDB 資料表中的股票市場資料。並行運 parallel 的不同應用程式可實時跟踪這些變化 value-at-risk，計算並根據股票價格變動自動重新平衡投資組合。
- 運輸車輛和工業設備中的感應器會將資料傳送至 DynamoDB 資料表。不同的應用程式會監控效能並在偵測到問題時傳送簡訊提醒，透過應用機器學習演算法預測任何潛在缺陷，以及將資料壓縮和封存到 Amazon Simple Storage Service (Amazon S3)。
- 應用程式在某位朋友上傳新圖片時，立即自動傳送通知給群組中所有朋友的行動裝置。
- 新客戶將資料新增至 DynamoDB 資料表。此事件呼叫另一個應用程式，該應用程式會將歡迎電子郵件傳送給新客戶。

DynamoDB 支援近乎即時的項目層級變更資料擷取紀錄串流。您可以建立使用這些串流並根據內容採取動作的應用程式。

以下影片將為您介紹變更資料擷取概念。

### [資料表容量模式](#)

#### 主題

- [變更資料擷取的串流選項](#)
- [使用 Kinesis Data Streams 來擷取 DynamoDB 的變更](#)
- [DynamoDB Streams 的變更資料擷取](#)

## 變更資料擷取的串流選項

DynamoDB 提供兩種用於變更資料擷取的串流模型：DynamoDB 專用 Kinesis Data Streams 和 DynamoDB Streams。

為了協助您為應用程式選擇合適的解決方案，下表摘要說明每種串流模型的特色。

| 屬性                              | DynamoDB 專用 Kinesis Data Streams      | DynamoDB Streams               |
|---------------------------------|---------------------------------------|--------------------------------|
| 資料保留                            | 最多 <a href="#">1 年</a> 。              | 24 小時。                         |
| Kinesis Client Library (KCL) 支援 | 支援 <a href="#">KCL 1.X 版和 2.X 版</a> 。 | 支援 <a href="#">KCL 1.X 版</a> 。 |

| 屬性     | DynamoDB 專用 Kinesis Data Streams                                                                                                                                                  | DynamoDB Streams                                                                     |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| 消費者數量  | 每個碎片最多 <a href="#">5 個同時</a> 消費者，每個具有 <a href="#">強化廣播功能</a> 的碎片最多 20 個同時消費者。                                                                                                     | 每個碎片最多 <a href="#">2 個同時</a> 消費者。                                                    |
| 輸送量配額  | 無限制。                                                                                                                                                                              | 根據 DynamoDB 表格和 AWS 區域的輸送量 <a href="#">配額</a> 而定。                                    |
| 記錄交付模式 | Kinesis Data Streams 使用 <a href="#">GetRecords</a> 和 <a href="#">增強型</a> <a href="#">散發功能</a> 透過 HTTP 提取模型，透過使用推送記錄透過 HTTP/2。 <a href="#">SubscribeToShard</a>                    | 使用 HTTP 提取模型 <a href="#">GetRecords</a> 。                                            |
| 記錄排序   | 每個串流紀錄上的時間戳記屬性可用來識別 DynamoDB 資料表中發生變更的實際順序。                                                                                                                                       | 對於 DynamoDB 資料表中的每個修改項目，串流紀錄的出現順序與項目的實際修改順序相同。                                       |
| 複製記錄   | 串流中偶爾會出現重複紀錄。                                                                                                                                                                     | 串流中沒有出現重複紀錄。                                                                         |
| 串流處理選項 | 使用 <a href="#">AWS Lambda</a> 、 <a href="#">Amazon Managed Service for Apache Flink</a> 、 <a href="#">Kinesis Data Firehose</a> 或 <a href="#">AWS Glue Streaming ETL</a> 來處理串流記錄。 | 使用 <a href="#">AWS Lambda</a> 或 <a href="#">DynamoDB Streams Kinesis 轉接器</a> 處理串流記錄。 |
| 耐久性等級  | <a href="#">可用區域</a> 會提供自動不中斷的容錯移轉。                                                                                                                                               | <a href="#">可用區域</a> 會提供自動不中斷的容錯移轉。                                                  |

您可以在同一個 DynamoDB 資料表上啟用這兩種串流模型。

下面這段影片將進一步說明這兩個選項之間的差異。

## [DynamoDB Streams 與 Kinesis Data Streams](#)

## 使用 Kinesis Data Streams 來擷取 DynamoDB 的變更

您可以使用 Amazon Kinesis Data Streams 來擷取 Amazon DynamoDB 的變更。

Kinesis Data Streams 會擷取任何 DynamoDB 資料表中的項目層級修改，並將其複製到您選擇的 [Kinesis 資料串流](#)。您的應用程式可以存取此串流，並以近乎即時的速度檢視項目層級的變更。您每小時可以持續擷取和儲存 TB 級的資料。您可以利用較長的資料保留時間，並透過增強的廣發功能，您可以同時觸及兩個以上的下游應用程式。其他優點包括額外的稽核和安全透明度。

Kinesis Data Streams 也可讓您存取 [Amazon 資料 Firehose](#) 和 [Amazon 阿帕奇 Flink 管理服務](#)。這可讓您建置應用程式來啟動即時儀表板、產生提醒、實作動態定價和廣告，以及執行複雜的資料分析及機器學習演算法。

### Note

使用 DynamoDB 專用 Kinesis 資料串流，會同時適用於資料串流 [Kinesis Data Streams 定價](#) 和來源資料表的 [DynamoDB 定價](#)。

## Kinesis Data Streams 如何與 DynamoDB 搭配運作

當 DynamoDB 資料表啟用 Kinesis 資料串流時，資料表會傳送資料記錄，以擷取該資料表資料的任何變更。此資料記錄包括：

- 最近建立、更新或刪除之任何項目的具體時間
- 該項目的主鍵
- 修改前記錄的快照
- 修改後記錄的快照

這些資料記錄會以接近即時的速度擷取和發佈，並予以發佈。將其寫入 Kinesis 資料串流後，就可以像其他記錄一樣讀取。您可以使用 Kinesis 用戶端程式庫、使用 AWS Lambda、呼叫 Kinesis Data Streams API，以及使用其他連線的服務。如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的 [從 Amazon Kinesis Data Streams 讀取資料](#)。

這些對資料的變更也會以非同步方式擷取。Kinesis 對其串流來源的資料表沒有效能影響。存放在 Kinesis 資料串流中的串流紀錄會採用靜態加密。如需詳細資訊，請參閱 [Amazon Kinesis Data Streams 中的資料保護](#)。

Kinesis 資料串流記錄的顯示順序可能會與發生項目變更時不同。相同的項目通知也可能會在串流中多次出現。您可以檢查 `ApproximateCreationDateTime` 屬性以識別發生項目修改的順序，並識別重複的記錄。

當您啟用 Kinesis 資料串流作為 DynamoDB 表格的串流目標時，您可以在毫秒或微秒內設定 `ApproximateCreationDateTime` 值的精確度。默認情況下，`ApproximateCreationDateTime` 指示以毫秒為單位的更改時間。此外，您可以在作用中的串流目的地上變更此值。在此類更新之後，寫入 Kinesis 的串流記錄將具有所需精確度的 `ApproximateCreationDateTime` 值。

寫入 DynamoDB 的二進位值必須使用 [base64 編碼格式](#) 進行編碼。然而，當資料記錄寫入 Kinesis 資料串流時，這些編碼的二進位值會使用 base64 編碼再次編碼。從 Kinesis 資料串流讀取這些記錄時，為了擷取原始二進位值，應用程式必須將這些值解碼兩次。

使用 Kinesis Data Streams 時，DynamoDB 會根據變更資料擷取單位收取費用。每個單一項目的 1 KB 變更會計為一個變更資料擷取單位。使用與 [寫入作業的容量單位耗用量](#) 相同的邏輯，以寫入串流之項目的「之前」和「之後」映像中較大者，計算每個項目的 KB 變化量。運作方式與 DynamoDB [隨需](#) 模式類似，您不需要為變更資料擷取單位佈建容量輸送量。

### 為 DynamoDB 資料表開啟 Kinesis 資料串流

您可以使用 AWS Management Console、AWS SDK 或 ()，從現有 DynamoDB 表格啟用或停用串流至 Kinesis 的串流。AWS Command Line Interface AWS CLI

- 您只能在與表格相同的 AWS 帳戶和 AWS 區域中將資料從 DynamoDB 串流至 Kinesis 資料串流。
- 您只能將 DynamoDB 資料表中的資料串流至一個 Kinesis 資料串流。

### 變更 DynamoDB 表格上的運動資料串流目的地

根據預設，所有 Kinesis 資料串流記錄都包含一個 `ApproximateCreationDateTime` 屬性。此屬性表示創建每個記錄時的大約時間的毫秒的時間戳。您可以變更這些值的精確度，方法是使用 <https://console.aws.amazon.com/kinesis>、SDK 或 AWS CLI

## Amazon DynamoDB 專用 Kinesis Data Streams 入門

本節說明如何將適用於 Amazon DynamoDB 的 Kinesis Data Streams 表與 Amazon DynamoDB 主控台、AWS Command Line Interface (AWS CLI) 和 API 搭配使用。

所有這些範例都使用 Music DynamoDB 資料表，該資料表是在 [DynamoDB 入門](#) 教學課程中建立的。

若要進一步了解如何建立取用者並將 Kinesis 資料串流連接到其他 AWS 服務，請參閱 [Amazon Kinesis 資料串流開發人員指南中的從 Kinesis 資料串流讀取資料](#)。

#### Note

第一次使用 KDS 碎片時，建議您將碎片設定為隨著使用模式縱向擴展和縮小。累積更多使用模式的相關資料後，您可以調整串流中的碎片以進行配對。

## Console

1. 登入 AWS Management Console 並開啟運動主控台，網址為 <https://console.aws.amazon.com/kinesis/>。
2. 選擇 Create data stream (建立資料串流)，並依照說明來建立名為 samplestream 的串流。
3. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
4. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
5. 選擇 Music (音樂) 資料表。
6. 選擇 Exports and streams (匯出與串流) 索引標籤。

The screenshot shows the 'Exports and streams' tab for a table named 'Music'. It includes a navigation bar with tabs for Overview, Indexes, Monitor, Global tables, Backups, Exports and streams, and Additional settings. The 'Exports to S3' section shows 0 exports and a search bar. The 'Amazon Kinesis data stream details' section shows the stream is 'On' with a destination stream named 'test' and a record timestamp precision of 'Microsecond'.

- (選擇性) 在 Amazon Kinesis 資料串流詳細資訊下，您可以將記錄時間戳記精確度從微秒 (預設) 變更為毫秒。
- 從下拉式清單選擇 samplestream (範例串流)。
- 選擇「開啟」按鈕。

## AWS CLI

- 使用 [create-stream 命令](#) 建立名為 samplestream 的 Kinesis 資料串流。

```
aws kinesis create-stream --stream-name samplestream --shard-count 3
```

請先參閱 [Kinesis Data Streams 的碎片管理考量事項](#)，再為 Kinesis 資料串流設定碎片數量。

- 使用 [describe-stream 命令](#) 確認 Kinesis 串流是否處於作用中狀態且可供使用。

```
aws kinesis describe-stream --stream-name samplestream
```

3. 使用 DynamoDB `enable-kinesis-streaming-destination` 命令在 DynamoDB 資料表上啟用 Kinesis 串流功能。將 `stream-arn` 值替換為上一個步驟中傳回的值 `describe-stream`。選擇性地啟用每筆記錄所傳回的時間戳記值的精確度更精確 (微秒) 的串流。

以微秒時間戳記精確度啟用串流：

```
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
```

或者使用默認時間戳記精度 (毫秒) 啟用流式傳輸：

```
aws dynamodb enable-kinesis-streaming-destination \
 --table-name Music \
 --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
```

4. 使用 DynamoDB `describe-kinesis-streaming-destination` 命令確認資料表上的 Kinesis 串流是否處於作用中狀態。

```
aws dynamodb describe-kinesis-streaming-destination --table-name Music
```

5. 使用 `put-item` 命令將資料寫入 DynamoDB 資料表，如《[DynamoDB 開發人員指南](#)》中所述。

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

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

6. 使用 Kinesis [get-records](#) CLI 命令來擷取 Kinesis 串流內容。然後使用下面的程式碼片段來將串流內容還原序列化。

```
/**
 * 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. 遵循《Kinesis Data Streams 開發人員指南》中的說明，使用 Java [建立](#)名為 samplestream 的 Kinesis 資料串流。



請先參閱 [Kinesis Data Streams 的碎片管理考量事項](#)，再為 Kinesis 資料串流設定碎片數量。

2. 使用以下程式碼片段來啟用 DynamoDB 資料表上的 Kinesis 串流。選擇性地啟用每筆記錄所傳回的時間戳記值的精確度更精確 (微秒) 的串流。

以微秒時間戳記精確度啟用串流：

```
EnableKinesisStreamingConfiguration enableKdsConfig =
 EnableKinesisStreamingConfiguration.builder()

 .approximateCreationDateTimePrecision(ApproximateCreationDateTimePrecision.MICROSECOND)
 .build();

EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
 .tableName(tableName)
 .streamArn(kdsArn)
 .enableKinesisStreamingConfiguration(enableKdsConfig)
 .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

或者使用默認時間戳記精度 ( 毫秒 ) 啟用流式傳輸：

```
EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
 .tableName(tableName)
 .streamArn(kdsArn)
 .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

3. 遵循《Kinesis Data Streams 開發人員指南》中的說明，從建立的資料串流中進行[讀取](#)。
4. 然後使用下面的程式碼片段將串流內容還原序列化

```
/**
 * 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());
}
```

## 變更使用中的 Amazon Kinesis 資料串流

本節說明如何使用主控台 AWS CLI 和 API 變更適用於 DynamoDB 的作用中 Kinesis Data Streams 設定。

### AWS Management Console

1. [開啟動 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>
2. 去你的桌子。
3. 選擇「匯出和串流」。

## AWS CLI

1. 呼叫describe-kinesis-streaming-destination以確認串流為ACTIVE。
2. 呼叫UpdateKinesisStreamingDestination，例如在此範例中：

```
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. 呼叫describe-kinesis-streaming-destination以確認串流為UPDATING。
4. describe-kinesis-streaming-destination定期呼叫，直到流狀態ACTIVE再次為止。時間戳記精確度更新通常需要 5 分鐘才會生效。一旦此狀態更新，表示更新已完成，而新的有效位數值將會套用至 future 的記錄。
5. 使用寫入表格putItem。
6. 使用 Kinesis get-records 指令取得串流內容。
7. 確認寫入ApproximateCreationDateTime的具有所需的精確度。

## 爪哇 API

1. 提供建構要UpdateKinesisStreamingDestination求和UpdateKinesisStreamingDestination回應的程式碼片段。
2. 提供建構DescribeKinesisStreamingDestination要求的程式碼片段和DescribeKinesisStreamingDestination response
3. describe-kinesis-streaming-destination定期呼叫，直到ACTIVE再次串流狀態為止，表示更新已完成，而新的精確度值將套用至 future 的記錄。
4. 執行對表格的寫入。
5. 從串流讀取並反序列化串流內容。
6. 確認寫入ApproximateCreationDateTime的具有所需的精確度。

## 在 DynamoDB 中使用 Kinesis Data Streams 設定碎片和監控變更資料擷取

### Kinesis Data Streams 的碎片管理考量事項

Kinesis 資料串流會計算其在[碎片](#)中的輸送量。在 Amazon Kinesis 資料串流中，您可以選擇資料串流的隨選模式和佈建模式。

如果 DynamoDB 寫入工作負載具有高度變化且無法預測，我們建議您針對 Kinesis 資料串流使用隨選模式。使用隨需模式時，不需要容量規劃，因為 Kinesis Data Streams 會自動管理碎片以提供必要的輸送量。

對於可預測的工作負載，您可以針對 Kinesis 資料串流使用佈建模式。在佈建模式下，您必須指定資料串流的碎片數目，以容納 DynamoDB 中的變更資料擷取記錄。若要判斷 Kinesis 資料串流支援 DynamoDB 表所需的碎片數目，您需要下列輸入值：

- DynamoDB 資料表紀錄的平均大小 (以位元組為單位) (`average_record_size_in_bytes`)。
- 您的 DynamoDB 資料表上將執行的每秒寫入操作數目上限。這包括應用程式執行的建立、刪除和更新作業，以及自動產生的作業，例如存留時間產生的刪除作業 (`write_throughput`)。
- 與建立或刪除操作相比，您在資料表上執行的更新和覆寫操作的百分比 (`percentage_of_updates`)。請注意，更新和覆寫操作會將已修改項目的舊映像和新映像複製到串流。因此產生兩倍大小的 DynamoDB 項目。

您可以使用下列公式中的輸入值來計算 Kinesis 資料串流所需的碎片數量 (`number_of_shards`)：

```
number_of_shards = ceiling(max(((write_throughput * (4+percentage_of_updates) * average_record_size_in_bytes) / 1024 / 1024), (write_throughput/1000)), 1)
```

例如，您的最大輸送量可能為每秒 1040 個寫入作業 (`write_throughput`)，平均記錄大小為 800 位元組 (`average_record_size_in_bytes`)。如果這些寫入作業中有 25% 是更新作業 (`percentage_of_updates`)，則您將需要兩個碎片 (`number_of_shards`) 來容納 DynamoDB 串流輸送量：

```
ceiling(max(((1040 * (4+25/100) * 800) / 1024 / 1024), (1040/1000)), 1).
```

使用公式計算 Kinesis 資料串流佈建模式所需的碎片數量之前，請考慮下列事項：

- 此公式有助於估計用於容納 DynamoDB 變更資料記錄所需的碎片數量。它不代表 Kinesis 資料串流中所需的碎片總數，例如支援其他 Kinesis 資料串流取用者所需的碎片數量。
- 如果您未設定處理尖峰輸送量的資料串流，則可能仍會在佈建模式中遇到讀取和寫入輸送量例外狀況。在這種情形下，您必須手動擴展資料串流以適應資料流量。
- 此公式會將變更記錄資料記錄串流至 Kinesis 資料串流之前，將 DynamoDB 產生的額外膨脹納入考量。

若要深入了解 Kinesis 資料串流上的容量模式，請參閱[選擇資料串流容量模式](#)。若要進一步了解不同容量模式之間的定價差異，請參閱 [Amazon Kinesis Data Streams 定價](#)。

## 使用 Kinesis Data Streams 監控變更資料擷取

DynamoDB 提供數個 Amazon CloudWatch 指標，可協助您監控變更資料擷取到 Kinesis 的複寫。如需 CloudWatch 量度的完整清單，請參閱[DynamoDB 指標和維度](#)。

為判斷串流是否有足夠的容量，建議您在啟用串流期間和生產環境期間監控下列項目：

- **ThrottledPutRecordCount**：Kinesis 資料串流容量不足而受到 Kinesis 資料串流限制的記錄數。儘管在特殊使用峰值期間，您可能會遇到調節的情況，仍應盡可能降低 **ThrottledPutRecordCount**。DynamoDB 會重試將調節記錄傳送到 Kinesis 資料串流，但這可能會導致較高的複寫延遲。

如果遇到過多且規律的調節，您可能需要按照觀察到的資料表寫入輸送量按比例增加 Kinesis 串流碎片的數量。若要進一步了解如何判斷 Kinesis 資料串流的大小，請參閱[判斷 Kinesis Data Stream 的初始大小](#)。

- **AgeOfOldestUnreplicatedRecord**：自 DynamoDB 資料表中出現尚未複寫到 Kinesis 資料串流的最舊項目層級變更以來經過的時間。在正常的操作下，**AgeOfOldestUnreplicatedRecord** 應該以毫秒為單位。當不成功的複寫嘗試是因客戶控制的組態選擇所引起時，此數字會隨著不成功複寫嘗試的增加而增加。

如果 **AgeOfOldestUnreplicatedRecord** 指標超過 168 小時，將自動停用從 DynamoDB 表格到 Kinesis 資料串流的項目層級變更複寫。

可能導致複寫嘗試失敗的客戶控制組態示例包括，佈建的 Kinesis 資料串流容量不足導致過度調節，或是手動更新 Kinesis 資料串流的存取原則因而拒絕 DynamoDB 新增資料至您的資料串流。為盡可能降低此指標，您可能需要確保妥善佈建您的 Kinesis 資料串流容量，並確保 DynamoDB 的許可保持不變。

- **FailedToReplicateRecordCount**：DynamoDB 無法複寫到您的 Kinesis 資料串流的記錄數目。大於 34KB 的某些項目可能會擴充大小，以變更大於 Kinesis Data Streams 1MB 項目大小限制的資料記錄。當這些大於 34KB 的項目包含大量的布林值或空白屬性值時，就會發生此大小擴充。布林值和空白屬性值會以 1 位元組形式儲存在 DynamoDB 中，但是在使用用於 Kinesis Data Streams 複寫的標準 JSON 將其序列化時，最多可擴充至 5 個位元組。DynamoDB 無法將這類變更記錄複寫到您的 Kinesis 資料串流。DynamoDB 會略過這些變更資料記錄，並自動繼續複寫後續記錄。

您可以建立 Amazon CloudWatch 警示，當前任何指標超過特定閾值時，傳送 Amazon 簡單通知服務 (Amazon SNS) 訊息以進行通知。

## 使用 Amazon Kinesis Data Streams 和 Amazon DynamoDB 專用的 IAM 政策

當您第一次啟用 Amazon Kinesis Data Streams 時，Amazon DynamoDB oDB 會自動為您建立 AWS Identity and Access Management (IAM) 服務連結角色。這個角色 (AWSServiceRoleForDynamoDBKinesisDataStreamsReplication) 可讓 DynamoDB 代表您管理對 Kinesis Data Streams 的項目層級變更的複寫。請勿刪除此服務連結角色。

如需服務連結角色的詳細資訊，請參閱 IAM 使用者指南中的 [使用服務連結角色](#)。

若要啟用 Amazon DynamoDB 專用 Amazon Kinesis Data Streams，您必須擁有下列資料表許可：

- dynamodb:EnableKinesisStreamingDestination
- kinesis:ListStreams
- kinesis:PutRecords
- kinesis:DescribeStream

若要為指定的 DynamoDB 資料表描述 Amazon DynamoDB 專用 Amazon Kinesis Data Streams，您必須擁有下列資料表許可。

- dynamodb:DescribeKinesisStreamingDestination
- kinesis:DescribeStreamSummary
- kinesis:DescribeStream

若要停用 Amazon DynamoDB 專用 Amazon Kinesis Data Streams，您必須擁有下列資料表許可。

- dynamodb:DisableKinesisStreamingDestination

若要更新適用於 Amazon DynamoDB 的 Amazon Kinesis 資料串流，您必須在表格上具有下列許可。

- dynamodb:UpdateKinesisStreamingDestination

下列範例顯示如何使用 IAM 政策授予 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 的許可。

## 範例：啟用 Amazon DynamoDB 專用 Amazon Kinesis Data Streams

下列身分與存取權管理政策授予許可，以便為表格啟用亞馬 Amazon Kinesis Data Streams。Music 它不會授與停用、更新或描述表格適用於 DynamoDB 的 Kinesis Data Streams 的權限。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"
 }
]
}
```

## 範例：更新亞馬遜運動資料串流

下列身分與存取權管理政策授予許可，以便更新表格的 Amazon Kinesis Data Streams。Music 它不會授予對表格啟用、停用或描述適用於 Amazon DynamoDB 的 Amazon Kinesis 資料串流的許可。Music

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "dynamodb:UpdateKinesisStreamingDestination"
]
 }
]
}
```

```

],
 "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
 }
]
}

```

### 範例：停用 Amazon DynamoDB 專用 Amazon Kinesis Data Streams

下列身分與存取權管理政策授予許可，以針對表格停用亞馬 Amazon Kinesis Data Streams。Music 它不會授予對表格啟用、更新或描述適用於 Amazon DynamoDB 的 Amazon Kinesis 資料串流的許可。Music

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "dynamodb:DisableKinesisStreamingDestination"
],
 "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
 }
]
}

```

### 範例：根據資源選擇性地為 Amazon DynamoDB 專用 Amazon Kinesis Data Streams 套用許可

下列 IAM 政策授予許可，以便為表格啟用和描述適用於 Amazon DynamoDB 的 Amazon Amazon Kinesis Data Streams，並拒絕對 Music 表格停用 Amazon Kinesis Data Streams 的許可。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"
 }
}
]
```

為 DynamoDB 專用 Kinesis Data Streams 使用服務連結角色

[亞馬遜運動資料串流使用 AWS Identity and Access Management \(IAM\) 服務連結角色](#)。服務連結角色是特殊類型的 IAM 角色，此角色可直接連結到 DynamoDB 專用 Kinesis Data Streams。服務連結角色是由適用於 DynamoDB 的 Kinesis Data Streams 預先定義，並包含服務代表您呼叫其他 AWS 服務所需的所有權限。

服務連結角色可讓 DynamoDB 專用 Kinesis Data Streams 的設定更為簡單，因為您不必手動新增必要的許可。DynamoDB 專用 Kinesis Data Streams 會定義其服務連結角色的許可，除非另有定義，否則僅有 DynamoDB 專用 Kinesis Data Streams 可以擔任其角色。定義的許可包括信任政策和許可政策，並且該許可政策不能連接到任何其他 IAM 實體。

如需關於支援服務連結角色的其他服務的資訊，請參閱[可搭配 IAM 運作的 AWS 服務](#)，並尋找 Service-Linked Role (服務連結角色) 欄顯示為 Yes (是) 的服務。選擇具有連結的 Yes (是)，以檢視該服務的服務連結角色文件。

DynamoDB 專用 Kinesis Data Streams 的服務連結角色許可

適用於 DynamoDB 的 Kinesis Data Streams 使用名為的服務連結角色。AWSServiceRoleForDynamoDBKinesisDataStreamsReplication 服務連結角色的目的是讓 Amazon DynamoDB 代表您管理對 Kinesis Data Streams 項目層級變更的複寫。

AWSServiceRoleForDynamoDBKinesisDataStreamsReplication 服務連結角色信任下列服務以擔任角色：

- `kinesisreplication.dynamodb.amazonaws.com`

此角色許可政策允許 DynamoDB 專用 Kinesis Data Streams 對指定資源完成下列動作：

- 動作：Kinesis stream 上的 Put records and describe

- 動作：開Generate data keys啟以將資料放AWS KMS在使用使用者產生 AWS KMS 金鑰加密的 Kinesis 串流上。

如需政策文件的確切內容，請參閱 [DynamoDB KinesisReplicationServiceRolePolicy](#)。

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的[服務連結角色許可](#)。

為 DynamoDB 專用 Kinesis Data Streams 建立服務連結角色

您不需要手動建立一個服務連結角色。當您在、或 AWS API 中啟用適用於 DynamoDB 的 Kinesis Data Streams 時 AWS Management Console，適用於 DynamoDB 的 Kinesis Data Streams 會為您建立服務連結角色。AWS CLI

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。在啟用 DynamoDB 專用 Kinesis Data Streams 時，DynamoDB 專用 Kinesis Data Streams 會再次為您建立服務連結角色。

為 DynamoDB 專用 Kinesis Data Streams 編輯服務連結角色

DynamoDB 專用 Kinesis Data Streams 不允許您編輯

AWSServiceRoleForDynamoDBKinesisDataStreamsReplication 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的[編輯服務連結角色](#)。

為 DynamoDB 專用 Kinesis Data Streams 刪除服務連結角色

您也可以使用 IAM 主控台 AWS CLI 或 AWS API 手動刪除服務連結角色。若要執行此操作，您必須先手動清除服務連結角色的資源，然後才能手動刪除它。

#### Note

若 DynamoDB 專用 Kinesis Data Streams 服務在您試圖刪除資源時正在使用該角色，刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

使用 IAM 手動刪除服務連結角色

使用 IAM 主控台或 AWS API 刪

除AWSServiceRoleForDynamoDBKinesisDataStreamsReplication服務連結角色。AWS CLI 如需詳細資訊，請參閱《IAM 使用者指南》中的[刪除服務連結角色](#)。

## DynamoDB Streams 的變更資料擷取

DynamoDB Streams 可擷取任何 DynamoDB 資料表中依時間順序排序的項目層級修改，並將此資訊存放於日誌中長達 24 小時。應用程式可以存取此日誌，並近乎即時地檢視資料項目修改前後的顯示內容。

靜態加密功能會加密 DynamoDB Streams 中的資料。如需詳細資訊，請參閱 [DynamoDB 靜態加密](#)。

DynamoDB 串流是 DynamoDB 資料表中項目變更資訊的排序流程。當您在資料表啟用串流時，DynamoDB 會擷取資料表中資料項目的每項修改資訊。

應用程式每次建立、更新或刪除資料表中的項目時，DynamoDB Streams 都會使用已修改項目的主索引鍵屬性寫入串流紀錄。串流紀錄包含 DynamoDB 資料表中單一項目資料修改的相關資訊。您可以設定串流，讓串流紀錄擷取其他資訊，例如修改項目的「之前」與「之後」影像。

DynamoDB Streams 可協助確保下列事項：

- 每個串流紀錄只在串流中出現一次。
- 對於 DynamoDB 資料表中的每個修改項目，串流紀錄的出現順序與項目的實際修改順序相同。

DynamoDB Streams 會近乎即時地寫入串流紀錄，讓您可以建立使用這些串流並根據內容採取動作的應用程式。

### 主題

- [DynamoDB Streams 的端點](#)
- [啟用串流](#)
- [讀取及處理串流](#)
- [DynamoDB Streams 和存留時間](#)
- [使用 DynamoDB Streams Kinesis 轉接器處理串流記錄](#)
- [DynamoDB Streams 低階 API : Java 範例](#)
- [DynamoDB Streams 和觸發器 AWS Lambda](#)

## DynamoDB Streams 的端點

AWS 為 DynamoDB 和動態資料庫 DynamoDB Streams 維護個別的端點。為了使用資料庫資料表與索引，應用程式必須能存取 DynamoDB 端點。為了讀取及處理 DynamoDB Streams 紀錄，應用程式必須能存取同一區域中的 DynamoDB Streams 端點。

DynamoDB Streams 端點的命名慣用格式為 `streams.dynamodb.<region>.amazonaws.com`。例如，如果使用端點 `dynamodb.us-west-2.amazonaws.com` 存取 DynamoDB，則將使用端點 `streams.dynamodb.us-west-2.amazonaws.com` 存取 DynamoDB Streams。

#### Note

如需 DynamoDB 和 DynamoDB Streams 區域與端點的完整清單，請參閱 AWS 一般參考中的 [區域與端點](#)。

開 AWS 發套件會為 DynamoDB 和動態資料庫 DynamoDB Streams 提供個別的用户端。視您需求的不同，應用程式可以存取 DynamoDB 端點、DynamoDB Streams 端點，或同時存取這兩種端點。若要連線到這兩種端點，您的應用程式必須具體化兩個用户端：一個用於 DynamoDB，另一個用於 DynamoDB Streams。

## 啟用串流

當您使用 AWS CLI 或其中一個 AWS SDK 建立新資料表時，您可以在新資料表上啟用串流。您也可以在此現有的資料表上啟用或停用串流，或變更串流的設定。DynamoDB Streams 會以非同步方式運作，因此若您啟用串流，也不會影響資料表的效能。

管理 DynamoDB Streams 最簡單的方式就是使用 AWS Management Console。

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在 DynamoDB 主控台儀表板上，選擇 Tables (資料表)，然後選取現有的資料表。
3. 選擇 Exports and streams (匯出與串流) 索引標籤。
4. 在 DynamoDB 串流詳細資料區段中，選擇開啟。
5. 在 [開啟 DynamoDB 串流] 頁面上，選擇修改表格中的資料時要寫入串流的資訊：
  - 僅限金鑰屬性：僅已修改項目的金鑰屬性。
  - 新映像：在修改後出現的整個項目。
  - 舊映像：在修改前出現的整個項目。
  - 新舊映像：項目的新舊映像。

如果設定符合您的需求，請選擇 [開啟串流]。

6. (選擇性) 若要停用現有串流，請選擇 DynamoDB 串流詳細資料下的關閉。

您也可以使用 `CreateTable` 或 `UpdateTable` API 操作來啟用或修改串流。`StreamSpecification` 參數可決定串流的設定方式：

- `StreamEnabled`：指定是否啟用 (`true`) 或停用 (`false`) 資料表的串流。
- `StreamViewType`：指定每次修改資料表資料時，將會寫入串流的資訊：
  - `KEYS_ONLY`：僅已修改項目的索引鍵屬性。
  - `NEW_IMAGE`：在修改後出現的整個項目。
  - `OLD_IMAGE`：在修改前出現的整個項目。
  - `NEW_AND_OLD_IMAGES`：項目的新舊映像。

您可以隨時啟用或停用串流。但若嘗試在已有串流的資料表中啟用串流，就會收到 `ResourceInUseException`。若嘗試在沒有串流的資料表中停用串流，就會收到 `ValidationException`。

當您將 `StreamEnabled` 設定為 `true` 時，DynamoDB 會建立新的串流，並對其指派唯一的串流描述項。如果您停用再重新啟用資料表串流，則會使用不同的串流描述項建立新的串流。

每個串流都可依 Amazon 資源名稱 (ARN) 唯一識別。以下是名為 `TestTable` 的 DynamoDB 資料表上串流的 ARN 範例。

```
arn:aws:dynamodb:us-west-2:111122223333:table/TestTable/stream/2015-05-11T21:21:33.291
```

若要判斷資料表的最新串流描述項，請發出 `DynamoDB DescribeTable` 請求，並尋找回應中的 `LatestStreamArn` 元素。

#### Note

一旦串流設定完畢，就無法再編輯 `StreamViewType`。若您需要變更已設定完畢的串流，就必須停用目前的串流並建立一個新的串流。

## 讀取及處理串流

若要讀取及處理串流，您的應用程式必須連線到 DynamoDB Streams 端點，並發出 API 請求。

串流由串流紀錄所組成。每個串流紀錄均代表串流所屬之 DynamoDB 資料表中的一項資料修改。每個串流紀錄會獲派一個序號，其反映出紀錄發佈至串流的順序。

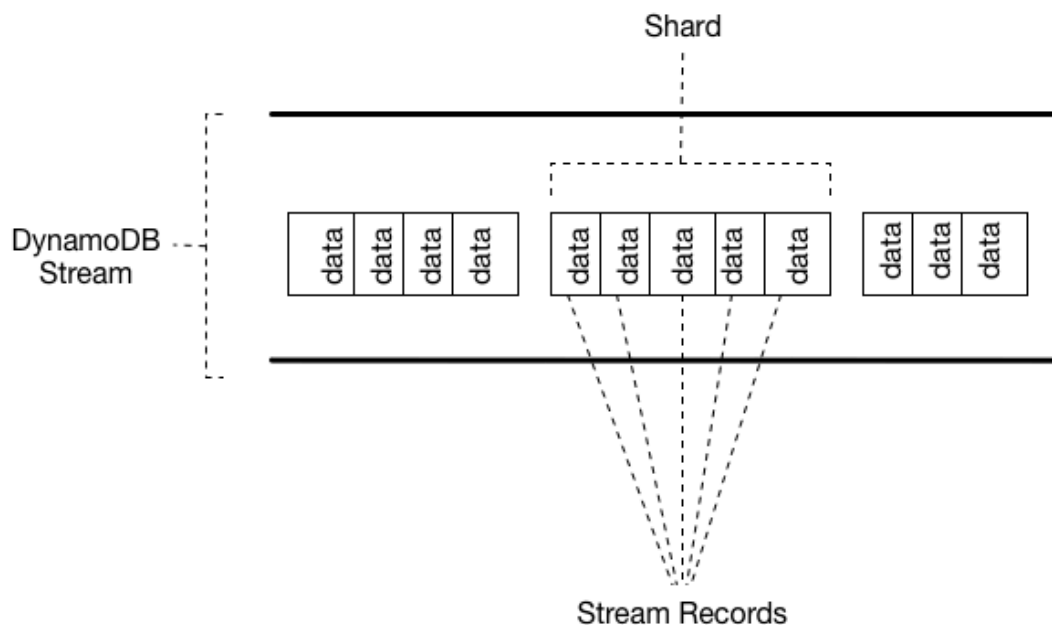
串流紀錄會整理成群組，即碎片。每個碎片皆代表多個串流紀錄的容器，並含有存取及重複處理這些紀錄所需的資訊。碎片內的串流紀錄會在 24 小時後自動移除。

碎片為暫時性：系統會視需要自動建立及刪除這些碎片。任何碎片還可分割成多個新的碎片，這也會自動發生 (父碎片也可能只有一個子碎片。) 碎片可能會為了回應其父資料表上的上層寫入活動而分割，讓應用程式可同時處理多個碎片中的紀錄。

如果您停用串流，所有開啟的碎片都會關閉。串流中的資料在 24 小時內仍可供讀取。

由於碎片具有系屬關係 (父系與子系)，因此應用程式一律必須先處理父碎片，再處理子碎片。這有利於確保串流紀錄也按照正確順序處理。(如果您使用的是 DynamoDB Streams Kinesis 轉接器，則系統會為您處理這個問題。應用程式會依正確順序處理碎片和串流紀錄。除了當應用程式正在執行時分割的碎片，其還會自動處理新的或過期碎片。如需詳細資訊，請參閱 [使用 DynamoDB Streams Kinesis 轉接器處理串流記錄](#)。)

下圖說明串流、串流中的碎片及碎片中的串流紀錄之間的關係。



#### Note

如果您執行的 `PutItem` 或 `UpdateItem` 操作不會變更項目中的任何資料，則 DynamoDB Streams 不會寫入該操作的串流紀錄。

若要存取串流及處理其中的串流紀錄，您必須執行下列操作：

- 判斷您要存取之串流的唯一 ARN。
- 判斷串流中包含您感興趣之串流紀錄的碎片。
- 存取碎片及擷取您要的串流紀錄。

#### Note

最多不得同時從同一個串流碎片讀取 2 個以上的處理程序。每個碎片具有 2 個以上的讀取器會導致調節。

DynamoDB Streams API 提供下列可讓應用程式使用的動作：

- [ListStreams](#)：傳回目前帳戶及端點的串流描述項清單。您可以只選擇性地請求特定資料表名稱的串流描述項。
- [DescribeStream](#)：傳回指定串流的詳細資訊。輸出包含與串流相關聯的碎片清單，包括碎片 ID。
- [GetShardIterator](#)：傳回碎片疊代運算，其描述了碎片內的位置。您可以請求疊代運算提供串流中最舊點、最新點或特定點的存取權。
- [GetRecords](#)：傳回指定碎片內的串流紀錄。您必須提供從 [GetShardIterator](#) 請求傳回的碎片疊代運算。

如需這些 API 操作的完整說明 (包括範例請求與回應)，請參閱《[Amazon DynamoDB Streams API 參考](#)》。

## DynamoDB Streams 的資料保留限制

DynamoDB Streams 中所有資料的生命週期皆為 24 小時。您可以擷取並分析任何指定資料表過去 24 小時的活動。但超過 24 小時的資料容易隨時受到裁剪 (移除) 的影響。

如果您停用資料表的串流，則串流中的資料在 24 小時內仍可供讀取。過了這段時間後，資料就會過期，且串流紀錄也會自動刪除。沒有手動刪除現有串流的機制。您必須等到保留限制過期 (24 小時)，所有串流紀錄才會被刪除。

## DynamoDB Streams 和存留時間

您可以備份或處理 [存留時間](#) (TTL) 刪除的項目，方法是在資料表上啟用 Amazon DynamoDB Streams 並處理過期項目的串流紀錄。如需詳細資訊，請參閱 [讀取及處理串流](#)。



串流紀錄包含使用者身分欄位 `Records[<index>].userIdentity`。

存留時間程序在過期後刪除的項目具有下列欄位：

- `Records[<index>].userIdentity.type`  
"Service"
- `Records[<index>].userIdentity.principalId`  
"dynamodb.amazonaws.com"

### Note

當您在全域表中使用 TTL 時，執行 TTL 的區域將會設定 `userIdentity` 欄位。複寫刪除作業時，不會在其他區域設定此欄位。

下列 JSON 顯示單一串流紀錄的相關部分。

```
"Records": [
 {
 ...
 "userIdentity": {
 "type": "Service",
 "principalId": "dynamodb.amazonaws.com"
 }
 ...
 }
]
```

使用 DynamoDB Streams 和 Lambda 封存 TTL 已刪除的項目

合併 [DynamoDB 存留時間 \(TTL\)](#)、[DynamoDB Streams](#) 和 [AWS Lambda](#) 可協助簡化封存資料、降低 DynamoDB 儲存成本並降低程式碼複雜性。使用 Lambda 做為串流使用者具有許多優勢，最顯著的是與其他使用者 (例如 Kinesis Client Library (KCL)) 相比，成本降低。透過 Lambda 使用事件時，您不會因為 DynamoDB 串流上的 `GetRecords` API 呼叫而需要付費，而且 Lambda 可藉由識別串流事件中的 JSON 模式提供事件篩選功能。透過事件模式內容篩選，您最多可以定義五個不同的篩選條件來控



制哪些事件要傳送到 Lambda 進行處理。這有助於減少對 Lambda 函數的叫用、簡化程式碼並降低總體成本。

雖然 DynamoDB Streams 包含所有資料修改，例如 Create、Modify 和 Remove 動作，這可能會導致不必要的叫用封存 Lambda 函數。例如，假設有一個資料表每小時有 200 萬個資料修改流入串流中，但其中不到 5% 的項目刪除將在 TTL 程序中過期且需要進行封存。搭配 [Lambda 事件來源篩選條件](#)，Lambda 函數每小時只會叫用 100,000 次。使用事件篩選，您只需為所需的叫用付費，而不是在沒有事件篩選的情況下為 200 萬次的叫用付費。

事件篩選套用至 [Lambda 事件來源映射](#)，它是一種資源，可從選定的事件 (DynamoDB 串流) 中讀取並叫用 Lambda 函數。在下圖中，您可以看到 Lambda 函數如何透過串流和事件篩選條件使用存留時間已刪除的項目。



## DynamoDB 存留時間事件篩選條件模式

將以下 JSON 新增至事件來源映射 [篩選條件](#)，可僅對 TTL 刪除的項目叫用 Lambda 函數：

```
{
 "Filters": [
 {
 "Pattern": { "userIdentity": { "type": ["Service"], "principalId":
["dynamodb.amazonaws.com"] } }
 }
]
}
```

## 建立 AWS Lambda 事件來源對應

使用以下程式碼片段建立已篩選的事件來源映射，您可以將它連接到資料表的 DynamoDB 串流。每個程式碼區塊都包含事件篩選條件模式。

## 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"]
 }
}
```

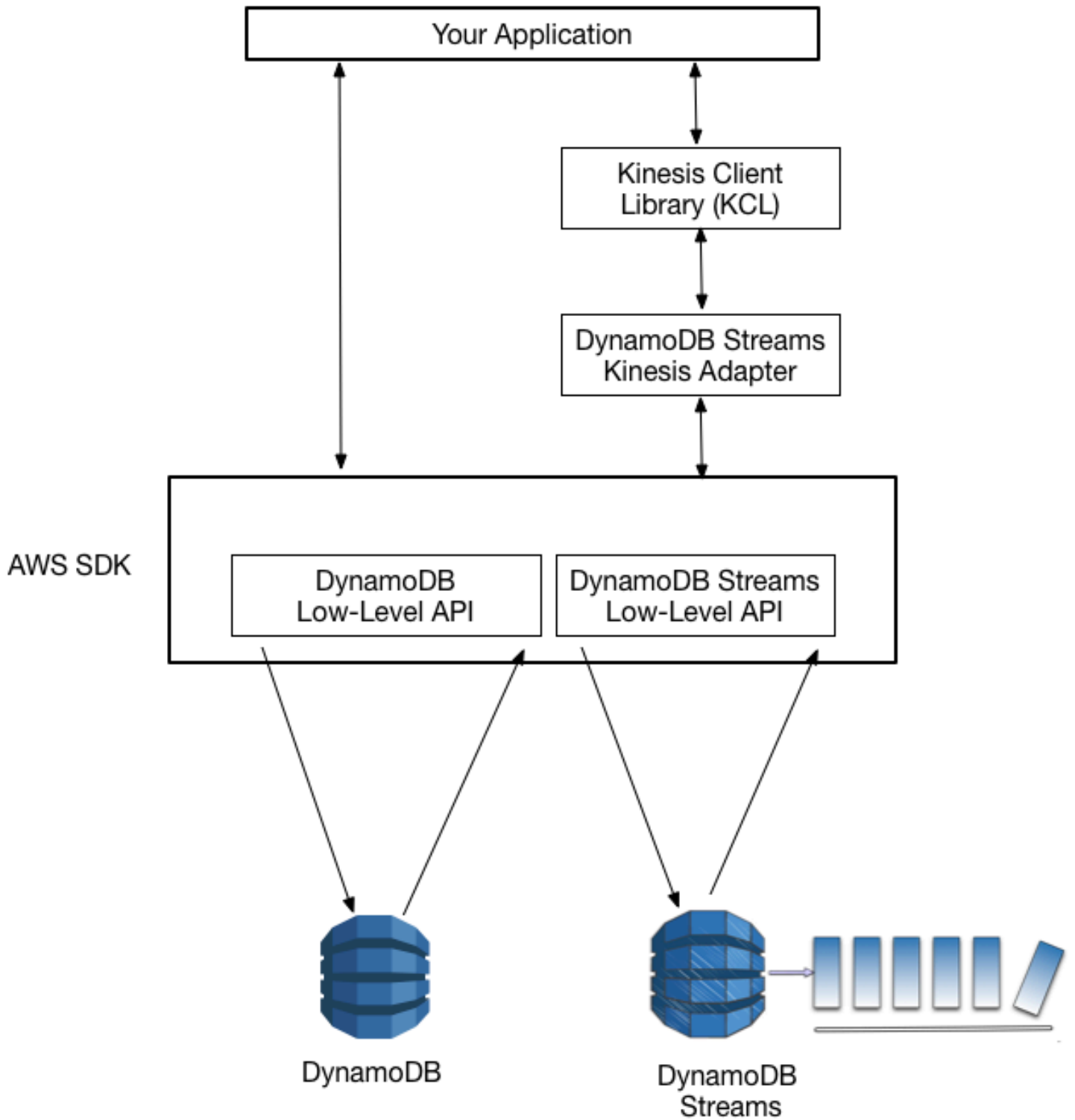
## 使用 DynamoDB Streams Kinesis 轉接器處理串流記錄

建議透過 Amazon Kinesis 轉接器耗用來自 Amazon DynamoDB 的串流。DynamoDB Streams API 與 Kinesis Data Streams API 相類似是刻意為之，後者是可即時處理大規模串流資料的服務。在這兩種服務中，資料串流由碎片組成，碎片是用於串流紀錄的容器。這兩種服務的 API 都包含 ListStreams、DescribeStream、GetShards 以及 GetShardIterator 操作。(雖然這些 DynamoDB Streams 動作與其在 Kinesis Data Streams 中的對應動作類似，但它們並非完全相同。)

您可以使用 Kinesis Client Library (KCL) 為 Kinesis Data Streams 撰寫應用程式。KCL 會在低階 Kinesis Data Streams API 上提供有用的抽象，可以簡化程式碼。如需 KCL 的詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的[使用 Kinesis Client Library 開發消費者](#)。

身為 DynamoDB Streams 使用者，您可以使用在 KCL 內找到的設計模式來處理 DynamoDB Streams 碎片和串流紀錄。為此，您可以使用 DynamoDB Streams Kinesis 轉接器。Kinesis 轉接器會實作 Kinesis Data Streams 界面，以便您將 KCL 用於耗用和處理來自 DynamoDB Streams 的紀錄。[如需有關如何設定和安裝 DynamoDB Streams Kinesis 動配接器的指示，請參閱存放庫。GitHub](#)

下圖顯示這些程式庫彼此如何互動。



有了 DynamoDB Streams Kinesis 轉接器，您就可以開始針對 KCL 界面進行開發，並將 API 呼叫順暢地導向 DynamoDB Streams 端點。

當應用程式啟動後，其會呼叫 KCL 以將工作者執行個體化。您必須為 Worker 提供應用程式的組態資訊，例如串流描述元和 AWS 認證，以及您所提供之記錄處理器類別的名稱。當其在記錄處理器中執行程式碼時，工作者會執行下列任務：

- 連線到串流
- 列舉串流內的碎片
- 與其他工作者 (若有) 協調碎片關聯性
- 為其所管理的每個碎片執行個體化記錄處理器
- 從串流提取紀錄
- 將記錄推送至對應的記錄處理器
- 對已處理的記錄執行檢查點作業
- 當工作者執行個體數目變更時，平衡碎片與工作者的關聯
- 當碎片進行分割時，平衡碎片與工作者的關聯

#### Note

如需此處所列 KCL 概念的描述，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的 [使用 Kinesis Client Library 開發消費者](#)。

如需使用串流的詳細資訊，AWS Lambda 請參閱 [DynamoDB Streams 和觸發器 AWS Lambda](#)

### 逐步解說：DynamoDB Streams Kinesis 轉接器

本節會逐步解說使用 Amazon Kinesis Client Library 與 Amazon DynamoDB Streams Kinesis 轉接器的 Java 應用程式。此應用程式示範資料複寫的範例，將一份資料表的寫入活動套用至第二份資料表，讓兩份資料表的內容保持同步。如需來源碼，請參閱「[完整程式：DynamoDB Streams Kinesis 轉接器](#)」。

此程式執行下列操作：

1. 建立兩個 DynamoDB 資料表，並命名為 KCL-Demo-src 和 KCL-Demo-dst。這些資料表各會啟用串流。
2. 在來源資料表中新增、更新與刪除項目，以產生更新活動。這會導致將資料寫入資料表的串流。
3. 從串流讀取紀錄、將其重新建構為 DynamoDB 請求，再將請求套用至目標資料表。
4. 掃描來源與目標資料表，以確定其內容相同。

## 5. 刪除資料表以清除。

下列章節將說明這些步驟，並在演練結尾顯示完整的應用程式。

### 主題

- [步驟 1：建立 DynamoDB 資料表](#)
- [步驟 2：在來源資料表中產生更新活動](#)
- [步驟 3：處理串流](#)
- [步驟 4：確定這兩個資料表的內容相同](#)
- [步驟 5：清除](#)
- [完整程式：DynamoDB Streams Kinesis 轉接器](#)

### 步驟 1：建立 DynamoDB 資料表

第一步是建立兩個 DynamoDB 資料表：一個來源資料表與一個目標資料表。來源資料表串流中的 `StreamViewType` 是 `NEW_IMAGE`。這表示每當在此資料表中修改項目時，項目的「修改後」影像就會寫入串流。串流可透過此方法追蹤資料表上的所有寫入活動。

下列範例顯示用來建立這兩份資料表的程式碼。

```
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)
```

## 步驟 2：在來源資料表中產生更新活動

下一步是在來源資料表上產生一些寫入活動。在此活動進行期間，來源資料表的串流也會近乎即時地更新。

此應用程式會使用呼叫 PutItem、UpdateItem 與 DeleteItem API 操作的方法，來定義小幫手類別，以寫入資料。下列程式碼範例示範如何使用這些方法。

```
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");
```

## 步驟 3：處理串流

程式現在會開始處理串流。DynamoDB Streams Kinesis 轉接器會作為 KCL 與 DynamoDB Streams 端點之間的透明層，讓程式碼可以充分利用 KCL，而不需要發出低階 DynamoDB Streams 呼叫。此程式會執行下列任務：

- 它會以遵守 KCL 界面定義的方法 (StreamsRecordProcessor、initialize 與 processRecords) 來定義紀錄處理器類別 shutdown。processRecords 方法包含從來源資料表串流讀取與寫入目標資料表所需的邏輯。
- 它會定義紀錄處理器類別的類別處理站 (StreamsRecordProcessorFactory)。這是使用 KCL 之 Java 程式的必要任務。
- 它會執行個體化與類別處理站相關聯的新 KCL Worker。
- 它會在紀錄處理完成時關閉 Worker。

若要進一步了解 KCL 界面定義，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的[使用 Kinesis Client Library 開發消費者](#)。



下列程式碼範例顯示 StreamsRecordProcessor 中的主迴圈。case 陳述式會根據串流紀錄中顯示的 OperationType 來決定要執行的動作。

```
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().getNewItem());
 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();
 }
 }
}
```

步驟 4：確定這兩個資料表的內容相同

此時，來源與目標資料表的內容會同步。此應用程式會對這兩個資料表發出 Scan 請求，以確認其內容事實上相同。

DemoHelper 類別包含呼叫低階 Scan API 的 ScanTable 方法。下列範例示範其使用方法。

```
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!");
}
```

## 步驟 5：清除

示範已完成，因此應用程式會刪除來源與目標資料表。請參閱以下程式碼範例。即使在刪除資料表後，其串流也會保持可用長達 24 小時，然後就自動刪除。

```
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
```

## 完整程式：DynamoDB Streams Kinesis 轉接器

以下是執行此[逐步解說：DynamoDB Streams Kinesis 轉接器](#)所述之任務的完整 Java 程式。當您執行它時，應該會看到類似如下的輸出。

```
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

若要執行此程式，請確保用戶端應用程式可以 CloudWatch 使用政策存取 DynamoDB 和 Amazon。如需詳細資訊，請參閱 [適用於 DynamoDB 的以身分為基礎的政策](#)。

此來源碼包含四個 .java 檔案：

- StreamsAdapterDemo.java

- StreamsRecordProcessor.java
- StreamsRecordProcessorFactory.java
- StreamsAdapterDemoHelper.java

## StreamsAdapterDemo. 爪哇

```
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. 爪哇

```
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().getNewItem());
 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. 爪哇

```
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. 爪哇

```
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);
 }
}
```

## DynamoDB Streams 低階 API : Java 範例

### Note

此頁面上的程式碼並不詳盡，並且無法應對使用 Amazon DynamoDB Streams 的所有案例。從 DynamoDB 使用串流紀錄的建議方式，是透過使用 Kinesis Client Library (KCL) 的 Amazon Kinesis 轉接器，如 [使用 DynamoDB Streams Kinesis 轉接器處理串流記錄](#) 中所述。

本節包含顯示作用中 DynamoDB Streams 的 Java 程式。此程式執行下列操作：

1. 建立啟用串流的 DynamoDB 資料表。
2. 說明此資料表的串流設定。
3. 修改資料表中的資料。
4. 描述串流中的碎片。
5. 讀取碎片中的串流紀錄。
6. 清除。

當您執行此程式時，會看到與下面類似的輸出。

```
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:
 100000000003218256368,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: 200000000003218256412,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:
 300000000003218256413,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 和觸發器 AWS Lambda

### 主題

- [教學 #1: 使用篩選器來處理 Amazon DynamoDB 的所有事件，並 AWS Lambda 使用 AWS CLI](#)
- [教學課程 #2：使用篩選條件來處理 Amazon DynamoDB 和 Lambda 的所有事件](#)
- [Lambda 的最佳實務](#)

Amazon DynamoDB 已與整合，AWS Lambda 因此您可以建立觸發器 — 可自動回應 DynamoDB Streams 中事件的程式碼片段。您可以利用觸發條件建立對 DynamoDB 資料表資料修改做出反應的應用程式。

如果您在表格上啟用 DynamoDB Streams，則可以將串流 Amazon 資源名稱 (ARN) 與您寫入的 AWS Lambda 函數產生關聯。然後，即可將該 DynamoDB 資料表的所有變動動作擷取為串流上的項目。例如，您可以設定觸發條件，以便在修改資料表中的項目時，新記錄會立即顯示在該資料表的串流中。

**Note**

您可以訂閱兩個以上的 Lambda 函數。如果您為一個 DynamoDB 串流訂閱兩個以上的 Lambda 函數，則可能會發生讀取節流。

此 [AWS Lambda](#) 服務會以每秒四次的速度輪詢串流以偵測新記錄。當有新的串流記錄可用時，系統會同步叫用 Lambda 函數。對於相同的 DynamoDB 串流，最多可以訂閱兩個 Lambda 函數。如果您訂閱兩個以上的 Lambda 函數到相同的 DynamoDB 串流，可能會發生讀取節流。

Lambda 函數可以傳送通知、啟動工作流程，或執行您指定的其他許多動作。您可以撰寫 Lambda 函數僅將每筆串流紀錄複製到持久性儲存，例如 Amazon S3 File Gateway (Amazon S3)，並在您的資料表中建立永久的寫入活動稽核追蹤。或者，假設您有一個會寫入 GameScores 表的手機遊戲應用程式。每當 TopScore 資料表的 GameScores 屬性更新時，對應的串流紀錄就會寫入資料表串流。然後，這個事件就會觸發在社交媒體網路張貼賀電的 Lambda 函數。(此函數也可以編寫為忽略所有非更新至 GameScores 或不修改 TopScore 屬性的串流紀錄)。

如果函數傳回錯誤，Lambda 會不斷重試批次直到處理成功或資料過期。您還可以設定 Lambda，使用較小批次重試、限制重試次數、在記錄太舊時捨棄，以及其他選項。

作為效能最佳實務，Lambda 函數必須為短期函數。為了避免產生不必要的處理延遲，此函數也不應執行複雜的邏輯。尤其對於高速串流而言，相較於同步長時間執行的 Lambda，觸發非同步後續處理 Step Function 工作流程是較佳的做法。

您無法在不同 AWS 帳戶中使用相同的 Lambda 觸發器。DynamoDB 表和 Lambda 函數都必須屬於相同 AWS 的帳戶。

如需詳細資訊 AWS Lambda，請參閱[AWS Lambda 開發人員指南](#)。

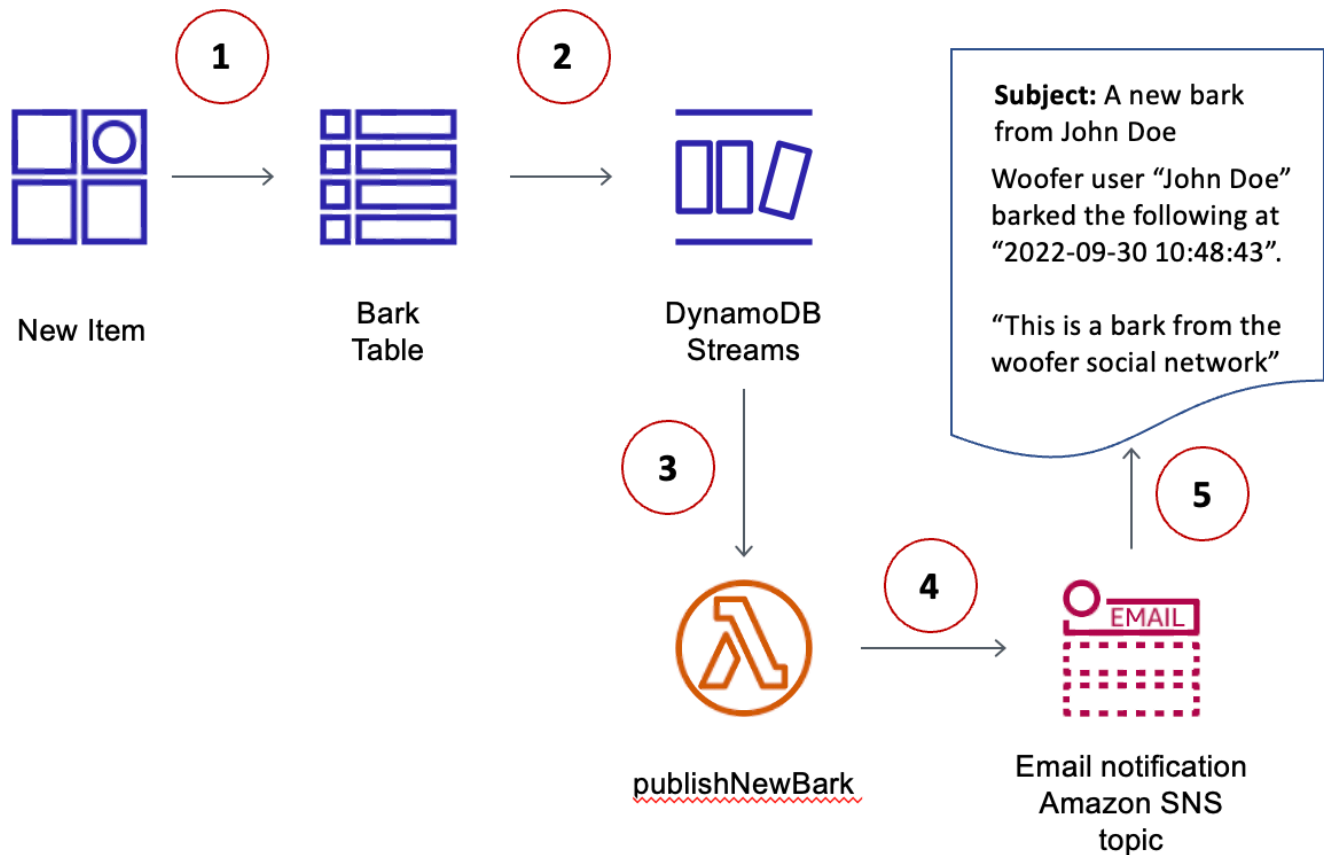
教學 #1: 使用篩選器來處理 Amazon DynamoDB 的所有事件，並 AWS Lambda 使用 AWS CLI

**主題**

- [步驟 1：建立啟用串流的 DynamoDB 資料表](#)
- [步驟 2：建立 Lambda 執行角色](#)
- [步驟 3：建立 Amazon SNS 主題](#)
- [步驟 4：建立並測試 Lambda 函數](#)
- [步驟 5：建立並測試觸發器](#)

在本教學課程中，您將建立 AWS Lambda 觸發器來處理 DynamoDB 表格的串流。

本教學案例為 Woofers，簡易的社交網路。Woofers 使用者使用互相傳送的 barks (簡短的文字訊息) 進行通訊。下圖顯示此應用程式的元件和工作流程。



1. 使用者將某個項目寫入 DynamoDB 資料表 (BarkTable)。資料表中的每個項目代表一個 bark。
2. 寫入新串流紀錄即反映新的項目已新增至 BarkTable。
3. 新的流紀錄觸發一個 AWS Lambda 函數 ( `publishNewBark` )。
4. 如果該串流紀錄指出新的項目已新增至 BarkTable，則 Lambda 函數會從串流紀錄讀取資料，將訊息發佈至 Amazon Simple Notification Service (Amazon SNS) 中的主題。
5. Amazon SNS 主題的訂閱者會收到此訊息。(在此教學中，唯一的訂閱者是電子郵件地址)。

## 開始之前

本自學課程使用 AWS Command Line Interface AWS CLI。若您尚未執行此作業，請遵循 [《AWS Command Line Interface 使用者指南》](#) 中的說明安裝及設定 AWS CLI。

## 步驟 1：建立啟用串流的 DynamoDB 資料表

在此步驟中，您會建立 DynamoDB 資料表 (BarkTable) 存放 Woofers 使用者的所有 bark。主索引鍵包含 Username (分割區索引鍵) 和 Timestamp (排序索引鍵)。這些屬性的類型皆為字串。

BarkTable 已啟用串流。稍後在本教學課程中，您會透過將 AWS Lambda 函數與串流產生關聯來建立觸發程序。

1. 輸入下列命令建立資料表。

```
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. 在輸出中，尋找 LatestStreamArn。

```
...
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

記下 *region* 和 *accountID*，因為您在本教學的其他步驟中會需要它們。

## 步驟 2：建立 Lambda 執行角色

在此步驟中，您會建立 AWS Identity and Access Management (IAM) 角色 (WoofersLambdaRole) 並為其指派許可。這個角色會由您在 [步驟 4：建立並測試 Lambda 函數](#) 中建立的 Lambda 函數所使用。

您也要為該角色建立政策。此政策包含 Lambda 函數在執行時間所需要全部許可。

1. 使用下列內容建立名為 trust-relationship.json 的檔案。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
```

```
 "Effect": "Allow",
 "Principal": {
 "Service": "lambda.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
```

2. 輸入下列命令建立 WoofersLambdaRole。

```
aws iam create-role --role-name WoofersLambdaRole \
 --path "/service-role/" \
 --assume-role-policy-document file://trust-relationship.json
```

3. 使用下列內容建立名為 role-policy.json 的檔案。( 替換 *region* 為您 *accountID* 的 AWS 地區和帳戶 ID。 )

```
{
 "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": [
 "*"
]
 }
]
```

政策有四個陳述式可讓 `WoofersLambdaRole` 執行下列操作：

- 執行 Lambda 函數 (`publishNewBark`)。您稍後要在本教學中建立此函數。
  - 訪問 Amazon CloudWatch 日誌。Lambda 函數會在執行時間將診斷檔寫入 CloudWatch 記錄檔。
  - 從 `BarkTable` 的 DynamoDB 串流讀取資料。
  - 將訊息發佈到 Amazon SNS。
4. 輸入下列命令將此政策連接至 `WoofersLambdaRole`。

```
aws iam put-role-policy --role-name WoofersLambdaRole \
 --policy-name WoofersLambdaRolePolicy \
 --policy-document file://role-policy.json
```

### 步驟 3：建立 Amazon SNS 主題

在此步驟中，您要建立 Amazon SNS 主題 (`woofersTopic`)，並用電子郵件地址訂閱此主題。您的 Lambda 函數會使用此主題發佈 `Woofers` 使用者的新 bark。

1. 輸入以下命令來建立新的 Amazon SNS 主題。

```
aws sns create-topic --name woofersTopic
```

2. 輸入下列命令使用電子郵件地址訂閱 `woofersTopic`。(將 *region* 和 *accountID* 更換為 AWS 區域和帳戶 ID，將 *example@example.com* 更換為有效的電子郵件地址。)

```
aws sns subscribe \
 --topic-arn arn:aws:sns:region:accountID:woofersTopic \
 --protocol email \
 --notification-endpoint example@example.com
```

3. Amazon SNS 會將確認訊息傳送至電子郵件地址。選擇該郵件中的 Confirm subscription (確認訂閱) 連結，完成訂閱程序。

#### 步驟 4：建立並測試 Lambda 函數

在此步驟中，您可以建立 AWS Lambda 函數 (publishNewBark) 來處理資料流記錄BarkTable。

publishNewBark 函數只處理對應至 BarkTable 新項目的串流事件。此函數會讀取此種事件的資料，然後呼叫 Amazon SNS 來進行發佈。

1. 使用下列內容建立名為 publishNewBark.js 的檔案。取代`region`您`accountID`的 AWS 地區和帳戶 ID。

```
'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. 建立 zip 檔以包含 publishNewBark.js。如果您有 zip 命令列公用程式，即可輸入下列命令執行此作業。

```
zip publishNewBark.zip publishNewBark.js
```

3. 當您建立 Lambda 函數時，您要為自己在 [步驟 2：建立 Lambda 執行角色](#) 中建立的 WoofersLambdaRole 指定 Amazon Resource Name (ARN)。輸入下列命令以擷取此 ARN。

```
aws iam get-role --role-name WoofersLambdaRole
```

在輸出中，尋找 WoofersLambdaRole 的 ARN。

```
...
"Arn": "arn:aws:iam::region:role/service-role/WoofersLambdaRole"
...
```

輸入下列命令建立 Lambda 函數。將 *roleARN* 取代為 WoofersLambdaRole 的 ARN。

```
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. 現在，測試 publishNewBark 確認能否作用。若要執行此作業，您要提供類似 DynamoDB Streams 中真實紀錄的輸入。

使用下列內容建立名為 payload.json 的檔案。*accountID* 用您 *region* 的帳戶 ID 替換 AWS 區域 和。

```
{
 "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"
}
]
```

輸入下列命令測試 `publishNewBark` 函數。

```
aws lambda invoke --function-name publishNewBark --payload file://payload.json --
cli-binary-format raw-in-base64-out output.txt
```

如果測試成功，您就會看到以下輸出。

```
{
 "StatusCode": 200,
 "ExecutedVersion": "$LATEST"
}
```

此外，`output.txt` 檔案還會包含以下文字。

```
"Successfully processed 1 records."
```

您也會在幾分鐘內收到新的電子郵件訊息。

#### Note

AWS Lambda 將診斷資訊寫入 Amazon CloudWatch 日誌。如果 Lambda 函數發生問題，您可以使用這些診斷進行疑難排解：

1. [請在以下位置開啟 CloudWatch 主控台。](https://console.aws.amazon.com/cloudwatch/) <https://console.aws.amazon.com/cloudwatch/>
2. 在導覽窗格中，選擇 Logs (日誌)。
3. 選擇下列的日誌群組：`/aws/lambda/publishNewBark`
4. 選擇最新的日誌串流，檢視函數的輸出 (和錯誤)。

## 步驟 5：建立並測試觸發器

在 [步驟 4：建立並測試 Lambda 函數](#) 中，您已測試過 Lambda 函數，確保其能正確執行。在此步驟中，透過將 Lambda 函數 (`publishNewBark`) 與事件來源 (BarkTable 串流) 建立關聯來建立觸發條件。

1. 當您建立觸發時，您需要指定 BarkTable 串流的 ARN。輸入下列命令以擷取此 ARN。

```
aws dynamodb describe-table --table-name BarkTable
```

在輸出中，尋找 `LatestStreamArn`。

```
...
```

```
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

2. 輸入以下命令來建立觸發。將 *streamARN* 更換為實際的串流 ARN。

```
aws lambda create-event-source-mapping \
 --region region \
 --function-name publishNewBark \
 --event-source streamARN \
 --batch-size 1 \
 --starting-position TRIM_HORIZON
```

3. 測試觸發。輸入下列命令將項目新增至 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"}
```

您應該會在幾分鐘內收到新的電子郵件訊息。

4. 開啟 DynamoDB 主控台，在 BarkTable 中多新增幾個項目。您必須指定 Username 和 Timestamp 屬性的值 (您也應該指定 Message 的值，雖然它不是必要的)。您應該會因為每個新增至 BarkTable 的項目而收到新的電子郵件訊息。

Lambda 函數只處理您新增至 BarkTable 的新項目。如果您要更新或刪除資料表中的項目，此函數不會執行任何作業。

#### Note

AWS Lambda 將診斷資訊寫入 Amazon CloudWatch 日誌。如果您的 Lambda 函數發生問題，您可以使用這些診斷進行疑難排解。

1. [請在以下位置開啟 CloudWatch 主控台。](https://console.aws.amazon.com/cloudwatch/) <https://console.aws.amazon.com/cloudwatch/>
2. 在導覽窗格中，選擇 Logs (日誌)。
3. 選擇下列的日誌群組：`/aws/lambda/publishNewBark`
4. 選擇最新的日誌串流，檢視函數的輸出 (和錯誤)。

## 教學課程 #2：使用篩選條件來處理 Amazon DynamoDB 和 Lambda 的所有事件

### 主題

- [整合練習 - AWS CloudFormation](#)
- [整合練習 - CDK](#)

在本教學課程中，您將建立 AWS Lambda 觸發器，以僅處理 DynamoDB 表中串流中的某些事件。

透過 [Lambda 事件篩選](#)，您可以使用篩選條件表達式來控制 Lambda 要傳送哪些事件給函數進行處理。對於每個 DynamoDB 串流，最多可以設定 5 個不同的篩選條件。如果您使用批次處理間隔，Lambda 會將篩選條件標準套用至每個新事件，以查看是否應將其納入目前的批次中。

篩選條件會透過稱為 `FilterCriteria` 的結構進行套用。`FilterCriteria` 的 3 項主要屬性是 `metadata properties`、`data properties` 和 `filter patterns`。

以下是 DynamoDB Streams 事件的範例結構：

```
{
 "eventID": "c9fbe7d0261a5163fcb6940593e41797",
 "eventName": "INSERT",
 "eventVersion": "1.1",
 "eventSource": "aws:dynamodb",
 "awsRegion": "us-east-2",
 "dynamodb": {
 "ApproximateCreationDateTime": 1664559083.0,
 "Keys": {
 "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
 "PK": { "S": "COMPANY#1000" }
 },
 "NewImage": {
 "quantity": { "N": "50" },
 "company_id": { "S": "1000" },
 "fabric": { "S": "Florida Chocolates" },
 "price": { "N": "15" },
 "stores": { "N": "5" },
 "product_id": { "S": "1000" },
 "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
 "PK": { "S": "COMPANY#1000" },
 "state": { "S": "FL" },
 "type": { "S": "" }
 }
 }
}
```

```

 },
 "SequenceNumber": "700000000000888747038",
 "SizeBytes": 174,
 "StreamViewType": "NEW_AND_OLD_IMAGES"
 },
 "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-StreamsSampleDDBTable-LU0I6UXQY7J1/stream/2022-09-30T17:05:53.209"
}

```

metadata properties 是事件物件的欄位。若是 DynamoDB Streams, metadata properties 是類似 dynamodb 或 eventName 的欄位。

data properties 是事件主體的欄位。若要篩選 data properties, 請務必將資料屬性包含在適當金鑰內的 FilterCriteria 之中。對於 DynamoDB 事件來源, 資料金鑰為 NewImage 或 OldImage。

最後, 篩選條件規則將會定義您要套用至特定屬性的篩選條件表達式。以下是一些範例：

| 比較運算子 | 範例                      | 規則語法 (部分)                                                                |
|-------|-------------------------|--------------------------------------------------------------------------|
| Null  | 產品類型為 Null              | { "product_type":<br>{ "S": null } }                                     |
| 空白    | 產品名稱為空                  | { "product_name":<br>{ "S": [ "" ] } }                                   |
| 等於    | 州等於佛羅里達州                | { "state": { "S":<br>["FL"] } }                                          |
| 及     | 產品原產州等於佛羅里達州, 且產品類別為巧克力 | { "state": { "S":<br>["FL"] }, "category": { "S": [ "CHOCOLAT<br>E"] } } |
| 或     | 產品原產州是佛羅里達州或加州          | { "state": { "S":<br>["FL", "CA"] } }                                    |
| Not   | 產品原產州不是佛羅里達州            | {"state": {"S":<br>[{"anything-but":<br>["FL"]}]]}                       |

| 比較運算子 | 範例              | 規則語法 (部分)                                             |
|-------|-----------------|-------------------------------------------------------|
| 存在    | 自製產品存在          | <code>{"homemade": {"S": [{"exists": true}]}}</code>  |
| 不存在   | 自製產品不存在         | <code>{"homemade": {"S": [{"exists": false}]}}</code> |
| 開頭為   | PK 以 COMPANY 開頭 | <code>{"PK": {"S": [{"prefix": "COMPANY"}]}}</code>   |

您可以為一個 Lambda 函數最多指定 5 個事件篩選模式。請注意，這 5 個事件中的每一個都將評估為邏輯 OR。因此，如果您設定兩個分別名為 `Filter_One` 和 `Filter_Two` 的篩選條件，Lambda 函數將會執行 `Filter_One OR Filter_Two`。

#### Note

[Lambda 事件篩選](#) 頁面中有部分選項可用於篩選和比較數值，但不適用於 DynamoDB 篩選條件事件，因為 DynamoDB 中的數字會儲存為字串。例如 `"quantity": { "N": "50" }`，從 "N" 屬性可以得知這是一個數字。

## 整合練習 - AWS CloudFormation

要在實踐中顯示事件過濾功能，以下是一個示例 CloudFormation 模板。此範本將產生一個簡單的 DynamoDB 資料表，其中含有分割區索引鍵 PK 和排序索引鍵 SK，且已啟用 Amazon DynamoDB Streams。它會建立一個 Lambda 函數和一個簡單的 Lambda 執行角色，以允許將日誌寫入 Amazon CloudWatch，並從 Amazon DynamoDB Streams 讀取事件。它也會在 DynamoDB Streams 和 Lambda 函數之間新增事件來源映射，以便每次在 Amazon DynamoDB Streams 中發生事件時，都可以執行該函數。

```
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.comAction:
  - sts:AssumeRole

Path: "/"

Policies:

- PolicyName: root

PolicyDocument:

Version: "2012-10-17"

Statement:

- Effect: Allow  
Action:
  - logs:CreateLogGroup
  - logs:CreateLogStream
  - logs:PutLogEventsResource: 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
```

部署此雲端編組範本之後，即可插入下列 Amazon DynamoDB 項目：

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

由於此雲端形成範本內嵌包含的簡單 lambda 函數，您將在 lambda 函數的 Amazon CloudWatch 日誌群組中看到事件，如下所示：

```
{
 "eventID": "c9fbe7d0261a5163fcb6940593e41797",
 "eventName": "INSERT",
 "eventVersion": "1.1",
 "eventSource": "aws:dynamodb",
 "awsRegion": "us-east-2",
 "dynamodb": {
 "ApproximateCreationDateTime": 1664559083.0,
 "Keys": {
 "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
 "PK": { "S": "COMPANY#1000" }
 },
 "NewImage": {
 "quantity": { "N": "50" },
 "company_id": { "S": "1000" },
 "fabric": { "S": "Florida Chocolates" },
 "price": { "N": "15" },
 "stores": { "N": "5" },
 "product_id": { "S": "1000" },
 "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
 "PK": { "S": "COMPANY#1000" },
 "state": { "S": "FL" },
 "type": { "S": "" }
 },
 "SequenceNumber": "700000000000888747038",
 "SizeBytes": 174,
 "StreamViewType": "NEW_AND_OLD_IMAGES"
 },
}
```

```
"eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-StreamsSampleDDBTable-LU0I6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

## 篩選條件範例

- 僅限符合指定州的產品

此範例會修改範 CloudFormation 本，以包含篩選條件，以符合來自佛羅里達州的所有產品，縮寫為「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
```

重新部署堆疊後，即可將下列 DynamoDB 項目新增至資料表。請注意，它不會出現在 Lambda 函數日誌中，因為此範例中的產品來自加州。

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

- 僅限以 PK 和 SK 中部分值開頭的項目

此範例會修改 CloudFormation 範本，使其包含下列條件：

```
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
```

請注意，AND 條件要求條件位於模式內部，且其中的索引鍵 PK 和 SK 位於相同表達式內並以逗號分隔。

以 PK 和 SK 中的部分值開頭或來自特定州。

此範例會修改 CloudFormation 範本，使其包含下列條件：

```
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
```

請注意，新增 OR 條件的方法是在篩選條件區段中引入新的模式。

## 整合練習 - CDK

以下範例 CDK 專案編組範本逐步解說事件篩選功能。在使用此 CDK 專案之前，必須先[安裝必要條件](#)，包括[執行準備指令碼](#)。

## 建立 CDK 專案

首先創建一個新的 AWS CDK 項目，通過在一個空目錄 `cdk init` 中調用。

```
mkdir ddb_filters
cd ddb_filters
cdk init app --language python
```

`cdk init` 命令使用專案資料夾的名稱來命名專案的各種元素，包括類別、子資料夾和檔案。資料夾名稱中的任何連字號都會轉換為底線。否則，該名稱應遵循 Python 識別符的形式。例如，不應以數字開頭或包含空格。

若要使用新專案，請啟用其虛擬環境。如此可將專案的相依性安裝在本機專案資料夾中，而不是全域安裝。

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

### Note

您可以將其識別為用來啟用虛擬環境的 Mac/Linux 命令。Python 範本包含一個批次檔案 `source.bat`，允許在 Windows 上使用相同的命令。傳統的 Windows 命令 `.venv\Scripts\activate.bat` 也同樣適用。如果您使用 AWS CDK Toolkit v1.70.0 或更早版本初始化 AWS CDK 項目，則虛擬環境位於 `.env` 目錄中而不是 `.venv`。

## 基本基礎架構

在您偏好的文字編輯器中開啟檔案 `./ddb_filters/ddb_filters_stack.py`。此檔案是在您建立 AWS CDK 專案時 auto 產生的。

接下來，新增函數 `_create_ddb_table` 和 `_set_ddb_trigger_function`。這些函數將會建立佈建模式或隨需模式的 DynamoDB 資料表，其中含有分割區索引鍵 PK 和排序索引鍵 SK，且依預設會啟用 Amazon DynamoDB Streams 以顯示新舊映像。

Lambda 函數會儲存在資料夾 `lambda` 內的檔案 `app.py` 中。此檔案會在稍後建立。它會包含環境變數 `APP_TABLE_NAME` (將成為此堆疊所建立 Amazon DynamoDB 資料表的名稱)。在相同的函數中，我們將對 Lambda 函數授予串流讀取權限。最後，它會訂閱 DynamoDB Streams 做為 Lambda 函數的事件來源。

在 `__init__` 方法中的檔案尾端，您將會呼叫相應的建構以在堆疊中對其初始化。對於需要其他元件和服務的較大專案，最好在基本堆疊之外定義這些建構。

```
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)
```

現在，我們將創建一個非常簡單的 lambda 函數，將日誌打印到 Amazon CloudWatch。若要執行此操作，請建立名為 lambda 的新資料夾。

```
mkdir lambda
touch app.py
```

使用您偏好的文字編輯器，將下列內容新增至 app.py 檔案：

```
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)
```

確定您位於 /ddb\_filters/ 資料夾中，並輸入下列命令以建立範例應用程式：

```
cdk deploy
```

在特定時間點，系統會要求您確認是否要部署解決方案。輸入 Y 以接受變更。

```
#####
+ # ${LambdaHandler/ServiceRole} # arn:${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
```

部署變更後，開啟 AWS 主控台並將一個項目新增至您的資料表。

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

記 CloudWatch 錄檔現在應該包含此項目的所有資訊。

### 篩選條件範例

- 僅限符合指定州的產品



開啟檔案 `ddb_filters/ddb_filters/ddb_filters_stack.py` 加以修改，使其納入符合所有等於 "FL" 之產品的篩選條件。這可以在第 45 行中 `event_subscription` 的正下方進行修改。

```
event_subscription.add_property_override(
 property_path="FilterCriteria",
 value={
 "Filters": [
 {
 "Pattern": json.dumps(
 {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
)
 },
]
 },
)
```

- 僅限以 PK 和 SK 中部分值開頭的項目

修改 Python 指令碼以納入下列條件：

```
event_subscription.add_property_override(
 property_path="FilterCriteria",
 value={
 "Filters": [
 {
 "Pattern": json.dumps(
 {
 "dynamodb": {
 "Keys": {
 "PK": {"S": [{"prefix": "COMPANY"}]},
 "SK": {"S": [{"prefix": "PRODUCT"}]},
 }
 }
 }
)
 },
]
 },
)
```

- 應以位於 PK 和 SK 中的部分值開頭或來自特定州。

修改 python 指令碼以納入下列條件：

```
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"]}}}
)
 },
]
 },
)
```

請注意，新增 OR 條件的方法是在篩選條件陣列中新增更多元素。

清除

請在工作目錄的底部找到篩選條件堆疊，然後執行 `cdk destroy`。系統會要求您確認刪除此資源：

```
cdk destroy
Are you sure you want to delete: DdbFiltersStack (y/n)? y
```

## Lambda 的最佳實務

AWS Lambda 函數在容器內運行-與其他函數隔離的執行環境。當您第一次執行函式時，AWS Lambda 會建立新容器並開始執行函式的程式碼。

Lambda 函數有一個處理常式，每次叫用只執行一次。此處理常式包含函數的主要商業邏輯。例如，[步驟 4：建立並測試 Lambda 函數](#) 中顯示的 Lambda 函數有一個可處理 DynamoDB Streams 紀錄的處理常式。

您也可以提供僅執行一次的初始化程式碼 (在建立容器之後，但在第一次 AWS Lambda 執行處理常式之前)。中顯示的 Lambda 函數[步驟 4：建立並測試 Lambda 函數](#)具有初始化程式碼，可 JavaScript 在 Node.js 中匯入的 SDK，並為 Amazon SNS 建立用戶端。這些物件應該只在處理常式外部定義一次。

函數運行後，AWS Lambda 可能會選擇重複使用容器以進行後續調用函數。在這種情況下，您的函數處理常式或許能夠重複使用您在初始化程式碼中定義的資源 (您完全無法控制 AWS Lambda 保留容器多長時間，或容器是否得以重複使用。)

對於使用的 DynamoDB 觸發器 AWS Lambda，我們建議您執行下列操作：

- AWS 服務客戶端應該在初始化代碼中實例化，而不是在處理程序中。這 AWS Lambda 允許在容器的生命週期內重複使用現有的連接。
- 一般而言，您不需要明確管理連線或實作連線集區，因為會為您 AWS Lambda 管理此連線。

DynamoDB 串流的 Lambda 取用者無法保證只有一次交付，而且可能導致偶爾重複。請確定您的 Lambda 函數程式碼是冪等的，以避免因重複處理而產生意外的問題。

如需詳細資訊，請參閱AWS Lambda 開發人員指南中的[使用 AWS Lambda 函數的最佳做法](#)。

## 使用 DynamoDB 的隨需備份與還原

您可以使用 DynamoDB 隨需備份功能，為法規合規需求建立資料表的完整備份以供長期保留與封存。只要在 AWS Management Console 上按一下或使用單一 API 呼叫，您隨時可以備份與還原資料表資料。備份與還原動作執行時，完全不會影響資料表效能或可用性。

以下影片將為您介紹備份和還原概念。

### [備份和還原](#)

建立和管理 DynamoDB 隨需備份有兩種可用選項：

- AWS Backup 服務
- DynamoDB

搭配 AWS Backup，您可以設定備份原則並監控 AWS 資源和內部部署工作負載集中在一個位置。使用 DynamoDB 與 AWS Backup，您可以將隨需備份複製到 AWS 帳戶和區域、將成本配置標籤新增至隨需備份，以及將隨選備份轉換至冷儲存裝置，以降低成本。若要使用這些進階功能，您必須[選擇加入](#)至 AWS Backup。選擇加入的選項適用於特定帳戶和 AWS 地區，因此您可能必須使用相同帳戶選擇加入多個地區。如需詳細資訊，請參閱 [AWS Backup 開發人員指南](#)。

隨需備份與還原程序擴展時，不會降低您應用程式的效能或可用性。它採用全新獨特的分散式技術，無論資料表大小，都可讓您在幾秒內完成備份。您可以在幾秒內跨數千個分割區建立一致的備份，無須擔心排程或長時間執行的備份程序。所有隨需備份皆會分類、可供探索，並保留到確實刪除為止。

此外，隨需備份與還原操作並不會影響效能或 API 延遲。無論是否刪除資料表，都會保留備份。如需詳細資訊，請參閱 [使用 DynamoDB 備份和還原](#)。

除了與備份儲存容量相關聯的標準定價之外，DynamoDB 隨需備份不會收取額外費用。DynamoDB 隨需備份不能複製到其他帳戶或區域。若要建立跨 AWS 帳戶和區域以及用於其他進階功能的備份副本，您應該使用 AWS Backup。如果您使用 AWS Backup 功能，將會按照 AWS Backup 向您收取費用。如需 AWS 區域可用性和定價的詳細資訊，請參閱 [Amazon DynamoDB 定價](#)。

## 主題

- [搭配 DynamoDB 使用 AWS Backup](#)
- [使用 DynamoDB 備份和還原](#)

## 搭配 DynamoDB 使用 AWS Backup

Amazon DynamoDB 可協助您透過 AWS Backup 中的加強備份功能符合法規合規和業務持續性需求。AWS Backup 是一項全受管的資料保護服務，可輕鬆跨 AWS 服務、雲端和內部部署環境集中與自動化備份。使用此服務，您可以設定備份原則並監控 AWS 資源集中在一個地方。若要使用 AWS Backup，您必須[選擇加入](#)。選擇加入的選項適用於特定帳戶和 AWS 地區，因此您可能必須使用相同帳戶選擇加入多個地區。如需詳細資訊，請參閱 [AWS 備份開發人員指南](#)。

Amazon DynamoDB 與 AWS Backup 自然整合。您可以使用 AWS Backup 自動排程、複製、標記和生命週期您的 DynamoDB 隨需備份。您可以繼續從 DynamoDB 主控台檢視和還原這些備份。您可以使用 DynamoDB 主控台、API 和 AWS 命令列界面 (AWS CLI) 以啟用 DynamoDB 資料表的自動備份。

**Note**

透過 DynamoDB 進行的任何備份將保持不變。您仍可以透過目前的 DynamoDB 工作流程建立備份。

透過 AWS Backup 增強的可用備份功能包括：

**排程備份**-您可以使用備份計劃設定 DynamoDB 資料表的定期排程備份。

**跨帳戶和跨區域複製**-您可以自動將備份複製到不同AWS地區或帳戶，可讓您支援資料保護需求。

**冷儲存分層**-您可以設定備份來實作生命週期規則，以刪除備份或將備份轉換至較冷的儲存體。這可協助您最佳化備份成本。

**標籤**-您可以自動將備份標記為計費和成本分配目的。

**加密**-透過 AWS Backup 受管的 DynamoDB 隨需備份現在會儲存在 AWS Backup 保存庫。這可讓您使用與 DynamoDB 資料表加密金鑰無關的 AWS KMS key 加密和保護備份。

**稽核備份**-您可以使用 AWS Backup Audit Manager 來稽核您的 AWS Backup 政策合規性，並尋找尚未與您定義控制項相容的備份活動和資源。您也可以使用它來自動產生每日和隨需報告的稽核記錄，以滿足您的備份控管目的。

**使用 WORM 模式來保護備份**— 您可以使用 AWS Backup 文件庫鎖定，以啟用備份的單寫多讀 (WORM) 設定。搭配AWS Backup文件庫鎖定，您可以新增額外的防禦層，以保護備份免於意外或惡意刪除作業、備份保留期間的變更，以及生命週期設定的更新。如需進一步了解，請參閱[AWS Backup 保存庫鎖](#)。

這些增強型備份功能於可用於所有 AWS 區域。若要進一步了解這些功能，請參閱[AWS Backup開發人員指南](#)。

**主題**

- [使用 AWS Backup 備份和還原 DynamoDB 資料表：運作方式](#)
- [使用 AWS Backup 建立 DynamoDB 資料表備份](#)
- [使用 AWS Backup 複製有 DynamoDB 資料表的備份](#)
- [從中還原 DynamoDB 資料表的備份 AWS Backup](#)
- [使用刪除 DynamoDB 資料表的備份 AWS Backup](#)

- [使用須知](#)

## 使用 AWS Backup 備份和還原 DynamoDB 資料表：運作方式

您可以使用隨需備份功能來建立 Amazon DynamoDB 資料表的完整備份。本節概述備份與還原程序期間所發生的情況。

### 備份

當您使用 AWS Backup 建立隨需備份時，系統會為請求的時間標記建立目錄。備份以非同步方式建立，它會套用所有變更，直到請求最後一個完整資料表快照為止。

每次您建立隨需備份時，系統都會備份整個資料表資料。您可建立的隨需備份數目不限。

#### Note

與 DynamoDB 備份不同的是，使用 AWS Backup 進行的備份不是即時的。

您無法在備份進行時，執行下列操作：

- 暫停或取消備份操作。
- 刪除備份的來源資料表。
- 在資料表的備份進行時，停用該資料表的備份。

AWS Backup 提供自動化備份排程、保留管理和生命週期管理。這樣就不需要自訂指令碼和手動程序。AWS Backup 會執行備份，並在到期時刪除備份。如需詳細資訊，請參閱 [AWS Backup 開發人員指南](#)。

如果使用主控台，任何使用 AWS Backup 建立的備份將會列於 Backups (備份) 分頁，Backup type (備份類型) 設為 AWS\_BACKUP。

#### Note

您無法使用 DynamoDB 主控台刪除標示為 Backup type (備份類型) 的 AWS\_BACKUP。若要管理這些備份，請使用 AWS Backup 主控台。

若要了解如何執行備份，請參閱「[備份 DynamoDB 資料表](#)」。

## 還原

還原資料表時，不會使用資料表的任何佈建輸送量。您可以從 DynamoDB 備份執行完整資料表還原，或設定目的地資料表設定。當您執行還原時，可以變更下列資料表設定：

- 全域次要索引 (GSI)
- 本機次要索引 (LSI)
- 帳單模式
- 佈建的讀取與寫入容量
- 加密設定

### Important

當您執行完整資料表還原時，會使用與來源資料表相同的佈建讀取容量單位與寫入容量單位來設定目標資料表，如請求備份時所記錄。還原程序也會還原本機次要索引與全域次要索引。

您可以將 DynamoDB 資料表資料的備份複製到不同的 AWS 區域，然後在該新區域中還原它。您可以在 AWS 商業區域、AWS 中國區域和 AWS GovCloud (US) 區域之間進行複製與還原備份。您只需為從來源區域中傳輸出來的資料，以及還原為目標區域中的新資料表付費。

AWS Backup 將恢復包含所有原始索引的資料表。

您必須在還原的資料表上手動進行下列設定：

- 自動調整規模政策
- AWS Identity and Access Management (IAM) 政策
- Amazon CloudWatch 指標和警示
- Tags (標籤)
- 串流設定
- 存留時間 (TTL) 設定
- 刪除保護設定
- 時間點復原 (PITR) 設定

您只能從備份將整個資料表資料還原為新的資料表。您只能在還原的資料表變為作用中之後，才可寫入資料表。

**Note**

AWS Backup 還原是非破壞性的。您無法在還原操作期間覆寫現有的資料表。

服務指標顯示 95% 的客戶資料表還原會在一小時內完成。不過，還原時間與資料表的組態 (例如資料表的大小和基礎分割區的數目) 和其他相關變數直接相關。規劃災難復原的最佳實務是定期記錄平均還原完成時間，並確認這些時間如何影響整體復原時間目標。

若要了解如何執行還原，請參閱「[從備份中還原 DynamoDB 資料表](#)」。

您可以使用 IAM 政策進行存取控制。如需詳細資訊，請參閱 [搭配 IAM 使用 DynamoDB 備份與還原](#)。

所有備份與還原主控台以及 API 動作都會擷取並記錄到 AWS CloudTrail 中，以記錄、持續監控與稽核。

## 使用 AWS Backup 建立 DynamoDB 資料表備份

本節說明如何開啟 AWS Backup 以從 DynamoDB 資料表建立隨需備份和排程備份。

### 主題

- [開啟 AWS Backup 功能](#)
- [隨需備份](#)
- [排程備份](#)

### 開啟 AWS Backup 功能

您必須開啟 AWS Backup 才能與 DynamoDB 搭配使用。

若要開啟 AWS Backup，請進行以下步驟：

1. 登入 AWS 管理主控台，並前往 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 在 [備份設定] 視窗中選擇開啟。
4. 確認畫面隨即出現。選擇開啟功能。

AWS Backup 功能現在可用於您的 DynamoDB 資料表。



如果您在 AWS Backup 功能開啟後選擇關閉它們，請依照下列步驟操作：

1. 登入 AWS 管理主控台，並前往 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 在 [備份設定] 視窗中選擇關閉。
4. 確認畫面隨即出現。選擇關閉功能。

如果您無法開啟或關閉 AWS Backup 功能，可能需由您的 AWS 管理員執行這些操作。

### 隨需備份

若要建立 DynamoDB 資料表的隨需備份，請依照下列步驟執行：

1. 登入 AWS 管理主控台，並前往 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 選擇 Create backup (建立備份)。
4. 從顯示的下拉式功能表中，選擇 Create an on-demand backup (建立隨需備份)。
5. 若要使用暖儲存和其他基本功能建立由 AWS Backup 管理的備份，請選擇 Default Settings (預設設定)。若要建立可轉換至冷儲存的備份，或使用 DynamoDB 功能而非 AWS Backup 建立備份，請選擇 Customize settings (自訂設定)。

若您要改為使用先前的 DynamoDB 功能來建立此備份，請選擇 Customize settings (自訂設定)，然後選擇 Backup with DynamoDB (使用 DynamoDB 進行備份)。

6. 當您完成設定後，選擇 Create backup (建立備份)。

### 排程備份

若要排程備份，請依照下列步驟執行。

1. 登入 AWS 管理主控台，並前往 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 從顯示的下拉式選單中，選擇使用 AWS Backup 排程備份。
4. 您會被導向至 AWS Backup 以建立備份計畫。

## 使用 AWS Backup 複製有 DynamoDB 資料表的備份

您可以製作現有備份的複製。您可以視需要將備份複製到多個 AWS 帳戶或 AWS 區域，或自動做為排程備份計劃的一部分。您也可以為 Amazon DynamoDB Encryption Client 自動化一系列跨帳戶和跨區域副本。

如果您有業務持續性或合規性要求，需要將備份儲存在與生產資料最短距離的位置，則跨區域複寫特別有用。

跨帳戶備份對於將備份安全地複製到組織內一個或多個 AWS 帳戶的運作或安全性考量相當有助益。如果不小心刪除您的原始備份，您可以將備份從其目的地帳戶複製到其來源帳戶，然後啟動還原。您必須在組織服務中有兩個屬於相同組織的帳戶，才能執行這項操作。

除非您另有指定，否則複本會繼承來源備份的組態，但有一個例外：如果您指定新副本「永不」過期。使用此設定，新複本仍會繼承其來源到期日。如果您希望新備份副本是永久性的，請將來源備份設定為永不過期，或指定新副本在建立後 100 年過期。

### Note

如果您要複製到其他帳戶，您必須先取得該帳戶的許可。

若要複製備份，請執行以下操作：

1. 登入 AWS 管理主控台，並前往 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 選取要複製的備份旁的核取方塊。
  - 如果要複製的備份顯示為灰色，則必須啟用 [AWS Backup 的進階功能](#)。然後建立一個新的備份。現在，您可以將此新備份複製到其他區域和帳戶，然後複製日後的任何其他新備份。
4. 請選擇 Copy (複製)。
5. 若您要複製備份到另一個帳戶或區域，請選取 Copy the recovery point to another destination (將復原點複製到另一個目的地)。然後選擇要複製到您帳戶中的其他區域，還是複製到不同區域的其他帳戶。

### Note

若要將備份還原到另一個區域或帳戶，您必須先將備份複製到該區域或帳戶。

6. 選取要將檔案複製到其中的所需保存庫。如果需要，您也可建立新的備份保存庫。
7. 選擇 Copy backup (複製備份)。

## 從中還原 DynamoDB 資料表的備份 AWS Backup

本節說明如何從中還原 DynamoDB 表格的備份。AWS Backup

### 主題

- [從中還原 DynamoDB 資料表 AWS Backup](#)
- [將 DynamoDB 資料表還原至其他區域或帳戶](#)

## 從中還原 DynamoDB 資料表 AWS Backup

若要從中還原 DynamoDB 表格 AWS Backup，請依照下列步驟執行：

1. [登入 AWS 管理主控台並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇 Backups (備份) 索引標籤。
4. 選取您要還原的上一個備份旁的核取方塊。
5. 選擇 Restore (還原)。您會被導向至 Restore table from backup (從備份還原資料表) 畫面。
6. 輸入新還原資料表的名稱、此新資料表將有的加密、要加密還原的金鑰，以及其他選項。
7. 完成時，請選擇 Restore (還原)。

## 將 DynamoDB 資料表還原至其他區域或帳戶

若要將 DynamoDB 資料表還原到其他區域或帳戶，您必須先將備份複製到新的區域或帳戶。為了複製到另一個帳戶，該帳戶必須先授予您許可。將 DynamoDB 備份複製到新的區域或帳戶後，可以使用上一節中的程序還原該備份。

## 使用刪除 DynamoDB 資料表的備份 AWS Backup

本節說明如何使用刪除 DynamoDB 表格的備份。AWS Backup

透過 Backup 功能建立的 DynamoDB B AWS Backup 會儲存在備 AWS 份保存庫中。

若要刪除這種備份，請執行以下操作：

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 在隨後的畫面上，選擇 [繼續 AWS Backup]。

您將被帶到 AWS Backup 主控台。若要深入瞭解如何刪除上的備份 AWS Backup 主控台，請參閱[刪除備份](#)。

如需有關詳細資訊，AWS Backup 請參閱AWS 規定指引 AWS Backup中的 [Backup 和復原使用](#)。

## 使用須知

本節說明 AWS Backup 和 DynamoDB 管理的隨需備份之間的技术差異。

AWS Backup 有一些與 DynamoDB 不同的工作流程和行為。其中包含：

Encryption (加密)-使用 AWS Backup 方案會儲存在加密的保存庫中，其金鑰由 AWS Backup 服務管理。保存庫具有存取控制政策，以提供額外的安全性。

Backup ARN-由 AWS Backup 建立的備份文件現在會有一個 AWS Backup ARN，這可能會影響使用者許可模型。Backup 資源名稱 (ARN) 將從 `arn:aws:dynamodb` 改變為 `arn:aws:backup`。

刪除備份-由AWS Backup建立的備份只能從AWS Backup保存庫中刪除。您將無法從 DynamoDB 主控台刪除AWS Backup檔案。

Backup 程序-與 DynamoDB 備份不同的是，使用AWS Backup進行的備份不是即時的。

計費-使用AWS Backup功能的 DynamoDB 資料表備份費用來自AWS Backup。

IAM 角色-如果您是透過 IAM 角色管理存取，則還需要使用下列新許可來設定新的 IAM 角色：

```
"dynamodb:StartAwsBackupJob",
"dynamodb:RestoreTableFromAwsBackup"
```

需要使用 `dynamodb:StartAwsBackupJob` 才能成功備份 AWS Backup 功能，以及需要 `dynamodb:RestoreTableFromAwsBackup` 才能從 AWS Backup 功能復原備份。

若要在完整的 IAM 政策中檢視這些許可，請參閱[使用 IAM](#) 的範例 8。

## 使用 DynamoDB 備份和還原

Amazon DynamoDB 支援獨立的隨需備份和還原功能。無論您是否使用 AWS Backup，這些功能都可供您使用。

您可以使用 DynamoDB 隨需備份功能針對法規合規需求，建立資料表的完整備份以供長期保留與封存。只要在 AWS 管理主控台上按一下或使用單一 API 呼叫，您隨時可以備份與還原資料表資料。備份與還原動作執行時，完全不會影響資料表效能或可用性。

您可以使用主控台、AWS 命令列界面 (AWS CLI) 或 DynamoDB API 建立資料表備份。如需詳細資訊，請參閱 [備份 DynamoDB 資料表](#)。

如需從備份還原資料表的資訊，請參閱「[從備份中還原 DynamoDB 資料表](#)」。

### 使用 DynamoDB 備份和還原 DynamoDB 資料表：運作方式

您可以使用 DynamoDB 隨需備份功能來建立 Amazon DynamoDB 資料表的完整備份。此功能獨立於 AWS Backup。本節概述 DynamoDB 備份與還原程序期間所發生的情況。

#### 備份

當您用 DynamoDB 建立隨需備份時，系統會為請求的時間標記建立目錄。備份以非同步方式建立，它會套用所有變更，直到請求最後一個完整資料表快照為止。系統會立即處理 DynamoDB 備份請求，該請求幾分鐘內就可還原。

#### Note

每次您建立隨需備份時，系統都會備份整個資料表資料。您可建立的隨需備份數目不限。

DynamoDB 中的所有備份在執行時皆不會使用資料表的任何佈建輸送量。

DynamoDB 備份並不保證項目之間具因果一致性，但是備份中更新之間的扭曲通常遠少於一秒。

您無法在備份進行時，執行下列操作：

- 暫停或取消備份操作。
- 刪除備份的來源資料表。
- 在資料表的備份進行時，停用該資料表的備份。

如果您不希望建立排程指令碼和清理任務，可以使用 AWS Backup 建立備份計畫，為 DynamoDB 資料表設置排程和保留政策。AWS Backup 會執行備份，並在備份到期時予以刪除。如需詳細資訊，請參閱《[AWS Backup 開發人員指南](#)》。

除了 AWS Backup，您可以使用 AWS Lambda 功能排程定期或未來備份。如需詳細資訊，請參閱[無伺服器解決方案以排定 Amazon DynamoDB 隨需備份](#)。

如果使用主控台，任何使用 AWS Backup 建立的備份將會列於 Backups (備份) 分頁，Backup type (備份類型) 設為 AWS。

#### Note

您無法使用 DynamoDB 主控台刪除標示為 Backup type (備份類型) 的 AWS。若要管理這些備份，請使用 AWS Backup 主控台。

若要了解如何執行備份，請參閱「[備份 DynamoDB 資料表](#)」。

## 還原

還原資料表時，不會使用資料表的任何佈建輸送量。您可以從 DynamoDB 備份執行完整資料表還原，或設定目的地資料表設定。當您執行還原時，可以變更下列資料表設定：

- 全域次要索引 (GSI)
- 本機次要索引 (LSI)
- 帳單模式
- 佈建的讀取與寫入容量
- 加密設定

#### Important

當您執行完整資料表還原時，會使用與來源資料表相同的佈建讀取容量單位與寫入容量單位來設定目標資料表，如請求備份時所記錄。還原程序也會還原本機次要索引與全域次要索引。

您也可以跨 AWS 區域還原 DynamoDB 資料表資料，將還原的資料表建立在不同於備份所在的區域。您可以在 AWS 商業區域、AWS 中國區域和 AWS GovCloud (US) 區域之間進行跨區域還原。您只需為從來源區域中傳輸出來的資料，以及還原為目標區域中的新資料表付費。

如果您選擇阻止在新還原的資料表上建立部分或全部次要索引，則還原會更快且更符合經濟效益。

您必須在還原的資料表上手動進行下列設定：

- 自動調整規模政策
- AWS Identity and Access Management (IAM) 政策
- Amazon CloudWatch 指標和警示
- Tags (標籤)
- 串流設定
- 存留時間 (TTL) 設定
- 刪除保護設定
- 時間點復原 (PITR) 設定

您只能從備份將整個資料表資料還原為新的資料表。您只能在還原的資料表變為作用中之後，才可寫入資料表。

#### Note

您無法在還原操作期間覆寫現有的資料表。

服務指標顯示 95% 的客戶資料表還原會在一小時內完成。不過，還原時間與資料表的組態 (例如資料表的大小和基礎分割區的數目) 和其他相關變數直接相關。規劃災難復原的最佳實務是定期記錄平均還原完成時間，並確認這些時間如何影響整體復原時間目標。

若要了解如何執行還原，請參閱「[從備份中還原 DynamoDB 資料表](#)」。

您可以使用 IAM 政策進行存取控制。如需詳細資訊，請參閱 [搭配 IAM 使用 DynamoDB 備份與還原](#)。

所有備份與還原主控台以及 API 動作都會擷取並記錄到 AWS CloudTrail 中，以記錄、持續監控與稽核。

## 備份 DynamoDB 資料表

本節說明如何使用 Amazon DynamoDB 主控台或 AWS Command Line Interface 來備份資料表。

### 主題

- [建立資料表備份 \(主控台\)](#)



- [建立資料表備份 \(AWS CLI\)](#)

## 建立資料表備份 (主控台)

請遵循下列步驟，使用 AWS Management Console 為現有的 Music 資料表建立名為 MusicBackup 的備份。

### 建立資料表備份

1. 登入 AWS Management Console，並在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 您可以執行下列其中一項操作來建立備份：
  - 在 Music 資料表的 Backups (備份) 索引標籤上，選擇 Create backup (建立備份)。
  - 在主控台左側的導覽窗格中，選擇 Backups (備份)。然後選擇 Create backup (建立備份)。
3. 請確定 Music 是資料表名稱，然後輸入 **MusicBackup** 做為備份名稱。然後，選擇 Create (建立) 建立備份。

## 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.

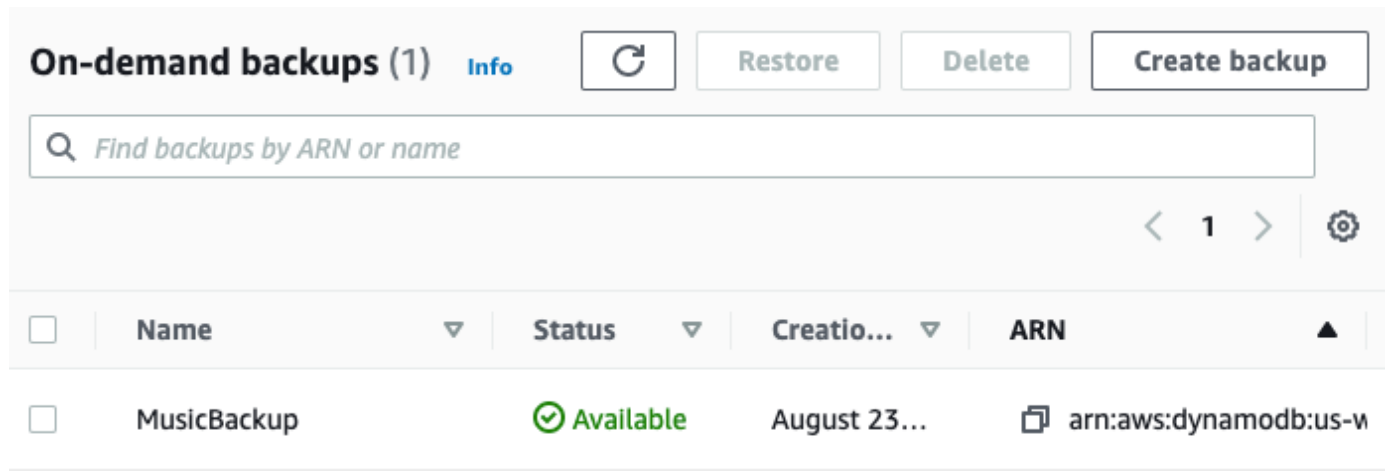
[Cancel](#)[Create backup](#)

### Note

如果您使用導覽窗格中的 Backups (備份) 區段建立備份，則不會為您預先選取資料表。您必須手動選擇備份的來源資料表名稱。



建立備份期間，備份狀態會設定為 **Creating** (建立中)。備份完成之後，備份狀態會變更為 **Available** (可用)。



The screenshot shows the 'On-demand backups (1)' 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 below the search bar lists the backup details:

| <input type="checkbox"/> | Name        | Status    | Creation Date | ARN                   |
|--------------------------|-------------|-----------|---------------|-----------------------|
| <input type="checkbox"/> | MusicBackup | Available | August 23...  | arn:aws:dynamodb:us-w |

## 建立資料表備份 (AWS CLI)

請遵循下列步驟，使用 AWS CLI 為現有的資料表 **Music** 建立備份。

### 建立資料表備份

- 為 **MusicBackup** 資料表建立名為 **Music** 的備份。

```
aws dynamodb create-backup --table-name Music \
--backup-name MusicBackup
```

建立備份期間，備份狀態會設定為 **CREATING**。

```
{
 "BackupDetails": {
 "BackupName": "MusicBackup",
 "BackupArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc",
 "BackupStatus": "CREATING",
 "BackupCreationDateTime": 1489602797.149
 }
}
```

備份完成後，其 BackupStatus 應該變更為 AVAILABLE。若要確認，請使用 describe-backup 命令。您可以從先前步驟的輸出或使用 backup-arn 命令取得 list-backups 的輸入值。

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/01489173575360-
b308cd7d
```

若要追蹤您的備份，您可以使用 list-backups 命令。它會列出處於 CREATING 或 AVAILABLE 狀態的所有備份。

```
aws dynamodb list-backups
```

list-backups 命令與 describe-backup 命令適用於查看備份之來源資料表的相關資訊。

## 從備份中還原 DynamoDB 資料表

本節說明如何使用 Amazon DynamoDB 主控台或 AWS Command Line Interface (AWS CLI) 從備份還原資料表。

### Note

如果您想要使用 AWS CLI，就必須先設定它。如需詳細資訊，請參閱 [存取 DynamoDB](#)。

## 主題

- [從備份中還原資料表 \(主控台\)](#)
- [從備份中還原資料表 \(AWS CLI\)](#)

## 從備份中還原資料表 (主控台)

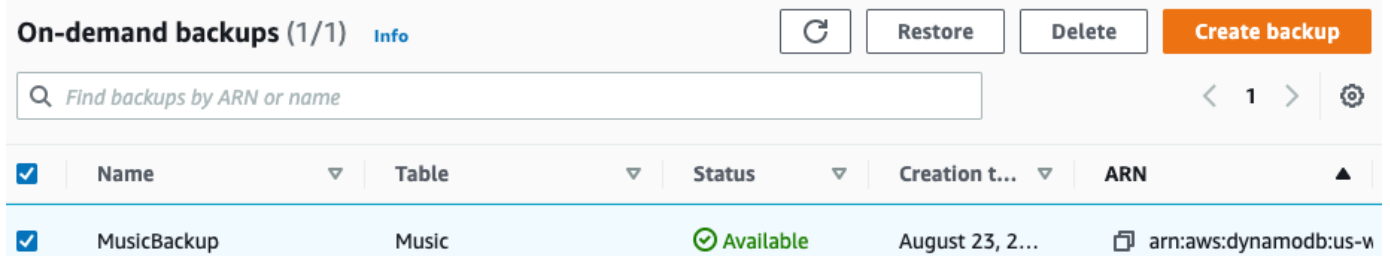
下列程序示範如何使用在「Music」教學中建立的 MusicBackup 檔案還原 [備份 DynamoDB 資料表](#) 資料表。

### Note

此程序假設在使用 Music 檔案還原資料表前 MusicBackup 資料表已不存在。

## 從備份還原資料表

1. 登入 AWS Management Console，並在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 在備份清單中，選擇 MusicBackup。



4. 選擇 Restore (還原)。
5. 輸入 **Music** 做為新資料表的名稱。確認備份名稱和其他備份詳細資訊。然後選擇 Restore table (還原資料表) 啟動還原程序。

### Note

您可以將資料表還原至相同的 AWS 區域，或還原至不同於備份所在的區域。您可以阻止在新還原的資料表上建立次要索引。此外，您可以指定不同的加密模式。從備份還原的資料表一律使用 DynamoDB 標準資料表類別建立。

# 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

正在還原的資料表會顯示為 Creating (正在建立) 狀態。還原程序完成後，Music 資料表的狀態會變更為 Active (作用中)。

## 從備份中還原資料表 (AWS CLI)

遵循這些步驟透過 AWS CLI 使用在「[備份 DynamoDB 資料表](#)」教學中建立的 MusicBackup 還原 Music 資料表。

### 從備份還原資料表

1. 使用 `list-backups` 命令確認您想要還原的備份。此範例使用 MusicBackup。

```
aws dynamodb list-backups
```

如需備份的其他詳細資訊，請使用 `describe-backup` 命令。您可以從上一步取得輸入 `backup-arn`。

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d
```

2. 從備份還原資料表。在本例中，MusicBackup 會將 Music 資料表還原至相同的 AWS 區域。

```
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. 利用自訂的資料表設定，從儲存貯體還原資料表。在本例中，MusicBackup 會還原 Music 資料表並指定還原資料表的加密模式。

#### Note

`sse-specification-override` 參數採用的數值與 `CreateTable` 命令中使用的 `sse-specification-override` 參數相同。如需進一步了解，請參閱 [在 DynamoDB 中管理加密資料表](#)。

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d
--sse-specification-override SSE-SUPPORTED
```

```
--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
```

您可以將資料表還原到不同於備份所在的 AWS 區域。

#### Note

- 執行跨區域還原必須使用 `sse-specification-override` 參數，而還原至與來源資料表相同的區域時則可選用此參數。
- 從命令列執行跨區域還原時，必須將預設 AWS 區域設定為所需的目的地區域。如需進一步了解，請參閱《AWS Command Line Interface 使用者指南》中的[命令列選項](#)。

```
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
```

您可以覆寫帳單模式及佈建給還原資料表的輸送量。

```
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
```

您可以阻止在還原的資料表上建立部分或全部次要索引。

#### Note

如果您阻止在還原的資料表上建立部分或全部次要索引，可加速還原且更符合經濟效益。

```
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

提供的次要索引應該符合現有索引。您無法在還原時建立新索引。

您可以使用不同覆寫的組合。例如，您可以使用單一全域次要索引，同時變更所佈建的輸送量，如下所示。

```
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
```

若要確認還原，請使用 `describe-table` 命令描述 Music 資料表。

```
aws dynamodb describe-table --table-name Music
```

正在從備份還原的資料表會顯示為 `Creating` (正在建立) 狀態。還原程序完成後，Music 資料表的狀態會變更為 `Active` (作用中)。

### Important

當還原正在進行時，請勿修改或刪除 IAM 角色政策，否則可能會造成非預期的行為。例如，假設您在資料表還原時移除資料表的寫入許可。在此案例中，基礎 `RestoreTableFromBackup` 操作會無法將任何還原的資料寫入資料表。

還原操作完成後，您就可以修改或刪除 IAM 角色政策。

涉及存取目標還原資料表的[來源 IP 限制](#)的 IAM 政策，應該將 `aws:ViaAWSService` 金鑰設定為 `false`，以此確保限制僅適用於委託人直接提出的請求。否則，還原將被取消。如果您的備份使用 AWS 受管金鑰 或客戶受管金鑰加密，請勿在還原進行期間停用或刪除金鑰，否則還原將會失敗。還原操作完成後，您可以變更還原資料表的加密金鑰，並停用或刪除舊的金鑰。

## 刪除 DynamoDB 資料表備份

本節說明如何使用 AWS Management Console 或 AWS Command Line Interface (AWS CLI) 來刪除 Amazon DynamoDB 資料表備份。

### Note

若要使用 AWS CLI，請先進行設定。如需詳細資訊，請參閱 [使用 AWS CLI](#)。

## 主題

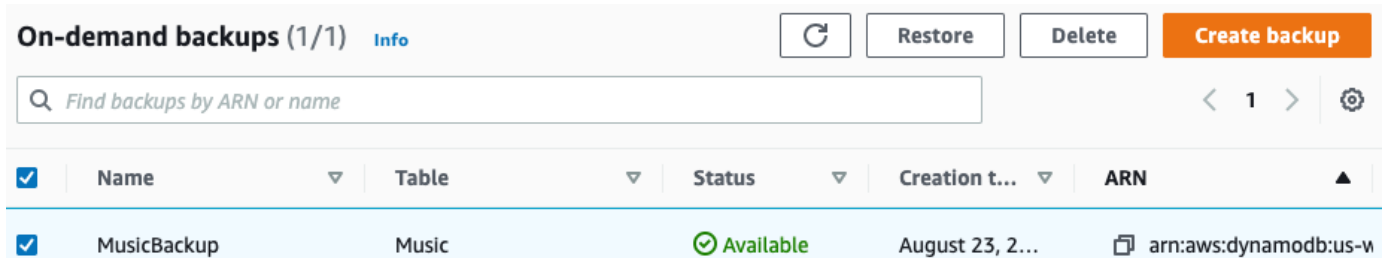
- [刪除資料表備份 \(主控台\)](#)
- [刪除資料表備份 \(AWS CLI\)](#)

## 刪除資料表備份 (主控台)

下列程序顯示如何使用主控台刪除 [備份 DynamoDB 資料表](#) 教學中所建立的 MusicBackup。

### 刪除備份

1. 登入 AWS Management Console，並在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Backups (備份)。
3. 在備份清單中，選擇 MusicBackup。



| On-demand backups (1/1) <a href="#">Info</a> |             |       |           |                 |                       |  |       | Restore | Delete | Create backup |
|----------------------------------------------|-------------|-------|-----------|-----------------|-----------------------|--|-------|---------|--------|---------------|
| Find backups by ARN or name                  |             |       |           |                 |                       |  | < 1 > |         |        |               |
| <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. 選擇 Delete (刪除)。確認您要刪除備份，方法是輸入 **delete**，然後按一下 Delete (刪除)。

### 刪除資料表備份 (AWS CLI)

下列範例會使用 AWS CLI 來刪除現有資料表 Music 資料表的備份。

```
aws dynamodb delete-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc
```

### 搭配 IAM 使用 DynamoDB 備份與還原

您可以使用 AWS Identity and Access Management (IAM) 限制對某些資源的 Amazon DynamoDB 備份與還原動作。在每份資料表中操作 CreateBackup 和 RestoreTableFromBackup API。

如需在 DynamoDB 中使用 IAM 政策的詳細資訊，請參閱 [適用於 DynamoDB 的以身分為基礎的政策](#)。

您可使用以下 IAM 政策範例，在 DynamoDB 中設定特定的備份與還原功能。

#### 範例 1：允許 CreateBackup 和 RestoreTableFromBackup 動作

下列 IAM 政策會授予許可，允許在所有資料表上執行 CreateBackup 與 RestoreTableFromBackup DynamoDB 動作：

```
{
 "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

DynamoDB `RestoreTableFromBackup` 權限對來源備份而言為必要，而目標資料表的 DynamoDB 讀取和寫入權限對於還原功能而言則為必要。

DynamoDB `RestoreTableToPointInTime` 權限對來源資料表而言為必要，而目標資料表的 DynamoDB 讀取和寫入權限對於還原功能而言則為必要。

### 範例 2：允許 `CreateBackup` 並拒絕 `RestoreTableFromBackup`

下列 IAM 政策授予 `CreateBackup` 動作的許可，並拒絕 `RestoreTableFromBackup` 動作：

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": ["dynamodb:CreateBackup"],
 "Resource": "*"
 },
 {
 "Effect": "Deny",
 "Action": ["dynamodb:RestoreTableFromBackup"],
 "Resource": "*"
 }
]
}
```

### 範例 3：允許 `ListBackups` 並拒絕 `CreateBackup` 及 `RestoreTableFromBackup`

下列 IAM 政策授予 `ListBackups` 動作的許可，並拒絕 `CreateBackup` 和 `RestoreTableFromBackup` 動作：

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
```

```

 "Action": ["dynamodb:ListBackups"],
 "Resource": "*"
 },
 {
 "Effect": "Deny",
 "Action": [
 "dynamodb:CreateBackup",
 "dynamodb:RestoreTableFromBackup"
],
 "Resource": "*"
 }
]
}

```

#### 範例 4：允許 ListBackups 並拒絕 DeleteBackup

下列 IAM 政策授予 ListBackups 動作的許可，並拒絕 DeleteBackup 動作：

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": ["dynamodb:ListBackups"],
 "Resource": "*"
 },
 {
 "Effect": "Deny",
 "Action": ["dynamodb>DeleteBackup"],
 "Resource": "*"
 }
]
}

```

#### 範例 5：允許所有資源的 RestoreTableFromBackup 和 DescribeBackup，並拒絕特定備份的 DeleteBackup

下列 IAM 政策會授予 RestoreTableFromBackup 和 DescribeBackup 動作的許可，並拒絕特定備份資源的 DeleteBackup 動作：

```

{

```

```

"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

DynamoDB `RestoreTableFromBackup` 權限對來源備份而言為必要，而目標資料表的 DynamoDB 讀取和寫入權限對於還原功能而言則為必要。

DynamoDB `RestoreTableToPointInTime` 權限對來源資料表而言為必要，而目標資料表的 DynamoDB 讀取和寫入權限對於還原功能而言則為必要。

## 範例 6：允許特定資料表的 CreateBackup

下列 IAM 政策只授予在 Movies 資料表上執行 CreateBackup 動作的許可：

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": ["dynamodb:CreateBackup"],
 "Resource": [
 "arn:aws:dynamodb:us-east-1:123456789012:table/Movies"
]
 }
]
}
```

## 範例 7：允許 ListBackups

下列 IAM 政策會授予 ListBackups 動作的許可：

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": ["dynamodb:ListBackups"],
 "Resource": "*"
 }
]
}
```

### Important

您無法授予對特定資料表執行 ListBackups 動作的許可。

## 範例 8：允許存取 AWS Backup 功能

您將需要 StartAwsBackupJob 動作的 API 許可，以便成功進行進階功能備份，以及 dynamodb:RestoreTableFromAwsBackup 動作，以成功還原該備份。

下列 IAM 政策授予 AWS Backup 觸發具有進階功能的備份和還原的許可。另請注意，如果資料表已加密，則政策將需要存取[AWS KMS 金鑰](#)。

```
{
 "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": "*"
 }
]
}
```

### 範例 9：拒絕特定來源資料表的 RestoreTableToPointInTime

以下 IAM 政策拒絕特定來源資料表 RestoreTableToPointInTime 動作的權限：

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Deny",
 "Action": [
 "dynamodb:RestoreTableToPointInTime"
],
 "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music"
 }
]
}
```

## 範例 10：拒絕為特定來源資料表的所有備份執行 RestoreTableFromBackup

以下 IAM 政策拒絕特定來源資料表中所有備份的 RestoreTableToPointInTime 動作權限：

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Deny",
 "Action": [
 "dynamodb:RestoreTableFromBackup"
],
 "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/*"
 }
]
}
```

## 適用於 DynamoDB 的 Point-in-time 復原

您可以建立 Amazon DynamoDB 表格的隨需備份，也可以使用復原來啟用 point-in-time 連續備份。如需隨需備份的詳細資訊，請參閱[使用 DynamoDB 的隨需備份與還原](#)。

Point-in-time 復原可協助保護 DynamoDB 資料表免於意外寫入或刪除作業。使用 point-in-time 復原功能，您不必擔心建立、維護或排程隨選備份的問題。例如，假設測試指令碼意外寫入至生產 DynamoDB 資料表。透過 point-in-time 復原，您可以將該資料表還原到過去 35 天內的任何時間點。啟用 point-in-time 復原之後，您可以從目前時間前五分鐘還原到 35 天前的任何時間點。DynamoDB 維護您資料表的增量備份。

此外，point-in-time 作業不會影響效能或 API 延遲。如需詳細資訊，請參閱[Point-in-time 恢復：它是如何工作的](#)。

您可以使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或 DynamoDB API 將 DynamoDB 表格還原到某個時間點。point-in-time 復原程序會還原至新資料表。如需詳細資訊，請參閱[還原 DynamoDB 資料表至某個時間點](#)。

以下視頻將為您介紹備份和還原概念，並討論有關 point-in-time 恢復的更多信息。

### [備份和還原](#)

如需 AWS 區域可用性和定價的詳細資訊，請參閱[Amazon DynamoDB 定價](#)。

## 主題

- [Point-in-time 恢復：它是如何工作的](#)
- [在您開始使用 point-in-time 復原之前](#)
- [還原 DynamoDB 資料表至某個時間點](#)

## Point-in-time 恢復：它是如何工作的

Amazon DynamoDB 資料庫 point-in-time 復原 (PITR) 可為您的 DynamoDB 表格資料提供自動備份。本章節將概要說明該過程如何在 DynamoDB 中運作。

### 啟用 point-in-time 復原

您可以使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或 DynamoDB API 啟用 point-in-time 復原功能。啟用時，point-in-time 復原會提供持續備份，直到您明確將其關閉為止。如需詳細資訊，請參閱 [還原 DynamoDB 資料表至某個時間點](#)。

啟用 point-in-time 復原之後，您可以還原到 `EarliestRestorableDateTime` 和內的任何時間點 `LatestRestorableDateTime`。`LatestRestorableDateTime` 通常比目前時間前五分鐘。

#### Note

point-in-time 復原程序會一律還原至新資料表。

### 使用 point-in-time 恢復恢復表

對於 `EarliestRestorableDateTime`，您可以將表格還原到過去 35 天內的任何時間點。保留期間為 35 天 (5 個行事曆週)，不能修改。任何數量的使用者都可以在特定帳戶中執行最多 50 次同時還原 (任何類型的還原)。

#### Important

如果您停用 point-in-time 復原並稍後在資料表上重新啟用它，則會重設可復原該資料表的開始時間。因此，您只能使用 `LatestRestorableDateTime` 立即還原該資料表。

使用 point-in-time 復原進行還原時，DynamoDB 會根據選取的日期和時間 (`day:hour:minute:second`) 將表格資料還原到新資料表的狀態。



還原資料表時，不會使用資料表的任何佈建輸送量。您可以使用 point-in-time 復原來執行完整的資料表還原，也可以設定目的地資料表設定。您可以變更還原的資料表上的以下資料表設定：

- 全域次要索引 (GSI)
- 本機次要索引 (LSI)
- 帳單模式
- 佈建的讀取與寫入容量
- 加密設定

#### Important

當您執行完整資料表還原時，會使用與來源資料表相同的佈建讀取容量單位與寫入容量單位來設定目標資料表，如請求備份時所記錄。例如，假設資料表的佈建輸送量最近降低至 50 個讀取容量單位及 50 個寫入容量單位。然後，您將資料表還原到三週前的狀態，當時的佈建輸送量設為 100 個讀取容量單位和 100 個寫入容量單位。在本例中，DynamoDB 會將資料表的資料還原到該時間點，並使用從該時間開始佈建的輸送量 (100 個讀取容量單位和 100 個寫入容量單位)。

您也可以跨區 AWS 域還原 DynamoDB 表格資料，以便在來源表格所在的不同區域中建立還原的表格。您可以在 AWS 商業區域、中國區域和 AWS GovCloud (美 AWS 國) 區域之間進行跨區域還原。您只需為從來源區域中傳輸出來的資料，以及還原為目標區域中的新資料表付費。

#### Note

如果來源或目的地區域是亞太區域 (香港) 或中東 (巴林)，則不支援跨區域還原。

如果您阻止在還原的資料表上建立部分或全部索引，可加速還原且更符合經濟效益。

您必須在還原的資料表上手動進行下列設定：

- 自動調整規模政策
- AWS Identity and Access Management (IAM) 政策
- Amazon CloudWatch 指標和警報
- 標籤

- 串流設定
- 存留時間 (TTL) 設定
- Point-in-time 復原設定
- 刪除保護設定

還原資料表的時間受多項因素影響。point-in-time 還原時間並不總是直接與表格的大小相關聯。如需詳細資訊，請參閱 [還原](#)。

## 刪除啟用 point-in-time 復原的表格

刪除已啟用 point-in-time 復原的表格時，DynamoDB 會自動建立稱為系統備份的備份快照，並將其保留 35 天 (無需額外費用)。您可以使用系統備份將已刪除的資料表還原到刪除之前的狀態。所有的系統備份都遵循 `###-###$DeletedTableBackup` 的標準命名慣例。

### Note

刪除啟用 point-in-time 恢復的表後，您可以使用系統還原將該表還原到單個時間點：刪除之前的那一刻。您無法將已刪除的表格還原到過去 35 天內任何時間點。

## 在您開始使用 point-in-time 復原之前

在 Amazon DynamoDB 表上啟用 point-in-time 復原 (PITR) 之前，請考慮下列事項：

- 如果您停用 point-in-time 復原並稍後在資料表上重新啟用它，則會重設可復原該資料表的開始時間。因此，您只能使用 `LatestRestorableDateTime` 立即還原該資料表。
- 您可以在全域表格的每個本機複本上啟用 point-in-time 復原功能。在還原資料表時，備份會還原到不屬於全域資料表的獨立資料表。如果您使用的是 [全域表 2019.11.21 \(目前\) 版本](#) 的全域表格，您可以從還原的表格建立新的全域表格。如需詳細資訊，請參閱 [全域資料表：運作方式](#)。
- 您也可以跨 AWS 區域還原 DynamoDB 資料表資料，將還原的資料表建立在不同於來源資料表所在的區域。您可以在 AWS 商業區域、中國區域和 AWS GovCloud (美AWS國) 區域之間進行跨區域還原。您只需為從來源區域中傳輸出來的資料，以及還原為目標區域中的新資料表付費。
- AWS CloudTrail 記錄所有控制台和 API 操作以進行 point-in-time 恢復，以啟用日誌記錄，持續監視和審計。如需更多詳細資訊，請參閱 [使用 AWS CloudTrail 記錄 DynamoDB 操作](#)。

## 還原 DynamoDB 資料表至某個時間點

Amazon DynamoDB 時間點復原 (PITR) 可持續備份 DynamoDB 資料表的資料。您可使用 DynamoDB 主控台或 AWS Command Line Interface (AWS CLI)，以將資料表還原至某個時間點。時間點復原過程會還原到新資料表。

如果您想要使用 AWS CLI，就必須先設定它。如需詳細資訊，請參閱 [存取 DynamoDB](#)。

### 主題

- [還原 DynamoDB 資料表至某個時間點 \(主控台\)](#)
- [還原資料表至某個時間點 \(AWS CLI\)](#)

## 還原 DynamoDB 資料表至某個時間點 (主控台)

以下範例示範如何使用 DynamoDB 主控台將名為 Music 的現有資料表還原至某個時間點。

### Note

此程序假設您已經啟用時間點復原。若要啟用 Music 資料表，在 Backups (備份) 索引標籤的 Point-in-time recover (PITR) (時間點復原) 區段中，選擇 Edit (編輯)，然後勾選 Enable point-in-time-recovery (啟用時間點復原)。

### 還原資料表至某個時間點

1. 登入 AWS Management Console，並在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 在資料表清單中，選擇 Music 資料表。
4. 在 Music 資料表 Backups (備份) 索引標籤上的 Point-in-time recovery (時間點復原) 區段，選擇 Restore (還原)。
5. 輸入 **MusicMinutesAgo** 做為新資料表的名稱。

### Note

您可以將資料表還原至相同的 AWS 區域，或還原至不同於來源資料表所在的區域。您也可以阻止在還原的資料表上建立次要索引。此外，您可以指定不同的加密模式。

- 若要確認可還原時間，請將還原日期和時間設為 Earliest (最近)。然後選擇 Restore (還原) 啟動還原程序。

正在還原的資料表會顯示為 Restoring (正在還原) 狀態。還原程序完成後，MusicMinutesAgo 資料表的狀態會變更為 Active (作用中)。

## 還原資料表至某個時間點 (AWS CLI)

以下程序顯示如何使用 AWS CLI 將名為 Music 的現有資料表還原至某個時間點。

### Note

此程序假設您已經啟用時間點復原。若要針對 Music 資料表啟用程序，請執行下列命令。

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

## 還原資料表至某個時間點

- 使用 Music 命令以確認為 describe-continuous-backups 資料表啟用了時間點復原。

```
aws dynamodb describe-continuous-backups \
 --table-name Music
```

連續備份 (在建立資料表時自動啟用) 和時間點復原已啟用。

```
{
 "ContinuousBackupsDescription": {
 "PointInTimeRecoveryDescription": {
 "PointInTimeRecoveryStatus": "ENABLED",
 "EarliestRestorableDateTime": 1519257118.0,
 "LatestRestorableDateTime": 1520018653.01
 },
 "ContinuousBackupsStatus": "ENABLED"
 }
}
```

2. 還原資料表至某個時間點。在本例中，Music 資料表會還原至相同 AWS 區域的 LatestRestorableDateTime (~5 分鐘前)。

```
aws dynamodb restore-table-to-point-in-time \
 --source-table-name Music \
 --target-table-name MusicMinutesAgo \
 --use-latest-restorable-time
```

#### Note

您也可還原至特定時間點。若要執行此作業，請使用 `--restore-date-time` 引數執行該命令，並指定時間戳記。您可以指定過去 35 天內的任何時間點。例如，下列命令可復原資料表至 `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
```

當復原到特定時間點時，指定 `--no-use-latest-restorable-time` 引數是選擇性的。

3. 利用自訂的資料表設定，將資料表還原至某個時間點。在此案例中，Music 資料表會還原至 LatestRestorableDateTime (~5分鐘以前)。

您可以為還原的資料表指定不同的加密模式，如下所示。

#### Note

`sse-specification-override` 參數採用的數值與 `CreateTable` 命令中使用的 `sse-specification-override` 參數相同。如需進一步了解，請參閱 [在 DynamoDB 中管理加密資料表](#)。

```
aws dynamodb restore-table-to-point-in-time \
 --source-table-name Music \
 --target-table-name MusicMinutesAgo \
 --use-latest-restorable-time \
 --sse-specification-override SSE_ON
```

```
--sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-a123-ab1234a1b234
```

您可以將資料表還原到不同於來源資料表所在的 AWS 區域。

#### Note

- 執行跨區域還原必須使用 `sse-specification-override` 參數，而還原至與來源資料表相同的區域時則可選用此參數。
- 執行跨區域還原必須提供 `source-table-arn` 參數。
- 從命令列執行跨區域還原時，必須將預設 AWS 區域設定為所需的目的地區域。如需進一步了解，請參閱《AWS Command Line Interface 使用者指南》中的[命令列選項](#)。

```
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
```

您可以覆寫帳單模式及佈建給還原資料表的輸送量。

```
aws dynamodb restore-table-to-point-in-time \
 --source-table-name Music \
 --target-table-name MusicMinutesAgo \
 --use-latest-restorable-time \
 --billing-mode-override PAY_PER_REQUEST
```

您可以阻止在還原的資料表上建立部分或全部次要索引。

#### Note

如果您阻止在新還原的資料表上建立部分或全部次要索引，可加速還原且更符合經濟效益。

```
aws dynamodb restore-table-to-point-in-time \
 --source-table-name Music \
 --target-table-name MusicMinutesAgo \
 --use-latest-restorable-time \
 --global-secondary-index-override '[]'
```

您可以使用不同覆寫的組合。例如，您可以使用單一全域次要索引，同時變更所佈建的輸送量，如下所示。

```
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_ONLY}" \
 \
 --sse-specification-override Enabled=true,SSEType=KMS \
 --use-latest-restorable-time
```

若要確認還原，請使用 `describe-table` 命令描述 `MusicEarliestRestorableDateTime` 資料表。

```
aws dynamodb describe-table --table-name MusicEarliestRestorableDateTime
```

正在還原的資料表會顯示為 `Creating` (正在建立) 的狀態，且復原會做為 `true` 進行。還原程序完成後，`MusicEarliestRestorableDateTime` 資料表的狀態會變更為 `Active` (作用中)。

### Important

復原過程中，請勿修改或刪除 AWS Identity and Access Management (IAM) 授予 IAM 實體的政策 (例如使用者、群組或角色) 許可以執行復原。否則，可能會造成意外行為。例如，假設您在資料表還原時移除資料表的寫入許可。在此案例中，基礎 `RestoreTableToPointInTime` 操作無法將任何還原的資料寫入資料表。涉及存取目標還原資料表的來源 IP 限制的 IAM 政策可能也會導致問題。

只有在復原操作完成後，才能修改或刪除許可。



# 使用 DynamoDB Accelerator (DAX) 的記憶體內加速

Amazon DynamoDB 是專為擴展與效能所設計。在大多數情況下，DynamoDB 回應時間可以測量到個位數毫秒。但是，有些使用案例需要以微秒為單位的回應時間。針對這些使用案例，DynamoDB Accelerator (DAX) 在存取最終一致資料時可提供快速的回應時間。

DAX 是與 DynamoDB 相容的快取服務，可讓您利用快速的記憶體內效能，滿足高需求的應用程式。DAX 可解決三種核心案例：

1. 作為記憶體內快取，DAX 會以十倍為單位 (從個位數毫秒到微秒) 來縮短最終一致讀取工作負載的回應時間。
2. DAX 可提供與 DynamoDB API 相容的受管服務，減少操作和應用程式的複雜度。因此，它只需要極少的功能變更，便能搭配現有應用程式使用。
3. 針對需要大量讀取或爆量的工作負載，DAX 會降低過度佈建讀取容量單位的需求，以此增加輸送量以及節省可能的操作成本。這對需要重複讀取個別索引鍵的應用程式特別有利。

DAX 支援伺服器端加密。使用靜態加密功能時，DAX 在磁碟上保留的資料會受到加密。DAX 將資料寫入磁碟，做為自主要節點到僅供讀取複本的變更環節之一。如需更多詳細資訊，請參閱 [DAX 靜態加密](#)。

DAX 也支援傳輸中加密功能，確保應用程式與叢集之間的所有請求和回應都經由 Transport Layer Security (TLS) 加密，並透過驗證叢集 x509 憑證對叢集的連線進行身分驗證。如需更多詳細資訊，請參閱 [DAX 傳輸中加密](#)。

## 主題

- [DAX 使用案例](#)
- [DAX 使用須知](#)
- [DAX 的運作方式](#)
- [DAX 叢集元件](#)
- [建立 DAX 叢集](#)
- [DAX 與 DynamoDB 一致性模式](#)
- [使用 DynamoDB Accelerator \(DAX\) 用戶端開發](#)
- [管理 DAX 叢集](#)
- [監控 DAX](#)
- [DAX T3/T2 爆量執行個體](#)

- [DAX 存取控制](#)
- [DAX 靜態加密](#)
- [DAX 傳輸中加密](#)
- [使用 DAX 的服務連結 IAM 角色](#)
- [跨多 AWS 帳戶存取 DAX](#)
- [DAX 叢集調整大小指南](#)
- [將 DAX 與 DynamoDB 搭配使用的最佳做法](#)
- [DAX API 參考](#)

## DAX 使用案例

DAX 可讓您以微秒延遲，存取 DynamoDB 資料表的最終一致資料。多可用區域 DAX 叢集每秒可處理數百萬個請求。

DAX 適合下列應用程式類型：

- 需要盡快讀取回應時間的應用程式。部分範例包括即時競標、社群遊戲與交易應用程式。DAX 為這些使用案例提供快速的記憶體內讀取效能。
- 比其他應用程式更常讀取少量項目的應用程式。例如，假設有一部電子商務系統，正對某項熱門產品展開一日促銷。在促銷期間，與其他所有產品相比，針對該產品 (及其在 DynamoDB 中的資料) 的需求會驟增。若要減輕「熱」鍵與不一致流量分佈的影響，您可以將讀取活動卸載至 DAX 快取，直到一日促銷結束為止。
- 需要大量讀取但對成本也很敏感的應用程式。透過 DynamoDB，您可以佈建應用程式所需的每秒讀取數目。如果讀取活動增加，您可以增加資料表的佈建讀取輸送量 (需額外付費)。或者，您可以將活動從應用程式卸載至 DAX 叢集，並減少需要額外購買的讀取容量單位數。
- 需要對大型資料集重複讀取的應用程式。這類應用程式可能會從其他應用程式重新分配資料庫資源。例如，長時間執行的區域天氣資料分析可能會暫時使用 DynamoDB 資料表中的所有讀取容量。這種情況會對其他需要存取相同資料的應用程式造成負面影響。透過 DAX，可改為對快取資料執行天氣分析。

DAX 不適合下列應用程式類型：

- 需要強烈一致讀取 (或無法容忍最終一致讀取) 的應用程式。
- 讀取回應時間不需要到微秒，或不需從基礎資料表卸載重複讀取活動的應用程式。
- 需要大量寫入或未執行太多讀取活動的應用程式。

- 已搭配 DynamoDB 使用不同快取解決方案，以及使用自己用戶端邏輯來使用該快取解決方案的應用程式。

## DAX 使用須知

- 如需提供 DAX 的 AWS 區域清單，請參閱 [Amazon DynamoDB 定價](#)。
- DAX 支援使用這些程式設計語言 AWS 所提供的用戶端，以圍棋、Java、Node.js、Python 和 .NET 編寫的應用程式。
- DAX 僅適用於 EC2-VPC 平台。
- DAX 叢集服務角色政策必須允許 dynamodb:DescribeTable 動作，才能維護有關 DynamoDB 資料表的中繼資料。
- DAX 叢集會維護其存放項目屬性名稱的中繼資料。它會無限期地維護該中繼資料 (即使項目過期或從快取中移出也一樣)。長期下來，使用屬性名稱未限制數量的應用程式可能會在 DAX 叢集中造成記憶體用盡。此限制僅適用於頂層屬性名稱，而非巢狀屬性名稱。有問題的頂層屬性名稱包括時間戳記、UUID 和工作階段 ID。

此限制僅適用於屬性名稱，而非其值。與以下相似的項目則不是問題。

```
{
 "Id": 123,
 "Title": "Bicycle 123",
 "CreationDate": "2017-10-24T01:02:03+00:00"
}
```

但與以下相似的項目，若數量夠多且每個都具有不同的時間戳記，則可能會造成問題。

```
{
 "Id": 123,
 "Title": "Bicycle 123",
 "2017-10-24T01:02:03+00:00": "created"
}
```

## DAX 的運作方式

Amazon DynamoDB Accelerator (DAX) 旨在於 Amazon Virtual Private Cloud (Amazon VPC) 環境內執行。Amazon VPC 服務會定義虛擬網路，與傳統資料中心幾乎一模一樣。使用 VPC，您可以控制其

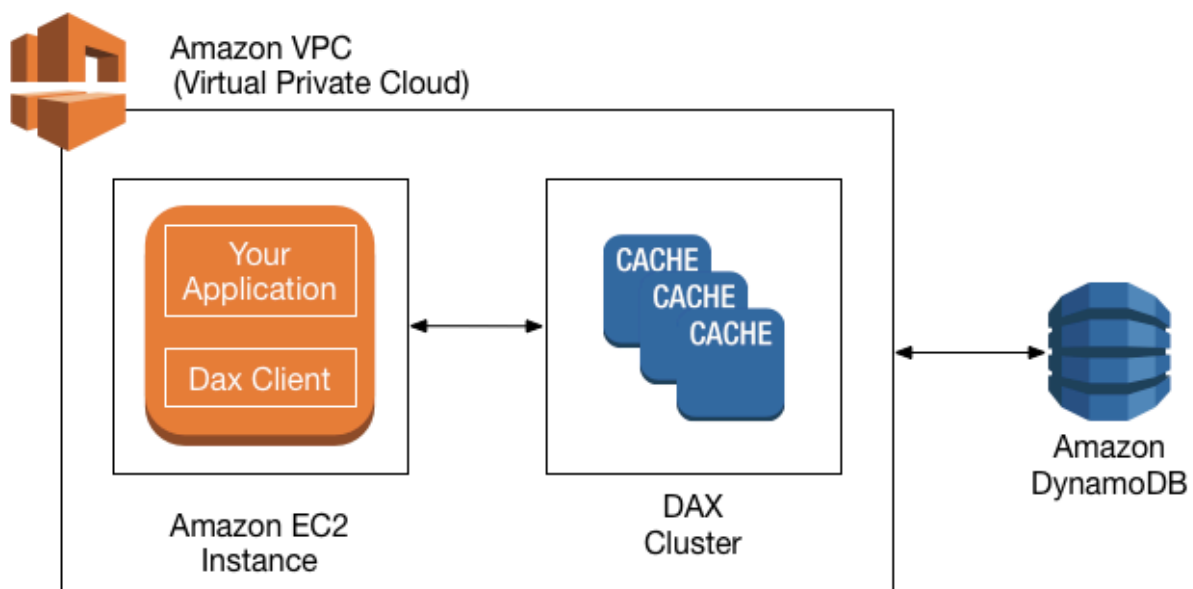
IP 地址範圍、子網路、路由表、網路閘道及安全設定。您可以在虛擬網路中啟動 DAX 叢集，然後使用 Amazon VPC 安全群組來控制對該叢集的存取。

### Note

如果您是在 2013 年 12 月 4 日之後建立 AWS 帳戶，則您在每個 AWS 區域中都會有預設的 VPC。VPC 已立即可供您使用，無須進行任何其他設定步驟。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的[預設 VPC 和預設子網路](#)。

下圖顯示 DAX 的高層級概觀。



若要建立 DAX 叢集，您可以使用 AWS Management Console。除非另有指定，否則您的 DAX 叢集會在您的預設 VPC 中執行。若要執行應用程式，請在 Amazon VPC 內啟動 Amazon EC2 執行個體。接著在 EC2 執行個體上部署應用程式 (使用 DAX 用戶端)。

DAX 用戶端會在執行時間，將應用程式的所有 DynamoDB API 請求導向該 DAX 叢集。若 DAX 可直接處理其中一個 API 請求，它便會執行該作業。否則，它會將請求傳遞給 DynamoDB。

DAX 叢集最後會將結果傳回您的應用程式。

### 主題

- [DAX 如何處理請求](#)
- [項目快取](#)
- [查詢快取](#)

## DAX 如何處理請求

DAX 叢集會包含一或多個節點。而每個節點都會執行自己的 DAX 快取軟體執行個體。其中一個節點會作為叢集的主要節點。其他節點 (如其存在) 則為讀取複本。如需詳細資訊，請參閱 [節點](#)。

您的應用程式可以透過指定 DAX 叢集的端點，存取 DAX。DAX 用戶端軟體可使用叢集端點，執行智慧型負載平衡和路由。

### 讀取操作

DAX 可以回應下列 API 呼叫：

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

若請求指定最終一致讀取 (預設行為)，它會嘗試從 DAX 讀取該項目：

- DAX 如有該項目 (快取命中)，DAX 不需存取 DynamoDB 就能將該項目傳回應用程式。
- DAX 若沒有該項目 (快取未中)，DAX 會將請求傳遞給 DynamoDB。當它從 DynamoDB 收到回應後，DAX 便會將結果傳回應用程式。但它也會將結果寫入主要節點上的快取。

#### Note

叢集中如有任何僅供讀取複本，DAX 會自動將該複本與主節點保持同步。如需詳細資訊，請參閱 [叢集](#)。

若請求指定強烈一致讀取，DAX 會將請求傳遞給 DynamoDB。DynamoDB 的結果將不會在 DAX 中進行快取。他們只會傳回應用程式。

## 寫入操作

下列 DAX API 操作會視為「全部寫入」：

- BatchWriteItem
- UpdateItem
- DeleteItem
- PutItem

這些操作會先將資料寫入 DynamoDB 資料表，然後再寫入 DAX 叢集。資料必須同時成功寫入資料表與 DAX，操作才算成功。

## 其他操作

DAX 無法識別任何管理資料表的 DynamoDB 操作 (例如 CreateTable、UpdateTable)。您的應用程式如需執行這些操作，它必須直接存取 DynamoDB 而非使用 DAX。

如需有關 DAX 和 DynamoDB 一致性的詳細資訊，請參閱 [DAX 與 DynamoDB 一致性模式](#)。

如需有關 DAX 中交易的運作方式資訊，請參閱 [使用 DynamoDB Accelerator \(DAX\) 中的交易 API](#)。

## 請求率限制

如果傳送至 DAX 的請求數超過節點的容量，DAX 會傳回 [ThrottlingException](#)，將速率限制為其接受其他請求的程度。DAX 會持續評估您的 CPU 使用率，以判斷其可以處理的請求量，同時維持良好的叢集狀態。

您可以監控 DAX 發佈至 Amazon CloudWatch 的 [ThrottledRequestCount 指標](#)。如果您每隔一段時間就會看到這些例外狀況，請考慮[擴展您的叢集](#)。

## 項目快取

DAX 會維護項目快取，存放來自 GetItem 及 BatchGetItem 操作的結果。快取中的項目代表來自 DynamoDB 的最終一致資料，並依其主索引鍵值存放。

應用程式傳送 GetItem 或 BatchGetItem 請求時，DAX 會嘗試使用指定的索引鍵值，直接從項目快取讀取項目。若找到項目 (快取命中)，DAX 會立即將其傳回應用程式。如果找不到項目 (快取未命中)，DAX 會將請求傳送至 DynamoDB。DynamoDB 使用最終一致讀取來處理請求，並將項目傳回 DAX。DAX 會將它們存放在項目快取中，然後傳回至應用程式。

項目快取有存留時間 (TTL) 設定，預設為 5 分鐘。DAX 會指派時間戳記給其寫入項目快取的每個快取。若項目存留在快取中的時間長於 TTL 設定，項目便會過期。若是對過期的項目發出 GetItem 請求，會將其視為快取未中。DAX 會因此而傳送 GetItem 請求給 DynamoDB。

#### Note

您可以在建立新的 DAX 叢集時，為項目快取指定 TTL 設定。如需詳細資訊，請參閱 [管理 DAX 叢集](#)。

DAX 也會為項目快取維護一份最近最少使用 (LRU) 的清單。LRU 清單會追蹤項目第一次寫入快取的時間，以及上一次從快取中讀取項目的時間。當項目快取變滿時，DAX 會移出較舊的項目 (無論項目過期與否)，以清出空間存放新的項目。LRU 演算法一律會為項目快取啟用，使用者無法加以設定。

如果您將項目快取 TTL 設定指定為零，項目快取中的項目僅會因 LRU 移出或[全部寫入](#)操作而重新整理。

如需 DAX 中項目快取的一致性詳細資訊，請參閱 [DAX 項目快取行為](#)。

## 查詢快取

DAX 也會維護查詢快取，存放來自 Query 及 Scan 操作的結果。此快取中的項目代表來自對 DynamoDB 資料表之查詢與掃描的結果集。這些結果集會依其參數值存放。

當應用程式傳送 Query 或 Scan 請求時，DAX 會嘗試使用指定的參數值，從查詢快取讀取相符的結果集。若有找到結果集 (快取命中)，DAX 會立即將其傳回應用程式。如果找不到結果集 (快取未中)，DAX 會將請求傳送至 DynamoDB。DynamoDB 使用最終一致讀取來處理請求，並將結果集傳回 DAX。DAX 會將它存放在查詢快取中，然後傳回至應用程式。

#### Note

您可以在建立新的 DAX 叢集時，為查詢快取指定 TTL 設定。如需詳細資訊，請參閱 [管理 DAX 叢集](#)。

DAX 也會為查詢快取維護一份 LRU 清單。清單會追蹤結果集第一次寫入快取的時間，以及上一次從快取讀取結果的時間。當查詢快取變滿時，DAX 會移出較舊的結果集 (無論結果集過期與否)，以清出空間存放新的結果集。LRU 演算法一律會為查詢快取啟用，使用者無法加以設定。

如果您將查詢快取 TTL 設定指定為零，查詢回應將不會快取。

如需 DAX 中查詢快取的一致性詳細資訊，請參閱 [DAX 查詢快取行為](#)。

## DAX 叢集元件

Amazon DynamoDB Accelerator (DAX) 叢集由 AWS 基礎設施元件組成。本節說明這些元件及其運作方式。

### 主題

- [節點](#)
- [叢集](#)
- [區域與可用區域](#)
- [參數群組](#)
- [安全群組](#)
- [叢集 ARN](#)
- [叢集端點](#)
- [節點端點](#)
- [子網路群組](#)
- [事件](#)
- [Maintenance window \(維護時段\)](#)

## 節點

節點是 DAX 叢集最小的建置區塊。每個節點都會執行 DAX 軟體的執行個體，並且維持快取資料的單一複本。

您可以使用下列兩種方式的其中之一，擴展您的 DAX 叢集：

- 為叢集新增更多節點。這會增加叢集整體的讀取輸送量。
- 使用更大的節點類型。更大的節點類型可提供更多容量，並可以增加輸送量。(您必須使用新的節點類型建立新叢集。)

叢集中每個節點的節點類型都相同，並且會執行相同的 DAX 快取軟體。如需可用的節點類型清單，請參閱 [Amazon DynamoDB 定價](#)。



## 叢集

叢集是 DAX 作為單一單位管理的一或多個節點的邏輯群組。叢集中的其中一個節點會指定為主要節點，其他節點 (若有的話) 則為僅供讀取複本。

主要節點負責下列項目：

- 滿足應用程式對快取資料的請求。
- 處理對 DynamoDB 的寫入操作。
- 根據叢集的移出政策，從快取中移出資料。

對主節點上的快取資料進行變更時，DAX 會使用複寫日誌將變更散佈到所有僅供讀取複本節點。當所有僅供讀取複本收到確認後，DynamoDB 會從主節點刪除複寫日誌。

僅供讀取複本負責下列項目：

- 滿足應用程式對快取資料的請求。
- 根據叢集的移出政策，從快取中移出資料。

但是，與主節點不同的是，僅供讀取複本不會對 DynamoDB 進行寫入。

僅供讀取複本還有另外兩種用途：

- 延展性。若您有大量的用戶端需要並行存取 DAX，您可以新增更多複本進行讀取擴展。DAX 會將負載平均分佈到叢集中的所有節點。(另一種增加輸送量的方式是使用更大的快取節點類型。)
- 高可用性。在主節點故障時，DAX 會自動容錯移轉至僅供讀取複本，並將其指派為新的主節點。若複本節點失敗，DAX 叢集中的其他節點仍然可以處理請求，直到可復原失敗的節點為止。為取得最大的容錯能力，建議您將僅供讀取複本部署在分離的可用區域中。這項組態可確保即使整個可用區域都不可使用時，DAX 叢集仍然能夠繼續運作。

DAX 叢集最多可支援每個叢集 11 個節點 (主節點，加上最多 10 個僅供讀取複本)。

### Important

針對生產用途，我們強烈建議搭配至少三個節點使用 DAX，並將每個節點置放在不同的可用區域中。要容錯的 DAX 叢集需要三個節點。

可使用一或兩個節點來部署 DAX 叢集，以供開發或測試工作負載使用。一和兩個節點叢集無法容錯，因此針對生產用途，我們不建議使用少於三個節點。若一或兩個節點叢集發生軟體或硬體錯誤，叢集可能會無法使用或遺失快取資料。

## 區域與可用區域

AWS 區域中的 DAX 叢集只能與位於相同區域的 DynamoDB 資料表互動。因此，請確認您在正確的區域中啟動您的 DAX 叢集。若您在其他區域中具有 DynamoDB 資料表，您必須在那些區域中啟動 DAX 叢集。

每個區域皆設計為與其他區域完全隔離。在每個區域中皆有多個可用區域。藉由在不同的可用區域中啟動您的節點，您可以實現最大的容錯能力。

### Important

請不要將您所有的叢集節點置放在單一可用區域中。在這項組態中，若發生可用區域故障的情況，您的 DAX 叢集將會無法使用。

針對生產用途，我們強烈建議搭配至少三個節點使用 DAX，並將每個節點置放在不同的可用區域中。要容錯的 DAX 叢集需要三個節點。

可使用一或兩個節點來部署 DAX 叢集，以供開發或測試工作負載使用。一和兩個節點叢集無法容錯，因此針對生產用途，我們不建議使用少於三個節點。若一或兩個節點叢集發生軟體或硬體錯誤，叢集可能會無法使用或遺失快取資料。

## 參數群組

「參數群組」可用來管理 DAX 叢集的執行時間設定。DAX 有數個參數，可讓您用來最佳化效能 (例如定義快取資料的 TTL 政策)。參數群組為可以套用到叢集的具名參數組。藉由執行此作業，您可以確保該叢集中的所有節點都以完全相同的方式設定。

## 安全群組

DAX 叢集會在 Amazon Virtual Private Cloud (Amazon VPC) 環境中執行。此環境是一個您 AWS 帳戶的專用虛擬網路，與其他 VPC 隔離。「安全群組」做為 VPC 的虛擬防火牆，可讓您控制傳入及傳出網路流量。

當您在 VPC 中啟動叢集時，您會為您的安全群組新增「輸入」規則，允許傳入網路流量。輸入規則會指定您叢集的通訊協定 (TCP) 和連接埠號碼 (8111)。在為您的安全群組新增此規則後，在您 VPC 中執行的應用程式便可存取 DAX 叢集。

## 叢集 ARN

每個 DAX 叢集都會獲得指派一個 Amazon 資源名稱 (ARN)。ARN 格式如下。

```
arn:aws:dax:region:accountID:cache/clusterName
```

您可以在 IAM 政策中使用叢集 ARN，定義 DAX API 操作的許可。如需更多詳細資訊，請參閱 [DAX 存取控制](#)。

## 叢集端點

每個 DAX 叢集都會提供「叢集端點」供您的應用程式使用。藉由使用此端點存取叢集，您的應用程式便不需要知道叢集中個別節點的主機名稱和連接埠號碼。您的應用程式會自動「得知」叢集中的所有節點，即使您新增或移除僅供讀取複本。

以下是 us-east-1 區域中未設為使用傳輸中加密的叢集端點範例。

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

以下是相同區域中未設為使用傳輸中加密的叢集端點範例。

```
daxs://my-encrypted-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

## 節點端點

DAX 叢集上的每個個別節點都有其自身的主機名稱和連接埠號碼。以下是「節點端點」的範例。

```
myDAXcluster-a.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111
```

您的應用程式可使用其端點直接存取節點。但是，我們建議您將 DAX 叢集作為單一單位處理，並改為使用叢集端點進行存取。叢集端點可讓您的應用程式無須維持節點的清單，並在您為叢集新增節點或從叢集移除節點時，將該清單保持在最新狀態。

## 子網路群組

對 DAX 叢集節點的存取會限制於在 Amazon VPC 環境中 Amazon EC2 執行個體上執行的應用程式。您可以使用「子網路群組」授予叢集從特定子網路上所執行 Amazon EC2 執行個體進行存取的權限。

子網路群組是子網路的集合 (一般是私有)，您可以為在 Amazon VPC 環境中執行的叢集指定這些子網路。

建立 DAX 叢集時，您必須指定子網路群組。DAX 會使用此子網路群組選取子網路及該子網路中的 IP 地址，以與您的節點建立關聯。

## 事件

DAX 會記錄您叢集內重要的事件，例如新增節點失敗、新增節點成功，或是變更安全群組。藉由監控重要的事件，您可以了解叢集目前的狀態，並根據事件採取正確的動作。您可以使用 DAX 管理 API 中的 AWS Management Console 或 DescribeEvents 動作存取這些事件。

您也可以請求將通知傳送至特定的 Amazon Simple Notification Service (Amazon SNS) 主題。您接著便會在您的 DAX 叢集中發生事件時立即得知。

## Maintenance window (維護時段)

每個叢集都有每週一次的維護時段，會在此期間套用任何系統變更。若您在建立或修改快取叢集時並未指定偏好的維護時段，則 DAX 會在該週隨機選取一天並指派 60 分鐘的維護時段。

60 分鐘的維護時段是從每個 AWS 區域之 8 小時時段內隨機選取的。下表列出每個區域的時段，預設維護時段會從此時段中指派。

| 區域代碼           | 區域名稱          | Maintenance window (維護時段)       |
|----------------|---------------|---------------------------------|
| ap-northeast-1 | 亞太區域 (東京)     | 下午 1 時至 9 時 (UTC)               |
| ap-southeast-1 | 亞太區域 (新加坡) 區域 | 下午 2 時至 10 時 (UTC)              |
| ap-southeast-2 | 亞太區域 (雪梨) 區域  | 中午 12 時至下午 8 時 (UTC)            |
| ap-south-1     | 亞太區域 (孟買) 區域  | 下午 5 時 30 分至次日上午 1 時 30 分 (UTC) |
| cn-northwest-1 | 中國 (寧夏) 區域    | 下午 11 時至次日上午 7 時 (UTC)          |
| cn-north-1     | 中國 (北京) 區域    | 下午 2 時至 10 時 (UTC)              |

| 區域代碼         | 區域名稱              | Maintenance window (維護時段) |
|--------------|-------------------|---------------------------|
| eu-central-1 | 歐洲 (法蘭克福) 區域      | 下午 11 時至次日上午 7 時 (UTC)    |
| eu-west-1    | 歐洲 (愛爾蘭) 區域       | 下午 10 時至次日上午 6 時 (UTC)    |
| eu-west-2    | 歐洲 (倫敦) 區域        | 下午 11 時至次日上午 7 時 (UTC)    |
| eu-west-3    | 歐洲 (巴黎) 區域        | 下午 11 時至次日上午 7 時 (UTC)    |
| sa-east-1    | 南美洲 (聖保羅) 區域      | 上午 1 時至上午 9 時 (UTC)       |
| us-east-1    | 美國東部 (維吉尼亞北部) 區域  | 上午 3 時至上午 11 時 (UTC)      |
| us-east-2    | 美國東部 (俄亥俄) 區域     | 下午 11 時至次日上午 7 時 (UTC)    |
| us-west-1    | 美國西部 (加利佛尼亞北部) 區域 | 上午 6 時至下午 2 時 (UTC)       |
| us-west-2    | 美國西部 (奧勒岡) 區域     | 上午 6 時至下午 2 時 (UTC)       |

維護時段應落在使用量最低的時段，因此可能需要不時進行調整。您可以指定最高達 24 小時的時間範圍，並讓任何您請求的維護活動在此期間進行。

## 建立 DAX 叢集

本節會帶您逐步在預設 Amazon Virtual Private Cloud (Amazon VPC) 環境中首次設定和使用 Amazon DynamoDB Accelerator (DAX)。您可以使用 AWS Command Line Interface (AWS CLI) 或 AWS Management Console 來建立您的第一個 DAX 叢集。

在您建立 DAX 叢集之後，您可以從相同 VPC 中執行的 Amazon EC2 執行個體存取它。您接著可以搭配應用程式使用您的 DAX 叢集。如需詳細資訊，請參閱 [使用 DynamoDB Accelerator \(DAX\) 用戶端開發](#)。

## 主題

- [為 DAX 建立可存取 DynamoDB 的 IAM 服務角色](#)
- [使用 AWS CLI 建立 DAX 叢集](#)
- [使用 AWS Management Console 建立 DAX 叢集](#)

## 為 DAX 建立可存取 DynamoDB 的 IAM 服務角色

若要讓 DAX 叢集代您存取 DynamoDB 資料表，您必須建立服務角色。服務角色是 AWS Identity and Access Management (IAM) 角色，可授權 AWS 服務代您執行作業。服務角色可允許 DAX 存取您的 DynamoDB 資料表，就像是您自己存取資料表一般。您需要建立服務角色，才能建立 DAX 叢集。

如果您是使用主控台，則建立叢集的工作流程會檢查是否有既有 DAX 服務角色。如果找不到，則主控台會為您建立新的服務角色。如需詳細資訊，請參閱 [the section called “步驟 2：建立 DAX 叢集”](#)。

如果您使用的是 AWS CLI，則必須指定您先前建立的 DAX 服務角色。否則，您需要事先建立新的服務角色。如需詳細資訊，請參閱 [步驟 1：使用 AWS CLI 為 DAX 建立可存取 DynamoDB 的 IAM 服務角色](#)。

### 建立服務角色所需的許可

AWS 受管 AdministratorAccess 政策提供了建立 DAX 叢集和服務角色所有需要的許可。若您的使用者已連接 AdministratorAccess，您便無需採取進一步的動作。

否則，您必須將下列許可新增至您的 IAM 政策，讓您的使用者可以建立服務角色：

- iam:CreateRole
- iam:CreatePolicy
- iam:AttachRolePolicy
- iam:PassRole

將這些許可連接到嘗試執行動作的使用者。

#### Note

iam:CreateRole、iam:CreatePolicy、iam:AttachRolePolicy 和 iam:PassRole 許可未包含在 DynamoDB 的 AWS 受管政策中。這是根據方案設計，因為這些許可可能會提供

權限提升：也就是使用者可以使用這些許可來建立新的管理員政策，然後將該政策連接至現有角色。因此，您 (DAX 叢集管理員) 必須將這些許可明確地新增至您的政策。

## 疑難排解

若您的使用者政策遺漏 `iam:CreateRole`、`iam:CreatePolicy` 和 `iam:AttachPolicy` 許可，您將會收到錯誤訊息。下表列出這些訊息，並說明如何修正問題。

| 如果您看到此錯誤訊息...                                                                                                                                                                            | 請執行下列動作：                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| User: arn:aws:iam:: <i>accountID</i> :user/ <i>userName</i> is not authorized to perform: iam:CreateRole on resource: arn:aws:iam:: <i>accountID</i> :role/service-role/ <i>roleName</i> | 將 <code>iam:CreateRole</code> 新增至使用者政策。       |
| User: arn:aws:iam:: <i>accountID</i> :user/ <i>userName</i> is not authorized to perform: iam:CreatePolicy on resource: policy <i>policyName</i>                                         | 將 <code>iam:CreatePolicy</code> 新增至使用者政策。     |
| User: arn:aws:iam:: <i>accountID</i> :user/ <i>userName</i> is not authorized to perform: iam:AttachRolePolicy on resource: role <i>daxServiceRole</i>                                   | 將 <code>iam:AttachRolePolicy</code> 新增至使用者政策。 |

如需 DAX 叢集管理所需 IAM 政策的詳細資訊，請參閱 [DAX 存取控制](#)。

## 使用 AWS CLI 建立 DAX 叢集

本節說明如何使用 AWS Command Line Interface (AWS CLI) 建立 Amazon DynamoDB Accelerator (DAX) 叢集。若您尚未執行此作業，您必須安裝及設定 AWS CLI。若要執行此作業，請參閱 AWS Command Line Interface 使用者指南中的以下說明：

- [安裝 AWS CLI](#)
- [設定 AWS CLI](#)

**⚠ Important**

若要利用 AWS CLI 管理 DAX 叢集，請安裝或升級至版本 1.11.110 或更新版本。

所有 AWS CLI 範例都會使用 us-west-2 區域及虛擬的帳戶 ID。

**主題**

- [步驟 1：使用 AWS CLI 為 DAX 建立可存取 DynamoDB 的 IAM 服務角色](#)
- [步驟 2：建立子網路群組](#)
- [步驟 3：使用 AWS CLI 建立 DAX 叢集](#)
- [步驟 4：使用 AWS CLI 設定安全群組傳入規則](#)

**步驟 1：使用 AWS CLI 為 DAX 建立可存取 DynamoDB 的 IAM 服務角色**

您必須先為 Amazon DynamoDB Accelerator (DAX) 叢集建立服務角色，才可建立該叢集。服務角色是 AWS Identity and Access Management (IAM) 角色，可授權 AWS 服務代您執行作業。服務角色可允許 DAX 存取您的 DynamoDB 資料表，就像是您自己存取資料表一般。

在此步驟中，您會建立 IAM 政策，然後將該政策連接到 IAM 角色。這可讓您將角色指派給 DAX 叢集，使其可代您執行 DynamoDB 操作。

**建立 DAX 的 IAM 服務角色**

1. 使用下列內容建立名為 service-trust-relationship.json 的檔案。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "dax.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
```

2. 建立服務角色。



```
aws iam create-role \
 --role-name DAXServiceRoleForDynamoDBAccess \
 --assume-role-policy-document file://service-trust-relationship.json
```

3. 使用下列內容建立名為 `service-role-policy.json` 的檔案。

```
{
 "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:*"
]
 }
]
}
```

使用您的 AWS 帳戶 ID 取代 *accountID*。若要尋找您的 AWS 帳戶 ID，請在主控台的右上角，選擇您的登入 ID。您的 AWS 帳戶 ID 會出現在下拉式選單中。

在範例中的 Amazon Resource Name (ARN) 里，*accountID* 必須是 12 位數字。請勿使用連字號或其他任何標點符號。

4. 建立服務角色的 IAM 政策。

```
aws iam create-policy \
 --policy-name DAXServicePolicyForDynamoDBAccess \
 --policy-document file://service-role-policy.json
```

在輸出中，記下您所建立政策的 ARN，如以下範例所示。

```
arn:aws:iam::123456789012:policy/DAXServicePolicyForDynamoDBAccess
```

5. 將該政策連接到服務角色。將以下程式碼中的 *arn* 取代成先前步驟中的實際角色 ARN。

```
aws iam attach-role-policy \
 --role-name DAXServiceRoleForDynamoDBAccess \
 --policy-arn arn
```

接下來，您需要為您的預設 VPC 指定子網路群組。子網路群組是您 VPC 中一或多個子網路的集合。請參閱 [步驟 2：建立子網路群組](#)。

## 步驟 2：建立子網路群組

請按照此程序操作，來使用 AWS Command Line Interface (AWS CLI) 為您的 Amazon DynamoDB Accelerator (DAX) 叢集建立子網路群組。

### Note

如果您已建立預設 VPC 的子網路群組，則可略過此步驟。

DAX 旨在於 Amazon Virtual Private Cloud (Amazon VPC) 環境內執行。如果您是在 2013 年 12 月 4 日之後建立 AWS 帳戶，則您在每個 AWS 區域中都會有預設的 VPC。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 [預設 VPC 和預設子網路](#)。

## 建立子網路群組

1. 若要判斷您預設 VPC 的識別符，請輸入以下命令。

```
aws ec2 describe-vpcs
```

在輸出中，記下您預設 VPC 的識別符，如以下範例所示。

```
vpc-12345678
```

2. 判斷與您預設 VPC 相關聯的子網路 ID。使用您的實際 VPC ID (例如 vpc-12345678) 取代 *vpcID*。

```
aws ec2 describe-subnets \
 --filters "Name=vpc-id,Values=vpcID" \
 --query "Subnets[*].SubnetId"
```

在輸出中，記下子網路識別碼 (例如 subnet-11111111)。

3. 建立子網路群組。確認您在 `--subnet-ids` 參數中至少指定一個子網路 ID。

```
aws dax create-subnet-group \
 --subnet-group-name my-subnet-group \
 --subnet-ids subnet-11111111 subnet-22222222 subnet-33333333 subnet-44444444
```

若要建立叢集，請參閱 [步驟 3：使用 AWS CLI 建立 DAX 叢集](#)。

### 步驟 3：使用 AWS CLI 建立 DAX 叢集

請遵循此程序，使用 AWS Command Line Interface (AWS CLI) 在您的預設 Amazon VPC 中建立 Amazon DynamoDB Accelerator (DAX) 叢集。

#### 建立 DAX 叢集

1. 取得您服務角色的 Amazon Resource Name (ARN)。

```
aws iam get-role \
 --role-name DAXServiceRoleForDynamoDBAccess \
 --query "Role.Arn" --output text
```

在輸出中，記下服務角色 ARN，如以下範例所示。

```
arn:aws:iam::123456789012:role/DAXServiceRoleForDynamoDBAccess
```

2. 建立 DAX 叢集。使用前一步驟中的 ARN，取代 *roleARN*。

```
aws dax create-cluster \
 --cluster-name mydaxcluster \
 --node-type dax.r4.large \
 --replication-factor 3 \
 --iam-role-arn roleARN \
 --subnet-group my-subnet-group \
 --sse-specification Enabled=true \
 --output text
```

```
--region us-west-2
```

叢集中的所有節點類型皆為 `dax.r4.large` (`--node-type`)。共有三個節點 (`--replication-factor`)：一個主節點和兩個複本。

#### Note

由於 `sudo` 和 `grep` 是保留的關鍵字，您無法在叢集名稱中使用這些字詞來建立 DAX 叢集。例如，`sudo` 和 `sudocluster` 是無效的叢集名稱。

若要檢視叢集狀態，請輸入以下命令。

```
aws dax describe-clusters
```

狀態會在輸出中顯示，例如 `"Status": "creating"`。

#### Note

建立叢集需要幾分鐘。當叢集準備就緒時，其狀態會變更為 `available`。同時，請繼續進行 [步驟 4：使用 AWS CLI 設定安全群組傳入規則](#)，並遵循其中的說明。

## 步驟 4：使用 AWS CLI 設定安全群組傳入規則

您 Amazon DynamoDB Accelerator (DAX) 叢集中的節點會使用您 Amazon VPC 的預設安全群組。對於預設安全群組，您必須為未加密的叢集授權 TCP 連接埠 8111 的輸入流量，或為加密的叢集授權連接埠 9111 的輸入流量。這可允許 Amazon VPC 中的 Amazon EC2 執行個體存取您的 DAX 叢集。

#### Note

若使用不同的安全群組 (而非 `default`) 啟動您的 DAX 叢集，您必須改為該群組執行此程序。

### 設定安全群組傳入規則

1. 若要判斷預設安全群組識別符，請輸入以下命令。使用您實際的 VPC ID (來自 [步驟 2：建立子網路群組](#)) 取代 `vpcID`。

```
aws ec2 describe-security-groups \
 --filters Name=vpc-id,Values=vpcID Name=group-name,Values=default \
 --query "SecurityGroups[*].{GroupName:GroupName,GroupId:GroupId}"
```

在輸出中，記下安全群組識別碼 (例如 `sg-01234567`)。

2. 然後輸入以下內容。將 *sgID* 取代成您實際的安全群組識別符。請將連接埠 8111 用於未加密的叢集，再將 9111 用於加密的叢集。

```
aws ec2 authorize-security-group-ingress \
 --group-id sgID --protocol tcp --port 8111
```

## 使用 AWS Management Console 建立 DAX 叢集

本節說明如何使用 AWS Management Console 建立 Amazon DynamoDB Accelerator (DAX) 叢集。

### 主題

- [步驟 1：使用 AWS Management Console 建立子網路群組](#)
- [步驟 2：使用 AWS Management Console 建立 DAX 叢集](#)
- [步驟 3：使用 AWS Management Console 設定安全群組傳入規則](#)

### 步驟 1：使用 AWS Management Console 建立子網路群組

請按照此程序操作，來使用 AWS Management Console 為您的 Amazon DynamoDB Accelerator (DAX) 叢集建立子網路群組。

#### Note

如果您已建立預設 VPC 的子網路群組，則可略過此步驟。

DAX 旨在於 Amazon Virtual Private Cloud (Amazon VPC) 環境內執行。如果您是在 2013 年 12 月 4 日之後建立 AWS 帳戶，則您在每個 AWS 區域中都會有預設的 VPC。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的[預設 VPC 和預設子網路](#)。

您必須在建立 DAX 叢集時指定子網路群組，這是建立程序的一部分。子網路群組是您 VPC 中一或多個子網路的集合。當您建立 DAX 叢集時，節點會部署到子網路群組內的子網路。

## 建立子網路群組

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在導覽窗格中，選擇 DAX 下方的 Subnet groups (子網路群組)。
3. 選擇 Create subnet group (建立子網路群組)。
4. 在 Create subnet group (建立子網路群組) 視窗中，執行下列作業：
  - a. Name (名稱)：輸入子網路群組的短名稱。
  - b. Description (描述)：輸入子網路群組的描述。
  - c. VPC ID：為您的 Amazon VPC 環境選擇識別碼。
  - d. Subnets (子網路)：從清單中選擇一或多個子網路。

### Note

子網路會分佈在多個可用區域中。如果您計劃建立多節點的 DAX 叢集 (主節點和一或多個僅供讀取複本)，建議您選擇多個子網路 ID。DAX 接著便可以將叢集節點部署到多個可用區域。若可用區域無法使用，DAX 會自動容錯移轉至仍然存活的可用區域。您的 DAX 叢集會繼續運作，而不會發生中斷。

當您滿意設定後，請選擇 Create subnet group (建立子網路群組)。

若要建立叢集，請參閱 [步驟 2：使用 AWS Management Console 建立 DAX 叢集](#)。

## 步驟 2：使用 AWS Management Console 建立 DAX 叢集

請遵循此程序，在您的預設 Amazon VPC 中建立 Amazon DynamoDB Accelerator (DAX) 叢集。

### 建立 DAX 叢集

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在導覽窗格中，選擇 DAX 下方的 Clusters (叢集)。
3. 選擇 Create cluster (建立叢集)。
4. 在 Create cluster (建立叢集) 視窗中，執行下列作業：
  - a. Cluster name (叢集名稱)：輸入您 DAX 叢集的短名稱。

**Note**

由於 `sudo` 和 `grep` 是保留的關鍵字，您無法在叢集名稱中使用這些字詞來建立 DAX 叢集。例如，`sudo` 和 `sudocluster` 是無效的叢集名稱。

- b. Cluster description (叢集描述)：輸入叢集的描述。
- c. Node type (節點類型)：為叢集中的所有節點選擇節點類型。
- d. Cluster size (叢集大小)：選擇叢集中的節點數。叢集組成包含一個主要節點和最多九個僅供讀取複本。

**Note**

如果想要建立單節點叢集，請選擇 1。您的叢集會包含一個主要節點。  
如果您想要建立多節點叢集，請選擇 3 (一個主節點和兩個讀取複本) 到 10 (一個主節點和九個讀取複本) 之間的數字。

**Important**

針對生產用途，我們強烈建議搭配至少三個節點使用 DAX，其中每個節點都置放在不同的可用區域中。要容錯的 DAX 叢集需要三個節點。

可使用一或兩個節點來部署 DAX 叢集，以供開發或測試工作負載使用。一和兩個節點叢集無法容錯，因此針對生產用途，我們不建議使用少於三個節點。若一或兩個節點叢集發生軟體或硬體錯誤，叢集可能會無法使用或遺失快取資料。

- e. 選擇 Next (下一步)。
- f. Subnet group (子網路群組)：選取 Choose existing (選擇現有)，然後選擇您在 [步驟 1：使用 AWS Management Console 建立子網路群組](#) 建立的子網路群組。
- g. 存取控制：選擇 default (預設值) 安全群組
- h. 可用區域 (AZ)：選擇 Automatic (自動)。
- i. 選擇下一步。
- j. IAM service role for DynamoDB access (DynamoDB 存取的 IAM 服務角色服務角色)：選擇 Create new (建立新的)，然後輸入以下資訊：

- IAM role name (IAM 角色名稱)：輸入 IAM 角色的名稱 (例如 DAXServiceRole)。主控台會建立新的 IAM 角色，且您的 DAX 叢集會在執行時間取得此角色。
  - 選取 Create policy (建立政策) 旁的方框。
  - IAM role policy (IAM 角色政策)：選擇 Read/Write (讀取/寫入)。這會允許 DAX 叢集在 DynamoDB 中執行讀取及寫入操作。
  - New IAM policy name (新 IAM 政策名稱)：此欄位會在您輸入 IAM 角色名稱時填入。您也可以輸入 IAM 政策的名称，例如 DAXServicePolicy。主控台即建立新的 IAM 政策，並將政策連接至 IAM 角色。
  - Access to DynamoDB tables (存取 DynamoDB 資料表)：選擇 All tables (所有資料表)。
- k. Encryption (加密)：選擇 Turn on encryption at rest (開啟靜態加密) 和 Turn on encryption in transit (開啟傳輸中加密)。如需詳細資訊，請參閱 [DAX 靜態加密](#) 和 [DAX 傳輸中加密](#)。

DAX 也需要個別的服務角色才能存取 Amazon EC2。DAX 會自動為您建立此服務角色。如需詳細資訊，請參閱[使用 DAX 的服務連結角色](#)。

5. 當您滿意設定後，選擇 Next (下一步)。
6. Parameter group (參數群組)：選擇 Choose existing (選擇現有)。
7. Maintenance window (維護時段)：若您對於套用的軟體更新沒有偏好，選擇 No preference (無偏好設定)，或選擇 Specify time window (指定時間範圍)，並針對維護時段的排程選項選擇 Weekday (工作日)、Time (UTC) (國際標準時間) 和 Start within (hours) (幾小時後開始)。
8. Tags (標籤)：選擇 Add new tag (新增標籤) 輸入鍵值對，以進行標記。
9. 選擇 Next (下一步)。

在 Review and create (檢閱和建立) 畫面上，您可以查看所有設定。若您已準備好建立叢集，請選擇 Create cluster (建立叢集)。

在 Clusters (叢集) 畫面中，您的 DAX 叢集會列出並顯示 Creating (正在建立) 狀態。

#### Note

建立叢集需要幾分鐘。當叢集準備就緒時，其狀態會變更為 Available (可用)。同時，請繼續進行 [步驟 3：使用 AWS Management Console 設定安全群組傳入規則](#)，並遵循其中的說明。



## 步驟 3：使用 AWS Management Console 設定安全群組傳入規則

您的 Amazon DynamoDB Accelerator (DAX) 叢集透過 TCP 連接埠 8111 (用於未加密的叢集) 或 9111 (用於加密的叢集) 進行通訊，因此您需要授權該連接埠的輸入流量。這可允許 Amazon VPC 中的 Amazon EC2 執行個體存取您的 DAX 叢集。

### Note

若使用不同的安全群組 (而非 default) 啟動您的 DAX 叢集，您必須改為該群組執行此程序。

### 設定安全群組傳入規則

1. 在 <https://console.aws.amazon.com/ec2/> 開啟 Amazon EC2 主控台。
2. 在導覽窗格中，選擇 Security Groups (安全群組)。
3. 選擇 default (預設) 安全群組。在 Actions (動作) 選單上，選擇 Edit inbound rules (編輯傳入規則)。
4. 選擇 Add Rule (新增規則) 並輸入以下資訊：
  - Port Range (連接埠範圍)：輸入 8111 (如果您的叢集未加密) 或 9111 (如果您的叢集已加密)。
  - Source (來源)：將此項保留為 Custom (自訂)，然後選擇右側的搜尋欄位，即會顯示下拉式選單。選擇預設安全群組的識別符。
5. 選擇 Save rules (儲存規則) 儲存變更。
6. 若要更新主控台的名稱，請前往 Name (名稱) 屬性並選擇顯示的 Edit (編輯) 選項。

## DAX 與 DynamoDB 一致性模式

Amazon DynamoDB Accelerator (DAX) 是全部寫入快取服務，設計目的是要簡化將快取新增至 DynamoDB 資料表的程序。因為 DAX 與 DynamoDB 分開操作，所以您務必要了解 DAX 和 DynamoDB 的一致性模式，確定應用程式如預期地運作。

在許多使用案例中，應用程式使用 DAX 的方式會影響 DAX 叢集內資料的一致性，以及 DAX 與 DynamoDB 之間資料的一致性。

### 主題

- [DAX 叢集節點之間的一致性](#)

- [DAX 項目快取行為](#)
- [DAX 查詢快取行為](#)
- [強烈一致和交易讀取](#)
- [負快取](#)
- [寫入策略](#)

## DAX 叢集節點之間的一致性

若要達到應用程式的高可用性，建議您使用至少三個節點來佈建 DAX 叢集。然後將這些節點置放到區域中的多個可用區域內。

DAX 叢集執行時，會複寫叢集中所有節點之間的資料 (假設您已佈建超過一個節點)。請考慮使用 DAX 執行成功 UpdateItem 的應用程式。此動作會使用新的值修改主要節點中的項目快取。該值接著會複寫到叢集中的所有其他節點。這項複寫最終會一致，而且通常不需要一秒就能完成。

在此情況下，根據每個用戶端所存取的節點，兩個用戶端可能會讀取相同 DAX 叢集中的相同索引鍵，但收到不同的值。在叢集中的所有節點之間完全複寫更新時，這些節點將會一致。(此行為與 DynamoDB 的最終一致性質相似。)

如果您要建置使用 DAX 的應用程式，則應該設計該應用程式，使其容忍最終一致資料。

## DAX 項目快取行為

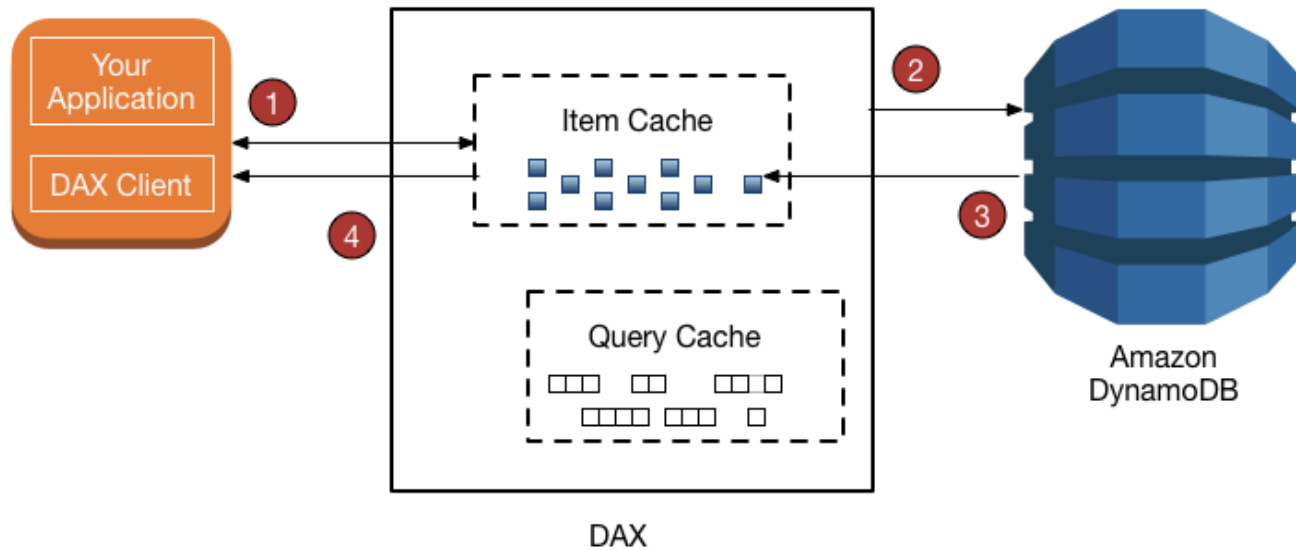
每個 DAX 叢集有兩個相異快取：項目快取與查詢快取。如需詳細資訊，請參閱 [DAX 的運作方式](#)。

本節會介紹讀取和寫入 DAX 項目快取的一致性隱憂。

### 讀取一致性

使用 DynamoDB，GetItem 操作預設會執行最終一致讀取。假設您搭配 DynamoDB 用戶端使用 UpdateItem。若您接著立即嘗試讀取相同項目，您可能會看到資料顯示的是更新前的內容。這是因為所有 DynamoDB 儲存體位置之間的傳播延遲。通常可在幾秒內達到一致性。因此若您重試讀取，您可能會看到更新後的項目。

當您搭配 DAX 用戶端使用 GetItem 時，操作 (在此案例中為最終一致讀取) 會以下列方式繼續。



1. DAX 用戶端發出 `GetItem` 請求。DAX 嘗試從項目快取讀取請求的項目。如果項目位於快取中（「快取命中」），則 DAX 會將其傳回應用程式。
2. 如果項目無法使用（「快取未中」），則 DAX 會對 DynamoDB 執行最終一致的 `GetItem` 操作。
3. DynamoDB 會傳回所請求的項目，而 DAX 會將其存放在項目快取中。
4. DAX 將項目傳回應用程式。
5. (未顯示) 如果 DAX 叢集包含超過一個節點，則會將項目複寫至叢集中的所有其他節點。

根據快取的存留時間 (TTL) 設定及最近最少使用 (LRU) 演算法，項目會保留在 DAX 項目快取中。如需詳細資訊，請參閱 [DAX 的運作方式](#)。

但是，在此期間，DAX 將不會從 DynamoDB 重新讀取項目。如果有人使用 DynamoDB 用戶端來更新項目，完全略過 DAX，則使用 DynamoDB 用戶端的 `GetItem` 請求所產生的結果會與使用 `GetItem` 用戶端的相同請求不同。在此情況下，除非 DAX 項目的 TTL 過期，否則 DAX 和 DynamoDB 將會針對相同索引鍵保留不一致的值。

如果應用程式修改基礎 DynamoDB 資料表中的資料，略過 DAX，則應用程式需要預期和容忍可能發生的資料不一致。

**Note**

除了 `GetItem`，DAX 用戶端還支援 `BatchGetItem` 請求。基本上 `BatchGetItem` 是一或多個 `GetItem` 請求的包裝函式，因此 DAX 會將每個請求視為個別的 `GetItem` 操作。

## 寫入一致性

DAX 是全部寫入快取，可簡化讓 DAX 項目快取與基礎 DynamoDB 資料表保持一致的程序。

DAX 用戶端支援與 DynamoDB 相同的寫入 API 操作

(`PutItem`、`UpdateItem`、`DeleteItem`、`BatchWriteItem` 和 `TransactWriteItems`)。當您搭配 DAX 用戶端使用這些操作時，項目會同時在 DAX 和 DynamoDB 中修改。DAX 會更新其項目快取中的項目，無論這些項目的 TTL 值為何。

例如，假設您從 DAX 用戶端發出 `GetItem` 請求，從 `ProductCatalog` 資料表讀取項目。(分割區索引鍵是 `Id`；沒有排序索引鍵。) 您擷取了 `Id` 為 101 的項目。該項目的 `QuantityOnHand` 值是 42。DAX 會將項目存放在其項目快取中，並具有特定的 TTL。針對此範例，假設 TTL 為 10 分鐘。然後，在 3 分鐘後，另一個應用程式使用 DAX 用戶端來更新相同的項目；因此，該項目的 `QuantityOnHand` 值現在是 41。假設未再次更新項目，則在下個 10 分鐘期間，任何後續讀取相同的項目都會傳回已快取的 `QuantityOnHand` 值 (41)。

## DAX 如何處理寫入

DAX 適用於需要高效能讀取的應用程式。身為全部寫入快取，DAX 會同步將您的全部寫入傳遞至 DynamoDB，然後自動地、非同步地將產生的更新複製到叢集中所有節點的項目快取。您不需要管理快取失效邏輯，因為 DAX 會為您自動進行處理。

DAX 支援下列寫入操作：`PutItem`、`UpdateItem`、`DeleteItem`、`BatchWriteItem` 和 `TransactWriteItems`。

當您將 `PutItem`、`UpdateItem`、`DeleteItem` 或 `BatchWriteItem` 請求傳送至 DAX 時，會執行下列作業：

- DAX 會將請求傳送至 DynamoDB。
- DynamoDB 回覆 DAX，確認寫入成功。
- DAX 將項目寫入至其項目快取。
- DAX 將成功傳回給申請者。

當您將 `TransactWriteItems` 請求傳送至 DAX 時，會執行下列作業：

- DAX 會將請求傳送至 DynamoDB。
- DynamoDB 回覆 DAX，確認交易完成。
- DAX 將成功傳回給申請者。
- 在背景中，DAX 會對 `TransactGetItems` 請求中的每個項目進行 `TransactWriteItems` 請求，以將項目存放在項目快取。`TransactGetItems` 會用來確保可[序列化隔離](#)。

若因為任何原因導致對 DynamoDB 的寫入失敗 (包括節流)，則項目便不會在 DAX 中進行快取。故障異常會傳回給申請者。除非是第一次將資料成功寫入 DAX，否則這會確保不會將資料寫入 DynamoDB 快取。

#### Note

每個對 DAX 進行的寫入都會改變項目快取的狀態。但是，寫入項目快取不會影響查詢快取。(DAX 項目快取和查詢快取的用途不同，並且會各自獨立操作。)

## DAX 查詢快取行為

DAX 會快取從 Query 獲得的結果，並在其查詢快取中進行 Scan 請求。但是，這些結果完全不會影響項目快取。在應用程式使用 DAX 發出 Query 或 Scan 請求後，結果集會存放於查詢快取而非項目快取中。您無法執行 Scan 操作來「預熱」項目快取，因為項目快取和查詢快取是不同的實體。

### 查詢-更新-查詢的一致性

項目快取的更新或基礎 DynamoDB 資料表的更新不會讓查詢快取中所存放的結果失效，或對其進行修改。

為了說明，請考慮以下情況。應用程式正在使用 `DocumentRevisions` 資料表，該資料表具有分割區索引鍵 `DocId` 和排序索引鍵 `RevisionNumber`。

1. 用戶端針對 `RevisionNumber` 大於或等於 5 的所有項目，發出 `DocId 101` 的 Query。DAX 會將結果集存放在查詢快取中，然後將結果集傳回給使用者。
2. 用戶端針對 `RevisionNumber` 為 20 的 `DocId 101` 發出 `PutItem` 請求。
3. 用戶端發出與步驟 1 所述相同的 Query (`DocId 101` 以及 `RevisionNumber >= 5`)。

在此案例中，步驟 3 發出的 Query 快取結果集會和步驟 1 快取的結果集完全相同。原因是，DAX 不會根據對個別項目的更新，使 Query 或 Scan 結果集無效。步驟 2 的 PutItem 操作只有在 Query 的 TTL 過期時，才會反映在 DAX 查詢快取中。

您的應用程式應該考慮查詢快取的 TTL 值，以及應用程式可以容忍查詢快取與項目快取之間不一致結果多長的時間。

## 強烈一致和交易讀取

若要執行強烈一致 GetItem、BatchGetItem、Query 或 Scan 請求，您必須將 ConsistentRead 參數設為 true。DAX 會將強烈一致讀取請求傳遞至 DynamoDB。收到來自 DynamoDB 的回應時，DAX 會將結果傳回用戶端，但不會快取結果。DAX 無法提供強烈一致讀取，因為它與 DynamoDB 無法緊密結合。基於此原因，任何從 DAX 進行的後續讀取都必須是最終一致讀取。任何後續的強烈一致讀取都必須傳遞至 DynamoDB。

DAX 處理 TransactGetItems 請求的方式與處理強烈一致讀取相同。DAX 會將所有 TransactGetItems 請求傳遞至 DynamoDB。收到來自 DynamoDB 的回應時，DAX 會將結果傳回用戶端，但不會快取結果。

## 負快取

在項目快取和查詢快取中，DAX 都支援負快取項目。「負快取項目」會在 DAX 於基礎 DynamoDB 資料表中找不到所請求項目時發生。DAX 會快取空的結果，並將該結果傳回使用者，而不是產生錯誤。

例如，假設應用程式將 GetItem 請求傳送至 DAX 叢集，而 DAX 項目快取中沒有相符項目。這會讓 DAX 從基礎 DynamoDB 資料表中讀取對應的項目。如果項目不存在於 DynamoDB，則 DAX 會將空項目存放至其項目快取，然後將空項目傳回應用程式。現在假設應用程式傳送同一項目的另一個 GetItem 請求。DAX 會找到項目快取中的空項目，並將它立即傳回給應用程式。它完全不會參考 DynamoDB。

負快取項目將會保留在 DAX 項目快取中，直到其項目 TTL 過期、呼叫 LRU，或是使用 PutItem、UpdateItem 或 DeleteItem 來修改項目。

DAX 查詢快取會以類似的方式處理負快取結果。如果應用程式執行 Query 或 Scan，而且 DAX 查詢快取未包含已快取的結果，則 DAX 會將請求傳送給 DynamoDB。如果結果集中沒有相符項目，則 DAX 會將空結果集存放至查詢快取，並將空結果集傳回應用程式。除非該結果集的 TTL 過期，否則後續的 Query 或 Scan 請求將會產生相同的 (空) 結果集。

## 寫入策略

DAX 的全部寫入行為適用於許多應用程式模式。但是，有些應用程式模式不適用全部寫入模式。

針對對延遲相當敏感的應用程式，全部寫入 DAX 會造成額外的網路躍點。因此寫入 DAX 會比直接寫入 DynamoDB 來得稍慢。若您的應用程式對寫入延遲相當敏感，您可以透過改為直接寫入 DynamoDB 來降低延遲。如需詳細資訊，請參閱 [繞過式寫入](#)。

針對寫入密集應用程式 (例如，執行大量資料載入的應用程式)，您可能會不想要透過 DAX 寫入所有資料，因為應用程式只會讀取該資料的極小部分。當您透過 DAX 寫入大量資料時，必須呼叫其 LRU 演算法來清出快取中的空間，以讀取新項目。這會降低 DAX 作為讀取快取的有效性。

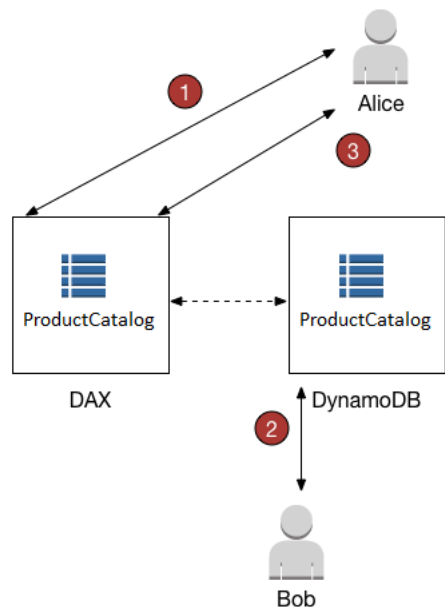
當您將項目寫入 DAX 時，項目快取狀態會變更，以容納新的項目。(例如，DAX 可能需要移出項目快取中較舊的資料，以清出空間來放置新項目。) 新項目將會根據快取的 LRU 演算法以及 TTL 設定保留在項目快取中。只要項目持續存在於項目快取中，DAX 就不會從 DynamoDB 重新讀取項目。

## 全部寫入

DAX 項目快取會實作全部寫入政策。如需詳細資訊，請參閱 [DAX 如何處理寫入](#)。

當您寫入項目時，DAX 會確保已快取的項目與存在於 DynamoDB 中的項目同步。針對在寫入項目之後需要立即重新讀取項目的應用程式，這十分有用。不過，如果其他應用程式直接寫入 DynamoDB 資料表，則 DAX 項目快取中的項目將不再與 DynamoDB 同步。

為了示範，請考慮正在使用 ProductCatalog 資料表的兩位使用者 (Alice 和 Bob)。Alice 使用 DAX 存取資料表，但 Bob 略過 DAX，並在 DynamoDB 中直接存取資料表。





1. Alice 更新中的 ProductCatalog 資料表中的項目。DAX 將請求轉送至 DynamoDB，且更新成功。DAX 接著將項目寫入至其項目快取，並將成功回應傳回給 Alice。從這個時間點開始，除非最後從快取中移出項目，否則任何從 DAX 讀取項目的使用者都會看到具有 Alice 更新的項目。
2. 稍後，Bob 更新了 Alice 寫入的相同 ProductCatalog 項目。不過，Bob 直接在 DynamoDB 中更新項目。DAX 不會回應透過 DynamoDB 進行的更新，來重新整理其項目快取。因此，DAX 使用者將不會看到 Bob 的更新。
3. Alice 再次從 DAX 中讀取項目。項目位於項目快取中，因此 DAX 會將它傳回 Alice，而不會存取 DynamoDB 資料表。

在此情況下，Alice 和 Bob 會看到相同 ProductCatalog 項目的不同呈現。除非 DAX 從項目快取中移出項目，或另一位使用者使用 DAX 再次更新相同的項目，否則就會是這種情況。

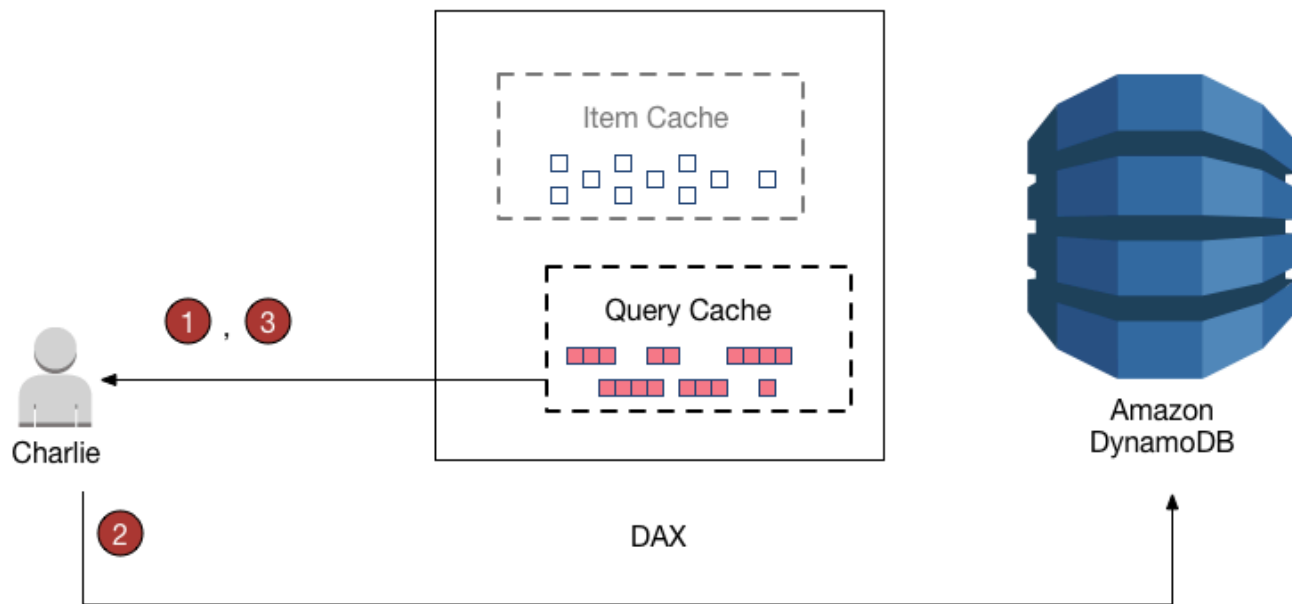
## 繞過式寫入

如果您的應用程式需要寫入大量資料 (例如大量資料載入)，則略過 DAX，將資料直接寫入至 DynamoDB 可能會相當合理。這種「繞過式寫入」策略可減少延遲。但是，項目快取將不會和 DynamoDB 中的資料保持同步。

如果您決定使用繞過式寫入策略，請記住，只要應用程式使用 DAX 用戶端讀取資料，DAX 就會填入其項目快取。這在某些情況下十分有益，因為它確定只會快取最常讀取的資料 (與最常寫入的資料相反)。

例如，請考慮想要使用 DAX 處理不同資料表 (GameScores 資料表) 的使用者 (Charlie)。GameScores 的分割區索引鍵是 UserId；因此，所有 Charlie 的分數都會有相同的 UserId。





1. Charlie 想要擷取其所有的分數，因此他傳送 Query 至 DAX。假設先前從未發出過此查詢，DAX 會將查詢轉送至 DynamoDB 進行處理。它會將結果存放在 DAX 查詢快取中，然後將結果傳回給 Charlie。除非移出結果集，否則結果集仍然會位於查詢快取中。
2. 現在假設 Charlie 玩 Meteor Blasters 遊戲，並得到高分。Charlie 將 UpdateItem 請求傳送至 DynamoDB，修改 GameScores 資料表中的項目。
3. 最後，Charlie 決定重新執行他稍早發出的 Query 來從 GameScores 擷取他的所有資料。在結果中，Charlie 看不到他在 Meteor Blasters 中的高分。原因是查詢結果來自查詢快取，而不是項目快取。兩個快取彼此獨立，因此其中一個快取中的變更不會影響另一個快取。

DAX 不會使用 DynamoDB 中的最新資料來重新整理查詢快取中的結果集。查詢快取中的每個結果集，其狀態都會是執行 Query 或 Scan 操作時的狀態。因此，Charlie 的 Query 結果不會反映其 PutItem 操作。除非 DAX 從查詢快取中移出結果集，否則就會是這種情況。

## 使用 DynamoDB Accelerator (DAX) 用戶端開發

若要從應用程式使用 DAX，您可以針對程式設計語言使用 DAX 用戶端。DAX 用戶端能對現有 Amazon DynamoDB 應用程式造成最少的干擾，您只需要對程式碼進行一些簡單修改。

**Note**

適用於各種程式設計語言的 DAX 用戶端可在以下網站取得：

- <http://dax-sdk.s3-website-us-west-2.amazonaws.com>

本節示範如何在預設 Amazon VPC 中啟動 Amazon EC2 執行個體、連線至執行個體，以及執行範例應用程式。本節也提供一些資訊，指導如何修改現有應用程式，好讓它可以使用 DAX 叢集。

**主題**

- [教學課程：使用 DynamoDB Accelerator \(DAX\) 執行範例應用程式](#)
- [修改現有應用程式以使用 DAX](#)

## 教學課程：使用 DynamoDB Accelerator (DAX) 執行範例應用程式

本教學課程示範如何在預設的 Virtual Private Cloud (VPC) 中啟動 Amazon EC2 執行個體、連線至執行個體，以及使用 Amazon DynamoDB Accelerator (DAX) 執行應用程式範例。

**Note**

為完成此教學課程，您預設的 VPC 中必須具備執行中的 DAX 叢集。若您尚未建立 DAX 叢集，請參閱 [建立 DAX 叢集](#) 來取得說明。

**主題**

- [步驟 1：啟動 Amazon EC2 執行個體](#)
- [步驟 2：建立使用者和政策](#)
- [步驟 3：設定 Amazon EC2 執行個體](#)
- [步驟 4：執行範例應用程式](#)

### 步驟 1：啟動 Amazon EC2 執行個體

當您的 Amazon DynamoDB Accelerator (DAX) 叢集可以使用時，您可以在預設 Amazon VPC 中啟動 Amazon EC2 執行個體。您接著便可以在該執行個體上安裝及執行 DAX 用戶端軟體。

## 啟動 EC2 執行個體

1. 登入 AWS Management Console 並開啟 Amazon EC2 主控台，網址為 <https://console.aws.amazon.com/ec2/>。

2. 選擇 Launch Instance (啟動執行個體) 並執行下列作業：

### 步驟 1：選擇 Amazon Machine Image (AMI)

1. 在 AMI 清單中，找到 Amazon Linux AMI 並選擇 Select (選取)。

### 步驟 2：選擇執行個體類型

1. 在執行個體類型的清單中，選擇 t2.micro。
2. 選擇 Next: Configure Instance Details (下一步：設定執行個體詳細資訊)。

### 步驟 3：設定執行個體詳細資訊

1. 前往 Network (網路)，然後選擇您的預設 VPC。
2. 選擇 Next: Add Storage (下一步：新增儲存體)。

### 步驟 4：新增儲存體

1. 選擇 Next: Add Tags (下一步：新增標籤) 以跳過此步驟。

### 步驟 5：新增標籤

1. 選擇 Next: Configure Security Group (下一步：設定安全群組) 以跳過此步驟。

### 步驟 6：設定安全群組

1. 選擇 Select an existing security group (選取現有的安全群組)。
2. 在安全群組清單中，選擇 default (預設)。這是您 VPC 的預設安全群組。
3. 選擇 Next: Review and Launch (下一步：檢閱和啟動)。

### 步驟 7：檢閱執行個體啟動

1. 選擇啟動。
3. 在 Select an existing key pair or create a new key pair (選取現有的金鑰對或建立新的金鑰對) 視窗中，執行下列其中一項作業：
  - 如果您沒有 Amazon EC2 金鑰對，請選擇 Create a new key pair (建立新的金鑰對) 並依照指示進行。系統會要求您下載私有金鑰檔案 (.pem 檔案)。您在稍後登入 Amazon EC2 執行個體時會需要此檔案。
  - 若您已擁有 Amazon EC2 金鑰對，請前往 Select a key pair (選取金鑰對)，然後從清單中選擇您的金鑰對。您必須已有可用的私有金鑰檔案 (.pem 檔案) 才能登入您的 Amazon EC2 執行個體。
4. 當設定好您的金鑰對後，請選擇 Launch Instances (啟動執行個體)。
5. 在主控台導覽窗格中，選擇 EC2 Dashboard (EC2 儀表板)，然後選擇您啟動的執行個體。在下方窗格的 Description (描述) 標籤上，找到您執行個體的 Public DNS (公有 DNS)，例如：ec2-11-22-33-44.us-west-2.compute.amazonaws.com。請記下此公有 DNS 名稱，因為您在 [步驟 3：設定 Amazon EC2 執行個體](#) 中將需要它。

#### Note

Amazon EC2 執行個體需要幾分鐘的時間才會變成可用。同時，請繼續進行 [步驟 2：建立使用者和政策](#)，並遵循其中的說明。

## 步驟 2：建立使用者和政策

在此步驟中，您可以使用政策建立使用者，該政策授予對 Amazon DynamoDB 加速器 (DAX) 叢集和 DynamoDB 的存取權。AWS Identity and Access Management 您接著便可以執行與您的 DAX 叢集互動的應用程式。

### 註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

### 若要註冊成為 AWS 帳戶

1. 開啟 <https://portal.aws.amazon.com/billing/signup>。
2. 請遵循線上指示進行。

部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

當您註冊一個時 AWS 帳戶，將創建 AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。您可以隨時登錄 <https://aws.amazon.com/> 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

## 建立具有管理權限的使用者

註冊後，請保護您的 AWS 帳戶 AWS 帳戶根使用者 AWS IAM Identity Center、啟用和建立系統管理使用者，這樣您就不會將 root 使用者用於日常工作。

## 保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶電子郵件地址，以帳戶擁有者身分登入。[AWS Management Console](#) 在下一頁中，輸入您的密碼。

如需使用根使用者登入的說明，請參閱 AWS 登入使用者指南中的[以根使用者身分登入](#)。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

如需指示，請參閱《IAM 使用者指南》中的[為 AWS 帳戶根使用者啟用虛擬 MFA 裝置 \(主控台\)](#)。

## 建立具有管理權限的使用者

1. 啟用 IAM Identity Center。

如需指示，請參閱 AWS IAM Identity Center 使用者指南中的[啟用 AWS IAM Identity Center](#)。

2. 在 IAM 身分中心中，將管理存取權授予使用者。

[若要取得有關使用 IAM Identity Center 目錄做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用 AWS IAM Identity Center 者存取」。](#)

## 以具有管理權限的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

如需使用 IAM 身分中心使用者 [登入的說明](#)，請參閱使用 AWS 登入者指南中的 [登入 AWS 存取入口網站](#)。

## 指派存取權給其他使用者

1. 在 IAM 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「[建立權限集](#)」。

2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「[新增群組](#)」。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 使用者和群組位於 AWS IAM Identity Center：

建立權限合集。請按照 AWS IAM Identity Center 使用者指南 中的 [建立權限合集](#) 說明進行操作。

- 透過身分提供者在 IAM 中管理的使用者：

建立聯合身分的角色。請按照 IAM 使用者指南 的 [為第三方身分提供者 \(聯合\) 建立角色](#) 中的指示進行操作。

- IAM 使用者：

- 建立您的使用者可擔任的角色。請按照 IAM 使用者指南 的 [為 IAM 使用者建立角色](#) 中的指示進行操作。

- (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南的 [新增許可到使用者 \(主控台\)](#) 中的指示。

若要使用 JSON 政策編輯器來建立政策

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 <https://console.aws.amazon.com/iam/>。
2. 在左側的導覽窗格中，選擇 Policies (政策)。

如果這是您第一次選擇 Policies (政策)，將會顯示 Welcome to Managed Policies (歡迎使用受管政策) 頁面。選擇 Get Started (開始使用)。

3. 在頁面頂端，選擇 Create policy (建立政策)。

- 在政策編輯器中，選擇 JSON 選項。
- 輸入或貼上 JSON 政策文件。如需有關 IAM 政策語言的詳細資訊，請參閱 [IAM JSON 政策參考](#)。
- 解決[政策驗證](#)期間產生的任何安全性警告、錯誤或一般性警告，然後選擇 Next (下一步)。

**Note**

您可以隨時切換視覺化與 JSON 編輯器選項。不過，如果您進行變更或在視覺化編輯器中選擇下一步，IAM 就可能調整您的政策結構，以便針對視覺化編輯器進行最佳化。如需詳細資訊，請參閱 IAM 使用者指南中的[調整政策結構](#)。

- (選擇性) 在中建立或編輯原則時 AWS Management Console，您可以產生可在 AWS CloudFormation 範本中使用的 JSON 或 YAML 原則範本。

若要這麼做，請在 [原則編輯器] 中選擇 [動作]，然後選擇 [產生 CloudFormation 範本]。若要進一步了解 AWS CloudFormation，請參閱《AWS CloudFormation 使用指南》中的[AWS Identity and Access Management 資源類型參考](#)。

- 將許可新增至政策後，請選擇下一步。
- 在檢視與建立頁面上，為您在建立的政策輸入政策名稱與描述 (選用)。檢視此政策中定義的許可，來查看您的政策所授予的許可。
- (選用) 藉由連接標籤作為鍵值組，將中繼資料新增至政策。如需有關在 IAM 中使用標籤的詳細資訊，請參閱《IAM 使用者指南》中的[標記 IAM 資源](#)。
- 選擇 Create policy (建立政策) 儲存您的新政策。

政策文件：複製並貼上以下文件以建立 JSON 政策。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Action": [
 "dax:*"
],
 "Effect": "Allow",
 "Resource": [
 "*"
]
 }
],
}
```

```
 "Action": [
 "dynamodb:*"
],
 "Effect": "Allow",
 "Resource": [
 "*"
]
 }
]
```

### 步驟 3：設定 Amazon EC2 執行個體

當您的 Amazon EC2 執行個體可用時，您可以登入執行個體並準備使用它。

#### Note

下列步驟假設您從執行 Linux 的電腦連線至 Amazon EC2 執行個體。如需其他連線方式，請參閱《Amazon EC2 Linux 執行個體使用者指南》中的[連線 Linux 執行個體](#)。

#### 設定 EC2 執行個體

1. 前往 <https://console.aws.amazon.com/ec2/> 開啟 Amazon EC2 主控台。
2. 使用 ssh 命令登入您的 Amazon EC2 執行個體，如以下範例所示。

```
ssh -i my-keypair.pem ec2-user@public-dns-name
```

您需要指定您的私有金鑰檔案 (.pem 檔案) 和您執行個體的公有 DNS 名稱。(請參閱 [步驟 1：啟動 Amazon EC2 執行個體](#))。

登入 ID 為 ec2-user。不需要任何密碼。

3. 登入 EC2 執行個體後，設定您的登入 AWS 資料，如下所示。輸入您的 AWS 存取金鑰 ID 和私密金鑰 (寄件人 [步驟 2：建立使用者和政策](#))，並將預設地區名稱設定為您目前的區域。(在以下範例中，預設區域名稱為 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]:
```

啟動並設定好 Amazon EC2 執行個體後，您可以使用其中一個提供的範例應用程式測試 DAX 的功能。如需詳細資訊，請參閱 [步驟 4：執行範例應用程式](#)。

## 步驟 4：執行範例應用程式

為協助您測試 Amazon DynamoDB Accelerator (DAX) 的功能，您可以在 Amazon EC2 執行個體上執行其中一個提供的應用程式範例。

### 主題

- [適用於 Go 的 DAX 開發套件](#)
- [Java 與 DAX](#)
- [.NET 和 DAX](#)
- [Node.js 和 DAX](#)
- [Python 和 DAX](#)

### 適用於 Go 的 DAX 開發套件

請按照此程序操作，在 Amazon EC2 執行個體上執行 Amazon DynamoDB Accelerator (DAX) 適用於 Go 的開發套件範例應用程式。

為 DAX 執行適用於 Go 的開發套件範例

1. 在您的 Amazon EC2 執行個體上設定適用於 Go 的開發套件：
  - a. 安裝 Go 程式設計語言 (Golang)。

```
sudo yum install -y golang
```

- b. 測試 Golang 是否已安裝並正常運作。

```
go version
```

這類的訊息應會出現。

```
go version go1.15.5 linux/amd64
```

其餘指示仰賴模組支援，成為 Go 版本 1.13 的預設。

## 2. 安裝範例 Golang 應用程式。

```
go get github.com/aws-samples/aws-dax-go-sample
```

## 3. 執行下列 Golang 程式。第一個程式會建立名為 TryDaxGoTable 的 DynamoDB 資料表。第二個程式會將資料寫入資料表。

```
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. 執行下列 Golang 程式。

```
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
```

記下計時資訊：GetItem、Query 和 Scan 測試所需要的毫秒數。

## 5. 在先前的步驟中，您已針對 DynamoDB 端點執行程式。現在，請再次執行程式，但這一次 GetItem、Query 和 Scan 操作會由您的 DAX 叢集處理。

若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

- 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- 使用 AWS CLI — 輸入下列指令。

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

如下列範例所示，叢集端點會在輸出上顯示。

```
{
 "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"
}
```

現在重新執行程式，但這一次，請將叢集端點做為命令列參數指定。

```
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
```

查看輸出的剩餘部分，並記下計時資訊。使用 DAX 的 GetItem、Query 和 Scan 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

6. 執行以下 Golang 程式，刪除 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 與 DAX

適用於 Java 2.x 的 DAX 開發套件與[適用於 Java 2.x 的 AWS 開發套件](#)相容。它以 Java 8+ 為基礎建置，並支援非封鎖 I/O，如需將 DAX 與 Java 1.x AWS 版 SDK 搭配使用的相關資訊，請參閱[搭配適用於 Java 1.x 的 AWS 開發套件使用 DAX](#)

## 使用用戶端作為 Maven 依存項目

遵循這些步驟，在您的應用程式中將 DAX SDK for Java 用戶端做為依存項目使用。

1. 下載並安裝 Apache Maven。如需詳細資訊，請參閱[下載 Apache Maven](#)和[安裝 Apache Maven](#)。
2. 將用戶端 Maven 依存項目新增至您應用程式的專案物件模型 (POM) 檔案。在此範例中，將 `x.x.x` 取代為用戶端的實際版本號碼。

```
<!--Dependency:-->
<dependencies>
 <dependency>
 <groupId>software.amazon.dax</groupId>
 <artifactId>amazon-dax-client</artifactId>
 <version>x.x.x</version>
 </dependency>
</dependencies>
```

## TryDax 範例程式碼

在設定工作空間並將 DAX 開發套件新增為依存項目之後，請將 [TryDax. 爪哇](#) 複製到您的專案中。

使用此命令運程式碼。

```
java -cp classpath TryDax
```

您應該會看到類似下列的輸出。

```
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.
```

記下計時資訊：GetItem 和 Query 測試所需要的毫秒數。在此例中，您已針對 DynamoDB 端點執行程式。您將再次執行該程式，但此次針對的是您的 DAX 叢集。

若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

- 在 DynamoDB 主控台中，選取您的 DAX 叢集。叢集端點會在主控台中顯示，如以下範例。

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- 使用 AWS CLI，輸入下列命令：

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

叢集端點地址、連接埠和 URL 會在輸出中顯示，如以下範例所示。

```
{
 "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"
}
```

現在重新執行程式，但這一次，請將叢集端點 URL 作為命令列參數指定。

```
java -cp classpath TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

查看輸出，並記下計時資訊。使用 DAX 的 `GetItem` 和 `Query` 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

## 軟體開發套件指標

使用適用於 Java 2.x 的 DAX SDK，您可以收集有關應用程式中服務用戶端的指標，並在 Amazon CloudWatch 中分析輸出。如需詳細資訊，請參閱[啟用 SDK 指標](#)。

### Note

適用於 Java 的 DAX 開發套件僅會收集 `ApiCallSuccessful` 和 `ApiCallDuration` 指標。

## TryDax. 爪哇

```
import java.util.Map;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.dax.ClusterDaxAsyncClient;
import software.amazon.dax.Configuration;

public class TryDax {
 public static void main(String[] args) throws Exception {
 DynamoDbAsyncClient ddbClient = DynamoDbAsyncClient.builder()
 .build();

 DynamoDbAsyncClient daxClient = null;
 if (args.length >= 1) {
 daxClient = ClusterDaxAsyncClient.builder()
 .overrideConfiguration(Configuration.builder()
```

```
 .url(args[0]) // e.g. dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com
 .build()
 }.build();
}

String tableName = "TryDaxTable";

System.out.println("Creating table...");
createTable(tableName, ddbClient);

System.out.println("Populating table...");
writeData(tableName, ddbClient, 100, 10);

DynamoDbAsyncClient testClient = null;
if (daxClient != null) {
 testClient = daxClient;
} else {
 testClient = ddbClient;
}

System.out.println("Running GetItem and Query tests...");
System.out.println("First iteration of each test will result in cache misses");
System.out.println("Next iterations are cache hits\n");

// GetItem
getItemTest(tableName, testClient, 100, 10, 5);

// Query
queryTest(tableName, testClient, 100, 2, 9, 5);

System.out.println("Deleting table...");
deleteTable(tableName, ddbClient);
}

private static void createTable(String tableName, DynamoDbAsyncClient client) {
 try {
 System.out.println("Attempting to create table; please wait...");

 client.createTable(CreateTableRequest.builder()
 .tableName(tableName)
 .keySchema(KeySchemaElement.builder()
 .keyType(KeyType.HASH)
 .attributeName("pk")
```

```
 .build(), KeySchemaElement.builder()
 .keyType(KeyType.RANGE)
 .attributeName("sk")
 .build())
 .attributeDefinitions(AttributeDefinition.builder()
 .attributeName("pk")
 .attributeType(ScalarAttributeType.N)
 .build(), AttributeDefinition.builder()
 .attributeName("sk")
 .attributeType(ScalarAttributeType.N)
 .build())
 .billingMode(BillingMode.PAY_PER_REQUEST)
 .build()).get();
client.waiter().waitUntilTableExists(DescribeTableRequest.builder()
 .tableName(tableName)
 .build()).get();
System.out.println("Successfully created table.");

} catch (Exception e) {
 System.err.println("Unable to create table: ");
 e.printStackTrace();
}
}

private static void deleteTable(String tableName, DynamoDbAsyncClient client) {
 try {
 System.out.println("\nAttempting to delete table; please wait...");
 client.deleteTable(DeleteTableRequest.builder()
 .tableName(tableName)
 .build()).get();
 client.waiter().waitUntilTableNotExists(DescribeTableRequest.builder()
 .tableName(tableName)
 .build()).get();
 System.out.println("Successfully deleted table.");

 } catch (Exception e) {
 System.err.println("Unable to delete table: ");
 e.printStackTrace();
 }
}

private static void writeData(String tableName, DynamoDbAsyncClient client, int
pkmax, int skmax) {
 System.out.println("Writing data to the table...");
```



```
int stringSize = 1000;
StringBuilder sb = new StringBuilder(stringSize);
for (int i = 0; i < stringSize; i++) {
 sb.append('X');
}
String someData = sb.toString();

try {
 for (int ipk = 1; ipk <= pkmax; ipk++) {
 System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
 for (int isk = 1; isk <= skmax; isk++) {
 client.putItem(PutItemRequest.builder()
 .tableName(tableName)
 .item(Map.of("pk", attr(ipk), "sk", attr(isk), "someData",
attr(someData)))
 .build()).get();
 }
 }
} catch (Exception e) {
 System.err.println("Unable to write item:");
 e.printStackTrace();
}

private static AttributeValue attr(int n) {
 return AttributeValue.builder().n(String.valueOf(n)).build();
}

private static AttributeValue attr(String s) {
 return AttributeValue.builder().s(s).build();
}

private static void getItemTest(String tableName, DynamoDbAsyncClient client, int
pk, int sk, int iterations) {
 long startTime, endTime;
 System.out.println("GetItem test - partition key 1-" + pk + " and sort keys 1-"
+ sk);

 for (int i = 0; i < iterations; i++) {
 startTime = System.nanoTime();
 try {
 for (int ipk = 1; ipk <= pk; ipk++) {
```

```
 for (int isk = 1; isk <= sk; isk++) {
 client.getItem(GetItemRequest.builder()
 .tableName(tableName)
 .key(Map.of("pk", attr(ipk), "sk", attr(isk)))
 .build()).get();
 }
 }
} catch (Exception e) {
 System.err.println("Unable to get item:");
 e.printStackTrace();
}
endTime = System.nanoTime();
printTime(startTime, endTime, pk * sk);
}
}

private static void queryTest(String tableName, DynamoDbAsyncClient client, int pk,
int sk1, int sk2, int iterations) {
 long startTime, endTime;
 System.out.println("Query test - partition key 1-" + pk + " and sort keys
between " + sk1 + " and " + sk2);

 for (int i = 0; i < iterations; i++) {
 startTime = System.nanoTime();
 for (int ipk = 1; ipk <= pk; ipk++) {
 try {
 // Pagination API for Query.
 client.queryPaginator(QueryRequest.builder()
 .tableName(tableName)
 .keyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
 .expressionAttributeValues(Map.of(":pkval", attr(ipk),
":skval1", attr(sk1), ":skval2", attr(sk2))))
 .build()).items().subscribe((item) -> {
 }).get();
 } catch (Exception e) {
 System.err.println("Unable to query table:");
 e.printStackTrace();
 }
 }
 endTime = System.nanoTime();
 printTime(startTime, endTime, pk);
 }
}
```

```
private static void printTime(long startTime, long endTime, int iterations) {
 System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
(1000000.0));
 System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
1000000.0));
}
}
```

## .NET 和 DAX

遵循下列步驟，在 Amazon EC2 執行個體上執行 .NET 範例。

### Note

本教學使用 .NET 6 SDK，但亦可與 .NET Core SDK 搭配使用。本教學課程示範如何在預設的 Amazon VPC 中執行程式來存取 Amazon DynamoDB Accelerator (DAX) 叢集。如果您願意，您可以使用 AWS Toolkit for Visual Studio 來撰寫 .NET 應用程式，並將其部署到您的 VPC 中。

如需詳細資訊，請參閱《AWS Elastic Beanstalk 開發人員指南》中的[使用 AWS Toolkit for Visual Studio 在 .NET 中建立和部署 Elastic Beanstalk 應用程式](#)。

## 執行 DAX 的 .NET 範例

1. 前往 [Microsoft Downloads page](#) (Microsoft 下載頁面) 並下載適用於 Linux 的最新 .NET 6 (或 .NET Core) SDK。下載的檔案為 `dotnet-sdk-N.N.N-linux-x64.tar.gz`。
2. 將 SDK 檔案解壓縮。

```
mkdir dotnet
tar zxvf dotnet-sdk-N.N.N-linux-x64.tar.gz -C dotnet
```

將 *N.N.N* 替換為 .NET SDK 的實際版本號碼 (例如：6.0.100)。

3. 驗證安裝。

```
alias dotnet=$HOME/dotnet/dotnet
dotnet --version
```

這應該會列印出 .NET SDK 的版本號碼。

**Note**

您可能會收到以下錯誤 (而不是版本號碼) :

錯誤 : libunwind.so.8 : 無法開啟共享物件檔案 : 找不到檔案或目錄

若要解決錯誤 , 請安裝 libunwind 套件。

```
sudo yum install -y libunwind
```

在執行此作業後 , 您應該能夠執行 `dotnet --version` 命令 , 而不會發生任何錯誤。

**4. 建立新的 .NET 專案。**

```
dotnet new console -o myApp
```

這需要幾分鐘的時間來執行 one-time-only 設定。完成時 , 請執行範例專案。

```
dotnet run --project myApp
```

您應該會收到以下訊息 : Hello World!

**5. myApp/myApp.csproj 檔案包含您專案的中繼資料。若要在您的應用程式中使用 DAX 用戶端 , 請修改檔案 , 讓它看起來如下。**

```
<Project Sdk="Microsoft.NET.Sdk">
 <PropertyGroup>
 <OutputType>Exe</OutputType>
 <TargetFramework>net6.0</TargetFramework>
 </PropertyGroup>
 <ItemGroup>
 <PackageReference Include="AWSSDK.DAX.Client" Version="*" />
 </ItemGroup>
</Project>
```

**6. 下載範例程式來源碼 (.zip 檔案)。**

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

下載完成後 , 解壓縮來源檔案。

```
unzip TryDax.zip
```

- 現在，請執行範例程式 (一次執行一個)。針對每一個程式，將其內容複製到 `myApp/Program.cs`，然後執行 `MyApp` 專案。

執行以下 .NET 程式。第一個程式會建立名為 `TryDaxTable` 的 DynamoDB 資料表。第二個程式會將資料寫入資料表。

```
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
```

- 接下來，請執行一些程式，以在您的 DAX 叢集上執行 `GetItem`、`Query` 和 `Scan` 操作。若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：
  - 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- 使用 AWS CLI — 輸入下列指令。

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

如下列範例所示，叢集端點會在輸出上顯示。

```
{
 "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"
}
```

現在執行下列程式，將您的叢集端點指定為命令列參數。(以您的實際 DAX 叢集端點取代範本端點。)

```
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.16fzcv.dax-clusters.us-east-1.amazonaws.com

cp TryDax/dotNet/05-Scan-Test.cs MyApp/Program.cs
dotnet run --project MyApp dax://my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com
```

記下計時資訊：GetItem、Query 和 Scan 測試所需要的毫秒數。

9. 執行以下 .NET 程式，以刪除 TryDaxTable。

```
cp TryDax/dotNet/06-DeleteTable.cs MyApp/Program.cs
dotnet run --project MyApp
```

如需這些程式的詳細資訊，請參閱下列各節：

- [政司司長 CreateTable](#)
- [02-Write-Data.cs](#)
- [GetItem-Test.cs](#)
- [04-Query-Test.cs](#)
- [05-Scan-Test.cs](#)
- [政司司長 DeleteTable](#)

政司司長 CreateTable

01-CreateTable.cs 程式會建立資料表 (TryDaxTable)。本節剩餘的 .NET 程式呈現於此資料表。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
 class Program
```

```
{
 public static async Task Main(string[] args)
 {
 AmazonDynamoDBClient client = new AmazonDynamoDBClient();

 var tableName = "TryDaxTable";

 var request = new CreateTableRequest()
 {
 TableName = tableName,
 KeySchema = new List<KeySchemaElement>()
 {
 new KeySchemaElement{ AttributeName = "pk",KeyType = "HASH"},
 new KeySchemaElement{ AttributeName = "sk",KeyType = "RANGE"}
 },
 AttributeDefinitions = new List<AttributeDefinition>() {
 new AttributeDefinition{ AttributeName = "pk",AttributeType = "N"},
 new AttributeDefinition{ AttributeName = "sk",AttributeType = "N"}
 },
 ProvisionedThroughput = new ProvisionedThroughput()
 {
 ReadCapacityUnits = 10,
 WriteCapacityUnits = 10
 }
 };

 var response = await client.CreateTableAsync(request);

 Console.WriteLine("Hit <enter> to continue...");
 Console.ReadLine();
 }
}
```

## 02-Write-Data.cs

02-Write-Data.cs 程式會將測試資料寫入 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();
 }
 }
}
```



## GetItem-Test.cs

03-GetItem-Test.cs 程式會在 GetItem 上執行 TryDaxTable 操作。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace ClientTest
{
 class Program
 {
 public static async Task Main(string[] args)
 {
 string endpointUri = args[0];
 Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

 var clientConfig = new DaxClientConfig(endpointUri)
 {
 AwsCredentials = FallbackCredentialsFactory.GetCredentials()
 };
 var client = new ClusterDaxClient(clientConfig);

 var tableName = "TryDaxTable";

 var pk = 1;
 var sk = 10;
 var iterations = 5;

 var startTime = System.DateTime.Now;

 for (var i = 0; i < iterations; i++)
 {
 for (var ipk = 1; ipk <= pk; ipk++)
 {
 for (var isk = 1; isk <= sk; isk++)
 {
 var request = new GetItemRequest()
 {
 TableName = tableName,
```

```

 Key = new Dictionary<string, AttributeValue>() {
 {"pk", new AttributeValue {N = ipk.ToString()} },
 {"sk", new AttributeValue {N = isk.ToString()} } }
 };
 var response = await client.GetItemAsync(request);
 Console.WriteLine($"GetItem succeeded for pk: {ipk},sk:
{isk}");
 }
}

var endTime = DateTime.Now;
TimeSpan timeSpan = endTime - startTime;
Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
milliseconds");

Console.WriteLine("Hit <enter> to continue...");
Console.ReadLine();
}
}
}
}

```

## 04-Query-Test.cs

04-Query-Test.cs 程式會在 Query 上執行 TryDaxTable 操作。

```

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
 class Program
 {
 public static async Task Main(string[] args)
 {
 string endpointUri = args[0];
 Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");
 }
 }
}

```

```
var clientConfig = new DaxClientConfig(endpointUri)
{
 AwsCredentials = FallbackCredentialsFactory.GetCredentials()
};
var client = new ClusterDaxClient(clientConfig);

var tableName = "TryDaxTable";

var pk = 5;
var sk1 = 2;
var sk2 = 9;
var iterations = 5;

var startTime = DateTime.Now;

for (var i = 0; i < iterations; i++)
{
 var request = new QueryRequest()
 {
 TableName = tableName,
 KeyConditionExpression = "pk = :pkval and sk between :skval1
and :skval2",
 ExpressionAttributeValues = new Dictionary<string,
AttributeValue>() {
 {":pkval", new AttributeValue {N = pk.ToString()} },
 {":skval1", new AttributeValue {N = sk1.ToString()} },
 {":skval2", new AttributeValue {N = sk2.ToString()} }
 }
 };
 var response = await client.QueryAsync(request);
 Console.WriteLine($"{i}: Query succeeded");
}

var endTime = DateTime.Now;
TimeSpan timeSpan = endTime - startTime;
Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
milliseconds");

Console.WriteLine("Hit <enter> to continue...");
Console.ReadLine();
}
```

```
 }
}
```

## 05-Scan-Test.cs

05-Scan-Test.cs 程式會在 Scan 上執行 TryDaxTable 操作。

```
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();
}
}
```

## 政司司長 DeleteTable

06-DeleteTable.cs 程式會刪除 TryDaxTable。請在完成測試之後執行此程式。

```
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 和 DAX

請按照此程序操作，在 Amazon EC2 執行個體上執行 Node.js 範例應用程式。

### 執行 DAX 的 Node.js 範例

1. 在您的 Amazon EC2 執行個體上設定 Node.js，如下所示：

a. 安裝節點版本管理工具 (nvm)。

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
```

b. 使用 nvm 安裝 Node.js。

```
nvm install 12.16.3
```

c. 測試 Node.js 已安裝且正常運作。

```
node -e "console.log('Running Node.js ' + process.version)"
```

這應該會顯示以下訊息。

```
Running Node.js v12.16.3
```

2. 使用節點套件管理工具 (npm) 安裝 DAX Node.js 用戶端。

```
npm install amazon-dax-client
```

3. 下載範例程式來源碼 (.zip 檔案)。

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

下載完成後，解壓縮來源檔案。

```
unzip TryDax.zip
```

- 執行下列 Node.js 程式。第一個程式會建立名為 TryDaxTable 的 Amazon DynamoDB 資料表。第二個程式會將資料寫入資料表。

```
node 01-create-table.js
node 02-write-data.js
```

- 執行下列 Node.js 程式。

```
node 03-getitem-test.js
node 04-query-test.js
node 05-scan-test.js
```

記下計時資訊：GetItem、Query 和 Scan 測試所需要的毫秒數。

- 在先前的步驟中，您已針對 DynamoDB 端點執行程式。請再次執行程式，但這一次 GetItem、Query 和 Scan 操作會由您的 DAX 叢集處理。

若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

- 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- 使用 AWS CLI — 輸入下列指令。

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

如下列範例所示，叢集端點會在輸出上顯示。

```
{
 "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"
}
```

現在重新執行程式，但這一次，請將叢集端點做為命令列參數指定。

```
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
```

查看輸出的剩餘部分，並記下計時資訊。使用 DAX 的 `GetItem`、`Query` 和 `Scan` 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

7. 執行以下 Node.js 程式，刪除 `TryDaxTable`。

```
node 06-delete-table
```

如需這些程式的詳細資訊，請參閱下列各節：

- [01-create-table.js](#)
- [02-write-data.js](#)
- [03-getitem-test.js](#)
- [04-query-test.js](#)
- [05-scan-test.js](#)
- [06-delete-table.js](#)

### 01-create-table.js

`01-create-table.js` 程式會建立資料表 (`TryDaxTable`)。本節剩餘的 Node.js 程式會依存此資料表。

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
 region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
 TableName: tableName,
 KeySchema: [
```



```
 { AttributeName: "pk", KeyType: "HASH" }, //Partition key
 { AttributeName: "sk", KeyType: "RANGE" }, //Sort key
],
 AttributeDefinitions: [
 { AttributeName: "pk", AttributeType: "N" },
 { AttributeName: "sk", AttributeType: "N" },
],
 ProvisionedThroughput: {
 ReadCapacityUnits: 10,
 WriteCapacityUnits: 10,
 },
};

dynamodb.createTable(params, function (err, data) {
 if (err) {
 console.error(
 "Unable to create table. Error JSON:",
 JSON.stringify(err, null, 2)
);
 } else {
 console.log(
 "Created table. Table description JSON:",
 JSON.stringify(data, null, 2)
);
 }
});
```

## 02-write-data.js

02-write-data.js 程式會將測試資料寫入 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

03-getitem-test.js 程式會在 TryDaxTable 上執行 GetItem 操作。

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
 region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
```

```
var daxClient = null;

if (process.argv.length > 2) {
 var dax = new AmazonDaxClient({
 endpoints: [process.argv[2]],
 region: region,
 });
 daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient !== null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 1;
var sk = 10;
var iterations = 5;

for (var i = 0; i < iterations; i++) {
 var startTime = new Date().getTime();

 for (var ipk = 1; ipk <= pk; ipk++) {
 for (var isk = 1; isk <= sk; isk++) {
 var params = {
 TableName: tableName,
 Key: {
 pk: ipk,
 sk: isk,
 },
 };
 };

 client.get(params, function (err, data) {
 if (err) {
 console.error(
 "Unable to read item. Error JSON:",
 JSON.stringify(err, null, 2)
);
 } else {
 // GetItem succeeded
 }
 });
 }
}

var endTime = new Date().getTime();
```

```
console.log(
 "\tTotal time: ",
 endTime - startTime,
 "ms - Avg time: ",
 (endTime - startTime) / iterations,
 "ms"
);
}
```

## 04-query-test.js

04-query-test.js 程式會在 Query 上執行 TryDaxTable 操作。

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
 region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
 var dax = new AmazonDaxClient({
 endpoints: [process.argv[2]],
 region: region,
 });
 daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient !== null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 5;
var sk1 = 2;
var sk2 = 9;
var iterations = 5;

var params = {
 TableName: tableName,
```

```
KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
ExpressionAttributeValues: {
 ":pkval": pk,
 ":skval1": sk1,
 ":skval2": sk2,
},
};

for (var i = 0; i < iterations; i++) {
 var startTime = new Date().getTime();

 client.query(params, function (err, data) {
 if (err) {
 console.error(
 "Unable to read item. Error JSON:",
 JSON.stringify(err, null, 2)
);
 } else {
 // Query succeeded
 }
 });

 var endTime = new Date().getTime();
 console.log(
 "\tTotal time: ",
 endTime - startTime,
 "ms - Avg time: ",
 (endTime - startTime) / iterations,
 "ms"
);
}
```

## 05-scan-test.js

05-scan-test.js 程式會在 Scan 上執行 TryDaxTable 操作。

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
 region: region,
```

```
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
 var dax = new AmazonDaxClient({
 endpoints: [process.argv[2]],
 region: region,
 });
 daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var iterations = 5;

var params = {
 TableName: tableName,
};
var startTime = new Date().getTime();
for (var i = 0; i < iterations; i++) {
 client.scan(params, function (err, data) {
 if (err) {
 console.error(
 "Unable to read item. Error JSON:",
 JSON.stringify(err, null, 2)
);
 } else {
 // Scan succeeded
 }
 });
}

var endTime = new Date().getTime();
console.log(
 "\tTotal time: ",
 endTime - startTime,
 "ms - Avg time: ",
 (endTime - startTime) / iterations,
 "ms"
);
```

## 06-delete-table.js

06-delete-table.js 程式會刪除 TryDaxTable。請在完成測試之後執行此程式。

```
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 和 DAX

請按照此程序操作，在 Amazon EC2 執行個體上執行 Python 範例應用程式。

## 執行 DAX 的 Python 範例

1. 使用 pip 公用程式安裝 DAX Python 用戶端。

```
pip install amazon-dax-client
```

2. 下載範例程式來源碼 (.zip 檔案)。

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

下載完成後，解壓縮來源檔案。

```
unzip TryDax.zip
```

3. 執行下列 Python 程式。第一個程式會建立名為 TryDaxTable 的 Amazon DynamoDB 資料表。第二個程式會將資料寫入資料表。

```
python 01-create-table.py
python 02-write-data.py
```

4. 執行下列 Python 程式。

```
python 03-getitem-test.py
python 04-query-test.py
python 05-scan-test.py
```

記下計時資訊：GetItem、Query 和 Scan 測試所需要的毫秒數。

5. 在先前的步驟中，您已針對 DynamoDB 端點執行程式。現在，請再次執行程式，但這一次 GetItem、Query 和 Scan 操作會由您的 DAX 叢集處理。

若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

- 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

```
dax://my-cluster.16fzcv.dax-clusters.us-east-1.amazonaws.com
```

- 使用 AWS CLI — 輸入下列指令。

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```



叢集端點會在輸出中顯示，如此範例所示。

```
{
 "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"
}
```

請重新執行程式，但這一次，請將叢集端點做為命令列參數指定。

```
python 03-getitem-test.py dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
python 04-query-test.py dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
python 05-scan-test.py dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

查看輸出的剩餘部分，並記下計時資訊。使用 DAX 的 GetItem、Query 和 Scan 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

6. 執行以下 Python 程式，刪除 TryDaxTable。

```
python 06-delete-table.py
```

如需這些程式的詳細資訊，請參閱下列各節：

- [01-create-table.py](#)
- [02-write-data.py](#)
- [03-getitem-test.py](#)
- [04-query-test.py](#)
- [05-scan-test.py](#)
- [06-delete-table.py](#)

01-create-table.py

01-create-table.py 程式會建立資料表 (TryDaxTable)。本節剩餘的 Python 程式呈現於此資料表。

```
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

02-write-data.py 程式會將測試資料寫入 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

03-getitem-test.py 程式會在 GetItem 上執行 TryDaxTable 操作。此示例是針對 eu-west-1 區域。

```
import argparse
```

```
import sys
import time
import amazondax
import boto3

def get_item_test(key_count, iterations, dyn_resource=None):
 """
 Gets items from the table a specified number of times. The time before the
 first iteration and the time after the last iteration are both captured
 and reported.

 :param key_count: The number of items to get from the table in each iteration.
 :param iterations: The number of iterations to run.
 :param dyn_resource: Either a Boto3 or DAX resource.
 :return: The start and end times of the test.
 """
 if dyn_resource is None:
 dyn_resource = boto3.resource('dynamodb')

 table = dyn_resource.Table('TryDaxTable')
 start = time.perf_counter()
 for _ in range(iterations):
 for partition_key in range(1, key_count + 1):
 for sort_key in range(1, key_count + 1):
 table.get_item(Key={
 'partition_key': partition_key,
 'sort_key': sort_key
 })
 print('.', end='')
 sys.stdout.flush()
 print()
 end = time.perf_counter()
 return start, end

if __name__ == '__main__':
 parser = argparse.ArgumentParser()
 parser.add_argument(
 'endpoint_url', nargs='?',
 help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.")
 args = parser.parse_args()

 test_key_count = 10
```

```
test_iterations = 50
if args.endpoint_url:
 print(f"Getting each item from the table {test_iterations} times, "
 f"using the DAX client.")
 # Use a with statement so the DAX client closes the cluster after completion.
 with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url,
region_name='eu-west-1') as dax:
 test_start, test_end = get_item_test(
 test_key_count, test_iterations, dyn_resource=dax)
else:
 print(f"Getting each item from the table {test_iterations} times, "
 f"using the Boto3 client.")
 test_start, test_end = get_item_test(
 test_key_count, test_iterations)
print(f"Total time: {test_end - test_start:.4f} sec. Average time: "
 f"{(test_end - test_start)/ test_iterations}.")
```

#### 04-query-test.py

04-query-test.py 程式會在 Query 上執行 TryDaxTable 操作。

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key

def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
 """
 Queries the table a specified number of times. The time before the
 first iteration and the time after the last iteration are both captured
 and reported.

 :param partition_key: The partition key value to use in the query. The query
 returns items that have partition keys equal to this value.
 :param sort_keys: The range of sort key values for the query. The query returns
 items that have sort key values between these two values.
 :param iterations: The number of iterations to run.
 :param dyn_resource: Either a Boto3 or DAX resource.
 :return: The start and end times of the test.
 """
```

```
if dyn_resource is None:
 dyn_resource = boto3.resource("dynamodb")

table = dyn_resource.Table("TryDaxTable")
key_condition_expression = Key("partition_key").eq(partition_key) & Key(
 "sort_key"
).between(*sort_keys)

start = time.perf_counter()
for _ in range(iterations):
 table.query(KeyConditionExpression=key_condition_expression)
 print(".", end="")
 sys.stdout.flush()
print()
end = time.perf_counter()
return start, end

if __name__ == "__main__":
 # pylint: disable=not-context-manager
 parser = argparse.ArgumentParser()
 parser.add_argument(
 "endpoint_url",
 nargs="?",
 help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
)
 args = parser.parse_args()

 test_partition_key = 5
 test_sort_keys = (2, 9)
 test_iterations = 100
 if args.endpoint_url:
 print(f"Querying the table {test_iterations} times, using the DAX client.")
 # Use a with statement so the DAX client closes the cluster after completion.
 with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
 test_start, test_end = query_test(
 test_partition_key, test_sort_keys, test_iterations, dyn_resource=dax
)
 else:
 print(f"Querying the table {test_iterations} times, using the Boto3 client.")
 test_start, test_end = query_test(
 test_partition_key, test_sort_keys, test_iterations
)
```

```
print(
 f"Total time: {test_end - test_start:.4f} sec. Average time: "
 f"{(test_end - test_start)/test_iterations}."
)
```

## 05-scan-test.py

05-scan-test.py 程式會在 Scan 上執行 TryDaxTable 操作。

```
import argparse
import time
import sys
import amazondax
import boto3

def scan_test(iterations, dyn_resource=None):
 """
 Scans the table a specified number of times. The time before the
 first iteration and the time after the last iteration are both captured
 and reported.

 :param iterations: The number of iterations to run.
 :param dyn_resource: Either a Boto3 or DAX resource.
 :return: The start and end times of the test.
 """
 if dyn_resource is None:
 dyn_resource = boto3.resource("dynamodb")

 table = dyn_resource.Table("TryDaxTable")
 start = time.perf_counter()
 for _ in range(iterations):
 table.scan()
 print(".", end="")
 sys.stdout.flush()
 print()
 end = time.perf_counter()
 return start, end

if __name__ == "__main__":
 # pylint: disable=not-context-manager
 parser = argparse.ArgumentParser()
 parser.add_argument(
```

```
 "endpoint_url",
 nargs="?",
 help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
)
 args = parser.parse_args()

 test_iterations = 100
 if args.endpoint_url:
 print(f"Scanning the table {test_iterations} times, using the DAX client.")
 # Use a with statement so the DAX client closes the cluster after completion.
 with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
 test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
 else:
 print(f"Scanning the table {test_iterations} times, using the Boto3 client.")
 test_start, test_end = scan_test(test_iterations)
 print(
 f"Total time: {test_end - test_start:.4f} sec. Average time: "
 f"{(test_end - test_start)/test_iterations}."
)
```

## 06-delete-table.py

06-delete-table.py 程式會刪除 TryDaxTable。請在完成測試 Amazon DynamoDB Accelerator (DAX) 功能之後執行此程式。

```
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!")
```

## 修改現有應用程式以使用 DAX

如果您已有使用 Amazon DynamoDB 的 Java 應用程式，則可以進行修改，使其可存取您的 DynamoDB Accelerator (DAX) 叢集。您不需要重新撰寫整個應用程式，因為 DAX Java 用戶端與適用於 Java 2.x 的 AWS 開發套件中所含的 DynamoDB 低階用戶端十分相似。如需更多詳細資訊，請參閱[使用 DynamoDB 中的項目](#)。

### Note

此範例使用適用於 Java 2.x 的 AWS 開發套件。如需適用於 Java 1.x 版本的舊式開發套件，請參閱[修改適用於 Java 1.x 的開發套件的現有應用程式來使用 DAX](#)。

若要修改您的程式，請將 DynamoDB 用戶端取代為 DAX 用戶端。

```
Region region = Region.US_EAST_1;

// Create an asynchronous DynamoDB client
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
 .region(region)
 .build();

// Create an asynchronous DAX client
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
 .overrideConfiguration(Configuration.builder()
 .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
 .region(region)
 .addMetricPublisher(cloudWatchMetricsPub) // optionally enable SDK
metric collection
 .build())
 .build();
```

您也可以使用高階程式庫 (適用於 Java 2.x 的 AWS 開發套件的一部分)，以 DAX 用戶端取代 DynamoDB 用戶端。

```
Region region = Region.US_EAST_1;
DynamoDbAsyncClient dax = ClusterDaxAsyncClient.builder()
 .overrideConfiguration(Configuration.builder()
 .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
 .region(region)
 .build())
 .build();

DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
 .dynamoDbClient(dax)
 .build();
```

如需詳細資訊，請參閱[映射 DynamoDB 資料表中的項目](#)。

## 管理 DAX 叢集

本節將介紹一些常見的 Amazon DynamoDB Accelerator (DAX) 叢集管理任務。

### 主題

- [管理 DAX 叢集的 IAM 許可](#)
- [擴展 DAX 叢集](#)
- [自訂 DAX 叢集設定](#)
- [配置 TTL 設定](#)
- [DAX 的標記支援](#)
- [AWS CloudTrail 整合](#)
- [刪除 DAX 叢集](#)

## 管理 DAX 叢集的 IAM 許可

當您使用 AWS Management Console 或 AWS Command Line Interface (AWS CLI) 管理 DAX 叢集時，強烈建議您縮減使用者可執行的動作範圍。這樣做有利於降低風險，同時遵循最低權限。

以下的討論著重於 DAX 管理 API 的存取控制。如需更細資訊，請參閱《Amazon DynamoDB API 參考》中的 [Amazon DynamoDB Accelerator](#)。

**Note**

如需管理 AWS Identity and Access Management (IAM) 許可的詳細資訊，請參閱下列主題：

- IAM 和建立 DAX 叢集：[建立 DAX 叢集](#)。
- IAM 與 DAX 資料平面操作：[DAX 存取控制](#)。

針對 DAX 管理 API，您無法將 API 動作範圍侷限於某項特定資源。Resource 元素必須設為 "\*"。這與 DAX 資料平面 API 操作不同，例如 GetItem、Query 和 Scan。資料平面操作是透過 DAX 用戶端公開，而且這些操作的範圍可以侷限於特定的資源。

為了示範，請考慮以下 IAM 政策文件。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Action": [
 "dax:*"
],
 "Effect": "Allow",
 "Resource": [
 "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
]
 }
]
}
```

假設此政策的目的是允許叢集的 DAX 管理 API 呼叫 DAXCluster01，且僅限該叢集。

現在假設使用者發出以下 AWS CLI 命令。

```
aws dax describe-clusters
```

此命令會因為未授權異常而失敗，因為基礎 DescribeClusters API 呼叫範圍無法侷限於特定叢集。即使此政策在語法上有效，命令仍會失敗，因為 Resource 元素必須設為 "\*"。但若使用者執行某個程式，將 DAX 資料平面呼叫 (例如 GetItem 或 Query) 傳送至 DAXCluster01，則這些呼叫會成功。這是因為 DAX 資料平面 API 的範圍可侷限在特定的資源 (此案例中為 DAXCluster01)。

如果您想要撰寫單一的全方位 IAM 政策，將 DAX 管理 API 和 DAX 資料平面 API 都包含在內，建議您在政策文件中包含兩個不同的陳述式。其中一個陳述式應該處理 DAX 資料平面 API，而另一個陳述式則處理 DAX 管理 API。

以下範例政策示範這種方法。請注意 DAXDataAPIs 陳述式的範圍如何侷限在 DAXCluster01 資源範圍內，但 DAXManagementAPIs 的資源必須是 "\*"。每個陳述式中顯示的動作都僅供示範使用。您可以視需要為您的應用程式自訂他們。

```
{
 "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": [
 "*"
]
}
]
```

## 擴展 DAX 叢集

調整 DAX 叢集有兩種可用選項。第一個選項是 horizontal scaling (水平擴展)，您可在此將僅供讀取複本新增至叢集。第二個選項是 vertical scaling (垂直擴展)，您可在此選取不同的節點類型。如果需要如何為應用程式選擇適當叢集大小和節點類型的建議，請參閱 [DAX 叢集調整大小指南](#)。

### 水平擴展

使用水平擴展，您可以將更多的唯讀複本新增至叢集，改善輸送量。單一 DAX 叢集最多支援 10 個僅供讀取複本，您可在叢集執行時新增或移除複本。

以下 AWS CLI 範例示範如何增加或減少節點數目。--new-replication-factor 引數會指定叢集中的節點總數。其中一個節點是主要節點，其他節點則是僅供讀取複本。

```
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**

當您修改複寫因素時，叢集狀態會變更為 `modifying`。當修改完成時，狀態會變更為 `available`。

## 垂直擴展

如果您有大型的資料工作集，您的應用程式可能得益於使用較大的節點類型。較大的節點可讓叢集在記憶體中存放更多資料、減少快取未中，改善應用程式的整體應用程式效能 (DAX 叢集中的所有節點都必須是相同的類型。)

如果 DAX 叢集的寫入操作或快取未中率很高，使用較大的節點類型應可讓您的應用程式受益。寫入操作和快取未中會消耗叢集主節點上的資源，因此，使用較大的節點類型可能會提高主要節點的效能，進而為這類的操作提供較高的輸送量。

您無法在執行中的 DAX 叢集上修改節點類型。而是必須使用所需節點類型建立新的叢集。如需支援的節點類型清單，請參閱「[節點](#)」。

您可以使用 AWS Management Console、[AWS CloudFormation](#)、AWS CLI 或 [AWS 開發套件](#) 建立新的 DAX 叢集叢集。(針對 AWS CLI，請使用 `--node-type` 參數來指定節點類型。)

## 自訂 DAX 叢集設定

當您建立 DAX 叢集時，會使用下列預設設定：

- 啟用自動快取移出，存留時間 (TTL) 為 5 分鐘
- 無偏好的可用區域
- 無偏好的維護時段
- 停用通知

針對新的叢集，您可以在建立期間自訂設定。若要在 AWS Management Console 中執行此作業，請清除 `Use default settings` (使用預設設定) 以修改下列設定：

- **Network and Security (網路與安全)**：允許您在目前 AWS 區域內執行不同可用區域中的個別 DAX 叢集節點。如果您選擇 `No Preference` (無偏好設定)，節點會自動分散在可用區域中。

- **Parameter Group (參數群組)**：套用到叢集中每個節點的一組具名參數。您可以使用參數群組指定快取 TTL 行為。您可以隨時變更參數群組中任何指定參數的數值 (預設參數群組 `default.dax.1.0` 除外)。
- **Maintenance Window (維護時段)**：軟體更新和修補程式套用到叢集中節點的每週時間期間。您可以選擇開始日、開始時間和維護時段的持續時間。如果您選擇 No Preference (無偏好設定)，即會在每個區域的 8 小時時段內隨機選取維護時段。如需詳細資訊，請參閱 [Maintenance window \(維護時段\)](#)。

### Note

您也可以隨時在執行中的叢集上變更 Parameter Group (參數群組) 和 Maintenance Window (維護視窗)。

發生維護事件時，DAX 可以使用 Amazon Simple Notification Service (Amazon SNS) 通知您。若要設定通知，請從 Topic for SNS notification (SNS 通知主題) 選擇器中選擇選項。您可以建立新的 Amazon SNS 主題，或使用現有的主題。

如需設定和訂閱 Amazon SNS 主題的詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的 [Amazon SNS 入門](#)。

## 配置 TTL 設定

DAX 會維護兩種從 DynamoDB 讀取的資料快取：

- **Item cache (項目快取)**：適用於使用 `GetItem` 或 `BatchGetItem` 擷取的項目。
- **Query cache (查詢快取)**：適用於使用 `Query` 或 `Scan` 擷取的結果集。

如需詳細資訊，請參閱 [項目快取](#) 及 [查詢快取](#)。

這些快取每個的預設 TTL 都是 5 分鐘。如果您想要使用不同的 TTL 設定，您可以使用自訂參數群組啟動 DAX 叢集。若要在主控台上執行此作業，請在導覽窗格中選擇 DAX | Parameter groups (DAX | 參數群組)。

您也可以使用 AWS CLI 執行這些任務。以下範例示範如何使用自訂參數群組啟動新的 DAX 叢集。在此範例中，項目快取的 TTL 會設為 10 分鐘，而查詢快取的 TTL 會設為 3 分鐘。

1. 建立新的參數群組。

```
aws dax create-parameter-group \
 --parameter-group-name custom-ttl
```

2. 將項目快取 TTL 設成 10 分鐘 (600000 毫秒)。

```
aws dax update-parameter-group \
 --parameter-group-name custom-ttl \
 --parameter-name-values "ParameterName=record-ttl-millis,ParameterValue=600000"
```

3. 將查詢快取 TTL 設成 3 分鐘 (180000 毫秒)。

```
aws dax update-parameter-group \
 --parameter-group-name custom-ttl \
 --parameter-name-values "ParameterName=query-ttl-millis,ParameterValue=180000"
```

4. 驗證參數設定是否正確。

```
aws dax describe-parameters --parameter-group-name custom-ttl \
 --query "Parameters[*].[ParameterName,Description,ParameterValue]"
```

您現在可以使用此參數群組啟動新的 DAX 叢集。

```
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

您無法修改正由執行中 DAX 執行個體使用的參數群組。

## DAX 的標記支援

包括 DynamoDB 在內的許多 AWS 服務，都支援標記：以使用者定義的名稱標記資源的能力。您可將標籤指派給 DAX 叢集，讓您快速識別有相同標籤的所有 AWS 資源，或按您指派的標籤分類您的 AWS 帳單。



如需詳細資訊，請參閱 [將標籤新增至資源](#)。

## 使用 AWS Management Console

### 管理 DAX 叢集標籤

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在導覽窗格中，選擇 DAX 下方的 Clusters (叢集)。
3. 選擇您想要使用的叢集。
4. 選擇 Tags (標籤) 索引標籤。您可以在此新增、列出、編輯或刪除您的標籤。

當您滿意設定後，請選擇 Apply Changes (套用變更)。

## 使用 AWS CLI

當您使用 AWS CLI 管理 DAX 叢集標籤時，您必須先決定叢集的 Amazon Resource Name (ARN)。以下範例示範如何決定名為 MyDAXCluster 叢集的 ARN。

```
aws dax describe-clusters \
 --cluster-name MyDAXCluster \
 --query "Clusters[*].ClusterArn"
```

在輸出中，ARN 看起來會如下：`arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster`

以下範例示範如何標記叢集。

```
aws dax tag-resource \
 --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
 --tags="Key=ClusterUsage,Value=prod"
```

列出叢集的所有標籤。

```
aws dax list-tags \
 --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

您可以指定標籤的鍵來移除標籤。

```
aws dax untag-resource \
 --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

```
--resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
--tag-keys ClusterUsage
```

## AWS CloudTrail 整合

DAX 與 AWS CloudTrail 整合，可讓您稽核 DAX 叢集活動。您可以使用 CloudTrail 日誌來檢視在叢集層級進行的所有變更。您也可以查看對叢集元件進行的變更，例如節點、子網路群組和參數群組。如需詳細資訊，請參閱 [使用 AWS CloudTrail 記錄 DynamoDB 操作](#)。

## 刪除 DAX 叢集

如果您不再使用 DAX 叢集，建議您刪除它，以避免支付未使用資源的費用。

您可以使用主控台或 AWS CLI 來刪除 DAX 叢集。以下是範例。

```
aws dax delete-cluster --cluster-name mydaxcluster
```

## 監控 DAX

監控是維持 Amazon DynamoDB 加速器 (DAX) 和解決方案的可靠性、可用性和效能的 AWS 重要組成部分。您應該從 AWS 解決方案的所有部分收集監視資料，以便在發生多點失敗時更輕鬆地偵錯。

在開始監控 DAX 之前，應先建立監控計畫，為下列問題提供解答：

- 監控目標是什麼？
- 要監控哪些資源？
- 監控這些資源的頻率為何？
- 要使用哪些監控工具？
- 誰將執行監控任務？
- 發生問題時應該通知誰？

### 主題

- [監控工具](#)
- [使用 Amazon 監控 CloudWatch](#)
- [使用 AWS CloudTrail 記錄 DAX 操作](#)

## 監控工具

AWS 提供可用來監控 Amazon DynamoDB 加速器 (DAX) 的工具。您可以設定其中一些工具為您進行監控，而有些工具需要手動介入。建議您盡可能自動化監控任務。

### 主題

- [自動化監控工具](#)
- [手動監控工具](#)

## 自動化監控工具

您可以使用下列自動化監控工具來監看 DAX，並在發生錯誤時進行回報：

- Amazon CloudWatch 警示 — 觀看您指定期間內的單一指標，並根據指定臨界值在多個時段內相對於指定閾值的指標值執行一或多個動作。動作是傳送至亞馬遜簡單通知服務 (Amazon SNS) 主題或 Amazon EC2 Auto Scaling 政策的通知。CloudWatch 警示不會僅因為處於特定狀態而叫用動作；狀態必須已變更並維持指定數目的期間。如需詳細資訊，請參閱 [使用 Amazon 監控指標 CloudWatch](#)。
- Amazon CloudWatch 日誌 — 監控、存放和存取來自 AWS CloudTrail 或其他來源的日誌檔。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的 [監控日誌檔](#)。
- Amazon E CloudWatch vents — 匹配事件並將其路由到一個或多個目標函數或串流，以進行變更、擷取狀態資訊並採取糾正措施。有關更多信息，請參閱 [Amazon 用 CloudWatch 戶指南中的 Amazon CloudWatch 事件是什麼](#)。
- AWS CloudTrail 防護記錄監控 — 在帳戶之間共用記 CloudTrail 錄檔、即時監控記錄檔案，方法是將 CloudWatch 記錄檔傳送至記錄檔、以 Java 撰寫記錄處理應用程式，以及驗證記錄檔在傳送之後是否未變更 CloudTrail。若要取得更多資訊，請參閱 [《使用指南》中的〈AWS CloudTrail 使用 CloudTrail 記錄檔〉](#)。

## 手動監控工具

監視 DAX 的另一個重要部分是手動監視 CloudWatch 警報未涵蓋的項目。DAX、CloudWatch Trusted Advisor、和其他 AWS Management Console 儀表板可提供您 AWS 環境狀態的 at-a-glance 檢視。建議您也查看 DAX 上的日誌檔。

- DAX 儀表板會顯示下列項目：
  - 服務運作狀態

- CloudWatch 首頁會顯示下列內容：
  - 目前警示與狀態
  - 警示與資源的圖表
  - 服務運作狀態

此外，您可以使用執行 CloudWatch 以下操作：

- 建立 [自訂儀表板](#) 以監控您注重的服務。
- 用於疑難排解問題以及探索驅勢的圖形指標資料。
- 搜尋並瀏覽所有資 AWS 源指標。
- 建立與編輯要通知發生問題的警示。

## 使用 Amazon 監控 CloudWatch

您可以使用 Amazon 監控 DynamoDB 加速器 (DAX) CloudWatch，該 Amazon 會從 DAX 收集原始資料並將其處理為可讀且接近即時的指標。會記錄兩週期間的統計數字。接著，您就可以存取歷史資訊，並更清楚 Web 應用程式或服務的執行效能。根據預設，DAX 指標資料會 CloudWatch 自動傳送至。有關更多信息，請參閱 [什麼是 Amazon CloudWatch？](#) 在 Amazon 用 CloudWatch 戶指南。

### 主題

- [如何使用 DAX 指標？](#)
- [檢視 DAX 指標和維度](#)
- [建立 CloudWatch 警示以監控 DAX](#)
- [生產監控](#)

### 如何使用 DAX 指標？

DAX 回報的指標可提供資訊，您可透過不同方式加以分析。下列清單顯示一些常見的指標用途。這些是建議，以協助您開始，而不是完整清單。

如何？	相關指標
判斷是否發生任何系統錯誤	監控 <code>FaultRequestCount</code> 以判斷是否有任何請求導致 HTTP 500 (伺服器錯誤) 代碼。這可能表示基礎表格的 <a href="#">SystemErrors 測量結果</a> 中存在 DAX 內部服務錯誤或 HTTP 500。

如何？	相關指標
判斷是否發生任何使用者錯誤	監控 <code>ErrorRequestCount</code> 以判斷是否有任何請求導致 HTTP 400 (用戶端器錯誤) 代碼。如果您發現錯誤數量不斷成長，您可能希望進行調查並確保您傳送的是正確的用戶端請求。
判斷是否發生任何快取遺漏	監控 <code>ItemCacheMisses</code> 以判斷在快取中未找到某項目的次數， <code>QueryCacheMisses</code> 和 <code>ScanCacheMisses</code> 來判斷在快取中未找到某查詢或掃描結果的次數。
監控快取命中率	<p>使用「<a href="#">CloudWatch 公制數學</a>」可使用數學運算式定義快取命中率量度。</p> <p>例如，針對項目快取，您可以使用表達式 <code>m1/SUM([m1, m2])*100</code>，其中 <code>m1</code> 是 <code>ItemCacheHits</code> 指標而 <code>m2</code> 是叢集的 <code>ItemCacheMisses</code> 指標。針對查詢和掃描快取，您可以使用對應的查詢和掃描快取指標來遵循相同的模式。</p>

## 檢視 DAX 指標和維度

當您與 Amazon DynamoDB 互動時，它會將指標和維度傳送到 Amazon CloudWatch。您可以使用下列程序來檢視 DynamoDB Accelerator (DAX) 的指標。

### 檢視指標 (主控台)

指標會先依服務命名空間分組，再依各命名空間內不同的維度組合分類。

1. [請在以下位置開啟 CloudWatch 主控台。](https://console.aws.amazon.com/cloudwatch/) <https://console.aws.amazon.com/cloudwatch/>
2. 在導覽窗格中，選擇 指標。
3. 選取 DAX 命名空間。

### 檢視指標 (AWS CLI)

- 在命令提示中，使用下列命令。

```
aws cloudwatch list-metrics --namespace "AWS/DAX"
```

## DAX 指標與維度

以下各節包含 DAX 傳送至的量度和維度 CloudWatch。

### DAX 指標

下列是 DAX 提供的指標。CloudWatch 只有當指標的值非零時，DAX 才會傳送指標。

#### Note

CloudWatch 以一分鐘的間隔彙總下列 DAX 指標：

- 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

並非所有統計數字，例如 Average 或 Sum，皆適用於所有指標。不過，所有這些值都可透過 DAX 主控台取得，或透過使用 CloudWatch 主控台 AWS CLI、或 AWS SDK 來取得所有量度。在下表中，每個指標皆有適用於該指標的有效統計數字列表。

指標	描述
CPUUtilization	<p>節點或叢集的 CPU 使用率百分比。</p> <p>單位：Percent</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"> <li>• Minimum</li> <li>• Maximum</li> <li>• Average</li> </ul>
CacheMemoryUtilization	<p>節點或叢集上項目快取和查詢快取正在使用的可用快取記憶體百分比。快取的資料會在記憶體使用率達到 100% 之前開始移除 (請參閱 EvictedSize 指標)。如果 CacheMemoryUtilization 在任何節點上達到 100%，寫入請求將進行調節，且您應該考慮切換到節點類型較大的叢集。</p> <p>單位：Percent</p>

指標	描述
	<p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li></ul>
NetworkBytesIn	<p>節點或叢集在所有網路介面上收到的位元組數目。</p> <p>單位：Bytes</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li></ul>
NetworkBytesOut	<p>節點或叢集在所有網路介面上送出的位元組數目。此指標識別單一節點或叢集上的傳出流量 (位元組數目)。</p> <p>單位：Bytes</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li></ul>
NetworkPacketsIn	<p>節點或叢集在所有網路介面上收到的封包數目。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li></ul>



指標	描述
NetworkPacketsOut	<p>節點或叢集在所有網路介面上送出的封包數目。此指標識別單一節點或叢集上的傳出流量 (封包數目)。</p> <p>單位 : Count</p> <p>有效的統計數字 :</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li></ul>
GetItemRequestCount	<p>節點或叢集處理的 GetItem 請求數量。</p> <p>單位 : Count</p> <p>有效的統計數字 :</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
BatchGetItemRequestCount	<p>節點或叢集處理的 BatchGetItem 請求數量。</p> <p>單位 : Count</p> <p>有效的統計數字 :</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>

指標	描述
BatchWriteItemRequestCount	<p>節點或叢集處理的 BatchWriteItem 請求數量。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
DeleteItemRequestCount	<p>節點或叢集處理的 DeleteItem 請求數量。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
PutItemRequestCount	<p>節點或叢集處理的 PutItem 請求數量。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>

指標	描述
UpdateItemRequestCount	<p>節點或叢集處理的 UpdateItem 請求數量。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
TransactWriteItemsCount	<p>節點或叢集處理的 TransactWriteItems 請求數量。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
TransactGetItemsCount	<p>節點或叢集處理的 TransactGetItems 請求數量。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>

指標	描述
ItemCacheHits	<p>從節點或叢集快取傳回項目的次數。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
ItemCacheMisses	<p>項目不在節點或叢集快取中且必須從 DynamoDB 擷取的次數。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
QueryCacheHits	<p>從節點或叢集快取傳回查詢結果的次數。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>

指標	描述
QueryCacheMisses	<p>查詢結果不在節點或叢集快取中且必須從 DynamoDB 擷取的次數。</p> <p>單位 : Count</p> <p>有效的統計數字 :</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
ScanCacheHits	<p>從節點或叢集快取傳回掃描結果的次數。</p> <p>單位 : Count</p> <p>有效的統計數字 :</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>

指標	描述
ScanCacheMisses	<p>掃描結果不在節點或叢集快取中且必須從 DynamoDB 擷取的次數。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
TotalRequestCount	<p>節點或叢集處理的請求總數。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>

指標	描述
ErrorRequestCount	<p>節點或叢集回報的導致使用者錯誤的請求總數。包含節點或叢集調節的請求。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
ThrottledRequestCount	<p>節點或叢集調節的請求總數。由 DynamoDB 調節的請求不包含在內，但可以使用 <a href="#">DynamoDB 指標</a> 監控这样的请求。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>

指標	描述
FaultRequestCount	<p>節點或叢集回報的導致內部錯誤的請求總數。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
FailedRequestCount	<p>節點或叢集回報的導致錯誤的請求總數。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
QueryRequestCount	<p>節點或叢集處理的查詢數目。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>



指標	描述
ScanRequestCount	<p>節點或叢集處理的掃描請求數目。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
ClientConnections	<p>用戶端連線至節點或叢集的同時連線數目。</p> <p>單位：Count</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
EstimatedDbSize	<p>節點或叢集在項目快取及查詢快取中快取資料量近似值。</p> <p>單位：Bytes</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li></ul>

指標	描述
EvictedSize	<p>節點或叢集為騰出空間給新請求的資料而移除的資料量。如果遺漏率升高，並且此指標也不斷增加，這可能表示您的工作集已經增加。您應該考慮切換到節點類型較大的叢集。</p> <p>單位：Bytes</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• Sum</li></ul>
CPUCreditUsage	<p>節點為 CPU 使用率花費的 CPU 點數數量。一個 CPU 額度等於一個 vCPU 以 100% 使用率執行 1 分鐘，或同等的 vCPU、使用率與時間的組合 (例如，一個 vCPU 以 50% 使用率執行 2 分鐘，或兩個 vCPU 以 25% 使用率執行 2 分鐘)。</p> <p>CPU 額度指標僅提供 5 分鐘頻率。如果您要指定大於 5 分鐘的期間，請使用 Sum 統計數字代替 Average。</p> <p>單位：Credits (vCPU-minutes)</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>

指標	描述
CPUCreditBalance	<p>自節點啟動或開始後，累積獲得的 CPU 點數數量。</p> <p>獲得額度後，額度會在額度餘額中累積，並在支付額度時，從額度餘額中移出。點數餘額有最大值限制，它取決於 DAX 節點大小。到達限制之後，任何獲得的新額度都會遭到捨棄。</p> <p>CPUCreditBalance 中的點數可供節點支付以大幅提升並超越基準 CPU 使用率。</p> <p>單位：Credits (vCPU-minutes)</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>
CPUSurplusCreditBalance	<p>當 CPUCreditBalance 值為 0 時，DAX 節點已支出的剩餘點數數量。</p> <p>CPUSurplusCreditBalance 值由獲得的 CPU 額度支付。如果剩餘點數超過節點在 24 小時期間可獲得的最大點數數量，超過最大值的支出剩餘點數將必須負擔額外的費用。</p> <p>單位：Credits (vCPU-minutes)</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>

指標	描述
CPUSurplusCreditsCharged	<p>若支出剩餘額度數量未由獲得的 CPU 額度付清，會產生額外的費用。</p> <p>發生以下情況時，將收取支出剩餘點數的費用：支出剩餘點數超過節點在 24 小時期間可獲得的最大點數數量。在小時結束或節點終止時，將收取超過最大值的支出剩餘點數的費用。</p> <p>單位：Credits (vCPU-minutes)</p> <p>有效的統計數字：</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>

**Note**

CPUCreditUsage、CPUCreditBalance、CPUSurplusCreditBalance 和 CPUSurplusCreditsCharged 指標僅適用於 T3 節點。

## DAX 指標的維度

DAX 指標是由帳戶、叢集 ID 或叢集 ID 和節點 ID 結合值量化。您可以使用 CloudWatch 主控台來擷取下表中任何維度的 DAX 資料。

維度	描述
Account	提供帳戶中所有節點的彙總統計數字。
ClusterId	將資料限制為叢集。
ClusterId, NodeId	將資料限制為叢集內的節點。

## 建立 CloudWatch 警示以監控 DAX

您可以建立 Amazon CloudWatch 警示，在警示狀態變更時傳送 Amazon 簡單通知服務 (Amazon SNS) 訊息。警示會在您指定的期間監看單一指標。警示會根據在數個期間與指定閾值相關的指標值，來執行一個或多個動作。此動作是傳送到 Amazon SNS 主題或 Auto Scaling 政策的通知。警示只會呼叫持續狀態變更的動作。CloudWatch 警報不會僅僅因為它們處於特定狀態而叫用動作。狀態必須已變更，且在指定的期間數內維持此狀態。

我要如何收到這些查詢快取遺漏的通知？

1. 建立 Amazon SNS 主題 (arn:aws:sns:us-west-2:522194210714:QueryMissAlarm)。

如需詳細資訊，請參閱 [Amazon CloudWatch 使用者指南中的設定 Amazon 簡單通知服務](#)。

2. 建立警示。

```
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. 測試警示。

```
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

您可以增加或減少閾值，使其符合您的應用程式需求。您也可以使用 [CloudWatch Metric Math](#) 來定義快取遺漏率量度，並設定該量度的警示。

如果請求導致叢集中的內部錯誤，我該如何收到通知？

1. 建立 Amazon SNS 主題 `arn:aws:sns:us-west-2:123456789012:notify-on-system-errors`。

如需詳細資訊，請參閱 [Amazon CloudWatch 使用者指南中的設定 Amazon 簡單通知服務](#)。

2. 建立警示。

```
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. 測試警示。

```
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
```

## 生產監控

您應在各個時間點和不同的負載條件下測量效能，以在您的環境中確立 DAX 正常效能的基準。當您監控 DAX 時，應該考慮存放歷史監控資料。這個存放的資料會提供基準，讓您與目前的效能資料比較，識別出正常的效能模式和效能異常狀況，再規劃方式來處理問題。

若要確立基準，您至少應在負載測試和生產階段監控下列項目。

- CPU 使用率和節流請求，可讓您判斷是否需要在叢集中使用大型節點類型。您可以透過 CPUUtilization CloudWatch 指標取得叢集的 CPU 使用率。
- 操作延遲 (在用戶端測量) 應該保持在應用程式的延遲需求之內。
- 錯誤率應該保持較低，如 ErrorRequestCount FaultRequestCount、和 FailedRequestCount CloudWatch 指標所示。
- 網路位元組耗用量，因此您可以判斷叢集中是否需要使用更多節點或更大的節點類型。NetworkBytesIn 和 NetworkBytesOut 指標可在中使用 CloudWatch，您應該將它們與執行個體可用基準頻寬進行比較，如 [此處](#) 所述。

#### Note

Amazon EC2 記錄的可用基準頻寬是以每秒 GB (Gbps) 為單位，而 NetworkBytesIn 和 NetworkBytesOut 指標是以每分鐘 GB (GBpm) 為單位。若要將 Gbps 轉換成 GBpm 並測量使用率，請將基準頻寬乘以 7.5。

- 快取記憶體使用率和移出的大小，如此就能判斷叢集的節點類型是否有足夠的記憶體可保留您的工作集，若是不足，則切換至較大的節點類型。

**Note**

在大量快取遺漏和寫入的情況下，快取記憶體使用率可能增加達 100%，且可能造成可用性停擺。

- 用戶端連接，可讓您監控叢集連接中任何無法解釋的峰值。

## 使用 AWS CloudTrail 記錄 DAX 操作

Amazon DynamoDB 加速器 (DAX) 與這項服務整合在一起 AWS CloudTrail，該服務可提供 DAX 中使用者、角色或 AWS 服務所採取的動作記錄。

若要進一步了解 DAX 和 CloudTrail，請參閱中的〈DynamoDB 硬碟加速器 (DAX)〉一節。[使用 AWS CloudTrail 記錄 DynamoDB 操作](#)

## DAX T3/T2 爆量執行個體

DAX 可讓您在固定效能執行個體 (例如 R4 和 R5) 與爆量效能執行個體 (例如 T2 和 T3) 之間選擇。爆量效能執行個體提供基準水準的 CPU 效能，並可在需要時大幅超越基準水準。

基準效能與大幅提升效能的能力，取決於 CPU 的點數。工作負載低於基準水準閾值時，爆量效能執行個體會以執行個體大小決定的速率持續累積 CPU 點數。當工作負載增加時，則可能會使用這些點數。一個 CPU 點數提供一分鐘、一個 CPU 核心的完整效能。

許多工作負載不需要一致的高 CPU 水準，但可在需要時受益於完全運用非常快速的 CPU。爆量效能執行個體是專為這些使用案例而設計。如果您的資料庫需要一致的高 CPU 效能，我們建議您使用固定效能執行個體。

## DAX T2 執行個體系列

DAX T2 執行個體是爆量效能執行個體，除了提供基準水準的 CPU 效能之外，還可在需要時大幅超越基準水準。T2 執行個體是需要可預測價格的測試與開發工作負載的理想選擇。DAX T2 執行個體專為標準模式設定，換言之，如果執行個體的累積點數不足，CPU 使用率則會逐漸降到基準水準。如需標準模式的詳細資訊，請參閱《Amazon EC2 Linux 執行個體使用者指南》中的[爆量效能執行個體的標準模式](#)。



## DAX T3 執行個體系列

DAX T3 執行個體是下一代爆量的一般用途執行個體類型，其可提供 CPU 基準效能，而且只要有所需要，就可大幅提升 CPU 用量。T3 執行個體在運算、記憶體和網路資源之間取得平衡，非常適合 CPU 使用率中等、遇到暫時使用量峰值的工作負載。DAX T3 執行個體設定為無限制模式，這表示它們可以在 24 小時時段內大幅提升並超越基準，但需額外付費。如需無限制模式的詳細資訊，請參閱《Amazon EC2 Linux 執行個體使用者指南》中的[爆量效能執行個體的無限模式](#)。

只要工作負載需要，DAX T3 執行個體就能維持高 CPU 效能。針對大多數的一般用途工作負載，T3 執行個體可提供足夠效能，不會收取任何額外費用。當 T3 執行個體的 CPU 平均使用率在 24 小時時段內等於或低於基準時，T3 執行個體的每小時價格將自動涵蓋所有暫時使用量峰值。

例如，`dax.t3.small` 執行個體以每小時 24 個 CPU 點數的速率持續接收點數。此功能提供相當於一個 CPU 核心 20% 的基準效能 ( $20\% \times 60 \text{ 分鐘} = 12 \text{ 分鐘}$ )。如果執行個體未使用其收到的點數，則會將這些點數儲存在其 CPU 點數餘額中，最多可達 576 個 CPU 點數。當 `t3.small` 執行個體需要爆量到一個核心的 20% 以上時，它會提取 CPU 點數餘額，以便自動處理此突增的使用量。

雖然 DAX T2 執行個體會 CPU 點數提取到零後限制在基準效能，但即使 CPU 點數餘額為零，DAX T3 執行個體仍可大幅提升並超越基準。對於大多數的工作負載 (CPU 平均使用率等於或低於基準效能)，`t3.small` 的每小時基本價格能涵蓋所有 CPU 爆量。如果執行個體在 CPU 點數餘額在提取到零後以平均 25% 的 CPU 使用率 (高於基準 5%) 執行 24 小時，則會額外收取 11.52 美分的費用 ( $9.6 \text{ 美分} / \text{vCPU-小時} \times 1 \text{ vCPU} \times 5\% \times 24 \text{ 小時}$ )。如需定價詳細資訊，請參閱 [Amazon DynamoDB 定價](#)。

## DAX 存取控制

DynamoDB Accelerator (DAX) 可與 DynamoDB 一起使用，為您的應用程式無縫新增快取層。但是，DAX 和 DynamoDB 具有不同的存取控制機制。這兩種服務都使用 AWS Identity and Access Management (IAM) 來實作各自的安全政策，但 DAX 和 DynamoDB 的安全模型不同。

我們強烈建議您了解兩種安全模型，以便為使用 DAX 的應用程式實作合適的安全措施。

本節說明 DAX 提供的存取控制機制，並提供範例 IAM 政策。您可以使用此政策來為您的需求量身打造。

使用 DynamoDB，您可以建立限制使用者在個別 DynamoDB 資源上可執行動作的 IAM 政策。例如，您可以建立使用者角色，只允許使用者對特定 DynamoDB 資料表執行唯讀動作。(如需詳細資訊，請參閱「[Amazon DynamoDB 的 Identity and Access Management](#)」。) 相較之下，DAX 安全模型著重在叢集安全，以及叢集代替您執行 DynamoDB API 動作的能力。

### ⚠ Warning

若您目前使用 IAM 角色及政策限制對 DynamoDB 資料表資料的存取，使用 DAX 可以推翻這些政策。例如，使用者可透過 DAX 存取 DynamoDB 資料表，但無法藉由直接存取 DynamoDB 來明確存取相同的資料表。如需詳細資訊，請參閱 [Amazon DynamoDB 的 Identity and Access Management](#)。

DAX 不會強制對 DynamoDB 上的資料執行使用者層級隔離。相反的，使用者會在存取該叢集時，繼承 DAX 叢集 IAM 政策的許可。因此，透過 DAX 存取 DynamoDB 資料表時，唯一有效的存取控制便是 DAX 叢集 IAM 政策的許可。其他任何許可都不會獲得承認。

若您需要隔離，我們建議您建立其他 DAX 叢集，並為每個叢集劃定 IAM 政策的範圍。例如，您可以建立多個 DAX 叢集，然後只允許每個叢集存取單一資料表。

## DAX 的 IAM 服務角色

建立 DAX 叢集時，您必須為叢集與 IAM 角色建立關聯。這稱為叢集的服務角色。

假設您想要建立名為 DAXCluster01 的新 DAX 叢集。您可以建立名為 DAX 的服務角色 ServiceRole，並將該角色與 Dax Cluster01 產生關聯。DAX 政策 ServiceRole 將會代表與 DaxCluster01 互動的使用者，定義 DaxCluster01 可執行的動作。

建立服務角色時，必須指定 DAX ServiceRole 與 DAX 服務本身之間的信任關係。信任關聯會判斷可取得角色及使用其許可的實體。以下是 DAX 的信任關係文件範例 ServiceRole：

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "dax.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
```

此信任關係可讓 DAX 叢集代表您執行 DAX，ServiceRole 並執行 DynamoDB API 呼叫。

允許的 DynamoDB API 動作會在您附加至 DAX 的 IAM 政策文件中說明。ServiceRole 以下為範例政策文件。

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

此政策允許 DAX 在 DynamoDB 資料表上執行必要 DynamoDB API 動作。DAX 需要 `dynamodb:DescribeTable` 動作才能維護資料表的相關中繼資料，其他則是在表內項目上執行的讀取與寫入動作。該資料表名為 `Books`，位於 `us-west-2` 區域，由 AWS 帳戶 ID `123456789012` 擁有。

#### Note

DAX 支援機制，以防止跨服務存取期間混淆的副問題。如需詳細資訊，請參閱《IAM 使用者指南》中的[混淆代理問題](#)。

## 允許 DAX 叢集存取的 IAM 政策

建立 DAX 叢集後，您需要授予使用者許可，讓使用者能夠存取 DAX 叢集。

例如，假設您希望將存取 DAXCluster01 的許可授予名為 Alice 的使用者。您必須先建立 IAM 政策 (AliceAccessPolicy) 來定義收件者可存取的 DAX 叢集和 DAX API 動作。您接著便會藉由將此政策連接到使用者 Alice，來授予存取。

下列政策文件會給予收件人 DAXCluster01 的完整存取。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Action": [
 "dax:*"
],
 "Effect": "Allow",
 "Resource": [
 "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
]
 }
]
}
```

政策文件允許 DAX 叢集的存取，但並未授予任何 DynamoDB 許可。(DynamoDB 許可會由 DAX 服務角色授予。)

針對使用者 Alice，您首先要使用先前顯示的政策文件建立 AliceAccessPolicy。您接著會將政策連接到 Alice。

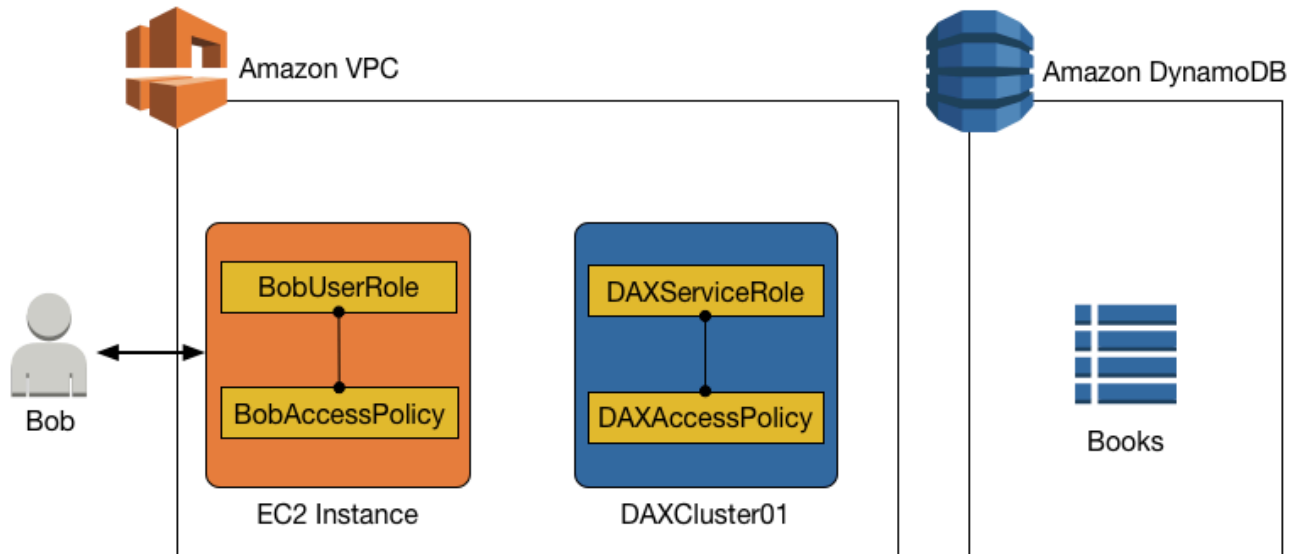
### Note

除了將政策連接到使用者之外，您也可以將政策連接到 IAM 角色。透過這種方式，所有取得該角色的使用者都會擁有您在政策中定義的許可。

使用者政策結合 DAX 服務角色，會判斷收件人可透過 DAX 存取的 DynamoDB 資源和 API 動作。

## 使用案例：存取 DynamoDB 和 DAX

下列案例可以幫助您進一步了解用於 DAX 的 IAM 政策。(本節剩餘的部分將參考這個案例)。以下圖表顯示案例的高層級概觀。

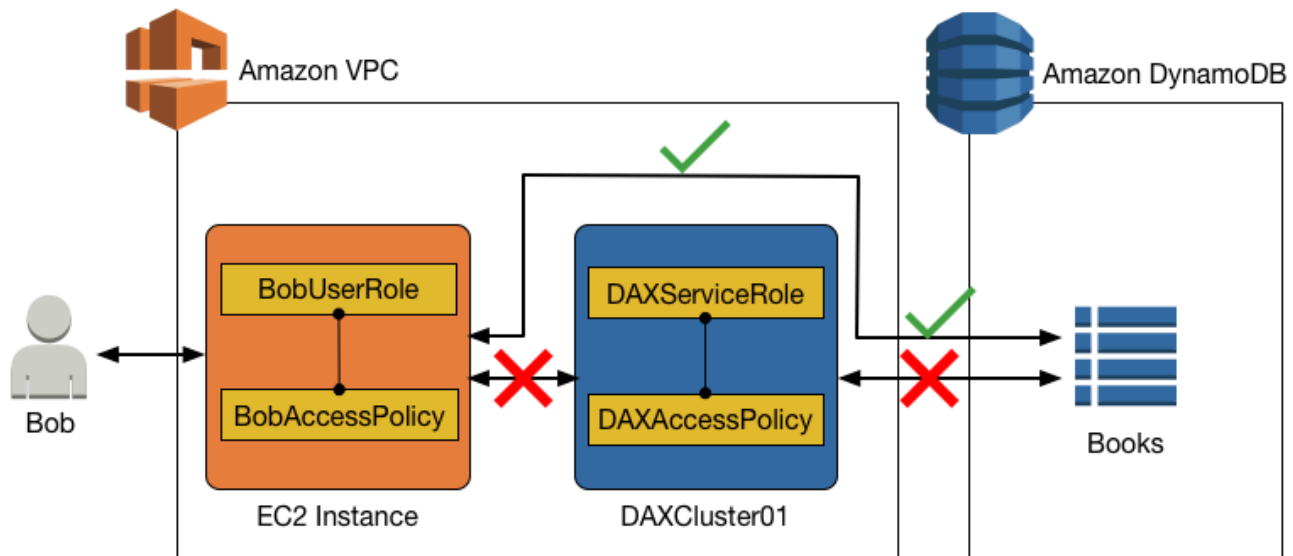


在此案例中，有下列實體：

- 使用者 (Bob)。
- IAM 角色 (BobUserRole)。Bob 會在執行時間取得此角色。
- IAM 政策 (BobAccessPolicy)。此政策會連接到 BobUserRole。BobAccessPolicy 會定義 BobUserRole 允許存取的 DynamoDB 和 DAX 資源。
- DAX 叢集 (DAXCluster01)。
- IAM 服務角色 (DAXServiceRole)。此角色允許 DAXCluster01 存取 DynamoDB。
- IAM 政策 (DAXAccessPolicy)。此政策會連接到 DAXServiceRole。DAXAccessPolicy 會定義 DAXCluster01 允許存取的 DynamoDB API 和資源。
- DynamoDB 資料表 (Books)。

BobAccessPolicy 和 DAXAccessPolicy 中的政策陳述式組合會決定 Bob 可以如何使用 Books 資料表。例如，Bob 可能可以直接 (使用 DynamoDB 端點)、間接 (使用 DAX 叢集) 或同時使用這兩種方式存取 Books。Bob 也可能可以從 Books 讀取資料、寫入資料到 Books，或同時具有這兩種許可。

## 存取 DynamoDB，但不可使用 DAX 存取



您可以允許直接存取 DynamoDB 資料表，同時防止透過 DAX 叢集間接進行存取。如需直接存取 DynamoDB，BobUserRole 的許可是由 BobAccessPolicy (連接到角色) 決定。

### DynamoDB 的唯讀存取 (僅限)

Bob 可以使用 BobUserRole 存取 DynamoDB。連接到此角色 (BobAccessPolicy) 的 IAM 政策會判斷 BobUserRole 可存取的 DynamoDB 資料表，以及 BobUserRole 可呼叫的 API。

考慮下列 BobAccessPolicy 的政策文件。

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

當此文件連接到 BobAccessPolicy 時，它會允許 BobUserRole 存取 DynamoDB 端點，並對 Books 資料表執行唯讀操作。

DAX 沒有出現在此政策中，因此透過 DAX 進行存取會遭到拒絕。

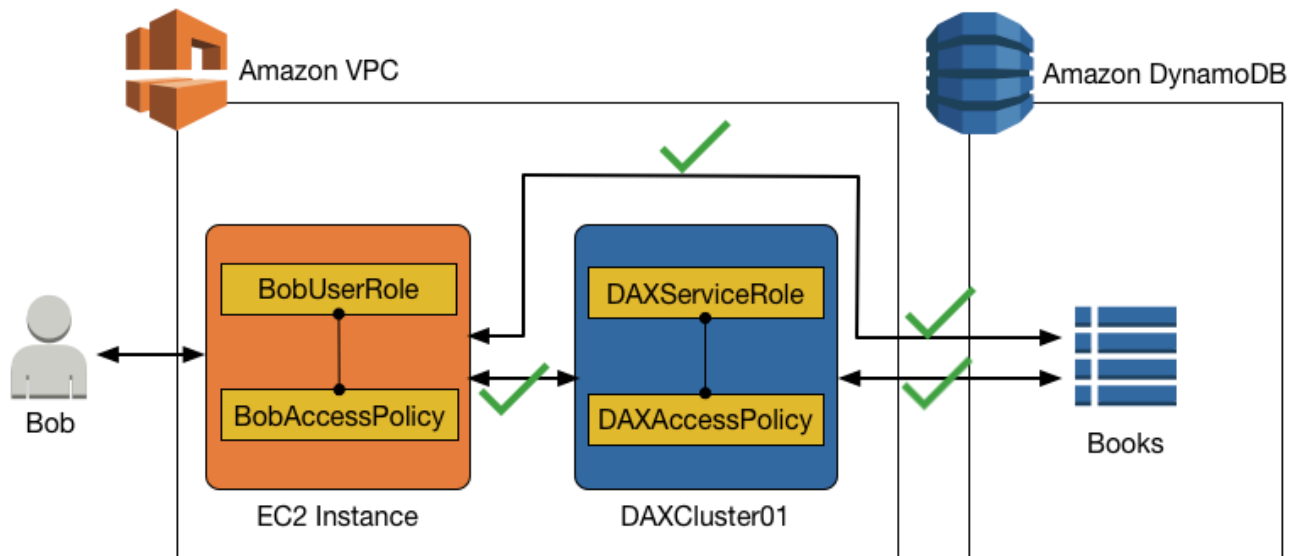
## DynamoDB 的讀取/寫入存取 (僅限)

若 BobUserRole 需要 DynamoDB 的讀寫存取，將適用以下政策。

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

同樣的，DAX 沒有出現在此政策中，因此透過 DAX 進行存取會遭到拒絕。

## 存取 DynamoDB 和 DAX



若要允許存取 DAX 叢集，您必須在 IAM 政策中包含 DAX 專屬的動作。

下列 DAX 專屬動作會對應到他們在 DynamoDB API 中名稱相似的動作：

- `dax:GetItem`
- `dax:BatchGetItem`
- `dax:Query`
- `dax:Scan`
- `dax:PutItem`
- `dax:UpdateItem`
- `dax>DeleteItem`
- `dax:BatchWriteItem`
- `dax:ConditionCheckItem`

這同樣適用於 `dax:EnclosingOperation` 條件金鑰。



## DynamoDB 的唯讀存取及 DAX 的唯讀存取

假設 Bob 需要透過 DynamoDB 及 DAX 對 Books 資料表進行唯讀存取。以下政策 (連接到 BobUserRole) 會授予這項存取。

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

政策具有 DAX 存取的陳述式 (DAXAccessStmt) 及另一個 DynamoDBAccess 的陳述式 (DynamoDBAccessStmt)。這些陳述式會允許 Bob 向 DAXCluster01 傳送 GetItem、BatchGetItem、Query 及 Scan 請求。

但是，DAXCluster01 的服務角色也需要對 DynamoDB 中 Books 資料表的唯讀存取。以下連接到 DAXServiceRole 的 IAM 政策會滿足這項需求。

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

## 讀寫存取 DynamoDB 及使用 DAX 進行唯讀存取

針對給定的使用者角色，您可以提供 DynamoDB 資料表的讀寫存取，同時也允許透過 DAX 進行唯讀存取。

針對 Bob，BobUserRole 的 IAM 政策需要允許對 Books 資料表進行 DynamoDB 讀取及寫入動作，同時也支援透過 DAXCluster01 進行唯讀動作。

以下 BobUserRole 的範例政策文件會授予這項存取。

```
{
 "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"
}
]
}

```

此外，DAXServiceRole 將需要允許 DAXCluster01 對 Books 資料表執行唯讀動作的 IAM 政策。

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "DynamoDBAccessStmnt",
 "Effect": "Allow",
 "Action": [
 "dynamodb:GetItem",
 "dynamodb:BatchGetItem",
 "dynamodb:Query",
 "dynamodb:Scan",
 "dynamodb:DescribeTable"
],
 "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
 }
]
}

```

## 讀寫存取 DynamoDB 及讀寫存取 DAX

現在假設 Bob 需要對 Books 資料表進行讀寫存取，無論是直接透過 DynamoDB，或間接透過 DAXCluster01。以下政策文件 (連接到 BobAccessPolicy) 會授予這項存取。

```

{

```

```

"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"
 }
]
}

```

此外，DAXServiceRole 將需要允許 DAXCluster01 對 Books 資料表執行讀寫動作的 IAM 政策。

```

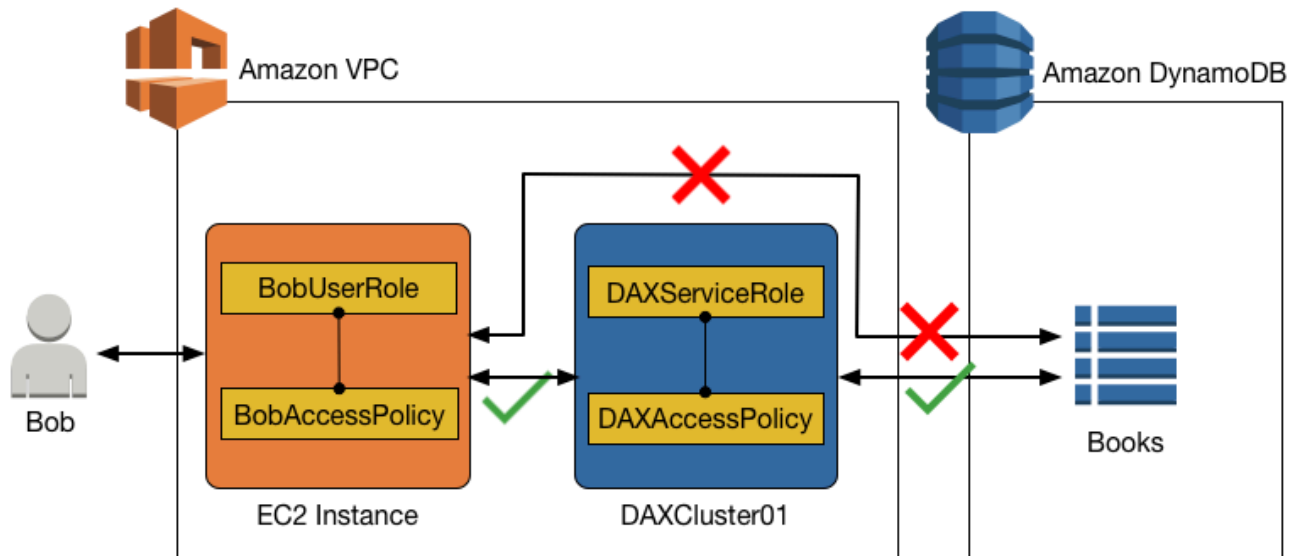
{
 "Version": "2012-10-17",
 "Statement": [

```

```
{
 "Sid": "DynamoDBAccessStmnt",
 "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"
}
]
```

## 透過 DAX 存取 DynamoDB，但不直接存取 DynamoDB

在此案例中，Bob 可透過 DAX 存取 Books 資料表，但無法直接存取 DynamoDB 中的 Books 資料表。因此，當 Bob 取得 DAX 的存取時，他也同時會取得透過其他方式皆無法取得的 DynamoDB 資料表的存取。當您為 DAX 服務角色設定 IAM 政策時，請記得任何透過使用者存取政策取得 DAX 叢集存取的使用者，都會取得在該政策中指定資料表的存取。在此情況下，BobAccessPolicy 可以存取 DAXAccessPolicy 中指定的資料表。



若您目前使用 IAM 角色及政策限制對 DynamoDB 資料表和資料的存取，使用 DAX 可以推翻這些政策。在下列政策中，Bob 可透過 DAX 存取 DynamoDB 資料表，但無法明確的直接存取 DynamoDB 中相同的資料表。

以下連接到 BobUserRole 的政策文件 (BobAccessPolicy) 會授予這項存取。

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

```
 }
]
}
```

在此存取政策中，沒有任何可直接存取 DynamoDB 的許可。

搭配 BobAccessPolicy，下列 DAXAccessPolicy 可讓 BobUserRole 存取 DynamoDB 資料表 Books，即使 BobUserRole 無法直接存取 Books 資料表。

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

如此範例所示，當您設定使用者存取原則和 DAX 叢集存取原則的存取控制時，您必須完全瞭解 end-to-end 存取權限，以確保遵守最低權限原則。您也必須確保給予使用者存取 DAX 叢集不會推翻先前建立的存取控制政策。

## DAX 靜態加密

Amazon DynamoDB Accelerator (DAX) 靜態加密可協助保護您的資料免於發生未經授權的基礎儲存存取，為資料提供另一層保護。組織政策、行業或政府法規，以及合規要求可能需要使用靜態加密來保護您的資料。您可以使用加密來提高部署在雲端中的應用程式資料安全性。

使用靜態加密時，磁碟上的 DAX 所維持的資料使用 256 位元進階加密標準 (也稱為 AES-256 加密) 來加密。DAX 將資料寫入磁碟，做為自主要節點到僅供讀取複本的變更環節之一。

DAX 靜態加密會自動與 AWS Key Management Service (AWS KMS) 整合，以管理用於加密叢集的單一服務預設金鑰。如果建立加密的 DAX 叢集時服務預設金鑰不存在，AWS KMS 會自動建立一個新的 AWS 受管金鑰。此金鑰可搭配未來建立的加密叢集使用。AWS KMS 結合了安全、高可用性的硬體和軟體，以提供針對雲端調整的金鑰管理系統。

在資料加密後，DAX 可以透明的方式處理您的資料解密，同時盡量降低對效能的影響。您不需要修改應用程式即可使用加密。

#### Note


DAX 不會為每個單一 DAX 操作呼叫 AWS KMS。DAX 只會在叢集啟動時使用金鑰。即使撤銷了存取，DAX 仍會繼續存取資料，直到叢集關閉。不支援客戶指定的 AWS KMS 金鑰。

DAX 靜態加密可用於下列叢集節點類型。

系列	節點類型
記憶體最佳化 (R4 與 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



系列	節點類型
	dax.r5.8xlarge
	dax.r5.12xlarge
	dax.r5.16xlarge
	dax.r5.24xlarge
一般用途 (T2)	dax.t2.small
	dax.t2.medium
一般用途 (T3)	dax.t3.small
	dax.t3.medium

 Important

DAX 靜態加密不支援 `dax.r3.*` 節點類型。

您可以在建立叢集後啟用或停用靜態加密。若在建立時未啟用靜態加密，您必須重新建立叢集以啟用靜態加密。

無需額外費用即可提供 DAX 靜態加密 (須支付 AWS KMS 加密金鑰費用)。如需定價資訊，請參閱 [Amazon DynamoDB 定價](#)。

## 使用 AWS Management Console 啟用靜態加密

請遵循下列步驟，使用主控台在資料表上啟用 DAX 靜態加密。

### 啟用 DAX 靜態加密

1. 登入 AWS Management Console，並在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在主控台左側的導覽窗格中，選擇 DAX 下的 Clusters (叢集)。
3. 選擇 Create cluster (建立叢集)。

- 針對 Cluster name (叢集名稱) 輸入您叢集的短名稱。為叢集中的所有節點選擇 Node type (節點類型)，而對於叢集大小，請使用 **3** 節點。
- 在 Encryption (加密) 中，請確認已選擇 Enable Encryption (啟用加密)。

### 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.

- 在選擇 IAM 角色、子網路群組、安全群組、以及叢集設定後，請選擇 Launch cluster (啟動叢集)。

若要確認叢集是否加密，檢查 Clusters (叢集) 窗格下的叢集詳細資訊。加密應 ENABLED (啟用)。

## DAX 傳輸中加密

Amazon DynamoDB Accelerator (DAX) 支援在應用程式與 DAX 叢集之間傳輸資料時進行加密，讓您能夠在具有嚴格加密要求的應用程式中使用 DAX。

無論您是否選擇傳輸中加密，應用程式與 DAX 叢集之間的流量都會保留在 Amazon VPC 中。此流量會路由至彈性網路介面，其中包含 VPC 中附加至叢集節點的私有 IP。您可以將 VPC 作為信任界限使用，透過標準工具 (例如安全群組、搭配網路 ACL 的子網路分隔和 VPC 流程追蹤) 掌控資料的安全性。DAX 傳輸中加密會新增至此機密性基準，確保應用程式與叢集之間的所有請求和回應都經由 Transport Layer Security (TLS) 加密，並透過驗證叢集 x509 憑證，對叢集的連線進行身份驗證。如果您在建立 DAX 叢集時選擇 [encryption at rest](#) (靜態加密)，也可以加密 DAX 寫入磁碟的資料。

您可以輕鬆透過 DAX 使用傳輸中加密功能。僅需在建立新叢集時選取此選項，並在應用程式中使用任何 [DAX 用戶端](#) 的最新版本。使用傳輸中加密的叢集不支援未加密的流量，因此不會設錯應用程式並略過加密。DAX 用戶端在建立連線時會使用叢集的 x509 憑證來驗證叢集的身分，以此確保 DAX 請求會前往預期的目的地。所有建立 DAX 叢集的方法都支援傳輸中加密功能：AWS Management Console、AWS CLI、所有開發套件和 AWS CloudFormation。

無法在現有 DAX 叢集上啟用傳輸中加密功能。若要在現有 DAX 應用程式中啟用傳輸中加密功能，請建立已啟用傳輸中加密功能的新叢集、將應用程式的流量轉移至該叢集，然後刪除舊叢集。

## 使用 DAX 的服務連結 IAM 角色

Amazon DynamoDB Accelerator (DAX) 使用 AWS Identity and Access Management(IAM) [服務連結角色](#)。服務連結角色是直接連結至 DAX 的一種特殊 IAM 角色類型。服務連結角色由 DAX 預先定義，並包含該服務代您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓設定 DAX 更為簡單，因為您不必手動新增必要的許可。DAX 定義其服務連結角色的許可，除非另有定義，否則僅有 DAX 可以擔任其角色。已定義的許可包括信任政策和許可政策。該許可政策無法連接至其他任何 IAM 實體。

您必須先刪除角色的相關資源，才能刪除角色。如此可保護您的 DAX 資源，避免您不小心移除資源的存取許可。

如需支援服務連結角色之其他服務的資訊，請參閱《IAM 使用者指南》中的[與 IAM 搭配運作的 AWS 服務](#)。尋找 Service-linked roles (服務連結角色) 欄中包含 Yes (是) 的服務。選擇 Yes (是) 連結，檢視該服務的服務連結角色文件。

### 主題

- [DAX 的服務連結角色許可](#)
- [建立 DAX 的服務連結角色](#)
- [編輯 DAX 的服務連結角色](#)
- [刪除 DAX 的服務連結角色](#)

## DAX 的服務連結角色許可

DAX 使用名為 `AWSServiceRoleForDAX` 的服務連結角色。此角色可讓 DAX 代表您的 DAX 叢集呼叫服務。

### Important

`AWSServiceRoleForDAX` 服務連結角色可讓設定和維護 DAX 叢集更為容易。但是，您仍然必須授予每個叢集存取 DynamoDB 的權限，才能使用它。如需詳細資訊，請參閱 [DAX 存取控制](#)。

`AWSServiceRoleForDAX` 服務連結角色信任下列服務擔任角色：

- `dax.amazonaws.com`

此角色許可政策允許 DAX 對指定資源完成下列動作：

- 在 ec2 上的動作：
  - `AuthorizeSecurityGroupIngress`
  - `CreateNetworkInterface`
  - `CreateSecurityGroup`
  - `DeleteNetworkInterface`
  - `DeleteSecurityGroup`
  - `DescribeAvailabilityZones`
  - `DescribeNetworkInterfaces`
  - `DescribeSecurityGroups`
  - `DescribeSubnets`
  - `DescribeVpcs`
  - `ModifyNetworkInterfaceAttribute`
  - `RevokeSecurityGroupIngress`

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的[服務連結角色許可](#)。

允許 IAM 實體建立 `AWSServiceRoleForDAX` 服務連結角色

新增以下政策陳述式到該 IAM 實體的許可中。

```
{
 "Effect": "Allow",
 "Action": [
 "iam:CreateServiceLinkedRole"
],
 "Resource": "*",
 "Condition": {"StringLike": {"iam:AWSServiceName": "dax.amazonaws.com"}}
}
```

## 建立 DAX 的服務連結角色

您不需要手動建立一個服務連結角色。當您建立叢集時，DAX 會為您建立服務連結角色。

### Important

若您是在 2018 年 2 月 28 日之前使用 DAX 服務，當其開始支援服務連結角色時，DAX 會在您的帳戶中建立 AWSServiceRoleForDAX 角色。如需詳細資訊，請參閱《IAM 使用者指南》中的[在我的 AWS 帳戶中出現的新角色](#)。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您建立執行個體或叢集時，DAX 會再次為您建立服務連結角色。

## 編輯 DAX 的服務連結角色

DAX 不允許您編輯 AWSServiceRoleForDAX 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的[編輯服務連結角色](#)。

## 刪除 DAX 的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。不過您必須先刪除您的所有 DAX 叢集，才能刪除服務連結角色。

## 清除服務連結角色

您必須先確認服務連結角色沒有作用中的工作階段，並移除該角色使用的資源，之後才能使用 IAM 將其刪除。

檢查服務連結角色是否於 IAM 主控台有作用中的工作階段

1. 登入 AWS Management Console，並開啟位於 <https://console.aws.amazon.com/iam/> 的 IAM 主控台。
2. 在 IAM 主控台的導覽窗格中，選擇 Roles (角色)。然後選擇 AWSServiceRoleForDAX 角色的名稱 (而非核取方塊)。
3. 在所選角色的 Summary (摘要) 頁面中，選擇 Access Advisor (存取 Advisor) 分頁。
4. 在 Access Advisor (存取 Advisor) 分頁中，檢閱服務連結角色的近期活動。

**Note**

如果您不確定 DAX 是否正在使用 `AWSServiceRoleForDAX` 角色，可嘗試刪除該角色。若服務正在使用該角色，則刪除會失敗，而您可以檢視角色使用於哪個區域。若該角色正受到使用，則您必須刪除您的 DAX 叢集，之後才能刪除該角色。您無法撤銷服務連結角色的工作階段。

如果您想要移除 `AWSServiceRoleForDAX` 角色，則必須先刪除所有 DAX 叢集。

### 刪除您的所有 DAX 叢集

使用這些步驟之一刪除您的每個 DAX 叢集。

#### 刪除 DAX 叢集 (主控台)

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 在導覽窗格中，選擇 DAX 下方的 Clusters (叢集)。
3. 選擇 Actions (動作)，然後選擇 Delete (刪除)。
4. 在 Delete cluster confirmation (刪除叢集確認) 方塊中，選擇 Delete (刪除)。

#### 刪除 DAX 叢集 (AWS CLI)

請參閱 AWS CLI 命令參考中的 [delete-cluster](#)。

#### 刪除 DAX 叢集 (API)

請參閱《Amazon DynamoDB API 參考》中的 [DeleteCluster](#)。

### 刪除服務連結角色

#### 使用 IAM 手動刪除服務連結角色

使用 IAM 主控台、IAM CLI 或 IAM API 刪除 `AWSServiceRoleForDAX` 服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的 [刪除服務連結角色](#)。

# 跨多 AWS 帳戶存取 DAX

想像您有一個 DynamoDB Accelerator (DAX) 叢集在一個 AWS 帳戶 (帳戶 A) 中執行，但您需要從另一個 AWS 帳戶 (帳戶 B) 中的 Amazon Elastic Compute Cloud (Amazon EC2) 執行個體存取 DAX 叢集。在本教學課程中，您使用來自帳戶 B 的 IAM 角色在帳戶 B 中啟動 EC2 執行個體，然後使用 EC2 執行個體的臨時安全登入資料來擔任帳戶 A 的 IAM 角色。最後，您使用在帳戶 A 中擔任 IAM 角色的臨時安全登入資料，透過 Amazon VPC 對等連線對帳戶 A 中的 DAX 叢集進行應用程式呼叫。為了執行這些工作，您在兩個 AWS 帳戶中都需要具備系統管理存取權。

## Important

DAX 叢集無法從其他帳戶存取 DynamoDB 資料表。

## 主題

- [設定 IAM](#)
- [設定 VPC](#)
- [修改 DAX 用戶端以允許跨帳戶存取權](#)

## 設定 IAM

1. 使用下列內容建立名為 `AssumeDaxRoleTrust.json` 的文字檔案，這可讓 Amazon EC2 代表您工作。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "ec2.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
```

2. 在帳戶 B 中，建立 Amazon EC2 可在啟動執行個體時使用的角色。

```
aws iam create-role \
 --role-name AssumeDaxRole \
 --assume-role-policy-document file://AssumeDaxRoleTrust.json
```

3. 使用下列內容建立名為 AssumeDaxRolePolicy.json 的文字檔案，這會允許在帳戶 B 中的 EC2 執行個體上執行的程式碼擔任帳戶 A 中的 IAM 角色。將 *accountA* 取代為帳戶 A 的實際 ID。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "sts:AssumeRole",
 "Resource": "arn:aws:iam::accountA:role/DaxCrossAccountRole"
 }
]
}
```

4. 將該政策新增至您剛建立的角色。

```
aws iam put-role-policy \
 --role-name AssumeDaxRole \
 --policy-name AssumeDaxRolePolicy \
 --policy-document file://AssumeDaxRolePolicy.json
```

5. 建立執行個體描述檔以允許執行個體使用該角色。

```
aws iam create-instance-profile \
 --instance-profile-name AssumeDaxInstanceProfile
```

6. 將角色與執行個體描述檔建立關聯。

```
aws iam add-role-to-instance-profile \
 --instance-profile-name AssumeDaxInstanceProfile \
 --role-name AssumeDaxRole
```

7. 使用下列內容建立名為 DaxCrossAccountRoleTrust.json 的文字檔案，這可讓帳戶 B 擔任帳戶 A 角色。將 *##B* 取代為帳戶 B 的實際 ID。

```
{
```



```
"Version": "2012-10-17",
"Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "AWS": "arn:aws:iam::accountB:role/AssumeDaxRole"
 },
 "Action": "sts:AssumeRole"
 }
]
```

8. 在帳戶 A 中，建立帳戶 B 可以擔任的角色。

```
aws iam create-role \
 --role-name DaxCrossAccountRole \
 --assume-role-policy-document file://DaxCrossAccountRoleTrust.json
```

9. 建立一個名為 `DaxCrossAccountPolicy.json` 的文字檔案以允許存取 DAX 叢集。將 *dax-cluster-arn* 取代為 DAX 叢集的正确 Amazon Resource Name (ARN)。

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

10. 在帳戶 A 中，將政策新增至角色。

```
aws iam put-role-policy \
 --role-name DaxCrossAccountRole \
 --policy-name DaxCrossAccountPolicy \
 --policy-document file://DaxCrossAccountPolicy.json
```

## 設定 VPC

1. 尋找帳戶 A DAX 叢集的子網路群組。將 *cluster-name* 取代為帳戶 B 必須存取的 DAX 叢集名稱。

```
aws dax describe-clusters \
 --cluster-name cluster-name \
 --query 'Clusters[0].SubnetGroup'
```

2. 使用該#####尋找該叢集的 VPC。

```
aws dax describe-subnet-groups \
 --subnet-group-name subnet-group \
 --query 'SubnetGroups[0].VpcId'
```

3. 使用該 *vpc-id* 尋找該 VPC 的 CIDR。

```
aws ec2 describe-vpcs \
 --vpc vpc-id \
 --query 'Vpcs[0].CidrBlock'
```

4. 從帳戶 B，使用不同於上一個步驟中找到的 CIDR 的非重疊 CIDR 來建立 VPC。然後，建立至少一個子網路。您可以使用 AWS Management Console 中的 [VPC 建立協助程式](#) 或 [AWS CLI](#)。
5. 從帳戶 B，請求與帳戶 A VPC 的對等連線，如[建立和接受 VPC 對等互連](#)中所述。從帳戶 A 接受連線。
6. 從帳戶 B 尋找新 VPC 的路由表。將 *vpc-id* 取代為您帳戶 B 中建立的 VPC ID。

```
aws ec2 describe-route-tables \
 --filters 'Name=vpc-id,Values=vpc-id' \
 --query 'RouteTables[0].RouteTableId'
```

7. 新增路由，將目的地為帳戶 A CIDR 的流量傳送至 VPC 對等連線。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
aws ec2 create-route \
 --route-table-id accountB-route-table-id \
 --destination-cidr accountA-vpc-cidr \
 --vpc-peering-connection-id peering-connection-id
```

8. 使用您之前找到的 *vpc-id*，從帳戶 A 尋找 DAX 叢集的路由表。

```
aws ec2 describe-route-tables \
 --filters 'Name=vpc-id, Values=accountA-vpc-id' \
 --query 'RouteTables[0].RouteTableId'
```

9. 從帳戶 A，新增路由，將目的地為帳戶 B CIDR 的流量傳送至 VPC 對等連線。使用您帳戶的正確值取代每個 *user input placeholder*。

```
aws ec2 create-route \
 --route-table-id accountA-route-table-id \
 --destination-cidr accountB-vpc-cidr \
 --vpc-peering-connection-id peering-connection-id
```

10. 從帳戶 B，在先前建立的 VPC 中啟動 EC2 執行個體。將其命名為 `AssumeDaxInstanceProfile`。您可以使用 AWS Management Console 中的 [啟動協助程式](#) 或 [AWS CLI](#)。記下執行個體的安全群組。
11. 從帳戶 A 尋找 DAX 叢集所使用的安全群組。記得將 *cluster-name* 取代為您的 DAX 叢集名稱。

```
aws dax describe-clusters \
 --cluster-name cluster-name \
 --query 'Clusters[0].SecurityGroups[0].SecurityGroupIdentifier'
```

12. 更新 DAX 叢集的安全群組，以允許來自您在帳戶 B 中建立之 EC2 執行個體的安全群組的傳入流量。請記得使用正確的帳戶值取代 *user input placeholders*。

```
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
```

此時，帳戶 B 的 EC2 執行個體上的應用程式可以使用執行個體描述檔來擔任 `arn:aws:iam::accountA-id:role/DaxCrossAccountRole` 角色並使用 DAX 叢集。

## 修改 DAX 用戶端以允許跨帳戶存取權

### Note

AWS Security Token Service (AWS STS) 登入資料是臨時登入資料。某些用戶端會自動處理重新整理，而有些則需要額外的邏輯來重新整理登入資料。我們建議您遵循適當說明文件的指引。

### Java

本節可協助您修改現有的 DAX 用戶端程式碼，以允許跨帳戶 DAX 存取。如果您還沒有 DAX 用戶端程式碼，可以在 [Java 與 DAX](#) 教學課程中找到可行的程式碼範例。

1. 新增以下匯入項目。

```
import com.amazonaws.auth.STSAssumeRoleSessionCredentialsProvider;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import
 com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
```

2. 從 AWS STS 取得登入資料提供者並建立 DAX 用戶端物件。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
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

本節可協助您修改現有的 DAX 用戶端程式碼，以允許跨帳戶 DAX 存取。如果您還沒有 DAX 用戶端程式碼，可以在 [.NET 和 DAX](#) 教學課程中找到可行的程式碼範例。

1. 將 [AWSSDK.SecurityToken](#) NuGet 套件新增到解決方案。

```
<PackageReference Include="AWSSDK.SecurityToken" Version="latest version" />
```

2. 使用 SecurityToken 和 SecurityToken.Model 套件。

```
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
```

3. 從 AmazonSimpleTokenService 取得臨時登入資料並建立 ClusterDaxClient 物件。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
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

本節可協助您修改現有的 DAX 用戶端程式碼，以允許跨帳戶 DAX 存取。如果您還沒有 DAX 用戶端程式碼，可以在 [GitHub 上找到可行的程式碼範例](#)。

1. 匯入 AWS STS 和工作階段套件。

```
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. 從 `AmazonSimpleTokenService` 取得臨時登入資料並建立 DAX 用戶端物件。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
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

本節可協助您修改現有的 DAX 用戶端程式碼，以允許跨帳戶 DAX 存取。如果您還沒有 DAX 用戶端程式碼，可以在 [Python 和 DAX 教學課程](#) 中找到可行的程式碼範例。

## 1. 匯入 boto3。

```
import boto3
```

2. 從 sts 取得臨時登入資料並建立 AmazonDaxClient 物件。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
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

本節可協助您修改現有的 DAX 用戶端程式碼，以允許跨帳戶 DAX 存取。如果您還沒有 DAX 用戶端程式碼，可以在 [Node.js](#) 和 [DAX](#) 教學課程中找到可行的程式碼範例。請記得將每個 *user input placeholder* 取代為您帳戶的正確值。

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');
const region = 'region';
const endpoints = [daxEndpoint1, ...];

const getCredentials = async() => {
 return new Promise((resolve, reject) => {
 const sts = new AWS.STS();
 const roleParams = {
 RoleArn: 'arn:aws:iam::accountA:role/RoleName',
 RoleSessionName: 'tryDax',
 };
 sts.assumeRole(roleParams, (err, session) => {
 if(err) {
 reject(err);
 } else {
```

```
 resolve({
 accessKeyId: session.Credentials.AccessKeyId,
 secretAccessKey: session.Credentials.SecretAccessKey,
 sessionToken: session.Credentials.SessionToken,
 });
 }
});
});
};

const createDaxClient = async() => {
 const credentials = await getCredentials();
 const daxClient = new AmazonDaxClient({endpoints: endpoints, region: region,
 accessKeyId: credentials.accessKeyId, secretAccessKey: credentials.secretAccessKey,
 sessionToken: credentials.sessionToken});
 return new AWS.DynamoDB.DocumentClient({service: daxClient});
};

createDaxClient().then((client) => {
 client.get(...);
 ...
}).catch((error) => {
 console.log('Caught an error: ' + error);
});
```

## DAX 叢集調整大小指南

本指南會為您提供建議，以便您為應用程式選擇合適的 Amazon DynamoDB Accelerator (DAX) 叢集大小和節點類型。這些指示會逐步引導您預估應用程式的 DAX 流量、選取叢集組態並加以測試。

如果您目前有 DAX 叢集，並且想評估叢集是否具有合適的節點數量與大小，請參考[擴展 DAX 叢集](#)。

### 主題

- [概觀](#)
- [預估流量](#)
- [負載測試](#)



## 概觀

不論您是要建立新的叢集或維護現有叢集，為工作負載適當調整 DAX 叢集都很重要。隨著時間經過與應用程式工作負載變更，您也應該定期審視擴展決策來確保合適性。

此程序通常遵循下列步驟：

1. 預估流量。在此步驟中，您將預測應用程式傳送至 DAX 的流量、流量的性質 (讀取比對寫入操作)，以及預計的快取命中率。
2. 負載測試。在此步驟中，您將建立叢集並向叢集傳送流量，來驗證上一個步驟的預估。請重複這個步驟直到找到合適的叢集組態為止。
3. 生產監控。當應用程式在生產階段使用 DAX 時，務必[監控叢集](#)以持續確認叢集仍會隨著工作負載變更正確調整。

## 預估流量

典型 DAX 工作負載的特性有三個主要因素：

- 快取命中率
- 每秒[讀取容量單位](#) (RCU)
- 每秒[寫入容量單位](#) (WCUs)

## 預估快取命中率

如果您已有 DAX 叢集，則可以使用 ItemCacheHits 和 ItemCacheMisses [Amazon CloudWatch 指標](#) 來判斷快取命中率。快取命中率等於  $\text{ItemCacheHits} / (\text{ItemCacheHits} + \text{ItemCacheMisses})$ 。如果您的工作負載包含 Query 或 Scan 操作，也請查看 QueryCacheHits、QueryCacheMisses、ScanCacheHits 與 ScanCacheMisses 指標。應用程式間的快取命中率會有所差異，而且非常容易受到叢集的存留時間 (TTL) 設定影響。使用 DAX 的應用程式通常有 85% 至 95% 的命中率。

## 預估讀取和寫入容量單位

[如果您的應用程式已有 DynamoDB 表格，請查看 ConsumedReadCapacityUnits 和 ConsumedWriteCapacityUnits CloudWatch 指標。](#) 使用 Sum 統計數字，除以週期中的秒數。

如果您也已經有 DAX 叢集，請記得此 DynamoDB ConsumedReadCapacityUnits 指標只會考量快取未中。因此，如果要了解 DAX 叢集每秒處理的讀取容量單位，請將數字除以快取未中率 (即 1：快取命中率)。

如果您還沒有 DynamoDB 表格，請參閱有關[讀取和寫入容量單位](#)的文件，以根據應用程式的預估請求率、每個請求存取的项目以及项目大小來估算流量。

預估流量時，請將後續增長和預期/非預期的尖峰納入考量，來確保叢集有足夠的流量增加空間。

## 負載測試

預估流量後的下一步是測試負載叢集組態。

1. 進行第一次負載測試時，建議您先從 `dax.r4.large` 節點類型開始，這是最低成本固定效能、記憶體最佳化的節點類型。
2. 容錯叢集至少需要三個節點，分布於三個可用區域。在此情況下，如果有一個可用區域失效，則可用區域的有效數量會減少三分之一。進行第一次負載測試時，建議您先從兩個節點的叢集開始，這會模擬三個節點叢集中有一個可用區域失效的情形。
3. 負載測試期間，請對您的測試叢集推動持續流量 (如上一步驟中所預估)。
4. 在負載測試期間監控叢集的效能。

理想情況下，您在負載測試期間推動的流量概況，應該盡可能地符合應用程式實際流量。這包括操作的分配 (例如 70%GetItem、25%Query 和 5%PutItem)、各操作的請求率、每個請求的项目存取數和项目大小的分佈。如果要達到與應用程式預期相當的快取命中率，請密切注意測試流量中的金鑰分佈情形。

### Note

對 T2 節點類型 (`dax.t2.small` 與 `dax.t2.medium`) 進行負載測試時請小心。T2 節點類型提供[高載 CPU 效能](#)，會根據節點的 CPU 點數餘額隨著時間變化。在 T2 節點上執行的 DAX 叢集可能看似正常運作，但只要有任何節點暴增超過本身執行個體的[基準效能](#)，節點就會消耗累積的 CPU 點數餘額。如果點數餘額不足，[效能會逐漸降低](#)至基準效能等級。

請在負載測試期間[監控您的 DAX 叢集](#)，來判斷為負載測試使用的節點類型是否合適。此外，也請在負載測試期間監控請求率與快取命中率，來確保測試基礎設施真的有在推動您想要的流量。

您應該留意所選叢集執行個體類型的網路位元組使用量。若超過 Amazon EC2 執行個體的可用基準頻寬，即表示您的叢集可能無法承受應用程式的工作負載，且需要進行擴展。

如果負載測試指出選取的叢集組態無法承受應用程式的工作負載，您應[切換至更大的節點類型](#)，尤其是在叢集的主節點上發現高 CPU 使用率、高移出率或高快取記憶體時更是如此。如果命中率一直很高，而且讀取與寫入流量的比率也很高的話，則建議您考慮[為叢集新增更多節點](#)。如果需要額外指導來了解何時該使用更大的節點類型 (垂直擴展) 或新增更多節點 (水平擴展)，則請參閱 [擴展 DAX 叢集](#)。

變更叢集組態後，請重複進行負載測試。

## 將 DAX 與 DynamoDB 搭配使用的最佳做法

當您將 DAX 與 DynamoDB 搭配使用時，建議您參考下列主題作為改善快取效能和可靠性的最佳實務。

- [DAX 叢集調整大小指南](#)
- [生產監控](#)

## DAX API 參考

如需 Amazon DynamoDB Accelerator (DAX) API 的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的 [Amazon DynamoDB Accelerator](#)。

# 適用於 DynamoDB 資料表的資料建模

在我們深入探討資料建模之前，請務必了解一些 DynamoDB 基礎知識。DynamoDB 是一個使用鍵值的 NoSQL 資料庫，擁有靈活的結構描述。除了每個項目的索引鍵屬性之外，資料屬性集可以統整在一起，也可以各自獨立。DynamoDB 索引鍵結構描述的形式可以是簡單主索引鍵，其中分割區索引鍵可唯一識別項目，也可以採用複合主鍵的形式，其中使用分割區索引鍵和排序索引鍵的組合來唯一定義項目。分割區索引鍵會經過雜湊處理，以判斷資料的實體位置並加以擷取。因此，請務必選擇高基數且可水平擴充的屬性作為分割區索引鍵，以確保資料均勻分佈。排序索引鍵屬性在索引鍵結構描述中是選用項目，而且有排序索引鍵就可讓您在 DynamoDB 中建立一對多關係的模型，以及建立項目集合。排序索引鍵也稱為範圍索引鍵，可用來排序項目集合中的項目，也能進行靈活的範圍型操作。

如需 DynamoDB 索引鍵結構描述的詳細資訊和最佳實務，可參考下列內容：

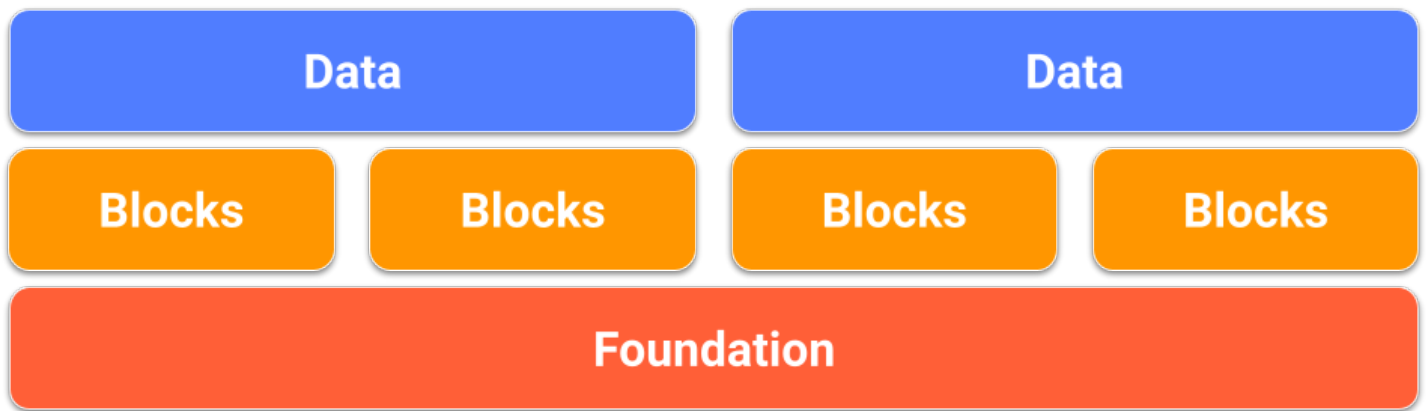
- [the section called “分割區與資料分佈”](#)
- [the section called “分割區索引鍵設計”](#)
- [the section called “排序索引鍵設計”](#)
- [選擇適合的 DynamoDB 分割區索引鍵](#)

通常需要有次要索引才能在 DynamoDB 中支援其他查詢模式。次要索引是影子資料表，其中相同的資料會透過與基底資料表不同的索引鍵結構描述來組織。本機次要索引 (LSI) 與基底資料表共用相同的分割區索引鍵，並允許使用替代的排序索引鍵來共用基底資料表的容量。全域次要索引 (GSI) 可以有與基底資料表不同的分割區索引鍵以及不同的排序索引鍵屬性，這表示 GSI 的輸送量管理與基底資料表無關。

如需次要索引和最佳實務的進一步詳細資訊，可參考下列內容：

- [the section called “使用索引”](#)
- [the section called “次要索引”](#)

現在讓我們來進一步了解資料建模。在 DynamoDB 或任何類似 NoSQL 資料庫上設計靈活且高度最佳化結構描述的程序，可能是一項要學習的極具挑戰性技能。本單元的目標是協助您發展心智圖，以設計讓您從使用案例進入生產環境的結構描述。我們將首先介紹任何設計 (單一資料表與多資料表設計) 的基礎選擇。接著，我們會檢閱多種設計模式 (建置區塊)，這些模式可供您的應用程式用來實現各種組織性或效能結果。最後，我們會介紹適用於不同使用案例和產業的各種完整結構描述設計套件。

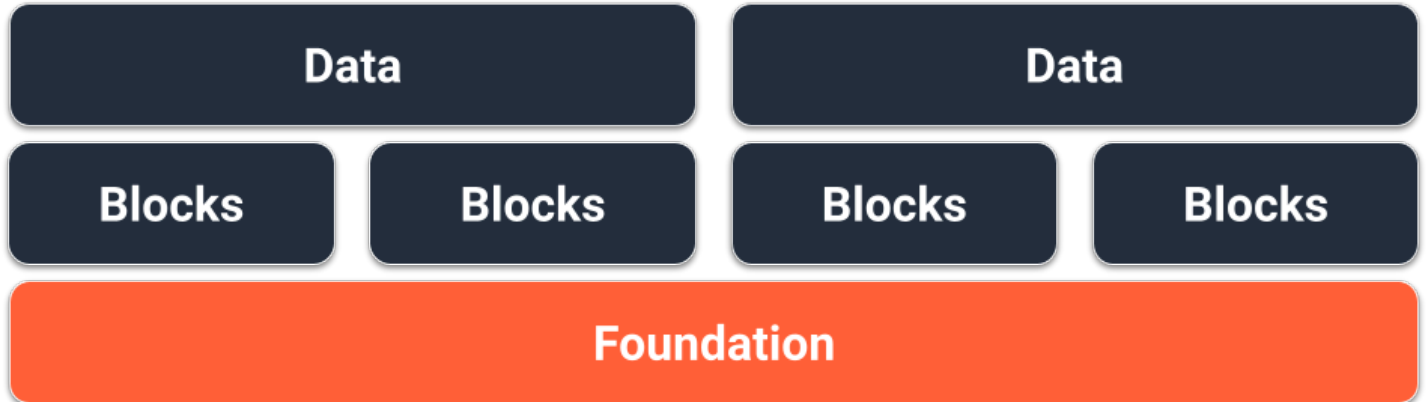


## 主題

- [DynamoDB 中的資料建模基礎](#)
- [DynamoDB 中的資料建模建置區塊](#)
- [DynamoDB 中的資料建模結構描述設計套件](#)

## DynamoDB 中的資料建模基礎

本節藉由檢驗兩種類型的資料表設計來介紹基礎層：單一資料表和多資料表。



## 單一資料表設計基礎

在我們 DynamoDB 結構描述的基礎上，其中一個選擇是單一資料表設計。單一資料表設計是一種模式，可讓您將多種資料類型 (實體) 儲存在單一 DynamoDB 資料表中。它旨在透過消除維護多個資料表和它們之間複雜關係的需要，來最佳化資料存取模式，提高效能並降低成本。這是有可能的，因為 DynamoDB 會將具有相同分割區索引鍵 (稱為項目集合) 的項目儲存在彼此相同的分割區上。在此設計中，不同類型的資料會以項目的形式儲存在同一個資料表中，並且每個項目由一個唯一的排序索引鍵來識別。

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

## 優點

- 資料位置性，可在單一資料庫呼叫中支援多個實體類型的查詢
- 降低讀取的整體財務和延遲成本：
  - 對總計小於 4KB 的兩個項目的單一查詢是 0.5 RCU 最終一致
  - 對總計小於 4KB 的兩個項目的兩個查詢是 1 RCU 最終一致 (每個 0.5 RCU)
  - 返回兩個單獨的數據庫調用的時間將平均高於單個調用
- 減少要管理的資料表數量：
  - 不需要跨多個 IAM 角色或政策維護許可
  - 資料表的容量管理會在所有實體中進行平均，通常會產生更可預測的取用模式
  - 監控需要更少的警示
  - 客戶管理的加密金鑰只需在一個資料表上進行輪換
- 資料表的平滑流量：
  - 透過將多種使用模式彙總到同一個資料表，整體使用量往往更平滑 (股票指數的績效往往比任何個股表現更平滑)，這對於透過佈建模式資料表達到更高的使用率有更好的效果

## 缺點

- 與關聯式資料庫相比，由於矛盾的設計，學習曲線可能會很陡峭
- 所有實體類型的資料需求必須一致
  - 備份是全部或沒有，因此，如果某些數據不是關鍵任務，請考慮將其保存在單獨的表格中
  - 資料表加密會在所有項目之間共用。對於具有個別租戶加密需求的多租戶應用程式，將需要用戶端加密
  - 混合了歷史資料和操作資料的資料表，不會因啟用不常存取的儲存類別而獲得更多優勢。如需更多資訊，請參閱 [資料表類別](#)
- 即使只需要處理一部分實體，所有變更的資料都會傳播至 DynamoDB Streams 流。
  - 由於有 Lambda 事件篩選條件，這在使用 Lambda 時不會影響您的帳單，但在使用 Kinesis Consumer Library 時會增加成本
- 使用 GraphQL 時，單一資料表設計將更加難以實現
- 如果您使用更高層級的 SDK 客戶端 (如 Java 的 [DynamoDBMapper](#) 或 [增強型用戶端](#))，那處理結果會比較困難，因為同一回應中的項目可能與不同的類別有關

## 使用情況

除非您的使用案例會受到上述其中一個缺點的嚴重影響，否則單一資料表設計是 DynamoDB 的建議設計模式。對於大多數客戶而言，長期利益會超過以這種方式設計資料表的短期挑戰。

## 多資料表設計基礎

在我們 DynamoDB 結構描述的基礎上，第二個選擇是多資料表設計。多資料表設計是一種模式，更像是傳統資料庫設計，您可以在每個 DynamoDB 資料表中儲存單一類型 (實體) 資料。每個資料表中的資料仍會依據分割索引鍵進行組織，因此單一實體類型內的效能會針對可擴展性和效能進行最佳化，但是跨多資料表的查詢必須獨立完成。

### 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

## 優點

- 對於那些不習慣使用單一資料表設計的人來說，該設計更簡單
- 由於每個解析程式會對應到單一實體 (資料表)，因此 GraphQL 解析程式的實現更容易
- 允許跨不同實體類型的唯一資料需求：
  - 可為關鍵任務的單一資料表進行備份
  - 可以每個資料表進行管理的資料表加密。對於具有個別租戶加密需求的多租戶應用程式，不同的租戶資料表可讓每個客戶擁有自己的加密金鑰
  - 不常存取的儲存類別只能在具有歷史資料的資料表上啟用，以實現完整的成本節省優勢。如需更多資訊，請參閱 [資料表類別](#)
- 每個資料表都有自己的變更資料串流，允許針對每種類型的項目 (而非單一整合式處理器) 設計專用的 Lambda 函數

## 缺點



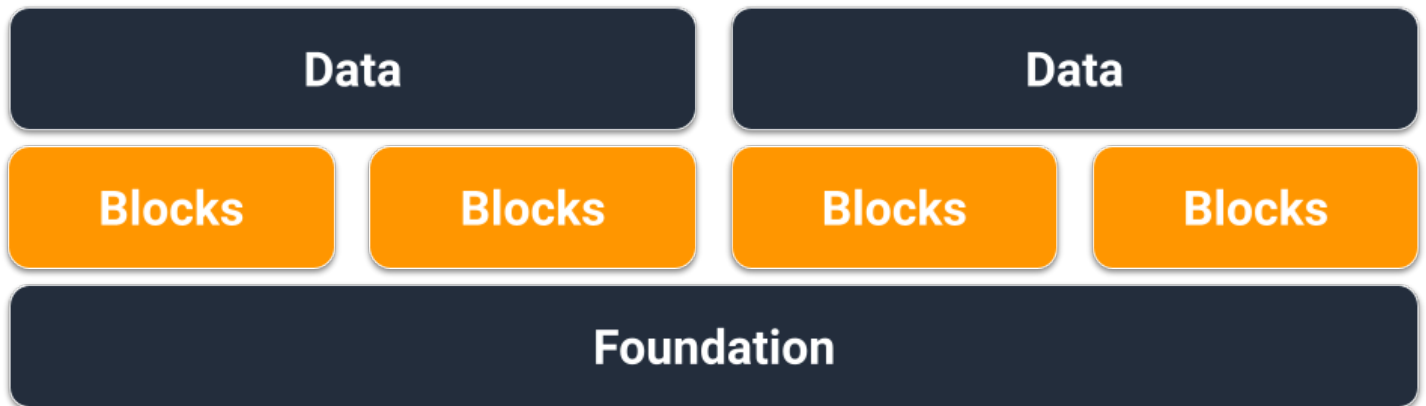
- 對於需要跨多資料表資料的存取模式，則需要從 DynamoDB 進行多次讀取，而且可能需要在用戶端程式碼上處理/加入資料。
- 多個表的操作和監視需要更多 CloudWatch 警報，並且每個表必須獨立調整
- 每個資料表的許可都需要單獨管理。未來新增資料表都將需要變更任何必要的 IAM 角色或政策

## 使用情況

如果您的應用程式的存取模式不需要一併查詢多個實體或資料表，那麼多資料表設計會是良好且足夠的方法。

## DynamoDB 中的資料建模建置區塊

本節介紹建置區塊層，為您提供了可在應用程式中使用的設計模式。



## 主題

- [複合排序索引鍵建置區塊](#)
- [多租戶建置區塊](#)
- [稀鬆索引建置區塊](#)
- [建置區塊生存時間](#)
- [封存建置區塊生存時間](#)
- [垂直分割建置區塊](#)
- [寫入碎片建置區塊](#)

## 複合排序索引鍵建置區塊

當人們思考 NoSQL 時，他們也可能認為它是非關聯式的。最終，DynamoDB 結構描述中無法放入關係是不合理的，它們只是看起來與關聯式資料庫及其外部索引鍵不同而已。其中一項可以用來在 DynamoDB 中開發資料邏輯階層的最關鍵模式，就是複合排序索引鍵。最常見的設計模式，是以井字號將階層中的每一層 (父層 > 子層 > 孫層) 隔開。例如 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

雖然 DynamoDB 中的分割區索引鍵一律需要確切的值才能查詢資料，但我們可以將部分條件從左至右套用至排序索引鍵，類似於遍歷二元樹。

在上面的範例中，我們有一間需要在使用者工作階段中維護購物車的電子商務商店。每當使用者登入時，他們可能希望看到整個購物車，包括儲存供日後使用的項目。但是，在他們進入結帳程序時，只應載入使用中購物車內的項目以供購買。由於這兩個 KeyConditions 都明確要求 CART 排序索引鍵，因此 DynamoDB 會在讀取時直接忽略額外的願望清單資料。雖然儲存的項目和使用中的項目都屬於同一個購物車，但我們需要在應用程式的不同部分中以不同的方式處理它們，因此在排序索引鍵的字首上套用 KeyCondition，是僅擷取應用程式每個部分所需資料的最佳方式。

### 此建置區塊的主要特徵

- 相關項目彼此在本機儲存，以便有效存取資料
- 使用 KeyCondition 表達式，可以選擇性擷取階層的子集，表示沒有浪費的 RCU
- 應用程式的不同部分可以將其項目儲存在特定字首下，防止被覆寫的項目或衝突的寫入

## 多租戶建置區塊

許多客戶使用 DynamoDB 來託管多租戶應用程式的資料。針對這些案例，我們想要設計結構描述，以便將單一租戶的所有資料保留在其本身的資料表邏輯分割區中。這會利用「項目集合」的概念，該術語是指 DynamoDB 資料表中具有相同分割區索引鍵的所有項目。如需 DynamoDB 如何處理多租戶的詳細資訊，請參閱 [Multitenancy on 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]

在此範例中，我們正在執行一個可能有成千上萬使用者的相片託管網站。每個使用者最初只會將相片上傳到自己的個人資料，但預設情況下，我們不允許使用者查看任何其他使用者的相片。理想情況下，應該為每個使用者呼叫 API 的授權新增額外隔離等級，以確保他們只從自己的分割請求資料，但在結構描述等級中，唯一分割區索引鍵就足夠了。

### 此建置區塊的主要特徵

- 任何一位使用者或租用戶讀取的資料量只能與其分割區中的項目總量相同
- 由於帳戶關閉或合規要求而需刪除租戶資料，可以巧妙而廉價的方式完成。只要執行分割區索引鍵等於其租戶 ID 的查詢，然後為傳回的每個主索引鍵執行 DeleteItem 操作

#### Note

以多租戶為設計理念，您可以在單一資料表中使用不同的加密金鑰提供者，以安全地隔離資料。[AWS適用於 Amazon DynamoDB 的資料庫加密 SDK](#) 可讓您在 DynamoDB 工作負載中納

入用戶端加密。您可以執行屬性層級加密，讓您在將特定屬性值儲存在 DynamoDB 資料表之前先加密，並搜尋加密的屬性，而無需事先解密整個資料庫。

## 稀疏索引建置區塊

有時，存取模式需要尋找符合罕見項目的項目，或是接收狀態的項目 (需要提升回應)。我們不會定期查詢這些項目的整個資料集，而是可以利用全域次要索引 (GSI) 稀疏的載入資料的事實。這表示只有基底資料表中具有在索引中定義之屬性項目，才會複寫至索引。

Primary key		Attributes		
Partition key: DeviceID	Sort key: State#Date			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:45:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:50:00	Operator	Date	
		Liz	2020-04-24	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	
		Liz	2020-04-11	
	NORMAL#2020-04-11T09:30:00	Operator	Date	
		Sue	2020-04-11	
	WARNING2#2020-04-11T09:25:00	Operator	Date	
		Sue	2020-04-11	
	WARNING3#2020-04-11T05:55:00	Operator	Date	
		Liz	2020-04-11	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	
		Sue	2020-04-27	
	WARNING4#2020-04-27T16:15:00	Operator	Date	EscalatedTo
		Sue	2020-04-27	Sara

Primary key		Attributes	
Partition key: EscalatedTo	Sort key: State#Date		
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator
		d#11223	Sue

在此範例中，我們可看到 IOT 使用案例，在欄位中的每個裝置都會定期回報狀態。對於大多數報告，我們期望裝置會報告一切正常，但有時可能會出現故障，必須提升給維修技術人員處理。對於具有升級的報告而言，系統會新增該項目的屬性 EscalatedTo，但不會顯示該屬性。本範例中的 GSI 由 EscalatedTo 分割，由於 GSI 會從基底資料表中引入金鑰，我們仍然可以看到哪個 DeviceID 報告了故障以及何時報告故障。

雖然在 DynamoDB 中的讀取成本比寫入低，但稀疏索引是非常強大的工具，適用於特定類型項目的執行個體罕見，但讀取以找到它們卻很常見的使用案例。

此建置區塊的主要特徵

- 稀疏 GSI 的寫入和儲存成本僅適用於符合金鑰模式的項目，因此 GSI 的成本可以大幅低於已複製所有項目的其他 GSI
- 複合排序索引鍵仍可用來進一步縮小符合所需查詢的項目範圍，例如，排序索引鍵可以使用時間戳記，以便只檢視最後 X 分鐘內報告的錯誤 (SK > 5 minutes ago, ScanIndexForward: False)

## 建置區塊生存時間

大多數資料都有一定的持續時間，可以視為值得保存在主要資料儲存中。為了協助在 DynamoDB 中的資料老化，它具有稱作存留時間 (TTL) 的功能。[TTL](#) 功能可讓您在資料表層級定義特定屬性，這些屬性需要監視具有紀元時間戳記的項目 (過去)。這使您可以免費從資料表中刪除過期記錄。

### Note

如果您使用的是[全域表 2019.11.21 \(目前\) 版本](#)的全域表，而且您也使用存留[時間功能](#)，DynamoDB 會將 TTL 刪除複寫到所有複本表格。初始 TTL 刪除不會在 TTL 過期發生時消耗區域中的寫入容量。但是，對複本資料表複寫的 TTL 刪除會消耗每個複本區域中的複寫寫入容量，並且將收取適當的費用。

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

在此範例中，我們有一個應用程式專門設計用來讓使用者建立短期訊息。在 DynamoDB 中建立訊息時，應用程式的程式碼會將 TTL 屬性設定為未來七天的日期。在大約七天內，DynamoDB 會看到這些項目的 epoch 時間戳記已經過去，並將其刪除。

由於 TTL 完成的刪除是免費的，因此強烈建議您使用此功能從資料表中移除歷史資料。這將減少每個月整體儲存體費用，並可能降低使用者讀取成本，因為他們查詢擷取的資料將較少。雖然 TTL 是在

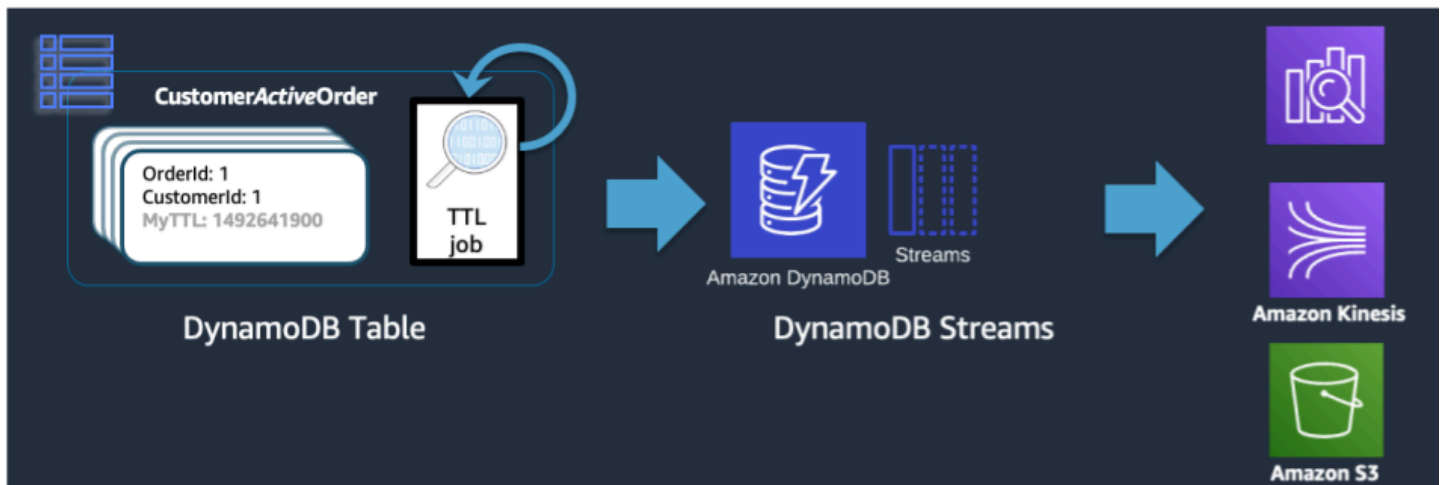
資料表層級啟用的，但要為哪些項目或實體建立 TTL 屬性，以及將 epoch 時間戳記設定為未來多久時間，將由您決定。

### 此建置區塊的主要特徵

- TTL 刪除會在幕後執行，不會影響您的資料表效能
- TTL 是一種非同步處理程序，大約每六個小時執行一次，但可能需要超過 48 小時才能刪除過期的記錄
  - 如果必須在 48 小時內清除過時的資料，請勿在鎖定記錄或狀態管理等使用案例中依賴 TTL 進行刪除
- 您可以將 TTL 屬性命名為有效的屬性名稱，但值必須是數字類型

## 封存建置區塊生存時間

雖然 TTL 是從 DynamoDB 刪除舊資料的有效工具，但許多使用案例都需要將資料封存的保存時間比主要資料儲存更長。在此情況下，我們可以利用 TTL 對記錄的定時刪除，將過期的記錄推送到長期資料儲存。



在 DynamoDB 完成 TTL 刪除時，它仍會作為 Delete 事件推送到 DynamoDB 串流中。在 DynamoDB TTL 是執行刪除的執行者時，principal:dynamodb 的串流記錄上會有一個屬性。使用 DynamoDB 串流的 Lambda 訂閱用戶，我們可以僅對 DynamoDB 主體屬性套用事件篩選條件，並知道與該篩選條件相符的任何記錄，都將被推送到 S3 Glacier 等封存儲存。

### 此建置區塊的主要特徵

- 一旦歷史項目不再需要 DynamoDB 的低延遲讀取，將它們遷移到像 S3 Glacier 這樣的較冷的儲存體服務可大幅降低儲存成本，同時滿足使用案例的資料合規需求

- 如果資料保留在 Amazon S3 中，則可以使用具成本效益的分析工具 (如 Amazon Athena 或 Redshift Spectrum) 來執行資料的歷史分析

## 垂直分割建置區塊

熟悉文件模型資料庫的使用者將熟悉在單一 JSON 文件中儲存所有相關資料的想法。雖然 DynamoDB 支援 JSON 資料類型，但它不支援在巢狀 JSON 上執行 KeyConditions。由於 KeyConditions 會決定要從磁碟讀取多少資料，以及查詢有效消耗了多少 RCU，因此可能會導致大規模的不足。為了更最佳化 DynamoDB 的寫入和讀取，我們建議您將文件的個別實體分割為個別 DynamoDB 項目，也稱為垂直分割。

```
{
 "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

如上所示，垂直分割是實際操作中單一資料表設計的關鍵範例，但如果需要，也可以跨多個資料表進行實作。由於 DynamoDB 帳單會以 1 KB 的增量進行寫入，因此理想情況下，您應該以產生小於 1 KB 項目的方式對文件進行分割。

### 此建置區塊的主要特徵

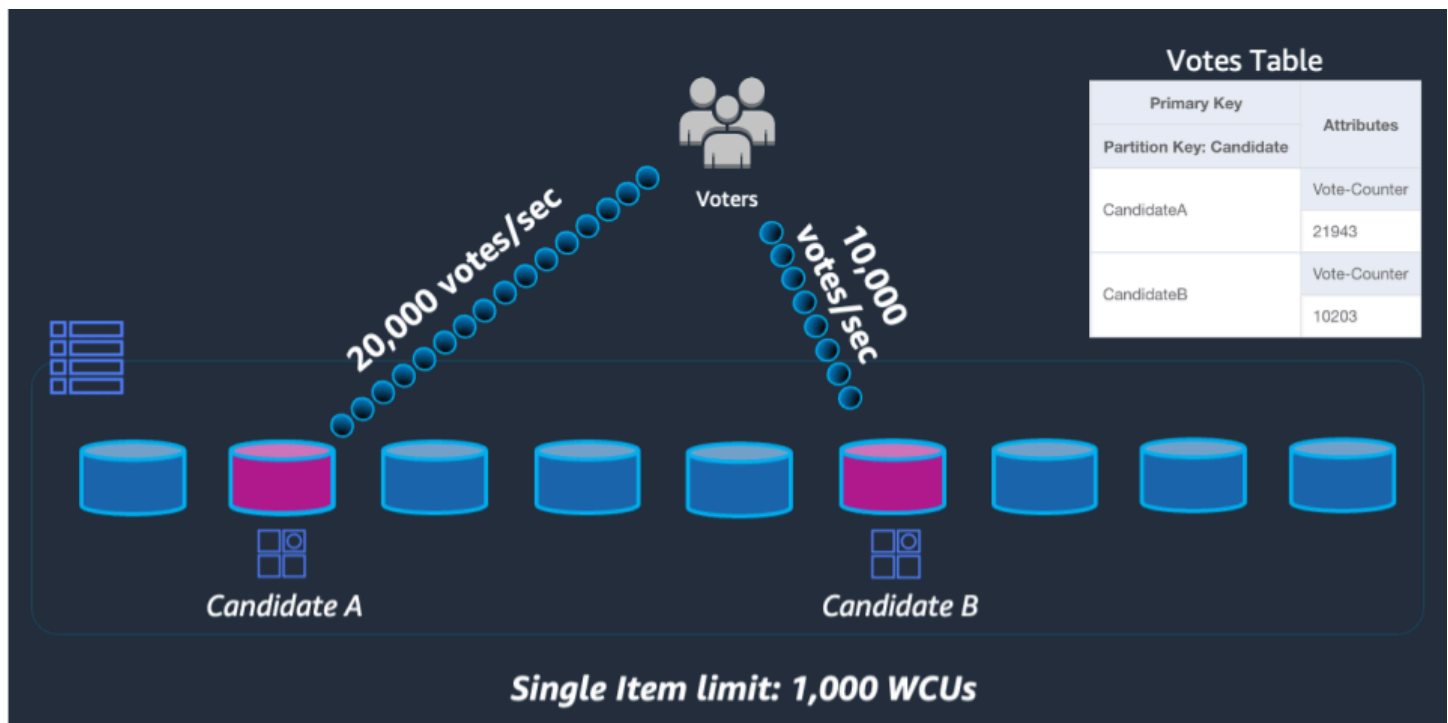
- 資料關係的階層是透過排序索引鍵字首維護的，因此如果需要，可以在用戶端重建單一文件結構
- 資料結構的單數元件可以獨立更新，導致小項目更新僅為 1 WCU
- 透過使用排序索引鍵 BeginsWith，應用程式可在單一查詢中擷取類似資料，彙總讀取成本以降低總成本/延遲
- 大型文件可輕易地超過 DynamoDB 中的 400 KB 個別項目大小限制，而垂直分割則有助於解決此限制

### 寫入碎片建置區塊

DynamoDB 的極少數硬性限制之一，就是單一實體分割區每秒可維持多少輸送量 (不一定是單一分割區索引鍵) 的限制。這些限制目前是：

- 1000 個 WCU (或每秒寫入 1000  $\leq$  1KB 的項目) 和 3000 RCU (或每秒讀取 3000  $\leq$  4KB) 強烈一致或
- 每秒 6000  $\leq$  4 KB 讀取最終一致

如果對資料表的要求超過上述任一限制，則會將錯誤傳回給 `ThroughputExceededException` 的用戶端 SDK，通常稱為限流。需要超過該限制的讀取操作使用案例，通常是將讀取快取放在 DynamoDB 前面，以達到最佳效果，但是寫入操作需要稱為寫入碎片的結構描述層級設計。



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

為了解決此問題，我們將在應用程式的 `UpdateItem` 代碼中的每個競爭者的分割區索引鍵結尾附加一個隨機整數。隨機整數產生器的範圍，將需要具有上限相符或超過指定競爭者每秒的寫入量除以

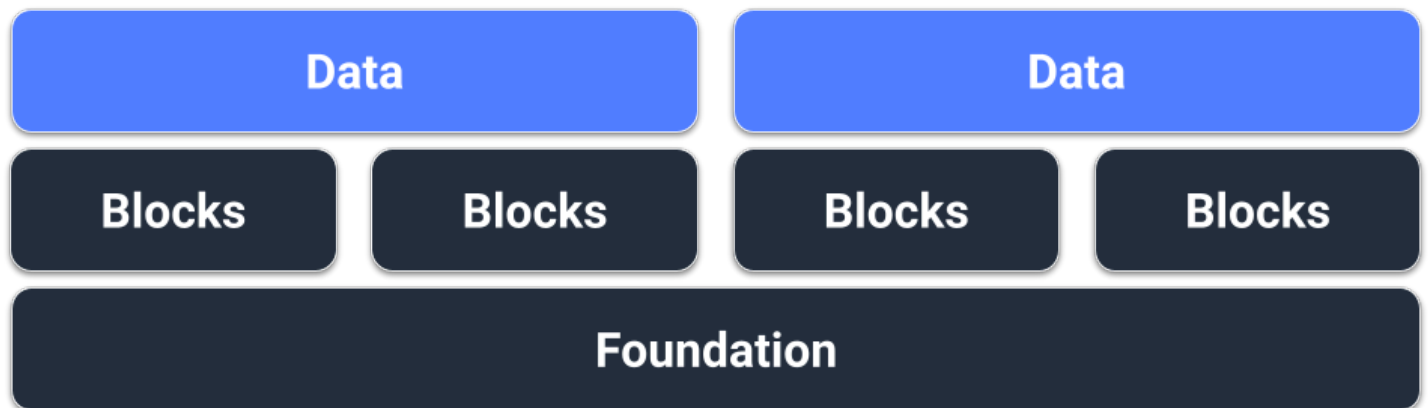
1000。為了支援每秒 20,000 次投票，它看起來會像 rand (0,19)。現在該資料儲存在單獨的邏輯分割區下，它必須在讀取時重新合併在一起。由於投票總數不需要是即時的，因此排定為每 X 分鐘讀取所有投票分割區的 Lambda 函數可能會偶爾為每位競爭者執行彙總，並將其寫回單一投票總記錄以供即時讀取。

### 此建置區塊的主要特徵

- 對於無法避免的指定分割區索引鍵具有極高寫入輸送量的使用案例，可以人為將寫入操作分散到多個 DynamoDB 分割區
- 具有低基數分割區索引鍵的 GSI 也應該使用此模式，因為 GSI 上的限流會套用背壓，以在基底資料表上進行寫入操作

## DynamoDB 中的資料建模結構描述設計套件

本節介紹資料層，用於詳細說明可在 DynamoDB 資料表設計中使用的不同範例。每一個範例將詳細說明其使用案例、存取模式、如何實現存取模式，以及最終結構描述的樣貌。



### 必要條件

在嘗試為 DynamoDB 設計結構描述之前，我們必須先收集有關結構描述需要支援的使用案例的一些先決條件資料。與關聯式資料庫不同，DynamoDB 預設會碎片化，這表示資料會在幕後存放在多部伺服器上，因此針對資料位置進行設計非常重要。我們需要為每個結構描述設計整理以下清單：

- 實體清單 (ER 圖)
- 每個實體的預估數量和輸送量
- 需要被支援的存取模式 (查詢和寫入)
- 資料保留要求

## 主題

- [DynamoDB 中的社交網路結構描述設計](#)
- [DynamoDB 中的遊戲設定檔結構描述設計](#)
- [DynamoDB 中的投訴管理系統結構描述設計](#)
- [DynamoDB 中的週期性付款結構描述設計](#)
- [監控 DynamoDB 中的裝置狀態更新](#)
- [使用 DynamoDB 做為線上商店的資料存放區](#)

## DynamoDB 中的社交網路結構描述設計

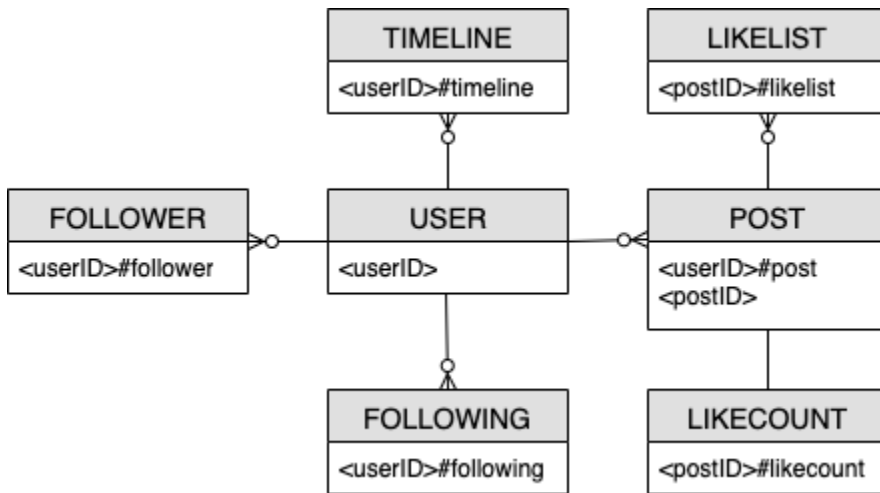
### 社交網路商業使用案例

此使用案例討論如何使用 DynamoDB 做為社交網路。社交網路是一種線上服務，可以讓不同的使用者與彼此互動。我們將設計的社交網路會讓使用者看到一個時間軸，其中包括他們的文章、他們的跟隨者、他們跟隨的人，以及他們跟隨的人撰寫的文章。此結構描述設計的存取模式為：

- 取得指定 userID 的使用者資訊
- 取得指定 userID 的跟隨者清單
- 取得指定 userID 的跟隨清單
- 取得指定 userID 的文章清單
- 取得指定 postID 中喜歡該篇文章的使用者清單
- 取得指定 postID 的喜歡計數
- 取得指定 userID 的時間軸

### 社交網路實體關係圖

這是我們將用於社交網路結構描述設計的實體關係圖 (ERD)。



## 社交網路存取模式

這些是我們針對社交網路結構描述設計考量的存取模式。

- getUserInfoByUserID
- getFollowerListByUserID
- getFollowingListByUserID
- getPostListByUserID
- getUserLikesByPostID
- getLikeCountByPostID
- getTimelineByUserID

## 社交網路結構描述設計演變

DynamoDB 是一種 NoSQL 資料庫，因此它不允許您執行聯結，這是一種結合多個資料庫資料的操作。不熟悉 DynamoDB 的客戶可能會在不需要時，將關聯式資料庫管理系統 (RDBMS) 設計理念 (例如為每個實體建立資料表) 套用至 DynamoDB。DynamoDB 的單一資料表設計之目的，是根據應用程式的存取模式，以預先聯結的形式寫入資料，然後立即使用資料，無需額外運算。如需詳細資訊，請參閱 [DynamoDB 中的單一資料表與多資料表設計](#)。

現在，讓我們逐步介紹如何發展結構描述設計以解決所有存取模式。

### 步驟 1：位址存取模式 1 (getUserInfoByUserID)

為了取得特定使用者的資訊，我們需要 [Query](#) 基底資料表，索引鍵條件為 `PK=<userID>`。查詢操作可讓您將結果分頁，這在使用者有許多跟隨者的情況下很實用。如需查詢的詳細資訊，請參閱 [DynamoDB 中的查詢操作](#)。

在我們的範例中，我們追蹤使用者的兩種類型資料：他們的「計數」和他們的「資訊」。使用者的「計數」反映他們有多少追隨者，他們追隨的使用者數量以及他們建立了多少文章。使用者的「資訊」反映了他們的個人資訊，例如他們的姓名。

我們看到這兩種資料由以下的兩個項目來表示。排序索引鍵 (SK) 中有「計數」的項目比含有「資訊」的項目更有可能變更。DynamoDB 會將項目的大小視為更新前後所顯示的大小，消耗的佈建輸送量將反映這些項目大小中較大的部分。因此即使您只更新一小部分項目的屬性，[UpdateItem](#) 還是會使用完整數量的佈建輸送量 (之前和之後項目大小中較大的一個)。您可以透過單一 [Query](#) 操作取得項目，並使用 [UpdateItem](#) 從現有的數字屬性中新增或減去。

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://...	...

### 步驟 2：位址存取模式 2 ([getFollowerListByUserID](#))

若要取得跟隨指定使用者的使用者清單，我們需要 [Query](#) 基底資料表，索引鍵條件為 `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				

### 步驟 3：位址存取模式 3 ([getFollowingListByUserID](#))

若要取得指定使用者跟隨的使用者清單，我們需要 [Query](#) 基底資料表，索引鍵條件為 `PK=<userID>#following`。然後，您可以將多個請求一起使用一個 [TransactWriteItems](#) 操作群組，並執行以下操作：

- 將使用者 A 新增到使用者 B 的跟隨者清單中，然後將使用者 B 的跟隨者計數增加一
- 將使用者 B 新增到使用者 A 的跟隨者清單中，然後將使用者 A 的跟隨者計數增加一

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				

#### 步驟 4：位址存取模式 4 (getPostListByUserID)

若要取得指定使用者建立的文章清單，我們需要 Query 基底資料表，索引鍵條件為 PK=<userID>#post。這裡需要注意的一件重要事情是，使用者的 postID 必須是增量的：第二個 postID 值必須大於第一個 postID 值 (因為使用者希望以排序方式查看自己的貼文)。您可以透過根據時間值 (例如通用唯一字典順序排序識別符 (ULID)) 產生 postID 來實現此目的。

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	

## 步驟 5：位址存取模式 5 (`getUserLikesByPostID`)

若要取得對指定使用者貼文按下喜歡的使用者清單，我們需要 Query 基底資料表，索引鍵條件為 `PK=<postID>#likelist`。這種方法是我們用於擷取跟隨者和跟隨清單中存取模式 2 (`getFollowerListByUserID`) 和存取模式 3 (`getFollowingListByUserID`) 相同的模式。

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				

## 步驟 6：位址存取模式 6 (`getLikeCountByPostID`)

要取得指定文章的喜歡計數，我們需要透過 `PK=<postID>#likecount` 的索引鍵條件在基底資料表上執行 [GetItem](#) 操作。每當有許多跟隨者 (例如名人) 的使用者建立文章時，此存取模式就會造成限流問題，因為當分割區的輸送量超過每秒 1000 WCU 時，就會發生限流問題。這個問題不是 DynamoDB 的結果，它只是出現在 DynamoDB 中，因為它位於軟體堆疊的末端。

您應該評估它是否對所有使用者同時查看類似計數真的很重要，或者是否可以隨著時間的推移逐漸發生。一般來說，文章的類似計數不需要立即 100% 準確。您可以在應用程式和 DynamoDB 之間放置佇列，以便定期進行更新以實作此策略。



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			

## 步驟 7：位址存取模式 7 (`getTimelineByUserID`)

如要取得指定使用者的時間軸，我們需要透過 `PK=<userID>#timeline` 的索引鍵條件在基底資料表上執行 Query 操作。讓我們考慮一個場景，即使用者的追隨者需要同步查看他們的文章。每次使用者寫一篇文章時，他們的跟隨者清單都會被讀取，並且他們的 `userID` 和 `postID` 會慢慢輸入到其所有跟隨者的時間軸索引鍵中。然後，當您的應用程式啟動時，您可以使用 Query 操作讀取時間軸索引鍵，並使用任何新項目的 [BatchGetItem](#) 操作來使用 `userID` 和 `postID` 的組合填充時間軸畫面。您無法使用 API 呼叫讀取時間軸，但是如果文章可以經常編輯，則這是一個更具成本效益的解決方案。

時間軸是一個顯示最近文章的地方，因此我們需要一種清理舊文章的方法。您可以使用 DynamoDB 的 [TTL](#) 功能免費執行此操作，而不是使用 WCU 刪除它們。

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			

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	Base 資料表/ GSI/LSI	操作	分割區索引鍵 值	排序索引鍵值	其他條件/篩 選條件
getUserInfoByUserID	BASE 資料表	Query	PK=<userID>		
getFollowerListByUserID	BASE 資料表	Query	PK=<userID>#follower		

存取模式	Base 資料表/ GSI/LSI	操作	分割區索引鍵 值	排序索引鍵值	其他條件/篩 選條件
getFollowingListByUserID	BASE 資料表	Query	PK=<userID>#following		
getPostListByUserID	BASE 資料表	Query	PK=<userID>#post		
getUserLikesByPostID	BASE 資料表	Query	PK=<postID>#likelist		
getLikeCountByPostID	BASE 資料表	GetItem	PK=<postID>#likecount		
getTimelineByUserID	BASE 資料表	Query	PK=<userID>#timeline		

## 社交網路最終結構描述

這是最終的結構描述設計。若要將此結構描述設計下載為 JSON 檔案，請參閱 GitHub 上的 [DynamoDB 範例](#)。

Base 資料表：

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			

## 使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 [NoSQL Workbench](#)，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視覺化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需更多詳細資訊，請參閱 [the section called “下載”](#)。
2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需更多詳細資訊，請參閱 [the section called “匯入現有的模型”](#)。
4. 一旦您匯入到 NoSQL Workbench 後，便可以編輯資料模型。如需更多詳細資訊，請參閱 [the section called “編輯現有的模型”](#)。

- 若要視覺化您的資料模型、新增範例資料，或從 CSV 檔案匯入範例資料，請使用 NoSQL Workbench 的[資料視覺化工具](#)功能。

## DynamoDB 中的遊戲設定檔結構描述設計

### 遊戲設定檔商業使用案例

本使用案例討論如何使用 DynamoDB 儲存遊戲系統的玩家設定檔。使用者 (在本案例中為玩家) 需要先建立設定檔，然後才能與許多現代遊戲 (尤其是線上遊戲) 進行互動。遊戲設定檔通常包括下列項目：

- 基本資訊，例如使用者名稱
- 遊戲資料，例如物品和裝備
- 遊戲記錄，例如任務和活動
- 朋友清單等社交資訊

為了滿足此應用程式的精細資料查詢存取要求，主索引鍵 (分割區索引鍵和排序索引鍵) 將使用通用名稱 (PK 和 SK)，因此它們可以用各種類型的值進行過載，如下所示。

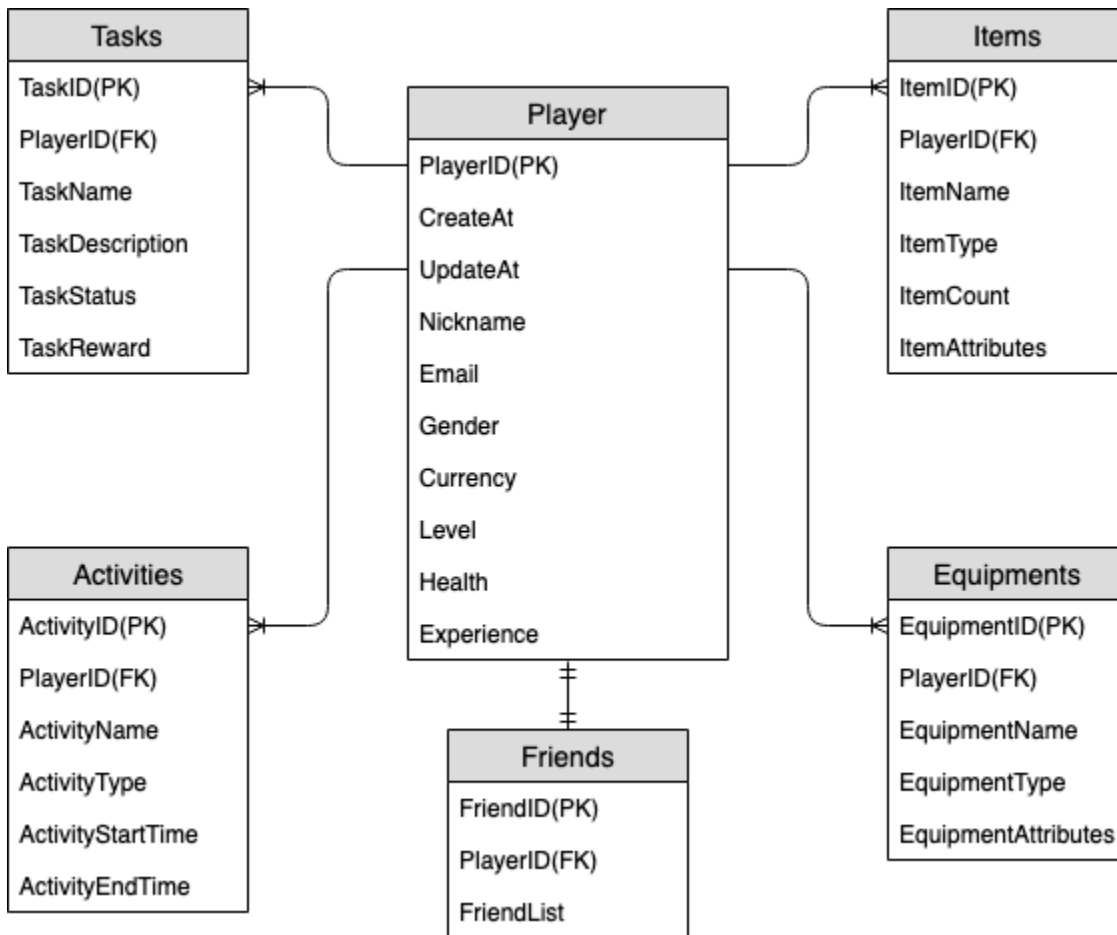
此結構描述設計的存取模式為：

- 取得使用者的好友清單
- 取得玩家的所有資訊
- 取得使用者的物品清單
- 從使用者的物品清單中取得特定項目
- 更新使用者的角色
- 更新使用者的物品計數

遊戲設定檔的大小在不同的遊戲中會有所不同。[壓縮大型屬性值](#)可以讓它們符合 DynamoDB 中的項目限制並減少成本。輸送量管理策略取決於各種因素，例如：玩家數量，每秒遊戲遊玩數量，以及工作負載的季節性。通常對於新推出的遊戲而言，玩家數量和受歡迎程度未知，因此我們將從[隨需輸送量模式](#)開始。

### 遊戲設定檔實體關係圖

這是我們將用於遊戲設定檔架構設計的實體關係圖 (ERD)。



## 遊戲設定檔存取模式

這些是我們針對社交網路結構描述設計考量的存取模式。

- getPlayerFriends
- getPlayerAllProfile
- getPlayerAllItems
- getPlayerSpecificItem
- updateCharacterAttributes
- updateItemCount

## 遊戲設定檔結構描述設計演進

從上面的 ERD，我們可以看到，這是一個 one-to-many 關係類型的數據建模。在 DynamoDB 中，one-to-many 資料模型可以組織成項目集合，這與傳統的關聯式資料庫不同，其中建立多個表格並透過

外部索引鍵連結。[項目集合](#)是一組項目，共用相同的分割區索引鍵值，但具有不同的排序索引鍵值。在項目集合中，每個項目都有唯一的排序索引鍵值，可將其與其他項目區分開來。考量這一點，讓我們對每種實體類型的 HASH 和 RANGE 值使用以下模式。

首先，我們使用通用名稱 (如 PK 和 SK) 將不同類型的實體儲存在同一個資料表中，以打造前瞻性的模型。為了更好的可讀性，我們可以包括字首來表示資料的類型或包含名為 Entity\_type 或 Type 的任意屬性。在目前範例中，我們使用以 player 開頭的字串將 player\_ID 儲存為 PK；使用 entity name# 作為 SK 的字首，並新增一個 Type 屬性來指示這筆資料是哪種實體類型。這使我們能夠支援將來儲存更多實體類型，並使用例如 GSI 超載和稀疏 GSI 等先進技術來滿足更多存取模式。

讓我們開始實作存取模式。新增玩家、新增裝備等存取模式，都可以透過 [PutItem](#) 操作來實現，所以我們可以忽略它們。在本文件中，我們將著重於上面列出的典型存取模式。

### 步驟 1：位址存取模式 1 (getPlayerFriends)

我們會透過此步驟解決存取模式 1 (getPlayerFriends)。在我們目前的設計中，朋友關係很簡單，遊戲中的朋友數量也很少。為簡單起見，我們使用清單資料類型來儲存朋友清單 (1:1 模型建置)。在這種設計中，我們使用 [GetItem](#) 來滿足這種存取模式。在 GetItem 操作中，我們明確提供了分割區索引鍵和排序索引鍵值以取得特定項目。

但是，如果一個遊戲有很多朋友，並且他們之間的關係很複雜 (例如友誼與邀請和接受組件是雙向的)，則必須使用一種 many-to-many 關係來單獨存儲每個朋友，以便擴展到無限的朋友列表大小。而且，如果友誼變更涉及同時對多個項目進行操作，則 DynamoDB 交易可用於將多個動作分組在一起，並將其提交為單一 all-or-nothing [TransactWriteItems](#) 或 [TransactGetItems](#) 作業。

Primary key		Attributes	
Partition key: PK	Sort key: SK	Type	FriendList
player001	FRIENDS#player001	Friends	<pre>{   "M": {     "FriendId": {       "S": "player002"     },     "FriendName": {       "S": "Alice"     }   },   "M": {     "FriendId": {       "S": "player003"     },     "FriendName": {       "S": "Bob"     }   } }</pre>

### 步驟 2：位址存取模式 2 (getPlayerAllProfile)、3 (getPlayerAllItems) 和 4 (getPlayerSpecificItem)

我們使用此步驟解決存取模式 2 (getPlayerAllProfile)、3 (getPlayerAllItems) 和 4 (getPlayerSpecificItem)。這三種存取模式的共同點是使用 [Query](#) 操作的範圍查詢。根據查詢的範圍，會使用 [索引鍵條件](#) 和 [篩選條件表達式](#)，這些表達式通常在實際開發中使用。

在查詢操作中，我們為分割區索引鍵提供單一值，並取得具有該分割區索引鍵值的所有項目。存取模式 2 (getPlayerAllProfile) 是以這種方式實現。或者，我們可以新增一個排序索引鍵條件表達式 - 確定要從資料表中讀取項目的字串。存取模式 3 (getPlayerAllItems) 是透過新增排序索引鍵 `begins_with ITEMS#` 的索引鍵條件來實現。此外，為了簡化應用程式端的開發，我們可以使用篩選條件表達式來實現存取模式 4 (getPlayerSpecificItem)。

以下是使用篩選條件表達式來篩選 Weapon 類別項目的虛擬程式碼範例：

```
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

篩選條件表達式會在查詢完成之後，結果傳回給用戶端之前進行套用。因此，無論是否存在篩選條件表達式，查詢都會消耗相同數量的讀取容量。

如果存取模式是查詢大型資料集並篩選出大量資料以僅保留一小部分資料，則適當的方法是更有效地設計 DynamoDB 分割區索引鍵和排序索引鍵。例如，在上面的範例中取得特定 `ItemType`，如果每個玩家都有很多物品並且查詢特定 `ItemType` 是典型的存取模式，將 `ItemType` 作為複合索引鍵帶入 SK 會更有效。資料模型看起來會像這樣：`ITEMS#ItemType#ItemId`。

步驟 3：位址存取模式 5 (`updateCharacterAttributes`)、和 6 (`updateItemCount`)

我們使用此步驟解決存取模式 5 (`updateCharacterAttributes`) 和 6 (`updateItemCount`)。當玩家需要修改角色時，例如減少貨幣，或者修改物品中某種武器數量時，可使用 [UpdateItem](#) 以實現這



些存取模式。如要更新玩家的貨幣，但確保它永遠不會低於最低金額，僅在餘額大於或等於最小金額時，我們才可以新增 [the section called “條件表達式”](#) 來減少餘額。此為一個虛擬程式碼範例：

```
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/play er001.png	500 <small>Updated to 500-Amount</small>	10	80	1000

使用 DynamoDB 進行開發並使用 [原子計數器](#) 減少庫存時，我們可以透過使用樂觀鎖定來確保等冪性。以下是原子計數器的虛擬程式碼範例：

```
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"}}	

此外，在玩家用貨幣購買物品的情況下，整個過程需要扣除貨幣並同時新增一個物品。我們可以使用 DynamoDB 交易將多個動作分組在一起，並將它們作為單個 all-or-nothing `TransactWriteItems` 或 `TransactGetItems` 操作提交。`TransactWriteItems` 是同步且冪等的寫入作業，可在單 all-or-nothing 一作業中將多達 100 個寫入動作分組。這些動作的完成具有不可分割性，也就是全部成功或全部失敗。交易有助於消除貨幣重複或消失的風險。如需交易的詳細資訊，請參閱 [DynamoDB 交易範例](#)。

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	Base 資料表/ GSI/LSI	作業	分割區索引鍵 值	排序索引鍵值	其他條件/篩 選條件
getPlayer Friends	基本資料表	GetItem	PK=PlayerID	SK="FRIENDS#playerID"	

存取模式	Base 資料表/ GSI/LSI	作業	分割區索引鍵 值	排序索引鍵值	其他條件/篩 選條件
getPlayerAll 設定檔	BASE 資料表	Query	PK=PlayerID		
getPlayerAll 項目	BASE 資料表	Query	PK=PlayerID	SK begins_wi th "ITEMS#"	
getPlayer Specific項目	BASE 資料表	Query	PK=PlayerID	SK begins_wi th "ITEMS#"	過濾壓縮表達 式:"ItemType =: 項目類 型":{「: 項目類型」 expressio nAttribut eValues:「武 器」}
updateCha racterAtt ributes	基本資料表	UpdateItem	PK=PlayerID	SK="#META DATA#play erID"	UpdateExp ression:「設 定貨幣 = 貨 幣-: 金額」: 「貨幣 >= Condition Expression: 最低金額」

存取模式	Base 資料表/ GSI/LSI	作業	分割區索引鍵 值	排序索引鍵值	其他條件/篩 選條件
updateItem mCount	基本資料表	UpdateItem	PK=PlayerID	SK ="ITEMS#I temID"	更新表達 式: 「設置 Item Count = Item Count - :共融」: { expressio n-tribu te-values : INR」: { 「N」: 「1」}}

## 遊戲設定檔最終結構描述

這是最終的結構描述設計。若要將此結構定義設計下載為 JSON 檔案，請參閱上的 [DynamoDB 範例](#)。  
GitHub

Base 資料表：

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-avatars/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"}}							

## 使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 [NoSQL Workbench](#)，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視覺化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需詳細資訊，請參閱 [the section called “下載”](#)。
2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需詳細資訊，請參閱 [the section called “匯入現有的模型”](#)。
4. 一旦您匯入到 NoSQL Workbench 後，便可以編輯資料模型。如需詳細資訊，請參閱 [the section called “編輯現有的模型”](#)。
5. 若要視覺化您的資料模型、新增範例資料，或從 CSV 檔案匯入範例資料，請使用 NoSQL Workbench 的 [資料視覺化工具](#) 功能。

## DynamoDB 中的投訴管理系統結構描述設計

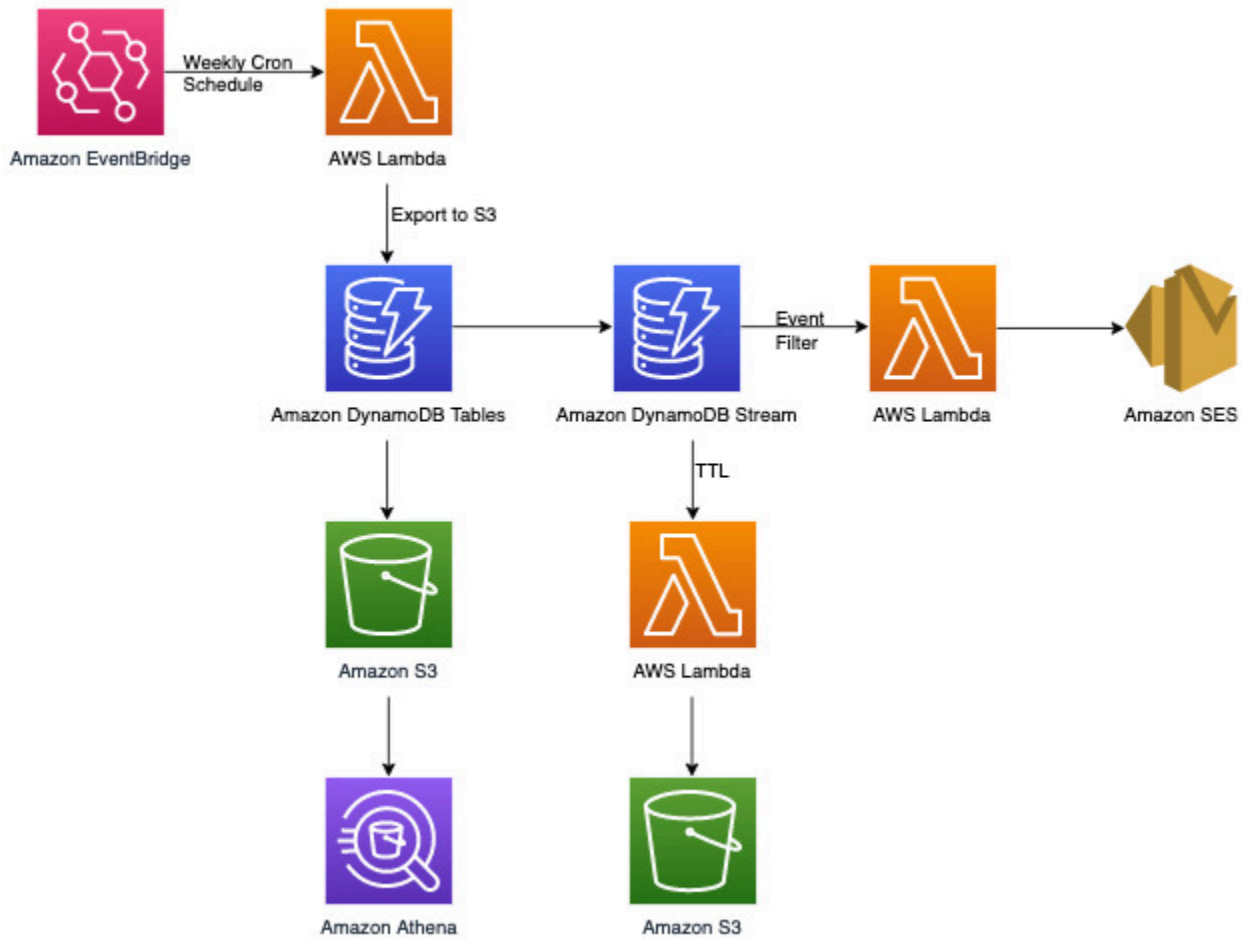
### 投訴管理系統商業使用案例

DynamoDB 是非常適合投訴管理系統 (或聯絡中心) 使用案例的資料庫，因為與其相關聯的大多數存取模式都是採用鍵值型交易查詢。這種情況的典型存取模式會是：

- 建立和更新投訴
- 呈報投訴
- 建立和讀取投訴的評論
- 取得某一位客戶的所有投訴
- 取得某一位客服人員的所有評論並取得所有呈報

某些評論可能包含描述投訴或解決方案的附件。雖然這些全都是鍵值存取模式，但可能會有其他需求，例如，有新評論新增至投訴時傳送通知，或是執行分析查詢，以便了解每週在嚴重性 (或客服人員表現) 方面的投訴分佈情形。與生命週期管理或合規有關的其他需求，可能包括從記錄投訴起算經過三年後封存投訴資料。

## 投訴管理系統架構圖



除了稍後將在「DynamoDB 資料建模」一節中討論的鍵值交易存取模式之外，還有三項非交易需求。上面的架構圖可分成以下三個工作流程：

1. 有新評論新增至投訴時傳送通知
2. 對每週資料執行分析查詢
3. 封存超過三年的資料

讓我們進一步深入探討上述每一項。

有新評論新增至投訴時傳送通知

我們可以使用下面的工作流程來達成此需求：



[DynamoDB Streams](#) 是一種變更資料擷取機制，用來記錄 DynamoDB 資料表上的所有寫入活動。您可以設定 Lambda 函數來觸發部分或全部變更。您可以在 Lambda 觸發程序上設定[事件篩選條件](#)，以篩選掉與使用案例無關的事件。在此範例中，我們只能在有新評論新增時使用篩選條件來觸發 Lambda，並傳送通知至可從 [AWS Secrets Manager](#) 或任何其他憑證存放區提取的相關電子郵件 ID。

### 對每週資料執行分析查詢

DynamoDB 適合用於主要進行線上交易處理 (OLTP) 的工作負載。對於具有分析需求的其他 10-20% 存取模式，可以使用受管的[匯出至 Amazon S3](#) 功能將資料匯出到 S3，這樣做不會影響 DynamoDB 資料表上的即時流量。讓我們來了解下面這個工作流程：



[Amazon EventBridge](#) 可用來定時觸發 AWS Lambda，它可讓您設定 cron 表達式，以便定期進行 Lambda 調用。Lambda 可以調用 `ExportToS3` API 呼叫並將 DynamoDB 資料儲存在 S3 中。隨後 SQL 引擎 (如[Amazon Athena](#)) 就能存取此 S3 資料，以對 DynamoDB 資料執行分析查詢，而不會影響資料表上的即時交易工作負載。依嚴重性層級尋找投訴數目的 Athena 範例查詢如下所示：

```

SELECT Item.severity.S as "Severity", COUNT(Item) as "Count"
FROM "complaint_management"."data"
WHERE NOT Item.severity.S = ''
GROUP BY Item.severity.S ;

```

產生的 Athena 查詢結果如下：

### Results (3)

#	Severity	Count
1	P3	1
2	P2	2
3	P1	1

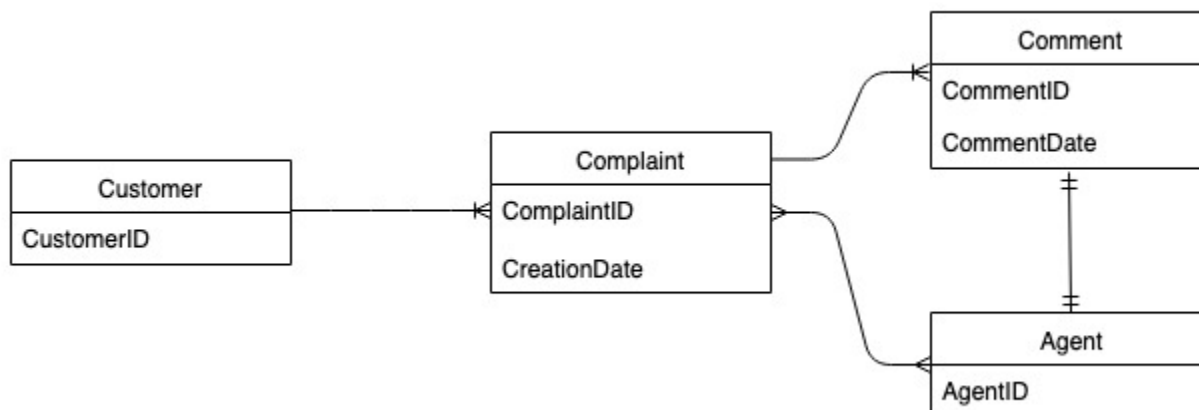
### 封存超過三年的資料

您可以利用 DynamoDB [存留時間 \(TTL\)](#) 功能刪除 DynamoDB 資料表中過時的資料，無需額外費用 (但 2019.11.21 (目前) 版的全域資料表複本除外，其中複寫到其他區域的 TTL 刪除項目會耗用寫入容量)。此資料隨即出現，且可從 DynamoDB Streams 取用並封存至 Amazon S3。此需求的工作流程如下：



### 投訴管理系統實體關係圖

這是我們將用於投訴管理系統結構描述設計的實體關係圖 (ERD)。





## 投訴管理系統存取模式

這些是我們針對投訴管理結構描述設計考量的存取模式。

1. createComplaint
2. updateComplaint
3. updateSeveritybyComplaintID
4. getComplaintByComplaintID
5. addCommentByComplaintID
6. getAllCommentsByComplaintID
7. getLatestCommentByComplaintID
8. getAComplaintbyCustomerIDAndComplaintID
9. getAllComplaintsByCustomerID
10. escalateComplaintByComplaintID
11. getAllEscalatedComplaints
12. getEscalatedComplaintsByAgentID (從最新到最舊的順序)
13. getCommentsByAgentID (兩個日期之間)

## 投訴管理系統結構描述設計演變

由於這是投訴管理系統，因此大多數存取模式都以投訴作為主要實體。高基數的 ComplaintID 將可確保資料在基礎分割區中均勻分佈，同時也是我們確定的存取模式最常見的搜尋條件。因此，ComplaintID 是此資料集中良好的分割區索引鍵選擇。

步驟 1：位址存取模式 1 (**createComplaint**)、2 (**updateComplaint**)、3 (**updateSeveritybyComplaintID**) 和 4 (**getComplaintByComplaintID**)

我們可以使用稱為「中繼資料」(或「AA」) 的通用排序索引鍵來儲存投訴專屬資訊，例如 CustomerID、State、Severity 和 CreationDate。我們使用單一操作搭配 PK=ComplaintID 和 SK="metadata" 來執行下列操作：

1. [PutItem](#) 用來建立新的投訴
2. [UpdateItem](#) 用來更新投訴中繼資料中的嚴重性或其他欄位
3. [GetItem](#) 用來提取投訴的中繼資料

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>

## 步驟 2：位址存取模式 5 (addCommentByComplaintID)

此存取模式須在投訴與投訴的評論之間採用一對多關係模型。我們將在這裡採用[垂直分割](#)技術來使用排序索引鍵，並建立具有不同類型資料的項目集合。只要查看存取模式 6 (getAllCommentsByComplaintID) 和 7 (getLatestCommentByComplaintID)，就會了解評論需依時間排序。我們也可以同時接收多個評論，如此就能使用[複合排序索引鍵](#)技術將時間和 CommentID 附加到排序索引鍵屬性中。

其他處理這類可能發生的評論衝突的選項，包括增加時間戳記的精細度，或是加上一個累加數字作為尾碼，而不要使用 Comment\_ID。在這種情況下，我們將在對應評論之項目的排序索引鍵值加上首碼「comm#」，以實現範圍型操作。

另外還需要確保投訴中繼資料中的 currentState 能反映新增新評論的狀態。新增評論可能表示，投訴已指派給客服人員或已解決等情況。為了在投訴中繼資料中綁定新增評論和更新目前狀態，我們將採用全有或全無的方式使用 TransactWriteItems API。產生的資料表狀態現在看起來像這樣：

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>

讓我們在資料表中新增更多資料，另外也新增 ComplaintID 作為有別於 PK 的另一個欄位，以便在 ComplaintID 需要其他索引時，讓模型能夠因應未來需要。另請注意，某些評論可能包含附件，我們會將這些附件儲存在 Amazon Simple Storage Service 中，並且只會在 DynamoDB 中維護其參考或 URL。最佳實務是盡可能保持交易資料庫精簡，以最佳化成本和效能。資料現在看起來像這樣：

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>

### 步驟 3：位址存取模式 6 (`getAllCommentsByComplaintID`) 和 7 (`getLatestCommentByComplaintID`)

若要取得投訴的所有評論，我們可以在排序索引鍵上使用 `query` 操作搭配 `begins_with` 條件。與其耗用額外的讀取容量來讀取中繼資料項目，隨後因篩選相關結果而造成額外負荷，倒不如使用項這樣的排序索引鍵條件幫助我們只讀取所要的內容。例如，[query](#) 操作搭配 `PK=Complaint123` 和 `SK begins_with comm#` 會傳回下列內容並略過中繼資料項目：

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>

既然我們在模式 7 (getLatestCommentByComplaintID) 中需要投訴的最新評論，那麼就使用兩個額外的查詢參數：

1. ScanIndexForward 應設定為 False，才能以遞減順序排序結果
2. Limit 應設定為 1，才能取得最新 (單獨一個) 評論

類似存取模式 6 (getAllCommentsByComplaintID)，我們使用 begins\_with comm# 作為排序索引鍵條件以略過中繼資料項目。現在，您可以使用查詢操作搭配 PK=Complaint123 和 SK=begin\_with comm#、ScanIndexForward=False、Limit 1，對此設計執行存取模式 7。結果會傳回以下目標項目：

Partition key: PK	Sort key: SK	Attributes					
	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
Complaint123	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

讓我們新增更多虛設資料到資料表中。

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>

#### 步驟 4：位址存取模式 8 (getAComplaintbyCustomerIDAndComplaintID) 和 9 (getAllComplaintsByCustomerID)

存取模式 8 (getAComplaintbyCustomerIDAndComplaintID) 和 9 (getAllComplaintsByCustomerID) 導入了新的搜尋條件：CustomerID。若要從現有資料表提取它，則須使用昂貴的 [Scan](#) 來讀取所有資料，然後篩選相關項目以找出所指的 CustomerID。我們只要建立 [全域次要索引 \(GSI\)](#) 並搭配 CustomerID 作為分割區索

引鍵，就能讓這項搜尋更有效率。務必記住客戶與投訴之間的一對多關係以及存取模式 9 (getAllComplaintsByCustomerID)，ComplaintID 會是排序索引鍵的合適選擇。

GSI 中的資料如下所示：

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>

此 GSI 上存取模式 8 (getAComplaintbyCustomerIDAndComplaintID) 的範例查詢會是：customer\_id=custXYZ、sort key=Complaint1321。結果會是：

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>
custXYZ	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

若要針對存取模式 9 (getAllComplaintsByCustomerID) 取得某一個客戶的所有投訴，GSI 上的查詢會是：customer\_id=custXYZ 作為分割區索引鍵條件。結果會是：

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>

### 步驟 5：位址存取模式 10 (`escalateComplaintByComplaintID`)

此存取將介紹呈報相關層面。若要呈報投訴，我們可以使用 `UpdateItem` 將 `escalated_to` 和 `escalation_time` 等屬性新增至現有投訴中繼資料項目中。DynamoDB 提供了靈活的結構描述設計，這表示非索引鍵屬性集可以合併在一起，也可以分散到不同的項目中。如需範例，請參閱下列內容：

#### 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	

### 步驟 6：位址存取模式 11 (`getAllEscalatedComplaints`) 和 12 (`getEscalatedComplaintsByAgentID`)

預計在整個資料集中，只有少數投訴會呈報。因此，在呈報相關屬性上建立索引將能實現高效率的查詢與符合成本效益的 GSI 儲存。我們可以利用[稀疏索引](#)技術來實現這個目標。具有分割區索引鍵 `escalated_to` 及排序索引鍵 `escalation_time` 的 GSI 如下所示：



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>

若要針對存取模式 11 (`getAllEscalatedComplaints`) 取得所有呈報的投訴，只要掃描此 GSI 即可。請注意，由於 GSI 的大小，此掃描將會是高效能且具有成本效益。若要取得特定客服人員的 (存取模式 12 (`getEscalatedComplaintsByAgentID`)) 的已呈報投訴，分割區索引鍵會是 `escalated_to=agentID`，且我們會將 `ScanIndexForward` 設定為 `False`，以便從最新到最舊排序。

### 步驟 7：位址存取模式 13 (`getCommentsByAgentID`)

對於最後一個存取模式，我們需要以新的維度執行查詢：AgentID。另外還需要依時間進行排序來讀取兩個日期之間的評論，因此我們會建立一個 GSI，並以 `agent_id` 作為分割區索引鍵及 `comm_date` 作為排序索引鍵。此 GSI 中的資料如下所示：

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>	
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"]

此 GSI 的範例查詢會是 `partition key agentID=AgentA` 和 `sort key=comm_date between (2023-04-30T12:30:00, 2023-05-01T09:00:00)`，其結果為：

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"]

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	Base 資料表/ GSI/LSI	操作	分割區索引鍵 值	排序索引鍵值	其他條件/篩 選條件
createCom plaint	BASE 資料表	PutItem	PK=compla int_id	SK=metadata	
updateCom plaint	BASE 資料表	UpdateItem	PK=compla int_id	SK=metadata	
updateSev eritybyCo mplaintID	BASE 資料表	UpdateItem	PK=compla int_id	SK=metadata	
getCompla intByComp laintID	BASE 資料表	GetItem	PK=compla int_id	SK=metadata	
addCommen tByCompla intID	BASE 資料表	TransactW riteltems	PK=compla int_id	SK=metada ta, SK=comm#c omm_date# comm_id	

存取模式	Base 資料表/ GSI/LSI	操作	分割區索引鍵 值	排序索引鍵值	其他條件/篩 選條件
getAllCommentsByComplaintID	BASE 資料表	Query	PK=complaint_id	SK begins_with "comm#"	
getLatestCommentByComplaintID	BASE 資料表	Query	PK=complaint_id	SK begins_with "comm#"	scan_index_forward=False, Limit 1
getAComplaintbyCustomerIDandComplaintID	Customer_complaint_GSI	Query	customer_id=customer_id	complaint_id = complaint_id	
getAllComplaintsByCustomerID	Customer_complaint_GSI	Query	customer_id=customer_id	N/A	
escalateComplaintByComplaintID	BASE 資料表	UpdateItem	PK=complaint_id	SK=metadata	
getAllEscalatedComplaints	Escalations_GSI	Scan	N/A	N/A	
getEscalatedComplaintsByAgentID (從最新到最舊的順序)	Escalations_GSI	Query	escalated_to=agent_id	N/A	scan_index_forward=False

存取模式	Base 資料表/ GSI/LSI	操作	分割區索引鍵 值	排序索引鍵值	其他條件/篩 選條件
getCommentsByAgentID (兩個日期之間)	Agents_Comments_GSI	Query	agent_id= agent_id	SK between (date1, date2)	

## 投訴管理系統最終結構描述

以下是最終結構描述設計。若要將此結構描述設計下載為 JSON 檔案，請參閱 GitHub 上的 [DynamoDB 範例](#)。

### 基底資料表

Primary key		Attributes							
Partition key: PK	Sort key: SK								
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA			
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>		
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB			
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00
Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text				
		comm4	2022-12-31T19:32:00	waiting	<comm text>				
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
Complaint0987	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custXYZ	Complaint0987	assigned	2023-06-10T12:30:08	P3	<description text>		

## Customer\_Complaint\_GSI

Primary key		Attributes							
Partition key: customer_id	Sort key: complaint_id								
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>		
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>		
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00

## Escalations\_GSI

Primary key		Attributes							
Partition key: escalated_to	Sort key: escalation_time								
AgentB	2023-01-03T04:00:07	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
	2023-05-15T14:00:00	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

## Agents\_Comments\_GSI

Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
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"]

## 使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 [NoSQL Workbench](#)，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視覺化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需更多詳細資訊，請參閱 [the section called “下載”](#)。

2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需更多詳細資訊，請參閱 [the section called “匯入現有的模型”](#)。
4. 一旦您匯入到 NoSQL Workbench 後，便可以編輯資料模型。如需更多詳細資訊，請參閱 [the section called “編輯現有的模型”](#)。
5. 若要視覺化您的資料模型、新增範例資料，或從 CSV 檔案匯入範例資料，請使用 NoSQL Workbench 的 [資料視覺化工具](#) 功能。

## DynamoDB 中的週期性付款結構描述設計

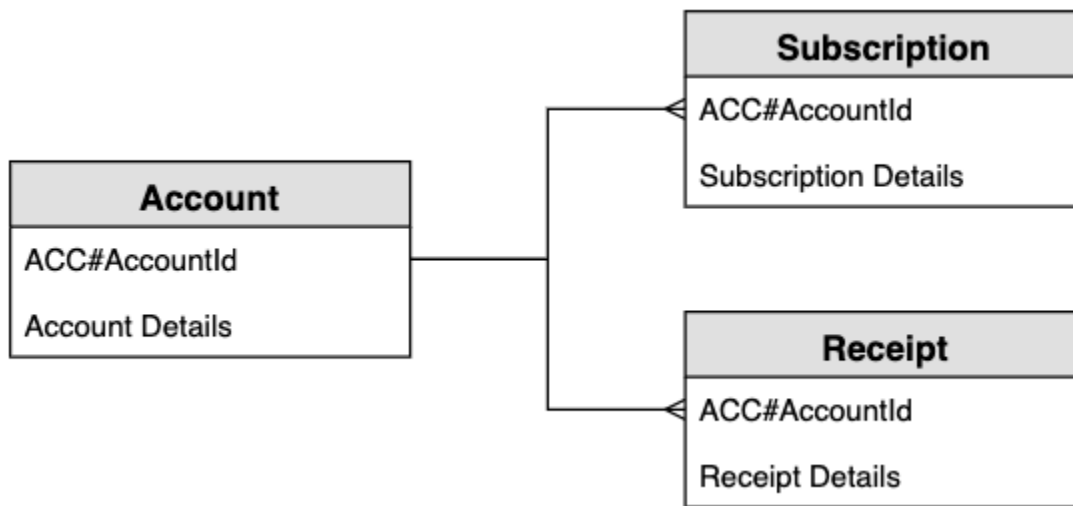
### 週期性付款商業使用案例

此使用案例將說明如何使用 DynamoDB 實作週期性付款系統。資料模型具有以下實體：帳戶、訂閱及收據。我們的使用案例包括以下細節：

- 每個帳戶可以有許多訂閱
- 需要處理下一次付款時，訂閱會有 NextPaymentDate，傳送電子郵件提醒給客戶時，則會有 NextReminderDate
- 有一個代表訂閱的項目會儲存並在付款處理完成時更新 (平均項目大小約為 1KB，輸送量取決於帳戶和訂閱的數目)
- 付款處理器還會在過程中建立一張收據並儲存在資料表中，而且會使用 [TTL](#) 屬性將它設定為在經過一段時間後過期。

### 週期性付款實體關係圖

這是我們將用於週期性付款系統結構描述設計的實體關係圖 (ERD)。



## 週期性付款系統存取模式

這些是我們針對週期性付款系統結構描述設計考量的存取模式。

1. `createSubscription`
2. `createReceipt`
3. `updateSubscription`
4. `getDueRemindersByDate`
5. `getDuePaymentsByDate`
6. `getSubscriptionsByAccount`
7. `getReceiptsByAccount`

## 週期性付款結構描述設計

通用名稱 PK 和 SK 用於金鑰屬性，以允許在相同資料表中儲存不同類型的實體，例如帳戶、訂閱和收據實體。使用者首先建立訂閱，這代表使用者同意在每個月的同一天支付一定金額來換取使用產品。使用者可以選擇要在每個月的哪一天處理付款。另外還會在處理付款之前傳送提醒。應用程式的運作方式是每天執行兩個批次任務：一個批次任務傳送當天應送出的提醒，另一個批次任務則處理當天應付的任何款項。

### 步驟 1：位址存取模式 1 (`createSubscription`)

存取模式 1 (`createSubscription`) 用於初次建立訂閱，並設定包括 `SKU`、`NextPaymentDate`、`NextReminderDate` 和 `PaymentDetails` 等詳細資訊。此步驟僅顯示具有一個訂閱的一個帳戶的資料表狀態。項目集合中可以有多個訂閱，因此這是一對多關係。

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 7ST"}}	2023-05-18T09:41:25.856Z

## 步驟 2：位址存取模式 2 (createReceipt) 和 3 (updateSubscription)

存取模式 2 (createReceipt) 用於建立收據項目。每個月處理付款後，付款處理器會將收據寫回基底資料表。項目集合中可以有多個收據，因此這是一對多關係。付款處理器也會更新訂閱項目 (存取模式 3 (updateSubscription)) 以更新下個月的 NextReminderDate 或 NextPaymentDate。

Primary key		Attributes									
Partition key: PK	Sort key: SK										
ACC#123	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card": {"S": "1234123412341234"}, "default-address": {"S": "12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

## 步驟 3：位址存取模式 4 (getDueRemindersByDate)

應用程式會分批處理當天的付款提醒。因此，應用程式需要存取不同維度的訂閱：日期，而非帳戶。這是一個很好的[全域次要索引 \(GSI\)](#) 使用案例。在此步驟中，我們會新增索引 GSI-1，它會使用 NextReminderDate 作為 GSI 分割區索引鍵。我們不需要複寫所有項目。這個 GSI 是[稀疏索引](#)，因此不會複寫收據項目。我們也不需要對應所有的屬性，只需要包括屬性的子集。下方影像顯示了 GSI-1 的結構描述，它提供了應用程式傳送提醒電子郵件所需的資訊。

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

## 步驟 4：位址存取模式 5 (getDuePaymentsByDate)

應用程式會分批處理當天的付款，採取與處理提醒相同的方式。我們會在此步驟中新增 GSI-2，它會使用 NextPaymentDate 作為 GSI 分割區索引鍵。我們不需要複寫所有項目。這個 GSI 是稀疏索引，因為不會複寫收據項目。下方影像顯示了 GSI-2 的結構描述。

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"}}		

## 步驟 5：位址存取模式 6 (getSubscriptionsByAccount) 和 7 (getReceiptsByAccount)



應用程式會在基底資料表上使用[查詢](#)來鎖定帳戶識別碼 (PK) 為目標，以擷取帳戶的所有訂閱，並使用範圍運算子取得 SK 開頭為「SUB#」的所有項目。應用程式也可以使用相同的查詢結構來擷取所有收據，方法是使用範圍運算子來取得 SK 開頭為「REC#」的所有項目。這樣就能滿足存取模式 6 (getSubscriptionsByAccount) 和 7 (getReceiptsByAccount)。應用程式使用這些存取模式讓使用者看見過去六個月內自己目前的訂閱及過去的收據。在此步驟中，我們不會變更資料表結構描述，而且可以在下面看到我們如何單獨鎖定存取模式 6 (getSubscriptionsByAccount) 中的訂閱項目。

Primary key		Attributes									
Partition key: PK	Sort key: SK										
	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.24Z	12.99	1700318200					
ACC#123	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card": {"S": "1234123412341234"}, "default-address": {"S": "12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	Base 資料表/ GSI/LSI	操作	分割區索引鍵值	排序索引鍵值
createSubscription	BASE 資料表	PutItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
createReceipt	BASE 資料表	PutItem	ACC#account_id	REC#<ReceiptDate>#SKU<SKUID>
updateSubscription	BASE 資料表	UpdateItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
getDueRemindersByDate	GSI-1	Query	<NextReminderDate>	
getDuePaymentsByDate	GSI-2	Query	<NextPaymentDate>	

存取模式	Base 資料表/ GSI/LSI	操作	分割區索引鍵值	排序索引鍵值
getSubscriptionsByAccount	BASE 資料表	Query	ACC#account_id	SK begins_with "SUB#"
getReceiptsByAccount	BASE 資料表	Query	ACC#account_id	SK begins_with "REC#"

## 週期性付款最終結構描述

以下是最終結構描述設計。若要將此結構描述設計下載為 JSON 檔案，請參閱 GitHub 上的 [DynamoDB 範例](#)。

### 基底資料表

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						
ACC#123	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
	s@s.com	28	12.99	2023-05-18T14:15:39.24Z	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					
2023-06-21	2023-05-21T14:15:39.24Z	SK	PK	SKU	Email	NextPaymentDate
		SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

### GSI-2

Primary key		Attributes									
Partition key: NextPaymentDate	Sort key: LastPaymentDate										
2023-06-28	2023-05-18T14:15:39.24Z	PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails			
		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"}}			

## 使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 [NoSQL Workbench](#)，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需更多詳細資訊，請參閱 [the section called “下載”](#)。
2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需更多詳細資訊，請參閱 [the section called “匯入現有的模型”](#)。
4. 一旦您匯入到 NoSQL Workbench 後，便可以編輯資料模型。如需更多詳細資訊，請參閱 [the section called “編輯現有的模型”](#)。
5. 若要視覺化您的資料模型、新增範例資料，或從 CSV 檔案匯入範例資料，請使用 NoSQL Workbench 的 [資料視覺化工具](#) 功能。

## 監控 DynamoDB 中的裝置狀態更新

本使用案例將說明如何使用 DynamoDB 監控 DynamoDB 中的裝置狀態更新 (或裝置狀態變更)

### 使用案例

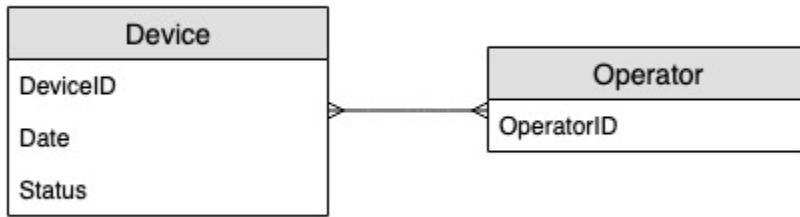
在 IoT 使用案例 (例如智慧工廠) 中，許多裝置需要由作業員監控，並定期將其狀態或記錄傳送至監控系統。當裝置發生問題時，裝置的狀態會從正常轉為警告。根據裝置中異常行為的嚴重性和類型，會有不同的日誌層級或狀態。然後，系統會指派作業員檢查裝置，並在需要時將問題呈報給主管。

此系統的典型存取模式包括：

- 建立裝置的日誌項目
- 取得特定裝置狀態的所有日誌，並顯示最新日誌
- 取得兩個日期之間，給指定作業員的所有日誌
- 取得呈報給指定主管的所有日誌
- 取得呈報給指定主管，並包含特定裝置狀態的所有日誌
- 取得呈報給指定主管，並包含特定日期之特定裝置狀態的所有日誌

### 實體關係圖

這是用於監控裝置狀態更新實體關係圖 (ERD)。



## 存取模式

須使用此存取模式監控裝置狀態更新。

1. `createLogEntryForSpecificDevice`
2. `getLogsForSpecificDevice`
3. `getWarningLogsForSpecificDevice`
4. `getLogsForOperatorBetweenTwoDates`
5. `getEscalatedLogsForSupervisor`
6. `getEscalatedLogsWithSpecificStatusForSupervisor`
7. `getEscalatedLogsWithSpecificStatusForSupervisorForDate`

## 架構設計演進

步驟 1：位址存取模式 1 (`createLogEntryForSpecificDevice`) 和 2 (`getLogsForSpecificDevice`)

裝置追蹤系統的擴展單位為個別裝置。在這個系統中，`deviceID` 會唯一識別裝置，進一步讓 `deviceID` 成為分割區索引鍵的理想候選者。每個設備會定期向追蹤系統發送資訊 (例如每五分鐘左右)，而這類排序會依日期為邏輯排序準則，因此成為排序索引鍵。此使用案例中的範例資料看起來應會與以下內容相似：

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
d#12345	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
d#54321	2020-04-11T09:25:00	State
		WARNING2
	2020-04-11T09:30:00	State
		NORMAL
d#11223	2020-04-27T16:10:00	State
		WARNING4
d#11223	2020-04-27T16:15:00	State
		WARNING4

若要擷取特定裝置的日誌項目，可以使用分割區索引鍵 `DeviceID="d#12345"` 執行[查詢](#)操作。

### 步驟 2：位址存取模式 3 (`getWarningLogsForSpecificDevice`)

由於 `State` 是非索引鍵屬性，必須使用[篩選條件表達式](#)，才可利用目前的結構描述處理存取模式 3。在 DynamoDB 中，使用索引鍵條件表達式讀取資料後，才會套用篩選條件表達式。例如，若想針對 `d#12345` 擷取警告日誌，利用分割區索引鍵 `DeviceID="d#12345"` 進行的查詢操作會讀取上表中的四個項目，然後篩選掉處於警告狀態的項目。這種方法無法有效地大規模進行。如果排除項目的比例較低或不常執行查詢，則篩選條件表達式是排除查詢項目的好方法。但是，我們可以持續改善資料表設計，以便更有效率地從資料表中擷取大量項目，並篩選掉大部分的項目。

若想改變存取模式的處理方法，可以透過[複合排序索引鍵](#)。您可以從排序索引鍵已改為 `State#Date` 的 [DeviceStateLog\\_3.json](#) 匯入範例資料。此排序索引鍵結合了 `State`、`#` 以及 `Date` 屬性。此範例中，`#` 用來作為分隔符。資料現在看起來會像這樣：

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

若僅想擷取裝置的警告日誌，可利用此結構描述更切合查詢。查詢的索引鍵條件使用分割區索引鍵 `DeviceID="d#12345"` 與排序索引鍵 `State#Date begins_with "WARNING"`。此查詢只會讀取三個與警告狀態相關的項目。

### 步驟 3：位址存取模式 4 (`getLogsForOperatorBetweenTwoDates`)

您可以匯入 [DeviceStateLog\\_4.json](#)，其中 `Operator` 屬性已新增至具有範例資料的 `DeviceStateLog` 資料表。

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	

由於 Operator 目前不是分割區索引鍵，因此無法根據以 OperatorID 為基礎的資料表執行直接鍵值對查詢。我們必須使用 OperatorID 上的全域次要索引，建立一個新的[項目集合](#)。存取模式必須根據日期進行查詢，因此日期是[全域次要索引 \(GSI\)](#) 的排序索引鍵屬性。這就是 GSI 現在的樣子：

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	



針對存取模式 4 (getLogsForOperatorBetweenTwoDates) , 您可以使用分割區索引鍵 OperatorID=Liz 查詢此 GSI , 以及 2020-04-11T05:58:00 和 2020-04-24T14:50:00 之間的排序索引鍵 Date。

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	

步驟 4：位址存取模式 5 (`getEscalatedLogsForSupervisor`)、6 (`getEscalatedLogsWithSpecificStatusForSupervisor`) 和 7 (`getEscalatedLogsWithSpecificStatusForSupervisorForDate`)

我們可以利用[疏鬆索引](#)處理這些存取模式。

全域次要索引依預設是疏鬆的，因此只有基底資料表中包含主索引鍵屬性的項目，才會實際出現在索引中。針對要建模的存取模式，這是另一種排除無關項目的方式。

您可以匯入 [DeviceStateLog\\_6.json](#)，其中 `EscalatedTo` 屬性已新增至具有範例資料的 `DeviceStateLog` 資料表。如前所述，並非所有日誌都會呈報給主管。

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

您現在可以建立分區索引鍵為 EscalatedTo、排序索引鍵為 State#Date 的新 GSI。請注意，只有當項目同時具有 EscalatedTo 和 State#Date 屬性時，才會顯示在索引中。

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

其餘的存取模式摘要如下：

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	Base 資料表/ GSI/LSI	操作	分割區索引鍵 值	排序索引鍵值	其他條件/篩 選條件
createLog EntryForS pecificDevice	基底資料表	PutItem	DeviceID= deviceId	State#Dat e=state#date	
getLogsFo rSpecific Device	BASE 資料表	Query	DeviceID= deviceId	State#Date begins_with "state1#"	ScanIndex Forward = False
getWarnin gLogsForS pecificDevice	BASE 資料表	Query	DeviceID= deviceId	State#Date begins_with "WARNING"	
getLogsFo rOperator BetweenTw oDates	GSI-1	Query	Operator= operatorN ame	Date between date1 and date2	
getEscala tedLogsFo rSupervisor	GSI-2	Query	Escalated To=superv isorName		
getEscala tedLogsWi thSpecifi	GSI-2	Query	Escalated To=superv isorName	State#Date begins_with "state1#"	

存取模式	Base 資料表/ GSI/LSI	操作	分割區索引鍵 值	排序索引鍵值	其他條件/篩 選條件
cStatusFo rSupervisor					
getEscala tedLogsWi thSpecifi cStatusFo rSupervis orForDate	GSI-2	Query	Escalated To=superv isorName	State#Dat e begins_wi th "state1#d ate1"	

## 最終結構描述

以下是最終結構描述設計。若要將此結構描述設計下載為 JSON 檔案，請參閱 GitHub 上的 [DynamoDB 範例](#)。

## 基底資料表

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				
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator	Date	State
		d#11223	Sue	2020-04-27T16:15:00	WARNING4

## 使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 [NoSQL Workbench](#)，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視覺化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需更多詳細資訊，請參閱 [the section called “下載”](#)。
2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需更多詳細資訊，請參閱 [the section called “匯入現有的模型”](#)。
4. 一旦您匯入到 NoSQL Workbench 後，便可以編輯資料模型。如需更多詳細資訊，請參閱 [the section called “編輯現有的模型”](#)。
5. 若要視覺化您的資料模型、新增範例資料，或從 CSV 檔案匯入範例資料，請使用 NoSQL Workbench 的 [資料視覺化工具](#) 功能。

## 使用 DynamoDB 做為線上商店的資料存放區

此使用案例說明如何將 DynamoDB 做為線上商店 (或電子商店) 的資料存放區。

### 使用案例

使用者可在線上商店瀏覽各種產品，並進行購買。結帳時客戶可以使用折扣碼或禮品卡付款，並用信用卡支付剩餘金額。從數個倉庫中找出購買產品後，便會運送到客戶提供的地址。線上商店存放區的典型存取模式包括：

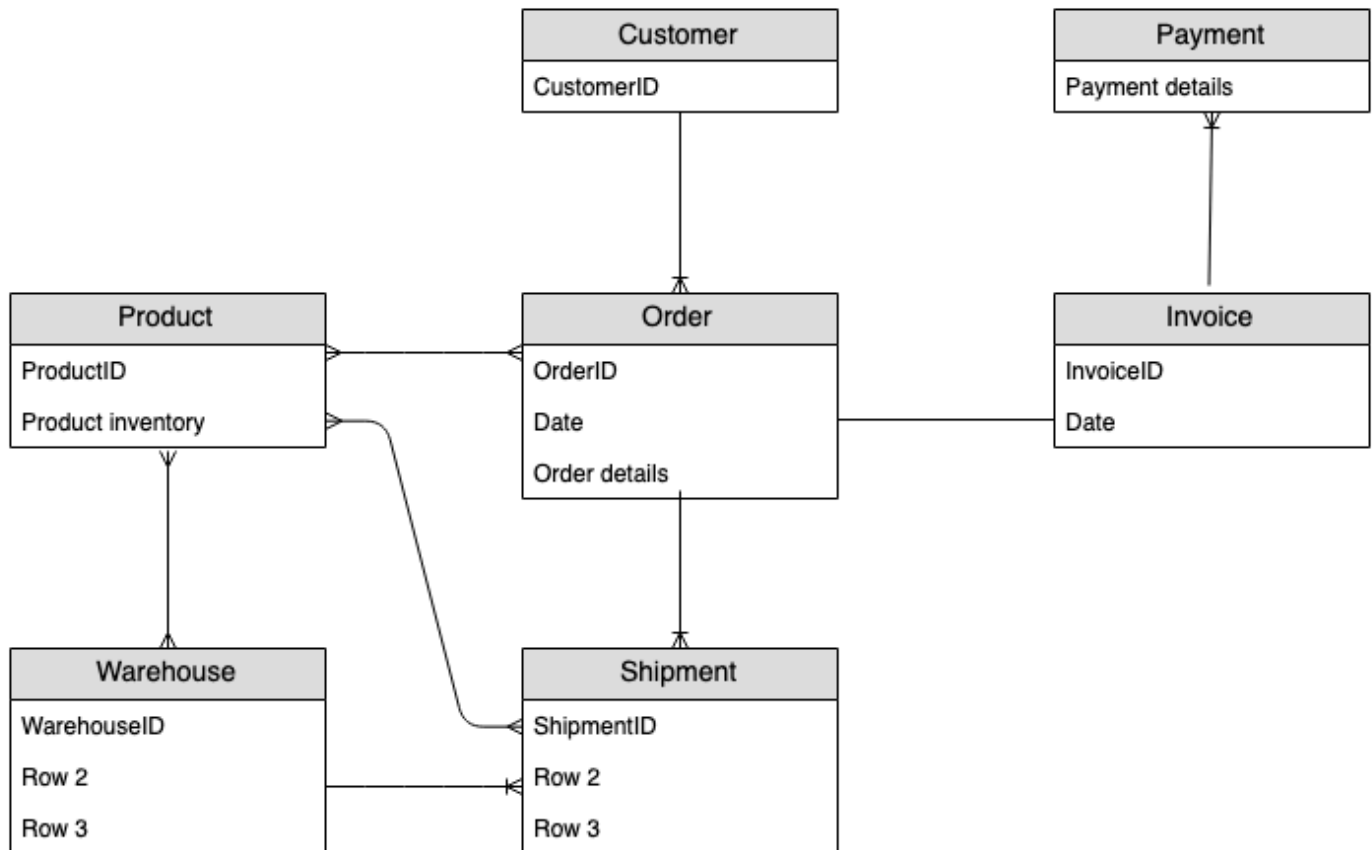
- 取得具指定 CustomerID 的客戶
- 取得具指定 ProductID 的產品
- 取得具指定 WarehouseID 的倉庫
- 透過 ProductID 取得所有倉庫的產品庫存
- 取得具指定 OrderID 的訂單



- 取得具指定 OrderID 的所有產品
- 取得具指定 OrderID 的發票
- 取得具指定 OrderID 的所有運送貨物
- 取得指定日期範圍內，具指定 ProductID 的所有訂單
- 取得具指定 InvoiceID 的發票
- 取得具指定 InvoiceID 的所有付款
- 取得具指定 ShipmentID 的所有運送詳細資訊
- 取得具指定 WarehouseID 的所有運送貨物
- 取得具指定 WarehouseID 的所有產品庫存
- 取得指定日期範圍內，具指定 CustomerID 的所有發票
- 取得指定日期範圍內，由指定 CustomerID 訂購的所有產品

## 實體關係圖

針對如何使用 DynamoDB 做為線上商店的資料存放區，這是用於建模的實體關係圖 (ERD)。



## 存取模式

針對如何使用 DynamoDB 做為線上商店的資料存放區，這些是可採用的存取模式。

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`

## 架構設計演進

使用 [DynamoDB 專用 NoSQL Workbench](#)，導入 [AnOnlineShop\\_1.json](#) 以創建一個名為的新數據模型 `AnOnlineShop` 和一個名為的新表。 `OnlineShop` 請注意，我們將分區索引鍵和排序索引鍵命名為通用名稱 PK 和 SK，藉此可在同一個資料表中儲存不同類型的實體。

### 步驟 1：位址存取模式 1 (`getCustomerByCustomerId`)

匯入 [AnOnlineShop\\_2.json](#) 以處理存取模式 1 () `getCustomerByCustomerId`。某些實體與其他實體並無關聯，因此我們將針對這些實體使用相同的 PK 和 SK 值。在範例資料中，請注意索引鍵字首為 `c#`，以和稍後會從其他實體新增的 `customerId` 進行區別。也請於其他實體重複執行此做法。

若要處理此存取模式，可以利用 `PK=customerId` 和 `SK=customerId` 執行 [GetItem](#) 操作。

## 步驟 2：位址存取模式 2 (getProductByProductId)

匯入 [AnOnlineShop\\_3.json](#) 以解決實體的 product 存取模式 2 (getProductByProductId)。產品實體字首為 p#，並且已使用相同的排序索引鍵屬性來儲存 customerID 與 productID。透過通用命名和 [垂直分割](#) 建立項目集合，可設計有效的單一資料表。

若要處理此存取模式，可以利用 PK=productId 和 SK=productId 執行 GetItem 操作。

## 步驟 3：位址存取模式 3 (getWarehouseByWarehouseId)

匯入 [AnOnlineShop\\_4.json](#) 以解決實體的 warehouse 存取模式 3 (getWarehouseByWarehouseId)。目前我們已將 customer、product 與 warehouse 實體加入同一個資料表，且這些實體均使用字首和 EntityType 屬性。類型屬性 (或字首命名) 能提升模型的可讀性。如果只單純將不同實體的英數 ID 儲存在同一屬性中，閱讀上會相當困難。若少了識別符，就會難以分辨不同實體。

若要處理此存取模式，可以利用 PK=warehouseId 和 SK=warehouseId 執行 GetItem 操作。

Base 資料表：

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"}}	

## 步驟 4：位址存取模式 4 (getProductInventoryByProductId)

匯入 [AnOnlineShop\\_5.json](#) 以解決存取模式 4 () getProductInventoryByProductId。warehouseItem 實體用於跟踪每個倉庫中的產品數量。當倉庫新增或移除產品時，通常會更新此項目。正如 ERD 中所看到的，product 和 warehouse 之間存在 many-to-many 關係。在這裡，從 product 到的 one-to-many 關係 warehouse 被建模為 warehouseItem。稍後，從 warehouse 到的 one-to-many 關係也 product 將被建模。

可透過查詢 `PK=ProductId` 和 `SK begins_with "w#"` 處理存取模式 4。

關於 `begins_with()` 和其他可套用至排序索引鍵的表達式，詳細資訊請參閱[索引鍵條件表達式](#)。

Base 資料表：

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"}}	

步驟 5：位址存取模式 5 (`getOrderDetailsByOrderId`) 和 6 (`getProductByOrderId`)

透過匯入 [AnOnlineShop\\_6.json](#) customerproduct，將更多、和warehouse項目新增至資料表。然後，導入 [AnOnlineShop\\_7.json](#) 以構建可以解決訪問模式 5 (`getOrderDetailsByOrderId`) 和 6 (`getProductByOrderId`) 的項目集合。order您可以看到之間的 one-to-many 關係，order並product建模為 OrderItem 實體。

若想處理存取模式 5 (`getOrderDetailsByOrderId`)，請透過 `PK=orderId` 查詢資料表，了解與訂單相關的所有資訊，包括 `customerId` 和已訂購產品。

Base 資料表：

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

若想處理存取模式 6 (getProductByOrderId)，僅能以 order 讀取產品，此方法可透過利用 PK=orderId 和 SK begins\_with “p#” 查詢資料表完成。

Base 資料表：

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

步驟 6：位址存取模式 7 (getInvoiceByOrderId)

匯入 [AnOnlineShop\\_8.json](#) 以將 invoice 實體新增至訂單項目集合，以處理存取模式 7 (getInvoiceByOrderId)。若要處理此存取模式，可以利用 PK=orderId 和 SK begins\_with “i#” 執行查詢操作。

Base 資料表：

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

### 步驟 7：位址存取模式 8 (`getShipmentByOrderId`)

匯入 [AnOnlineShop\\_9.json](#) 以將shipment實體新增至訂單項目集合，以解決存取模式 8 (`getShipmentByOrderId`)。我們可透過在單一資料表設計中新增更多實體類型，擴展同一個的垂直分割模型。請注意，訂單項目集合中項目間的關係，並不同於 `order`、`shipment`、`orderItem` 及 `invoice` 實體間所擁有的關係。

若想透過 `orderId` 取得運送貨物，您可以利用 `PK=orderId` 和 `SK begins_with "sh#"` 執行查詢操作。

Base 資料表：

Primary key		Attributes				
Partition key: PK	Sort key: SK					
o#12345	c#12345	EntityType	Date			
		order	2020-06-21T19:10:00			
	i#55443	EntityType	Amount		Date	
		invoice	400		2020-06-21T19:18:00	
	p#12345	EntityType	Price		Quantity	
		orderItem	100		2	
p#99887	EntityType	Price		Quantity		
	orderItem	40		5		
o#12345	sh#88899	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

## 步驟 8：位址存取模式 9 (getOrderByProductIdForDateRange)

在上個步驟中，我們建立了訂單項目集合。此存取模式具有新的查詢維度 (ProductID 和 Date)，會要求您掃描整個資料表並篩選相關記錄，以擷取目標項目。為處理這類存取模式，必須建立 [全域次要索引 \(GSI\)](#)。匯入 [AnOnlineShop\\_10.json](#) 以使用 GSI 建立新的項目集合，以便從多個訂單項目集合擷取 orderItem 資料。目前資料內有 GSI1-PK 和 GSI1-SK，分別為 GSI1 的分割區索引鍵和排序索引鍵。

DynamoDB 會自動將資料表中包含 GSI 索引鍵屬性的項目填入 GSI，不須額外手動插入 GSI。

若想處理存取模式 9，請利用 GSI1-PK=productId 和 GSI1SK between (date1, date2) 查詢 GSI1。

## Base 資料表：

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

步驟 9：位址存取模式 10 (**getInvoiceByInvoiceId**) 和 11 (**getPaymentByInvoiceId**)

匯入 [AnOnlineShop\\_11.json](#) 以解決存取模式 10 (**getInvoiceByInvoiceId**) 和 11 (**getPaymentByInvoiceId**)，這兩者都與之相關。invoice 即使這兩種存取模式並不相同，仍須透過索引鍵條件以實現。Payments 的定義為 invoice 實體上具地圖資料類型的屬性。

**Note**

為了儲存不同實體的資訊，GSI1-PK 和 GSI1-SK 會進行多載，以便從同一個 GSI 取得多個存取模式。如需 GSI 多載的詳細資訊，請參閱 [多載全域次要索引](#)。

若想處理存取模式 10 和 11，請利用 GSI1-PK=invoiceId 和 GSI1-SK=invoiceId 查詢 GSI1。

## 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

步驟 10：位址存取模式 12 (`getShipmentDetailsByShipmentId`) 和 13 (`getShipmentByWarehouseId`)

匯入 [AnOnlineShop\\_12.json](#) 以解決存取模式 12 (`getShipmentDetailsByShipmentId`) 和 13 (`getShipmentByWarehouseId`)。

請注意，shipmentItem 實體已新增至基底資料表上的訂單項目集合，以在單一查詢作業中擷取訂單的所有詳細資訊。

Base 資料表：

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": "Slanbar svagen" }, "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": "Slanbar svagen" }, "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					

分GS11割區和排序索引鍵已經用來建立shipment和之間的 one-to-many 關係shipmentItem。若想處理存取模式 12 (getShipmentDetailsByShipmentId) , 請利用 GS11-PK=shipmentId 和 GS11-SK=shipmentId 查詢 GS11。

GS11 :

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": "Slanbar svagen"},           "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": "Slanbar svagen"},           "Number":           {"S": "34"},           "ZipCode":           {"S": "41787"}         }	Express	2020-06-22T10:20:00

我們需要創建另一個 GSI ( GSI2 ) 來 shipment 為訪問模式 13 ( getShipmentByWarehouseId ) 之間 warehouse 和之間的新 one-to-many 關係建模。若想處理此存取模式，請利用 GSI2-PK=warehouseId 和 GSI2-SK begins\_with "sh#" 查詢 GSI2。

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",             "Country":             "S": "Vastra Gotaland"           },           "City":           {             "S": "Goteborg",             "Street":             "S": "Slanbar svagen",             "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",             "Country":             "S": "Vastra Gotaland"           },           "City":           {             "S": "Goteborg",             "Street":             "S": "Slanbar svagen",             "Number":             {               "S": "34"             },             "ZipCode":             {               "S": "41787"             }           }         }	Express	2020-06-22T10:20:00

步驟 11：位址存取模式 14 (`getProductInventoryByWarehouseId`)、15 (`getInvoiceByCustomerIdForDateRange`) 和 16 (`getProductsByCustomerIdForDateRange`)

匯入 [AnOnlineShop\\_13.json](#) 以新增與下一組存取模式相關的資料。若想處理存取模式 14 (`getProductInventoryByWarehouseId`)，請利用 `GSI2-PK=warehouseId` 和 `GSI2-SK begins_with "p#"` 查詢 GSI2。

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	

若想處理存取模式 15 (getInvoiceByCustomerIdForDateRange) , 請利用 GSI2-PK=customerId 和 GSI2-SK between (i#date1, i#date2) 查詢 GSI2。

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	

若想處理存取模式 16 (getProductsByCustomerIdForDateRange) , 請利用 GSI2-PK=customerId 和 GSI2-SK between (p#date1, p#date2) 查詢 GSI2。

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

在 [NoSQL Workbench](#) 中，面向代表適用於 DynamoDB 的應用程式不同資料存取模式。面向為您提供了一種查看資料表中的資料子集的方法，而無需查看不符合面向限制的記錄。面向是一種視覺資料建模工具，在 DynamoDB 中不作為可用的建構存在，因為其純粹用於輔助存取模式建模。

匯入 [AnOnlineShop\\_facets.json](#) 以查看此使用案例的方面。

下表摘要整理了所有存取模式，以及結構描述設計處理這些模式的方式：

存取模式	Base 資料表/ GSI/LSI	操作	分割區索引鍵值	排序索引鍵值
getCustomerByCustomerId	基本資料表	GetItem	PK=customerId	SK=customerId
getProductById	基本資料表	GetItem	PK=productId	SK=productId
getWarehouseByWarehouseId	基本資料表	GetItem	PK=warehouseId	SK=warehouseId
getProductInventoryByProductId	BASE 資料表	Query	PK=productId	SK begins_with "w#"
getOrderDetailsByOrderId	BASE 資料表	Query	PK=orderId	
getProductByOrderId	BASE 資料表	Query	PK=orderId	SK begins_with "p#"
getInvoiceByOrderId	BASE 資料表	Query	PK=orderId	SK begins_with "i#"
getShipmentByOrderId	BASE 資料表	Query	PK=orderId	SK begins_with "sh#"
getOrderByIdForDateRange	GSI1	Query	PK=productId	SK between date1 and date2
getInvoiceById	GSI1	Query	PK=invoiceId	SK=invoiceId
getPaymentByInvoiceId	GSI1	Query	PK=invoiceId	SK=invoiceId



存取模式	Base 資料表/ GSI/LSI	操作	分割區索引鍵值	排序索引鍵值
getShipmentDetailsByShipmentId	GSI1	Query	PK=shipmentId	SK=shipmentId
getShipmentByWarehouseId	GSI2	Query	PK=warehouseId	SK begins_with "sh#"
getProductInventoryByWarehouseId	GSI2	Query	PK=warehouseId	SK begins_with "p#"
getInvoiceByCustomerIdForDateRange	GSI2	Query	PK=customerId	SK between i#date1 and i#date2
getProductsByCustomerIdForDateRange	GSI2	Query	PK=customerId	SK between p#date1 and p#date2

## 網上商店最終模式

以下是最終結構描述設計。若要將此結構定義設計下載為 JSON 檔案，請參閱上的 [DynamoDB 設計模式](#)。GitHub

## 基底資料表

Primary key		Attributes			
Partition key: PK	Sort key: SK				
c#12345	c#12345	EntityType	Email	Name	
		customer	samaneh@example.com	Samaneh	
c#23456	c#23456	EntityType	Email	Name	
		customer	kathleen@example.com	Kathleen	
c#54321	c#54321	EntityType	Email	Name	
		customer	henrik@example.com	Henrik	
p#12345	p#12345	EntityType	Detail	Price	
		product	{"Name": {"S": "Options Open"}, "Description": {"S": "The latest album"}}	100	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#12345	50
p#99887	p#99887	EntityType	Detail	Price	
		product	{"Name": {"S": "The Book"}, "Description": {"S": "The best book ever"}}	40	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#99887	4
	w#12376	EntityType	Quantity		
warehouseItem		4			
w#12345	w#12345	EntityType	Address		
		warehouse	{"Country": {"S": "Sweden"}, "County": {"S": "Vastra Gotaland"}, "City": {"S": "Goteborg"}, "Street": {"S": "MainStreet"}, "Number": {"S": "20"}, "ZipCode": {"S": "41111"}}		

線上商店

		EntityType	Address		
			{"Country": {"S": "Sweden"}, "County": {"S": "Vastra Gotaland"}, "City": {"S": "Goteborg"}, "Street": {"S": "MainStreet"}, "Number": {"S": "20"}, "ZipCode": {"S": "41111"}}		

## GS11

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":"Slanbar svagen"},                     "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":"Slanbar svagen"},                     "Number":                     {"S":"34"},                     "ZipCode":                     {"S":"41787"}                 }	Express	2020-06-22T10:20:00
線上商店	sh#98765	o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country":                     {"S":"Sweden"},                     "Country":                     {"S":"Vastra Gotaland"},                     "City":                     {"S":"Goteborg"},                     "Street":                     {"S":"Slanbar svagen"},                     "Number":                     {"S":"34"},                     "ZipCode":                     {"S":"41787"}                 }	Express	API 版本 2012-08-10 1354 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
線上商店									API 版本 2012-08-10 1356

## 使用 NoSQL Workbench 與此結構描述設計

您可以將此最終結構描述匯入 [NoSQL Workbench](#)，這是為 DynamoDB 提供資料建模、資料視覺化，和查詢開發功能的視覺化工具，以進一步探索和編輯新專案。請依照下列步驟以開始使用：

1. 下載 NoSQL Workbench。如需詳細資訊，請參閱 [the section called “下載”](#)。
2. 下載上面列出的 JSON 結構描述檔案，該檔案已經是 NoSQL Workbench 模型格式。
3. 將 JSON 結構描述檔案匯入到 NoSQL Workbench。如需詳細資訊，請參閱 [the section called “匯入現有的模型”](#)。
4. 一旦您匯入到 NoSQL Workbench 後，便可以編輯資料模型。如需詳細資訊，請參閱 [the section called “編輯現有的模型”](#)。
5. 若要視覺化您的資料模型、新增範例資料，或從 CSV 檔案匯入範例資料，請使用 NoSQL Workbench 的 [資料視覺化工具](#) 功能。

# 從關聯式資料庫移轉至 DynamoDB

將關聯式資料庫移轉至 DynamoDB 需要仔細規劃，以確保成功取得成功。本指南將協助您瞭解此程序的運作方式、您擁有哪些工具，以及如何評估潛在的移轉策略，並選取符合您需求的移轉策略。

## 主題

- [遷移至 DynamoDB 支援的原因](#)
- [將關聯式資料庫移轉至 DynamoDB 時的考量事項](#)
- [了解移轉至 DynamoDB 的運作方式](#)
- [協助移轉至 DynamoDB 的工具](#)
- [選擇適當的策略以移轉至 DynamoDB](#)
- [執行離線 DynamoDB 至](#)
- [執行與 DynamoDB 支援的混合式移轉](#)
- [透過 1:1 移轉每個表格來執行線上移轉至 DynamoDB](#)
- [使用自訂臨時資料表執行線上移轉至 DynamoDB](#)

## 遷移至 DynamoDB 支援的原因

遷移到 Amazon DynamoDB 可為企業和組織帶來一系列令人信服的優勢。以下是 DynamoDB 成為資料庫移轉的吸引力選擇的一些關鍵優勢：

- **可擴充性：**DynamoDB 專為處理大量工作負載而設計，並可無縫調整以適應不斷增長的資料量和流量。使用 DynamoDB，您可以根據需求輕鬆擴展或縮減資料庫，確保應用程式能夠處理突然尖峰的流量，而不會影響效能。
- **效能：**DynamoDB 提供低延遲資料存取，讓應用程式能夠以優異的速度擷取和處理資料。它的分散式架構可確保讀取和寫入操作分佈在多個節點上，即使在高請求率下也能提供一致的 10 毫秒回應時間。
- **完全受管：**DynamoDB 是由 AWS 這表示可以 AWS 處理資料庫管理的作業層面，包括佈建、組態、修補、備份和擴展。這可讓您更專注於開發應用程式，減少資料庫管理工作。
- **無伺服器架構：**DynamoDB 支援無伺服器模型 (稱為 [DynamoDB 隨需](#))，您只需為應用程式發出的實際讀取和寫入請求付費，無需預先佈建容量。此 pay-per-request 模型提供了成本效益和最低的營運開銷，因為您只需支付消耗的資源費用，而無需佈建和監控容量。



- **NoSQL 彈性**：與傳統的關聯式資料庫不同，DynamoDB 遵循 NoSQL 資料模型，提供結構描述設計的彈性。使用 DynamoDB，您可以存放結構化、半結構化和非結構化資料，因此非常適合處理各種不斷發展的資料類型。這種靈活性使開發週期更快，更容易適應不斷變化的業務需求。
- **高可用性和耐久性**：DynamoDB 可跨區域內多個可用區域複寫資料，確保高可用性和資料耐久性。它會自動處理複寫、容錯移轉和復原，將資料遺失或服務中斷的風險降到最低。DynamoDB 提供高達 99.999% 的可用性 SLA。
- **安全性和合規性**：DynamoDB 與之整合，可 AWS Identity and Access Management 提供精細的存取控制。它提供靜態和傳輸中的加密，以確保數據的安全性。DynamoDB 也遵守各種合規性標準，包括 HIPAA、PCI DSS 和 GDPR，讓您能夠符合法規要求。
- **與 AWS 生態系統整合**：DynamoDB 是 AWS 生態系統的一部分，可與其他 AWS 服務 (例如 AWS Lambda AWS CloudFormation、和) 無縫整合。AWS AppSync 此整合可讓您建置無伺服器架構、利用基礎架構即程式碼，以及建立即時資料驅動的應用程式。

## 將關聯式資料庫移轉至 DynamoDB 時的考量事項

關係數據庫系統和 NoSQL 數據庫有不同的優勢和短處。這些差異使得兩個系統之間的數據庫設計不同。

	關聯式資料庫	NoSQL database (NoSQL 資料庫)
查詢資料庫	在關聯式資料庫中，可以靈活地查詢資料，但查詢相對昂貴，而且在高流量情況下無法很好地擴展 (請參閱在 <a href="#">DynamoDB 中製作關聯式資料模型的第一步</a> )。關聯式資料庫應用程式可以在預存程序、SQL 子查詢、大量更新查詢和彙總查詢中實作商務邏輯。	在如 DynamoDB 這類的 NoSQL 資料庫中，可以少數方式有效地查詢資料，但在外部的查詢就非常昂貴而且速度緩慢。對 DynamoDB 的寫入是單一字元。先前在預存程序中執行的應用程式商業邏輯必須重構為在 DynamoDB 以外執行的自訂程式碼 (例如 Amazon EC2 或) 上執行的自訂程式碼。AWS Lambda
設計資料庫	您可以針對彈性進行設計，而不必擔心實作細節或效能。查	您可以專門設計結構描述，使最常見和最重要的查詢盡可能快且價格低廉。系統會打造您

	關聯式資料庫	NoSQL database (NoSQL 資料庫)
	詢最佳化通常不會影響結構描述設計，但標準化相當重要。	的資料結構，以符合企業使用案例的特定要求。

為 NoSQL 數據庫設計需要與關係數據庫管理系統 ( RDBMS ) 設計不同的心態。針對 RDBMS，您可以建立標準化的資料模型，而不需考量存取模式。您可以在稍後有新問題與查詢要求時擴展此模型。您可以將每種資料整理至其資料表。

使用 NoSQL 設計時，在知道需要回答的問題之前，您不應該開始為 DynamoDB 設計結構描述。了解業務問題和應用程序的讀寫模式至關重要。您也應該致力於在 DynamoDB 應用程式中維護盡可能少的表格。擁有較少的資料表可讓事物更具擴展性，需要較少的許可管理，並減少 DynamoDB 應用程式的額外負荷。它還可以幫助降低整體備份成本。

針對 DynamoDB 建立關聯式資料模型，以及建置新版前端應用程式的工作是[另](#)一個主題。本指南假設您建置了使用 DynamoDB 的新版應用程式，但您仍需要確定在切換期間移轉和同步處理歷史資料的最佳方式。

## 尺寸考量

您儲存在 DynamoDB 資料表中的每個項目 (列) 的大小上限為 400 KB。如需詳細資訊，請參閱 [配額和限制](#)。項目大小由項目中所有屬性名稱和屬性值的總大小決定。如需詳細資訊，請參閱 [the section called “項目大小和格式”](#)。

如果您的應用程式需要在某個項目中存放超過 DynamoDB 大小限制允許的資料，請將該項目分解為項目集合、壓縮項目資料，或將該項目作為物件存放在 Amazon Simple Storage Service (Amazon S3) 中，同時將 Amazon S3 物件識別碼存放在 DynamoDB 項目中。請參閱 [the section called “大型項目”](#)。更新項目的成本取決於項目的完整大小。對於需要頻繁更新現有項目的工作負載，具有一個或兩個 KB 的小項目的更新成本將比較大的項目低。[the section called “處理項目收集”](#) 如需項目集合的詳細資訊，請參閱。

在選擇分區和排序關鍵屬性、其他表格設定、項目大小和結構，以及是否要建立次要索引時，請務必檢閱 [DynamoDB 建模文件](#) 以及的指南。[the section called “成本最佳化”](#) 請務必測試您的移轉計劃，以便 DynamoDB 解決方案具有成本效益，並符合 DynamoDB 的功能和限制。

# 了解移轉至 DynamoDB 的運作方式

在檢閱我們可用的移轉工具之前，請考慮 DynamoDB 如何處理寫入。

## Note

DynamoDB 會自動將您的資料分片並分配到多個共用伺服器 and 儲存位置，因此您無法直接將大量資料集直接匯入生產伺服器。

默認和最常見的寫操作是單個 [PutItem](#) API 操作。您可以在迴圈中執行 PutItem 作業來處理資料集。DynamoDB 支援幾乎無限制的並行連線，因此假設您可以設定和執行大規模多執行緒載入常式 (例如 MapReduce 或 Spark)，寫入速度僅受目標資料表的容量限制 (通常也是無限制的)。

將資料載入 DynamoDB 時，瞭解載入器的寫入速度非常重要。如果您載入的項目 (列) 大小為 1KB 或更小，則此速度只是每秒的項目數。然後可以使用足夠的 WCU (寫入容量單位) 佈建目標資料表來處理此速率。如果您的載入器在任何一秒鐘內超過佈建的容量，則可能會限制或完全拒絕額外的要求。您可以在 DynamoDB 主控台監控索引標籤中的 CloudWatch 圖表中查看節流。

可以執行的第二個操作是使用調用的相關 API [BatchWriteItem](#)。BatchWriteItem 允許您將最多 25 個寫入請求合併到一個 API 調用中。這些資料會由服務接收，並以個別要 PutItem 的形式處理至資料表。選擇時 BatchWriteItem，在使用進行單例調用時，您將無法獲得 AWS SDK 隨附的自動重試的優勢。PutItem 因此，如果有任何錯誤 (例如節流異常)，則必須在響應調用中查找任何失敗寫入的列表。BatchWriteItem 如需在節流圖表中偵測到節流警告時處理節 CloudWatch 流警告的詳細資訊，請參閱 [the section called “限流”](#)

使用 [DynamoDB 從 S3 匯入功能可執行第三種類型的資料匯入](#)。此功能可讓您在 Amazon S3 中暫存大型資料集，並要求 DynamoDB 自動將資料匯入新的資料表。導入不是即時的，並且需要與數據集大小成比例的時間。但是，它提供了便利性，因為它不需要編寫 ETL 平台或自訂 DynamoDB 程式碼。匯入功能有限制，因此在停機時間可接受時適用於移轉。S3 中的資料會載入匯入所建立的新資料表，無法將資料載入任何現有資料表。不會執行任何資料轉換，因此需要上游程序將最終格式的資料準備並存放到 S3 儲存貯體。

## 協助移轉至 DynamoDB 的工具

您可以使用數種常見的移轉和 ETL 工具將資料移轉至 DynamoDB。

許多客戶選擇撰寫自己的移轉指令碼和工作，以便為移轉程序建立自訂資料轉換。如果您計劃操作具有大量寫入流量或常規大型批量負載工作的大容量 DynamoDB 表，您可能希望自己建置遷移工具，以便

在大量寫入流量下對 DynamoDB 的行為充滿信心。執行實務移轉時，可以在專案早期體驗例如節流處理和有效率的資料表佈建等案例。

Amazon 提供了一系列可以利用的數據工具，包括 [Apache Kafka 的數據 AWS Database Migration Service \( DMS \)](#)、[AWS Glue](#)、[Amazon EMR](#) 和 [Amazon 託管流](#)。所有這些工具都可用於執行停機時間遷移，而某些可以利用關聯式資料庫變更資料擷取 (CDC) 功能的工具也可以支援線上移轉。在選擇最佳工具時，將有助於考慮組織在每種工具上的技能和經驗，以及每種工具的功能，性能和成本。

## 選擇適當的策略以移轉至 DynamoDB

大型關聯式資料庫應用程式可能會跨越一百或多個表格，並支援多種不同的應用程式功能。進行大型移轉時，請考慮將應用程式分解為較小的元件或微型服務，並一次移轉一小組資料表。然後，您可以在浪潮中將其他元件移轉到 DynamoDB。

選取移轉策略時，某些參數可能會引導您使用一個或另一個解決方案。鑑於我們的需求和可用資源，我們可以在決策樹中展示這些選項，以簡化可用的選項。這裡簡要提到了這些概念（但將在指南後面進行更深入的介紹）：

- **離線遷移**：如果您的應用程式能夠在移轉期間容許某些停機時間，就會大幅簡化遷移程序。
- **混合式移轉**：此方法可允許移轉期間的部分運作時間，例如允許讀取但不寫入，或允許讀取和插入，但不允許更新和刪除。
- **線上移轉**：移轉期間需要零停機時間的應用程式不易移轉，而且可能需要大量規劃和自訂開發。其中一個關鍵決定是估計和權衡建置自訂移轉程序的成本，以及在切換期間停機時間的業務所產生的成本。

If	及	然後
在維護時段期間，您可以將應用程式關閉一段時間，以執行資料移轉。這是離線遷移		使用完整載入工作來使用 AWS DMS 和執行離線移轉。VIEW 如果需要，請使用 SQL 預先塑形源數據。

If	及		然後
<p>您可以在移轉期間以唯讀模式執行應用程式。這是一個混合遷移</p>			<p>停用應用程式或來源資料庫內的寫入。使用完整載入工作來使用 AWS DMS 和執行離線移轉。</p>
<p>在遷移期間，您可以使用讀取和新記錄插入來執行應用程式，但不會進行更新或刪除。這是一個混合遷移</p>	<p>您具備應用程式開發技能，並且可以更新現有的關聯式應用程式，以執行雙重寫入，包括對所有新記錄的 DynamoDB</p>		<p>使用完整載入工作來使用 AWS DMS 和執行離線移轉。同時，部署允許讀取和執行雙重寫入的現有應用程式版本。</p>
<p>您需要以最少的停機時間進行遷移。這是線上移轉</p>	<p>您正在將來源表格 1 對 1 移轉至 DynamoDB，而無需進行重大結構描述變更</p>		<p>用 AWS DMS 於執行線上資料移轉。執行批量載入工作，然後執行 CDC 同步工作</p>
	<p>您正在按照單一資料表理念將來源資料表合併為較少的 DynamoDB 表格</p>	<p>您在 SQL 主機上具備後端資料庫開發技能和備用容量</p>	<p>在 SQL 數據庫中創建 NoSQL 就緒表。使用 JOIN，聯合，視圖，觸發器，存儲過程填充並同步它</p>
		<p>您沒有 SQL 主機上的後端資料庫開發技能和備用容量</p>	<p>考慮混合式或離線移轉方法</p>

If	及	然後
	您可以略過遷移歷史交易資料，或將其存檔到 Amazon S3 中以代替遷移。您只需要遷移幾個小型靜態表	撰寫指令碼或使用任何 ETL 工具移轉資料表。VIEW 如果需要，請使用 SQL 預先塑形源數據。

## 執行離線 DynamoDB 至

離線移轉適用於允許停機時間視窗執行移轉的情況。關聯式資料庫通常每個月至少需要一些停機時間來進行維護和修補，或者硬體升級或主要發行版本升級可能需要較長的停機時間。

Amazon S3 可以用作移轉期間的暫存區域。您可以使用從 S3 匯入功能，將以 CSV (逗號分隔值) 或 DynamoDB JSON 格式儲存的資料自動匯入到新的 [DynamoDB](#) 表格中。

### 計劃

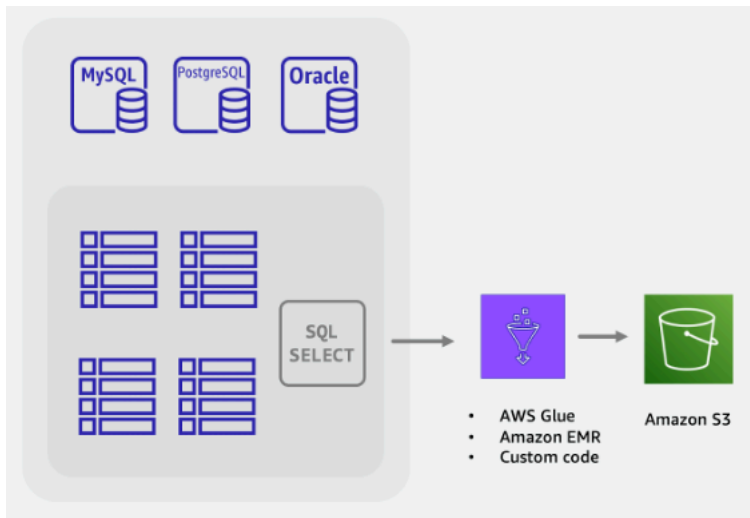
使用 Amazon S3 執行離線遷移

### 工具

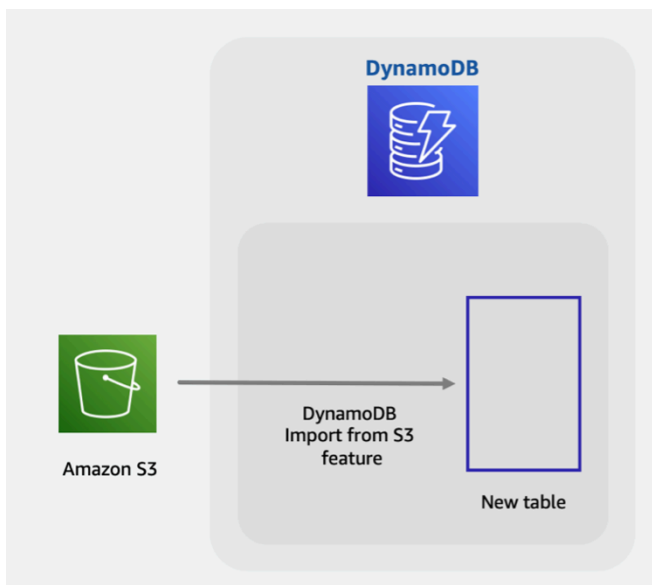
- 擷取和轉換 SQL 資料並將其存放在 S3 儲存貯體的 ETL 任務，例如：
  - AWS Glue
  - Amazon EMR
  - 您自己的自訂程式碼
- 從 S3 匯入功能

### 離線遷移步驟：

1. 建立可查詢 SQL 資料庫的 ETL 任務、將資料表資料轉 DynamoDB JSON 或 CSV 格式，然後將其儲存至 S3 儲存貯體。



2. 會叫用 DynamoDB 從 S3 匯入功能來建立新表格，並自動從 S3 儲存貯體載入資料。



完全離線遷移既簡單又直接，但應用程式擁有者和使用者可能不受歡迎。如果應用程式可以在移轉期間提供降低等級的服務，而不是完全沒有服務，使用者將會受益。

您可以新增功能以在離線移轉期間停用寫入，同時允許正常繼續讀取。在移轉關聯式資料時，應用程式使用者仍然可以安全地瀏覽和查詢現有資料。如果這是您要尋找的內容，請繼續閱讀以了解[混合式遷移](#)。

## 執行與 DynamoDB 支援的混合式移轉

雖然所有資料庫應用程式都執行讀取和寫入作業，規劃混合式或線上移轉時，應考慮所執行的寫入作業類型。資料庫寫入可分為三個儲存貯體：插入、更新和刪除。某些應用程式不會對現有記錄執行任何



更新。其他應用程式可能不需要立即處理刪除作業，例如，可能會在月底將刪除項目延遲至大量清理程序。這些類型的應用程式可以更輕鬆地遷移，同時允許部分運作時間。

## 計劃

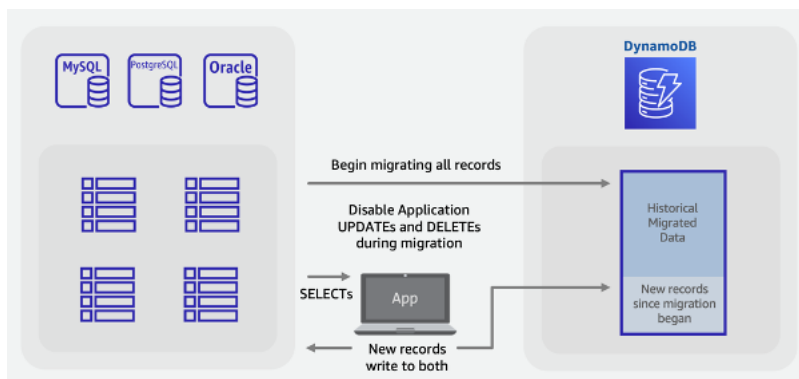
### 使用應用程式雙重寫入執行混合線上/離線移轉

## 工具

- 擷取和轉換 SQL 資料並將其存放在 S3 儲存貯體的 ETL 任務，例如：
  - AWS Glue
  - Amazon EMR
  - 您自己的自訂程式碼

### 混合式移轉步驟：

1. 建立目標 DynamoDB 資料表。此表格將接收歷史大量資料和新的即時資料
2. 建立舊版應用程式版本，在對 SQL 資料庫和 DynamoDB 進行雙重寫入時執行所有插入作業時停用刪除和更新
3. 開始 ETL 工作以移轉現有資料並同時部署新的應用程式版本
4. ETL 工作完成時，DynamoDB 將擁有所有現有和新記錄，並準備好進行應用程式切換



### Note

ETL 工作會直接從 SQL 寫入 DynamoDB 料庫。我們無法像離線遷移範例中那樣使用 S3 匯入功能，因為在整個匯入完成之前，目標資料表不會變為公開，而且可用於其他寫入。



## 透過 1:1 移轉每個表格來執行線上移轉至 DynamoDB

許多關聯式資料庫都有一項稱為「變更資料擷取」(CDC) 的功能，其中資料庫可讓使用者要求對特定時間點之前或之後發生的資料表變更清單。CDC 會使用內部記錄檔來啟用此功能，而且不需要資料表具有任何時間戳記資料行即可運作。

將 SQL 表的結構定義遷移到 NoSQL 數據庫時，您可能需要將數據合併並重新塑形為較少的表格。這樣做可讓您在單一位置收集資料，並避免在多步驟讀取作業中手動連結相關資料。但是，並非總是需要單一表格資料整形，有時您會將表格 1 對 1 移轉到 DynamoDB。這些 1 對 1 資料表移轉不太複雜，因為您可以利用來源資料庫 CDC 功能，使用支援此類移轉類型的常見 ETL 工具。每行的數據仍然可以轉換為新格式，但每個表的範圍保持不變。

請考慮將 SQL 表格 1 對 1 移轉至 DynamoDB，並注意沒有伺服器端連結。

### 計劃

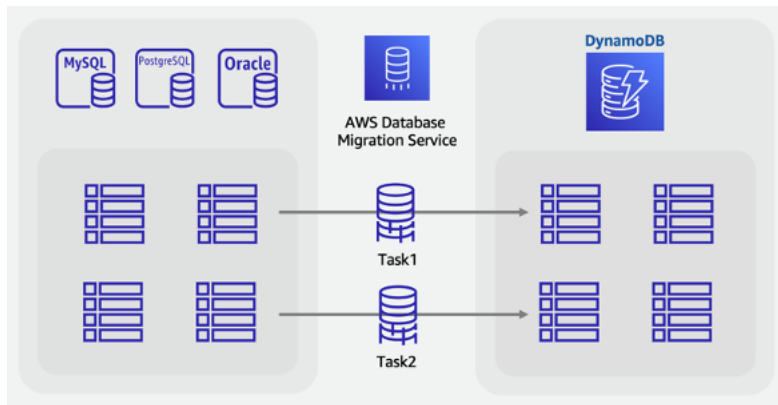
使用下列方法，將每個表格線上移轉到 DynamoDB 中 AWS DMS

### 工具

- [AWS Database Migration Service \(DMS\)](#) 是一種 ETL 工具，可以大量載入歷史資料，也可以利用 CDC 同步處理來源和目標資料表

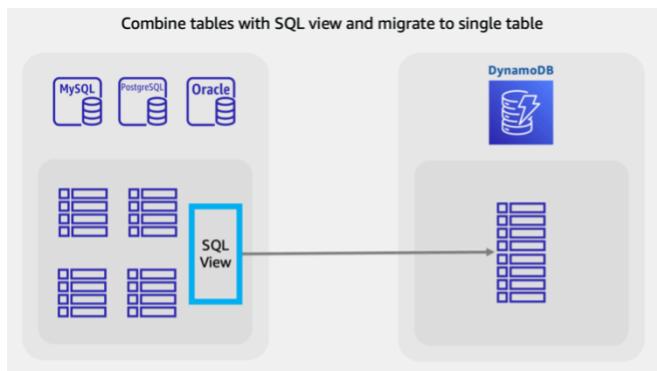
### 線上移轉步驟：

1. 識別將要移轉的來源結構描述中的表格
2. 在具有類似索引鍵結構的 DynamoDB 中建立相同數量的表格
3. 在中建立複製伺服器 AWS DMS 並設定來源端點和目標端點
4. 定義所需的任何每行轉換（例如連接列或將日期轉換為 ISO-8601 字符串格式）
5. 針對「全載」和「變更資料擷取」的每個表格建立移轉任務
6. 監視這些工作，直到進行中的複寫階段已開始
7. 此時，您可以執行任何驗證稽核，然後將使用者切換到讀取和寫入 DynamoDB 的應用程式



## 使用自訂臨時資料表執行線上移轉至 DynamoDB

您可能希望合併資料表以利用唯一的 NoSQL 存取模式 (例如，將四個舊版資料表轉換為單一 DynamoDB 表格)。單一索引鍵值文件要求或預先群組項目集合的查詢傳回通常會比執行多資料表聯結的 SQL 資料庫具有更好的延遲時間。但是，這會使遷移任務變得更加困難。SQL VIEW 可以在源數據庫中完成工作，以準備表示一個集合中所有四個表的單個數據集。



此視圖可能會將 JOIN 表格轉換為非規範化形式，或者可以使用 SQL 將實體標準化並堆疊表。UNION [本片](#) 涵蓋重塑關聯式資料的重要決策。對於離線移轉，使用檢視合併資料表是形狀 DynamoDB 單一表格結構定義資料的好方法。

但是，對於具有即時變更資料的線上移轉，您無法利用 CDC 功能，因為這些功能僅支援單一 VIEW 資料表查詢，而不支援。如果您的資料表包含上次更新的時間戳記資料行，而且這些資料行已合併到中 VIEW，則您可以建置自訂 ETL 工作，使用這些工作來透過同步處理實現大量載入。

應對此挑戰的一個新穎方法是使用標準 SQL 功能 (例如檢視、預存程序和觸發程序)，以建立採用最終所需 DynamoDB NoSQL 格式的新 SQL 表格。

如果您的資料庫伺服器可以配置額外的儲存空間，則可以在移轉開始之前建立此單一臨時資料表。這可透過撰寫將從現有資料表讀取的預存程序、視需要轉換資料，以及寫入新的臨時資料表來完成。可以添

加一組觸發器，以實時將表中的更改複製到臨時表中。如果每個公司原則不允許觸發程序，變更預存程序也可以達到相同的結果。您可以在任何寫入資料的程序中加入幾行程式碼，以便另外將相同的變更寫入暫存資料表。

將此暫存資料表與舊版應用程式資料表完全同步化，將為您提供即時移轉的絕佳起點。使用資料庫 CDC 來完成即時移轉的工具，例如 AWS DMS，現在可以針對此資料表使用。這種方法的優點是它使用關聯式資料庫引擎中已知的 SQL 技能和功能。

## 計劃

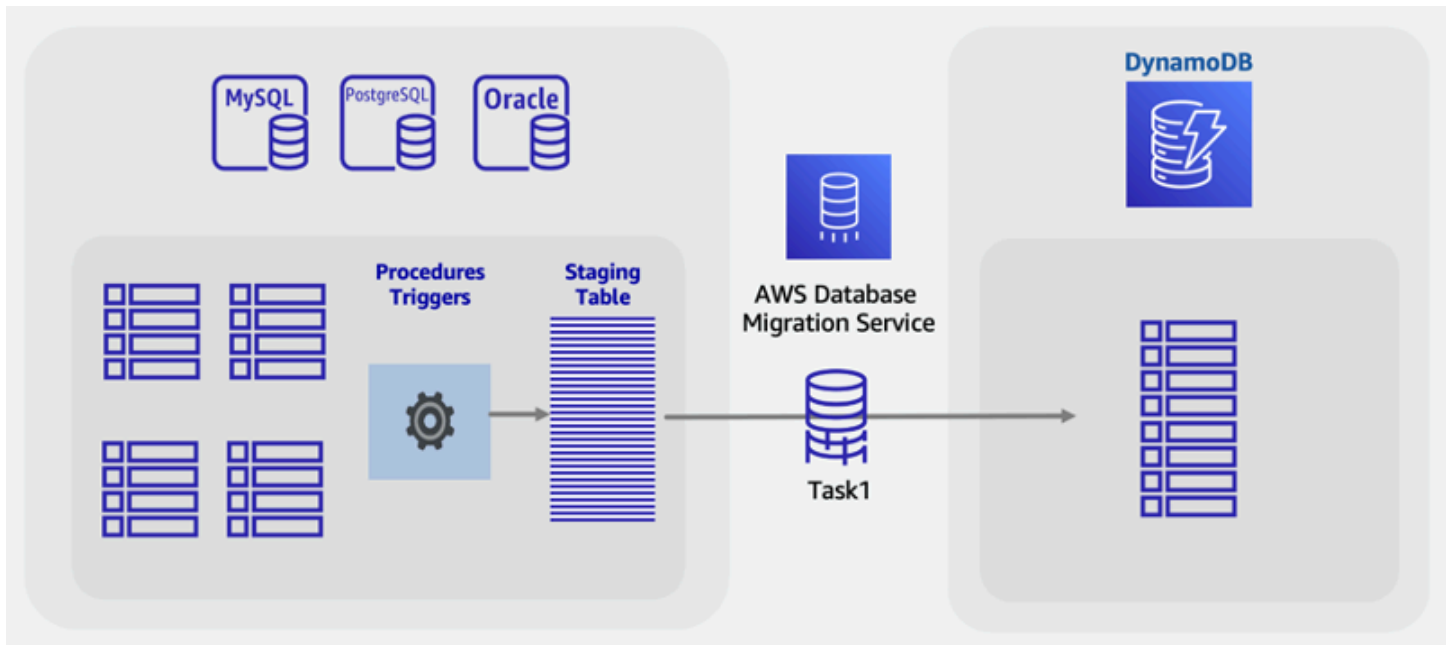
### 使用 SQL 臨時資料表執行線上移轉 AWS DMS

## 工具

- 自訂 SQL 預存程序或觸發程序
- [AWS Database Migration Service \(DMS\)](#)，可將即時臨時資料表移轉至 DynamoDB 的 ETL 工具

## 線上移轉步驟：

1. 在來源關聯式資料庫引擎中，請確定有一些額外的磁碟空間和處理容量。
2. 在 SQL 資料庫中建立新的臨時資料表，並啟用時間戳記或 CDC 功能
3. 撰寫並執行預存程序，將現有的關聯式資料表資料複製到臨時資料表
4. 在對現有資料表執行正常寫入時，部署觸發程序或修改現有程序，以雙重寫入新的臨時資料表
5. 執行 AWS DMS 以將此來源表格移轉並同步至目標 DynamoDB 表格



本指南介紹了將關聯式資料庫資料移轉至 DynamoDB 的幾個考量和方法，重點是將停機時間降至最低，並使用常見的資料庫工具和技術。如需詳細資訊，請參閱下列內容：

- [AWS DMS 使用者指南](#)
- [AWS Glue 使用者指南](#)
- [從資料庫管理系統移轉至 DynamoDB 的最佳作法](#)

# DynamoDB 專用 NoSQL Workbench

Amazon DynamoDB 專用 NoSQL Workbench 是用於現代資料庫開發和操作的跨平台用戶端 GUI 應用程式。適用於 Windows、macOS 和 Linux。NoSQL Workbench 是視覺化開發工具，提供了資料模型建立、資料視覺化和查詢開發功能，協助您設計、建立、查詢及管理 DynamoDB 資料表。NoSQL Workbench 現在包含 DynamoDB 本機版做為安裝程序的選用部分，可讓您更輕鬆地在 DynamoDB 本機版中建立資料模型。若要深入了解 DynamoDB 本機版及其要求，請參閱 [設定 DynamoDB Local \(可下載版本\)](#)。

## 建立資料模型

借助 DynamoDB 專用 NoSQL Workbench，您可以使用滿足您應用程式資料存取模式的現有資料模型，來建置新的資料模型或設計模型。您也可以程序結束時，匯入及匯出設計好的資料模型。如需詳細資訊，請參閱 [使用 NoSQL Workbench 建立資料模型](#)。

## 資料視覺化

資料模型視覺化工具提供畫布，您可在此映射查詢以及視覺化應用程式的存取模式 (面向)，不必編寫程式碼。每個面向都會對應 DynamoDB 中不同的存取模式。您可以自動產生範例資料，以便在資料模型中使用。如需詳細資訊，請參閱 [視覺化資料存取模式](#)。

## 建立操作

NoSQL Workbench 提供強大的圖形使用者界面供您開發及測試查詢。您可以使用 operation builder (操作建置器) 來檢視、探索及查詢即時資料集。結構化操作生成器支持投影表達式，條件表達式，並生成多種語言的示例代碼。您可以將表格從一個 Amazon DynamoDB 帳戶直接複製到不同區域中的另一個帳戶。您也可以直接在 DynamoDB 本機和 Amazon DynamoDB 帳戶之間複製表格，以便在開發環境之間更快速地複製表格的金鑰結構描述 (以及選擇性的 GSI 結構描述和項目)。如需詳細資訊，請參閱 [使用 NoSQL Workbench 探索資料集與建立操作](#)。

下面的視頻詳細介紹了與 NoSQL 工作台數據建模的概念。

## 主題

- [下載 DynamoDB 專用 NoSQL Workbench](#)
- [安裝 DynamoDB 專用 NoSQL Workbench](#)
- [使用 NoSQL Workbench 建立資料模型](#)
- [視覺化資料存取模式](#)

- [使用 NoSQL Workbench 探索資料集與建立操作](#)
- [NoSQL Workbench 的範例資料模型](#)
- [NoSQL Workbench 的版本歷史記錄](#)

## 下載 DynamoDB 專用 NoSQL Workbench

請依照這些說明下載 Amazon DynamoDB 專用 NoSQL Workbench 與 DynamoDB 本機版\*。

先決條件

Ubuntu 安裝需要兩個先決條件的軟體：libfuse2 和捲曲。

利布夫塞 2

從 Ubuntu 22.04 開始，默認情況下不再安裝庫庫 2。要解決此問題，請運行 `sudo add-apt-repository universe && sudo apt install libfuse2` 以安裝最新的 [Ubuntu 版本](#)。

curl

更新 Ubuntu，運行 `sudo apt update && sudo apt upgrade`

接下來，安裝 cURL，執行：`sudo apt install curl`

若要下載 NoSQL Workbench 和 DynamoDB 本機版

1. 下載適用您作業系統的 NoSQL Workbench。

作業系統	下載連結
macOS (英特爾) **	<a href="#">下載適用於 macOS (英特爾)</a>
蘋果矽	<a href="#">下載適用於 macOS (蘋果矽)</a>
Windows	<a href="#">適用於 Windows 的下載</a>
Linux***	<a href="#">下載 Linux 版本</a>

\* NoSQL Workbench 包含 DynamoDB 本機版做為安裝程序的選用部分。

\*\* 如果您嘗試開啟 NoSQL Workbench 時出現警告訊息，指出該應用程式並未由已識別的開發人員向 Apple 註冊，請執行以下操作：

1. 找到應用程式，然後將其打開。
2. 按住 Control 鍵並按一下 App 圖像，然後從快捷鍵選單中選擇「開啟」。

這會將應用程式儲存為安全性設定的例外狀況。如同開啟任何已註冊的應用程式一樣，按兩下應用程式即可開啟該 app。

\*\*\* NoSQL 工作台支持 Ubuntu 12.04，Fedora 21 和 Debian 8，或這些 Linux 發行版的任何較新版本。

2. 啟動已下載的應用程式，並依照步驟安裝 NoSQL Workbench。

#### Note

若要執行本機 DynamoDB，需要 Java 執行階段環境 (JRE) 11.x 版或更新版本。

## 安裝 DynamoDB 專用 NoSQL Workbench

請依照下列步驟在支援的平台上安裝 NoSQL Workbench 和 DynamoDB 本機版。

### Windows

在 Windows 安裝 NoSQL Workbench

1. 執行 NoSQL Workbench 安裝程式應用程式，並選擇安裝語言。然後選擇 OK (確定) 以開始設定。如需下載 NoSQL Workbench 的詳細資訊，請參閱 [下載 DynamoDB 專用 NoSQL Workbench](#)。
2. 選擇 Next (下一步) 繼續設定，然後在下一個畫面選擇 Next (下一步)。
3. Install DynamoDB Local (安裝 DynamoDB 本機版) 核取方塊會預設選取，以在安裝過程中包含 DynamoDB 本機版。請保持選取此選項，確保 DynamoDB 本機版順利安裝，且目的地路徑將與 NoSQL Workbench 的安裝路徑相同。若清除此選項的核取方塊，會略過 DynamoDB 本機版的安裝程序，且安裝路徑僅適用於 NoSQL Workbench。

選擇安裝軟體目的地，然後選擇 Next (下一步)。

**Note**

如果您選擇不包括 DynamoDB 本機做為安裝的一部分，請清除安裝 DynamoDB 本機核取方塊，選擇下一步，然後跳至步驟 6。您可以稍後再個別下載 DynamoDB 本機版。如需詳細資訊，請參閱 [設定 DynamoDB Local \(可下載版本\)](#)。

請在此步驟中設定您的安裝路徑。

4. 選擇 DynamoDB 本機版要使用的連接埠號碼。預設連接埠為 8000。輸入連接埠號碼後，請選擇 Next (下一步)。
5. 選擇 Next (下一步) 以開始設定。
6. 設定完成後，請選擇 Finish (完成)，關閉設定畫面。
7. 在安裝路徑中開啟應用程式，例如 `/programs/DynamoDBWorkbench/`。

## macOS

若要在 macOS 上安裝 NoSQL Workbench

1. 執行 NoSQL Workbench 安裝程式應用程式，並選擇安裝語言。然後選擇 OK (確定) 以開始設定。如需下載 NoSQL Workbench 的詳細資訊，請參閱 [下載 DynamoDB 專用 NoSQL Workbench](#)。
2. 選擇 Next (下一步) 繼續設定，然後在下一個畫面選擇 Next (下一步)。
3. Install DynamoDB local (安裝 DynamoDB 本機版) 核取方塊會預設選取，以在安裝過程中包含 DynamoDB 本機版。請保持選取此選項，確保 DynamoDB 本機版順利安裝，且目的地路徑將與 NoSQL Workbench 的安裝路徑相同。若清除此選項，會略過 DynamoDB 本機版的安裝程序，且安裝路徑僅適用於 NoSQL Workbench。

選擇安裝軟體目的地，然後選擇 Next (下一步)。

**Note**

如果您選擇不包括 DynamoDB 本機做為安裝的一部分，請清除安裝 DynamoDB 本機核取方塊，選擇下一步，然後跳至步驟 6。您可以稍後再個別下載 DynamoDB 本機版。如需詳細資訊，請參閱 [設定 DynamoDB Local \(可下載版本\)](#)。



請在此步驟中設定您的安裝路徑。

4. 選擇 DynamoDB 本機版要使用的連接埠號碼。預設連接埠為 8000。輸入連接埠號碼後，請選擇 Next (下一步)。
5. 選擇 Next (下一步) 以開始設定。
6. 設定完成後，請選擇 Finish (完成)，關閉設定畫面。
7. 在安裝路徑中開啟應用程式，例如 /Applications/DynamoDBWorkbench/。

#### Note

適用於 macOS 的 NoSQL Workbench 會執行自動更新。若要取得更新通知，請在「系統偏好設定」>「通知」中啟用對 NoSQL Workbench 的通知。

## Linux

若要在 Linux 上安裝 NoSQL Workbench

1. 執行 NoSQL Workbench 安裝程式應用程式，並選擇安裝語言。然後選擇 OK (確定) 以開始設定。如需下載 NoSQL Workbench 的詳細資訊，請參閱 [下載 DynamoDB 專用 NoSQL Workbench](#)。
2. 選擇 Forward (轉送) 繼續設定，然後在下一個畫面選擇 Forward (轉送)。
3. Install DynamoDB local (安裝 DynamoDB 本機版) 核取方塊會預設選取，以在安裝過程中包含 DynamoDB 本機版。請保持選取此選項，確保 DynamoDB 本機版順利安裝，且目的地路徑將與 NoSQL Workbench 的安裝路徑相同。若清除此選項，會略過 DynamoDB 本機版的安裝程序，且安裝路徑僅適用於 NoSQL Workbench。

選擇安裝軟體目的地，然後選擇 Forward (轉送)。

#### Note

如果您選擇不包括 DynamoDB 本機做為安裝的一部分，請清除安裝 DynamoDB 本機核取方塊，選擇轉寄，然後跳至步驟 6。您可以稍後再個別下載 DynamoDB 本機版。如需詳細資訊，請參閱 [設定 DynamoDB Local \(可下載版本\)](#)。

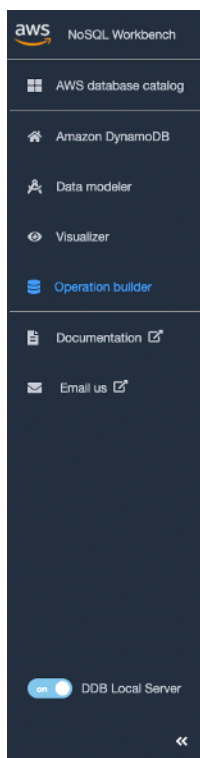
請在此步驟中設定您的安裝路徑。

4. 選擇 DynamoDB 本機版要使用的連接埠號碼。預設連接埠為 8000。輸入連接埠號碼後，請選擇 Forward (轉送)。
5. 選擇 Forward (轉送) 以開始設定。
6. 設定完成後，請選擇 Finish (完成)，關閉設定畫面。
7. 在安裝路徑中開啟應用程式，例如 `/usr/local/programs/DynamoDBWorkbench/`。

### Note

如果您選擇將 DynamoDB 本機版做為安裝 NoSQL Workbench 的一部分，則 DynamoDB 本機版將預先設定為預設選項。若要編輯預設選項，請修改位於 `/資源/ddblocal_LocalStart` 腳本/目錄中的 DDB 指令碼。您可以在安裝過程提供的路徑找到此選項。若要進一步了解 DynamoDB 本機版選項，請參閱 [DynamoDB Local 使用須知](#)。

如果您選擇將 DynamoDB 本機版做為安裝 NoSQL Workbench 的一部分，您可以存取切換，以啟用和停用 DynamoDB 本機版，如下圖所示。



# 使用 NoSQL Workbench 建立資料模型

您可以使用 Amazon DynamoDB 專用 NoSQL Workbench 中的資料模型建立工具，建立新的資料模型或根據能滿足您應用程式資料存取模式的現有資料模型設計模型。資料模型建立工具包含幾個協助您開始的範例資料模型。

## 主題

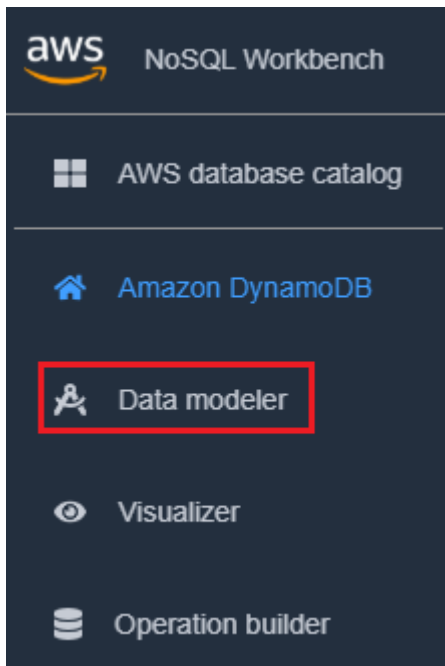
- [建立新的資料模型](#)
- [匯入現有的資料模型](#)
- [匯出資料模型](#)
- [編輯現有的資料模型](#)

## 建立新的資料模型

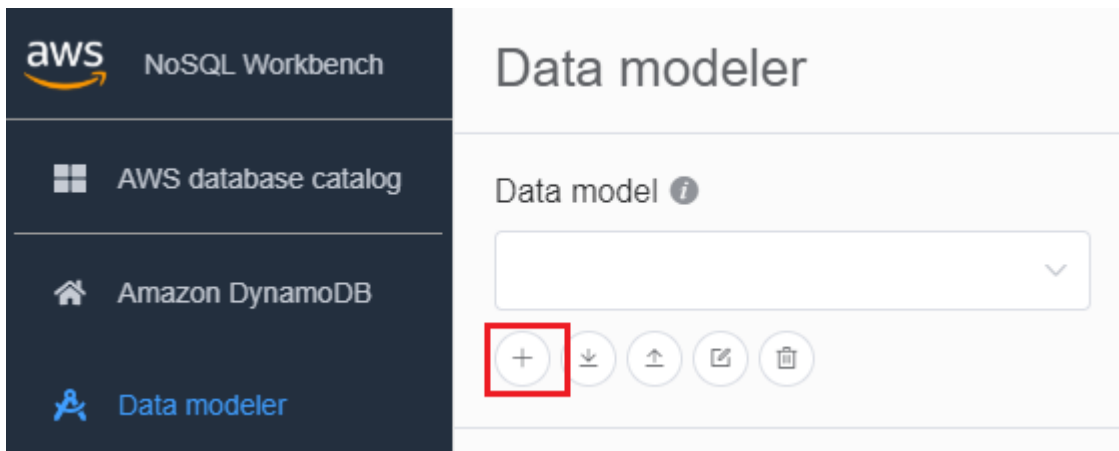
使用 NoSQL Workbench，在 Amazon DynamoDB 中依照這些步驟建立新的資料模型。

### 建立新的資料模型

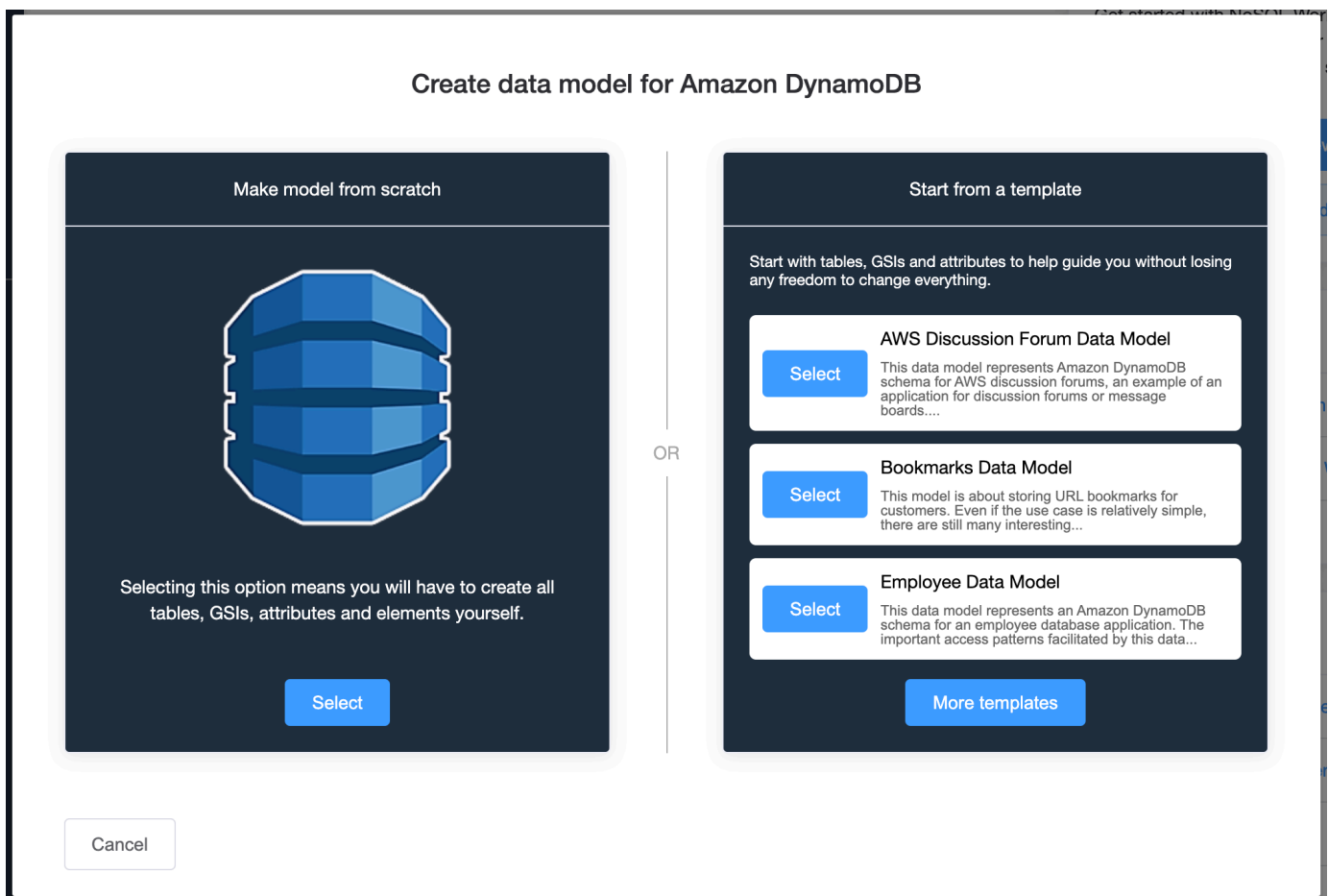
1. 開啟 NoSQL Workbench，然後在左側的導覽窗格中，選擇 Data modeler (資料模型建立工具) 圖示。



2. 選擇 Create data model (建立資料模型)。

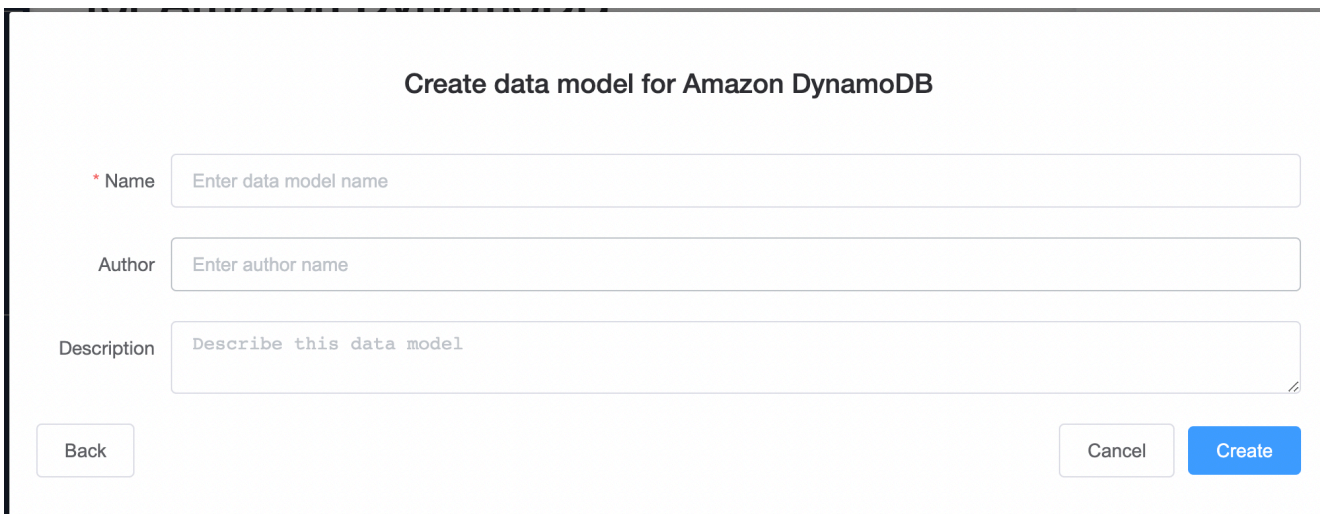


Create data model (建立資料模型) 有兩種選擇：從頭開始建立模型或從模板建立。



### Make model from scratch

若要從頭開始建立模型，請輸入資料模型的名稱、作者和說明。完成時，請選擇 Create (建立)。



**Create data model for Amazon DynamoDB**

\* Name

Author

Description

### Start from a template

從範本開始可讓您選擇要從範本開始建立的範例模型。選擇 **More templates (更多範本)**，查看更多範本選項。選擇 **Select (選取)**，瀏覽您要使用的範本。

輸入所選範本的資料模型名稱、作者和說明。您可以選擇 **Schema only (僅結構描述)** 或 **Schema with sample data (有範例資料的結構描述)**。

- **Schema only (僅結構描述)** 使用主索引鍵 (分割區和排序索引鍵) 和其他屬性，建立一個空的資料模型。
- **Schema with sample data (有範例資料的結構描述)** 會建立一個資料模型，其中包含主索引鍵 (分割區和排序索引鍵) 和其他屬性的範例資料。

完成此資訊後，請選擇 **Create (建立)** 來建立模型。

## Create data model for Amazon DynamoDB

Data Model

Template

\* Save as

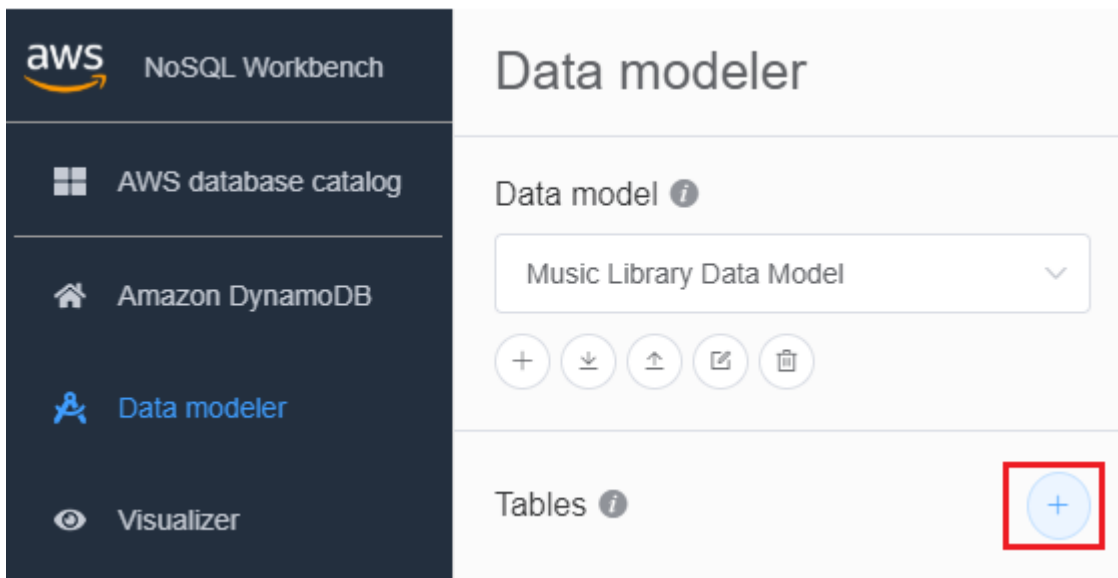
Author

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. 建立模型後，選擇 Add table (新增資料表)。



如需資料表的詳細資訊，請參閱[在 DynamoDB 中使用資料表](#)。

## 4. 指定下列內容：

- Table Name (資料表名稱) – 輸入資料表的唯一名稱。
- 分割區索引鍵：輸入分割區索引鍵名稱並指定其類型。或者，您也可以選擇更精細的資料類型格式來產生範例資料。
- 若要新增排序索引鍵：
  1. 選取 Add sort key (新增排序索引鍵)。
  2. 指定排序索引鍵名稱及其類型。或者，您可以選擇更精細的資料類型格式來產生範例資料。

**Note**

若要深入了解主索引鍵設計、設計、有效使用分割區索引鍵以及使用排序索引鍵，請參閱以下內容：

- [主索引鍵](#)
- [有效設計和使用分區索引鍵的最佳實務](#)
- [使用排序索引鍵以組織資料的最佳實務](#)

5. 若要新增其他屬性，請針對各屬性執行以下作業：
  1. 選擇新增屬性。
  2. 指定屬性名稱和其類型。或者，您可以選擇更精細的資料類型格式來產生範例資料。
6. 新增面向：

您可以選擇性地新增面向。面向是 NoSQL Workbench 中的虛擬建構模組。它不是 DynamoDB 本身中的功能建構模組。

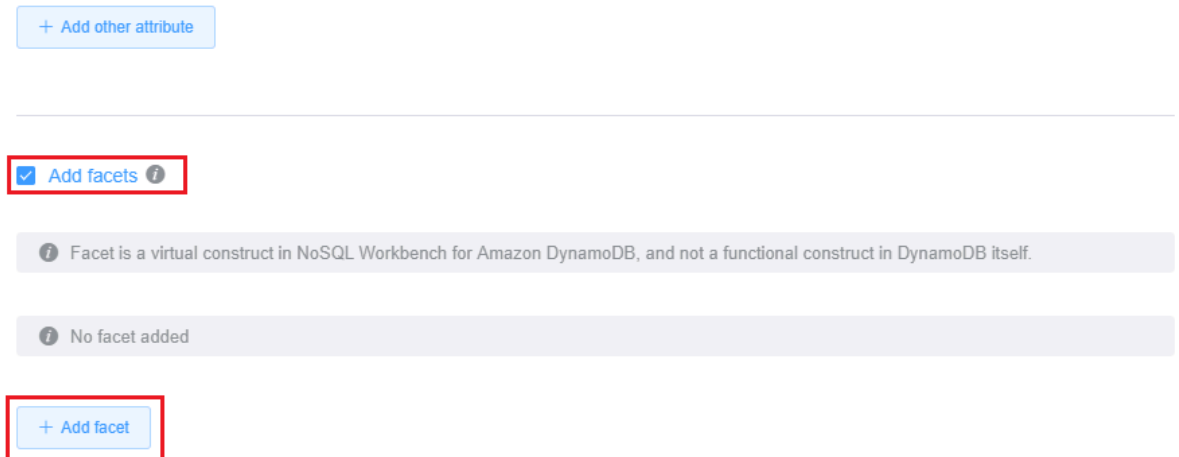
**Note**

NoSQL Workbench 中的面向可協助您針對 Amazon DynamoDB，使用資料表中的資料子集將應用程式的不同資料存取模式視覺化。如需進一步了解面向，請參閱 [檢視資料存取模式](#)。

若要新增面向，

- 選取 Add facets (新增面向)。

- 選擇 Add facet (新增面向)。



- 指定下列內容：
  - Facet name (面向名稱)。
  - 分割區索引鍵別名有助於分辨此面向視圖。
  - Sort key alias (排序鍵別名)。
  - 選擇屬於此面向的 Other attributes (其他屬性)。

選擇 Add facet (新增面向)。



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  ⓘ

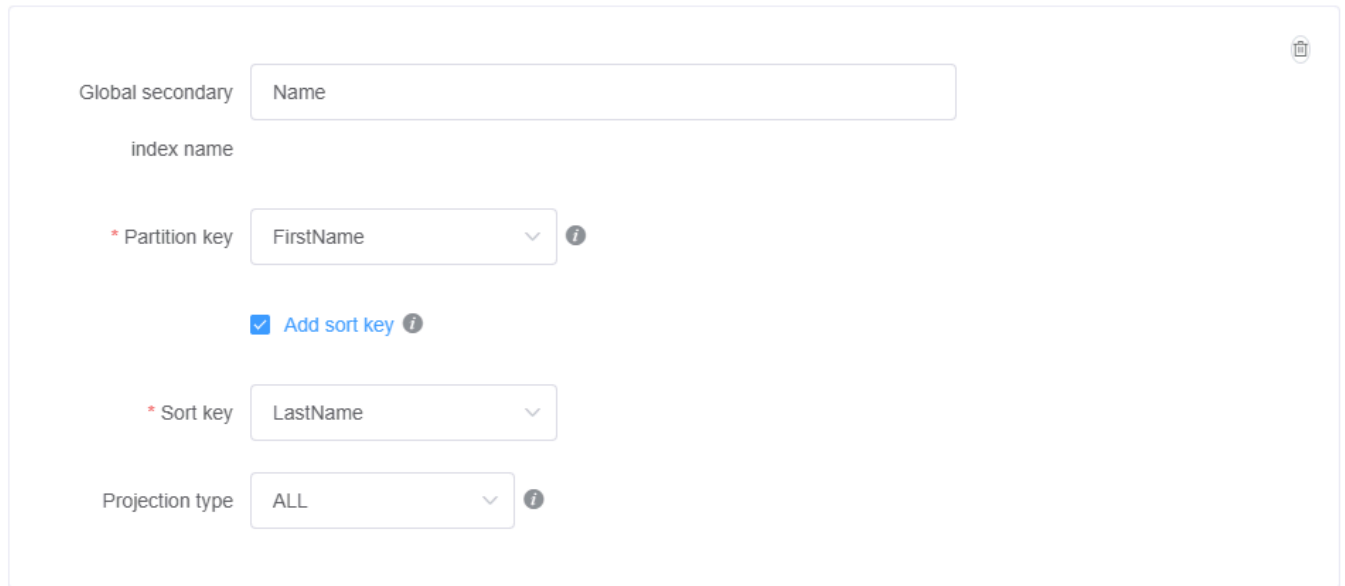
Other attributes  ⓘ

若您想新增更多面向，請重複執行此步驟。

7. 若要新增全域次要索引，請選擇 Add global secondary index (新增全域次要索引)。

指定 Global secondary index name (全域次要索引名稱)、Partition key (分割區索引鍵) 屬性和 Projection type (投射類型)。

## Global secondary indexes

[+ Add global secondary index](#)

如需在 DynamoDB 中使用全域次要索引的詳細資訊，請參閱[全域次要索引](#)。

- 根據預設，您的資料表將使用佈建的容量模式，並對讀取和寫入容量啟用 Auto Scaling 功能。若想變更這些設定，請取消勾選容量設定下的從基底資料表繼承容量設定。

選取所需的容量模式、讀取和寫入容量以及自動調整規模 IAM 角色 (如果適用)。

如需 DynamoDB 容量設定的詳細資訊，請參閱[輸送量容量](#)。

- 儲存對資料表設定所做的編輯。

[Cancel](#)[Save edits](#)

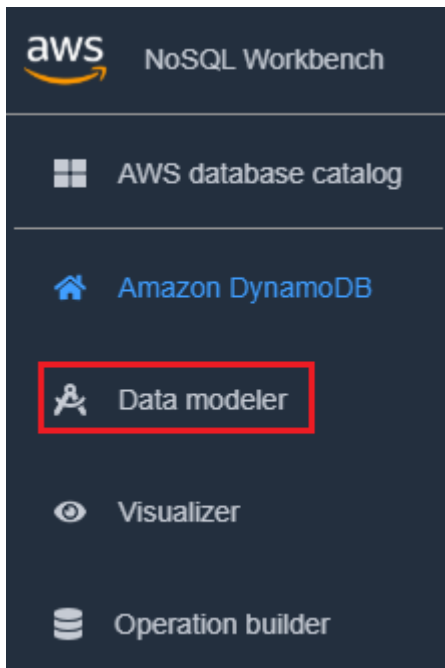
如需有關 CreateTable API 作業的詳細資訊，請參閱 Amazon DynamoDB API 參考[CreateTable](#)中的。

## 匯入現有的資料模型

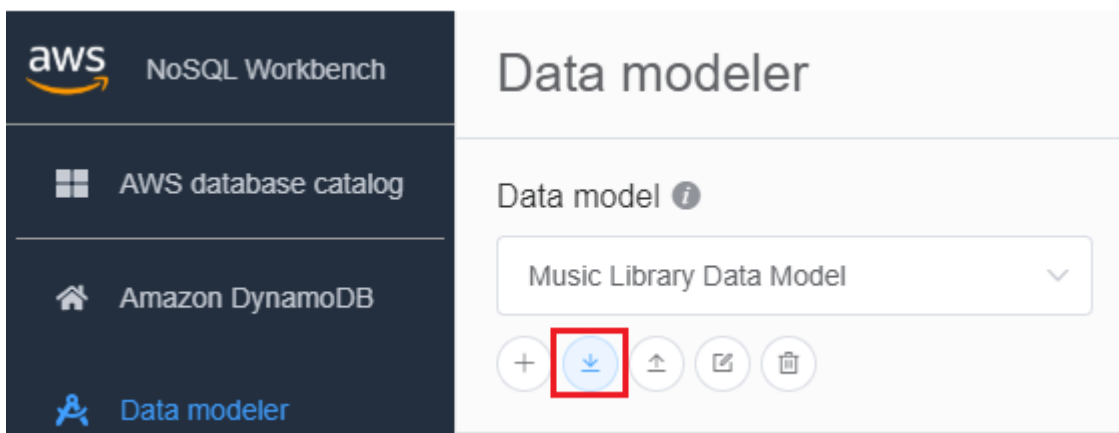
您可以使用 Amazon DynamoDB 專用 NoSQL Workbench，藉由匯入與修改現有的模型來建立資料模型。您能夠以 NoSQL Workbench 模型格式或 [AWS CloudFormation JSON 範本格式](#) 匯入資料模型。

## 匯入資料模型

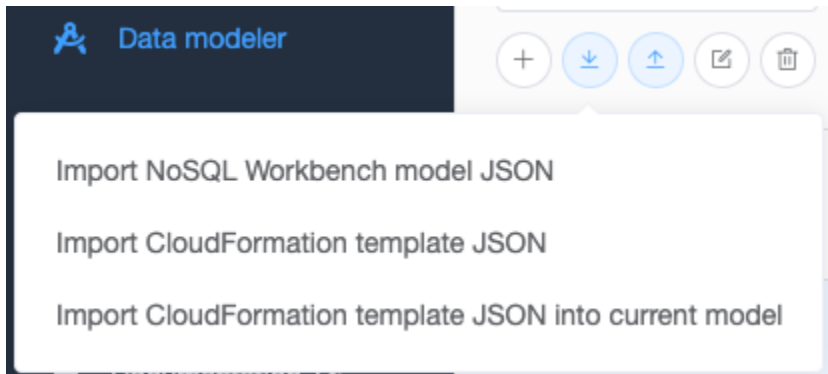
1. 在 NoSQL Workbench 的左側導覽窗格中，選擇 Data modeler (資料模型建立工具) 圖示。



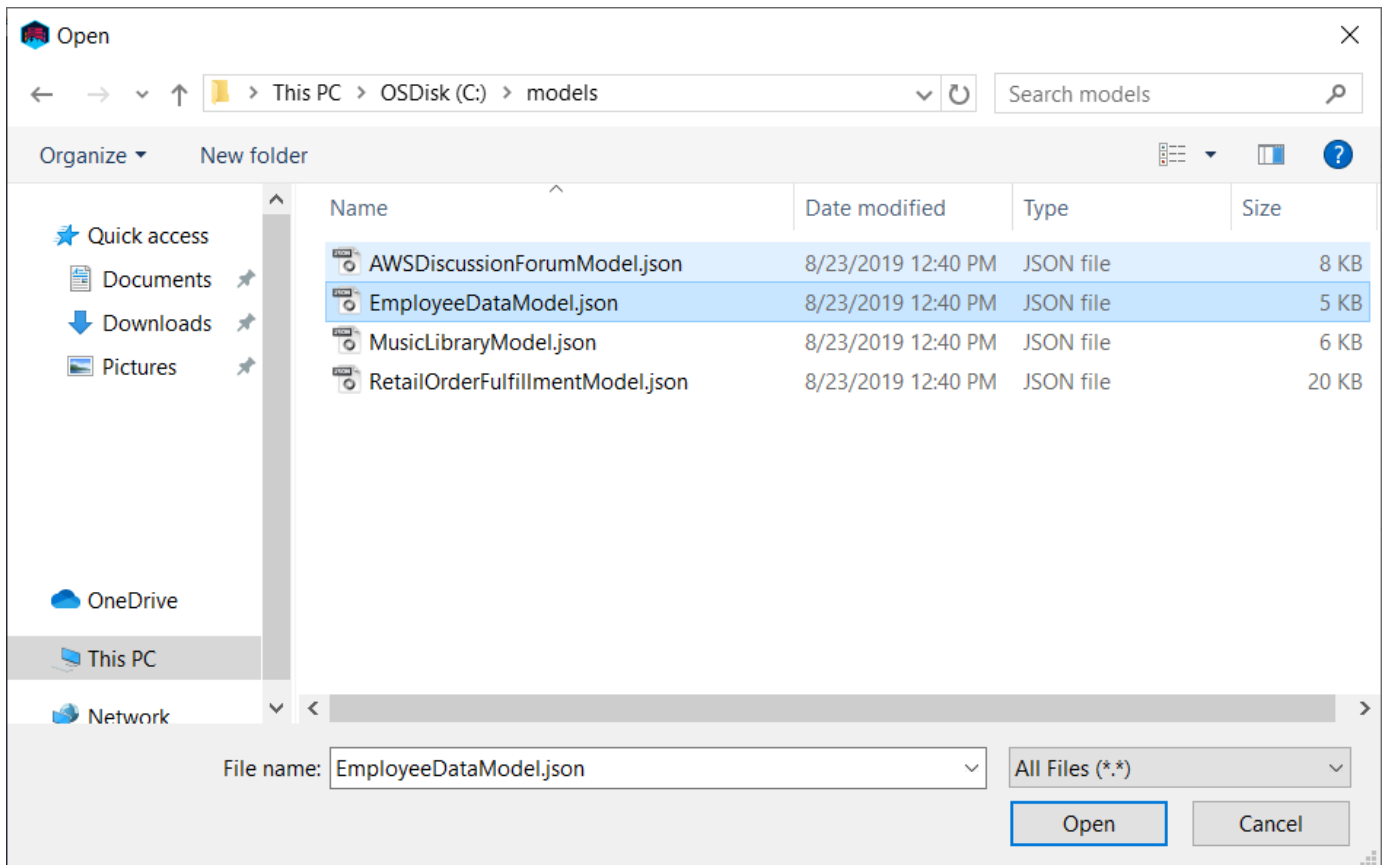
2. 將指標停留在 Import data model (匯入資料模型)。



在下拉式清單中，選擇您要匯入的模型是否為 NoSQL 工作台模型格式或 CloudFormation JSON 範本格式。如果您在 NoSQL 工作台中打開了現有的數據模型，則可以選擇將模 CloudFormation 板導入當前模型。



### 3. 選擇要匯入的模型。



4. 如果您要匯入的模型是 CloudFormation 範本格式，您會看到要匯入的表格清單，並有機會指定資料模型名稱、作者和說明。

## Create data model for Amazon DynamoDB

**i** 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

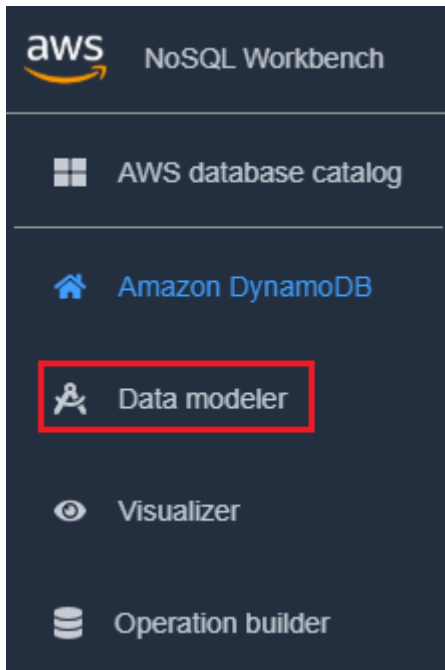
Create

## 匯出資料模型

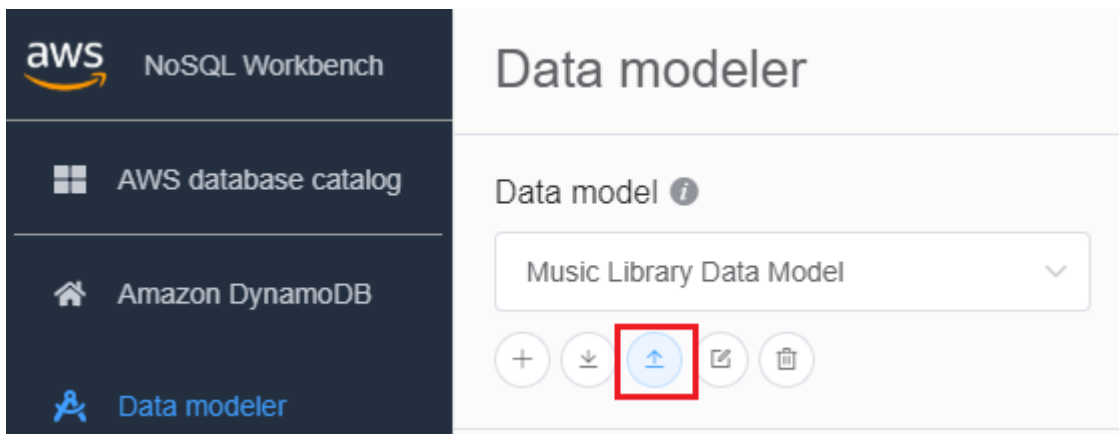
在使用 Amazon DynamoDB 專用 NoSQL Workbench 建立資料模型後，即可以 NoSQL Workbench 模型格式或 [AWS CloudFormation JSON 範本格式](#) 來儲存和匯出該模型。

### 匯出資料模型

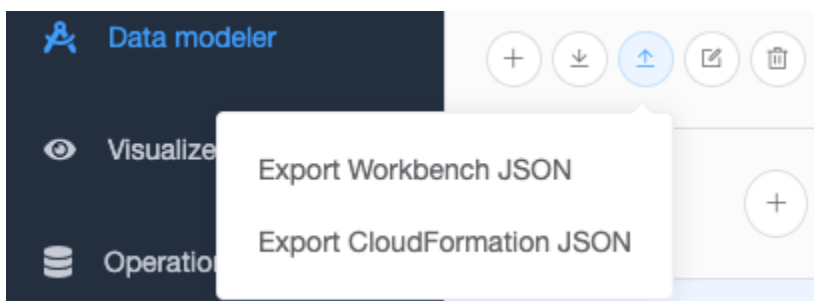
1. 在 NoSQL Workbench 的左側導覽窗格中，選擇 Data modeler (資料模型建立工具) 圖示。



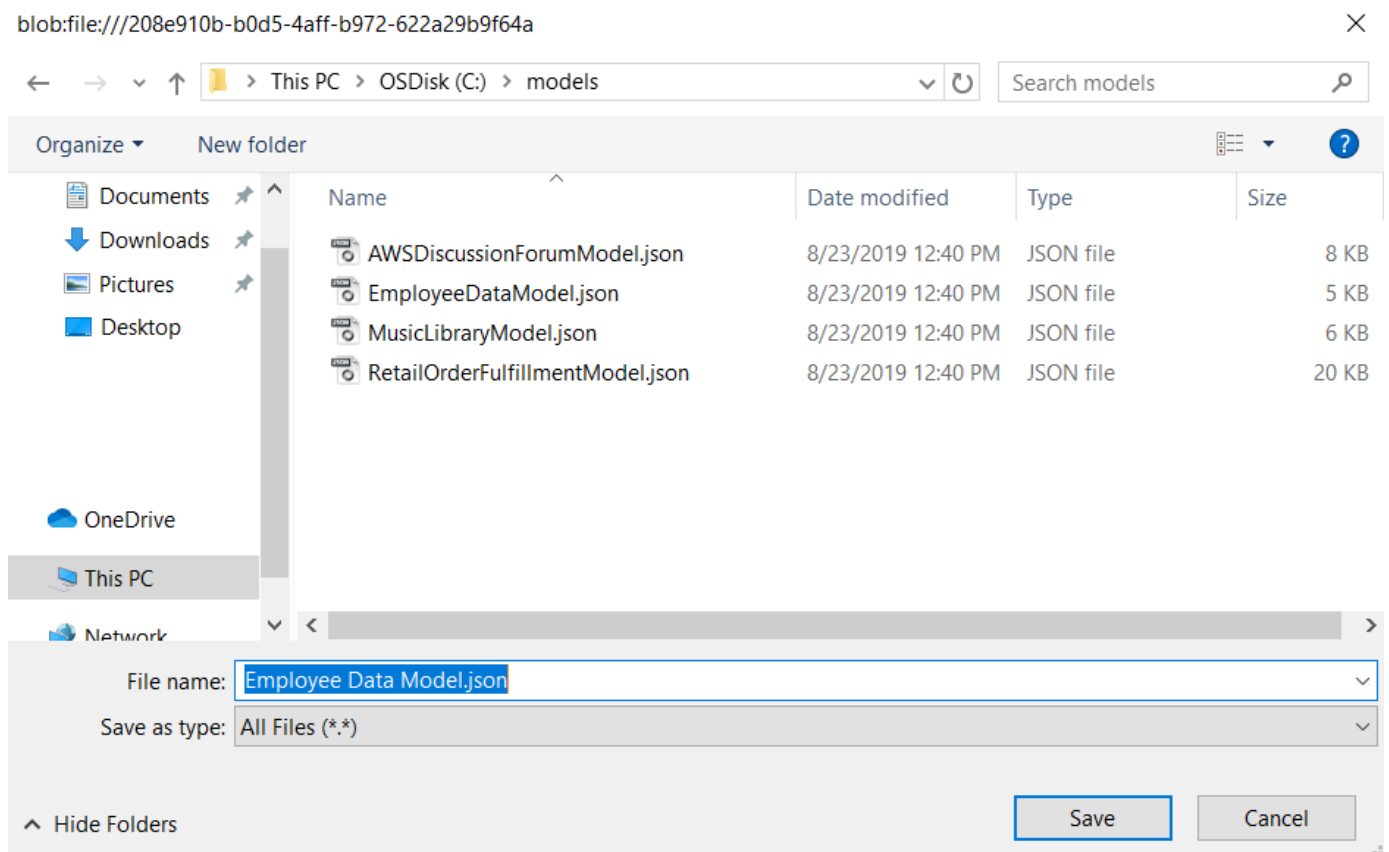
2. 將指標停留在 Export data model (匯出資料模型)。



在下拉式清單中，選擇是以 NoSQL 工作台模型格式還是 CloudFormation JSON 範本格式匯出資料模型。



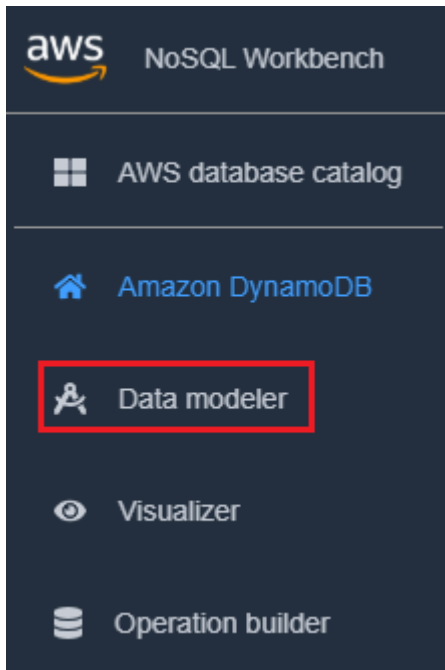
3. 選擇儲存模型的位置。



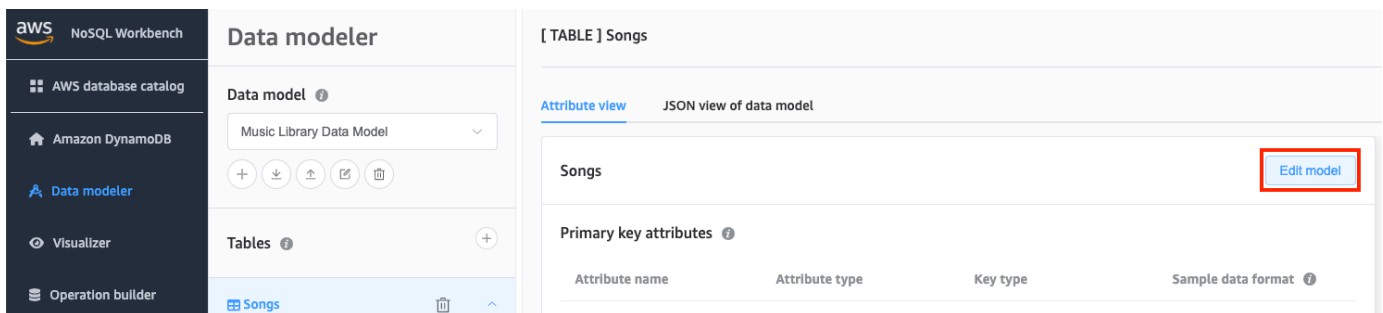
## 編輯現有的資料模型

### 編輯現有的使用者

1. 在 NoSQL Workbench 的左側導覽窗格中，選擇 Data modeler (資料模型建立工具) 按鈕。



2. 選取資料模型，然後選擇要編輯的資料表。選擇編輯模型



3. 執行所需編輯，然後選擇 Save edits (儲存編輯)。

手動編輯現有的模型並新增面向

1. 匯出您的模型。如需詳細資訊，請參閱 [匯出資料模型](#)。
2. 在編輯器中開啟匯出的檔案。
3. 找到您想要建立面向的資料表 DataModel 物件。

新增代表資料表所有面向的 TableFacets 陣列。

針對每個面向新增一個物件至 TableFacets 陣列。每個陣列元素都具有下列屬性：

- FacetName – 面向的名稱。此值在整個模型中必須是唯一的。



- **PartitionKeyAlias** – 資料表分割區索引鍵的易記名稱。當您在 NoSQL Workbench 中檢視面向時會顯示此別名。
- **SortKeyAlias** – 資料表排序索引鍵的易記名稱。當您在 NoSQL Workbench 中檢視面向時會顯示此別名。如果資料表未定義任何排序索引鍵，即不需要此屬性。
- **NonKeyAttributes** – 存取模式所需的屬性名稱陣列。這些名稱必須與針對此資料表定義的屬性名稱相符。

```
{
 "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. 您現在可以將修改過的模型匯入 NoSQL Workbench。如需詳細資訊，請參閱 [匯入現有的資料模型](#)。

# 視覺化資料存取模式

您可以使用 Amazon DynamoDB 專用 NoSQL Workbench 的視覺化工具功能，在應用程式中映射查詢及視覺化不同的存取模式 (稱之為面向)。每個面向都會對應 DynamoDB 中不同的存取模式。您也可以將資料手動新增至資料模型，或從 MySQL 匯入資料。

## 主題

- [將範例資料新增至資料模型](#)
- [從 CSV 檔案匯入範例資料](#)
- [檢視資料存取模式](#)
- [使用彙總檢視在資料模型中檢視所有資料表](#)
- [向 DynamoDB 遞交資料模型](#)

## 將範例資料新增至資料模型

將範例資料新增至模型，可讓您在視覺化模型及其各種資料存取模式 (或面向) 時顯示資料。

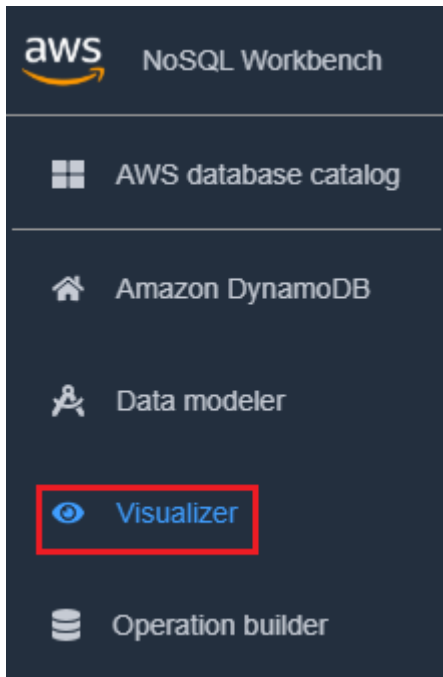
有兩種方法可以新增範例資料。一種是使用我們的範例資料自動產生工具。另一種是一次新增一項資料。

使用 Amazon DynamoDB 專用 NoSQL Workbench，依照這些步驟將範例資料新增至資料模型。

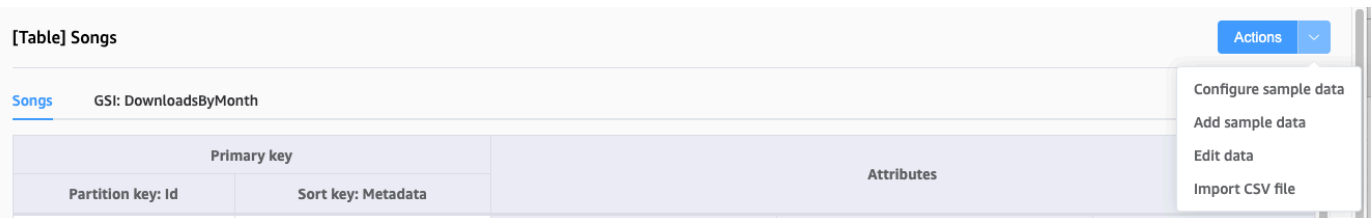
### 若要自動產生範例資料

自動產生範例資料可協助您產生 1 到 5000 列的資料，供您立即使用。您可以指定精細的範例資料類型，以根據您的設計和測試需求建立真實的資料。若要利用此功能產生真實的資料，您需要在資料模型建立工具中指定屬性的範例資料類型格式。如需指定範例資料類型格式，請參閱 [建立新的資料模型](#)。

1. 在左側的導覽窗格中，選擇 visualizer (視覺化工具) 圖示。



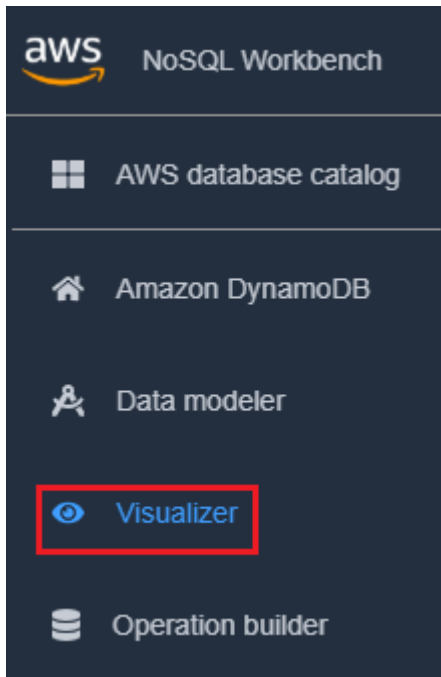
2. 在視覺化工具中，選取資料模型並選擇資料表。
3. 選擇動作下拉式清單，並選取新增範例資料。



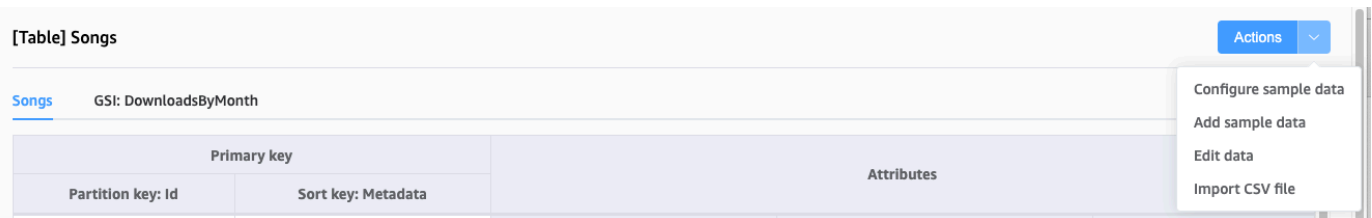
4. 輸入您要產生的範例資料項目數量，然後選取確認。

若要一次新增一項範例資料

1. 在左側的導覽窗格中，選擇 visualizer (視覺化工具) 圖示。



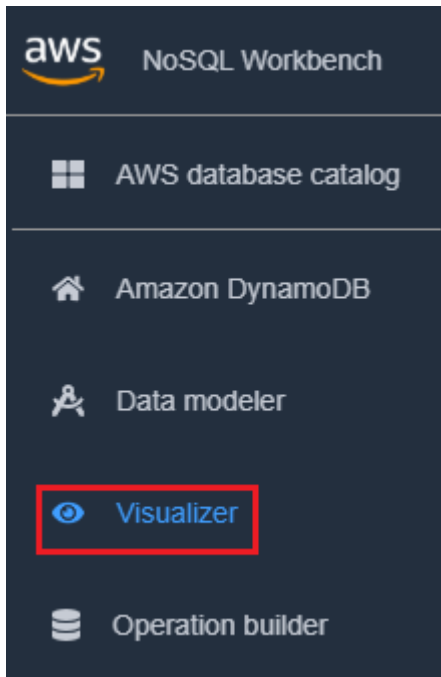
2. 在視覺化工具中，選取資料模型並選擇資料表。
3. 選擇動作下拉式清單，並選取編輯資料。



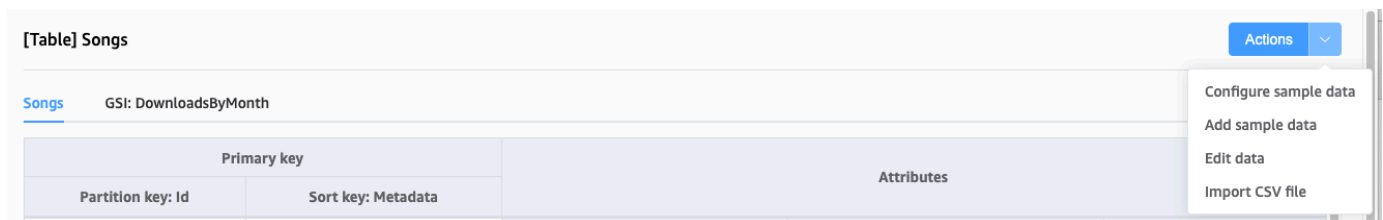
4. 選擇新增新列。將範例資料輸入空白的文字方塊，然後再次選擇新增新列以新增其他列。完成後，選擇儲存變更。

若要刪除範例資料

1. 在左側的導覽窗格中，選擇 visualizer (視覺化工具) 圖示。



2. 在視覺化工具中，選取資料模型並選擇資料表。
3. 選擇動作下拉式清單，並選取編輯資料。



4. 選取您要刪除的每一列資料旁的刪除圖示。

## 從 CSV 檔案匯入範例資料

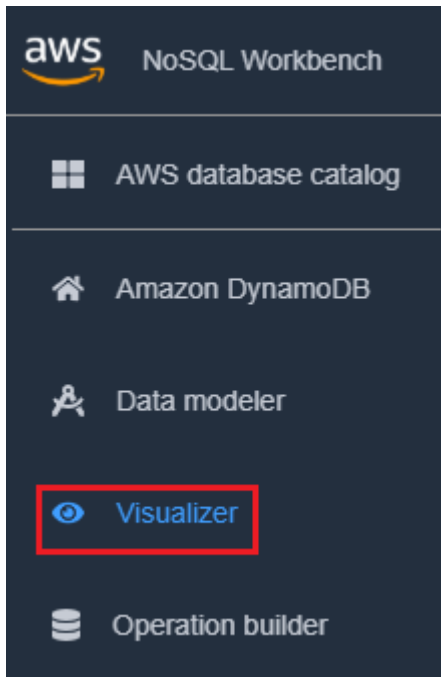
如果您有 CSV 檔案預先存在範例資料，您可以將其匯入。這使您可以快速填入模型取樣資料，而無需逐行輸入。

CSV 檔案中的欄名稱必須與資料模型中的屬性名稱相符，但不需要以相同的順序排列。例如，如果您的資料模型具有名為LoginAlias、FirstName，以及LastName，您的 CSV 列可以是LastName、FirstName，以及LoginAlias。

從 CSV 檔案匯入的資料一次限制為 150 列。

若要從 CSV 檔案匯入資料至 NoSQL 工作台

1. 在左側的導覽窗格中，選擇 visualizer (視覺化工具) 圖示。



2. 在視覺化工具中，選取資料模型並選擇資料表。
3. 選擇動作下拉式清單，並選取編輯資料。
4. 再次選擇動作下拉式清單，並選取匯入 CSV 檔案。
5. 選取 CSV 檔案，然後選擇 Open (開啟)。CSV 檔案中的資料會附加至您的資料表。

#### Note

如果您的 CSV 檔案包含與資料表中已有項目具有相同索引鍵的一或多列，您可以選擇覆寫現有項目或將它們附加到資料表結尾。如果您選擇附加項目，字尾「-Copy」會新增至每個重複項目的索引鍵，以區分重複項目與資料表中已有的項目。

## 檢視資料存取模式

在 NoSQL Workbench 中，面向代表適用於 Amazon DynamoDB 的應用程式不同資料存取模式。當一個排序索引鍵代表多個資料類型時，面向可協助您將資料模型視覺化。面向為您提供了一種查看資料表中的資料子集的方法，而無需查看不符合面向限制的記錄。面向是一種視覺資料建模工具，在 DynamoDB 中不作為可用的建構存在，因為其純粹用於輔助存取模式建模。

若要查看面向的範例，您可以匯入我們的其中一個包含面向的範例資料模型，作為資料模型範本。

## 匯入資料模型範例

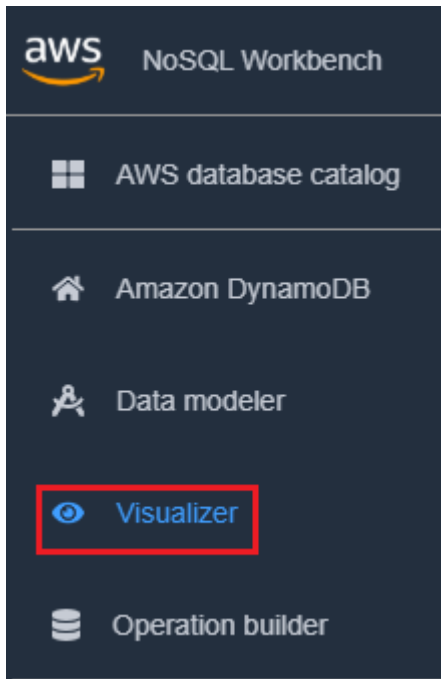
1. 在左側,選擇 Amazon DynamoDB。
2. 在 Sample data models (資料模型範例) 區段中，將指標暫留在 Music Library Data Model (音樂庫資料模型) 上，然後選擇 Import (匯入)。

The screenshot shows the AWS NoSQL Workbench interface. On the left is a dark sidebar with navigation options: 'AWS database catalog', 'Amazon DynamoDB' (highlighted with a red box), 'Data modeler', 'Visualizer', 'Operation builder', 'Documentation', and 'Email us'. The main area displays a table of 'Sample data models' with columns for 'Data model name' and 'Skill level'. The 'Music Library Data Model' row is selected, and its 'Import' button is highlighted with a red box.

Data model name	Skill level
> 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. 在左側的導覽窗格中，選擇 visualizer (視覺化工具) 圖示。





4. 選擇 Songs (歌曲) 資料表並將其展開 您將看到資料的彙總檢視。

 A screenshot of the AWS NoSQL Workbench Visualizer interface. The left sidebar shows the 'Songs' table selected under the 'Visualizer' menu. The main area displays the 'Aggregate view' for the 'Songs' table. The table has a primary key 'Id' and a sort key 'Metadata'. The data is grouped by 'Id'.
 

Primary key		Attributes		
Partition key: Id	Sort key: Metadata			
	Details	Title	Artist	TotalDownloads
		Wild Love	Argyboots	3
	Did-9349823681	DownloadTimestamp		
		2018-01-01T00:00:07		
1	Did-9349823682	DownloadTimestamp		
		2018-01-01T00:01:08		

5. 選擇 Facets (面向) 下拉式選單箭頭，以展開可用的面向。

6. 選擇 SongDetails (歌曲詳細資訊) 面向，以使用套用的歌曲詳細資訊面向將資料視覺化。

 A screenshot of the AWS NoSQL Workbench Visualizer interface. The left sidebar shows the 'Songs' table selected. The main area displays the '[FACET] SongDetails' view. The table has columns for 'SongId (Partition key) : String', 'Metadata (Sort key) : String', 'Title : String', 'Artist : String', and 'TotalDownloads : String'.
 

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

您也可以使用資料模型建立工具來編輯面向定義。如需詳細資訊，請參閱 [編輯現有的資料模型](#)。

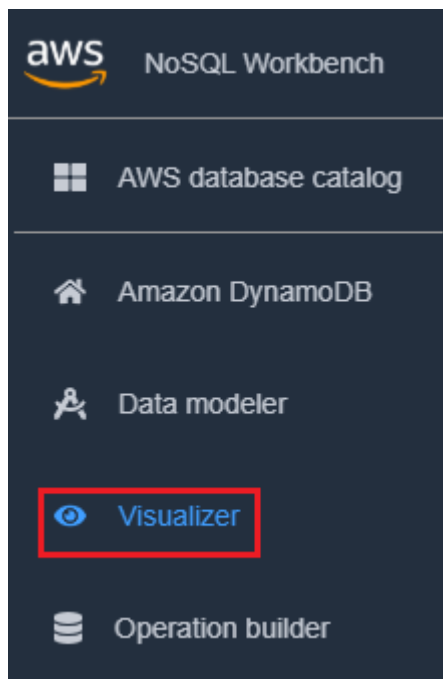
## 使用彙總檢視在資料模型中檢視所有資料表

Amazon DynamoDB 專用 NoSQL Workbench 中的 Aggregate View (彙總檢視) 代表資料模型中的所有資料表。每份資料表都會顯示以下資訊：

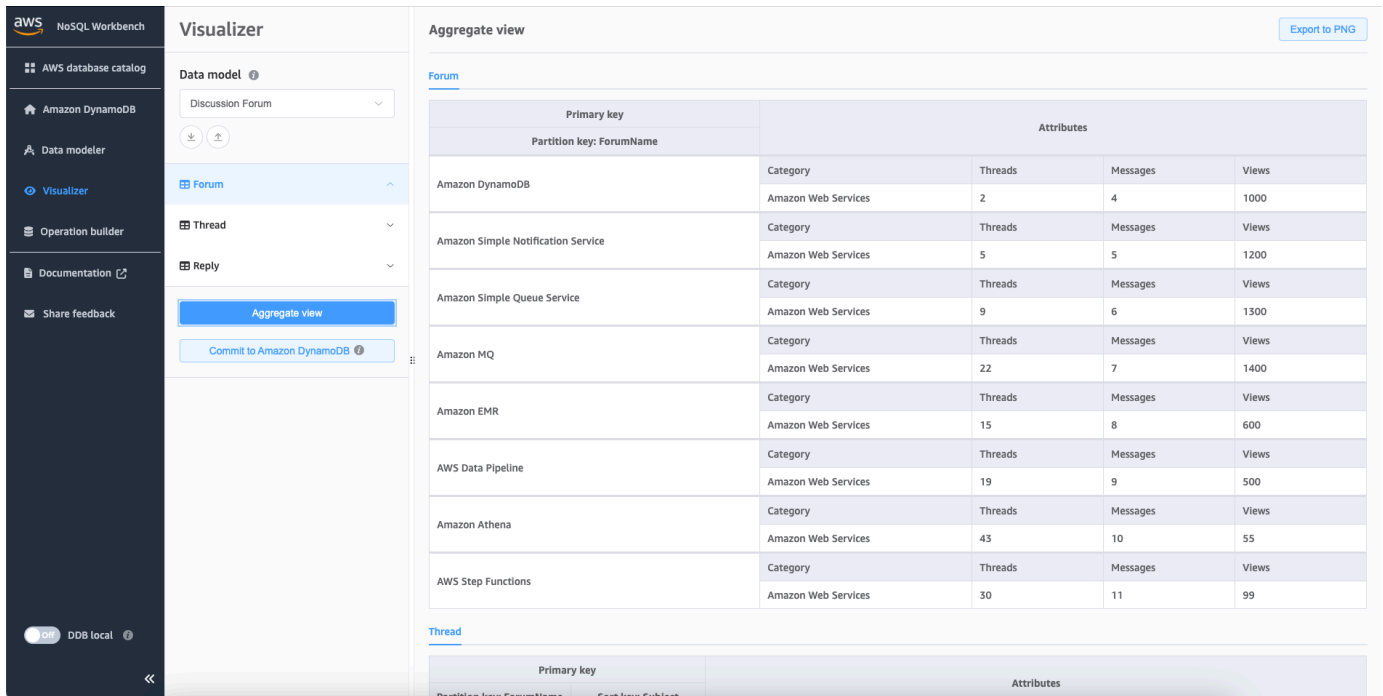
- 資料表欄位名稱
- 範例資料
- 所有與資料表相關聯的全域次要索引。每個索引都會顯示下列資訊：
  - 索引欄位名稱
  - 範例資料

### 檢視所有資料表資訊

1. 在左側的導覽窗格中，選擇 visualizer (視覺化工具) 圖示。



2. 在視覺化工具中，選擇 Aggregate view (彙總檢視)。



Primary key	Attributes			
Partition key: ForumName	Category	Threads	Messages	Views
Amazon DynamoDB	Amazon Web Services	2	4	1000
Amazon Simple Notification Service	Amazon Web Services	5	5	1200
Amazon Simple Queue Service	Amazon Web Services	9	6	1300
Amazon MQ	Amazon Web Services	22	7	1400
Amazon EMR	Amazon Web Services	15	8	600
AWS Data Pipeline	Amazon Web Services	19	9	500
Amazon Athena	Amazon Web Services	43	10	55
AWS Step Functions	Amazon Web Services	30	11	99

## 向 DynamoDB 遞交資料模型

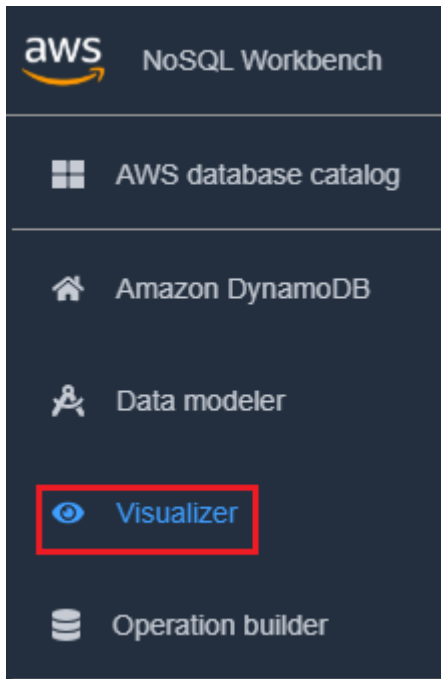
當您滿意資料模型後，即可向 Amazon DynamoDB 遞交模型。

### Note

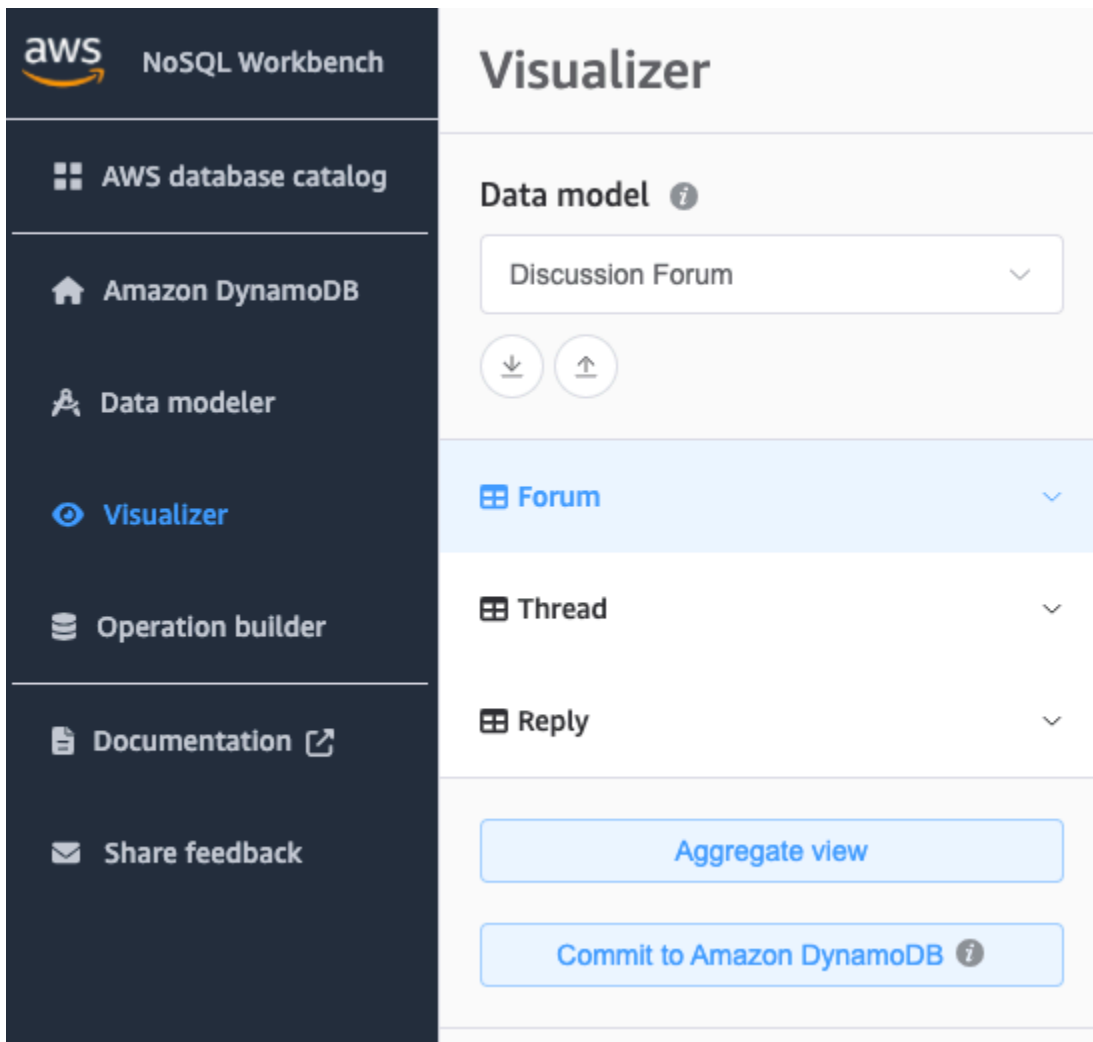
- 此動作會導致針對資料模型中呈現的資料表及全域次要索引，在 AWS 中建立伺服器端資源。
- 使用下列特性建立資料表：
  - Auto scaling 設為目標使用率的 70%。
  - 佈建的容量設為 5 個讀取容量單位和 5 個寫入容量單位。
  - 使用 10 個讀取容量單位和 5 個寫入容量單位的佈建容量建立全域次要索引。

## 向 DynamoDB 遞交資料模型

1. 在左側的導覽窗格中，選擇 visualizer (視覺化工具) 圖示。



2. 選擇 Commit to DynamoDB (遞交給 DynamoDB)。



3. 選擇 Add new remote connection (新增新遠端連線) 標籤，藉以選擇已有的現存連線，或建立新連線。

- 指定下列資訊，新增新的連線：
  - 帳戶別名
  - AWS 區域
  - 存取金鑰 ID
  - 私密存取金鑰

如需如何取得存取金鑰的詳細資訊，請參閱[取得 AWS 存取金鑰](#)。

- 您可以選擇指定下列項目：
  - [Session token \(工作階段字符\)](#)
  - [IAM 角色 ARN](#)

- 如果您不想要註冊免費方案帳戶，且寧可使用 [DynamoDB Local \(可下載版本\)](#)：
  1. 選擇 Add a new DynamoDB local connection (新增 DynamoDB 本機連線) 標籤。
  2. 指定 Connection name (連線名稱) 和 Port (連接埠)。
- 4. 選擇 Commit (遞交)。

#### Note

若您將 DynamoDB 本機版安裝做為 NoSQL Workbench 安裝的一部分，則需使用 NoSQL Workbench 畫面左下角的 DynamoDB local Server (DynamoDB 本機版伺服器)，開啟 DynamoDB 本機版。如需有關切換的詳細資訊，請參閱 [安裝 DynamoDB 專用 NoSQL Workbench](#)。

## 使用 NoSQL Workbench 探索資料集與建立操作

Amazon DynamoDB 專用 NoSQL Workbench 會提供強大的圖形使用者界面來開發及測試查詢。您可以使用 NoSQL Workbench 中的操作建置器來檢視、探索及查詢即時資料集。結構化操作生成器支持投影表達式，條件表達式，並生成多種語言的示例代碼。您可以將表格從一個 Amazon DynamoDB 帳戶直接複製到不同區域中的另一個帳戶。您也可以直接在 DynamoDB 本機和 Amazon DynamoDB 帳戶之間複製表格，以便在開發環境之間更快地複製資料表的金鑰結構描述 (以及選擇性的 GSI 結構描述和項目)。您可以在作業建置器中儲存多達 50 個 DynamoDB 資料作業。

### 主題

- [連線至即時資料集](#)
- [建置複雜的操作](#)
- [使用 NoSQL 工作台複製表格](#)
- [將資料匯出到 CSV 檔案](#)

## 連線至即時資料集

若要使用 NoSQL 工作台連線到 Amazon DynamoDB 表格，您必須先連線到您的帳戶。AWS

### 新增資料庫連線

1. 在 NoSQL Workbench 的左側導覽窗格中，選擇 Operation builder (操作建置器) 圖示。

2. 選擇 Add Connection (新增連線)。
3. 指定下列資訊：
  - Account Alias (帳戶別名)
  - AWS 區域
  - 存取金鑰 ID
  - 私密存取金鑰


如需如何取得存取金鑰的詳細資訊，請參閱[取得 AWS 存取金鑰](#)。

您可以選擇指定下列項目：

- [Session token \(工作階段字符\)](#)
  - [IAM 角色 ARN](#)
4. 選擇 Connect (連線)。

如果您不想要註冊免費方案帳戶，且寧可使用 [DynamoDB Local \(可下載版本\)](#)：

- a. 在連線畫面上選擇 Local (本機) 標籤。
- b. 指定下列資訊：
  - 連線名稱 (Connection name)
  - 連接埠
- c. 選擇 connect (連接) 按鈕。

 Note

若要連線至 DynamoDB 本機，請使用終端機手動啟動 DynamoDB 本機 (請參閱在[電腦上部署本機](#))，或使用 [NoSQL 工作台導覽功能表中的 DDB 本機](#) 切換直接啟動 DynamoDB 本機。確保連線連接埠與您的 DynamoDB 本機連接埠相同。

5. 對已建立的連線選擇 Open (開啟)。

連線至 DynamoDB 資料庫後，左窗格即會顯示可用的資料表清單。選擇資料表之一會傳回資料表所存放資料的範例。

您現在可以針對選取的資料表執行隨機操作查詢。

若要在表格上執行查詢，請參閱〈[建築作業](#)〉的下一節，請參閱〈[建置複雜的操作](#)〉。

## 建置複雜的操作

Amazon DynamoDB 專用 NoSQL Workbench 中的操作建置器提供視覺化界面，可讓您在其中執行複雜的資料平面操作。它包括對投射表達式和條件表達式的支援。建立操作後，您可以儲存操作以供日後使用 (最多可儲存 50 個操作)。然後，您可以在 Saved Operations (已儲存操作) 選單中瀏覽常用的資料平面操作清單，再使用它們來自動填入和建置新操作。您也可以使用多種語言產生這些操作的範本程式碼。

NoSQL Workbench 支援建置 DynamoDB 專用 [PartiQL](#) 陳述式，可讓您使用 SQL 相容查詢語言來與 DynamoDB 互動。NoSQL Workbench 也支援建置 DynamoDB CRUD API 操作。

若要使用 NoSQL Workbench 來建立操作，請在左側導覽窗格中選擇 Operation builder (操作建置器) 圖示。

主題

- [建置 PartiQL 陳述式](#)
- [建置 API 操作](#)

## 建置 PartiQL 陳述式

若要使用 NoSQL 工作台為 [DynamoDB 陳述式建立部分](#) SQL，請選擇靠近 NoSQL 工作台使用者介面頂端的 PartiQL 編輯器。

您可以在操作建置器中建立以下 PartiQL 陳述式類型。

主題

- [單一陳述式](#)
- [交易](#)
- [批次](#)

## 單一陳述式

若要執行或產生 PartiQL 陳述式的程式碼，請執行以下操作。



1. 選擇視窗頂端附近的 PartiQL 編輯器。
2. 輸入有效的 [PartiQL statement](#) (PartiQL 陳述式)。
3. 如果您的陳述式使用參數：
  - a. 選擇 Optional request parameters (選用的請求參數)。
  - b. 選擇 Add new parameters (新增新參數)。
  - c. 輸入屬性類型和值。
  - d. 如果希望新增其他參數，請重複步驟 b 和 c。
4. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

從顯示的標籤中選取所需的語言。您現在可以複製此程式碼，並使用在您的應用程式中。
5. 若希望立即執行此操作，請選擇 Run (執行)。
6. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)。然後輸入操作的名稱，並選擇 Save (儲存)。

## 交易

若要執行或產生 PartiQL 交易的程式碼，請執行以下操作。

1. 從更多操作下拉列表中選擇部分 QL 交易。
2. 選擇 Add a new statement (新增新陳述式)。
3. 輸入有效的 [PartiQL statement](#) (PartiQL 陳述式)。

### Note

同一個 PartiQL 交易請求中不能同時支援讀取和寫入操作。SELECT 陳述式不能與 INSERT、UPDATE 和 DELETE 陳述式存在於同一個請求中。如需詳細資訊，請參閱[使用 DynamoDB 專用 PartiQL 執行交易](#)。

4. 如果您的陳述式使用參數
  - a. 選擇 Optional request parameters (選用的請求參數)。
  - b. 選擇 Add new parameters (新增新參數)。
  - c. 輸入屬性類型和值。
  - d. 如果希望新增其他參數，請重複步驟 b 和 c。

5. 如果希望新增更多陳述式，請重複步驟 2 至 4。
6. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

從顯示的標籤中選取所需的語言。您現在可以複製此程式碼，並使用在您的應用程式中。

7. 若希望立即執行此操作，請選擇 Run (執行)。
8. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)。然後輸入操作的名稱，並選擇 Save (儲存)。

## 批次

若要執行或產生 PartiQL 批次的程式碼，請執行以下操作。

1. 選擇分區批次來自更多操作下拉列表。
2. 選擇 Add a new statement (新增新陳述式)。
3. 輸入有效的 [PartiQL statement](#) (PartiQL 陳述式)。

### Note

同一個 PartiQL 批次請求中不能同時支援讀取和寫入操作，也就是說 SELECT 陳述式不能與 INSERT、UPDATE 和 DELETE 陳述式存在於同一個請求中。不允許對相同項目執行寫入操作。與 BatchGetItem 操作一樣，僅支持單例讀操作。不支援掃描和查詢操作。如需詳細資訊，請參閱 [使用 DynamoDB 專用 PartiQL 執行批次操作](#)。

4. 如果您的陳述式使用參數：
  - a. 選擇 Optional request parameters (選用的請求參數)。
  - b. 選擇 Add new parameters (新增新參數)。
  - c. 輸入屬性類型和值。
  - d. 如果希望新增其他參數，請重複步驟 b 和 c。
5. 如果希望新增更多陳述式，請重複步驟 2 至 4。
6. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

從顯示的標籤中選取所需的語言。您現在可以複製此程式碼，並使用在您的應用程式中。

7. 若希望立即執行此操作，請選擇 Run (執行)。
8. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)。然後輸入操作的名稱，並選擇 Save (儲存)。

## 建置 API 操作

若要使用 NoSQL 工作台建置 DynamoDB CRUD API，請從 NoSQL 工作台使用者介面左側選取作業產生器。

然後選取 [開啟] 並選擇連線。

您可以在操作建置器中執行以下操作。

- [刪除資料表](#)
- [建立資料表](#)
- [更新資料表](#)
  
- [放置項目](#)
- [更新項目](#)
- [刪除項目](#)
- [查詢](#)
- [掃描](#)
- [交易取得項目](#)
- [交易寫入項目](#)

### 刪除資料表

若要執行 Delete Table 業，請執行下列動作。

1. 從「表格」區段中找到您要刪除的表格。
2. 從水平省略符號功能表中選取刪除表格。
3. 輸入表格名稱以確認要刪除表格。
4. 選取刪除。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的[刪除資料表](#)。

### 刪除 GSI

若要執行 Delete GSI 業，請執行下列動作。

1. 從 [表格] 區段中找到您要刪除之資料表的 GSI。
2. 從水平省略符號功能表中選取 [刪除 GSI]。
3. 輸入 GSI 名稱以確認您要刪除 GSI。
4. 選取刪除。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的[刪除資料表](#)。

## 建立資料表

若要執行作Create Table業，請執行下列動作。

1. 選擇 [表格] 區段旁邊的 + 圖示。
2. 輸入所需的資料表名稱。
3. 建立分割區索引鍵。
4. 選用性：建立排序索引鍵。
5. 若要自訂容量設定，請取消勾選 [使用預設容量設定] 旁邊的方塊。
  - 您現在可以選取 Provisioned (已佈建的) 或 On-demand capacity (隨需容量)。若選取 Provisioned (已佈建的)，您可以設定最小和最大讀寫容量單位。您還可以啟用或停用自動調整規模。
  - 如果表格目前設定為隨選，您將無法指定佈建的輸送量。
  - 如果您從隨需切換到佈建輸送量，則自動調度資源將自動套用至所有 GSI，最小值：1，最大值：10；目標：70%。
6. 選取略過 GSI 並建立以建立不含 GSI 的此表格。或者，您可以選取「下一步」以使用此新表格建立 GSI。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的[建立資料表](#)。

## 建立 GSI

若要執行作Create GSI業，請執行下列動作。

1. 尋找您要新增 GSI 的資料表。
2. 從水平省略符號功能表中，選取建立 GSI。
3. 在索引名稱下命名您的 GSI。

4. 建立分割區索引鍵。
5. 選用性：建立排序索引鍵。
6. 從下拉式清單中選擇投影類型選項。
7. 選取「建立 GSI」。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的[建立資料表](#)。

### 更新資料表

若要使用操作更新表格的容量設定，請執行以下 Update Table 操作。

1. 尋找您要更新容量設定的表格。
2. 從水平省略符號功能表中，選取 [更新容量設定]。
3. 選取已佈建或隨選容量。

選取佈建後，您可以設定最小和最大讀取和寫入容量單位。您還可以啟用或停用自動調整規模。

4. 選取 Update (更新)。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的[更新資料表](#)。

### 更新 GSI

要使用操作更新 GSI 的容量設置，請執行以下 Update Table 操作。

#### Note

依預設，全域次要索引會繼承基底資料表的容量設定。只有當基礎表格處於佈建容量模式時，全域次要索引才能有不同容量模式。當您對佈建模式資料表建立全域次要索引時，必須為該索引的預期工作負載指定讀取與寫入容量單位。如需詳細資訊，請參閱[全域次要索引的佈建輸送量考量](#)。

1. 找到您要更新容量設定的 GSI。
2. 從水平省略符號功能表中，選取 [更新容量設定]。
3. 您現在可以選取 Provisioned (已佈建的) 或 On-demand capacity (隨需容量)。

選取佈建後，您可以設定最小和最大讀取和寫入容量單位。您還可以啟用或停用自動調整規模。

#### 4. 選取 Update (更新)。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的[更新資料表](#)。

##### 放置項目

您可以使用 Put Item 操作來創建某個物件。若要執行或產生 Put Item 操作的程式碼，請執行以下操作。

1. 查找您要在其中創建物件的表格。
2. 從「操作」下拉列表中選擇「創建物件」。
3. 輸入分割區索引鍵值。
4. 輸入排序索引鍵值 (如有)。
5. 若想要新增非索引鍵屬性，請執行以下作業：
  - a. 選取 [+ 新增其他屬性]。
  - b. 指定 Attribute name (屬性名稱)、Type (類型) 和 Value (值)。
6. 如果必須滿足某項條件表達式，Put Item 操作才能成功，請執行以下作業：
  - a. 選擇 Condition (條件)。
  - b. 指定屬性名稱、比較運算子、屬性類型和屬性值。
  - c. 如需其他條件，請再次選擇 Condition (條件)。

如需詳細資訊，請參閱 [條件表達式](#)。

7. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

從顯示的標籤中選取所需的語言。您現在可以複製此程式碼，並使用在您的應用程式中。

8. 若希望立即執行此操作，請選擇 Run (執行)。
9. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)，然後輸入操作的名稱，再選擇 Save (儲存)。

如需有關此作業的詳細資訊，請參閱 Amazon DynamoDB API 參考 [PutItem](#) 中的。

##### 更新項目

若要執行或產生 Update Item 操作的程式碼，請執行以下操作：

1. 尋找您要在其中更新項目的表格。
2. 選取項目。
3. 輸入所選表達式的屬性名稱和屬性值。
4. 如果您要新增更多運算式，請在 [更新運算式] 下拉式清單中選擇另一個運算式，然後選取 + 圖示。
5. 如果必須滿足某項條件表達式，Update Item 操作才能成功，請執行以下作業：
  - a. 選擇 Condition (條件)。
  - b. 指定屬性名稱、比較運算子、屬性類型和屬性值。
  - c. 如需其他條件，請再次選擇 Condition (條件)。

如需詳細資訊，請參閱 [條件表達式](#)。

6. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

選擇您想要的語言標籤。您現在可以複製此程式碼，並使用在您的應用程式中。
7. 若希望立即執行此操作，請選擇 Run (執行)。
8. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)，然後輸入操作的名稱，再選擇 Save (儲存)。

如需有關此作業的詳細資訊，請參閱 Amazon DynamoDB API 參考 [UpdateItem](#) 中的。

## 刪除項目

若要執行 Delete Item 業，請執行下列動作。

1. 尋找您要在其中刪除項目的表格。
2. 選取項目。
3. 從動作下拉式清單中，選取刪除項目。
4. 選取「刪除」(Delete) 以確認您要刪除該項目。

如需有關此作業的詳細資訊，請參閱 Amazon DynamoDB API 參考 [DeleteItem](#) 中的。

## 複製項目

您可以透過建立具有相同屬性的新項目來複製項目。若要複製項目，請執行下列操作。

1. 尋找您要在其中複製項目的表格。
2. 選取項目。
3. 從「作業」下拉式清單中選取「複製項目」。
4. 指定新的分割區索引鍵。
5. 指定新的排序關鍵字 (如有必要)。
6. 選取 [執行]。

如需有關此作業的詳細資訊，請參閱 Amazon DynamoDB API 參考 [DeleteItem](#) 中的。

## Query

若要執行或產生 Query 操作的程式碼，請執行以下操作。

1. 從 NoSQL 維護作業使用者介面頂端選取查詢。
2. 指定分割區索引鍵值。
3. 如果 Query 操作需要排序索引鍵：
  - a. 選取 Sort key (排序索引鍵)。
  - b. 指定比較運算子和屬性值。
4. 選取「查詢」以執行此查詢作業。如果需要更多選項，請選中更多選項複選框，然後繼續執行以下步驟。
5. 如果不是所有屬性都應傳回並附帶操作結果，請選取 Projection expression (投射表達式)。
6. 選擇 + 圖示。
7. 輸入附帶查詢結果傳回的屬性。
8. 如需更多屬性，請選擇 +。
9. 如果必須滿足某項條件表達式，Query 操作才能成功，請執行以下作業：
  - a. 選擇 Condition (條件)。
  - b. 指定屬性名稱、比較運算子、屬性類型和屬性值。
  - c. 如需其他條件，請再次選擇 Condition (條件)。

如需詳細資訊，請參閱 [條件表達式](#)。

10. 若要產生程式碼，請選擇 Generate code (產生程式碼)。



選擇您想要的語言標籤。您現在可以複製此程式碼，並使用在您的應用程式中。

11. 若希望立即執行此操作，請選擇 Run (執行)。
12. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)，然後輸入操作的名稱，再選擇 Save (儲存)。

如需此操作的詳細資訊，請參閱《Amazon DynamoDB API 參考》中的 [Query](#)。

## Scan

若要執行或產生 Scan 操作的程式碼，請執行以下操作。

1. 從 NoSQL 工作台使用者介面頂端選取掃描。
2. 選取 [掃描] 按鈕以執行此基本掃描作業。如果需要更多選項，請選中更多選項複選框，然後繼續執行以下步驟。
3. 指定屬性名稱以篩選掃描結果。
4. 如果不是所有屬性都應傳回並附帶操作結果，請選取 Projection expression (投射表達式)。
5. 如果必須滿足某項條件表達式，scan 操作才能成功，請執行以下作業：
  - a. 選擇 Condition (條件)。
  - b. 指定屬性名稱、比較運算子、屬性類型和屬性值。
  - c. 如需其他條件，請再次選擇 Condition (條件)。

如需詳細資訊，請參閱 [條件表達式](#)。

6. 若要產生程式碼，請選擇 Generate code (產生程式碼)。

選擇您想要的語言標籤。您現在可以複製此程式碼，並使用在您的應用程式中。

7. 若希望立即執行此操作，請選擇 Run (執行)。
8. 若想儲存此操作以供日後使用，請選擇 Save operation (儲存操作)，然後輸入操作的名稱，再選擇 Save (儲存)。

## TransactGetItems

若要執行或產生 TransactGetItems 操作的程式碼，請執行以下操作。

1. 從 NoSQL 工作台 UI 頂端的「更多作業」下拉式清單中，選擇 TransactGetItems。

2. 選擇附近的 + 圖標 TransactGetItem。
3. 指定分割區索引鍵。
4. 指定排序鍵 (如有必要)。
5. 選取執行以執行作業，選取儲存作業以儲存作業，或選取產生程式碼以產生程式碼。

如需交易的詳細資訊，請參閱 [Amazon DynamoDB Transactions](#)。

## TransactWriteItems

若要執行或產生 TransactWriteItems 操作的程式碼，請執行以下操作。

1. 從 NoSQL 工作台 UI 頂端的「更多作業」下拉式清單中，選擇 TransactWriteItems。
2. 從「作業」下拉式清單中選擇作業。
3. 選擇附近的 + 圖標 TransactWriteItem。
4. 在「作業」下拉式清單中，選擇您要執行的作業。
  - 如需 DeleteItem，請遵循 [刪除項目](#) 操作的說明。
  - 如需 PutItem，請遵循 [放置項目](#) 操作的說明。
  - 如需 UpdateItem，請遵循 [更新項目](#) 操作的說明。

若要變更動作順序，請在左側清單中選擇動作，然後選擇上下箭號在清單中上下移動該動作。

若要刪除動作，請在清單中選擇動作，然後選擇 Delete (刪除) (回收桶) 圖示。

5. 選取執行以執行作業，選取儲存作業以儲存作業，或選取產生程式碼以產生程式碼。

如需交易的詳細資訊，請參閱 [Amazon DynamoDB Transactions](#)。

## 使用 NoSQL 工作台複製表格

複製資料表會在您的開發環境之間複製資料表的金鑰結構描述 (以及選擇性的 GSI 結構描述和項目)。您可以將 DynamoDB 本機之間的表格複製到 Amazon DynamoDB 帳戶，甚至可以將表格從一個帳戶複製到不同區域中的另一個帳戶，以加快實驗速度。

### 複製表格

1. 在作業產生器中，選取您的連線和區域 (DynamoDB 本機無法使用區域選項)。

2. 連線至 DynamoDB 後，瀏覽您的表格並選取要複製的表格。
3. 從水平省略符號功能表中，選取複製選項。
4. 輸入您的克隆目標詳細信息：
  - a. 選取連線。
  - b. 選取一個區域 (區域不適用於 DynamoDB 本機)。
  - c. 輸入新表格名稱。
  - d. 選擇翻製選項：
    - i. 依預設，金鑰結構描述處於選取狀態，且無法取消選取。默認情況下，克隆表將復制您的主鍵和排序鍵 (如果可用)。
    - ii. 如果要複製的資料表具有 GSI，則預設會選取 GSI 結構描述。克隆表將復制您的 GSI 主鍵和排序鍵 (如果可用)。您可以選擇取消選取 GSI 結構描述，以略過複製 GSI 結構描述。複製資料表會將基礎資料表的容量設定複製為 GSI 的容量設定。複製完成後，您可以使用作業建置器中的作業來更新資料表的 GSI 容量設定。UpdateTable
5. 輸入要複製的項目數。若只要複製金鑰結構描述和選擇性地複製 GSI 結構描述，您可以將要複製的項目值保持在 0。可複製的項目數目上限為 5000 個。
6. 選擇容量模式：
  - a. 依預設會選取隨選模式。DynamoDB 隨需提供讀取和寫入請求的 pay-per-request 定價，因此您只需按使用量付費。若要深入了解，請參閱 [DynamoDB 隨選模式](#)。
  - b. 佈建模式可讓您指定應用程式所需的每秒讀取和寫入次數。您可以使用自動調整規模來自動調整資料表的佈建容量，以回應流量的變動。若要深入了解，請參閱 [DynamoDB 佈建模式](#)。
7. 選取複製以開始複製。
8. 克隆過程將在後台運行。複製表格狀態發生變更時，作業建置器索引標籤會顯示通知。您可以選取「作業建置器」頁籤，然後選取箭頭按鈕來存取此狀態。箭頭按鈕位於選單側邊欄底部附近的複製表格狀態 Widget 上。

## 將資料匯出到 CSV 檔案

您可以從作業建置器將查詢的結果匯出到 CSV 檔案。這可讓您將資料載入試算表或使用偏好的程式設計語言來處理資料。

### 匯出至 CSV

1. 在作業產生器中，執行您選擇的作業，例如掃描或查詢。

**Note**

- 您只能將讀取 API 作業和 PartiQL 陳述式的結果匯出至 CSV 檔案。您無法從交易讀取陳述式匯出結果。
- 目前，您可以一次將結果匯出至 CSV 檔案。如果結果有多頁，您必須個別匯出每個頁面。

2. 從結果中選取要匯出的項目。
3. 在「動作」下拉式清單中，選擇「匯出為 CSV」。
4. 為 CSV 檔案選擇檔案名稱及位置，然後選擇 Save (儲存)。

## NoSQL Workbench 的範例資料模型

模型建立工具和視覺化工具的首頁會顯示 NoSQL Workbench 隨附的許多範例模型。本節說明這些模型及其潛在用途。

### 主題

- [員工資料模型](#)
- [開發論壇資料模型](#)
- [音樂資料庫資料模型](#)
- [滑雪渡假村資料模型](#)
- [信用卡優惠資料模型](#)
- [書籤資料模型](#)

## 員工資料模型

這個資料模型是一個簡介模型。它代表了員工的基本詳細資料，例如唯一的別名、名字、姓氏、頭銜、主管和技能。

該資料模型會描繪了一些技術，例如處理複雜工作的屬性，像是擁有不只一個技能。此模型也會透過次要索引 DirectExports 的範例，說明主管與其直屬員工的一對多關係。

此資料模型有助於以下存取模式：

- 使用員工的登入別名擷取員工記錄，您可透過名為 Employee 的表格來加快完成此動作。
- 按名稱搜尋員工，您可透過名為 Name 的員工表格全域次要索引來加快完成此動作。
- 使用管理員的登入別名擷取管理員的所有直屬報告，您可透過名為 DirectReports 的員工表全域次要索引來加快完成此動作。

## 開發論壇資料模型

這個資料模型代表開發論壇。使用此模型，客戶可以與開發人員社群互動、提出問題，以及回應其他客戶的貼文。每一種 AWS 服務都有專用的論壇。任何人都可以在論壇張貼訊息、開始新的主題，每個主題都會收到許多回覆。

此資料模型有助於以下存取模式：

- 使用論壇名稱擷取論壇記錄，您可透過名為 Forum 的表格來加快完成此動作。
- 擷取某個論壇的特定主題或所有主題，您可透過名為 Thread 的資料表來加快完成此動作。
- 使用張貼使用者的電子郵件地址搜尋回覆，您可透過名為 PostedBy-Message-Index 的 Reply 資料表的全域次要索引來加快完成此動作。

## 音樂資料庫資料模型

此資料模型代表具有大量歌曲選集的音樂資料庫，能以近乎即時的方式展示下載次數最多的歌曲。

此資料模型有助於以下存取模式：

- 擷取一首歌曲記錄，您可透過名為 Songs 的表格加快完成此動作。
- 擷取特定的下載記錄或歌曲的所有下載記錄，您可透過名為 Songs 的表格加快完成此動作。
- 擷取歌曲特定的每月下載次數記錄，您可透過名為 Song 的表格加快完成此動作。
- 擷取歌曲的所有記錄 (包括歌曲記錄、下載記錄和每月下載次數記錄)，您可透過名為 Songs 的表格加快完成此動作。
- 搜尋最多人下載的歌曲，您可透過名為 DownloadsByMonth 的表格的全域次要索引加快完成此動作。

## 滑雪渡假村資料模型

此資料模型代表一個滑雪渡假村，其中有每部滑雪纜車每天收集到的大量資料集合。

此資料模型有助於以下存取模式：

- 擷取指定滑雪纜車或整體渡假村的所有資料 (動態和靜態)，您可透過名為 SkiLifts 的表格加快完成此動作。
- 在特定日期擷取滑雪纜車或整體渡假村的所有動態資料 (包括特別的纜車乘客，積雪覆蓋率、雪崩危險性和纜車狀態)，您可透過名為 SkiLifts 的表格加快完成此動作。
- 擷取特定滑雪纜車的所有靜態資料 (包括纜車是否是給有經驗的乘客使用、纜車升高的垂直高度 (英尺)、纜車乘坐時間)，您可透過名為 SkiLifts 的表格加快完成此動作。
- 擷取對特定滑雪纜車或整體渡假村記錄的資料日期 (依特別的乘客總數排序)，您可透過名為 SkiLiftsByRiders 的 SkiLifts 表格的全域次要索引加快完成此動作。

## 信用卡優惠資料模型

此資料模型由信用卡優惠應用程式使用。

信用卡供應商會不時提供優惠。這些優惠包括不收費的餘額轉移、提高信用額度、降低利率、現金回饋和航空公司里程數。客戶接受或拒絕這些優惠後，相應的優惠狀態會隨之更新。

此資料模型有助於以下存取模式：

- 使用 AccountId 擷取帳戶記錄，您可透過主表格加快完成此動作。
- 擷取有少量預估項目的所有帳戶，您可透過次要索引 AccountIndex 加快完成此動作。
- 使用 AccountId 擷取帳戶和所有與這些帳戶相關的優惠，您可透過主表格加快完成此動作。
- 使用 AccountId 和 OfferId 擷取與這些帳戶相關的帳戶和特定優惠記錄，您可透過主表格加快完成此動作。
- 使用 AccountId、OfferType 和 Status 擷取所有與帳戶相關且為 OfferType 特定的所有 ACCEPTED/DECLINED 優惠記錄，您可透過次要索引 GSI1 加快完成此動作。
- 使用 OfferId 擷取優惠和相關的優惠項目記錄，您可透過主表格加快完成此動作。

## 書籤資料模型

這個資料模型可供客戶用來存放書籤。

一個客戶可以有很多個書籤，且一個書籤可以屬於許多客戶。這個資料模型代表多對多的關係。

此資料模型有助於以下存取模式：

- `customerId` 的單一查詢現在可以傳回客戶資料以及書籤。
- 查詢 `ByEmail` 索引會透過電子郵件地址傳回客戶資料。請注意，此索引不會擷取書籤。
- 查詢 `ByUrl` 索引會透過 URL 取得書籤資料。請注意，我們有 `CustomerID` 作為索引的排序金鑰，因為相同的 URL 可以被多個客戶加入書籤。
- 查詢 `ByCustomerFolder` 索引會依資料夾取得每位客戶的書籤。

## NoSQL Workbench 的版本歷史記錄

下表說明在各版 NoSQL Workbench 用戶端工具的重大變更。

版本	變更	描述	日期
3.13.0	NoSQL 工作台操作生成器的改進	NoSQL 工作台現在包括對暗模式的本機支持。改進了操作生成器中的表格和項目操作。項目結果和操作生成器請求信息以 JSON 格式提供。	2024年4月24日
3.12.0	使用 NoSQL 工作台複製表格並使用的傳回容量	您現在可以在 DynamoDB 本機和 DynamoDB Web 服務帳戶之間複製表格，或在 DynamoDB Web 服務帳戶之間複製表格，以加快開發迭代速度。使用「作業建置器」執行作業後，檢視使用的 RCU 或 WCU。我們修復了從 CSV 文件導入時覆蓋數據的問題。	2024年2月26日
3.11.0	本機改良功能	您現在可以在啟動內建 DynamoDB 本機	2024年1月17日

版本	變更	描述	日期
		執行個體時指定連接埠。NoSQL 工作台現在可以安裝在沒有管理員權限的視窗上。我們更新了數據模型模板。	
3.10.0	對蘋果矽的原生支持	NoSQL 工作台現在包括 Mac 與蘋果矽的原生支持。您現在可以為類型的屬性設定範例資料產生格式Number。	2023 年 12 月 5 日
3.9.0	資料模型建立工具改進功能	視覺化工具現在可支援將資料模型提交至 DynamoDB 本機，並且可選擇覆寫現有資料表。	2023 年 11 月 3 日
3.8.0	產生範例資料	NoSQL Workbench 現在支援為您的 DynamoDB 資料模型產生範例資料。	2023 年 9 月 25 日
3.6.0	操作建置器的改善項目	操作建置器的連線管理改善項目。資料模型建立工具中的屬性名稱現在無需刪除資料即可變更。其他錯誤修復。	2023 年 4 月 11 日



版本	變更	描述	日期
3.5.0	Support 新 AWS 區域	NoSQL Workbench 現在支援 ap-south-2、ap-southeast-3、ap-southeast-4、eu-central-2、eu-south-2、me-central-1 和 me-west-1 區域。	2023 年 2 月 23 日
3.4.0	DynamoDB 本機版的支援	NoSQL Workbench 現在支援 DynamoDB 本機版做為安裝程序的一部分。	2022 年 12 月 6 日
3.3.0	控制平面操作支援	操作建置器現在支援控制平面操作。	2022 年 6 月 1 日
3.2.0	CSV 匯入和匯出	您現在可以從視覺化檢視工具中的 CSV 檔案匯入範例資料，也可以將作業產生器查詢的結果匯出至 CSV 檔案。	2021 年 10 月 11 日
3.1.0	儲存操作	NoSQL Workbench 中的操作建置器現在支援儲存操作以供日後使用。	2021 年 7 月 12 日
3.0.0	容量設 CloudFormation 定和匯入/匯出	Amazon DynamoDB 專用 NoSQL Workbench 現在不僅支援為資料表指定讀取/寫入容量模式，還能以 AWS CloudFormation 格式匯入和匯出資料模型。	2021 年 4 月 21 日

版本	變更	描述	日期
2.2.0	PartiQL 支援	Amazon DynamoDB 專用 NoSQL Workbench 新增支援建置 DynamoDB 的 PartiQL 陳述式。	2020 年 12 月 4 日
1.1.0	支援 Linux。	Linux Ubuntu、Fedora 與 Debian 均支援 Amazon DynamoDB 專用 NoSQL Workbench。	2020 年 5 月 4 日
1.0.0	Amazon DynamoDB 專用 NoSQL Workbench : GA 版。	Amazon DynamoDB 專用 NoSQL Workbench 現已正式推出。	2020 年 3 月 2 日
0.4.1	支援 IAM 角色和臨時安全登入資料。	Amazon DynamoDB 專用 NoSQL Workbench 新增對 AWS Identity and Access Management (IAM) 角色和臨時安全登入資料的支援。	2019 年 12 月 19 日
0.3.1	支援 <a href="#">DynamoDB 本機版 (可下載版本)</a> 。	NoSQL Workbench 現在支援連接至 <a href="#">DynamoDB 本機版 (可下載版本)</a> ，以設計、建立、查詢和管理 DynamoDB 資料表。	2019 年 11 月 8 日

版本	變更	描述	日期
0.2.1	已發行 NoSQL Workbench 預覽。	這是 DynamoDB 專用 NoSQL Workbench 的初始版本。使用 NoSQL Workbench 設計、建立、查詢及管理 DynamoDB 資料表。	2019 年 9 月 16 日

# 使用開發套件的 DynamoDB 程式碼範例 AWS

下列程式碼範例說明如何搭配 AWS 軟體開發套件 (SDK) 使用 DynamoDB。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境和跨服務範例中查看內容中的動作。

Scenarios (案例) 是向您展示如何呼叫相同服務中的多個函數來完成特定任務的程式碼範例。

Cross-service examples (跨服務範例) 是跨多個 AWS 服務執行的應用程式範例。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含入門相關資訊和舊版 SDK 的詳細資訊。

開始使用

## Hello DynamoDB

下列程式碼範例示範如何開始使用 DynamoDB。

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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();
 }
 }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[ListTables](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

### C MakeLists.txt 的 CMake 文件的代碼。

```
Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

Set the AWS service components used by this project.
set(SERVICE_COMPONENTS dynamodb)
```

```
Set this project's name.
project("hello_dynamodb")

Set the C++ standard to use to build this target.
At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
 string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
 list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
 # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

 # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
 # and set the proper subdirectory to the
 executables' location.

 AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
 hello_dynamodb.cpp)

target_link_libraries(${PROJECT_NAME}
 ${AWSSDK_LINK_LIBRARIES})
```

hello\_dynamodb.cpp 來源檔案的程式碼。

```
#include <aws/core/Aws.h>
```

```
#include <aws/dynamodb/DynamoDBClient.h>
#include <aws/dynamodb/model/ListTablesRequest.h>
#include <iostream>

/*
 * A "Hello DynamoDB" starter application which initializes an Amazon DynamoDB
 (DynamoDB) client and lists the
 * DynamoDB tables.
 *
 * main function
 *
 * Usage: 'hello_dynamodb'
 *
 */

int main(int argc, char **argv) {
 Aws::SDKOptions options;
 // Optionally change the log level for debugging.
 // options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
 Aws::InitAPI(options); // Should only be called once.

 int result = 0;
 {
 Aws::Client::ClientConfiguration clientConfig;
 // Optional: Set to the AWS Region (overrides config file).
 // clientConfig.region = "us-east-1";

 Aws::DynamoDB::DynamoDBClient dynamodbClient(clientConfig);
 Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
 listTablesRequest.SetLimit(50);
 do {
 const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
dynamodbClient.ListTables(
 listTablesRequest);
 if (!outcome.IsSuccess()) {
 std::cout << "Error: " << outcome.GetError().GetMessage() <<
std::endl;
 result = 1;
 break;
 }

 for (const auto &tableName: outcome.GetResult().GetTableNames()) {
 std::cout << tableName << std::endl;
 }
 }
 }
}
```

```
 listTablesRequest.SetExclusiveStartTableName(
 outcome.GetResult().GetLastEvaluatedTableName());
 } while (!listTablesRequest.GetExclusiveStartTableName().empty());
}

Aws::ShutdownAPI(options); // Should only be called once.
return result;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考 [ListTables](#) 中的。

## Java

適用於 Java 2.x 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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!");
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[ListTables](#)中的。

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考[ListTables](#)中的。

## 程式碼範例

- [使用開發套件執行的 DynamoDB 作 AWS](#)
- [搭BatchExecuteStatement配 AWS 開發套件或 CLI 使用](#)
- [搭BatchGetItem配 AWS 開發套件或 CLI 使用](#)

- [搭BatchWriteItem配 AWS 開發套件或 CLI 使用](#)
- [搭CreateTable配 AWS 開發套件或 CLI 使用](#)
- [搭DeleteItem配 AWS 開發套件或 CLI 使用](#)
- [搭DeleteTable配 AWS 開發套件或 CLI 使用](#)
- [搭DescribeTable配 AWS 開發套件或 CLI 使用](#)
- [搭ExecuteStatement配 AWS 開發套件或 CLI 使用](#)
- [搭GetItem配 AWS 開發套件或 CLI 使用](#)
- [搭ListTables配 AWS 開發套件或 CLI 使用](#)
- [搭PutItem配 AWS 開發套件或 CLI 使用](#)
- [搭Query配 AWS 開發套件或 CLI 使用](#)
- [搭Scan配 AWS 開發套件或 CLI 使用](#)
- [搭UpdateItem配 AWS 開發套件或 CLI 使用](#)
- [搭UpdateTable配 AWS 開發套件或 CLI 使用](#)
- [使用開發套件的 DynamoDB 案例 AWS](#)
  - [使用開發套件加速使用 DAX 讀取 AWS](#)
  - [使用開發套件開始使用 DynamoDB 資料表、項目和查詢 AWS](#)
  - [使用批次的 PartiQL 陳述式和開發套件來查詢 DynamoDB 資料表 AWS](#)
  - [使用 PartiQL 和開發套件查詢 DynamoDB 資料表 AWS](#)
  - [使用 SDK 為 DynamoDB 使用文件模型 AWS](#)
  - [使用 SDK 為 DynamoDB 使用高階物件持續性模型 AWS](#)
- [使用開發套件的 DynamoDB 的無伺服器範例 AWS](#)
  - [從 DynamoDB 觸發程序叫用 Lambda 函數](#)
  - [使用 DynamoDB 觸發程序報告 Lambda 函數的批次項目失敗](#)
- [使用開發套件的 DynamoDB 跨服務範例 AWS](#)
  - [建置應用程式以將資料提交至 DynamoDB 資料表](#)
  - [建立 API Gateway REST API 以追蹤 COVID-19 資料](#)
  - [使用 Step Functions 建立傳訊應用程式](#)
  - [建立相片資產管理應用程式，讓使用者以標籤管理相片](#)
  - [建立 Web 應用程式以追蹤 DynamoDB 資料](#)
- [使用 API Gateway 建立 websocket 聊天應用程式](#)

- [使用開發套件使用 Amazon 重新認知功能偵測影像中的個人防護裝置 AWS](#)
- [從瀏覽器調用 Lambda 函數](#)
- [使用開發套件監控 Amazon DynamoDB 的效能 AWS](#)
- [使用 SDK 儲存 EXIF 和其他影像資訊 AWS](#)
- [使用 API Gateway 來調用 Lambda 函數](#)
- [使用 Step Functions 呼叫 Lambda 函數](#)
- [使用排程事件來調用 Lambda 函數](#)

## 使用開發套件執行的 DynamoDB 作 AWS

下列程式碼範例示範如何使 AWS 用開發套件執行個別 DynamoDB 動作。這些摘錄會呼叫 DynamoDB API，是必須在內容中執行之大型程式的程式碼摘錄。每個範例都包含一個連結 GitHub，您可以在其中找到設定和執行程式碼的指示。

下列範例僅包含最常使用的動作。如需完整清單，請參閱 [《Amazon DynamoDB API 參考》](#)。

### 範例

- [搭BatchExecuteStatement配 AWS 開發套件或 CLI 使用](#)
- [搭BatchGetItem配 AWS 開發套件或 CLI 使用](#)
- [搭BatchWriteItem配 AWS 開發套件或 CLI 使用](#)
- [搭CreateTable配 AWS 開發套件或 CLI 使用](#)
- [搭DeleteItem配 AWS 開發套件或 CLI 使用](#)
- [搭DeleteTable配 AWS 開發套件或 CLI 使用](#)
- [搭DescribeTable配 AWS 開發套件或 CLI 使用](#)
- [搭ExecuteStatement配 AWS 開發套件或 CLI 使用](#)
- [搭GetItem配 AWS 開發套件或 CLI 使用](#)
- [搭ListTables配 AWS 開發套件或 CLI 使用](#)
- [搭PutItem配 AWS 開發套件或 CLI 使用](#)
- [搭Query配 AWS 開發套件或 CLI 使用](#)
- [搭Scan配 AWS 開發套件或 CLI 使用](#)
- [搭UpdateItem配 AWS 開發套件或 CLI 使用](#)
- [搭UpdateTable配 AWS 開發套件或 CLI 使用](#)

## 搭配BatchExecuteStatement配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用BatchExecuteStatement。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [使用多批 PartiQL 陳述式查詢資料表](#)

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用多批 INSERT 陳述式新增項目。

```
/// <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!;
 }
}
```

使用多批 SELECT 陳述式取得項目。

```
/// <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;
 }
}
```

使用多批 UPDATE 陳述式更新項目。

```
 /// <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;
}
```

使用多批 DELETE 陳述式刪除項目。

```
/// <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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[BatchExecuteStatement](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用多批 INSERT 陳述式新增項目。

```
// 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

std::vector<Aws::String> titles;
std::vector<float> ratings;
std::vector<int> years;
std::vector<Aws::String> plots;
Aws::String doAgain = "n";
do {
 Aws::String aTitle = askQuestion(
 "Enter the title of a movie you want to add to the table: ");
 titles.push_back(aTitle);
 int aYear = askQuestionForInt("What year was it released? ");
 years.push_back(aYear);
 float aRating = askQuestionForFloatRange(
 "On a scale of 1 - 10, how do you rate it? ",
 1, 10);
 ratings.push_back(aRating);
}
```

```

 Aws::String aPlot = askQuestion("Summarize the plot for me: ");
 plots.push_back(aPlot);

 doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
} while (doAgain == "y");

std::cout << "Adding " << titles.size()
 << (titles.size() == 1 ? " movie " : " movies ")
 << "to the table using a batch \"INSERT\" statement." << std::endl;

{
 Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
 titles.size());

 std::stringstream sqlStream;
 sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {"
 << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
 << INFO_KEY << "': ?}";

 std::string sql(sqlStream.str());

 for (size_t i = 0; i < statements.size(); ++i) {
 statements[i].SetStatement(sql);

 Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
 attributes.push_back(
 Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

 attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

 // Create attribute for the info map.
 Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

 std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
 = Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
 ALLOCATION_TAG.c_str());
 ratingAttribute->SetN(ratings[i]);
 infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

 std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
 Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
 ALLOCATION_TAG.c_str());
 plotAttribute->SetS(plots[i]);
 }
}

```

```

 infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
 attributes.push_back(infoMapAttribute);
 statements[i].SetParameters(attributes);
 }

 Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

 request.SetStatements(statements);

 Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
 request);
 if (!outcome.IsSuccess()) {
 std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
 << std::endl;
 return false;
 }
}

```

使用多批 SELECT 陳述式取得項目。

```

// 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;
}
}
```

使用多批 UPDATE 陳述式更新項目。

```
// 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;
 }
}
```

使用多批 DELETE 陳述式刪除項目。



```
// 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;
 }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[BatchExecuteStatement](#)中的。

## Go

## SDK for Go V2

 Note

還有更多關於 [GitHub](#)。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用多批 INSERT 陳述式新增項目。

```
// 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
}
```

使用多批 SELECT 陳述式取得項目。

```
// 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
}
```

```
}
```

使用多批 UPDATE 陳述式更新項目。

```
// 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
}
```

使用多批 DELETE 陳述式刪除項目。

```
// 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
}
```

定義此範例中使用的影片結構。

```
// 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"])
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考 [BatchExecuteStatement](#) 中的。

## JavaScript

適用於 JavaScript (v3) 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

使用 PartiQL 建立一批項目。

```
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;
};
```

使用 PartiQL 取得一批項目。

```
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;
};
```

使用 PartiQL 更新一批項目。

```
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;
};
```



使用 PartiQL 刪除一批項目。

```
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;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [BatchExecuteStatement](#) 中的。

## PHP

## 適用於 PHP 的開發套件

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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,
],
],
]);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考[BatchExecuteStatement](#)中的。

## Python

適用於 Python (Boto3) 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[BatchExecuteStatement](#)中的 Python (博托 3) API 參考。

## Ruby

適用於 Ruby 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用 PartiQL 讀取一批項目。

```
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
```

使用 PartiQL 刪除一批項目。

```
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
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考[BatchExecuteStatement](#)中的。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭BatchGetItem配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用BatchGetItem。

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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);
 }
 }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[BatchGetItem](#)中的。

## Bash

## AWS CLI 與 Bash 腳本

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
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
}

```

此範例中使用的公用程式函數。

```

#####
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
}

```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[BatchGetItem](#)中的。

## C++

## 適用於 C++ 的 SDK

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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;
}

```



- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考 [BatchGetItem](#) 中的。

## CLI

### AWS CLI

若要從表格擷取多個項目

下列 `batch-get-items` 範例會使用三個 `GetItem` 要求的批次，從 `MusicCollection` 表格讀取多個項目，並要求作業使用的讀取容量單位數目。該命令只返回 `AlbumTitle` 屬性。

```
aws dynamodb batch-get-item \
 --request-items file://request-items.json \
 --return-consumed-capacity TOTAL
```

`request-items.json` 的內容：

```
{
 "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"
 }
}
```

輸出：

```
{
```

```
"Responses": {
 "MusicCollection": [
 {
 "AlbumTitle": {
 "S": "Somewhat Famous"
 }
 },
 {
 "AlbumTitle": {
 "S": "Blue Sky Blues"
 }
 },
 {
 "AlbumTitle": {
 "S": "Louder Than Ever"
 }
 }
],
 "UnprocessedKeys": {},
 "ConsumedCapacity": [
 {
 "TableName": "MusicCollection",
 "CapacityUnits": 1.5
 }
]
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的 [Batch 操作](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考 [BatchGetItem](#) 中的。

## Java

適用於 Java 2.x 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

示範如何使用服務用戶端取得批次項目。

```
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.");
}
}
```

示範如何使用服務用戶端和分頁器取得批次項目。

```
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());
 });
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[BatchGetItem](#)中的。

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱[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;
};
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [BatchGetItem](#) 中的。

適用於 JavaScript (v2) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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);
 });
 }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [BatchGetItem](#) 中的。

## PowerShell

### 適用的工具 PowerShell

範例 1：從 DynamoDB 資料表 SongTitle 「音樂」和「歌曲」中取得含有「路上某處」的項目。

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

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list = New-Object
'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
Amazon.DynamoDBv2.Model.AttributeValue]]'
```



```

$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
 'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
 'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem

```

輸出：

Name	Value
----	-----
Artist	No One You Know
SongTitle	Somewhere Down The Road
AlbumTitle	Somewhat Famous
CriticRating	10
Genre	Country
Price	1.94
Artist	No One You Know
SongTitle	Somewhere Down The Road
AlbumTitle	Somewhat Famous
CriticRating	10
Genre	Country
Price	1.94

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 指令程[BatchGetItem](#)式參考中的。

## Python

適用於 Python (Boto3) 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[BatchGetItem](#)中的 Python (博托 3) API 參考。

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [BatchGetItem](#) 中的斯威夫特 API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭BatchWriteItem配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用BatchWriteItem。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [開始使用資料表、項目和查詢](#)

### .NET

#### AWS SDK for .NET

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

將一批項目寫入電影資料表。

```
/// <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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考 [BatchWriteItem](#) 中的。

## Bash

## AWS CLI 與 Bash 腳本

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
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
}

```

此範例中使用的公用程式函數。

```

#####
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
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[BatchWriteItem](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[BatchWriteItem](#)中的。

## CLI

### AWS CLI

若要將多個項目新增至表格

下列batch-write-item範例會使用三個PutItem要求的批次，將三個新項目新增至MusicCollection資料表。它也會要求作業使用的寫入容量單位數目，以及作業所修改之任何項目集合的相關資訊。

```
aws dynamodb batch-write-item \
 --request-items file://request-items.json \
 --return-consumed-capacity INDEXES \
 --return-item-collection-metrics SIZE
```

request-items.json 的內容：

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

```

輸出：

```

{
 "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
 }
 }
 }
]
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的 [Batch 操作](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考 [BatchWriteItem](#) 中的。

## Go

## SDK for Go V2

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考 [BatchWriteItem](#) 中的。

## Java

適用於 Java 2.x 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

通過使用服務客戶端插入許多項目到表中。

```
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);
 }
 }
}
```

使用增強型用戶端將許多項目插入資料表。

```
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);
 }
}
}
}

```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考 [BatchWriteItem](#) 中的。

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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);
}
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考[BatchWriteItem](#)中的。

適用於 JavaScript (v2) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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);
 }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [BatchWriteItem](#) 中的。

## PHP

## 適用於 PHP 的開發套件

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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;
 }
 }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考 [BatchWriteItem](#) 中的。

## PowerShell

### 適用的工具 PowerShell

範例 1：建立新項目，或以 DynamoDB 表格中的新項目取代現有項目「音樂」和「歌曲」。

```
$item = @{
 SongTitle = 'Somewhere Down The Road'
 Artist = 'No One You Know'
 AlbumTitle = 'Somewhat Famous'
 Price = 1.94
 Genre = 'Country'
 CriticRating = 10.0
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item
```

輸出：

```
$requestItem = @{
 'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
 'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 指令程 [BatchWriteItem](#) 式參考中的。

## Python

### 適用於 Python (Boto3) 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[BatchWriteItem](#)中的 Python (博托 3) API 參考。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
 end
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
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考 [BatchWriteItem](#) 中的。

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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)
}
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [BatchWriteItem](#) 中的斯威夫特 API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭 CreateTable 配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用 CreateTable。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [使用 DAX 加速讀取](#)
- [開始使用資料表、項目和查詢](#)

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
 /// <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.Write(".");
 }
 while (status != "ACTIVE");

 return status == TableStatus.ACTIVE;
 }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[CreateTable](#)中的。

## Bash

### AWS CLI 與 Bash 腳本

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
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
 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
}

```

此範例中使用的公用程式函數。

```

#####
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
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[CreateTable](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[CreateTable](#)中的。

## CLI

### AWS CLI

#### 範例 1：若要建立含標籤的資料表

下列create-table範例會使用指定的屬性和索引鍵結構描述來建立名為的資料表MusicCollection。此表格使用佈建的輸送量，並使用預設 AWS 擁有的 CMK 進行靜態加密。此指令也會將標籤套用至資料表，其中的索引鍵Owner和值為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
```

輸出：

```
{
 "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"
 }
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

### 範例 2：在隨選模式中建立表格

下列範例會建立名為 MusicCollection 使用隨選模式而非佈建輸送量模式的資料表。這對於工作負載不可預測的表格非常有用。

```

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

```

輸出：

```
{
```

```
"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"
 }
}
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

範例 3：建立資料表並使用客戶管理的 CMK 加密

下列範例會建立名為的資料表，MusicCollection並使用客戶管理的 CMK 加密資料表。

```
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
```

輸出：

```
{
 "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"
 }
}
}
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

範例 4：若要建立具有本機次要索引的資料表

下列範例會使用指定的屬性和索引鍵結構描述，建立名為MusicCollection本機次要索引的資料表AlbumTitleIndex。

```

aws dynamodb create-table \
 --table-name MusicCollection \
 --attribute-definitions AttributeName=Artist,AttributeType=S
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\"]
 }
 }
]

```

```
]"
```

輸出：

```
{
 "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"
 }
]
 }
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

#### 範例 5：若要建立具有全域次要索引的資料表

下列範例會建立名 GameScores 為全域次要索引的資料表 GameTitleIndex。基礎資料表具有分割區索引鍵 UserId 和排序索引鍵 GameTitle，可讓您有效率地找到個別使用者在特定遊戲中的最佳分數，而 GSI 具有分割區索引鍵 GameTitle 和排序索引鍵 TopScore，可讓您快速找到特定遊戲的整體最高分數。

```

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
 }
 }
]"

```

輸出：

```

{
 "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"
 }
]
}
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

#### 範例 6：一次建立含有多個全域次要索引的表格

下列範例會建立一個以兩個全域次要索引命名GameScores的資料表。GSI 結構描述是透過檔案傳遞的，而不是透過命令列傳遞。

```

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

```

gsi.json 的內容：

```

[
 {
 "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
 }
 }
]
}
```

輸出：

```
{
 "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"
 }
]
}
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的 [表格的基本操作](#)。

範例 7：若要建立啟用「串流」的表格

下列範例會建立一個名為GameScores用 DynamoDB Streams 的資料表。每個項目的新舊圖像都將被寫入流中。

```
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
```

輸出：

```
{
 "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"
}
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格的基本操作](#)。

範例 8：若要建立已啟用「僅限金鑰串流」的資料表

下列範例會建立一個名為 GameScores 用 DynamoDB Streams 的資料表。只有修改項目的索引鍵屬性會寫入資料流。

```

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

```

輸出：

```

{
 "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
}
}

```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的變更 DynamoDB [串流的資料擷取](#)。

範例 9：若要建立具有標準不常存取類別的資料表

下列範例會建立名為的資料表，GameScores並指派標準不常存取 (DynamoDB 標準 — IA) 表類別。此表類別針對主要成本的儲存進行了最佳化。

```
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
```

輸出：

```
{
 "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
}
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[表格類別](#)。

#### 範例 10：建立啟用刪除保護的資料表

下列範例會建立名為的資料表，GameScores 並啟用刪除保護。

```
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
```

輸出：

```
{
 "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
}
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[使用刪除保護](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考[CreateTable](#)中的。

Go

SDK for Go V2

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考[CreateTable](#)中的。

## Java

### 適用於 Java 2.x 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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 "";
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[CreateTable](#)中的。

## JavaScript

適用於 JavaScript (v3) 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
};
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [CreateTable](#) 中的。

## 適用於 JavaScript (v2) 的開發套件

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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);
 }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [CreateTable](#) 中的。

## Kotlin

### 適用於 Kotlin 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
 }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [CreateTable](#) 中的 Kotlin API 參考。

## PHP

適用於 PHP 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

建立 資料表。

```
$tableName = "ddb_demo_table_{$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],
]);
}

```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考 [CreateTable](#) 中的。

## PowerShell

### 適用的工具 PowerShell

範例 1：此範例會建立名為 Thread 的資料表，其主索引鍵由 'ForumName' (雜湊金鑰類型) 和 'Subject' (索引鍵類型範圍) 組成。用來建構資料表的結構描述可以傳送至每個指令程式，如所示或使用 -Schema 參數指定。

```

$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"

```

```
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

輸出：

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

範例 2：此範例會建立名為 Thread 的資料表，其主索引鍵由 'ForumName' (雜湊金鑰類型) 和 'Subject' (索引鍵類型範圍) 組成。也會定義本機次要索引。本地輔助索引的鍵將從表上的主哈希鍵自動設置 ( ForumName )。用來建構資料表的結構描述可以傳送至每個指令程式，如所示或使用 -Schema 參數指定。

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

輸出：

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

範例 3：此範例顯示如何使用單一管線建立名為 Thread 的資料表，其主索引鍵由 'ForumName' (索引鍵類型 hash) 和 'Subject' (索引鍵類型範圍) 和本機次要索引組成。如果沒有

從管線或-Schema 參數提供，Add-DDB KeySchema 和 Add-DDB 會為您IndexSchema 建立新 TableSchema 物件。

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

輸出：

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

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 指令程[CreateTable](#)式參考中的。

## Python

適用於 Python (Boto3) 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

建立用於存放電影資料的資料表。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[CreateTable](#)中的 Python (博托 3) API 參考。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
end

```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考[CreateTable](#)中的。

## Rust

### 適用於 Rust 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```

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))
 }
}
}
```

- 如需 API 的詳細資訊，請參閱 AWS SDK [CreateTable](#) 中的 Rust API 參考資料。

## SAP ABAP

## 適用於 SAP ABAP 的開發套件

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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.
```

- 如需 API 詳細資訊，請參閱 AWS SDK [CreateTable](#) 中的 SAP ABAP API 參考資料。

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
///
/// 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
 }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [CreateTable](#) 中的斯威夫特 API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭 `DeleteItem` 配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用 `DeleteItem`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [開始使用資料表、項目和查詢](#)

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
///
/// <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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[DeleteItem](#)中的。



## Bash

## AWS CLI 與 Bash 腳本

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
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
```

```
}
```

此範例中使用的公用程式函數。

```

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
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[DeleteItem](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[DeleteItem](#)中的。

## CLI

### AWS CLI

#### 範例 1：若要刪除項目

下列delete-item範例會從MusicCollection表格中刪除項目，並要求有關已刪除之項目以及請求所使用容量的詳細資訊。

```
aws dynamodb delete-item \
 --table-name MusicCollection \
```

```
--key file://key.json \
--return-values ALL_OLD \
--return-consumed-capacity TOTAL \
--return-item-collection-metrics SIZE
```

key.json 的內容：

```
{
 "Artist": {"S": "No One You Know"},
 "SongTitle": {"S": "Scared of My Shadow"}
}
```

輸出：

```
{
 "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
]
 }
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的 [撰寫項目](#)。

## 範例 2：若要有條件地刪除項目

下列範例只會刪除 ProductCatalog 表格中的某 ProductCategory 個物件，Sporting Goods 或 Gardening Supplies 且其價格在 500 到 600 之間。它返回有關被刪除的項目的詳細信息。

```
aws dynamodb delete-item \
 --table-name ProductCatalog \
 --key '{"Id":{"N":"456"}}' \
 --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (#P
 between :lo and :hi)" \
 --expression-attribute-names file://names.json \
 --expression-attribute-values file://values.json \
 --return-values ALL_OLD
```

names.json 的內容：

```
{
 "#P": "Price"
}
```

values.json 的內容：

```
{
 ":cat1": {"S": "Sporting Goods"},
 ":cat2": {"S": "Gardening Supplies"},
 ":lo": {"N": "500"},
 ":hi": {"N": "600"}
}
```

輸出：

```
{
 "Attributes": {
 "Id": {
 "N": "456"
 },
 "Price": {
 "N": "550"
 },
 },
}
```

```
 "ProductCategory": {
 "S": "Sporting Goods"
 }
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[撰寫項目](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考 [DeleteItem](#) 中的。

Go

SDK for Go V2

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考 [DeleteItem](#) 中的。

## Java

### 適用於 Java 2.x 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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);
 }
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考 [DeleteItem](#) 中的。

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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;
};
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [DeleteItem](#) 中的。

## 適用於 JavaScript (v2) 的開發套件

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

從資料表刪除項目。

```
// 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);
 }
});
```

使用 DynamoDB 文件用戶端從資料表刪除項目。

```
// 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);
 }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [DeleteItem](#) 中的。

## Kotlin

### 適用於 Kotlin 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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")
}
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [DeleteItem](#) 中的 Kotlin API 參考。

## PHP

### 適用於 PHP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
$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,
]);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考 [DeleteItem](#) 中的。

## PowerShell

### 適用的工具 PowerShell

範例 1：移除符合所提供金鑰的 DynamoDB 項目。

```
$key = @{
 SongTitle = 'Somewhere Down The Road'
 Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 指令程 [DeleteItem](#) 式參考中的。

## Python

### 適用於 Python (Boto3) 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

您可以指定條件，僅在符合特定條件時刪除項目。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[DeleteItem](#)中的 Python (博托 3) API 參考。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
end
```

```
Deletes a movie from the table.
#
@param title [String] The title of the movie to delete.
@param year [Integer] The release year of the movie to delete.
def delete_item(title, year)
 @table.delete_item(key: {"year" => year, "title" => title})
rescue Aws::DynamoDB::Errors::ServiceError => e
 puts("Couldn't delete movie #{title}. Here's why:")
 puts("\t#{e.code}: #{e.message}")
 raise
end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考 [DeleteItem](#) 中的。

## Rust

### 適用於 Rust 的 SDK

#### Note

還有更多關於 [GitHub](#)。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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)),
 }
}
```

- 如需 API 的詳細資訊，請參閱 AWS SDK [DeleteItem](#) 中的 Rust API 參考資料。

## SAP ABAP

### 適用於 SAP ABAP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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.
```

- 如需 API 詳細資訊，請參閱 AWS SDK [DeleteItem](#) 中的 SAP ABAP API 參考資料。

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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)
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [DeleteItem](#) 中的斯威夫特 API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭配 DeleteTable 配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用 DeleteTable。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [使用 DAX 加速讀取](#)
- [開始使用資料表、項目和查詢](#)

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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;
 }
}

```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考 [DeleteTable](#) 中的。

## Bash

### AWS CLI 與 Bash 腳本

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```

#####
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
}
```

此範例中使用的公用程式函數。



```
#####
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."
 fi
}
```

```
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
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[DeleteTable](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[DeleteTable](#)中的。

## CLI

### AWS CLI

#### 刪除表格的步驟

下列delete-table範例會刪除資MusicCollection料表。

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

輸出：

```
{
 "TableDescription": {
 "TableStatus": "DELETING",
 "TableSizeBytes": 0,
 "ItemCount": 0,
 "TableName": "MusicCollection",
 "ProvisionedThroughput": {
 "NumberOfDecreasesToday": 0,
 "WriteCapacityUnits": 5,
 }
 }
}
```


```
 "ReadCapacityUnits": 5
 }
}
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[刪除表格](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考[DeleteTable](#)中的。

Go

SDK for Go V2

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考 [DeleteTable](#) 中的。

## Java

### 適用於 Java 2.x 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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!");
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考 [DeleteTable](#) 中的。

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [DeleteTable](#) 中的。

### 適用於 JavaScript (v2) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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);
 }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [DeleteTable](#) 中的。

## Kotlin

### 適用於 Kotlin 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
 val request = DeleteTableRequest {
 tableName = tableNameVal
 }

 DynamoDbClient { region = "us-east-1" }.use { ddb ->
 ddb.deleteTable(request)
 println("$tableNameVal was deleted")
 }
}
```



- 有關 API 的詳細信息，請參閱 AWS SDK [DeleteTable](#) 中的 Kotlin API 參考。

## PHP

適用於 PHP 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
public function deleteTable(string $TableName)
{
 $this->customWaiter(function () use ($TableName) {
 return $this->dynamoDbClient->deleteTable([
 'TableName' => $TableName,
]);
 });
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考 [DeleteTable](#) 中的。

## PowerShell

適用的工具 PowerShell

範例 1：刪除指定的資料表。在進行作業之前，系統會提示您進行確認。

```
Remove-DDBTable -TableName "myTable"
```

範例 2：刪除指定的資料表。在進行作業之前，系統不會提示您進行確認。

```
Remove-DDBTable -TableName "myTable" -Force
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 指令程 [DeleteTable](#) 式參考中的。

## Python

### 適用於 Python (Boto3) 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[DeleteTable](#)中的 Python (博托 3) API 參考。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考[DeleteTable](#)中的。

## Rust

### 適用於 Rust 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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())),
 }
}
```

- 如需 API 的詳細資訊，請參閱 AWS SDK [DeleteTable](#) 中的 Rust API 參考資料。

## SAP ABAP

### 適用於 SAP ABAP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

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.
```

- 如需 API 詳細資訊，請參閱 AWS SDK [DeleteTable](#) 中的 SAP ABAP API 參考資料。

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
///
/// 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)
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [DeleteTable](#) 中的斯威夫特 API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭 DescribeTable 配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用 DescribeTable。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [開始使用資料表、項目和查詢](#)

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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}");
 }

```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考 [DescribeTable](#) 中的。

## Bash

### AWS CLI 與 Bash 腳本

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```

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

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

local table_status
table_status=$(
 aws dynamodb describe-table \
 --table-name "$table_name" \
 --output text \
 --query 'Table.TableStatus'
)

local error_code=${?}

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

echo "$table_status"
```



```
 return 0
}
```

此範例中使用的公用程式函數。

```
#####
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
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[DescribeTable](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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();
}

```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考 [DescribeTable](#) 中的。

## CLI

## AWS CLI

## 若要描述表格

下列describe-table範例說明資MusicCollection料表。

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

## 輸出：

```
{
 "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
 }
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[說明表格](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考[DescribeTable](#)中的。

Go

SDK for Go V2

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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, ¬FoundEx) {
 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
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考[DescribeTable](#)中的。

## Java

### 適用於 Java 2.x 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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!");
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[DescribeTable](#)中的。

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
};
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [DescribeTable](#) 中的。

適用於 JavaScript (v2) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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);
 }
});
```

```
}
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [DescribeTable](#) 中的。

## PowerShell

### 適用的工具 PowerShell

範例 1：傳回指定資料表的詳細資訊。

```
Get-DDBTable -TableName "myTable"
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 指令程 [DescribeTable](#) 式參考中的。

## Python

### 適用於 Python (Boto3) 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[DescribeTable](#)中的 Python (博托 3) API 參考。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考[DescribeTable](#)中的。

## SAP ABAP

### 適用於 SAP ABAP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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.
```

- 如需 API 詳細資訊，請參閱 AWS SDK [DescribeTable](#) 中的 SAP ABAP API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭ExecuteStatement配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用ExecuteStatement。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [使用 PartiQL 查詢資料表](#)

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用 INSERT 陳述式新增項目。

```
/// <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;
}
```

使用 SELECT 陳述式取得項目。

```
/// <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;
}
```

使用 SELECT 陳述式取得項目清單。

```
/// <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;
}
```

使用 UPDATE 陳述式更新項目。

```
/// <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;
}
```



使用 DELETE 陳述式刪除單個影片。

```
/// <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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[ExecuteStatement](#)中的。

## C++

## 適用於 C++ 的 SDK

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用 INSERT 陳述式新增項目。

```
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;
 }
 }
}

```

使用 SELECT 陳述式取得項目。

```

// 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;
 }
 }
}
}

```

使用 UPDATE 陳述式更新項目。

```

// 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;
}
}

```

使用 DELETE 陳述式刪除項目。

```

// 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;
 }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考 [ExecuteStatement](#) 中的。

## Go

### SDK for Go V2

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

使用 INSERT 陳述式新增項目。

```
// 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
}
```

```
}
```

使用 SELECT 陳述式取得項目。

```
// 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
}
```

使用 SELECT 陳述式取得項目清單並推斷結果。

```
// 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
}
```

使用 UPDATE 陳述式更新項目。

```
// 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
}
```

使用 DELETE 陳述式刪除項目。

```
// 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
}
```

```
}
```

定義此範例中使用的影片結構。

```
// 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"])
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考[ExecuteStatement](#)中的。

## JavaScript

適用於 JavaScript (v3) 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用 PartiQL 建立項目。

```
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;
};
```

使用 PartiQL 取得項目。

```
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;
};
```

使用 PartiQL 更新項目。

```
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;
};
```

使用 PartiQL 刪除項目。

```
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;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考[ExecuteStatement](#)中的。

## PHP

### 適用於 PHP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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,
]);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考[ExecuteStatement](#)中的。

## Python

適用於 Python (Boto3) 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[ExecuteStatement](#)中的 Python (博托 3) API 參考。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用 PartiQL 選取單一項目。

```
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

```

使用 PartiQL 更新單一項目。

```

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

```

使用 PartiQL 新增單一項目。

```

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

```

使用 PartiQL 刪除單一項目。

```

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]
 }
 @dynamodb.client.execute_statement(request)
end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考[ExecuteStatement](#)中的。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭GetItem配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用GetItem。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [使用 DAX 加速讀取](#)
- [開始使用資料表、項目和查詢](#)

### .NET

#### AWS SDK for .NET

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
 /// <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;
 }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[GetItem](#)中的。

## Bash

### AWS CLI 與 Bash 腳本

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
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 getopt "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
}

```

```
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
}
```

此範例中使用的公用程式函數。

```
#####
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
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[GetItem](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[GetItem](#)中的。

## CLI

### AWS CLI

#### 範例 1：讀取表格中的項目

下列 `get-item` 範例會從資料 `MusicCollection` 表擷取項目。該表具有 `hash-and-range` 主鍵（`Artist` 和 `SongTitle`），因此您必須同時指定這兩個屬性。此命令也會要求作業所耗用之讀取容量的相關資訊。

```
aws dynamodb get-item \
```

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

key.json 的內容：

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

輸出：

```
{
 "Item": {
 "AlbumTitle": {
 "S": "Songs About Life"
 },
 "SongTitle": {
 "S": "Happy Day"
 },
 "Artist": {
 "S": "Acme Band"
 }
 },
 "ConsumedCapacity": {
 "TableName": "MusicCollection",
 "CapacityUnits": 0.5
 }
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的讀取項目](#)。

範例 2：若要使用一致讀取來讀取項目

下列範例會使用強烈一致的讀取，從資料 MusicCollection 表擷取項目。

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

key.json 的內容：

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

輸出：

```
{
 "Item": {
 "AlbumTitle": {
 "S": "Songs About Life"
 },
 "SongTitle": {
 "S": "Happy Day"
 },
 "Artist": {
 "S": "Acme Band"
 }
 },
 "ConsumedCapacity": {
 "TableName": "MusicCollection",
 "CapacityUnits": 1.0
 }
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的讀取項目](#)。

範例 3：擷取料號的特定屬性

下列範例會使用投影運算式，僅擷取所需項目的三個屬性。

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

names.json 的內容：

```
{
```

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

輸出：

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

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的讀取項目](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考 [GetItem](#) 中的。

Go

SDK for Go V2

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考 [GetItem](#) 中的。

## Java

適用於 Java 2.x 的 SDK

### Note

還有更多關於 [GitHub](#)。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

使用取得表格中的項目 `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);
 }
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[GetItem](#)中的。



## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考[GetItem](#)中的。

### 適用於 JavaScript (v2) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

從資料表取得項目。

```
// 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);
 }
});
```

使用 DynamoDB 文件用戶端從資料表取得項目。

```
// 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);
 }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [GetItem](#) 中的。

## Kotlin

### 適用於 Kotlin 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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)
 }
 }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [GetItem](#) 中的 Kotlin API 參考。

## PHP

適用於 PHP 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
$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,
]);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考 [GetItem](#) 中的。

## PowerShell

適用的工具 PowerShell

範例 1：傳回含有分區索引鍵 SongTitle 和排序索引鍵演出者的 DynamoDB 項目。

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

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

輸出：

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

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 指令程[GetItem](#)式參考中的。

## Python

適用於 Python (Boto3) 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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"]
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[GetItem](#)中的 Python (博托 3) API 參考。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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

```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考 [GetItem](#) 中的。

## SAP ABAP

### 適用於 SAP ABAP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```

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.

```

- 如需 API 詳細資訊，請參閱 AWS SDK [GetItem](#) 中的 SAP ABAP API 參考資料。

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [GetItem](#) 中的斯威夫特 API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭 ListTables 配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用 ListTables。

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[ListTables](#)中的。

## Bash

### AWS CLI 與 Bash 腳本

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
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"
 fi
}
```

```

 return 1
fi

echo "$response" | tr -s "[:space:]" "\n"

return 0
}

```

此範例中使用的公用程式函數。

```

#####
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
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[ListTables](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[ListTables](#)中的。

## CLI

### AWS CLI

#### 範例 1：列出表格

下列list-tables範例會列出與目前 AWS 帳戶和 Region 相關聯的所有表格。

```
aws dynamodb list-tables
```

輸出：

```
{
 "TableNames": [
 "Forum",
 "ProductCatalog",
 "Reply",
 "Thread"
]
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[列出表格名稱](#)。

### 範例 2：限制頁面大小

下列範例會傳回所有現有資料表的清單，但是每次呼叫中只會擷取一個項目，如果需要執行多個呼叫以取得整個清單。在大量資源上執行清單命令時，限制頁面大小非常有用，這可能會導致使用預設頁面大小 1000 時發生「逾時」錯誤。

```
aws dynamodb list-tables \
 --page-size 1
```

輸出：

```
{
 "TableNames": [
 "Forum",
 "ProductCatalog",
 "Reply",
 "Thread"
]
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[列出表格名稱](#)。

### 範例 3：限制傳回的項目數

下列範例會將傳回的項目數限制為 2。回應包含用來擷取下一頁結果的 NextToken 值。

```
aws dynamodb list-tables \
 --max-items 2
```

輸出：

```
{
 "TableNames": [
 "Forum",
 "ProductCatalog"
],
 "NextToken":
 "abCDeFGhiJKlmnOPqrSTuvwXYZ1aBCdEFghijK7LM51n0pqRSTuv3WxY3ZabC5dEFghI2Jk3LmnoPQ6RST9"
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[列出表格名稱](#)。

#### 範例 4：擷取下一頁結果

下列命令會使用上一次呼叫list-tables命令的NextToken值來擷取另一個結果頁面。由於在這種情況下的響應不包括一個NextToken值，我們知道我們已經達到了結果的結果。

```
aws dynamodb list-tables \
 --starting-token
 abCDeFGhiJKlmnOPqrSTuvwXYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9
```

輸出：

```
{
 "TableNames": [
 "Reply",
 "Thread"
]
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[列出表格名稱](#)。

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[ListTables](#)中的。

Go

SDK for Go V2

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考[ListTables](#)中的。

## Java

### 適用於 Java 2.x 的 SDK

#### Note

還有更多關於 [GitHub](#)。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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!");
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[ListTables](#)中的。

## JavaScript

適用於 JavaScript (v3) 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
};
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [ListTables](#) 中的。

適用於 JavaScript (v2) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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);
 }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [ListTables](#) 中的。

## Kotlin

### 適用於 Kotlin 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
suspend fun listAllTables() {
 DynamoDbClient { region = "us-east-1" }.use { ddb ->
 val response = ddb.listTables(ListTablesRequest {})
 response.tableNames?.forEach { tableName ->
 println("Table name is $tableName")
 }
 }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [ListTables](#) 中的 Kotlin API 參考。

## PHP

### 適用於 PHP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
public function listTables($exclusiveStartTableName = "", $limit = 100)
{
 $this->dynamoDbClient->listTables([
 'ExclusiveStartTableName' => $exclusiveStartTableName,
 'Limit' => $limit,
]);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考[ListTables](#)中的。

## PowerShell

### 適用的工具 PowerShell

示例 1：返回所有表的詳細信息，自動迭代，直到服務指示沒有其他表存在。

```
Get-DDBTableList
```

範例 2：手動重複執行所有資料表的詳細資訊，每次呼叫最多可傳回 10 個資料表，直到服務指出沒有其他資料表存在為止。

```
$nextToken = $null
do {
 Get-DDBTableList -ExclusiveStartTableName $nextToken -Limit 10
 $nextToken = $AWSHistory.LastServiceResponse.LastEvaluatedTableName
} while ($nextToken -ne $null)
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 指令程[ListTables](#)式參考中的。

## Python

### 適用於 Python (Boto3) 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[ListTables](#)中的 Python (博托 3) API 參考。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

判斷資料表是否存在。

```
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
end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考[ListTables](#)中的。

## Rust

### 適用於 Rust 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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::
```

判斷資料表是否存在。

```
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()),
 }
}
```

- 如需 API 的詳細資訊，請參閱 AWS SDK [ListTables](#) 中的 Rust API 參考資料。

## SAP ABAP

適用於 SAP ABAP 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

TRY.



```
oo_result = lo_dyn->listtables().
" You can loop over the oo_result to get table properties like this.
LOOP AT oo_result->get_tablenames() INTO DATA(lo_table_name).
 DATA(lv_tablename) = lo_table_name->get_value().
ENDLOOP.
DATA(lv_tablecount) = lines(oo_result->get_tablenames()).
MESSAGE 'Found ' && lv_tablecount && ' tables' TYPE 'I'.
CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
 DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
 MESSAGE lv_error TYPE 'E'.
ENDTRY.
```

- 如需 API 詳細資訊，請參閱 AWS SDK [ListTables](#) 中的 SAP ABAP API 參考資料。

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [ListTables](#) 中的斯威夫特 API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭PutItem配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用PutItem。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [使用 DAX 加速讀取](#)
- [開始使用資料表、項目和查詢](#)

## .NET

### AWS SDK for .NET

#### Note

還有更多關於 [GitHub](#)。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
 /// <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;
 }
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考 [PutItem](#) 中的。

## Bash

### AWS CLI 與 Bash 腳本

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
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
}
```

此範例中使用的公用程式函數。

```
#####
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
}

```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[PutItem](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[PutItem](#)中的。

## CLI

### AWS CLI

範例 1：若要將項目新增至表格

下列 `put-item` 範例會將新項目新增至 `MusicCollection` 表格中。

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

`item.json` 的內容：

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

輸出：

```
{
 "ConsumedCapacity": {
 "TableName": "MusicCollection",
 "CapacityUnits": 1.0
 },
 "ItemCollectionMetrics": {
 "ItemCollectionKey": {
 "Artist": {
```

```

 "S": "No One You Know"
 }
},
"SizeEstimateRangeGB": [
 0.0,
 1.0
]
}
}

```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的撰寫項目](#)。

**範例 2：** 若要有條件地覆寫資料表中的項目

下列 `put-item` 範例只有在現有項目具有值為的 `AlbumTitle` 屬性時，才會覆寫 `MusicCollection` 表格中的 `Greatest Hits` 現有項目。該命令返回項目的先前值。

```

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

```

`item.json` 的內容：

```

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

```

`names.json` 的內容：

```

{
 "#A": "AlbumTitle"
}

```

`values.json` 的內容：

```

{

```

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

輸出：

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

如果密鑰已經存在，您應該看到以下輸出：

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

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[撰寫項目](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考 [PutItem](#) 中的。

Go

SDK for Go V2

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考 [PutItem](#) 中的。

## Java

### 適用於 Java 2.x 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

使用將項目放入表格中 [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);
 }
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考 [PutItem](#) 中的。

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [PutItem](#) 中的。

適用於 JavaScript (v2) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

將項目放入資料表。

```
// 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);
 }
 });
```

使用 DynamoDB 文件用戶端將項目放入資料表。

```
// 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);
 }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [PutItem](#) 中的。

## Kotlin

### 適用於 Kotlin 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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.")
 }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [PutItem](#) 中的 Kotlin API 參考。

## PHP

## 適用於 PHP 的開發套件

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考[PutItem](#)中的。

## PowerShell

### 適用的工具 PowerShell

範例 1：建立新料號，或以新料號取代現有料號。

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

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 指令程 [PutItem](#) 式參考中的。

## Python

### 適用於 Python (Boto3) 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[PutItem](#)中的 Python (博托 3) API 參考。

## Ruby

適用於 Ruby 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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

```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考 [PutItem](#) 中的。

## Rust

### 適用於 Rust 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```

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,
})
}
```

- 如需 API 的詳細資訊，請參閱 AWS SDK [PutItem](#) 中的 Rust API 參考資料。



## SAP ABAP

### 適用於 SAP ABAP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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.
```

- 如需 API 詳細資訊，請參閱 AWS SDK [PutItem](#) 中的 SAP ABAP API 參考資料。

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

**Note**

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [PutItem](#) 中的斯威夫特 API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭 Query 配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用 Query。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [使用 DAX 加速讀取](#)
- [開始使用資料表、項目和查詢](#)

## .NET

### AWS SDK for .NET

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// <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;
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for .NET API 參考》中的 [Query](#)。

## Bash

### AWS CLI 與 Bash 腳本

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
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
}

```

此範例中使用的公用程式函數。

```

#####
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
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [Query](#)。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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);
 }
 } while (true);
}
```

```

 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;
}

```

- 如需 API 的詳細資訊，請參閱《AWS SDK for C++ API 參考》中的 [Query](#)。

## CLI

### AWS CLI

#### 範例 1：若要查詢資料表

下列 query 範例會查詢 MusicCollection 資料表中的項目。該表有一個 hash-and-range 主鍵 (Artist 和 SongTitle)，但這個查詢只指定哈希鍵值。它返回名為「沒有人你認識」的藝術家的歌曲標題。

```
aws dynamodb query \
 --table-name MusicCollection \
 --projection-expression "SongTitle" \
 --key-condition-expression "Artist = :v1" \
 --expression-attribute-values file://expression-attributes.json \
 --return-consumed-capacity TOTAL
```

expression-attributes.json 的內容：

```
{
 ":v1": {"S": "No One You Know"}
}
```

輸出：

```
{
 "Items": [
 {
 "SongTitle": {
 "S": "Call Me Today"
 },
 "SongTitle": {
 "S": "Scared of My Shadow"
 }
 }
],
 "Count": 2,
 "ScannedCount": 2,
 "ConsumedCapacity": {
 "TableName": "MusicCollection",
```

```
 "CapacityUnits": 0.5
 }
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的使用 DynamoDB 中的查詢](#)。

範例 2：若要使用強烈一致的讀取來查詢資料表，並以遞減順序遍歷索引

下列範例會執行與第一個範例相同的查詢，但會以相反順序傳回結果，並使用強式一致的讀取。

```
aws dynamodb query \
 --table-name MusicCollection \
 --projection-expression "SongTitle" \
 --key-condition-expression "Artist = :v1" \
 --expression-attribute-values file://expression-attributes.json \
 --consistent-read \
 --no-scan-index-forward \
 --return-consumed-capacity TOTAL
```

expression-attributes.json 的內容：

```
{
 ":v1": {"S": "No One You Know"}
}
```

輸出：

```
{
 "Items": [
 {
 "SongTitle": {
 "S": "Scared of My Shadow"
 }
 },
 {
 "SongTitle": {
 "S": "Call Me Today"
 }
 }
],
 "Count": 2,
 "ScannedCount": 2,
 "ConsumedCapacity": {
```

```
 "TableName": "MusicCollection",
 "CapacityUnits": 1.0
 }
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的使用 DynamoDB 中的查詢](#)。

### 範例 3：篩選出特定結果

下列範例會查詢 MusicCollection 但排除 AlbumTitle 屬性中具有特定值的結果。請注意，這不會影響 ScannedCount 或 ConsumedCapacity，因為篩選器會在項目讀取之後套用。

```
aws dynamodb query \
 --table-name MusicCollection \
 --key-condition-expression "#n1 = :v1" \
 --filter-expression "NOT (#n2 IN (:v2, :v3))" \
 --expression-attribute-names file://names.json \
 --expression-attribute-values file://values.json \
 --return-consumed-capacity TOTAL
```

values.json 的內容：

```
{
 ":v1": {"S": "No One You Know"},
 ":v2": {"S": "Blue Sky Blues"},
 ":v3": {"S": "Greatest Hits"}
}
```

names.json 的內容：

```
{
 "#n1": "Artist",
 "#n2": "AlbumTitle"
}
```

輸出：

```
{
 "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
}
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的使用 DynamoDB 中的查詢](#)。

#### 範例 4：僅擷取料號計數

下列範例會擷取符合查詢的項目計數，但不會擷取任何項目本身。

```
aws dynamodb query \
 --table-name MusicCollection \
 --select COUNT \
 --key-condition-expression "Artist = :v1" \
 --expression-attribute-values file://expression-attributes.json
```

expression-attributes.json 的內容：

```
{
 ":v1": {"S": "No One You Know"}
}
```

輸出：

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

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的使用 DynamoDB 中的查詢](#)。

### 範例 5：若要查詢索引

下列範例會查詢本機次要索引 AlbumTitleIndex。查詢會傳回已投影至本機次要索引的基底資料表中的所有屬性。請注意，查詢本機次要索引或全域次要索引時，您也必須使用 table-name 參數提供基底資料表的名稱。

```
aws dynamodb query \
 --table-name MusicCollection \
 --index-name AlbumTitleIndex \
 --key-condition-expression "Artist = :v1" \
 --expression-attribute-values file://expression-attributes.json \
 --select ALL_PROJECTED_ATTRIBUTES \
 --return-consumed-capacity INDEXES
```

expression-attributes.json 的內容：

```
{
 ":v1": {"S": "No One You Know"}
}
```

輸出：

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

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的使用 DynamoDB 中的查詢](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [Query](#)。

Go

SDK for Go V2

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for Go API 參考》](#) 中的 Query。

## Java

### 適用於 Java 2.x 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用查詢表格 [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;
}
}
```

使用 `DynamoDbClient` 和次要索引查詢資料表。

```
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);
 }
}
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 [Query](#)。

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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;
};
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 的詳細資訊，請參閱 [《AWS SDK for JavaScript API 參考》](#) 中的 Query。

適用於 JavaScript (v2) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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);
 }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 的詳細資訊，請參閱 [《AWS SDK for JavaScript API 參考》](#) 中的 Query。

## Kotlin

### 適用於 Kotlin 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
 }
}
```

- 如需 API 的詳細資訊，請參閱 [《適用於 Kotlin 的 AWS SDK API 參考》](#) 中的 [Query](#)。

## PHP

## 適用於 PHP 的開發套件

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
$birthKey = [
 'Key' => [
 'year' => [
 'N' => "$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);
 }
```

- 如需 API 的詳細資訊，請參閱《AWS SDK for PHP API 參考》中的 [Query](#)。

## PowerShell

### 適用的工具 PowerShell

範例 1：叫用傳回具有指定和演出者的 DynamoDB 項目 SongTitle 的查詢。

```
$invokeDDBQuery = @{
 TableName = 'Music'
 KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
 ExpressionAttributeValues = @{
 ':SongTitle' = 'Somewhere Down The Road'
 ':Artist' = 'No One You Know'
 } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

輸出：

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

- 如需 API 詳細資訊，請參閱在 AWS Tools for PowerShell 指令程式參考中 [查詢](#)。

## Python

### 適用於 Python (Boto3) 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用索引鍵條件表達式查詢項目。

```
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"]
```

查詢項目並投影以傳回資料子集。

```
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"]
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的 AWS SDK API 參考》中的 [Query](#)。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for Ruby API 參考》](#) 中的 Query。

## Rust

### 適用於 Rust 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

尋找在指定年份製作的電影。

```
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![])
 }
}

```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [Query](#)。

## SAP ABAP

### 適用於 SAP ABAP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```

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

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的 AWS SDK API 參考》中的 [Query](#)。

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Swift 的 AWS SDK API 參考》中的 [Query](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭Scan配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用Scan。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [使用 DAX 加速讀取](#)
- [開始使用資料表、項目和查詢](#)

## .NET

### AWS SDK for .NET

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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;
}

```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for .NET API 參考》](#) 中的 Scan。

## Bash

### AWS CLI 與 Bash 腳本

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```

#####
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}" ;;
 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 "$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
}

```

此範例中使用的公用程式函數。

```

#####
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
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [Scan](#)。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
//! Scan an Amazon DynamoDB table.
/*!
 \sa scanTable()
 \param tableName: Name for the DynamoDB table.
 \param projectionExpression: An optional projection expression, ignored if
 empty.
 \param clientConfiguration: AWS client configuration.
 \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::scanTable(const Aws::String &tableName,
 const Aws::String &projectionExpression,
 const Aws::Client::ClientConfiguration
&clientConfiguration) {
 Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
```

```
Aws::DynamoDB::Model::ScanRequest request;
request.SetTableName(tableName);

if (!projectionExpression.empty())
 request.SetProjectionExpression(projectionExpression);

Aws::Vector<Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>>
all_items;
Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
last_evaluated_key; // Used for pagination;
do {
 if (!last_evaluated_key.empty()) {
 request.SetExclusiveStartKey(last_evaluated_key);
 }
 const Aws::DynamoDB::Model::ScanOutcome &outcome =
dynamoClient.Scan(request);
 if (outcome.IsSuccess()) {
 // Reference the retrieved items.
 const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
 all_items.insert(all_items.end(), items.begin(), items.end());

 last_evaluated_key = outcome.GetResult().GetLastEvaluatedKey();
 }
 else {
 std::cerr << "Failed to Scan items: " <<
outcome.GetError().GetMessage()
 << std::endl;
 return false;
 }
} while (!last_evaluated_key.empty());

if (!all_items.empty()) {
 std::cout << "Number of items retrieved from scan: " << all_items.size()
 << std::endl;
 // Iterate each item and print.
 for (const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&itemMap: all_items) {
 std::cout << "*****"
 << std::endl;
 // Output each retrieved field and its value.
 for (const auto &itemEntry: itemMap)
 std::cout << itemEntry.first << ": " << itemEntry.second.GetS()
```

```
 << std::endl;
 }
}

else {
 std::cout << "No items found in table: " << tableName << std::endl;
}

return true;
}
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for C++ API 參考》](#) 中的 Scan。

## CLI

### AWS CLI

#### 掃描表格

下列 scan 範例會掃描整個 MusicCollection 表格，然後將結果縮小為演出者「沒有您認識的人」的歌曲。對於每個項目，僅返回專輯標題和歌曲標題。

```
aws dynamodb scan \
 --table-name MusicCollection \
 --filter-expression "Artist = :a" \
 --projection-expression "#ST, #AT" \
 --expression-attribute-names file://expression-attribute-names.json \
 --expression-attribute-values file://expression-attribute-values.json
```

expression-attribute-names.json 的內容：

```
{
 "#ST": "SongTitle",
 "#AT": "AlbumTitle"
}
```

expression-attribute-values.json 的內容：

```
{
```

```
":a": {"S": "No One You Know"}
}
```

輸出：

```
{
 "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
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的在 Dynamo DB 中使用掃描](#)。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [Scan](#)。

Go

SDK for Go V2

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for Go API 參考》](#) 中的 Scan。

## Java

適用於 Java 2.x 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

使用掃描 Amazon DynamoDB 表。 [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.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);
 }
}
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for Java 2.x API 參考》](#) 中的 Scan。

## JavaScript

適用於 JavaScript (v3) 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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;
};
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for JavaScript API 參考》](#) 中的 Scan。

適用於 JavaScript (v2) 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
// 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 + ")"
);
 });
 }
});
```

- 如需詳細資訊，請參閱 [《AWS SDK for JavaScript 開發人員指南》](#)。
- 如需 API 的詳細資訊，請參閱 [《AWS SDK for JavaScript API 參考》](#) 中的 Scan。

## Kotlin

### 適用於 Kotlin 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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]}")
 }
 }
 }
}
```

```
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Kotlin 的 AWS SDK API 參考》中的 [Scan](#)。

## PHP

### 適用於 PHP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
$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);
}
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for PHP API 參考》](#) 中的 Scan。

## PowerShell

### 適用的工具 PowerShell

範例 1：傳回「音樂」資料表中的所有項目。

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

輸出：

Name	Value
----	-----
Genre	Country
Artist	No One You Know
Price	1.94
CriticRating	9
SongTitle	Somewhere Down The Road
AlbumTitle	Somewhat Famous
Genre	Country
Artist	No One You Know
Price	1.98
CriticRating	8.4
SongTitle	My Dog Spot
AlbumTitle	Hey Now

範例 2：傳回「音樂」資料表中 CriticRating 大於或等於 9 的項目。

```
$scanFilter = @{
 CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
 AttributeValueList = @(@{N = '9'})
 ComparisonOperator = 'GE'
 }
}
```

```
Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

輸出：

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

- 如需 API 詳細資訊，請參閱在AWS Tools for PowerShell 指令程式參考中[掃描](#)。

## Python

適用於 Python (Boto3) 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的 AWS SDK API 參考》中的 [Scan](#)。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
```

- 如需 API 的詳細資訊，請參閱 [《AWS SDK for Ruby API 參考》](#) 中的 Scan。

## Rust

### 適用於 Rust 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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(())
}
```

- 如需 API 的詳細資訊，請參閱《適用於 Rust 的 AWS SDK API 參考》中的 [Scan](#)。

## SAP ABAP

### 適用於 SAP ABAP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的 AWS SDK API 參考》中的 [Scan](#)。

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的 AWS SDK API 參考》中的 [Scan](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭UpdateItem配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用UpdateItem。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [開始使用資料表、項目和查詢](#)

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
 /// <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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[UpdateItem](#)中的。

## Bash

## AWS CLI 與 Bash 腳本

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
#####
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
}

```

此範例中使用的公用程式函數。

```

#####
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
}
```

```
}
```

- 如需 API 詳細資訊，請參閱AWS CLI 命令參考[UpdateItem](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
//! 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();
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考 [UpdateItem](#) 中的。

## CLI

## AWS CLI

## 範例 1：若要更新表格中的項目

下列 `update-item` 範例會更新 `MusicCollection` 資料表中的項目。它增加了一個新的屬性 (`Year`) 並修改 `AlbumTitle` 屬性。在回應中傳回項目中的所有屬性，就像它們在更新之後所顯示的那樣。

```
aws dynamodb update-item \
 --table-name MusicCollection \
 --key file://key.json \
 --update-expression "SET #Y = :y, #AT = :t" \
 --expression-attribute-names file://expression-attribute-names.json \
 --expression-attribute-values file://expression-attribute-values.json \
 --return-values ALL_NEW \
 --return-consumed-capacity TOTAL \
 --return-item-collection-metrics SIZE
```

`key.json` 的內容：

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

`expression-attribute-names.json` 的內容：

```
{
 "#Y": "Year", "#AT": "AlbumTitle"
}
```

`expression-attribute-values.json` 的內容：

```
{
 ":y": {"N": "2015"},
 ":t": {"S": "Louder Than Ever"}
}
```

輸出：

```
{
 "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
]
}
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的[撰寫項目](#)。

#### 範例 2：若要有條件地更新料號

下列範例會更新 MusicCollection 表格中的項目，但前提是現有項目尚未具有 Year 屬性時。

```
aws dynamodb update-item \
 --table-name MusicCollection \
 --condition-expression 'attribute_not_exists(year)'
```

```
--key file://key.json \
--update-expression "SET #Y = :y, #AT = :t" \
--expression-attribute-names file://expression-attribute-names.json \
--expression-attribute-values file://expression-attribute-values.json \
--condition-expression "attribute_not_exists(#Y)"
```

key.json 的內容：

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

expression-attribute-names.json 的內容：

```
{
 "#Y": "Year",
 "#AT": "AlbumTitle"
}
```

expression-attribute-values.json 的內容：

```
{
 ":y": {"N": "2015"},
 ":t": {"S": "Louder Than Ever"}
}
```

如果項目已具有Year屬性，DynamoDB 會傳回下列輸出。

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
operation: The conditional request failed
```

如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的 [撰寫項目](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考 [UpdateItem](#) 中的。

## Go

## SDK for Go V2

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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"])
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考 [UpdateItem](#) 中的。

## Java

適用於 Java 2.x 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

使用更新表格中的項目 [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!");
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[UpdateItem](#)中的。

## JavaScript

適用於 JavaScript (v3) 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

此範例使用文件用戶端來簡化 DynamoDB 中處理項目的作業。如需 API 詳細資訊，請參閱 [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;
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [UpdateItem](#) 中的。

## Kotlin

適用於 Kotlin 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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")
 }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [UpdateItem](#) 中的 Kotlin API 參考。

## PHP

### 適用於 PHP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
 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
]
],
]);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考[UpdateItem](#)中的。

## PowerShell

### 適用的工具 PowerShell

範例 1：使用分區索引鍵 `SongTitle` 和排序索引鍵演出者，將 DynamoDB 項目上的類型屬性設定為「Rap」。

```
$key = @{
 SongTitle = 'Somewhere Down The Road'
 Artist = 'No One You Know'
```

```

} | ConvertTo-DDBItem

$updateDdbItem = @{
 TableName = 'Music'
 Key = $key
 UpdateExpression = 'set Genre = :val1'
 ExpressionAttributeValue = (@{
 ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
 })
}
Update-DDBItem @updateDdbItem

```

輸出：

Name	Value
----	-----
Genre	Rap

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 指令程[UpdateItem](#)式參考中的。

## Python

適用於 Python (Boto3) 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用更新表達式更新項目。

```

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"]
```

使用包含算術運算的更新表達式更新項目。

```
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"]
```

僅在項目符合特定條件時更新項目。

```
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"]
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[UpdateItem](#)中的 Python (博托 3) API 參考。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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
 end
end
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考[UpdateItem](#)中的。

## SAP ABAP

### 適用於 SAP ABAP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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.
```

- 如需 API 詳細資訊，請參閱 AWS SDK [UpdateItem](#) 中的 SAP ABAP API 參考資料。

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

**Note**

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
/// 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
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [UpdateItem](#) 中的斯威夫特 API 參考。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 搭 UpdateTable 配 AWS 開發套件或 CLI 使用

下列程式碼範例會示範如何使用 UpdateTable。

### CLI

#### AWS CLI

##### 範例 1：修改表格的計費模式

下列 update-table 範例會增加資 MusicCollection 料表上佈建的讀取和寫入容量。

```
aws dynamodb update-table \
 --table-name MusicCollection \
 --billing-mode PROVISIONED \
```

```
--provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10
```

輸出：

```
{
 "TableDescription": {
 "AttributeDefinitions": [
 {
 "AttributeName": "AlbumTitle",
 "AttributeType": "S"
 },
 {
 "AttributeName": "Artist",
 "AttributeType": "S"
 },
 {
 "AttributeName": "SongTitle",
 "AttributeType": "S"
 }
],
 "TableName": "MusicCollection",
 "KeySchema": [
 {
 "AttributeName": "Artist",
 "KeyType": "HASH"
 },
 {
 "AttributeName": "SongTitle",
 "KeyType": "RANGE"
 }
],
 "TableStatus": "UPDATING",
 "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
 "ProvisionedThroughput": {
 "LastIncreaseDateTime": "2020-07-28T13:18:18.921000-07:00",
 "NumberOfDecreasesToday": 0,
 "ReadCapacityUnits": 15,
 "WriteCapacityUnits": 10
 },
 "TableSizeBytes": 182,
 "ItemCount": 2,
 "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
```



```
 "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
 "BillingModeSummary": {
 "BillingMode": "PROVISIONED",
 "LastUpdateToPayPerRequestDateTime":
"2020-07-28T13:14:48.366000-07:00"
 }
 }
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的更新表格](#)。

## 範例 2：建立全域次要索引

下列範例會將全域次要索引新增至 MusicCollection 料表。

```
aws dynamodb update-table \
 --table-name MusicCollection \
 --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
 --global-secondary-index-updates file://gsi-updates.json
```

gsi-updates.json 的內容：

```
[
 {
 "Create": {
 "IndexName": "AlbumTitle-index",
 "KeySchema": [
 {
 "AttributeName": "AlbumTitle",
 "KeyType": "HASH"
 }
],
 "ProvisionedThroughput": {
 "ReadCapacityUnits": 10,
 "WriteCapacityUnits": 10
 },
 "Projection": {
 "ProjectionType": "ALL"
 }
 }
 }
]
```

輸出：

```
{
 "TableDescription": {
 "AttributeDefinitions": [
 {
 "AttributeName": "AlbumTitle",
 "AttributeType": "S"
 },
 {
 "AttributeName": "Artist",
 "AttributeType": "S"
 },
 {
 "AttributeName": "SongTitle",
 "AttributeType": "S"
 }
],
 "TableName": "MusicCollection",
 "KeySchema": [
 {
 "AttributeName": "Artist",
 "KeyType": "HASH"
 },
 {
 "AttributeName": "SongTitle",
 "KeyType": "RANGE"
 }
],
 "TableStatus": "UPDATING",
 "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
 "ProvisionedThroughput": {
 "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
 "NumberOfDecreasesToday": 0,
 "ReadCapacityUnits": 15,
 "WriteCapacityUnits": 10
 },
 "TableSizeBytes": 182,
 "ItemCount": 2,
 "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
 "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
 "BillingModeSummary": {
 "BillingMode": "PROVISIONED",
```

```
 "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
 },
 "GlobalSecondaryIndexes": [
 {
 "IndexName": "AlbumTitle-index",
 "KeySchema": [
 {
 "AttributeName": "AlbumTitle",
 "KeyType": "HASH"
 }
],
 "Projection": {
 "ProjectionType": "ALL"
 },
 "IndexStatus": "CREATING",
 "Backfilling": false,
 "ProvisionedThroughput": {
 "NumberOfDecreasesToday": 0,
 "ReadCapacityUnits": 10,
 "WriteCapacityUnits": 10
 },
 "IndexSizeBytes": 0,
 "ItemCount": 0,
 "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
 }
]
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的更新表格](#)。

範例 3：若要在表格上啟用 DynamoDB Streams

下列命令會在資料表上啟用 DynamoDB Streams。MusicCollection

```
aws dynamodb update-table \
 --table-name MusicCollection \
 --stream-specification StreamEnabled=true,StreamViewType=NEW_IMAGE
```

輸出：

```
{
 "TableDescription": {
 "AttributeDefinitions": [
 {
 "AttributeName": "AlbumTitle",
 "AttributeType": "S"
 },
 {
 "AttributeName": "Artist",
 "AttributeType": "S"
 },
 {
 "AttributeName": "SongTitle",
 "AttributeType": "S"
 }
],
 "TableName": "MusicCollection",
 "KeySchema": [
 {
 "AttributeName": "Artist",
 "KeyType": "HASH"
 },
 {
 "AttributeName": "SongTitle",
 "KeyType": "RANGE"
 }
],
 "TableStatus": "UPDATING",
 "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
 "ProvisionedThroughput": {
 "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
 "NumberOfDecreasesToday": 0,
 "ReadCapacityUnits": 15,
 "WriteCapacityUnits": 10
 },
 "TableSizeBytes": 182,
 "ItemCount": 2,
 "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
 "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
 "BillingModeSummary": {
 "BillingMode": "PROVISIONED",
```

```
 "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
 },
 "LocalSecondaryIndexes": [
 {
 "IndexName": "AlbumTitleIndex",
 "KeySchema": [
 {
 "AttributeName": "Artist",
 "KeyType": "HASH"
 },
 {
 "AttributeName": "AlbumTitle",
 "KeyType": "RANGE"
 }
],
 "Projection": {
 "ProjectionType": "INCLUDE",
 "NonKeyAttributes": [
 "Year",
 "Genre"
]
 },
 "IndexSizeBytes": 139,
 "ItemCount": 2,
 "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
 }
],
 "GlobalSecondaryIndexes": [
 {
 "IndexName": "AlbumTitle-index",
 "KeySchema": [
 {
 "AttributeName": "AlbumTitle",
 "KeyType": "HASH"
 }
],
 "Projection": {
 "ProjectionType": "ALL"
 },
 "IndexStatus": "ACTIVE",
 "ProvisionedThroughput": {
 "NumberOfDecreasesToday": 0,
```

```
 "ReadCapacityUnits": 10,
 "WriteCapacityUnits": 10
 },
 "IndexSizeBytes": 0,
 "ItemCount": 0,
 "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
 }
],
"StreamSpecification": {
 "StreamEnabled": true,
 "StreamViewType": "NEW_IMAGE"
},
"LatestStreamLabel": "2020-07-28T21:53:39.112",
"LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112"
}
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的更新表格](#)。

#### 範例 4：啟用伺服器端加密

下列範例會在 MusicCollection 資料表上啟用伺服器端加密。

```
aws dynamodb update-table \
 --table-name MusicCollection \
 --sse-specification Enabled=true,SSEType=KMS
```

輸出：

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

```
 "AttributeName": "SongTitle",
 "AttributeType": "S"
 }
],
"TableName": "MusicCollection",
"KeySchema": [
 {
 "AttributeName": "Artist",
 "KeyType": "HASH"
 },
 {
 "AttributeName": "SongTitle",
 "KeyType": "RANGE"
 }
],
"TableStatus": "ACTIVE",
"CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
"ProvisionedThroughput": {
 "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
 "NumberOfDecreasesToday": 0,
 "ReadCapacityUnits": 15,
 "WriteCapacityUnits": 10
},
"TableSizeBytes": 182,
"ItemCount": 2,
"TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
"TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
"BillingModeSummary": {
 "BillingMode": "PROVISIONED",
 "LastUpdateToPayPerRequestDateTime":
"2020-07-28T13:14:48.366000-07:00"
},
"LocalSecondaryIndexes": [
 {
 "IndexName": "AlbumTitleIndex",
 "KeySchema": [
 {
 "AttributeName": "Artist",
 "KeyType": "HASH"
 },
 {
 "AttributeName": "AlbumTitle",
 "KeyType": "RANGE"
 }
]
 }
]
```

```
 }
],
 "Projection": {
 "ProjectionType": "INCLUDE",
 "NonKeyAttributes": [
 "Year",
 "Genre"
]
 },
 "IndexSizeBytes": 139,
 "ItemCount": 2,
 "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
}
],
"GlobalSecondaryIndexes": [
 {
 "IndexName": "AlbumTitle-index",
 "KeySchema": [
 {
 "AttributeName": "AlbumTitle",
 "KeyType": "HASH"
 }
],
 "Projection": {
 "ProjectionType": "ALL"
 },
 "IndexStatus": "ACTIVE",
 "ProvisionedThroughput": {
 "NumberOfDecreasesToday": 0,
 "ReadCapacityUnits": 10,
 "WriteCapacityUnits": 10
 },
 "IndexSizeBytes": 0,
 "ItemCount": 0,
 "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
 }
],
"StreamSpecification": {
 "StreamEnabled": true,
 "StreamViewType": "NEW_IMAGE"
},
"LatestStreamLabel": "2020-07-28T21:53:39.112",
```



```
 "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112",
 "SSEDescription": {
 "Status": "UPDATING"
 }
}
}
```

如需詳細資訊，請參閱 [Amazon DynamoDB 開發人員指南中的更新表格](#)。

- 如需 API 詳細資訊，請參閱 AWS CLI 命令參考 [UpdateTable](#) 中的。

## PowerShell

適用的工具 PowerShell

範例 1：更新指定表格的佈建輸送量。

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 指令程 [UpdateTable](#) 式參考中的。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用開發套件的 DynamoDB 案例 AWS

下列程式碼範例說明如何在具 AWS 有開發套件的 DynamoDB 中實作常見案例。這些案例會向您展示如何呼叫 DynamoDB 中的多個函數來完成特定任務。每個案例都包含一個連結 GitHub，您可以在其中找到如何設定和執程式碼的指示。

### 範例

- [使用開發套件加速使用 DAX 讀取 AWS](#)
- [使用開發套件開始使用 DynamoDB 資料表、項目和查詢 AWS](#)
- [使用批次的 PartiQL 陳述式和開發套件來查詢 DynamoDB 資料表 AWS](#)
- [使用 PartiQL 和開發套件查詢 DynamoDB 資料表 AWS](#)
- [使用 SDK 為 DynamoDB 使用文件模型 AWS](#)
- [使用 SDK 為 DynamoDB 使用高階物件持續性模型 AWS](#)

## 使用開發套件加速使用 DAX 讀取 AWS

以下程式碼範例顯示做法：

- 使用 DAX 和 SDK 用戶端建立資料並將其寫入資料表。
- 使用兩個用戶端取得、查詢和掃描表格，並比較其效能。

如需詳細資訊，請參閱[使用 DynamoDB Accelerator 用戶端開發](#)。

### Python

適用於 Python (Boto3) 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用 DAX 或 Boto3 用戶端建立資料表。

```
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.")

```

將測試資料寫入資料表。

```

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)
```

為 DAX 用戶端和 Boto3 用戶端多次迭代取得項目，並報告每個迭代所花費的時間。

```
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}}."
)
```

為 DAX 用戶端和 Boto3 用戶端查詢多次迭代的資料表，並報告每個迭代所花費的時間。

```
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}."
)
```

為 DAX 用戶端和 Boto3 用戶端掃描多次迭代的資料表，並報告每個迭代所花費的時間。

```
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}."
)
```

## 刪除 資料表。

```
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!")
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的 AWS SDK API 參考資料》中的下列主題。
  - [CreateTable](#)
  - [DeleteTable](#)
  - [GetItem](#)
  - [PutItem](#)
  - [查詢](#)
  - [掃描](#)

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用開發套件開始使用 DynamoDB 資料表、項目和查詢 AWS

下列程式碼範例示範如何：

- 建立可存放電影資料的資料表。
- 放入、取得和更新資料表中的單個電影。
- 將影片資料從範例 JSON 檔案寫入資料表。
- 查詢特定年份發表的電影。
- 掃描某個年份範圍內發表的電影。
- 從資料表刪除電影，然後刪除資料表。

## .NET

### AWS SDK for .NET

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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();
 }
}
```

建立包含電影資料的資料表。

```
 /// <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;
}
```

新增單一電影到資料表。

```
 /// <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;
 }
```

更新資料表中的單一項目。

```
 /// <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;
}
```

從電影資料表擷取單一項目。

```
/// <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;
}
```

將一批項目寫入電影資料表。

```
/// <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;
}
```

從資料表刪除單一項目。

```
/// <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;
 }
}
```

在資料表中查詢特定年份發表的電影。

```
/// <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;
}
```

在資料表中掃描某個年份範圍內發表的電影。

```
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;
}

```

刪除電影資料表。

```

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;
}
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for .NET API 參考》中的下列主題。
  - [BatchWriteItem](#)
  - [CreateTable](#)
  - [DeleteItem](#)
  - [DeleteTable](#)
  - [DescribeTable](#)
  - [GetItem](#)
  - [PutItem](#)
  - [查詢](#)
  - [掃描](#)
  - [UpdateItem](#)

## Bash

### AWS CLI 與 Bash 腳本

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

DynamoDB 入門案例。

```
#####
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
}

```

此案例中使用的 DynamoDB 函數。

```

#####
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
```

```
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
}
```

此案例中使用的公用程式函數。

```
#####
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
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - [BatchWriteItem](#)
  - [CreateTable](#)
  - [DeleteItem](#)
  - [DeleteTable](#)
  - [DescribeTable](#)
  - [GetItem](#)
  - [PutItem](#)
  - [查詢](#)
  - [掃描](#)
  - [UpdateItem](#)

## C++

## 適用於 C++ 的 SDK

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
{
 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;
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for C++ API 參考》中的下列主題。
  - [BatchWriteItem](#)
  - [CreateTable](#)
  - [DeleteItem](#)
  - [DeleteTable](#)
  - [DescribeTable](#)
  - [GetItem](#)
  - [PutItem](#)
  - [查詢](#)
  - [掃描](#)
  - [UpdateItem](#)

Go

SDK for Go V2

 Note

還有更多關於 [GitHub](#)。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

執行互動式案例以建立資料表並對其執行動作。

```
// 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))
}
```

定義此範例中使用的影片結構。

```
// 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"])
}
```

建立呼叫 DynamoDB 動作的結構和方法。

```
// 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, ¬FoundEx) {
 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
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Go API 參考》中的下列主題。
  - [BatchWriteItem](#)
  - [CreateTable](#)
  - [DeleteItem](#)
  - [DeleteTable](#)
  - [DescribeTable](#)
  - [GetItem](#)
  - [PutItem](#)
  - [查詢](#)
  - [掃描](#)
  - [UpdateItem](#)

## Java

適用於 Java 2.x 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

建立 DynamoDB 資料表。

```
// 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);
 }
}
```

建立 Helper 函數以下載並擷取範例 JSON 檔案。

```
// 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++;
 }
}
```

從資料表取得項目。

```
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);
 }
}
```

完整範例。

```
/**
 * 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);
 }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - [BatchWriteItem](#)
  - [CreateTable](#)
  - [DeleteItem](#)
  - [DeleteTable](#)
  - [DescribeTable](#)
  - [GetItem](#)
  - [PutItem](#)
  - [查詢](#)
  - [掃描](#)
  - [UpdateItem](#)

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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.");
 };
```

- 如需 API 詳細資訊，請參閱《AWS SDK for JavaScript API 參考》中的下列主題。
  - [BatchWriteItem](#)
  - [CreateTable](#)
  - [DeleteItem](#)
  - [DeleteTable](#)
  - [DescribeTable](#)
  - [GetItem](#)
  - [PutItem](#)
  - [查詢](#)
  - [掃描](#)
  - [UpdateItem](#)

## Kotlin

### 適用於 Kotlin 的 SDK

#### Note

還有更多關於 [GitHub](#)。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

### 建立 DynamoDB 資料表。

```
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}")
}
}
```

建立 Helper 函數以下載並擷取範例 JSON 檔案。

```
// 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.")
 }
}
```

從資料表取得項目。

```
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)
 }
 }
}
```

## 完整範例。

```
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]}")
 }
 }
 }
}
}
```

- 如需 API 詳細資訊，請參閱《AWS 適用於 Kotlin 的 SDK API 參考》中的下列主題。
  - [BatchWriteItem](#)
  - [CreateTable](#)
  - [DeleteItem](#)
  - [DeleteTable](#)
  - [DescribeTable](#)
  - [GetItem](#)
  - [PutItem](#)
  - [查詢](#)
  - [掃描](#)
  - [UpdateItem](#)

## PHP

## 適用於 PHP 的開發套件

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
namespace DynamoDb\Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;
use DynamoDb\DynamoDBService;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithDynamoDB
{
 public function run()
 {
 echo("\n");
 echo("-----\n");
 print("Welcome to the Amazon DynamoDB getting started demo using PHP!
\n");
 echo("-----\n");

 $uuid = uniqid();
 $service = new DynamoDBService();

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

```
echo "Waiting for table...";
$service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
$tableName]);
echo "table $tableName found!\n";

echo "What's the name of the last movie you watched?\n";
while (empty($movieName)) {
 $movieName = testable_readline("Movie name: ");
}
echo "And what year was it released?\n";
$movieYear = "year";
while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
 $movieYear = testable_readline("Year released: ");
}

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

echo "How would you rate the movie from 1-10?\n";
$rating = 0;
while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
|| $rating > 10) {
 $rating = testable_readline("Rating (1-10): ");
}
echo "What was the movie about?\n";
while (empty($plot)) {
 $plot = testable_readline("Plot summary: ");
}
$key = [
 'Item' => [
 'title' => [
 'S' => $movieName,
],
 'year' => [
 'N' => $movieYear,
```



```

],
]
];
$attributes = ["rating" =>
 [
 'AttributeName' => 'rating',
 'AttributeType' => 'N',
 'Value' => $rating,
],
 'plot' => [
 'AttributeName' => 'plot',
 'AttributeType' => 'S',
 'Value' => $plot,
]
];
$service->updateItemAttributesByKey($tableName, $key, $attributes);
echo "Movie added and updated.";

$batch = json_decode(loadMovieData());

$service->writeBatch($tableName, $batch);

$movie = $service->getItemByKey($tableName, $key);
echo "\n\nThe movie {$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);
 }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for PHP API 參考》中的下列主題。
  - [BatchWriteItem](#)
  - [CreateTable](#)
  - [DeleteItem](#)
  - [DeleteTable](#)
  - [DescribeTable](#)
  - [GetItem](#)
  - [PutItem](#)
  - [查詢](#)
  - [掃描](#)
  - [UpdateItem](#)

## Python

適用於 Python (Boto3) 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

建立封裝 DynamoDB 資料表的類別。

```
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
```

建立 Helper 函數以下載並擷取範例 JSON 檔案。

```
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]
```

執行互動式案例以建立資料表並對其執行動作。

```
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}")

```

此案例使用以下協助類別在命令提示中提出問題。

```
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
```

- 如需 API 的詳細資訊，請參閱《適用於 Python (Boto3) 的 AWS SDK API 參考資料》中的下列主題。
  - [BatchWriteItem](#)

- [CreateTable](#)
- [DeleteItem](#)
- [DeleteTable](#)
- [DescribeTable](#)
- [GetItem](#)
- [PutItem](#)
- [查詢](#)
- [掃描](#)
- [UpdateItem](#)

## Ruby

適用於 Ruby 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

建立封裝 DynamoDB 資料表的類別。

```
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"}
]
)
end
```

```
],
 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
```

建立 Helper 函數以下載並擷取範例 JSON 檔案。

```
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
```

執行互動式案例以建立資料表並對其執行動作。

```
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
end
end
```

```
 end
 else
 puts("I don't know about any movies released between #{years[:start]} "\
 "and #{years[:end]}".)
 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
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Ruby API 參考》中的下列主題。
  - [BatchWriteItem](#)
  - [CreateTable](#)
  - [DeleteItem](#)
  - [DeleteTable](#)
  - [DescribeTable](#)
  - [GetItem](#)
  - [PutItem](#)

- [查詢](#)
- [掃描](#)
- [UpdateItem](#)

## SAP ABAP

適用於 SAP ABAP 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執程式碼範例儲存庫](#)。

```
" 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_dynresourceindex.
MESSAGE 'The table or index does not exist' TYPE 'E'.
CATCH /aws1/cx_dynresourceinuse.
MESSAGE 'The table cannot be deleted as it is in use' TYPE 'E'.
ENDTRY.
```


- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的 AWS SDK API 參考》中的下列主題。
  - [BatchWriteItem](#)
  - [CreateTable](#)
  - [DeleteItem](#)
  - [DeleteTable](#)
  - [DescribeTable](#)
  - [GetItem](#)
  - [PutItem](#)
  - [查詢](#)
  - [掃描](#)
  - [UpdateItem](#)

## Swift

### 適用於 Swift 的 SDK

#### Note

這是適用於預覽版本 SDK 的發行前版本文件。內容可能變動。

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

一個用於處理對適用於 Swift 的 SDK 之 DynamoDB 呼叫的 Swift 類別。

```
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)
}
}
```

MovieTable 類別用來表示電影的結構。

```
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
}
}
```

使用 MovieTable 類別來存取 DynamoDB 資料庫的程式。

```
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)
 }
 }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的 AWS SDK API 參考》中的下列主題。
  - [BatchWriteItem](#)
  - [CreateTable](#)
  - [DeleteItem](#)
  - [DeleteTable](#)
  - [DescribeTable](#)
  - [GetItem](#)
  - [PutItem](#)
  - [查詢](#)
  - [掃描](#)
  - [UpdateItem](#)

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用批次的 PartiQL 陳述式和開發套件來查詢 DynamoDB 資料表 AWS

下列程式碼範例示範如何：

- 透過執行多個 SELECT 陳述式取得一批項目。
- 透過執行多個 INSERT 陳述式新增一批項目。
- 透過執行多個 UPDATE 陳述式更新一批項目。
- 透過執行多個 DELETE 陳述式刪除一批項目。

### .NET

#### AWS SDK for .NET

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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
 {
```

```
25) for (var indexOffset = 0; indexOffset < 250; indexOffset +=
 {
 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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考[BatchExecuteStatement](#)中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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.GetResponses();

 for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
 responses) {
 const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 &item = response.GetItem();

 printMovieInfo(item);
 }
 }
 else {
```

```
 std::cerr << "Failed to retrieve the movie information: "
 << outcome.GetError().GetMessage() << std::endl;
 return false;
 }
}

// 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

for (size_t i = 0; i < titles.size(); ++i) {
 ratings[i] = askQuestionForFloatRange(
 Aws::String("\nLet's update your the movie, \"" + titles[i] +
 ".\nYou rated it " + std::to_string(ratings[i])
 + ", what new rating would you give it? ", 1, 10));
}

std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

{
 Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
 titles.size());

 std::stringstream sqlStream;
 sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
 << INFO_KEY << "." << RATING_KEY << "=? WHERE "
 << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

 std::string sql(sqlStream.str());

 for (size_t i = 0; i < statements.size(); ++i) {
 statements[i].SetStatement(sql);

 Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
 attributes.push_back(
 Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
 attributes.push_back(
 Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

 attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
 statements[i].SetParameters(attributes);
 }
}
```

```
Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

request.SetStatements(statements);
Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
 request);
if (!outcome.IsSuccess()) {
 std::cerr << "Failed to update movie information: "
 << outcome.GetError().GetMessage() << std::endl;
 return false;
}
}

std::cout << "Retrieving the updated movie data with a batch \"SELECT\"
statement."
 << std::endl;

// 5. Get the updated data for multiple movies using "Select" statements.
(BatchExecuteStatement)
{
 Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
 titles.size());
 std::stringstream sqlStream;
 sqlStream << "SELECT * FROM \" << MOVIE_TABLE_NAME << "\" WHERE "
 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

 std::string sql(sqlStream.str());

 for (size_t i = 0; i < statements.size(); ++i) {
 statements[i].SetStatement(sql);
 Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
 attributes.push_back(
 Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));
 attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
 statements[i].SetParameters(attributes);
 }

 Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

 request.SetStatements(statements);

 Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
```

```

 request);
 if (outcome.IsSuccess()) {
 const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

 const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

 for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
 const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

 printMovieInfo(item);
 }
 }
 else {
 std::cerr << "Failed to retrieve the movies information: "
 << outcome.GetError().GetMessage() << std::endl;
 return false;
 }
}

std::cout << "Deleting the movie data with a batch \"DELETE\" statement."
 << std::endl;

// 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
{
 Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
 titles.size());
 std::stringstream sqlStream;
 sqlStream << "DELETE FROM \"" << MOVIE_TABLE_NAME << "\" WHERE "
 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

 std::string sql(sqlStream.str());

 for (size_t i = 0; i < statements.size(); ++i) {
 statements[i].SetStatement(sql);
 Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
 attributes.push_back(
 Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));
 attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
 }
}

```



```
 statements[i].SetParameters(attributes);
 }

 Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

 request.SetStatements(statements);

 Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
 request);

 if (!outcome.IsSuccess()) {
 std::cerr << "Failed to delete the movies: "
 << outcome.GetError().GetMessage() << std::endl;
 return false;
 }
}

return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
 \sa createMoviesDynamoDBTable()
 \param clientConfiguration: AWS client configuration.
 \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
 const Aws::Client::ClientConfiguration &clientConfiguration) {
 Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

 bool movieTableAlreadyExisted = false;

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

 Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
 yearAttributeDefinition.SetAttributeName(YEAR_KEY);
 yearAttributeDefinition.SetAttributeType(
 Aws::DynamoDB::Model::ScalarAttributeType::N);
 request.AddAttributeDefinitions(yearAttributeDefinition);

 Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
 yearAttributeDefinition.SetAttributeName(TITLE_KEY);
```

```
yearAttributeDefinition.SetAttributeType(
 Aws::DynamoDB::Model::ScalarAttributeType::S);
request.AddAttributeDefinitions(yearAttributeDefinition);

Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
 Aws::DynamoDB::Model::KeyType::HASH);
request.AddKeySchema(yearKeySchema);

Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
 Aws::DynamoDB::Model::KeyType::RANGE);
request.AddKeySchema(yearKeySchema);

Aws::DynamoDB::Model::ProvisionedThroughput throughput;
throughput.WithReadCapacityUnits(
 PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
 PROVISIONED_THROUGHPUT_UNITS);
request.SetProvisionedThroughput(throughput);
request.SetTableName(MOVIE_TABLE_NAME);

std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
 request);
if (!result.IsSuccess()) {
 if (result.GetError().GetErrorType() ==
 Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
 std::cout << "Table already exists." << std::endl;
 movieTableAlreadyExisted = true;
 }
 else {
 std::cerr << "Failed to create table: "
 << result.GetError().GetMessage();
 return false;
 }
}
}

// Wait for table to become active.
if (!movieTableAlreadyExisted) {
 std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
 << "' to become active...." << std::endl;
```

```
 if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
clientConfiguration)) {
 return false;
 }
 std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
 << std::endl;
 }

 return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
 \sa deleteMoviesDynamoDBTable()
 \param clientConfiguration: AWS client configuration.
 \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
 const Aws::Client::ClientConfiguration &clientConfiguration) {
 Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

 Aws::DynamoDB::Model::DeleteTableRequest request;
 request.SetTableName(MOVIE_TABLE_NAME);

 const Aws::DynamoDB::Model::DeleteTableOutcome &result =
dynamoClient.DeleteTable(
 request);
 if (result.IsSuccess()) {
 std::cout << "Your table \""
 << result.GetResult().GetTableDescription().GetTableName()
 << " was deleted.\n";
 }
 else {
 std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
 << std::endl;
 }

 return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
 \sa waitTableActive()
 \param waitTableActive: The DynamoDB table's name.
```

```
\param 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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[BatchExecuteStatement](#)中的。

## Go

## SDK for Go V2

 Note

還有更多關於 [GitHub](#)。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

執行一個情境，該情境建立資料表並執行多批 PartiQL 查詢。

```
// 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))
}
```

定義此範例中使用的影片結構。

```
// 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"])
}
```



## 建立執行 PartiQL 陳述式的結構和方法。

```
// 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
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考[BatchExecuteStatement](#)中的。

## Java

### 適用於 Java 2.x 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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);
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[BatchExecuteStatement](#)中的。

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

執行批次 PartiQL 陳述式。

```
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/q1-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.");
};

```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考 [BatchExecuteStatement](#) 中的。

## Kotlin

### 適用於 Kotlin 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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")
 }
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [BatchExecuteStatement](#) 中的 Kotlin API 參考。

## PHP

### 適用於 PHP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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,
],
],
]);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考[BatchExecuteStatement](#)中的。

## Python

### 適用於 Python (Boto3) 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

建立一個類別，該類別可以執行多批 PartiQL 陳述式。

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint

import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLBatchWrapper:
```

```

"""
Encapsulates a DynamoDB resource to run PartiQL statements.
"""

def __init__(self, dyn_resource):
 """
 :param dyn_resource: A Boto3 DynamoDB resource.
 """
 self.dyn_resource = dyn_resource

def run_partiql(self, statements, param_list):
 """
 Runs a PartiQL statement. A Boto3 resource is used even though
 `execute_statement` is called on the underlying `client` object because
the
 resource transforms input and output from plain old Python objects
(POPOs) to
 the DynamoDB format. If you create the client directly, you must do these
transforms yourself.

 :param statements: The batch of PartiQL statements.
 :param param_list: The batch of PartiQL parameters that are associated
with
 each statement. This list must be in the same order as
the
 statements.

 :return: The responses returned from running the statements, if any.
 """
 try:
 output = self.dyn_resource.meta.client.batch_execute_statement(
 Statements=[
 {"Statement": statement, "Parameters": params}
 for statement, params in zip(statements, param_list)
]
)
 except ClientError as err:
 if err.response["Error"]["Code"] == "ResourceNotFoundException":
 logger.error(
 "Couldn't execute batch of PartiQL statements because the
table "
 "does not exist."
)
 else:

```

```
 logger.error(
 "Couldn't execute batch of PartiQL statements. Here's why:
%s: %s",
 err.response["Error"]["Code"],
 err.response["Error"]["Message"],
)
 raise
 else:
 return output
```

執行一個情境，該情境建立資料表並批次執行 PartiQL 查詢。

```
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}")
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[BatchExecuteStatement](#)中的 Python (博托 3) API 參考。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

執行一個情境，該情境會建立資料表並執行批次 PartiQL 查詢。

```
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
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考[BatchExecuteStatement](#)中的。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用 PartiQL 和開發套件查詢 DynamoDB 資料表 AWS

下列程式碼範例示範如何：

- 透過執行 SELECT 陳述式取得項目。
- 透過執行 INSERT 陳述式新增項目。
- 透過執行 UPDATE 陳述式更新項目。
- 透過執行 DELETE 陳述式刪除項目。

### .NET

#### AWS SDK for .NET

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for .NET API 參考 [ExecuteStatement](#) 中的。

## C++

### 適用於 C++ 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
// 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;
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for C++ API 參考[ExecuteStatement](#)中的。

Go

SDK for Go V2

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

執行一個情境，該情境建立資料表並執行 PartiQL 查詢。

```
// 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))
}
```

定義此範例中使用的影片結構。



```
// 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"])
}
```

建立執行 PartiQL 陳述式的結構和方法。

```
// 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
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Go API 參考[ExecuteStatement](#)中的。

## Java

### 適用於 Java 2.x 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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());
}
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Java 2.x API 參考[ExecuteStatement](#)中的。

## JavaScript

適用於 JavaScript (v3) 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

執行單一 PartiQL 陳述式。

```
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.");
};
```

- 如需 API 詳細資訊，請參閱 AWS SDK for JavaScript API 參考[ExecuteStatement](#)中的。

## Kotlin

### 適用於 Kotlin 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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)
}
```

- 有關 API 的詳細信息，請參閱 AWS SDK [ExecuteStatement](#) 中的 Kotlin API 參考。

## PHP

### 適用於 PHP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

```
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,
]);
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for PHP API 參考 [ExecuteStatement](#) 中的。

## Python

適用於 Python (Boto3) 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在 [AWS 設定和執行程式碼範例儲存庫](#)。

建立一個類別，該類別可以執行 PartiQL 陳述式。

```
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
```

執行一個情境，該情境建立資料表並執行 PartiQL 查詢。

```
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}")
```

- 如需 API 的詳細資訊，請參閱AWS 開發套件[ExecuteStatement](#)中的 Python (博托 3) API 參考。

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

執行一個情境，該情境建立資料表並執行 PartiQL 查詢。

```
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
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Ruby API 參考[ExecuteStatement](#)中的。

## Rust

### 適用於 Rust 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

```
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(())
}
```

- 如需 API 的詳細資訊，請參閱 AWS SDK [ExecuteStatement](#) 中的 Rust API 參考資料。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用 SDK 為 DynamoDB 使用文件模型 AWS

下列程式碼範例顯示如何使用 DynamoDB 和 SDK 的文件模型執行建立、讀取、更新和刪除 (CRUD) 和批次作業。AWS

如需詳細資訊，請參閱[文件模型](#)。

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用文件模型執行 CRUD 操作。

```
///
/// <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);
 }
}
}
```

使用文件模型執行批次寫入操作。

```
/// <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();
}
}
```

使用文件模型掃描資料表。

```
/// <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)
 {
```

```
0. // Assume there is a price error. So we scan to find items priced <
 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}");
 }
}
```

使用文件模型搜尋和掃描資料表。

```
/// <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 name 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}");
 }
}
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用 SDK 為 DynamoDB 使用高階物件持續性模型 AWS

下列程式碼範例示範如何使用 DynamoDB 和 SDK 的物件持續性模型來執行建立、讀取、更新和刪除 (CRUD) 和批次作業。AWS

如需詳細資訊，請參閱[物件持久性模型](#)。

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[AWS 設定和執行程式碼範例儲存庫](#)。

使用高階物件持久性模型執行 CRUD 操作。

```
/// <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");
 }
 }
}
```

使用高階物件持久性模型執行批次寫入操作。

```
/// <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);
 }
}
```

使用高階物件持久性模型將任意資料映射至資料表。

```
/// <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);
 }
}
```

使用高階物件持久性模型搜尋和掃描資料表。

```
/// <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}");
 }
 }
}
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用開發套件的 DynamoDB 的無伺服器範例 AWS

下列程式碼範例說明如何搭配開發套件使用 AWS DynamoDB。

### 範例

- [從 DynamoDB 觸發程序叫用 Lambda 函數](#)
- [使用 DynamoDB 觸發程序報告 Lambda 函數的批次項目失敗](#)

## 從 DynamoDB 觸發程序叫用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，該函數會接收透過從 DynamoDB 串流接收記錄而觸發的事件。此函數會擷取 DynamoDB 承載並記錄記錄內容。

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 .NET 與 Lambda 一起使用 DynamoDB 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace AWSLambda_DDB;

public class Function
{
 public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
context)
 {
 context.Logger.LogInformation($"Beginning to process
{dynamoEvent.Records.Count} records...");

 foreach (var record in dynamoEvent.Records)
 {
 context.Logger.LogInformation($"Event ID: {record.EventID}");
 }
 }
}
```

```
 context.Logger.LogInformation($"Event Name: {record.EventName}");

 context.Logger.LogInformation(JsonSerializer.Serialize(record));
 }

 context.Logger.LogInformation("Stream processing complete.");
}
}
```

## Go

### SDK for Go V2

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Go 使用與 Lambda 一起使用 DynamoDB 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
error) {
 if len(event.Records) == 0 {
 return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
 LogDynamoDBRecord(record)
 }
}
```

```
 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

### 使用 Lambda 使用 JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
 console.log(JSON.stringify(event, null, 2));
 event.Records.forEach(record => {
 logDynamoDBRecord(record);
 });
};

const logDynamoDBRecord = (record) => {
 console.log(record.eventID);
 console.log(record.eventName);
 console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

## 使用 Lambda 使用 TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
export const handler = async (event, context) => {
 console.log(JSON.stringify(event, null, 2));
 event.Records.forEach(record => {
 logDynamoDBRecord(record);
 });
}
const logDynamoDBRecord = (record) => {
 console.log(record.eventID);
 console.log(record.eventName);
 console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

## PHP

### 適用於 PHP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 PHP 與 Lambda 一起使用 DynamoDB 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;
```

```
require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
 private StderrLogger $logger;

 public function __construct(StderrLogger $logger)
 {
 $this->logger = $logger;
 }

 /**
 * @throws JsonException
 * @throws \Bref\Event\InvalidLambdaEvent
 */
 public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
 {
 $this->logger->info("Processing DynamoDb table items");
 $records = $event->getRecords();

 foreach ($records as $record) {
 $eventName = $record->getEventName();
 $keys = $record->getKeys();
 $old = $record->getOldImage();
 $new = $record->getNewImage();

 $this->logger->info("Event Name:". $eventName. "\n");
 $this->logger->info("Keys:". json_encode($keys). "\n");
 $this->logger->info("Old Image:". json_encode($old). "\n");
 $this->logger->info("New Image:". json_encode($new));

 // TODO: Do interesting work based on the new data

 // Any exception thrown will be logged and the invocation will be
 marked as failed
 }

 $totalRecords = count($records);
 $this->logger->info("Successfully processed $totalRecords items");
 }
}

$logger = new StderrLogger();
```

```
return new Handler($logger);
```

## Python

### 適用於 Python (Boto3) 的 SDK

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用使用 Python 與 Lambda 一起使用動 DynamoDB 事件。

```
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
SPDX-License-Identifier: Apache-2.0

import json

def lambda_handler(event, context):
 print(json.dumps(event, indent=2))

 for record in event['Records']:
 log_dynamodb_record(record)

def log_dynamodb_record(record):
 print(record['eventID'])
 print(record['eventName'])
 print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

## Ruby

### 適用於 Ruby 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用紅寶石與 Lambda 一起使用 DynamoDB 事件。

```
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
SPDX-License-Identifier: Apache-2.0

def lambda_handler(event:, context:)
 return 'received empty event' if event['Records'].empty?

 event['Records'].each do |record|
 log_dynamodb_record(record)
 end

 "Records processed: #{event['Records'].length}"
end

def log_dynamodb_record(record)
 puts record['eventID']
 puts record['eventName']
 puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
end
```

## Rust

適用於 Rust 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 與 Lambda 一起使用 DynamoDB 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
 event::dynamodb::{Event, EventRecord},
};
```



```
// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

 let records = &event.payload.records;
 tracing::info!("event payload: {:?}",records);
 if records.is_empty() {
 tracing::info!("No records found. Exiting.");
 return Ok(());
 }

 for record in records{
 log_dynamo_dbrecord(record);
 }

 tracing::info!("Dynamo db records processed");

 // Prepare the response
 Ok(())
}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
 tracing::info!("EventId: {}", record.event_id);
 tracing::info!("EventName: {}", record.event_name);
 tracing::info!("DynamoDB Record: {:?}", record.change);
 Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
 tracing_subscriber::fmt()
 .with_max_level(tracing::Level::INFO)
 .with_target(false)
```

```
.without_time()
.init();

let func = service_fn(function_handler);
lambda_runtime::run(func).await?;
Ok(())
}
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用 DynamoDB 觸發程序報告 Lambda 函數的批次項目失敗

下列程式碼範例說明如何針對接收來自 DynamoDB 串流之事件的 Lambda 函數實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

.NET

AWS SDK for .NET

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 .NET 報告使用 Lambda 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))
]
```

```
namespace AWSLambda_DDB;

public class Function
{
 public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

 {
 context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
 List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
 StreamsEventResponse streamsEventResponse = new StreamsEventResponse();


 foreach (var record in dynamoEvent.Records)
 {
 try
 {
 var sequenceNumber = record.Dynamodb.SequenceNumber;
 context.Logger.LogInformation(sequenceNumber);
 }
 catch (Exception ex)
 {
 context.Logger.LogError(ex.Message);
 batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
 }
 }

 if (batchItemFailures.Count > 0)
 {
 streamsEventResponse.BatchItemFailures = batchItemFailures;
 }

 context.Logger.LogInformation("Stream processing complete.");
 return streamsEventResponse;
 }
}
```

## Go

## SDK for Go V2

 Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Go 使用 Lambda 報告批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
(*BatchResult, error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
 // Process your record
 curRecordSequenceNumber = record.Change.SequenceNumber
 }

 if curRecordSequenceNumber != "" {
 batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
curRecordSequenceNumber})
 }
}
```

```
}

batchResult := BatchResult{
 BatchItemFailures: batchItemFailures,
}

return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

## Java

適用於 Java 2.x 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Java 使用 Lambda 報告批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 Serializable> {

 @Override
```

```
public StreamsEventResponse handleRequest(DynamodbEvent input, Context
context) {

 List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
 String curRecordSequenceNumber = "";

 for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
input.getRecords()) {
 try {
 //Process your record
 StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
 curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

 } catch (Exception e) {
 /* Since we are working with streams, we can return the failed
item immediately.
 Lambda will immediately begin to retry processing from this
failed item onwards. */
 batchItemFailures.add(new
StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
 return new StreamsEventResponse(batchItemFailures);
 }
 }

 return new StreamsEventResponse();
}
}
```

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

### 使用 Lambda 使用報告批次項目失敗 JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
export const handler = async (event) => {
 const records = event.Records;
 let curRecordSequenceNumber = "";

 for (const record of records) {
 try {
 // Process your record
 curRecordSequenceNumber = record.dynamodb.SequenceNumber;
 } catch (e) {
 // Return failed record's sequence number
 return { batchItemFailures: [{ itemIdentifier:
curRecordSequenceNumber }] };
 }
 }

 return { batchItemFailures: [] };
};
```

## 使用 Lambda 使用報告批次項目失敗 TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBBatchItemFailure, DynamoDBStreamEvent } from "aws-lambda";

export const handler = async (event: DynamoDBStreamEvent):
Promise<DynamoDBBatchItemFailure[]> => {

 const batchItemsFailures: DynamoDBBatchItemFailure[] = []
 let curRecordSequenceNumber

 for(const record of event.Records) {
 curRecordSequenceNumber = record.dynamodb?.SequenceNumber

 if(curRecordSequenceNumber) {
 batchItemsFailures.push({
 itemIdentifier: curRecordSequenceNumber
 })
 }
 }
}
```

```
 return batchItemsFailures
}
```

## PHP

### 適用於 PHP 的開發套件

#### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 PHP 使用 Lambda 報告批次項目失敗。

```
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
SPDX-License-Identifier: Apache-2.0
<?php

using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
 private StderrLogger $logger;
 public function __construct(StderrLogger $logger)
 {
 $this->logger = $logger;
 }

 /**
 * @throws JsonException
 * @throws \Bref\Event\InvalidLambdaEvent
 */
 public function handle(mixed $event, Context $context): array
```



```
{
 $dynamoDbEvent = new DynamoDbEvent($event);
 $this->logger->info("Processing records");

 $records = $dynamoDbEvent->getRecords();
 $failedRecords = [];
 foreach ($records as $record) {
 try {
 $data = $record->getData();
 $this->logger->info(json_encode($data));
 // TODO: Do interesting work based on the new data
 } catch (Exception $e) {
 $this->logger->error($e->getMessage());
 // failed processing the record
 $failedRecords[] = $record->getSequenceNumber();
 }
 }
 $totalRecords = count($records);
 $this->logger->info("Successfully processed $totalRecords records");

 // change format for the response
 $failures = array_map(
 fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
 $failedRecords
);

 return [
 'batchItemFailures' => $failures
];
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

適用於 Python (Boto3) 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Python 使用 Lambda 報告批次項目失敗。

```
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
SPDX-License-Identifier: Apache-2.0
def handler(event, context):
 records = event.get("Records")
 curRecordSequenceNumber = ""

 for record in records:
 try:
 # Process your record
 curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
 except Exception as e:
 # Return failed record's sequence number
 return {"batchItemFailures":[{"itemIdentifier":
curRecordSequenceNumber}]}

 return {"batchItemFailures":[]}
```

## Ruby

適用於 Ruby 的開發套件

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用紅寶石使用 Lambda 報告批次項目失敗。

```
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
 records = event["Records"]
 cur_record_sequence_number = ""

 records.each do |record|
 begin
 # Process your record
 cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
 rescue StandardError => e
 # Return failed record's sequence number
 return {"batchItemFailures" => [{"itemIdentifier" =>
cur_record_sequence_number}]}
 end
 end

 {"batchItemFailures" => []}
end
```

## Rust

適用於 Rust 的 SDK

### Note

還有更多關於 GitHub。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 使用 Lambda 報告批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
 event::dynamodb::{Event, EventRecord, StreamRecord},
 streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
```

```
/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
 let stream_record: &StreamRecord = &record.change;

 // process your stream record here...
 tracing::info!("Data: {:?}", stream_record);

 Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
Result<DynamoDbEventResponse, Error> {
 let mut response = DynamoDbEventResponse {
 batch_item_failures: vec![],
 };

 let records = &event.payload.records;

 if records.is_empty() {
 tracing::info!("No records found. Exiting.");
 return Ok(response);
 }

 for record in records {
 tracing::info!("EventId: {}", record.event_id);

 // Couldn't find a sequence number
 if record.change.sequence_number.is_none() {
 response.batch_item_failures.push(DynamoDbBatchItemFailure {
 item_identifer: Some("").to_string(),
 });
 return Ok(response);
 }

 // Process your record here...
 if process_record(record).is_err() {
 response.batch_item_failures.push(DynamoDbBatchItemFailure {
 item_identifer: record.change.sequence_number.clone(),
 });
 /* Since we are working with streams, we can return the failed item
 immediately.
```

```
 Lambda will immediately begin to retry processing from this failed
 item onwards. */
 return Ok(response);
 }
}

tracing::info!("Successfully processed {} record(s)", records.len());

Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
 tracing_subscriber::fmt()
 .with_max_level(tracing::Level::INFO)
 // disable printing the name of the module in every log line.
 .with_target(false)
 // disabling time is handy because CloudWatch will add the ingestion
 time.
 .without_time()
 .init();

 run(service_fn(function_handler)).await
}
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用開發套件的 DynamoDB 跨服務範例 AWS

下列範例應用程式使用 AWS 開發套件將 DynamoDB 與其他應用程式結合。AWS 服務每個範例都包含一個連結 GitHub，您可以在其中找到如何設定和執行應用程式的指示。

### 範例

- [建置應用程式以將資料提交至 DynamoDB 資料表](#)
- [建立 API Gateway REST API 以追蹤 COVID-19 資料](#)
- [使用 Step Functions 建立傳訊應用程式](#)
- [建立相片資產管理應用程式，讓使用者以標籤管理相片](#)

- [建立 Web 應用程式以追蹤 DynamoDB 資料](#)
- [使用 API Gateway 建立 websocket 聊天應用程式](#)
- [使用開發套件使用 Amazon 重新認知功能偵測影像中的個人防護裝置 AWS](#)
- [從瀏覽器調用 Lambda 函數](#)
- [使用開發套件監控 Amazon DynamoDB 的效能 AWS](#)
- [使用 SDK 儲存 EXIF 和其他影像資訊 AWS](#)
- [使用 API Gateway 來調用 Lambda 函數](#)
- [使用 Step Functions 呼叫 Lambda 函數](#)
- [使用排程事件來調用 Lambda 函數](#)

## 建置應用程式以將資料提交至 DynamoDB 資料表

下列程式碼範例顯示如何建置應用程式，以將資料提交至 Amazon DynamoDB 資料表，並在使用者更新資料表時通知您。

### Java

適用於 Java 2.x 的 SDK

示範如何建立動態 Web 應用程式，以使用 Amazon DynamoDB Java API 提交資料，以及使用 Amazon Simple Notification Service Java API 傳送文字訊息。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- DynamoDB
- Amazon SNS

### JavaScript

適用於 JavaScript (v3) 的開發套件

此範例說明如何建置應用程式，讓使用者將資料提交至 Amazon DynamoDB 資料表，以及使用 Amazon Simple Notification Service (Amazon SNS) 傳送文字訊息給管理員。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例也可在 [AWS SDK for JavaScript v3 開發人員指南](#) 中取得。

此範例中使用的服務

- DynamoDB
- Amazon SNS

## Kotlin

適用於 Kotlin 的 SDK

示範如何使用 Amazon DynamoDB Kotlin API 建立提交資料的原生 Android 應用程式，並使用 Amazon SNS Kotlin API 傳送文字訊息。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- DynamoDB
- Amazon SNS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 建立 API Gateway REST API 以追蹤 COVID-19 資料

以下程式碼範例示範如何建立 REST API，此 API 使用虛構資料模擬追蹤美國 COVID-19 每日病例的系統。

## Python

適用於 Python (Boto3) 的 SDK

示範如何搭配使用 AWS Chalice，AWS SDK for Python (Boto3) AWS Lambda 以建立使用 Amazon API Gateway 和 Amazon DynamoDB 的無伺服器 REST API。REST API 使用虛構資料模擬追蹤美國 COVID-19 每日病例的系統。了解如何：

- 使用 AWS Chalice 定義 Lambda 函數中的路由，這些函數會呼叫以處理透過 API Gateway 傳送的 REST 要求。
- 使用 Lambda 函數在 DynamoDB 資料表中擷取和存放資料，以便為 REST 請求提供服務。

- 在 AWS CloudFormation 範本中定義資料表結構和資訊安全角色資源。
- 使用 AWS Chalice 並 CloudFormation 封裝和部署所有必要的資源。
- 用 CloudFormation 於清理所有創建的資源。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- API Gateway
- AWS CloudFormation
- DynamoDB
- Lambda

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用 Step Functions 建立傳訊應用程式

下列程式碼範例會示範如何建立從資料庫資料表擷取訊息記錄的 AWS Step Functions Messenger 應用程式。

### Python

適用於 Python (Boto3) 的 SDK

示範如何使用 AWS SDK for Python (Boto3) 與建立信 AWS Step Functions 使應用程式，以從 Amazon DynamoDB 表擷取訊息記錄，並透過 Amazon Simple Queue Service (Amazon SQS) 傳送這些記錄。狀態機與一個 AWS Lambda 功能集成在數據庫中掃描未發送的消息。

- 建立從 Amazon DynamoDB 資料表擷取和更新訊息記錄的狀態機器。
- 更新狀態機器定義，以便也向 Amazon Simple Queue Service (Amazon SQS) 傳送訊息。
- 開始和停用狀態機器執行。
- 使用服務整合從狀態機器連接至 Lambda、DynamoDB 和 Amazon SQS。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- DynamoDB



- Lambda
- Amazon SQS
- Step Functions

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 建立相片資產管理應用程式，讓使用者以標籤管理相片

下列程式碼範例示範如何建立無伺服器應用程式，讓使用者以標籤管理相片。

### .NET

#### AWS SDK for .NET

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

如要深入探索此範例的來源，請參閱[AWS 社群](#)上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

### C++

#### 適用於 C++ 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## Java

適用於 Java 2.x 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 [GitHub](#)。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## JavaScript

適用於 JavaScript (v3) 的開發套件

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 [GitHub](#)。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## Kotlin

適用於 Kotlin 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 [GitHub](#)。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## PHP

適用於 PHP 的開發套件

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 [GitHub](#)。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## Rust

適用於 Rust 的 SDK

顯示如何開發照片資產管理應用程式，以便使用 Amazon Rekognition 偵測圖片中的標籤，並將其儲存以供日後擷取。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例 [GitHub](#)。

如要深入探索此範例的來源，請參閱 [AWS 社群](#) 上的文章。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 建立 Web 應用程式以追蹤 DynamoDB 資料

下列程式碼範例說明如建立 Web 應用程式追蹤 Amazon DynamoDB 資料表中的工作項目，並且使用 Amazon Simple Email Service (Amazon SES) 傳送報告。

## .NET

### AWS SDK for .NET

說明如何使用 Amazon DynamoDB .NET API 來建立可追蹤 DynamoDB 工作資料的動態 Web 應用程式。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- DynamoDB
- Amazon SES

## Java

### 適用於 Java 2.x 的 SDK

說明如何使用 Amazon DynamoDB API 來建立可追蹤 DynamoDB 工作資料的動態 Web 應用程式。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- DynamoDB
- Amazon SES

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

說明如何使用 Amazon DynamoDB API 來建立可追蹤 DynamoDB 工作資料的動態 Web 應用程式。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- DynamoDB

- Amazon SES

## Kotlin

### 適用於 Kotlin 的 SDK

說明如何使用 Amazon DynamoDB API 來建立可追蹤 DynamoDB 工作資料的動態 Web 應用程式。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- DynamoDB
- Amazon SES

## Python

### 適用於 Python (Boto3) 的 SDK

示範如何使用建立 REST 服務，AWS SDK for Python (Boto3) 以追蹤 Amazon DynamoDB 中的工作項目，以及使用亞馬遜簡單電子郵件服務 (Amazon SES) 傳送電子郵件報告。這個範例使用 Flask Web 框架來處理 HTTP 路由，並與 React 網頁整合以呈現功能完整的 Web 應用程式。

- 建置整合的 AWS 服務燒瓶 REST 服務。
- 讀取、寫入和更新 DynamoDB 資料表中儲存的工作項目。
- 使用 Amazon SES 傳送工作項目的電子郵件報告。

如需有關如何設定和執行的完整原始程式碼和指示，請參閱上的[AWS 程式碼範例儲存庫](#)中的完整範例 GitHub。

此範例中使用的服務

- DynamoDB
- Amazon SES

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用 API Gateway 建立 websocket 聊天應用程式

下列程式碼範例示範如何建立由建置於 Amazon API Gateway 上的 websocket API 提供服務的聊天應用程式。

### Python

適用於 Python (Boto3) 的 SDK

示範如何使用 AWS SDK for Python (Boto3) 與 Amazon API Gateway V2 搭配使用，以建立與亞馬遜動態 B 整合的 AWS Lambda 網路通訊端 API。

- 建立由 API Gateway 提供服務的 websocket API。
- 定義 Lambda 處理常式，該常式將連接存放在 DynamoDB 中，並將訊息傳送給其他聊天參與者。
- 連接至 websocket 聊天應用程式，並使用 Websockets 套件傳送訊息。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用開發套件使用 Amazon 重新認知功能偵測影像中的個人防護裝置 AWS

以下程式碼範例說明如何建置可使用 Amazon Rekognition 在映像中偵測個人防護裝備 (PPE) 的應用程式。

### Java

適用於 Java 2.x 的 SDK

說明如何建立使用個人防護裝備偵測影像的 AWS Lambda 功能。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

### 此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

示範如何搭配使用 Amazon Rekognition AWS SDK for JavaScript 來建立應用程式，以偵測位於亞馬遜簡單儲存服務 (Amazon S3) 儲存貯體中的映像中的個人防護設備 (PPE)。該應用程式將結果儲存到 Amazon DynamoDB 資料表中，並使用 Amazon Simple Email Service (Amazon SES) 向管理員傳送包含結果的電子郵件通知。

了解如何：

- 使用 Amazon Cognito 建立未經身分驗證的使用者。
- 使用 Amazon Rekognition 分析映像中是否具有 PPE。
- 驗證 Amazon SES 的電子郵件地址。
- 以結果更新 DynamoDB 資料表。
- 使用 Amazon SES 傳送電子郵件通知。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

### 此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 從瀏覽器調用 Lambda 函數

下列程式碼範例會示範如何從瀏覽器叫用 AWS Lambda 函數。



## JavaScript

### 適用於 JavaScript (v2) 的開發套件

您可以建立以瀏覽器為基礎的應用程式，使用 AWS Lambda 函數更新具有使用者選擇的 Amazon DynamoDB 表格。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- DynamoDB
- Lambda

### 適用於 JavaScript (v3) 的開發套件

您可以建立以瀏覽器為基礎的應用程式，使用 AWS Lambda 函數更新具有使用者選擇的 Amazon DynamoDB 表格。此應用程序使用 AWS SDK for JavaScript v3。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- DynamoDB
- Lambda

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用開發套件監控 Amazon DynamoDB 的效能 AWS

下列程式碼範例顯示如何設定應用程式使用 DynamoDB 來監視效能。

### Java

#### 適用於 Java 2.x 的 SDK

此範例顯示如何設定 Java 應用程式以監視 DynamoDB 的效能。應用程式會將測量結果資料傳送至您 CloudWatch 可以監視效能的位置。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- CloudWatch
- DynamoDB

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用 SDK 儲存 EXIF 和其他影像資訊 AWS

以下程式碼範例顯示做法：

- 從 JPG、JPEG 或 PNG 檔案中取得 EXIF 資訊。
- 將映像檔案上傳至 Amazon S3 儲存貯體。
- 使用 Amazon Rekognition 識別檔案中的三個主要屬性 (標籤)。
- 將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表。

### Rust

#### 適用於 Rust 的 SDK

從 JPG、JPEG 或 PNG 檔案獲取 EXIF 資訊，將映像檔案上傳至 Amazon S3 儲存貯體，使用 Amazon Rekognition 識別三個主要屬性 (Amazon Rekognition 中的標籤)，然後將 EXIF 和標籤資訊新增至區域中的 Amazon DynamoDB 資料表中。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- DynamoDB
- Amazon Rekognition
- Amazon S3

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用 API Gateway 來調用 Lambda 函數

下列程式碼範例說明如何建立 Amazon API Gateway 叫用的 AWS Lambda 函數。

## Java

### 適用於 Java 2.x 的 SDK

示範如何使用 Lambda Java 執行階段 API 建立 AWS Lambda 函數。此範例會呼叫不同的 AWS 服務來執行特定使用案例。此範例示範如何建立 Amazon API Gateway 調用的 Lambda 函數，該函數會掃描 Amazon DynamoDB 資料表中的工作週年紀念日，並使用 Amazon Simple Notification Service (Amazon SNS) 傳送文字訊息給您的員工，在他們的週年紀念日向他們道賀。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

## JavaScript

### 適用於 JavaScript (v3) 的開發套件

示範如何使用 Lambda JavaScript 執行階段 API 建立 AWS Lambda 函數。此範例會呼叫不同的 AWS 服務來執行特定使用案例。此範例示範如何建立 Amazon API Gateway 調用的 Lambda 函數，該函數會掃描 Amazon DynamoDB 資料表中的工作週年紀念日，並使用 Amazon Simple Notification Service (Amazon SNS) 傳送文字訊息給您的員工，在他們的週年紀念日向他們道賀。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例也可在 [AWS SDK for JavaScript v3 開發人員指南](#)中取得。

此範例中使用的服務

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用 Step Functions 呼叫 Lambda 函數

下列程式碼範例會示範如何建立依序叫用 AWS Lambda 函式的 AWS Step Functions 狀態機器。

### Java

#### 適用於 Java 2.x 的 SDK

說明如何使用 AWS Step Functions 和建立 AWS 無伺服器工作流程。AWS SDK for Java 2.x 每個工作流程步驟都是使用 AWS Lambda 函數來實作。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例中使用的服務

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

### JavaScript

#### 適用於 JavaScript (v3) 的開發套件

說明如何使用 AWS Step Functions 和建立 AWS 無伺服器工作流程。AWS SDK for JavaScript 每個工作流程步驟都是使用 AWS Lambda 函數來實作。

Lambda 是一項運算服務，可讓您執行程式碼，而無需佈建或管理伺服器。Step Functions 是一種無伺服器協同運作服務，可讓您結合 Lambda 函數和其他 AWS 服務來建置關鍵業務應用程式。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例也可在 [AWS SDK for JavaScript v3 開發人員指南](#) 中取得。

此範例中使用的服務

- DynamoDB

- Lambda
- Amazon SES
- Step Functions

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱[搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

## 使用排程事件來調用 Lambda 函數

下列程式碼範例說明如何建立 Amazon EventBridge 排程事件所叫用的 AWS Lambda 函數。

### Java

#### 適用於 Java 2.x 的 SDK

說明如何建立叫用 AWS Lambda 函數的 Amazon EventBridge 排程事件。設定 EventBridge 為在叫用 Lambda 函數時使用 cron 運算式來排程。在此範例中，您會使用 Lambda Java 執行期 API 建立 Lambda 函數。此範例會呼叫不同的 AWS 服務來執行特定使用案例。此範例示範如何建立應用程式，將行動裝置文字訊息傳送給員工，在他們的週年紀念日向他們道賀。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

#### 此範例中使用的服務

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

### JavaScript

#### 適用於 JavaScript (v3) 的開發套件

說明如何建立叫用 AWS Lambda 函數的 Amazon EventBridge 排程事件。設定 EventBridge 為在叫用 Lambda 函數時使用 cron 運算式來排程。在此範例中，您可以使用 Lambda JavaScript 執行階段 API 建立 Lambda 函數。此範例會呼叫不同的 AWS 服務來執行特定使用案例。此範例示範如何建立應用程式，將行動裝置文字訊息傳送給員工，在他們的週年紀念日向他們道賀。

有關如何設置和運行的完整源代碼和說明，請參閱中的完整示例[GitHub](#)。

此範例也可在 [AWS SDK for JavaScript v3 開發人員指南](#) 中取得。

此範例中使用的服務

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配開發套件使用 DynamoDB AWS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

# Amazon DynamoDB 中的安全與合規

雲安全 AWS 是最高的優先級。身為 AWS 客戶，您可以從資料中心和網路架構中獲益，該架構專為滿足對安全性最敏感的組織的需求而打造。

安全是 AWS 與您之間共同承擔的責任。[共同責任模型](#) 將此描述為雲端的安全和雲端內的安全：

- 雲端的安全性 — AWS 負責保護在 AWS 雲端中執行 AWS 服務的基礎架構。AWS 還為您提供可以安全使用的服務。第三方稽核人員定期檢測及驗證安全的效率也是我們 [AWS 合規計劃](#) 的一部分。若要進一步了解適用於 DynamoDB 的合規計劃，請參閱 [合規計劃範圍內的 AWS 服務](#)。
- 雲端中的安全性 — 您的責任取決於您使用的 AWS 服務。您也必須對資料敏感度、組織要求，以及適用法律和法規等其他因素負責。

本文件會協助您了解使用 DynamoDB 時共同的責任模型的適用情形。下列主題將示範如何設定 DynamoDB 以達到您的安全和合規目標。您也將學習如何使用其他可 AWS 協助您監控和保護 DynamoDB 資源的服務。

## 主題

- [AWS 亞馬遜動態 B 的受管政策](#)
- [針對 DynamoDB 使用以資源為基礎的政策](#)
- [DynamoDB 中的資料保護](#)
- [AWS Identity and Access Management \(IAM\)](#)
- [DynamoDB 的業界合規驗證](#)
- [Amazon DynamoDB 中的資料復原及災難復原功能](#)
- [Amazon DynamoDB 中的基礎設施安全性](#)
- [AWS PrivateLink 適用於動態支援](#)
- [Amazon DynamoDB 中的組態與漏洞分析](#)
- [Amazon DynamoDB 的安全最佳實務](#)

## AWS 亞馬遜動態 B 的受管政策

DynamoDB 使用 AWS 受管政策來定義服務執行特定動作所需的一組權限。DynamoDB 會維護和更新其 AWS 受管理的政策。您無法變更 AWS 受管理原則中的權限。如需 AWS 受管政策的詳細資訊，請參閱 IAM 使用者指南中的 [AWS 受管政策](#)。

DynamoDB 偶爾可能會向 AWS 受管理的原則新增其他權限，以支援新功能。此類型的更新會影響已連接政策的所有身分識別 (使用者、群組和角色)。當新功能啟動或新作業可用時，最有可能更新 AWS 受管理的原則。DynamoDB 不會從 AWS 受管政策移除權限，因此政策更新不會破壞您現有的權限。

## AWS 受管政策:DynamoDB ReplicationServiceRolePolicy

您無法將 `DynamoDBReplicationServiceRolePolicy` 政策附加至 IAM 實體。此政策會連接到服務連結角色，而此角色可讓 DynamoDB 代表您執行動作。如需詳細資訊，請參閱[搭配全域資料表使用 IAM](#)。

此政策會授予許可，允許服務連結角色執行全域資料表複本之間的資料。也會授與管理許可，以代表您管理全域資料表複本。

### 許可詳細資訊

此政策會授予下列許可：

- `dynamodb` – 執行資料複寫並管理資料表複本。
- `application-autoscaling`— 檢索和管理表格 AutoScaling 設置
- `account` – 擷取區域狀態以評估複本可存取性。
- `iam`— 在服務連結角色不存 AutoScaling 在的情況下，建立應用程式的服務連結角色。

您可以在[這裡](#)找到此受管政策的定義。

## AWS 受管政策：AmazonDynamoDB ReadOnlyAccess

您可將 `AmazonDynamoDBReadOnlyAccess` 政策連接到 IAM 身分。

此政策授予對亞馬遜動態 B 的唯讀存取權。

### 許可詳細資訊

此政策包含以下許可：

- `Amazon DynamoDB`— 提供對亞馬遜動態 B 的唯讀存取權。
- `Amazon DynamoDB Accelerator (DAX)`— 提供 Amazon DynamoDB 加速器 (DAX) 的唯讀存取權。
- `Application Auto Scaling`— 允許主參與者從 Application Auto Scaling 檢視組態。這是必要的，使用者才能檢視附加至資料表的自動調度資源調整原則。



- CloudWatch— 允許主參與者檢視中 CloudWatch 配置的測量結果資料和警示。這是必要的，因此使用者可以檢視已針對表格設定的可計費表格大小和 CloudWatch 警示。
- AWS Data Pipeline— 可讓主參與者檢視 AWS Data Pipeline 及關聯的物件。
- Amazon EC2— 允許主體檢視 Amazon EC2 虛擬私人雲端、子網路和安全群組。
- IAM— 允許主體檢視 IAM 角色。
- AWS KMS— 允許主參與者檢視中配置的 AWS KMS 金鑰。這是必要的，以使用戶可 AWS KMS keys 以查看他們在其帳戶中創建和管理。
- Amazon SNS— 允許主體按主題列出 Amazon SNS 主題和訂閱。
- AWS Resource Groups— 可讓主參與者檢視資源群組及其查詢。
- AWS Resource Groups Tagging— 允許主參與者列出區域中所有已標記或先前標記的資源。
- Kinesis— 允許主參與者檢視 Kinesis 資料串流說明。
- Amazon CloudWatch Contributor Insights— 允許主參與者檢視參與者見解規則所收集的  
時間序列資料。

若要以 JSON 格式檢閱原則，請參閱 [AmazonDynamoDB ReadOnlyAccess](#)。

## 受管政策的 DynamoDB 更新 AWS

此表格顯示 DynamoDB AWS 存取管理原則的更新。

變更	描述	變更日期
AmazonDynamoDBReadOnlyAccess 更新現有政策	AmazonDynamoDBReadOnlyAccess 新增權限 dynamodb:GetResourcePolicy。此權限可讓您存取附加至 DynamoDB 資源的讀取以資源為基礎的原則。	2024年3月20日
DynamoDBReplicationServiceRolePolicy 更新現有政策	DynamoDBReplicationServiceRolePolicy 新增權限 dynamodb:GetResourcePolicy。此權限允許服務連結角	2023年12月15日

變更	描述	變更日期
	色讀取附加至 DynamoDB 資源的以資源為基礎的政策。	
DynamoDBReplicationServiceRolePolicy 更新現有政策	DynamoDBReplicationServiceRolePolicy 新增權限 <code>account:ListRegions</code> 。此權限允許服務連結角色評估複本的存取	2023 年 5 月 10 日
DynamoDBReplicationServiceRolePolicy 已新增至受管政策清單	已新增有關 DynamoDB 全域資料表服務連結角色使用之受管政策 <code>DynamoDBReplicationServiceRolePolicy</code> 的資訊。	2023 年 5 月 10 日
DynamoDB 全域資料表已開始追蹤其變更	DynamoDB 全域資料表開始追蹤其 AWS 受管理原則的變更。	2023 年 5 月 10 日

## 針對 DynamoDB 使用以資源為基礎的政策

DynamoDB 針對資料表、索引和串流支援以資源為基礎的政策。以資源為基礎的策略可讓您定義存取權限，方法是指定每個資源的存取權限，以及允許這些資源對每個資源執行的動作。

您可以將以資源為基礎的政策附加至 DynamoDB 資源，例如資料表或串流。在此政策中，您可以為 Identity and Access Management (IAM) [主體指定許可](#)，[這些主體](#) 可以對這些 DynamoDB 資源執行特定動作。例如，附加至資料表的原則將包含資料表及其索引的存取權限。因此，以資源為基礎的政策可以透過在資源層級定義權限，協助您簡化 DynamoDB 表、索引和串流的存取控制。您可以附加至 DynamoDB 資源的原則大小上限為 20 KB。

使用以資源為基礎的政策的一大好處是簡化跨帳戶存取控制，以便為不同的 IAM 主體提供跨帳戶存取權。AWS 帳戶如需詳細資訊，請參閱 [跨帳戶存取的資源型政策](#)。

以資源為基礎的政策也支援與 [IAM 存取分析器](#) 外部存取分析器和 [封鎖公用存取 \(BPA\)](#) 功能的整合。IAM 存取分析器報告對以資源為基礎的政策中指定的外部實體的跨帳戶存取。它還提供了可見

性，以幫助您調整權限並符合最低權限原則。BPA 可協助您防止公開存取 DynamoDB 表、索引和串流，並且會在以資源為基礎的原則建立和修改工作流程中自動啟用。

## 主題

- [使用以資源為基礎的策略建立資料表](#)
- [將策略附加到現有表格](#)
- [將資源型政策附加至串流](#)
- [從表格中移除以資源為基礎的政策](#)
- [跨帳戶存取以資源為基礎的政策](#)
- [使用資源型原則封鎖公用存取](#)
- [資源型政策支援的 API 作業](#)
- [使用 IAM 身分型政策和以 DynamoDB 資源為基礎的政策進行授權](#)
- [資源型政策範例](#)
- [資源型政策考量事項](#)
- [資源型政策最佳做法](#)

## 使用以資源為基礎的策略建立資料表

您可以在使用 DynamoDB 主控台、[CreateTable](#) API、[AWS SDK](#) 或範本建立表格時新增以資源為基礎的政策。AWS CLI AWS CloudFormation

### AWS CLI

下列範例會建立 *MusicCollection* 使用 create-table AWS CLI 命令名稱的資料表。此命令也包含將以資源為基礎的原則新增至資料表的 resource-policy 參數。此原則可讓使用者 *John* 在資料表上執行 [RestoreTableToPointInTimeGetItem](#)、和 [PutItem](#) API 動作。

請記住將 ## 文本替換為特定於資源的信息。

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

```

```
"{
 \"Version\": \"2012-10-17\",
 \"Statement\": [
 {
 \"Effect\": \"Allow\",
 \"Principal\": {
 \"AWS\": \"arn:aws:iam::123456789012:user/John\"
 },
 \"Action\": [
 \"dynamodb:RestoreTableToPointInTime\",
 \"dynamodb:GetItem\",
 \"dynamodb:DescribeTable\"
],
 \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
 }
]
}"
```

## AWS Management Console

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在儀表板上，選擇 [建立資料表]。
3. 在表格詳細資訊中，輸入表格名稱、分割區索引鍵和排序索引鍵詳細資訊。
4. 在 [表格設定] 中選擇 [自訂設定]。
5. (選擇性) 指定表格類別、容量計算器、讀取/寫入容量設定、次要索引、靜態加密和刪除保護的選項。
6. 在以資源為基礎的策略中，新增政策以定義表格及其索引的存取權限。在此原則中，您可以指定有權存取這些資源的人員，以及允許這些資源對每個資源執行的動作。若要新增原則，請執行下列其中一個動作：
  - 輸入或貼上 JSON 政策文件。如需 IAM 政策語言的詳細資訊，請參閱 [IAM 使用者指南中的使用 JSON 編輯器](#) 建立政策。

### Tip

若要在 Amazon DynamoDB 開發人員指南中查看以資源為基礎的政策範例，請選擇政策範例。

- 選擇「新增陳述式」以新增陳述式，並在提供的欄位中輸入資訊。針對您想要新增的任意數量陳述式重複此步驟。

**⚠ Important**

儲存原則之前，請務必先解決任何安全性警告、錯誤或建議。

下列 IAM 政策範例可讓使用者 *John* 在資料表上執行 [RestoreTableToPointInTimeGetItem](#)、和 [PutItem](#) API 動作 *MusicCollection*。

請記住將##文本替換為特定於資源的信息。

```
{
 "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. (選用) 選擇右下角的 Preview external access (預覽外部存取)，以預覽新政策會如何影響資源的公開和跨帳戶存取權。在儲存政策之前，您可以檢查它是否引入新的 IAM Access Analyzer 問題清單，或是解決現有的問題清單。如果您沒有看到作用中的分析器，請選擇 Go to Access Analyzer (移至 Access Analyzer)，以在 IAM Access Analyzer 中 [建立帳戶分析器](#)。如需詳細資訊，請參閱 [預覽存取權](#)。
8. 選擇 建立資料表。

## AWS CloudFormation 範本

## Using the AWS::DynamoDB::Table resource

下列 CloudFormation 範本會使用 [:: DynamoDB::AWS:](#) 資料表資源建立包含串流的資料表。此範本還包括附加至資料表和串流的資源型政策。

```
{
 "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",
```



```

"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:UpdateItem\",
 \"dynamodb:UpdateTable\"
],
 \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
 }
]
}"

```

## AWS CLI 有條件地更新現有策略的範例

若要有條件地更新資料表的現有資源型政策，您可以使用選 `expected-revision-id` 用參數。下列範例只會在 DynamoDB 中存在原則，且其目前修訂 ID 與提供 `expected-revision-id` 的參數相符時，才會更新該原則。

```

aws dynamodb put-resource-policy \
--resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
--expected-revision-id 1709841168699 \
--policy \
 "{
 \"Version\": \"2012-10-17\",
 \"Statement\": [
 {
 \"Effect\": \"Allow\",
 \"Principal\": {
 \"AWS\": \"arn:aws:iam::111122223333:user/John\"
 },
 \"Action\": [
 \"dynamodb:GetItem\",
 \"dynamodb:UpdateItem\",
 \"dynamodb:UpdateTable\"
],
 \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
 }
]
 }"

```

## AWS Management Console

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。

2. 從儀表板中選擇現有表格。
3. 瀏覽至「權限」索引標籤，然後選擇「建立表格原則」。
4. 在以資源為基礎的策略編輯器中，新增您要附加的策略，然後選擇 [建立策略]。

下列 IAM 政策範例可讓使用者 *John* 在名為的 [GetItem](#) 現有資料表上執行 [UpdateItem](#)、[UpdateTable](#) API 動作 *MusicCollection*。 [PutItem](#)

請記住將 ## 文本替換為特定於資源的信息。

```
{
 "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

下列 IAM 政策範例使用此方 `putResourcePolicy` 法將以資源為基礎的政策附加至現有資料表。此原則可讓使用者對現有資料表執行 [GetItem](#) API 動作。

```
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" +
 " \"Action\": [\n" +
 " \"dynamodb:GetItem\"\n" +
 "],\n" +
 " \"Resource\": \"\" + tableArn + "\"\n" +
 " }\n" +
 "]\n" +
 "}";
}
}

```

## 將資源型政策附加至串流

您可以使用 [DynamoDB 主控台](#)、[PutResourcePolicyAPI](#)、[AWS SDK](#) 或 [範本](#)，將以資源為基礎的政策附加至現有資料表的串流，或修改現有政策。 [AWS CLI](#) [AWS CloudFormation](#)

### Note

使用 [CreateTable](#) 或 [UpdateTable](#) API 建立串流時，您無法將原則附加至串流。不過，您可以在刪除資料表後修改或刪除策略。您也可以修改或刪除已停用串流的政策。

## AWS CLI

下列 IAM 政策範例使用 `put-resource-policy` AWS CLI 命令，將以資源為基礎的政策附加至名 *MusicCollection* 為的資料表串流。此範例可讓使用者 *John* 在串流上執行 [GetRecords](#)、[GetShardIterator](#)、和 [DescribeStream](#) API 動作。

請記住將 `##` 文本替換為特定於資源的信息。

```
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. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在 DynamoDB 主控台儀表板上，選擇 [表格]，然後選取現有的表格。  
  
請確定您選取的表格已開啟串流。如需有關開啟資料表串流的資訊，請參閱[啟用串流](#)。
3. 選擇許可索引標籤標籤。
4. 在使用中串流的資源型政策中，選擇 [建立串流政策]。

- 在以資源為基礎的政策編輯器中，新增政策以定義串流的存取權限。在此原則中，您可以指定誰可以存取串流，以及允許他們在串流上執行的動作。若要新增原則，請執行下列其中一個動作：
  - 輸入或貼上 JSON 政策文件。如需 IAM 政策語言的詳細資訊，請參閱 [IAM 使用者指南中的使用 JSON 編輯器](#) 建立政策。

**i** Tip

若要在 Amazon DynamoDB 開發人員指南中查看以資源為基礎的政策範例，請選擇政策範例。

- 選擇「新增陳述式」以新增陳述式，並在提供的欄位中輸入資訊。針對您想要新增的任意數量陳述式重複此步驟。

**⚠** Important

儲存原則之前，請務必先解決任何安全性警告、錯誤或建議。

- (選用) 選擇右下角的 Preview external access (預覽外部存取)，以預覽新政策會如何影響資源的公開和跨帳戶存取權。在儲存政策之前，您可以檢查它是否引入新的 IAM Access Analyzer 問題清單，或是解決現有的問題清單。如果您沒有看到作用中的分析器，請選擇 Go to Access Analyzer (移至 Access Analyzer)，以在 IAM Access Analyzer 中 [建立帳戶分析器](#)。如需詳細資訊，請參閱 [預覽存取權](#)。
- 選擇建立政策。

下列 IAM 政策範例會將以資源為基礎的政策附加至名 *MusicCollection* 為的資料表串流。此範例可讓使用者 *John* 在串流上執行 [GetRecords](#)、[GetShardIterator](#)、和 [DescribeStream](#) API 動作。

請記住將 ## 文本替換為特定於資源的信息。

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

## 從表格中移除以資源為基礎的政策

您可以使用 DynamoDB 主控台、[DeleteResourcePolicy](#) API、AWS SDK 或範本，從現有表格中刪除以資源為基礎的政策。AWS CLI AWS CloudFormation

### AWS CLI

下列範例會使用 `delete-resource-policy` AWS CLI 命令，從名 *MusicCollection* 為的資料表中移除以資源為基礎的政策。

請記住將 `##` 文本替換為特定於資源的信息。

```
aws dynamodb delete-resource-policy \
 --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

### AWS Management Console

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在 DynamoDB 主控台儀表板上，選擇 [表格]，然後選取現有的表格。
3. 選擇許可。
4. 從 [管理原則] 下拉式清單中選擇 [刪除原則]
5. 在 [刪除資源型原則以資源為基礎的資源] 對話方塊中，輸入 **confirm** 以確認刪除動作。
6. 選擇刪除。



## 跨帳戶存取以資源為基礎的政策

使用以資源為基礎的策略，您可以提供對不同資源的跨帳戶存取。AWS 帳戶如果您有與 AWS 區域資源相同的分析器，則資源型政策允許的所有跨帳戶存取將透過 IAM Access Analyzer 外部存取發現項目報告。IAM Access Analyzer 會比對 IAM [政策文法](#)和[最佳實務](#)來執行政策檢查，以驗證您的政策。這些檢查會產生問題清單並提供可行的建議，協助您撰寫具有功能性且符合安全最佳實務的政策。您可以在 [DynamoDB](#) 主控台的「權限」索引標籤中，從 IAM 存取分析器檢視作用中的發現項目。

如需使用 IAM 存取分析器驗證政策的相關資訊，請參閱 [IAM 使用者指南中的 IAM 存取分析器政策驗證](#)。若要檢視 IAM Access Analyzer 傳回的警告、錯誤和建議清單，請參閱 [IAM Access Analyzer 政策檢查參考](#)。

若要將存取帳戶 B 中表格 B 的 [GetItem](#) 權限授與帳戶 A 中的使用者 A，請執行下列步驟：

1. 將以資源為基礎的政策附加至表格 B，以授與使用者 A 執行 GetItem 動作的權限。
2. 將以身分識別為基礎的原則附加至使用者 A，授與其在表格 B 上執行 GetItem 動作的權限。

使用 [DynamoDB 主控台](#) 中提供的預覽外部存取選項，您可以預覽新政策如何影響資源的公開和跨帳戶存取。在儲存政策之前，您可以檢查它是否引入新的 IAM Access Analyzer 問題清單，或是解決現有的問題清單。如果您沒有看到作用中的分析器，請選擇 Go to Access Analyzer (移至 Access Analyzer)，在 IAM Access Analyzer 中 [建立帳戶分析器](#)。如需詳細資訊，請參閱 [預覽存取權](#)。

DynamoDB 資料平面和控制平面 API 中的表名稱參數可接受表格的完整 Amazon 資源名稱 (ARN)，以支援跨帳戶操作。如果您只提供資料表名稱參數，而不是完整的 ARN，則會在要求程式所屬帳戶中的資料表上執行 API 作業。如需使用跨帳戶存取的策略範例，請參閱 [跨帳戶存取的資源型政策](#)。

即使其他帳戶的主體正在讀取或寫入擁有者帳戶中的 DynamoDB 表，也會向資源擁有者的帳戶收費。如果表格已佈建輸送量，則來自擁有者帳戶和其他帳戶中要求程式的所有要求總和將決定要求是否會受到限制 (如果自動調度資源功能已停用) 或在啟用自動資源調度資源時向上/縮減。

請求將記錄在所有者和請求者帳戶的 CloudTrail 日誌中，以便兩個帳戶中的每個帳戶都可以跟踪哪個帳戶訪問了哪些數據。

### Note

[控制平面 API](#) 的跨帳戶存取具有較低的每秒交易 (TPS) 限制為 500 個要求。

## 使用資源型原則封鎖公用存取

[封鎖公用存取 \(BPA\)](#) 是一項功能，可識別並防止附加以資源為基礎的政策，這些政策可授予公開存取 [Amazon Web Services \(AWS\)](#) 帳戶中的 DynamoDB 表格、索引或串流。使用雙酚 A，您可以防止公開存取 DynamoDB 資源。BPA 會在建立或修改以資源為基礎的原則期間執行檢查，並透過 DynamoDB 協助改善您的安全狀態。

BPA 會使用 [自動推理](#) 來分析以資源為基礎的原則所授予的存取權，並在管理資源型原則時找到此類權限時發出警示。分析會驗證所有以資源為基礎的政策陳述式、動作以及政策中使用的條件金鑰集的存取。

### Important

BPA 可防止透過直接附加至 DynamoDB 資源 (例如表格、索引和串流) 以資源為基礎的原則授予公開存取，從而協助保護您的資源。除了使用 BPA 之外，請仔細檢查以下政策，以確認它們不授予公眾訪問權限：

- 附加至關聯 AWS 主體 (例如 IAM 角色) 的身分識別型政策
- 附加至關聯 AWS 資源 (例如 AWS Key Management Service (KMS) 金鑰) 的以資源為基礎的政策

您必須確定 [主參與者](#) 不包含 \* 項目，或其中一個指定的條件索引鍵會限制主參與者對資源的存取。如果以資源為基礎的政策授予對您表格、索引或串流的公用存取權 AWS 帳戶，DynamoDB 將阻止您建立或修改原則，直到政策中的規格得到更正並視為非公開為止。

您可以在 Principal 區塊內指定一或多個主參與者，將原則設為非公用。下列以資源為基礎的原則範例會透過指定兩個主體來封鎖公用存取。

```
{
 "Effect": "Allow",
 "Principal": {
 "AWS": [
 "123456789012",
 "111122223333"
]
 },
 "Action": "dynamodb:*",
 "Resource": "*"
}
```

透過指定特定條件金鑰來限制存取的原則也不會被視為公用。除了評估以資源為基礎的原則中指定的主體之外，下列[受信任條件索引鍵](#)還可用來完成非公用存取以資源為基礎的原則評估：

- `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`

此外，若要讓以資源為基礎的政策是非公開的，Amazon 資源名稱 (ARN) 和字串金鑰的值不得包含萬用字元或變數。如果以資源為基礎的政策使用 `aws:PrincipalIsAWSService` 金鑰，則必須確定已將索引鍵值設定為 `true`。

以下策略限制了對指定帳戶 John 中的用戶的訪問。該條件使得 `Principal` 約束，不被視為公共。

```
{
 "Effect": "Allow",
 "Principal": {
 "AWS": "*"
 },
 "Action": "dynamodb:*",
 "Resource": "*",
 "Condition": {
 "StringEquals": {
```

```
 "aws:PrincipalArn": "arn:aws:iam::123456789012:user/John"
 }
}
```

下列非公用資源型原則範例會限制sourceVPC使用運算子。StringEquals

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "AWS": "*"
 },
 "Action": "dynamodb:*",
 "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
 "Condition": {
 "StringEquals": {
 "aws:SourceVpc": [
 "vpc-91237329"
]
 }
 }
 }
]
}
```

## 資源型政策支援的 API 作業

本主題列出以資源為基礎的原則支援的 API 作業。但是，對於跨帳戶存取，您只能透過以資源為基礎的政策使用一組特定的 DynamoDB API。您無法將以資源為基礎的政策附加到資源類型，例如備份和匯入。與在這些資源類型上操作的 API 相對應的 IAM 動作會排除在以資源為基礎的政策中支援的 IAM 動作之外。由於資料表管理員在相同帳戶內設定內部資料表設定，因此 API (例如[UpdateTimeToLive](#)和[DisableKinesisStreamingDestination](#)) 不支援透過以資源為基礎的政策跨帳戶存取。

支援跨帳戶存取的 DynamoDB 資料平面和控制平面 API 也支援資料表名稱多載，讓您指定資料表 ARN 而非資料表名稱。您可以在這些 API 的TableName參數中指定資料表 ARN。但是，並非所有這些 API 都支持跨帳戶訪問。

下表列出以資源為基礎的政策和跨帳戶存取的 API 層級支援。

API 動作	資源型政策支援	跨帳戶支援
Data Plane - Tables/indexes		
<a href="#">DeleteItem</a>	是	是
<a href="#">GetItem</a>	是	是
<a href="#">PutItem</a>	是	是
<a href="#">查詢</a>	是	是
<a href="#">掃描</a>	是	是
<a href="#">UpdateItem</a>	是	是
<a href="#">TransactGetItems</a>	是	是
<a href="#">TransactWriteItems</a>	是	是
<a href="#">BatchGetItem</a>	是	是
<a href="#">BatchWriteItem</a>	是	是
PartiQL		
<a href="#">BatchExecuteStatement</a>	是	否
<a href="#">ExecuteStatement</a>	是	否
<a href="#">ExecuteTransaction</a>	是	否
Control Plane - Tables		
<a href="#">CreateTable</a>	否	否
<a href="#">DeleteTable</a>	是	是
<a href="#">DescribeTable</a>	是	是
<a href="#">UpdateTable</a>	是	是
Version 2019.11.21 (Current) global tables		

API 動作	資源型政策支援	跨帳戶支援
<a href="#">DescribeTableReplicaAutoScaling</a>	是	否
<a href="#">UpdateTableReplicaAutoScaling</a>	是	否
Version 2017.11.29 (Legacy) global table		
<a href="#">CreateGlobalTable</a>	否	否
<a href="#">DescribeGlobalTable</a>	否	否
<a href="#">DescribeGlobalTableSettings</a>	否	否
<a href="#">ListGlobalTables</a>	否	否
<a href="#">UpdateGlobalTable</a>	否	否
<a href="#">UpdateGlobalTableSettings</a>	否	否
Tags		
<a href="#">ListTagsOfResource</a>	是	是
<a href="#">TagResource</a>	是	是
<a href="#">UntagResource</a>	是	是
Backup/Restore		
<a href="#">CreateBackup</a>	是	否
<a href="#">DescribeBackup</a>	否	否
<a href="#">DeleteBackup</a>	否	否
<a href="#">RestoreTableFromBackup</a>	否	否
Continuous Backup/Restore (PITR)		

API 動作	資源型政策支援	跨帳戶支援
<a href="#">DescribeContinuousBackups</a>	是	否
<a href="#">RestoreTableToPointInTime</a>	是	否
<a href="#">UpdateContinuousBackups</a>	是	否
Contributor Insights		
<a href="#">DescribeContributorInsights</a>	是	否
<a href="#">ListContributorInsights</a>	否	否
<a href="#">UpdateContributorInsights</a>	是	否
Export		
<a href="#">DescribeExport</a>	否	否
<a href="#">ExportTableToPointInTime</a>	是	否
<a href="#">ListExports</a>	否	否
Import		
<a href="#">DescribeImport</a>	否	否
<a href="#">ImportTable</a>	否	否
<a href="#">ListImports</a>	否	否
Kinesis		
<a href="#">DescribeKinesisStreamingDestination</a>	是	否
<a href="#">DisableKinesisStreamingDestination</a>	是	否
<a href="#">EnableKinesisStreamingDestination</a>	是	否

API 動作	資源型政策支援	跨帳戶支援
<a href="#">UpdateKinesisStreamingDestination</a>	是	否
Resource policies		
<a href="#">GetResourcePolicy</a>	是	否
<a href="#">PutResourcePolicy</a>	是	否
<a href="#">DeleteResourcePolicy</a>	是	否
Time-to-Live		
<a href="#">DescribeTimeToLive</a>	是	否
<a href="#">UpdateTimeToLive</a>	是	否
Others		
<a href="#">DescribeLimits</a>	否	否
<a href="#">DescribeEndpoints</a>	否	否
<a href="#">ListBackups</a>	否	否
<a href="#">ListTables</a>	否	否

下表列出 DynamoDB Streams API 的 API 層級支援，適用於以資源為基礎的政策和跨帳戶存取。

API 動作	資源型政策支援	跨帳戶支援
<a href="#">DescribeStream</a>	是	是
<a href="#">GetRecords</a>	是	是
<a href="#">GetShardIterator</a>	是	是
<a href="#">ListStreams</a>	否	否



## 使用 IAM 身分型政策和以 DynamoDB 資源為基礎的政策進行授權

以身分識別為基礎的政策會附加至身分，例如 IAM 使用者、使用者群組和角色。這些是 IAM 政策文件，用於控制身分可以執行哪些動作、哪些資源以及在什麼情況下。您可以[管理](#)身分識別型原則，也可以是[內嵌](#)原則。

以資源為基礎的政策是您附加至資源的 IAM 政策文件，例如 DynamoDB 表。這些政策會授予指定的主體許可，允許在該資源上執行特定的動作，並且定義資源所適用的條件。例如，DynamoDB 表格的以資源為基礎的政策也包含與該表關聯的索引。資源型政策是內嵌政策。不存在受管的資源型政策。

如需這些政策的詳細資訊，請參閱 IAM 使用者指南中的以[身分識別為基礎的政策和以資源為基礎的政策](#)。

如果 IAM 主體來自與資源擁有者相同的帳戶，則以資源為基礎的政策就足以指定資源的存取權限。您仍然可以選擇使用以 IAM 身分識別為基礎的政策以及以資源為基礎的政策。對於跨帳戶存取，您必須明確允許身分識別和資源策略的存取，如中[跨帳戶存取以資源為基礎的政策](#)所指定。當您使用這兩種類型的策略時，系統會按照[確定帳戶內是否允許或拒絕請求中所述評估策略](#)。

### 資源型政策範例

當您在以資源為基礎的策略的Resource欄位中指定 ARN 時，只有當指定的 ARN 與所附加的 DynamoDB 資源的 ARN 相符時，該策略才會生效。

#### Note

請記住將##文本替換為特定於資源的信息。

### 以資源為基礎的資源策略

下列附加至 DynamoDB 表格名*MusicCollection*為的以資源為基礎的政策將授予 IAM 使用者 *John* 和 *Jane* 對資源執行[GetItem](#)和[BatchGetItem](#)動作的權限。*MusicCollection*

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

## 串流以資源為基礎的政策

下列附加至名為的 DynamoDB 串流的以資源為基礎的政策可授2024-02-12T18:57:26.492予 IAM 使用者 *John* 和 *Jane* 對資源執行[GetRecords](#)和 [DescribeStream](#)API 動作的權限。[GetShardIterator](#)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"
]
 }
]
```

## 基於資源的策略，用於訪問對指定的資源執行所有操作

若要允許使用者對資料表執行所有動作，以及包含資料表的所有相關索引，您可以使用萬用字元 (\*) 來代表動作以及與資料表相關聯的資源。使用萬用字元作為資源，將允許使用者存取 DynamoDB 表及其所有相關索引，包括尚未建立的索引。例如，下列原則會授與使用者 *John* 對 *MusicCollection* 表格及其所有索引執行任何動作的權限，包括 future 將要建立的任何索引。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "1111",
 "Effect": "Allow",
 "Principal": "arn:aws:iam::111122223333:user/John",
 "Action": "dynamodb:*",
 "Resource": [
 "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
 "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/index/*"
]
 }
]
}
```

## 跨帳戶存取的資源型政策

您可以為跨帳戶 IAM 身分指定許可，以存取 DynamoDB 資源。例如，您可能需要來自信任帳戶的使用者才能取得讀取表格內容的存取權，條件是他們只能存取這些項目中的特定項目和特定屬性。下列原則允許從受信任的 AWS 帳戶 識別碼 *1111111111* 存取使用者 *John*，以使用 API 存取帳戶 *123456789012* 中的資料表中的資料。[GetItem](#) 該策略可確保使用者只能存取具有主索引鍵 *Jane* 的項目，而且使用者只能擷取屬性 *Artist* 和 *SongTitle*，但不能擷取其他屬性。

**⚠ Important**

如果您未指定SPECIFIC\_ATTRIBUTES條件，您會看到傳回之項目的所有屬性。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "CrossAccountTablePolicy",
 "Effect": "Allow",
 "Principal": {
 "AWS": "arn:aws:iam::111111111111:user/John"
 },
 "Action": "dynamodb:GetItem",
 "Resource": [
 "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
],
 "Condition": {
 "ForAllValues:StringEquals": {
 "dynamodb:LeadingKeys": "Jane",
 "dynamodb:Attributes": [
 "Artist",
 "SongTitle"
]
 },
 "StringEquals": {
 "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
 }
 }
 }
]
}
```

除了先前以資源為基礎的政策之外，附加至使用者 *John* 的身分型政策也需要允許跨帳戶存取的 GetItem API 動作才能運作。##### *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"
]
 },
 "StringEquals": {
 "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
 }
 }
}

```

使用者約翰可以透過在 `table-name` 參數中指定資料表 ARN 來提出 `GetItem` 要求，以存取帳戶 **123456789012** *MusicCollection* 中的資料表。

```

aws dynamodb get-item \
 --table-name arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
 --key '{"Artist": {"S": "Jane"}}' \
 --projection-expression 'Artist, SongTitle' \
 --return-consumed-capacity TOTAL

```

## 具有 IP 位址條件的資源型政策

您可以套用條件來限制來源 IP 位址、虛擬私有雲端 (VPC) 和 VPC 端點 (VPCE)。您可以根據原始請求的來源位址指定權限。例如，只有從特定 IP 來源 (例如企業 VPN 端點) 存取 DynamoDB 資源時，您可能想要允許使用者存取 DynamoDB 資源。在 `Condition` 陳述式中指定這些 IP 位址。

下列範例允許使用者 *John* 在來源 IP 為和時存取任何 DynamoDB 資源。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"
]
 }
 }
 }
]
 }
}

```

**##### DynamoDB ##### VPC ##### vpce-1a2b3c4d#**

```

{
 "Id": "PolicyId",
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "AccessToSpecificVPCEOnly",
 "Principal": "*",
 "Action": "dynamodb:*",
 "Effect": "Deny",
 "Resource": "*",
 "Condition": {
 "StringNotEquals": {
 "aws:sourceVpce": "vpce-1a2b3c4d"
 }
 }
 }
]
}

```

## 使用 IAM 角色的以資源為基礎的政策

您也可以在以資源為基礎的政策中指定 IAM 服務角色。擔任此角色的 IAM 實體受到針對該角色指定的允許動作以及資源型政策中特定資源集合的限制。

下列範例可讓 IAM 實體在 *MusicCollection* 和 DynamoDB 資源上執行所有動 *Dynamo MusicCollection* DB 資源動作。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "1111",
 "Effect": "Allow",
 "Principal": { "AWS": "arn:aws:iam::111122223333:role/John" },
 "Action": "dynamodb:*",
 "Resource": [
 "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
 "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/*"
]
 }
]
}
```

## 資源型政策考量事項

當您為 DynamoDB 資源定義以資源為基礎的政策時，需要考量下列事項：

### 一般考量

- 以資源為基礎的政策文件所支援的大小上限為 20 KB。根據此限制計算原則大小時，DynamoDB 會計算空格。
- 在成功更新相同資源的策略之後，會封鎖指定資源的後續政策更新 15 秒。
- 目前，您只能將以資源為基礎的政策附加至現有串流。您無法在建立串流時將原則附加至串流。

### 全域資料表考量

- [全域表格版本 2017.11.29 \(舊版\)](#) 複本不支援以資源為基礎的政策。
- 在以資源為基礎的政策中，如果針對 DynamoDB 服務連結角色 (SLR) 複寫全域表資料的動作遭到拒絕，則新增或刪除複本將失敗並顯示錯誤。

- [:AWS: DynamoDB:: GlobalTable](#) 資源不支援建立複本，並在部署堆疊更新的區域以外的區域中，將以資源為基礎的原則新增至相同堆疊更新中的該複本。

## 跨帳戶考量

- 使用資源型政策的跨帳戶存取不支援含 AWS 受管金鑰的加密資料表，因為您無法授與受管理 KMS 原則的跨帳戶存取 AWS 權。

## AWS CloudFormation 考量

- 以資源為基礎的政策不支援漂移偵測。如果您在 AWS CloudFormation 堆疊範本之外更新以資源為基礎的政策，則需要使用變更來更新 CloudFormation 堆疊。
- 以資源為基礎的政策不支援額外變更。如果您在範本之外新增、更新或刪除原則，則 CloudFormation 範本中沒有變更原則時，將不會覆寫變更。

例如，假設您的範本包含以資源為基礎的政策，您稍後會在範本外部進行更新。如果您未對範本中的原則進行任何變更，DynamoDB 中更新的原則將不會與範本中的原則同步。

相反地，假設您的範本不包含以資源為基礎的政策，但您可以在範本之外新增政策。只要您未將此原則新增至範本，就不會從 DynamoDB 中移除。當您將原則新增至範本並更新堆疊時，DynamoDB 中的現有政策將會更新，以符合範本中定義的原則。

## 資源型政策最佳做法

本主題說明為 DynamoDB 資源定義存取權限的最佳做法，以及這些資源允許的動作。

### 簡化對 DynamoDB 資源的存取控制

如果需要存取 DynamoDB 資源的 AWS Identity and Access Management 主體與資源擁有者屬於相同的主體，則不需要每個主體都需要以 IAM 身分識別 AWS 帳戶 為基礎的政策。被連接到給定的資源基於資源的策略就足夠了。這種類型的配置簡化了訪問控制。

### 使用以資源為基礎的政策保護您的 DynamoDB 資源

對於所有 DynamoDB 表格和串流，請建立以資源為基礎的政策，以強制執行這些資源的存取控制。以資源為基礎的原則可讓您在資源層級集中管理權限、簡化 DynamoDB 表、索引和串流的存取控制，並減少管理額外負荷。如果未針對資料表或串流指定以資源為基礎的政策，則會隱含拒絕資料表或串流的存取，除非與 IAM 主體相關聯的身分型政策允許存取。



## 套用最低權限許可

當您針對 DynamoDB 資源使用以資源為基礎的政策設定權限時，僅授與執行動作所需的權限。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。探索工作負載或使用案例所需的許可時，您可能會從廣泛許可開始。隨著使用案例的成熟，您可以設法減少授予的許可，以便朝向最低權限的目標邁進。

### 分析跨帳戶存取活動，以產生最低權限原則

IAM Access Analyzer 會報告對以資源為基礎的政策中指定的外部實體的跨帳戶存取，並提供可見性以協助您精簡許可並符合最低權限。如需政策產生的詳細資訊，請參閱 [IAM Access Analyzer 政策產生](#)。

### 使用 IAM 存取分析器產生最低權限政策

若只授予執行任務所需的許可，您可以根據在 AWS CloudTrail 中記錄的存取活動產生政策。IAM 存取分析器會分析政策使用的服務和動作。

## DynamoDB 中的資料保護

Amazon DynamoDB 提供高耐用性儲存基礎設施，專為任務關鍵型及主要資料儲存體所設計。資料會以備援方式存放在 Amazon DynamoDB 區域中多個設施的多部裝置上。

DynamoDB 可保護儲存在靜態的使用者資料，以及內部部署用戶端與 DynamoDB 之間，以及 DynamoDB 與相同區域內的其他 AWS 資源之間傳輸中的資料。AWS

### 主題

- [DynamoDB 靜態加密](#)
- [DynamoDB Accelerator 中的資料保護](#)
- [網際網路流量隱私權](#)

## DynamoDB 靜態加密

儲存在 Amazon DynamoDB 中的所有使用者資料都會完全靜態加密。DynamoDB 靜態加密使用存放在 [AWS Key Management Service \(AWS KMS\)](#) 的加密金鑰提供加強的安全性。此功能協助降低了保護敏感資料所涉及的操作負擔和複雜性。您可以透過靜態加密，建立符合嚴格加密合規和法規要求，而且對安全性要求甚高的應用程式。

每當資料存放到耐用的媒體時，DynamoDB 靜態加密就會透過保護加密資料表中的資料：包括其主索引鍵、本機及全域次要索引、串流、全域資料表、備份和 DynamoDB Accelerator (DAX) 叢集，來提供

另一層資料保護。組織政策、行業或政府法規，以及合規要求可能會經常需要使用靜態加密來增加應用程式的資料安全性。

靜態加密與 AWS KMS 整合，用於管理用於加密表格的加密金鑰。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的 [AWS Key Management Service 概念](#)。

建立新資料表時，您可以選擇下列其中一種 AWS KMS key 類型來加密資料表。您可以隨時在這些金鑰類型之間切換。

- AWS 擁有的金鑰 — 預設加密類型。此金鑰是由 DynamoDB 所擁有 (不需額外費用)。
- AWS 受管金鑰 — 密鑰存儲在您的帳戶中，由管理 AWS KMS (AWS KMS 收費)。
- 客戶受管金鑰 – 金鑰會存放在您的帳戶中，並且由您建立、擁有且管理。您可以完全控制 KMS 金鑰 (AWS KMS 需付費)。

如需金鑰類型的詳細資訊，請參閱 [客戶金鑰和 AWS 金鑰](#)。

#### Note

- 建立啟用靜態加密的新 DAX 叢集時，AWS 受管金鑰 會用來加密叢集中的靜態資料。
- 如果您的資料表有排序索引鍵，則某些標示範圍界限的排序索引鍵將會以純文字的格式存放在資料表中繼資料中。

當您存取加密的資料表，DynamoDB 會以透明方式解密資料表資料。您不必變更任何代碼或應用程式來使用或管理加密的資料表。DynamoDB 會繼續提供您預期的個位數毫秒的相同延遲，且所有 DynamoDB 查詢會在加密資料上順暢地運作。

您可以在建立新表格時指定加密金鑰，或使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或 Amazon DynamoDB API 在現有資料表上切換加密金鑰。如要瞭解如何作業，請參閱 [在 DynamoDB 中管理加密資料表](#)。

使用的 AWS 擁有的金鑰 靜態加密不收取額外費用。但是，客戶管理的金鑰需要支付 AWS 受管金鑰和 AWS KMS 費用。如需定價的詳細資訊，請參閱 [AWS KMS 定價](#)。

DynamoDB 靜態加密可在所有 AWS 區域使用，包括中 AWS 國 (北京) 和 AWS 中國 (寧夏) 區域以及 AWS GovCloud (美國) 區域。如需詳細資訊，請參閱 [靜態加密：如何運作](#) 及 [DynamoDB 靜態加密使用須知](#)。

## 靜態加密：如何運作

Amazon DynamoDB 靜態加密功能會使用 256 位元的進階加密標準 (AES-256) 來加密您的資料，此項標準有助於防止對底層儲存未經授權的存取，進而保護您的資料。

靜態加密與 AWS Key Management Service (AWS KMS) 整合，用於管理用於加密表格的加密金鑰。

### Note

在 2022 年 5 月，將輪 AWS KMS 換日程 AWS 受管金鑰 從每 3 年 (約 1,095 天) 變更為每年 (約 365 天)。

新功能會 AWS 受管金鑰 在建立一年後自動輪換，之後大約每年輪換一次。

現 AWS 受管金鑰 有檔案會在最近輪替一年後自動輪換，之後每年輪換一次。

### AWS 擁有的金鑰

AWS 擁有的金鑰 不會儲存在您的 AWS 帳戶中。它們是 AWS 擁有和管理以在多個 AWS 帳戶中使用的 KMS 金鑰集合的一部分。AWS 服務可用 AWS 擁有的金鑰 來保護您的資料。AWS 擁有的金鑰 DynamoDB 所使用的每年輪換一次 (大約 365 天)。

您無法檢視、管理 AWS 擁有的金鑰、使用或稽核其使用狀況。不過，您不需要執行任何工作或變更任何程式，即可保護加密您的資料的金鑰。

使用時不會向您收取月費或使用費 AWS 擁有的金鑰，也不會計入您帳戶的 AWS KMS 配額。

### AWS 受管金鑰

AWS 受管金鑰 是您帳戶中的 KMS 金鑰，由與整合的 AWS 服務代表您建立、管理和使用 AWS KMS。您可以檢視您的帳戶中的 AWS 受管金鑰，檢視其金鑰政策，以及在 AWS CloudTrail 日誌中稽核其使用方式。不過，您無法管理這些 KMS 金鑰或變更其許可。

靜態加密會自動整 AWS KMS 合，以 AWS 受管金鑰 管理用於加密表格的 DynamoDB (aws/dynamodb)。如果在您建立加密的 DynamoDB 表時不存在，AWS KMS 會自動為您建立新金鑰。AWS 受管金鑰 此金鑰會與 future 建立的加密資料表一起使用。AWS KMS 結合安全、高可用性的硬體和軟體，提供專為雲端擴充的金鑰管理系統。

如需有關管理的權限的詳細資訊 AWS 受管金鑰，請參閱《AWS Key Management Service 開發人員指南》[AWS 受管金鑰中的授權使用](#)。

## 客戶受管金鑰

客戶受管金鑰是您 AWS 帳戶中您建立、擁有和管理的 KMS 金鑰。您可以完全控制這些 KMS 金鑰，包括建立和維護其金鑰政策、IAM 政策和授與、啟用和停用它們、輪換其密碼編譯材料、新增標籤、建立參考別名，以及排程供刪除。如需有關管理客戶受管金鑰的詳細資訊，請參閱[客戶受管金鑰政策](#)。

當您指定一個客戶受管金鑰作為資料表層級加密金鑰時，DynamoDB 資料表、本機和全域次要索引及串流皆會以相同的客戶受管金鑰加密。隨需備份會以於備份建立時指定的資料表層級加密金鑰加密。更新資料表層級加密金鑰並不會變更與現有隨需備份相關的加密金鑰。

將客戶受管金鑰的狀態設為停用，或排定其刪除時間，可避免所有使用者和 DynamoDB 服務可在資料表上加密或解密資料，以及執行讀取或寫入操作。DynamoDB 必須能夠存取您的加密金鑰，才能確保您可以繼續存取資料表並避免資料遺失。

如果您停用客戶受管金鑰或排定其刪除的時間，您的資料表狀態就會成為 Inaccessible (無法存取)。為確保您可以持續使用資料表進行作業，您必須在七天內為指定的加密金鑰提供 DynamoDB 存取權限。只要服務偵測到您的加密金鑰無法存取，DynamoDB 就會寄送電子通知提醒您。

### Note

- 如果您的客戶受管金鑰仍然無法由 DynamoDB 服務存取且時間超過七天，資料表就會存檔並無法再存取。DynamoDB 會建立資料表的隨需備份，並會向您收取費用。您可以使用隨需備份還原您的資料至新的資料表。若要開始還原，最後一個在資料表中的客戶受管金鑰必須啟用，且 DynamoDB 必須可以存取該資料表。
- 如果用來加密全域資料表複本的客戶受管金鑰無法存取，DynamoDB 會從複寫群組中移除此複本。將不會刪除複本，且複寫將會在偵測到客戶受管金鑰為無法存取後 20 小時停止進出此區域。

如需更多詳細資訊，請參閱[啟用金鑰](#)和[刪除金鑰](#)。

## 使用注意事項 AWS 受管金鑰

除非 Amazon DynamoDB 可以存取您帳戶中存放的 KMS 金鑰，否則無法讀取您 AWS KMS 的表格資料。DynamoDB 使用信封加密和金鑰階層來加密資料。您的 AWS KMS 加密金鑰會用來加密此金鑰階層的根金鑰。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的[封套加密](#)。

您可以使用 AWS CloudTrail 和 Amazon CloudWatch 日誌來追蹤 DynamoDB 代表您傳送 AWS KMS 的請求。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的[監控 DynamoDB 與 AWS KMS 的互動](#)。

不會 AWS KMS 針對每個 DynamoDB 作業呼叫。每個具有作用中流量的呼叫者每 5 分鐘會重新整理一次金鑰。

請確定您已設定好 SDK，來重複使用連線。否則，您將會遇到 DynamoDB 的延遲，必須為每個 DynamoDB 作業重新建立新的 AWS KMS 快取項目。此外，您可能不得不面對更高 AWS KMS 和 CloudTrail 成本。例如，若要使用 Node.js SDK 來執行這項動作，您可以在 keepAlive 開啟時，建立新的 HTTPS 代理。如需詳細資訊，請參閱《AWS SDK for JavaScript 開發人員指南》中的[在 Node.js 中設定 keepAlive](#)。

## DynamoDB 靜態加密使用須知

在您使用 Amazon DynamoDB 中靜態加密時請考量以下項目。

所有資料表資料都會加密

已對所有 DynamoDB 資料表資料啟用伺服器端靜態加密，且無法停用。您無法僅加密資料表中的項目子集。

只有持久性儲存媒體上的資料是靜態 (靜止) 時，靜態加密才會加密資料。如果對傳輸中資料或使用中資料有資料安全性的疑慮，您可能需要進行額外的措施：

- 傳輸中的資料：DynamoDB 中的所有資料都是在傳輸中加密的。根據預設，與 DynamoDB 之間的通訊會使用 HTTPS 協定，其使用 Secure Sockets Layer (SSL)/Transport Layer Security (TLS) 加密保護網路流量。
- 使用中資料：在將資料傳送至 DynamoDB 前使用用戶端加密來保護資料。如需詳細資訊，請參閱《Amazon DynamoDB Encryption Client 開發人員指南》中的[用戶端與伺服器端加密](#)。

您可以將串流與加密的資料表搭配著使用。DynamoDB Streams 一律會以資料表層級加密金鑰加密。如需詳細資訊，請參閱[DynamoDB Streams 的變更資料擷取](#)。

DynamoDB 的備份會受到加密，且從備份恢復的資料表也已啟用加密。您可以使用 AWS 擁有的金鑰 AWS 受管金鑰、或客戶管理的金鑰來加密備份資料。如需詳細資訊，請參閱[使用 DynamoDB 的隨需備份與還原](#)。

本機次要索引和全域次要索引會使用與基礎資料表相同的金鑰加密。



## 加密類型

### Note

Global Table 版本 2017 不支援客戶受管金鑰。如果要在 DynamoDB Global Table 中使用客戶受管金鑰，則需要將資料表升級為 Global Table 版本 2019，然後將其啟用。

在上 AWS Management Console，加密類型是 KMS 當您使用 AWS 受管金鑰 或客戶管理金鑰來加密資料時。使用 AWS 擁有的金鑰時，加密類型為 DEFAULT。在 Amazon DynamoDB API 中，加密類型是 KMS 當您使用 AWS 受管金鑰 或客戶受管金鑰時。在缺少加密類型的情況下，會使用 AWS 擁有的金鑰來加密資料。您可以隨時在 AWS 擁有的金鑰 AWS 受管金鑰、和客戶管理的金鑰之間切換。您可以使用主控台、AWS Command Line Interface (AWS CLI) 或 Amazon DynamoDB API 來切換加密金鑰。

請注意，使用客戶受管金鑰時有下列限制：

- 您無法搭配 DynamoDB Accelerator (DAX) 叢集使用客戶受管金鑰。如需詳細資訊，請參閱 [DAX 靜態加密](#)。
- 您可以使用客戶受管金鑰來加密使用交易的資料表。不過，為了確保交易傳播的持久性，服務會暫時儲存交易請求的副本，並使用 AWS 擁有的金鑰進行加密。資料表和次要索引中已遞交的資料一律會使用客戶受管金鑰進行靜態加密。
- 您可以使用客戶受管金鑰來加密使用 Contributor Insights 的資料表。不過，傳輸目標的 Amazon CloudWatch 資料會使用 AWS 擁有的金鑰。
- 當您轉換為新的客戶管理金鑰時，請務必將原始金鑰保持啟用狀態，直到程序完成為止。AWS 在使用新密鑰加密數據之前，仍然需要原始密鑰來解密數據。當表格的 SSEDescription 狀態為已啟用且顯示新客戶受管金鑰 MasterKeyArn 的 KMS 時，此程序將會完成。此時可以停用或排程刪除原始金鑰。
- 顯示新的客戶受管金鑰後，資料表和任何新的隨需備份都會使用新的金鑰加密。
- 任何現有的隨需備份都會使用建立備份時使用的客戶受管金鑰進行加密。您將需要相同的金鑰來還原這些備份。您可以使用 DescribeBackup API 來檢視該備份的 SSEDescription，以識別每個備份建立期間的金鑰。
- 如果您停用客戶受管金鑰或排定其刪除的時間，所有在 DynamoDB Streams 中的資料仍有 24 小時的生命週期。任何未擷取的活動資料如果超過 24 小時，則符合裁剪的資格。
- 如果您停用客戶受管金鑰或排定其刪除的時間，存留時間 (TTL) 刪除就會持續 30 分鐘。這些 TTL 刪除會持續發出至 DynamoDB Streams，並符合標準裁剪/保留間隔。

如需更多詳細資訊，請參閱[啟用金鑰](#)和[刪除金鑰](#)。

## 使用 KMS 金鑰和資料金鑰

DynamoDB 靜態加密功能使用 AWS KMS key 和資料金鑰階層來保護您的表格資料。當 DynamoDB 串流、全域資料表和備份寫入到持久性媒體時，DynamoDB 會使用相同的金鑰階層來保護 DynamoDB 串流、全域資料表和備份。

我們建議您先規劃加密策略，再於 DynamoDB 中使用資料表。如果您將敏感或機密資料儲存在 DynamoDB，請考慮在計劃中加入用戶端加密。如此一來，您就能盡量靠近資料來源進行加密，並確保資料在整個生命週期受到保護。如需詳細資訊，請參閱 [DynamoDB 加密客戶端](#) 文件。

## AWS KMS key

靜態加密會保護 AWS KMS key 下的 DynamoDB 資料表。根據預設，DynamoDB 會使用 [AWS 擁有的金鑰](#)，這是一種在 DynamoDB 服務帳戶中建立和管理的多租用戶加密金鑰。但是，您可以使用 [客戶受管金鑰](#) 或 AWS 帳戶中 DynamoDB (aws/dynamodb) 加密您的 DynamoDB 資料表。您可以為每個資料表選取不同的 KMS 金鑰。您為資料表選取的 KMS 金鑰也會用來加密其本機和全域次要索引、串流和備份。

建立或更新資料表時，您可以選取資料表的 KMS 金鑰。您可以隨時在 DynamoDB 主控台中或使用作業變更資料表的 KMS 金鑰。[UpdateTable](#) 切換金鑰的程序是無縫的，且不需要停機時間或降低服務效能。

### Important

DynamoDB 僅支援[對稱 KMS 金鑰](#)。您無法使用[非對稱 KMS 金鑰](#)來加密您的 DynamoDB 資料表。

使用客戶受管金鑰來取得下列功能：

- 您可以建立和管理 KMS 金鑰，包括設定[金鑰政策](#)、[IAM 政策](#)和[授予](#)來控制對 KMS 金鑰的存取。您可以[啟用和停用](#) KMS 金鑰、啟用和停用[自動金鑰輪換](#)，以及於不再使用時[刪除 KMS 金鑰](#)。
- 您可以搭配[匯入的金鑰材料](#)使用客戶受管金鑰，或是使用位於您所擁有及管理[自訂金鑰存放區](#)中的客戶受管金鑰。
- 您可以透過檢查記錄中的 DynamoDB API 呼叫來稽核 DynamoDB 表的加密和解密。[AWS KMS AWS CloudTrail](#)

AWS 受管金鑰 如果您需要下列任一功能，請使用：

- 您可以[檢視 KMS 金鑰](#)和[檢視其金鑰政策](#)。(您無法變更金鑰政策。)
- 您可以透過檢查記錄中的 DynamoDB API 呼叫來稽核 DynamoDB 表的加密和解密。[AWS KMS/AWS CloudTrail](#)

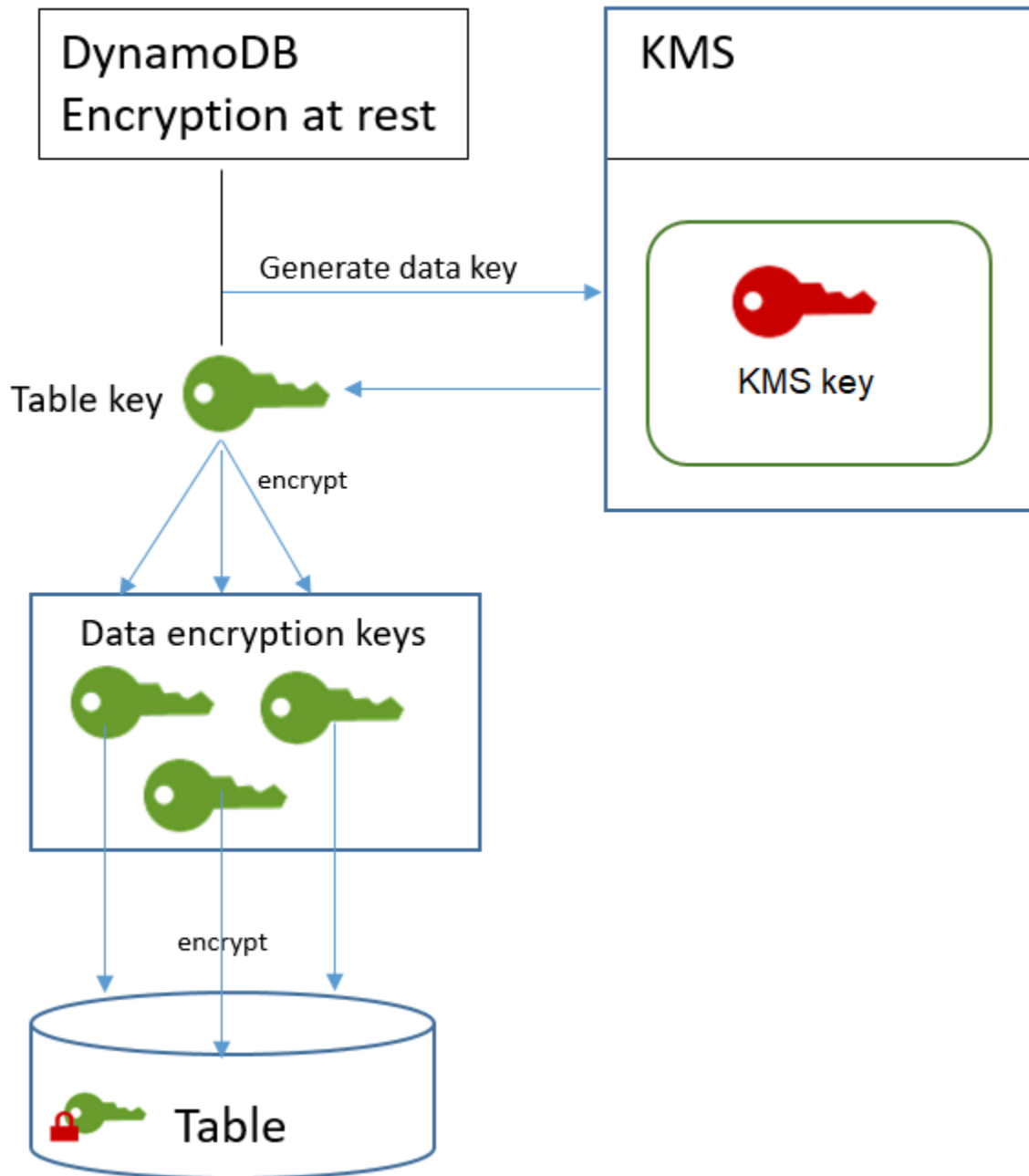
不過，AWS 擁有的金鑰 這是免費的，且其使用不會計入[AWS KMS 資源或要求配額](#)。客戶受管金鑰 AWS 受管金鑰 [會向這些 KMS 金鑰收取](#)每個 API 呼叫和 AWS KMS 配額的費用。

## 資料表金鑰

DynamoDB 對資料表使用 KMS 金鑰來為資料表[產生](#)和加密唯一的[資料金鑰](#)，稱為資料表金鑰。資料表金鑰會在加密資料表的生命週期內持續存在。

資料表金鑰用作金鑰加密金鑰。DynamoDB 使用此資料表金鑰來保護用於加密資料表資料的資料加密金鑰。DynamoDB 會為資料表中的每個基礎結構產生唯一的資料加密金鑰，但多個資料表項目可能受到相同的資料加密金鑰保護。





當您第一次存取加密的資料表時，DynamoDB 會傳送要求 AWS KMS 給使用 KMS 金鑰來解密資料表金鑰。然後，它使用純文字資料表金鑰來解密資料加密金鑰，並使用純文字資料加密金鑰來解密資料表資料。

DynamoDB 會在其外儲存和使用表格金鑰和資料加密金鑰。AWS KMS 它使用 [進階加密標準](#) (AES) 加密和 256 位元加密金鑰來保護所有金鑰。然後，它會將加密金鑰與加密資料一起存放，以便這些資料可用來隨需解密資料表。

如果您變更了資料表的 KMS 金鑰，DynamoDB 便會產生新的資料表金鑰。然後，使用新的資料表金鑰來重新加密資料表金鑰。

## 資料表金鑰快取

AWS KMS 為了避免呼叫每個 DynamoDB 作業，DynamoDB 會快取記憶體中每個呼叫者的純文字表格鍵。如果 DynamoDB 在閒置五分鐘後收到快取表格金鑰的請求，則會傳送新 AWS KMS 要求以解密資料表金鑰。此呼叫將擷取自上次解密資料表金鑰請求以來，對 AWS KMS 或 AWS Identity and Access Management (IAM) 中 KMS 金鑰存取政策所做的任何變更。

## 授權使用您的 KMS 金鑰

如果您使用[客戶受管金鑰](#)或帳戶中的[AWS 受管金鑰](#)保護您的 DynamoDB 資料表，該 KMS 金鑰的政策必須允許 DynamoDB 代您使用該金鑰。AWS 受管金鑰針對 DynamoDB 的授權內容包含其金鑰原則，並授與委派使用權限的權限。

您可以完全控制客戶受管金鑰的政策和授予，因為 AWS 受管金鑰在您的帳戶中，您可以檢視其政策和授予。但是，因為它是由管理的 AWS，因此您無法變更原則。

DynamoDB 不需要其他授權即可使用預設值[AWS 擁有的金鑰](#)來保護您的 AWS 帳戶

## 主題

- [金鑰政策 AWS 受管金鑰](#)
- [客戶受管金鑰的金鑰政策](#)
- [使用授予來授權 DynamoDB](#)

## 金鑰政策 AWS 受管金鑰

當 DynamoDB 在密碼編譯操作中使用適用於 DynamoDB (aws/dynamodb) 的[AWS 受管金鑰](#)時，它是代表正在存取[DynamoDB 資源](#)的使用者進行這項操作。上的金鑰原則授與帳戶中的所有 AWS 受管金鑰使用者可以將指定作業使用的權限。AWS 受管金鑰但是，只有在 DynamoDB 代表使用者提出請求時才會授予許可。金鑰原則中的[ViaService 條件](#)不允許任何使用者使用，AWS 受管金鑰除非要求源自 DynamoDB 服務。

此金鑰原則與所有政策一樣 AWS 受管金鑰，由建立 AWS。您不能改變它，但可以隨時檢視它。如需詳細資訊，請參閱[檢視金鑰政策](#)。

金鑰政策中的政策陳述式具有下列效果：

- 當請求來自 DynamoDB 代表他們時，允許帳戶中的使用者在密碼編譯作業中使用 DynamoDB。AWS 受管金鑰 此政策也允許使用者為 KMS 金鑰 [建立授予](#)。
- 允許帳戶中授權的 IAM 身分檢視 DynamoDB AWS 受管金鑰 的屬性，並 [撤銷允許 DynamoDB 使用 KMS 金鑰的授予](#)。DynamoDB 使用 [授予](#) 進行持續的維護操作。
- 允許 DynamoDB 執行唯讀作業，以便在您的帳戶中尋找 AWS 受管金鑰 適用於 DynamoDB 的項目。

```
{
 "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" : "*"
}]
}
```

## 客戶受管金鑰的金鑰政策

當您選取[客戶受管金鑰](#)來保護 DynamoDB 資料表時，DynamoDB 會取得許可，代替進行選取的委託人使用 KMS 金鑰。該委託人 (使用者或角色) 必須在 KMS 金鑰上擁有 DynamoDB 需要的許可。您可以在[金鑰政策](#)、[IAM 政策](#)或[授予](#)中提供這些許可。

至少，DynamoDB 在客戶受管金鑰上需要具備下列許可：

- [kms:Encrypt](#)
- [kms:Decrypt](#)
- [公里 : ReEncrypt\\*](#) ( 對於 [公里 : ReEncryptFrom](#) 和 [公里 : ReEncryptTo](#) )
- [公里 : GenerateDataKey\\*](#) ( 對於 [公里 : GenerateDataKey](#) 和 [公里 : GenerateDataKeyWithoutPlaintext](#) )
- [公里 : DescribeKey](#)
- [公里 : CreateGrant](#)

例如，以下範例金鑰政策只會提供必要許可。政策具有下列效果：

- 允許 DynamoDB 在密碼編譯操作中使用 KMS 金鑰並建立授予，但只有在其代替具備使用 DynamoDB 許可帳戶中的委託人時才能進行。如果政策陳述式中指定的委託人沒有使用 DynamoDB 的許可，呼叫便會失敗，即使呼叫是來自 DynamoDB 服務也一樣。
- [kms: ViaService](#) 條件金鑰只有在要求來自 DynamoDB 代表政策陳述式中列出的主體時，才允許使用權限。這些委託人無法直接呼叫這些操作。請注意，`kms:ViaService` 值 (`dynamodb.*.amazonaws.com`) 在區域位置中有星號 (\*)。DynamoDB 需要獨立於任何特定權限，AWS 區域 因此可以進行跨區域呼叫以支援 [DynamoDB 全域表](#)。
- 給予 KMS 金鑰管理員 (可取得 `db-team` 角色的使用者) 對 KMS 金鑰的唯讀存取權以及撤銷授予的許可，包括 [DynamoDB 需要用來保護資料表的授予](#)。

在使用範例金鑰原則之前，請先以您的實際主參與者取代範例主參與者。AWS 帳戶

```
{
 "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": "*"
 }
]
}
```

## 使用授予來授權 DynamoDB

除了金鑰政策外，DynamoDB 會使用授予在客戶受管金鑰或 AWS 受管金鑰 for DynamoDB (aws/dynamodb) 上設定許可。若要檢視您帳戶中 KMS 金鑰的授權，請使用此[ListGrants](#)作業。DynamoDB 不需要授予或任何其他許可，即可使用 [AWS 擁有的金鑰](#) 來保護您的資料表。

DynamoDB 在執行背景系統維護和持續資料保護任務時使用授予許可。它還使用授與來產生[資料表金鑰](#)。

每個授與都專屬於一個資料表。如果帳戶包含使用相同 KMS 金鑰加密的多個資料表，則每個資料表有每一種類型的授予。授權受 [DynamoDB 加密內容](#) 的限制，其中包括資料表名稱和 AWS 帳戶 ID，並且包含[淘汰授權](#) (如果不再需要) 的權限。

如要建立授予，DynamoDB 必須擁有代替建立加密資料表使用者呼叫 CreateGrant 的許可。對於 AWS 受管金鑰，DynamoDB 會從[金鑰原則](#)取得kms:CreateGrant權限，只有[CreateGrant](#)在 DynamoDB 代表授權使用者提出要求時，才允許帳戶使用者呼叫 KMS 金鑰。

金鑰政策也會允許帳戶[撤銷 KMS 金鑰上的授予](#)。不過，如果您撤銷作用中加密資料表的授予，DynamoDB 將無法保護和維護資料表。

### DynamoDB 加密內容

[加密內容](#)是一組金鑰/值對，其中包含任意非私密資料。當您在加密資料的要求中包含加密內容時，AWS KMS 密碼編譯會將加密內容繫結至加密的資料。若要解密資料，您必須傳遞相同的加密內容。

DynamoDB 會在所有密 AWS KMS 碼編譯作業中使用相同的加密內容。如果使用[客戶受管金鑰](#)或 [AWS 受管金鑰](#) 來保護您的 DynamoDB 資料表，您可以使用加密內容來識別在稽核記錄和日誌中使用 KMS 金鑰的情況。它也會以純文字顯示在日誌中，例如[AWS CloudTrail](#)和 [Amazon CloudWatch](#) 日誌。

加密內容也可以用作政策和授予中的授權條件。DynamoDB 使用加密內容來限制允許存取客戶管理金鑰或您帳戶和區 AWS 受管金鑰 域的[授權](#)。

在其請求中 AWS KMS，DynamoDB 會使用具有兩個金鑰值配對的加密內容。

```
"encryptionContextSubset": {
 "aws:dynamodb:tableName": "Books"
 "aws:dynamodb:subscriberId": "111122223333"
}
```

- 資料表 – 第一個鍵值對會識別 DynamoDB 正在加密的資料表。金鑰為 aws:dynamodb:tableName。值是資料表的名稱。

```
"aws:dynamodb:tableName": "<table-name>"
```

例如：

```
"aws:dynamodb:tableName": "Books"
```

- 帳戶 – 第二個鍵值對會識別 AWS 帳戶。金鑰為 `aws:dynamodb:subscriberId`。值是帳戶 ID。

```
"aws:dynamodb:subscriberId": "<account-id>"
```

例如：

```
"aws:dynamodb:subscriberId": "111122223333"
```

## 監控動 DynamoDB 的互動 AWS KMS

如果您使用[客戶受管金鑰](#)或[AWS 受管金鑰](#)保護 DynamoDB 表，則可以使用 AWS CloudTrail 記錄來追蹤 DynamoDB 代表您傳送的請求。AWS KMS

本節將討論 `GenerateDataKey`、`Decrypt` 和 `CreateGrant` 請求。此外，DynamoDB 會使用[DescribeKey](#)作業來判斷您選取的 KMS 金鑰是否存在於帳戶和區域中。當您刪除資料表時，它也會使用[RetireGrant](#)作業移除授權。

## GenerateDataKey

當您在資料表上啟用靜態加密時，DynamoDB 會建立唯一的資料表金鑰。它會傳送[GenerateDataKey](#)要求 AWS KMS，以指定資料表的 KMS 金鑰。

記錄 `GenerateDataKey` 操作的事件類似於以下範例事件。使用者是 DynamoDB 服務帳戶。參數包括秘密 KMS 金鑰的 Amazon 資源名稱 (ARN)、需要 256 位元金鑰的金鑰規範，以及識別資料表和 AWS 帳戶的[加密內容](#)。

```
{
 "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"
}

```

## 解密

當您存取加密的 DynamoDB 資料表時，DynamoDB 需要解密資料表金鑰以便解密階層下方的金鑰。然後解密資料表中的資料。解密資料表金鑰。DynamoDB 會將[解密](#)請求傳送到指 AWS KMS 定資料表的 KMS 金鑰。

記錄 Decrypt 操作的事件類似於以下範例事件。使用者是您存取資料表的使用 AWS 帳戶者中的主參與者。這些參數包括加密資料表金鑰 (做為[密文](#) Blob)，以及識別資料表和 AWS 帳戶 AWS KMS 從加密文字衍生出 KMS 金鑰的識別碼。

```

{
 "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

當您使用[客戶受管金鑰](#)或 [AWS 受管金鑰](#) 保護您的 DynamoDB 資料表時，DynamoDB 會使用[授權](#)，以允許服務執行持續資料保護和維護及耐用性任務。[AWS 擁有的金鑰](#) 上不需要這些授予。

DynamoDB 建立的授予專屬於特定資料表。[CreateGrant](#) 請求中的主參與者是建立表格的使用者。

記錄 CreateGrant 操作的事件類似於以下範例事件。參數包括資料表的 KMS 金鑰的 Amazon 資源名稱 (ARN)、承授者委託人和淘汰委託人 (DynamoDB 服務)，以及授予涵蓋的操作。它還包含一個限制，要求所有加密操作使用指定的[加密內容](#)。

```

{
 "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"
}
```

## 在 DynamoDB 中管理加密資料表

您可以使用 AWS Management Console 或 AWS Command Line Interface (AWS CLI) 在新表格上指定加密金鑰，並更新 Amazon DynamoDB 中現有表格上的加密金鑰。

### 主題

- [指定新資料表的加密金鑰](#)
- [更新加密金鑰](#)

### 指定新資料表的加密金鑰

請依照這些步驟，使用 Amazon DynamoDB 主控台或 AWS CLI 在新資料表上指定加密金鑰。

### 建立加密資料表 (主控台)

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇 Create Table (建立資料表)。針對資料表名稱，輸入 **Music**。針對主索引鍵，輸入 **Artist**，對於排序索引鍵，輸入 **SongTitle**，兩者都做為字串。
4. 在 Settings (設定) 中，確認選取 Customize settings (自訂設定)。

#### Note

如果選取 [使用預設設定]，則會在靜態時加密資料表，且不需額外付費。AWS 擁有的金鑰

5. 在「靜態加密」下，選擇加密類型- AWS 擁有的金鑰 AWS 受管金鑰、或客戶管理的金鑰。
  - 由 Amazon DynamoDB 擁有。AWS 擁有的金鑰，專門由 DynamoDB 擁有和管理。使用此金鑰不會向您收取額外費用。
  - AWS 受管理的金鑰。金鑰別名：aws/dynamodb。密鑰存儲在您的帳戶中，由 AWS Key Management Service (AWS KMS) 管理。AWS KMS 費用適用。
  - 儲存在您的帳戶中，且由您擁有和管理。客戶受管金鑰。密鑰存儲在您的帳戶中，由 AWS Key Management Service (AWS KMS) 管理。AWS KMS 費用適用。

**Note**

如果您選擇擁有和管理自己的金鑰，請確保已正確設定 KMS 金鑰政策。如需詳細資訊及範例，請參閱[客戶受管金鑰的金鑰政策](#)。

- 選擇 Create table (建立資料表) 以建立加密資料表。若要確認加密類型，選取 Overview (概觀) 標籤中的資料表詳細資訊並檢閱 Additional details (其他詳細資訊) 區段。

### 建立加密資料表 (AWS CLI)

使用建 AWS CLI 立具有 Amazon DynamoDB 的預設 AWS 擁有的金鑰金鑰 AWS 受管金鑰、或客戶受管金鑰的表格。

使用預設值建立加密資料表 AWS 擁有的金鑰

- 建立加密的 Music 資料表，如下所示。

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

**Note**

此表格現在已使用 DynamoDB 服務帳戶 AWS 擁有的金鑰 中的預設值加密。

若要使用適用 AWS 受管金鑰 於 DynamoDB 建立加密資料表

- 建立加密的 Music 資料表，如下所示。

```
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
```

```
--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
```

資料表描述的 SSEDescription 狀態會設為 ENABLED，而 SSEType 設為 KMS。

```
"SSEDescription": {
 "SSEType": "KMS",
 "Status": "ENABLED",
 "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-a123-ab1234a1b234",
}
```

使用 DynamoDB 由客戶受管金鑰建立加密資料表

- 建立加密的 Music 資料表，如下所示。

```
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
```

資料表描述的 SSEDescription 狀態會設為 ENABLED，而 SSEType 設為 KMS。

```
"SSEDescription": {
 "SSEType": "KMS",
 "Status": "ENABLED",
```

```
"KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-a123-ab1234a1b234",
}
```

## 更新加密金鑰

您也可以使用 DynamoDB 主控台或隨時更新 AWS 擁有的金鑰、AWS 受管金鑰和客戶受管金鑰之間現有資料表的加密金鑰。AWS CLI

### 更新加密金鑰 (主控台)

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇您要更新的資料表。
4. 選取 Actions (動作) 下拉式清單，然後選取 Update settings (更新設定) 選項。
5. 轉至 Additional settings (其他設定) 索引標籤。
6. 在 Encryption (加密) 下，選擇 Manage encryption (管理加密)。
7. 選擇加密類型：
  - 由 Amazon DynamoDB 擁有。AWS KMS 金鑰由 DynamoDB 擁有及管理。使用此金鑰不會向您收取額外費用。
  - AWS 託管密鑰密鑰別名：aws/dynamodb。密鑰存儲在您的帳戶中，並由管理 AWS Key Management Service。 (AWS KMS)。AWS KMS 費用適用。
  - 儲存在您的帳戶中，且由您擁有和管理。密鑰存儲在您的帳戶中，並由管理 AWS Key Management Service。 (AWS KMS)。AWS KMS 費用適用。

#### Note

如果您選擇擁有和管理自己的金鑰，請確保已正確設定 KMS 金鑰政策。如需詳細資訊，請參閱[客戶受管金鑰的金鑰政策](#)。

然後，選擇 Save (儲存) 以更新加密資料表。若要確認加密類型，檢查 Overview (概觀) 索引標籤下的資料表詳細資訊。

## 更新加密金鑰 (AWS CLI)

下列範例顯示如何使用 AWS CLI 更新加密資料表。

### 使用預設值更新加密資料表 AWS 擁有的金鑰

- 更新加密的 Music 資料表，如下列範例所示。

```
aws dynamodb update-table \
 --table-name Music \
 --sse-specification Enabled=false
```

#### Note

此表格現在已使用 DynamoDB 服務帳戶 AWS 擁有的金鑰 中的預設值加密。

若要使用適用 AWS 受管金鑰 於 DynamoDB 的加密資料表更新

- 更新加密的 Music 資料表，如下列範例所示。

```
aws dynamodb update-table \
 --table-name Music \
 --sse-specification Enabled=true
```

資料表描述的 SSEDescription 狀態會設為 ENABLED，而 SSEType 設為 KMS。

```
"SSEDescription": {
 "SSEType": "KMS",
 "Status": "ENABLED",
 "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

使用 DynamoDB 由客戶受管金鑰更新加密資料表

- 更新加密的 Music 資料表，如下列範例所示。

```
aws dynamodb update-table \
 --table-name Music \
 --sse-specification Enabled=true
```



```
--table-name Music \
--sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

資料表描述的 SSEDescription 狀態會設為 ENABLED，而 SSEType 設為 KMS。

```
"SSEDescription": {
 "SSEType": "KMS",
 "Status": "ENABLED",
 "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

## DynamoDB Accelerator 中的資料保護

Amazon DynamoDB Accelerator (DAX) 靜態加密可協助保護您的資料免於發生未經授權的基礎儲存存取，為資料提供另一層保護。組織政策、行業或政府法規，以及合規要求可能需要使用靜態加密來保護您的資料。您可以使用加密來提高部署在雲端中的應用程式資料安全性。

如需 DAX 中資料保護的詳細資訊，請參閱 [DAX 靜態加密](#)。

## 網際網路流量隱私權

Amazon DynamoDB 和現場部署應用程式之間以及 DynamoDB 與相同區域內的其他 AWS 資源之間的連線都受到保護。AWS

## 端點所需的政策

Amazon DynamoDB 提供 [DescribeEndpoints](#) API，可讓您列舉區域端點資訊。對於來自 VPC 端點的請求，IAM 和虛擬私有雲端 (VPC) 端點政策都必須使用 IAM dynamodb:DescribeEndpoints 動作針對請求身分和存取管理 (IAM) 主體授權 DescribeEndpoints API 呼叫。否則，將拒絕存取 DescribeEndpoints API。存取 DynamoDB 的公有端點時，DescribeEndpointsAPI 呼叫的 IAM 和 VPC 端點政策授權步驟不適用。

以下是端點政策的範例。

```
{
 "Version": "2012-10-17",
 "Statement": [

```

```
{
 {
 "Effect": "Allow",
 "Principal": "(Include IAM Principals)",
 "Action": "dynamodb:DescribeEndpoints",
 "Resource": "*"
 }
}
```

服務和內部部署用戶端與應用程式之間的流量。

您的私人網路之間有兩個連線選項，以及 AWS：

- 一個 AWS Site-to-Site VPN 連接。如需詳細資訊，請參閱《AWS Site-to-Site VPN 使用者指南》中的[什麼是 AWS Site-to-Site VPN ?](#)。
- 一個 AWS Direct Connect 連接。如需詳細資訊，請參閱《AWS Direct Connect 使用者指南》中的[什麼是 AWS Direct Connect ?](#)。

透過網路存取 DynamoDB 是透過 AWS 已發佈的 API 進行的。用戶端必須支援 Transport Layer Security (TLS) 1.2。我們建議使用 TLS 1.3。用戶端也必須支援具備完整轉寄密碼 (PFS) 的密碼套件，例如暫時性 Diffie-Hellman (DHE) 或橢圓曲線 Diffie-Hellman Ephemeral (ECDHE)。現代系統 (如 Java 7 和更新版本) 大多會支援這些模式。此外，您必須使用存取金鑰 ID，以及與 IAM 委託人相關聯的私密存取金鑰來簽署請求，或者您可以使用 [AWS Security Token Service \(STS\)](#) 來產生臨時安全憑證來簽署請求。

## 相同區域中 AWS 資源間的流量

適用於 DynamoDB 的 Amazon Virtual Private Cloud (Amazon VPC) 端點是 VPC 中的邏輯實體，僅允許連線到 DynamoDB。Amazon VPC 會將請求路由至 DynamoDB，並將回應路由回 VPC。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的[VPC 端點](#)。如需可從 VPC 端點控制存取權限的政策範例，請參閱[使用 IAM 政策控制對 DynamoDB 的存取](#)。

### Note

無法透過 AWS Site-to-Site VPN 或 AWS Direct Connect 存取 Amazon VPC 端點。

# AWS Identity and Access Management (IAM)

AWS Identity and Access Management 是協助系統管理員安全地控制 AWS 資源存取的 AWS 服務。管理員可以控制透過驗證 (已登入) 和授權 (具有許可) 來使用 Amazon DynamoDB 和 DynamoDB Accelerator 資源的對象。您可以使用 IAM 來管理存取許可，並為 Amazon DynamoDB 和 DynamoDB Accelerator 實作安全政策。IAM 是一項無需額外付費即可使用的 AWS 服務。

## 主題

- [Amazon DynamoDB 的 Identity and Access Management](#)
- [使用 IAM 政策條件進行精細定義存取控制](#)
- [DynamoDB Accelerator 中的身分和存取管理](#)

## Amazon DynamoDB 的 Identity and Access Management

AWS Identity and Access Management (IAM) 可協助管理員安全地控制 AWS 資源存取權。AWS 服務 IAM 管理員可以控制誰能「完成身分驗證」(已登入) 和「獲得授權」(具有許可) 而得以使用 DynamoDB 資源。IAM 是您可以使用的 AWS 服務，無需額外付費。

## 主題

- [物件](#)
- [使用身分驗證](#)
- [使用政策管理存取權](#)
- [Amazon DynamoDB 如何搭配 IAM 運作](#)
- [Amazon DynamoDB 以身分為基礎的政策範例](#)
- [Amazon DynamoDB 身分識別和存取疑難排解](#)
- [避免購買 DynamoDB 預留容量的 IAM 政策](#)

## 物件

您使用 AWS Identity and Access Management (IAM) 的方式會因您在 DynamoDB 中執行的工作而有所不同。

服務使用者：如果使用 DynamoDB 執行任務，管理員會為您提供所需的憑證和許可。隨著您為了執行作業而使用的 DynamoDB 功能數量變多，您可能會需要額外的許可。了解存取許可的管理方式可協助

您向管理員請求正確的許可。若您無法存取 DynamoDB 中的某項功能，請參閱 [Amazon DynamoDB 身分識別和存取疑難排解](#)。

**服務管理員：**如果您負責公司內的 DynamoDB 資源，您可能具備 DynamoDB 的完整存取權限。您的任務是判斷服務使用者應存取的 DynamoDB 功能及資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司可搭配 DynamoDB 使用 IAM 的方式，請參閱 [Amazon DynamoDB 如何搭配 IAM 運作](#)。

**IAM 管理員：**如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 DynamoDB 存取權的詳細資訊。若要檢視您可以在 IAM 中使用的範例 DynamoDB 身分型政策，請參閱 [Amazon DynamoDB 以身分為基礎的政策範例](#)。

## 使用身分驗證

驗證是您 AWS 使用身分認證登入的方式。您必須以 IAM 使用者身分或假設 IAM 角色進行驗證 (登入 AWS)。AWS 帳戶根使用者

您可以使用透過 AWS 身分識別來源提供的認證，以聯合身分識別身分登入。AWS IAM Identity Center (IAM 身分中心) 使用者、貴公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料都是聯合身分識別的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使 AWS 用同盟存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需有關登入的詳細資訊 AWS，請參閱《AWS 登入 使用指南》AWS 帳戶中的 [如何登入](#) 您的。

如果您 AWS 以程式設計方式存取，請 AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以使用您的認證以加密方式簽署要求。如果您不使用 AWS 工具，則必須自行簽署要求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱 IAM 使用者指南中的 [簽署 AWS API 請求](#)。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如，AWS 建議您使用多重要素驗證 (MFA) 來增加帳戶的安全性。如需更多資訊，請參閱 AWS IAM Identity Center 使用者指南中的 [多重要素驗證](#) 和 IAM 使用者指南中的 [在 AWS 中使用多重要素驗證 \(MFA\)](#)。

## AWS 帳戶 根使用者

當您建立時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取該帳戶中的所有資源 AWS 服務和資源。此身分稱為 AWS 帳戶 root 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱 IAM 使用者指南中的 [需要根使用者憑證的任務](#)。

## 聯合身分

最佳作法是要求人類使用者 (包括需要系統管理員存取權的使用者) 使用與身分識別提供者的同盟，才能使用臨時認證 AWS 服務 來存取。

聯合身分識別是來自企業使用者目錄的使用者、Web 身分識別提供者、Identity Center 目錄，或使用透過身分識別來源提供的認證進行存取 AWS 服務 的任何使用者。AWS Directory Service 同盟身分存取時 AWS 帳戶，他們會假設角色，而角色則提供臨時認證。

對於集中式存取權管理，我們建議您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中建立使用者和群組，也可以連線並同步到自己身分識別來源中的一組使用者和群組，以便在所有應用程式 AWS 帳戶 和應用程式中使用。如需 IAM Identity Center 的相關資訊，請參閱 [AWS IAM Identity Center 使用者指南中的什麼是 IAM Identity Center ?](#)。

## IAM 使用者和群組

[IAM 使用者](#)是您內部的身分，具 AWS 帳戶 有單一人員或應用程式的特定許可。建議您盡可能依賴暫時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱 [IAM 使用者指南](#)中的為需要長期憑證的使用案例定期輪換存取金鑰。

[IAM 群組](#)是一種指定 IAM 使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。如需進一步了解，請參閱 IAM 使用者指南中的[建立 IAM 使用者 \(而非角色\) 的時機](#)。

## IAM 角色

[IAM 角色](#)是您 AWS 帳戶 內部具有特定許可的身分。它類似 IAM 使用者，但不與特定的人員相關聯。您可以[切換角色，在中暫時擔任 IAM 角色](#)。AWS Management Console 您可以透過呼叫 AWS CLI 或 AWS API 作業或使用自訂 URL 來擔任角色。如需使用角色的方法更多相關資訊，請參閱 IAM 使用者指南中的[使用 IAM 角色](#)。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 – 若要向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱 [IAM 使用者指南](#)中的為第三方身分提供者建立角色。如果您使用 IAM Identity Center，

則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的[許可集](#)。

- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權 – 您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的委託人) 存取您帳戶中的資源。角色是授予跨帳戶存取權的主要方式。但是，對於某些策略 AWS 服務，您可以將策略直接附加到資源 (而不是使用角色作為代理)。若要了解跨帳戶存取權角色和資源型政策間的差異，請參閱 IAM 使用者指南中的[IAM 角色與資源類型政策的差異](#)。
- 跨服務訪問 — 有些 AWS 服務 使用其他 AWS 服務功能。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
- 轉寄存取工作階段 (FAS) — 當您使用 IAM 使用者或角色在中執行動作時 AWS，您會被視為主體。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 會使用主體呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。只有當服務收到需要與其 AWS 服務 他資源互動才能完成的請求時，才會發出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱 [《轉發存取工作階段》](#)。
- 服務角色 – 服務角色是服務擔任的 [IAM 角色](#)，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需更多資訊，請參閱 IAM 使用者指南中的[建立角色以委派許可給 AWS 服務](#)。
- 服務連結角色 — 服務連結角色是連結至 AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的中，AWS 帳戶 且屬於服務所有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 — 您可以使用 IAM 角色來管理在 EC2 執行個體上執行的應用程式以及發出 AWS CLI 或 AWS API 請求的臨時登入資料。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並提供給其所有應用程式，請建立連接至執行個體的執行個體設定檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得暫時憑證。如需更多資訊，請參閱 IAM 使用者指南中的[利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式](#)。

若要了解是否要使用 IAM 角色或 IAM 使用者，請參閱 IAM 使用者指南中的[建立 IAM 角色 \(而非使用者\) 的時機](#)。

## 使用政策管理存取權

您可以透 AWS 過建立原則並將其附加至 AWS 身分識別或資源來控制中的存取。原則是一個物件 AWS，當與身分識別或資源相關聯時，會定義其權限。AWS 當主參與者 (使用者、root 使用者或角色



工作階段) 提出要求時，評估這些原則。政策中的許可決定是否允許或拒絕請求。大多數原則會以 AWS 以 JSON 文件的形式儲存在中。如需 JSON 政策文件結構和內容的更多相關資訊，請參閱 IAM 使用者指南中的 [JSON 政策概觀](#)。

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該原則的使用者可以從 AWS Management Console、AWS CLI、或 AWS API 取得角色資訊。

## 身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱 IAM 使用者指南中的 [建立 IAM 政策](#)。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管理的策略是獨立策略，您可以將其附加到您的 AWS 帳戶。受管政策包括 AWS 受管政策和客戶管理的策略。若要了解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的 [在受管政策和內嵌政策間選擇](#)。

## 資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中 [指定主體](#)。主參與者可以包括帳戶、使用者、角色、同盟使用者或。AWS 服務

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用 IAM 的 AWS 受管政策。

## 存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些委託人 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 和 Amazon VPC 是支援 ACL 的服務範例。AWS WAF若要進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的[存取控制清單 \(ACL\) 概觀](#)。

## 其他政策類型

AWS 支援其他較不常見的原則類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 Principal 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可範圍的更多相關資訊，請參閱 IAM 使用者指南中的[IAM 實體許可範圍](#)。
- 服務控制策略 (SCP) — SCP 是 JSON 策略，用於指定中組織或組織單位 (OU) 的最大權限。AWS Organizations 是一種用於分組和集中管理您企業擁有的多個 AWS 帳戶的服務。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 限制成員帳戶中實體的權限，包括每個 AWS 帳戶根使用者帳戶。如需組織和 SCP 的更多相關資訊，請參閱 AWS Organizations 使用者指南中的[SCP 運作方式](#)。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需更多資訊，請參閱 IAM 使用者指南中的[工作階段政策](#)。

## 多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。要了解如何在涉及多個政策類型時 AWS 確定是否允許請求，請參閱《IAM 使用者指南》中的[政策評估邏輯](#)。

## Amazon DynamoDB 如何搭配 IAM 運作

在您使用 IAM 管理 DynamoDB 的存取權之前，請了解搭配 DynamoDB 使用的 IAM 功能有哪些。

您可以搭配 Amazon DynamoDB 使用的 IAM 功能

IAM 功能	DynamoDB 支援
<a href="#">身分型政策</a>	是
<a href="#">資源型政策</a>	是



IAM 功能	DynamoDB 支援
<a href="#">政策動作</a>	是
<a href="#">政策資源</a>	是
<a href="#">政策條件索引鍵</a>	是
<a href="#">ACL</a>	否
<a href="#">ABAC(政策中的標籤)</a>	部分
<a href="#">臨時憑證</a>	是
<a href="#">主體許可</a>	是
<a href="#">服務角色</a>	是
<a href="#">服務連結角色</a>	否

若要取得 DynamoDB 和其他 AWS 服務如何搭配大多數 IAM 功能運作的高階檢視，請參閱 IAM 使用者指南中的[搭配 IAM 使用的 AWS 服務](#)。

適用於 DynamoDB 的以身分為基礎的政策

支援身分型政策 是

身分型政策是可以連接到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱《IAM 使用者指南》中的[建立 IAM 政策](#)。

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。您無法在身分型政策中指定主體，因為這會套用至連接的使用者或角色。如要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的[IAM JSON 政策元素參考](#)。

DynamoDB 以身分為基礎的政策範例

若要檢視 DynamoDB 以身分為基礎的政策範例，請參閱[Amazon DynamoDB 以身分為基礎的政策範例](#)。

## DynamoDB 內以資源為基礎的政策

支援以資源基礎的政策 是

資源型政策是附加到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中[指定主體](#)。主參與者可以包括帳戶、使用者、角色、同盟使用者或。AWS 服務

若要啟用跨帳戶存取，您可以指定在其他帳戶內的所有帳戶或 IAM 實體，作為資源型政策的主體。新增跨帳戶主體至資源型政策，只是建立信任關係的一半。當主體和資源位於不同時 AWS 帳戶，受信任帳戶中的 IAM 管理員也必須授與主體實體 (使用者或角色) 權限，才能存取資源。其透過將身分型政策連接到實體來授與許可。不過，如果資源型政策會為相同帳戶中的主體授予存取，這時就不需要額外的身分型政策。如需詳細資訊，請參閱《IAM 使用者指南》中的[IAM 角色與資源型政策有何差異](#)。

### DynamoDB 的政策動作

支援政策動作 是

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 Action 元素描述您可以用來允許或拒絕政策中存取的動作。原則動作通常與關聯的 AWS API 作業具有相同的名稱。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯動作的許可。

若要查看 DynamoDB 動作的清單，請參閱服務授權參考中的[Amazon DynamoDB 定義的動作](#)。

DynamoDB 中的政策動作會在動作之前使用以下字首：

```
aws
```

若要在單一陳述式中指定多個動作，請用逗號分隔。

```
"Action": [
```

```
"aws:action1",
"aws:action2"
]
```

若要檢視 DynamoDB 以身分為基礎的政策範例，請參閱 [Amazon DynamoDB 以身分為基礎的政策範例](#)。

## DynamoDB 的政策資源

支援政策資源 **是**

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Resource JSON 政策元素可指定要套用動作的物件。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 [Amazon Resource Name \(ARN\)](#) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出操作)，請使用萬用字元 (\*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

若要查看 DynamoDB 資源類型清單及其 ARN，請參閱《服務授權參考》中的 [Amazon DynamoDB 定義的資源](#)。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 [Amazon DynamoDB 定義的動作](#)。

若要檢視 DynamoDB 以身分為基礎的政策範例，請參閱 [Amazon DynamoDB 以身分為基礎的政策範例](#)。

## DynamoDB 的政策條件索引鍵

支援服務特定政策條件金鑰 **是**

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Condition 元素 (或 Condition 區塊) 可讓您指定使陳述式生效的條件。Condition 元素是選用項目。您可以建立使用[條件運算子](#)的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個索引鍵，AWS 會使用邏輯 AND 操作評估他們。如果您為單一條件索引鍵指定多個值，請使用邏輯 OR 運算來 AWS 評估條件。必須符合所有條件，才會授與陳述式的許可。

您也可以指定條件時使用預留位置變數。例如，您可以只在使用者使用其 IAM 使用者名稱標記時，將存取資源的許可授予該 IAM 使用者。如需更多資訊，請參閱 IAM 使用者指南中的[IAM 政策元素：變數和標籤](#)。

AWS 支援全域條件金鑰和服務特定條件金鑰。若要查看所有 AWS 全域條件金鑰，請參閱《IAM 使用者指南》中的[AWS 全域條件內容金鑰](#)。

若要查看 DynamoDB 條件索引鍵的清單，請參閱《服務授權參考》中的[Amazon DynamoDB 的條件索引鍵](#)。若要了解您可以搭配哪些動作和資源使用條件索引鍵，請參閱[Amazon DynamoDB 定義的動作](#)。

若要檢視 DynamoDB 以身分為基礎的政策範例，請參閱[Amazon DynamoDB 以身分為基礎的政策範例](#)。

#### DynamoDB 中的存取控制清單 (ACL)

支援 ACL 否

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

#### 搭配 DynamoDB 的屬性型存取控制 (ABAC)

支援 ABAC (政策中的標籤) 部分

屬性型存取控制 (ABAC) 是一種授權策略，可根據屬性來定義許可。在中 AWS，這些屬性稱為標籤。您可以將標籤附加到 IAM 實體 (使用者或角色) 和許多 AWS 資源。為實體和資源加上標籤是 ABAC 的第一步。您接著要設計 ABAC 政策，允許在主體的標籤與其嘗試存取的資源標籤相符時操作。

ABAC 在成長快速的環境中相當有幫助，並能在政策管理變得繁瑣時提供協助。

若要根據標籤控制存取，請使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件金鑰，在政策的 [條件元素](#) 中，提供標籤資訊。

如果服務支援每個資源類型的全部三個條件金鑰，則對該服務而言，值為 Yes。如果服務僅支援某些資源類型的全部三個條件金鑰，則值為 Partial。

如需 ABAC 的詳細資訊，請參閱《IAM 使用者指南》中的 [什麼是 ABAC?](#)。如要查看含有設定 ABAC 步驟的教學課程，請參閱《IAM 使用者指南》中的 [使用屬性型存取控制 \(ABAC\)](#)。

將臨時憑證與 DynamoDB 搭配使用

支援臨時憑證 是

當您使用臨時憑據登錄時，某些 AWS 服務 不起作用。如需其他資訊，包括哪些 AWS 服務 與臨時登入資料 [搭配 AWS 服務 使用](#)，請參閱 IAM 使用者指南中的 IAM。

如果您使用除了使用者名稱和密碼以外的任何方法登入，則您正在 AWS Management Console 使用臨時認證。例如，當您 AWS 使用公司的單一登入 (SSO) 連結存取時，該程序會自動建立暫時認證。當您以使用者身分登入主控台，然後切換角色時，也會自動建立臨時憑證。如需切換角色的詳細資訊，請參閱《IAM 使用者指南》中的 [切換至角色 \(主控台\)](#)。

您可以使用 AWS CLI 或 AWS API 手動建立臨時登入資料。然後，您可以使用這些臨時登入資料來存取 AWS。AWS 建議您動態產生臨時登入資料，而不是使用長期存取金鑰。如需詳細資訊，請參閱 [IAM 中的暫時性安全憑證](#)。

DynamoDB 的跨服務主體許可

支援轉寄存取工作階段 (FAS) 是

當您使用 IAM 使用者或角色在中執行動作時 AWS，您會被視為主體。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 會使用主體呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。只有當服務收到需要與其 AWS 服務 他資源互動才能完成的請求時，才會發出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱 [《轉發存取工作階段》](#)。

DynamoDB 的服務角色

支援服務角色 是

服務角色是服務擔任的 [IAM 角色](#)，可代您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需更多資訊，請參閱 IAM 使用者指南中的 [建立角色以委派許可給 AWS 服務](#)。

#### Warning

變更服務角色的許可有可能會讓 DynamoDB 功能出現故障。只有 DynamoDB 提供指引時，才能編輯服務角色。

## DynamoDB 的服務連結角色

支援服務連結角色。

否

服務連結角色是一種連結至 AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的中，AWS 帳戶且屬於服務所有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

如需建立或管理服务連結角色的詳細資訊，請參閱 [可搭配 IAM 運作的 AWS 服務](#)。在表格中尋找服務，其中包含服務連結角色欄中的 Yes。選擇是連結，以檢視該服務的服務連結角色文件。

## Amazon DynamoDB 以身分為基礎的政策範例

根據預設，使用者和角色不具備建立或修改 DynamoDB 資源的許可。他們也無法使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或 AWS API 來執行工作。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

若要了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的 [建立 IAM 政策](#)。

如需 DynamoDB 所定義之動作和資源類型的詳細資訊，包括每種資源類型的 ARN 格式，請參閱《服務授權參考》中 [Amazon DynamoDB 適用的動作、資源和條件索引鍵](#)。

### 主題

- [政策最佳實務](#)
- [使用 DynamoDB 主控台](#)
- [允許使用者檢視他們自己的許可](#)
- [搭配 Amazon DynamoDB 使用身分型政策](#)



## 政策最佳實務

以身分為基礎的政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 DynamoDB 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管原則並邁向最低權限權限 — 若要開始將權限授與使用者和工作負載，請使用可授與許多常見使用案例權限的 AWS 受管理原則。它們在您的 AWS 帳戶。建議您透過定義特定於您使用案例的 AWS 客戶管理政策，進一步降低使用權限。如需更多資訊，請參閱 IAM 使用者指南中的 [AWS 受管政策](#) 或 [任務職能的 AWS 受管政策](#)。
- 套用最低許可許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的權限。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱 IAM 使用者指南中的 [IAM 中的政策和許可](#)。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。您也可以使用條件來授與對服務動作的存取權 (如透過特定) 使用這些動作 AWS 服務，例如 AWS CloudFormation。如需更多資訊，請參閱 IAM 使用者指南中的 [IAM JSON 政策元素：條件](#)。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您編寫安全且實用的政策。如需更多資訊，請參閱 IAM 使用者指南中的 [IAM Access Analyzer 政策驗證](#)。
- 需要多因素身份驗證 (MFA) — 如果您的案例需要 IAM 使用者或根使用者 AWS 帳戶，請開啟 MFA 以獲得額外的安全性。若要在呼叫 API 作業時請求 MFA，請將 MFA 條件新增至您的政策。如需更多資訊，請參閱 [IAM 使用者指南](#) 中的設定 MFA 保護的 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱 IAM 使用者指南中的 [IAM 安全最佳實務](#)。

### 使用 DynamoDB 主控台

若要存取 Amazon DynamoDB 主控台，您必須擁有最基本的一組許可。這些權限必須允許您列出和檢視有關 AWS 帳戶如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

您不需要為僅對 AWS CLI 或 AWS API 進行呼叫的使用者允許最低主控台權限。反之，只需允許存取符合他們嘗試執行之 API 操作的動作就可以了。

為確保使用者和角色仍可使用 DynamoDB 主控台，請同時將 `DynamoDB ConsoleAccess` 或 `ReadOnly` AWS 受管理的政策附加到實體。如需詳細資訊，請參閱《IAM 使用者指南》中的 [新增許可到使用者](#)。

## 允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此原則包含在主控台上或以程式設計方式使用 AWS CLI 或 AWS API 完成此動作的權限。

```
{
 "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": "*"
 }
]
}
```

## 搭配 Amazon DynamoDB 使用身分型政策

本主題涵蓋在 Amazon DynamoDB 中使用以身分為基礎的 AWS Identity and Access Management (IAM) 政策，並提供範例。這些範例會示範帳戶管理員如何將許可政策連接至 IAM 身分 (使用者、群組和角色)，並藉此授與許可，以取得在 Amazon DynamoDB 資源上執行操作的許可。



本主題中的各節涵蓋下列內容：

- [使用 Amazon DynamoDB 主控台所需的 IAM 許可](#)
- [AWS 適用於 Amazon DynamoDB 的受管 \(預先定義\) IAM 政策](#)
- [客戶受管政策範例](#)

以下是許可政策的範例。

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

上述政策具有一個陳述式 `dynamodb:DescribeTable` , `dynamodb:Query` 可針對 `us-west-2` AWS 區域中某個資料表 (由指定的 AWS 帳戶擁有 `dynamodb:Scan`) 上的三個 DynamoDB 動作 (、和) 授與權限。 `account-id` Resource 值中的 Amazon Resource Name (ARN) 會指定許可適用的資料表。

使用 Amazon DynamoDB 主控台所需的 IAM 許可

若要使用 DynamoDB 主控台，使用者必須具有允許使用者使用其 AWS 帳戶 DynamoDB 資源的最低權限集。除了這些 DynamoDB 許可之外，主控台還需要以下許可：

- Amazon CloudWatch 許可以顯示指標和圖形。
- AWS Data Pipeline 匯出和匯入 DynamoDB 資料的權限。
- AWS Identity and Access Management 存取匯出和匯入所需角色的權限。
- Amazon 簡單通知服務許可，可在觸發 CloudWatch 警報時通知您。
- AWS Lambda 處理 DynamoDB Streams 記錄的權限。

如果您建立比最基本必要許可更嚴格的 IAM 政策，則對於採取該 IAM 政策的使用者而言，主控台就無法如預期運作。為確保這些使用者仍然可以使用 DynamoDB 主控台，請同時將 AmazonDynamoDBReadOnlyAccess AWS 受管理的政策附加到使用者，如中所述。[AWS 適用於 Amazon DynamoDB 的受管 \(預先定義\) IAM 政策](#)

您不需要為只對 AWS CLI 或 Amazon DynamoDB API 撥打電話的使用者允許最低主控台許可。

#### Note

如果您參考 VPC 人雲端端點，您還需要使用 IAM 動作 (Dynamodb:) 授權請求 IAM 主體的 DescribeEndpoints API 呼叫。DescribeEndpoints 如需更多資訊，請參閱[端點所需的政策](#)。

### AWS 適用於 Amazon DynamoDB 的受管 (預先定義) IAM 政策

AWS 透過提供由建立和管理的獨立 IAM 政策來解決一些常見使用案例 AWS。這些 AWS 受管理的政策會為常見使用案例授與必要的權限，因此您可以避免調查需要哪些權限。如需詳細資訊，請參閱《IAM 使用者指南》中的 [AWS 受管政策](#)。

下列 AWS 受管政策 (您可以附加至帳戶中的使用者) 專屬於 DynamoDB，並依使用案例案例分組：

- AmazonDynamo資料庫 ReadOnlyAccess — 透過授與對 DynamoDB 資源的唯讀存取權。AWS Management Console
- AmazonDynamo資料庫 FullAccess — 透過授與對 DynamoDB 資源的完整存取權。AWS Management Console

您可以登入 IAM 主控台並在該處搜尋特定政策，以檢閱這些 AWS 受管許可政策。

#### Important

最佳實務是建立自訂 IAM 政策，授予[最低權限](#)給需要的使用者、角色或群組。

### 客戶受管政策範例

在本節中，您可以找到授予各種 DynamoDB 動作許可之政策範例。當您使用 AWS SDK 或 AWS CLI 在您使用主控台時，需要授予特定的其他許可給主控台。如需詳細資訊，請參閱 [使用 Amazon DynamoDB 主控台所需的 IAM 許可](#)。

**Note**

下列所有策略範例均使用其中一個 AWS 區域，並包含虛擬帳號 ID 和表格名稱。

範例：

- [將許可授與資料表上所有 DynamoDB 動作的 IAM 政策](#)
- [用來授予 DynamoDB 資料表項目唯讀許可的 IAM 政策](#)
- [授予特定 DynamoDB 資料表及其索引存取權限的 IAM 政策](#)
- [讀取、寫入、更新和刪除 DynamoDB 資料表存取權限的 IAM 政策](#)
- [用於在同一帳戶中分隔 DynamoDB 環境的 IAM 政策 AWS](#)
- [避免購買 DynamoDB 預留容量的 IAM 政策](#)
- [僅授予 DynamoDB 串流讀取存取權限的 IAM 政策 \(不適用於資料表\)](#)
- [允許 AWS Lambda 函數存取 DynamoDB 串流記錄的 IAM 政策](#)
- [允許對 DynamoDB Accelerator \(DAX\) 叢集進行讀取和寫入的 IAM 政策](#)

《IAM 使用者指南》包含[三個其他 DynamoDB 範例](#)：

- [Amazon DynamoDB：允許存取特定資料表](#)
- [Amazon DynamoDB：允許存取特定資料欄](#)
- [Amazon DynamoDB：允許根據 Amazon Cognito ID 對 DynamoDB 進行資料列層級存取](#)

將許可授與資料表上所有 DynamoDB 動作的 IAM 政策

下列政策會對資料表上名為 Books 的所有 DynamoDB 動作授予許可。在中指定的資源 ARN Resource 可識別特定 AWS 區域中的表格。如果以萬用字元 (\*) 代替 ResourceBooks ARN 中的資料表名稱，則允許對帳戶中的所有資料表執行所有 DynamoDB 動作。在此或任何 IAM 政策上使用萬用字元之前，請仔細考慮可能發生的安全性影響。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "AllAPIActionsOnBooks",
 "Effect": "Allow",
```

```
 "Action": "dynamodb:*",
 "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
 }
]
}
```

### Note

這是使用萬用字元 (\*) 允許所有動作的範例，包括管理、資料操作、監控和購買 DynamoDB 預留容量。反之，最佳實務是明確指定要授予的每個動作，以及只指定該使用者、角色或群組所需的動作。

用來授予 DynamoDB 資料表項目唯讀許可的 IAM 政策

下列許可政策只會授予 GetItem、BatchGetItem、Scan、Query 和 ConditionCheckItem DynamoDB 動作的許可，進而在 Books 資料表上設定唯讀存取權。

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

授予特定 DynamoDB 資料表及其索引存取權限的 IAM 政策

下列政策會針對名為 Books 的 DynamoDB 資料表上的資料修改動作和該表所有的索引授予許可。如需索引運作方式的詳細資訊，請參閱 [使用次要索引來改善資料存取](#)。

```
{
```

```
"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/*"
]
 }
]
```

## 讀取、寫入、更新和刪除 DynamoDB 資料表存取權限的 IAM 政策

如果您需要允許應用程式建立、讀取、更新和刪除 Amazon DynamoDB 資料表、索引和串流中的資料，請使用此政策。視情況取代「AWS 地區」名稱、您的帳戶 ID 以及資料表名稱或萬用字元 (\*)。

```
{
 "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/*"
]
 }
]
}

```

若要擴展此原則以涵蓋此帳戶之所有區 AWS 域中的所有 DynamoDB 表格，請使用萬用字元 (\*) 作為區域和資料表名稱。例如：

```

"Resource": [
 "arn:aws:dynamodb:*:123456789012:table/*",
 "arn:aws:dynamodb:*:123456789012:table/*/index/*"
]

```

## 用於在同一帳戶中分隔 DynamoDB 環境的 IAM 政策 AWS

假設您有不同的環境，且每個環境都會維護各自版本的資料表 (名為 ProductCatalog)。如果您在同一 AWS 帳戶中建立兩個 ProductCatalog 資料表，則在一個環境中工作可能會因為權限的設定方式而影響另一個環境。例如，並行控制平面作業數目的配額 (例如 CreateTable) 是在 AWS 帳戶層級設定的。

因此，一個環境中的每個動作都會減少另一個環境中可用的操作數目。還有一種風險是，一個環境中的程式碼可能會在另一個環境中意外存取資料表。

### Note

若想區隔生產和測試工作負載來協助控制事件的潛在「爆炸範圍」，最佳實務是為測試和生產工作負載建立不同的 AWS 帳戶。如需詳細資訊，請參閱 [AWS 帳戶管理與區隔](#)。

進一步假設您有兩位開發人員 (Amit 和 Alice) 正在測試 ProductCatalog 資料表。您的開發人員可以共享相同的測試 AWS 帳戶，而不是每個開發人員都需要單獨的 AWS 帳戶。在此測試帳戶中，您可以建立相同資料表的複本以供每位開發人員處理 (例如 Alice\_ProductCatalog 和 Amit\_ProductCatalog)。在這種情況下，您可以創建用戶愛麗絲和阿米特在您為測試環境創建的 AWS 帳戶。您接著可以將許可授予這些使用者，以對他們擁有的資料表執行 DynamoDB 動作。

若要授予這些 IAM 使用者許可，您可以執行下列其中一項：

- 為每位使用者建立不同的政策，然後分別將每個政策連接至其使用者。例如，您可以將下列政策連接至使用者 Alice，允許她存取 Alice\_ProductCatalog 資料表上的所有 DynamoDB 動作：

```
{
 "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/*"
 }
]
}

```

然後，您可以為使用者 Amit 建立具有不同資源 (Amit\_ProductCatalog 資料表) 的類似政策。

- 您可以使用 IAM 政策變數來撰寫單一政策，並將它連接至群組，而不是將政策連接至個別使用者。您需要建立群組，而在此範例中，會將使用者 Alice 和 Amit 新增至群組。下列範例授予對 `${aws:username}_ProductCatalog` 資料表執行所有 DynamoDB 動作的許可。評估政策時，政策變數 `${aws:username}` 會取代為申請者的使用者名稱。例如，如果 Alice 傳送新增項目的請求，則只有在 Alice 將項目新增至 Alice\_ProductCatalog 資料表時，才允許此動作。

```

{
 "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

使用 IAM 政策變數時，您必須在政策中明確地指定 2012-10-17 版本的 IAM 政策語言。IAM 政策語言 (2008-10-17) 的預設版本不支援政策變數。

您可以對資料表使用萬用字元 (\*) 授予許可 (而不是如同一般作法將特定資料表識別為資源)，在這些資料表中，資料表名稱的前面會加上提出請求的使用者，如下範例所示。

```
"Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/${aws:username}_*"
```

## 避免購買 DynamoDB 預留容量的 IAM 政策

使用 Amazon DynamoDB 預留容量，您會支付一次性預付費用，並承諾在一段時間支付最低消費額，以大幅降低成本。您可以使用 AWS Management Console 檢視和購買保留容量。不過，您可能不希望組織中的所有使用者都可以購買預留容量。如需預留容量的詳細資訊，請參閱 [Amazon DynamoDB 定價](#)。

DynamoDB 提供下列 API 操作，以控制對預留容量管理的存取：

- `dynamodb:DescribeReservedCapacity`：傳回目前生效的預留容量購買。
- `dynamodb:DescribeReservedCapacityOfferings`：傳回 AWS 目前所提供之預留容量方案的詳細資訊。
- `dynamodb:PurchaseReservedCapacityOfferings`：執行實際購買的預留容量。

會 AWS Management Console 使用這些 API 動作來顯示保留容量資訊並進行購買。您無法從應用程式呼叫這些操作，因為只有從主控台才能加以存取。不過，您可以在 IAM 許可政策中允許或拒絕存取這些操作。

下列政策允許使用者使用 AWS Management Console — 檢視預留容量購買和供應項目，但新購買會遭拒。

```
{
 "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:*"
 }
]
}
```

```
}
```

請注意，此政策使用萬用字元 (\*) 來允許所有、和描述許可，以拒絕為所有人購買 DynamoDB 預留容量。

僅授予 DynamoDB 串流讀取存取權限的 IAM 政策 (不適用於資料表)

當您為資料表啟用 DynamoDB Streams 時，會擷取資料表中每個資料項目的修改資訊。如需詳細資訊，請參閱 [DynamoDB Streams 的變更資料擷取](#)。

在部分情況下，建議您防止應用程式讀取 DynamoDB 資料表中的資料，但同時仍然允許存取該資料表的串流。例如，您可以設定 AWS Lambda 為在偵測到項目更新時輪詢串流並叫用 Lambda 函數，然後執行其他處理。

下列動作可用來控制對 DynamoDB Streams 的存取：

- dynamodb:DescribeStream
- dynamodb:GetRecords
- dynamodb:GetShardIterator
- dynamodb:ListStreams

下列範例政策會將存取名為 GameScores 之資料表上串流的許可授予使用者。ARN 中的萬用字元 (\*) 可符合與該資料表建立關聯的任何串流。

```
{
 "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/*"
 }
]
}
```

```
}
```

請注意，此政策會授予存取 GameScores 資料表串流的權限，但不允許存取資料表本身。

允許 AWS Lambda 函數存取 DynamoDB 串流記錄的 IAM 政策

如果您希望根據 DynamoDB 串流中的事件執行某些動作，可以撰寫由這些事件觸發的 AWS Lambda 函數。這類 Lambda 函數需要從 DynamoDB Streams 讀取資料的許可。如需有關搭配使用 Lambda 和 DynamoDB Streams 的詳細資訊，請參閱 [DynamoDB Streams 和觸發器 AWS Lambda](#)。

若要將許可授予 Lambda，請使用與 Lambda 函數 IAM 角色相關聯的許可政策（也稱為執行角色）。在您建立 Lambda 函數時，請指定此政策。

例如，您可以將下列許可政策與執行角色建立關聯，以授予執行所列 DynamoDB Streams 動作的 Lambda 許可。

```
{
 "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/*"
 }
]
}
```

如需詳細資訊，請參閱《AWS Lambda 開發人員指南》中的 [AWS Lambda 許可](#)。

允許對 DynamoDB Accelerator (DAX) 叢集進行讀取和寫入的 IAM 政策

以下政策允許對 DynamoDB Accelerator (DAX) 叢集（而非相關聯的 DynamoDB 資料表），執行讀取、寫入、更新和刪除。若要使用此原則，請替換 AWS 區域名稱、您的帳戶 ID 和 DAX 叢集的名稱。

**Note**

此政策允許存取 DAX 叢集，但不允許存取相關聯的 DynamoDB 資料表。請確定您的 DAX 叢集具有正確政策，可代表您在 DynamoDB 資料表上執行這些相同操作。

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

若要擴充此原則以涵蓋帳戶所有區 AWS 域的 DAX 存取權，請使用萬用字元 (\*) 做為地區名稱。

```
"Resource": "arn:aws:dax:*:123456789012:cache/MyDAXCluster"
```

## Amazon DynamoDB 身分識別和存取疑難排解

請使用以下資訊來協助您診斷和修正使用 DynamoDB 和 IAM 時發生的常見問題。

### 主題

- [我未獲授權，不得在 DynamoDB 中執行動作](#)
- [我沒有授權執行 iam : PassRole](#)

- [我想允許我以外的人員存 AWS 帳戶 取我的 DynamoDB 資源](#)

我未獲授權，不得在 DynamoDB 中執行動作

如果 AWS Management Console 告訴您您沒有執行動作的授權，則您必須聯絡管理員以尋求協助。您的管理員是提供您使用者名稱和密碼的人員。

下列範例錯誤會在 mateojackson 使用者嘗試使用主控台檢視一個虛構 *my-example-widget* 資源的詳細資訊，但卻無虛構 `aws:GetWidget` 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
aws:GetWidget on resource: my-example-widget
```

在此情況下，Mateo 會請求管理員更新他的政策，允許他使用 *my-example-widget* 動作存取 `aws:GetWidget` 資源。

我沒有授權執行 iam : PassRole

如果您收到錯誤，告知您無權執行 `iam:PassRole` 動作，您的政策必須更新，允許您將角色傳遞給 DynamoDB。

有些 AWS 服務 允許您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 marymajor 的 IAM 使用者嘗試使用主控台在 DynamoDB 中執行動作時，發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 `iam:PassRole` 動作。

如果您需要協助，請聯絡您的 AWS 系統管理員。您的管理員提供您的簽署憑證。

我想允許我以外的人員存 AWS 帳戶 取我的 DynamoDB 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 DynamoDB 是否支援這些功能，請參閱 [Amazon DynamoDB 如何搭配 IAM 運作](#)。
- 若要了解如何提供對您所擁有資源 AWS 帳戶 的存取權，請參閱 [IAM 使用者指南中您擁有的另一 AWS 帳戶 個 IAM 使用者提供存取權限](#)。
- 若要了解如何將資源存取權提供給第三方 AWS 帳戶，請參閱 IAM 使用者指南中的 [提供第三方 AWS 帳戶 擁有的存取權](#)。
- 若要了解如何透過聯合身分提供存取權，請參閱 IAM 使用者指南中的 [將存取權提供給在外部進行身分驗證的使用者 \(聯合身分\)](#)。
- 若要了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱 IAM 使用者指南中的 [IAM 角色 與資源型政策的差異](#)。

## 避免購買 DynamoDB 預留容量的 IAM 政策

使用 Amazon DynamoDB 預留容量，您會支付一次性預付費用，並承諾在一段時間支付最低消費額，以大幅降低成本。您可以使用 AWS Management Console 檢視和購買保留容量。不過，您可能不希望組織中的所有使用者都可以購買預留容量。如需預留容量的詳細資訊，請參閱 [Amazon DynamoDB 定價](#)。

DynamoDB 提供下列 API 操作，以控制對預留容量管理的存取：

- `dynamodb:DescribeReservedCapacity`：傳回目前生效的預留容量購買。
- `dynamodb:DescribeReservedCapacityOfferings`：傳回 AWS 目前所提供之預留容量方案的詳細資訊。
- `dynamodb:PurchaseReservedCapacityOfferings`：執行實際購買的預留容量。

會 AWS Management Console 使用這些 API 動作來顯示保留容量資訊並進行購買。您無法從應用程式呼叫這些操作，因為只有從主控台才能加以存取。不過，您可以在 IAM 許可政策中允許或拒絕存取這些操作。

下列政策允許使用者使用 AWS Management Console — 檢視預留容量購買和供應項目，但新購買會遭拒。

```
{
 "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:*"
 }
]
```

請注意，此政策使用萬用字元 (\*) 來允許所有、和描述許可，以拒絕為所有人購買 DynamoDB 預留容量。

## 使用 IAM 政策條件進行精細定義存取控制

您在 DynamoDB 中授予許可時，可以指定條件，以決定許可政策的生效方式。

### 概觀

在 DynamoDB 中，您可以選擇在使用 IAM 政策授予許可時指定條件 (請參閱 [Amazon DynamoDB 的 Identity and Access Management](#))。例如，您可以：

- 授予許可，允許使用者唯讀存取資料表或次要索引中的特定項目和屬性。
- 授予許可，允許使用者根據該使用者的身分唯寫存取資料表中的特定屬性。

在 DynamoDB 中，您可以使用條件金鑰在 IAM 政策中指定條件，如下節中的使用案例所述。

#### Note

某些 AWS 服務也支援以標籤為基礎的條件，但 DynamoDB 不支援。

### 許可使用案例

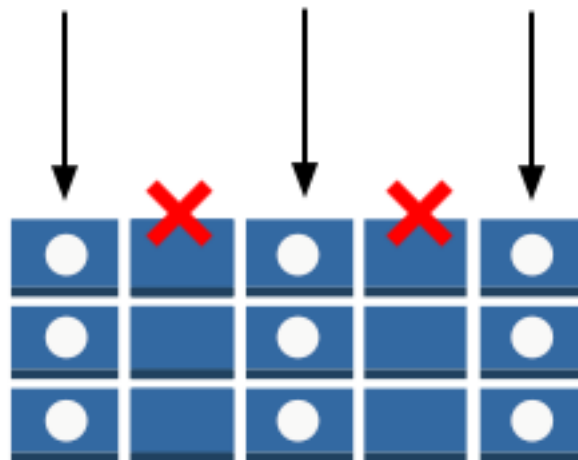
除了控制對 DynamoDB API 動作的存取之外，您也可以控制對個別資料項目和屬性的存取。例如，您可以執行下列動作：



- 授予資料表的許可，但根據特定主索引鍵值來限制存取該資料表中的特定項目。範例可能是遊戲的社交聯網應用程式，而所有使用者的已儲存遊戲資料都會存放在單一資料表中，但使用者無法存取他們未擁有的資料項目，如下圖所示：



- 隱藏資訊，讓使用者只能看到一部分的屬性。範例可能是根據使用者位置來顯示附近機場之航班資料的應用程式。航空公司名稱、到達和起飛時間，以及航班編號全部會顯示。不過，會隱藏駕駛員名稱或乘客數目這類屬性，如下圖所示：



若要實作這類精細定義存取控制，您可以撰寫 IAM 許可政策，以指定條件來存取安全登入資料和相關聯許可。您接著將政策套用至使用 IAM 主控台所建立的使用者、群組或角色。您的 IAM 政策可以限制對資料表中個別項目的存取、對這些項目中屬性的存取，或同時限制兩者的存取。

您可以選擇性地使用 Web 聯合身分來控制透過 Login with Amazon、Facebook 或 Google 進行身分驗證之使用者的存取。如需詳細資訊，請參閱 [使用 Web 聯合身分](#)。

您可以使用 IAM Condition 元素來實作精細定義存取控制政策。將 Condition 元素新增至許可政策，即可根據特定商業需求來允許或拒絕存取 DynamoDB 資料表和索引中的項目和屬性。

例如，試想有一個能讓玩家選取和玩各種不同遊戲的手機遊戲應用程式。應用程式會使用名為 GameScores 的 DynamoDB 資料表記錄高分和其他使用者資料。資料表中的每個項目都是依使用者 ID 和使用所玩遊戲的名稱進行唯一識別。GameScores 資料表的主索引鍵包含分割區索引鍵 (UserId) 與排序索引鍵 (GameTitle)。使用者只可存取與其使用者 ID 建立關聯的遊戲資料。想要玩遊戲的使用者必須屬於名為 GameRole 且已連接安全政策的 IAM 角色。

若要管理應用程式中的使用者許可，您可以撰寫下列這類許可政策：

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

除了授予 GameScores 資料表 (Action 元素) 上特定 DynamoDB 動作 (Resource 元素) 的許可之外，Condition 元素還會使用 DynamoDB 特有的下列條件金鑰來限制許可，如下所示：

- `dynamodb:LeadingKeys`：此條件金鑰可讓使用者只存取分割區索引鍵值符合其使用者 ID 的項目。此 ID `#{www.amazon.com:user_id}` 是替換變數。如需替換變數的詳細資訊，請參閱 [使用 Web 聯合身分](#)。
- `dynamodb:Attributes`：此條件金鑰會限制對所指定屬性的存取，因此只有許可政策中所列的動作才能傳回這些屬性的值。此外，`StringEqualsIfExists` 子句可以確保應用程式必須一律提供要處理的特定屬性清單，以及確保應用程式無法請求所有屬性。

評估 IAM 政策時，結果一律是 `true` (允許存取) 或 `false` (拒絕存取)。如果 Condition 元素的任一部分是 `false`，則整個政策會評估為 `false`，並拒絕存取。

#### Important

如果您使用 `dynamodb:Attributes`，則必須指定政策中所列資料表和任何次要索引之所有主索引鍵和索引鍵屬性的名稱。否則，DynamoDB 無法使用這些索引鍵屬性來執行所請求的動作。

IAM 政策文件只能包含下列 Unicode 字元：水平定位字元 (U+0009)、換行字元 (U+000A)、歸位字元 (U+000D)，以及 U+0020 到 U+00FF 範圍內的字元。

## 指定條件：使用條件金鑰

AWS 為支援 IAM 進行存取控制的所有 AWS 服務提供一組預先定義的條件金鑰 (AWS 寬度條件金鑰)。例如，您可以使用 `aws:SourceIp` 條件金鑰先檢查申請者的 IP 地址，再允許執行動作。如需詳細資訊和 AWS 寬金鑰清單，請參閱《IAM 使用者指南》中的 [條件可用金鑰](#)。

下表顯示適用於 DynamoDB 的 DynamoDB 服務專屬條件金鑰。

DynamoDB 條件金鑰	描述
dynamodb: LeadingKeys	代表資料表的第一個索引鍵屬性，換言之，即分割區索引鍵。索引鍵名稱 LeadingKeys 是複數，即使索引鍵與單一項目動作一起使用也是一樣。此外，在條件中使用 ForAllValues 時，您必須使用 LeadingKeys 修飾詞。
dynamodb:Select	<p>代表 Query 或 Scan 請求的 Select 參數。Select 可以是任何一個值：</p> <ul style="list-style-type: none"> <li>• ALL_ATTRIBUTES</li> <li>• ALL_PROJECTED_ATTRIBUTES</li> <li>• SPECIFIC_ATTRIBUTES</li> <li>• COUNT</li> </ul>
dynamodb: Attributes	<p>代表請求中的屬性名稱清單，或從請求中傳回的屬性。Attributes 值的指定方式相同，而且意義與特定 DynamoDB API 動作的參數相同，如下所示：</p> <ul style="list-style-type: none"> <li>• AttributesToGet 使用者：BatchGetItem, GetItem, Query, Scan</li> <li>• AttributeUpdates 使用者：UpdateItem</li> <li>• Expected 使用者：DeleteItem, PutItem, UpdateItem</li> <li>• Item 使用者：PutItem</li> <li>• ScanFilter 使用者：Scan</li> </ul>
dynamodb: ReturnValues	代表請求的 ReturnValues 參數。ReturnValues 可以是下列的任何值：

DynamoDB 條件金鑰	描述
	<ul style="list-style-type: none"><li>• ALL_OLD</li><li>• UPDATED_OLD</li><li>• ALL_NEW</li><li>• UPDATED_NEW</li><li>• NONE</li></ul>
dynamodb:ReturnConsumedCapacity	代表請求的 ReturnConsumedCapacity 參數。ReturnConsumedCapacity 可以是下列的一個值： <ul style="list-style-type: none"><li>• TOTAL</li><li>• NONE</li></ul>

## 限制使用者存取

許多 IAM 許可政策可讓使用者只存取資料表中分割區索引鍵值符合使用者識別符的項目。例如，遊戲應用程式先前限制是使用此方式所存取，因此使用者只能存取與其使用者 ID 建立關聯的遊戲資料。IAM 替換變數 `${www.amazon.com:user_id}`、`${graph.facebook.com:id}` 和 `${accounts.google.com:sub}` 包含用於 Login with Amazon、Facebook 及 Google 的使用者識別符。若要了解應用程式如何登入其中一個身分提供者，並取得這些識別符，請參閱 [使用 Web 聯合身分](#)。

### Note

下節中的每個範例都會將 Effect 子句設定為 Allow，而且只指定允許的動作、資源及參數。只允許存取 IAM 政策中明確列出的項目。

在某些情況下，有可能會重新撰寫這些政策，讓它們成為拒絕類型 (即，將 Effect 子句設定為 Deny，並反轉政策中的所有邏輯)。不過，建議您避免在 DynamoDB 中使用拒絕類型政策，因為它們比允許類型政策更難正確地撰寫。此外，DynamoDB API 的未來變更 (或現有 API 輸入的變更) 也可能會導致無效的拒絕類型政策。

## 範例政策：使用條件進行精細定義存取控制

本節顯示數個政策，來實作 DynamoDB 資料表和索引上的精細定義存取控制。

**Note**

所有範例都會使用 us-west-2 區域，並且包含虛構帳戶 ID。

以下影片說明使用 IAM 政策條件在 DynamoDB 中進行精細的存取控制。

1：授予許可，以限制存取具有特定分割區索引鍵值的項目

下列許可政策授予許可，以允許 GamesScore 資料表上的一組 DynamoDB 動作。它會使用 dynamodb:LeadingKeys 條件金鑰，只限制下列項目的使用者動作：其 UserID 主索引鍵值符合此應用程式之 Login with Amazon 唯一使用者 ID 的項目。

**Important**

動作清單不包含 Scan 的許可，因為 Scan 會傳回所有項目，不論前導索引鍵為何。

```
{
 "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

使用政策變數時，您必須在政策中明確指定版本 2012-10-17。存取原則語言的預設版本 2008-10-17 不支援政策變數。

若要實作唯讀存取權，您可以移除任何可修改資料的動作。在下列政策中，條件中只會包含提供唯讀存取權的動作。

```

{
 "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**

如果您使用 `dynamodb:Attributes`，則必須指定政策中所列資料表和任何次要索引之所有主索引鍵和索引鍵屬性的名稱。否則，DynamoDB 無法使用這些索引鍵屬性來執行所請求的動作。

## 2：授予許可，以限制存取資料表中的特定屬性

下列許可政策透過新增 `dynamodb:Attributes` 條件金鑰，來允許只存取資料表中的兩個特定屬性。在條件式寫入或掃描篩選條件中，可以讀取、寫入或評估這些屬性。

```
{
 "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

政策會採用允許清單方式，以允許存取一組指定的屬性。您可以撰寫同等的政策，改為拒絕存取其他屬性。我們不建議使用此拒絕清單方式。使用者可以決定這些受拒絕的屬性，請遵循最低權限政策，如 Wikipedia (網址為 [http://en.wikipedia.org/wiki/Principle\\_of\\_least\\_privilege](http://en.wikipedia.org/wiki/Principle_of_least_privilege)) 中所述，並使用允許清單方式來列舉所有允許值，而不是指定拒絕屬性。

此政策不允許 PutItem、DeleteItem 或 BatchWriteItem。這些動作一律會取代整個先前項目，以允許使用者刪除不允許它們存取之屬性的先前值。

許可政策中的 StringEqualsIfExists 子句確保下列狀況：

- 如果使用者指定 Select 參數，則其值必須是 SPECIFIC\_ATTRIBUTES。此需求防止 API 動作傳回任何不允許的屬性，例如來自索引投影。
- 如果使用者指定 ReturnValues 參數，則其值必須是 NONE、UPDATED\_OLD 或 UPDATED\_NEW。這是必要項目，因為 UpdateItem 動作也會執行隱含讀取操作來檢查項目在取代前是否存在，因此，請求時可以傳回先前的屬性值。以這種方式限制 ReturnValues 可確定使用者只能讀取或寫入允許的屬性。
- StringEqualsIfExists 子句可確保在允許動作內容中，一個請求只能使用其中一個參數：Select 或 ReturnValues。

以下是此政策的某些變化：

- 若只要允許讀取動作，您可以從允許動作清單中移除 UpdateItem。因為其餘動作不接受 ReturnValues 動作，所以您可以從條件中移除 ReturnValues。您也可以將 StringEqualsIfExists 變更為 StringEquals，因為 Select 參數一律會有值 (除非特別指定，否則為 ALL\_ATTRIBUTES)。
- 若只要允許寫入動作，您可以從允許動作清單中移除所有項目，但 UpdateItem 除外。因為 UpdateItem 未使用 Select 參數，所以您可以從條件中移除 Select。您也必須將 StringEqualsIfExists 變更為 StringEquals，因為 ReturnValues 參數一律會有值 (除非特別指定，否則為 NONE)。

- 若要允許名稱符合某模式的所有屬性，請使用 `StringLike`，而非 `StringEquals`，並使用多字元模式相符萬用字元 (\*)。

### 3：授予許可，防止特定屬性的更新

下列許可政策限制只更新 `dynamodb:Attributes` 條件金鑰所識別之特定屬性的使用者存取。`StringNotLike` 條件防止應用程式更新使用 `dynamodb:Attributes` 條件金鑰所指定的屬性。

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

注意下列事項：

- `UpdateItem` 動作 (如同其他寫入動作) 需要項目的讀取存取，以便其傳回更新前後的值。在政策中，您會限制只存取特定屬性的動作，且這些屬性僅限於允許藉由指定 `dynamodb:ReturnValues`

條件金鑰來進行更新者。條件金鑰會限制請求中的 `ReturnValues`，只指定 `NONE`、`UPDATED_OLD` 或 `UPDATED_NEW`，而且未包含 `ALL_OLD` 或 `ALL_NEW`。

- `PutItem` 和 `DeleteItem` 動作會取代整個項目，因此允許應用程式修改任何屬性。因此，限制應用程式只更新特定屬性時，您不應該授予這些 API 的許可。

#### 4：授予許可，只查詢索引中的投影屬性

下列許可政策使用 `dynamodb:Attributes` 條件金鑰，允許對次要索引 (`TopScoreDateTimeIndex`) 進行查詢。政策也會限制只請求已投影到索引之特定屬性的查詢。

若需要應用程式在查詢中指定屬性清單，政策也會指定 `dynamodb:Select` 條件金鑰需要 DynamoDB Query 動作的 `Select` 參數是 `SPECIFIC_ATTRIBUTES`。屬性清單限制為使用 `dynamodb:Attributes` 條件金鑰所提供的特定清單。

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

```
]
}
```

下列許可政策類似，但查詢必須請求所有已投影到索引的屬性。

```
{
 "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：授予許可，以限制存取特定屬性和分割區索引鍵值

下列許可政策允許資料表和資料表索引 (指定於 Action 元素) 的特定 DynamoDB 動作 (指定於 Resource 元素)。此政策使用 `dynamodb:LeadingKeys` 條件金鑰來限制分割區索引鍵值符合使用者 Facebook ID 之項目的許可。

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

#### 注意下列事項：

- 政策 (UpdateItem) 所允許的寫入動作只能修改 attribute-A 或 attribute-B。
- 因為政策允許 UpdateItem，所以應用程式可以插入新項目，而隱藏屬性在新項目中會是 Null。如果這些屬性投影到 TopScoreDateTimeIndex，則政策將有助於防止會致使從資料表擷取的查詢。
- 應用程式無法讀取 dynamodb:Attributes 中所列屬性以外的任何屬性。在此政策下，應用程式必須將讀取請求中的 Select 參數設定為 SPECIFIC\_ATTRIBUTES，而且只能請求列於允許清單上的屬性。對於寫入請求，應用程式無法將 ReturnValues 設定為 ALL\_OLD 或 ALL\_NEW，而且無法根據任何其他屬性來執行條件式寫入操作。

## 相關主題

- [Amazon DynamoDB 的 Identity and Access Management](#)
- [DynamoDB API 許可：動作、資源和條件參考](#)

## 使用 Web 聯合身分

如果您要撰寫以大量使用者為目標的應用程式，也可以選擇使用 web 聯合身分進行身分驗證和授權。Web 聯合身分讓您不再需要建立個別的使用者。相反地，使用者可以登入身分識別提供者，然後從 AWS Security Token Service (AWS STS) 取得臨時安全登入資料。然後，應用程式可以使用這些憑據訪問 AWS 服務。

Web 聯合身分支援下列身分提供者：

- 登入 Amazon
- Facebook
- Google

## Web 聯合身分的其他資源

下列資源可協助您進一步了解 Web 聯合身分：

- AWS 開發人員部落格中的[使用 AWS SDK for .NET 的 Web 聯合身分](#)文章，會介紹如何使用 Facebook 的 Web 聯合身分。它包含 C# 中的程式碼片段，顯示如何採用具有 Web 身分的 IAM 角色，以及如何使用臨時安全登入資料存取 AWS 資源。
- [AWS Mobile SDK for iOS](#) 和 [AWS Mobile SDK for Android](#) 包含範例應用程式。這些應用程式所包含的程式碼示範如何呼叫身分提供者，接著示範如何使用這些供應商中的資訊來取得和使用暫時安全憑證。
- [與行動應用程式的 Web 身分聯盟](#)文章討論 Web 身分聯盟，並顯示如何使用 Web 身分聯合來存取 AWS 資源的範例。

## Web 聯合身分的範例政策

若要顯示如何與 DynamoDB 使用網頁身分同盟，請重新瀏覽中介紹的 GameScores 表格。[使用 IAM 政策條件進行精細定義存取控制](#) 以下是的主鍵 GameScores。

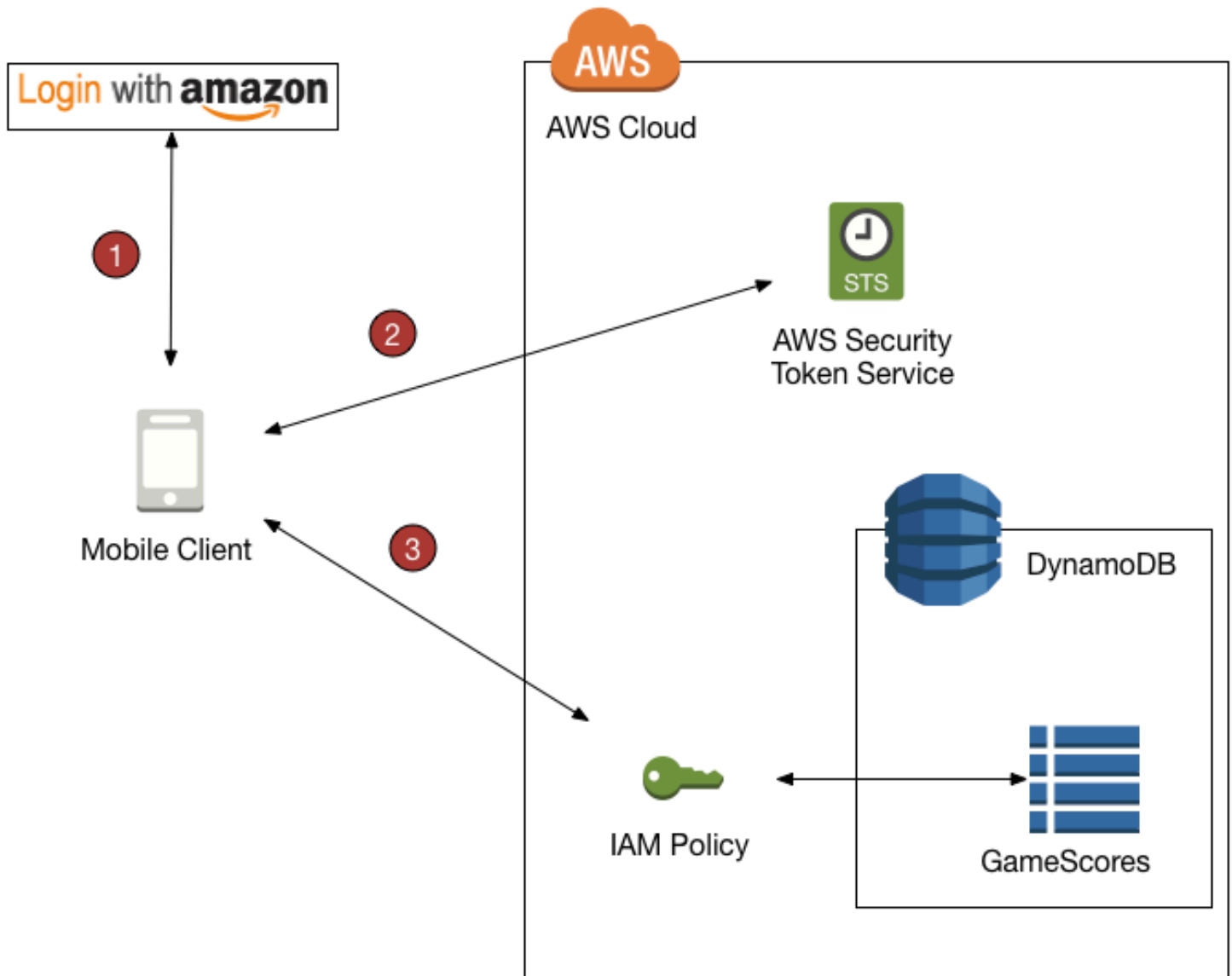
資料表名稱	主索引鍵類型	分割區索引鍵名稱和類型	排序索引鍵名稱和類型
GameScores ( <u>UserId</u> , <u>GameTitle</u> , ...)	複合	屬性名稱： UserId 類型： 字串	屬性名稱： GameTitle 類型： 字串

現在假設手機遊戲應用程式使用此資料表，而且應用程式需要支援數千或甚至數百萬使用者。在這種規模下，管理單個應用程式用戶變得非常困難，並且保證每個用戶只能訪問GameScores表中自己的數據。好消息是，許多使用者都擁有第三方身分提供者 (例如 Facebook、Google 或 Login with Amazon) 的帳戶。因此可以合理地運用其中一個提供者來進行身分驗證任務。

若要使用 Web 聯合身分來執行這個操作，則應用程式開發人員必須向身分提供者 (例如 Login with Amazon) 註冊應用程式，並取得唯一應用程式 ID。接下來，開發人員需要建立 IAM 角色。(在此範例中，此角色的名稱為GameRole。) 該角色必須附加 IAM 政策文件，以指定應用程式可存取GameScores表格的條件。

使用者想要玩遊戲時，會從遊戲應用程式登入其 Login with Amazon 帳戶。然後，該應用程式調用 AWS Security Token Service ( AWS STS )，提供使用 Amazon 應用程式 ID 登錄並請求成員資格GameRole。AWS STS 根據GameRole政策文件，將暫時 AWS 認證傳回應用程式，並允許其存取GameScores表格。

下圖顯示這些部分如何一起使用。



## Web 聯合身分概觀

1. 應用程式會呼叫第三方身分提供者，以對使用者和應用程式進行身分驗證。身分提供者會將 Web 身分字符合傳回給應用程式。
2. 該應用程式調用 AWS STS 並傳遞 Web 身份令牌作為輸入。AWS STS 授權該應用程式並為其提供臨時 AWS 訪問憑據。允許應用程式擔任 IAM 角色 (GameRole)，並根據角色的安全政策存取 AWS 資源。
3. 應用程式會呼叫 DynamoDB 來存取資料表。GameScores 因為它已經假設 GameRole，所以應用程式受到與該角色相關聯的安全性原則的約束。政策文件會防止應用程式存取不屬於使用者的資料。



再次，以下是顯示的GameRole安全策略[使用 IAM 政策條件進行精細定義存取控制](#)：

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

子Condition句會決定應用程式可以看見中GameScores的哪些項目。作法是比較 Login with Amazon ID 與 UserId 中的 GameScores 分割區索引鍵值。只有使用此政策中所列的其中一個 DynamoDB 動作才能處理屬於目前使用者的項目。但無法存取資料表中的其他項目。甚至，只能存取政策中所列的特定屬性。

## 準備使用 Web 聯合身分

如果您是應用程式開發人員，並想要針對應用程式使用 Web 聯合身分，請遵循下列步驟進行：

1. 向第三方身分提供者註冊為開發人員。下列外部連結提供向所支援之身分提供者註冊的資訊：
  - [Login with Amazon 開發人員中心](#)
  - Facebook 網站上的[註冊](#)
  - 在 Google 網站上[使用 OAuth 2.0 存取 Google API](#)
2. 向身分提供者註冊應用程式。當您這麼做時，提供者會將您應用程式的唯一 ID 提供給您。如果您想要應用程式使用多個身分提供者，則需要向每個供應商取得應用程式 ID。
3. 建立一或多個 IAM 角色。每個應用程式的各個身分供應商都需要一個角色。例如，您可能會建立使用者使用 Login with Amazon 登入之應用程式可擔任的角色、使用者使用 Facebook 登入之相同應用程式可擔任的次要角色，以及使用者使用 Google 登入之應用程式可擔任的第三個角色。

在建立角色程序期間，您需要將 IAM 政策連接至此角色。您的政策文件應該定義應用程式所需的 DynamoDB 資源，以及存取這些資源的許可。

如需詳細資訊，請參閱《IAM 使用者指南》中的[關於 Web 聯合身分](#)。

### Note

作為替代方案 AWS Security Token Service，您可以使用 Amazon Cognito。Amazon Cognito 是管理行動應用程式臨時登入資料的偏好服務。如需詳細資訊，請參閱《Amazon Cognito 開發人員指南》中的[取得憑證](#)。

## 使用 DynamoDB 主控台產生 IAM 政策

DynamoDB 主控台可協助您建立 IAM 政策，以與 Web 聯合身分搭配使用。若要執行此操作，您可以選擇 DynamoDB 資料表，並指定要包含在政策中的身分提供者、動作和屬性。DynamoDB 主控台接著會產生可連接至 IAM 角色的政策。

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在導覽窗格中，選擇 Tables (資料表)。
3. 在資料表清單中，選擇想要用於建立 IAM 政策的資料表。
4. 選取動作按鈕，然後選擇建立存取控制政策。
5. 選擇政策的身分提供者、動作和屬性。

完成您要的設定後，選擇產生政策。即會出現產生的政策。

6. 選擇請參閱文件，然後遵循所需的步驟將產生的政策附加到 IAM 角色。

### 撰寫應用程式使用 Web 聯合身分

若要使用 Web 聯合身分，您的應用程式必須假設您建立的 IAM 角色。完成後，應用程式會遵守您連接到角色的存取政策。

在執行時間，如果您的應用程式使用 Web 聯合身分，則必須遵循下列步驟進行：

1. 使用第三方身分提供者進行身分驗證。您的應用程式必須使用身分提供者所提供的界面來呼叫身分提供者。對使用者進行身分驗證的確切方式取決於供應商以及執行應用程式的平台。一般而言，如果使用者尚未登入，則身分供應商會負責顯示該供應商的登入頁面。

身分提供者對使用者進行身分驗證之後，供應商會將 Web 身分字符傳回給應用程式。此字符的格式取決於供應商，但通常是極長的字元字串。

2. 取得臨時 AWS 安全登入資料。若要執行此操作，您的應用程式會將 `AssumeRoleWithWebIdentity` 請求傳送至 AWS Security Token Service (AWS STS)。此請求包含以下項目：

- 先前步驟中的 Web 身分字符
- 來自身分提供者的應用程式 ID
- 針對此應用程式的這個身分提供者所建立之 IAM 角色的 Amazon Resource Name (ARN)

AWS STS 傳回一組在特定時間後過期的 AWS 安全認證 (預設為 3,600 秒)。

以下是 AWS STS 中 `AssumeRoleWithWebIdentity` 動作的範例請求和回應。Web 身分字符取自 Login with Amazon 身分提供者。

```
GET / HTTP/1.1
Host: sts.amazonaws.com
```

```
Content-Type: application/json; charset=utf-8
URL: https://sts.amazonaws.com/?ProviderId=www.amazon.com
&DurationSeconds=900&Action=AssumeRoleWithWebIdentity
&Version=2011-06-15&RoleSessionName=web-identity-federation
&RoleArn=arn:aws:iam::123456789012:role/GameRole
&WebIdentityToken=Atza|IQEBLjAsAhQluyKqyBiYZ8-kclvGTYM81e...(remaining characters
omitted)
```

```
<AssumeRoleWithWebIdentityResponse
 xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
 <AssumeRoleWithWebIdentityResult>
 <SubjectFromWebIdentityToken>amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE</
SubjectFromWebIdentityToken>
 <Credentials>
 <SessionToken>AQoDYXdzEMf//////////wEa8AP6nNDwcSLnf+cHupC...(remaining
characters omitted)</SessionToken>
 <SecretAccessKey>8Jhi60+EWUUbBUSHTesjTxqQtM8UKvsM6XAjdA==</SecretAccessKey>
 <Expiration>2013-10-01T22:14:35Z</Expiration>
 <AccessKeyId>06198791C436IEXAMPLE</AccessKeyId>
 </Credentials>
 <AssumedRoleUser>
 <Arn>arn:aws:sts::123456789012:assumed-role/GameRole/web-identity-federation</
Arn>
 <AssumedRoleId>AR0AJU4SA2VW5SZRF2YMG:web-identity-federation</AssumedRoleId>
 </AssumedRoleUser>
 </AssumeRoleWithWebIdentityResult>
 <ResponseMetadata>
 <RequestId>c265ac8e-2ae4-11e3-8775-6969323a932d</RequestId>
 </ResponseMetadata>
</AssumeRoleWithWebIdentityResponse>
```

### 3. 存取 AWS 資源。AWS STS 的回應包含應用程式存取 DynamoDB 資源所需的資訊：

- AccessKeyId、SecretAccessKey 和 SessionToken 欄位包含只適用於此使用者和此應用程式的安全登入資料。
- Expiration 欄位表示這些登入資料的時間限制，在此時間之後就不再有效。
- AssumedRoleId 欄位包含應用程式已擔任之工作階段特定 IAM 角色的名稱。應用程式會在此工作階段期間遵守 IAM 政策文件中的存取控制。
- SubjectFromWebIdentityToken 欄位包含出現在此特定身分提供者的 IAM 政策變數中的唯一 ID。以下是所支援供應商的 IAM 政策變數，及其範例值：

政策變數	範例值
<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

如需使用這些政策變數的範例 IAM 政策，請參閱 [範例政策：使用條件進行精細定義存取控制](#)。

如需如何 AWS STS 產生臨時存取登入資料的詳細資訊，請參閱 [IAM 使用指南中的要求臨時安全登入資料](#)。

## DynamoDB API 許可：動作、資源和條件參考

當您設定 [Amazon DynamoDB 的 Identity and Access Management](#) 並撰寫可連接至 IAM 身分 (身分類型政策) 的許可政策時，可以參考《IAM 使用者指南》中的 [Amazon DynamoDB 的動作、資源和條件金鑰](#)。此頁面列出每個 DynamoDB API 作業、您可以授與執行動作權限的對應動作，以及您可以授與權限的 AWS 資源。您在政策的 Action 欄位中指定動作，然後在政策的 Resource 欄位中指定資源值。

您可以在 DynamoDB 政策中使用 AWS 寬條件金鑰來表示條件。如需 AWS 全金鑰的完整清單，請參閱 [IAM 使用者指南中的 IAM JSON 政策元素參考資料](#)。

除了廣泛的條件 AWS 鍵之外，DynamoDB 還有其自己的特定金鑰，您可以在條件中使用。如需詳細資訊，請參閱 [使用 IAM 政策條件進行精細定義存取控制](#)。

### 相關主題

- [Amazon DynamoDB 的 Identity and Access Management](#)
- [使用 IAM 政策條件進行精細定義存取控制](#)

## DynamoDB Accelerator 中的身分和存取管理

DynamoDB Accelerator (DAX) 可與 DynamoDB 一起使用，為您的應用程式無縫新增快取層。但是，DAX 和 DynamoDB 具有不同的存取控制機制。這兩種服務都使用 AWS Identity and Access Management (IAM) 來實作各自的安全政策，但 DAX 和 DynamoDB 的安全模型不同。

如需有關 DAX 中 Identity and Access Management 的詳細資訊，請參閱 [DAX 存取控制](#)。

## DynamoDB 的業界合規驗證

若要瞭解 AWS 服務 是否屬於特定規範遵循方案的範圍內，請參閱[AWS 服務 遵循規範計劃](#)方案中的，並選擇您感興趣的合規方案。如需一般資訊，請參閱[AWS 規範計劃](#)。

您可以使用下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱[下載中的報告中的](#) AWS Artifact。

您在使用時的合規責任取決 AWS 服務 於資料的敏感性、公司的合規目標以及適用的法律和法規。AWS 提供下列資源以協助遵循法規：

- [安全性與合規性快速入門指南](#) — 這些部署指南討論架構考量，並提供部署以安全性和合規性 AWS 為重點的基準環境的步驟。
- [在 Amazon Web Services 上架構 HIPAA 安全性與合規性](#) — 本白皮書說明公司如何使用建立符合 HIPAA 資格的應 AWS 應用程式。

### Note

並非所有人 AWS 服務 都符合 HIPAA 資格。如需詳細資訊，請參閱 [HIPAA 資格服務參照](#)。

- [AWS 合規資源](#) — 此工作簿和指南集合可能適用於您的產業和所在地。
- [AWS 客戶合規指南](#) — 透過合規的角度瞭解共同的責任模式。這份指南總結了在多個架構 (包括美國國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO)) 中，保 AWS 服務 護指引並對應至安全控制的最佳實務。
- [使用 AWS Config 開發人員指南中的規則評估資源](#) — 此 AWS Config 服務會評估您的資源組態符合內部實務、產業準則和法規的程度。
- [AWS Security Hub](#) — 這 AWS 服務 提供了內部安全狀態的全面視圖 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱 [Security Hub controls reference](#)。
- [Amazon GuardDuty](#) — 透過監控環境中的 AWS 帳戶可疑和惡意活動，藉此 AWS 服務 偵測您的工作負載、容器和資料的潛在威脅。GuardDuty 可協助您滿足特定合規性架構所要求的入侵偵測需求，如 PCI DSS 等各種合規性需求。
- [AWS Audit Manager](#) — 這 AWS 服務 有助於您持續稽核您的 AWS 使用情況，以簡化管理風險以及遵守法規和業界標準的方式。

# Amazon DynamoDB 中的資料復原及災難復原功能

AWS 全球基礎設施是以 AWS 區域與可用區域為中心建置的。AWS 區域提供多個分開且隔離的實際可用區域，它們以低延遲、高輸送量和高度備援聯網功能相互連結。透過可用區域，您所設計與操作的應用程式和資料庫，就能夠在可用區域之間自動容錯移轉，而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如果您需要在更大的地理距離內複製資料或應用程式，請使用 AWS 本地區域。AWS 本地區域是旨在補足現有 AWS 區域的單一資料中心。如同所有 AWS 區域，AWS 本地區域會與其他 AWS 區域完全隔離。

如需 AWS 區域與可用區域的詳細資訊，請參閱 [AWS 全球基礎設施](#)。

除了 AWS 全球基礎設施外，Amazon DynamoDB 還提供數種支援資料復原和備份需求的功能。

## 隨需備份與還原

DynamoDB 提供隨需備份功能。它可讓您建立資料表的完整備份來長期保留與封存。如需詳細資訊，請參閱 [DynamoDB 的隨需備份與還原](#)。

## 時間點復原

時間點復原有助於保護您的 DynamoDB 資料表免遭意外寫入或刪除操作。有了時間點復原，就無需為建立、維護或排程隨需備份而煩惱。如需詳細資訊，請參閱 [DynamoDB 的時間點復原](#)。

## 跨 AWS 區域同步的全域資料表

DynamoDB 會自動將資料表的資料與傳輸流分散到足夠數量的伺服器上，以處理您的輸送量和儲存需求，同時保持快速且一致的效能。您的所有資料皆儲存於固態硬碟 (SSD) 上，並會自動複製到 AWS 區域中的多個可用區域，以提供內建的高可用性和資料耐久性。您可以使用全域資料，表讓 DynamoDB 資料表在 AWS 區域中保持同步。

# Amazon DynamoDB 中的基礎設施安全性

作為受管服務，Amazon DynamoDB 受到 AWS 全球網路安全程序的保護，如 AWS Well-Architected 的 [架構中的基礎設施保護](#) 中所述。

您可以使用 AWS 已發佈的 API 呼叫透過網路存取 DynamoDB。用戶端可以使用 TLS (傳輸層安全性) 1.2 或 1.3 版。用戶端還必須支援具備完全正向加密 (PFS) 功能的密碼套件，例如臨時 Diffie-Hellman (DHE) 或橢圓曲線臨時 Diffie-Hellman (ECDHE)。現代系統(如 Java 7 和更新版本)大多會支援這些模



式。此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。或者，您可以使用 [AWS Security Token Service](#) (AWS STS) 來產生暫時安全登入資料來簽署請求。

您也可以使用 DynamoDB 的 Virtual Private Cloud (VPC) 端點，讓 VPC 中的 Amazon EC2 執行個體能夠使用其私有 IP 地址來存取 DynamoDB，卻不會暴露於公有網際網路。如需更多詳細資訊，請參閱 [使用 Amazon VPC 端點來存取 DynamoDB](#)。

## 使用 Amazon VPC 端點來存取 DynamoDB

基於安全考量，許多 AWS 客戶會在 Amazon Virtual Private Cloud (Amazon VPC) 環境中執行其應用程式。運用 Amazon VPC，您可以將 Amazon EC2 執行個體啟動到 Virtual Private Cloud 上，此 Virtual Private Cloud 與其他網路 (包括公有網際網路) 在邏輯上隔離。使用 Amazon VPC，您可以控制其 IP 地址範圍、子網路、路由表、網路閘道及安全設定。

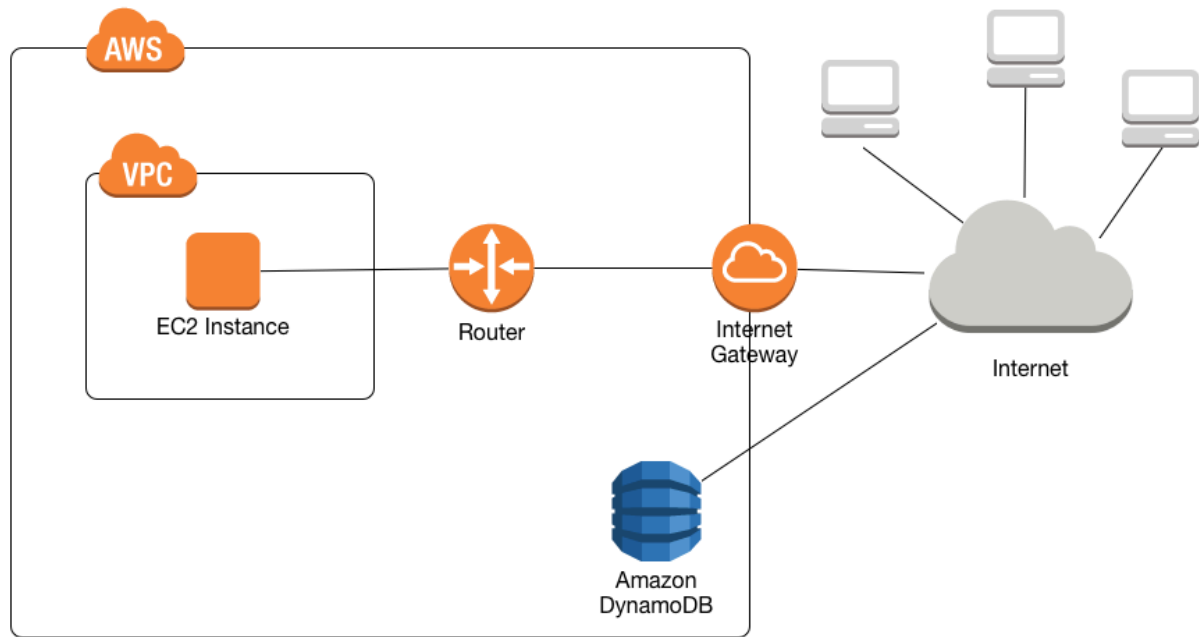
### Note

如果您是在 2013 年 12 月 4 日 AWS 帳戶之後建立的，則每個 AWS 區域 VPC 都已經有預設的 VPC。預設的 VPC 已可供使用：您可以立即開始使用，不需執行任何其他設定步驟。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 [預設 VPC 和預設子網路](#)。

若要存取公有網際網路，您的 VPC 必須擁有網際網路閘道，也即將 VPC 連線到網際網路的虛擬路由器。這可讓在 VPC 中的應用程式在 Amazon EC2 上執行，以便存取 Amazon DynamoDB 等網際網路資源。

根據預設，與 DynamoDB 之間的通訊會使用 HTTPS 協定，其使用 SSL/TLS 加密保護網路流量。下圖顯示 VPC 中的 Amazon EC2 執行個體存取 DynamoDB，方法是讓 DynamoDB 使用網際網路閘道而非 VPC 端點。





許多客戶對於透過公有的網際網路來傳送和接收資料，都會有合法的隱私與安全性疑慮。您可以使用虛擬私有網路 (VPN)，來轉傳通過您自己企業網路基礎設施的所有 DynamoDB 網路傳輸資料，以解決前述的問題。但是這個方法可能會帶來頻寬和可用性的挑戰。

DynamoDB 的 VPC 端點可以減輕這些挑戰的阻礙。DynamoDB 的 VPC 端點可讓 VPC 中的 Amazon EC2 執行個體使用私有 IP 地址來存取 DynamoDB，卻不需暴露於公有網際網路。您的 EC2 執行個體不需公有 IP 地址，您也不需要網際網路閘道、NAT 裝置或 VPC 中的虛擬私有閘道。您可以使用端點政策來控制對 DynamoDB 的存取。您 VPC 與 AWS 服務之間的流量都會保持在 Amazon 網路的範圍內。

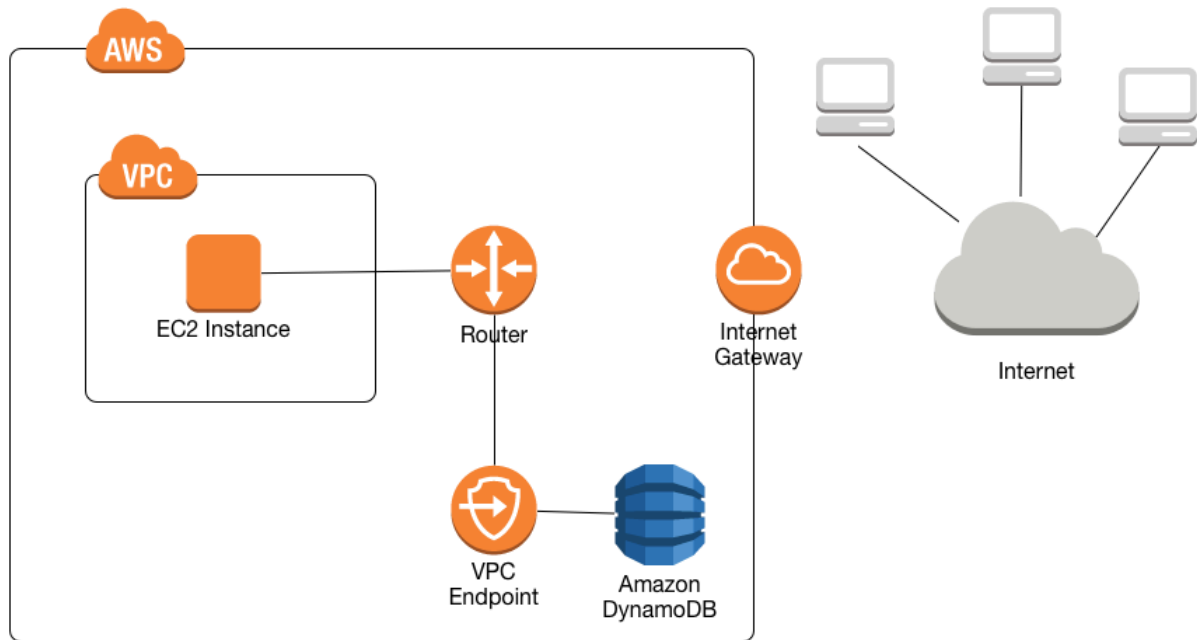
#### Note

即使您使用公有 IP 位址，執行個體和託管服務之間的所有 VPC 通訊都會在 AWS 網路中保持私有狀態。來自 AWS 網路且具有 AWS 網路目的地的封包會保留在 AWS 全球網路上，但往返 AWS 中國區域的流量除外。

當您建立 DynamoDB 使用的 VPC 端點時，區域 (例如 `dynamodb.us-west-2.amazonaws.com`) 內傳送到 DynamoDB 端點的所有要求，都會轉傳到 Amazon 網路中的私有 DynamoDB 端點。您不需要修改

VPC 中在 EC2 執行個體上執行的應用程式。端點的名稱會保持不變，但 DynamoDB 的路由會完全在 Amazon 網路中，不會存取公有網際網路。

下圖顯示 VPC 中的 EC2 執行個體如何使用 VPC 端點來存取 DynamoDB。



如需詳細資訊，請參閱 [the section called “教學課程：針對 DynamoDB 使用 VPC 端點”](#)。

## 共用 Amazon VPC 端點和 DynamoDB

若要允許透過 VPC 子網路的閘道端點存取 DynamoDB 服務，您必須擁有該 VPC 子網路的擁有者帳戶許可。

一旦 VPC 子網路的閘道端點取得 DynamoDB 存取權，任何具有該子網路存取權的 AWS 帳戶都可以使用 DynamoDB。這表示 VPC 子網路中的所有帳戶使用者，都可以使用他們擁有存取權的任何 DynamoDB 資料表。這包括與 VPC 子網路以外不同帳戶關聯的 DynamoDB 資料表。VPC 子網路擁有者仍可自行決定限制子網路中任何特定使用者透過閘道端點使用 DynamoDB 服務。

## 教學課程：針對 DynamoDB 使用 VPC 端點

此節會逐步說明如何設定並使用 DynamoDB 的 VPC 端點。

### 主題

- [步驟 1：啟動 Amazon EC2 執行個體](#)
- [步驟 2：設定您的 Amazon EC2 執行個體](#)
- [步驟 3：建立 DynamoDB 的 VPC 端點](#)
- [步驟 4：\(選用\) 清除](#)

### 步驟 1：啟動 Amazon EC2 執行個體

在此步驟中，您會在預設的 Amazon VPC 中啟動 Amazon EC2 執行個體。您可以接著為 DynamoDB 建立並使用 VPC 端點。

1. 在 <https://console.aws.amazon.com/ec2/> 開啟 Amazon EC2 主控台。
2. 選擇 Launch Instance (啟動執行個體) 並執行下列作業：

#### 步驟 1：選擇 Amazon Machine Image (AMI)

- 在 AMI 清單的第一項上，前往 Amazon Linux AMI 並選擇 Select (選取)。

#### 步驟 2：選擇執行個體類型

- 在執行個體類型清單的第一項上，選擇 t2.micro。
- 選擇 Next: Configure Instance Details (下一步：設定執行個體詳細資訊)。

#### 步驟 3：設定執行個體詳細資訊

- 前往 Network (網路)，然後選擇您的預設 VPC。

選擇 Next: Add Storage (下一步：新增儲存體)。

#### 步驟 4：新增儲存體

- 選擇 Next: Tag Instance (下一步：為執行個體新增標籤) 以跳過此步驟。

#### 步驟 5：將執行個體加上標籤

- 選擇 Next: Configure Security Group (下一步：設定安全群組) 以跳過此步驟。

#### 步驟 6：設定安全群組

- 選擇 Select an existing security group (選取現有的安全群組)。
- 在安全群組清單中，選擇 default (預設)。這是您 VPC 的預設安全群組。
- 選擇 Next: Review and Launch (下一步：檢閱和啟動)。

### 步驟 7：檢閱執行個體啟動

- 選擇啟動。
3. 在 Select an existing key pair or create a new key pair (選取現有的金鑰對或建立新的金鑰對) 視窗中，執行下列其中一項作業：
    - 如果您沒有 Amazon EC2 金鑰對，請選擇 Create a new key pair (建立新的金鑰對) 並依照指示進行。系統會要求您下載私有金鑰檔案 (.pem 檔案)，您之後登入 Amazon EC2 執行個體時會需要此檔案。
    - 若您已擁有 Amazon EC2 金鑰對，請前往 Select a key pair (選取金鑰對)，然後從清單中選擇您的金鑰對。您必須已具備可用的私有金鑰檔案 (.pem 檔案) 才能登入您的 Amazon EC2 執行個體。
  4. 當您設定好金鑰對後，請選擇 Launch Instances (啟動執行個體)。
  5. 返回 Amazon EC2 主控台首頁，然後選擇您啟動的執行個體。在下方窗格的 Description (描述) 標籤上，找到您執行個體的 Public DNS (公有 DNS)。例如：`ec2-00-00-00-00.us-east-1.compute.amazonaws.com`。

請記下此公有 DNS 名稱，因為在此教學課程 ([步驟 2：設定您的 Amazon EC2 執行個體](#)) 的下一個步驟中您會需要此名稱。

#### Note

Amazon EC2 執行個體會需要幾分鐘的時間才會變成可用。在您繼續進行下一個步驟之前，請先確定 Instance State (執行個體狀態) 為 `running`，並已通過其所有 Status Checks (狀態檢查)。

### 步驟 2：設定您的 Amazon EC2 執行個體

在 Amazon EC2 執行個體可用時，您將能夠登入該執行個體並準備第一次使用。

**Note**

下列步驟假設您從執行 Linux 的電腦連線至 Amazon EC2 執行個體。如需其他連線方式，請參閱《Linux 執行個體適用的 Amazon EC2 使用者指南》中的[連線至您的 Linux 執行個體](#)。

1. 您需要允許 SSH 流量傳入 Amazon EC2 執行個體。若要執行此動作，您需要建立新的 EC2 安全群組，然後將安全群組指派給 EC2 執行個體。
  - a. 在導覽窗格中，選擇 Security Groups (安全群組)。
  - b. 選擇 Create Security Group (建立安全群組)。在 Create Security Group (建立安全群組) 視窗中，執行下列動作：
    - Security group name (安全群組名稱)：為您的安全群組輸入名稱。例如：my-ssh-access
    - Description (描述)：為安全群組輸入簡短描述。
    - VPC：請選擇預設 VPC。
    - 在 Security group rules (安全群組規則) 區段中，選擇 Add Rule (新增規則)，並執行下列動作：
      - Type (類型)：選擇 SSH。
      - Source (來源)：選擇 My IP (我的 IP)。

當您滿意設定後，請選擇 Create (建立)。
  - c. 在導覽窗格中，選擇執行個體。
  - d. 選擇您在 [步驟 1：啟動 Amazon EC2 執行個體](#) 中啟動的 Amazon EC2 執行個體。
  - e. 選擇 Actions (動作) --> Networking (聯網) --> Change Security Groups (變更安全群組)。
  - f. 在 Change Security Groups (變更安全群組) 下，選取您在前述步驟中建立的安全群組 (例如：my-ssh-access)。此外，還應選擇現有的 default 安全群組。完成設定後，請選擇 Assign Security Groups (指派安全群組)。
2. 使用 ssh 命令登入您的 Amazon EC2 執行個體，如下列範例所示。

```
ssh -i my-keypair.pem ec2-user@public-dns-name
```

您會需要指定您的私有金鑰檔案 (.pem 檔案) 和您執行個體的公有 DNS 名稱。(請參閱 [步驟 1：啟動 Amazon EC2 執行個體](#))。

登入 ID 為 `ec2-user`。不需要任何密碼。

3. 設定您的 AWS 登入資料，如下所示。根據提示輸入您的 AWS 存取金鑰 ID、秘密金鑰和預設區域名稱。

```
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]:
```

您現在可以開始建立 DynamoDB 的 VPC 端點。

### 步驟 3：建立 DynamoDB 的 VPC 端點

在此步驟中，您要建立 DynamoDB 的 VPC 端點並對其進行測試，確保其可以正常運作。

1. 在開始之前，請確認您可以使用 DynamoDB 的公有端點來與 DynamoDB 通訊。

```
aws dynamodb list-tables
```

輸出會顯示您目前擁有的 DynamoDB 資料表清單。(如果沒有任何資料表，清單將是空白。)

2. 請確認 DynamoDB 是可用的服務，可在目前 AWS 區域中建立 VPC 端點。(命令會以粗體文字顯示，後面接著輸出範例。)

```
aws ec2 describe-vpc-endpoint-services
```

```
{
 "ServiceNames": [
 "com.amazonaws.us-east-1.s3",
 "com.amazonaws.us-east-1.dynamodb"
]
}
```

在範例輸出中，DynamoDB 是其中一個可用的服務，因此您可以繼續為其建立 VPC 端點。

3. 確定您的 VPC 識別碼。

```
aws ec2 describe-vpcs
```

```
{
 "Vpcs": [
 {
 "VpcId": "vpc-0bbc736e",
 "InstanceTenancy": "default",
 "State": "available",
 "DhcpOptionsId": "dopt-8454b7e1",
 "CidrBlock": "172.31.0.0/16",
 "IsDefault": true
 }
]
}
```

在範例輸出中，VPC ID 為 vpc-0bbc736e。

4. 建立 VPC 端點。對於 --vpc-id 參數，請指定上一個步驟的 VPC ID。使用 --route-table-ids 參數，將端點與路由表相關聯。

```
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. 確認您可以透過 VPC 端點存取 DynamoDB。

```
aws dynamodb list-tables
```

如果需要，您可以嘗試其他 DynamoDB AWS CLI 命令。如需詳細資訊，請參閱 [《AWS CLI 命令參考》](#)。

#### 步驟 4：(選用) 清除

如果您要刪除自己在此教學課程中建立的資源，請按下列步驟執行：

若要移除 DynamoDB 的 VPC 端點

1. 登入 Amazon EC2 執行個體。
2. 決定 VPC 端點 ID。

```
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"
 }
}
```

在範例輸出中，VPC 端點 ID 為 vpce-9b15e2f2。

3. 刪除 VPC 端點。

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2
```

```
{
 "Unsuccessful": []
}
```

空白陣列 [] 表示成功 (沒有未成功的請求)。



## 若要終止 Amazon EC2 執行個體

1. 在 <https://console.aws.amazon.com/ec2/> 開啟 Amazon EC2 主控台。
2. 在導覽窗格中，選擇執行個體。
3. 選擇 Amazon EC2 執行個體。
4. 依序選擇 Actions (動作)、Instance State (執行個體狀態) 和 Terminate (終止)。
5. 在確認視窗中，請選擇 Yes, Terminate (是，終止)。

## AWS PrivateLink 適用於動態支援

使 AWS PrivateLink 用 DynamoDB，您可以在虛擬私有雲端 (Amazon VPC) 中佈建介面 Amazon VPC 端點 (介面端點)。這些端點可透過 VPN 直接從內部部署的應用程式存取 AWS Direct Connect，或透過 AWS 區域過 [Amazon VPC](#) 對等互連的不同應用程式存取。使用 AWS PrivateLink 和介面端點，您可以簡化從應用程式到 DynamoDB 的私有網路連線。

VPC 中的應用程式不需要公用 IP 位址即可與 DynamoDB 介面虛擬私人雲端端點進行通訊，以進行 DynamoDB 作業。介面端點由一個或多個彈性網路界面 (ENI) 表示，這些界面是從 Amazon VPC 中的子網路指派私有 IP 地址。透過介面端點向 DynamoDB 發出的請求會保留在 Amazon 網路上。您也可以透過 AWS Direct Connect 或 AWS Virtual Private Network (AWS VPN) 從現場部署應用程式存取 Amazon VPC 中的介面端點。有關如何將 Amazon VPC 與現場部署網路連接的詳細資訊，請參閱 [AWS Direct Connect 使用者指南](#) 和使用 [AWS Site-to-Site VPN 者指南](#)。

如需有關介面端點的一般資訊，請參閱 AWS PrivateLink 指南中的 [介面 Amazon VPC 端點 \(AWS PrivateLink\)](#)。

### 主題

- [Amazon VPC 端點的類 Amazon DynamoDB](#)
- [使用 Amazon DynamoDB 時 AWS PrivateLink 的注意事項](#)
- [建立 Amazon VPC 端點](#)
- [存取 Amazon DynamoDB 介面端點](#)
- [從 DynamoDB 介面端點存取表格並控制 API 作業](#)
- [更新內部部署 DNS 組態](#)
- [為 DynamoDB 建立 Amazon VPC 私人雲端端點政策](#)

## Amazon VPC 端點的類 Amazon DynamoDB

您可以使用兩種類型的 Amazon VPC 端點來存取 Amazon DynamoDB：閘道端點和介面端點 (使用)。AWS PrivateLink 閘道端點是您在路由表中指定的閘道，可透過網路從 Amazon VPC 存取 DynamoDB。AWS 介面端點可使用私有 IP 地址，透過使用 Amazon VPC 對等或從 Amazon VPC 內部、內部部署或從另一個 Amazon VPC 中的 Amazon VPC 路由請求路 AWS 區域 由到 DynamoDB，以擴展閘道端點的功能。AWS Transit Gateway 如需詳細資訊，請參閱 [什麼是 Amazon VPC 對等互連？](#) 和 [Transit Gateway 與 Amazon VPC 對等互連](#)。

介面端點與閘道端點相容。如果 Amazon VPC 中有現有的閘道端點，則可以在同一個 Amazon VPC 中使用這兩種類型的端點。

閘道 DynamoDB 點	DynamoDB 支援的介面端點
在這兩種情況下，您的網路流量都會保留在 AWS 網路上。	
使用 Amazon DynamoDB 公有 IP 地址	使用 Amazon VPC 中的私有 IP 地址存取 Amazon DynamoDB
不允許從內部部署存取	允許從內部部署存取
不允許從其他人存取 AWS 區域	AWS 區域 透過使用 Amazon VPC 對等互連，允許從另一個端點的 Amazon VPC 端點進行存取，或 AWS Transit Gateway
不計費	計費

如需有關閘道端點的詳細資訊，請參閱 AWS PrivateLink 指南中的 [閘道 Amazon VPC 端點](#)。

### 使用 Amazon DynamoDB 時 AWS PrivateLink 的注意事項

Amazon VPC 考量適用 AWS PrivateLink 於 Amazon DynamoDB。如需詳細資訊，請參閱《AWS PrivateLink 指南》中的 [介面端點考量事項](#) 和 [AWS PrivateLink 配額](#)。此外，適用下列限制。

AWS PrivateLink Amazon DynamoDB 不支援下列項目：

- [聯邦資訊處理標準 \(FIPS\) 端點](#)
- Transport Layer Security (TLS) 1.1
- 私人和混合網域名稱系統 (DNS) 服務

AWS PrivateLink 目前不支援 Amazon DynamoDB Streams 端點。

您每秒最多可以為啟用的每個 AWS PrivateLink 端點提交 50,000 個請求。

### Note

AWS PrivateLink 端點的網路連線逾時不在 DynamoDB 錯誤回應的範圍內，因此必須由連線到端點的應用程式正確處理。 PrivateLink

## 建立 Amazon VPC 端點

若要建立 Amazon 虛擬私人雲端介面端點，請參閱指[AWS PrivateLink 南中的建立 Amazon VPC 端點](#)。

## 存取 Amazon DynamoDB 介面端點

當您建立介面端點時，DynamoDB 會產生兩種特定端點的 DynamoDB DNS 名稱：區域和區域名稱。

- 區域 DNS 名稱包括唯一的 Amazon VPC 端點識別碼、服務識別碼 AWS 區域、和名稱 `vpce.amazonaws.com` 中的。例如，對於 Amazon VPC 端點識別碼 `vpce-1a2b3c4d`，產生的 DNS 名稱可能 `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` 與。
- 地區 DNS 名稱包含可用區域，例如 `vpce-1a2b3c4d-5e6f-us-east-1a.dynamodb.us-east-1.vpce.amazonaws.com`。如果您的架構可隔離可用區域，則可以使用此選項。例如，您可以將其用於故障遏止或降低區域資料傳輸成本。

您可以從 DynamoDB 公用 DNS 網域解析端點特定的 DNS 名稱。

## 從 DynamoDB 介面端點存取表格並控制 API 作業

您可以使用 AWS CLI 或 AWS 開發套件存取 DynamoDB 資料表，並透過 DynamoDB 介面端點控制 API 作業。

### AWS CLI 例子

若要透過 DynamoDB 介面端點在 AWS CLI 命令中存取 DynamoDB 表格或 DynamoDB 控制 API 作業，請使用和參數。 `--region --endpoint-url`

範例：建立 VPC 端點

```
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
```

### 範例：修改 VPC 端點

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
any additional parameters needed, see Privatelink documentation for more details
```

### 範例：使用端點 URL 列出資料表

在下列範例中，將 VPC 端點識別碼的區域 `us-east-1` 和 DNS 名稱取代為您自己 `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` 的資訊。

```
aws dynamodb --region us-east-1 --endpoint https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com list-tables
```

## AWS SDK 範例

若要在使用開發套件時透過 DynamoDB 介面端點存取 DynamoDB 表格或 DynamoDB 控制 API 作業，請將您的 AWS 開發套件更新為最新版本。然後，將您的用戶端設定為使用端點 URL 透過 DynamoDB 介面端點存取表格或 DynamoDB 控制項 API 操作。

### SDK for Python (Boto3)

#### 範例：使用端點 URL 存取 DynamoDB 表格

在下列範例中，將區域 `us-east-1` 和 VPC 端點 ID `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` 取代為您的資訊。

```
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

範例：使用端點 URL 存取 DynamoDB 表格

在下列範例中，將區域 `us-east-1` 和 VPC 端點 ID `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` 取代為您的資訊。

```
//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

範例：使用端點 URL 存取 S3 儲存貯體

在下列範例中，以您自己的資訊取代區域 `us-east-1` 和 VPC 人雲端端點識別碼 `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com`。

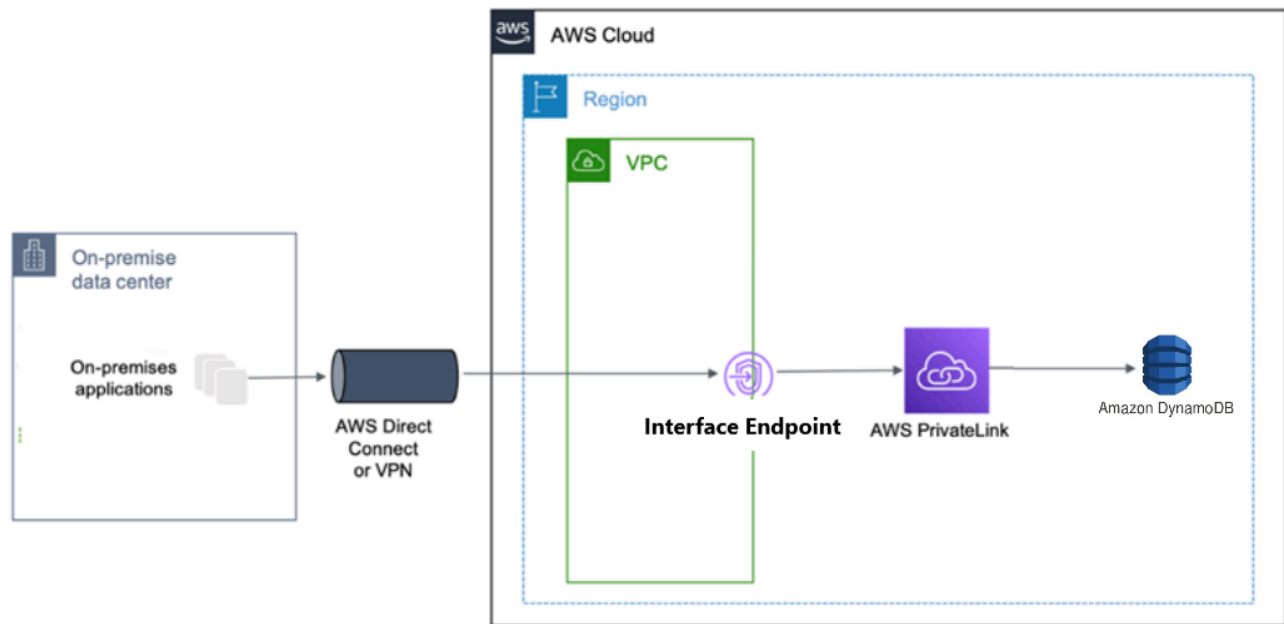
```
Region region = Region.US_EAST_1;
dynamoDbClient = DynamoDbClient.builder().region(region)
 .endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com"))
 .build();
```

## 更新內部部署 DNS 組態

使用端點特定的 DNS 名稱存取 DynamoDB 的介面端點時，您不需要更新內部部署 DNS 解析器。您可以從公用 DynamoDB DNS 網域使用介面端點的私人 IP 位址來解析端點特定的 DNS 名稱。

使用介面端點存取 DynamoDB，而不需要 Amazon VPC 中的閘道端點或網際網路閘道

Amazon VPC 中的介面端點可以透過 Amazon 網路將 Amazon 內 VPC 人雲端應用程式和現場部署應用程式路由到 DynamoDB，如下圖所示。

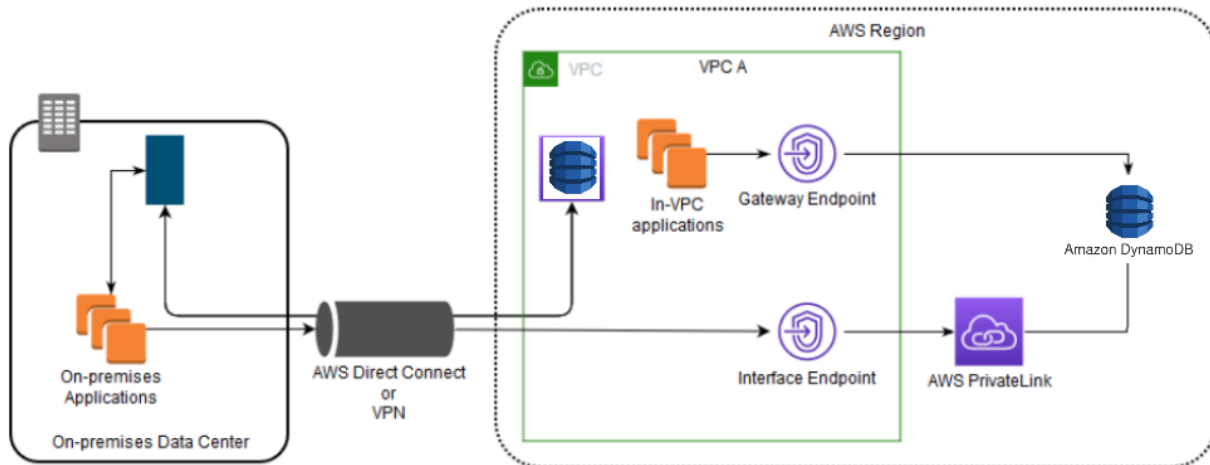


此圖展示了以下要點：

- 您的現場部署網路使 AWS VPN 用 AWS Direct Connect 或連接到 Amazon VPC A。
- 您的現場部署和 Amazon VPC A 中的應用程式會使用端點特定的 DNS 名稱，透過 DynamoDB 介面端點存取 DynamoDB。
- 現場部署應用程式透過 AWS Direct Connect (或 AWS VPN) 將資料傳送到 Amazon VPC 中的介面端點。AWS PrivateLink 透過網路將資料從介面端點移至 DynamoDB。AWS
- Amazon 中的 VPC 應用程式也會將流量傳送到介面端點。AWS PrivateLink 透過網路將資料從介面端點移至 DynamoDB。AWS

## 使用閘道端點並將端點連接在同一個 Amazon VPC 中，以存取 DynamoDB

您可以建立介面端點，並將現有閘道端點保留在相同的 Amazon VPC 中，如下圖所示。透過採用此方法，您可以允許 Amazon 內 VPC 人雲端應用程式繼續透過閘道端點存取 DynamoDB，但不會計費。然後，只有您的內部部署應用程式會使用介面端點來存取 DynamoDB。若要以這種方式存取 DynamoDB，您必須更新內部部署應用程式，以使用 DynamoDB 的端點特定 DNS 名稱。



此圖展示了以下要點：

- 現場部署應用程式使用端點特定的 DNS 名稱，透過 AWS Direct Connect (或) 將資料傳送到 Amazon VPC 內的介面端點。AWS VPN AWS PrivateLink 透過網路將資料從介面端點移至 DynamoDB。AWS
- Amazon 內 VPC 應用程式會使用預設的區域 DynamoDB 名稱，將資料傳送到透過網路連線到 DynamoDB 的閘道端點。AWS

如需有關閘道端點的詳細資訊，請參閱 [Amazon VPC 使用者指南中的閘道 Amazon VPC 端點](#)。

## 為 DynamoDB 建立 Amazon VPC 私人雲端端點政策

您可以將端點政策附加到 Amazon VPC 端點，以控制對 DynamoDB 的存取。此政策會指定下列資訊：

- 可以執行動作的 AWS Identity and Access Management (IAM) 主體
- 可執行的動作
- 可在其中執行動作的資源

### 主題

- [範例：限制從 Amazon VPC 端點存取特定資料表](#)

## 範例：限制從 Amazon VPC 端點存取特定資料表

您可以建立端點策略，限制只能存取特定 DynamoDB 表格。如果您的 Amazon VPC AWS 服務 中有其他使用表格的政策，則此類型的政策非常有用。下表原則限制只能存取 `DOC-EXAMPLE-TABLE` 若要使用此端點策略，請 `DOC-EXAMPLE-TABLE` 以資料表的名稱取代。

```
{
 "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/*"]
 }
]
}
```

## Amazon DynamoDB 中的組態與漏洞分析

AWS 會處理訪客作業系統 (OS) 和資料庫修補、防火牆組態和災難復原等基本安全任務。這些程序已由適當的第三方進行檢閱並認證。如需詳細資訊，請參閱以下 資源：

- [Amazon DynamoDB 的合規驗證](#)
- [共同的責任模型](#)
- [Amazon Web Services：安全程序概觀](#) (白皮書)

以下安全最佳實務也可處理 Amazon DynamoDB 中的組態和漏洞分析：

- [使用 AWS Config 規則 監控 DynamoDB 的合規性](#)
- [使用 AWS Config 監控 DynamoDB 組態](#)



# Amazon DynamoDB 的安全最佳實務

在您開發和實作自己的安全政策時，可考慮使用 Amazon DynamoDB 提供的多種安全功能。以下最佳實務為一般準則，並不代表完整的安全解決方案。這些最佳實務可能不適用或無法滿足您的環境需求，因此請將其視為實用建議就好，而不要當作是指示。

## 主題

- [DynamoDB 預防性安全最佳實務](#)
- [DynamoDB 偵測性安全最佳實務](#)

## DynamoDB 預防性安全最佳實務

下列最佳實務有助於預測並預防 Amazon DynamoDB 中的安全性事件。

### 靜態加密

DynamoDB 使用儲存在 [AWS Key Management Service \(AWS KMS\)](#) 中的加密金鑰，為資料表、索引、串流和備份中的所有使用者資料進行靜態加密。如此可透過保護您的資料免於發生未經授權的基礎儲存體存取，為資料提供另一層保護。

您可以指定 DynamoDB 是否應使用 AWS 擁有的金鑰 (預設加密類型) AWS 受管金鑰、或客戶受管金鑰來加密使用者資料。如需詳細資訊，請參閱 [Amazon DynamoDB 靜態加密](#)。

### 使用 IAM 角色驗證對 DynamoDB 的存取權

若要讓使用者、應用程式和其他 AWS 服務存取 DynamoDB，他們必須在其 AWS API 請求中包含有效的 AWS 認證。您不應將 AWS 登入資料直接存放在應用程式或 EC2 執行個體中。這些是不會自動輪換的長期登入資料，因此如果遭到盜用，可能會對業務造成嚴重的影響。IAM 角色可讓您取得可用來存取 AWS 服務和資源的臨時存取金鑰。

如需詳細資訊，請參閱 [Amazon DynamoDB 的 Identity and Access Management](#)。

### 使用 IAM 政策進行 DynamoDB 基礎授權

授予許可時，會決定誰可以取得這些許可、可以取得哪些 DynamoDB API 的許可，以及可以對這些資源進行的特定動作。對降低錯誤或惡意意圖所引起的安全風險和影響而言，實作最低權限是其中關鍵。

將許可政策連接至 IAM 身分 (即使用者、群組和角色)，藉此授予可在 DynamoDB 資源上執行操作的許可。

您可以使用下列內容執行這項作業：

- [AWS 受管理 \(預先定義\) 策略](#)
- [客戶受管政策](#)

### 使用 IAM 政策條件進行精細定義存取控制

您在 DynamoDB 中授予許可時，可以指定條件，以決定許可政策的生效方式。對降低錯誤或惡意意圖所引起的安全風險和影響而言，實作最低權限是其中關鍵。

您可以指定使用 IAM 政策授予許可時的條件。例如，您可以執行下列動作：

- 授予許可，允許使用者唯讀存取資料表或次要索引中的特定項目和屬性。
- 授予許可，允許使用者根據該使用者的身分唯寫存取資料表中的特定屬性。

如需詳細資訊，請參閱[使用 IAM 政策條件精細定義存取控制](#)。

### 使用 VPC 端點和政策來存取 DynamoDB

如果只需要從 Virtual Private Cloud (VPC) 內存取 DynamoDB，則應使用 VPC 端點來限制僅來自所需 VPC 的存取。這樣做可防止流量周遊開放式網際網路，也可防止流量受到該環境的限制。

使用 DynamoDB 的 VPC 端點可讓您使用下列內容來控制和限制存取：

- VPC 端點政策：這些政策會套用在 DynamoDB VPC 端點上。它們可讓您控制和限制 API 對 DynamoDB 資料表的存取權限。
- IAM 政策：利用使用連接至使用者、群組或角色的 `aws:sourceVpce` 條件，您可以強制要求對 DynamoDB 資料表進行的所有存取都要透過指定的 VPC 端點。

如需詳細資訊，請參閱[Amazon DynamoDB 端點](#)。

### 考慮用戶端加密

我們建議您先規劃加密策略，再於 DynamoDB 中使用資料表。如果您將敏感或機密資料儲存在 DynamoDB，請考慮在計劃中加入用戶端加密。如此一來，您就能盡量靠近資料來源進行加密，並確保資料在整個生命週期受到保護。將您傳輸中和靜態的敏感資料加密，有助於確保您的明文資料不會被任何第三方取得。

[AWS Database Encryption SDK for DynamoDB](#) 是一個軟體程式庫，可協助您保護在將資料表資料傳送至 DynamoDB 之前的安全性。其會加密、簽署、驗證和解密 DynamoDB 資料表項目。您可以控制要加密和簽署的屬性。

### 主索引鍵考量

請勿在資料表和全域次要索引的[主索引](#)中使用敏感名稱或敏感的純文字資料。鍵名稱將顯示在您的表格定義中。例如，任何具有呼叫權限的人都可以存取主索引鍵名稱[DescribeTable](#)。關鍵值可能會

顯示在您的 [AWS CloudTrail](#) 和其他日誌中。此外，DynamoDB 會使用金鑰值散發資料和路由請求，而且 AWS 管理員可能會觀察這些值以維持服務的健康狀態。

如果您需要在表格或 GSI 金鑰值中使用敏感資料，建議您使用 end-to-end 戶端加密。這可讓您對資料執行鍵值參考，同時確保資料在 DynamoDB 相關記錄中永遠不會以未加密的方式顯示。若要達成此目的，其中一種方法是使用 [DynamoDB 的 AWS 資料庫加密 SDK](#)，但這不是必要的。如果您使用自己的解決方案，我們應該始終使用足夠安全的加密算法。您不應該使用非加密選項（如散列），因為在大多數情況下它們不被認為是足夠安全的。

如果您的主鍵名稱是敏感的，我們建議您 `sk` 改用 `pk` 和。這是一般的最佳作法，可讓您的分割區索引鍵設計變得有彈性。

如果您擔心什麼是正確的選擇，請務必諮詢您的安全專家或 AWS 客戶團隊。

## DynamoDB 偵測性安全最佳實務

下列 Amazon DynamoDB 最佳實務能幫助您偵測潛在安全弱點與事件。

### 用 AWS CloudTrail 於監控 AWS 受管理的 KMS 金鑰使用量

如果您正在使用 [AWS 受管金鑰](#) 用靜態加密，則會登入此金鑰的使用情況 AWS CloudTrail。CloudTrail 通過記錄在您帳戶上採取的操作來提供用戶活動的可見性。CloudTrail 記錄每個動作的重要資訊，包括提出要求的人員、使用的服務、執行的動作、動作的參數，以及 AWS 服務傳回的回應元素。此資訊可協助您追蹤對 AWS 資源所做的變更，並疑難排解作業問題。CloudTrail 可以更輕鬆地確保遵守內部政策和法規標準。

您可以使用 CloudTrail 稽核金鑰使用情況。CloudTrail 會建立包含帳戶 AWS API 呼叫歷史記錄和相關事件的記錄檔。這些記錄檔包括使用 AWS Management Console、AWS SDK 和命令行工具所發出的所有 AWS KMS API 要求，以及透過整合式 AWS 服務提出的要求。您可以使用這些日誌檔來取得使用 KMS 金鑰的時間、所請求的操作、請求者的身分、請求的來源 IP 地址等資訊。如需詳細資訊，請參閱 [《AWS CloudTrail 使用者指南》](#) 中的 [記錄 AWS KMS API 呼叫和 AWS CloudTrail](#)。

### 使用以下方式監控動態 B 作業 CloudTrail

CloudTrail 可同時監視控制平面事件和資料平面事件。控制平面操作可讓您建立及管理 DynamoDB 資料表。它們也可讓您使用索引、串流，以及相依於資料表的其他物件。資料平面操作可讓您對資料表中的資料執行建立、讀取、更新與刪除（也稱為 CRUD）動作。某些資料平面操作也可讓您從次要索引中讀取資料。若要在中啟用資料平面事件的記錄 CloudTrail，您需要啟用中的資料平面 API 活動記錄 CloudTrail。如需更多資訊，請參閱 [記錄追蹤的資料事件](#)。

DynamoDB 中發生活動時，該活動會與事件歷史記錄中的其他 AWS 服務 CloudTrail 事件一起記錄在事件中。如需詳細資訊，請參閱[使用 AWS CloudTrail 來記錄 DynamoDB 操作](#)。您可以在帳戶中查看，搜索和下載最近的事 AWS 件。如需詳細資訊，請參閱 AWS CloudTrail 使用指南中的[檢視具有 CloudTrail 事件歷程記錄](#)的事件。

[如需 AWS 帳戶中持續的事件記錄 \(包括 DynamoDB 的事件\)，請建立追蹤。](#) 追蹤可 CloudTrail 將日誌檔交付到 Amazon Simple Storage Service (Amazon S3) 儲存貯體。根據預設，當您在主控台上建立追蹤時，追蹤會套用至所有 AWS 區域。該追蹤會記錄來自 AWS 分割區中所有區域的事件，並將日誌檔案交付到您指定的 S3 儲存貯體。此外，您還可以設定其他 AWS 服務，以進一步分析 CloudTrail 記錄中收集的事件資料並採取行動。

## 使用 DynamoDB Streams 監控資料平面操作

DynamoDB 已與整合，AWS Lambda 因此您可以建立觸發器 (可自動回應 DynamoDB Streams 中事件的程式碼片段)。您可以利用觸發條件建立對 DynamoDB 資料表資料修改做出反應的應用程式。

如果您在資料表中啟用 DynamoDB Streams，就可以建立串流 Amazon Resource Name (ARN) 與您撰寫之 Lambda 函數的關聯。修改表格中的項目之後，表格的資料流中會立即顯示新記錄。AWS Lambda 輪詢串流，並在偵測到新串流記錄時同步叫用 Lambda 函數。Lambda 函數可以執行您指定的任何動作，例如傳送通知或啟動工作流程。

如需範例，請參閱[教學課程：搭配 Amazon DynamoDB Streams 使用 AWS Lambda](#)。此範例會接收 DynamoDB 事件輸入、處理其包含的訊息，並將一些傳入的事件資料寫入 Amazon CloudWatch 日誌。

## 使用以下方式監控動態 B 組態 AWS Config

您可以使用 [AWS Config](#) 持續監控並記錄 AWS 資源的變更。您也可以使用 AWS Config 來清查您的 AWS 資源。當偵測到先前狀態的變更時，Amazon Simple Notification Service (Amazon SNS) 會通知您檢閱並採取行動。請遵循[AWS Config 使用主控台設定中的](#)指引，確保包含 DynamoDB 資源類型。

您可以進行設 AWS Config 定，將組態變更和通知串流至 Amazon SNS 主題。例如，更新資源時，您可以收到傳送至您電子郵件的通知，因此您可以檢視變更。您也可以根據資源 AWS Config 評估自訂或受管規則時收到通知。

如需範例，請參閱 AWS Config 開發人員指南中的[AWS Config 傳送至 Amazon SNS 的通知](#)主題。

## 監控 DynamoDB 是否符合規則 AWS Config

AWS Config 持續追蹤資源中發生的組態變更。其會檢查這些變更是否違反您規則中的任何條件。如果資源違反規則，會將資源和規則 AWS Config 標記為不符合標準。

透過使 AWS Config 用評估資源組態，您可以評估資源組態是否符合內部實務、產業準則和法規。AWS Config 提供[AWS 受管規則](#)，這些規則是預先定義的可自訂規則，可 AWS Config 用來評估您的 AWS 資源是否符合常見的最佳作法。

為您的 DynamoDB 資源加上標籤，以便進行識別和自動化

您可以使用標籤的形式將中繼 AWS 資料指派給資源。每個標記都是由客戶定義金鑰和選用值組成的簡單標籤，能夠更輕鬆地管理、搜尋和篩選資源。

標記允許實現分組控制。雖然標籤不具固有類型，但能讓您依用途、擁有者、環境或其他條件分類資源。下列是一些範例：

- 安全性：用於確定加密等需求。
- 機密性：資源支援的特定資料機密等級識別符。
- 環境：用來區分開發、測試和生產基礎設施。

如需詳細資訊，請參閱 [AWS 標記策略](#)和 [DynamoDB 的標記](#)。

使用監控與安全最佳實務相關的 Amazon DynamoDB 使用情況。AWS Security Hub

Security Hub 會透過安全控制來評估資源組態和安全標準，協助您遵守各種合規架構。

如需有關使用 Security Hub 評估 DynamoDB 資源的詳細資訊，請參閱《AWS Security Hub 使用者指南》中的 [Amazon DynamoDB 控制項](#)。

# 在 DynamoDB 支援中監控和記錄

監控是維持 DynamoDB 和解決方案的可靠性、可用性和效能的重要組成部分 AWS。您應該從 AWS 解決方案的所有部分收集監控資料，以便輕鬆偵錯多點故障。

## 主題

- [監控計畫](#)
- [效能基準](#)
- [整合服務](#)
- [自動化監控工具](#)
- [使用 Amazon 監控指標 CloudWatch](#)
- [使用 AWS CloudTrail 記錄 DynamoDB 操作](#)
- [使用 DynamoDB 的 CloudWatch 參與者洞察分析資料存取](#)

## 監控計畫

在開始監視 DynamoDB 之前，請先建立監視計劃，其中包含下列問題的答案：

- 監控目標是什麼？
- 要監控哪些資源？
- 監控這些資源的頻率為何？
- 要使用哪些監控工具？
- 誰將執行監控任務？
- 發生問題時應該通知誰？

## 效能基準

透過測量不同時間和不同負載條件下的效能，為環境中的正常 DynamoDB 效能建立基準。當您監控 DynamoDB 時，應該考慮存放歷史監控資料。這個存放的資料會提供基準，讓您與目前的效能資料比較，識別出正常的效能模式和效能異常狀況，再規劃方式來處理問題。若要建立基準，您至少必須監控下列項目：

- 在指定時段使用的讀取或寫入容量單位數目，讓您可追蹤已使用多少佈建的輸送量。



- 在指定時段內超出資料表佈建寫入容量或佈建讀取容量的請求，讓您可判斷哪些請求超出資料表的佈建輸送量配額。
- 系統錯誤，可讓您判斷是否有任何請求導致錯誤。

## 整合服務

DynamoDB 會代表您自動監控您的表格，並透過 Amazon 報告指標。CloudWatch 此外，DynamoDB 與下列項 AWS 服務 目整合，可協助您監控 DynamoDB 資源並進行疑難排解。

- AWS CloudTrail 擷取由您或代表您發出的 API 呼叫和相關事件，AWS 帳戶 並將日誌檔傳遞到您指定的 Amazon S3 儲存貯體。如需詳細資訊，請參閱 [使用 AWS CloudTrail 記錄 DynamoDB 操作](#)。
- Contributor Insights 是一種診斷工具，可讓您一目了然地識別資料表或索引中最常存取和節流的金鑰。如需詳細資訊，請參閱 [使用 DynamoDB 的 CloudWatch 參與者洞察分析資料存取](#)。

## 自動化監控工具

AWS 提供您可用來監視 DynamoDB 的各種工具。建議您盡可能自動化監控任務。您可以使用下列自動化監控工具來監看 DynamoDB，並在發生錯誤時進行回報：

- AWS CloudTrail 警示 — 監視您指定期間內的單一測量結果，並根據指定臨界值在數個期間內相對於指定臨界值的測量結果值執行一或多個動作。

動作是傳送至亞馬遜簡單通知服務 (Amazon SNS) 主題或 Amazon EC2 Auto Scaling 政策的通知。AWS CloudTrail 警示不會僅因為處於特定狀態而叫用動作；狀態必須已變更並維持指定數目的期間。如需詳細資訊，請參閱 [使用 Amazon 監控指標 CloudWatch](#)。

- AWS CloudTrail 日誌監控 — 在帳戶之間共享日誌文件，通過將日 AWS CloudTrail 誌文件發送到 AWS CloudTrail 日誌中實時監控日誌文件，使用 Java 編寫日誌處理應用程序，並驗證您的日誌文件在交付後沒有更改 AWS CloudTrail。如需詳細資訊，請參閱 AWS CloudTrail 使用者指南中的「[什麼是 Amazon CloudWatch 日誌](#)」。

## 使用 Amazon 監控指標 CloudWatch

您可以使用來監控 DynamoDB CloudWatch，該模式會收集 DynamoDB 的原始資料，並將其處理為可讀且接近即時的指標。這些統計資料會保留一段時間，因此您可以存取歷史資訊，以便更好地瞭解 Web 應用程式或服務的執行情況。根據預設，DynamoDB 指標資料會自動傳送至 CloudWatch。如需詳細資訊，請參閱 [什麼是 Amazon CloudWatch ?](#) 和 Amazon CloudWatch 用戶指南中的 [指標保留](#)。

## 主題

- [如何使用 DynamoDB 指標？](#)
- [在 CloudWatch 主控台中檢視指標](#)
- [檢視指標 AWS CLI](#)
- [DynamoDB 指標和維度](#)
- [建立 CloudWatch 鬧鐘](#)

## 如何使用 DynamoDB 指標？

DynamoDB 回報的指標可提供能讓您以不同方式分析的資訊。下列清單顯示一些常見的指標用途。這些是協助您開始的建議，而不是完整清單。

### 如何使用 DynamoDB 指標？

如何？	相關指標
如何監控表格上 TTL 刪除的速率？	您可以監控指定時段內的 <code>TimeToLiveDeletedItemCount</code> ，以追蹤您資料表的 TTL 刪除率。如需使用 <code>TimeToLiveDeletedItemCount</code> 指標的無伺服器應用程式範例，請參閱 <a href="#">使用 DynamoDB 存留時間 (TTL) 自動將項目存檔到 S3 AWS Lambda 和 Amazon 資料 Firehose</a> 。
如何判斷我的佈建輸送量正在使用多少？	您可以監控指定時段內的 <code>ConsumedReadCapacityUnits</code> 或 <code>ConsumedWriteCapacityUnits</code> ，以追蹤佈建輸送量的使用數量。
如何判斷哪些要求超出表格的佈建輸送量配額？	如果請求內的任何事件超過佈建輸送量的配額，則 <code>ThrottledRequests</code> 會增加 1。然後，若要深入了解哪些事件調節了請求，請將 <code>ThrottledRequests</code> 與資料表和其索引的 <code>ReadThrottleEvents</code> 與 <code>WriteThrottleEvents</code> 指標比較。
如何確定是否發生任何系統錯誤？	您可以監控 <code>SystemErrors</code> ，以判斷是否有任何請求導致 HTTP 500 (伺服器錯誤) 代碼。這項指標通常應該等於零。否則建議您予以調查。
如何監視表格作業的延遲值？	您可以監控 <code>SuccessfulRequestLatency</code> 和跟踪平均延遲。偶爾的延遲峰值無需擔心。但是，如果平均延遲很高，則可能



## 如何？

## 相關指標

是您必須解決的潛在問題。如需詳細資訊，請參閱[對 Amazon DynamoDB 中的延遲問題進行疑難排解](#)。

## 在 CloudWatch 主控台中檢視指標

測量結果會先依服務命名空間分組，然後再依每個命名空間內的各種維度組合分組。

若要在 CloudWatch 主控台中檢視指標

1. [請在以下位置開啟 CloudWatch 主控台](https://console.aws.amazon.com/cloudwatch/)。 <https://console.aws.amazon.com/cloudwatch/>
2. 在導覽窗格中，選擇「度量」、「所有量度」。
3. 選取 DynamoDB 命名空間。您也可以選擇用途命名空間來查看 DynamoDB 使用量指標。如需使用量指標的詳細資訊，請參閱[AWS 使用量指標](#)。
4. 瀏覽標籤會顯示命名空間中的所有測量結果。
5. (選擇性) 若要將量度圖形新增至 CloudWatch 儀表板，請選擇動作 > 新增至儀表板。

## 檢視指標 AWS CLI

若要使用取得測量結果資訊 AWS CLI，請使用 CloudWatch 指令 `list-metrics`。以下範例會列出 AWS/DynamoDB 命名空間中的所有指標。

```
aws cloudwatch list-metrics --namespace "AWS/DynamoDB"
```

若要取得指標統計數字，請使用 `get-metric-statistics` 命令。下列命令會取得特定 24 小時期間內資料表 `ProductCatalog` 的 `ConsumedReadCapacityUnits` 統計資料，其粒度為 5 分鐘。

```
aws cloudwatch get-metric-statistics --namespace AWS/DynamoDB \
 --metric-name ConsumedReadCapacityUnits \
 --start-time 2023-11-01T00:00:00Z \
 --end-time 2023-11-02T00:00:00Z \
 --period 360 \
 --statistics Average \
 --dimensions Name=TableName,Value=ProductCatalog
```

範例輸出如下所示：

```
{
 "Datapoints": [
 {
 "Timestamp": "2023-11-01T 09:18:00+00:00",
 "Average": 20,
 "Unit": "Count"
 },
 {
 "Timestamp": "2023-11-01T 04:36:00+00:00",
 "Average": 22.5,
 "Unit": "Count"
 },
 {
 "Timestamp": "2023-11-01T 15:12:00+00:00",
 "Average": 20,
 "Unit": "Count"
 },
 ...
 {
 "Timestamp": "2023-11-01T 17:30:00+00:00",
 "Average": 25,
 "Unit": "Count"
 }
],
 "Label": " ConsumedReadCapacityUnits "
}
```

## DynamoDB 指標和維度

當您與 DynamoDB 互動時，它會將量度和維度傳送至。CloudWatch

### 檢視指標和維度

CloudWatch 顯示 DynamoDB 支援的下列度量：

#### DynamoDB 指標

##### Note

Amazon 以一分鐘的間隔 CloudWatch 彙總這些指標：

- ConditionalCheckFailedRequests

- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits
- ReadThrottleEvents
- ReturnedBytes
- ReturnedItemCount
- ReturnedRecordsCount
- SuccessfulRequestLatency
- SystemErrors
- TimeToLiveDeletedItemCount
- ThrottledRequests
- TransactionConflict
- UserErrors
- WriteThrottleEvents

對於所有其他 DynamoDB 指標，彙總的間隔為五分鐘。

並非所有統計數字，例如 Average 或 Sum，皆適用於所有指標。不過，所有這些值都可透過 Amazon DynamoDB 主控台取得，或透過使用主控 CloudWatch 台或所有指標的 AWS 開發套件來取得。AWS CLI

在以下列表中，每個指標皆有適用於該指標的一組有效統計數字。

可用指標的列表

- [AccountMaxReads](#)
- [AccountMaxTableLevelReads](#)
- [AccountMaxTableLevelWrites](#)
- [AccountMaxWrites](#)
- [AccountProvisionedReadCapacityUtilization](#)
- [AccountProvisionedWriteCapacityUtilization](#)
- [AgeOfOldestUnreplicatedRecord](#)
- [ConditionalCheckFailedRequests](#)
- [ConsumedChangeDataCaptureUnits](#)

- [ConsumedReadCapacityUnits](#)
- [ConsumedWriteCapacityUnits](#)
- [FailedToReplicateRecordCount](#)
- [MaxProvisionedTableReadCapacityUtilization](#)
- [MaxProvisionedTableWriteCapacityUtilization](#)
- [OnDemandMaxReadRequestUnits](#)
- [OnDemandMaxWriteRequestUnits](#)
- [OnlineIndexConsumedWriteCapacity](#)
- [OnlineIndexPercentageProgress](#)
- [OnlineIndexThrottleEvents](#)
- [PendingReplicationCount](#)
- [ProvisionedReadCapacityUnits](#)
- [ProvisionedWriteCapacityUnits](#)
- [ReadThrottleEvents](#)
- [ReplicationLatency](#)
- [ReturnedBytes](#)
- [ReturnedItemCount](#)
- [ReturnedRecordsCount](#)
- [SuccessfulRequestLatency](#)
- [SystemErrors](#)
- [TimeToLiveDeletedItemCount](#)
- [ThrottledPutRecordCount](#)
- [ThrottledRequests](#)
- [TransactionConflict](#)
- [UserErrors](#)
- [WriteThrottleEvents](#)

### AccountMaxReads

帳戶可以使用的讀取容量單位數目上限。此限制不適用於隨選資料表或全域次要索引。

單位 : Count

有效的統計數字：

- **Maximum**：帳戶可以使用的讀取容量單位數目上限。

### AccountMaxTableLevelReads

帳戶的資料表或全域次要索引可以使用的讀取容量單位數目上限。對於隨選資料表，此限制會限制資料表或全域次要索引可使用的最大讀取請求單位數。

單位：Count

有效的統計數字：

- **Maximum**：帳戶的資料表或全域次要索引可以使用的讀取容量單位數目上限。

### AccountMaxTableLevelWrites

帳戶的資料表或全域次要索引可以使用的寫入容量單位數目上限。對於隨選資料表，此限制會限制資料表或全域次要索引可使用的寫入請求單位上限。

單位：Count

有效的統計數字：

- **Maximum**：帳戶的資料表或全域次要索引可以使用的寫入容量單位數目上限。

### AccountMaxWrites

帳戶可以使用的寫入容量單位數目上限。此限制不適用於隨選資料表或全域次要索引。

單位：Count

有效的統計數字：

- **Maximum**：帳戶可以使用的寫入容量單位數目上限。

### AccountProvisionedReadCapacityUtilization

帳戶可以使用的佈建讀取容量單元百分比。

單位：Percent

有效的統計數字：

- **Maximum**：帳戶使用的佈建讀取容量單元百分比上限。
- **Minimum**：帳戶使用的佈建讀取容量單元百分比下限。
- **Average**：帳戶使用的佈建讀取容量單元平均百分比。指標會每隔五分鐘發佈一次。因此，如果您快速調整佈建的讀取容量單位，此統計數字可能無法反映真實的平均值。

#### AccountProvisionedWriteCapacityUtilization

帳戶使用的佈建寫入容量單元百分比。

單位：Percent

有效的統計數字：

- **Maximum**：帳戶使用的佈建寫入容量單元百分比上限。
- **Minimum**：帳戶使用的佈建寫入容量單元百分比下限。
- **Average**：帳戶使用的佈建寫入容量單元平均百分比。指標會每隔五分鐘發佈一次。因此，如果您快速調整佈建的寫入容量單位，此統計數字可能無法反映真實的平均值。

#### AgeOfOldestUnreplicatedRecord

自尚未複製到 Kinesis Data Streams 的紀錄首次出現在 DynamoDB 資料表中以來經過的時間。

單位：Milliseconds

維度：TableName, DelegatedOperation


有效的統計數字：

- **Maximum**.
- **Minimum**.
- **Average**.

#### ConditionalCheckFailedRequests

執行條件式寫入的失敗嘗試次數。PutItem、UpdateItem 以及 DeleteItem 操作可讓您提供邏輯條件，該條件必須評估為 true，才能繼續操作。若此條件評估為

false , ConditionalCheckFailedRequests 會遞增 1。對於 PartiQL Update 和 Delete 陳述式 (其中提供了一個評估為 false 的邏輯條件) , ConditionalCheckFailedRequests 也會遞增 1。

 Note

失敗的條件式寫入會導致 HTTP 400 錯誤 (錯誤的請求)。這些事件會反映在 ConditionalCheckFailedRequests 指標而非 UserErrors 指標中。

單位 : Count

維度 : TableName

有效的統計數字 :

- Minimum
- Maximum
- Average
- SampleCount
- Sum

ConsumedChangeDataCaptureUnits

已耗用的變更資料擷取單位數目。

單位 : Count

維度 : TableName, DelegatedOperation

有效的統計數字 :

- Minimum
- Maximum
- Average

## ConsumedReadCapacityUnits

在指定時段內，使用的佈建與隨需讀取容量單位數目，可讓您追蹤已使用多少佈建的輸送量。您可以擷取資料表及其所有全域次要索引或特定全域次要索引的總消耗讀取容量。如需詳細資訊，請參閱[讀取/寫入容量模式](#)。

此 `TableName` 維度會針對資料表傳回 `ConsumedReadCapacityUnits`，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 `ConsumedReadCapacityUnits`，您必須指定 `TableName` 和 `GlobalSecondaryIndexName`。

### Note

在 Amazon DynamoDB 中，消耗的容量指標會以 CloudWatch 一分鐘的間隔回報為平均值。這表示只持續一秒鐘的容量消耗量短暫而激烈的尖峰可能無法準確反映在 CloudWatch 圖表中，這可能導致該分鐘的明顯消耗率降低。

使用 `Sum` 統計數字來計算消耗的輸送量。例如，取完整一分鐘的 `Sum` 值，再將該值除以一分鐘的秒數 (60)，得出每秒平均 `ConsumedReadCapacityUnits`。您可以將計算的數值與您提供給 DynamoDB 的佈建輸送量數值進行比較。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- `Minimum`：對資料表或索引的任何個別請求所消耗的讀取容量單位數目下限。
- `Maximum`：對資料表或索引的任何個別請求消耗的讀取容量單位數目上限。
- `Average`：每個請求消耗的平均讀取容量。量。

### Note

此 `Average` 數值受樣本數值會為零的閒置時段所影響。

- `Sum`：所耗用的讀取容量單位總數。這是 `ConsumedReadCapacityUnits` 指標最實用的統計數字。
- `SampleCount`— 傳送給 DynamoDB 支援的讀取請求數目。如果沒有消耗讀取容量，則傳回 0。



**Note**

此 SampleCount 數值受樣本數值會為零的閒置時段所影響。

## ConsumedWriteCapacityUnits

在指定時段內，使用的佈建與隨需寫入容量單位數目，可讓您追蹤已使用多少佈建的輸送量。您可以擷取資料表及其所有全域次要索引或特定全域次要索引的消耗總計寫入容量。如需詳細資訊，請參閱[讀取/寫入容量模式](#)。

此 TableName 維度會針對資料表傳回 ConsumedWriteCapacityUnits，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 ConsumedWriteCapacityUnits，您必須指定 TableName 和 GlobalSecondaryIndexName。

**Note**

使用 Sum 統計數字來計算消耗的輸送量。例如，取得一分鐘範圍內的 Sum 值，然後除以分鐘 (60) 中的秒數，以計算 ConsumedWriteCapacityUnits 每秒的平均值 (認識到此平均值並未反白顯示該分鐘期間發生的寫入活動中的任何大型但短暫的尖峰)。您可以將計算的數值與您提供給 DynamoDB 的佈建輸送量數值進行比較。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：


- Minimum：對資料表或索引的任何個別請求消耗的寫入容量單位數目下限。
- Maximum：對資料表或索引的任何個別請求消耗的寫入容量單位數目上限。
- Average：每個請求消耗的平均寫入容量。

**Note**

此 Average 數值受樣本數值會為零的閒置時段所影響。

- Sum：所耗用的寫入容量單位總數。這是 ConsumedWriteCapacityUnits 指標最實用的統計數字。

- `SampleCount` : 對 DynamoDB 的寫入請求數量，即使沒有消耗任何寫入容量也一樣。

 Note

此 `SampleCount` 數值受樣本數值會為零的閒置時段所影響。

### `FailedToReplicateRecordCount`

DynamoDB 無法複製到您的 Kinesis 資料串流的記錄數目。

單位 : Count

尺寸: `TableName`, `DelegatedOperation`

有效的統計數字 :

- Sum

### `MaxProvisionedTableReadCapacityUtilization`

帳戶最高佈建讀取資料表或全域次要索引所使用的佈建讀取容量單位百分比。

單位 : Percent

有效的統計數字 :

- Maximum – 帳戶最高佈建讀取資料表或全域次要索引所使用的佈建讀取容量單位最大百分比。
- Minimum – 帳戶最高佈建讀取資料表或全域次要索引所使用的佈建讀取容量單位最小百分比。
- Average – 帳戶最高佈建讀取資料表或全域次要索引所使用的平均佈建讀取容量單位百分比。指標會每隔五分鐘發佈一次。因此，如果您快速調整佈建的讀取容量單位，此統計數字可能無法反映真實的平均值。

### `MaxProvisionedTableWriteCapacityUtilization`

帳戶最高佈建寫入資料表或全域次要索引所使用的佈建寫入容量百分比。

單位 : Percent

有效的統計數字 :

- **Maximum**：帳戶最高佈建寫入資料表或全域次要索引所使用的佈建寫入容量單位百分比上限。
- **Minimum**：帳戶最高佈建寫入資料表或全域次要索引所使用的佈建寫入容量單位百分比下限。
- **Average**：帳戶最高佈建寫入資料表或全域次要索引所使用的平均佈建寫入容量單位百分比。指標會每隔五分鐘發佈一次。因此，如果您快速調整佈建的寫入容量單位，此統計數字可能無法反映真實的平均值。

## OnDemandMaxReadRequestUnits

資料表或全域次要索引的指定隨選讀取請求單位數目。

若要檢視OnDemandMaxReadRequestUnits表格，您必須指定TableName。若要檢視全域次要索引的 OnDemandMaxReadRequestUnits，您必須指定 TableName 和 GlobalSecondaryIndexName。

單位：計數

尺寸: TableName, GlobalSecondaryIndexName

有效的統計數字：

- **Minimum**— 隨選讀取請求單位的最低設定。如果您使用增UpdateTable加讀取要求單位，此量度會顯示此ReadRequestUnits期間內隨需的最低值。
- **Maximum**— 按需讀取請求單位的最高設定。如果您使用UpdateTable減少讀取要求單位，此量度會顯示此ReadRequestUnits期間內隨選的最高值。
- **Average**— 平均按需讀取請求單位。OnDemandMaxReadRequestUnits 指標會每隔五分鐘發佈一次。因此，如果您快速調整隨選讀取請求單位，此統計資料可能不會反映真正的平均值。

## OnDemandMaxWriteRequestUnits

資料表或全域次要索引的指定隨選寫入要求單位數目。

若要檢視OnDemandMaxWriteRequestUnits表格，您必須指定TableName。若要檢視全域次要索引的 OnDemandMaxWriteRequestUnits，您必須指定 TableName 和 GlobalSecondaryIndexName。

單位：Count

尺寸: TableName, GlobalSecondaryIndexName

有效的統計數字：

- **Minimum**— 隨選寫入請求單位的最低設定。如果您使用增UpdateTable加寫入要求單位，此量度會顯示此WriteRequestUnits期間內隨需的最低值。
- **Maximum**— 按需寫入請求單位的最高設定。如果您使用UpdateTable減少寫入要求單位，此量度會顯示此WriteRequestUnits期間內隨需的最高值。
- **Average**— 平均按需寫入請求單位。OnDemandMaxWriteRequestUnits 指標會每隔五分鐘發佈一次。因此，如果您快速調整隨選寫入請求單位，此統計資料可能不會反映真正的平均值。

### OnlineIndexConsumedWriteCapacity

將新的全域次要索引新增至資料表時，所消耗的寫入容量單位數目。如果索引的寫入容量太低，可能會調節回填階段期間傳入的寫入活動。這可能會增加建立索引所需的時間。您應該在建立索引的同時監控此統計數字，以便判斷索引的寫入容量是否佈建不足。

即使索引仍在建置中，您也可以使用 UpdateTable 操作調整索引的寫入容量。

索引的ConsumedWriteCapacityUnits度量不包括建立索引期間使用的寫入輸送量。

#### Note

如果新全域次要索引的回填階段快速完成 (不到幾分鐘)，則可能不會發出此指標，如果基底資料表在索引中要回填的項目很少或沒有，就可能發生這種情況。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

## OnlineIndexPercentageProgress

新的全域次要索引新增至資料表時的完成百分比。DynamoDB 必須先為新索引分配資源，然後將資料表中的屬性回填至索引。對於大型資料表，此程序可能需要很長的時間。您應該監控此統計數字，以便在 DynamoDB 建置索引時檢視相對進度。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

## OnlineIndexThrottleEvents

將新的全域次要索引新增至資料表時，所發生的寫入調節事件數目。這些事件表示需要較長的時間才能完成索引建立，因為傳入寫入活動超過索引的佈建寫入輸送量。

即使索引仍在建置中，您也可以使用 UpdateTable 操作調整索引的寫入容量。

索引的 WriteThrottleEvents 量度不包含索引建立期間發生的任何節流事件。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

## PendingReplicationCount

[全域表格版本 \(舊版\)](#) 指標 (僅限全域資料表)。寫入一個複本列表，但尚未寫入全域資料表中的另一個複本的項目更新數目。

單位：Count

維度：TableName, ReceivingRegion

有效的統計數字：

- Average
- Sample Count
- Sum

## ProvisionedReadCapacityUnits

資料表或全域次要索引佈建的讀取容量單位數目。此 TableName 維度會針對資料表傳回 ProvisionedReadCapacityUnits，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 ProvisionedReadCapacityUnits，您必須指定 TableName 和 GlobalSecondaryIndexName。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- Minimum：佈建讀取容量的最低設定。如果使用 UpdateTable 增加讀取容量，此指標會顯示在此時段內佈建的 ReadCapacityUnits 最低數值。
- Maximum：佈建讀取容量的最高設定。如果使用 UpdateTable 減少讀取容量，此指標會顯示在此時段內佈建的 ReadCapacityUnits 最高數值。
- Average：平均佈建讀取容量。ProvisionedReadCapacityUnits 指標會每隔五分鐘發佈一次。因此，如果您快速調整佈建的讀取容量單位，此統計數字可能無法反映真實的平均值。

## ProvisionedWriteCapacityUnits

資料表或全域次要索引佈建的寫入容量單位數目。

此 `TableName` 維度會針對資料表傳回 `ProvisionedWriteCapacityUnits`，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 `ProvisionedWriteCapacityUnits`，您必須指定 `TableName` 和 `GlobalSecondaryIndexName`。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- **Minimum**：佈建寫入容量的最低設定。如果使用 `UpdateTable` 增加寫入容量，此指標會顯示在此時段內佈建的 `WriteCapacityUnits` 最低數值。
- **Maximum**：佈建寫入容量的最高設定。如果使用 `UpdateTable` 減少寫入容量，此指標會顯示在此時段內佈建的 `WriteCapacityUnits` 最高數值。
- **Average**：佈建平均的寫入容量。`ProvisionedWriteCapacityUnits` 指標會每隔五分鐘發佈一次。因此，如果您快速調整佈建的寫入容量單位，此統計數字可能無法反映真實的平均值。

## ReadThrottleEvents

對 DynamoDB 的請求超過資料表或全域次要索引的佈建讀取容量單位。

單一請求可能會導致多個事件。例如，`BatchGetItem` 讀取 10 個項目會作為 10 個 `GetItem` 事件處理。針對每個事件，如果該事件經調節，則 `ReadThrottleEvents` 會遞增 1。除非全部 10 個 `GetItem` 事件經過調節，否則整個 `BatchGetItem` 的 `ThrottledRequests` 指標不會遞增。

此 `TableName` 維度會針對資料表傳回 `ReadThrottleEvents`，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 `ReadThrottleEvents`，您必須指定 `TableName` 和 `GlobalSecondaryIndexName`。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- **SampleCount**
- **Sum**

## ReplicationLatency

(此指標適用於 DynamoDB 全域資料表。) 在 DynamoDB Streams 中顯示一個複本列表的更新項目到該項目出現在全域資料表的另一個複本中的經過時間。

單位 : Milliseconds

維度 : TableName, ReceivingRegion

有效的統計數字 :

- Average
- Minimum
- Maximum

## ReturnedBytes

GetRecords 操作 (Amazon DynamoDB Streams) 在指定時段內傳回的位元組數目。

單位 : Bytes

維度 : Operation, StreamLabel, TableName

有效的統計數字 :

- Minimum
- Maximum
- Average
- SampleCount
- Sum

## ReturnedItemCount

在指定時段內由 Query、Scan 或 ExecuteStatement (選取) 操作所傳回的項目數。

傳回的項目數量不一定與評估的項目數量相同。例如，假設您請求的是資料表上的 Scan 或具有 100 個項目的索引，但指定了縮小結果的 FilterExpression，故只返回 15 個項目。在這種情況下，來自 Scan 的回應會包含 100 個 ScanCount 和 15 個返回項目的 Count。

單位 : Count



維度：TableName, Operation

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

ReturnedRecordsCount

GetRecords 操作 (Amazon DynamoDB Streams) 在指定時段內傳回的串流紀錄數目。

單位：Count

維度：Operation, StreamLabel, TableName

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount
- Sum

SuccessfulRequestLatency

在指定時段內對 DynamoDB 或 Amazon DynamoDB Streams 發起的成功請求延遲。SuccessfulRequestLatency 可以提供兩種不同類型的資訊：

- 成功請求的經過時間 (Minimum、Maximum、Sum 或 Average)。
- 成功請求的數量 (SampleCount)。

SuccessfulRequestLatency 僅反映 DynamoDB 或 Amazon DynamoDB 串流中的活動，不會考慮網路延遲或用戶端活動。

單位：Milliseconds

維度：TableName, Operation, StreamLabel

有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount

SystemErrors

對 DynamoDB 或 Amazon DynamoDB Streams 發起的能在指定時段內產生 HTTP 500 狀態碼的請求。HTTP 500 通常表示內部服務錯誤。

單位：Count

維度：TableName, Operation

有效的統計數字：

- Sum
- SampleCount

TimeToLiveDeletedItemCount

存留時間 (TTL) 在指定時段內刪除的項目數量。此指標可協助您監控資料表上的 TTL 刪除率。

單位：Count

尺寸：TableName

有效的統計數字：

- Sum

ThrottledPutRecordCount

因 Kinesis Data Streams 容量不足而受到調節的 Kinesis Data Streams 的紀錄數目。

單位：Count

尺寸：TableName, DelegatedOperation


有效的統計數字：

- Minimum
- Maximum
- Average
- SampleCount

ThrottledRequests

對 DynamoDB 的請求超過資源 (例如資料表或索引) 的佈建輸送量限制。

如果請求內的任何事件超過佈建輸送量的限額，則 ThrottledRequests 會遞增 1。例如，如果使用全域次要索引更新資料表中的項目，則會有多個事件：寫入資料表和寫入每個索引。如果其中一或多個事件經過調節，則 ThrottledRequests 會遞增 1。


 Note

在批次處理請求中 (BatchGetItem 或 BatchWriteItem)，ThrottledRequests 只有在每個批次中的請求進行調節時才會遞增。

如果批次內的任何個別請求經過調節，則會遞增下列一個指標：

- ReadThrottleEvents：在 BatchGetItem 內調節的 GetItem 事件。
- WriteThrottleEvents：在 BatchWriteItem 內調節的 PutItem 或 DeleteItem 事件。

若要深入了解哪些事件調節了請求，請將 ThrottledRequests 與資料表和其索引的 ReadThrottleEvents 和 WriteThrottleEvents 進行比較。

 Note

經調節的請求會導致 HTTP 400 狀態碼。所有這些事件都會反映在 ThrottledRequests 指標而非 UserErrors 指標中。

單位：Count

維度：TableName, Operation

有效的統計數字：

- Sum
- SampleCount

TransactionConflict


因同一項目上並行請求之間的交易衝突而拒絕的項目層級請求。如需詳細資訊，請參閱在 [DynamoDB 中處理交易衝突](#)。

單位：Count

維度：TableName


有效的統計數字：

- Sum：因交易衝突而拒絕的項目層級請求數目。

 Note

如果對 `TransactWriteItems` 或 `TransactGetItems` 的呼叫中有多個項目層級請求遭拒絕，則對於每個項目層級的 `Put`、`Update`、`Delete` 或 `Get` 請求，Sum 會遞增 1。

- SampleCount：因交易衝突而拒絕的請求數目。

 Note

如果對 `TransactWriteItems` 或 `TransactGetItems` 的呼叫中有多個項目層級請求遭拒絕，則 `SampleCount` 只會遞增 1。

- Min：對 `TransactWriteItems`、`TransactGetItems`、`PutItem`、`UpdateItem` 或 `DeleteItem` 的呼叫中遭拒絕的項目層級請求數目下限。
- Max：對 `TransactWriteItems`、`TransactGetItems`、`PutItem`、`UpdateItem` 或 `DeleteItem` 的呼叫中被拒絕的項目層級請求數目上限。
- Average：對 `TransactWriteItems`、`TransactGetItems`、`PutItem`、`UpdateItem` 或 `DeleteItem` 的呼叫中被拒絕的項目層級請求平均數。

## UserErrors

對 DynamoDB 或 Amazon DynamoDB Streams 發起的能在指定時段內產生 HTTP 400 狀態碼的請求。HTTP 400 通常表示用戶端錯誤，例如參數組合無效、嘗試更新不存在的資料表或請求簽章不正確。

有些例外狀況會記錄關於 UserErrors 的指標，範例如下：

- ResourceNotFoundException
- ValidationException
- TransactionConflict

所有這些事件都會反映在 UserErrors 指標中，但下列項目除外：

- ProvisionedThroughputExceededException— 請參閱本節中的 ThrottledRequests 指標。
- ConditionalCheckFailedException— 請參閱本節中的 ConditionalCheckFailedRequests 指標。

UserErrors 代表針對目前 AWS 區域和目前帳戶的 DynamoDB 串流請求或 Amazon DynamoDB 串流請求的 HTTP 400 錯誤彙總。AWS

單位：Count

有效的統計數字：

- Sum
- SampleCount

## WriteThrottleEvents

對 DynamoDB 的請求超過資料表或全域次要索引的佈建寫入容量單位。

單一請求可能會導致多個事件。例如，在附有三個全域次要索引的資料表上提出 PutItem 請求會導致四個事件：一個資料表寫入事件、三個索引寫入事件。針對每個事件，如果該事件經調節，則 WriteThrottleEvents 指標會遞增 1。針對單一 PutItem 請求，如果有任何事件經過調節，則 ThrottledRequests 也會遞增 1。針對 BatchWriteItem，除非全部個別 PutItem 或 DeleteItem 事件經過調節，否則整個 BatchWriteItem 的 ThrottledRequests 指標不會遞增。

此 TableName 維度會針對資料表傳回 WriteThrottleEvents，但不會針對任何全域次要索引傳回。若要檢視全域次要索引的 WriteThrottleEvents，您必須指定 TableName 和 GlobalSecondaryIndexName。

單位：Count

維度：TableName, GlobalSecondaryIndexName

有效的統計數字：

- Sum
- SampleCount

用量指標

中的使用量度可 CloudWatch 讓您透過視覺化主 CloudWatch 控制台中的指標、建立自訂儀表板、偵測 CloudWatch 異常偵測活動中的變更，以及設定警示，以便在使用量接近閾值時提醒您，藉此主動管理使用情況。

DynamoDB 也將這些使用量指標與 Service Quotas 整合。您可以用 CloudWatch 來管理帳戶對服務配額的使用情況。如需詳細資訊，請參閱[視覺化您的 Service Quotas](#) 和[設定警報](#)。

可用使用量指標的清單

- [AccountProvisionedWriteCapacityUnits](#)
- [AccountProvisionedReadCapacityUnits](#)
- [TableCount](#)

AccountProvisionedWriteCapacityUnits

帳戶所有資料表或全域次要索引佈建的寫入容量單位數目總和。

單位：Count

有效的統計數字：

- Minimum：在一時間段內佈建的最低寫入容量單位數目。
- Maximum：在一時間段內佈建的最大寫入容量單位數目。
- Average — 在一段時間內佈建的寫入容量單位帳戶的平均數量。

此指標會每隔五分鐘發佈一次。因此，如果您快速調整佈建的寫入容量單位，此統計數字可能無法反映真實的平均值。

### AccountProvisionedReadCapacityUnits

所有資料表或全域次要索引佈建的讀取容量單位數目。

單位：Count

有效的統計數字：

- Minimum：在一時間段內佈建的最低讀取容量單位數目。
- Maximum：在一時間段內佈建的最大讀取容量單位數目。
- Average — 在一段時間內佈建的讀取容量單位帳戶的平均數量。

此指標會每隔五分鐘發佈一次。因此，如果您快速調整佈建的讀取容量單位，此統計數字可能無法反映真實的平均值。

### TableCount

帳戶中作用中資料表的數目。

單位：Count

有效的統計數字：

- Minimum：在一時間段內的最小資料表數目。
- Maximum：在一時間段內的最大資料表數目。
- Average — 在一時間段內的平均資料表數目。

## 了解 DynamoDB 的指標和維度

DynamoDB 的指標條件由帳戶的數值、資料表名稱、全域次要索引名稱或操作進行限定。您可以使用 CloudWatch 主控台沿下表中的任何維度擷取 DynamoDB 資料。

可用維度的列表

- [DelegatedOperation](#)
- [GlobalSecondaryIndexName](#)

- [作業](#)
- [OperationType](#)
- [動詞](#)
- [ReceivingRegion](#)
- [StreamLabel](#)
- [TableName](#)

## DelegatedOperation

此維度將資料限制在 DynamoDB 代表您執行的操作中。擷取下列操作：

- Kinesis Data Streams 的變更資料擷取。

## GlobalSecondaryIndexName

此維度將資料限制為資料表上的全域次要索引。若您指定 GlobalSecondaryIndexName，您也必須指定 TableName。

## 作業

此維度將資料限制為下列其中一項 DynamoDB 操作：

- PutItem
- DeleteItem
- UpdateItem
- GetItem
- BatchGetItem
- Scan
- Query
- BatchWriteItem
- TransactWriteItems
- TransactGetItems
- ExecuteTransaction
- BatchExecuteStatement



- `ExecuteStatement`

此外，您可以將資料限制為下列 Amazon DynamoDB Streams 操作：

- `GetRecords`

### OperationType

此維度將資料限制為下列其中一種操作類型：

- `Read`
- `Write`

此維度針對回應 `ExecuteTransaction` 和 `BatchExecuteStatement` 請求發出。

### 動詞

此維度將資料限制為下列其中一個 DynamoDB PartiQL 動詞：

- `Insert` (插入) : `PartiQLInsert`
- `Select` (選取) : `PartiQLSelect`
- `Update` (更新) : `PartiQLUpdate`
- `Delete` (刪除) : `PartiQLDelete`

此維度針對回應 `ExecuteStatement` 操作發出。

### ReceivingRegion

此維度將資料限制在特定 AWS 區域。它可與源自 DynamoDB 全域資料表內複本列表的指標搭配使用。

### StreamLabel

此維度將資料限制為特定串流標籤。它與源自 Amazon DynamoDB Streams `GetRecords` 操作的指標搭配使用。

### TableName

此維度將資料限制為特定資料表。此值可以是目前區域和目前 AWS 帳戶中的任何表格名稱。

## 建立 CloudWatch 鬧鐘

**CloudWatch 警示**會監視指定期間內的單一量度，並根據一段時間內相對於臨界值的指標值，執行一或多個指定的動作。此動作是傳送到 Amazon SNS 主題或 Auto Scaling 政策的通知。您也可以將警示新增至儀表板，以便在多個地區監視和接收有關資 AWS 源和應用程式的警示。您可以建立的警報數量沒有限制。CloudWatch 警示不會僅因為處於特定狀態而叫用動作；狀態必須已變更並維持指定數目的期間。如需建議的 DynamoDB 警示清單，請參閱[建議](#)的警示。

### Note

建立 CloudWatch 警示時，您必須指定所有必要的維度，因為不 CloudWatch 會彙總遺失維度的量度。建立 CloudWatch 警示時，建立遺失尺寸的警示不會導致錯誤。

假設您有一個佈建的表格，其中包含五個讀取容量單位。您希望在耗用整個佈建的讀取容量之前收到通知，因此您決定建立 CloudWatch 警示，以便在耗用的容量達到您為表格佈建的 80% 時收到通知。您可以在 CloudWatch 主控台中建立警報，也可以使用 AWS CLI。

### 在 CloudWatch 控制台中創建警報

在 CloudWatch 主控台中建立鬧鐘

1. 請登入 AWS Management Console 並開啟 CloudWatch 主控台，網址為 <https://console.aws.amazon.com/cloudwatch/>。
2. 在導覽窗格中，選擇 Alarms (警示)、All alarms (所有警示)。
3. 選擇 Create alarm (建立警示)。
4. **ConsumeReadCapacityUnits**在「測量結果名稱」資料欄中尋找含有您要監督之表格的資料列。選取此資料列旁的核取方塊，然後選擇「選取測量結果」。
5. 在「指定量度和條件」下，選擇「總和」做為統計值。選擇一個 1 分鐘的時間段。
6. 在 Conditions (條件) 下，指定以下內容：
  - a. 對於閾值類型，選擇靜態。
  - b. 對於「無論何**ConsumedReadCapacityUnits**時」，選擇「大於/等於」，並將臨界值指定為 240。
7. 選擇下一步。
8. 在 [通知] 下，選擇**In alarm**並選取警示ALARM處於狀態時通知的 SNS 主題。
9. 完成時，請選擇下一步。

10. 輸入警示的名稱和說明，然後選擇 Next (下一步)。
11. 在 Preview and create (預覽及建立) 下，請確認資訊和條件都是您希望的內容，然後選擇 Create alarm (建立警示)。

## 在中建立警示 AWS CLI

```
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
```

測試警示。

```
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
```

## 更多 AWS CLI 範例

下列程序說明如果您的要求超過表格的佈建完成配額，系統會如何收到通知。

1. 創建一個 Amazon SNS 主題 `arn:aws:sns:us-east-1:123456789012:requests-exceeding-throughput`。如需詳細資訊，請參閱 [設定 Amazon Simple Notification Service](#)。
2. 建立警示。

```
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:requests-exceeding-throughput
```

```
-\\-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. 測試警示。

```
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
```

下列程序說明系統錯誤時如何收到通知。

1. 創建一個 Amazon SNS 主題 `arn:aws:sns:us-east-1:123456789012:notify-on-system-errors`。如需詳細資訊，請參閱 [設定 Amazon Simple Notification Service](#)。
2. 建立警示。

```
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. 測試警示。

```
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
```

## 使用 AWS CloudTrail 記錄 DynamoDB 操作

DynamoDB 與服務整合在一起 AWS CloudTrail，該服務可提供 DynamoDB 中使用者、角色或 AWS 服務所採取的動作記錄。CloudTrail 將 DynamoDB 的所有 API 呼叫擷取為事件。擷取的呼叫包括來自 DynamoDB 主控台的呼叫，以及對 DynamoDB API 操作 (同時使用 PartiQL 和傳統 API) 的程式碼呼叫。如果您建立追蹤，您可以啟用持續交付 CloudTrail 事件到 Amazon S3 儲存貯體，包括 DynamoDB 的事件。如果您未設定追蹤，您仍然可以在 [事件歷程記錄] 中檢視 CloudTrail 主控台中最近的事件。使用收集的資訊 CloudTrail，您可以判斷向 DynamoDB 發出的請求、提出請求的 IP 位址、提出請求的人員、提出請求的時間以及其他詳細資訊。

如需強大的監控和警示功能，您也可以將 CloudTrail 事件與 [Amazon CloudWatch 日誌](#) 整合。若要加強對 DynamoDB 服務活動的分析，並識別 AWS 帳戶的活動變更，您可以使用 [Amazon Athena](#) 查詢 AWS CloudTrail 日誌。例如，您可以使用查詢來識別趨勢，並依屬性 (例如來源 IP 地址或使用者) 進一步隔離活動。

若要進一步了解 CloudTrail，包括如何設定和啟用它，請參閱 [AWS CloudTrail 使用者指南](#)。

### 主題

- [DynamoDB 資訊 CloudTrail](#)
- [了解資料 DynamoDB 日誌檔案項目](#)

## DynamoDB 資訊 CloudTrail

CloudTrail 在您創建 AWS 帳戶時，您的帳戶已啟用。當 DynamoDB 中發生受支援的事件活動時，該活動會與事件歷史記錄中的其他 AWS 服務 CloudTrail 事件一起記錄在事件中。您可以在帳戶中查看，搜索和下載最近的事 AWS 件。如需詳細資訊，請參閱 [使用 CloudTrail 事件歷程記錄](#)。

如需 AWS 帳戶中持續的事件記錄 (包括 DynamoDB 的事件)，請建立追蹤。追蹤可 CloudTrail 將日誌檔交付到 Amazon S3 儲存貯體。根據預設，當您在主控台中建立追蹤時，追蹤會套用至所有 AWS 區域。追蹤記錄來自 AWS 分區中所有區域的事件，並將日誌檔傳送到您指定的 Amazon S3 儲存貯體。

此外，您還可以設定其他 AWS 服務，以進一步分析 CloudTrail 記錄中收集的事件資料並採取行動。如需詳細資訊，請參閱下列內容：

- [建立追蹤的概觀](#)
- [CloudTrail 支援的服務與整合](#)
- [設定 Amazon SNS 通知 CloudTrail](#)
- [從多個區域接收 CloudTrail 日誌文件並從多個帳戶接收 CloudTrail 日誌文件](#)

## 控制平面事件 CloudTrail

預設會將下列 API 動作記錄為 CloudTrail 檔案中的事件：

### 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](#)

## 資料平面事件 CloudTrail

若要在 CloudTrail 檔案中啟用下列 API 動作的記錄功能，您需要啟用中的資料平面 API 活動記錄 CloudTrail。如需更多資訊，請參閱[記錄追蹤的資料事件](#)。

您可以依資源類型篩選資料平面事件，以精細控制您要選擇性地記錄哪些 DynamoDB API 呼叫並支付費用。CloudTrail 例如，藉由指定 `AWS::DynamoDB::Stream` 作為資源類型，您就能只記錄 DynamoDB 串流 API 的呼叫。對於已啟用串流的資料表，資料平面事件中的資源欄位會同時包含 `AWS::DynamoDB::Stream` 和 `AWS::DynamoDB::Table`。如果您指定 `AWS::DynamoDB::Table` 作為資源類型，則會根據預設同時記錄 DynamoDB 資料表和 DynamoDB 串流事件。如果您不想要記錄串流事件，可以新增額外的[篩選條件](#)來排除串流事件。如需詳細資訊，請參閱 AWS CloudTrail API 參考[DataResource](#)中的。

## Amazon DynamoDB

- [BatchExecuteStatement](#)
- [BatchGetItem](#)
- [BatchWriteItem](#)
- [DeleteItem](#)
- [ExecuteStatement](#)
- [ExecuteTransaction](#)
- [GetItem](#)
- [PutItem](#)
- [查詢](#)
- [掃描](#)
- [TransactGetItems](#)
- [TransactWriteItems](#)
- [UpdateItem](#)



**Note**

DynamoDB 即時時間資料平面動作不會由下列人員記錄 CloudTrail

## DynamoDB Streams

- [GetRecords](#)
- [GetShardIterator](#)

## 了解資料 DynamoDB 日誌檔案項目

追蹤是一種組態，可讓事件以日誌檔的形式傳遞到您指定的 Amazon S3 儲存貯體。CloudTrail 記錄檔包含一或多個記錄項目。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根或使用者憑證提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

**Note**

非關鍵屬性值將在使用 PartiQL API 的操作 CloudTrail 日誌中編輯，並且不會出現在使用傳統 API 的操作日誌中。

如需詳細資訊，請參閱[CloudTrail 使用 userIdentity 元素](#)。

下列範例示範這些事件類型的 CloudTrail 記錄檔：

## Amazon DynamoDB

- [UpdateTable](#)
- [DeleteTable](#)
- [CreateCluster](#)

- [PutItem \(成功\)](#)
- [UpdateItem \(不成功\)](#)
- [TransactWriteItems \(成功\)](#)
- [TransactWriteItems \(與 TransactionCanceledException\)](#)
- [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 boto-core/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 (成功)

```

{
 "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 (不成功)

```

{
 "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 (成功)

```

{
 "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 (與 TransactionCanceledException)

```
{
 "Records": [
 {
 "eventVersion": "1.06",
 "userIdentity": {
 "type": "AssumedRole",
 "principalId": "AKIAIOSFODNN7EXAMPLE",
 "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
 "accountId": "111122223333",
 "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
 "sessionContext": {
 "sessionIssuer": {
 "type": "Role",
 "principalId": "AKIAI44QH8DHBEXAMPLE",
 "arn": "arn:aws:iam::444455556666:role/admin-role",
 "accountId": "444455556666",
 "userName": "bob"
 },
 "attributes": {
 "creationDate": "2020-09-03T22:14:13Z",
 "mfaAuthenticated": "false"
 }
 }
 },
 "eventTime": "2019-02-01T00:42:34Z",
 "eventSource": "dynamodb.amazonaws.com",
 "eventName": "TransactWriteItems",
 "awsRegion": "us-west-2",
 "sourceIPAddress": "192.0.2.0",
 "userAgent": "aws-cli/1.16.93 Python/3.4.7
Linux/4.9.119-0.1.ac.277.71.329.metal1.x86_64 botocore/1.12.83",
 "errorCode": "TransactionCanceledException",
 "errorMessage": "Transaction cancelled, please refer cancellation reasons
for specific reasons [ConditionalCheckFailed, None]",
 "requestParameters": {
 "requestItems": [
 {
 "operation": "Put",
 "tableName": "Music",
```

```
 "key": {
 "Artist": "No One You Know",
 "SongTitle": "Call Me Today"
 },
 "items": [
 "Artist",
 "SongTitle",
 "AlbumTitle"
],
 "conditionExpression": "#AT = :A",
 "expressionAttributeNames": {
 "#AT": "AlbumTitle"
 },
 "returnValuesOnConditionCheckFailure": "ALL_OLD"
 },
 {
 "operation": "Update",
 "tableName": "Music",
 "key": {
 "Artist": "No One You Know",
 "SongTitle": "Call Me Tomorrow"
 },
 "updateExpression": "SET #AT = :newval",
 "ConditionExpression": "attribute_not_exists(Rating)",
 "ExpressionAttributeNames": {
 "#AT": "AlbumTitle"
 },
 "returnValuesOnConditionCheckFailure": "ALL_OLD"
 },
 {
 "operation": "Delete",
 "TableName": "Music",
 "key": {
 "Artist": "No One You Know",
 "SongTitle": "Call Me Yesterday"
 },
 "conditionExpression": "#P between :lo and :hi",
 "expressionAttributeNames": {
 "#P": "Price"
 },
 "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
 },
 {
 "operation": "ConditionCheck",
```

```

 "TableName": "Music",
 "Key": {
 "Artist": "No One You Know",
 "SongTitle": "Call Me Now"
 },
 "ConditionExpression": "#P between :lo and :hi",
 "ExpressionAttributeNames": {
 "#P": "Price"
 },
 "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
 }
],
"returnConsumedCapacity": "TOTAL",
"returnItemCollectionMetrics": "SIZE"
},
"responseElements": null,
"requestID": "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|AAAAAAAAAAAAH7HF3xwDQHBrvk2UBZ1PKh8bX3F
+JeH0rFwHCE7dz4VGV1ZoJ5bMxQwkmerA3wzCTL+zSseGLdSXNJP14EwrjLNvDNoZeRSJ/
n6xc3I4NYOptR4zR8d7VrjMAD6h5nR12NtxGIgJ/
dVsUpLuWsHyCW3PPbKsM1JSruVRWoitRhSd3S6s1EWEpB0bDC7+
+ISH5mXrCH0nvyezQKlqNshTSPZ5jWwqRj2VNSXCMTGXv9P01/
U0bp0UI2cuRTchgUpPSe3ur2sQrRj3K1bmIyCz7P
+H3CYlugafi8fQ5kipDSkESkIWS605ejzibWKg/3izms1eVIm/
zLFdEeihCYJ7G8fpHUSLX5JAK3ab68aUXGSFEZLONntgNIhQkcMo00/
mJlaIgkEdBUyqvZ01vtKUBH5YonIrZqSUhv8Coc+mh24v0g1YI+SPIXlr
+Ln154BG6AjirmaScjHACVXoPDxPsXSJXC4c9HjoC3YSskCPV7uWi0f65/
n7JAT3cscKX2ISaLHwYzJPaMBSftx0geRLm3BnisL32nT8uTj2gF/
PUrEjdyoqTX7EerQpcaekXm0gay5Kh8n4T2uPdM83f356vRpar/
DDp8pLFD0ddb6Yvz7zU2zGdAvTod3IScC1GpTqcjRxaMh1BVZy1TnI9Cs
+7fXMdUF6xYScjR2725icFBNLojSFVDmsfHabXaCEpmeuXZsLbp5CjcPAHa66R8mQ5tSoFjrzoEzeB4uconEXAMPLE=="
 },
 "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"
 }
]
```

## 使用 DynamoDB 的 CloudWatch 參與者洞察分析資料存取

Amazon Amazon DynamoDB 參與 CloudWatch 者深入解析是一種診斷工具，可讓您一目了然地識別資料表或索引中最常存取和節流的金鑰。此工具使用[CloudWatch 貢獻者見解](#)。

透過在資料表或全域次要索引上啟用 DynamoDB 的參與 CloudWatch 者深入解析，您可以檢視這些資源中存取最多和節流的項目。

### Note

CloudWatch DynamoDB 的參與者深入解析會收取費用。如需有關定價的詳細資訊，請參閱[Amazon CloudWatch 定價](#)。

### 主題

- [CloudWatch DynamoDB 的貢獻者深入解析：其運作方式](#)
- [開始使用 DynamoDB 的 CloudWatch 參與者深入解析](#)
- [將 IAM 與 DynamoDB 的 CloudWatch 貢獻者深入解析搭配使用](#)

## CloudWatch DynamoDB 的貢獻者深入解析：其運作方式

Amazon DynamoDB 與[CloudWatch 參與者見解](#)整合，以提供資料表或全域次要索引中存取次數最多和節流項目的相關資訊。DynamoDB 會透過 CloudWatch 參與者見解[規則](#)、[報表和報表資料圖形](#)，為您[提供此資訊](#)。

如需有關參 CloudWatch 與者深入解析的詳細資訊，請參閱 Amazon [使用者指南中的使用參與者深入解析來分析高基數資料](#)。CloudWatch

下列各節說明 DynamoDB 參與 CloudWatch 者深入解析的核心概念和行為。

## 主題

- [CloudWatch DynamoDB 規則的參與者深入解析](#)
- [瞭解 DynamoDB 圖形的 CloudWatch 參與者深入解析](#)
- [與其他 DynamoDB 功能互動](#)
- [CloudWatch 針對 DynamoDB 計費提供者深入解析](#)

## CloudWatch DynamoDB 規則的參與者深入解析

[當您在資料表或全域次要索引上啟用 DynamoDB 的 CloudWatch 參與者深入解析時，DynamoDB 會代表您建立下列規則：](#)

- **最常存取項目 (分割區索引鍵)：** 在您的資料表或全域次要索引中辨識最常存取項目的分割區索引鍵。

CloudWatch 規則名稱格式：DynamoDBContributorInsights-PKC-[resource\_name]-[creationtimestamp]

- **最常調節項目 (分割區索引鍵)：** 在您的資料表或全域次要索引中辨識最常調節項目的分割區索引鍵。

CloudWatch 規則名稱格式：DynamoDBContributorInsights-PKT-[resource\_name]-[creationtimestamp]

### Note

當您在 DynamoDB 表上啟用參與者深入解析時，您仍會受到貢獻者見解規則限制的約束。如需更多相關資訊，請參閱 [CloudWatch Service Quotas](#)。

如果您的資料表或全域次要索引有排序索引鍵，DynamoDB 也會建立以下排序索引鍵專用的規則：

- **最常存取項目 (分割區和排序索引鍵)：** 在您的資料表或全域次要索引中辨識最常存取項目的分割區和排序索引鍵。

CloudWatch 規則名稱格式：DynamoDBContributorInsights-SKC-[resource\_name]-[creationtimestamp]

- **最常調節項目 (分割區和排序索引鍵)：** 在您的資料表或全域次要索引中辨識最常調節項目的分割區和排序索引鍵。

CloudWatch 規則名稱格式：DynamoDBContributorInsights-SKT-[resource\_name]-[creationtimestamp]

### Note

- 您無法使用 CloudWatch 主控台或 API 直接修改或刪除 DynamoDB CloudWatch 參與者深入解析所建立的規則。在資料表或全域次要索引上停用 DynamoDB 的參與 CloudWatch 者深入解析會自動刪除為該資料表或全域次要索引建立的規則。
- 當您將 [GetInsightRuleReport](#) 作業與 DynamoDB 建立的 CloudWatch 參與者見解規則搭配使用時，只會 Maximum 傳回有用的統計 MaxContributorValue 資料。此清單中的其他統計數字不會傳回有意義的值。
- CloudWatch DynamoDB 的貢獻者深入解析有 25 個貢獻者的限制。請求超過 25 個參與者會傳回錯誤。

您可以使用 [DynamoDB 的 CloudWatch 參與者深入解析規則建立 CloudWatch 警示](#)。這可讓您在任何項目超過或達到 ConsumedThroughputUnits 或 ThrottleCount 的特定門檻時收到通知。如需詳細資訊，請參閱針對 [參與者深入解析度量資料設定警示](#)。

## 瞭解 DynamoDB 圖形的 CloudWatch 參與者深入解析

CloudWatch DynamoDB 的參與者深入解析會在 DynamoDB 和 CloudWatch 主控台上顯示兩種類型的圖形：存取次數最多的項目和節流最多的項目。

### 最常存取項目

使用此圖表在資料表或全域次要索引中辨識最常存取的項目。圖表的 Y 軸代表 ConsumedThroughputUnits，而 X 軸代表時間。每個主要 N 金鑰都會以其各自的顏色顯示，而且在 X 軸下方也有圖例說明。

DynamoDB 會使用 ConsumedThroughputUnits 評量金鑰的存取頻率，且此評量數值結合讀取和寫入流量。ConsumedThroughputUnits 的定義如下：

- 佈建：(3 x 已消耗寫入容量單位) + 已消耗讀取容量單位
- 隨需：(3 x 寫入請求單位) + 讀取請求單位

在 DynamoDB 主控台上，圖表中的每個資料點都代表 1 分鐘內 ConsumedThroughputUnits 的最大值。例如，圖表值 180,000 ConsumedThroughputUnits 代表該項目持續存取的頻率為在 1 分鐘內，每 60 秒的每個項目最大輸送量 1,000 個寫入要求單位或 3,000 個讀取要求單位 (3,000 x 60 秒)。也就是說，圖表中呈現的數值代表每 1 分鐘內最高的每分鐘流量。您可以在 CloudWatch 主控台上變更指標的時間粒ConsumedThroughputUnits度 (例如，檢視 5 分鐘量度而非 1 分鐘)。

如果您看到多條緊密聚集的線，沒有任何明顯的異常，就表示您於特定時期內部同項目的工作負載相對來說還算平衡。如果圖表中出現孤立點而不是相連的線，就表示有一個項目僅於短時間內經常存取。

如果您的資料表或全域次要索引排序索引鍵，DynamoDB 就會建立兩個圖表：一個是最常存取的分割區索引鍵，另一個則是最常存取的分割區 + 排序索引鍵配對。您可以在分割區索引鍵 (僅限圖) 中檢視分割區索引鍵層級的流量。您可以在分割區 + 排序索引鍵圖表中檢視項目層級的流量。

### 最常調節項目

使用此圖表在資料表或全域次要索引中辨識最常調節的項目。圖表的 Y 軸代表 ThrottleCount，而 X 軸代表時間。前 N 個按鍵中的每個按鍵都會以自己的顏色顯示，並在 x 軸下方顯示一個圖例。

DynamoDB 會使用 ThrottleCount 來評估調節頻率，也就是 ProvisionedThroughputExceededException、ThrottlingException 和 RequestLimitExceeded 錯誤計數。

不會測量因全域次要索引的寫入容量不足而造成的寫入調節。您可以使用全域次要索引的最常存取項目圖形，識別可能會造成寫入調節的不平衡存取模式。如需詳細資訊，請參閱[全域次要索引的佈建輸送量考量](#)。

在 DynamoDB 主控台上，圖形中的每個資料點代表 1 分鐘內的調節事件計數。

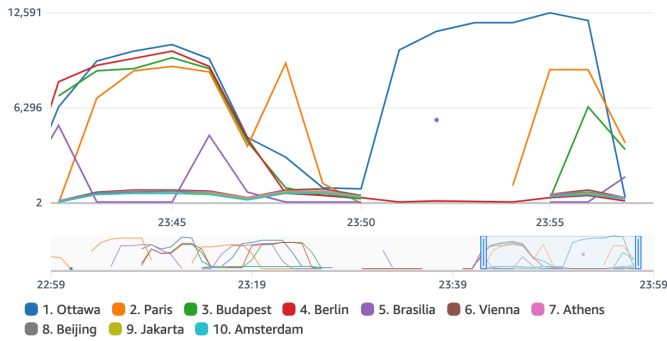
如果此圖表沒有顯示任何資料，就表示您的要求沒有受到調節。如果圖表中出現孤立點而不是相連的線，表示有一個項目僅於短時間內經常調節。

如果您的資料表或全域次要索引有排序索引鍵，DynamoDB 就會建立兩個圖表：一個是最常調節的分割區索引鍵，另一個則是最常調節的分割區 + 排序索引鍵配對。您可以查看分割區索引區中的分割區索引區層級調節計數 (僅圖表)，以及分割區 + 排序索引鍵圖表中的項目層級調節計數。

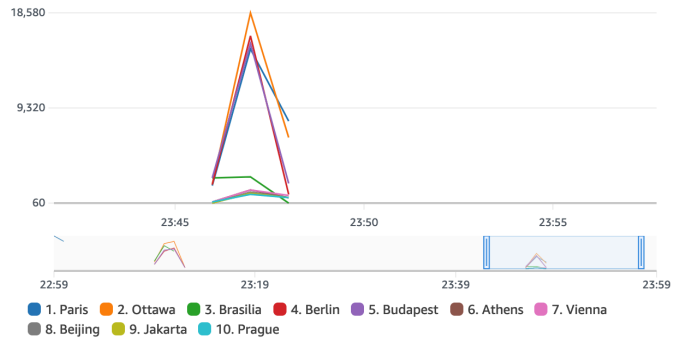
### 報告範例

下列是針對同時有分割區索引鍵和排序索引鍵的資料表產生的報告範例。

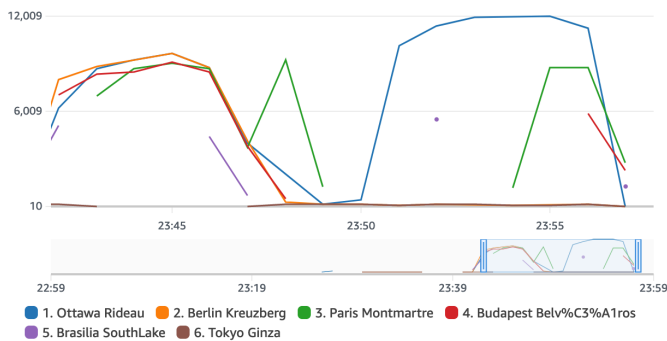
Most accessed keys (partition key)



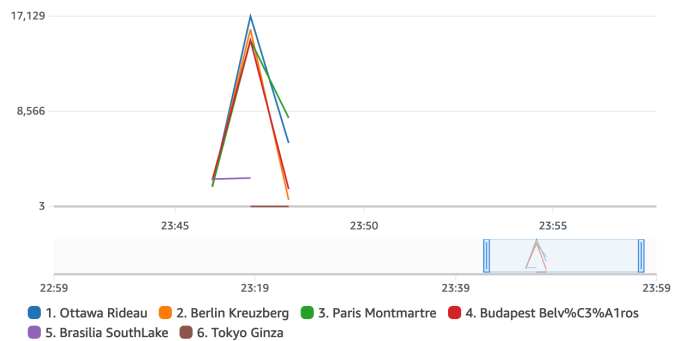
Most throttled keys (partition key)



Most accessed keys (partition and sort keys)



Most throttled keys (partition and sort keys)



## 與其他 DynamoDB 功能互動

下列各節說明 DynamoDB 的參與 CloudWatch 者深入解析如何運作並與 DynamoDB 中的其他數個功能互動。

### 全域資料表

CloudWatch DynamoDB 的參與者深入解析會將全域資料表複本監視為不同的資料表。一個區域中複本的參與者見解圖表可能不會顯示與另一個 AWS 區域相同的模式。這是因為寫入資料於所有全域資料表中的複本複製，但每個複本都可做為區域性讀取流量。

### DynamoDB Accelerator (DAX)

CloudWatch DynamoDB 的參與者深入解析不會顯示 DAX 快取回應。它只會顯示存取資料表或全域次要索引的回應。

#### **i** Note

DynamoDB CCI 不支援 PartiQL 請求。



## 靜態加密

CloudWatch DynamoDB 的參與者深入解析不會影響加密在 DynamoDB 中的運作方式。在中 CloudWatch 發佈的主索引鍵資料會使用 AWS 擁有的金鑰。不過，DynamoDB 也支援 AWS 受管金鑰和客戶管理的金鑰。

CloudWatch DynamoDB 的參與者深入解析圖形會以純文字顯示經常存取項目的分區索引鍵和排序索引鍵 (如果適用)。如果您需要使用金 AWS 鑰管理服務 (KMS) 來加密此資料表的分區金鑰，並使用 AWS 受管金鑰 或客戶受管金鑰排序金鑰資料，則不應為此表啟用 DynamoDB 的參與 CloudWatch 者深入解析。

如果您需要使用 AWS 受管金鑰 或客戶受管金鑰加密主要金鑰資料，則不應為該表啟用 DynamoDB 的參與 CloudWatch 者深入解析。

## 精細定義存取控制

CloudWatch 對於具有精細存取控制 (FGAC) 的資料表，DynamoDB 的參與者深入解析功能不同。換句話說，任何具有適當 CloudWatch 權限的使用者都可以在參與者深入 CloudWatch 解析圖表中檢視受 FGAC 保護的主索引鍵。

如果資料表的主索引鍵包含您不想發佈至的 FGAC 保護資料 CloudWatch，則不應為該表啟用 DynamoDB 的參與 CloudWatch 者深入解析。

## 存取控制

您可以使用 AWS Identity and Access Management (IAM) 限制 DynamoDB 控制平面權限和資料平面許可，來控制對 DynamoDB CloudWatch 參與者見解的存取。CloudWatch 如需詳細資訊，請參閱[使用 IAM 搭配 DynamoDB 的參與 CloudWatch 者深入解析](#)。

## CloudWatch 針對 DynamoDB 計費提供者深入解析

DynamoDB 的 CloudWatch 參與者深入解析的費用會顯示在每月帳單的[CloudWatch](#)區段中。這些費用是根據處理的 DynamoDB 事件數來計算。對於啟用 DynamoDB CloudWatch 參與者深入解析的資料表和全域次要索引，透過[資料平面](#)作業寫入或讀取的每個項目都代表一個事件。

如果資料表或全域次要索引包含排序索引鍵，則讀取或寫入的每個項目都代表兩個事件。這是因為 DynamoDB 識別來自不同時間序列的首位參與者：一個僅用於分割區，另一個用於分割區索引鍵和排序索引鍵對。

例如，假設您的應用程式要執行下列 DynamoDB 操作：GetItem、PutItem 和放置 5 個項目的 BatchWriteItem

- 如果您的資料表或全域次要索引只有分割區索引鍵，則會產生 7 個事件 (1 個 GetItem、1 個 PutItem 和 5 個 BatchWriteItem)。
- 如果您的資料表或全域次要索引有分割區索引鍵和排序索引鍵，則會產生 14 個事件 (2 個 GetItem、2 個 PutItem 和 10 個 BatchWriteItem)。
- 無論傳回的項目數為何，Query 操作一律產生 1 個事件。

與其他 DynamoDB 功能不同，DynamoDB 的參與 CloudWatch 者深入解析計費不會因下列情況而有所不同：

- [容量模式](#) (佈建與隨需)
- 是否執行讀取或寫入請求
- 讀取或寫入項目的大小 (KB)

## 開始使用 DynamoDB 的 CloudWatch 參與者深入解析

本節說明如何搭配 Amazon DynamoDB 主控台或 () 使用亞馬遜參與 CloudWatch 者深入解析。AWS Command Line Interface AWS CLI

在下列範例中，您會用到 [DynamoDB 入門](#) 教學課程中定義的資料表。

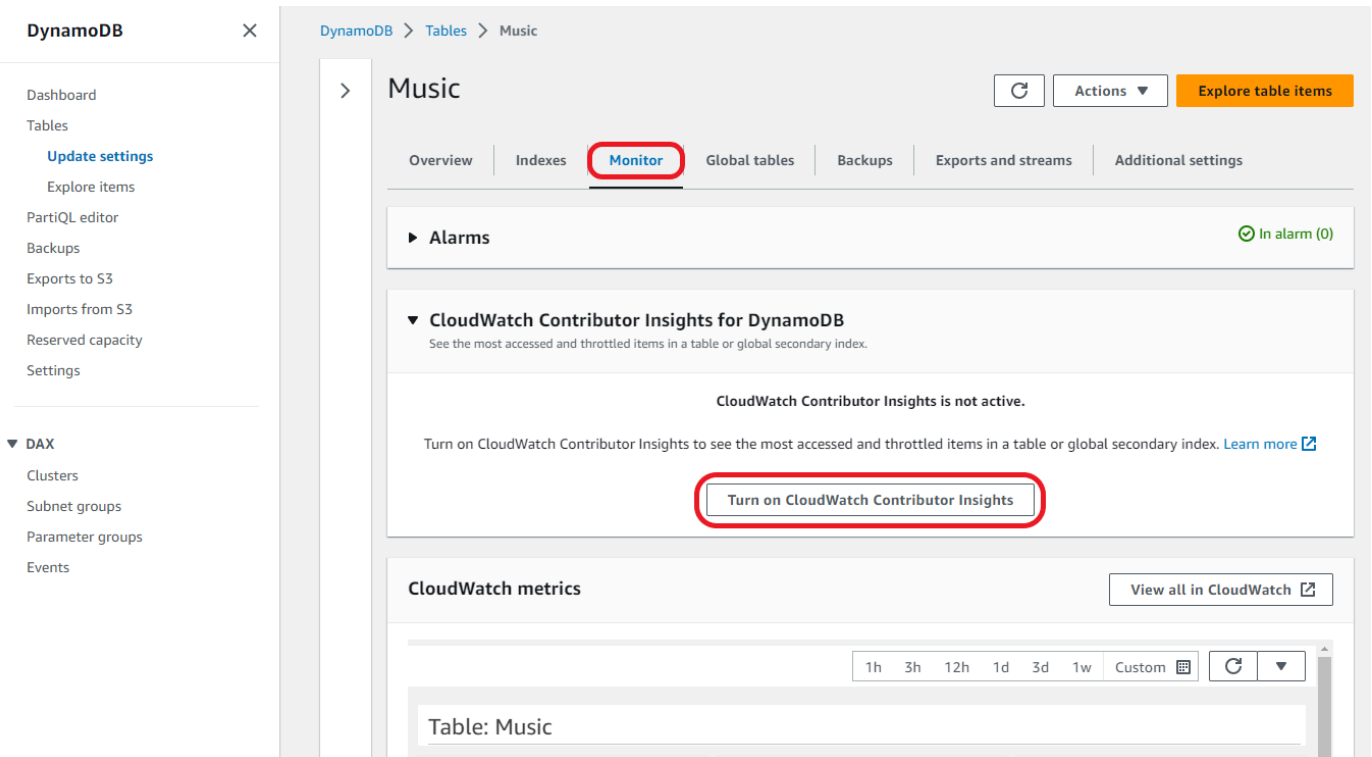
### 主題

- [使用貢獻者見解 \(主控台\)](#)
- [使用 Contributor Insights \(AWS CLI\)](#)

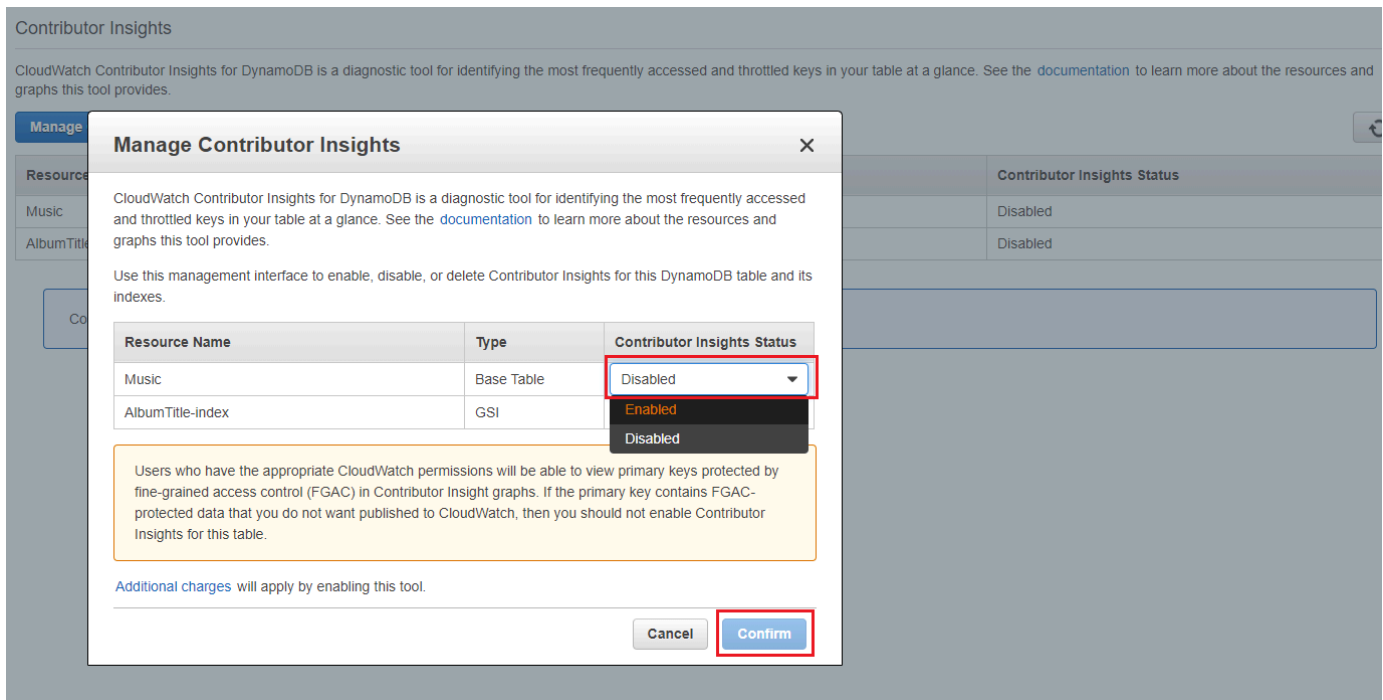
### 使用貢獻者見解 (主控台)

若要在主控台中使用貢獻者深入解析

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇 Music 資料表。
4. 選擇 監控 索引標籤。
5. 選擇 [開啟貢獻者見解 CloudWatch 貢獻者見解]。

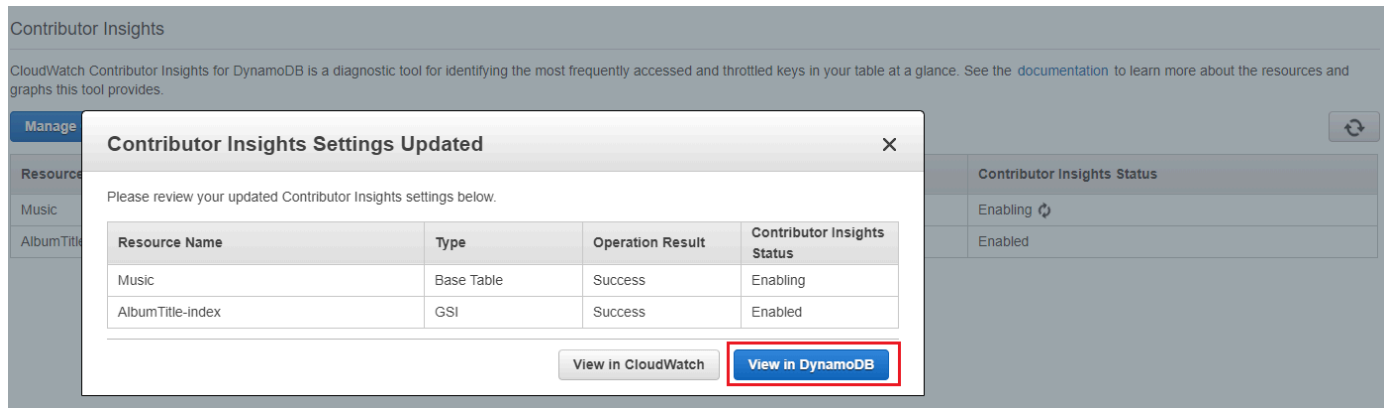


6. 進入 Manage Contributor Insights (管理 Contributor Insights) 對話方塊，在 Contributor Insights Status (Contributor Insights 狀態) 下，為 Music 基礎資料表和 AlbumTitle-index 全域次要索引選擇 Enabled (啟用)。然後選擇 Confirm (確認)。

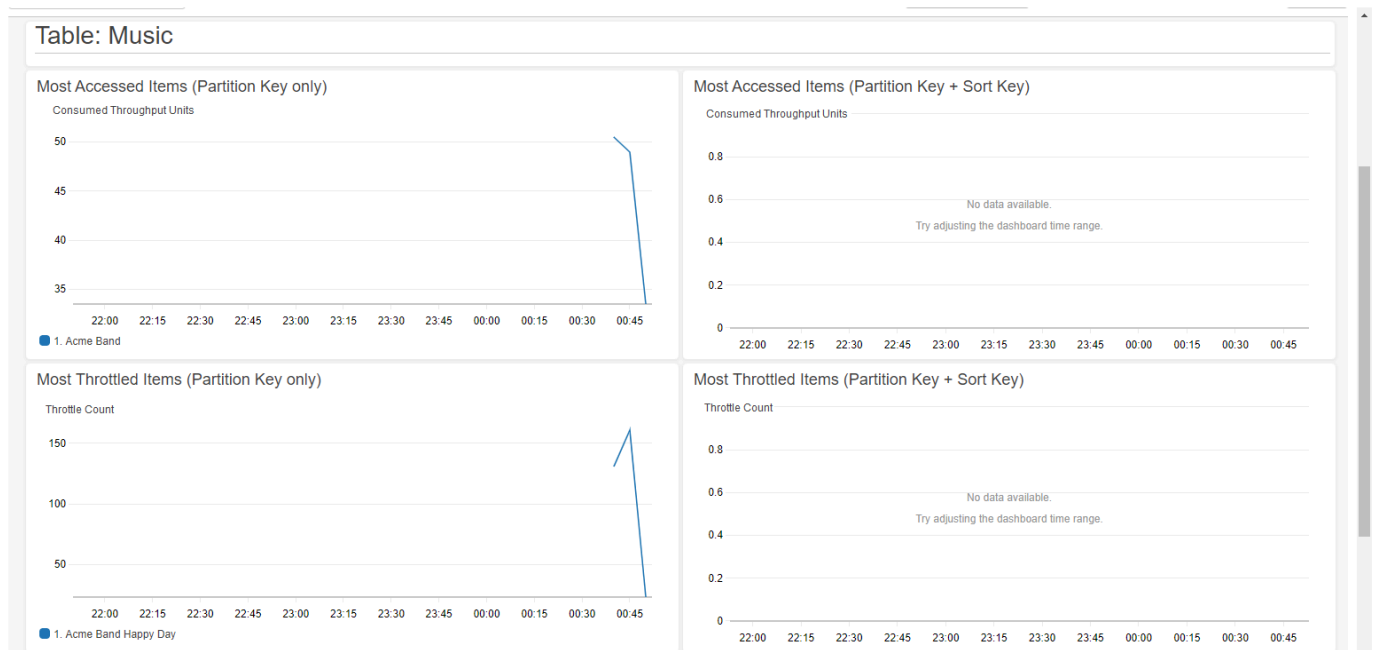


如果操作失敗，請參閱 Amazon DynamoDB API 參考 [DescribeContributorInsightsFailureException](#) 中的可能原因。

## 7. 選擇 View in DynamoDB (在 DynamoDB 中檢視)。



## 8. Contributor Insights 圖表現在可以在 Music 資料表的 Contributor Insights 索引標籤找到。



## 建立 CloudWatch 鬧鐘

請按照以下步驟創建 CloudWatch 警報，並在任何分區密鑰消耗超過 50,000 時收到通知 [ConsumedThroughputUnits](#)。

1. 請登入 AWS Management Console 並開啟 CloudWatch 主控台，網址為 <https://console.aws.amazon.com/cloudwatch/>
2. 在主控台左側的導覽窗格中，選擇 Contributor Insights。

3. 選 DynamoDB ContributorInsights 樂規則。
4. 選擇 Actions (動作) 下拉式功能表。
5. 選擇 View in metrics (在指標中檢視)。
6. 選擇 Max Contributor Value (作者值上限)。

### Note

只有 Max Contributor Value 和 Maximum 會傳回有用的統計數字。此清單中的其他統計數字不會傳回有意義的值。

The screenshot shows the AWS IAM console interface. In the left-hand navigation pane, the 'Contributor Insights' link is highlighted with a red box. In the main content area, the 'Actions' dropdown menu is open, and the 'View in metrics' and 'Max Contributor Value' options are highlighted with red boxes. The background shows a graph titled 'DynamoDBContributorInsights-PKC-Music-1580235665872' with a 'No data available' message.

7. 在 Actions (動作) 欄中，選擇 Create Alarm (建立警示)。

The screenshot shows the AWS IAM console interface. In the main content area, the 'Actions' dropdown menu is open, and the 'Create alarm' option is highlighted with a red box. The background shows a graph titled 'Untitled graph' with a 'DynamoDBContributorInsights-PKC-Music-1580235665872 MaxContributorValue' metric selected. The 'Math expression' field is set to 'DynamoDBContributorInsights-PKC-Music-1580235665872 MaxContrib...'.

8. 輸入值 50000 做為閾值，然後選擇 Next (下一步)。

The screenshot shows the AWS CloudWatch console interface for configuring an alarm. The 'Conditions' section is expanded, showing the following configuration:

- Threshold type:**  Static (Use a value as a threshold)
- Whenever DynamoDBContributorInsights-PKC-Music-1587490256272 MaxContributorValue is...**
- Define the alarm condition:**  Greater (> threshold)
- than...** Define the threshold value:  (Must be a number)
- Additional configuration:** (collapse arrow)

The 'Next' button is highlighted in orange.

9. 如需如何設定 [CloudWatch 警示通知的詳細資訊](#)，請參閱使用 [Amazon 警示](#)。

## 使用 Contributor Insights (AWS CLI)

若要在中使用貢獻者見解 AWS CLI

1. 在基底資料表 Music 上啟用 DynamoDB 的 CloudWatch 參與者深入解析。

```
aws dynamodb update-contributor-insights --table-name Music --contributor-insights-action=ENABLE
```

2. 在 AlbumTitle-index 全域次要索引上啟用 DynamoDB 專用 Contributor Insights。

```
aws dynamodb update-contributor-insights --table-name Music --index-name AlbumTitle-index --contributor-insights-action=ENABLE
```

3. 取得 Music 資料表及其所有索引的狀態和規則。

```
aws dynamodb describe-contributor-insights --table-name Music
```

4. 在 AlbumTitle-index 全域次要索引上停用 DynamoDB 的 CloudWatch 參與者深入解析。

```
aws dynamodb update-contributor-insights --table-name Music --index-name
AlbumTitle-index --contributor-insights-action=DISABLE
```

## 5. 取得 Music 資料表及其所有索引的狀態。

```
aws dynamodb list-contributor-insights --table-name Music
```

## 將 IAM 與 DynamoDB 的 CloudWatch 貢獻者深入解析搭配使用

當您第一次為 Amazon DynamoDB 啟用亞馬遜 CloudWatch 參與者洞察時，DynamoDB 會自動為您建立 AWS Identity and Access Management (IAM) 服務連結角色。此角色 `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 可讓 DynamoDB 代表您管理 CloudWatch 參與者見解規則。請勿刪除此服務連結角色。如果刪除，您所有管理的角色都不會再於您刪除資料表或全域次要索引時清理。

如需服務連結角色的詳細資訊，請參閱 IAM 使用者指南中的 [使用服務連結角色](#)。

需要以下許可：

- 若要啟用或停用 DynamoDB 的 CloudWatch 參與者深入解析，您必須擁有資料表或索引的 `dynamodb:UpdateContributorInsights` 權限。
- 若要檢視 DynamoDB 圖形的 CloudWatch 參與者深入解析，您必須擁有權限。 `cloudwatch:GetInsightRuleReport`
- 若要針對指定 DynamoDB 資料表或索引描述 DynamoDB 的 CloudWatch 參與者深入解析，您必須擁有權限。 `dynamodb:DescribeContributorInsights`
- 若要針對每個資料表和全域次要索引列出 DynamoDB 狀態的參與 CloudWatch 者深入解析，您必須擁有權限。 `dynamodb:ListContributorInsights`

## 範例：啟用或停用 DynamoDB 的 CloudWatch 參與者深入解析

下列 IAM 政策授予許可，以啟用或停用 DynamoDB 的 CloudWatch 參與者深入解析。

```
{
 "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/*"
 }
]
}

```

對於由 KMS 金鑰加密的資料表，使用者必須擁有 kms:Decrypt 許可，才能更新 Contributor Insights。

```

{
 "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"
]
 }
]
}

```



```
],
 }
]
}
```

## 範例：擷取 CloudWatch 貢獻者見解規則報表

下列 IAM 政策授予擷取 CloudWatch 參與者見解規則報表的權限。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "cloudwatch:GetInsightRuleReport"
],
 "Resource": "arn:aws:cloudwatch:*:*:insight-rule/
DynamoDBContributorInsights*"
 }
]
}
```

## 範例：根據資源選擇性地套用 DynamoDB 權限的 CloudWatch 參與者深入解析

下列 IAM 政策會授予允許 `ListContributorInsights` 和 `DescribeContributorInsights` 動作的許可，並拒絕特定全域次要索引的 `UpdateContributorInsights` 動作。

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

## 使用服務連結角色取得 DynamoDB 的 CloudWatch 參與者深入解析

CloudWatch [DynamoDB 的參與者深入解析會使用 AWS Identity and Access Management \(IAM\) 服務連結角色](#)。服務連結角色是一種獨特的 IAM 角色類型，可直接連結至 DynamoDB 的參與 CloudWatch 者深入解析。DynamoDB 的參與 CloudWatch 者深入解析會預先定義服務連結角色，並包含服務代表您呼叫其他 AWS 服務所需的所有權限。

服務連結角色可讓您更輕鬆地為 DynamoDB 設定參與 CloudWatch 者深入解析，因為您不必手動新增必要的權限。CloudWatch DynamoDB 的參與者深入解析會定義其服務連結角色的權限，除非另有定義，否則只有 DynamoDB 的參與者深入解析可以 CloudWatch 擔任其角色。定義的許可包括信任政策和許可政策，並且該許可政策不能連接到任何其他 IAM 實體。

如需關於支援服務連結角色的其他服務的資訊，請參閱[可搭配 IAM 運作的 AWS 服務](#)，並尋找 Service-Linked Role (服務連結角色) 欄顯示為 Yes (是) 的服務。選擇具有連結的是，以檢視該服務的服務連結角色文件。

### DynamoDB 參與 CloudWatch 者深入解析的服務連結角色權限

CloudWatch DynamoDB 的參與者深入解析會使用名為的服務連結角色。AWSServiceRoleForDynamoDBCloudWatchContributorInsights服務連結角色的目的是讓 Amazon DynamoDB 代表您管理 CloudWatch 針對 DynamoDB 表和全域次要索引建立的 Amazon 參與者見解規則。

AWSServiceRoleForDynamoDBCloudWatchContributorInsights 服務連結角色信任下列服務以擔任角色：

- contributorinsights.dynamodb.amazonaws.com

角色權限原則允許 DynamoDB 的 CloudWatch 參與者深入解析對指定的資源完成下列動作：

- 動作：DynamoDBContributorInsights 上的 Create and manage Insight Rules

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的[服務連結角色許可](#)。

### 為 DynamoDB 的參與 CloudWatch 者深入解析建立服務連結角色

您不需要手動建立一個服務連結角色。當您在 AWS Management Console、或 AWS API 中啟用參與者深入解析時 AWS CLI，DynamoDB 的 CloudWatch 參與者深入解析會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您啟用參與者深入解析時，DynamoDB 的參與 CloudWatch 者深入解析會再次為您建立服務連結角色。

### 為 DynamoDB 的參與 CloudWatch 者深入解析編輯服務連結角色

CloudWatch DynamoDB 的參與者深入解析不允許您編輯服務連結 `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱 IAM 使用者指南中的[編輯服務連結角色](#)。

### 刪除 DynamoDB 參與 CloudWatch 者深入解析的服務連結角色

您不需要手動刪除 `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 角色。當您停用 AWS Management Console、或 AWS API 中的參與者深入解析時 AWS CLI，DynamoDB 的 CloudWatch 參與者深入解析會清除資源。

您也可以使用 IAM 主控台 AWS CLI 或 AWS API 手動刪除服務連結角色。若要執行此操作，您必須先手動清除服務連結角色的資源，然後才能手動刪除它。

#### Note

如果 DynamoDB 服務的 CloudWatch 參與者深入解析在您嘗試刪除資源時正在使用該角色，則刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

### 使用 IAM 手動刪除服務連結角色

#### 使用 IAM 主控台或 AWS API 刪

除 `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` 服務連結角色。AWS CLI 如需詳細資訊，請參閱《IAM 使用者指南》中的[刪除服務連結角色](#)。

# 使用 DynamoDB 進行設計和架構的最佳實務

使用本節快速找到使用 Amazon DynamoDB 時能將效能發揮到最高且輸送量成本降至最低的建議。

## 主題

- [DynamoDB 的 NoSQL 設計](#)
  - [使用刪除保護來保護資料表](#)
  - [使用 DynamoDB Well-Architected Lens 來最佳化您的 DynamoDB 工作負載](#)
  - [有效設計和使用分區索引鍵的最佳實務](#)
  - [使用排序索引鍵以組織資料的最佳實務](#)
  - [使用 DynamoDB 中次要索引的最佳實務](#)
  - [儲存大型項目和屬性的最佳實務](#)
  - [在 DynamoDB 中處理時間序列資料的最佳實務](#)
  - [管理 many-to-many 關係的最佳做法](#)
  - [實作混合資料庫系統的最佳實務](#)
  - [在 DynamoDB 中製作關聯式資料模型的最佳實務](#)
  - [查詢和掃描資料的最佳實務](#)
  - [DynamoDB 資料表設計的最佳實務](#)
  - [DynamoDB 全域資料表設計的最佳實務](#)
  - [在 DynamoDB 中管理控制平面的最佳實務](#)
  - [瞭解 AWS 帳單和用量報告的最佳做法](#)
  - [切換容量模式時的考量](#)
- 
- [使用 Amazon DynamoDB 時 AWS PrivateLink 的注意事項](#)

## DynamoDB 的 NoSQL 設計

如 Amazon DynamoDB 這類的 NoSQL 資料庫會使用備用模型來管理資料，例如鍵值的配對或文件儲存。當您從關聯式資料庫管理系統切換至如 DynamoDB 這類的 NoSQL 資料庫時，請務必了解主要差異和特定設計方法。

## 主題

- [關聯式資料設計與 NoSQL 間的差異](#)
- [NoSQL 設計的兩個主要概念](#)
- [NoSQL 設計方法](#)
- [DynamoDB 專用 NoSQL Workbench](#)

## 關聯式資料設計與 NoSQL 間的差異

關聯式資料庫管理系統 (RDBMS) 和 NoSQL 資料庫各有優劣：

- RDBMS 可以彈性地查詢資料，但查詢相對昂貴，而且在高流量的狀況下擴展不易 (請參閱 [在 DynamoDB 中製作關聯式資料模型的第一步](#))。
- 在如 DynamoDB 這類的 NoSQL 資料庫中，可以少數方式有效地查詢資料，但在外部的查詢就非常昂貴而且速度緩慢。

這些差異讓兩種系統之間的資料庫設計不同：

- 在 RDBMS 中，您會針對靈活性而進行設計，而不需擔心實行詳細資訊或效能。查詢最佳化通常不會影響結構描述設計，但標準化相當重要。
- 在 DynamoDB 中，您會特別設計結構描述，盡可能以最快最便宜的方式進行常用與重要的查詢。系統會打造您的資料結構，以符合企業使用案例的特定要求。

## NoSQL 設計的兩個主要概念

NoSQL 設計思維與 RDBMS 設計不同。針對 RDBMS，您可以直接建立標準化的資料模型，而不需考量存取模式。您可以在稍後有新問題與查詢要求時擴展此模型。您可以將每種資料整理至其資料表。

NoSQL 設計的不同之處

- 相反地，針對 DynamoDB，您不應開始設計結構描述，除非您知道其將需要回答的問題。必須事先了解企業問題和應用程式使用案例。
- 您在 DynamoDB 應用程式中維護的資料表應越少越好。擁有較少的資料表可讓事物更具擴展性，需要較少的許可管理，並減少 DynamoDB 應用程式的額外負荷。它還可以幫助降低整體備份成本。

## NoSQL 設計方法

設計 DynamoDB 應用程式的第一步即是辨識系統必須滿足的特定查詢模式。

特別是，請務必了解應用程式存取模式的三種基本屬性，再開始進行：

- **資料大小：**了解資料一次儲存與要求的方式，是協助判斷分割資料的最有效方法。
- **資料形狀：**NoSQL 資料庫會組織資料，而非在資料處理時重新改造資料 (如 RDBMS 系統)，如此其在資料庫中的狀態就會與將受到查詢的項目相對應。此為提升速度與可擴展性的關鍵因素。
- **資料速度：**DynamoDB 會隨著增加程序查詢可用的實體分割區數與有效地在這些分割區間散佈資料來擴展。事先了解尖峰查詢負載，將可能有助於判斷如何分割資料以有效使用輸入/輸出容量。

在您識別特定查詢要求後，您可以根據管理效能的一般原則來組織資料：

- **將相關的資料保持在一起。** 20 年前針對路由表最佳化的研究發現，「參照本地性」是提升回應時間的最重要的單一因素，也就是將相關資料放在同一個位置。這在 NoSQL 系統也同樣適用，將相關資料保持在鄰近位置對成本與效能有重大影響。您應盡可能將 NoSQL 系統中的相關項目放置在靠近的位置，而非在多個資料表之間分配相關的資料項目。

作為一般規則，您在 DynamoDB 應用程式中維護的資料表應越少越好。

例外狀況包含大量時間序列資料涉及其中的案例，或存取模式極為不同的資料集。含反轉索引的單一資料表通常可以啟用簡單查詢，來建立並擷取您應用程式所需的複雜階層資料結構。

- **使用排序。** 若相關項目的索引鍵設計為能一起排序，則可以一起分組，以更有效地進行查詢。此為重要的 NoSQL 設計策略。
- **發佈佇列。** 亦請注意不要針對資料庫的一部分集中進行大量查詢 (這會超過輸入/輸出容量)。而您應盡可能將資料索引鍵平均分配到這些分割區，避免「熱點」。
- **使用全域次要索引。** 透過建立特定全域次要索引，您可以啟用主要資料表可支援的不同查詢，這能維持速度且費用相對便宜。

這些一般原則會轉換為常見設計模式，讓您可以在 DynamoDB 使用它們來有效打造資料模型。

## DynamoDB 專用 NoSQL Workbench

[DynamoDB 專用 NoSQL Workbench](#) 是跨平台用戶端 GUI 應用程式，可用於現代資料庫開發和操作。適用於 Windows、macOS 和 Linux。NoSQL Workbench 是視覺化開發工具，提供了資料模型建立、資料視覺化、範例資料產生和查詢開發功能，協助您設計、建立、查詢及管理 DynamoDB 資料表。借助 DynamoDB 專用 NoSQL Workbench，您可以使用滿足您應用程式資料存取模式的現有資料模型，來建置新的資料模型或設計模型。您也可以在此時，匯入及匯出設計好的資料模型。如需更多資訊，請參閱[使用 NoSQL Workbench 建立資料模型](#)

## 使用刪除保護來保護資料表

刪除保護可防止您的資料表被意外刪除。本節說明使用刪除保護的一些最佳實務。

- 對於所有使用中的生產資料表，最佳實務是開啟刪除保護設定，保護這些資料表以免遭到意外刪除。這同樣適用於全域複本。
- 提供應用程式開發使用案例時，如果資料表管理工作流程包括經常刪除和重新建立開發和臨時資料表，則可以關閉刪除保護設定。這會允許獲授權的 IAM 主體故意刪除此類資料表。

如需刪除保護的詳細資訊，請參閱 [使用刪除保護](#)。

## 使用 DynamoDB Well-Architected Lens 來最佳化您的 DynamoDB 工作負載

本節描述 Amazon DynamoDB Well-Architected Lens，其收集了用於設計架構良好 DynamoDB 工作負載的設計原則和指引。

### 最佳化 DynamoDB 資料表上的成本

本節涵蓋如何為現有 DynamoDB 資料表最佳化成本的最佳實務。您應查看以下策略，了解哪一項成本最佳化策略最適合您的需求，並反覆採用這些策略。每個策略都會概述可能會影響您成本的原因、應查看的跡象，以及如何降低成本的規範式引導。

#### 主題

- [在資料表層級評估您的成本](#)
- [評估您的資料表容量模式](#)
- [評估您資料表的自動擴展設定](#)
- [評估您的資料表類別選擇](#)
- [識別您未使用的資源](#)
- [評估您的資料表用量模式](#)
- [評估您的 Streams 使用量](#)
- [評估您是否具有適當大小的佈建容量](#)



## 在資料表層級評估您的成本

在中找到的 Cost Explorer 工具 AWS Management Console 可讓您查看依類型劃分的成本，例如讀取、寫入、儲存和備份費用。您還可以查看依期間 (例如月份或日期) 彙總的這些成本。

管理員面臨的挑戰之一，在於只需要檢閱個別特定資料表成本的情況。部分資料可透過 DynamoDB 主控台或呼叫 DescribeTable API 取得，但 Cost Explorer 依預設不允許按照特定資料表相關的成本進行篩選或分組。本節將說明如何使用標記功能，在 Cost Explorer 中執行個別資料表的成本分析。

### 主題

- [如何檢視單一 DynamoDB 資料表的成本](#)
- [Cost Explorer 的預設檢視](#)
- [如何在 Cost Explorer 中使用和套用資料表標籤](#)

### 如何檢視單一 DynamoDB 資料表的成本

Amazon DynamoDB AWS Management Console 和 DescribeTable API 都會顯示單一表格的相關資訊，包括主索引鍵結構描述、資料表上的任何索引，以及表格以及任何索引的大小和項目計數。將資料表的大小加上索引的大小，即可計算資料表的每月儲存成本。例如，在 us-east-1 區域中，每 GB 為 0.25 美元。

如果資料表處於佈建容量模式，也會傳回目前的 RCU 和 WCU 設定值。這些資訊可用於計算資料表目前的讀取和寫入成本，但這些成本可能變動，尤其是在設定了資料表的 Auto Scaling 功能後。

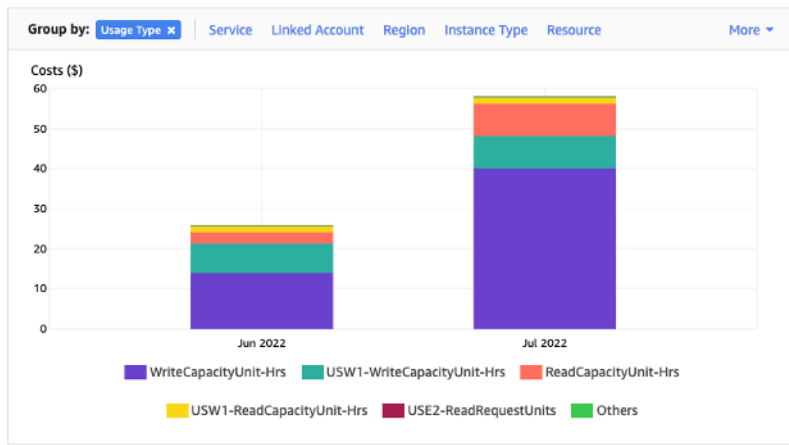
#### Note

如果資料表處於隨需容量模式，則 DescribeTable 對於估計輸送量成本將沒有幫助，因為這些成本是根據任一期間內的實際用量而非佈建用量進行計費。

### Cost Explorer 的預設檢視

Cost Explorer 的預設檢視會提供圖表以顯示耗用資源 (例如輸送量和儲存體) 的成本。您可以選擇按期間分組成本，例如按月或按日總計。儲存、讀取、寫入和其他功能的成本也可以進行細分和比較。





## 如何在 Cost Explorer 中使用和套用資料表標籤

根據預設，Cost Explorer 不會提供任何一個特定資料表的成本摘要，因為它會將多個資料表的成本合併為一項總數。不過，您可以使用 [AWS 資源標記](#)，以中繼資料標籤來識別各個資料表。標籤是一種鍵值對，可用於各種用途，例如找出所有屬於特定專案或部門的資源。在此範例中，我們假設您有一個名為 MyTable。

1. 使用表格名稱的鍵和的值來設置一個標籤。MyTable
2. [在 Cost Explorer 中啟用此標籤](#)，然後篩選標籤值，以深入了解各個資料表的成本。

### Note

標籤可能需要一到兩天的時間才會開始出現在 Cost Explorer 中

您可以在主控台中自行設定中繼資料標籤，也可以透過 AWS CLI 或 AWS SDK 等自動化設定中繼資料標籤。請考慮在組織的新標籤建立程序中要求設定 table\_name 標籤。對於現有的資料表，可使用一項 Python 公用程式來尋找這些標記，並將這些標記套用到您帳戶中特定區域內全部現有的資料表。GitHub 如需詳細資訊，請參閱上的 [同名表格標記器](#)。

## 評估您的資料表容量模式

本節概述如何為 DynamoDB 資料表選取適當的容量模式。每種模式都經過調整，以滿足不同工作負載在回應輸送量變化以及該用量計費方式方面的需求。在制定決策時，必須均衡考量這些因素。

### 主題

- [有哪些可用的資料表容量模式](#)

- [何時選取隨需容量模式](#)
- [何時選取佈建容量模式](#)
- [選擇資料表容量模式時應考慮的其他因素](#)

有哪些可用的資料表容量模式

建立 DynamoDB 資料表時，您必須選取隨需或佈建容量模式。每 24 小時，您可以將模式切換為讀取/寫入容量模式一次。唯一的例外是，如果您將佈建模式表格切換為隨選模式：您可以在相同的 24 小時期間內切換回已佈建模式。

**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**

### 隨需容量模式

隨需容量模式的設計是為了讓使用者不必再規劃或佈建 DynamoDB 資料表的容量。在此模式下，資料表不需擴充或縮減任何資源，就能立即容納資料表收到的請求 (最多可達到先前資料表尖峰輸送量的兩倍)。

隨需資料表是按資料表的實際請求數量計費，因此只需依照用量而非佈建項目付費。

### 佈建容量模式

佈建的容量模式是較傳統的模型，您可以在其中定義表格可用於請求的容量，無論是直接或藉由 auto 擴展的協助。由於會在任何指定時間為資料表佈建特定容量，因此是根據佈建的容量而非請求數目進行計費。超過配置的容量也可能導致資料表拒絕請求，並降低應用程式使用者的體驗。

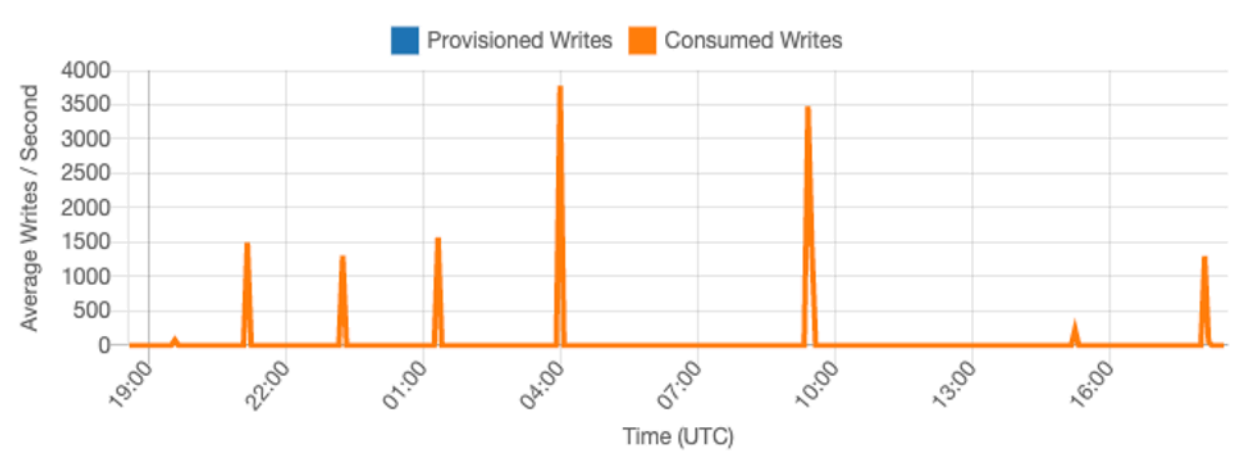
佈建容量模式必須避免過度或不足的情況，採均衡方式對資料表進行佈建，以維持低度限流並妥善控管成本。

### 何時選取隨需容量模式

在進行成本最佳化時，如果您的工作負載類似下圖所示，則隨需模式將是最佳選擇。

下列因素會造成此類型的工作負載：

- 無法預測的請求時機 (導致流量尖峰)
- 變動的請求量 (由批次工作負載導致)
- 在特定的一小時內降至零或低於峰值的 18% (由開發或測試環境導致)



對於具有上述因素的工作負載，使用 auto 擴展來維持表格上足夠的容量以回應流量尖峰，可能會導致表格佈建過度，並且成本超過必要，或者表格正在佈建和請求受到不必要的限制。

由於隨需資料表是針 pay-per-request 對讀取和寫入請求計費，因此您只需按使用量付費，輕鬆平衡成本和效能。您也可以選擇為個別隨選資料表和全域次要索引設定每秒最大讀取或寫入 (或兩者) 輸送量，以協助保持成本和用量限制。如需詳細資訊，請參閱 [隨選表格的最大輸送量](#)。您應該定期評估隨需資料表，以確認工作負載仍具有上述因素。如果工作負載達到穩定，請考慮變更為佈建模式以進一步最佳化成本。

### 何時選取佈建容量模式

對於佈建容量模式而言，理想的工作負載必須具有更可預測的用量模式，如下圖所示。

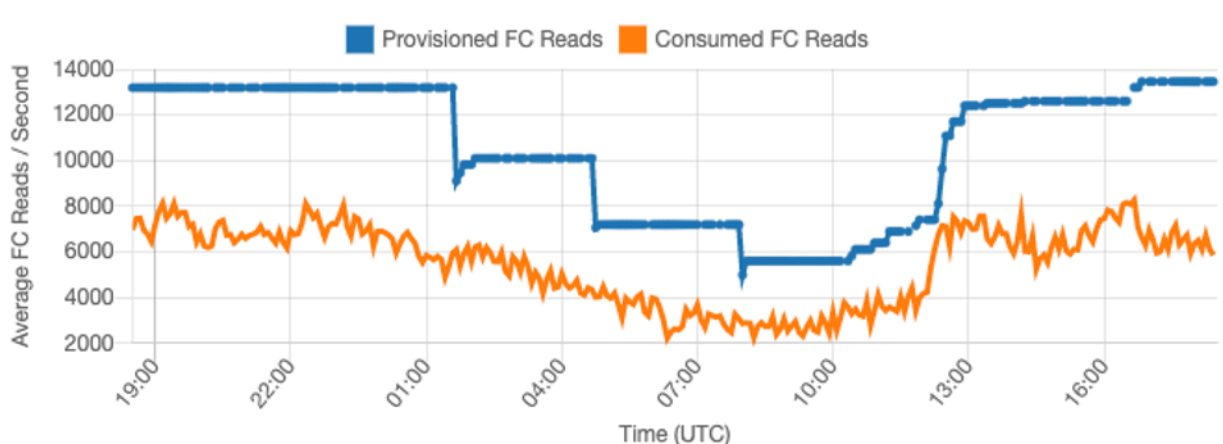
#### Note

我們建議您在對已佈建容量採取動作之前，先檢閱精細的期間 (例如 14 天或 24 小時) 的指標。

下列因素會造成此類型的工作負載：

- 特定一小時或一天內的可預測/周期性流量

• 有限度的短期突發流量



由於在特定一小時或一天內的流量較為穩定，因此較能依據實際使用容量的近似值來設定資料表的佈建容量。基本上，佈建容量資料表的成本最佳化必須藉由實際操作，盡量使佈建容量 (藍線) 接近實際使用容量 (橘線)，同時不增加資料表上的 `ThrottledRequests`。兩條線之間的空間表示浪費的容量，但同時也是預防因限流而影響使用者體驗的保障。

DynamoDB 為佈建容量資料表提供自動調整規模功能，將代表您自動平衡佈建容量。如此可讓您追蹤全天的使用容量，並根據少數幾項變數來設定資料表的容量。

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](#)

## 容量單位下限

您可以設定資料表的容量下限，藉此限制限流的程度，但這麼做不會降低資料表的成本。如果資料表會在低用量期間之後出現突發高用量，則可以設定用量下限以預防自動調整規模功能將資料表容量設定得過低。

## 容量單位上限

您可以設定資料表的容量上限，以避免資料表的規模調整到高於預期的程度。如果不需要進行大規模負載測試，請考慮為開發或測試資料表套用上限。您可以為任何資料表設定上限，但是在生產環境中使用時，請務必定期依據資料表基準評估此設定，以預防意外限流的狀況。

## 目標使用率

對於佈建容量資料表，設定資料表的目標使用率是實現成本最佳化的主要方法。對此設定較低的百分比值，將會增加資料表過度佈建程度而提高成本，但同時降低限流的風險。對此設定較高的百分比值，將會降低資料表過度佈建程度，但同時提高限流的風險。

## 選擇資料表容量模式時應考慮的其他因素

在兩種模式之間進行抉擇時，還有一些其他因素值得納入考量。

## 預留容量

對於佈建容量資料表，DynamoDB 提供預留容量的購買功能，適用於讀取和寫入容量 (目前不適用於複寫寫入容量單位 (rWCU) 和標準 - IA 資料表)。如果您選擇購買此類預留容量，則可以大幅降低資料表成本。

在兩種資料表模式之間進行抉擇時，請考慮這項額外的折扣因素對資料表成本的影響程度。在許多情況下，即使是較難預測的工作負載，在過度佈建的具有預留容量的佈建容量資料表上執行也會更為便宜。

## 改善工作負載的可預測性

在某些情況下，工作負載可能顯得同時具有可預測和無法預測的模式。雖然隨需資料表可以輕鬆支援這種情況，但若可改善無法預測的工作負載模式，可能會有助於提高成本效益。

造成這些模式的最常見原因之一是批次匯入。這種類型的流量可能經常超過資料表的基準容量，導致執行期間發生限流。若要在佈建容量資料表上執行這類工作負載，請考慮下列選項：

- 如果批次發生在排程時間，則可排程在其執行之前增加自動調整規模的容量。

- 如果批次是隨機發生，請考慮嘗試延長執行時間，而不是盡速執行
- 為匯入增加期間，匯入的速度開始很小，但會在幾分鐘內緩慢增加，直到 auto Scaling 有機會開始調整資料表容量

## 評估您資料表的自動擴展設定

本節概述如何評估 DynamoDB 資料表上的自動擴展設定。[Amazon DynamoDB 自動擴展](#)是一款功能，可根據應用程式流量和目標使用率指標，管理資料表和全域次要索引 (GSI) 輸送量。如此可確保您的資料表或 GSI 具有應用程式模式所需的容量。

AWS auto Scaling 服務會監控您目前的資料表使用率，並將其與目標使用率值進行比較：TargetValue。若需增加或減少分配的容量，您會收到通知。

### 主題

- [瞭解您的自動擴展設定](#)
- [如何識別目標使用率低的資料表 \(<=50%\)](#)
- [如何處理具有季節性差異的工作負載](#)
- [如何處理具有未知模式的尖峰工作負載](#)
- [如何處理具有連結應用程式的工作負載](#)

### 瞭解您的自動擴展設定

您的營運團隊需定義目標使用率的正確值、初始步驟和最終值。這可讓您根據歷史應用程式使用情況來定義值，用來觸發 AWS 自動擴展政策。使用率目標是套用自動擴展規則之前的一段時間內，需要達成的總容量百分比。

當您設定高使用率目標 (目標約 90%) 時，表示您的流量須在一段時間內高於 90%，才會啟用自動擴展。除非您的應用程式狀態穩定，且不會接收到流量尖峰，否則不應使用高使用率目標。

當您設定非常低使用率 (目標小於 50%) 時，表示您的應用程式須達到佈建容量的 50%，才會觸發自動擴展政策。除非您的應用程式流量成長快速，否則這通常會變成未使用容量和資源浪費。

### 如何識別目標使用率低的資料表 (<=50%)

您可以使用 AWS CLI 或 AWS Management Console 來監視和識別 DynamoDB TargetValues 資源中的 auto 擴展政策：

## AWS CLI

1. 執行下列命令，傳回完整的資源清單：

```
aws application-autoscaling describe-scaling-policies --service-namespace dynamodb
```

此命令將傳回發給所有 DynamoDB 資源的自動擴展政策完整清單。若您只想從特定資料表擷取資源，您可以新增 `-resource-id` parameter。例如：

```
aws application-autoscaling describe-scaling-policies --service-namespace dynamodb --resource-id "table/<table-name>"
```

2. 執行下列命令，僅傳回特定 GSI 的自動擴展政策

```
aws application-autoscaling describe-scaling-policies --service-namespace dynamodb --resource-id "table/<table-name>/index/<gsi-name>"
```

我們對自動擴展政策感興趣的值於下面強調顯示。我們希望確保目標值大於 50%，以避免過度佈建。您應該會獲得類似以下的結果：

```
{
 "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. 登入 AWS Management Console 並瀏覽至「[開始使用](#)」中的 [CloudWatch 服務頁面 AWS Management Console](#)。如有必要，請選取適當的 AWS 區域。
2. 在左側導覽列中，選取 Tables (資料表)。在 Tables (資料表) 頁面上，選取資料表 Name (名稱)。
3. 在 Additional Settings (其他設定) 標籤的 Table Details (資料表詳細資訊) 上，檢閱資料表的自動擴展設定。



Overview | Indexes | Monitor | Global tables | Backups | Exports and streams | **Additional settings**

### 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

針對索引，請展開 Index Capacity (索引容量) 標籤，檢閱索引的自動擴展設定。

### 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

若您的目標使用率值小於或等於 50%，您應探索資料表使用率指標，查看指標是否 [佈建不足或過度佈建](#)。

### 如何處理具有季節性差異的工作負載

請考慮下列情況：您的應用程式大部分時間都以最小平均值運作，但使用率目標很低，因此應用程式可以快速回應特定時間發生的事件，且您擁有足夠容量避免受到限流。應用程式在正常辦公時間 (上午 9 點至下午 5 點) 非常忙碌，但下班時間僅在基礎層級運作時，這種情況就很常見。由於部分使用者在上午 9 點前開始連線，因此應用程式會使用此低閾值快速提升，以便在尖峰時段達到所需容量。

此情況可能如下所示：

- 下午 5 點至上午 9 點之間，ConsumedWriteCapacity 單位停留在 90 和 100 之間
- 使用者在上午 9 點前開始連線到應用程式，且容量單位大幅增加 (您看到的最大值為 1500 WCU)
- 平均而言，在工作期間，應用程式使用量介於 800 到 1200

若上一個情況適用於您，請考慮使用 [排程的自動擴展](#)，您的資料表仍可設定應用程式自動擴展規則，但因目標使用率較低，只會在您需要的特定間隔佈建額外容量。

您可 AWS CLI 以使用執行下列步驟來建立排定的 auto 調整規則，該規則將根據一天中的時間和星期幾執行。

1. 將您的 DynamoDB 資料表或 GSI 向 Application Auto Scaling 註冊為可擴展的目標。可擴展的目標是 Application Auto Scaling 可橫向擴展和縮減的資源。

```
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. 根據您的需求設定排定的動作。

我們需要兩個規則來處理此情況：一個用於縱向擴展，另一個用於縮減規模。第一個規則縱向擴展排程的動作：

```
aws application-autoscaling put-scheduled-action \
 --service-namespace dynamodb \
 --scalable-dimension dynamodb:table:WriteCapacityUnits \
 --resource-id table/<table-name>
```

```
--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"
```

第二個規則縮減規模排程的動作：

```
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. 執行以下命令，驗證這兩個規則皆已啟用。

```
aws application-autoscaling describe-scheduled-actions --service-namespace dynamodb
```

您應該會取得如下結果：

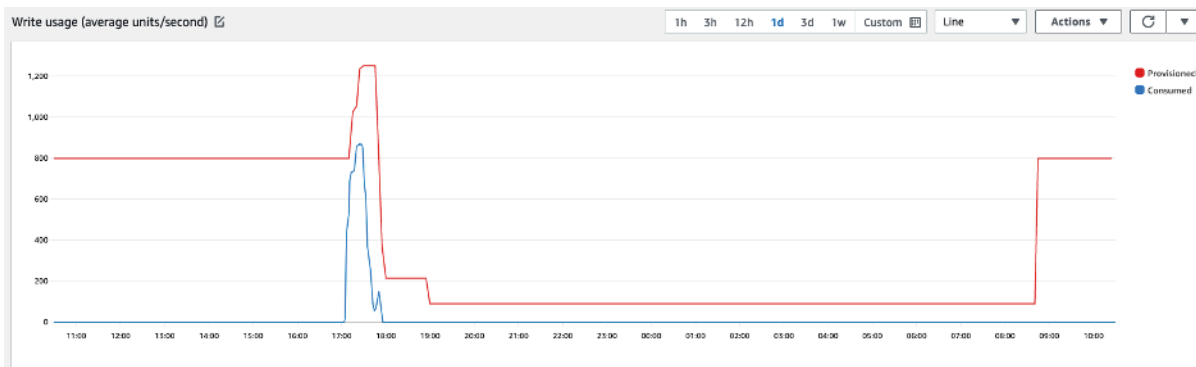
```
{
 "ScheduledActions": [
 {
 "ScheduledActionName": "my-5-8-scheduled-down-action",
 "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-5-8-scheduled-down-action",
 "ServiceNamespace": "dynamodb",
 "Schedule": "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"
 },
 {
```

```

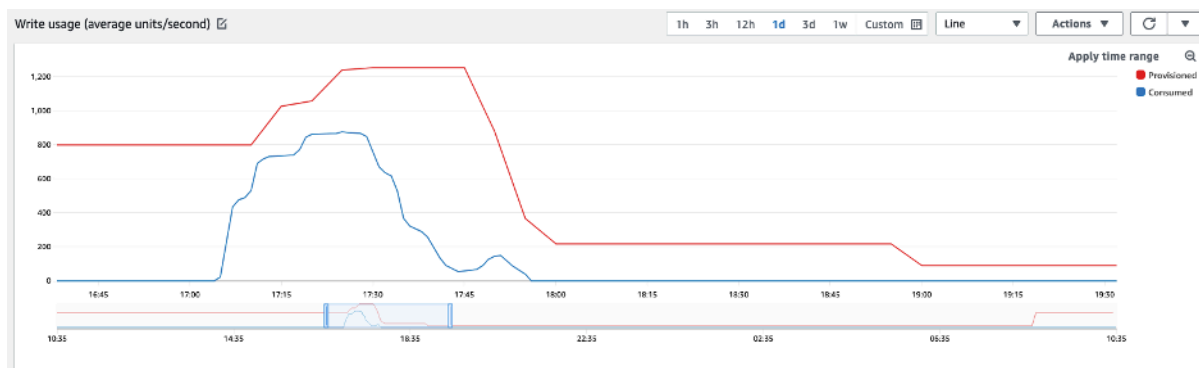
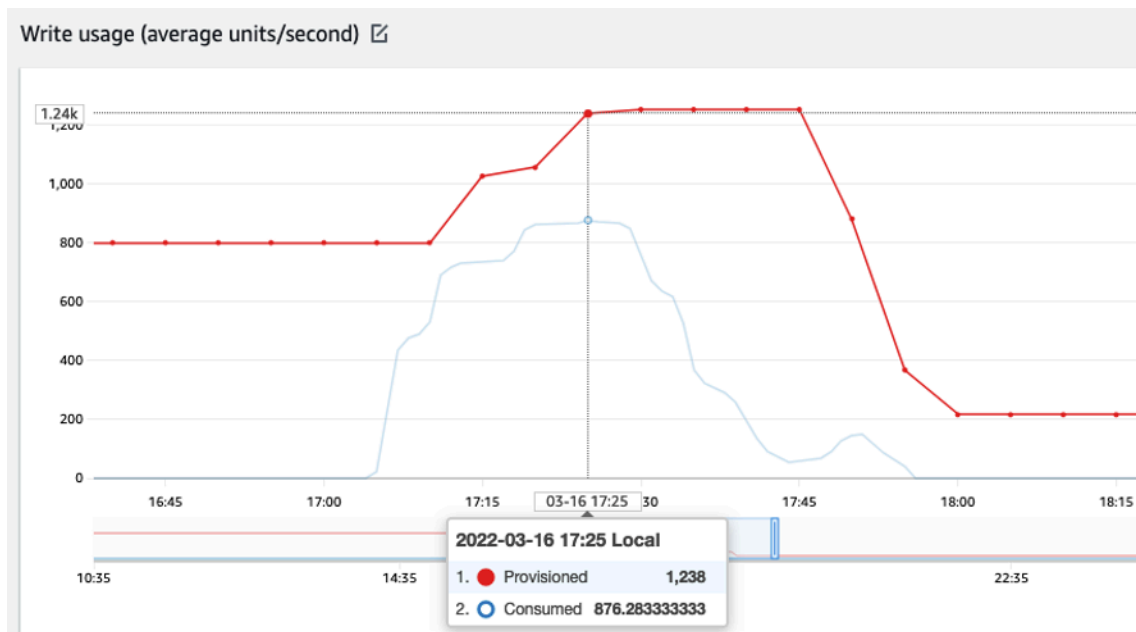
 "ScheduledActionName": "my-8-5-scheduled-action",
 "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
 table/<table-name>:scheduledActionName/my-8-5-scheduled-action",
 "ServiceNamespace": "dynamodb",
 "Schedule": "cron(45 8 ? * MON-FRI *)",
 "Timezone": "Australia/Brisbane",
 "ResourceId": "table/<table-name>",
 "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
 "ScalableTargetAction": {
 "MinCapacity": 800,
 "MaxCapacity": 1500
 },
 "CreationTime": "2022-03-15T17:28:57.816000+10:00"
 }
]
}

```

下圖顯示一律保持 70% 目標使用率的範例工作負載。請注意持續套用自動擴展規則的方式，同時輸送量不會降低。



放大時可以看到應用程式中，有一個尖峰觸發了 70% 的自動擴展閾值，強制啟動自動擴展，並提供資料表所需的額外容量。排程的自動擴展動作會影響最大值與最小值，您須設定這些值。



## 如何處理具有未知模式的尖峰工作負載

在此情況中，應用程式會使用非常低的使用率目標，因為您仍不確定應用程式模式，同時要確保工作負載不受限流。

請考慮改用[隨需容量模式](#)。隨需資料表非常適合流量模式不明的尖峰工作負載。使用隨需容量模式，您可以按請求支付應用程式在資料表上執行的資料讀取和寫入費用。您不需要指定預期應用程式執行的讀取和寫入輸送量，因為 DynamoDB 會立即因應工作負載的升降。

## 如何處理具有連結應用程式的工作負載

在此情況中，應用程式會依賴其他系統，像是在批次處理的情況，您可能會根據應用程式邏輯中的事件，遇到流量出現大峰值。

請考慮開發自訂自動擴展邏輯，以回應這些事件，讓您可以根據特定需要，提升資料表容量與 TargetValues。您可以受益於 Lambda Amazon EventBridge 和 Step Functions 等 AWS 服務的組合，以回應您的特定應用程式需求。

## 評估您的資料表類別選擇

本節概述如何為 DynamoDB 資料表選取適當的資料表類別。隨著標準不常存取 (Standard Infrequent-Access, 標準 - IA) 資料表類別的推出，您現在可利用此功能將資料表最佳化，實現較低的儲存成本或輸送量成本。

### 主題

- [有哪些可用的資料表類別](#)
- [何時選取 DynamoDB 標準資料表類別](#)
- [何時選取 DynamoDB 標準 - IA 資料表類別](#)
- [選擇資料表類別時應考慮的其他因素](#)

### 有哪些可用的資料表類別

建立 DynamoDB 資料表時，您必須選取 DynamoDB 標準或 DynamoDB 標準 - IA 資料表類別。資料表類別可以在 30 天內變更兩次，因此可以在日後隨時變更。選取任一資料表類別，都不會影響資料表的效能、可用性、可靠性或持久性。

## 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

## 標準資料表類別

新資料表預設的選項是標準資料表類別。對於具有頻繁存取資料的資料表，此選項可維持 DynamoDB 的原始計費餘額，有利於在輸送量和儲存成本之間取得平衡。

### 標準 - IA 資料表類別

對於需要長期儲存資料且不常更新或讀取的工作負載，標準 - IA 資料表類別可提供較低的儲存成本 (約降低 60%)。由於此類別已針對非頻繁的存取模式進行最佳化，因此讀取和寫入會以稍高於標準資料表類別的費率 (約提高 25%) 來計費。

### 何時選取 DynamoDB 標準資料表類別

如果資料表的儲存成本佔資料表每月總成本約 50% 以下，則最適合選擇 DynamoDB 標準資料表類別。這項成本結餘表示一項工作負載定期存取或更新 DynamoDB 內儲存項目的情況。

### 何時選取 DynamoDB 標準 - IA 資料表類別

如果資料表的儲存成本佔資料表每月總成本約 50% 以上，則最適合選擇 DynamoDB 標準 - IA 資料表類別。這項成本結餘表示一項工作負載每月建立或讀取項目少於儲存數量的情況。

標準 - IA 資料表類別的常見用途是將不常存取的資料移至個別的標準 - IA 資料表。如需進一步資訊，請參閱[使用 Amazon DynamoDB 標準 - IA 資料表類別來最佳化工作負載的儲存成本](#)。

### 選擇資料表類別時應考慮的其他因素

在兩種資料表類別之間進行抉擇時，還有一些其他因素值得納入決策考量。

### 預留容量

目前不支援使用標準 - IA 資料表類別來購買資料表的預留容量。在從具有預留容量的標準資料表轉換為不含預留容量的標準 - IA 資料表時，可能不會為您帶來可見的成本效益。

## 識別您未使用的資源

本節概述如何定期評估未使用的資源。隨著應用程式需求演變，您應確保沒有任何資源仍未使用且因此產生不必要的 Amazon DynamoDB 成本。以下描述的程序將使用 Amazon CloudWatch 指標識別未使用的資源，並協助您識別這些資源並採取行動以降低成本。

您可以使用來監控 DynamoDB CloudWatch，該模式會收集 DynamoDB 的原始資料，並將其處理為可讀且接近即時的指標。這些統計資料會保留一段時間，以便您存取歷史資訊，並更清楚了解自己的使用率。根據預設，DynamoDB 指標資料會自動傳送至 CloudWatch。如需詳細資訊，請參閱[什麼是 Amazon CloudWatch ?](#) 和 Amazon CloudWatch 用戶指南中的指[標保留](#)。

## 主題

- [如何識別未使用資源](#)
- [識別未使用的資料表資源](#)
- [清除未使用的資料表資源](#)
- [識別未使用的 GSI 資源](#)
- [清除未使用的 GSI 資源](#)
- [清除未使用的全域資料表](#)
- [清理未使用的備份或 point-in-time 復原 \(PITR\)](#)

### 如何識別未使用資源

若要識別未使用的資料表或索引，我們會查看 30 天期間的下列 CloudWatch 量度，以瞭解資料表是否有任何使用中的讀取或寫入，或是全域次要索引 (GSI) 的任何讀取：

#### [ConsumedReadCapacityUnits](#)

在指定時段使用的讀取容量單位數目，可讓您追蹤已使用多少使用容量。您可以擷取資料表及其所有全域次要索引或特定全域次要索引的總消耗讀取容量。

#### [ConsumedWriteCapacityUnits](#)

在指定時段使用的寫入容量單位數目，可讓您追蹤已使用多少使用容量。您可以擷取資料表及其所有全域次要索引或特定全域次要索引的消耗總計寫入容量。

### 識別未使用的資料表資源

Amazon CloudWatch 是一項監控和可觀察性服務，提供 DynamoDB 表格指標，您將用來識別未使用的資源。CloudWatch 您可以透過以 AWS Management Console 及透過檢視度量 AWS Command Line Interface。

### AWS Command Line Interface

若要透過檢視表格量度 AWS Command Line Interface，您可以使用下列命令。

1. 首先評估資料表的讀取：

```
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>
```

為了避免將資料表誤認為未使用，請評估較長期間內的指標。選擇適當的開始時間和結束時間範圍 (例如 30 天) 及適當的期間 (例如 86400)。

在傳回的資料中，任何大於 0 的總和都表示您所評估的資料表在該期間內曾經接收讀取流量。

下列結果顯示在評估期間接收讀取流量的資料表：

```
{
 "Timestamp": "2022-08-25T19:40:00Z",
 "Sum": 36023355.0,
 "Unit": "Count"
},
{
 "Timestamp": "2022-08-12T19:40:00Z",
 "Sum": 38025777.5,
 "Unit": "Count"
},
```

下列結果顯示在評估期間未接收讀取流量的資料表：

```
{
 "Timestamp": "2022-08-01T19:50:00Z",
 "Sum": 0.0,
 "Unit": "Count"
},
{
 "Timestamp": "2022-08-20T19:50:00Z",
 "Sum": 0.0,
 "Unit": "Count"
},
```

## 2. 接下來，評估資料表的寫入數：

```
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>
```

為了避免將資料表誤認為未使用，建議您評估較長期間內的指標。選擇適當的開始時間和結束時間範圍 (例如 30 天) 及適當的期間 (例如 86400)。

在傳回的資料中，任何大於 0 的總和都表示您所評估的資料表在該期間內曾經接收讀取流量。

下列結果顯示在評估期間接收寫入流量的資料表：

```
{
 "Timestamp": "2022-08-19T20:15:00Z",
 "Sum": 41014457.0,
 "Unit": "Count"
},
{
 "Timestamp": "2022-08-18T20:15:00Z",
 "Sum": 40048531.0,
 "Unit": "Count"
},
```

下列結果顯示在評估期間曾接收寫入流量的資料表：

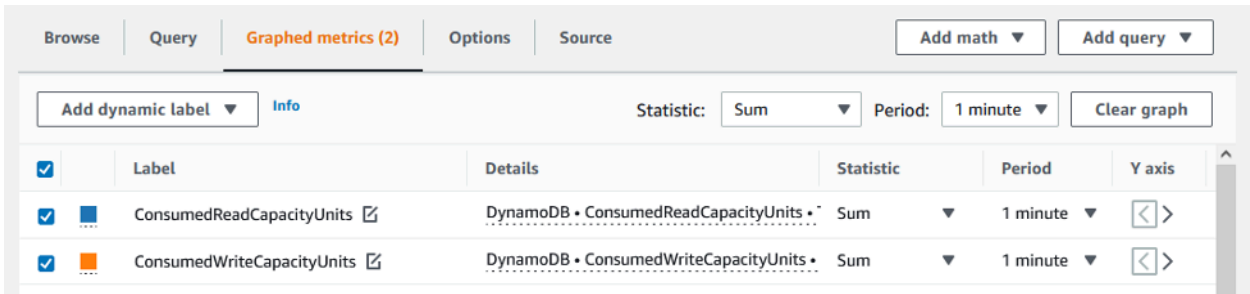
```
{
 "Timestamp": "2022-07-31T20:15:00Z",
 "Sum": 0.0,
 "Unit": "Count"
},
{
 "Timestamp": "2022-08-19T20:15:00Z",
 "Sum": 0.0,
 "Unit": "Count"
},
```

## AWS Management Console

採用下列步驟，即可透過 AWS Management Console 評估資源使用率。

1. 登入主 AWS 控制台並瀏覽至 <https://console.aws.amazon.com/cloudwatch/> 的 CloudWatch 服務頁面。如有必要，請在主機右上方選取適當的 AWS 區域。
2. 在左側導覽窗格中，找到 Metrics (指標) 區段並選取 All metrics (所有指標)。

3. 這個動作會開啟一個具有兩個面板的儀表板。在頂部面板中，您會看到目前的圖表化指標。在底部，您將能選取可供進行圖表化的指標。在底部面板中選取 DynamoDB。
4. 在 DynamoDB 指標選取面板中，選取 Table Metrics (資料表指標) 類別，以顯示目前區域中資料表的指標。
5. 向下捲動選單找出您的資料表名稱，然後為您的資料表選取指標 ConsumedReadCapacityUnits 和 ConsumedWriteCapacityUnits。
6. 選取 Graphed metrics (2) (圖表化指標 (2)) 索引標籤，然後將 Statistic (統計數據) 欄調整為 Sum (總和)。



7. 為了避免將資料表誤認為未使用，建議您評估較長期間內的指標。在圖表面板頂端，選擇適當的時間範圍 (例如 1 個月) 來評估您的資料表。選取 Custom (自訂)，在下拉式清選中選取 1 Months (1 個月)，然後選擇 Apply (套用)。

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

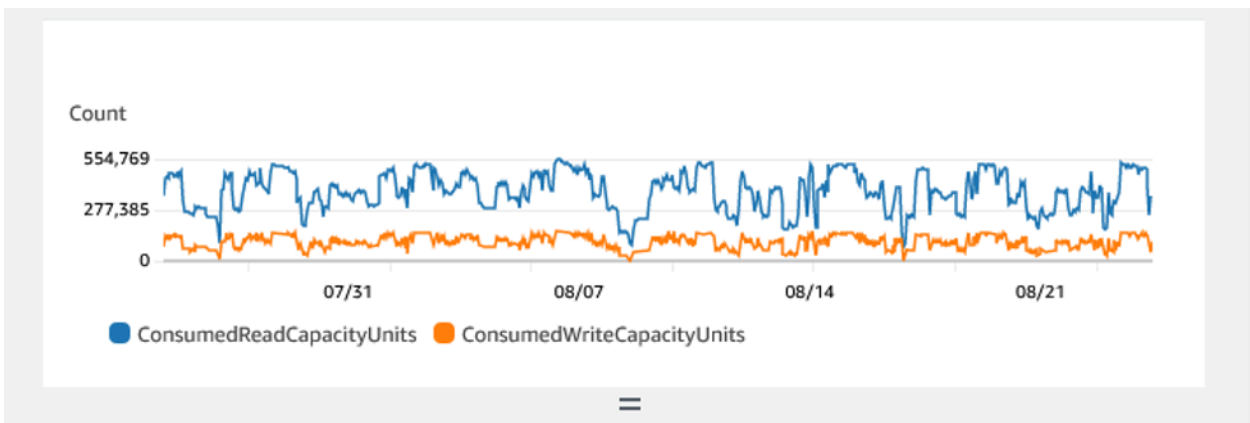
Months 3 6 12 15

1 Months ▼

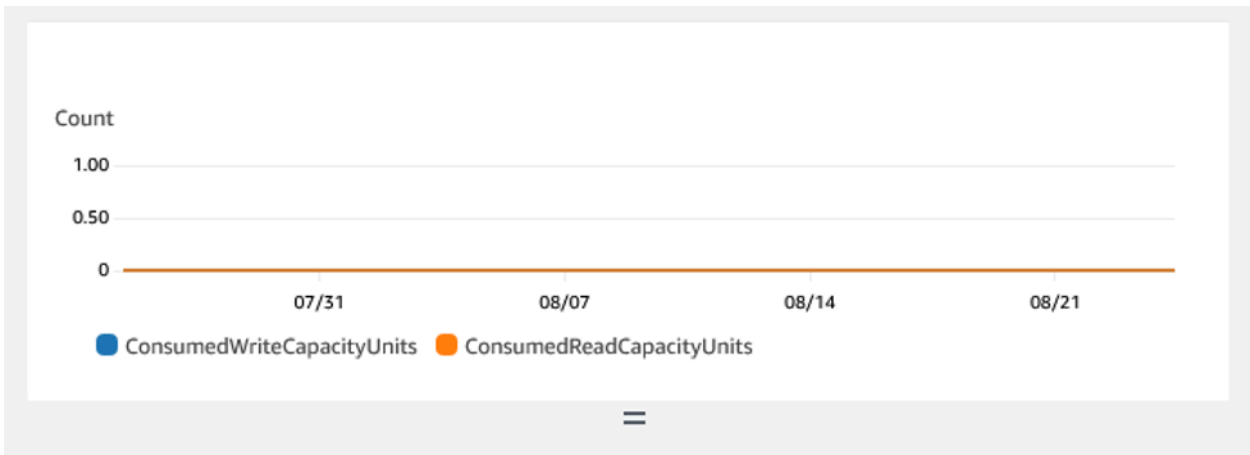
Clear Cancel Apply

8. 評估資料表的圖表化指標，判斷是否已使用該資料表。如果指標超過 0，就表示在評估期間內已使用該資料表。讀取和寫入均為 0 的平坦圖形，表示該資料表尚未使用。

下圖顯示具有讀取流量的資料表：



下圖顯示沒有讀取流量的資料表：



## 清除未使用的資料表資源

如果找出未使用的資料表資源，您可以透過下列方式降低其持續產生的成本。

### Note

如果找出未使用的資料表，但仍希望保留以備日後需要時可供存取，請考慮將其轉換為隨需模式。否則，您可以考慮備份和刪除整個資料表。

## 容量模式

DynamoDB 會對 DynamoDB 資料表中讀取、寫入及儲存資料等活動收取費用。

DynamoDB 有「隨需」和「佈建」[兩種容量模式](#)，分別提供適用於處理資料表讀取和寫入的特定計費選項。讀取/寫入容量模式可控制您變更讀取與寫入傳輸量以及管理容量的方式。

若為隨需模式資料表，不需要指定您預期應用程式將進行的讀取和寫入輸送量。DynamoDB 會將依據您的應用程式在資料表上執行的讀取與寫入請求單位，向您收取與寫入的費用。如果您的資料表/索引沒有任何活動，則不需要支付輸送量的費用，但仍會產生儲存費用。

## 資料表類別

DynamoDB 也提供兩種[資料表類別](#)，旨在協助您最佳化成本。預設值為 DynamoDB 標準資料表類別，建議大多數工作負載使用。DynamoDB 標準-不常存取 (DynamoDB 標準-IA) 資料表類別針對以儲存為主要成本的資料表進行最佳化。

如果您的資料表或索引沒有任何活動，儲存體可能成為主要成本來源，而變更資料表類別可大幅節省成本。

## 刪除資料表

如果您發現未使用的資料表並希望予以刪除，建議您先備份或匯出資料。

透過 AWS Backup 進行的備份可利用冷儲存分層，進一步降低成本。如需如何透過 AWS Backup 啟用備份功能的資訊，以及如何使用生命週期將備份移至冷存放裝置的資訊，請參閱文件說明文件。[搭配 DynamoDB 使用 AWS Backup](#)

或者，您可以選擇將資料表的資料匯出到 S3。若要執行此操作，請參閱[匯出至 Amazon S3](#) 文件。資料匯出後，如果希望利用 S3 Glacier Instant Retrieval、S3 Glacier Flexible Retrieval 或 S3 Glacier Deep Archive 進一步降低成本，請參閱[管理儲存生命週期](#)。

備份資料表之後，即可選擇透過 AWS Management Console 或 AWS Command Line Interface 加以刪除。

## 識別未使用的 GSI 資源

識別未使用全域次要索引的步驟類似於識別未使用資料表的步驟。如果寫入基底資料表的項目包含用作 GSI 分割索引鍵的屬性，則 DynamoDB 會將這些項目複製到 GSI 中，因此在基底資料表處於使用狀態時，未使用的 GSI 仍可能具有大於 0 的 ConsumedWriteCapacityUnits。因此，您只需評估 ConsumedReadCapacityUnits 指標即可判斷 GSI 是否未使用。

若要透過檢視 GSI 指標 AWS CLI，您可以使用下列命令來評估表格讀取：

```
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>
```

為了避免將資料表誤認為未使用，建議您評估較長期間內的指標。選擇適當的開始時間和結束時間範圍 (例如 30 天) 及適當的期間 (例如 86400)。

在傳回的資料中，任何大於 0 的總和都表示您所評估的資料表在該期間內曾經接收讀取流量。

下列結果顯示 GSI 在評估期間內接收讀取流量：

```
{
 "Timestamp": "2022-08-17T21:20:00Z",
 "Sum": 36319167.0,
 "Unit": "Count"
```

```
 },
 {
 "Timestamp": "2022-08-11T21:20:00Z",
 "Sum": 1869136.0,
 "Unit": "Count"
 },
```

下列結果顯示 GSI 在評估期間內接收最低讀取流量：

```
 {
 "Timestamp": "2022-08-28T21:20:00Z",
 "Sum": 0.0,
 "Unit": "Count"
 },
 {
 "Timestamp": "2022-08-15T21:20:00Z",
 "Sum": 2.0,
 "Unit": "Count"
 },
```

下列結果顯示 GSI 在評估期間未接收讀取流量：

```
 {
 "Timestamp": "2022-08-17T21:20:00Z",
 "Sum": 0.0,
 "Unit": "Count"
 },
 {
 "Timestamp": "2022-08-11T21:20:00Z",
 "Sum": 0.0,
 "Unit": "Count"
 },
```

## 清除未使用的 GSI 資源

如果找出未使用的 GSI，您可以選擇將其刪除。由於 GSI 中的所有資料也位於基底資料表中，因此在刪除 GSI 之前，不需進行額外的備份。如果日後再次需要 GSI，則可以將其重新加入資料表中。

如果找出不常使用的 GSI，則應考慮變更應用程式中的設計，以便將其刪除或降低其成本。例如，雖然應謹慎使用 DynamoDB 掃描，因為其可能消耗大量系統資源，但如果它們支援的存取模式很少使用，則可能比 GSI 更具成本效益。

此外，如果需要 GSI 來支援「不常存取」模式，請考慮投射一組限制程度較高的屬性。雖然這可能需要對基底資料表進行後續查詢以支援「不常存取」模式，但可能有助於大幅降低儲存和寫入成本。

## 清除未使用的全域資料表

Amazon DynamoDB 全域資料表提供全受管解決方案，用以部署多個區域和多個作用中資料庫，而不需要建置和維護您自己的複寫解決方案。

對於使用者近處的資料以及用於災難復原的次要區域，理想的做法是使用全域資料表來提供低延遲的存取管道。

如果為資源啟用了全域表選項，以便提供資料的低延遲存取，但不是災難復原策略的一部分，請透過評估其 CloudWatch 指標來驗證這兩個複本是否主動提供讀取流量。如果其中一個複本不提供讀取流量，則可能就是未使用的資源。

如果全域資料表是災難復原策略的一部分，則在作用中/待命模式下可能會有一個複本未接收讀取流量。

## 清理未使用的備份或 point-in-time 復原 (PITR)

DynamoDB 提供兩種備份樣式。Point-in-time Recovery 提供 35 天的連續備份功能，協助您防止意外寫入或刪除，而隨需備份則可建立快照，並可長期儲存。兩種類型的備份都會產生相關成本。

請參閱有關 [使用 DynamoDB 的隨需備份與還原](#) 和 [適用於 DynamoDB 的 Point-in-time 復原](#) 的說明文件，以判斷您的資料表是否啟用了可能不再需要的備份。

## 評估您的資料表用量模式

本節概述如何判斷您有效地使用 DynamoDB 資料表。某些用量模式對 DynamoDB 來說並非最理想的，且從效能和成本角度來看，這些模式仍有最佳化空間。

### 主題

- [執行較少高度一致性讀取操作](#)
- [執行較少的讀取操作交易](#)
- [執行較少掃描](#)
- [縮短屬性名稱](#)
- [啟用存留時間 \(TTL\)](#)
- [以跨區域備份取代全域資料表](#)



## 執行較少高度一致性讀取操作

DynamoDB 可讓您根據每個請求設定[讀取一致性](#)。根據預設，讀取請求最終會保持一致。最終一致讀取費用為 0.5 RCU，資料上限為 4 KB。

分散式工作負載的多數部分都具有彈性，可以容忍最終一致性。但是，有些存取模式要求高度一致性讀取。高度一致性讀取費用為 1 RCU，資料上限為 4 KB，實際上是讀取成本的兩倍。DynamoDB 讓您在同一個資料表上使用這兩種一致性模式。

您可以評估工作負載和應用程式的程式碼，確定是否僅在需要時使用高度一致性讀取。

## 執行較少的讀取操作交易

DynamoDB 可讓您以某種 all-or-nothing 方式將某些動作分組，這表示您可以使用 DynamoDB 執行 ACID 交易。但是，如同關聯式資料庫，並非每個動作都需要遵循這種方法。

上限 4 KB 的[交易讀取操作](#)會耗用 2 RCU，而非以最終一致方式讀取相同數量資料的預設 0.5 RCU。寫入操作的成本會加倍，也就是說，最多 1 KB 的交易寫入等於 2 WCU。

若要判斷表格上的所有作業是否都是交易，可以將表格的 CloudWatch 指標向下篩選為交易 API。若交易 API 是資料表 SuccessfulRequestLatency 指標下唯一可用的圖表，這就表示此資料表中每項操作都是交易。或者，若整體容量使用率趨勢符合交易 API 趨勢，請考慮重新檢視應用程式設計，因為該設計似乎由交易 API 主導。

## 執行較少掃描

會廣泛使用 Scan 操作，通常源於對 DynamoDB 資料執行分析查詢的需求。在大型資料表上執行頻繁 Scan 操作可能既低效又昂貴。

更好的替代方法是使用[匯出至 S3](#) 功能，並選擇一個時間點將資料表狀態匯出到 S3。AWS 提供像 Athena 這樣的服務，然後可以用來對資料執行分析查詢，而不會消耗資料表中的任何容量。

您可以使用 Scan 的 SuccessfulRequestLatency 指標下的 SampleCount 統計資料，決定 Scan 操作的頻率。若 Scan 操作確實十分頻繁，則應重新評估存取模式和資料模型。

## 縮短屬性名稱

DynamoDB 項目的總大小是其屬性名稱長度和值的總和。較長的屬性名稱不僅產生更多儲存成本，也導致更多 RCU/WCU 耗用。建議您選擇較短的屬性名稱，而不是較長的屬性名稱。較短的屬性名稱有助於限制項目大小少於 4KB/1KB，之後您將耗用額外的 RCU/WCU 來存取資料。

## 啟用存留時間 (TTL)

[存留時間 \(TTL\)](#) 可識別比您設定的到期時間更早的項目，並將其從資料表中移除。若您的資料已不重要，則在資料表上啟用 TTL 可協助您縮減資料並節省儲存成本。

TTL 的另一個益處為，過期項目發生在 DynamoDB 串流上，因此您不僅從資料中移除該資料，而是可以從串流中使用這些項目，並封存到成本較低的儲存層。此外，透過 TTL 刪除項目不需額外費用 — 它不會消耗容量，且設計清理應用程式也不會產生額外負荷。

## 以跨區域備份取代全域資料表

[全域表格](#) 可讓您在不同區域中維護多個使用中複本資料表 — 這些資料表皆可接受寫入操作，並彼此複寫資料。您可以輕鬆設定複本，系統會為您管理同步處理。使用最後一個寫入獲勝策略，將複本收斂到一致。

若您僅使用全域資料表做為容錯移轉或災難復原 (DR) 策略的一部分，您可以將資料表取代為跨區域備份複本，以達到相對較寬鬆的復原點目標，和復原時間目標需求。若您不需要快速的本地存取和五個九的高可用性，維護全域資料表複本可能不是容錯移轉的最佳方法。

或者，您可以考慮使用 `B AWS ackup` 來管理 DynamoDB 備份。與使用全域資料表相比，您可以排程定期備份並跨區域複製備份，以更符合成本效益的方式滿足 DR 需求。

## 評估您的 Streams 使用量

本章節概述如何評估 DynamoDB Streams 使用量。某些使用量模式對 DynamoDB 來說並非最理想的，且從效能和成本角度來看，這些模式仍有最佳化空間。

您有兩個原生串流整合，適用於串流和事件驅動使用案例：

- [Amazon DynamoDB Streams](#)
- [Amazon Kinesis Data Streams](#)

此頁面將著重於這些選項的成本最佳化策略。若您想了解如何在兩個選項間選擇，請參閱 [變更資料擷取的串流選項](#)。

## 主題

- [最佳化 DynamoDB Streams 的成本](#)
- [最佳化 Kinesis Data Streams 的成本](#)

- [兩種 Streams 使用量類型的成本最佳化策略](#)

### 最佳化 DynamoDB Streams 的成本

如 DynamoDB Streams 的[定價頁面](#)所述，無論資料表的輸送容量模式為何，DynamoDB 都會根據對資料表 DynamoDB 串流發出的讀取請求量計費。對 DynamoDB 串流發出的讀取請求，與對 DynamoDB 資料表發出的讀取請求不同。

串流中的每個讀取請求都採用 GetRecords API 呼叫形式，可在回應中傳回最多 1000 筆記錄或 1 MB 的記錄，以先到達者為準。[其他 DynamoDB 串流 API](#) 不會收取任何費用，且 DynamoDB Streams 不會因閒置而收費。換句話說，若沒有對 DynamoDB 串流發出讀取請求，則在資料表上啟用 DynamoDB 串流將不會產生任何費用。

以下是 DynamoDB Streams 的部分消費者應用程式：

- AWS Lambda 函數
- Amazon Kinesis Data Streams 型應用程式
- 使用 AWS SDK 建置的客戶消費者應用程式

以 AWS Lambda 為基礎的 DynamoDB Streams 消費者所發出的讀取請求是免費的，而任何其他類型的消費者呼叫則會收費。每個月，非 Lambda 消費者發出的前 2,500,000 個讀取要求也是免費。這適用於對 AWS 帳戶中每 AWS 個區域的任何 DynamoDB Streams 發出的所有讀取請求。

### 監控您的 DynamoDB Streams 使用量

帳單主控台上的 DynamoDB Streams 費用會針對帳戶中跨 AWS 區域的所有 DynamoDB Streams 分組在一起。AWS 目前不支援標記 DynamoDB Streams，因此無法使用成本分配標籤來識別 DynamoDB Streams 的精細成本。您可以在 DynamoDB 串流層級取得 GetRecords 磁碟區，以計算每個串流的費用。該磁碟區由 DynamoDB 串流的 CloudWatch 量度 SuccessfulRequestLatency 及其 SampleCount 統計資料表示。此指標也會包含全域表執行持續複寫的 GetRecords 呼叫，以及 AWS Lambda 取用者所發出的呼叫，兩者都不收費。如需 DynamoDB Streams 發佈的其他 CloudWatch 量度的相關資訊，請參閱。[DynamoDB 指標和維度](#)

### 使用 AWS Lambda 做為消費者

評估使用 AWS Lambda 函數做為 DynamoDB Streams 的取用者是否可行，因為這樣可以消除與從 DynamoDB 串流讀取相關的成本。另一方面，DynamoDB Streams Kinesis 轉接器或以 SDK 為基礎的消費者應用程式將根據他們對 DynamoDB 串流發出的 GetRecords 呼叫量計費。

Lambda 函數呼叫將根據標準 Lambda 定價收費，但 DynamoDB Streams 不會產生任何費用。Lambda 會輪詢您的 DynamoDB 串流中的碎片，其記錄的基本速率為每秒 4 次。當記錄可用時，Lambda 會調用您的函數，並等待結果。如果處理成功，Lambda 會恢復輪詢，直到收到多筆記錄。

調校以 DynamoDB Streams Kinesis 轉接器為基礎的消費者應用程式

由於非 Lambda 消費者發出的讀取要求會針對 DynamoDB Streams 收費，因此需在近乎即時的要求與消費者應用程式須輪詢 DynamoDB 串流的次數間找出平衡。

使用以 DynamoDB Streams Kinesis 轉接器為基礎的應用程式輪詢 DynamoDB Streams 的頻率由設定的 `idleTimeBetweenReadsInMillis` 值決定。此參數決定消費者在處理碎片前應等待的時間 (以毫秒為單位)，以防先前對相同碎片進行的 `GetRecords` 叫用未傳回任何記錄。此參數的預設值為 1000 ms。若不需要近乎即時的處理，則可以增加此參數，讓消費者應用程式在 DynamoDB Streams 呼叫上進行較少的 `GetRecords` 呼叫，並進行最佳化。

最佳化 Kinesis Data Streams 的成本

當 Kinesis 資料串流設定為目的地，為 DynamoDB 資料表提供變更資料擷取事件時，Kinesis 資料串流可能需單獨的大小管理，但這會影響整體成本。DynamoDB 會根據變更資料擷取單位 (CDU) 計費，其中每單位皆由最多 1 KB DynamoDB 項目大小所組成，該項目則為 DynamoDB 服務嘗試傳送至目的地 Kinesis 動資料串流的內容。

除了 DynamoDB 服務費用外，也會產生標準的 Kinesis 資料串流費用。如[定價頁面](#)所述，服務定價會根據容量模式 (佈建和隨需) 而不同，這些模式與 DynamoDB 資料表容量模式不同，且為使用者定義。在高階上，Kinesis Data Streams 會根據容量模式以及 DynamoDB 服務擷取到的串流資料，按小時計費。視 Kinesis 資料串流的使用者組態而定，可能產生額外費用，例如資料擷取 (針對隨需模式)、延長資料保留 (超過預設 24 小時)，以及強化廣發消費者擷取。

監控 Kinesis Data Streams 使用量

適用於 DynamoDB 的 Kinesis Data Streams 會發佈 DynamoDB 的指標，以及標準的 Kinesis 料串流指標。CloudWatch Kinesis 服務可能會限制 DynamoDB 服務的 Put 嘗試，因為 Kinesis Data Streams 容量不足，或是由相依元件 (例如可能設定為加密靜態 Kinesis 資料串流資料的 AWS KMS 服務) 所造成的限制。

若要進一步了解 DynamoDB 服務針對 Kinesis 資料串流發佈的 CloudWatch 指標，請參閱。[使用 Kinesis Data Streams 監控變更資料擷取](#) 為避免因限流而產生額外的服務重試成本，請確定在佈建模式中，Kinesis 資料串流的大小適中。

## 為 Kinesis Data Streams 選擇合適的容量模式

Kinesis Data Streams 有兩種容量模式受支援：佈建模式和隨需模式。

- 如果涉及 Kinesis 資料串流的工作負載具有可預測或準確預測的應用程式流量 (流量一致或逐漸增加)，那麼 Kinesis Data Stream 的佈建模式就很適合，且更具成本效益
- 若該工作負載是新工作負載，有無法預測的應用程式流量，或您不想管理容量，那麼 Kinesis Data Streams 的隨需模式就很適合，且更具成本效益

最佳化成本的最佳實務是評估與 Kinesis 資料串流相關聯的 DynamoDB 資料表是否具有可預測的流量模式，該模式可利用 Kinesis Data Streams 的佈建模式。如果工作負載是新的，您可以使用 Kinesis Data Streams 的隨選模式幾週，檢閱指標以瞭解流 CloudWatch 量模式，然後根據工作負載的性質將相同的「串流」切換至佈建模式。在佈建模式中，可以遵循 Kinesis Data Streams 的碎片管理考量事項，估算碎片數量。

使用 DynamoDB 專用 Kinesis Data Streams，評估您的消費者應用程式

由於 Kinesis Data Streams 不會對 DynamoDB Streams 等 GetRecords 呼叫數量計費，因此只要頻率低於 GetRecords 的限流限制，消費者應用程式就能盡量執行呼叫。針對 Kinesis Data Streams 的隨需模式，資料讀取按 GB 計費。針對 Kinesis Data Streams 的佈建模式，若資料保留時間少於 7 天，則不收取讀取費用。若 Lambda 函數為 Kinesis Data Streams 消費者，Lambda 會在 Kinesis 串流輪詢每個碎片，以每秒一次的基本速率記錄。

## 兩種 Streams 使用量類型的成本最佳化策略

### 針對 AWS Lambda 消費者的事件

Lambda 事件篩選可讓您根據篩選條件捨棄事件，使其無法在 Lambda 函數叫用批次中使用。如此一來，便可最佳化 Lambda 成本，以在消費者函數邏輯中處理或捨棄不需要的串流記錄。若要深入了解如何設定事件篩選和寫入篩選條件，請參閱 [Lambda 事件篩選](#)。

### 調整 AWS Lambda 費者

您可以調校 Lambda 組態參數，進一步最佳化成本，例如提升 BatchSize 以在每次呼叫處理更多內容、啟用 BisectBatchOnFunctionError 以防止處理重複項目 (會產生額外成本)，以及設定 MaximumRetryAttempts 以防產生太多重試次數。根據預設，失敗的消費者 Lambda 叫用會無限重試，直到記錄從串流到期，DynamoDB Streams 的期限約為 24 小時，針對 Kinesis Data Streams 則可設定為 24 小時至最長 1 年。可在《Lambda AWS 開發人員指南》<https://docs.aws.amazon.com/lambda/latest/dg/with-ddb.html#services-ddb-params> 找到其他 Lambda 設定選項，包含上述適用於 DynamoDB 串流消費者的選項。



## 評估您是否具有適當大小的佈建容量

本節概述如何評估 DynamoDB 資料表是否有適當大小的佈建。隨著工作負載的演變，您應該適時修改操作程序，尤其是在佈建模式下設定 DynamoDB 資料表時，可能會有過度佈建或佈建不足的風險。

以下描述的程序需要統計資訊，這些資訊應從支援您生產應用程式的 DynamoDB 資料表擷取。若要瞭解您的應用程式行為，您應定義足以從應用程式擷取資料季節性的時段。例如，若應用程式顯示每週模式，三週時間便應足夠來分析應用程式輸送量需求。

若您不知道從何下手，請使用至少一個月的資料用量進行以下計算。

評估容量時，DynamoDB 資料表可以獨立設定讀取容量單位 (RCU) 和寫入容量單位 (WCU)。如果資料表設定了任何全域次要索引 (GSI)，則需指定耗用輸送量，這也與基礎資料表的 RCU 和 WCU 無關。

### Note

本機次要索引 (LSI) 會耗用基礎資料表的容量。

### 主題

- [如何擷取 DynamoDB 資料表上的耗用指標](#)
- [如何識別佈建不足的 DynamoDB 資料表](#)
- [如何識別過度佈建的 DynamoDB 資料表](#)

### 如何擷取 DynamoDB 資料表上的耗用指標

若要評估表格和 GSI 容量，請監視下列 CloudWatch 測量結果，然後選取適當的維度來擷取表格或 GSI 資訊：

讀取容量單位	寫入容量單位
ConsumedReadCapacityUnits	ConsumedWriteCapacityUnits
ProvisionedReadCapacityUnits	ProvisionedWriteCapacityUnits
ReadThrottleEvents	WriteThrottleEvents

您可以透過 AWS CLI 或執行此操作 AWS Management Console。

## AWS CLI

在擷取資料表使用量指標之前，我們需要先使用 CloudWatch API 擷取一些歷史資料點。

首先要建立兩個檔案：`write-calc.json` 和 `read-calc.json`。這些檔案代表資料表或 GSI 的計算。您需要更新部分欄位 (如下表所示)，以符合您的環境。

欄位名稱	定義	範例
<code>&lt;table-name&gt;</code>	您要分析的資料表名稱	SampleTable
<code>&lt;period&gt;</code>	您將用來評估使用率目標的期間 (以秒為單位)	針對 1 小時的時間，您應指定：3600
<code>&lt;start-time&gt;</code>	評估間隔的開始時間，以 ISO8601 格式顯示	2022-02-21T23:00:00
<code>&lt;end-time&gt;</code>	評估間隔的結束時間，以 ISO8601 格式顯示	2022-02-22T06:00:00

寫入計算檔案將擷取指定日期範圍時段內佈建和耗用的 WCU 數。同時也會產生用於分析的使用率百分比。`write-calc.json` 檔案的完整內容應如下所示：

```
{
 "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
}
```



讀取計算檔案會使用類似檔案。此檔案將擷取指定日期範圍時段內佈建和耗用的 RCU 數。read-calc.json 檔案的內容應如下所示：

```
{
 "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
}
```

一旦您建立了檔案，就可以開始擷取使用率資料。

1. 若要擷取寫入使用率資料，請發出下列命令：

```
aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
```

2. 若要擷取讀取使用率資料，請發出下列命令：

```
aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
```

這兩個查詢結果將為一系列 JSON 格式的資料點，這些資料點會用於分析。您的結果將取決於您指定的資料點數量、期間以及您自己的特定工作負載資料。這可能看起來如下：

```
{
 "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,
 40.947256944444445,
 25.618194444444444,
 0.0
],
 "StatusCode": "Complete"
}
],
"Messages": []
}
```

#### Note

若您指定一個短期和一個長期時間範圍，您可能需要修改指令碼中預設為 24 的 MaxDatapoints。這代表每小時產生一個資料點，每天 24 個資料點。

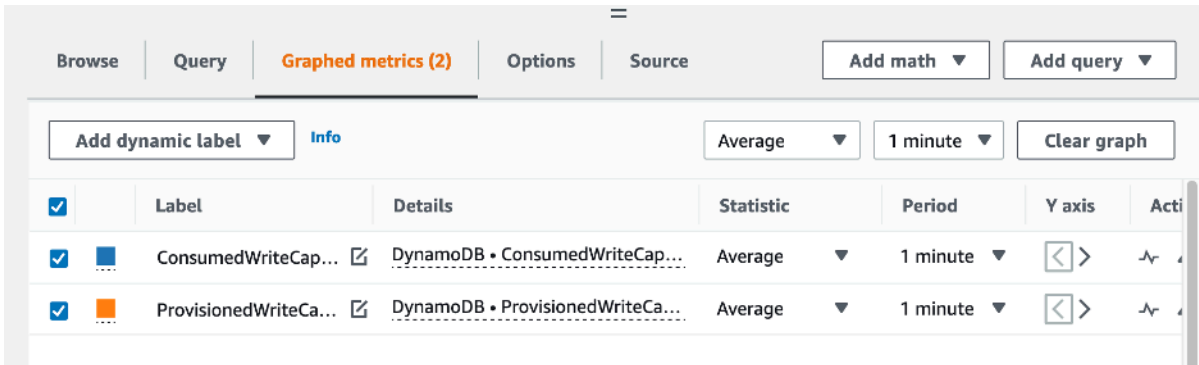
## AWS Management Console

1. 登入 AWS Management Console 並瀏覽至「[開始使用](#)」中的 [CloudWatch 服務頁面 AWS Management Console](#)。如有必要，請選取適當的 AWS 區域。
2. 在左側導覽窗格中找到指標區段，並選取 All metrics (所有指標)。
3. 這會開啟一份儀表板，其中包含兩個面板。頂部面板顯示圖形，底部面板則顯示您欲繪製圖形的指標。選取 DynamoDB 面板。
4. 從子面板選取 Table Metrics (資料表指標) 類別。接著會顯示目前區域中的資料表。
5. 向下捲動選單找出您的資料表名稱，然後選取寫入操作指標：ConsumedWriteCapacityUnits 和 ProvisionedWriteCapacityUnits。

**Note**

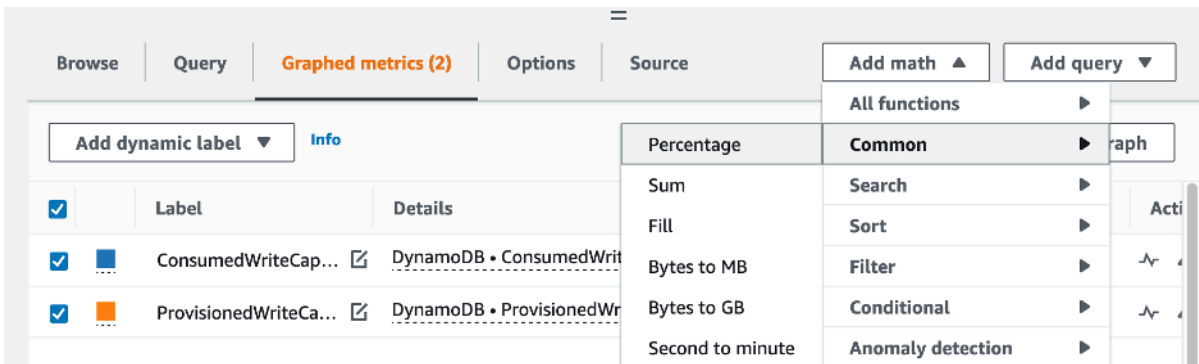
此範例討論寫入操作指標，但您也可以用這些步驟來繪製讀取操作指標的圖形。

6. 選取 Graphed metrics (2) (圖表化指標(2)) 標籤，修改公式。依預設，CloudWatch 會為圖表選取統計函數平均值。

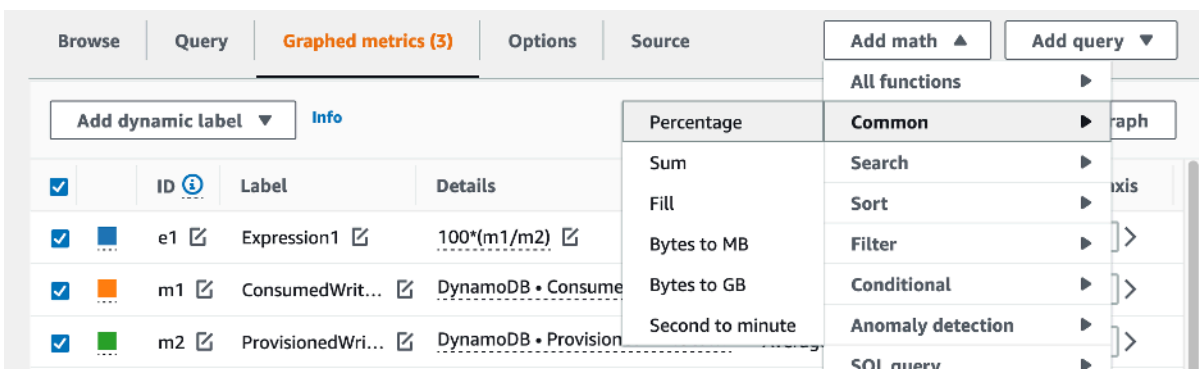


7. 同時選取兩個圖形化指標 (左側的核取方塊) 後，選取選單 Add math (新增數學)，再選取 Common (一般) 及 Percentage (百分比) 函數。重複該過程兩次。

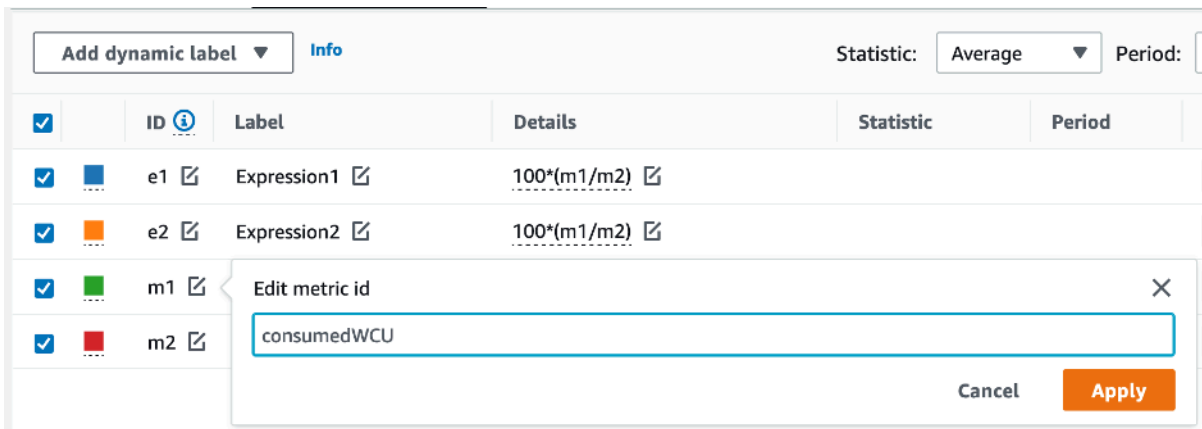
第一次選取 Percentage (百分比) 函數：



第二次選取 Percentage (百分比) 函數：



8. 此時，選單底部應有四個指標。接著進行 ConsumedWriteCapacityUnits 計算。為了保持一致，我們需要匹配我們在 AWS CLI 部分中使用的名稱。按一下 m1 ID，並將此值變更為 consumedWCU。

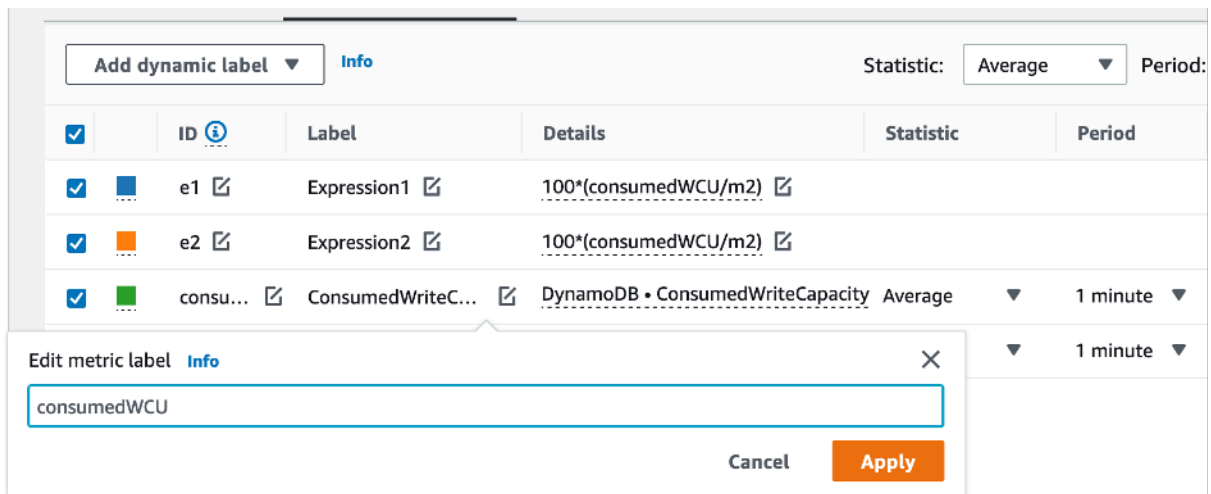


ID	Label	Details	Statistic	Period
e1	Expression1	100*(m1/m2)	Average	1 minute
e2	Expression2	100*(m1/m2)	Average	1 minute
m1	ConsumedWriteC...	DynamoDB • ConsumedWriteCapacity	Average	1 minute
m2	ConsumedWriteC...	DynamoDB • ConsumedWriteCapacity	Average	1 minute

Edit metric id

consumedWCU

Cancel Apply



ID	Label	Details	Statistic	Period
e1	Expression1	100*(consumedWCU/m2)	Average	1 minute
e2	Expression2	100*(consumedWCU/m2)	Average	1 minute
consumedWCU	ConsumedWriteC...	DynamoDB • ConsumedWriteCapacity	Average	1 minute

Edit metric label

consumedWCU

Cancel Apply

9. 將統計資料從 Average (平均值) 變更為 Sum (總和)。此動作會自動建立另一個稱為 ANOMALY\_DETECTION\_BAND 的指標。針對本程序的範圍，我們可以移除新產生的 ad1 metric 上的核取方塊，加以忽略。

The screenshot shows the 'Graphed metrics (4)' tab in the Amazon CloudWatch console. A search bar is at the top right. Below it, there are tabs for 'Browse', 'Query', 'Graphed metrics (4)', 'Options', and 'Source'. A dropdown menu is open, showing options for 'Standard', 'Average', 'Minimum', 'Maximum', 'Sum', and 'Sample count'. The 'Sum' option is highlighted. Below the menu, there is a table with columns for 'ID', 'Label', and 'Details'. The table contains five rows of metrics, including 'e1', 'e2', 'consumedWCU', and 'm2'.

ID	Label	Details
e1	Expression1	$100 * (\text{consumedWCU} / \text{m2})$
e2	Expression2	$100 * (\text{consumedWCU} / \text{m2})$
consumedWCU	consumedWCU	DynamoDB • ConsumedWriteCapacity
m2	ProvisionedWriteCapacity	DynamoDB • ProvisionedWriteCapacity

The screenshot shows the 'Graphed metrics (4/5)' tab in the Amazon CloudWatch console. The 'Statistic' dropdown is set to '(multiple)' and the 'Period' is set to '1 minute'. The table shows the same metrics as the previous screenshot, but with the 'Statistic' column added. The 'Statistic' column contains 'Sum' for 'consumedWCU' and 'Average' for 'm2'.

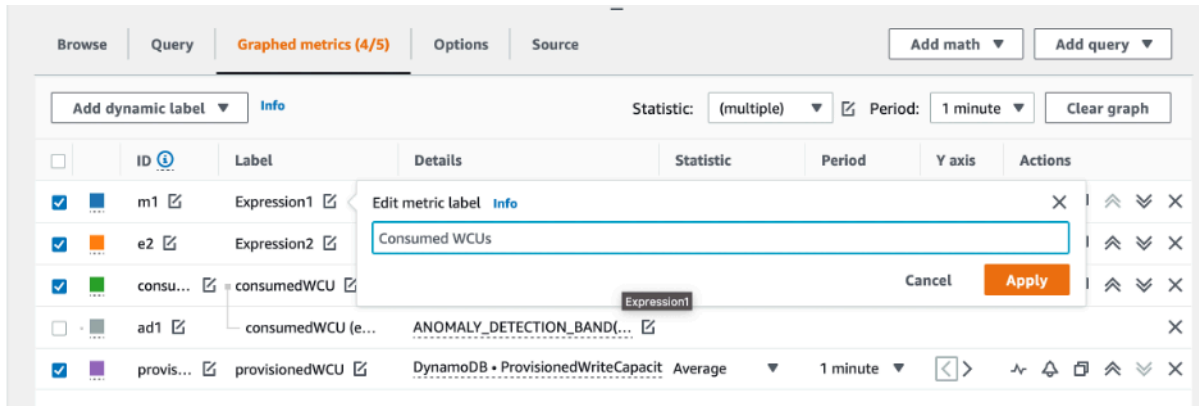
ID	Label	Details	Statistic	Period	Y axis	Act
e1	Expression1	$100 * (\text{consumedWCU} / \text{m2})$				
e2	Expression2	$100 * (\text{consumedWCU} / \text{m2})$				
consumedWCU	consumedWCU	DynamoDB • ConsumedWriteCapacity	Sum	1 minute		
ad1	consumedWCU (e...)	ANOMALY_DETECTION_BAND(...)				
m2	ProvisionedWriteCapacity	DynamoDB • ProvisionedWriteCapacity	Average	1 minute		

10. 重複步驟 8，將 m2 ID 重新命名為 provisionedWCU。將統計資料設定保留為 Average (平均值)。

The screenshot shows the 'Graphed metrics (4/5)' tab in the Amazon CloudWatch console. The 'Statistic' dropdown is set to '(multiple)' and the 'Period' is set to '1 minute'. The table shows the same metrics as the previous screenshot, but with the 'Statistic' column added. An 'Edit metric label' dialog box is open, showing the label 'provisionedWCU'.

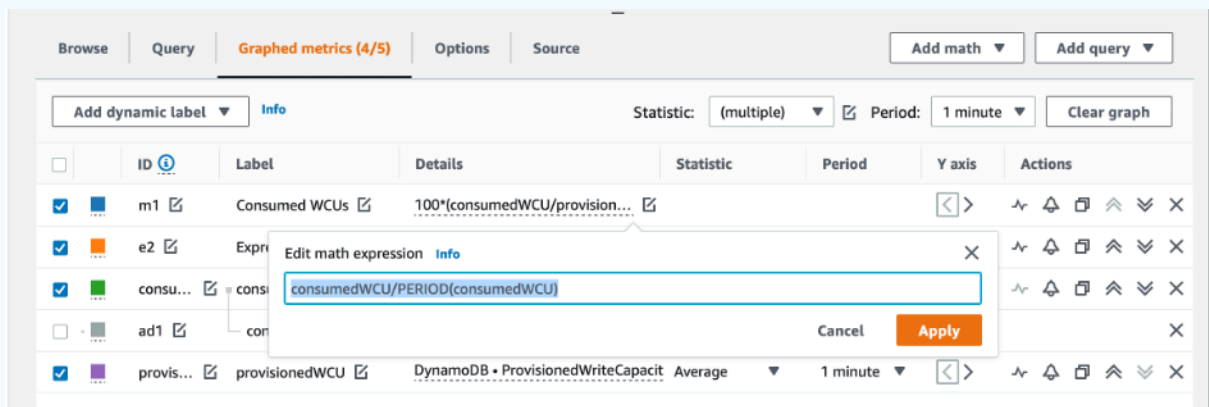
ID	Label	Details	Statistic	Period	Y axis	Act
e1	Expression1	$100 * (\text{consumedWCU} / \text{provisionedWCU})$				
e2	Expression2	$100 * (\text{consumedWCU} / \text{provisionedWCU})$				
consumedWCU	consumedWCU	DynamoDB • ConsumedWriteCapacity	Sum	1 minute		
ad1	consumedWCU (e...)	ANOMALY_DETECTION_BAND(...)				
provisionedWCU	ProvisionedWriteCapacity	DynamoDB • ProvisionedWriteCapacity	Average	1 minute		

## 11. 選取 Expression1 標籤，並將值更新為 m1，將標籤更新為 Consumed WCUs。

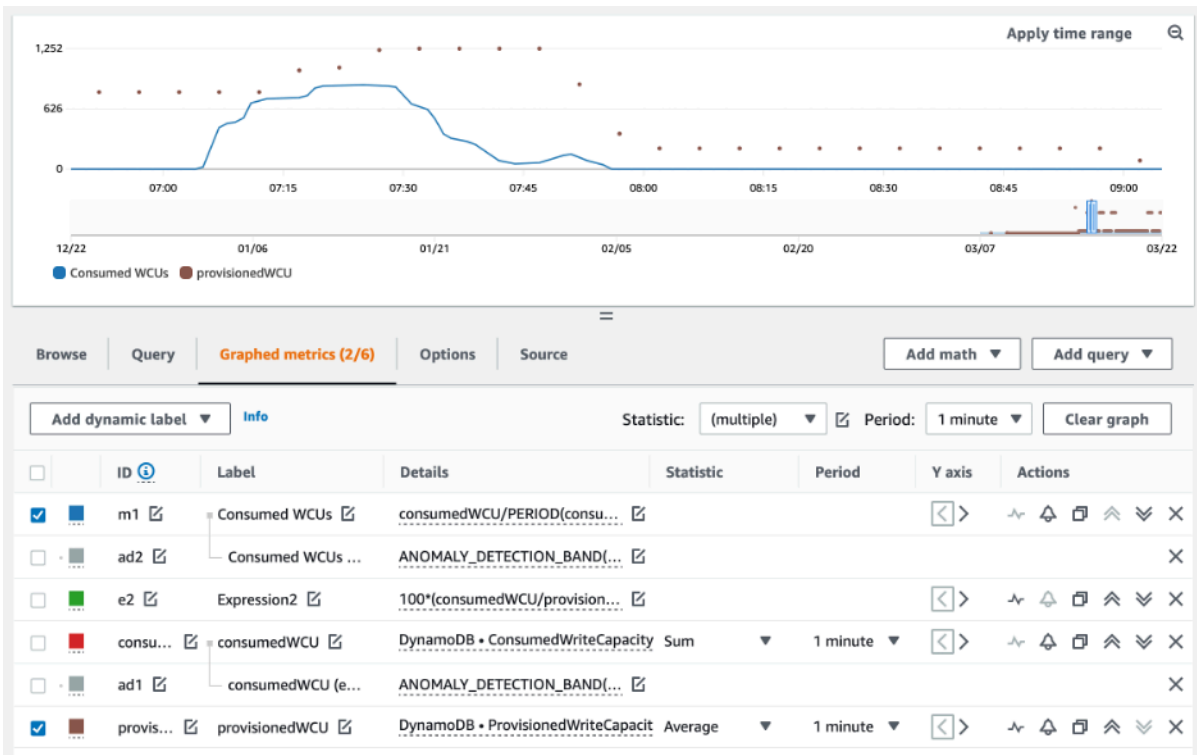


### Note

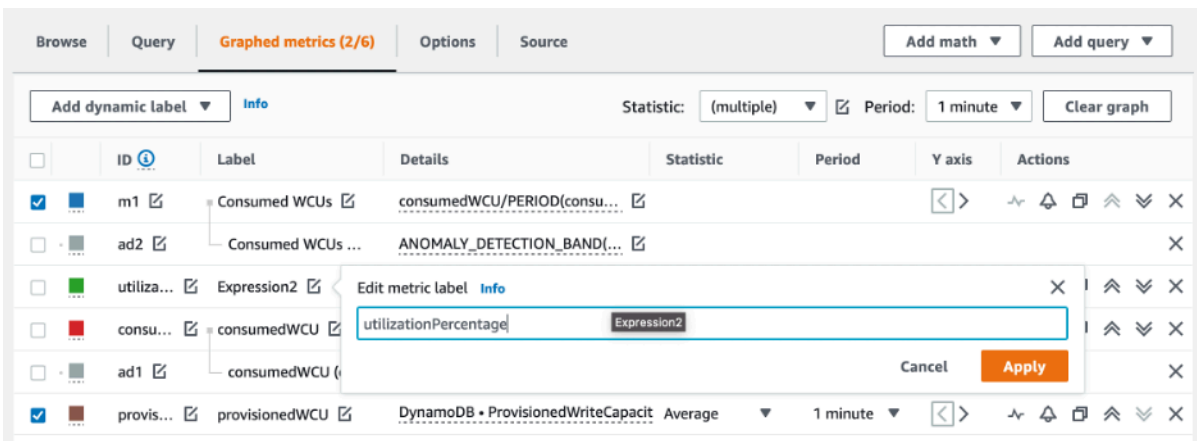
請確認您只選取 m1 (左側的核取方塊) 及 provisionedWCU，適當地視覺化資料。按一下 Details (詳細資料)，並將公式變更為  $\text{consumedWCU}/\text{PERIOD}(\text{consumedWCU})$ ，以更新公式。此步驟也可能產生另一個 ANOMALY\_DETECTION\_BAND 指標，但針對此程序的範圍，我們可加以忽略。



## 12. 您現在應有兩個圖形：一個表示資料表上已佈建的 WCU，另一個表示已耗用的 WCU。圖形的形狀可能與下方形狀不同，但您可以將其做為參考：



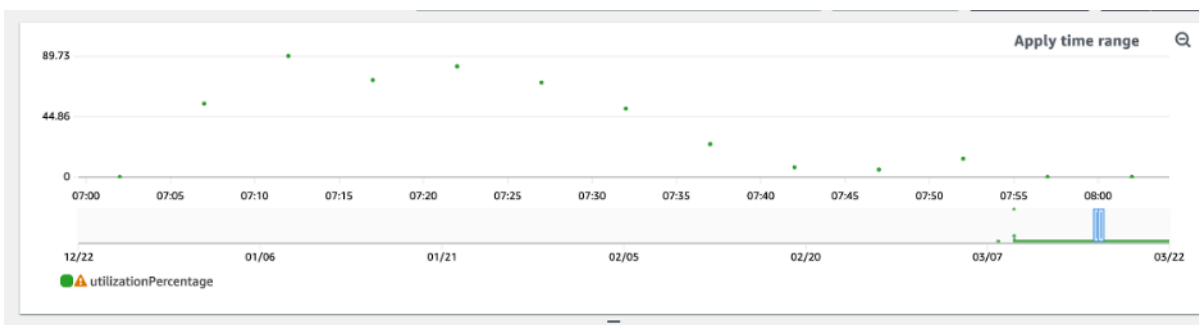
13. 選取 Expression2 圖形 (e2) 以更新百分比公式。將標籤和 ID 重新命名為 utilizationPercentage。將公式重新命名，以符合  $100*(m1/provisionedWCU)$ 。



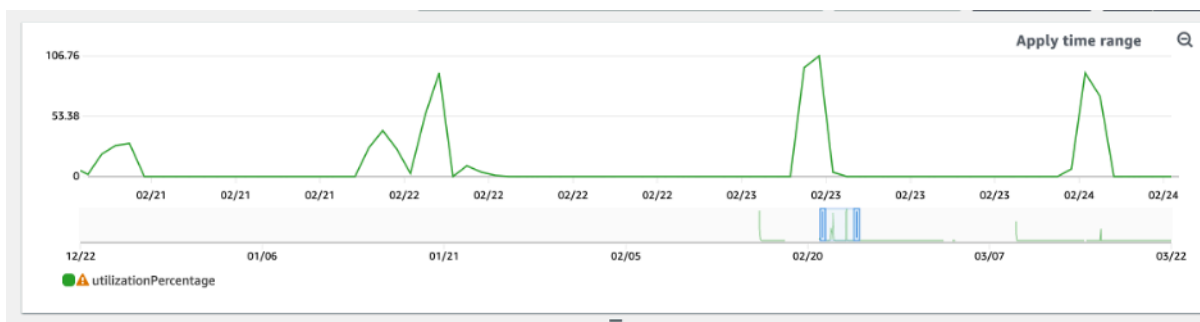


The screenshot shows the Amazon CloudWatch console interface. At the top, there are tabs for 'Browse', 'Query', 'Graphed metrics (2/6)', 'Options', and 'Source'. Below these are controls for 'Add dynamic label', 'Statistic: (multiple)', 'Period: 1 minute', and 'Clear graph'. A table lists several metrics with columns for ID, Label, Details, Statistic, Period, Y axis, and Actions. An 'Edit math expression' dialog box is open over the 'consumedWCU' metric, displaying the expression  $100 * (m1 / \text{provisionedWCU})$ . The dialog has 'Cancel' and 'Apply' buttons.

14. 除了 utilizationPercentage 之外，從所有指標移除核取方塊，視覺化您的使用率模式。預設間隔設為 1 分鐘，但您可以根據需要修改。



這是一個更長時間及 1 小時的視圖。您可以看到有些間隔使用率高於 100%，但這個特定的工作負載具有較長的間隔，使用率為零。



您可能得到與此範例中的圖片不同的結果。取決於您工作負載中的資料。使用率超過 100% 的間隔容易發生限流。DynamoDB 提供 [高載容量](#)，但是一旦高載容量完成，高於 100% 的任何項目都會受到限流。

## 如何識別佈建不足的 DynamoDB 資料表

針對多數工作負載，當資料表持續耗用超過其佈建容量的 80%，就會被視為佈建不足。

**高載容量**是一項 DynamoDB 功能，可讓客戶暫時耗用比原先佈建更多的 RCU/WCU (超過資料表定義的每秒佈建輸送量)。高載容量是為了吸收由於特殊事件或使用量尖峰而突增的流量。高載容量不會一直持續執行。一旦耗盡未使用的 RCU 和 WCU，若您嘗試耗用超過佈建的容量，就會受到限流。當應用程式流量接近 80% 使用率時，限流風險就會大幅提高。

80% 使用率規則會因資料的季節性和流量成長而有所不同。請考量下列情況：

- 如果您的流量在過去 12 個月內穩定保持約 90% 的使用率，那麼您的資料表容量剛好
- 如果您的應用程式流量在 3 個月內以每月 8% 的速度增加，您將達到 100%
- 如果您的應用程式流量在略多於 4 個月的期間以 5% 的速度增加，您仍會達到 100%

上述查詢結果能讓您得知使用率的情況。您可以使用這些結果做為指引，進一步評估其他指標，以協助您根據需要增加資料表容量 (例如：每月或每週成長率)。與您的營運團隊合作，為工作負載和資料表定義合理的百分比。

在部分特殊情況下，每天或每週進行分析時，資料會出現偏斜。例如，如果季節性應用程式在工作時間內使用量激增 (但在工作時間以外降到幾乎為零)，您可以透過[排程 auto 調整規模](#)，在其中指定一天中的小時 (以及星期幾) 以增加佈建容量以及何時減少容量，從而獲益。如果您的季節性不太明顯，您也可以從 [DynamoDB 表格 auto 擴展](#) 配置中受益，而不是瞄準更高的容量以滿足繁忙時間。

#### Note

當您為基礎資料表建立 DynamoDB 自動擴展組態時，請記得將任何與資料表相關的 GSI 加入其他組態。

## 如何識別過度佈建的 DynamoDB 資料表

從以上指令碼獲得的查詢結果提供了執行部分初始分析所需的資料點。若您的資料集在數個間隔內顯示的使用率低於 20%，表示資料表可能過度佈建。若要進一步判斷是否需減少 WCU 和 RCU 數量，您應重新檢視間隔中的其他讀數。

當您的資料表包含數個較低的使用間隔時，您可以從使用 auto 資源調整規模政策中獲益，無論是排定 auto 調度資源調整規模，或只針對以使用率為基礎的資料表設定預設的 auto 調整資源調度

如果您的工作負載使用率低到高節流率 (間隔中的 Max (ThrottleEvents) /Min (ThrottleEvents))，則當您的工作負載非常尖銳，其中流量在某些日子 (或數小時) 增加很多時，可能會發生這種情況，但通常流量一直很低。在這些情況下，使用[排程的 auto 調整](#)可能會有所幫助。

AWS [Well-Architected 的架構](#) 可協助雲端架構師為各種應用程式和工作負載建置安全、高效能、彈性和高效率的基礎架構。AWS Well-Architected 以六個支柱為基礎，包括卓越營運、安全性、可靠性、效能效率、成本最佳化和永續性，為客戶和合作夥伴提供一致的方法，以評估架構並實作可擴充的設計。

AWS [Well-Architected 的鏡頭](#) 將「Well-Architected」提供的指南擴展到特定的行業和技術領域。Amazon DynamoDB Well-Architected Lens 著重於 DynamoDB 工作負載。它提供最佳實務、設計原則和問題，以評估並檢閱 DynamoDB 工作負載。完成 Amazon DynamoDB Well-Architected Lens 檢閱，將為您提供有關建議設計原則的教育和指引，因為它與每個 AWS Well-Architected 支柱相關。本指引是以我們與不同產業、客群、規模和地區之客戶合作的經驗為基礎。

作為 Well-Architected Lens 檢閱的直接結果，您將收到可行建議的摘要，以最佳化並改善 DynamoDB 工作負載。

## 進行 Amazon DynamoDB Well-Architected Lens 檢閱

DynamoDB Well-Architected 的鏡頭審查通常由 AWS 解決方案架構師與客戶一起執行，但也可以由客戶作為自助服務來執行。儘管建議您檢閱全部六個 Well-Architected 支柱，作為 Amazon DynamoDB Well-Architected Lens 的一部分，但您也可以決定先將重點放在一或多個支柱上。

有關執行 Amazon DynamoDB Well-Architected 的鏡頭審查的其他資訊和指示，請參閱 [此影片](#) 和 [DynamoDB Well-Architected 的鏡頭頁面](#)。GitHub

## Amazon DynamoDB Well-Architected Lens 的支柱

Amazon DynamoDB Well-Architected Lens 是以六個支柱為中心而建置的：

### 效能效率支柱

效能效率支柱包括能夠有效率地使用運算資源，以滿足系統需求，並隨著需求變更與技術發展來保持該效率需求。

此支柱的主要 DynamoDB 設計原則是以 [建模資料](#)、[選擇分割區索引鍵](#) 和 [排序索引鍵](#)，以及根據應用程式存取模式 [定義次要索引](#) 為主。其他考量事項包括為工作負載選擇最佳輸送量模式、進行 AWS SDK 調整，以及在適當的情況下使用最佳快取策略。若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 效能效率支柱的 [深入探討影片](#)。

### 成本最佳化支柱

成本最佳化支柱著重於避免不必要的成本。

重要主題包括了解和控制花錢之處、選取最合適且正確數量的資源類型、分析一段時間後的花費、設計資料模型以最佳化應用程式特定存取模式的成本，以及擴展以符合業務需求而不會超支。

DynamoDB 的關鍵成本最佳化設計原則是以下列動作為主：為表格選擇最合適的容量模式和資料表類別，以及使用隨需容量模式或佈建容量模式搭配自動擴展來避免過度佈建容量。其他考量事項包括有效率的資料建模和查詢，以減少耗用的容量量、以折扣價保留部分已消耗容量、將項目大小降到最小、識別和移除未使用的資源，以及使用 [TTL](#) 免費自動刪除過時資料。若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 成本最佳化支柱的[深入探討影片](#)。

如需 DynamoDB 成本最佳化最佳實務的其他資訊，請參閱[成本最佳化](#)。

### 卓越營運支柱

卓越營運支柱著重於執行和監控系統，以提供商業價值並持續改善流程和程序。重要主題包括自動化變更、回應事件，以及定義管理日常作業的標準。

DynamoDB 的主要卓越營運設計原則包括透過 Amazon 監控 DynamoDB 指標，以 CloudWatch AWS Config 及在違反預先定義閾值或偵測到不合規規則時自動發出警示和修復。其他考量為透過基礎架構即程式碼定義 DynamoDB 資源，以及利用標籤來改善 DynamoDB 資源的組織、識別和成本會計。若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 卓越營運支柱的[深入探討影片](#)。

### 可靠性支柱

可靠性支柱著重於確保工作負載如預期般正確且一致地執行其預期功能。彈性工作負載可快速從失敗中復原，以滿足業務和客戶需求。重要主題包括分散式系統設計、復原規劃，以及如何處理變更。

DynamoDB 的基本可靠性設計原則圍繞根據您的 RPO 和 RTO 需求選擇備份策略和保留，針對多區域工作負載使用 DynamoDB 全域表或具有低 RTO 的跨區域災難復原案例，透過在 AWS SDK 中設定和使用這些功能來實作重試邏輯，並在應用程式中透過 Amazon 自動修復和預先定義的警示監控 DynamoDB 指標。CloudWatch 若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 可靠性支柱的[深入探討影片](#)。

### 安全支柱

安全支柱著重於保護資訊和系統。重要主題包括資料的機密性和完整性、識別和管理誰可以透過權限管理做什麼、保護系統，以及建立控制項來偵測安全事件。

DynamoDB 的主要安全設計原則是使用 HTTPS 加密傳輸中的資料、選擇靜態資料加密的金鑰類型，以及定義 IAM 角色和政策，以驗證、授權和提供 DynamoDB 資源的精細存取權。其他考量事項包括透過稽核 DynamoDB 控制平面和資料平面作業。AWS CloudTrail 若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 安全支柱的[深入探討影片](#)。

如需 DynamoDB 安全性的其他資訊，請參閱[安全性](#)。

## 永續性支柱

永續性支柱著重於將執行雲端工作負載時對環境的影響降到最低。重要主題包括永續性的共同責任模式、了解影響，以及充分利用以將所需資源降至最少，並減少下游影響。

DynamoDB 的主要永續性設計原則包括識別和移除未使用的 DynamoDB 資源、搭配自動擴展使用隨需容量模式或佈建容量模式來避免過度佈建容量、有效查詢以減少耗用的容量，以及透過壓縮資料和使用 TTL 刪除過時的資料來減少儲存體使用量。若要進一步了解這些設計原則，請觀看這部有關 DynamoDB Well-Architected Lens 永續性支柱的[深入探討影片](#)。

## 有效設計和使用分區索引鍵的最佳實務

唯一識別 Amazon DynamoDB 資料表中每個項目的主索引鍵，可以是簡易 (僅分割區索引鍵) 或複合 (分割區索引鍵與排序索引鍵的組合)。

一般來說，您應該設計您的應用程式，使其在資料表中的所有邏輯分割區索引鍵及其次要索引中統一進行活動。您可以確定應用程式需要的存取模式，並估計每個資料表和次要索引所需的讀取和寫入單位。

根據預設，資料表中的每個分割區都會盡可能提供 3,000 個 RCU 和 1,000 WCU 的完整容量。資料表中所有分割區的總輸送量可能受制於佈建模式下的佈建輸送量或隨選模式下的資料表層級輸送量。如需詳細資訊，請參閱 [Service Quotas](#)。

### 主題

- [設計分割區索引鍵以分配您的工作負載](#)
- [使用寫入碎片以平均分佈工作負載](#)
- [在資料上傳期間有效地分配寫入活動](#)

## 設計分割區索引鍵以分配您的工作負載

資料表主索引鍵的分割區索引鍵部分，決定了已儲存資料表資料的邏輯分割區。因此會影響底層的實體分割區。不能有效分配 I/O 請求的分割區索引鍵設計可能會造成「經常性」分割區，因此導致限流並且無法有效地使用已佈建的 I/O 容量。

資料表已佈建傳輸量的最佳使用情況，不僅取決於各個項目的工作負載模式，還取決於分割區索引鍵設計。這並不表示您必須存取所有分割區索引鍵值才能達到有效率的傳輸量層級，也不表示所存取之分割區索引鍵值的百分比必須很高。這表示您的工作負載存取的分割區索引鍵值越獨特，這些請求將越多地

分佈在已分割區空間中。一般而言，隨著所存取的分割區索引鍵值與分割區索引鍵值總數的比率成長，您會更有效率地利用已佈建的傳輸量。

以下是一些常用分割區索引鍵結構描述的已佈建傳輸量效率比較。

分割區索引鍵值	一致性
使用者 ID，其中應用程式有許多使用者。	好
狀態碼，其中只有一些可能的狀態碼。	不好
項目建立日期，捨入到最接近的時間週期 (例如日期、小時、分鐘)。	不好
裝置 ID，其中每部裝置會依相當類似的間隔存取資料。	好
裝置 ID，其中即使追蹤多個裝置，其中一部裝置也會明顯比其他所有裝置更熱門。	不好

如果單一資料表只有少量的分割區索引鍵值，請考慮在更多相異分割區索引鍵值之間分佈您的寫入操作。換句話說，請結構化主索引鍵元素，以避免某個「經常性」(高度請求) 的分割區索引鍵值使整體效能變慢。

例如，請考慮使用具有複合主索引鍵的資料表。分割區索引鍵代表項目的建立日期，已捨入到最接近的日期。排序索引鍵是項目識別符。在指定的日期 (假設是2014-07-09)，所有新項目都會寫入單一分割區索引鍵值 (和對應的實體分割區)。

如果資料表完全符合單一分割區 (考慮到資料會隨時間成長)，而且您應用程式的讀取與寫入輸送量需求不超過單一分割區的讀取與寫入能力，則您的應用程式應不會遇到由於分割所造成的非預期調節。

若要使用適用於 DynamoDB 的 NoSQL Workbench 來協助視覺化您的分割區索引鍵設計，請參閱 [使用 NoSQL Workbench 建立資料模型](#)。

## 使用寫入碎片以平均分佈工作負載

在 Amazon DynamoDB 中的分割區索引鍵空間分配寫入的一個好方法是拓展空間。您可以數種不同的方式來執行此動作：您可以在分割區索引鍵值中新增隨機數字，以便在分割區之間分配項目。或者，您可以使用根據您查詢內容計算的數字。



## 使用隨機尾碼的碎片

要在分割區索引鍵空間中更平均地分佈負載的方法，即是在分割區索引鍵值的結尾新增一個亂數。如此您就能在較大的空間中將寫入隨機化。

例如，對於代表今天日期的分割區索引鍵，您可以選擇 1 到 200 間的一個亂數，並將其串連到日期做為尾碼。這會產生分割區索引鍵值，例如 2014-07-09.1、2014-07-09.2... 到 2014-07-09.200。因為您隨機化了分割區索引鍵，每天對資料表的寫入會平均分配在多個分割區中。這可帶來更優良的平行處理與更高的整體傳輸量。

然而，若要讀取指定日期的所有項目，您需要查詢所有尾碼的項目，接著合併結果。例如，您會先對分割區索引鍵值 2014-07-09.1 發出 Query 請求。再對 2014-07-09.2 發出另一個 Query，依此類推直到 2014-07-09.200。最後，您的應用程式需要合併所有該等 Query 請求的結果。

## 使用計算尾碼的碎片

隨機化的策略可大幅改善寫入傳輸量。由於您不知道寫入項目時所使用的尾碼值，因此很難讀取特定項目。若要讓讀取個別項目變得比較容易，您可以使用不同的策略。不使用亂數以在分割區中分佈項目，而是使用您可以計算的數字 (根據您想要查詢之項目)。

考量先前的範例，其中資料表會使用分割區索引鍵中的今天日期。現在假設每個項目有可存取的 OrderId 屬性，且您經常需要依訂單 ID (除了日期) 來尋找項目。在您的應用程式將項目寫入資料表之前，可根據訂單 ID 計算雜湊尾碼並將其附加至分割區索引鍵日期。計算應會產生介於 1 到 200 之間的數字，此分佈非常平均，與隨機策略產生的結果相當類似。

一個簡單的計算便足以說明，例如訂單 ID 中字元之 UTF-8 字碼指標值的乘積：模數 200，+ 1。分割區索引鍵值會是與計算結果串聯的日期。

有了這項策略，即可在分割區索引鍵值之間，以及實體分割區之間平均分配寫入。您可以輕鬆地對特定項目與日期執行 GetItem 操作，因為您可以計算對特定 OrderId 值的分割區索引鍵值。

若要讀取指定日期的所有項目，您仍然需要 Query 每個 2014-07-09.N 索引鍵 (其中 N 是 1-200)，而且您的應用程式需要合併所有結果。好處是您能避免讓單一「經常性」分割區索引鍵值承受所有工作負載。

### Note

如需為了處理大量時間序列資料而設計的有效策略，請參閱 [時間序列資料](#)。

## 在資料上傳期間有效地分配寫入活動

通常，當您從其他資料來源載入資料時，Amazon DynamoDB 會將表格資料分割在多個伺服器上。如果您同時將資料上傳到所有已配置的伺服器，就能獲得更好的效能。

例如，假設您想要將使用者訊息上傳至使用複合主索引鍵的 DynamoDB 資料表，該資料表將 `UserID` 作為分割區索引鍵並將 `MessageID` 作為排序索引鍵。

當您上傳資料時，您可以採取的一種方法是一個接一個地上傳每位使用者的所有訊息項目：

UserID	MessageID
U1	1
U1	2
U1	...
U1	...最多可達 100
U2	1
U2	2
U2	...
U2	...最多可達 200

這個案例中的問題是您沒有將寫入請求分配到跨分割區索引鍵值的 DynamoDB。您一次需要一個分割區索引鍵值並上傳其所有項目，再轉到下一個分割區索引鍵值並執行相同操作。

在幕後，DynamoDB 會跨多個伺服器分割資料表中的資料。若要充分使用為資料表佈建的所有輸送容量，您必須將工作負載分配至各分割區索引鍵值。若您將不均勻數量的上傳工作導向所有具有相同分割區索引鍵值的項目，便無法完全使用 DynamoDB 為資料表佈建的所有資源。

您可以使用排序索引鍵，從每個分割區索引鍵值載入一個項目，再從每個分割區索引鍵值載入另一個項目，依此類推：



UserID	MessageID
U1	1
U2	1
U3	1
...	...
U1	2
U2	2
U3	2
...	...

此序列中的每次上傳都會使用不同的分割區索引鍵值，讓更多 DynamoDB 伺服器同時處於忙碌狀態，並改善您的輸送量效能。

## 使用排序索引鍵以組織資料的最佳實務

在 Amazon DynamoDB 表格中，唯一識別表格中每個項目的主索引鍵可由分區索引鍵和排序索引鍵組成。

精心設計的排序索引鍵有兩個主要優點：

- 它們將相關資訊集中在一個可以有效率查詢的地方。仔細設計的排序索引鍵，讓您可以透過例如 `begins_with`、`between`、`>`、`<` 等運算子，使用範圍查詢來擷取經常需要的相關項目群組。
- 複合排序索引鍵可讓您在資料中定義階層式 (one-to-many) 關係，以便在階層的任何層級進行查詢。

例如，在列出地理位置的表中，您可以按以下方式建構排序索引鍵。

```
[country]#[region]#[state]#[county]#[city]#[neighborhood]
```

這可以讓您在任何一種此彙整層級的位置列表進行有效率的範圍查詢，從 `country` 到一個 `neighborhood`，以及其之間的所有內容。

## 使用排序索引鍵進行版本控制

許多應用程式需要維護項目層級修訂的歷史記錄，以用於稽核或合規目的，並能輕鬆擷取最新版本。有一種有效的設計模式，是使用排序索引鍵字首來完成此操作：

- 在每個新項目建立該項目的兩個複本：一個複本在排序索引鍵字首應有版本號的字首為零 (例如 v0\_)，並且應有一個版本號的字首為一 (v1\_)。
- 每次更新項目時，都使用更新版本的排序索引鍵中更高版本的字首，並將更新的內容複製到版本字首為零的項目中。這代表任何項目的最新版本都可以使用零字首輕鬆定位。

例如，零件製造商可能會使用如下圖所示的結構描述。

Primary Key		Data-Item Attributes...								
Partition Key	Sort Key	Attribute 1		Attribute 2		Attribute 3		Attribute 4		...
Equipment_ID	(varies)									
Equipment_1	Details	Name: Biophasic Cardiometer <i>(equipment name)</i>	Factory_ID: S14_Tukwilla <i>(factory where manufactured)</i>	Line_ID: R_7 <i>(assembly-line ID)</i>						
	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.				

Equipment\_1 項目會經過各種稽核人員的一系列稽核。每個新稽核的結果都會在資料表中的新項目中擷取，從第一號版本開始，然後遞增每個連續修訂的編號。

增加每個新修訂版本時，應用程式層會將零版本項目 (具有等於 v0\_Audit 的排序索引鍵) 的內容替換為新版本的內容。

無論應用程式何時需要擷取最近的稽核狀態，都可以查詢排序索引鍵開頭的 v0\_。

如果應用程式需要擷取整個修訂歷史記錄，可查詢該項目分割區索引鍵下的所有項目並篩選掉 v0\_ 項目。

如果您在排序索引鍵開頭之後的排序索引鍵中包括個別部分 ID，此設計也適用於跨設備多個部分的稽核。

# 使用 DynamoDB 中次要索引的最佳實務

次要索引對於支援您應用程式所需的查詢模式而言通常很重要。同時，過度使用次要索引或未有效地使用次要索引可能會增加非必要的成本並降低效能。

## 內容

- [DynamoDB 中次要索引的一般準則](#)
  - [有效率地使用索引](#)
  - [謹慎選擇投影](#)
  - [最佳化頻繁查詢，避免擷取](#)
  - [在建立本機次要索引時請留意項目集合大小限制](#)
- [利用疏鬆索引](#)
  - [DynamoDB 中疏鬆索引的範例](#)
- [對具體化彙總查詢使用全域次要索引](#)
- [多載全域次要索引](#)
- [針對選擇性資料表查詢使用全域次要索引寫入碎片](#)
- [使用全域次要索引建立最終一致複本](#)

## DynamoDB 中次要索引的一般準則

Amazon DynamoDB 支援兩種次要索引：

- 全域次要索引 (GSI)：是包含分割區索引鍵或排序索引鍵的索引，這些索引鍵可與基底資料表上的索引鍵不同。全域次要索引之所以視為全域，是因為索引上的查詢可以跨越所有分割區之間基礎資料表中的所有資料。全域次要索引沒有大小限制，也有自己的讀取和寫入活動佈建輸送量設定，而且這些設定與資料表的佈建輸送量設定無關。
- 本機次要索引 (LSI)：與基底資料表擁有相同分區索引鍵但不同排序索引鍵的索引。本機次要索引的「本機」概念是指，本機次要索引的每個分割區都會列入基礎資料表分割區的範圍，其中分割區索引鍵的值皆相同。因此，針對每個分割區索引鍵值，所有已索引項目的大小總計不得超過 10 GB。此外，本機次要索引與其建立索引的資料表共用適用於讀取和寫入活動的佈建輸送量設定。

DynamoDB 中的每個資料表最多可以有 20 個全域次要索引 (預設配額) 以及 5 個本機次要索引。

全域次要索引通常比本機次要索引更實用。決定要使用的索引類型也將取決於應用程式的需求。如需全域次要索引與本機次要索引的比較，以及如何在它們之間進行選擇的詳細資訊，請參閱[the section called “使用索引”](#)。

以下是在 DynamoDB 中建立索引時需要謹記在心的一些常見原則與設計模式：

## 主題

- [有效率地使用索引](#)
- [謹慎選擇投影](#)
- [最佳化頻繁查詢，避免擷取](#)
- [在建立本機次要索引時請留意項目集合大小限制](#)

## 有效率地使用索引

將索引數量降到最低。請不要在您不常查詢的屬性上建立次要索引。不常使用的索引不僅會增加儲存體和 I/O 成本，而不會改善應用程式的效能。

## 謹慎選擇投影

由於次要索引會使用儲存體和佈建輸送量，建議您將索引的大小維持得越小越好。此外，相較於查詢完整資料表，索引的大小越小所能帶來的效能優勢越大。若您的查詢通常只會傳回一小部分的屬性，且這些屬性的總大小遠小於整個項目的大小，建議您只投影會定期請求的屬性。

相較於讀取，若您預期資料表上將會有許多寫入活動，遵循這些最佳實務：

- 考慮投影少量的屬性，以將寫入到索引的項目大小減至最小。但是，此作法僅適用於若投影屬性大小大於單一寫入容量單位 (1 KB)。例如，若一個索引項目的大小只有 200 位元組，DynamoDB 會將此大小向上四捨五入為 1 KB。換句話說，只要索引項目很小，您便可以投影更多屬性，而無須支付額外的成本。
- 避免投影您已知在查詢中很少會需要用到的屬性。每次您更新在索引中投影的屬性後，更新索引也不會產生額外的成本。您仍可以更高佈建傳輸量成本在 Query 中擷取非投影屬性，但查詢成本可能會比經常更新索引的成本少很多。
- 只有在您希望查詢傳回不同排序索引鍵排序的整個資料表項目，才指定 ALL。為所有屬性建立投影之後便不再需要擷取資料表，但在大多數的案例中，這樣會讓您的儲存體和寫入活動成本加倍。

如下一區段所述，平衡盡可能將索引大小降到最低的需求來達成將擷取減到最低的需求。

## 最佳化頻繁查詢，避免擷取

若要在延遲盡可能最低的前提下取得最快速的查詢，請投影所有您預期該查詢要傳回的屬性。特別是，如果您對未投影的屬性查詢本機次要索引，DynamoDB 會自動從需要在資料表讀取整個項目的資料表擷取那些屬性。此引發的延遲與其他 I/O 操作是可以避免的。

請謹記「偶爾」的查詢可能經常是「必要」的查詢。若因為您預計偶爾才會查詢某些屬性，而不打算投影，請考慮可能的結果，否則您可能會後悔沒有投影那些屬性。

如需資料表擷取的詳細資訊，請參閱 [本機次要索引的佈建輸送量考量](#)。

## 在建立本機次要索引時請留意項目集合大小限制

項目集合是資料表和其本機次要索引中所有具有相同分割區索引鍵的項目。由於項目集合的大小不可超過 10 GB，可能會耗盡特定分割區索引鍵值的空間。

當您新增或更新資料表項目時，DynamoDB 會更新任何受到影響的本機次要索引。若建立索引的屬性已在資料表中定義，本機次要索引便會一同成長。

當您建立本機次要索引時，建議您思考會有多少資料寫入索引，以及那些資料項目中有多少具有相同的分割區索引鍵。若您預期具有特定分割區索引鍵值的資料表和索引項目總和可能會超過 10 GB，請考慮是否要避免建立索引。

若您無法避免建立本機次要索引，您必須預計項目集合的大小上限，並在超過之前採取動作。最佳作法是，在撰寫項目時，您應該利用 [ReturnItemCollectionMetrics](#) 參數來監視和警示接近 10GB 大小限制的項目集合大小。超過最大項目集合大小將導致寫入嘗試失敗。您可以在項目收集大小影響您的應用程式之前監視和警示項目集合大小，以減輕項目集合大小問題。

### Note

建立之後，您就無法刪除本機次要索引。

如需取得在限制內操作及採取正確動作的策略，請參閱「[項目集合大小限制](#)」。

## 利用疏鬆索引

針對資料表中的任何項目，DynamoDB 在索引的排序索引鍵值存在於項目中時，才會寫入相對應的索引項目。若排序索引鍵並未出現在每個資料表項目中，或分割區索引鍵並未出現於項目中，則該索引便可稱為疏鬆。

稀疏索引在對小型子區塊的資料表進行查詢時很有用。例如，假設您有一個資料表，其中存放您所有的客戶訂單並內含以下索引鍵屬性：

- 分區索引鍵: `CustomerId`
- 排序索引鍵：`OrderId`

若要追蹤開啟的訂單，您可以在尚未運送的項目訂單中插入名為 `isOpen` 的屬性。然後，隨著訂單運送時，您可以刪除該屬性。當您在 `CustomerId` (分割區索引鍵) 和 `isOpen` (排序索引鍵) 上建立索引時，只有已定義 `isOpen` 的那些訂單會顯示在其中。當您有上千筆訂單，但只有部分的訂單處於開啟狀態，對開啟訂單查詢索引會比掃描整個資料表快得多且節省更多費用。

相較於使用 `isOpen` 之類的類型屬性，您可以使用對索引排序順序有用的屬性值。例如，您可以使用設定為訂單下訂日期的 `OrderOpenDate` 屬性，接著在訂單履行後將其刪除。如此一來，在您查詢稀疏索引時，系統會傳回依訂單下訂的日期所排序的項目。

## DynamoDB 中疏鬆索引的範例

全域次要索引依預設是稀疏的。在您建立全域次要索引時，指定一個分割區索引鍵和 (選擇性) 一個排序索引鍵。只有在基本資料表中含有那些屬性的項目會出現在索引中。

透過將全域次要索引設計為稀疏，您可以使用低於基本資料表的寫入傳輸量來佈建該索引，同時仍達成優異的效能。

例如，遊戲應用程式可能會追蹤每位使用者的所有比分，但一般來說只需要查詢少數較高的計分。以下設計模式能有效處理這種情況：

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 <i>(game score)</i>	Date: 2017-11-14 <i>(date of game)</i>		
	Game_2	Score: 69,450 <i>(game score)</i>	Date: 2017-12-31 <i>(date of game)</i>		
	Game_3	Score: 135,900 <i>(game score)</i>	Date: 2018-01-19 <i>(date of game)</i>	Award: Champ <i>(type of award)</i>	
Padma	Game_4	Score: 25,350 <i>(game score)</i>	Date: 2018-01-27 <i>(date of game)</i>		
	Game_5	Score: 69,450 <i>(game score)</i>	Date: 2028-01-19 <i>(date of game)</i>		
	Game_6	Score: 147,300 <i>(game score)</i>	Date: 2018-02-02 <i>(date of game)</i>	Award: Champ <i>(type of award)</i>	
	Game_7	Score: 169,100 <i>(game score)</i>	Date: 2018-03-10 <i>(date of game)</i>	Award: Champ <i>(type of award)</i>	

在此範例中，Rick 玩了三款遊戲並在其中一款遊戲中達到 Champ 狀態。Padma 玩了四款遊戲並在其中兩款遊戲中達到 Champ 狀態。請注意，Award 屬性僅在使用者獲得獎項時的項目中才會出現。關聯的全域次要索引看起來應如下：

GSI	Primary Key		Projected Attributes...		
	Partition Key	Sort 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

全域次要索引僅包含常受到查詢的高計分，也就是基本資料表中的小子集項目。

## 對具體化彙總查詢使用全域次要索引

快速變更的資料中維護接近即時彙總和索引鍵指標，對於企業來說變得越來越重要。例如，音樂資料庫可能想要以接近即時的速度展示最常下載的歌曲。

考慮下列音樂資料庫資料表配置：



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>					

此範例的資料表會使用 songID 做為分割區索引鍵來儲存歌曲。您可以在此資料表上啟用 Amazon DynamoDB Streams 並將 Lambda 函數附加到該串流中，如此隨著每個歌曲的下載，系統會使用 Partition-Key=SongID 和 Sort-Key=DownloadID 來將項目新增至資料表。隨著這些更新的進行，更新會在 DynamoDB Streams 中觸發 Lambda 函數。Lambda 函數可能會彙總並依照 songID 分組下載並更新頂層項目、Partition-Key=songID 和 Sort-Key=Month。請記住，如果 lambda 執行在寫入新的彙總值之後失敗，則可能會重試並將該值多次彙總，留下近似值。

若要以單位毫秒延遲，接近即時的速度讀取更新，則使用全域次要索引，內含查詢條件 Month=2018-01、ScanIndexForward=False、Limit=1。

在此使用的另一個最佳化方式是將全域次要索引定為稀疏索引，且僅在需要查詢即時擷取資料的項目時才供使用。在需要排名前十流行歌曲或在該月下載的任何歌曲的資訊時，全域次要索引可以做為額外工作流程。

## 多載全域次要索引

雖然 Amazon DynamoDB 每個資料表具有 20 個全域次要索引的預設配額，實際上，您可以在遠超過 20 個以上的資料欄位間進行索引。與在關聯式資料庫管理系統 (RDBMS) 中的資料表不同 (其中結構描述是統一的)，DynamoDB 中的資料表可以一次保留許多不同種類的資料項目。此外，在不同項目中的相同屬性可能包含完整不同種類的資訊。

考量 DynamoDB 資料表配置的下列範例，該資料表中會儲存各種不同的資料。



Primary Key		Data-Item Attributes...				
Partition Key	Sort Key	Attribute 1		Attribute 2	...	
HR-974 <i>(employee ID)</i>	Employee_Name	Data:	Murphy, John <i>(employee name)</i>	Start:	2008-11-08 <i>(start date)</i>	...etc.
	YYYY-Q1	Data:	\$5,477 <i>(order totals in USD)</i>	Name:	Murphy, John <i>(employee name)</i>	
	HR_confidential	Data:	2008-11-08 <i>(hire date)</i>	Name:	Murphy, John <i>(employee name)</i>	...etc.
	Warehouse_01	Data:	Murphy, John <i>(employee name)</i>			
	v0_Job_title	Data:	Operator-1 <i>(job title)</i>	Start:	2008-11-08 <i>(start date)</i>	...etc.
	v1_Job_title	Data:	Operator-2 <i>(job title)</i>	Start:	2016-11-04 <i>(start date)</i>	...etc.
	v2_Job_title	Data:	Supervisor-1 <i>(job title)</i>	Start:	2017-11-01 <i>(start date)</i>	...etc.

Data 屬性 (這是所有項目共有的屬性) 會根據其父項目擁有不同的內容。如果您為使用資料表排序索引鍵做為其分割區索引鍵和 Data 屬性做為其排序索引鍵的資料表建立全域次要索引，您可以使用單一全域次要索引來進行多種不同的查詢。這些查詢可能包含以下項目：

- 使用 Employee\_Name 作為分割區索引鍵值並將員工的姓名 (例如 Murphy, John) 作為排序索引鍵值，即可在全域次要索引中按姓名查找員工。
- 使用全域次要索引，透過搜尋倉庫 ID (像是 Warehouse\_01) 來尋找在特定倉庫工作的所有員工。
- 取得最近的雇用清單，查詢在 HR\_confidential 上的全域次要索引作為分割區索引鍵值，並在排序索引鍵值中使用日期範圍。

## 針對選擇性資料表查詢使用全域次要索引寫入碎片

應用程式經常需要識別 Amazon DynamoDB 資料表中符合特定條件的一小部分項目子集。當這些項目隨機分配到資料表的分割區索引鍵時，您可以求助資料表掃描來檢索它們。該選項可能非常昂貴，但是當資料表中有大量物件符合搜索條件時，它的效果很好。但是，當索引鍵空間很大且搜索條件十分有選擇性時，這種策略可能會導致很多不必要的處理。

更好的解決方案是查詢資料。若要在整個索引鍵空間中啟用選擇性查詢，您可以透過新增包含 (0-N) 數值的屬性到您將用於全域次要索引分割區索引鍵的每個項目，以此使用寫入分片。

以下是在「嚴重事件」工作流程中使用的結構描述範例：

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.

使用此結構描述設計，事件項目會分配在 GSI 上的 0-N 分割區，允許使用組合索引鍵上的排序條件進行分散讀取，從而檢索指定時間段內具有給定狀態的所有項目。

此結構描述模式以最低的成本提供高度選擇性的結果集，而不需要掃描資料表。

## 使用全域次要索引建立最終一致複本

使用全域次要索引建立資料表的最終一致複本。建立複本可讓您執行下列動作：

- 為不同的讀取器設定不同的佈建讀取容量。例如，假設您有兩個應用程式：一個應用程式處理高優先順序查詢，並需要最高層級的讀取效能；另一個應用程式則處理低優先順序查詢，且容許調節讀取活動。

如果這兩個應用程式都從同一個資料表讀取資料，則來自低優先順序應用程式的高讀取負載可能會耗用資料表的所有可用讀取容量。這將調節高優先級應用程式的讀取活動。

相反地，您可以透過全域次要索引建立複本，其讀取容量可與資料表本身分開設定。然後，您可以讓低優先級的應用程式查詢複本而非資料表。

- 完全消除資料表中的讀取。例如，您可能有一個應用程式會從網站擷取大量點擊流活動，但您不想冒著讓讀取干擾該活動的風險。您可以隔離此資料表並避免其他應用程式讀取 (請參閱 [使用 IAM 政策條件進行精細定義存取控制](#))，同時讓其他應用程式讀取使用全域次要索引建立的複本。

若要建立複本，請設定具有與資料表相同的索引鍵結構描述的全域次要索引，並將部分或全部的非索引鍵屬性投影到其中。在應用程式中，您可以將部分或所有讀取活動導向此全域次要索引，而不是父資料表。然後，您可以調整全域次要索引的佈建讀取容量，以便在不必要變更父資料表的佈建讀取容量情況下處理該讀取。

寫入父資料表與寫入資料出現在索引中的時間之間，總是會有短暫的傳播延遲。換句話說，您的應用程式應該考慮到全域次要索引複本只是與父資料表有最終一致性。

您可以建立多個全域次要索引複本來支援不同的讀取模式。在建立複本時，只需投入每個讀取模式實際需要的屬性。然後，應用程式可以耗用較少佈建的讀取容量來僅取得所需的資料，而不必從父資料表讀取項目。隨著時間推移，這項最佳化可以大幅節省成本。

## 儲存大型項目和屬性的最佳實務

Amazon DynamoDB 會將您存放在表格中的每個項目的大小限制為 400 KB (請參閱 [Amazon DynamoDB 中的服務、帳戶和資料表配額](#))。若您的應用程式需要在項目中存放比 DynamoDB 大小限制所允許大小更多的資料，您可以嘗試壓縮一或多個大型屬性，或將項目分成多個項目 (即以排序索引鍵編製索引)。您也可以將項目作為物件存放在 Amazon Simple Storage Service (Amazon S3) 中，接著在您的 DynamoDB 項目中存放 Amazon S3 物件的識別符。

最佳作法是，在撰寫項目時應該利用 [ReturnConsumedCapacity](#) 參數來監視和警示接近 400 KB 項目大小上限的項目大小。超過最大項目大小將導致寫入嘗試失敗。監視和警示項目大小可讓您在項目大小問題影響您的應用程式之前緩解這些問題。

## 壓縮大型屬性值

壓縮大型屬性值可以讓它們符合 DynamoDB 中的項目限制，並減少您的儲存成本。壓縮演算法 (例如 GZIP 或 LZ0) 會產生二進位輸出，然後您可以將其儲存在項目內的 Binary 屬性類型中。

舉例來說，假設一個儲存由論壇使用者撰寫之郵件的表格。這些消息通常包含長字符串的文本，這些文本是壓縮的候選人。雖然壓縮可以減少項目大小，但缺點是壓縮的屬性值對於篩選沒有用處。

如需示範如何在 DynamoDB 中壓縮這類訊息的範本程式碼，請參閱以下內容：

- [範例：使用 AWS SDK for Java Document API 處理二進位類型屬性](#)
- [範例：使用 AWS SDK for .NET 低階 API 處理二進位類型屬性](#)

## 垂直分割

處理大型項目的替代解決方案是將它們分解為較小的數據塊，並通過分區鍵值關聯所有相關項目。然後，您可以使用排序索引鍵字符串來識別儲存在其旁邊的關聯資訊。這樣做，並讓多個項目由相同的分區索引鍵值分組，您正在創建一個 [項目集合](#)。

如需此方法的詳細資訊，請參閱：

- [在 Amazon DynamoDB 中使用垂直分割有效率地擴展資料](#)
- [使用以下方法在 Amazon DynamoDB 中實作垂直分割 AWS Glue](#)

## 在 Amazon S3 中存放大型屬性值

如先前所述，您可以使用 Amazon S3 來存放在 DynamoDB 項目中不符合的大型屬性值。您可以將它們作為物件存放在 Amazon S3 中，接著在您的 DynamoDB 項目中存放物件的識別符。

您也可以使用 Amazon S3 中的物件中繼資料支援，將連結提供回 DynamoDB 中的父項目。在 Amazon S3 中以物件 Amazon S3 中繼資料的形式存放項目的主索引鍵值。此通常可協助維護 Amazon S3 物件。

例如，假設在「ProductCatalog」一節中定義的 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 資料表。此資料表中的項目會存放項目價格、描述、書籍作者和其他產品維度的資訊。如

果您希望存放的每個產品影像太大而無法在項目中容納，您可以將影像存放在 Amazon S3 而不是在 DynamoDB 中。

實作此策略時，請注意下列各項：

- DynamoDB 不支援跨 Amazon S3 和 DynamoDB 的交易。因此，應用程式必須處理任何錯誤，其可能會包含清理遺棄的 Amazon S3 物件。
- Amazon S3 會限制物件識別碼的長度。因此，您必須以不會產生過長物件識別碼或違反其他 Amazon S3 限制的方法來整理您的資料。

有關如何使用 Amazon S3 的詳細資訊，請參閱[Amazon Simple Storage Service 使用者指南](#)。

## 在 DynamoDB 中處理時間序列資料的最佳實務

Amazon DynamoDB 中的一般設計原則會建議您盡量減少使用的資料表數量。對於大多數應用程式，您只需要一張資料表。不過，對於時間序列資料，您通常可以在每個期間針對每個應用程式使用一個資料表，這是最好的處理方式。

### 時間序列資料的設計模式

假設一個典型的時間序列情況，這時您想追蹤大量事件。您的寫入存取模式是所有正在記錄的事件都記有今天的日期。您的讀取存取模式可能是讀取今天最頻繁的活動，昨天的活動則較少讀存，然後更舊的活動則極少讀存。處理這種情況的一個方法，就是將目前的日期和時間內建到主索引鍵中。

以下設計模式通常能有效處理這種情況：

- 每個期間建立一個資料表，並且佈建所需的讀取和寫入容量，以及必要的索引。
- 在每個期間結束之前，先為下一個期間預先建立資料表。在目前時段結束時，將事件流量引導至新資料表。您可以為這些資料表指派名稱，來為其所記錄的期間命名。
- 一旦資料表不再被寫入，會將其佈建的寫入容量減少到較低的值（例如，1 個 WCU），並佈建適當的讀取容量。隨著舊資料表的存在時間拉長，逐漸地縮減其佈建的讀取容量。您可以針對內容極少用到或不再需要的資料表，選擇封存或刪除這些資料表。

這個概念是針對目前將會產生最高流量的期間，配置所需的資源，然後針對目前未經常使用的舊資料表，縮減佈建的容量，以節省成本。根據您的業務需求，您可能需要考慮寫入分片的方法，來將流量平均地分配到邏輯分割區索引鍵。如需詳細資訊，請參閱[使用寫入碎片以平均分佈工作負載](#)。



## 時間序列資料表範例

下列是時間序列資料的範例，其中針對目前的資料表佈建了較高的讀取/寫入容量，並縮減舊資料表的佈建容量，因為這些資料表不常存取。

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	...
...	...	...	...	...

## 管理 many-to-many 關係的最佳做法

鄰接表是一種設計模式，對於在 Amazon DynamoDB 中建 many-to-many 立關係的模型非常有用。整體來說，它們提供在 DynamoDB 中呈現圖形資料 (節點和邊緣) 的方式。

## 相鄰清單設計模式

當應用程式的不同實體之間具有 many-to-many 關係時，該關係可以被建模為一個鄰接表。在此模式中，所有頂層實體 (相當於圖形模型的節點) 會使用分割區索引鍵來呈現。任何具其他實體的關係 (圖形中的邊緣) 會透過將排序索引鍵值設為目標實體 ID (目標節點) 來以分割區中項目的形式呈現。

此模式的優點包含可將資料複製降到最低並簡化查詢模式，以尋找與目標實體相關 (有目標節點的邊緣) 的所有實體 (節點)。

此模式的實用實際範例為可開立多帳單發票的發票系統。一個帳單可屬於多個發票。此範例中的分割區索引鍵是 InvoiceID 或 BillID。BillID 分割區擁有專屬於帳單的所有屬性。InvoiceID 分割區有儲存特定發票屬性的項目，且擁有累計至發票的每個 BillID 項目。

結構描述看起來類似如下。

	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 in this invoice..
		Bill_ID: Bill-4224687 <i>(bill ID)</i>	Dated: 2018-01-09 <i>(date created)</i>	Attributes of this bill in this invoice..
	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 in this invoice..
	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...	

使用上述結構描述，您會發現可使用資料表上的主索引鍵來查詢發票的所有帳單。若要查詢包含部分帳單的所有發票，針對資料表的排序索引鍵建立全域次要索引。

全域次要索引的投影看起來類似如下。

	Primary Key	Projected Attributes...	
	Partition Key		
Bill-4224663	Bill_ID: Bill-4224663 <i>(table primary key)</i>	Attributes of this bill...	
	Inv_ID: Invoice-92551 <i>(table primary key)</i>	Attributes of this bill <i>in this invoice..</i>	
Bill-4224687	Bill_ID: Bill-4224687 <i>(table primary key)</i>	Attributes of this bill...	
	Inv_ID: Invoice-92551 <i>(table primary key)</i>	Attributes of this bill <i>in this invoice..</i>	
	Inv_ID: Invoice-92552 <i>(table primary key)</i>	Attributes of this bill <i>in this invoice..</i>	
Invoice-92551	Inv_ID: Invoice-92551 <i>(table primary key)</i>	Attributes of this invoice...	
Invoice-92552	Inv_ID: Invoice-92552 <i>(table primary key)</i>	Attributes of this invoice...	

## 具體化圖形模式

許多應用程式會根據對等間的排名、實體間的一般關係、鄰接實體狀態和圖形樣式流程的其他類型來建置。如需這些類型的應用程式，請考量以下結構描述設計模式。



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)	
	PLACE 4 BIRTH	Data	GSI PK		
		USA Texas Austin	Hash(Person.Data)		
	SKILL 6	Data	GSI PK		
		Java Developer Senior	Hash(Person.Data)		
	2	DATE 2	Data	GSI PK	
		1980-12-19	0		
	3	PLACE 3	Data	GSI PK	
		UK England London	0		
	4	PLACE 4	Data	GSI PK	
		USA Texas Austin	0		
	5	DATE 2 BIRTH	Data	GSI PK	
			1980-12-19	Hash(Person.Data)	
		PERSON 5	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PERSON 1 FRIEND	Data	GSI PK	
		John Doe	Hash(Person.Data)		
PLACE 3 BIRTH	Data	GSI PK			
	UK England London	Hash(Person.Data)			
SKILL 7	Data	GSI PK			
	Guitar Advanced	Hash(Person.Data)			
6	SKILL 6	Data	GSI PK		
	Java Developer	0			
7	SKILL 7	Data	GSI PK		
		Guitar	0		

Primary Key		Attributes		
GSI PK	GSI 1 SK (Data)			Graph Projections
GSI 1	O-N	NodeID	TypeTarget	...
		2	DATE 2	
		NodeID	TypeTarget	
		1	DATE 2 BIRTH	
		NodeID		
		5	SKILL 7	
		NodeID		
		7	SKILL 7	
		NodeID		
		5	TypeTarget	
		NodeID	TypeTarget	
		5	Person 5	
		NodeID	TypeTarget	
		1	Person 5 FRIEND	
		NodeID	TypeTarget	
		6	SKILL 6	
		NodeID		
		1	SKILL 6	
		NodeID		
		1	Person 1	
NodeID	TypeTarget			
5	Person 1 FRIEND			
NodeID	TypeTarget			
3	PLACE 3			
NodeID	TypeTarget			
1	PLACE 3 BIRTH			
NodeID	TypeTarget			
4	PLACE 4			
NodeID	TypeTarget			
5	PLACE 4 BIRTH			

	Primary Key		Attributes		
	GSI PK	GSI 2 SK (TypeTarget)	NodeID	Data	Graph Projections
GSI 2	O-N	DATE 2	NodeID	Data	...
			2	1980-12-19	
		DATE 2 BIRTH	NodeID		
			1		
			NodeID		
			5		
		PERSON 1	NodeID	Data	
			1	John Doe	
		PERSON 1 FRIEND	NodeID		
			5		
		PERSON 5	NodeID	Data	
			5	Jane Smith	
		PERSON 5 FRIEND	NodeID		
			1		
		PLACE 3	NodeID	Data	
			3	UK England London	
		PLACE 3 BIRTH	NodeID		
			5		
PLACE 4	NodeID	Data			
	4	USA texas Austin			
PLACE 4 BIRTH	NodeID				
	1				
SKILL 6	NodeID	Data			
	6	Java Developer			
	NodeID	Data			
	1	Java Developer Senior			
SKILL 7	NodeID	Data			
	7	Guitar			
	NodeID	Data			
	5	Guitar Advanced			

上述結構描述顯示的圖形資料結構會以內含項目的資料分割區組來定義，其內含項目會定義圖形的邊緣與節點。邊緣項目包含 Target 和 Type 屬性。這些屬性用作複合索引鍵名稱 "TypeTarget" 的一部分，以識別主要資料表或第二個全域次要索引中的分割區中的項目。

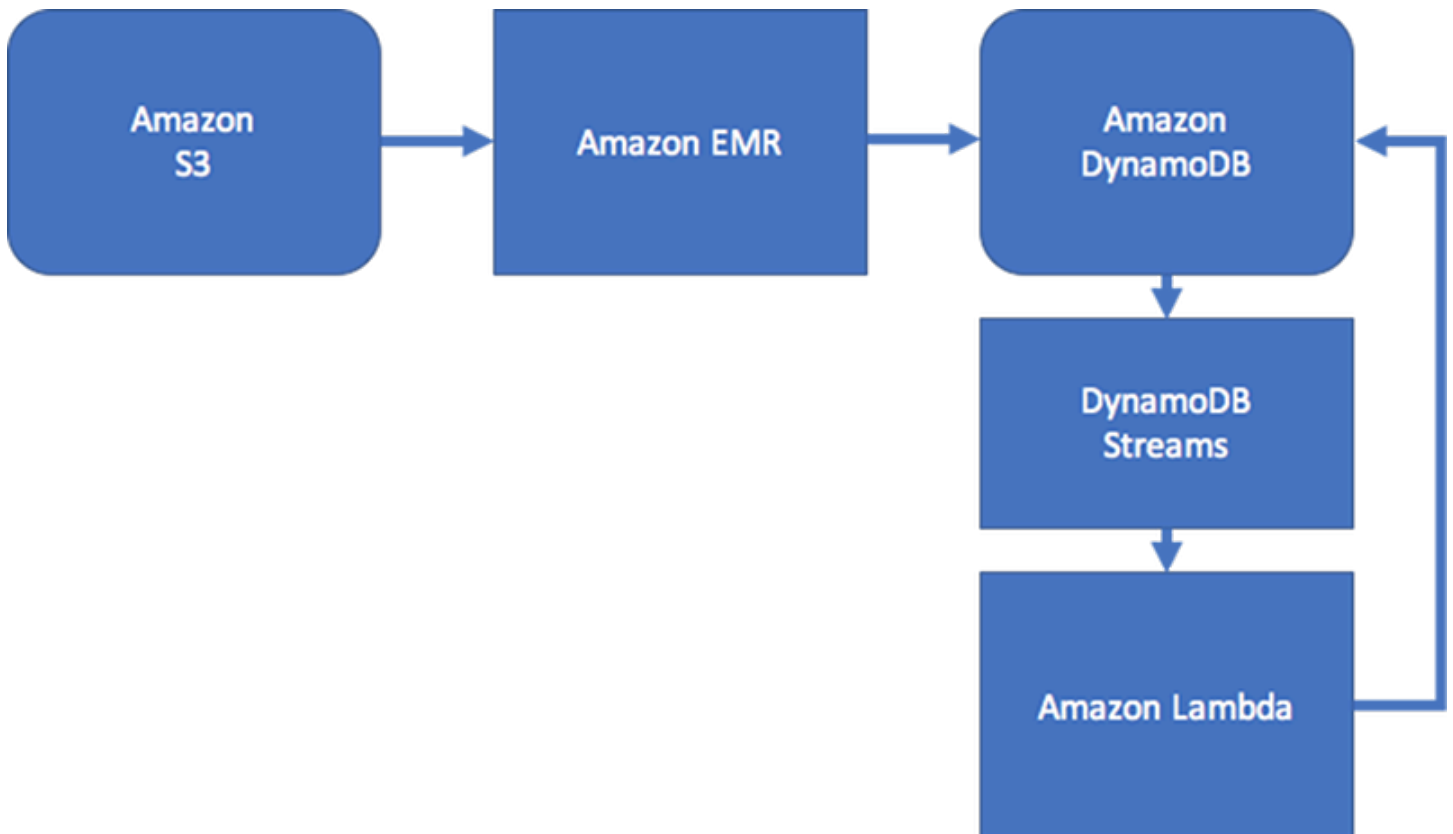
第一個全域次要索引會根據 Data 屬性建置。此屬性會如先前數個不同屬性類型 (也就是 Dates、Names、Places 和 Skills) 所述，使用全域次要索引多載。在這裡，一個全域次要索引可有效地對四個不同的屬性製作索引。

隨著您將項目插入資料表，您可以使用智慧分片策略來使用大型彙總 (生日、技能)，在避免經常性讀取/寫入問題時所需的許多邏輯分割區中，對全域次要索引散佈項目。

結合設計模式後，您能得到一個功能扎實的資料庫，可有效進行即時圖形流程。這些流程可以為建議的引擎、社交應用程式、節點排名、子樹系彙總和其他一般圖形使用者案例提供高效能鄰近實體狀態和邊緣彙總查詢。

如果您的使用案例對即時資料一致性並不敏感，您可以使用已排定的 Amazon EMR 程序，將適用於流程的相關圖形摘要彙總填入邊緣。如果您的應用程式不需要立即知道邊緣新增至圖形的時間，您可以使用排定的程序來彙總結果。

若要維持一定程度的一致性，設計可以包含 Amazon DynamoDB Streams 與 AWS Lambda 來處理邊緣更新。其可以使用 Amazon EMR 任務來驗證定期間隔的結果。下圖說明此方法。在社交應用程式會常常用到此方法，其中即時查詢的成本相當高，且立即知道個別使用者更新的需求很低。



IT 服務管理 (ITSM) 和安全性應用程式通常需要對由複雜邊際彙總構成的實體狀態變更回應做出即時回應。此類應用程式需要可以支援第二和第三層級關係之即時多節點彙總或複雜邊際尋訪的系統。若您的使用案例需要這些類型的即時圖形查詢工作流程，我們建議您使用 [Amazon Neptune](#) 來管理這些工作流程。

#### Note

如果您需要查詢高度連線的資料集，或執行需要以毫秒延遲周遊多個節點 (也稱為多躍點查詢) 的查詢，則應考慮使用 [Amazon Neptune](#)。Amazon Neptune 是專為高效能圖形資料庫引擎所打造，該服務將存放的數十億筆關係最佳化，且查詢圖形時只會有數毫秒的延遲。

## 實作混合資料庫系統的最佳實務

在某些情形中，從一或多個關聯式資料庫管理系統中 (RDBMS) 遷移到 Amazon DynamoDB 可能不是那麼有利。此時建議您建立混合系統。

## 如果不希望將所有項目遷移至 DynamoDB

例如，某些組織大量投資於程式碼，以便產生會計或作業所需的各種報告。產生報告所需的時間對他們來說並不重要。關聯式系統的彈性非常適合此類工作，且在 NoSQL 細節中重新建立所有報告可能相當昂貴。

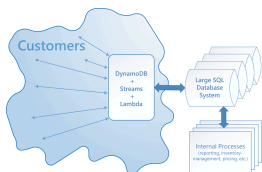
某些組織也維持各種在數十年間取得或繼承而來的舊式關聯式系統。從這些系統遷移資料的風險與價格可能太高，不值得做遷移。

然而，組織可能會發現他們的營運需依賴高流量的客戶服務網站，必須要有毫秒級的回應能力。除非投資高額 (通常高得難以接受) 的費用，否則關聯式系統無法擴充到符合此要求的地步。

在這類情況中，一個可能的解法是建立混合系統，其中 DynamoDB 建立在一或多個儲存在關聯式系統中的具體化資料檢視，並針對此檢視處理高流量要求。此類系統將先前處理客戶服務流量時所需的伺服器硬體、維護和 RDBMS 授權降到最低，藉此降低成本。

## 混合系統的實作方式

DynamoDB 可充分利用 DynamoDB Streams，並 AWS Lambda 與一或多個現有的關聯式資料庫系統無縫整合：



整合 DynamoDB Streams 且 AWS Lambda 可提供數個優點的系統：

- 以具體化檢視的持續快取形式運作。
- 其可以設定為當資料受到查詢或在 SQL 系統中遭到修改時逐漸填入資料。這意味著整個檢視不需要預先填入。反過來說，更可有效率地使用佈建輸送容量。
- 此管理成本較低且可用性高又可靠。

在實作此類整合時，基本上必須提供三種相互操作性。



1. 逐漸填入 DynamoDB 快取。查詢某個項目時，先在 DynamoDB 中尋找。若未找到，則在 SQL 中尋找，並將其載入 DynamoDB。

2. 透過 DynamoDB 快取寫入。客戶在 DynamoDB 中變更某數值時，系統會觸發 Lambda 函數以將新資料寫入至 SQL 系統。
3. 從 SQL 系統中更新 DynamoDB。當內部處理程序 (像是庫存管理與定價) 變更 SQL 系統中的值時，系統會觸發儲存的程序以將變更傳播至 DynamoDB 具體化的檢視。

這些操作是直覺化的，且並不是每個情況都需要所有操作。

混合解決方案在您想要主要依賴 DynamoDB 時很有用，但您或許也要維持小型關聯式系統，以供一次性查詢或供需要特別安全性或非緊急的操作使用。

## 在 DynamoDB 中製作關聯式資料模型的最佳實務

本節提供在 Amazon DynamoDB 中建構關聯式資料模型的最佳實務。首先，我們要介紹傳統的資料模型建構概念。接著會說明使用 DynamoDB 相較於傳統關聯式資料庫管理系統的優點，讓您了解它如何消除 JOIN 操作的需求並減少額外負荷。

然後我們將說明如何設計可有效擴展的 DynamoDB 資料表。最後我們會提供一個範例來說明，如何在 DynamoDB 中建模關聯式資料的模型。

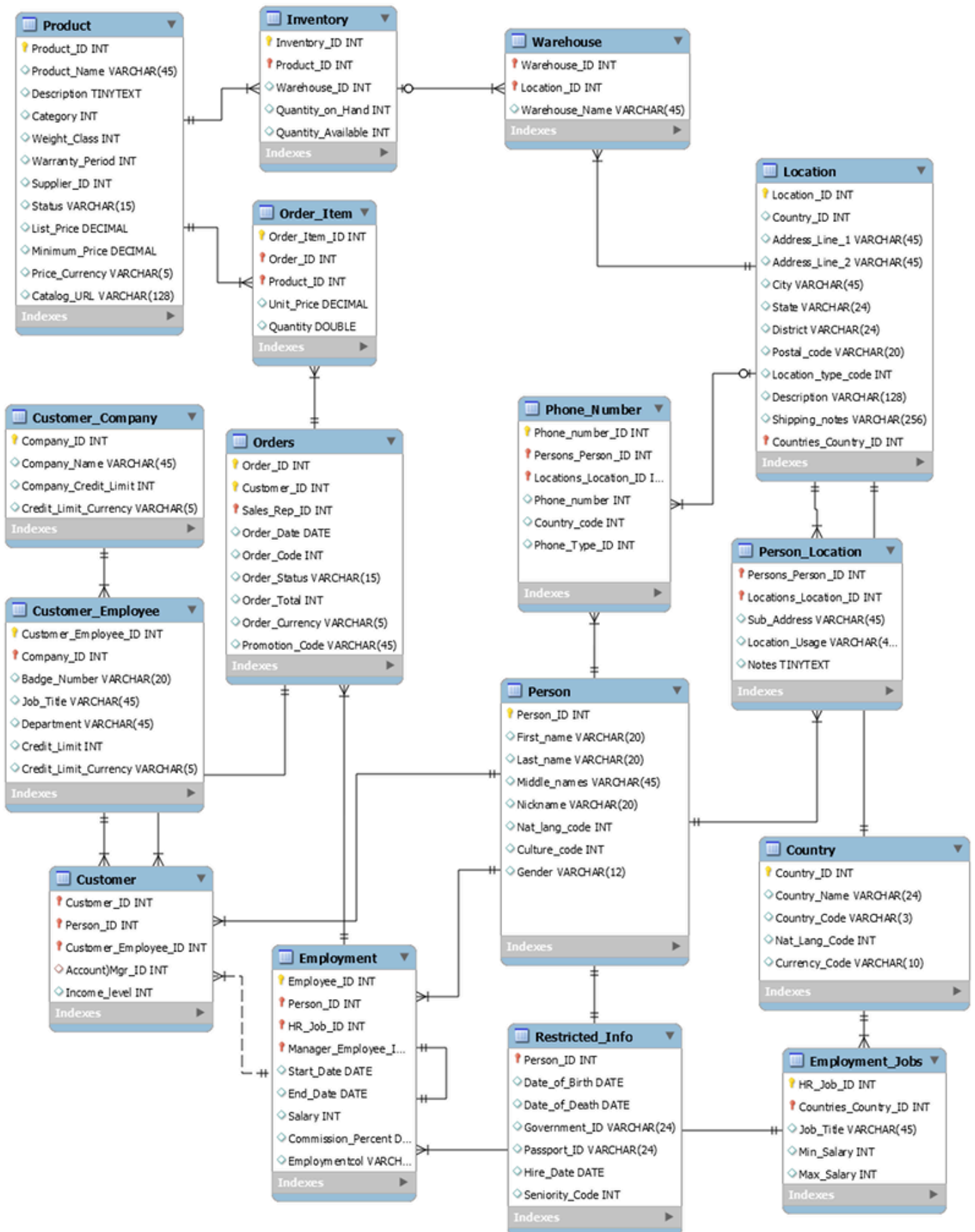
### 主題

- [傳統關聯式資料庫模型](#)
- [DynamoDB 如何免除 JOIN 操作的需求](#)
- [DynamoDB 交易如何消除寫入程序的負荷](#)
- [在 DynamoDB 中製作關聯式資料模型的第一步](#)
- [在 DynamoDB 中打造關聯式資料模型的範例](#)

## 傳統關聯式資料庫模型

傳統關聯式資料庫管理系統 (RDBMS) 會將資料儲存在標準化的關聯式結構中。關聯式資料模型的目標是減少資料的重複 (透過標準化)，以支援參考完整性並減少資料異常。

下列結構描述是一般訂單輸入應用程式的關聯式資料模型範例。此應用程式支援人力資源結構描述，並用它來為虛擬製造商的營運和業務支援系統提供支援。





由於 DynamoDB 是非關聯式資料庫服務，相較於傳統關聯式資料庫管理系統，它提供了更多優勢。

## DynamoDB 如何免除 JOIN 操作的需求

RDBMS 使用結構式查詢語言 (SQL) 將資料傳回至應用程式。由於資料模型的標準化，此類查詢通常需要使用 JOIN 運算子來合併來自一或多個資料表的資料。

例如，若要在可以運送任何項目的所有倉庫中，依庫存中的數量來排序產生購買順序項目清單，您可以對上述結構描述發出以下 SQL 查詢。

```
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
```

此類 SQL 查詢提供存取資料的彈性 API，但他們需要的處理量很大。查詢中的每個聯結都會增加查詢的執行期複雜度，因為每個資料表的資料必須暫存，然後再組裝以傳回結果集。

可能影響執行查詢時間長短的其他因素包括資料表的大小，以及要聯結的欄是否有索引。上述查詢會在數個資料表中起始複雜查詢，接著排序結果集。

免除使用 JOINS 的需求，是 NoSQL 資料建模的核心。這就是為什麼我們建置 DynamoDB 來支援 Amazon.com，以及為什麼 DynamoDB 可以在任何規模提供一致的效能。有鑑於 SQL 查詢和 JOINS 的執行期複雜性，RDBMS 效能不會大規模固定不變。這會隨著客戶應用程式的增長而導致效能問題。

雖然資料標準化確實可減少儲存到磁碟的資料量，但通常影響效能的最受限資源是 CPU 時間和網路延遲。

DynamoDB 的建置目的是消除 JOINS (並鼓勵取消資料標準化) 和最佳化資料庫架構，以便透過對項目的單一請求來完全回應應用程式查詢，將這兩種限制降到最低。這些品質讓 DynamoDB 能夠在任何規模下提供個位數的毫秒速度效能。這是因為 DynamoDB 操作的執行期複雜性對於常見存取模式來說是固定的，不會隨著資料大小而改變。

## DynamoDB 交易如何消除寫入程序的負荷

可能降低 RDBMS 速度的另一個因素，是使用交易來寫入標準化結構描述。如範例所示，大部分線上交易處理 (OLTP) 應用程式使用的關聯式資料結構儲存在 RDBMS 中時，必須細分並分散在多個邏輯資料表。



因此，符合 ACID 交易架構是必須的，以避免在應用程式嘗試讀取正處於寫入過程之物件時可能發生的競爭條件和資料完整性問題。此類交易框架與關聯式結構描述結合使用時，會大幅增加寫入程序的負荷。

在 DynamoDB 中實作交易會禁止 RDBMS 上常見的擴展問題發生。DynamoDB 以單一 API 呼叫的形式發出交易，並限制該單一交易中可存取的項目數量，藉此達到這個目的。長時間執行的交易可能因為交易永不關閉，所以長時間或永久鎖定資料，進而導致操作問題。

為了防止 DynamoDB 中發生此類問題，會使用兩個不同的 API 操作來實作交易：TransactWriteItems 和 TransactGetItems。這些 API 操作沒有 RDBMS 中常見的開始和結束語義。此外，DynamoDB 在交易中有 100 個項目的存取限制，同樣是為了防止交易長時間執行。若要進一步了解 DynamoDB 交易，請參閱[使用交易](#)。

基於這些原因，當您的企業需要對高流量查詢提供低延遲回應時，在技術與經濟的考量上通常會利用 NoSQL 系統。Amazon DynamoDB 可避免這些問題，協助解決限制關聯式系統可擴展性的問題。

基於以下原因，RDBMS 的效能通常無法適當地擴展：

- 因此會使用昂貴的聯結來重新組合必要的查詢結果檢視。
- 資料庫將資料標準化並存放在多個資料表中，這些資料表需要多個查詢來寫入至磁碟中。
- 這通常會引發 ACID 合規交易系統的效能成本，

DynamoDB 順利擴展的原因為下：

- 結構描述靈活性可讓 DynamoDB 在單一項目中存放複雜階層資料。
- 複合索引鍵設計可讓您將相關的項目存放在相同資料表的鄰近位置。
- 交易以單一操作形式執行。可存取的項目數限制為 100 個，以避免操作長時間執行。

對資料存放的查詢變得簡單多了，通常是使用以下格式：

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

與先前範例中的 RDBMS 相比，DynamoDB 傳回要求資料的程序簡單許多。

## 在 DynamoDB 中製作關聯式資料模型的第一步

### Important

NoSQL 設計思維與 RDBMS 設計不同。針對 RDBMS，您可以建立標準化的資料模型，而不需考量存取模式。您可以在稍後有新問題與查詢要求時擴展此模型。相反地，在 Amazon DynamoDB 中，您不應開始設計結構描述，除非您知道其需要回答的問題。您絕對必須事先了解企業問題和應用程式使用案例。

若要開始設計能有效擴展的 DynamoDB 資料表，您必須先執行數個步驟，以識別操作所需的存取模式與其需要支援的企業支援系統 (OSS/BSS)：

- 針對新應用程式，檢視活動與目標相關的使用者案例。記錄您識別的各種使用案例，並分析案例需要的存取模式。
- 針對現有應用程式，分析查詢記錄以找出人員目前使用系統的方式與索引鍵存取模式為何。

完成此程序後，您將收到一個清單，其看起來可能與以下類似。

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

在實際應用中，您的清單可能會更長。但此集合代表您可能會在生產環境中找到的查詢模式複雜度範圍。

DynamoDB 結構描述設計的常見方法是辨識應用程式層實體，並使用取消標準化與複合索引鍵彙總來降低查詢複雜度。

在 DynamoDB 中，此表示使用複合排序索引鍵、多載全域次要索引、分割區資料表/索引和其他設計模式。您可以使用這些元素來建構資料，如此應用程式就可以使用對資料表或索引的單一查詢來擷取特定

存取模式所需的項目。在 [關聯式模型](#) 中顯示您可以用來製作標準化結構描述模型的主要模式是相鄰清單模式。此設計中使用的其他模式可能包含全域次要索引寫入分片、全域次要索引多載、複合索引鍵和具體化的彙總。

### Important

一般來說，您在 DynamoDB 應用程式中維護的資料表應越少越好。例外狀況包含大量時間序列資料涉及其中的案例，或存取模式極為不同的資料集。含反轉索引的單一資料表通常可以啟用簡單查詢，來建立並擷取您應用程式所需的複雜階層資料結構。

若要使用適用於 DynamoDB 的 NoSQL Workbench 來協助視覺化您的分割區索引鍵設計，請參閱 [使用 NoSQL Workbench 建立資料模型](#)。

## 在 DynamoDB 中打造關聯式資料模型的範例

此範例說明如何在 Amazon DynamoDB 中打造關聯式資料模型。DynamoDB 資料表設計會與 [關聯式模型](#) 中顯示的關聯式排序項目結構描述相對應。其遵循 [相鄰清單設計模式](#)，此為在 DynamoDB 中呈現關聯式資料結構的常見方式。

該設計模式需要您定義一組實體類型，其通常會與關聯式結構描述中的各種資料表相關聯。系統接著會使用複合 (分割區索引鍵與排序索引鍵) 主要索引鍵來將實體項目新增至資料表。這些實體項目的分割區索引鍵為唯一辨識項目的屬性，且在所有項目上通常稱為 PK。排序索引鍵屬性包含您可以用於已反轉索引或全域次要索引的屬性值。此通常稱為 SK。

您定義的下列實體會支援關聯式排序項目結構描述。

1. HR-Employee : PK : EmployeeID、SK : 員工名稱
2. HR-Region : PK : RegionID、SK : 區域名稱
3. HR-國家-PK: CountryId, SK: 國家名稱
4. HR-Location : PK : LocationID、SK : 國家/地區名稱
5. HR-Job : PK : JobID、SK : 工作標題
6. HR-部門-PK : DepartmentID 件 , SK : DepartmentName
7. OE-客戶-PK : CustomerID , SK : 識別碼 AccountRep
8. OE-Order : PK OrderID、SK : CustomerID
9. OE-Product : PK : ProductID、SK : 產品名稱
10. OE-Warehouse : PK : WarehouseID、SK : 區域名稱

將這些實體項目新增至資料表後，您可以將邊緣項目新增至實體項目分割區來定義其中的關係。下列資料表示範此步驟。

在此範例中，資料表上的 Employee、Order 和 Product Entity 分割區擁有其他邊緣項目，其中包含對資料表上其他實體項目的指標。接著，定義少數全域次要索引 (GSI) 來支援先前定義的所有存取模式。實體項目不會全都將相同類型的值用在主要索引鍵或排序索引鍵屬性。只需要擁有要插入資料表上的主要索引鍵與排序索引鍵屬性。

事實是，這些實體中有部分會使用適當的名稱，而其他會使用其他實體 ID 做為排序索引鍵值，來允許相同的全域次要索引支援多種類型的查詢。此技巧稱作「GSI 多載」。其會有效地將 20 個全域次要索引的預設限制降到最低，以用於包含多個項目類型的資料表。下表以 GSI 1 顯示這種情況。

GSI 2 設計為支援常見的應用程式存取模式，以在擁有特定狀態的資料表上取得所有項目。針對在可用狀態間含不平均項目發佈的大型資料表，此存取模式可能會造成經常性索引鍵值，除非項目發佈的範圍是在可平行查詢的多於一個邏輯分割區中。此設計模式稱為 write sharding。

若要針對 GSI 2 達成此模式，應用程式會將 GSI 2 主要索引鍵屬性新增至每個「訂單」項目。其會填入 0-N 範圍中的某個亂數，其中 N 通常可以使用以下公式來計算，除非有特定不這麼做的原因。

```
ItemsPerRCU = 4KB / AvgItemSize

PartitionMaxReadRate = 3K * ItemsPerRCU

N = MaxRequiredIO / PartitionMaxReadRate
```

例如，假設您預期以下情況：

- 系統中將會有高達 200 萬的訂單，5 年內將成長至 300 萬。
- 在特定時間內這些訂單有高達 20% 會是 OPEN (開放) 狀態。
- 平均訂單記錄大約為 100 個位元組，在訂單分割區中有三個 OrderItem 記錄，每筆記錄大約為 50 個位元組，為您提供 250 個位元組的平均訂單實體大小。

針對該資料表，N 因素計算看起來會與以下類似。

```
ItemsPerRCU = 4KB / 250B = 16

PartitionMaxReadRate = 3K * 16 = 48K

N = (0.2 * 3M) / 48K = 13
```

在此案例中，您需要將所有訂單發佈在 GSI 2 上至少 13 個邏輯分割區中，以確保含 OPEN 狀態的所有 Order 項目的讀取不會對實體儲存層造成經常性分割區。此為填充此數字以允許在資料集異常狀況的最佳實務。因此使用  $N = 15$  的模型可能適用於此。如先前所述，您透過將 0-N 值新增至每個 Order 與 OrderItem 記錄的 GSI 2 PK 屬性來執行。

此明細假設需要搜尋所有 OPEN 發票的存取模式並不常發生，因此您可以使用高載容量來滿足要求。您可以使用 State 和 Date Range 排序索引鍵 (Sort Key) 條件來查詢下列全域次要索引，以依需要產生子集或為給定狀態的所有 Orders。

在此範例中，項目會隨機在 15 邏輯分割區中發佈。此結構會在存取模式需要大量將擷取的項目時運作。因此，15 個執行緒的其中之一不可能會傳回可能代表浪費容量的空結果集。即使未傳回任何項目或未寫入任何資料，查詢一律會使用 1 個讀取容量單位 (RCU) 或 1 個寫入容量單位 (WCU)。

如果存取模式需要對此全域次要索引進行高速查詢 (其會傳回稀疏結果集)，最好使用雜湊演算法來發佈項目 (不使用隨機模式)。在此案例中，執行時間執行查詢時，您可能會選取已知的屬性，並在插入項目時，將該屬性雜湊至 0-14 索引鍵空間。您可以有效率地透過全域次要索引來讀取它們。

最後，您可以再次瀏覽先前定義的存取模式。以下是您可以與應用程式 DynamoDB 版本一起使用以進行容納的存取模式清單與查詢條件。

	存取模式	查詢條件
1	按員工 ID 查詢員工詳細資訊	資料表上的主索引鍵 ，ID=「HR-EMPLOYEE」
2	按員工姓名查詢員工詳細資訊	使用 GSI-1，PK=「員工名稱」
3	僅取得員工的目前工作詳細資訊	資料表上的主索引鍵，PK=HR-EMPLOYEE-1，SK 開頭為「JH」
4	取得某個日期範圍內的客戶訂單	使用 GSI-1，PK= 客戶 1，SK=「狀態日期」，對於每個 StatusCode
5	針對所有客戶顯示某個日期範圍內處於 OPEN (未決) 狀態的所有訂單	使用 GSI-2，PK=[0..N] 範圍的平行查詢，SK 介於 OPEN-Date1 和 OPEN-Date2 之間

	存取模式	查詢條件
6	最近僱用的所有員工	使用 GSI-1, PK=「HR-CONFIDENTIAL」, SK > date1
7	尋找特定倉庫中的所有員工	使用 GSI-1, PK=WAREHOUSE1
8	取得產品的所有 Orderitems (訂單項目), 包括倉庫位置庫存	使用 GSI-1, PK=PRODUCT1
9	按客戶代表取得客戶	使用 GSI-1, PK=ACCOUNT-REP
10	按客戶代表和日期取得訂單	對於每個使用 GSI-1, PK= 帳戶代表, SK=「狀態日期」 StatusCode
11	取得特定職稱的所有員工	使用 GSI-1, PK=JOBTITLE
12	按產品和倉庫取得庫存	資料表上的主索引鍵 , PK=OE-PRODUCT1, SK =PRODUCT1
13	取得產品總庫存	資料表上的主索引鍵 , PK=OE-PRODUCT1, SK =PRODUCT1
14	按訂單總計和銷售週期排名取得客戶代表	使用 GSI-1, PK= 年-第一 季, = 假 scanIndexForward

## 查詢和掃描資料的最佳實務

本節會介紹在 Amazon DynamoDB 中使用 Query 和 Scan 操作的一些最佳實務。

### 掃描的效能考量

一般而言, Scan 操作效率低於 DynamoDB 中的其他操作。Scan 操作會一律掃描整個資料表或次要索引。然後它會篩選數值來提供您想要的結果, 基本上增加了從結果集刪除資料的額外步驟。

如果可行，您應該避免在大型資料表上或能移除許多結果的篩選條件上使用 Scan 操作。此外，隨著資料表或索引的增長，Scan 操作會變慢。所以此 Scan 操作會檢查每個項目的請求數值，並且可能會在單一操作中耗用大型資料表或索引的佈建輸送量。為了能擁有更快的回應時間，請設計資料表和索引，讓您的應用程式可以使用 Query 而非 Scan。(對於資料表，您也可以考慮使用 GetItem 和 BatchGetItem API。)

您也可以設計應用程式來使用 Scan 操作，最大限度地減少對請求率的影響。這可能包括比起 Scan 操作，使用全域次要索引更有效率的建模。有關此流程的更多資訊，請參閱以下影片。

## [建模低速存取模式](#)

### 避免讀取活動突發尖峰

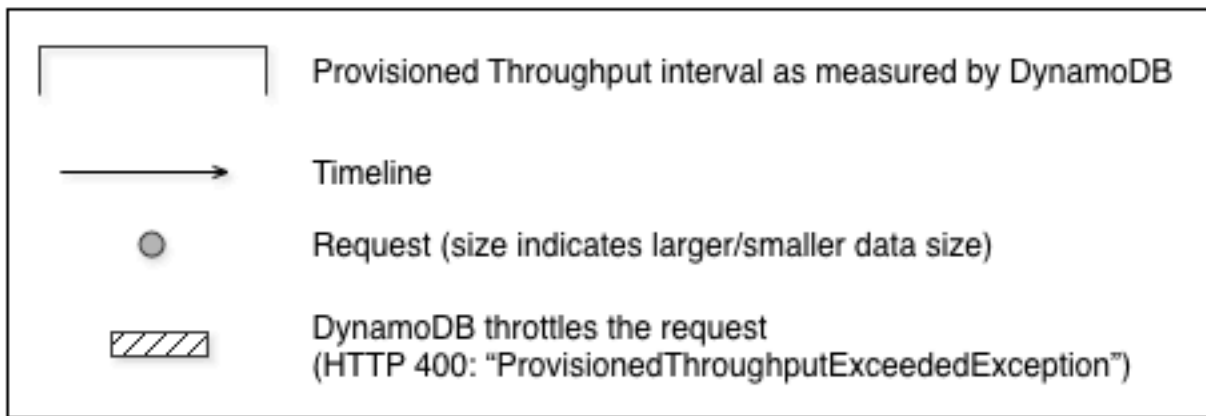
建立資料表時，您可以設定其讀取和寫入容量單位需求。針對讀取，容量單位會以每秒強烈一致的 4 KB 資料讀取請求數來表示。對於最終一致讀取，讀取容量單位為每秒兩個 4 KB 讀取請求。Scan 操作預設會執行最終一致讀取，而且最多可傳回 1 MB (一頁) 的資料。因此，單一 Scan 請求可能會消耗  $(1 \text{ MB 頁面大小} / 4 \text{ KB 項目大小}) / 2$  (最終一致讀取) = 128 個讀取操作。相對地，如果您請求強烈一致讀取，則 Scan 操作會耗用兩倍的佈建輸送量：256 個讀取操作。

與資料表設定的讀取容量相比，這代表在使用時突發尖峰。此掃描使用容量單位可防止同一資料表的其他可能更重要的請求使用可用容量單位。因此對於那些請求，您可能會得到一個 ProvisionedThroughputExceeded 例外狀況。

問題不僅僅是 Scan 使用的容量單位突然增加。掃描也可能會耗用相同分割區的所有容量單位，因為掃描會請求讀取該分割區上彼此相鄰的項目。這表示請求會落在相同的分割區上，造成其所有容量單位被消耗，並調節對該分割區的其他請求。如果讀取資料的請求分散在多個分割區，則操作不會調節特定的分割區。

下圖說明 Query 和 Scan 操作的容量單位使用量激增的影響，以及其對同一資料表的其他請求的影響。





### 1. Good: Even distribution of requests and size



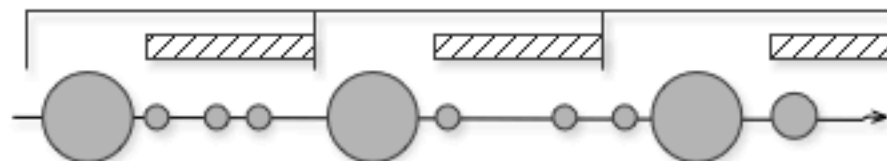
### 2. Not as Good: Frequent requests in bursts



### 3. Bad: A few random large requests



### 4. Bad: Large scan operations



如此處所示，使用量激增會以數種方式影響資料表的佈建輸送量：

#### 1. 良好：請求和大小分配均勻



2. 不太好：頻繁的突發請求
3. 不良：幾個隨機的大型請求
4. 不良：大型掃描操作

除了使用大型 Scan 操作，您可以使用下列技巧，將該操作對資料表佈建輸送量的影響降至最低。

- 縮減頁面大小

由於掃描操作會讀取整個頁面 (預設為 1 MB)，您可以設定較小的頁面大小來降低掃描操作的影響。Scan 操作提供了 Limit 參數，可讓您用來設定請求的頁面大小。每個擁有較小頁面大小的 Query 或 Scan 請求都會使用較少的讀取操作，並會在各請求之間建立「暫停」。例如，假設每個項目是 4 KB，而且您將頁面大小設定為 40 個項目。Query 請求將僅消耗 20 個最終一致讀取操作或 40 個強烈一致讀取操作。較大數量的較小 Query 或 Scan 操作會允許您的其他關鍵請求在不受調節的情況下成功進行。

- 隔離掃描操作

DynamoDB 旨在輕鬆實現可擴展性。因此，應用程式可以為不同的目的建立資料表，甚至可能會跨多個資料表複製內容。您想要在未採用「任務關鍵型」流量的資料表上執行掃描。某些應用程式處理此負載的方式為每小時在兩個資料表 (一個用於關鍵流量，另一個則用於簿記) 之間輪換流量。其他應用程式可以透過在兩個資料表 (「任務關鍵型」和「陰影」資料表) 上執行每次寫入來處理負載。

將您的應用程式設定為重試接收表示您已超過佈建輸送量的回應代碼的任何請求。您也可以使用 UpdateTable 操作為資料表增加佈建的輸送量。如果工作負載有暫時的激增，導致輸送量偶爾超出佈建層級，則使用指數退避重試請求。如需實作指數退避的詳細資訊，請參閱 [錯誤重試和指數退避](#)。

## 利用平行掃描

許多應用程式可以從使用平行 Scan 操作而非循序掃描中獲益。例如，處理大型歷史資料表的應用程式，執行平行掃描的速度會比循序掃描的速度要快得多。背景「掃描程式」程序中的多個工作者執行緒，可以低優先順序掃描資料表，而不會影響生產流量。在每個這些例子中，平行 Scan 的使用方式使其不會剝奪佈建輸送量資源的其他應用程式。

雖然平行掃描可能很有幫助，但它們可能會對佈建輸送量造成大量需求。使用平行掃描時，您的應用程式有多個工作者都在同時執行 Scan 操作。這會快速消耗資料表的佈建讀取容量。在這種情況下，其他需要存取資料表的應用程式可能會受到調節。

如果符合下列條件，則平行掃描會是正確的選擇：

- 資料表大小為 20 GB 或更大。
- 該資料表的佈建讀取輸送量尚未完全使用。
- 循序 Scan 操作太慢。

## 選擇 TotalSegments

TotalSegments 的最佳設定取決於您的特定資料、資料表的佈建輸送量設定以及您的效能需求。您可能需要進行實驗，才能正確設定。我們建議您從簡單的比率開始，例如每 2 GB 資料一個區段。例如，對於 30 GB 的資料表，您可以設定 TotalSegments 為 15 個區段 (30 GB/2 GB)。您的應用程式將使用 15 個工作者，每個工作者掃描不同的區段。

您也可以根據用戶端資源為 TotalSegments 選擇一個數值。您可以將 TotalSegments 設定為 1 到 1000000 之間的任何數字，而 DynamoDB 則可讓您掃描該數目的區段。例如，如果用戶端限制可以同時運行的執行緒數量，您就可以逐漸增加 TotalSegments，直到應用程式得到最佳 Scan 效能。

監控平行掃描以最佳化佈建輸送量的使用，同時確保其他應用程式不會耗盡資源。如果您沒有消耗所有佈建的輸送量，但仍然在 Scan 請求中經歷調節，則增加 TotalSegments 的數值。如果 Scan 請求所耗用的佈建輸送量超過您想使用的量，則減少 TotalSegments 的數值。

## DynamoDB 資料表設計的最佳實務

Amazon DynamoDB 中的一般設計原則會建議您盡量減少使用的資料表數量。大多數情況下，我們建議您考慮使用單一資料表。但是，如果單一或少量表格不可行，則可能可以使用以下指引。

- 每個帳戶限制資料表不得增加超過每個帳戶 10,000 個。如果您的應用程式需要更多資料表，請在多個帳戶間分散資料表。如需詳細資訊，請參閱 [Amazon DynamoDB 中的服務、帳戶和資料表配額](#)。
- 對於可能影響資料表管理的並行控制平面操作，請考慮控制平面限制。
- 與 AWS 解決方案架構師合作，驗證多租戶設計的設計模式。

## DynamoDB 全域資料表設計的最佳實務

全域資料表建立於 Amazon DynamoDB 全域佈局的基礎之上，提供您全受管、多區域以及多個作用中的資料庫，為大幅度擴展的全域應用程式提供快速、本機、讀取與寫的效能。透過全域資料表，您的資料會自動在您選擇的 AWS 區域中複製。由於全域資料表使用現有的 DynamoDB API，因此您的應用程式無須變更。使用全域資料表沒有前期成本或承諾，您僅需為使用的資源付費。

### 主題

- [DynamoDB 全域資料表設計的方案指引](#)
- [有關 DynamoDB 全域資料表設計的關鍵事實](#)
- [使用案例](#)
- [全域資料表的寫入模式](#)
- [使用全域資料表的請求路由](#)
- [使用全域資料表疏散區域](#)
- [全域資料表的輸送容量規劃](#)
- [全域資料表和常見問題的就緒性檢查清單](#)

## DynamoDB 全域資料表設計的方案指引

有效使用全域資料表需要仔細考慮因素，如偏好的寫入模式、路由模型和疏散程序。您必須在每個區域檢測您的應用程式，並準備好調整路由或執行疏散以維護全域狀況。獲得的好處是擁有全球分佈式的資料集，具有低延遲讀取和寫入，以及 99.999% 的服務等級協議。

## 有關 DynamoDB 全域資料表設計的關鍵事實

- 全域資料表有兩個版本：目前版本的[全域表 2019.11.21 \(目前版本\)](#) (有時稱為「V2」)，以及[全域表格版本 \(舊版\)](#) (有時稱為「V1」)。本指南僅著重於目前版本 V2。
- 如果不使用全域資料表，則 DynamoDB 就是一種區域服務。具有高可用性，並且在本質上對區域基礎架構故障具有彈性，包括整個可用區域 (AZ) 的故障。單一區域 DynamoDB 資料表具有 99.99% 的可用性<https://aws.amazon.com/dynamodb/sla>服務水準協議 (SLA)。
- 使用全域資料表，DynamoDB 讓資料表可以在兩個或多個區域之間複寫其資料。多區域 DynamoDB 資料表具有 99.999% 的可用性 SLA。通過適當的計畫，全域資料表有助於建立一個具有彈性並抵抗區域故障的架構。
- 全域資料表採用主動-主動式複寫模型。從 DynamoDB 的角度來看，每個區域中的資料表具有相同地位，可以接受讀取和寫入請求。收到寫入請求後，本機複本資料表會在背景作業將複本寫入其他參與的區域。
- 單獨複製項目。單一交易中更新的項目可能無法一起複製。
- 來源區域中的每個資料表分割區都會與其他分割區同步寫入複本。遠端區域內的寫入順序可能與來源區域內發生的寫入順序不相符。如需有關資料表分割區的詳細資訊，請參閱部落格文章[擴展 DynamoDB：分割區、快捷鍵和熱分割會如何影響效能](#)。
- 新寫入的項目通常會在一秒內傳播到所有複本列表。靠近區域的傳播速度往往更快。

- Amazon CloudWatch 為每個區域對提供一個ReplicationLatency指標。指標會根據查看項目到達時間，並將其到達時間與初始寫入時間進行比較並計算平均值。計時會儲存 CloudWatch 在來源區域內。檢視平均和最大計時有助於判斷平均和最差情況的複寫延遲。此延遲沒有 SLA。
- 如果相同項目在兩個不同的區域幾乎同時 (在此 ReplicationLatency 視窗內) 更新，並且第二次寫入發生在第一次寫入複製之前，則可能會發生寫入衝突。全域資料表會根據寫入的時間戳記，解決最後一個寫入獲勝機制之間的衝突。第一次寫入會「輸」給第二次寫入。這些衝突不會記錄在 CloudWatch 或中 AWS CloudTrail。
- 每個項目都有一個作為私有系統屬性保留的最後寫入時間戳記。最後一個寫入獲勝方法是通過使用條件寫入來實現，要求傳入項目的時間戳記大於現有項目的時間戳記。
- 全域資料表會將所有項目複寫到所有參與的區域。如果您想要有不同的複寫範圍，您可以建立不同的資料表，並為每個資料表提供不同的參與區域。
- 即使複本區域離線或 ReplicationLatency 增長，本機區域也會接受寫入。本機資料表會繼續嘗試將項目複製到遠端資料表，直到每個項目成功為止。
- 雖然不太可能，但是萬一區域完全離線，當稍後重新上線時，所有擱置的輸出和輸入複寫都會重新嘗試。不需要特殊動作即可使資料表恢復同步。最後一個寫入獲勝機制可確保資料最終一致。
- 您可以隨時將新區域新增至 DynamoDB 資料表。DynamoDB 將處理初始同步和進行中的複寫。如果區域遭到移除，即使是原始區域，也只會刪除該區域的資料表。
- DynamoDB 沒有全域端點。所有請求都會傳送至區域端點，然後存取該區域本機的全域資料表執行個體。
- 不應對 DynamoDB 跨區域呼叫。最佳實務是讓一個區域中的運算層直接存取該區域的本機 DynamoDB 端點。如果在區域內偵測到問題，無論這些問題是位於 DynamoDB 層或周圍堆疊中，則應將最終使用者流量路由至託管在不同區域的不同計算層。由於全域資料表複本，不同區域將具有相同資料的本地複本供其本機使用。某些情況下，某個區域中的運算層可能會將請求傳遞至另一個區域的運算層進行處理，但這不應該直接存取遠端 DynamoDB 端點。如需此特定使用案例的詳細資訊，請參閱 [運算層請求路由](#)。

## 使用案例

全域表提供以下常見優點：

- 較低的讀取延遲。您可以將資料複本置於靠近終端使用者的位置，以減少讀取期間的網路延遲。快取保持與 ReplicationLatency 值一樣新。
- 較低的寫入延遲。您可以寫入附近的區域，以減少網路延遲和寫入所花費的時間。必須小心路由寫入流量，以確保沒有衝突。路由技術在 [使用全域資料表的請求路由](#) 中有更詳細的討論。

- 提高彈性和災難復原。使用復原點目標 (RPO) 和復原時間點目標 (RTO)，當區域效能降低或完全中斷時，您可以疏散該區域 (移除部分或全部傳送至該區域的請求)。使用全域資料表也會將 [DynamoDB SLA](#) 從 99.99% 提高到 99.999%。
- 無縫區域遷移。您可以新增一個新區域，然後刪除舊區域，將部署從一個區域遷移到另一個區域，所有動作都不會導致資料層停機。例如，您可以針對訂單管理系統使用 DynamoDB 全域資料表，實現可靠的大規模低延遲處理，同時還能保持 AZ 和區域故障的彈性。

## 全域資料表的寫入模式

全域表在資料表永遠處於主動-主動資料表層級。不過，您可能想要透過控制路由，將它們用作主動-被動的方式。例如，您可能決定將寫入請求路由到單一區域，以避免潛在的寫入衝突。

受管寫入模式有三種主要分類：

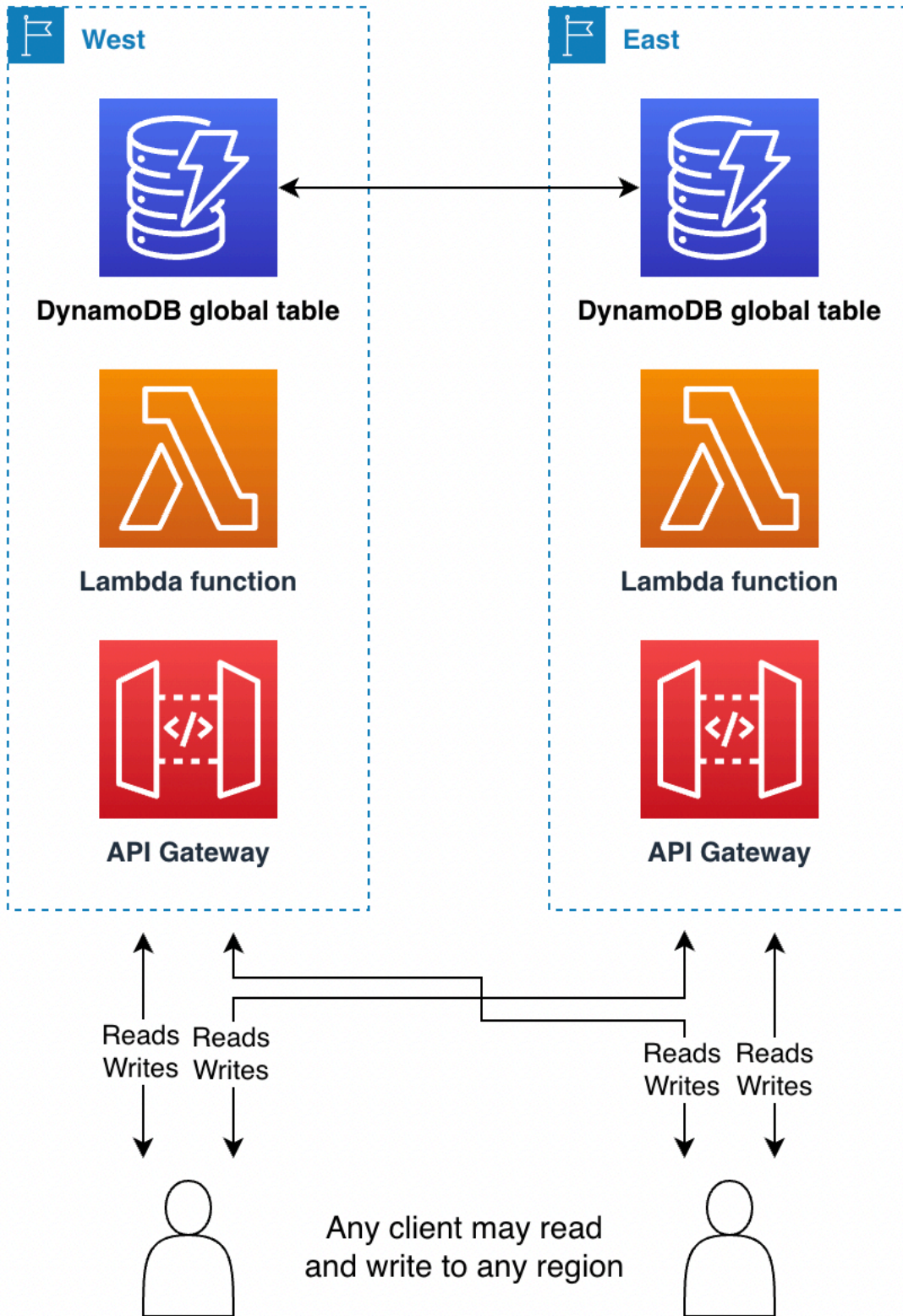
- 寫入任何區域模式 (無優先層級)
- 寫入單一區域模式 (單一優先層級)
- 寫入您的區域模式 (混合優先層級)

您應該考慮哪種寫入模式適合您的使用案例。此選項會影響您的路由請求、疏散區域以及處理災難復原的方式。整體最佳實務可能會因應用程式的寫入模式而有所不同。

### 寫入任何區域模式 (無優先層級)

寫入任何區域模式是完全主動-主動的方式，並且不會任何寫入位置施加限制。任何區域隨時可以接受寫入。這是最簡單的模式。只能適用於某些類型的應用程式。非常適用於所有寫入器都是等冪性時，因為可以安全地複寫，以便跨區域並行進行或重複寫入操作時不會發生衝突。例如，使用者更新其聯絡資料。這種模式也適用於等冪性的特殊情況 (一個僅能新增的資料集)，其中所有寫入都是確定性主索引鍵下的唯一插入。最後，此模式非常適合可以接受寫入衝突風險的情況之下。





寫入任何區域模式是最直覺式的實作架構。因為任何區域都可以隨時成為寫入目標，路由會更加容易。容錯移轉比較容易，因為可以重播任何最近的寫入到任何次要區域的次數。盡可能之下，您應該以此為寫入模式進行設計。

例如，影片串流服務通常會使用全域資料表來追蹤書籤、評論、觀看狀態旗標等。這些部署可以使用寫入任何區域模式，只需確保每次寫入都是等冪性，並且項目的下一個正確值不依賴於其目前的值。對於直接指派使用者新狀態的使用者更新，例如，設定最新的時間代碼、指派新的審核或設定新的監看狀態，就會發生這種情況。如果使用者的寫入請求被路由到不同的區域，最後一個寫入操作將持續，並且全域狀態將根據最後的指派進行結算。在最後延遲 `ReplicationLatency` 值之後，此模式下的讀取操作最終會變成一致。

在另一個範例中，一家金融服務公司使用全域資料表作為系統的一部分來維護每個客戶使用借記卡購買的動作記錄，以計算該客戶的現金回饋獎勵。新的交易會從世界各地進行串流並前往多個區域。對於他們當前不利用全局表的設計，他們使用每個客戶的單個 `RunningBalance` 項目。客戶動作會使用 `ADD` 運算式更新餘額，該運算式不是等冪性的，因為新的正確值取決於目前值。這意味著如果在不同區域中大約同一時間，有兩個寫入操作達到相同餘額，則餘額會失去同步。

這家公司可以通過對 `DynamoDB` 全域資料表仔細的重新設計，來實現寫入任何區域模式。新的設計可以遵循「事件串流」模式-本質上是一個帶有僅允許新增的工作流程分類帳。每個客戶動作都會將新項目新增到為該客戶維護的項目收集器。項目集合是共用主索引鍵，具有不同排序索引鍵的一組項目。每個附加客戶動作的寫入動作都是等冪性插入，使用客戶 ID 做為分割區索引鍵，而交易 ID 做為排序索引鍵。這種設計使餘額的計算涉入更多，因為需要 `Query` 拉動項目，搭配用戶端運算。但優點是，能夠使所有寫入都是等冪性，提供顯著的路由和容錯移轉簡化。如需更多資訊，請參閱[使用全域資料表的請求路由](#)。

舉第三個範例，假設有一位客戶在進行線上位置刊登廣告。他們已經決定，為了能夠實現寫入任何區域模式的設計簡化，可以接受低風險的資料丟失。當他們投放廣告時，他們只需幾毫秒就能擷取足夠的中繼資料來決定要顯示的廣告，然後記錄廣告曝光次數，這樣相同的廣告就不會向該使用者重複播放。透過全域資料表，他們可以取得全球終端使用者低延遲讀取和低延遲寫入。他們可以在單個項目中記錄使用者的所有廣告曝光次數，並將其表示為不斷增長的清單。他們可以使用一個項目而不是附加到項目集合中，因為，這樣他們就可以在每次寫入操作中刪除較舊的廣告曝光，而無需支付刪除費用。這項寫入操作不是等冪性的，因此，如果同一位使用者在大約相同時間看到在多個區域投放的廣告，就可能會有一個廣告印象寫入覆寫另一個區域。對於線上位置刊登廣告，使用者偶爾會看到重複播放廣告的風險，值得擁有更簡單、更有效率的設計。

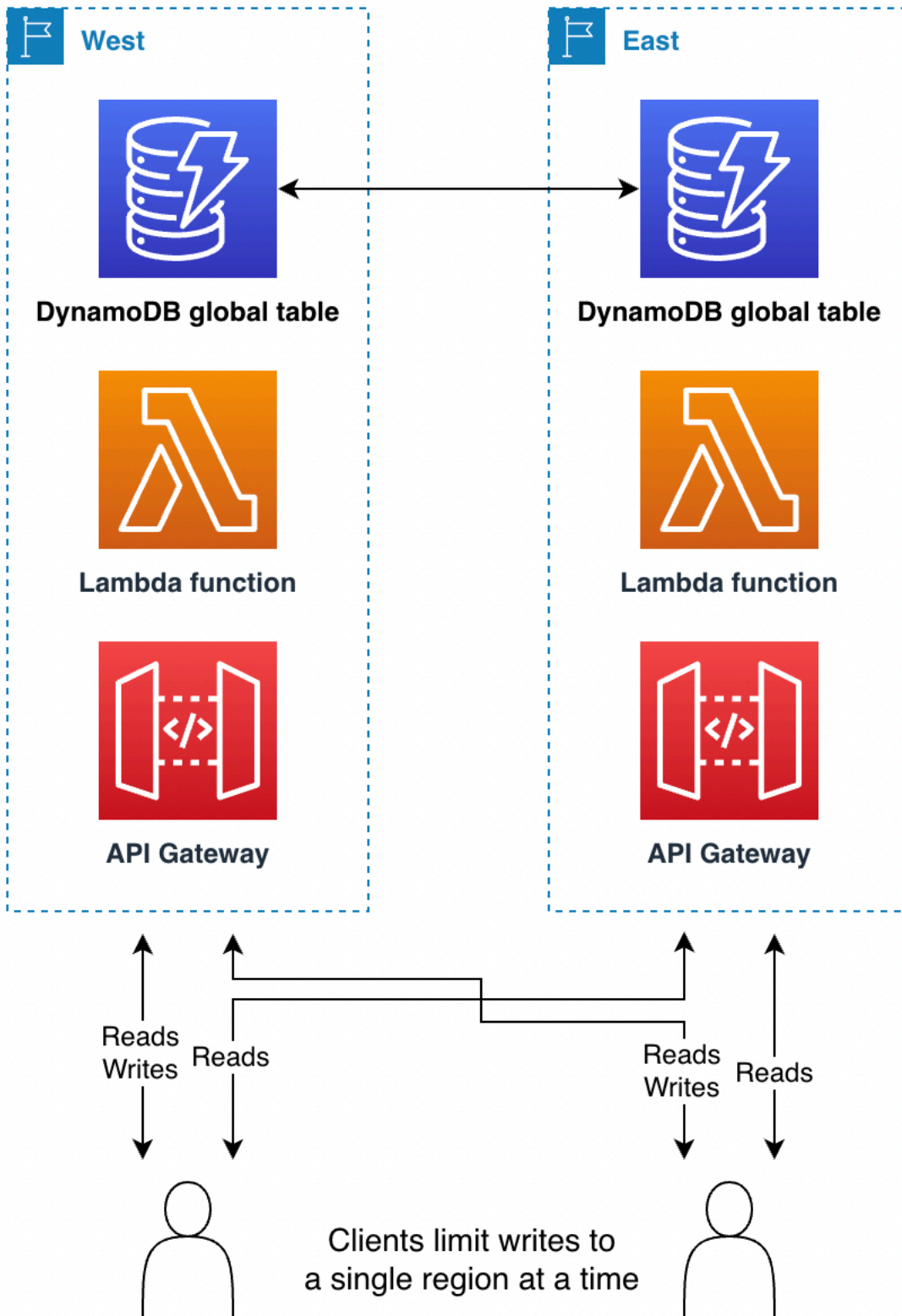
## 單一優先層級 (「寫入單一區域」)

寫入單一區域模式是主動-被動，並將所有資料表寫入路由到單一主動區域。請注意，`DynamoDB` 沒有單一使用中區域的概念；`DynamoDB` 外部路由的應用程式會管理此功能。寫入單一區域模式可確保一

次只寫入單一區域，以避免寫入衝突。當您想要使用條件表達式或交易時，這種寫入模式很有用，因為除非您知道自己在使用最新資料，否則將無法運作。因此，使用條件表達式和交易需要將所有寫入請求發送到具有最新資料的區域。

最終一致讀取可以移至任何複本區域，以達到較低的延遲。高度一致性讀取必須移至單一主要區域。





有時需要變更使用中的區域回應區域故障，以便於處理資料。[使用全域資料表疏散區域](#) 便是一種這類使用案例的範例。有些客戶會定期變更目前使用中的區域，例如「日後追蹤」部署。這會將使用中區域放置在最活躍的地理位置附近，進而提供最低延遲的讀取和寫入。還具有每天調用區域更改代碼路徑的附加優點，確保在任何災難復原之前都經過充分的測試。

被動區域可能會在 DynamoDB 周圍保留一組縮小規模的基礎設施，只有當成為主動區域時才會建立起來。有關指示燈和暖待機設計的更深入討論，請參閱[上的災難恢復 \(DR\) 架構 AWS，第 III 部分：指示燈和暖待機](#)。

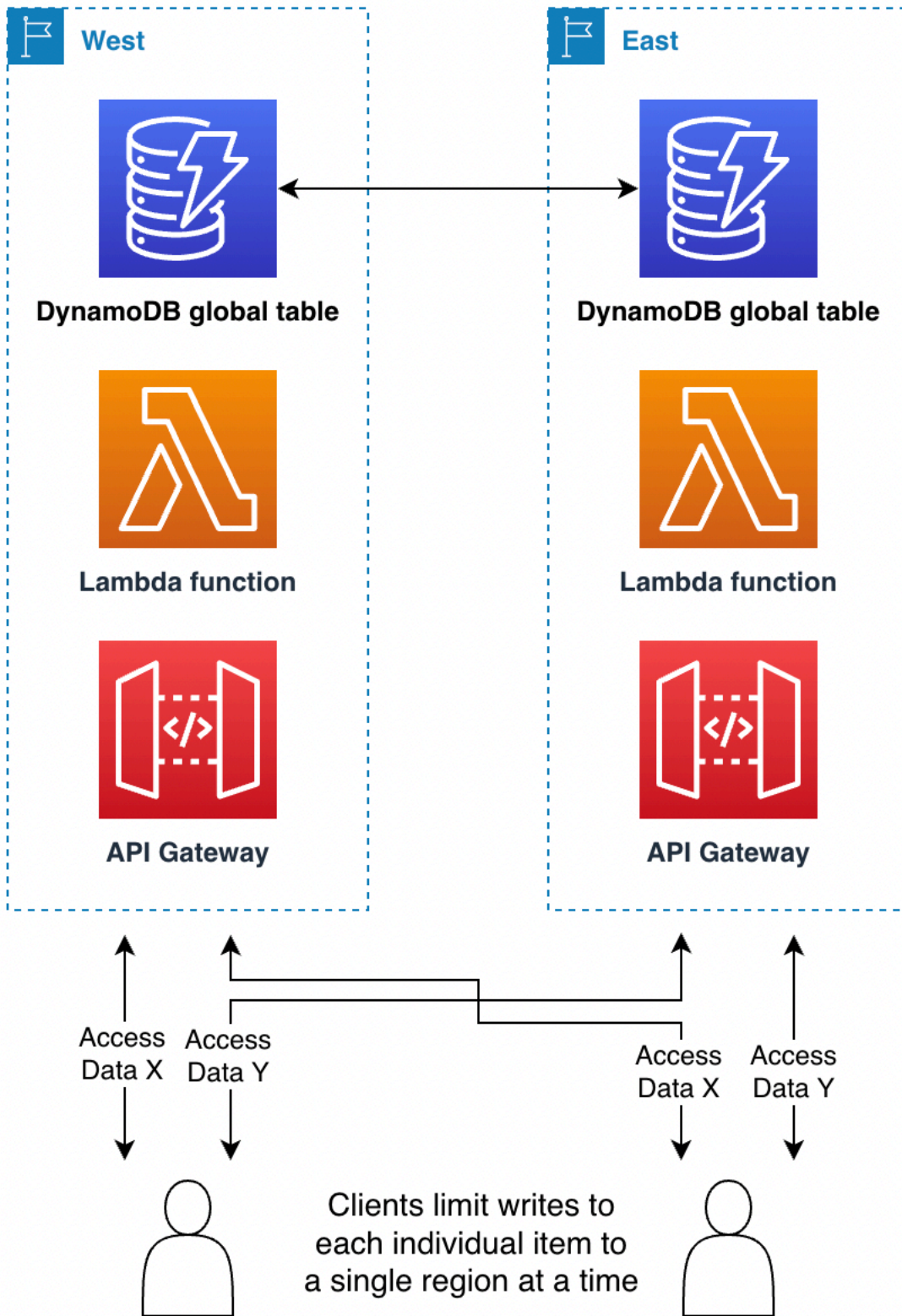
利用全域資料表實現低延遲全域分散讀取時，使用寫入單一區域模式會非常合適。例如，一家大型社群媒體公司擁有數百萬使用者和數十億貼文。每個使用者在建立帳戶時都會被分配到一個區域，該區域位於靠近他們的地理位置。所有資料都會進入該非全域資料表。該公司使用一個單獨的全域資料表來保存使用者到其主區域的映射，採用寫入單一區域模式。世界各地僅保留僅供讀取複本，以幫助直接找到每個使用者的資料，並將附加延遲降至最低。很少進行更新 (僅在將使用者的主要區域從一個區域移動到另一個區域時)，並且始終通過一個區域進行寫入，以避免發生寫入衝突的可能性。

作為另一個例子，假設有一位實作每日現金回饋計算的金融服務客戶。他們使用寫入任何區域模式來計算餘額，但使用寫入單一區域模式來追蹤實際的現金回饋付款。如果他們想為客戶提供每一天花費 10 美元即回饋 1 美分，他們需要 Query 將前一天的所有交易計算總支出，將現金回饋決策寫入新的資料表，刪除查詢的項目集並其標記為已消耗，並替換為單一項目儲存作為提醒金額，用於次日的計算。這項工作牽涉交易，因此寫入單一區域模式將能夠更好地工作。只要工作負載沒有重疊的機會，即使在同一個資料表上，應用程式也可能會混合寫入模式。

## 混合優先層級 (「寫入您的區域」)

寫入您的區域模式會將不同的資料子集指派給不同的區域，並且只允許透過其主區域的項目進行寫入操作。此模式為主動-被動模式，但會根據項目指派使用中的區域。每個區域都是對自己本身的非重疊資料集都是優先層級，而且必須保護寫入以確保適當的位置。

此模式類似於寫入單一區域模式，不同之處在於它可以實現更低的延遲寫入，因為與每個終端使用者相關的資料可以放置在距離該使用者更接近的網絡中。該模式也會在區域之間更均勻地分散周圍的基礎結構，並且容錯移轉案例期間需要建立基礎設施的工作較少，因為所有區域都會有一部分使用中的基礎設施。



可以用多種方式來決定項目的主區域：

- 固有：資料的某些方面清楚地表示其所在的區域，例如其分割區索引鍵。例如，客戶和有關該客戶的所有資料將在客戶資料中標記為歸屬特定區域。這項技術於[使用區域固定為 Amazon DynamoDB 全域資料表的項目設定主區域](#)中有說明
- 協商：每個資料集的主區域會以某種外部方式進行協商，例如，有一個單獨的全域服務來維護指派。指派可能是一個有限的持續時間，之後需要重新協商。
- 資料表導向：與單一複寫全域資料表不同，而是複寫區域和全域資料表一樣多。每個資料表的名稱都表示其主區域。標準操作中，所有資料都會寫入主區域，而其他區域則保留唯讀副本。在容錯移轉期間，另一個區域將暫時為該資料表承擔寫入職責。

例如，假設您在為一家遊戲公司工作。您需要為全球所有遊戲玩家提供低延遲的讀取和寫入。您可以將每個玩家的主區域放置到最接近玩家的區域。該區域需要所有的讀取和寫入，確保始終具有強大的 read-after-write 一致性。但是，如果該玩家外出旅行，或其主區域遭遇中斷，則將在替代區域提供玩家資料的完整副本。因此，能夠將玩家分配到不同主區域非常實用。

另一個例子，假設您在視訊會議公司工作。每個電話會議的中繼資料都會指派給特定區域。來電者可以使用離他們最近的區域以獲得最低延遲。如果發生區域中斷，使用全域資料表可讓您快速復原，因為系統可以將通話處理轉移到擁有複寫資料副本的不同區域。

## 使用全域資料表的請求路由

也許，全域資料表部署中最複雜的部分是管理請求路由。請求必須首先從終端使用者以某種方式選擇和路由到區域。請求遇到該區域中的某些服務堆疊，其中包括可能由 AWS Lambda 函數、容器或 Amazon 彈性運算雲端 (Amazon EC2) 節點支援的負載平衡器組成的運算層，以及可能包括其他資料庫在內的其他服務。該運算層與 DynamoDB 通訊應該使用該區域的本機端點來執行此操作。全域資料表中的資料會複寫到所有其他參與的區域，而且每個區域在其 DynamoDB 資料表周圍都有類似的服務堆疊。

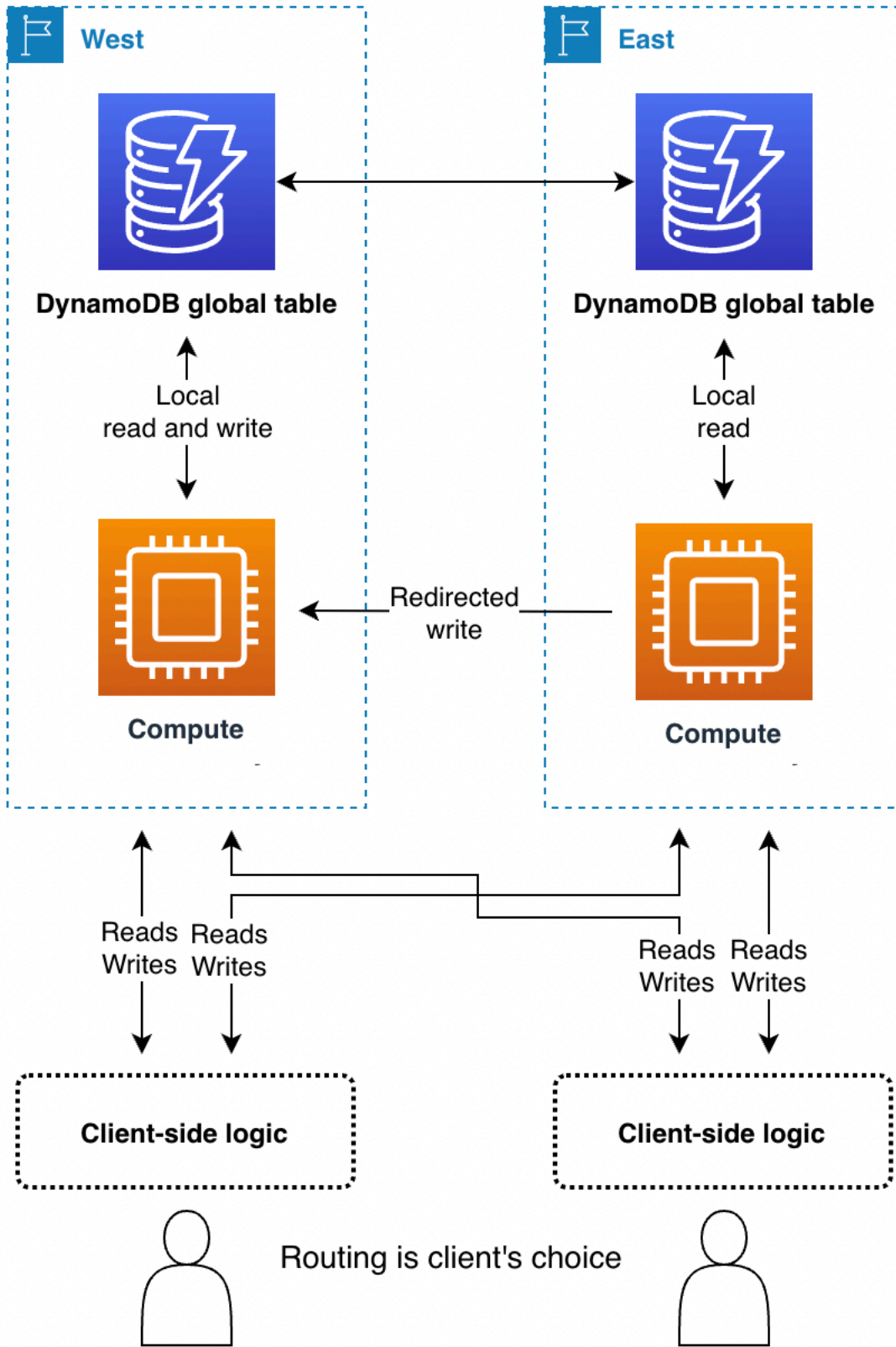
全域資料表會為不同區域中的每個堆疊提供相同資料的本機複本。如果本機 DynamoDB 資料表發生問題，您可以考慮為單一區域中的單一堆疊進行設計，並預期會對次要區域的 DynamoDB 端點進行遠端呼叫。這不是最佳實務。跨區域相關的延遲可能比本機存取高 100 倍。在本地執行一 back-and-forth 系列 5 個請求可能需要幾毫秒，但穿越全球時可能需要幾秒鐘的時間。建議最好將終端使用者路由到另一個區域進行處理。為了確保彈性，您需要跨越多個區域進行複寫，複寫運算層和資料層。

有許多替代技術可將終端使用者請求路由到區域進行處理。最佳選擇取決於您的寫入模式和容錯移轉的考量。本節討論四個選項：用戶端驅動、運算層、Route 53 和 Global Accelerator。

## 用戶端驅動的請求路由

透過用戶端導向的要求路由，使用者用戶端 (例如應用程式、與 JavaScript 其他用戶端的網頁) 將追蹤有效的應用程式端點。在這種情況下，將像是 Amazon API Gateway 這樣的應用程式端點，而不是文字 DynamoDB 端點。然後，終端使用者用戶端使用自己的嵌入邏輯來選擇要與哪個區域進行通訊。它可以根據隨機選擇、觀察到的最低延遲、觀察到的最高頻寬測量或本機執行的運作狀態檢查來選擇。





用戶端驅動的請求路由優點在於，它可以適應現實世界的公共網際網路流量狀況等情況，以便在發現任何性能下降時切換區域。用戶端必須瞭解所有可用端點，但啟動新的區域端點並不常見。

透過寫入任何區域模式，用戶端可以單方面選取其偏好的端點。如果對某個區域的存取受損，用戶端可以路由到另一個端點。

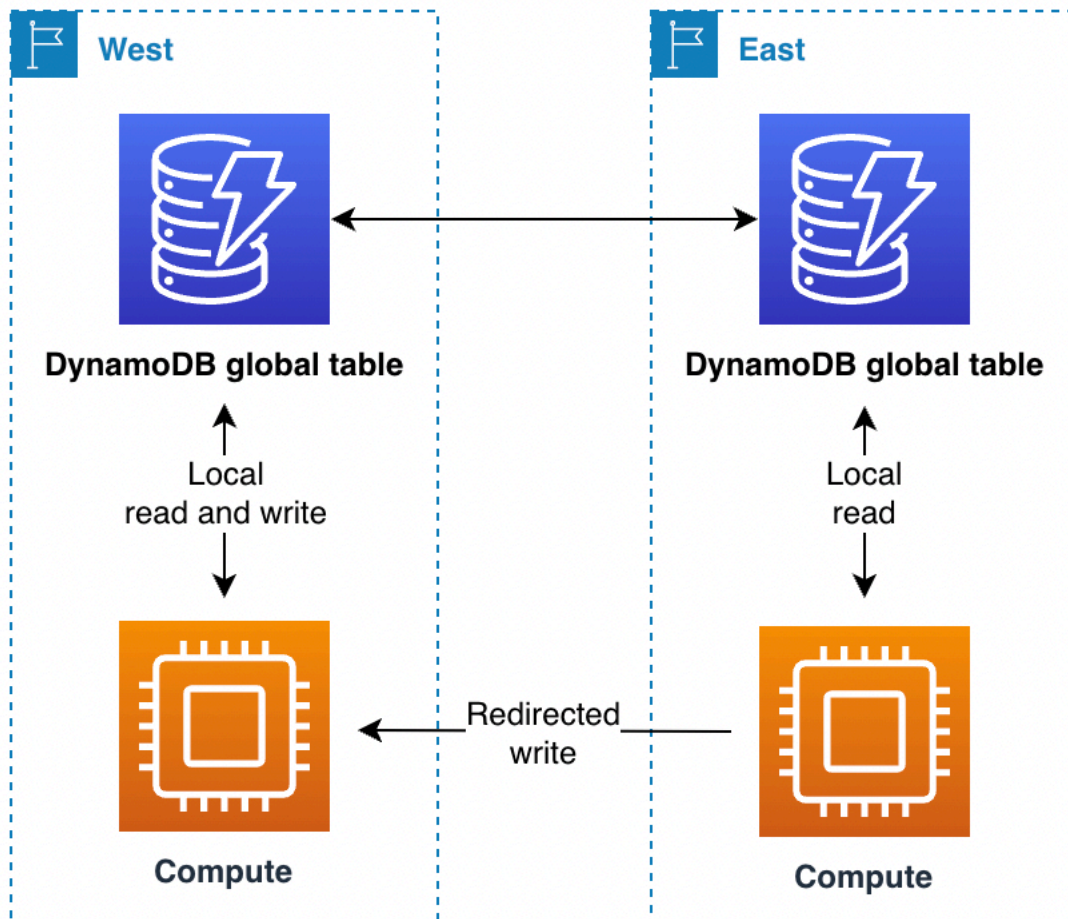
透過寫入單一區域模式，用戶端需要一種機制，將寫入路由到目前使用中的區域。這可能與經驗測試目前接受寫入的區域一樣基本 (注意任何寫入拒絕並退回替代)，或者與呼叫全域協調器查詢目前的應用程式狀態一樣複雜 (可能建立在 Route 53 應用程式復原控制器 (ARC) 路由控制項上，該控制項提供 5 區域定量驅動系統以維護全域狀態以滿足此類需求)。用戶端可以決定讀取是否可以移轉到任何區域以取得最終一致性，還是必須路由到使用中的區域以取得高度一致性。如需進一步資訊，請參閱 [Route 53 的運作方式](#)。

使用寫入您的區域模式時，用戶端必須確定其處理之資料集的主區域。例如，如果用戶端對應一個使用者帳戶，且每個使用者帳戶都連結至一個區域，則用戶端可以從全域登入系統要求適當的端點。

例如，透過網路協助使用者管理其業務金融的金融服務公司，可以使用全域資料表和寫入您的區域模式。每個使用者都必須登入中央服務。該服務會傳回憑證，以及這些憑證將在該區域使用的端點。憑證僅在短時間內有效。之後，網頁會自動協商一個新的登入，提供一個可能將使用者活動重定導向新區域的機會。

## 運算層請求路由

透過運算層請求路由，在運算層中執行的程式碼會決定是否要在本機處理請求，或將它傳遞給在另一個區域中執行的自身複本。當您使用寫入單一區域模式時，運算層可能會偵測到它不是使用中的區域，並允許本機讀取操作，同時將所有寫入操作轉送至另一個區域。此運算層程式碼必須知道資料拓撲和路由規則，並根據指定哪些區域為哪些資料使用中的最新設定，強制可靠的執行這些規則。區域內的外部軟體堆疊不需要知道微服務如何路由讀取和寫入請求。在穩健的設計中，接收區域會驗證其是否為寫入操作的目前主要項目。如果不是，則會產生錯誤，指出需要全域狀態需要修改。如果主要區域正在變更，則接收區域也可能會緩衝寫入操作一段時間。在所有情況下，區域中的運算堆疊只會寫入其本機 DynamoDB 端點，但運算堆疊可能會彼此通訊。

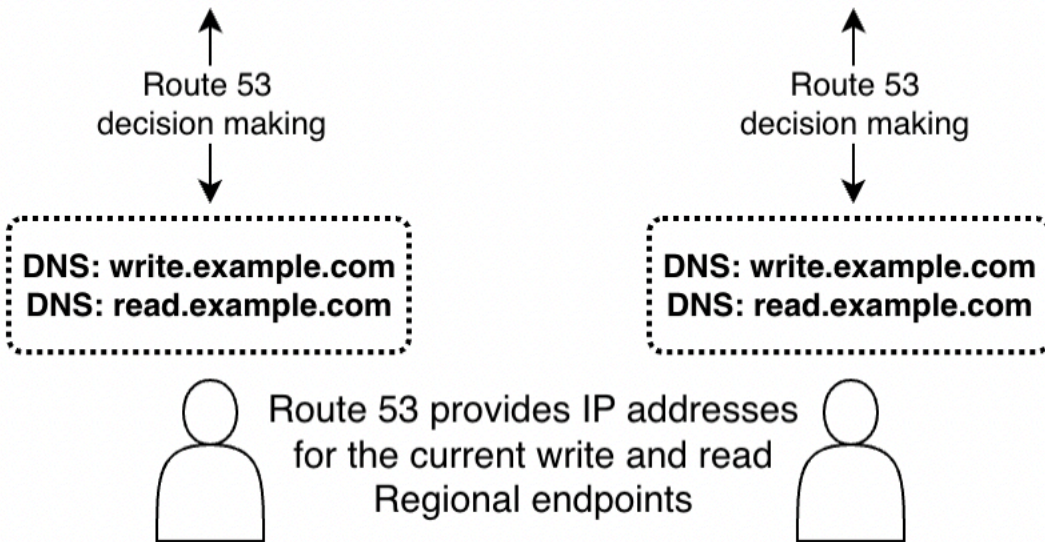
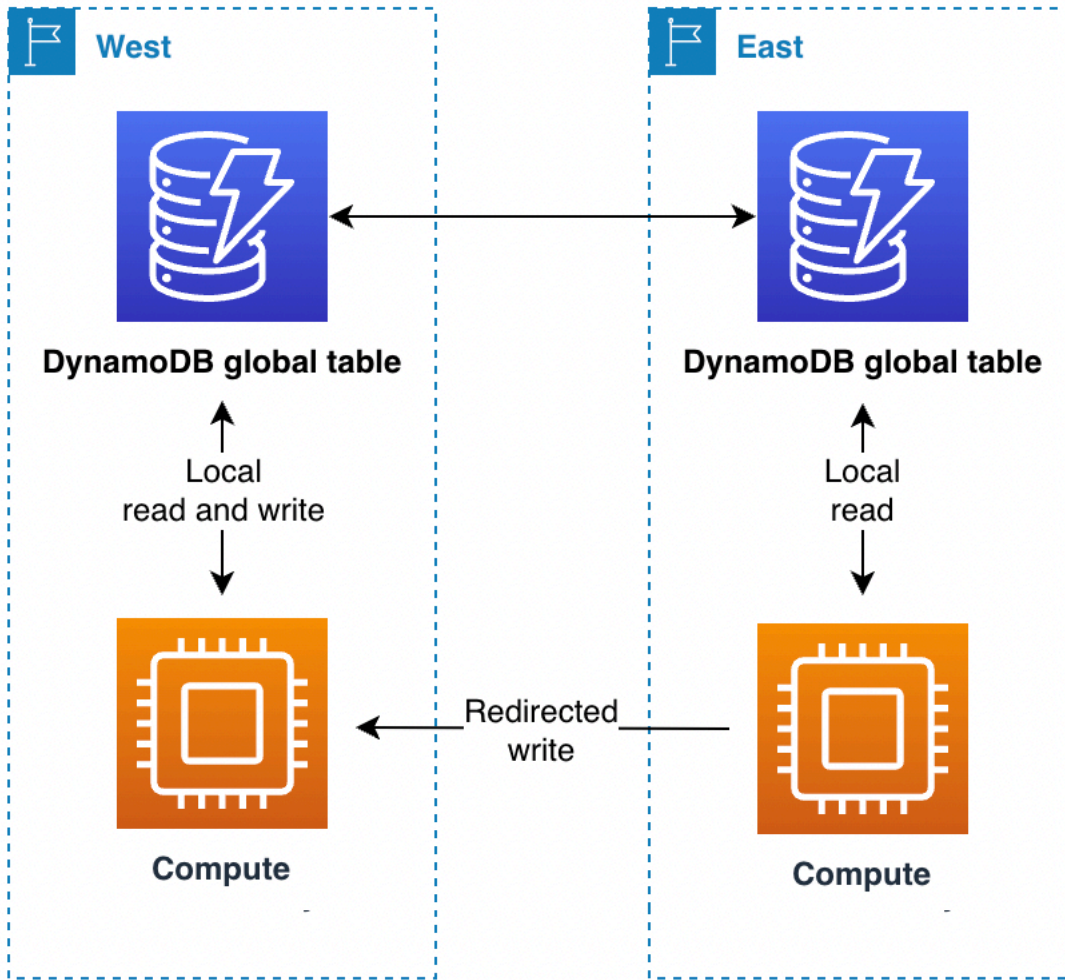


在這個案例中，假設金融服務公司使用「follow-the-sun 單一主要」模型。他們使用系統和程式庫來進行此路由程序。他們 AWS 的整體系統維持整體狀態，類似於 Route 53 ARC 佈線控制。他們使用全域資料表來追蹤哪個區域是主要區域，以及排定下一個主要交換器的時間。所有的讀取和寫入操作都經過程式庫，該程式庫與它們的系統進行協調。該程式庫允許在本機上以低延遲執行讀取操作。對於寫入操作，應用程式會檢查本機區域是否為目前的主要區域。如果是，則寫入操作會直接完成。如果不是，則該程式庫會將寫入任務轉傳到目前主要區域中的程式庫。接收程式庫會確認它也會將自己視為主要區域，如果不是，則會引發錯誤，表示全域狀態的傳播延遲。此方法不直接寫入遠端 DynamoDB 端點，進而提供驗證優勢。

## Route 53 請求路由

Amazon Route 53 應用程式復原控制器是一種網域名稱服務 (DNS) 技術。使用 Route 53 時，用戶端會透過查詢已知的 DNS 網域名稱來請求其端點，Route 53 會傳回與其認為最合適的區域端點對應 IP 地址。Route 53 有一份[用於決定適當區域的路由政策的清單](#)。Route 53 也可以執行[容錯移轉路由](#)，以將流量路由傳送出運作狀態檢查失敗的區域。





- 使用寫入任何區域模式，或者結合後台運算層請求路由使用，Route 53 可以取得完整存取權限，以根據任何複雜的內部規則 (例如，最近網絡鄰近的區域或最近的地理位置或任何其他選擇) 返回區域。
- 使用寫入單一區域模式，可以將 Route 53 設定為返回目前使用中區域 (使用 Route 53 ARC)。

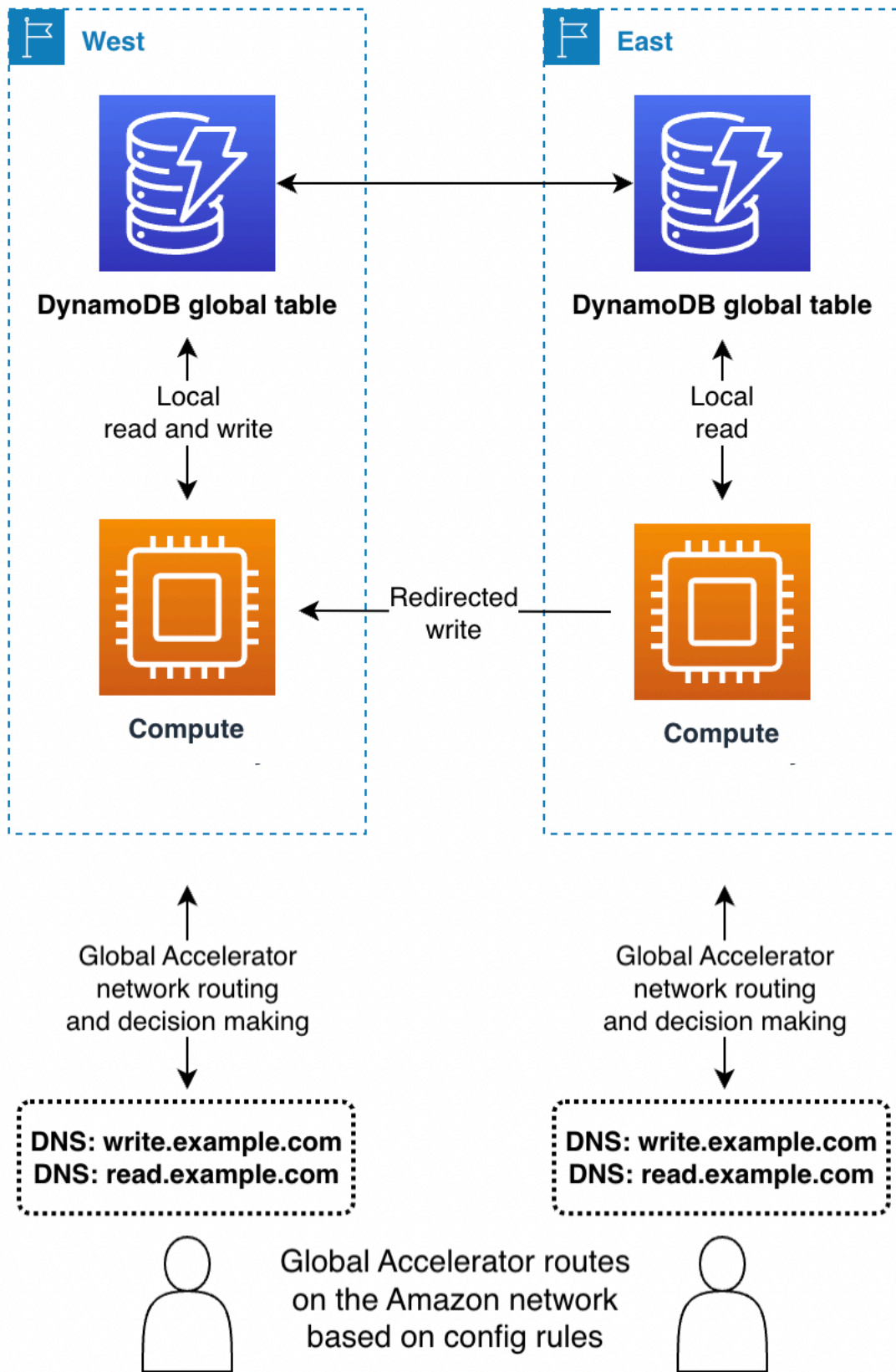
#### Note

用戶端會快取 Route 53 回應中的 IP 地址一段時間，該時間由網域名稱上的存留時間 (TTL) 設定所指定的時間。較長的 TTL 會延長復原時間點目標 (RTO)，讓所有用戶端辨識新的端點。60 秒是容錯移轉使用的典型值。並非所有軟體都能完全遵守 DNS TTL 逾期。

- 寫入您的區域模式時，最好避免使用 Route 53，除非您還使用運算層請求路由。

## Global Accelerator 請求路由

用戶端使用 [AWS Global Accelerator](#) 在 Route 53 中查詢已知的網域名稱。但是，用戶端不會取回對應於區域端點的 IP 位址，而是接收任何傳播靜態 IP 位址，該 IP 位址會路由到最近的 AWS 邊緣位置。從該節點開始，所有流量都會路由到私人 AWS 網路，以及路由到由全域加速器中維護的路由規則所選擇的區域中的某些端點 (例如負載平衡器或 API Gateway)。與使用 Route 53 規則為基礎的路由相比，Global Accelerator 請求路由的延遲較低，因為它可以減少公用網際網路上的流量。此外，由於 Global Accelerator 不依賴 DNS TTL 逾期來變更路由規則，因此可以更快速地調整路由。



- 透過寫入任何區域模式，或者如果與後端的計算層請求路由結合使用，Global Accelerator 可以無縫工作。用戶端會連線到最近的邊緣節點，而且不需要擔心由哪個區域接收請求。
- 使用寫入單一區域的 Global Accelerator 路由規則時，必須將請求傳送到目前使用中的區域。您可以使用運作狀態檢查，以人為方式報告任何區域的故障，而這些區域並未被您的全域系統視為使用中區域。與 DNS 一樣，如果請求可能來自任何區域，則可以使用替代 DNS 網域名稱來路由讀取請求。
- 使用寫入您的區域模式時，最好避免使用 Global Accelerator，除非您同時使用運算層請求路由。

## 使用全域資料表疏散區域

撤離區域是將讀取和寫入活動從該區域遷移出去的過程。最常見的是寫入活動，偶爾會有讀取活動。

### 疏散即時區域

出於多種原因，您可能決定疏散即時區域。逃避可能是常規業務活動的一部分，例如您使用的是 follow-the-sun 寫入一個區域模式。疏散也可能是由於商業決定變更目前作用中的區域、因應 DynamoDB 外部軟體堆疊的故障，或是因為您遇到一般問題，例如區域內的延遲高於平時。

通過寫入任何區域模式，疏散即時區域非常簡單。您可以透過任何路由系統將流量路由到備用區域，並讓已經在疏散區域中發生的寫入操作照常複製。

使用寫入單一區域和寫入您的區域模式時，您必須確保在新的使用中區域中開始寫入之前，使用中區域的所有寫入都已被完整記錄、串流處理和全域傳播。這是必要的，以確保未來的寫入是針對最新版本的資料。

假設區域 A 處於作用中狀態，而區域 B 是被動的 (無論是針對全域資料表或區域 A 中的項目來說)。執行疏散的典型機制是暫停寫入 A、等待充分的時間讓這些操作完全傳播到 B、更新架構堆疊以辨識 B 為使用中，然後繼續寫入操作至 B。沒有指標能夠保證區域 A 的資料已完全複製到區域 B。如果區域 A 狀況良好，請暫停寫入操作至區域 A，然後等待 10 倍 ReplicationLatency 最近指標的最大值，判斷複製是否完成。如果區域 A 狀況不佳並顯示延遲增加的其他區域，則您可以選擇較大的倍數作為等待時間。

### 疏散離線區域

有一個特殊情況需要考慮：如果區域 A 在毫無通知的情況下完全離線怎麼辦？雖然極不可能發生，但仍需審慎考慮。如果發生這種情況，區域 A 中尚未傳輸的任何寫入操作都會在區域 A 重新上線後保留和傳播。寫操作不會遺失，但它們的傳播會無限期延遲。



在這種情況下如何進行應用程式的決策。對於業務連續性，寫入操作可能需要繼續進行新的主要區域 B。不過，如果區域 B 中的某個項目在區域 A 的寫入操作擱置傳播時收到更新，則會在最後一個寫入獲勝模型下抑制傳播。區域 B 中的任何更新都可能抑制傳入的寫入請求。

透過寫入任何區域模式，讀取和寫入可以在區域 B 中繼續，相信區域 A 中的項目最終會傳播到區域 B，並察覺遺失項目的可能性，直到區域 A 重新連線為止。可能的話，您應該考慮重新播放最近的寫入流量 (例如，通過上游事件來源)，以填補任何可能遺失的空白寫入操作，並讓最後一個寫入獲勝衝突解決方案抑制傳入寫入操作的最後傳播。

使用其他寫入模式，您必須考慮工作可以稍微 out-of-date 觀察世界的程度。區域 A 重新上線之前，將丟失一些持續時間較短的寫入操作 (如，ReplicationLatency 追蹤)。業務能夠繼續進行嗎？在某些使用案例中，能夠繼續進行，但在其他情況下，如果沒有額外的緩解機制，可能無法繼續進行。

例如，假設即使在區域故障後，您也需要不間斷維持可用的信用餘額。您可以將餘額分成兩個不同的項目，一個在區域 A 和一個在區域 B，每個項目從可用餘額的一半開始。這將使用寫入您的區域模式。每個區域中處理的交易更新會針對餘額的本機複本寫入。如果區域 A 完全離線，工作仍然可以在區域 B 中繼續進行交易處理，並且寫入操作將限制為區域 B 中所保留的餘額部分，像這樣拆分餘額會導致複雜性，但即使在不確定的待處理寫入操作下，可以提供一個安全業務復原的範例。

作為另一個例子，假設您在擷取 Web 表單資料。您可以使用[開放式並行控制 \(OCC\)](#) 將版本指派給資料項目，並將最新版本嵌入 Web 表單中做為隱藏欄位。每次提交時，只有當資料庫中的版本仍與建置表單的版本相符時，寫入操作才會成功。如果版本不相符，則可以根據資料庫中目前版本重新整理 (或仔細合併) Web 表單，並且使用者可以再次進行操作。OCC 模型通常可以防止其他用戶端覆寫和產生新版本的資料，但也可以在用戶端遇到較舊版本資料的容錯移轉期間提供幫助。

假設您是使用時間戳記作為版本。假設表單首先是在 12:00 針對區域 A 建置，但是 (在容錯移轉之後) 嘗試寫入區域 B，並注意到資料庫中的最新版本是 11:59。在此情況中，用戶端可以等候 12:00 版本傳播到區域 B，然後在該版本之上寫入，或是在 11:59 建置並建立新的 12:01 版本 (寫入後，會在區域 A 復原之後抑制傳入版本)。

作為最後一個範例，金融服務公司將客戶帳戶及其財務交易的相關資料保存在 DynamoDB 資料庫中。如果區域 A 完全中斷，他們想要確保與其帳戶相關的任何寫入活動在區域 B 中完全可用，或者想要隔離他們已知的部分帳戶，直到區域 A 重新連線為止。他們沒有暫停所有業務，而是決定只對確定有未傳播交易的一小部分帳戶暫停業務。為了達成此目的，他們使用了第三個區域，我們將呼叫區域 C。在處理區域 A 中的任何寫入操作之前，他們在區域 C 中對那些暫緩操作 (例如，帳戶新的交易計數) 建立摘要彙總。此彙總足以讓區域 B 判斷其檢視是否完全為最新狀態。此動作會有效鎖定帳戶，從寫入區域 C 開始，直到區域 A 接受寫入操作且區域 B 收到為止。除非作為容錯移轉程序的一部分外，不會使用區域 C 中的資料，之後區域 B 可以和區域 C 交叉比對資料，以檢查其帳戶是否已過期。這些帳戶將被標記為隔離，直到區域 A 復原將部分資料傳播到區域 B。

如果區域 C 當機，則可以改用新的區域 D。區域 C 中的資料非常暫時，幾分鐘後，區域 D 就會有足夠的執行中寫入作業 up-to-date 記錄，以便完全有用。如果區域 B 當機，區域 A 可以繼續接受與區域 C 合作的寫入請求。這家公司願意接受更高的延遲寫入 (到兩個區域：C 和 A)，並且很高興擁有資料模型，可以摘要彙總帳戶狀態。

## 全域資料表的輸送容量規劃

將流量從一個區域移轉到另一個區域需要仔細考慮與容量有關的 DynamoDB 資料表設定。

管理寫入容量時的一些考量：

- 全域資料表必須處於隨需模式，或啟用自動擴展的佈建。
- 如果使用自動擴展進行佈建，則會跨區域複寫寫入設定 (最小、最大和目標使用率)。雖然自動擴展的設定是同步的，但實際佈建的寫入容量可能會在區域之間獨立浮動。
- 您會看到佈建寫入容量不同的原因之一是 TTL 功能。當您在 DynamoDB 中啟用 TTL 時，您可以指定屬性名稱，其值表示項目的到期時間，以秒為單位的 Unix 紀元時間格式。之後，DynamoDB 可刪除項目，而不會產生寫入成本。有了全域資料表，您可以在任何區域設定 TTL，該設定便會自動複寫到另一個區域關聯的全域資料表。當某個項目符合通過 TTL 規則刪除的條件時，該工作可以在任何區域完成。執行刪除操作時不會耗用於源資料表上的寫入單位，但複本表格會取得該刪除操作的複寫寫入操作，並將產生複製寫入單元成本。
- 如果您使用自動擴展，請確定佈建的寫入容量上限設定足以處理所有寫入操作以及所有潛在的 TTL 刪除操作。自動擴展會根據每個區域的寫入耗用進行調整。隨需資料表沒有最大佈建的寫入容量設定，但是資料表層級最大寫入輸送量限制會指定隨需資料表允許的最大持續寫入容量。預設限制為 40,000，但可以調整。我們建議您設定充足，以處理隨需資料表可能需要的所有寫入操作 (包括 TTL 寫入操作)。當您設定全域資料表時，所有參與區域的這個值必須相同。

管理讀取容量時的一些考量：

- 允許區域之間的讀取容量管理設定不同，因為需假設不同的區域可能具有獨立的讀取模式。當第一次將全域複本新增至資料表時，會傳播來源區域的容量。建立之後，您可以調整讀取容量設定，而這個設定不會傳輸到另一端。
- 使用 DynamoDB 自動擴展時，請確保佈建的最大讀取容量設定充足，以處理所有區域的所有讀取操作。在標準操作期間，讀取容量可能會分散到區域，但在容錯移轉期間，資料表應該能夠自動適應增加的讀取工作負載。隨需資料表沒有最大佈建的讀取容量設定，但是資料表層級最大讀取輸送量限制會指定隨需資料表允許的最大持續讀取容量。預設限制為 40,000，但可以調整。如果所有讀取操作都要路由到此單一區域，我們建議您將它設定充足，以處理資料表可能需要的所有讀取操作。

- 如果一個區域中的資料表通常不會收到讀取流量，但在容錯移轉後可能必須吸收大量的讀取流量，您可以提高資料表的佈建讀取容量，等待資料表完成更新，然後再縮減佈建資料表。您可以將資料表保留為佈建模式，也可以將其切換為隨需模式。這會預熱資料表，以接受更高層級的讀取流量。

Route 53 ARC 具有[整備檢查](#)，可用於確認 DynamoDB 區域是否具有類似的資料表設定和帳戶配額，無論您是否使用 Route 53 路由傳送請求。這些就緒性檢查還可以幫助調整帳戶層級的配額，以確保符合。

## 全域資料表和常見問題的就緒性檢查清單

部署全域資料表時，請使用下列檢查清單來制定決策和執行任務。

- 決定有多少以及哪些區域應參與全域資料表。
- 決定應用程式的寫入模式。如需詳細資訊，請參閱[全域資料表的寫入模式](#)。
- 根據您的寫入模式計畫[使用全域資料表的請求路由](#)策略。
- 根據您的寫入模式和路由策略來定義

---

**撤離區域**是將讀取和寫入活動從該區域遷移出去的過程。最常見的是寫入活動，偶爾會有讀取活動。

---

### 疏散即時區域

---

出於多種原因，您可能決定疏散即時區域。逃避可能是常規業務活動的一部分，例如您使用的是 follow-the-sun 寫入一個區域模式。疏散也可能是由於商業決定變更目前作用中的區域、因應 DynamoDB 外部軟體堆疊的故障，或是因為您遇到一般問題，例如區域內的延遲高於平時。

---

通過寫入任何區域模式，疏散即時區域非常簡單。您可以透過任何路由系統將流量路由到備用區域，並讓已經在疏散區域中發生的寫入操作照常複製。

---

使用寫入單一區域和寫入您的區域模式時，您必須確保在新的使用中區域中開始寫入之前，使用中區域的所有寫入都已被完整記錄、串流處理和全域傳播。這是必要的，以確保未來的寫入是針對最新版本的資料。

---

假設區域 A 處於作用中狀態，而區域 B 是被動的 (無論是針對全域資料表或區域 A 中的項目來說)。執行疏散的典型機制是暫停寫入 A、等待充分的時間讓這些操作完全傳播到 B、更新架構堆疊以辨識 B 為使用中，然後繼續寫入操作至 B。沒有指標能夠保證區域 A 的資料已完全複製到區域 B。如果區域 A 狀況良好，請暫停寫入操作至區域 A，然後等待 10 倍 ReplicationLatency

---

最近指標的最大值，判斷複製是否完成。如果區域 A 狀況不佳並顯示延遲增加的其他區域，則您可以選擇較大的倍數作為等待時間。

## 疏散離線區域

有一個特殊情況需要考慮：如果區域 A 在毫無通知的情況下完全離線怎麼辦？雖然極不可能發生，但仍需審慎考慮。如果發生這種情況，區域 A 中尚未傳輸的任何寫入操作都會在區域 A 重新上線後保留和傳播。寫操作不會遺失，但它們的傳播會無限期延遲。

在這種情況下如何進行應用程式的決策。對於業務連續性，寫入操作可能需要繼續進行新的主要區域 B。不過，如果區域 B 中的某個項目在區域 A 的寫入操作擱置傳播時收到更新，則會在最後一個寫入獲勝模型下抑制傳播。區域 B 中的任何更新都可能會抑制傳入的寫入請求。

透過寫入任何區域模式，讀取和寫入可以在區域 B 中繼續，相信區域 A 中的項目最終會傳播到區域 B，並察覺遺失項目的可能性，直到區域 A 重新連線為止。可能的話，您應該考慮重新播放最近的寫入流量 (例如，通過上游事件來源)，以填補任何可能遺失的空白寫入操作，並讓最後一個寫入獲勝衝突解決方案抑制傳入寫入操作的最後傳播。

使用其他寫入模式，您必須考慮工作可以稍微 out-of-date 觀察世界的程度。區域 A 重新上線之前，將丟失一些持續時間較短的寫入操作 (如，ReplicationLatency 追蹤)。業務能夠繼續進行嗎？在某些使用案例中，能夠繼續進行，但在其他情況下，如果沒有額外的緩解機制，可能無法繼續進行。

例如，假設即使在區域故障後，您也需要不間斷維持可用的信用餘額。您可以將餘額分成兩個不同的項目，一個在區域 A 和一個在區域 B，每個項目從可用餘額的一半開始。這將使用寫入您的區域模式。每個區域中處理的交易更新會針對餘額的本機複本寫入。如果區域 A 完全離線，工作仍然可以在區域 B 中繼續進行交易處理，並且寫入操作將限制為區域 B 中所保留的餘額部分，像這樣拆分餘額會導致複雜性，但即使在不確定的待處理寫入操作下，可以提供一個安全業務復原的範例。

作為另一個例子，假設您在擷取 Web 表單資料。您可以使用開放式並行控制 (OCC) 將版本指派給資料項目，並將最新版本嵌入 Web 表單中做為隱藏欄位。每次提交時，只有當資料庫中的版本仍與建置表單的版本相符時，寫入操作才會成功。如果版本不相符，則可以根據資料庫中目前版本重新整理 (或仔細合併) Web 表單，並且使用者可以再次進行操作。OCC 模型通常可以防止其他用戶端覆寫和產生新版本的資料，但也可以在用戶端遇到較舊版本資料的容錯移轉期間提供幫助。

假設您是使用時間戳記作為版本。假設表單首先是在 12:00 針對區域 A 建置，但是 (在容錯移轉之後) 嘗試寫入區域 B，並注意到資料庫中的最新版本是 11:59。在此情況中，用戶端可以等候



12:00 版本傳播到區域 B，然後在該版本之上寫入，或是在 11:59 建置並建立新的 12:01 版本 (寫入後，會在區域 A 復原之後抑制傳入版本)。

作為最後一個範例，金融服務公司將客戶帳戶及其財務交易的相關資料保存在 DynamoDB 資料庫中。如果區域 A 完全中斷，他們想要確保與其帳戶相關的任何寫入活動在區域 B 中完全可用，或者想要隔離他們已知的部分帳戶，直到區域 A 重新連線為止。他們沒有暫停所有業務，而是決定只對確定有未傳播交易的一小部分帳戶暫停業務。為了達成此目的，他們使用了第三個區域，我們將呼叫區域 C。在處理區域 A 中的任何寫入操作之前，他們在區域 C 中對那些暫緩操作 (例如，帳戶新的交易計數) 建立摘要彙總。此彙總足以讓區域 B 判斷其檢視是否完全為最新狀態。此動作會有效鎖定帳戶，從寫入區域 C 開始，直到區域 A 接受寫入操作且區域 B 收到為止。除非作為容錯移轉程序的一部分外，不會使用區域 C 中的資料，之後區域 B 可以和區域 C 交叉比對資料，以檢查其帳戶是否已過期。這些帳戶將被標記為隔離，直到區域 A 復原將部分資料傳播到區域 B。

如果區域 C 當機，則可以改用新的區域 D。區域 C 中的資料非常暫時，幾分鐘後，區域 D 就會有足夠的執行中寫入作業 up-to-date 記錄，以便完全有用。如果區域 B 當機，區域 A 可以繼續接受與區域 C 合作的寫入請求。這家公司願意接受更高的延遲寫入 (到兩個區域：C 和 A)，並且很高興擁有資料模型，可以摘要彙總帳戶狀態。

疏散計畫。

- 擷取每個區域的運作狀態、延遲和錯誤的指標。如需 DynamoDB 指標的清單，請參閱[監控 Amazon DynamoDB 的操作感知 AWS 部落格文章](#)，以取得要觀察的指標清單。您還應該使用 [Synthetic Canaries](#) (旨在檢測故障的人工請求，以煤礦中的金絲雀命名)，以及即時觀察客戶流量。並非所有問題都會出現在 DynamoDB 指標中。
- 為 ReplicationLatency 中任何持續增加設定警示。增加可能表示意外設定錯誤，而全域資料表在不同區域中有不同的寫入設定，這會導致複寫請求失敗並增加延遲。這也可能表示存在區域中斷。一個[很好的例子](#)是如果最近的平均值超過 180,000 毫秒，則發出警示。您也可能會看到 ReplicationLatency 降到 0，這表示停止複寫。
- 為每個全域資料表指派充足的最高讀取和寫入設定值。
- 事先確定疏散區域的原因。如果決定涉及人為判斷，請記錄所有考量因素。這項工作應提前仔細完成，而不是在壓力下進行。
- 為每個動作維護執行手冊，作為當疏散區域時必須採取的措施。通常，全域資料表涉及的工作很少，但移動堆疊的其餘部分可能很複雜。

#### Note

最佳實務是僅依賴資料平面操作，而非控制平面操作，因為在區域故障期間，某些控制平面操作可能會遭到降級。

如需詳細資訊，請參閱部 AWS 部落格文章[使用 Amazon DynamoDB 全域表建置彈性應用程式：第 4 部分](#)。

- 定期測試執行手冊的各個方面，包括區域疏散。未經測試的執行手冊是不可靠的執行手冊。
- 請考慮使用 Resilience Hub 來評估整個應用程式 (包括全域資料表) 的彈性。它透過儀表板提供整體應用程式產品組合彈性狀態的全面檢視。
- 請考慮使用 Route 53 ARC 整備檢查來評估應用程式目前的組態，並追蹤與最佳實務之間的任何差異。
- 撰寫用於 Route 53 或 Global Accelerator 使用的運作狀態檢查時，僅通過延時來確定 DynamoDB 端點是否已啟動是不夠的。這無法涵蓋許多故障模式，例如 IAM 組態錯誤、程式碼部署問題、DynamoDB 外部堆疊中的故障、高於平均讀取或寫入延遲等等。最好執行一組執行完整資料庫流程的呼叫。

## 部署全域資料表的常見問答集 (FAQ)

DynamoDB 全域資料表的整體使用方式有哪些實用原則？

DynamoDB 全域資料表的控制項很少，但仍需要考慮很多因素。您必須決定寫入模式、路由模型和疏散程序。您必須在每個區域檢測您的應用程式，並準備好調整路由或執行疏散以維護全域狀況。獲得的好處是擁有全球分佈式的資料集，具有低延遲讀取和寫入，以及 99.999% 的服務等級協議。

全域資料表的定價為何？

傳統 DynamoDB 資料表的寫入是以寫入容量單位 (WCU，適用於佈建資料表) 或寫入請求單位 (WRU，適用於隨需資料表) 定價。如果您寫入一個 5 KB 的項目，則會產生 5 個單位的費用。全域資料表的寫入是以複寫寫入容量單位 (rWCU，適用於佈建資料表) 或複寫寫入請求單位 (rWRU，適用於隨需資料表) 定價。

rWCU 和 rWRU 包括管理複寫所需的串流媒體基礎設施成本。因此，它們的價格比 WCU 和 WRU 高出 50%。需支付跨區域資料傳輸費用。

直接寫入或複寫寫入項目的每個區域都會產生複寫寫入單位費用。

寫入全域次要索引 (GSI) 會視為本機寫入，並使用一般的寫入單位。

rWCU 目前沒有可用的預留容量。對於具有消耗寫入單元的 GSI 的資料表來說，購買預留容量可能仍然有所幫助。

將新區域新增至全域資料表時，初始啟動程序的費用如同每 GB 還原的資料一樣，再加上跨區域資料傳輸費用。

全域資料表支援哪些區域？

[全域表版本 2019.11.21 \(目前版本\)](#) 在大多數地區都有提供。當您新增複本時，您可以在 DynamoDB 主控台的區域下拉式清單中看到最新的清單。

GSI 如何處理全域資料表？

在[全域表版本 2019.11.21 \(目前\)](#) 中，當您在一個區域中建立 GSI 時，它會在其他參與的區域中自動建立並自動回填。

如何停止全域資料表的複寫？

刪除複本資料表的方式，與刪除其他資料表相同。刪除全域資料表將停止複寫到該區域，並刪除保留在該區域中的資料表複本。不過，您不能在將表的複本作為獨立實體保留的同時停止複制，也無法暫停複寫。

DynamoDB 串流如何與全域資料表互動？

每個全域資料表都會根據其寫入資料表產生獨立串流，無論是從哪裡開始。您可以選擇在一個區域或所有區域中 (獨立) 使用此 DynamoDB 串流。如果您想要處理本機寫入，而不是複寫的寫入操作，可以將自己的區域屬性新增至每個項目。然後，您可以使用 Lambda 事件篩選條件，呼叫 Lambda 函數在本機區域中進行寫入。這有助於插入和更新操作，可惜不能用於刪除操作。

全域資料表如何處理交易？

交易操作僅在最初寫入發生的區域內提供原子性、一致性、隔離性和耐久性 (ACID) 保證。全域資料表不支援跨區域交易。例如，如果您有一個全域資料表，其在美國東部 (俄亥俄) 和美國西部 (奧勒岡) 區域有複本，並且在美國東部 (俄亥俄) 區域執行 `TransactWriteItems` 操作，在這些變更進行複寫之後，您可能會在美國西部 (奧勒岡) 區域看到部分完成的交易。當變更已在來源區域遞交後，這些變更才會複寫至其他區域。

全域資料表如何與 DynamoDB Accelerator (DAX) 快取互動？

全域資料表會透過直接更新 DynamoDB 來略過 DAX，因此 DAX 不會察覺其持有過時資料。當快取的 TTL 到期時，才會重新整理 DAX 快取。

是否會傳播資料表上的標籤？

否，標籤不會自動傳播。

我應該備份所有區域中的資料表還是只備份一個？

答案是取決於備份的目的。如果您想要確保資料持久性，則 DynamoDB 已適當提供該保護。該服務確保的就是持久性。如果您想要保留歷史記錄的快照 (例如，為了符合法規要求)，則在一個區域中進行備份應該就足夠了。您可以使用將備份複製到其他區域 AWS Backup。如果您想要復原錯誤刪除或修改的資料，請在一個區域中使用 [DynamoDB 時間點復原 \(PITR\)](#)。

如何使用部署全域資料表 AWS CloudFormation？

CloudFormation 將 DynamoDB 資料表和全域資料表代表為兩個獨立的資源：`AWS::DynamoDB::Table` 和 `AWS::DynamoDB::GlobalTable`。其中一種方法是使用 `GlobalTable` 建構模組來建立可能是全域資料表的所有資料表。然後，您可以將它們保留為獨立資料表來啟動，並在稍後視需要新增至區域。

在中 CloudFormation，無論複本數目為何，每個全域表都由單一區域中的單一堆疊控制。當您部署範本時，CloudFormation 會建立並更新所有複本，做為單一堆疊作業的一部分。您不應該在多個區域中部署相同的 [AWS::DynamoDB::GlobalTable](#) 資源。這會導致錯誤且不支援。如果您在多個區域中部署應用程式範本，則可以使用條件在單一區域中建立 `AWS::DynamoDB::GlobalTable` 資源。您也可以選擇在與應用程式分開的堆疊中定義 `AWS::DynamoDB::GlobalTable` 資源，並確保其部署到單一區域。

如果您有一般資料表，且想要將其轉換為全域資料表，同時保持管理，CloudFormation 則將刪除原則設定為 `Retain`、從堆疊中移除該資料表、將資料表轉換為主控台的全域資料表，然後將全域資料表作為新資源匯入堆疊。

目前不支援跨帳戶複寫。

## 在 DynamoDB 中管理控制平面的最佳實務

### Note

DynamoDB 引入每秒 2,500 個請求的控制平面限流，並提供重試選項。請參閱以下更多詳細資訊。

DynamoDB 控制平面操作可讓您管理 DynamoDB 資料表，以及相依於索引等資料表的物件。如需有關這些操作的詳細資訊，請參閱 [控制平台](#)。

在某些情況下，您可能需要採取動作，並使用控制平面呼叫傳回的資料做為業務邏輯的一部分。例如，您可能需要知道 `DescribeTable` 返回的 `ProvisionedThroughput` 值。在這些情況下，請遵循下列最佳實務：

- 請勿過度查詢 DynamoDB 控制平面。
- 請勿在同一個程式碼中混用控制平面呼叫和資料平面呼叫。
- 處理控制平面請求上的節流，並使用輪詢重試。
- 從單一用戶端叫用和追蹤特定資源的變更。
- 比起在短時間間隔內多次擷取相同資料表的資料，請快取資料以進行處理。

## 瞭解 AWS 帳單和用量報告的最佳做法

本文件說明與 DynamoDB 相關費用的 `UsageType` 帳單代碼。

AWS 提供成本和用量報告 (CUR)，其中包含所使用服務的資料。您可以使 AWS Cost and Usage Report 用 CSV 格式將帳單報告發佈到 Amazon S3。設定 CUR 時，您可以選擇按小時、日或月劃分時間週期，也可以選擇是否要按資源 ID 劃分使用量。如需產生 CUR 的詳細資訊，請參閱 [建立成本與使用量報告](#)

在 CSV 匯出中，您會找到每一行列出的相關屬性。以下是可能包含的屬性範例：

- `行項目/UsageStartDate`：以世界標準時間為單位的行項目的開始日期和時間 (含在內)。
- `lineitem/UsageEndDate`：對應行項目的結束日期和時間 (以 UTC 為單位)，不包括在內。
- `行程/ProductCode`：對於 DynamoDB，這將是「數據庫」AmazonDynamo
- `lineitem/UsageType`：用法類型的特定描述代碼，如本文件所列舉
- `明細行項目/作業`：提供費用內容的名稱，例如產生費用的作業名稱 (選擇性)。
- `lineitem/ResourceId`：產生使用情況之資源的識別碼。如果 CUR 包含依資源 ID 的劃分，則可使用此選項。
- `lineitem/UsageAmount`：指定期間內產生的使用量。
- `行程/UnblendedCost`：這種用法的成本。
- `行項目/LineItemDescription`：行項目的文字說明。

如需 CUR 資料說明的詳細資訊，請參閱 [成本與使用量報告 \(CUR\) 2.0](#)。請注意，確切的名稱會根據上下文而有所不同。

A UsageType 是具有值的字符串ReadCapacityUnit-Hrs，例如USW2-ReadRequestUnits，EU-WriteCapacityUnit-Hrs，或USE1-TimedPITRStorage-ByteHrs。每個用法類型都以選擇性的區域前置詞開頭。如果不存在，則表示 us-east-1 區域。如果存在，下表將短帳單區域代碼映射到常規的區域代碼和名稱。

例如，名為的用法USW2-ReadRequestUnits表示 us-west-2 中使用的讀取請求單位。

帳單區域代碼	區域代碼	區域名稱
AF1	af-south-1	非洲 (開普敦)
APE1	ap-east-1	亞太區域 (香港)
四月一日	ap-northeast-1	亞太區域 (東京)
四月二日	ap-northeast-2	亞太區域 (首爾)
四月三日	ap-northeast-3	亞太區域 (大阪)
年月 1 日	ap-south-1	亞太區域 (孟買)
APS2	ap-south-2	亞太區域 (海德拉巴)
APS3	ap-southeast-1	亞太區域 (新加坡)
APS4	ap-southeast-2	亞太區域 (悉尼)
APS5	ap-southeast-3	亞太區域 (雅加達)
APS6	ap-southeast-4	亞太區域 (墨爾本)
罐 1	ca-central-1	加拿大 (中部)
歐盟	eu-central-1	歐洲 (法蘭克福)
EUC1	eu-central-2	歐洲 (蘇黎世)
UUN1	eu-north-1	歐洲 (斯德哥爾摩)
EUS1	eu-south-1	歐洲 (米蘭)
EUS2	eu-south-2	歐洲 (西班牙)



帳單區域代碼	區域代碼	區域名稱
歐盟 1	eu-west-1	歐洲 (愛爾蘭)
歐盟 2	eu-west-2	歐洲 (倫敦)
EUW3	eu-west-3	Europe (Paris)
ILC1	中央 -1	以色列 (特拉維夫)
麥克 1	me-central-1	中東 (阿拉伯聯合大公國)
網格 1	me-south-1	Middle East (Bahrain)
表 1	sa-east-1	南美洲 (聖保羅)
使用 1 (預設值)	us-east-1	美國東部 (維吉尼亞北部)
使用 2	us-east-2	美國東部 (俄亥俄)
UGE1	us-gov-east-1	美國東部政府
UGW1	us-gov-west-1	美國政府西部
我們 1	us-west-1	美國西部 (加利佛尼亞北部)
使用 2	us-west-2	美國西部 (奧勒岡)

在下列各節中，我們會在完成 DynamoDB 的費用時使用REG-UsageType模式，其中 REG 會指定使用發生的區域，而 UsageType 是收費類型的程式碼。例如，如果您在 CSV 檔案USW1-ReadCapacityUnit-Hrs中看到的行項目，表示已佈建讀取容量的 US-West-1 產生用量。在這種情況下，清單會說REG-ReadCapacityUnit-Hrs。

## 主題

- [吞吐量容量](#)
- [串流](#)
- [儲存](#)
- [備份與恢復](#)
- [資料傳輸](#)

- [CloudWatch 貢獻者洞察](#)
- [DynamoDB Accelerator \(DAX\)](#)

## 吞吐量容量

### 佈建的容量讀取和寫入

在佈建容量模式下建立 DynamoDB 表時，您可以指定預期應用程式所需的讀取和寫入容量。用法類型取決於您的表格類別 (標準或標準-不常存取)。您可以根據每秒的使用率佈建讀取和寫入，但費用是根據佈建的容量每小時定價。

UsageType	單位	精細程度	描述
註冊 ReadCapacityUnit-小時	RCU 小時	小時	使用標準表格類別在佈建容量模式下讀取的費用。
REG-IA-小時 ReadCapacityUnit	RCU 小時	小時	使用標準 — IA 表格類別在佈建容量模式下讀取的費用。
註冊 WriteCapacityUnit-小時	世界大學時間	小時	使用標準表格類別在佈建容量模式下寫入的費用。
REG-IA-小時 WriteCapacityUnit	世界大學時間	小時	使用標準 — IA 表格類別佈建容量模式的寫入費用。

### 保留容量讀取和寫入

使用預留容量，您會支付一次性預付費用並承諾一段時間的最低佈建消費額。預留容量以折扣的小時費率計費。任何超過預留容量的佈建容量都會依標準佈建容量費率計費。保留容量可用於使用標準表格類別之 DynamoDB 表格上的單一區域、佈建的讀取和寫入容量單位 (RCU 和 WCU)。1 年期和 3 年的預留容量均使用相同的 SKU 計費。



UsageType	單位	精細程度	描述
註冊-: 動態. 讀 HeavyUsage	RCU 小時	預先, 然後每月	預留容量讀取的費用： ：一次性預付費用和每月開始時的每月費用，涵蓋該月所有折扣的承諾 RCU 小時。將具有匹配的零成本 REG ReadCapacityUnit-HRS 明細項目。
註冊-: 動態. 寫 HeavyUsage	世界大學時間	預先, 然後每月	保留容量寫入的費用： ：一次性預付費用和每月開始時的每月費用，涵蓋該月所有折扣的已承諾 WCU 小時。將具有匹配的零成本 REG WriteCapacityUnit-HRS 明細項目。

## 隨需容量讀取和寫入

在隨需容量模式下建立 DynamoDB 表時，您只需為應用程式執行的讀取和寫入付費。讀取和寫入要求的價格取決於您的資料表類別。

UsageType	單位	精細程度	描述
註冊 ReadRequestUnits	RRU	單位	使用標準表格類別以隨需容量模式進行讀取的費用。
瑞嘉 ReadRequestUnits	RRU	單位	使用標準 — IA 表格類別，以隨需容量模式進行讀取的費用。

UsageType	單位	精細程度	描述
註冊 WriteRequestUnits	WRU	單位	使用標準表格類別以隨需容量模式進行寫入的費用。
瑞嘉 WriteRequestUnits	WRU	單位	使用標準 — IA 表格類別，以隨需容量模式進行寫入的費用。

## 全域資料表讀取和寫入

DynamoDB 會根據每個複本表格上使用的資源，收取全域表使用量的費用。對於佈建的全域表格，全域表的寫入要求是以複寫的 WCU (RWCU) 而非標準 WCU 來衡量，而全域表格中全域次要索引的寫入則以 WCU 來衡量。對於隨選全域資料表，寫入要求是以複寫的 WRU (RWRU) 來衡量，而不是標準的 WRU。複製所耗用的 RWCU 或 RWRU 數目取決於您使用的全域表格版本。價格取決於您的桌子類別。

寫入全域次要索引 (GSI) 的費用是使用標準寫入單位 (WCU 和 WRU) 計費。讀取要求和資料儲存的計費方式與單一區域資料表相同。

如果您新增表格複本以在新區域中建立或擴充全域表，DynamoDB 會在新增的區域 (每 GB 還原的資料) 收取表格還原費用。還原的資料會以 REG-RestoreDataSize 位元組的形式收費。[使用 DynamoDB 的隨需備份與還原](#) 有關詳細信息，請參閱。跨區域複寫以及將複本新增至包含資料的表格，也會產生資料傳出的費用。

當您為 DynamoDB 全域表選取隨需容量模式時，您只需為應用程式在每個複本表格上使用的資源付費。

UsageType	單位	精細程度	描述
註冊 ReplWriteCapacityUnit-小時	RWCU 小時	小時	全域表、已佈建、標準表格類別。
REG-IA-小時 ReplWriteCapacityUnit	RWCU 小時	小時	全域表、佈建、標準 — IA 表類別。

UsageType	單位	精細程度	描述
註冊 ReplWrite RequestUnits	RWRU	單位	全局表，按需，標準表類。
瑞嘉 ReplWrite RequestUnits	RWRU	單位	全局表，按需，標準-IA 表類

## 串流

DynamoDB 具有兩種串流技術：DynamoDB Streams 和 Kinesis。每個都有單獨的定價。

DynamoDB Streams 會針對以讀取請求單位讀取資料收取費用。每個 GetRecords API 呼叫都會以串流讀取請求的形式計費。作為 DynamoDB 觸發程序的一部分或 DynamoDB 全域資料表 AWS Lambda 作為複寫的一部分所呼叫的 GetRecords API 呼叫，不需支付任何費用。

UsageType	單位	精細程度	描述
REG 流 RequestsCount	計數	單位	讀取動態資料庫串流的請求單位。

Amazon Kinesis Data Streams 會按變更資料擷取單位收取費用。DynamoDB 會針對每次寫入收取一個變更資料擷取單位的費用 (最多 1 KB)。對於大於 1 KB 的項目，需要額外的變更資料擷取單元。您只需為應用程式執行的寫入付費，而不必管理表格上的輸送量容量。

UsageType	單位	精細程度	描述
註冊 ChangeDataCaptureUnits-Kinesis	疾控中心單位	單位	變更 Kinesis 資料串流的資料擷取單位。

## 儲存

DynamoDB 會新增資料的原始位元組大小以及依據您啟用的功能而定的每個項目儲存額外負荷，來測量應計費資料的大小。

**Note**

CUR 中的儲存使用量值與使用時的儲存體值相比，會更高 DescribeTable，因為 DescribeTable 不包括每個項目的儲存額外負荷。

儲存體是按小時計算，但按月計算，按小時平均費用計算。

雖然儲存區 UsageType 使用 ByteHrs 做為尾碼，但 CUR 中的儲存空間使用量是以 GB 為單位，並以 GB-月計價。

UsageType	單位	精細程度	描述
註冊 TimedStorage-ByteHrs	GB	月	DynamoDB 表和索引所使用的儲存空間數量，用於具有標準資料表類別的表格。
瑞嘉--TimedStorage-ByteHrs	GB	月	DynamoDB 表和索引所使用的儲存空間數量，用於標準 — IA 表類別的表格。

## 備份與恢復

DynamoDB 提供兩種備份類型：時間點復原 (PITR) 備份和隨選備份。使用者也可以從這些備份還原至 DynamoDB 資料表。以下費用是指備份和還原。

新增或移除 Backup 時，會在當月第一個月產生備份儲存體費用，並在整個月內進行調整。如需詳細資訊，請參閱 [了解 Amazon DynamoDB 隨需備份和計費](#) 部落格

UsageType	單位	精細程度	描述
註冊 TimedBackupStorage-ByteHrs	GB	月	DynamoDB 表格和本機次要索引的隨選備份所使用的儲存體。

UsageType	單位	精細程度	描述
蒂姆皮特儲存-ByteHrs	GB	月	point-in-time復原 (PITR) 備份所使用的儲存體。DynamoDB 會在整個月內持續監控已啟用 PITR 的表格大小，以決定您的備份費用和儲存帳單 (只要啟用 PITR)。
註冊-字RestoreDataSize節	GB	大小	從 DynamoDB 備份還原的資料總大小 (包括表格資料、本機次要索引和全域次要索引)，以 GB 為單位。

## AWS Backup

AWS Backup 這是一項全受管備份服務，可讓您輕鬆集中並自動化雲端和內部部署 AWS 服務之間的資料備份。AWS Backup 需支付儲存 (暖或冷儲存)、還原活動以及跨區域資料傳輸的費用。以下 UsageType 費用顯示在「AWS Backup」 ProductCode 而不是「AmazonDynamo數據庫」下。

UsageType	單位	精細程度	描述
註冊 WarmStorage---Dynamo ByteHrs DB	GB	月	DynamoDB 備份在 AWS Backup 整個月內管理的儲存使用，以 GB-月為單位。
註冊 CrossRegion---Dynamo WarmBytes DB	GB	大小	在同一帳戶內或不同帳戶傳輸到不同 AWS 區域的資料。將備份從一個區域複製到另一個區域時，會產生跨區域傳輸費用。費用一律會計

UsageType	單位	精細程度	描述
			入資料傳輸來源的帳戶。
註冊恢復-DynamoDB WarmBytes	GB	大小	從暖儲存區還原的資料總大小，以 GB 為單位。
註冊 ColdStorage--- Dynamo ByteHrs DB	GB	月	DynamoDB 備份在 AWS Backup 整個月內管理的冷儲存使用，以 GB 月為單位。
註冊恢復-DynamoDB ColdBytes	GB	月	從冷儲存還原的資料總大小，以 GB 為單位。

## 匯出與匯入

您可以將資料從 DynamoDB 匯出到 Amazon S3，或將資料從 Amazon S3 匯入新的 DynamoDB 資料表。

雖然 UsageType 使用 Bytes 做為尾碼，但 CUR 中的匯出和匯入用量是以 GB 為單位測量和定價。

UsageType	單位	精細程度	描述
註冊-字ExportDataSize節	GB	大小	將資料匯出至 S3 的費用。DynamoDB 會根據建立匯出時指定時間點的 DynamoDB 基底資料表 (表資料和本機次要索引) 的大小，針對您匯出的資料收費。

UsageType	單位	精細程度	描述
註冊-字ImportDataSize節	GB	大小	從 S3 匯入資料的費用。大小是根據 Amazon S3 中資料的未壓縮物件大小來計算。匯入至含有 GSI 的資料表不會產生額外費用。
註冊-字IncrementalExportDataSize節	GB	大小	從連續備份處理以產生增量匯出的資料大小的費用。

## 資料傳輸

資料傳輸活動可能會顯示與 DynamoDB 服務相關聯。DynamoDB 不會對輸入資料傳輸收費，而且不會對 DynamoDB 與相同 AWS 區域內的其他 AWS 服務之間傳輸的資料收費 (換句話說，每 GB 0.00 美元)。跨區 AWS 域傳輸的資料 (例如，在美國東部 [維吉尼亞北部] 區域的 DynamoDB 與歐洲 [愛爾蘭] 區域的 Amazon EC2 之間傳輸的資料) 將在傳輸過程中兩端收費。

UsageType	單位	精細程度	描述
註冊-DataTransfer 在字節內	GB	單位	從網際網路傳入至 DynamoDB 的資料。
註冊-輸出Data Transfer字節	GB	單位	從 DynamoDB 傳出至網際網路的資料。

## CloudWatch 貢獻者洞察

CloudWatch DynamoDB 的參與者深入解析是一種診斷工具，用於識別 DynamoDB 表格中最常存取和限制的金鑰。下列 UsageType 費用會顯示在「AmazonCloudWatch」 ProductCode 而非「AmazonDynamoDB」之下。

UsageType	單位	精細程度	描述
註冊表順序：ContributorEventsManaged	處理的事件	單位	處理的 DynamoDB 件數量。例如，對於啟用「CloudWatch 參與者見解」的表格，每當某個項目被讀取或寫入時，它都會計為一個事件。如果資料表具有排序索引鍵，則會產生兩個事件的費用。
註冊表順序：ContributorRulesManaged	規則計數	月	當您啟用 Cloud Watch 參與者深入解析時，DynamoDB 會建立規則來識別存取次數最多的項目和節流最多的金鑰。針對為記錄 CloudWatch 參與者深入解析而設定的每個實體 (資料表和 GSI) 新增的規則，會產生此費用。

## DynamoDB Accelerator (DAX)

DynamoDB 加速器 (DAX) 會根據為服務選取的執行個體類型，按小時計費。下列費用是指佈建的 DynamoDB 加速器執行個體。以下 UsageType 費用顯示在「亞馬遜 DAX」 ProductCode 而不是「AmazonDynamoDB」下。

UsageType	單位	精細程度	描述
註冊-: 達克斯-NodeUsage <INSTANCE TYPE>	節點小時	小時	特定執行個體類型的每小時使用量。定價是從節點啟動到終止



UsageType	單位	精細程度	描述
			為止，每節點消耗的小時數計算。消耗的每個不足節點小時將按一小時計費。DAX 針對 DAX 叢集中的每個節點收取費用。如果您的叢集具有多個節點，您會在帳單報告中看到多個明細項目。

例證類型將是一個值，例如下表中的一個值。如需節點類型的詳細資訊，請參閱[節點](#)。

<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	

## 切換容量模式時的考量

建立 DynamoDB 資料表時，您必須選取隨需或佈建容量模式。

您可以隨時將表格從隨選模式切換為佈建的容量模式。當您在容量模式之間進行多次切換時，適用下列條件：

- 您可以隨時在隨選模式下將新建立的表格切換為佈建的容量模式。不過，您只能在資料表建立時間戳記後 24 小時切換回隨選模式。
- 您可以隨時以隨需模式將現有表格切換為已佈建容量模式。但是，您只能在指示切換到隨選的最後一個時間戳記後 24 小時切換回隨選模式。

## 主題

- [從佈建的容量模式切換至隨需容量模式](#)
- [從隨需容量模式切換至佈建容量模式](#)

## 從佈建的容量模式切換至隨需容量模式

在佈建模式中，您可以根據預期的應用程式需求設定讀取和寫入容量。當您將資料表從佈建的模式更新為隨需模式時，不需要指定您預期應用程式將進行的讀取和寫入輸送量。DynamoDB 隨需為讀取和寫入請求提供簡單的 pay-per-request 定價，因此您只需按使用量付費，輕鬆平衡成本和效能。您可以選擇性地為個別隨選資料表和關聯的全域次要索引設定最大讀取或寫入 (或兩者) 輸送量，以協助保持成本和用量限制。如需有關設定特定資料表或索引之最大輸送量的詳細資訊，請參閱[隨選表格的最大輸送量](#)。

當您從佈建的容量模式切換到隨需容量模式時，DynamoDB 會對表格和分區的結構進行多項變更。此程序需要幾分鐘的時間。在切換期間，您資料表提供的傳輸量將與先前佈建的寫入容量單位與讀取容量單位一致。

### 隨需容量模式的初始輸送量

如果您最近第一次將現有的資料表切換為隨選容量模式，表格會有下列先前的尖峰設定，即使該表格之前未提供使用隨選容量模式的流量。

以下是可能的情況範例：

- 設定在 4000 WCU 和 12,000 RCU 以下的任何已佈建表格，先前從未針對更多項目進行佈建。當您第一次將此表切換為隨需時，DynamoDB 將確保向外延展以立即維持至少 4,000 個寫入單位/秒和每秒 12,000 個讀取單位。
- 已佈建的表格，設定為 8,000 個 WCU 和 24,000 個 RCU。當您將此表格切換為隨選時，它將繼續能夠隨時維持至少 8,000 個寫入單位/秒和 24,000 個讀取單位。

- 設定為 8,000 個 WCU 和 24,000 RCU 的佈建資料表，在持續期間每秒消耗 6,000 個寫入單位和 18,000 個讀取單位。當您將此表格切換為隨選時，它將繼續維持至少 8,000 個寫入單位/秒和 24,000 個讀取單位/秒。先前的流量可能會進一步允許資料表在不限流的情況下，維持更高層級的流量。
- 先前設定為 10,000 個 WCU 和 10,000 RCU 的佈建資料表，但目前佈建 10 個 RCU 和 10 個 WCU。當您將此表格切換為隨選時，它將能夠維持至少 10,000 個寫入單位/秒和 10,000 個讀取單位/秒。

## 自動縮放設定

當您將資料表從佈建的模式更新為隨需模式時：

- 如果您使用的是主控台，將會刪除所有的自動調整規模設定 (如果有的話)。
- 如果您使用的是 AWS CLI 或 AWS SDK，則會保留所有 auto 縮放設定。當您再次將資料表更新為佈建的計費模式時，可以套用這些設定。

## 從隨需容量模式切換至佈建容量模式

從隨需模式切換回佈建容量模式時，則該資料表提供的傳輸量將與原先設定為隨需容量模式時達到的先前峰值一致。

## 管理容量

在將資料表從隨需模式更新為佈建的模式時，請考量下列的事項：

- 如果您使用的是 AWS CLI 或 AWS SDK，請使用 Amazon 查看歷史使用量 (和ConsumedReadCapacityUnits指標) 以決定新的輸送量設定，CloudWatch 以選擇表格ConsumedWriteCapacityUnits和全域次要索引的正確佈建容量設定。

### Note

如果您將全域資料表切換為佈建的模式，決定新的輸送量設定時，請針對基礎資料表和全域次要索引，檢視涵跨其所有區域複本的最大耗用量。

- 如果您要從隨選模式切換回佈建模式，請務必將初始佈建的單位設定為足以在轉換期間處理您的資料表或索引容量。

## 管理 Auto Scaling

當您將資料表從佈建的模式更新為隨需模式時：

- 如果您使用的是主控台，建議您使用下列預設值啟用 auto 縮放功能：
  - 目標使用率：70%
  - 佈建容量下限：5 個單位
  - 佈建容量上限：區域最大值
- 如果您使用 AWS CLI 或 SDK，則會保留先前的 auto 縮放設定 (如果有的話)。

# 將 DynamoDB 與其他 AWS 服務搭配使用

Amazon DynamoDB 已與其他 AWS 服務整合，可讓您自動執行重複執行任務，或者建置跨多個服務的應用程式。

## 主題

- [使用 Amazon Cognito 在您的檔案中設定 AWS 登入資料](#)
- [將資料從 DynamoDB 載入 Amazon Redshift](#)
- [在 Amazon EMR 上使用 Apache Hive 處理 DynamoDB 資料](#)
- [與 Amazon S3 整合](#)
- [DynamoDB 零 ETL 與 Amazon 服務整合 OpenSearch](#)
- [與 DynamoDB 整合的最佳做法](#)

## 使用 Amazon Cognito 在您的檔案中設定 AWS 登入資料

建議使用 Amazon Cognito 來獲得您 Web 和行動應用程式的 AWS 登入資料。Amazon Cognito 可協助您避免在檔案上直接寫入 AWS 登入資料。這會使用 AWS Identity and Access Management (IAM) 角色來為應用程式的已驗證及未驗證使用者生成暫時性的登入資料。

例如，若要設定您的 JavaScript 檔案以使用 Amazon Cognito 未驗證角色來存取 Amazon DynamoDB Web 服務，請遵循以下方法：

設定登入資料來與 Amazon Cognito 互動

1. 建立允許未驗證身分的 Amazon Cognito 身分集區。

```
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"
}
```

- 將下列政策複製到名為 `myCognitoPolicy.json` 的檔案。使用在上一步驟取得的 `IdentityPoolId`，來取代身分集區 ID (`us-west-2:12345678-1ab2-123a-1234-a12345ab12`)。

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

- 建立採用上述政策的 IAM 角色。如此一來，Amazon Cognito 會成為可擔任 `Cognito_DynamoPoolUnauth` 角色的信任實體。

```
aws iam create-role --role-name Cognito_DynamoPoolUnauth \
--assume-role-policy-document file://PathToFile/myCognitoPolicy.json --output json
```

- 連接受管政策 (`AmazonDynamoDBFullAccess`)，將 DynamoDB 的完整存取權授予 `Cognito_DynamoPoolUnauth` 角色。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess \
--role-name Cognito_DynamoPoolUnauth
```

**Note**

或者，您也可以將精細定義存取權授予 DynamoDB。如需詳細資訊，請參閱[使用 IAM 政策條件精細定義存取控制](#)。

5. 取得並複製 IAM 角色 Amazon 資源名稱 (ARN)。

```
aws iam get-role --role-name Cognito_DynamoPoolUnauth --output json
```

6. 將 Cognito\_DynamoPoolUnauth 角色新增至 DynamoPool 身分集區。指定的格式是 KeyName=string，其中 KeyName 是 unauthenticated，而字串是在上一步中取得的角色 ARN。

```
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. 在您的檔案中指定 Amazon Cognito 登入資料。並依照上一步的程式碼修改 IdentityPoolId 與 RoleArn。

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
 IdentityPoolId: "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
 RoleArn: "arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth"
});
```

您現在已可使用 Amazon Cognito 登入資料，對 DynamoDB Web 服務執行您的 JavaScript 程式。如需詳細資訊，請參閱《AWS SDK for JavaScript 入門指南》中的[在 Web 瀏覽器設定登入資料](#)。

## 將資料從 DynamoDB 載入 Amazon Redshift

Amazon Redshift 可為 Amazon DynamoDB 補充進階商業智慧功能和強大的 SQL 界面。當您將資料從 DynamoDB 資料表複製到 Amazon Redshift 時，可以對該資料執行複雜的資料分析查詢，包括與 Amazon Redshift 叢集中其他資料表的聯結。

在佈建輸送量方面，DynamoDB 資料表的複製操作會計入該資料表的讀取容量。資料複製之後，Amazon Redshift 中的 SQL 查詢無論如何都不會影響 DynamoDB。原因是您的查詢是處理 DynamoDB 中資料的複本，而不是 DynamoDB 本身。

您必須先建立 DynamoDB 資料表作為資料的目標，才能從 Amazon Redshift 資料表載入資料。請記住，您會將資料從 NoSQL 環境複製至 SQL 環境，而且某個環境中的特定規則不適用於另一個環境。以下是一些需要考量的差異：

- DynamoDB 資料表名稱最多可以包含 255 個字元，包括「.」（點）和「-」（破折號）字元，並且區分大小寫。Amazon Redshift 資料表名稱不得超過 127 個字元以，而且不能包含點或破折號，也不區分大小寫。此外，資料表名稱不能與任何 Amazon Redshift 保留字衝突。
- DynamoDB 不支援 SQL 的 NULL 概念。您必須指定 Amazon Redshift 如何解譯 DynamoDB 中的空屬性值或空白屬性值，將它們視為 NULL 或空欄位。
- DynamoDB 資料類型未直接與 Amazon Redshift 的資料類型相對應。您必須確定 Amazon Redshift 資料表中每個欄位的資料類型和大小都正確，才能容納 DynamoDB 的資料。

以下是 Amazon Redshift SQL 中的 COPY 命令範例：

```
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;
```

在此範例中，DynamoDB 中的來源資料表是 my-favorite-movies-table。Amazon Redshift 中的目標資料表是 favoritemovies。readratio 50 子句會調節使用的佈建輸送量百分比，在此情況下，COPY 命令不會使用超過 50% 為 my-favorite-movies-table 佈建的讀取容量單位。強烈建議將此比率設定為小於平均未用佈建輸送量的值。

如需將 DynamoDB 中的資料載入 Amazon Redshift 的詳細說明，請參閱《[Amazon Redshift 資料庫開發人員指南](#)》中的下列章節：

- [從 DynamoDB 資料表載入資料](#)
- [COPY 命令](#)
- [COPY 範例](#)



# 在 Amazon EMR 上使用 Apache Hive 處理 DynamoDB 資料

Amazon DynamoDB 已與 Apache Hive 整合；Apache Hive 是一種在 Amazon EMR 上執行的資料倉儲應用程式。Hive 可以在 DynamoDB 資料表中讀取和寫入資料，讓您能夠：

- 使用類似 SQL 的語言 (HiveQL) 查詢即時 DynamoDB 資料。
- 將資料從 DynamoDB 資料表複製到 Amazon S3 儲存貯體，反之亦然。
- 將資料從 DynamoDB 資料表複製到 Hadoop 分散式檔案系統 (HDFS) 中，反之亦然。
- 在 DynamoDB 資料表上執行聯結操作。

## 主題

- [概要](#)
- [教學課程：使用 Amazon DynamoDB 和 Apache Hive](#)
- [在 Hive 中建立外部資料表](#)
- [處理 HiveQL 陳述式](#)
- [查詢 DynamoDB 中的資料](#)
- [在 Amazon DynamoDB 之中複製和貼入資料](#)
- [效能調校](#)

## 概要

Amazon EMR 服務可讓您輕鬆快速地以符合成本效益的方式來處理相當大量的資料。若要使用 Amazon EMR，您可以啟動執行 Hadoop 開放原始碼架構的 Amazon EC2 執行個體受管叢集。Hadoop 為實作 MapReduce 演算法的分散式應用程式，其中任務會映射到叢集中的多個節點。每個節點會與其他節點平行處理其指定的工作。最後，在單個節點上會減少輸出，產生最終結果。

您可以選擇啟動 Amazon EMR 叢集，以便維持其持續性或暫時性：

- 持續性叢集會一直執行，直到您將其關閉為止。持續性叢集非常適合用於資料分析、資料倉儲或任何其他互動式用途。
- 暫時性叢集執行的時間足以處理任務流程，然後自動關閉。暫時性叢集非常適合用於定期處理任務，如執行指令碼。

如需 Amazon EMR 架構和管理的相關資訊，請參閱 [《Amazon EMR 管理指南》](#)。

在啟動 Amazon EMR 叢集時，請指定 Amazon EC2 執行個體的初始編號和類型。您還可以指定要在叢集上執行的其他分散式應用程式 (除了 Hadoop 本身)。這些應用程式包含 Hue、Mahout、Pig、Spark。

如需 Amazon EMR 應用程式的相關資訊，請參閱 [《Amazon EMR 版本指南》](#)。

根據叢集組態，您可能具有下列一或多種節點類型：

- **領導節點**：管理叢集，協調 MapReduce 可執行文件和原始資料子集到核心和任務執行個體群組的分佈。該節點也會追蹤每個已執行任務的狀態，並監控執行個體群組的運作狀態。一個叢集中只有一個領導節點。
- **核心節點**：會執行 MapReduce 任務，然後使用 Hadoop 分散式檔案系統 (HDFS) 存放資料。
- **任務節點 (選用)**：執行 MapReduce 任務。

## 教學課程：使用 Amazon DynamoDB 和 Apache Hive

在本教學課程中，您將啟動 Amazon EMR 叢集，然後使用 Apache Hive 處理存放在 DynamoDB 資料表中的資料。

Hive 是 Hadoop 的資料倉儲應用程式，允許您用其處理和分析來自多個來源的資料。Hive 提供類似 SQL 的語言 (HiveQL)，可讓您處理本機存放在 Amazon EMR 叢集或外部資料來源 (如 Amazon DynamoDB) 中的資料。

如需詳細資訊，請參閱 [Hive 教學課程](#)。

### 主題

- [開始之前](#)
- [步驟 1：建立 Amazon EC2 金鑰對](#)
- [步驟 2：啟動 Amazon EMR 叢集](#)
- [步驟 3：連接到領導節點](#)
- [步驟 4：將資料載入 HDFS](#)
- [步驟 5：將資料複製到 DynamoDB](#)
- [步驟 6：查詢 DynamoDB 資料表中的資料](#)
- [步驟 7：\(選用\) 清除](#)

## 開始之前

在此教學課程中，您將需執行下列項目：

- 一個 AWS 帳戶。如果您沒有帳戶，請參閱 [註冊成為 AWS](#)。
- SSH 用戶端 (Secure Shell)。您可以使用 SSH 用戶端連線到 Amazon EMR 叢集的領導節點並執行互動式命令。預設情況下，大多數的 Linux、Unix 和 Mac OS X 裝置都可以使用 SSH 用戶端。Windows 使用者可以下載並安裝支援 SSH 的 [PuTTY](#) 用戶端。

### 下一步驟

#### [步驟 1：建立 Amazon EC2 金鑰對](#)

#### 步驟 1：建立 Amazon EC2 金鑰對

在此步驟中，您要建立連線到 Amazon EMR 領導節點並執行 Hive 命令所需的 Amazon EC2 金鑰對。

1. 請登入 AWS Management Console，並在 <https://console.aws.amazon.com/ec2/> 開啟 Amazon EC2 主控台。
2. 選擇區域 (例如 US West (Oregon))。這應與 DynamoDB 資料表所在的區域相同。
3. 在導覽窗格中，選擇 Key Pairs (金鑰對)。
4. 選擇 Create Key Pair (建立金鑰對)。
5. 在 Key pair name (金鑰對名稱) 中，為金鑰對輸入名稱 (例如 mykeypair)，然後選擇 Create (建立)。
6. 下載私有金鑰檔案。檔案名稱以 .pem 結尾 (例如 mykeypair.pem)。將此私有金鑰檔案保存在安全的地方。您需要它來存取使用此金鑰對啟動的任何 Amazon EMR 叢集。

#### Important

如果遺失金鑰對，您便無法連線到 Amazon EMR 叢集的領導節點。

如需存取金鑰對的詳細資訊，請參閱《Amazon EC2 Linux 執行個體使用者指南》中的 [Amazon EC2 金鑰對](#)。

### 下一步驟

#### [步驟 2：啟動 Amazon EMR 叢集](#)

## 步驟 2：啟動 Amazon EMR 叢集

在此步驟中，您將設定並啟動 Amazon EMR 叢集。DynamoDB 的 Hive 和儲存處理常式已安裝在叢集上。

1. 請在 <https://console.aws.amazon.com/emr> 開啟 Amazon EMR 主控台。
2. 選擇 Create Cluster (建立叢集)。
3. 在 Create Cluster : Quick Options (建立叢集：快速選項) 頁面上，執行下列動作：
  - a. 在 Cluster name (叢集名稱) 中，為叢集輸入名稱 (例如 My EMR cluster)。
  - b. 在 EC2 key pair (EC2 金鑰對) 中，選擇您之前建立的金鑰對。

將其他設定保留為各自的預設設定。

4. 選擇 Create cluster (建立叢集)。

啟動叢集需要幾分鐘的時間。您可以使用 Amazon EMR 主控台中的 Cluster Details (叢集詳細資訊) 頁面監控此流程。

當狀態變更為 Waiting 時，即叢集已準備就緒。

### 叢集日誌檔案和 Amazon S3

Amazon EMR 叢集會產生日誌檔案，其中包含叢集狀態和偵錯資訊的相關資訊。Create Cluster : Quick Options (建立叢集：快速選項) 的預設設定包含設定 Amazon EMR 記錄。

如果尚不存在，則 AWS Management Console 會建立 Amazon S3 儲存貯體。儲存貯體名稱為 `aws-logs-account-id-region`，其中 *account-id* 是 AWS 帳戶號碼、*region* 是啟動叢集的區域 (例如 `aws-logs-123456789012-us-west-2`)。

#### Note

您可以使用 Amazon S3 主控台來檢視日誌檔案。如需詳細資訊，請參閱《Amazon EMR 管理指南》中的[檢視日誌檔案](#)。

除了日誌記錄之外，您還可以將此儲存貯體用於其他目的。例如，您可以使用儲存貯體作為儲存 Hive 指令碼的位置，或將資料從 Amazon DynamoDB 匯出到 Amazon S3 時作為目的地。

### 下一步驟

## 步驟 3：連接到領導節點

### 步驟 3：連接到領導節點

當 Amazon EMR 叢集狀態變更為 Waiting 時，您將能夠使用 SSH 連線到領導節點並執行命令列操作。

1. 在 Amazon EMR 主控台中，選擇叢集的名稱以檢視其狀態。
2. 在 Cluster Details (叢集詳細資訊) 頁面中，尋找 Leader public DNS (領導公有 DNS) 欄位。這是 Amazon EMR 叢集領導節點的公有 DNS 名稱。
3. 在 DNS 名稱右側，選擇 SSH 連結。
4. 請按使用 SSH 連線到領導節點中的說明進行。

根據您的作業系統，選擇 Windows 索引標籤或 Mac/Linux 索引標籤，然後按照連線到領導節點的說明進行。

使用 SSH 或 PuTTY 連線到領導節點後，您應該會看到類似下列內容的命令提示：

```
[hadoop@ip-192-0-2-0 ~]$
```

下一步驟

## 步驟 4：將資料載入 HDFS

### 步驟 4：將資料載入 HDFS

在此步驟中，您要將資料檔案複製到 Hadoop 分散式檔案系統 (HDFS)，然後建立會映射到資料檔案的外部 Hive 資料表。

下載範例資料

1. 下載範例資料封存 (features.zip)：

```
wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/features.zip
```

2. 從封存擷取 features.txt 檔案：

```
unzip features.zip
```

### 3. 檢視 features.txt 檔案的前幾列：

```
head features.txt
```

結果看起來會與下列類似：

```
1535908|Big Run|Stream|WV|38.6370428|-80.8595469|794
875609|Constable Hook|Cape|NJ|40.657881|-74.0990309|7
1217998|Gooseberry Island|Island|RI|41.4534361|-71.3253284|10
26603|Boone Moore Spring|Spring|AZ|34.0895692|-111.410065|3681
1506738|Missouri Flat|Flat|WA|46.7634987|-117.0346113|2605
1181348|Minnow Run|Stream|PA|40.0820178|-79.3800349|1558
1288759|Hunting Creek|Stream|TN|36.343969|-83.8029682|1024
533060|Big Charles Bayou|Bay|LA|29.6046517|-91.9828654|0
829689|Greenwood Creek|Stream|NE|41.596086|-103.0499296|3671
541692|Button Willow Island|Island|LA|31.9579389|-93.0648847|98
```

features.txt 檔案包含來自美國地名委員會 ([http://geonames.usgs.gov/domestic/download\\_data.htm](http://geonames.usgs.gov/domestic/download_data.htm)) 的資料子集。每一列中的欄位代表下列項目：

- 功能 ID (唯一識別碼)
- 名稱
- 類別 (湖泊、森林、溪流等)
- 州
- 緯度 (度數)
- 經度 (度數)
- 高度 (英尺)

### 4. 在命令提示中，輸入以下命令：

```
hive
```

命令提示會變更為此項目：hive>

### 5. 輸入下列 HiveQL 陳述式來建立原生 Hive 資料表：

```
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. 輸入下列 HiveQL 陳述式來載入包含資料的資料表：

```
LOAD DATA
LOCAL
INPATH './features.txt'
OVERWRITE
INTO TABLE hive_features;
```

7. 您現在有一個原生 Hive 資料表已填入來自 `features.txt` 檔案的資料。若要進行驗證，請輸入下列 HiveQL 陳述式：

```
SELECT state_alpha, COUNT(*)
FROM hive_features
GROUP BY state_alpha;
```

輸出應顯示州清單以及每個州的地理特徵數量。

## 下一步驟

### [步驟 5：將資料複製到 DynamoDB](#)

#### 步驟 5：將資料複製到 DynamoDB

在此步驟中，您會將資料從 Hive 資料表 (`hive_features`) 複製到 DynamoDB 中的新資料表。

1. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
2. 選擇 Create Table (建立資料表)。
3. 在 Create DynamoDB table (建立 DynamoDB 資料表) 頁面中，執行下列作業：
  - a. 在 Table (資料表) 中輸入 **Features**。
  - b. 對於 Primary key (主索引鍵)，在 Partition key (分割區索引鍵) 欄位中輸入 **Id**。將資料類型設定為 Number (數字)。

清除 Use default settings (使用預設設定)。為 Provisioned Capacity (佈建容量)，輸入下列內容：

- 讀取容量單位：10
- 寫入容量單位：10

選擇 Create (建立)。

4. 在 Hive 提示中輸入下列 HiveQL 陳述式：

```
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:Alpha");
```

您已經在 DynamoDB 中的 Hive 和 Features (特徵) 資料表之間建立了映射。

5. 輸入下列 HiveQL 陳述式，將資料匯入到 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 會提交將由 Amazon EMR 叢集處理的 MapReduce 任務。完成任務需要幾分鐘的時間。



6. 驗證資料已載入到 DynamoDB :
  - a. 在 DynamoDB 主控台的導覽窗格中，選擇 Tables (資料表)。
  - b. 選擇 Features (特徵) 資料表，然後選擇 Items (項目) 索引標籤來檢視資料。

下一步驟

### [步驟 6：查詢 DynamoDB 資料表中的資料](#)

#### 步驟 6：查詢 DynamoDB 資料表中的資料

在此步驟中，您將使用 HiveQL 來查詢 DynamoDB 中的 Features (特徵) 資料表。嘗試下列 Hive 查詢：

1. 所有功能類型 (feature\_class) 依字母順序排序：

```
SELECT DISTINCT feature_class
FROM ddb_features
ORDER BY feature_class;
```

2. 所有以字母「M」開頭的湖泊：

```
SELECT feature_name, state_alpha
FROM ddb_features
WHERE feature_class = 'Lake'
AND feature_name LIKE 'M%'
ORDER BY feature_name;
```

3. 至少三個特徵高於一英里 (5,280 英尺) 的州：

```
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;
```

下一步驟

### [步驟 7：\(選用\) 清除](#)

## 步驟 7：(選用) 清除

您已完成教學課程，可以繼續閱讀本節，進一步了解如何在 Amazon EMR 中使用 DynamoDB 資料。如果繼續閱讀，您可以決定讓 Amazon EMR 叢集保持正常運作。

如果不再需要叢集，您應該終止它並移除任何相關聯的資源。這將協助您避免為不需要的資源付費。

1. 終止 Amazon EMR 叢集：
  - a. 請在 <https://console.aws.amazon.com/emr> 開啟 Amazon EMR 主控台。
  - b. 選擇 Amazon EMR 叢集，選擇 Terminate (終止)，然後確認。
2. 刪除 DynamoDB 中的 Features (特徵) 資料表：
  - a. 請在 <https://console.aws.amazon.com/dynamodb/> 開啟 DynamoDB 主控台。
  - b. 在導覽窗格中，選擇 Tables (資料表)。
  - c. 選擇 Features (特徵) 資料表。在 Actions (動作) 選單中，選擇 Delete Table (刪除資料表)。
3. 刪除含有 Amazon EMR 日誌檔案的 Amazon S3 儲存貯體：
  - a. 請在 <https://console.aws.amazon.com/s3/> 開啟 Amazon S3 主控台。
  - b. 從儲存貯體清單中，選擇 `aws-logs-accountID-region`，其中 *accountID* 是您的 AWS 帳戶號碼、*region* 是您啟動叢集的區域。
  - c. 在 Actions (動作) 選單中，選擇 Delete (刪除)。

## 在 Hive 中建立外部資料表

在 [教學課程：使用 Amazon DynamoDB 和 Apache Hive](#) 中，您已建立映射到 DynamoDB 資料表的外部 Hive 資料表。在針對外部資料表發出 HiveQL 陳述式時，讀取和寫入操作會傳遞到 DynamoDB 資料表。

您可以將外部資料表視為指向在別處管理和存放的資料來源的指標。在此情況下，基礎資料來源即為 DynamoDB 資料表。(資料表必須已存在。您無法從 Hive 內建立、更新或刪除 DynamoDB 資料表。) 您可以使用 CREATE EXTERNAL TABLE 陳述式建立外部資料表。之後，您可以使用 HiveQL 來處理 DynamoDB 中的資料，就好像資料本機存放在 Hive 中一樣。

**Note**

您可以使用 INSERT 陳述式將資料插入到外部資料表中，並使用 SELECT 陳述式從中選取資料。不過，您無法使用 UPDATE 或 DELETE 陳述式來運用資料表中的資料。

如果不再需要外部資料表，您可以使用 DROP TABLE 陳述式將其移除。在本案例中，DROP TABLE 僅移除 Hive 中的外部資料表。它不會影響基礎 DynamoDB 資料表或其中的任何資料。

**主題**

- [建立外部資料表語法](#)
- [資料類型映射](#)

**建立外部資料表語法**

以下內容顯示用於建立映射到 DynamoDB 資料表的外部 Hive 資料表的 HiveQL 語法：

```
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..."
);
```

第 1 列為 CREATE EXTERNAL TABLE 陳述式的開始，可讓您在其中提供要建立的 Hive 資料表 (*hive\_table*)。

第 2 列指定 *hive\_table* 的資料欄和資料類型。您需要定義與 DynamoDB 資料表中屬性對應的資料欄和資料類型。

第 3 列是 STORED BY 子句，可讓您在其中指定類別來處理 Hive 和 DynamoDB 資料表之間的資料管理。針對 DynamoDB，STORED BY 應設定為 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'。

第 4 列為 TBLPROPERTIES 子句的開始，可讓您在其中為 DynamoDBStorageHandler 定義下列參數：

- `dynamodb.table.name` : DynamoDB 資料表的名稱。
- `dynamodb.column.mapping` : Hive 資料表中資料欄名稱的配對，以及其在 DynamoDB 資料表中的對應屬性。每個配對形式為 `hive_column_name:dynamodb_attribute_name`，且配對以逗號分隔。

注意下列事項：

- Hive 資料表名稱的名稱不一定要與 DynamoDB 資料表名稱相同。
- Hive 資料表資料欄名稱不一定要與 DynamoDB 資料表中的名稱相同。
- 由 `dynamodb.table.name` 指定的資料表必須存在於 DynamoDB 中。
- 針對 `dynamodb.column.mapping` :
  - 您必須映射 DynamoDB 資料表的金鑰結構描述屬性。這包含分割區索引鍵和排序索引鍵 (若有)。
  - 您不需要映射 DynamoDB 資料表的非索引鍵屬性。不過，在查詢 Hive 資料表時，您不會看到這些屬性的任何資料。
  - 如果 Hive 資料表資料欄的資料類型與 DynamoDB 屬性的資料類型不相容，在查詢 Hive 資料表時，您會在這些資料欄中看到 NULL。

#### Note

此 `CREATE EXTERNAL TABLE` 陳述式不會對 `TBLPROPERTIES` 子句執行任何驗證。您為 `dynamodb.table.name` 和 `dynamodb.column.mapping` 提供的值僅在您嘗試存取資料表時由 `DynamoDBStorageHandler` 類別評估。

## 資料類型映射

以下資料表顯示 DynamoDB 資料類型和相容的 Hive 資料類型：

DynamoDB 資料類型	Hive 資料類型
字串	STRING
Number	BIGINT 或 DOUBLE
: 二進位	BINARY
String Set	ARRAY<STRING>

DynamoDB 資料類型	Hive 資料類型
Number Set	ARRAY<BIGINT> 或 ARRAY<DOUBLE>
Binary Set	ARRAY<BINARY>

### Note

下列 DynamoDB 資料類型不受 `DynamoDBStorageHandler` 類別支援，因此無法與 `dynamodb.column.mapping` 搭配使用：

- Map
- 列出
- Boolean
- Null

但是，如果您需要使用這些資料類型，則可以建立名為 `item` 的單一實體來代表整個 DynamoDB 項目，做為對應中索引鍵和值的字串對應。如需詳細資訊，請參閱 [在無資料欄映射的情況下複製資料](#)

如果要映射 Number 類型的 DynamoDB 屬性，則必須選擇適當的 Hive 類型：

- Hive BIGINT 類型用於 8 位元組帶正負號的整數。該類型與 Java 中的 `long` 資料類型相同。
- Hive DOUBLE 類型用於 8 位元雙精度浮點數。該類型與 Java 中的 `double` 類型相同。

如果 DynamoDB 中儲存的數值資料精確度高於您選擇的 Hive 資料類型，則存取 DynamoDB 資料可能會導致精確度損失。

如果您將 Binary 類型的資料從 DynamoDB 匯出到 (Amazon S3) 或 HDFS，則資料會以 Base64-encoded 的字串存放。如果您將資料以 Binary 類型從 Amazon S3 或 HDFS 匯入 DynamoDB，則必須確保資料已編碼為 Base64 字串。

## 處理 HiveQL 陳述式

Hive 是在 Hadoop 上執行的應用程式；Hadoop 是用於執行 MapReduce 任務的批次導向框架。在發出 HiveQL 陳述式時，Hive 會確定它是否可以立即返回結果，或者它是否必須提交 MapReduce 任務。

例如，考量 ddb\_features 資料表 (從 [教學課程：使用 Amazon DynamoDB 和 Apache Hive](#))。下列 Hive 查詢會列印州縮寫和各州境內山峰的數量：

```
SELECT state_alpha, count(*)
FROM ddb_features
WHERE feature_class = 'Summit'
GROUP BY state_alpha;
```

Hive 不會立即返回結果。相反，它會提交由 Hadoop 架構處理的 MapReduce 任務。Hive 將等待任務完成，然後才會顯示查詢結果：

```
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)
```

## 監控和取消任務

當 Hive 啟動 Hadoop 任務時，它會列印該任務的輸出。任務完成狀態隨著任務的進行而更新。在某些情況下，狀態可能會長時間無法進行更新。(在查詢具有較低佈建讀取容量設定的大型 DynamoDB 資料表時，可能會發生這種情況。)

如需在完成之前取消任務，您可以在任何時候輸入 **Ctrl+C**。

## 查詢 DynamoDB 中的資料

以下範例顯示使用 HiveQL 查詢存放在 DynamoDB 中的資料的一些方法。

這些範例參考教學課程 ([步驟 5：將資料複製到 DynamoDB](#)) 中的 `ddb_features` 資料表。

### 主題

- [使用彙總函數](#)
- [使用 GROUP BY 和 HAVING 子句](#)
- [加入兩個 DynamoDB 資料表](#)
- [聯結來自不同來源的資料表](#)

## 使用彙總函數

HiveQL 提供用於摘要資料值的內建函數。例如，您可以使用 MAX 函數尋找所選資料欄的最大值。下列範例會傳回科羅拉多州最高山峰的海拔。

```
SELECT MAX(elev_in_ft)
FROM ddb_features
WHERE state_alpha = 'CO';
```

## 使用 GROUP BY 和 HAVING 子句

您可以使用 GROUP BY 子句跨多筆記錄收集資料。這常會與 SUM、COUNT、MIN 或 MAX 等彙總函數搭配使用。您也可以使用 HAVING 子句捨棄任何不符合特定條件的結果。

以下範例返回 `ddb_features` 資料表中具有五個以上特徵的州的最高海拔清單。

```
SELECT state_alpha, max(elev_in_ft)
```

```
FROM ddb_features
GROUP BY state_alpha
HAVING count(*) >= 5;
```

## 加入兩個 DynamoDB 資料表

以下範例示範將另一個 Hive 資料表 (`east_coast_states`) 映射到 DynamoDB 中的資料表。此 SELECT 陳述式是這兩個資料表之間的聯結。聯結會在叢集上運算並傳回。聯結不會在 DynamoDB 中發生。

考慮名為 `EastCoastStates` 的 DynamoDB 資料表，其中包含下列資料：

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

假設該資料表可用作名為 `east_coast_states` 的 Hive 外部資料表：

```
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");
```

下列聯結會傳回美國東海岸至少具有三個特徵的州：

```
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;
```

## 聯結來自不同來源的資料表

在以下範例中，s3\_east\_coast\_states 是與 Amazon S3 中存放的 CSV 檔案相關聯的 Hive 資料表。此 ddb\_features 資料表與 DynamoDB 中的資料相關聯。下列範例會聯結這兩個資料表，傳回名稱以「New」開頭的州的地理特徵。

```
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%';
```

## 在 Amazon DynamoDB 之中複製和貼入資料

在 [教學課程：使用 Amazon DynamoDB 和 Apache Hive](#) 中，您會將資料從原生 Hive 資料表複製到外部 DynamoDB 資料表，然後查詢外部 DynamoDB 資料表。該資料表為外部資料表，因為它存在於 Hive 之外。即使捨棄映射到它的 Hive 資料表，DynamoDB 中的資料表也不會受到影響。

Hive 是在 DynamoDB 資料表、Amazon S3 儲存貯體、原生 Hive 資料表和 Hadoop 分散式檔案系統 (HDFS) 之間複製資料的絕佳解決方案。本節將提供這些操作的範例。

### 主題

- [在 DynamoDB 和原生 Hive 資料表之間複製資料](#)
- [在 DynamoDB 與 Amazon S3 之間複製資料](#)
- [在 DynamoDB 與 HDFS 之間複製資料](#)
- [使用資料壓縮](#)
- [讀取不可列印的 UTF-8 字元資料](#)

## 在 DynamoDB 和原生 Hive 資料表之間複製資料

如果 DynamoDB 資料表中有資料，您可以將這些資料複製到原生 Hive 資料表。這會為您提供截至資料複製時間的資料快照。

如果您需要執行許多 HiveQL 查詢，但又不想消耗 DynamoDB 佈建的輸送容量，則可以決定這樣做。由於原生 Hive 資料表中的資料是 DynamoDB 資料的複本，而不是「即時」資料，您不應期望查詢可獲得最新資料。

### Note

本節中的範例假設您會遵循 [教學課程：使用 Amazon DynamoDB 和 Apache Hive](#) 中的步驟，並且在 DynamoDB 中有一個名為 `ddb_features` 的外部資料表。

## Example 從 DynamoDB 到原生 Hive 資料表

您可以建立一個原生 Hive 資料表並使用 `ddb_features` 的資料填入內容，如下所示：

```
CREATE TABLE features_snapshot AS
SELECT * FROM ddb_features;
```

您可以隨時重新整理資料：

```
INSERT OVERWRITE TABLE features_snapshot
SELECT * FROM ddb_features;
```

在這些範例中，子查詢 `SELECT * FROM ddb_features` 將從 `ddb_features` 擷取所有資料。如果您只想複製資料的子集，則可以在子查詢中使用 `WHERE` 子句。

下列範例會建立原生 Hive 資料表，其中只包含一些湖泊和山峰的屬性：

```
CREATE TABLE lakes_and_summits AS
SELECT feature_name, feature_class, state_alpha
FROM ddb_features
WHERE feature_class IN ('Lake', 'Summit');
```

## Example 從原生 Hive 資料表到 DynamoDB

使用下列 HiveQL 陳述式，將資料從本機 Hive 資料表複製到 `ddb_features`：

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM features_snapshot;
```

## 在 DynamoDB 與 Amazon S3 之間複製資料

如果 DynamoDB 資料表中有資料，則可以使用 Hive 將資料複製到 Amazon S3 儲存貯體。

如果您想要在 DynamoDB 資料表中建立資料的封存，則可執行此操作。例如，假設您的測試環境需要在 DynamoDB 中使用一組基準測試資料。您可以將基準資料複製到 Amazon S3 儲存貯體，然後執行測試。之後，您可以將 Amazon S3 儲存貯體中的基準資料還原到 DynamoDB，以此重設測試環境。

如果透過 [教學課程：使用 Amazon DynamoDB 和 Apache Hive](#) 作業，則表示您已經擁有一個包含 Amazon EMR 記錄的 Amazon S3 儲存貯體。如果您知道儲存貯體的根路徑，則可以將此儲存貯體用於本節中的範例：

1. 請在 <https://console.aws.amazon.com/emr> 開啟 Amazon EMR 主控台。
2. 對於 Name (名稱)，選擇您的叢集。
3. URI 會列在 Configuration Details (組態詳細資訊) 下方的 Log URI (記錄 URI)。
4. 記下儲存貯體的根路徑。命名慣例是：

```
s3://aws-logs-accountID-region
```

其中 *accountID* 是您的 AWS 帳戶 ID，而區域則是該儲存貯體的 AWS 區域。

### Note

對於這些範例，我們將使用儲存貯體內的子路徑，如下列範例所示：

```
s3://aws-logs-123456789012-us-west-2/hive-test
```

下列程序假設您會遵循教學課程中的步驟，並且在 DynamoDB 中有一個名為 `ddb_features` 的外部資料表。

### 主題

- [使用 Hive 預設格式複製資料](#)
- [以使用者指定格式複製資料](#)
- [在無資料欄映射的情況下複製資料](#)
- [檢視 Amazon S3 中的資料](#)

## 使用 Hive 預設格式複製資料

### Example 從 DynamoDB 到 Amazon S3

使用 INSERT OVERWRITE 陳述式直接寫入至 Amazon S3。

```
INSERT OVERWRITE DIRECTORY 's3://aws-logs-123456789012-us-west-2/hive-test'
SELECT * FROM ddb_features;
```

Amazon S3 中的資料檔案如下所示：

```
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
```

每個欄位都以 SOH 字元分隔 (標題的開頭, 0x01)。在檔案中, SOH 會顯示為 ^A。

### Example 從 Amazon S3 到 DynamoDB

1. 建立一個指向 Amazon S3 中未格式化資料的外部資料表。

```
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. 將資料複製到 DynamoDB。

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_unformatted;
```

## 以使用者指定格式複製資料

如果您想要指定自己的欄位分隔符號字元，則可建立映射至 Amazon S3 儲存貯體的外部資料表。您可以使用此技術建立具有逗號分隔值 (CSV) 的資料檔案。

### Example 從 DynamoDB 到 Amazon S3

1. 建立映射至 Amazon S3 的 Hive 外部資料表。執行此操作時，請確定資料類型與 DynamoDB 外部資料表的資料類型一致。

```
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. 從 DynamoDB 複製資料。

```
INSERT OVERWRITE TABLE s3_features_csv
SELECT * FROM ddb_features;
```

Amazon S3 中的資料檔案如下所示：

```
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 從 Amazon S3 到 DynamoDB

使用單一 HiveQL 陳述式，您可以使用來自 Amazon S3 的資料填入 DynamoDB 資料表：

```
INSERT OVERWRITE TABLE ddb_features
```

```
SELECT * FROM s3_features_csv;
```

## 在無資料欄映射的情況下複製資料

您可以使用原始格式從 DynamoDB 複製資料，並將其寫入至 Amazon S3，而無需指定任何資料類型或資料欄映射。您可以使用此方法建立 DynamoDB 資料的封存，並將其存放到 Amazon S3 中。

### Example 從 DynamoDB 到 Amazon S3

1. 建立與 DynamoDB 資料表相關聯的外部資料表。(此 HiveQL 陳述式中沒有 `dynamodb.column.mapping`。)

```
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. 建立另一個與 Amazon S3 儲存貯體相關聯的外部資料表。

```
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. 將資料從 DynamoDB 複製到 Amazon S3。

```
INSERT OVERWRITE TABLE s3_features_no_mapping
 SELECT * FROM ddb_features_no_mapping;
```

Amazon S3 中的資料檔案如下所示：

```
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
```

每個欄位都以 STX 字元 (文字的開頭, 0x02) 開頭, 並以 ETX 字元 (文字結尾, 0x03) 結尾。在此檔案中, STX 會顯示為 **^B**, ETX 會顯示為 **^C**。

### Example 從 Amazon S3 到 DynamoDB

使用單一 HiveQL 陳述式, 您可以使用來自 Amazon S3 的資料填入 DynamoDB 資料表:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM s3_features_no_mapping;
```

### 檢視 Amazon S3 中的資料

如果您使用 SSH 連線到領導節點, 則可以使用 AWS Command Line Interface (AWS CLI) 來存取 Hive 寫入至 Amazon S3 的資料。

以下步驟的撰寫是假設您已使用本節其中一個程序, 將資料從 DynamoDB 複製到 Amazon S3。

1. 如果您目前處於 Hive 命令提示字元, 請離開至 Linux 命令提示字元。

```
hive> exit;
```

2. 列出 Amazon S3 儲存貯體中的 Hive 測試目錄內容。(這是 Hive 從 DynamoDB 複製資料的位置。)

```
aws s3 ls s3://aws-logs-123456789012-us-west-2/hive-test/
```

回應外觀會與下列類似:

```
2016-11-01 23:19:54 81983 000000_0
```

檔案名稱 (000000\_0) 是由系統產生的。

3. (選用) 您可以將資料檔案從 Amazon S3 複製到領導節點上的本機檔案系統。執行此操作之後, 您可以使用標準 Linux 命令列公用程式來使用檔案中的資料。

```
aws s3 cp s3://aws-logs-123456789012-us-west-2/hive-test/000000_0 .
```

回應外觀會與下列類似：

```
download: s3://aws-logs-123456789012-us-west-2/hive-test/000000_0
to ./000000_0
```

**Note**

領導節點上的本機檔案系統容量有限。請勿將此命令與大於本機檔案系統中可用空間的檔案搭配使用。

## 在 DynamoDB 與 HDFS 之間複製資料

如果 DynamoDB 資料表中有資料，則可以使用 Hive 將資料複製到 Hadoop 分散式檔案系統 (HDFS)。

如果正在執行需要來自 DynamoDB 之資料的 MapReduce 任務，您可能會執行這項操作。如果您將資料從 DynamoDB 複製到 HDFS，Hadoop 可以處理此資訊，同時會使用 Amazon EMR 叢集中的所有可用節點。完成 MapReduce 任務後，您就能將結果從 HDFS 寫入至 DDB。

在下列範例中，Hive 將在下列 HDFS 目錄讀取及寫入內容：`/user/hadoop/hive-test`

**Note**

本節中的範例假設您會遵循 [教學課程：使用 Amazon DynamoDB 和 Apache Hive](#) 中的步驟，並且在 DynamoDB 中有一個名為 `ddb_features` 的外部資料表。

### 主題

- [使用 Hive 預設格式複製資料](#)
- [以使用者指定格式複製資料](#)
- [在無資料欄映射的情況下複製資料](#)
- [存取 HDFS 中的資料](#)

### 使用 Hive 預設格式複製資料

#### Example 從 DynamoDB 到 HDFS

使用 `INSERT OVERWRITE` 陳述式直接寫入至 HDFS。



```
INSERT OVERWRITE DIRECTORY 'hdfs:///user/hadoop/hive-test'
SELECT * FROM ddb_features;
```

HDFS 中的資料檔案外觀如下：

```
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
```

每個欄位都以 SOH 字元分隔 (標題的開頭, 0x01)。在檔案中, SOH 會顯示為 ^A。

Example 從 HDFS 到 DynamoDB

1. 建立映射至 HDFS 中未格式化資料的外部資料表。

```
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. 將資料複製到 DynamoDB。

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_unformatted;
```

以使用者指定格式複製資料

如果要使用不同的欄位分隔符號字元, 則可以建立映射至 HDFS 目錄的外部資料表。您可以使用此技術建立具有逗號分隔值 (CSV) 的資料檔案。

Example 從 DynamoDB 到 HDFS

1. 建立映射至 HDFS 的 Hive 外部資料表。執行此操作時, 請確定資料類型與 DynamoDB 外部資料表的資料類型一致。

```
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. 從 DynamoDB 複製資料。

```
INSERT OVERWRITE TABLE hdfs_features_csv
SELECT * FROM ddb_features;
```

HDFS 中的資料檔案外觀如下：

```
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 從 HDFS 到 DynamoDB

使用單一 HiveQL 陳述式，您可以使用來自 HDFS 的資料填入 DynamoDB 資料表：

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_csv;
```

## 在無資料欄映射的情況下複製資料

您可以使用原始格式從 DynamoDB 複製資料，並將其寫入至 HDFS，而無需指定任何資料類型或資料欄映射。您可以使用此方法建立 DynamoDB 資料的封存，並將其存放到 HDFS 中。

**Note**

如果 DynamoDB 資料表包含 Map、List、Boolean 或 Null 類型的屬性，這便是您可以使用 Hive 將資料從 DynamoDB 複製到 HDFS 的唯一方法。

**Example 從 DynamoDB 到 HDFS**

1. 建立與 DynamoDB 資料表相關聯的外部資料表。(此 HiveQL 陳述式中沒有 `dynamodb.column.mapping`。)

```
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. 建立另一個與 HDFS 目錄相關聯的外部資料表。

```
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. 將資料從 DynamoDB 複製到 HDFS。

```
INSERT OVERWRITE TABLE hdfs_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

HDFS 中的資料檔案外觀如下：

```
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
```

每個欄位都以 STX 字元 (文字的開頭, 0x02) 開頭, 並以 ETX 字元 (文字結尾, 0x03) 結尾。在此檔案中, STX 會顯示為 ^B, ETX 會顯示為 ^C。

### Example 從 HDFS 到 DynamoDB

使用單一 HiveQL 陳述式, 您可以使用來自 HDFS 的資料填入 DynamoDB 資料表:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM hdfs_features_no_mapping;
```

### 存取 HDFS 中的資料

HDFS 是一個分散式檔案系統, 可供 Amazon EMR 叢集中的所有節點存取。如果您使用 SSH 連線到領導節點, 則可以使用命令列工具來存取 Hive 寫入至 HDFS 的資料。

HDFS 與領導節點上的本機檔案系統不盡相同。您無法使用標準 Linux 命令 (例如 cat、cp、mv 或 rm) 處理 HDFS 中的檔案及目錄。這些任務要使用 `hadoop fs` 命令來執行。

以下步驟的撰寫是假設您已使用本節其中一個程序, 將資料從 DynamoDB 複製到 HDFS。

1. 如果您目前處於 Hive 命令提示字元, 請離開至 Linux 命令提示字元。

```
hive> exit;
```

2. 列出 HDFS 中 `/user/hadoop/hive-test` 目錄的內容。(這是 Hive 從 DynamoDB 複製資料的位置。)

```
hadoop fs -ls /user/hadoop/hive-test
```

回應外觀會與下列類似:

```
Found 1 items
-rw-r--r-- 1 hadoop hadoop 29504 2016-06-08 23:40 /user/hadoop/hive-test/000000_0
```

檔案名稱 (000000\_0) 是由系統產生的。

3. 檢視檔案的內容:

```
hadoop fs -cat /user/hadoop/hive-test/000000_0
```

**Note**

在此範例中，檔案相對較小 (大約 29 KB)。請審慎對非常大或包含不可列印字元的檔案使用此命令。

4. (選用) 您可以將資料檔案從 HDFS 複製到領導節點上的本機檔案系統。執行此操作之後，您可以使用標準 Linux 命令列公用程式來使用檔案中的資料。

```
hadoop fs -get /user/hadoop/hive-test/000000_0
```

此命令不會覆寫檔案。

**Note**

領導節點上的本機檔案系統容量有限。請勿將此命令與大於本機檔案系統中可用空間的檔案搭配使用。

## 使用資料壓縮

在使用 Hive 在不同的資料來源之間複製資料時，您可以要求即時資料壓縮。Hive 提供了幾個壓縮解碼器。您可以在 Hive 工作階段期間選擇一個。在執行此操作時，會以指定格式來壓縮資料。

以下範例會使用 Lempel-Ziv-Oberhumer (LZO) 演算法壓縮資料。

```
SET hive.exec.compress.output=true;
SET io.seqfile.compression.type=BLOCK;
SET mapred.output.compression.codec = com.hadoop.compression.lzo.LzopCodec;

CREATE EXTERNAL TABLE lzo_compression_table (line STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE lzo_compression_table SELECT *
FROM hiveTableName;
```

Amazon S3 中產生的檔案會具有系統產生的名稱，且以 .lzo 結尾 (例如，8d436957-57ba-4af7-840c-96c2fc7bb6f5-000000.lzo)。

可用的壓縮轉碼器為：

- `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`

## 讀取不可列印的 UTF-8 字元資料

您可以在建立 Hive 資料表時使用 `STORED AS SEQUENCEFILE` 子句，以此讀取和寫入不可列印的 UTF-8 字元資料。SequenceFile 是 Hadoop 二進位檔案格式。您需要使用 Hadoop 來讀取此檔案。下列範例顯示如何將資料從 DynamoDB 匯出到 Amazon S3。您可以使用此功能處理不可列印的 UTF-8 編碼字元。

```
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;
```

## 效能調校

在建立映射至 DynamoDB 資料表的 Hive 外部資料表時，您不會從 DynamoDB 消耗任何讀取或寫入容量。不過，Hive 資料表上的讀取和寫入活動 (例如 `INSERT` 或 `SELECT`) 會直接轉譯為基礎 DynamoDB 資料表上的讀取和寫入作業。

Amazon EMR 上的 Apache Hive 會實作自己的邏輯來平衡 DynamoDB 資料表上的輸入/輸出負載，並盡量減少超出資料表佈建輸送量的可能性。每個 Hive 查詢結束時，Amazon EMR 都會傳回執行時間指標，包括超出佈建輸送量的次數。您可以將此資訊與 DynamoDB 資料表上的 CloudWatch 指標一起使用，改善後續要求的效能。

Amazon EMR 主控台可為叢集提供基本監控工具。如需詳細資訊，請參閱《Amazon EMR 管理指南》中的[檢視及監控叢集](#)。

您也可以使用以 Web 類型工具 (例如 Hue、Ganglia 和 Hadoop Web 介面) 來監控叢集及 Hadoop 任務。如需詳細資訊，請參閱《Amazon EMR 管理指南》中的[檢視 Amazon EMR 叢集上託管的 Web 介面](#)。

本節說明可在外部 DynamoDB 資料表上調校 Hive 作業效能所採取的步驟。

主題

- [DynamoDB 佈建輸送量](#)
- [調整映射器](#)
- [其他主題](#)

## DynamoDB 佈建輸送量

在針對外部 DynamoDB 資料表發出 HiveQL 陳述式時，DynamoDBStorageHandler 類別會產生適當的低階 DynamoDB API 請求 (這些請求會消耗佈建的輸送量)。如果 DynamoDB 資料表上的讀取或寫入容量不足，則會對請求進行調節，繼而導致 HiveQL 效能緩慢。因此，您應該確保資料表擁有足夠的輸送容量。

例如，假設您已為 DynamoDB 資料表佈建 100 單位的讀取容量。這將讓您每秒讀取 409,600 個位元組 (100 × 4 KB 讀取容量單位大小)。現在假設資料表包含 20 GB 的資料 (21,474,836,480 個位元組)，而且您想要使用 SELECT 陳述式來選擇使用 HiveQL 的所有資料。您可以估計查詢執行需要多長時間，如下所示：

「 $21,474,836,480 / 409,600 = 52,429$  秒 = 14.56 小時」

在此案例中，DynamoDB 資料夾是瓶頸。這將無助於新增更多 Amazon EMR 節點，因為 Hive 輸送量限制為每秒只有 409,600 個位元組。減少 SELECT 陳述式所需時間的唯一方法，是增加 DynamoDB 資料表佈建的讀取容量。

您可以執行類似的計算，估計將大量載入資料帶入映射至 DynamoDB 資料表之 Hive 外部資料表所需的時間。決定每個項目所需的寫入容量單位總數 (小於 1KB = 1、1-2KB = 2，以此類推)，並將其乘以要載入的項目數。這將為您提供所需的寫入容量單位數目。將該數目除以每秒配置的寫入容量單位數目。這將產生載入資料表所需的秒數。

您應該定期監控資料表的 CloudWatch 指標。如需在 DynamoDB 主控台取得快速概觀，請選擇您的資料表，然後選擇 Metrics (指標) 索引標籤。您可以從此處檢視已耗用的讀取和寫入容量單位，以及已調節的讀取和寫入請求。

## 讀取容量

Amazon EMR 會根據資料表佈建輸送量設定，管理您的 DynamoDB 資料表的請求負載。不過，如果在任務輸出中發現大量的 `ProvisionedThroughputExceeded` 訊息，您可以調整預設讀取率。若要執行此操作，您可以修改 `dynamodb.throughput.read.percent` 組態變數。您可以使用 SET 命令在 Hive 命令提示字元中設定此變數：

```
SET dynamodb.throughput.read.percent=1.0;
```

此變數僅持續用於目前的 Hive 工作階段。如果您退出 Hive 並於之後返回查看，`dynamodb.throughput.read.percent` 會傳回其預設值。

`dynamodb.throughput.read.percent` 的值可能介於 0.1 及 1.5 (含) 之間。0.5 代表預設讀取率，表示 Hive 會嘗試使用資料表一半的讀取容量。如果將值增加到 0.5 以上，Hive 會增加請求率；如果將值減少到 0.5 以下，則會降低讀取請求率。(根據不同因素，例如在 DynamoDB 資料表中是否有統一金鑰分佈等，實際讀取率可能會有所不同。)

如果您注意到 Hive 經常耗盡資料表的佈建讀取容量，或者如果讀取請求調節過多，請嘗試將 `dynamodb.throughput.read.percent` 減少到 0.5 以下。如果資料表有足夠的讀取容量，並且您想要提升 HiveQL 作業的回應能力，則可以將值設定在 0.5 以上。

## 寫入容量

Amazon EMR 會根據資料表佈建輸送量設定，管理您的 DynamoDB 資料表的請求負載。不過，如果在任務輸出中發現大量的 `ProvisionedThroughputExceeded` 訊息，您可以調整預設寫入率。若要執行此操作，您可以修改 `dynamodb.throughput.write.percent` 組態變數。您可以使用 SET 命令在 Hive 命令提示字元中設定此變數：

```
SET dynamodb.throughput.write.percent=1.0;
```

此變數僅持續用於目前的 Hive 工作階段。如果您退出 Hive 並於之後返回查看，`dynamodb.throughput.write.percent` 會傳回其預設值。

`dynamodb.throughput.write.percent` 的值可能介於 0.1 及 1.5 (含) 之間。0.5 代表預設寫入率，表示 Hive 會嘗試使用資料表一半的寫入容量。如果將值增加到 0.5 以上，Hive 會增加請求率；如果將值減少到 0.5 以下，則會降低寫入請求率。(根據不同因素，例如在 DynamoDB 資料表中是否有統一金鑰分佈等，實際寫入率可能會有所不同。)



如果您注意到 Hive 經常耗盡資料表的佈建寫入容量，或者如果寫入請求調節過多，請嘗試將 `dynamodb.throughput.write.percent` 減少到 0.5 以下。如果資料表有足夠的容量，並且您想要提升 HiveQL 作業的回應能力，則可以將值設定在 0.5 以上。

在使用 Hive 將資料寫入 DynamoDB 時，請確認寫入容量單位數大於叢集中的映射器數量。例如，考慮使用由 10 個 m1.xlarge 節點的 Amazon EMR 叢集。此 m1.xlarge 節點類型提供 8 個映射器任務，因此叢集總共有 80 個映射器 (10 × 8)。如果 DynamoDB 資料表的寫入容量單位少於 80 個，則 Hive 寫入作業可能會耗用該資料表的所有寫入輸送量。

若要判斷 Amazon EMR 節點類型的映射器數目，請參閱《Amazon EMR 開發人員指南》中的[任務組態](#)。

如需映射器的詳細資訊，請參閱 [調整映射器](#)。

## 調整映射器

Hive 啟動 Hadoop 任務時，此任務會由一或多個映射器任務處理。假設 DynamoDB 資料表擁有足夠的輸送容量，您可以修改叢集中的映射器數目，潛在改善效能。

### Note

在 Hadoop 任務中使用的映射器任務數目會受到輸入分割影響，其中 Hadoop 會將資料細分成邏輯區塊。如果 Hadoop 未執行足夠的輸入分割，則您的寫入作業可能會無法使用 DynamoDB 資料表中所有可用的寫入輸送量。

## 增加映射器的數目

Amazon EMR 中的每個映射器都有每秒 1 MiB 的最大讀取率。叢集中的映射器數目視叢集中的節點大小而定。(如需節點大小和每個節點映射器數目的相關資訊，請參閱《Amazon EMR 開發人員指南》中的[任務組態](#)。)

如果您的 DynamoDB 資料表擁有足夠的讀取輸送量，您可以執行下列其中一項操作來嘗試增加映射器的數目：

- 增加叢集中節點的大小。例如，如果您的叢集正在使用 m1.large 節點 (每個節點有三個映射器)，則可以嘗試升級至 m1.xlarge 節點 (每個節點有八個映射器)。
- 增加叢集中節點的數目。例如，如果您有 m1.xlarge 節點的三節點叢集，則可用的映射器總共有 24 個。如果將叢集 (擁有相同類型的節點) 大小加倍，則會有 48 個映射器。

您可以使用 AWS Management Console 來管理叢集中節點的大小或數目。(您可能需要重新啟動叢集，才能讓這些變更生效。)

另一種增加映射器數目的方式，便是修改 `mapred.tasktracker.map.tasks.maximum` Hadoop 組態參數。(這是 Hadoop 參數，而非 Hive 參數。您無法從命令提示字元以互動方式加以修改。) 若想增加 `mapred.tasktracker.map.tasks.maximum` 的值，您只需增加映射器的數目，無需增加節點大小或數量。不過，如果您設定的值太高，叢集節點可能會耗盡記憶體。

在第一次啟動 Amazon EMR 叢集時，您可以將 `mapred.tasktracker.map.tasks.maximum` 設定為自舉動作。如需詳細資訊，請參閱《Amazon EMR 管理指南》中的 [\(選用\) 建立引導操作來安裝其他軟體](#)。

### 減少映射器的數目

如果使用 SELECT 陳述式從映射至 DynamoDB 的外部 Hive 資料表中選取資料，Hadoop 任務可視需要使用任意數量的任務，最多可達叢集中的映射器數目上限。在此案例中，長時間執行的 Hive 查詢可能會消耗 DynamoDB 資料表的所有佈建讀取容量，對其他使用者造成負面影響。

您可以使用 `dynamodb.max.map.tasks` 參數來設定映射任務的上限：

```
SET dynamodb.max.map.tasks=1
```

此數值必須等於或大於 1。在 Hive 處理查詢時，從 DynamoDB 資料表讀取時，產生的 Hadoop 作業使用不會超過 `dynamodb.max.map.tasks`。

### 其他主題

下列是使用 Hive 存取 DynamoDB 來調整應用程式的一些更多方法。

#### Retry duration (重試持續時間)

依預設，如果 Hive 未在兩分鐘內從 DynamoDB 傳回任何結果，Hive 將會重新執行 Hadoop 任務。您可以藉由修改 `dynamodb.retry.duration` 參數來調整此間隔：

```
SET dynamodb.retry.duration=2;
```

此值必須是非零的整數，代表重試間隔中的分鐘數。`dynamodb.retry.duration` 預設值為 2 (分鐘)。

## 平行資料請求

針對單一資料表的多個資料請求，無論是來自一名以上的使用者還是一個以上的應用程式，都會耗盡讀取佈建輸送量，降低效能。

## 處理持續時間

DynamoDB 中的資料一致性取決於每個節點讀取和寫入操作的順序。當 Hive 查詢正在進行中時，另一個應用程式可以將新資料載入 DynamoDB 資料表，或修改或刪除現有的資料。在此案例中，Hive 查詢的結果有可能無法反映執行查詢的過程中所做的資料變更。

## 請求時間

當對 DynamoDB 資料表的需求較低時，排程存取 DynamoDB 資料表的 Hive 查詢可改善效能。例如，若您大多數的應用程式使用者住在舊金山，您可以選擇在大多數的使用者仍在睡眠中的太平洋標準時間 (PST) 早上 4 點匯出每日資料，而不更新您 DynamoDB 資料庫中的紀錄。

# 與 Amazon S3 整合

Amazon DynamoDB 匯入與匯出功能提供簡易且有效率的方式，可在 Amazon S3 和 DynamoDB 資料表之間移動資料，無需撰寫任何程式碼。

DynamoDB 匯入和匯出功能可協助您移動、轉換和複製 DynamoDB 表格帳戶。您可以從 S3 來源匯入，也可以將 DynamoDB 表資料匯出到 Amazon S3，並使用 Athena、Amazon SageMaker 等 AWS 服務，並分析您的資料並 AWS Lake Formation 擷取可操作的見解。您也可以將資料直接匯入新的 DynamoDB 資料表，以便建置具有 10 毫秒的規模效能之新應用程式、促進資料表和帳戶之間的資料共用，並簡化災難復原和業務持續性計畫。

## 主題

- [從 Amazon S3 匯入 DynamoDB 資料：運作方式](#)
- [DynamoDB 資料匯出至 Amazon S3：運作方式](#)

## 從 Amazon S3 匯入 DynamoDB 資料：運作方式

若要將資料匯入 DynamoDB，您的資料必須位於 Amazon S3 儲存貯體中，且為 CSV、JSON 或 Amazon Ion 格式。資料可壓縮為 ZSTD 或 GZIP 格式，也可以直接匯入為未壓縮格式。來源資料可以是單一 Amazon S3 物件，或使用相同字首的多個 Amazon S3 物件。

您的資料將匯入到新的 DynamoDB 資料表中，該資料表將在您發起匯入請求時建立。您可以使用次要索引建立此資料表，然後在匯入完成後立即查詢並更新所有主要索引和次要索引的資料。您也可以在此匯入完成後新增全域資料表複本。

### Note

在 Amazon S3 匯入流程期間，DynamoDB 會建立一個新的目標資料表，並將其匯入。此功能目前不支援匯入至現有資料表。

從 Amazon S3 匯入不會耗用新資料表的寫入容量，因此您不需要佈建任何額外容量即可將資料匯入 DynamoDB。資料匯入定價取決於 Amazon S3 中來源資料的未壓縮大小，該資料會在匯入時進行處理。由於格式化或來源資料的不一致，導致無法載入資料表中的項目，也會在匯入流程中計費。如需詳細資訊，請參閱 [Amazon DynamoDB 定價](#)。

您可以從不同帳戶所擁有的 Amazon S3 儲存貯體匯入資料，前提是您有讀取該儲存貯體的正确許可。新資料表也可能位於與來源 Amazon S3 儲存貯體不同的區域。如需詳細資訊，請參閱 [Amazon Simple Storage Service 設定和許可](#)。

匯入時間與 Amazon S3 中的資料特性直接相關。這包括資料大小、資料格式、壓縮方法、資料分佈的一致性、Amazon S3 物件數量以及其他相關變數。具體而言，索引鍵均勻分佈的資料集，匯入速度會比偏斜的資料集更快。例如，若您的次要索引鍵是以月份分割，而您的資料全部來自十二月，那麼匯入資料就會需要更久時間。

與索引鍵相關聯的屬性在基礎資料表中應為唯一。如有任何索引鍵並非唯一，則匯入程序會覆寫相關聯的項目，直到僅留下最後一次覆寫。例如，若主索引鍵是月份，而有多個項目設定為 9 月份，則每個新項目都會覆寫先前寫入的項目，並僅保留一個「月份」主索引鍵設定為 9 月的項目。此情況下，匯入資料表描述中處理的物件數，與目標資料表中的物件數量不相符。

AWS CloudTrail 記錄表格匯入的所有主控台和 API 動作。如需詳細資訊，請參閱 [使用 AWS CloudTrail 記錄 DynamoDB 操作](#)。

以下影片介紹如何直接從 Amazon S3 匯入到 DynamoDB。

## [從 Amazon S3 匯入](#)

### 主題

- [請求在 DynamoDB 中匯入資料表](#)
- [DynamoDB 的 Amazon S3 匯入格式](#)
- [匯入格式配額與驗證](#)

- [將 Amazon S3 匯入到 DynamoDB 的最佳實務](#)

## 請求在 DynamoDB 中匯入資料表

DynamoDB 匯入功能可讓您將 Amazon S3 儲存貯體中的資料匯入至新的 DynamoDB 資料表。[您可以使用 DynamoDB 主控台、CLI CloudFormation 或 DynamoDB API 來請求資料表匯入。](#)

如果要使用 AWS CLI，必須先對其進行配置。如需詳細資訊，請參閱 [存取 DynamoDB](#)。

### Note

- 匯入資料表功能可與多個不同的 AWS 服務 (例如 Amazon S3 和 CloudWatch) 在開始匯入之前，請確定叫用匯入 API 的使用者或角色具有該功能所依賴之所有服務和資源權限。
- 請不要在匯入進行時修改 Amazon S3 物件，因為這可能會導致操作失敗或遭到取消。

如需如何處理錯誤和故障診斷的詳細資訊，請參閱 [匯入格式配額與驗證](#)。

## 主題

- [設定 IAM 許可](#)
- [使用 AWS Management Console 請求匯入](#)
- [獲取有關過去導入的詳細信息 AWS Management Console](#)
- [使用要求匯入 AWS CLI](#)
- [獲取有關過去導入的詳細信息 AWS CLI](#)

## 設定 IAM 許可

您可以從您擁有讀取許可的任何 Amazon S3 儲存貯體匯入資料。來源儲存貯體無需與來源資料表位於相同區域或擁有相同的擁有者。您的 AWS Identity and Access Management (IAM) 必須包含來源 Amazon S3 儲存貯體上的相關動作，以及提供偵錯資訊所需的 CloudWatch 許可。以下所示為政策範例。

```
{
 "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/*"
 }
]
}

```

## Amazon S3 許可

從另一個帳戶擁有的 Amazon S3 儲存貯體來源上開始匯入時，請確保該角色或使用者可以存取 Amazon S3 物件。您可以執行 Amazon S3 GetObject 命令並使用憑證以確認這點。使用 API 時，Amazon S3 儲存貯體擁有者參數會預設為目前使用者的帳戶 ID。對於跨帳戶匯入，請確定此參數已正確填入儲存貯體擁有者的帳戶 ID。下列程式碼為來源帳戶中 Amazon S3 儲存貯體政策的範例。

```
{
 "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

建立要匯入的新表格時，如果您選取不屬於 DynamoDB 擁有的靜態加密金鑰，則必須提供操作使用客戶管理金鑰加密的 DynamoDB 表所需的 AWS KMS 權限。如需詳細資訊，請參閱[授權使用 AWS KMS 金鑰](#)。如果 Amazon S3 物件使用伺服器端加密 KMS (SSE-KMS) 加密，請確保啟動匯入的角色或使用者具有使用金鑰解密的存取權。AWS KMS 此功能不支援客戶提供加密金鑰 (SSE-C) 加密的 Amazon S3 物件。

## CloudWatch 權限

發起匯入的角色或使用者，將需要與匯入相關聯的日誌群組和日誌串流之建立和管理員權限。

## 使用 AWS Management Console 請求匯入

以下範例示範如何使用 DynamoDB 主控台將現有資料匯入名為 MusicCollection 的新的資料表。

### 請求資料表匯入

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在主控台左側的導覽窗格中，選擇 Exports to S3 (從 S3 匯入)。
3. 在出現的頁面上，選取 Import from S3 (從 S3 匯入)。
4. 選擇 Import from S3 (從 S3 匯入)。
5. 在來源 S3 網址中，輸入 Amazon S3 來源網址。



如果您擁有來源儲存貯體，請選擇瀏覽 S3 進行搜尋。或者，您也可以使用下列格式輸入值區的 URL — `s3://bucket/prefix`。這 `prefix` 是一個 Amazon S3 key prefix。這可能是您要匯入的 Amazon S3 物件名稱，或是您要匯入的所有 Amazon S3 物件共用的 key prefix 置詞。

#### Note

您不能使用與 DynamoDB 匯出請求相同的前置詞。匯出功能會為所有匯出建立資料夾結構和資訊清單檔案。如果您使用相同的 Amazon S3 路徑，則會導致錯誤。

而是將匯入指向資料夾，該資料夾包含來自該特定匯出的資料。在這種情況下，正確路徑的格式將是 `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/Data/`，其中 `XXXXXXXX-XXXXXX` 是導出 ID。您可以在導出 ARN 中找到導出 ID，它具有以下格式 — `arn:aws:dynamodb:<Region>:<AccountID>:table/<TableName>/export/<XXXXXXXX-XXXXXX>`。例如 `arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/export/01234567890123-a1b2c3d4`。

6. 指定您是否為 S3 bucket owner (S3 儲存貯體擁有者)。如果來源時段屬於其他帳戶，請選取不同的 AWS 科目。然後輸入儲存貯體擁有者的帳戶 ID。
7. 在 Import file compression (匯入檔案壓縮)，請選取 No compression (無壓縮)、GZIP (GZIP) 或者 ZSTD (ZSTD)，視情況而定。
8. 選取適當的匯入檔案格式。選項包括 DynamoDB JSON (DynamoDB JSON)、Amazon Ion (Amazon Ion) 或者 CSV (CSV)。如果您選擇 CSV (CSV)，您將有兩個額外的選項：CSV header (CSV 標頭) 和 CSV delimiter character (CSV 分隔符號)。

針對 CSV header (CSV 標頭)，請選擇是否要從檔案的第一行取得標頭或自訂。如果您選擇 Customize your headers (自訂標頭)，您可以指定您要匯入的標頭值。用此方法指定的 CSV 標頭區分大小寫，且應包含目標資料表的索引鍵。

針對 CSV delimiter character (CSV 分隔符號)，您可以設置用以分隔項目的字元。根據預設，會選取逗號。如果您選擇 Custom delimiter character (自訂分隔符號字元)，分隔符號必須符合正則表達式模式：`[, ; : | \t ]`。

9. 選擇 Next (下一步) 按鈕，然後選擇建立以儲存資料的新資料表選項。



**Note**

「主索引鍵」和「排序索引鍵」必須符合檔案中的屬性，否則匯入將會失敗。屬性區分大小寫。

10. 選擇 Next (下一步) 以再次檢視您的匯入選項，然後按一下 Import (匯入)，以啟動匯入任務。首先，您會看到「資料表」中列出的新資料表狀態為「建立中」。目前無法存取資料表。
11. 匯入完成後，狀態將顯示為「啟用」，您可以開始使用資料表。

### 獲取有關過去導入的詳細信息 AWS Management Console

您可以按一下導覽側列中的 Import from S3 (從 S3 匯入)，然後選取 Imports (匯入) 索引標籤，找到您過去曾執行的匯入任務資訊。匯入面板包含您在過去 90 天內建立的所有匯入清單。選取 Imports (匯入) 索引標籤中所列出任務的 ARN，即可擷取該匯入的相關資訊，包括您選擇的任何進階組態設定。

### 使用要求匯入 AWS CLI

下列範例會從名為字首儲存貯體的 S3 儲存貯體，將 CSV 格式的資料匯入到名為 target-table 的新資料表。

```
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**

如果您選擇使用受 AWS Key Management Service (AWS KMS) 保護的金鑰加密匯入，則金鑰必須與目的地 Amazon S3 儲存貯體位於相同的區域。

### 獲取有關過去導入的詳細信息 AWS CLI

您可以使用 list-imports 命令尋找過去曾執行的匯入任務資訊。此命令會傳回您在過去 90 天內建立的所有匯入清單。請注意，雖然匯入任務中繼資料會在 90 天後過期，也無法在此清單中找到早於該時間的任務，但 DynamoDB 不會刪除 Amazon S3 儲存貯體或匯入期間建立的任何物件。

```
aws dynamodb list-imports
```

若要擷取特定匯入任務的詳細資訊 (包括任何進階組態設定)，請使用 `describe-import` 命令。

```
aws dynamodb describe-import \
 --import-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/exp
```

## DynamoDB 的 Amazon S3 匯入格式

DynamoDB 可以匯入三種格式的資料：CSV、JSON 和 Amazon Ion。

### 主題

- [CSV](#)
- [DynamoDB JSON](#)
- [Amazon Ion](#)

### CSV

CSV 格式的檔案由以換行符分隔的多個項目組成。根據預設，DynamoDB 會將匯入檔案的第一行解讀為標頭，並預期欄會以逗號分隔。只要標頭符合檔案中的欄數，您也可以定義欲套用的標頭。如果標頭定義明確，檔案的第一行將會匯入為值。

#### Note

從 CSV 檔案匯入時，除了基礎資料表的雜湊範圍與索引鍵和次要索引外，所有欄都會匯入為 DynamoDB 字串。

### 轉義雙逸出

CSV 檔案中存在的任何雙引號字元都必須逸出。如果未經逸出 (如下列範例)，則匯入將失敗：

```
id,value
"123",Women's Full Lenth Dress
```

如果使用兩組雙引號逸出，則相同的匯入將成功：

```
id,value
```

```
""""123""",Women's Full Lenth Dress
```

當文字經過正確逸出和匯入之後，就會如原始 CSV 檔案中一樣顯示：

```
id,value
"123",Women's Full Lenth Dress
```

## DynamoDB JSON

DynamoDB JSON 格式的檔案可能由多個項目物件組成。每個個別物件均採用 DynamoDB 的標準封送處理 JSON 格式，而新行則用作項目分隔符號。新增功能中，預設支援從時間點的匯出可作為匯入來源。

### Note

新行會用作 DynamoDB JSON 格式檔案的項目分隔符號，不應在項目物件中使用。

```
{
 "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

新行會用作 DynamoDB JSON 格式檔案的項目分隔符號，不應在項目物件中使用。

```
{
 "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](#) 是輸入豐富、自行描述、階層資料序列化格式，旨在處理快速開發、解耦及日常遇到的效率挑戰等問題，同時設計以服務為導向的大規模架構。

將資料匯入為 Ion 格式時，DynamoDB 資料表中使用的 DynamoDB 資料類型會映射至 Ion 資料類型。

	Ion 至 DynamoDB 資料類型轉換	B
1	Ion Data Type	DynamoDB Representation
2	string	String (s)
3	bool	Boolean (BOOL)
4	decimal	Number (N)
5	blob	Binary (B)

	Ion 至 DynamoDB 資料類型轉換	B
6	list (with type annotation \$dynamodb_SS, \$dynamodb_NS, or \$dynamodb_BS)	Set (SS, NS, BS)
7	list	List
8	struct	Map

Ion 檔案中的項目以新行分隔。每行都以 Ion 版本標記開頭，後面接著 Ion 格式的項目。

#### Note

下列範例中，為了便於閱讀，Ion 格式檔案的項目採用多行格式。

```
$ion_1_0 {
 Item:{
 Authors:$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"
 }
}
```

## 匯入格式配額與驗證

### 匯入配額

Amazon S3 的 DynamoDB 匯入可支援多達 50 個並行匯入任務，在 us-east-1、us-west-2 和 eu-west-1 區域的匯入來源物件大小總計達 15TB。在所有其他區域中，最多支援 50 個並行匯入任務，大

小總計為 1TB。每個匯入任務在所有區域中最多可以使用 50,000 個 Amazon S3 物件。這些預設配額會套用至每個帳戶。如果您認為需要修改這些配額，請聯絡您的客戶團隊，我們會 case-by-case 根據需要進行考量。如需 DynamoDB 限制的詳細資訊，請參閱 [Service Quotas](#)。

## 驗證錯誤

在匯入流程期間，DynamoDB 可能會在剖析資料時遇到錯誤。DynamoDB 會針對每個錯誤發出 CloudWatch 記錄，並保留遇到的錯誤總數計數。如果 Amazon S3 物件本身格式錯誤或其內容無法形成 DynamoDB 項目，我們可能會略過處理物件的剩餘部分。

### Note

如果 Amazon S3 資料來源有多個共用相同索引鍵的項目，這些項目將會覆寫，直到僅留下一個項目為止。這看起來像是匯入了 1 個項目而忽略了其他項目。重複的項目將以隨機順序覆寫，不會計為錯誤，也不會發出到 CloudWatch 記錄檔。

匯入完成後，您可以查看匯入的項目總數、錯誤總數，以及處理的項目總數。如需進一步的疑難排解，您也可以檢查所匯入項目的總大小與所處理資料的總大小。

匯入錯誤分為三類：API 驗證錯誤、資料驗證錯誤和組態錯誤。

## 驗證錯誤

API 驗證錯誤是來自同步 API 的項目層級錯誤。常見原因是許可問題、缺少必要參數以及參數驗證失敗。ImportTable 請求所擲回的例外狀況中，包含 API 呼叫失敗原因的詳細資訊。

## 驗證錯誤

資料驗證錯誤可能發生在項目層級或檔案層級。匯入期間，會先根據 DynamoDB 規則驗證項目，然後再匯入目標資料表。當項目驗證失敗且未匯入時，匯入任務會略過該項目，並繼續下一個項目。在工作結束時，匯入狀態設定為 FAILED FailureCode，ItemValidationError 且 FailureMessage 「有些項目驗證檢查失敗且未匯入。請檢查 CloudWatch 錯誤日誌以獲取更多詳細信息。」

造成資料驗證錯誤的常見原因包括物件無法剖析、物件格式不正確 (輸入指定 DYNAMODB\_JSON，但物件不在 DYNAMODB\_JSON 中)，以及結構描述與指定來源資料表的金鑰不相符。

## 組態錯誤

組態錯誤通常是因為權限驗證而導致的工作流程錯誤。匯入工作流程會在接受請求之後檢查某些權限。如果在呼叫 Amazon S3 等任何必要相依性時發生問題，或程序 CloudWatch 會將匯入狀態標示為「失



敗」。所以 `failureCode` 和 `failureMessage` 指向失敗原因。在適用的情況下，失敗訊息也包含要求識別碼，您可以用來調查中失敗的原因 CloudTrail。

常見的組態錯誤包括 Amazon S3 儲存貯體的 URL 錯誤，以及沒有存取 Amazon S3 儲存貯體、CloudWatch 日誌和用於解密 Amazon S3 物件的 AWS KMS 金鑰的權限。如需詳細資訊，請參閱[使用與資料金鑰](#)。

## 驗證 Amazon S3 物件

若要驗證來源 S3 物件，請執行下列步驟。

### 1. 驗證資料格式與壓縮類型

- 請確認指定字首下的所有相符 Amazon S3 物件都具有相同的格式 (DYNAMODB\_JSON、DYNAMODB\_ION、CSV)
- 請確認指定字首下的所有相符 Amazon S3 物件都以相同的方式壓縮 (GZIP、ZSTD、無)

#### Note

Amazon S3 對象不需要具有相應的擴展名 ( `.csv` / `.json` / `.ion` / `.gz` / `.zstd` 等 )，因為調用中指定的輸入格式優先。ImportTable

### 2. 驗證匯入資料是否符合所需的資料表結構描述

- 請確定來源資料中的每個項目都有主索引鍵。匯入的排序索引鍵為選用。
- 請確定與主索引鍵和排序索引鍵相關聯的屬性類別符合 Table 和 GSI 結構描述中的屬性類別，如資料表建立參數所指定

## 故障診斷

### CloudWatch 日誌

對於匯入失敗的工作，詳細的錯誤訊息會張貼至 CloudWatch 記錄檔。若要存取這些記錄檔，請先 ImportArn 從輸出擷取，然後使用此命令進行描述匯入：

```
aws dynamodb describe-import --import-arn arn:aws:dynamodb:us-east-1:ACCOUNT:table/
target-table/import/01658528578619-c4d4e311
}
```

輸出範例：

```
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."
 }
}

```

從上述回應擷取日誌群組並匯入 ID，並用其擷取錯誤日誌。匯入 ID 是 ImportArn 欄位的最後一個路徑元素。日誌群組的名稱為 /aws-dynamodb/imports。錯誤日誌串流的名稱為 import-id/error。此例子中，為 01658528578619-c4d4e311/error。

缺少項目中的金鑰 pk

如果來源 S3 物件不包含作為參數的主索引鍵，匯入將會失敗。例如，當您將匯入的主索引鍵定義為欄名稱「pk」時。

```

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"}'

```

來源物件 import-test.csv 中缺少欄「pk」，其中包含下列內容：

```

title,artist,year_of_release
The Dark Side of the Moon,Pink Floyd,1973

```

由於資料來源中缺少主索引鍵，因此這項匯入會因為項目驗證錯誤而失敗。

範例 CloudWatch 錯誤記錄檔：

```

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"
}

```

此錯誤日誌表示「一個或多個參數值無效：缺少項目中的金鑰 pk」。由於此匯入工作失敗，現在存在資料表「target-table」但其為空白，因為沒有匯入任何項目。已處理第一個項目，物件的項目驗證失敗。

若要修正此問題，請先將「target-table」刪除 (若不再需要)。然後使用來源物件中存在的主索引鍵欄名稱，或將來源資料更新為：

```

pk,title,artist,year_of_release
Albums::Rock::Classic::1973::AlbumId::ALB25,The Dark Side of the Moon,Pink Floyd,1973

```

目標資料表存在

當您啟動匯入工作並收到回應時，如下所示：

```

An error occurred (ResourceInUseException) when calling the ImportTable operation:
Table already exists: target-table

```

若要修正此錯誤，您必須選擇不存在的資料表名稱，然後再一次匯入。

指定的儲存貯體不存在

如果來源值區不存在，匯入將會失敗，並將錯誤訊息詳細資訊記錄在中 CloudWatch。

描述匯入範例：

```

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)"
}
}
```

此 `FailureCode` 為 `S3NoSuchBucket`，其中 `FailureMessage` 包含請求 ID 等詳細資訊以及擲出錯誤服務。由於在將資料匯入資料表之前就已發現錯誤，因此不會建立新的 DynamoDB 資料表。部分情況下，若資料開始匯入後才遇到這些錯誤，就會保留含有部分匯入資料的資料表。

若要修正此錯誤，請確定來源 Amazon S3 儲存貯體存在，然後重新啟動匯入程序。

## 將 Amazon S3 匯入到 DynamoDB 的最佳實務

以下為將資料從 Amazon S3 匯入到 DynamoDB 的最佳實務。

保持在 50,000 個 S3 物件的限制以下

每個匯入任務最多支援 50,000 個 S3 物件。如果您的資料集包含超過 50,000 個物件，請考慮將它們合併為較大的物件。

避免過大的 S3 物件

S3 物件會平行匯入。擁有眾多中型 S3 物件允許平行執行，而不會產生過大負荷。對於 1 KB 以下的項目，請考慮在每個 S3 物件中放置 400 萬個項目。如果您的平均項目大小較大，請在每個 S3 物件中按比例放置較少的項目。

隨機化已排序的資料

如果 S3 物件以排序順序保存資料，它會建立滾動的經常性分割區。這種情況是由其中一個分割區接收所有活動，接著是下一個分割區，依此類推。按排序順序的資料的定義是 S3 物件中的循序項目，將在匯入期間寫入相同目標分割區。資料以排序順序排列的一個常見情況是 CSV 檔案，其中項目會依分割區索引鍵排序，以便讓重複的項目共用相同的分割區索引鍵。

若要避免滾動的經常性分割區，我們建議您在這些情況下隨機化順序。這可以透過散佈寫入操作來改善效能。如需詳細資訊，請參閱 [在資料上傳期間有效地分配寫入活動](#)。

壓縮資料，將 S3 物件總大小保持在區域限制以下

在從 [S3 匯入程序](#) 中，要匯入之 S3 物件資料的總大小有限制。us-east-1、us-west-2 和 eu-west-1 區域中，限制為 15 TB，在所有其他區域中為 1 TB。此限制是以原始 S3 物件大小為基礎。

壓縮可讓更多原始資料符合限制。如果單獨壓縮不足以讓匯入符合限制範圍，您也可以聯絡 [AWS Premium Support](#) 請求提高配額。

注意項目大小如何影響效能

如果您的平均項目大小非常小 (低於 200 位元組)，匯入程序可能會比較大的項目大小更久。

考慮在沒有全域次要索引的情形下匯入

匯入任務的持續時間可能取決於是否存在一或多個全域次要索引 (GSI)。如果您計劃使用低基數的分割區索引鍵來建立索引，那麼如果將索引建立延遲到匯入任務完成之後 (而不是包含在匯入工作中)，匯入速度可能會更快。

**Note**

在匯入期間建立 GSI 不會產生寫入費用 (在匯入後建立 GSI 則會)。

## DynamoDB 資料匯出至 Amazon S3：運作方式

DynamoDB 匯出至 S3 是一種全受管解決方案，可將 DynamoDB 資料大規模匯出至 Amazon S3 儲存貯體。使用 DynamoDB 匯出到 S3 時，您可以從[point-in-time 復原 \(PITR\)](#) 視窗內的任何時間將資料從 Amazon DynamoDB 表格匯出到 Amazon S3 儲存貯體。您需在資料表上啟用 PITR，才能使用匯出功能。此功能可讓您使用其他 AWS 服務 (例如 Athena、Amazon、AWS Glue Amazon SageMaker EMR 和) 對資料執行分析和 AWS Lake Formation 複雜查詢。

DynamoDB 匯出至 S3 可讓您從 DynamoDB 資料表匯出完整資料和增量資料。匯出不會使用任何[讀取容量單位 \(RCU\)](#)，也不會影響資料表效能和可用性。支援的匯出檔案格式為 DynamoDB JSON 和 Amazon Ion 格式。您也可以將資料匯出到另一個 AWS 帳戶擁有的 S3 儲存貯體和不同 AWS 區域。您的數據始終是加密的 end-to-end。

DynamoDB 完整匯出的費用是根據匯出完成之時間點的 DynamoDB 資料表大小 (資料表資料和本機次要索引) 來計費。DynamoDB 增量匯出的費用是根據匯出期間內從連續備份處理的資料大小來計費。其他費用包括將匯出的資料儲存在 Amazon S3 中，以及對 Amazon S3 儲存貯體發出的 PUT 請求。如需這些費用的詳細資訊，請參閱 [Amazon DynamoDB 定價](#) 和 [Amazon S3 定價](#)。

如需 Service Quotas 的詳細資訊，請參閱 [資料表匯出至 Amazon S3](#)。

### 主題

- [請求在 DynamoDB 中匯出資料表](#)
- [DynamoDB 資料表匯出輸出格式](#)

## 請求在 DynamoDB 中匯出資料表

DynamoDB 表格匯出可讓您將表格資料匯出到 Amazon S3 儲存貯體，讓您能夠使用其他 AWS 服務 (例如 Athena、Amazon、Amazon SageMaker、AWS Glue 亞馬遜 EMR 和) 對資料執行分析和複雜查詢。AWS Lake Formation 您可以使用 AWS Management Console、或 DynamoDB API 請求資料表匯出。AWS CLI

**Note**

不支援請求者支付 Amazon S3 儲存貯體費用。

DynamoDB 同時支援完整匯出和增量匯出：

- 透過完整匯出，您可以將表格的完整快照從 point-in-time 復原 (PITR) 時段內的任何時間點匯出到 Amazon S3 儲存貯體。
- 使用增量匯出時，您可以將 PITR 時段中指定期間內變更、更新或刪除的 DynamoDB 資料表中的資料匯出至 Amazon S3 儲存貯體。

### 主題

- [必要條件](#)
- [使用 AWS Management Console 請求匯出](#)
- [獲取有關過去出口的詳細信息 AWS Management Console](#)
- [使用 AWS CLI 請求匯出](#)
- [獲取有關過去出口的詳細信息 AWS CLI](#)
- [使用 AWS SDK 請求匯出](#)
- [使用 AWS SDK 獲取有關過去導出的詳細信息](#)

### 必要條件

#### 啟用 PITR

若要使用匯出至 S3 功能，您必須在表格上啟用 PITR。如需如何啟用 PITR 的詳細資訊，請參閱 [Point-in-time 復原](#)。如果您要求匯出未啟用 PITR 的資料表，您的要求將失敗，並顯示例外狀況訊息：「呼叫ExportTableToPointInTime作業時發生錯誤 (PointInTimeRecoveryUnavailableException)：表格 'my-dynamodb-table」未啟用時間點復原。

#### 設定 S3 許可

您可以將資料表資料匯出至您擁有寫入許可的任何 Amazon S3 儲存貯體。目的地值區不需要位於相同的 AWS 區域，也不需要與來源資料表擁有者擁有相同的擁有者。您的 AWS Identity and Access Management (IAM) 政策需要允許您執行 S3 動作 (s3:AbortMultipartUploads3:PutObject、



和 `s3:PutObjectAcl`) 和 DynamoDB 匯出動作 (`dynamodb:ExportTableToPointInTime`)。以下是範例政策範例，該政策將授與您的使用者執行匯出至 S3 儲存貯體的權限。

```
{
 "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/*"
 }
]
}
```

如果您需要寫入另一個帳戶中的 S3 儲存貯體，或者您沒有寫入權限，則 S3 儲存貯體擁有者必須新增儲存貯體政策，以允許您從 DynamoDB 匯出到該儲存貯體。以下是目標 S3 儲存貯體的範例政策。

```
{
 "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/*"
 }
]
}
```

```
 }
]
}
```

在匯出過程中撤銷這些許可會產生部分檔案。

#### Note

如果您要匯出的目標資料表或儲存貯體使用客戶受管金鑰加密，該 KMS 金鑰的政策必須允許 DynamoDB 使用該金鑰。此許可是透過觸發匯出任務的 IAM 使用者/角色授予。如需有關加密及其最佳實務的詳細資訊，請參閱 [DynamoDB 如何使用 AWS KMS](#) 以及 [使用自訂 KMS 金鑰](#)。

使用 AWS Management Console 請求匯出

以下範例示範如何使用 DynamoDB 主控台匯出名為 MusicCollection 的現有資料表。

#### Note

此程序假設您已啟用 point-in-time 復原。若要為 MusicCollection 表格啟用此功能，請在表格的「總覽」頁籤的「表格詳細資訊」段落中，選擇啟用 P oint-in-time 復原功能。

請求資料表匯出

1. [登入 AWS Management Console 並開啟 DynamoDB 支援主控台](https://console.aws.amazon.com/dynamodb/)，網址為 <https://console.aws.amazon.com/dynamodb/>。
2. 在主控台左側的導覽窗格中，選擇 Exports to S3 (匯出至 S3)。
3. 選取匯出至 S3 按鈕。
4. 選擇來源資料表和目的地 S3 儲存貯體。如果您的帳戶擁有該目的地儲存貯體，您可以使用 Browse S3 (瀏覽 S3) 按鈕尋找它。如果不是，請使用 `s3://bucketname/prefix` format。輸入儲存貯體的 URL；**prefix** 是有助於您整理目的地儲存貯體的選用資料夾。
5. 選擇完整匯出或增量匯出。完整匯出會輸出資料表在您所指定時間點的完整資料表快照。增量匯出會輸出在指定匯出期間對資料表所做的變更。您的輸出已壓縮，因此它僅包含匯出期間內項目的最終狀態。即使項目在同一匯出期間內有多個更新，該項目也只會出現在匯出中出現一次。

## Full export

1. 選取您要匯出完整資料表快照的來源時間點。這個時間點可以是 PITR 時段內的任何時間點。或者，您可以選取目前時間以匯出最新快照。

### 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.



For date, use YYYY/MM/DD format. For time, use 24-hour format.

2. 對於匯出的檔案格式，請選擇 DynamoDB JSON 和 Amazon Ion 兩者之一。根據預設，您的資料表會以 DynamoDB JSON 格式在時間點復原時段中的最近可還原時間匯出，並使用 Amazon S3 金鑰 (SSE-S3) 進行加密。您可以視需要變更這些匯出設定。

**Note**

如果您選擇使用受 AWS Key Management Service (AWS KMS) 保護的金鑰加密匯出，金鑰必須與目的地 S3 儲存貯體位於相同的區域。

### Exported file format [Info](#)

**DynamoDB JSON****Amazon Ion**

Open-source text format, which is a superset of JSON.

## Incremental export

1. 選取您要匯出增量資料的匯出期間。選擇在 PITR 時段內的開始時間。匯出期間必須至少為 15 分鐘，且不超過 24 小時。匯出期間的開始時間包含在內，結束時間則不包含在內。

### 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. 選擇絕對模式或相對模式。
  - a. 絕對模式會依您指定的期間匯出增量資料。

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. 相對模式會在相對於匯出任務提交時間的匯出期間匯出增量資料。

**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**

- 對於匯出的檔案格式，請選擇 DynamoDB JSON 和 Amazon Ion 兩者之一。根據預設，您的資料表會以 DynamoDB JSON 格式在時間點復原時段中的最近可還原時間匯出，並使用 Amazon S3 金鑰 (SSE-S3) 進行加密。您可以視需要變更這些匯出設定。

**i** Note

如果您選擇使用受 AWS Key Management Service (AWS KMS) 保護的金鑰加密匯出，金鑰必須與目的地 S3 儲存貯體位於相同的區域。

Exported file format | **Info**

**DynamoDB JSON**

Amazon Ion

Open-source text format, which is a superset of JSON.

- 對於匯出檢視類型，請選取新舊映像或僅限新映像。新映像提供項目的最新狀態。舊映像提供項目在指定的「開始日期和時間」之前的狀態。預設設定為新舊映像。如需新映像和舊映像的詳細資訊，請參閱 [增量匯出輸出](#)。

## Export view type

- New and old images
- New images only

### 6. 選擇「匯出」開始。

匯出的資料在交易上不一致。您的交易操作可以在兩個導出輸出之間切斷。匯出中反映的交易作業可以修改項子集，而同一筆交易中的另一個修改子集則不會反映在相同的匯出要求中。不過，匯出最後會是一致的。如果交易在匯出期間遭到撕裂，您將會在下一個連續匯出中保留剩餘的交易，而不會有重複項目。用於匯出的期間是以內部系統時鐘為基礎，且可能與您的應用程式本機時鐘相差一分鐘。

### 獲取有關過去出口的詳細信息 AWS Management Console

您可以選擇導覽側邊欄中的 [匯出至 S3] 區段，找到過去執行的匯出工作的相關資訊。此區段包含您在過去 90 天內建立的所有匯出清單。選取匯出索引標籤中所列工作的 ARN，以擷取有關該匯出的資訊，包括您選擇的任何進階組態設定。請注意，雖然匯出任務中繼資料會在 90 天後過期，也無法在此清單中找到早於該時間的任務，但只要儲存貯體政策允許，S3 儲存貯體中的物件就會保留。DynamoDB 絕不會刪除在匯出時其在您的 S3 儲存貯體中建立的任何物件。

### 使用 AWS CLI 請求匯出

下列範例示範如何使用將名 AWS CLI 為的現有資料表匯出 MusicCollection 至名為的 S3 儲存貯體 ddb-export-musiccollection。

#### Note

此程序假設您已啟用 point-in-time 復原。若要針對 MusicCollection 資料表啟用程序，請執行下列命令。

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

## Full export

以下命令會將 MusicCollection 匯出至名為 ddb-export-musiccollection-9012345678 的 S3 儲存貯體 (字首為 2020-Nov)。資料表資料會以 DynamoDB JSON 格式在時間點復原時段中的特定時間匯出，並使用 Amazon S3 金鑰 (SSE-S3) 進行加密。

### Note

如果要求跨帳戶資料表匯出，請務必包含 `--s3-bucket-owner` 選項。

```
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

下列命令會藉由提供新的 `--export-type` 和 `--incremental-export-specification` 來執行增量匯出。將任何斜體內容替換為您自己的值。時間是以自 epoch 起算的秒數來指定。

```
aws dynamodb export-table-to-point-in-time \
 --table-arn arn:aws:dynamodb:REGION:ACCOUNT:table/TABLENAME \
 --s3-bucket BUCKET --s3-prefix PREFIX \
 --incremental-export-specification
 ExportFromTime=1693569600,ExportToTime=1693656000,ExportViewType=NEW_AND_OLD_IMAGES
 \
 --export-type INCREMENTAL_EXPORT
```

### Note

如果您選擇使用受 AWS Key Management Service (AWS KMS) 保護的金鑰加密匯出，金鑰必須與目的地 S3 儲存貯體位於相同的區域。



## 獲取有關過去出口的詳細信息 AWS CLI

使用 `list-exports` 命令即可找到您過去曾執行之匯出請求的相關資訊。此命令會傳回您在過去 90 天內建立的所有匯出清單。請注意，雖然匯出任務中繼資料會在 90 天後過期，也無法使用 `list-exports` 命令傳回早於該時間的任務，但只要儲存貯體政策允許，S3 儲存貯體中的物件就會保留。DynamoDB 絕不會刪除在匯出時其在您的 S3 儲存貯體中建立的任何物件。

在匯出成功或失敗之前，其狀態會是 PENDING。如果成功，狀態會變更為 COMPLETED。如果失敗，狀態會變更為 FAILED `failure_message` 與 `failure_reason`。

在以下範例中，我們使用選用 `table-arn` 參數，只列出特定資料表的匯出資料。

```
aws dynamodb list-exports \
 --table-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog
```

若要擷取特定匯出任務的詳細資訊 (包括任何進階組態設定)，請使用 `describe-export` 命令。

```
aws dynamodb describe-export \
 --export-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4
```

## 使用 AWS SDK 請求匯出

您可以使用這些程式碼片段，使用您選擇的 AWS SDK 要求匯出資料表。

### Python

#### 完整匯出

```
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'
)
```

## 增量匯出

```
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'
)
```

## 使用 AWS SDK 獲取有關過去導出的詳細信息

您可以使用這些程式碼片段，取得有關使用您選擇的 AWS SDK 過去表格匯出的詳細資訊。

### Python

#### 完整匯出

```
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',
)
)
```

## 增量匯出

```
import boto3

client = boto3.client('dynamodb')

https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/describe_export.html

print(
 client.describe_export(
 ExportArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE/
export/01695353076000-06e2188f',
)['ExportDescription']
)
```

## DynamoDB 資料表匯出輸出格式

DynamoDB 資料表匯出內容除了包含資料表資料的檔案之外，還包括資訊清單檔案。這些檔案全都儲存於您在[匯出請求](#)中指定的 Amazon S3 儲存貯體。以下各節說明每個輸出物件的格式和內容。

### 完整匯出輸出

### 資訊清單檔案

DynamoDB 會針對每次匯出請求，在指定的 S3 儲存貯體中建立資訊清單檔案及其檢查總和檔案。

```
export-prefix/AWS DynamoDB/ExportId/manifest-summary.json
export-prefix/AWS DynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWS DynamoDB/ExportId/manifest-files.json
export-prefix/AWS DynamoDB/ExportId/manifest-files.checksum
```

在請求資料表匯出時，您可以選擇 **export-prefix**。其有助於您在目的地 S3 儲存貯體中整理檔案。**ExportId** 是由服務產生的唯一權杖，可確保匯出至相同 S3 儲存貯體和 **export-prefix** 的多筆資料不會相互覆寫。

此匯出功能會為每個分割區建立至少 1 個檔案。對於空白的分割區，您的匯出請求將會建立空白檔案。每個檔案的所有項目都來自該特定分割區的雜湊金鑰空間。

### Note

DynamoDB 也會在資訊清單檔案所在的相同目錄中建立名為 `_started` 的空白檔案。此檔案驗證目的地儲存貯體可寫入並已開始進行匯出。您可以安全地刪除此檔案。

## 摘要資訊清單

`manifest-summary.json` 檔案包含匯出任務的摘要資訊。這可讓您知道共用資料夾中的哪些資料檔案與此匯出相關聯。其格式如下：

```
{
 "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"
}
```

## 檔案資訊清單

`manifest-files.json` 檔案包含內含匯出資料表資料的檔案資訊。檔案採用 [JSON Lines](#) 格式，因此新行將作為項目分隔符號使用。在下列範例中，為了便於閱讀，檔案資訊清單的資料檔案詳細資訊會採用多行格式。

```
{
```

```

"itemCount": 8,
 "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
 "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
 "dataFileS3Key": "AWS DynamoDB/01693685827463-2d8752fd/data/asdl123dasas.json.gz"
}

```

## 資料檔案

DynamoDB 會以兩種格式匯出資料表資料：DynamoDB JSON 和 Amazon Ion。無論選擇哪種格式，您的資料都會寫入多個以金鑰命名的壓縮檔案。這些檔案也會在 `manifest-files.json` 檔案中列出。

完整匯出後，S3 儲存貯體的目錄結構將包含匯出 ID 資料夾下的所有資訊清單檔案和資料檔案。

```

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

以 DynamoDB JSON 格式匯出的資料表由多個 Item 物件組成。每個個別物件均採用 DynamoDB 的標準封送處理 JSON 格式。

建立 DynamoDB JSON 匯出資料的自訂剖析器時，格式為 [JSON Lines](#)。這表示將新行用作項目分隔符號。許多 AWS 服務 (例如 Athena 和 AWS Glue) 都會自動剖析此格式。

在下列範例中，為了便於閱讀，DynamoDB JSON 匯出資料的單一項目採用多行格式。

```

{
 "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](#) 是輸入豐富、自行描述、階層資料序列化格式，旨在處理快速開發、解耦及日常遇到的效率挑戰等問題，同時設計以服務為導向的大規模架構。DynamoDB 支援匯出 Ion [文字格式](#) 的資料表資料，亦即 JSON 的超集。

將資料表匯出成 Ion 格式時，資料表中使用的 DynamoDB 資料類型會映射至 [Ion 資料類型](#)。DynamoDB 設定使用 [Ion 類型註釋](#)，以釐清在來源資料表中使用的資料類型。

## DynamoDB 至 Ion 資料類型轉換

DynamoDB 資料類型	Ion 表示法
String (S)	string

DynamoDB 資料類型	Ion 表示法
Boolean (BOOL)	bool
Number (N)	decimal
Binary (B)	blob
Set (SS、NS、BS)	list (包含類型註解 \$dynamodb_SS、\$dynamodb_NS 或 \$dynamodb_BS)
清單	list
Map	struct

Ion 匯出中的項目以新行分隔。每行都以 Ion 版本標記開頭，後面接著 Ion 格式的項目。在下列範例中，為了便於閱讀，Ion 匯出的項目採用多行格式。

```
$ion_1_0 {
 Item:{
 Authors:$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"
 }
}
```

## 增量匯出輸出

## 資訊清單檔案

DynamoDB 會針對每次匯出請求，在指定的 S3 儲存貯體中建立資訊清單檔案及其檢查總和檔案。

```
export-prefix/AWS DynamoDB/ExportId/manifest-summary.json
export-prefix/AWS DynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWS DynamoDB/ExportId/manifest-files.json
```

```
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

在請求資料表匯出時，您可以選擇 **export-prefix**。其有助於您在目的地 S3 儲存貯體中整理檔案。**ExportId** 是由服務產生的唯一權杖，可確保匯出至相同 S3 儲存貯體和 **export-prefix** 的多筆資料不會相互覆寫。

此匯出功能會為每個分割區建立至少 1 個檔案。對於空白的分割區，您的匯出請求將會建立空白檔案。每個檔案的所有項目都來自該特定分割區的雜湊金鑰空間。

### Note

DynamoDB 也會在資訊清單檔案所在的相同目錄中建立名為 `_started` 的空白檔案。此檔案驗證目的地儲存貯體可寫入並已開始進行匯出。您可以安全地刪除此檔案。

### 摘要資訊清單

`manifest-summary.json` 檔案包含匯出任務的摘要資訊。這可讓您知道共用資料夾中的哪些資料檔案與此匯出相關聯。其格式如下：

```
{
 "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"
```



```
}
```

## 檔案資訊清單

`manifest-files.json` 檔案包含內含匯出資料表資料的檔案資訊。檔案採用 [JSON Lines](#) 格式，因此新行將作為項目分隔符號使用。在下列範例中，為了便於閱讀，檔案資訊清單的資料檔案詳細資訊會採用多行格式。

```
{
 "itemCount": 8,
 "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
 "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
 "dataFileS3Key": "AWS DynamoDB/data/sgad6417s6vss4p7owp0471bcq.json.gz"
}
```

## 資料檔案

DynamoDB 會以兩種格式匯出資料表資料：DynamoDB JSON 和 Amazon Ion。無論選擇哪種格式，您的資料都會寫入多個以金鑰命名的壓縮檔案。這些檔案也會在 `manifest-files.json` 檔案中列出。

增量匯出的資料檔案全部包含在 S3 儲存貯體的常用資料資料夾中。您的資訊清單檔案會在匯出 ID 資料夾下。

```
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
```

# ...

在匯出檔案中，每個項目的輸出都包含時間戳記，代表該項目在您的資料表中更新的時間，還有指出該項目是 insert、update 或 delete 操作的資料結構。時間戳記是以內部系統時鐘為基礎，且可能因應用程式時脈而有所不同。對於增量匯出，您可以為輸出結構選擇兩種匯出檢視類型：新舊映像或僅限新映像。

- 新映像提供項目的最新狀態
- 舊映像提供項目在指定的開始日期和時間之前的狀態

如果您想要查看項目在匯出期間內的變更情形，檢視類型會很有幫助。它對於有效率地更新下游系統來說也很實用，特別是在這些下游系統的分割區索引鍵與 DynamoDB 分割區索引鍵不同的情況下。

您可以藉由查看輸出的結構，推斷增量匯出輸出中的項目是 insert、update 或 delete。下表針對兩種匯出檢視類型摘要說明增量匯出結構及其對應操作。

作業	僅限新映像	新舊映像
Insert	按鍵 + 新映像	按鍵 + 新映像
更新	按鍵 + 新映像	鍵 + 新映像 + 舊映像
Delete	鍵	按鍵 + 舊映像
Insert + delete	無輸出	無輸出

## DynamoDB JSON

DynamoDB JSON 格式的資料表匯出包含中繼資料時間戳記，該時間戳記指出項目的寫入時間，後面接著項目的索引鍵和值。以下顯示使用新舊映像作為匯出檢視類型的 DynamoDB JSON 輸出範例。

```
// 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](#) 是輸入豐富、自行描述、階層資料序列化格式，旨在處理快速開發、解耦及日常遇到的效率挑戰等問題，同時設計以服務為導向的大規模架構。DynamoDB 支援匯出 Ion [文字格式](#) 的資料表資料，亦即 JSON 的超集。

將資料表匯出成 Ion 格式時，資料表中使用的 DynamoDB 資料類型會映射至 [Ion 資料類型](#)。DynamoDB 設定使用 [Ion 類型註釋](#)，以釐清在來源資料表中使用的資料類型。

### DynamoDB 至 Ion 資料類型轉換

DynamoDB 資料類型	Ion 表示法
String (S)	string
Boolean (BOOL)	bool
Number (N)	decimal
Binary (B)	blob
Set (SS、NS、BS)	list (包含類型註解 \$dynamodb_SS、\$dynamodb_NS 或 \$dynamodb_BS)
清單	list
Map	struct

Ion 匯出中的項目以新行分隔。每行都以 Ion 版本標記開頭，後面接著 Ion 格式的項目。在下列範例中，為了便於閱讀，Ion 匯出的項目採用多行格式。

```
$ion_1_0 {
 Record:{
 Keys:{
 ISBN:"333-3333333333"
 },
 Metadata:{
 WriteTimestampMicros:1684374845117899.
 },
 OldImage:{
 Authors:$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"
 }
}
}
```

## DynamoDB 零 ETL 與 Amazon 服務整合 OpenSearch

Amazon DynamoDB 透過 DynamoDB 外掛程式提供與 Amazon OpenSearch 服務的零 ETL 整合，以進行擷取。OpenSearch Amazon OpenSearch 擷取提供全受管、無程式碼的體驗，可將資料擷取至 Amazon 服務。OpenSearch

使用用於 OpenSearch 擷取的 DynamoDB 外掛程式，您可以使用一或多個 DynamoDB 表作為擷取一或多個服務索引的來源。OpenSearch 您可以瀏覽和設定 OpenSearch 擷取管道，將 DynamoDB 做為來源，從中的 OpenSearch 擷取或 DynamoDB 整合。AWS Management Console

- 按照 OpenSearch 擷取入門指南中的說明[開始使用 OpenSearch 擷取](#)。
- [在適用於擷取的 DynamoDB 外掛程式文件中了解 DynamoDB 外掛程式的先決條件和所有設定選項。OpenSearch](#)

### 運作方式

此外掛程式使用 [DynamoDB 匯出至 Amazon S3](#) 來建立要載入的初始快照。OpenSearch 載入快照之後，外掛程式會使用 DynamoDB Streams 以近乎即時的方式複製任何進一步的變更。在 OpenSearch 擷取中，每個項目都會作為事件處理，並且可以使用處理器外掛程式進行修改。您可以刪除屬性或創建複合屬性，並通過路由將其發送到不同的索引。

您必須啟用 [point-in-time 復原 \(PITR\)](#) 才能使用匯出到 Amazon S3。您也必須啟用 [DynamoDB Streams](#) (在選取 [新映像和舊映像] 選項的情況下)，才能使用它。排除匯出設定，可以在不建立快照的情況下建立管線。

您也可以透過排除串流設定來建立僅包含快照且無更新的管道。外掛程式不會在您的資料表上使用讀取或寫入輸送量，因此可以安全使用，而不會影響您的生產流量。在建立此整合或其他整合之前，您應該考慮串流上的 parallel 消費者數量有限制。如需其他考量，請參閱 [the section called “整合最佳做法”](#)。

對於簡單的管道，單一 OpenSearch 運算單元 (OCU) 每秒可處理約 1 MB 的寫入。這相當於大約 1000 個寫入請求單位 (WCU)。根據管道的複雜性和其他因素，您可能實現的效果或多或少。

OpenSearch 擷取支援導致無法復原錯誤的事件無法復原的無效字母佇列 (DLQ)。此外，即使 DynamoDB、管道或 Amazon OpenSearch 服務發生服務中斷，管道也可以從中斷的位置繼續執行，而無需使用者介入。

如果中斷持續超過 24 小時，這可能會導致更新遺失。不過，管道會繼續處理還原可用性時仍然可用的更新。除非它們在無效字母佇列中，否則您需要執行新的索引構建來修復由於丟棄事件而導致的任何不規則行為。

如需外掛程式的所有設定和詳細資訊，請參閱 [OpenSearch 擷取 DynamoDB](#) 外掛程式文件。

## 透過主控台整合建立體驗

DynamoDB 和 OpenSearch 服務在中具有整合式體驗 AWS Management Console，可簡化入門程序。當您執行這些步驟時，服務會自動選取 DynamoDB 藍圖並為您新增適當的 DynamoDB 資訊。

若要建立整合，請依照 [OpenSearch 擷取入門指南中的說明](#) 進行操作。當您進入「[步驟 3：建立管線](#)」時，請使用下列步驟取代步驟 1 和 2：

1. 導覽至動 DynamoDB 主控台。
2. 在左側導覽窗格中，選擇 [整合]。
3. 選取您要複寫的目標 DynamoDB 資料表。OpenSearch
4. 選擇建立。

從這裡，您可以繼續學習本自學課程的其餘部分。

## 後續步驟

若要更好地瞭解 DynamoDB 如何與 OpenSearch 服務整合，請參閱下列內容：

- [開始使用 Amazon OpenSearch 攝入](#)
- [外掛程式設定和需求](#)

## 處理索引的重大變更

OpenSearch 可以動態地向索引添加新屬性。但是，在為給定鍵設置映射模板後，您需要採取其他操作來對其進行更改。此外，如果您的變更需要重新處理 DynamoDB 表中的所有資料，則需要採取步驟啟動全新匯出。

### Note

在所有這些選項中，如果 DynamoDB 表與您指定的對應範本發生類型衝突，您仍可能會遇到問題。確保您已啟用無效字母隊列 (DLQ) (即使在開發中)。這使得更容易理解在索引中索引時導致衝突的記錄可能會出現什麼問題 OpenSearch。

### 主題

- [運作方式](#)
- [刪除索引並重設管線 \(以管線為中心的選項\)](#)
- [重新建立索引並重設管線 \(以索引為中心的選項\)](#)
- [創建一個新的索引和接收器 \(在線選項\)](#)
- [避免和調試類型衝突的最佳實踐](#)

## 運作方式

以下是處理索引的重大變更時所採取的動作的快速概觀。請參 step-by-step 閱以下各節中的程序。

- **停止並啟動管線**：此選項會重設管線的狀態，管線將以新的完整匯出重新啟動。它是非破壞性的，因此不會刪除您的索引或 DynamoDB 中的任何資料。如果您在執行此操作之前並未建立新索引，您可能會看到版本衝突造成的大量錯誤，因為匯出會嘗試插入比索引\_version中目前文件舊的文件。您可以放心地忽略這些錯誤。管道停止時，不會向您收取任何費用。
- **更新管線**：此選項會以 [藍/綠](#) 方式更新管線中的組態，而不會遺失任何狀態。如果您對管線進行重大變更 (例如將新的路由、索引或索引鍵新增至現有索引)，您可能需要對管線進行完整重設，然後重新建立索引。此選項不會執行完整匯出。
- **刪除並重新建立索引**：此選項會移除索引上的資料和對應設定。您應該先執行此操作，然後再對映進行任何重大變更。它會中斷任何依賴索引的應用程式，直到重新建立並同步處理索引為止。刪除索引



並不會啟動全新匯出。您應該只在更新管道後刪除索引。否則，您的索引可能會在您更新設定之前重新建立。

## 刪除索引並重設管線 (以管線為中心的選項)

如果您仍在開發中，此方法通常是最快的選擇。您將在 OpenSearch Service 中刪除索引，然後[停止並啟動](#)管道，以啟動所有資料的全新匯出。這可確保對應範本與現有索引不會發生衝突，也不會遺失未完整處理表格中的資料。

1. 透過AWS Management Console，或搭配或 SDK 使用 StopPipelineAPI 作業來停AWS CLI止管線。
2. 使用[新的變更更新管線組態](#)。
3. 透過 REST API 呼叫或 OpenSearch 儀表板刪除 OpenSearch 服務中的索引。
4. 透過主控台或搭配或 SDK 使用 StartPipeline API 作業來啟動管線。AWS CLI

### Note

這會啟動全新的完整匯出，因此會產生額外費用。

5. 監視是否有任何未預期的問題，因為會產生全新的匯出來建立新索引。
6. 確認該指數符合您在 OpenSearch 服務中的期望。

匯出完成並繼續從串流讀取之後，您的 DynamoDB 表資料現在就可以在索引中使用。

## 重新建立索引並重設管線 (以索引為中心的選項)

如果您需要在從 DynamoDB 恢復管線之前對 OpenSearch Service 中的索引設計進行大量迭代，則此方法很有效。當您想要在搜索模式上快速迭代並希望避免在每次迭代之間等待新的導出完成時，這對於開發非常有用。

1. 透過AWS Management Console、或使用或 SDK 呼叫 StopPipelineAPI 作業來停AWS CLI止管線。
2. 使用您要使用的 OpenSearch 對應範本刪除並重新建立索引。您可以手動插入一些範例資料，以確認您的搜尋是否如預期般運作。如果您的範例資料可能與 DynamoDB 中的任何資料衝突，請務必先將其刪除，然後再進行下一個步驟。
3. 如果您的管道中有索引範本，請將其移除，或將其取代為您已在 OpenSearch Service 中建立的索引範本。請確定索引的名稱與管線中的名稱相符。

#### 4. 透過主控台或使用或 SDK 呼叫 StartPipeline API 作業來啟動管線。AWS CLI

##### Note

這將啟動全新的完整導出，這將產生額外費用。

#### 5. 監視是否有任何未預期的問題，因為會產生全新的匯出來建立新索引。

匯出完成並繼續從串流讀取之後，您應該是 DynamoDB 表格資料現在可以在索引中使用。

### 創建一個新的索引和接收器（在線選項）

如果您需要更新映射模板，但當前正在生產中使用索引，則此方法可以很好地工作。這將創建一個全新的索引，您需要在同步和驗證後將其移動到應用程序。

##### Note

這將在流上創建另一個消費者。如果您還有其他消費者（如AWS Lambda或全局表），則可能是一個問題。您可能需要暫停對現有管道的更新，以建立載入新索引的容量。

1. [使用新設定和不同索引名稱建立新管線。](#)
2. 監視新索引是否有任何未預期的問題。
3. 將應用程式交換到新索引。
4. 驗證一切正常後，停止並刪除舊管道。

### 避免和調試類型衝突的最佳實踐

- 始終使用無效字母隊列（DLQ），以便在發生類型衝突時更容易進行調試。
- 始終使用具有映射和設置的索引模板include\_keys。雖然 OpenSearch Service 會動態映射新的鍵，但這可能會導致意外行為的問題（例如GeoPoint，期望某些東西是 a，但它被創建為string或object）或錯誤（例如number，具有long與float值的混合）。
- 如果您需要保持現有索引在生產環境中正常工作，您也可以通過在管道配置文件中重命名[索引來替換以前的任何刪除索引步驟](#)。這將創建一個全新的索引。然後，您的應用程序將需要更新以指向新索引後，它完成。
- 如果您有使用處理器修復的類型轉換問題，則可以使用來測試此問題UpdatePipeline。若要這麼做，您需要執行停止並啟動或[處理無效字母佇列](#)，以修正任何先前略過的有錯誤的文件。

## 與 DynamoDB 整合的最佳做法

將 DynamoDB 與其他服務整合時，您應始終遵循使用每個個別服務的最佳做法。不過，您應該考慮一些特定於整合的最佳作法。

### 主題

- [在 DynamoDB 資料庫中建立快照](#)
- [在 DynamoDB 中擷取資料變更](#)
- [DynamoDB 的零 ETL 整合 OpenSearch](#)

### 在 DynamoDB 資料庫中建立快照

- 一般而言，我們建議使用[匯出至 Amazon S3](#) 來建立初始複寫的快照。它既符合成本效益，也不會與應用程式的流量競爭以取得輸送量。您還可以考慮備份並還原到新表，然後進行掃描操作。這樣可以避免與應用程式競爭輸送量，但通常比匯出成本效益低。
- 執行匯出StartTime時，請務必設定一個。這可讓您輕鬆判斷從何處開始變更資料擷取 (CDC)。
- 使用匯出至 S3 時，請在 S3 儲存貯體上設定生命週期動作。通常，設定為 7 天的到期動作是安全的，但您應該遵循公司可能具有的任何準則。即使您在擷取後明確刪除項目，此動作也有助於找出問 catch，這有助於減少不必要的成本並防止違反原則。

### 在 DynamoDB 中擷取資料變更

- 如果您需要接近即時的 CDC，請使用 [DynamoDB Streams](#) 或 [Amazon Kinesis Data Streams \(KDS\)](#)。當您決定要使用哪一個時，通常會考慮哪一個最容易與下游服務搭配使用。如果您需要在分區索引鍵層級提供順序事件處理，或者您的項目非常大，請使用 DynamoDB Streams。
- 如果您不需要接近即時的 CDC，您可以使用[匯出到 Amazon S3 進行增量匯出](#)，僅匯出兩個時間點之間發生的變更。

如果您使用匯出至 S3 來產生快照，這會特別有用，因為您可以使用類似的程式碼來處理增量匯出。一般而言，匯出至 S3 比先前的串流選項便宜一些，但成本通常不是使用哪個選項的主要因素。

- 一般而言，您只能同時擁有兩個 DynamoDB 串流的取用者。在規劃整合策略時，請考慮這一點。
- 請勿使用掃描來偵測變更。這可能會在小規模上工作，但很快就變得不切實際。

## DynamoDB 的零 ETL 整合 OpenSearch

DynamoDB 具有與 Amazon 服務的[零 ETL](#) 整合。OpenSearch 如需詳細資訊，請參閱[用於 OpenSearch 擷取的 DynamoDB 外掛程式](#)和 Amazon 服務的[特定最佳實務](#)。OpenSearch

### 組態

- 僅索引您需要執行搜尋的資料。始終使用映射模板 ( `template_type: index_template`和`template_content` ) 並`include_keys`實現此操作。
- 監視記錄檔中是否有與類型衝突相關的錯誤。OpenSearch Service 會預期給定索引鍵的所有值都具有相同的類型。如果存在不匹配，它會生成異常。如果遇到這些錯誤之一，您可以添加一個處理器來 `catch` 給定的密鑰始終是相同的值。
- 通常使用中`primary_key`繼資料值做為`document_id`值。在 OpenSearch 服務中，文件 ID 相當於 DynamoDB 中的主索引鍵。使用主索引鍵可讓您輕鬆找到文件，並確保更新會一致地複製到文件中，而不會發生衝突。

您可以使用輔助函數`getMetadata`來獲取主鍵 ( 例如，`document_id: "${getMetadata('primary_key')}`" )。如果您使用的是複合主鍵，輔助函數將為您連接在一起。

- 一般而言，請使用`action`設定的中`opensearch_action`繼資料值。這將確保以 OpenSearch 服務中的資料符合 DynamoDB 中的最新狀態的方式複寫更新。

您可以使用輔助函數`getMetadata`來獲取主鍵 ( 例如，`action: "${getMetadata('opensearch_action')}`" )。您還可以通過過濾等用`dynamodb_event_name`例獲取流事件類型。但是，通常不應將其用於`action`設置。

### 可觀測性

- 請務必在 OpenSearch 接收器上使用無效字母佇列 (DLQ) 來處理捨棄的事件。DynamoDB 的結構通常低於 OpenSearch 服務，並且始終可能發生意想不到的事情。使用無效字母佇列，您可以復原個別事件，甚至自動執行復原程序。這將幫助您避免需要重建整個索引。
- 一律設定警示，告知您的複寫延遲未超過預期金額。假設一分鐘沒有警報過於嘈雜，通常是安全的。這可能會有所不同，具體取決於您的寫入流量有多高，以及管道上的 OpenSearch 計算單元 (OCU) 設定。

如果您的複寫延遲超過 24 小時，串流將開始丟棄事件，而且除非您從頭開始完全重建索引，否則您將會遇到準確性問題。

## 擴展

- 對管道使用 auto 擴展來協助擴展或縮減 OCU，以最適合工作負載。
- 對於沒有 auto 擴展的佈建輸送量表格，我們建議您根據寫入容量單位 (WCU) 除以 1000 來設定 OCU。將最小值設定為低於該金額的 1 OCU (但至少 1)，並將最大值設定為高於該金額至少 1 個 OCU。
- 公式：

```
OCU_minimum = GREATEST((table_WCU / 1000) - 1, 1)
OCU_maximum = (table_WCU / 1000) + 1
```


- 範例：您的表格已佈建 25000 個 WCU。您管道的作業單元應設定為至少 24 個 (25000/1000-1)，最大值至少應設定 26 個 (25000/1000 + 1)。
- 對於具有 auto 調整規模的佈建輸送量表格，建議您根據最小和最大 WCU (除以 1000) 來設定 OCU。從 DynamoDB 將最小值設定為低於最小值的 1 個 OCU，並將最大值設定為至少高於 DynamoDB 上限的 1 個 OCU。
- 公式：

```
OCU_minimum = GREATEST((table_minimum_WCU / 1000) - 1, 1)
OCU_maximum = (table_maximum_WCU / 1000) + 1
```

- 範例：您的資料表具有至少 8000 個且最大值為 14000 的 auto 擴展政策。您的管道的作業單元應設定為最少 7 個 (8000/1000-1)，最多設定為 15 個 (14000/1000 + 1)。
- 對於隨需輸送量表格，我們建議根據您的典型尖峰和谷值設定 OCU，以便每秒寫入請求單位。您可能需要在較長的時間段內進行平均值，具體取決於您可用的彙總。從 DynamoDB 將最小值設定為低於最小值的 1 個 OCU，並將最大值設定為至少高於 DynamoDB 上限的 1 個 OCU。
- 公式：

```
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
```

- 範例：您的資料表的平均谷數為每秒 300 個寫入請求單位，平均峰值為 4300。您管道的作業單元應設定至少為 1 (300/1000-1，但至少 1 個)，最多可設定 5 個 (4300/1000 + 1)。
- 遵循擴展目標 OpenSearch 服務索引的最佳做法。如果您的索引縮放不足，它會減慢 DynamoDB 的擷取速度，並可能導致延遲。

 Note

GREATEST 是一個 SQL 函數，給定一組參數，返回具有最大值的參數。

# Amazon DynamoDB 中的服務、帳戶和資料表配額

本節說明 Amazon DynamoDB 中的當前配額 (過去稱為限制)。各項配額除非另有說明，否則都是區域特定規定。

## 主題

- [讀取/寫入容量模式與輸送量](#)
- [預留容量](#)
- [匯入配額](#)
- [Contributor Insights](#)
- [資料表](#)
- [全域資料表](#)
- [次要索引](#)
- [分割區索引鍵和排序索引鍵](#)
- [命名規則](#)
- [資料類型](#)
- [項目](#)
- [Attributes](#)
- [表達式參數](#)
- [DynamoDB 交易](#)
- [DynamoDB Streams](#)
- [DynamoDB Accelerator \(DAX\)](#)
- [API 特定限制](#)
- [DynamoDB 靜態加密](#)
- [資料表匯出至 Amazon S3](#)
- [備份和還原](#)

## 讀取/寫入容量模式與輸送量

您可以隨時將表格從隨選模式切換為佈建的容量模式。當您在容量模式之間進行多次切換時，適用下列條件：

- 您可以隨時在隨選模式下將新建立的表格切換為佈建的容量模式。不過，您只能在資料表建立時間戳記後 24 小時切換回隨選模式。
- 您可以隨時以隨需模式將現有表格切換為已佈建容量模式。但是，您只能在指示切換到隨選的最後一個時間戳記後 24 小時切換回隨選模式。

如需在讀取和寫入容量模式之間切換的詳細資訊，請參閱[切換容量模式時的考量](#)。

## 容量單位大小 (已佈建的資料表)

一個讀取容量單位 = 每秒一個高度一致性讀取，或每秒兩個最終一致讀取，適用於大小上限為 4 KB 的項目。

一個寫入容量單位 = 每秒一個寫入，適用於大小上限為 1 KB 的項目。

交易讀取請求需要 2 個讀取容量單位才能執行每秒讀取一個大小上限為 4 KB 的項目。

交易讀取請求需要兩個寫入容量單位才能執行每秒寫入一個大小上限為 1 KB 的項目。

## 請求單位大小 (隨需資料表)

一個讀取請求單位 = 每秒一個高度一致性讀取，或每秒兩個最終一致讀取，適用於大小上限為 4 KB 的項目。

一個寫入請求單位 = 每秒一個寫入，項目大小上限為 1 KB。

交易讀取請求需要兩個讀取請求單位，才能執行每秒讀取一個大小上限為 4 KB 的項目。

交易讀取請求需要兩個寫入請求單位，才能執行每秒寫入一個大小上限為 1 KB 的項目。

## 輸送量預設配額

AWS 在您的帳戶可在區域內佈建和使用的輸送量上設置一些預設配額。

帳戶層級的讀取輸送量和帳戶層級的寫入輸送量配額適用於帳戶層級。這些帳戶層級配額適用於給定區域中您帳戶的所有資料表和全域次要索引的佈建輸送容量總和。帳戶的所有可用輸送量可佈建至單一資料表或多份資料表。這些配額僅適用於使用佈建容量模式的資料表。

資料表層級的讀取輸送量和資料表層級的寫入輸送量配額，會以不同方式套用至使用佈建容量模式的資料表以及使用隨需容量模式的資料表。



對於佈建容量模式資料表和 GSI，配額是可佈建給區域中任何資料表或其任何 GSI 之讀取和寫入容量單位的最大數量。任何個別資料表及其所有 GSI 的總計也必須低於帳戶層級的讀取和寫入輸送量配額。這是對所有佈建資料表及其 GSI 的總計必須維持在帳戶層級讀取和寫入輸送量配額之下要求的額外補充。

對於隨需容量模式資料表和 GSI，資料表層級配額是可用於任何資料表或該表中任何個別 GSI 的最大讀取和寫入容量單位。在隨需模式下，不會將帳戶層級的讀取和寫入輸送量配額套用至資料表。

根據預設，以下是帳戶套用的輸送量配額。

	On-Demand	佈建	可調整
Per table	40,000 read request units and 40,000 write request units	40,000 read capacity units and 40,000 write capacity units	是
Per account	Not applicable	80,000 read capacity units and 80,000 write capacity units	是
Minimum throughput for any table or global secondary index	Not applicable	1 read capacity unit and 1 write capacity unit	是

您可以使用 [Service Quotas 主控台](#)、[AWS API](#) 與 [AWS CLI](#) 在需要時請求增加可調配額的配額。

針對帳戶層級輸送量配額，您可以使用 [Service Quotas 主控台](#)、[AWS CloudWatch 主控台](#)、[AWS API](#) 和 [AWS CLI](#) 建立 CloudWatch 警示，並在目前的使用量達到已套用配額值的指定百分比時自動收到通知。使用 CloudWatch 您還可以通過查看 AccountProvisionedReadCapacityUnits 和使用指標來監視您的使用 AccountProvisionedWriteCapacityUnits AWS 情況。若要進一步了解使用量指標，請參 [AWS 使用量指標](#)。

## 提高或降低輸送量 (已佈建的資料表)

### 提高佈建輸送量

您可以視需要經常提高 `ReadCapacityUnits` 或 `WriteCapacityUnits`，使用 `AWS Management Console` 或 `UpdateTable` 操作。在單一呼叫中，您可以提高資料表、該資料表中任何全域次要索引，或這些項目任意組合的佈建輸送量。新的設定值要在 `UpdateTable` 操作完成後才會生效。

當您新增佈建容量時，不能超過您的每個帳戶配額，而且 `DynamoDB` 不允許您非常快速地提高佈建容量。在這些限制之外，您可以將您資料表的佈建容量提高至您需要的數量。如需每個帳戶配額的詳細資訊，請參閱前一節 [輸送量預設配額](#)。

### 降低佈建輸送量

您可以在 `UpdateTable` 操作中降低每個資料表和全域次要索引的 `ReadCapacityUnits` 或 `WriteCapacityUnits` (或兩個都降低)。新的設定值要在 `UpdateTable` 操作完成後才會生效。

您每天可以在 `DynamoDB` 資料表上執行的佈建容量減少的次數存在預設配額。一天是根據國際標準時間 (UTC) 來定義。在給定的一天，只要您在當天還沒有執行任何其他減少，您可以在一小時內執行最多四次減少。之後，您可以每小時額外減少一次 (每 60 分鐘一次)。這有效地使一天中減少的最大數量達到 27 倍。

您可以使用 [Service Quotas 主控台](#)、[AWS API](#)與[AWS CLI](#)在需要時請求增加配額。

#### Important

資料表和全域次要索引調降限制是分開的，所以特定資料表的任何全域次要索引皆有各自的調降限制。但如果單一請求降低了資料表和全域次要索引的輸送量，只要其中之一超過目前的限制就會遭到拒絕。請求未得到部分處理。

### Example

在一天的第一個 4 小時中，具有全域次要索引的資料表可依下列方式修改：

- 調降資料表的 `WriteCapacityUnits` 或 `ReadCapacityUnits` (或兩者) 四次。
- 調降全域次要索引的 `WriteCapacityUnits` 或 `ReadCapacityUnits` (或兩者) 四次。

在同一天結束時，資料表和全域次要索引的輸送量有可能各調降 27 次。

## 預留容量

AWS 針對您的帳戶可以購買的有效預留容量設定預設配額。配額限制是寫入容量單位 (WCU) 和讀取容量單位 (RCU) 的預留容量總和。

	作用中的預留容量	可調整
每個帳戶	1,000,000 個佈建容量單位 (WCU _ RCU)	是

如果您嘗試單次購買超過 1,000,000 個佈建容量單位，將會收到此服務配額限制錯誤訊息。如果您擁有作用中的預留容量，並嘗試購買額外的預留容量，導致作用中的佈建容量單位總數超過 1,000,000 個，也會收到此服務配額限制錯誤訊息。

如果您的預留容量需求超過 1,000,000 個佈建容量單位，可以向[支援](#)團隊請求增加配額。

## 匯入配額

Amazon S3 的 DynamoDB 匯入可支援多達 50 個並行匯入任務，在 us-east-1、us-west-2 和 eu-west-1 區域的匯入來源物件大小總計達 15TB。在所有其他區域中，最多支援 50 個並行匯入任務，大小總計為 1TB。每個匯入任務在所有區域中最多可以使用 50,000 個 Amazon S3 物件。如需匯入和驗證的詳細資訊，請參閱[匯入格式配額與驗證](#)。

## Contributor Insights

當您在 DynamoDB 表上啟用客戶深入解析時，您仍會受到貢獻者見解規則限制的約束。如需更多相關資訊，請參閱 [CloudWatch Service Quotas](#)。

## 資料表

### 資料表大小

資料表大小沒有任何實際限制。就項目數或位元組數而言，資料表是沒有限制的。

### 每個帳戶每個區域的資料表數目上限

對於任何 AWS 帳戶，每個 AWS 區域都有 2,500 個表格的初始配額。

如果單一帳戶需要超過 2,500 個資料表，請聯絡您的 AWS 客戶團隊，了解將資料表數量增加到最多 10,000 個的可能性。如需超過 10,000 個資料表，建議的最佳實務是設定多個帳戶，每個帳戶最多可以提供 10,000 個資料表。

您可以使用 [Service Quotas 主控台](#)、[AWS API](#) 與 [AWS CLI](#) 查看帳戶中最大資料表數目的預設和套用配額值，並在需要時請求增加配額。您也可以開立票證給 [AWS 支援部門](#) 以申請增加配額。

您可以使用 [Service Quotas 主控台](#)、[AWS API](#) 和 [AWS CLI](#) 建立 CloudWatch 警示，以便在目前使用量達到目前配額的指定百分比時自動收到通知。使用 CloudWatch 您還可以通過查看使用指標來監視您的使用 TableCount AWS 情況。若要進一步了解使用量指標，請參 [AWS 使用量指標](#)。

## 全域資料表

AWS 在使用全域資料表時，在您可以佈建或使用的輸送上放置一些預設配額。

	On-Demand	佈建
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

事務性操作僅在最初寫入的 AWS 區域內提供原子性，一致性，隔離性和持久性 (ACID) 保證。全域資料表不支援跨區域交易。例如，假設您在美國東部 (俄亥俄) 和美國西部 (奧勒岡) 區域有一個包含複本的全域資料表，而且您在美國東部 (維吉尼亞北部) 區域執行 TransactWriteItems 作業。這麼做會使變更受到複寫，所以您可能會在美國西部 (奧勒岡) 區域中看到部分已完成交易。當變更已在來源區域遞交後，這些變更才會複寫至其他區域。

### Note

在某些情況下，您可能需要透過申請提高配額限制 AWS Support。如果您適用下列任一項，請參閱 <https://aws.amazon.com/support>：

- 如果您要為一個設定為使用 40,000 個以上的寫入容量單位 (WCU) 的資料表新增複本，您必須要求增加新增複本 WCU 配額的服務配額。
- 如果您要將一個複本或多個複本在 24 小時內新增到一個目標區域且合計大於 10TB，則必須針對您的新增複本資料回填配額請求新增服務配額。
- 如果您遇到類似下列的錯誤：
  - 無法在 'example\_table\_A' 區域中建立資料表 'example\_table' 的複本，因為超過 'example\_region\_B' 區域中目前帳戶的限制。

## 次要索引

### 每份資料表的次要索引

您可以定義 5 個本機次要索引的上限。

每個資料表有 20 個全域次要索引的預設配額。您可以使用 [Service Quotas 主控台](#)、[AWS API](#) 與 [AWS CLI](#) 檢查適用於您帳戶的每個資料表的全域次要索引預設配額和目前配額，並在需要時請求增加配額。您也可以開立票證至 <https://aws.amazon.com/support> 以請求增加配額。

每個 UpdateTable 操作只能建立或刪除一個全域次要索引。

### 每份資料表的預估次要索引屬性

您最多可以投影 100 個屬性到資料表所有的區域和全域次要索引。這只適用於使用者指定的投影屬性。

在 CreateTable 操作中，如果您指定 INCLUDE 的 ProjectionType，則 NonKeyAttributes 中指定的屬性總數，即所有次要索引總數，絕對不能超過 100。如果您將相同的屬性名稱投影到兩個不同的索引，在判斷總數時，這會計算為兩個不同的屬性。

這項限制不適用於含 KEYS\_ONLY 或 ALL 的 ProjectionType 的次要索引。

## 分割區索引鍵和排序索引鍵

### 分割區索引鍵長度

分割區索引鍵值的長度下限為 1 個位元組。長度上限為 2048 個位元組。

## 分割區索引鍵值

資料表或次要索引中，不同的分割區索引鍵值數目沒有實際限制。

## 排序索引鍵長度

排序索引鍵值的長度下限為 1 個位元組。長度上限為 1024 個位元組。

## 排序索引鍵值

一般而言，每個分割區索引鍵值的相異排序索引鍵值數目沒有實際限制。

例外狀況是有次要索引的資料表。項目集合是具有相同分割區索引鍵屬性值的一組項目。在全域次要索引中，項目集合與基底資料表相互獨立 (而且可以具有不同的分割區索引鍵屬性)，但在本機次要索引中，索引檢視會與資料表中的項目共同存在於相同的分割區中，並共用相同的分割區索引鍵屬性。鑑於此位置資訊，當資料表擁有一或多個 LSI 時，項目集合無法分發至多個分割區。

對於具有一或多個 LSI 的資料表，項目集合的大小不得超過 10GB。這包括所有基底資料表項目，以及具有與分割區索引鍵屬性相同值的所有投影 LSI 視圖。分割區的大小上限為 10GB。如需詳細資訊，請參閱[項目集合大小限制](#)。

## 命名規則

### 資料表名稱和次要索引名稱

資料表和次要索引名稱至少必須為 3 字元長，但不能超過 255 個字元。以下為允許的字元：

- A-Z
- a-z
- 0-9
- \_ (底線)
- - (連字號)
- . (點號)

### 屬性名稱

一般而言，屬性名稱至少必須為一個字元長，但不能超過 64 KB。



# 項目

## 項目大小

DynamoDB 的項目大小上限是 400 KB，包括屬性名稱二進位長度 (UTF-8 長度) 和屬性值長度 (另一個二進位長度)。屬性名稱算作大小限制的一部分。

例如，假設一個有兩個屬性的項目：一個屬性名為 "shirt-color"，其值為 "R"，另一個屬性名為 "shirt-size"，其值為 "M"。該項目的大小總計為 23 位元組。

## 具有本機次要索引之資料表的項目大小

針對資料表中每一個本機次要索引，每一個於下列總計的限制為 400 KB：

- 資料表的項目資料大小。
- 本機次要索引中對應項目 (包括其索引鍵值及投影屬性) 的大小。

## Attributes

### 每個項目的屬性名稱/值對

每個項目的屬性累積大小必須符合 DynamoDB 項目大小上限 (400 KB)。

### 清單、映射或集合的值數目

清單、映射或集合中值數目不限，只要含有值的項目符合 400 KB 項目大小限制。

### 屬性值

如果屬性未用作為資料表或索引的索引鍵屬性，則允許空字串和二進位屬性值。集合、清單和映射類型中允許空字串和二進位值。屬性值不得為空的集合 (字串集合、數字集合或二進位集合)。不過，允許空白清單與映射。

### 巢狀屬性深度

DynamoDB 支援巢狀屬性，最多 32 層深。



## 表達式參數

表達式參數包括 ProjectionExpression、ConditionExpression、UpdateExpression 和 FilterExpression。

### 長度

任何表達式字串的長度上限為 4 KB。例如，ConditionExpression a=b 的大小是 3 個位元組。

任何單一表達式屬性名稱或表達式屬性值的長度上限是 255 個位元組。例如，#name 是 5 個位元組；:val 是 4 個位元組。

表達式中所有替換變數的長度上限為 2 MB。這是所有 ExpressionAttributeNames 和 ExpressionAttributeValues 的長度加總。

### 運算子和運算元

UpdateExpression 中允許的運算子或函數數目上限為 300。例如，UpdateExpression SET a = :val1 + :val2 + :val3 包含兩個 "+" 運算子。

IN 比較子的運算元數目上限為 100。

### 保留字

DynamoDB 不會阻止您使用與保留字衝突的名稱。(如需完整清單，請參閱「[DynamoDB 中的保留字](#)」。)

但如果您在表達式參數中使用了保留字，您必須也要指定 ExpressionAttributeNames。如需詳細資訊，請參閱 [DynamoDB 中的表達式屬性名稱](#)。

## DynamoDB 交易

DynamoDB 交易 API 操作有下列限制：

- 一個交易不能包含超過 100 個不同的動作。
- 一個交易不能包含超過 4 MB 的資料。
- 交易中的任何兩個動作皆無法對相同資料表中的相同項目進行操作。例如，您不能在一個交易中，對相同項目同時進行 ConditionCheck 與 Update。

- 交易無法在多個 AWS 帳戶或區域中的表格上進行操作。
- 事務性操作僅在最初寫入的 AWS 區域內提供原子性，一致性，隔離性和持久性 (ACID) 保證。全域資料表不支援跨區域交易。舉例來說，假設您在美國東部 (俄亥俄) 與美國西部 (奧勒岡) 區域中有具有複本的全域資料表，並且在美國東部 (維吉尼亞北部) 區域中執行 `TransactWriteItems` 操作。這麼做會使變更受到複寫，所以您可能會在美國西部 (奧勒岡) 區域中看到部分已完成交易。當變更已在來源區域遞交後，這些變更才會複寫至其他區域。

## DynamoDB Streams

### DynamoDB Streams 中的碎片同時讀取

對於不是全域資料表的單一區域資料表，您最多可以設計兩個程序，以便同時讀取同一個 DynamoDB Streams 碎片。超過此限制會導致請求調節。對於全域資料表，我們建議您將同時讀取的數量限制為一個，以避免請求調節。

### 啟用 DynamoDB Streams 之資料表的寫入容量上限

AWS 在啟用 DynamoDB Streams 的情況下，針對 DynamoDB 資料表的寫入容量設置一些預設配額。這些預設配額僅適用於佈建讀/寫容量模式下的資料表。以下是預設情況下適用於您帳戶的輸送量配額。

- 美國東部 (維吉尼亞北部)、美國東部 (俄亥俄)、美國西部 (加利佛尼亞北部)、美國西部 (奧勒岡)、南美洲 (聖保羅)、歐洲 (法蘭克福)、歐洲 (愛爾蘭)、亞太區域 (東京)、亞太區域 (首爾)、亞太區域 (新加坡)、亞太區域 (雪梨)、中國 (北京) 區域：
  - 每份資料表：40,000 個寫入容量單位
- 所有其他區域：
  - 每份資料表：10,000 個寫入容量單位

您可以使用 [Service Quotas 主控台](#)、[AWS API](#) 與 [AWS CLI](#) 檢查已啟用 DynamoDB Streams 之資料表的最大寫入容量，以及您的帳戶上適用的預設配額和目前配額，並在需要時請求增加配額。您也可以開立票證給 [AWS 支援部門](#) 以申請增加配額。

#### Note

佈建輸送量配額也適用於已啟用 DynamoDB Streams 的 DynamoDB 資料表。當您請求增加已啟用 Streams 的資料表的寫入容量配額時，請確保也請求增加此資料表已佈建的輸送容量。如

需詳細資訊，請參閱[輸送量預設配額](#)。在處理較高輸送量的 DynamoDB Streams 時，也適用其他配額。如需詳細資訊，請參閱 [Amazon DynamoDB Streams API 參考指南](#)。

## DynamoDB Accelerator (DAX)

### AWS 區域可用性

如需可在其中使用 DAX 的 AWS 區域請 [DynamoDB](#)，請參閱中的。AWS 一般參考

### 節點

DAX 叢集包含一個主節點和 0 到 10 個僅供讀取複本節點。

單一 AWS 區域中的節點總數 (每個 AWS 帳戶) 不得超過 50 個。

### 參數群組

每個區域最多可以建立 20 個 DAX 參數群組。

### 子網路群組

每個區域最多可以建立 50 個 DAX 子網路群組。

您可在子網路群組內定義最多 20 個子網路。

## API 特定限制

### CreateTable/UpdateTable/DeleteTable/PutResourcePolicy/DeleteResourcePolicy

一般而言，您最多可以使用任何組合同時執行 500

[CreateTable](#)個[DeleteTable](#)[PutResourcePolicy](#)、[DeleteResourcePolicy](#)要

求。[UpdateTable](#)換言之，CREATING、UPDATING 或 DELETING 狀態的資料表總數不能超過 500。

您可以在一組表格中CreateTable，每秒最多提交 2,500 個可變

(DeleteTableUpdateTablePutResourcePolicy、[DeleteResourcePolicy](#)) 控制平面 API 請求。不過，PutResourcePolicy和DeleteResourcePolicy要求的個別限制較低。如需詳細資訊，請參閱PutResourcePolicy和的下列配額詳細資料DeleteResourcePolicy。

CreateTable以及PutResourcePolicy包含以資源為基礎之政策的要求，會計為原則的每個 KB 的兩個額外要求。例如，原則大小為 5 KB 的CreateTable或請PutResourcePolicy求將計為 11 個請求。請求 1 個，以資源為基礎的策略為 10 (2 x 5 KB)。CreateTable同樣地，大小為 20 KB 的原則會計為 41 個請求。請求為 1，而資源型政策則為 40 (2 x 20 KB)。CreateTable

### **PutResourcePolicy**

您每秒最多可以在一組表中提交 25 個 PutResourcePolicy API 請求。成功要求個別資料表之後，接下來的 15 秒內不會支援新的要PutResourcePolicy求。

以資源為基礎的政策文件所支援的大小上限為 20 KB。根據此限制計算原則大小時，DynamoDB 會計算空格。

### **DeleteResourcePolicy**

您每秒最多可以在一組表格中提交 50 個 DeleteResourcePolicy API 請求。成功PutResourcePolicy要求個別資料表後，接下來的 15 秒內不支援要DeleteResourcePolicy求。

### **BatchGetItem**

單一 BatchGetItem 操作最多可擷取 100 個項目。所有擷取項目大小總計不能超過 16 MB。

### **BatchWriteItem**

單一 BatchWriteItem 操作最多可以包含 25 個 PutItem 或 DeleteItem 請求。所有寫入項目大小總計不能超過 16 MB。

### **DescribeStream**

您可以以每秒 10 次的最高速率撥打電話DescribeStream。

### **DescribeTableReplicaAutoScaling**

DescribeTableReplicaAutoScaling 方式僅支援每秒 10 個請求。

### **DescribeLimits**

DescribeLimits 應只能定期呼叫。如果每分鐘呼叫一次以上，您應該會收到調節錯誤。

## **DescribeContributorInsights/ListContributorInsights/UpdateContributorInsights**

DescribeContributorInsights、ListContributorInsights 和 UpdateContributorInsights 應只能定期呼叫。DynamoDB 於每個 API 支援最多每秒五個請求。

## **DescribeTable/ListTables/GetResourcePolicy**

您每秒最多可以提交 2,500 個要求，結合唯讀 (DescribeTableListTables、和 GetResourcePolicy) 控制平面 API 要求。GetResourcePolicyAPI 的個別限制為每秒 100 個要求較低。

## **Query**

Query 的結果集受到每次呼叫為 1 MB 的限制。您可以使用查詢回應的 LastEvaluatedKey 擷取更多結果。

## **Scan**

Scan 的結果集受到每次呼叫為 1 MB 的限制。您可以使用掃描回應的 LastEvaluatedKey 擷取更多結果。

## **UpdateKinesisStreamingDestination**

執行 UpdateKinesisStreamingDestination 操作時，您可以在 24 小時內設定 ApproximateCreationDateTimePrecision 最多 3 次的新值。

## **UpdateTableReplicaAutoScaling**

UpdateTableReplicaAutoScaling 方式僅支援每秒 10 個請求。

## **UpdateTimeToLive**

每小時針對每個指定資料表，此 UpdateTableTimeToLive 方法僅支援一個啟用或停用 Time to Live (TTL) 的請求。完整處理此變更最多可能需要一個小時。在此一小時持續時間 UpdateTimeToLive 內對同一個表格進行任何其他呼叫都會產生 ValidationException。

## DynamoDB 靜態加密

您可以在 AWS 擁有的金鑰、a 和客戶管理的金鑰之間切換，最多可以在每個 24 小時視窗中隨時切換四次，從建立表格的時間開始。AWS 受管金鑰如果過去 6 小時內沒有變化，則允許額外變更。這會有效地將一天的變更次數增加到 8 次 (第一個 6 小時內 4 次變更，以及一天中後續每 6 小時 1 次的變更)。

即使上述配額已用盡，您也可以視需要切換加密金鑰以使用。AWS 擁有的金鑰

除非您請求較高的數量，否則配額如下。若要請求提高服務配額，請參閱 <https://aws.amazon.com/support>。

## 資料表匯出至 Amazon S3

完整匯出：所有執行中的資料表可以匯出最多 300 個並行匯出任務，或匯出最多 100TB。匯出排入佇列之前，會檢查這兩項限制。

增量匯出：在最短 15 分鐘、最長 24 小時的匯出時段內，可同時匯出最多 300 個並行任務，或 100TB 的資料表大小。

## 備份和還原

當您使用 DynamoDB 隨需或連續備份來還原表格資料時，最多可以執行 50 TB 的同時還原。使用時 AWS Backup，您最多可以執行 50 個並行還原，總計 25 TB。如需備份的詳細資訊，請參閱 [使用 DynamoDB 的隨需備份與還原](#)。

## 低階 API 參考

《[Amazon DynamoDB API 參考](#)》包含下列支援的完整操作清單：

- [DynamoDB](#)。
- [DynamoDB Streams](#)。
- [DynamoDB Accelerator \(DAX\)](#)。

# 疑難排解 Amazon DynamoDB

下列主題提供當您在使用 Amazon DynamoDB 時，可能遇到的錯誤和問題的故障診斷建議。如果您發現未列在此處的問題，您可以使用此頁面上的 Feedback (意見回饋) 按鈕來報告。

如需更多故障診段建議和常見支援問題的解答，請瀏覽 [AWS 知識中心](#)。

## 主題

- [對 Amazon DynamoDB 中的延遲問題進行疑難排解](#)
- [使用佈建容量模式的 DynamoDB 表格的節流問題](#)

## 對 Amazon DynamoDB 中的延遲問題進行疑難排解

如果您的工作負載似乎遇到高延遲，您可以分析 `CloudWatchSuccessfulRequestLatency` 指標，並檢查平均延遲以查看其是否與 DynamoDB 相關。所報告 `SuccessfulRequestLatency` 中的某些變化是正常的，偶爾的尖峰 (尤其是在 `Maximum` 統計資料中) 也不用擔心。但是，如果 `Average` 統計資料顯示急劇增加且持續存在，則應該查看 AWS Service Health 儀表板和您的 Personal Health 儀表板，以取得更多資訊。一些可能的原因包括資料表中項目的大小 (1kb 項目和 400kb 項目的延遲會有所不同) 或查詢的大小 (10 個項目相較於 100 個項目)。

如有必要，請考慮使用 AWS Support 建立支援案例，然後根據您的執行手冊，繼續評估您應用程式的任何可用回復選項 (例如，如果您有多區域架構，則撤離一個區域)。您應該記錄請求 ID，以便在您開啟支援案例 AWS Support 時提供這些 ID 的緩慢要求。

`SuccessfulRequestLatency` 指標僅測量 DynamoDB 服務內部的延遲 - 不包括用戶端活動和網路往返時間。若要深入了解用戶端呼叫 DynamoDB 服務的整體延遲，您可以在 AWS SDK 中啟用延遲指標記錄。

### Note

對於大部分單一操作 (透過完全指定主索引鍵的值來套用至單一項目的操作)，DynamoDB 提供個位數的毫秒 `Average SuccessfulRequestLatency`。此值不包括存取 DynamoDB 端點之呼叫者程式碼的傳輸負荷。對於多項目資料操作，延遲會因結果集的大小、傳回資料結構的複雜度，以及套用的任何條件表達式和篩選條件表達式等因素而有所不同。對於具有相同參數之相同資料集的重複多項操作，DynamoDB 提供高度一致的 `Average SuccessfulRequestLatency`。



請考慮下列一或多種策略來減少延遲：

- 調整請求逾時和重試行為：從用戶端到 DynamoDB 的路徑需周遊許多元件，每個元件在設計時都考慮到備援。想想網路恢復能力範圍、TCP 封包逾時以及 DynamoDB 本身的分散式架構。預設 SDK 行為旨在為大部分應用程式找到適當的平衡點。如果最佳延遲是您的最高優先順序，則應考慮調整 SDK 的預設請求逾時以及重試設定，以密切追蹤用戶端所測量到成功請求的典型延遲。如果一個請求花費的時間明顯比正常時間還要久，最終成功的機率會降低 - 如果能快速檢錯並發出新請求，那麼新請求可能會採取不同的路徑並且快速成功。請記住，在這些設定中過於積極可能會帶來不利影響。您可以在[針對延遲感知的 Amazon DynamoDB 應用程式調整 AWS Java SDK HTTP 請求設定](#)中找到此主題的實用討論。
- 減少用戶端與 DynamoDB 端點之間的距離：如果您的使用者遍布全球，請考慮使用 [全域資料表：DynamoDB 的多區域複寫](#)。使用全域資料表，您可以指定希望資料表可用的 AWS 區域。從本機全域資料表複本讀取資料，可大幅減少使用者的延遲。此外，請考慮使用 DynamoDB [閘道端點](#)，將用戶端流量保留在 VPC 中。
- 使用快取：如果您的流量需要大量讀取，請考慮使用快取服務，例如 [使用 DynamoDB Accelerator \(DAX\) 的記憶體內加速](#)。DAX 是適用於 DynamoDB 的全受管、高可用性記憶體快取，即使每秒數百萬個請求也能將時間從毫秒縮短到微秒，提供高達 10 倍的效能改進。
- 重複使用連線：DynamoDB 請求透過預設為 HTTPS 的已驗證工作階段發出。啟動連線需要時間，因此第一個請求的延遲時間會高於一般請求。經由已啟動連線的請求，可以提供 DynamoDB 一致的低延遲。因此，如果沒有發出其他請求，您可能希望每 30 秒發出一個「保持連線」GetItem 請求，以避免建立新連線的延遲。
- 使用最終一致讀取：如果您的應用程式不需要高度一致性讀取，請考慮使用預設的最終一致讀取。最終一致讀取的成本較低，也不太可能發生暫時性延遲增加。如需更多詳細資訊，請參閱 [讀取一致性](#)。

## 使用佈建容量模式的 DynamoDB 表格的節流問題

如果您的應用程式超過資料表或索引上佈建的輸送容量，將會請求調節。調節可防止您的應用程式使用太多容量單位。當 DynamoDB 節流讀取或寫入作業時，它會傳回給呼叫者 `ProvisionedThroughputExceededException`。應用程式接著可以採取適當動作，例如在重試請求之前等待短間隔。

本主題討論如何疑難排解常見的節流問題，以及如何用 CloudWatch 來調查問題可能來自何處。

主題

- [針對限流問題進行疑難排解](#)

- [使用 CloudWatch 指標調查節流問題](#)

## 針對限流問題進行疑難排解

對於疑難排解看似與節流有關的問題，重要的第一步是確認節流是來自 DynamoDB 還是來自應用程式。

以下是一些常見案例，以及協助解決問題的可能步驟。

DynamoDB 表格似乎具有足夠的佈建容量，但請求正在限制

當輸送量低於每分鐘的平均值，但超過每秒可用量時，就會發生這種情況。DynamoDB 只會向分鐘層級量度報告 CloudWatch，這些量度是以一分鐘的總和和計算得出的平均值。但是 DynamoDB 本身會套用每秒的速率限制。因此，如果在該分鐘的一小部分 (例如幾秒鐘或更短的時間) 內發生過多的輸送量，則可將該分鐘剩餘時間的請求加以限流。

例如，如果我們在一個表上佈建了 60 個 WCU，那麼它可以在一分鐘內執行 3600 個寫入操作。但是，如果所有 3600 WCU 請求都在同一秒鐘內命中，則該分鐘的其餘部分將受到限流。

解決這種案例的一種方法是將一些抖動和指數退避新增至 API 呼叫。如需詳細資訊，請參閱這篇有關[指數退避和抖動](#)的貼文。

自動縮放功能已啟用，但表格仍在進行節流

此情況可能發生在流量突增期間。當 2 個資料點在一分鐘範圍內違反設定的目標使用率值時，就可以觸發自動擴展。因此，由於消耗的容量在兩分鐘內高於目標使用率，因此可能會發生 auto 調整規模。但是，如果尖峰相隔超過一分鐘，則可能不會觸發 auto 縮放。

同樣地，當連續 15 個資料點低於目標使用率時，就會觸發縮減規模事件。在任何一種情況下，在觸發 auto 縮放之後，都會叫用 UpdateTable API 作業。然後可能需要幾分鐘的時間來更新表格或索引的佈建容量。在此期間，任何超過表格先前佈建容量的要求都會受到限制。

總而言之，auto 擴展需要連續的資料點，其中違反了目標使用率值以擴展 DynamoDB 表。因此，不建議將 auto 擴展作為處理 spikey 工作負載的解決方案。如需詳細資訊，請參閱[auto 調整成本最佳化文件](#)。

熱鍵可能導致節流問題

在 DynamoDB 中，沒有高基數的分割區索引鍵可能會產生許多僅以幾個分割區為目標的請求。如果產生的熱分割區超過每秒 3000 RCU 或 1000 WCU 的分割區限制，這可能會導致節流。診斷工具

CloudWatch 參與者見解 (CCI) 可以透過針對每個資料表的項目存取模式提供 CCI 圖形來協助偵錯此問題。您可以持續監控 DynamoDB 資料表中最常存取的索引鍵和其他流量趨勢。如需有關 CloudWatch 貢獻者深入解析的詳細資訊，請參閱 DynamoDB 的參[CloudWatch 與者深入解析](#)。如需詳細資訊，請參閱[設計分割區索引鍵以分配您的工作負載](#)和[選擇正確的 DynamoDB 分區金鑰](#)。

您的資料表流量超出表格層級輸送量配額。

資料表層級的讀取輸送量和資料表層級的寫入輸送量配額適用於任何區域的帳戶層級。這些配額適用於同時具有佈建容量模式和隨需容量模式的資料表。根據預設，放置在資料表上的輸送量配額為 40,000 個讀取請求單位和 40,000 個寫入請求單位。如果資料表中的流量超過這個配額，資料表可能會遭到限流。有關如何防止這種情況發生的詳細資訊，請參閱[監視 DynamoDB 的操作感知](#)。

若要解決此問題，請使用 Service Quotas 主控台來提高您帳戶的資料表層級讀取或寫入輸送量配額。

## 使用 CloudWatch 指標調查節流問題

以下是節流事件期間要監視的一些 DynamoDB 指標。您可以使用這些來協助找出建立限制要求的作業，並找出根本問題。

- **ThrottledRequests**
  - 一個限制的請求可以包含多個限制事件，因此事件與請求相比，與示例更相關。例如，當您使用 GSI 更新資料表中的項目時，會有多個事件：資料表的寫入作業，以及每個索引的寫入作業。即使其中一個或多個事件被限制，也只會有一個。ThrottledRequest
- **ReadThrottleEvents**
  - 注意超出資料表或 GSI 佈建 RCU 的請求。
- **WriteThrottleEvents**
  - 注意超出資料表或 GSI 佈建 WCU 的請求。
- **OnlineIndexConsumedWriteCapacity**
  - 將 GSI 新增至資料表時，請注意耗用的 WCU 數目。請注意，GSI 的 ConsumedWriteCapacityUnits 不包括建立索引期間耗用的 WCU。
  - 如果您將 GSI 的 WCU 設定得太低，則回填階段期間的傳入寫入活動可能會受到限制。
- **Provisioned Read/Write**
  - 檢視在指定時段，針對資料表或指定的全域次要索引耗用了多少已佈建的讀取或寫入容量單位。
  - 請注意，TableName 維度預設僅針對資料表傳回 ProvisionedReadCapacityUnits。若要檢視全域次要索引佈建的讀取或寫入容量單位數目，您必須同時指定 TableName 和 GlobalSecondaryIndexName。

- Consumed Read/Write
  - 檢視指定時段內耗用了多少個讀取或寫入容量單位。

如需 DynamoDB CloudWatch 度量的詳細資訊，請參閱 [DynamoDB 指標和維度](#)

# DynamoDB 附錄

## 主題

- [對 SSL/TLS 連線建立問題進行疑難排解](#)
- [監控工具](#)
- [範例資料表和資料](#)
- [建立範例資料表和上傳資料](#)
- [使用 AWS SDK for Python \(Boto\) 的 DynamoDB 範例應用程式：井字遊戲](#)
- [使用以下方式匯出和匯入 DynamoDB 資料 AWS Data Pipeline](#)
- [適用於 Titan 的 Amazon DynamoDB 儲存後端](#)
- [DynamoDB 中的保留字](#)
- [舊式條件式參數](#)
- [舊版低階 API \(2011-12-05\)](#)
- [AWS 適用於 Java 1.x 的開發套件範例](#)

## 對 SSL/TLS 連線建立問題進行疑難排解

Amazon DynamoDB 正在移動端點，以便保護由 Amazon Trust Services (ATS) 憑證授權機構而非第三方憑證授權機構所簽署的憑證。2017 年 12 月，我們推出 EU-WEST-3 (巴黎) 區域，附有由 Amazon Trust Services 發行的安全憑證。所有於 2017 年 12 月之後推出的新區域都具有附帶 Amazon Trust Services 發行的憑證的終端節點。本指南說明如何對 SSL/TLS 連線建立問題進行驗證和疑難排解。

## 測試您的應用程式或服務

大多數 AWS 開發套件和命令列界面 (CLI) 都支援 Amazon 信任服務憑證授權單位。如果您使用的是在二零一三年十月二十九日之前發行的開發 AWS 套件版本，您必須升級。 .NET，Java，PHP JavaScript，圍棋和 C ++ SDK 和 CLI 不捆綁任何證書，它們的證書來自基礎操作系統。自 2015 年 6 月 10 日起，Ruby 開發套件已包含至少一個必要 CA。在該日期之前，Ruby 第 2 版開發套件並無憑證套件。如果您使用 AWS SDK 的不受支援、自訂或修改版本，或使用自訂信任存放區，則可能沒有 Amazon 信任服務憑證授權單位所需的支援。

若要驗證能否存取 DynamoDB 端點，您需要進行測試來存取 EU-WEST-3 區域中的 DynamoDB API 或 DynamoDB Streams API，並驗證 TLS 交握是否成功。您需要在這種測試中存取的特定端點是：

- DynamoDB : <https://dynamodb.eu-west-3.amazonaws.com>
- DynamoDB Streams : <https://streams.dynamodb.eu-west-3.amazonaws.com>

如果應用程式不支援 Amazon Trust Services 憑證授權機構 (CA) , 則會看到下列其中一項失敗 :

- SSL/TLS 溝通錯誤
- 在軟體收到錯誤指出 SSL/TLS 溝通失敗之前, 會有長時間延遲。延遲時間依用戶端的重試策略和逾時組態而定。

## 測試您的用戶端瀏覽器

若要確認瀏覽器是否可連線到 Amazon DynamoDB, 請開啟以下網址 : <https://dynamodb.eu-west-3.amazonaws.com>。如果測試成功, 則會看到如下的訊息 :

```
healthy: dynamodb.eu-west-3.amazonaws.com
```

如果測試不成功, 則會顯示類似以下內容的錯誤 : <https://untrusted-root.badssl.com/>。

## 更新您的軟體應用程式用戶端

如果應用程式尚未支援下列任何 CA, 則在存取 DynamoDB 或 DynamoDB Streams API 端點時 (無論是透過瀏覽器還是編寫程式的方式), 均需更新用戶端機器上的信任 CA 清單 :

- Amazon 根 CA 1
- Starfield Services 根憑證授權機構 : G2
- Starfield 類別 2 憑證授權機構

如果用戶端已經信任上述三個 CA 之中的任一個, 則這些用戶端會信任 DynamoDB 使用的憑證, 不需要進行任何動作。不過, 如果用戶端尚未信任上述任何 CA, 則與 DynamoDB 或 DynamoDB Streams API 的 HTTPS 連線將會失敗。欲了解更多信息, 請訪問此博客文章 : <https://aws.amazon.com/blogs/security/how-to-prepare-for-aws-move-to-its--own-certificate-authority/>。

## 更新您的用戶端瀏覽器

只要更新瀏覽器即可更新瀏覽器中的憑證套件。常用瀏覽器的說明可以在瀏覽器網站上找到 :

- Chrome : <https://support.google.com/chrome/answer/95414?hl=en>

- 火狐瀏覽器:<https://support.mozilla.org/en-US/kb/update-firefox-latest-version>
- Safari : <https://support.apple.com/en-us/HT204416>
- 互聯網瀏覽器:<https://support.microsoft.com/en-us/help/17295/windows-internet-explorer-which-version-is-installed> 版本 #ie = 其他

## 手動更新您的憑證套件

如果您無法存取 DynamoDB API 或 DynamoDB Streams API，則需要更新憑證服務包。為此，您需要匯入至少一個必要 CA。您可以在此尋找這些 CA：<https://www.amazontrust.com/repository/>。

下列作業系統和程式設計語言支援 Amazon Trust Services 憑證授權機構：

- 具備自 2005 年 1 月起之更新的 Microsoft Windows 版本，Windows Vista、Windows 7、Windows Server 2008 和更新版本。
- Mac OS X 10.4 搭配 Java for MacOS X 10.4 第 5 版、Mac OS X 10.5 和更新版本。
- Red Hat Enterprise Linux 5 (2007 年 3 月)、Linux 6 和 Linux 7 以及 CentOS 5、CentOS 6、CentOS 7
- Ubuntu 8.10
- Debian 5.0
- Amazon Linux (所有版本)
- Java 1.4.2\_12、Java 5 更新版本 2 及所有更新版本，包括 Java 6、Java 7、Java 8

如果您仍然無法連線，請參閱您的軟體文件、作業系統廠商，或聯絡 Sup AWS port <https://aws.amazon.com/support> 以取得進一步協助。

## 監控工具

AWS 提供您可用來監視 DynamoDB 的工具。您可以設定其中一些工具為您進行監控，有些工具需要手動介入。建議您盡可能自動化監控任務。

### 自動化監控工具

您可以使用下列自動化監控工具來監看 DynamoDB，並在發生錯誤時進行回報：

- Amazon CloudWatch 警示 — 觀看您指定期間內的單一指標，並根據指定臨界值在多個時段內相對於指定閾值的指標值執行一或多個動作。動作是傳送至亞馬遜簡單通知服務 (Amazon SNS) 主題或



Amazon EC2 Auto Scaling 政策的通知。CloudWatch 警示不會僅因為處於特定狀態而叫用動作；狀態必須已變更並維持指定數目的期間。

- Amazon CloudWatch 日誌 — 監控、存放和存取來自 AWS CloudTrail 或其他來源的日誌檔。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的 [監控日誌檔](#)。
- Amazon E CloudWatch vents — 匹配事件並將其路由到一個或多個目標函數或串流，以進行變更、擷取狀態資訊並採取糾正措施。有關更多信息，請參閱 [Amazon 用 CloudWatch 戶指南中的 Amazon CloudWatch 事件是什麼](#)。
- AWS CloudTrail 日誌監控 — 在帳戶之間共享日誌文件，通過將日 CloudTrail 誌文件發送到 CloudWatch 日誌中實時監控日誌文件，在 Java 中寫入日誌處理應用程序，並驗證您的日誌文件在交付之後沒有更改 CloudTrail。若要取得更多資訊，請參閱 [《使用指南》中的〈AWS CloudTrail 使用 CloudTrail 記錄檔〉](#)。

## 手動監控工具

監視 DynamoDB 的另一個重要部分是手動監視 CloudWatch 警報未涵蓋的項目。DynamoDB、CloudWatch Trusted Advisor、和其他 AWS 主控台儀表板可提供您環境狀態的 at-a-glance AWS 檢視。我們建議您也檢查 DynamoDB 上的記錄檔。

- DynamoDB 儀表板會顯示：
  - 最近的提醒
  - 總容量
  - 服務運作狀態
- CloudWatch 主頁顯示：
  - 目前警示與狀態
  - 警示與資源的圖表
  - 服務運作狀態

此外，您可以使用執行 CloudWatch 以下操作：

- 建立 [自定義儀表板](#) 來監控您注重的服務
- 用於疑難排解問題以及探索驅勢的圖形指標資料。
- 搜尋並瀏覽所有資 AWS 源指標
- 建立與編輯要通知發生問題的警示



## 範例資料表和資料

所以此《Amazon DynamoDB 開發人員指南》使用範列表來說明 DynamoDB 的各個層面。

資料表名稱	主索引鍵
ProductCatalog	簡單主索引鍵： <ul style="list-style-type: none"><li>• Id (數字)</li></ul>
Forum	簡單主索引鍵： <ul style="list-style-type: none"><li>• Name (字串)</li></ul>
Thread	複合主索引鍵： <ul style="list-style-type: none"><li>• ForumName (字串)</li><li>• Subject (字串)</li></ul>
Reply	複合主索引鍵： <ul style="list-style-type: none"><li>• Id (字串)</li><li>• ReplyDateTime (字串)</li></ul>

此 Reply 資料表有一個名為 PostedBy-Message-Index 的全域次要索引。此索引會加速對 Reply 資料表的兩個非索引鍵屬性的查詢。

索引名稱	主索引鍵
PostedBy-Message-Index	複合主索引鍵： <ul style="list-style-type: none"><li>• PostedBy (字串)</li><li>• Message (字串)</li></ul>

如需這些資料表的詳細資訊，請參閱 [步驟 1：建立資料表](#) 和 [第 2 步：使用控制台或將數據寫入表 AWS CLI](#)。

## 範例資料檔案

### 主題

- [ProductCatalog 範例資料](#)
- [Forum 範例資料](#)
- [Thread 範例資料](#)
- [Reply 範例資料](#)

下列各部分會顯示用於載入 ProductCatalog、Forum、Thread 和 Reply 的資料表。

每個資料檔案會包含多個 PutRequest 元素，每個元素則會包含一個項目。借助 AWS Command Line Interface(AWS CLI)，可將這些 PutRequest 元素用作 BatchWriteItem 操作的輸入。

如需詳細資訊，請參閱 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 中的 [第 2 步：使用控制台或將數據寫入表 AWS CLI](#)。

### ProductCatalog 範例資料

```
{
 "ProductCatalog": [
 {
 "PutRequest": {
 "Item": {
 "Id": {
 "N": "101"
 },
 "Title": {
 "S": "Book 101 Title"
 },
 "ISBN": {
 "S": "111-1111111111"
 },
 "Authors": {
 "L": [
 {
 "S": "Author1"
 }
]
 },
 "Price": {
```

```
 "N": "2"
 },
 "Dimensions": {
 "S": "8.5 x 11.0 x 0.5"
 },
 "PageCount": {
 "N": "500"
 },
 "InPublication": {
 "BOOL": true
 },
 "ProductCategory": {
 "S": "Book"
 }
 }
 },
 {
 "PutRequest": {
 "Item": {
 "Id": {
 "N": "102"
 },
 "Title": {
 "S": "Book 102 Title"
 },
 "ISBN": {
 "S": "222-2222222222"
 },
 "Authors": {
 "L": [
 {
 "S": "Author1"
 },
 {
 "S": "Author2"
 }
]
 },
 "Price": {
 "N": "20"
 },
 "Dimensions": {
 "S": "8.5 x 11.0 x 0.8"
 }
 }
 }
 }
}
```

```
 },
 "PageCount": {
 "N": "600"
 },
 "InPublication": {
 "BOOL": true
 },
 "ProductCategory": {
 "S": "Book"
 }
 }
},
{
 "PutRequest": {
 "Item": {
 "Id": {
 "N": "103"
 },
 "Title": {
 "S": "Book 103 Title"
 },
 "ISBN": {
 "S": "333-3333333333"
 },
 "Authors": {
 "L": [
 {
 "S": "Author1"
 },
 {
 "S": "Author2"
 }
]
 },
 "Price": {
 "N": "2000"
 },
 "Dimensions": {
 "S": "8.5 x 11.0 x 1.5"
 },
 "PageCount": {
 "N": "600"
 }
 }
 }
}
```

```
 "InPublication": {
 "BOOL": false
 },
 "ProductCategory": {
 "S": "Book"
 }
 }
},
{
 "PutRequest": {
 "Item": {
 "Id": {
 "N": "201"
 },
 "Title": {
 "S": "18-Bike-201"
 },
 "Description": {
 "S": "201 Description"
 },
 "BicycleType": {
 "S": "Road"
 },
 "Brand": {
 "S": "Mountain A"
 },
 "Price": {
 "N": "100"
 },
 "Color": {
 "L": [
 {
 "S": "Red"
 },
 {
 "S": "Black"
 }
]
 },
 "ProductCategory": {
 "S": "Bicycle"
 }
 }
 }
}
```

```
 }
 },
 {
 "PutRequest": {
 "Item": {
 "Id": {
 "N": "202"
 },
 "Title": {
 "S": "21-Bike-202"
 },
 "Description": {
 "S": "202 Description"
 },
 "BicycleType": {
 "S": "Road"
 },
 "Brand": {
 "S": "Brand-Company A"
 },
 "Price": {
 "N": "200"
 },
 "Color": {
 "L": [
 {
 "S": "Green"
 },
 {
 "S": "Black"
 }
]
 },
 "ProductCategory": {
 "S": "Bicycle"
 }
 }
 }
 },
 {
 "PutRequest": {
 "Item": {
 "Id": {
 "N": "203"
```

```
 },
 "Title": {
 "S": "19-Bike-203"
 },
 "Description": {
 "S": "203 Description"
 },
 "BicycleType": {
 "S": "Road"
 },
 "Brand": {
 "S": "Brand-Company B"
 },
 "Price": {
 "N": "300"
 },
 "Color": {
 "L": [
 {
 "S": "Red"
 },
 {
 "S": "Green"
 },
 {
 "S": "Black"
 }
]
 },
 "ProductCategory": {
 "S": "Bicycle"
 }
 }
},
{
 "PutRequest": {
 "Item": {
 "Id": {
 "N": "204"
 },
 "Title": {
 "S": "18-Bike-204"
 }
 }
 }
}
```

```
 "Description": {
 "S": "204 Description"
 },
 "BicycleType": {
 "S": "Mountain"
 },
 "Brand": {
 "S": "Brand-Company B"
 },
 "Price": {
 "N": "400"
 },
 "Color": {
 "L": [
 {
 "S": "Red"
 }
]
 },
 "ProductCategory": {
 "S": "Bicycle"
 }
 }
},
{
 "PutRequest": {
 "Item": {
 "Id": {
 "N": "205"
 },
 "Title": {
 "S": "18-Bike-204"
 },
 "Description": {
 "S": "205 Description"
 },
 "BicycleType": {
 "S": "Hybrid"
 },
 "Brand": {
 "S": "Brand-Company C"
 },
 "Price": {
```



```

 "N": "500"
 },
 "Color": {
 "L": [
 {
 "S": "Red"
 },
 {
 "S": "Black"
 }
]
 },
 "ProductCategory": {
 "S": "Bicycle"
 }
 }
 }
}

```

## Forum 範例資料

```

{
 "Forum": [
 {
 "PutRequest": {
 "Item": {
 "Name": {"S": "Amazon DynamoDB"},
 "Category": {"S": "Amazon Web Services"},
 "Threads": {"N": "2"},
 "Messages": {"N": "4"},
 "Views": {"N": "1000"}
 }
 }
 },
 {
 "PutRequest": {
 "Item": {
 "Name": {"S": "Amazon S3"},
 "Category": {"S": "Amazon Web Services"}
 }
 }
 }
]
}

```

```
 }
]
}
```

## Thread 範例資料

```
{
 "Thread": [
 {
 "PutRequest": {
 "Item": {
 "ForumName": {
 "S": "Amazon DynamoDB"
 },
 "Subject": {
 "S": "DynamoDB Thread 1"
 },
 "Message": {
 "S": "DynamoDB thread 1 message"
 },
 "LastPostedBy": {
 "S": "User A"
 },
 "LastPostedDateTime": {
 "S": "2015-09-22T19:58:22.514Z"
 },
 "Views": {
 "N": "0"
 },
 "Replies": {
 "N": "0"
 },
 "Answered": {
 "N": "0"
 },
 "Tags": {
 "L": [
 {
 "S": "index"
 },
 {
 "S": "primarykey"
 }
]
 }
 }
 }
]
 }
}
```

```
 {
 "S": "table"
 }
]
}
},
{
 "PutRequest": {
 "Item": {
 "ForumName": {
 "S": "Amazon DynamoDB"
 },
 "Subject": {
 "S": "DynamoDB Thread 2"
 },
 "Message": {
 "S": "DynamoDB thread 2 message"
 },
 "LastPostedBy": {
 "S": "User A"
 },
 "LastPostedDateTime": {
 "S": "2015-09-15T19:58:22.514Z"
 },
 "Views": {
 "N": "3"
 },
 "Replies": {
 "N": "0"
 },
 "Answered": {
 "N": "0"
 },
 "Tags": {
 "L": [
 {
 "S": "items"
 },
 {
 "S": "attributes"
 }
]
 }
 }
 }
}
```

```

 "S": "throughput"
 }
]
 }
},
{
 "PutRequest": {
 "Item": {
 "ForumName": {
 "S": "Amazon S3"
 },
 "Subject": {
 "S": "S3 Thread 1"
 },
 "Message": {
 "S": "S3 thread 1 message"
 },
 "LastPostedBy": {
 "S": "User A"
 },
 "LastPostedDateTime": {
 "S": "2015-09-29T19:58:22.514Z"
 },
 "Views": {
 "N": "0"
 },
 "Replies": {
 "N": "0"
 },
 "Answered": {
 "N": "0"
 },
 "Tags": {
 "L": [
 {
 "S": "largeobjects"
 },
 {
 "S": "multipart upload"
 }
]
 }
 }
 }
}

```

```
 }
 }
]
}
```

## Reply 範例資料

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

# 建立範例資料表和上傳資料

## 主題

- [建立範例資料表並使用 AWS SDK for Java](#)
- [建立範例資料表並使用 AWS SDK for .NET](#)

在 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 中，請先以 DynamoDB 主控台建立資料表，然後使用 AWS CLI 將資料新增至資料表中。本附錄提供以程式設計方式建立資料表和新增資料的程式碼。

## 建立範例資料表並使用 AWS SDK for Java

下列 Java 程式碼範例會建立資料表，並將資料上傳至資料表。在 [在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 會顯示產生的資料表結構和資料。如需使用 Eclipse 執行此程式碼的 step-by-step 指示，請參閱 [Java 程式碼範例](#)。

```
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());
 }
}
}
```

## 建立範例資料表並使用 AWS SDK for .NET

下列 C# 程式碼範例會建立資料表，並將資料上傳至資料表。[在 DynamoDB 中建立資料表，以及載入程式碼範例的資料](#) 會顯示產生的資料表結構和資料。step-by-step 如需在 Visual Studio 中執行此程式碼的指示，請參閱[.NET 程式碼範例](#)。

```
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);
 }
}
}
```

## 使用 AWS SDK for Python (Boto) 的 DynamoDB 範例應用程式：井字遊戲

### 主題

- [步驟 1：在本機上部署及測試](#)
- [步驟 2：檢查資料模型和實作詳細資訊](#)
- [步驟 3：使用 DynamoDB 服務在生產環境中部署](#)
- [步驟 4：清除資源](#)

井字遊戲是建置在 Amazon DynamoDB 上的範例 Web 應用程式。該應用程式使用 AWS SDK for Python (Boto) 發出必要的 DynamoDB 呼叫，在 DynamoDB 資料表中存放遊戲資料，及使用 Python 的 Web 框架 Flask 示範 DynamoDB 中的端對端應用程式開發，包含如何為資料建立模型。它也示範

在 DynamoDB 中為資料建立模型的最佳實務，包含您為遊戲應用程式建立的資料表、您定義的主索引鍵、根據您的查詢需求所需的其他索引，以及使用串連值屬性。

您在 Web 上玩井字遊戲應用程式的程序如下所示：

1. 您要登入應用程式首頁。
2. 接著邀請另一位使用者做為您的對手玩遊戲。

在另一位使用者接受您的邀請前，遊戲狀態都會維持在 PENDING。對手接受邀請後，遊戲狀態會變更為 IN\_PROGRESS。

3. 遊戲會在對手登入並接受邀請後開始。
4. 應用程式會將所有遊戲的移動和狀態資訊存放在 DynamoDB 資料表。
5. 遊戲會以獲勝或平手結束，這會將遊戲狀態設為 FINISHED。

我們會依步驟說明端對端應用程式的建置練習：

- [步驟 1：在本機上部署及測試](#)：在本節中，您會在您的本機電腦上下載、部署及測試應用程式。您會在可下載版 DynamoDB 中建立必要的資料表。
- [步驟 2：檢查資料模型和實作詳細資訊](#)：本節會先詳細說明資料模型，包含索引和使用串連值屬性。接著會說明應用程式運作的方式。
- [步驟 3：使用 DynamoDB 服務在生產環境中部署](#)：本節重點為生產環境中的部署考量。在此步驟中，您會使用 Amazon DynamoDB 服務建立資料表，並使用 AWS Elastic Beanstalk 部署應用程式。當您在生產環境中部署該應用程式時，您還要授予適當的許可，讓應用程式可存取 DynamoDB 資料表。本節說明會帶您演練端對端生產部署。
- [步驟 4：清除資源](#)：本節重點說明此範例未涵蓋的部分。同時也會提供步驟，讓您移除在先前步驟中建立的 AWS 資源，以免產生任何費用。

## 步驟 1：在本機上部署及測試

### 主題

- [1.1：下載並安裝必要套件](#)
- [1.2：測試遊戲應用程式](#)

在此步驟中您會在您的本機電腦上下載、部署及測試井字遊戲應用程式。您並不會使用 Amazon DynamoDB Web 服務，而是將 DynamoDB 下載至您的電腦，並在該位置建立必要的資料表。

## 1.1：下載並安裝必要套件

若要在本機上測試此應用程式，您需要下列項目：

- Python
- Flask (Python 的微框架)
- AWS SDK for Python (Boto)
- 在您電腦上執行的 DynamoDB
- Git

若要取得這些工具，請執行下列作業：

1. 安裝 Python。如需逐步說明，請參閱 [Download Python](#)。

井字遊戲應用程式已使用 Python 2.7 版測試。

2. 使用 Python Package Installer (PIP) 安裝 Flask 和 AWS SDK for Python (Boto)：

- 安裝 PIP。

如需說明，請參閱 [Install PIP](#)。在安裝頁面上，選擇 get-pip.py 連結，然後儲存檔案。接著以管理員身分開啟命令終端機，在命令提示中輸入如下內容。

```
python.exe get-pip.py
```

在 Linux 上，您不需要指定 .exe 副檔名。只要指定 python get-pip.py 就可以了。

- 利用 PIP 使用下列程式碼安裝 Flask 和 Boto 套件。

```
pip install Flask
pip install boto
pip install configparser
```

3. 將 DynamoDB 下載到您的電腦。如需執行方式的說明，請參閱 [設定 DynamoDB Local \(可下載版本\)](#)。
4. 下載井字遊戲應用程式：
  - a. 安裝 Git。如需說明，請參閱 [git Downloads](#)。
  - b. 執行下列程式碼以下載應用程式。

```
git clone https://github.com/awslabs/dynamodb-tictactoe-example-app.git
```

## 1.2：測試遊戲應用程式

若要測試井字遊戲應用程式，您需要在您的電腦本機上執行 DynamoDB。

執行井字遊戲應用程式

1. 啟動 DynamoDB。
2. 啟動井字遊戲應用程式的 Web 伺服器。

若要執行此作業，請開啟命令終端機，導覽至您下載井字遊戲應用程式的所在資料夾，並使用下列程式碼在本機上執行應用程式。

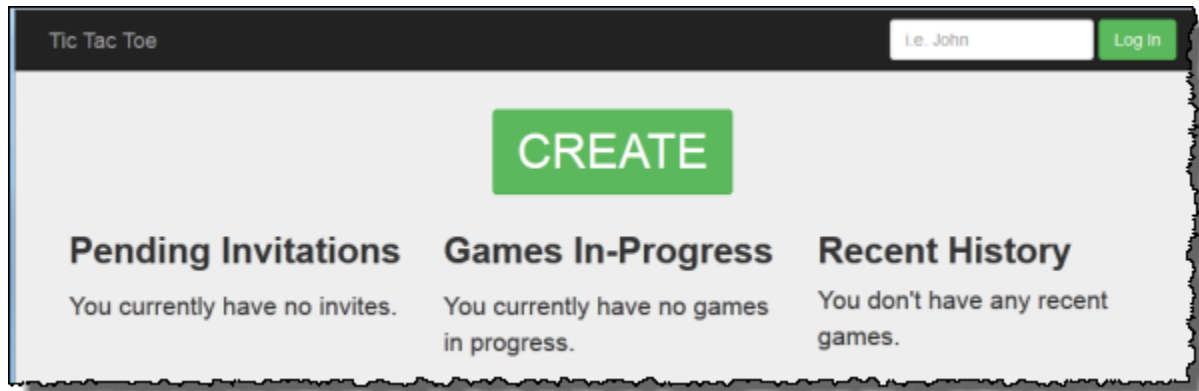
```
python.exe application.py --mode local --serverPort 5000 --port 8000
```

在 Linux 上，您不需要指定 .exe 副檔名。

3. 開啟您的 Web 瀏覽器，然後輸入如下內容。

```
http://localhost:5000/
```

瀏覽器會顯示出首頁。



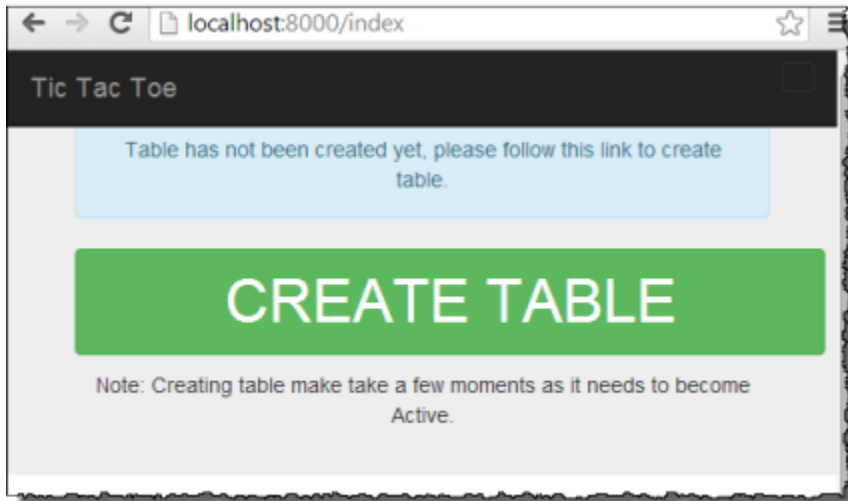
4. 在 Log in (登入) 方塊中輸入 **user1**，以 user1 身分登入。



**Note**

此範例應用程式不會執行任何使用者身分驗證。使用者 ID 僅會用於識別玩家。若兩名玩家使用同一個別名登入，應用程式仍會運作，如同您在兩個不同的瀏覽器中玩遊戲一樣。

5. 若這是您第一次玩遊戲，則會顯示一個頁面，要求您在 DynamoDB 中建立必要的資料表 (Games)。選擇 CREATE TABLE (建立資料表)。



6. 選擇 CREATE (建立) 建立第一個井字遊戲。
7. 在 Choose an Opponent (選擇對手) 方塊中輸入 **user2**，然後選擇 Create Game! (建立遊戲！)



執行此作業會在 Games 資料表中新增項目，而建立遊戲。它會將遊戲狀態設為 PENDING。

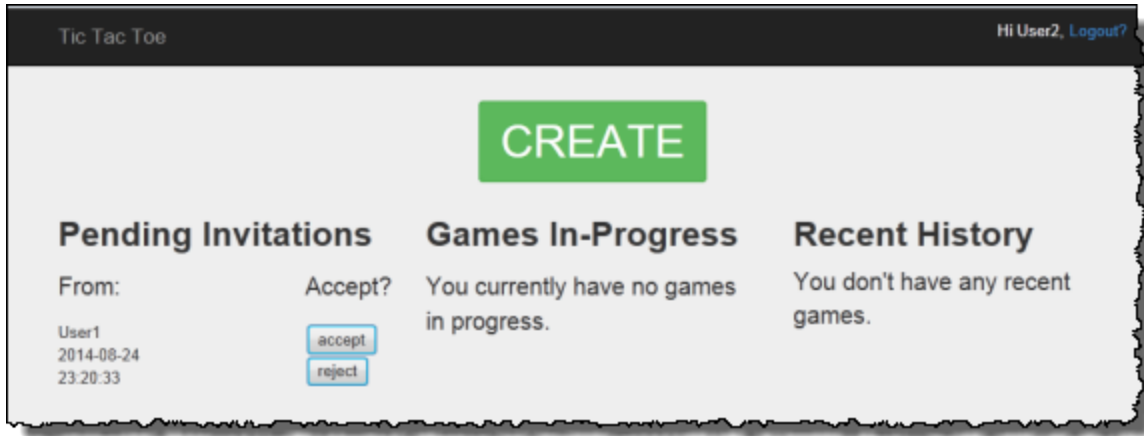
8. 開啟另一個瀏覽器視窗，並輸入如下內容。

`http://localhost:5000/`

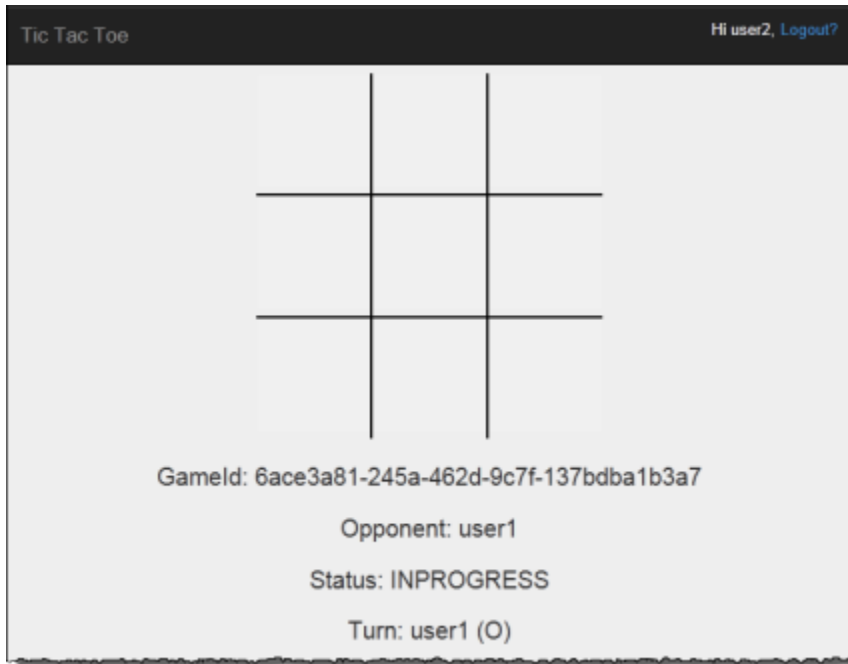
瀏覽器會透過 Cookie 傳遞資訊，因此建議您使用無痕 (incognito) 模式或隱私瀏覽，以免留下您的 Cookie。

9. 以 user2 登入。

此時將出現一個頁面，顯示 user1 發送的待定邀請。



10. 選擇 accept (接受) 接受邀請。



遊戲頁面會隨即顯示，其中有一個空白的井字遊戲方格。該頁面也會顯示相關的遊戲資訊，例如遊戲 ID、目前輪到哪一方及遊戲狀態。

11. 玩遊戲。

使用者每移動一步，Web 服務就會傳送請求到 DynamoDB，以條件式更新 Games 資料表中的遊戲項目。例如，條件會確保移動有效、方格可供使用者選擇，以及輪到該使用者移動。若移動有效，更新操作會為棋盤上的選擇新增對應屬性。更新操作也會將現有屬性的值設為可移動下一步的使用者。

在遊戲頁面上，應用程式每秒皆會發出非同步的 JavaScript 呼叫，最多達 5 分鐘，以檢查 DynamoDB 中的遊戲狀態是否已變更。若狀態已變更，應用程式會以新的資訊更新頁面。5 分鐘後，應用程式會停止發出請求，您必須重新整理頁面才能取得更新資訊。

## 步驟 2：檢查資料模型和實作詳細資訊

### 主題

- [2.1：基本資料模型](#)
- [2.2：應用程式的實際運作 \(程式碼演練\)](#)

### 2.1：基本資料模型

此範例應用程式的 DynamoDB 資料模型概念重點如下：

- **Table (資料表)：**在 DynamoDB 中，資料表為項目 (即紀錄) 的集合，而每一個項目都是名稱/值對的集合，稱為屬性。

在此井字遊戲範例中，應用程式會將所有遊戲資料存放在 Games 資料表。應用程式會為每一場遊戲在資料表中建立一個項目，並將所有遊戲資料以屬性的形式存放。一場井字遊戲最多可移動九步。因為 DynamoDB 資料表沒有結構描述，在必要屬性僅為主索引鍵的情況下，應用程式可針對每個遊戲項目存放不同數目的屬性。

Games 資料表具有由單一屬性 GameId 組成的簡易主索引鍵，其類型為字串。該應用程式會為每一場遊戲指派唯一的 ID。如需 DynamoDB 主索引鍵的詳細資訊，請參閱 [主索引鍵](#)。

當使用者透過邀請另一名使用者玩遊戲，來起始井字遊戲時，應用程式會在 Games 資料表中，以存放遊戲中繼資料的屬性建立新的項目，如下列範例：

- HostId，起始遊戲的使用者。
- Opponent，獲邀玩遊戲的使用者。
- 輪到玩遊戲的使用者。起始遊戲的使用者先開始玩遊戲。
- 在棋盤上使用 O 符號的使用者。起始遊戲的使用者使用 O 符號。

此外，應用程式也會建立 StatusDate 串連屬性，將初始遊戲狀態標記為 PENDING。下列螢幕擷取畫面顯示範例項目在 DynamoDB 主控台內的樣子：

Attribute	Type	Value
GameId (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"

隨著遊戲的進行，應用程式會為遊戲中的每步移動，新增一個屬性到資料表。屬性名稱為棋盤位置，例如 TopLeft 或 BottomRight。例如，移動可能是值為 TopLeft 的 O 屬性、值為 TopRight 的 O 屬性，及值為 BottomRight 的 X 屬性。屬性值不是 O 就是 X，取決於移動的使用者。例如，請參閱下列棋盤。

You Tie		
O	X	O
O	O	X
X	O	X

GameId: 5ca60639-bef9-4c03-83e9-bb7abe4debca  
Opponent: user2  
Status: FINISHED  
Turn: N/A

- Concatenated value attributes (串連值屬性)：StatusDate 屬性為串連值屬性的範例。在此方法中，並非透過建立個別屬性來存放遊戲狀態 (PENDING、IN\_PROGRESS 和 FINISHED) 及日期 (最後一步的移動時間)，而是將其合併為單一屬性，例如 IN\_PROGRESS\_2014-04-30 10:20:32。

然後應用程式會使用 StatusDate 屬性，將 StatusDate 指定為索引的排序索引鍵，來建立次要索引。使用 StatusDate 串連值屬性的好處會在接下來討論的索引中進一步說明。

- Global secondary indexes (全域次要索引)：您可以使用資料表的主索引鍵 (GameId) 有效率地查詢資料表，來尋找遊戲項目。若要查詢資料表上主索引鍵屬性以外的屬性，DynamoDB 支援建立次要索引。在此範例應用程式中，您會建置下列兩個次要索引：

Local Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Index Size (Bytes)*	Item Count*
This table has no local secondary indexes.					

Global Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Status	Read Capacity Units	Write Capacity Units	Last Decrease Time	Last Increase Time	Index Size (Bytes)*	Item Count*
hostStatusDate	HostId (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125
oppStatusDate	Opponent (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125

- HostId-StatusDate-index。此索引的分割區索引鍵為 HostId，排序索引鍵為 StatusDate。您可以使用此索引查詢 HostId，例如尋找由特定使用者發起的遊戲。
- OpponentId-StatusDate-index。此索引的分割區索引鍵為 OpponentId，排序索引鍵為 StatusDate。您可以使用此索引查詢 Opponent，例如尋找對手為特定使用者的遊戲。

這些索引稱為全域次要索引，因為這些索引中的分割區索引鍵與資料表主索引鍵所使用的分割區索引鍵 (GameId) 不同。

請注意，這兩種索引皆指定 StatusDate 做為排序索引鍵。這可讓您進行下列作業：

- 您可以使用 `BEGINS_WITH` 比較運算子查詢。例如，您可以尋找由特定使用者發起且屬性為 `IN_PROGRESS` 的所有遊戲。在此案例中，`BEGINS_WITH` 運算子會檢查開頭為 StatusDate 的 `IN_PROGRESS` 值。
- DynamoDB 會根據排序索引鍵值，將項目依排序存放於索引中。因此若所有狀態字首都相同 (例如 `IN_PROGRESS`)，日期部分所使用的 ISO 格式會將項目從最舊排序到最新。此方法可讓特定查詢有效率地執行，例如下列查詢：
  - 擷取最多 10 個最近由登入使用者發起的 `IN_PROGRESS` 遊戲。針對此查詢，您要指定 `HostId-StatusDate-index` 索引。
  - 擷取最多 10 個最近登入使用者為對手的 `IN_PROGRESS` 遊戲。針對此查詢，您要指定 `OpponentId-StatusDate-index` 索引。

如需次要索引的詳細資訊，請參閱「[使用次要索引來改善資料存取](#)」。

## 2.2：應用程式的實際運作 (程式碼演練)

此應用程式有兩個主要頁面：

- Home page (首頁)：此頁面為使用者提供簡易登入、可建立新井字遊戲的 CREATE (建立) 按鈕、正在進行的遊戲清單、遊戲歷史記錄，以及任何作用中的待定遊戲邀請。

首頁不會自動重新整理，您必須重新整理頁面才能重新整理清單。

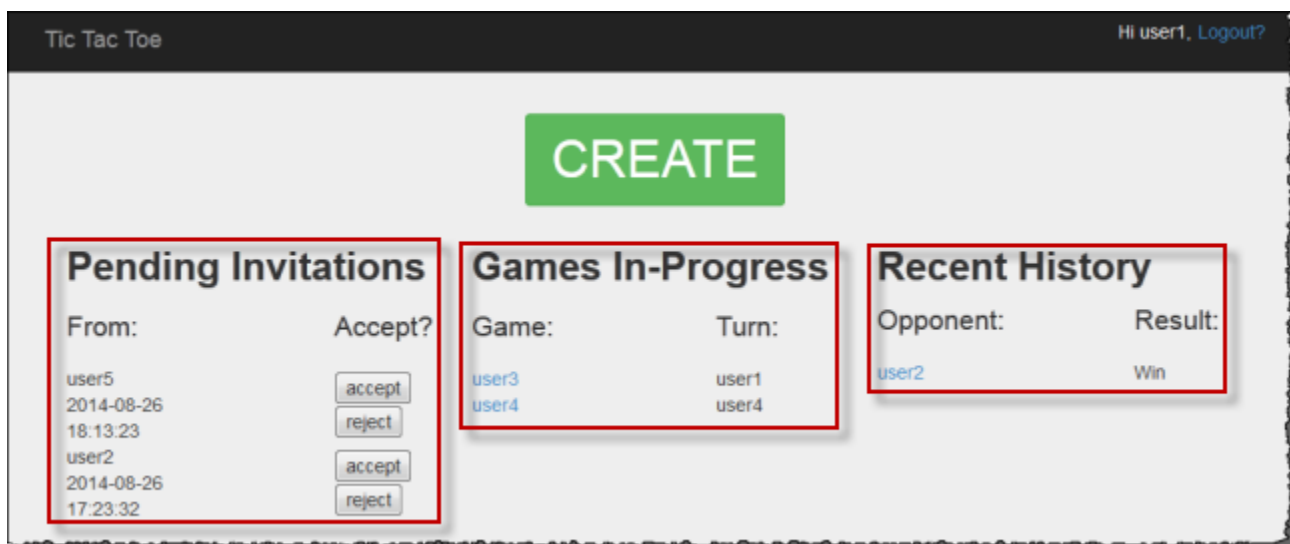
- Game page (遊戲頁面)：此頁面會顯示使用者玩的井字遊戲方格。

應用程式會每秒自動更新遊戲頁面一次。您瀏覽器中的 JavaScript 會每秒呼叫 Python Web 伺服器一次，查詢 Games 資料表以檢查資料表中的遊戲項目是否已變更。若已變更，JavaScript 會觸發頁面重新整理，讓使用者可看到更新的棋盤。

讓我們了解應用程式的詳細運作方式。

## 首頁

在使用者登入後，應用程式會顯示下列三項資訊清單。



- Invitations (邀請)：此清單會顯示最多 10 則最近其他使用者發送並正等待登入使用者接受的邀請。在先前的螢幕擷取畫面中，user1 有來自 user5 和 user2 的待定邀請。
- Games In-Progress (正在進行的遊戲) - 此清單會顯示最多 10 個最近正在進行的遊戲。這些遊戲是使用者正在玩的遊戲，其狀態為 IN\_PROGRESS。在螢幕擷取畫面中，user1 正在與 user3 和 user4 玩井字遊戲。
- Recent History (近期歷史記錄) - 此清單會顯示最多 10 個最近使用者完成的遊戲，其狀態為 FINISHED。在螢幕擷取畫面中顯示的遊戲裡，user1 先前與 user2 玩過遊戲。清單會為每個完成的遊戲顯示遊戲結果。

在程式碼中，`index` 函數 (位於 `application.py` 中) 會發出下列三個呼叫，來擷取遊戲狀態資訊：

```
inviteGames = controller.getGameInvites(session["username"])
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames = controller.getGamesWithStatus(session["username"], "FINISHED")
```

這些呼叫各會從 DynamoDB 傳回 Game 物件所包裝的項目清單。在檢視中從這些物件擷取資料非常容易。`index` 函數會將這些物件清單傳遞到檢視，以轉譯 HTML。

```
return render_template("index.html",
 user=session["username"],
 invites=inviteGames,
 inprogress=inProgressGames,
 finished=finishedGames)
```

井字遊戲應用程式會定義 Game 類別，主要用以存放從 DynamoDB 擷取的遊戲資料。這些函數會傳回 Game 物件的清單，讓您可將應用程式剩餘的部分與有關 Amazon DynamoDB 項目的程式碼隔離。因此，這些函數可協助您將應用程式程式碼和資料存放層的詳細資訊分離。

此處說明的架構模式又稱為模型-檢視-控制器 (MVC) UI 模式。在此案例中，Game 物件執行個體 (代表資料) 為模型，而 HTML 頁面為檢視。控制器則分為兩個檔案。`application.py` 檔案具有 Flask 框架的控制器邏輯，商業邏輯則隔離在 `gameController.py` 檔案中。意即應用程式會將與 DynamoDB 開發套件有關的所有事物存放在 `dynamodb` 資料夾中其自身的個別檔案裡。

讓我們檢閱三個函數，及其如何使用全域次要索引查詢 Games 資料表，來擷取相關資料。

使用 `getGameInvites` 擷取待定遊戲邀請的清單

`getGameInvites` 函數會擷取 10 個最近待定邀請的清單。這些遊戲已由使用者建立，但對手尚未接受遊戲邀請。在對手接受邀請前，這些遊戲皆會維持在 PENDING 狀態。若對手拒絕邀請，應用程式會從資料表移除對應項目。

函數指定查詢的方式如下所示：

- 其指定 `OpponentId-StatusDate-index` 索引，以搭配下列索引鍵值和比較運算子使用：
  - 分割區索引鍵為 `OpponentId`，並使用索引鍵 *user ID*。
  - 排序索引鍵為 `StatusDate`，並使用比較運算子和索引鍵值 `beginswith="PENDING_"`。

您可以使用 `OpponentId-StatusDate-index` 索引擷取登入使用者獲邀的遊戲，即對手為登入使用者的遊戲。



- 查詢會將結果限制為 10 個項目。

```
gameInvitesIndex = self.cm.getGamesTable().query(
 Opponent__eq=user,
 StatusDate__beginswith="PENDING_",
 index="OpponentId-StatusDate-index",
 limit=10)
```

在索引中，針對每個 OpponentId (分割區索引鍵)，DynamoDB 都會依 StatusDate (排序索引鍵) 讓項目保持排序。因此，查詢傳回的遊戲將會是 10 個最近的遊戲。

使用 `getGamesWithStatus` 擷取特定狀態的遊戲清單

對手接受遊戲邀請後，遊戲狀態會變更為 `IN_PROGRESS`。遊戲完成後，該狀態會變更為 `FINISHED`。

尋找正在進行之遊戲的查詢與尋找已完成遊戲的查詢皆相同，唯一相異點在於狀態值不同。因此，應用程式會定義 `getGamesWithStatus` 函數，其使用狀態值做為參數。

```
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames = controller.getGamesWithStatus(session["username"], "FINISHED")
```

下節會討論正在進行的遊戲，但該說明也同樣適用於已完成的遊戲。

指定使用者正在進行的遊戲清單同時包含下列項目：

- 由使用者發起之正在進行的遊戲
- 對手為該使用者之正在進行的遊戲

`getGamesWithStatus` 函數會執行下列兩項查詢，每次都會使用適當的次要索引。

- 函數會使用 `HostId-StatusDate-index` 索引查詢 Games 資料表。針對該索引，查詢會指定主索引鍵值：即分割區索引鍵 (`HostId`) 和排序索引鍵 (`StatusDate`) 兩者的值，以及比較運算子。

```
hostGamesInProgress = self.cm.getGamesTable ().query(HostId__eq=user,
 StatusDate__beginswith=status,
 index="HostId-StatusDate-index",
 limit=10)
```

請注意比較運算子的 Python 語法：



- `HostId__eq=user` 會指定對等比較運算子。
- `StatusDate__beginswith=status` 會指定 `BEGINS_WITH` 比較運算子。
- 函數會使用 `OpponentId-StatusDate-index` 索引查詢 `Games` 資料表。

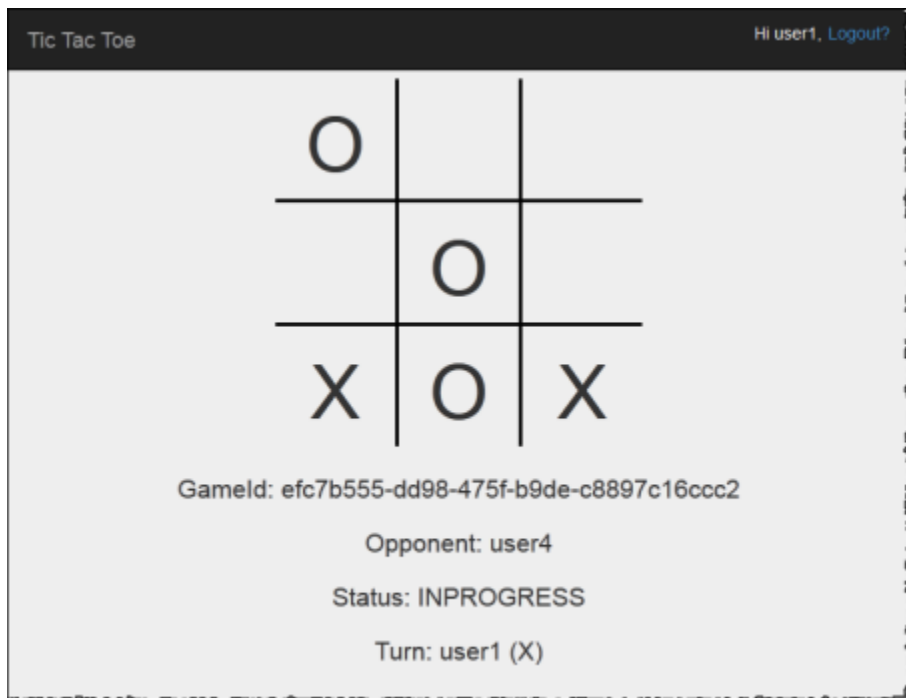
```
oppGamesInProgress = self.cm.getGamesTable().query(Opponent__eq=user,
 StatusDate__beginswith=status,
 index="OpponentId-StatusDate-index",
 limit=10)
```

- 函數接著會合併兩個清單、排序，然後為前 0 到 10 個項目建立 `Game` 物件的清單，並將清單傳回呼叫函數 (即 `index`)。

```
games = self.mergeQueries(hostGamesInProgress,
 oppGamesInProgress)
return games
```

## 遊戲頁面

遊戲頁面是使用者玩井字遊戲的位置。它會顯示遊戲方格和有關遊戲的資訊。下列螢幕擷取畫面顯示正在進行的範例遊戲：



應用程式會在下列情況中顯示遊戲頁面：

- 使用者建立遊戲，並邀請另一名使用者玩遊戲。

在此案例中，頁面會顯示使用者為發起人，且遊戲狀態為 PENDING，等待對手接受。

- 使用者在首頁上接受其中一個待定邀請。

在此案例中，頁面會顯示使用者為對手，且遊戲狀態為 IN\_PROGRESS。

使用者在棋盤上的選擇會產生對應用程式的表單 POST 請求。意即，Flask 會使用 HTML 表單資料呼叫 `selectSquare` 函數 (位於 `application.py` 中)。然後此函數會呼叫 `updateBoardAndTurn` 函數 (位於 `gameController.py` 中) 來更新遊戲項目，如下所示：

- 它會新增移動專屬的屬性。
- 它會將 Turn 屬性值更新為移動下一步的使用者。

```
controller.updateBoardAndTurn(item, value, session["username"])
```

如果項目更新成功，函數會傳回 `true`，否則傳回 `false`。請注意下列關於 `updateBoardAndTurn` 函數的事項：

- 函數會呼叫 SDK for Python 的 `update_item` 函數，對現有項目進行有限次數的更新。函數會映射到 DynamoDB 中的 `UpdateItem` 操作。如需詳細資訊，請參閱 [UpdateItem](#)。

#### Note

`UpdateItem` 和 `PutItem` 操作之間的差異在於 `PutItem` 會取代整個項目。如需詳細資訊，請參閱 [PutItem](#)。

在 `update_item` 呼叫方面，程式碼會識別下列項目：

- Games 資料表的主索引鍵 (即 `ItemId`)。

```
key = { "GameId" : { "S" : gameId } }
```

- 要新增的屬性 (目前使用者移動專屬的屬性) 及其值 (例如 `TopLeft="X"`)。

```
attributeUpdates = {
 position : {
```

```
 "Action" : "PUT",
 "Value" : { "S" : representation }
 }
}
```

- 條件必須為 true 才能更新：
  - 遊戲必須正在進行。即 StatusDate 屬性值的開頭必須為 IN\_PROGRESS。
  - 目前輪次必須是 Turn 屬性指定的有效使用者輪次。
  - 方格必須可供使用者選擇。即對應到該方格的屬性不得存在。

```
expectations = {"StatusDate" : {"AttributeValueList": [{"S" : "IN_PROGRESS_"}],
 "ComparisonOperator": "BEGINS_WITH"},
 "Turn" : {"Value" : {"S" : current_player}},
 position : {"Exists" : False}}
```

函數現在會呼叫 update\_item 更新項目。

```
self.cm.db.update_item("Games", key=key,
 attribute_updates=attributeUpdates,
 expected=expectations)
```

函數傳回後，selectSquare 函數會呼叫 redirect，如下列範例所示。

```
redirect("/game="+gameId)
```

此呼叫會使瀏覽器重新整理。在此重新整理中，應用程式會檢查遊戲是否以獲勝或平手結束。若遊戲已結束，則應用程式會根據結果更新遊戲項目。

## 步驟 3：使用 DynamoDB 服務在生產環境中部署

### 主題

- [3.1：建立 Amazon EC2 的 IAM 角色](#)
- [3.2：在 Amazon DynamoDB 中建立遊戲資料表](#)
- [3.3：組合及部署井字遊戲應用程式的程式碼](#)
- [3.4：設定 AWS Elastic Beanstalk 環境](#)

在先前的章節中，您已在電腦本機上使用 DynamoDB 本機版部署和測試井字遊戲應用程式。現在，您會在生產環境中部署應用程式，如下所示：

- 使用 AWS Elastic Beanstalk 部署應用程式，該服務簡單易用，可供您部署和擴展 Web 應用程式和服務。如需詳細資訊，請參閱[將 Flask 應用程式部署到 AWS Elastic Beanstalk](#)。

Elastic Beanstalk 會啟動一或多個 Amazon Elastic Compute Cloud (Amazon EC2) 執行個體，您可以透過 Elastic Beanstalk 加以設定，且您的井字遊戲應用程式也會在這些執行個體上執行。

- 使用 Amazon DynamoDB 服務，建立位於 AWS 上而非您電腦本機上的 Games 資料表。

此外，您也必須設定許可。根據預設，您建立的任何 AWS 資源 (例如 DynamoDB 中的 Games 資料表) 皆為私有。只有資源擁有者 (即建立 Games 資料表的 AWS 帳戶) 才能存取此資料表。因此根據預設，您的井字遊戲應用程式無法更新 Games 資料表。

若要授予必要許可，請建立 AWS Identity and Access Management (IAM) 角色並授予此角色存取 Games 資料表的許可。您的 Amazon EC2 執行個體首先會擔任此角色。AWS 會在回應中傳回暫時安全登入資料，Amazon EC2 執行個體可用來代替井字遊戲應用程式更新 Games 資料表。當您設定 Elastic Beanstalk 應用程式時，您可以指定 Amazon EC2 執行個體可以擔任的 IAM 角色。如需 IAM 角色的詳細資訊，請參閱《Amazon EC2 Linux 執行個體使用者指南》中的[Amazon EC2 適用的 IAM 角色](#)。

#### Note

在您為井字遊戲應用程式建立 Amazon EC2 執行個體前，必須先決定您希望 Elastic Beanstalk 建立執行個體的 AWS 區域。在您建立 Elastic Beanstalk 應用程式後，您要在組態檔案中提供同一個區域名稱和端點。井字遊戲應用程式會使用此檔案中的資訊，來建立 Games 資料表，並在特定的 AWS 區域中傳送後續請求。DynamoDB Games 資料表和 Elastic Beanstalk 啟動的 Amazon EC2 執行個體皆必須位於同一個區域。如需可用區域清單，請參閱 Amazon Web Services 一般參考 中的[Amazon DynamoDB](#)。

總而言之，您會執行下列作業，以在生產環境中部署井字遊戲應用程式：

1. 使用 IAM 服務建立 IAM 角色。您會將政策連接到此角色，授予 DynamoDB 動作的許可，使其可存取 Games 資料表。
2. 將井字遊戲應用程式的程式碼和組態檔案組合，並建立 .zip 檔案。您會使用這個 .zip 檔案，將井字遊戲應用程式的程式碼提供給 Elastic Beanstalk，以放置在您的伺服器上。如需建立套件的詳細資訊，請參閱《AWS Elastic Beanstalk 開發人員指南》中的[建立應用程式來源套件](#)。

在組態檔案 (beanstalk.config) 中，您要提供 AWS 區域和端點的資訊。井字遊戲應用程式會使用此資訊判斷要通訊的 DynamoDB 區域。

3. 設定 Elastic Beanstalk 環境。Elastic Beanstalk 會啟動一或多個 Amazon EC2 執行個體，並在上部署您的井字遊戲應用程式套件。Elastic Beanstalk 環境就緒後，您要新增 CONFIG\_FILE 環境變數，來提供組態檔案名稱。
4. 建立 DynamoDB 資料表。使用 Amazon DynamoDB 服務，您會在 AWS 上建立 Games 資料表，而非在您的電腦本機上。請記得，此資料表具有由字串類型之 GameId 分割區索引鍵組成的簡易主索引鍵。
5. 在生產環境中測試遊戲。

### 3.1：建立 Amazon EC2 的 IAM 角色

建立 Amazon EC2 類型的 IAM 角色，可讓執行您井字遊戲應用程式的 Amazon EC2 執行個體擔任正確的角色，並發出應用程式請求，來存取 Games 資料表。當您建立角色時，請選擇 Custom Policy (自訂政策) 選項，並將下列政策複製並貼上。

```
{
 "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/*"
]
 }
]
}
```

如需進一步說明，請參閱《IAM 使用者指南》中的[為 AWS 服務 \(AWS Management Console\) 建立角色](#)。

## 3.2：在 Amazon DynamoDB 中建立遊戲資料表

DynamoDB 中的 Games 資料表會存放遊戲資料。若沒有該資料表，應用程式會為您建立資料表。在此案例中，我們會讓應用程式建立 Games 資料表。

## 3.3：組合及部署井字遊戲應用程式的程式碼

若您遵循此範例的步驟，則您已有下載完成的井字遊戲應用程式。若沒有，請下載應用程式，並將所有檔案解壓縮到您本機電腦上的資料夾。如需指示，請參閱[步驟 1：在本機上部署及測試](#)。

當您解壓縮所有檔案後，您會有一個 code 資料夾。若要將此資料夾傳遞給 Elastic Beanstalk，您要將此資料夾的內容組合成 .zip 檔案。首先，請在該資料夾新增組態檔案。您的應用程式會使用區域和端點的資訊在指定區域中建立 DynamoDB 資料表，並使用指定端點發出後續的資料表操作請求。

1. 切換到您下載井字遊戲應用程式的所在資料夾。
2. 在應用程式的根資料夾中，使用下列內容建立名為 `beanstalk.config` 的文字檔案。

```
[dynamodb]
region=<AWS region>
endpoint=<DynamoDB endpoint>
```

例如，您可以使用下列內容。

```
[dynamodb]
region=us-west-2
endpoint=dynamodb.us-west-2.amazonaws.com
```

如需可用區域清單，請參閱《Amazon Web Services 一般參考》中的[Amazon DynamoDB](#)。

### Important

組態檔案中指定的區域為井字遊戲應用程式在 DynamoDB 中建立 Games 資料表的位置。您必須在同一個區域中建立下一節討論的 Elastic Beanstalk 應用程式。

**Note**

當您建立 Elastic Beanstalk 應用程式時，您要請求啟動您可選擇類型的環境。若要測試井字遊戲範例應用程式，您可以選擇 Single Instance (單一執行個體) 環境類型，並跳過下列步驟，進行下一步。

但是，Load balancing, autoscaling (負載平衡、自動調整規模) 環境類型可提供具高可用性且具可擴展性的環境，您應在建立和部署其他應用程式時加以考慮。若您選擇此環境類型，您還要產生 UUID，並將其新增到組態檔案，如下所示。

```
[dynamodb]
region=us-west-2
endpoint=dynamodb.us-west-2.amazonaws.com
[flask]
secret_key= 284e784d-1a25-4a19-92bf-8eeb7a9example
```

在用戶端與伺服器端的通訊中，當伺服器傳送回應時，基於安全性考量，伺服器會傳送簽署過的 Cookie，讓用戶端在下次請求時傳回伺服器。當只有一個伺服器時，伺服器可在啟動時於本機產生加密金鑰。當有多個伺服器時，他們都必須知道同一個加密金鑰，否則無法讀取對等伺服器設定的 Cookie。將 secret\_key 新增到組態檔案，可以告知所有伺服器使用此加密金鑰。

3. 壓縮應用程式根資料夾的內容 (其中包含 beanstalk.config 檔案)，例如 TicTacToe.zip。
4. 將 .zip 檔案檔案上傳到 Amazon Simple Storage Service (Amazon S3) 儲存貯體。在下一節中，您會將此 .zip 檔案提供給 Elastic Beanstalk，以上傳到伺服器。

如需如何上傳檔案至 Amazon S3 儲存貯體的說明，請參閱《Amazon Simple Storage Service 使用者指南》中的[建立儲存貯體](#)及[新增物件至儲存貯體](#)。

### 3.4 : 設定 AWS Elastic Beanstalk 環境

在此步驟中，您會建立 Elastic Beanstalk 應用程式，其為包含環境的元件集合。針對此範例，您會啟動一個 Amazon EC2 執行個體，以部署並執行您的井字遊戲應用程式。

1. 輸入下列自訂 URL 設定 Elastic Beanstalk 主控台，以設定環境。

```
https://console.aws.amazon.com/elasticbeanstalk/?region=<AWS-Region>#/
newApplication
```

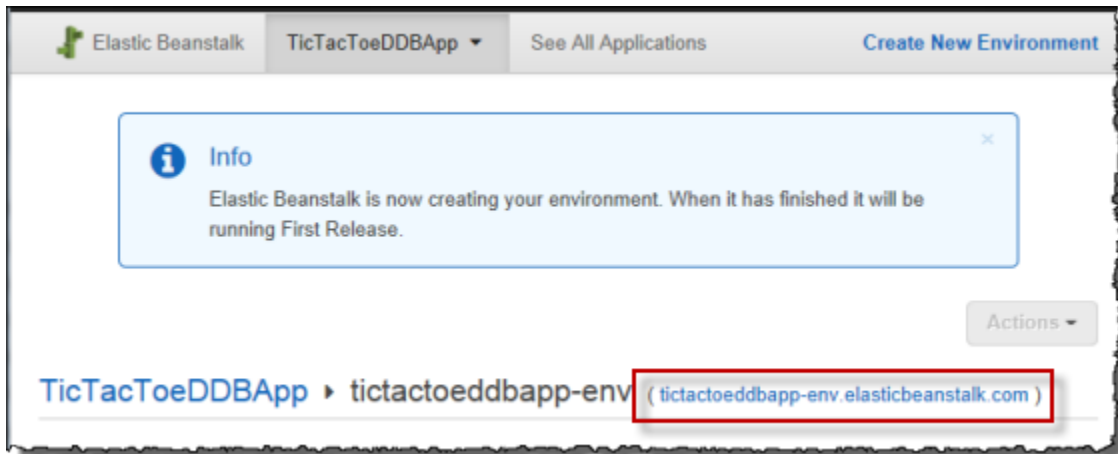
```
?applicationName=TicTacToeyour-name
&solutionStackName=Python
&sourceBundleUrl=https://s3.amazonaws.com/<bucket-name>/TicTacToe.zip
&environmentType=SingleInstance
&instanceType=t1.micro
```

如需自訂 URL 的詳細資訊，請參閱《AWS Elastic Beanstalk 開發人員指南》中的[建構立即啟動 URL](#)。在 URL 方面，請注意下列事項：

- 您必須提供 AWS 區域名稱 (與您在組態檔案中提供的名稱相同)、Amazon S3 儲存貯體名稱及物件名稱。
- 為進行測試，URL 會請求 SingleInstance 環境類型及 t1.micro 執行個體類型。
- 應用程式名稱必須是唯一的。因此，在先前的 URL 中，建議您在 applicationName 之前加上您的名稱。

執行此操作會開啟 Elastic Beanstalk 主控台。在某些案例中，您可能需要登入。

2. 在 Elastic Beanstalk 主控台中，選擇 Review and Launch (檢閱和啟動)，然後選擇 Launch (啟動)。
3. 請記下 URL，以供未來參考。此 URL 會開啟您的井字遊戲應用程式首頁。

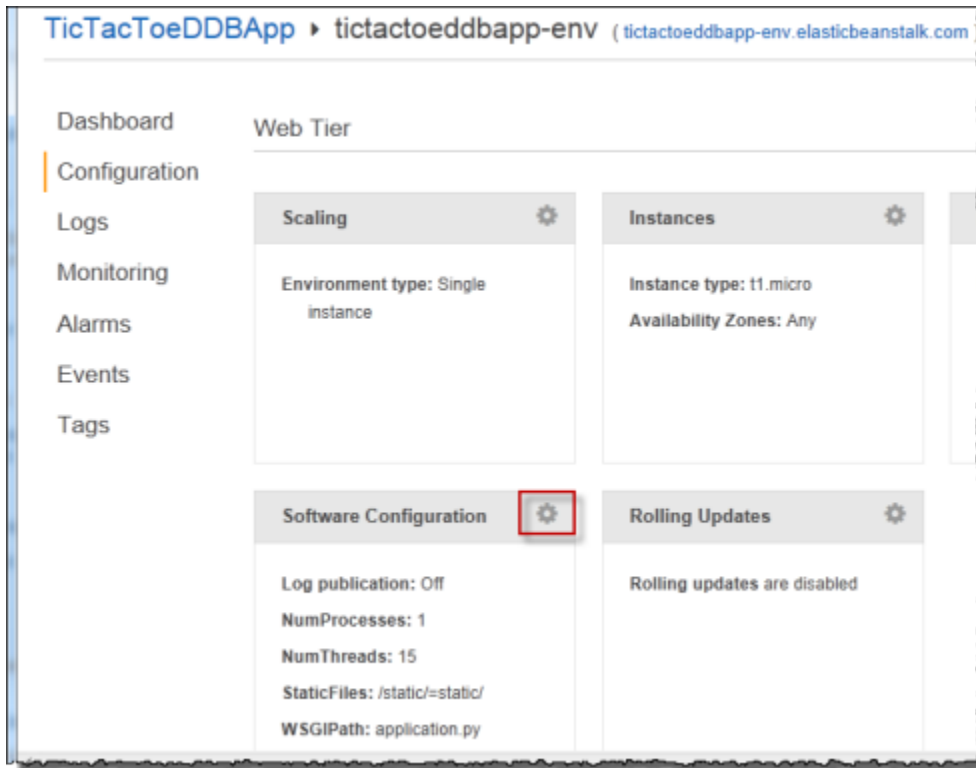


4. 設定井字遊戲應用程式，讓其可得知組態檔案的位置。

Elastic Beanstalk 建立應用程式後，請選擇 Configuration (組態)。

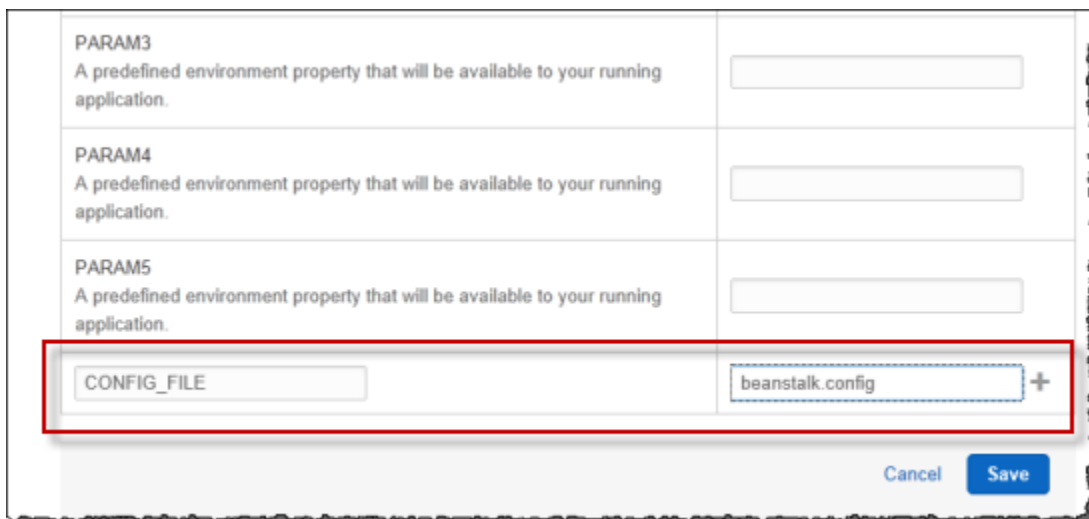
- a. 選擇 Software Configuration (軟體組態) 旁邊的齒輪圖示，如下列螢幕擷取畫面所示。





- b. 在 Environment Properties (環境屬性) 區段的結尾，輸入 **CONFIG\_FILE** 及其值 **beanstalk.config**，然後選擇 Save (儲存)。

完成環境更新需要幾分鐘的時間。

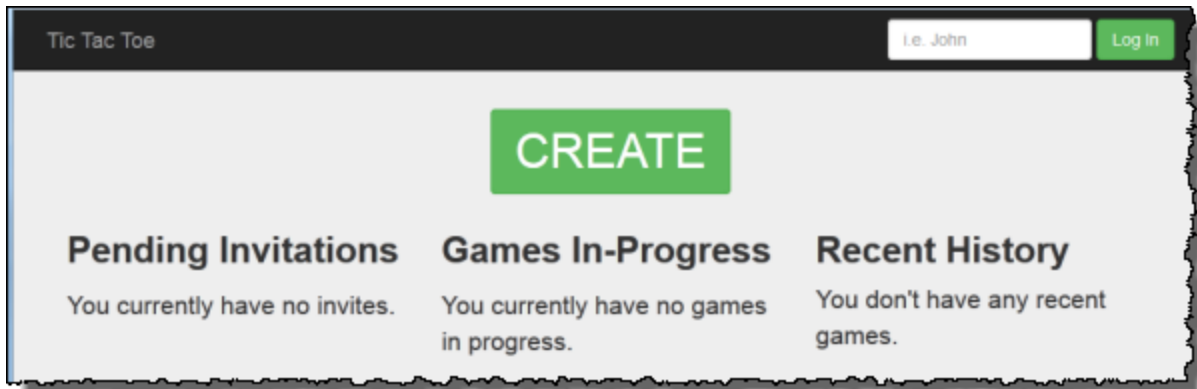


更新完成後，您便可以玩遊戲。

5. 在瀏覽器中，輸入您在先前步驟中複製的 URL，如下列範例所示。

`http://<pen-name>.elasticbeanstalk.com`

執行此作業會開啟應用程式首頁。



6. 以 `testuser1` 登入，然後選擇 **CREATE** (建立)，以開始新的井字遊戲。
7. 在 **Choose an Opponent** (選擇對手) 方塊中輸入 `testuser2`。



8. 開啟另一個瀏覽器視窗。

確認您已清除您瀏覽器視窗中的所有 Cookie，以免使用同一個使用者登入。

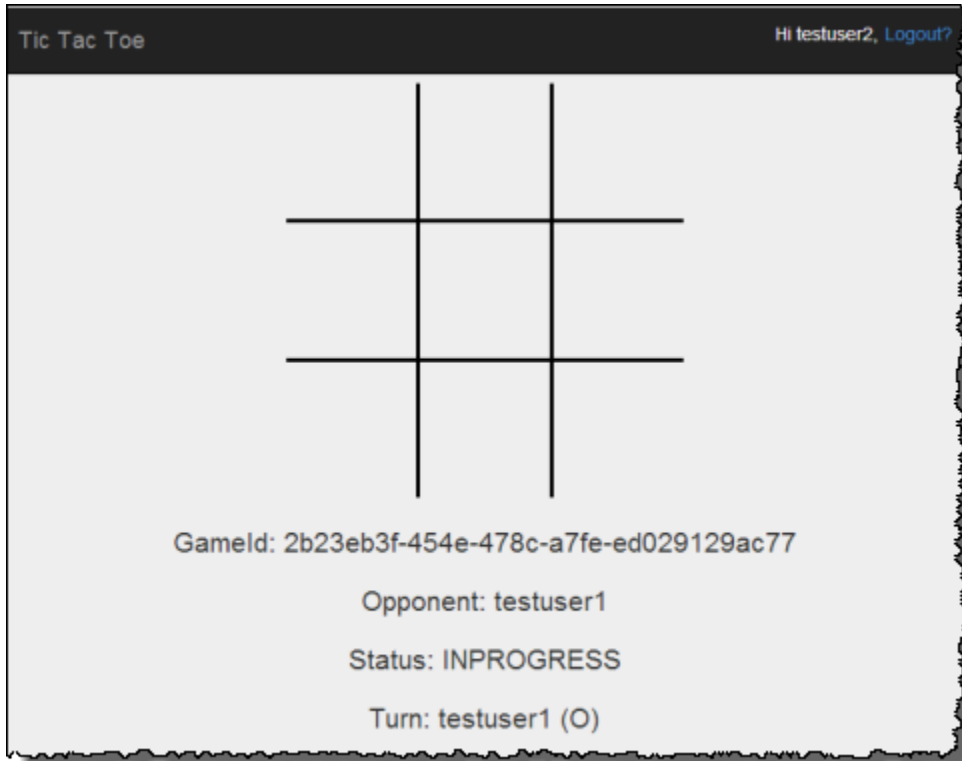
9. 輸入同一個 URL，以開啟應用程式首頁，如下列範例所示。

`http://<env-name>.elasticbeanstalk.com`

10. 以 `testuser2` 登入。
11. 針對待定邀請清單中 `testuser1` 發送的邀請，選擇 `accept` (接受)。



12. 現在遊戲頁面便會出現。



testuser1 和 testuser2 都可玩遊戲。每移動一步，應用程式就會將該移動儲存在 Games 資料表中的對應項目裡。

## 步驟 4：清除資源

現在您已完成井字遊戲應用程式的部署和測試。該應用程式涵蓋 Amazon DynamoDB 上的端對端 Web 應用程式開發，但不包括使用者身分驗證。應用程式在首頁使用登入資訊，只是為了在建立遊戲時新增玩家的名稱。在生產應用程式中，您要新增必要的程式碼，才能執行使用者登入和身分驗證。

若您已完成測試，可以移除您建立用來測試井字遊戲應用程式的資源，以免產生任何費用。

### 移除您建立的資源

1. 移除您在 DynamoDB 中建立的 Games 資料表。
2. 終止 Elastic Beanstalk 環境，以便釋放 Amazon EC2 執行個體。
3. 刪除您建立的 IAM 角色。
4. 移除您在 Amazon S3 中建立的物件。

## 使用以下方式匯出和匯入 DynamoDB 資料 AWS Data Pipeline

您可以使用 AWS Data Pipeline 將資料從 DynamoDB 表格匯出到 Amazon S3 儲存貯體中的檔案。您也可以使用主控台，將 Amazon S3 中的資料匯入至相同 AWS 區域或不同區域中的 DynamoDB 資料表。

### Note

DynamoDB 主控台現在原生支援從 Amazon S3 匯入和匯出。這些流程與 AWS Data Pipeline 匯入流程不相容。如需詳細資訊，請參閱[從 Amazon S3 匯入](#)、[從 Amazon S3 匯出](#)和部落格文章[將 Amazon DynamoDB 資料表的資料匯出至 Amazon S3 中的資料湖](#)。

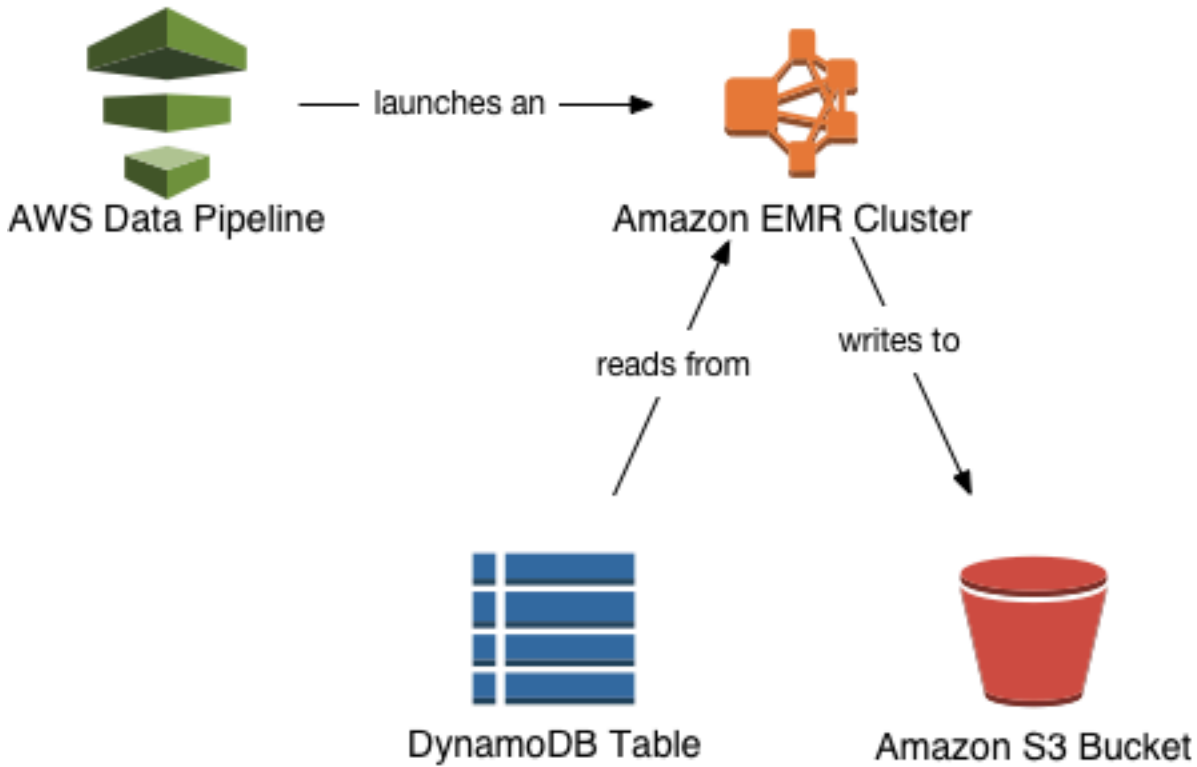
在許多情況下，匯出和匯入資料的功能都十分有用。例如，假設您要維護一組基準資料，以進行測試。您可以將基準資料放入 DynamoDB 資料表，並將它匯出至 Amazon S3。然後，在您執行可修改測試資料的應用程式之後，將 Amazon S3 中的基準匯回 DynamoDB 資料表，即可「重設」資料集。另一個範例包含意外刪除資料，甚至意外 DeleteTable 操作。在這些情況下，您可以還原 Amazon S3 中先前匯出檔案中的資料。您甚至可以複製某個 AWS 區域內 DynamoDB 資料表中的資料，並將資料存放在 Amazon S3 中，然後將 Amazon S3 的資料匯入至第二個區域內的相同 DynamoDB 資料表中。第二個區域中的應用程式接著可以存取其最近 DynamoDB 端點，並使用其專屬資料複本，而且網路延遲較短。

**⚠ Important**

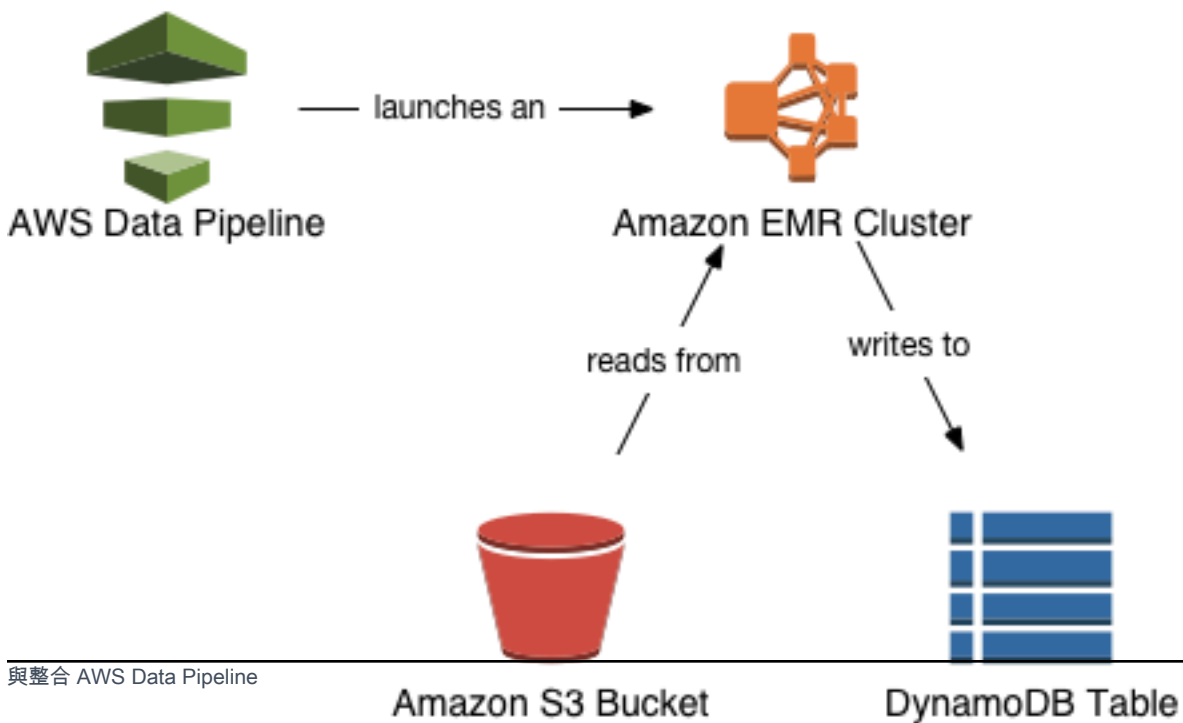
DynamoDB 備份與還原是一項全受管功能。您可以備份資料量從幾 MB 到數百 TB 不等的資料表，而且不會影響您生產應用程式的效能與可用性。只要按一下 AWS Management Console 或單一 API 呼叫，即可還原資料表。我們強烈建議您使用 DynamoDB 的原生備份和還原功能，而不要使用 AWS Data Pipeline。如需詳細資訊，請參閱 [使用 DynamoDB 的隨需備份與還原](#)。

下圖顯示使用 AWS Data Pipeline 匯出和匯入 DynamoDB 資料的概觀。

## Exporting Data from DynamoDB to Amazon S3



## Importing Data from Amazon S3 to DynamoDB



若要匯出 DynamoDB 表格，您可以使用 AWS Data Pipeline 主控台建立新的管道。管道會啟動 Amazon EMR 叢集來執行實際匯出。Amazon EMR 會讀取 DynamoDB 的資料，並將資料寫入至 Amazon S3 儲存貯體中的匯出檔案。

此步驟與匯入類似，差異在於讀取 Amazon S3 儲存貯體中的資料，並將其寫入至 DynamoDB 資料表。

#### Important

匯出或匯入 DynamoDB 資料時，使用的基礎 AWS 服務會產生額外費用：

- AWS Data Pipeline：管理匯入/匯出工作流程。
- Amazon S3：包含您從 DynamoDB 匯出或匯入至 DynamoDB 的資料。
- Amazon EMR：執行受管 Hadoop 叢集來執行 DynamoDB 與 Amazon S3 之間的讀取和寫入。叢集組態是一個 m3.xlarge 執行個體領導節點和一個 m3.xlarge 執行個體核心節點。

如需詳細資訊，請參閱 [AWS Data Pipeline 定價](#)、[Amazon EMR 定價](#)和 [Amazon S3 定價](#)。

## 匯出和匯入資料的必要條件

當您用 AWS Data Pipeline 於匯出和匯入資料時，您必須指定允許管線執行的動作，以及管線可以使用哪些資源。允許的動作和資源是使用 AWS Identity and Access Management (IAM) 角色定義的。

您也可以建立 IAM 政策，並將它們連接至使用者、角色或群組來控制存取。這些政策可讓您指定允許哪些使用者匯入和匯出 DynamoDB 資料。

#### Important

如果使用者想要與 AWS 之外的 AWS Management Console 授與程式設計存取 AWS 取權的方式取決於正在存取的使用者類型。

若要授予使用者程式設計存取權，請選擇下列其中一個選項。

哪個使用者需要程式設計存取權？	到	By
人力身分  (IAM Identity Center 中管理的使用者)	使用臨時登入資料來簽署對 AWS CLI、AWS SDK 或 AWS API 的程式設計要求。	請依照您要使用的介面所提供的指示操作。 <ul style="list-style-type: none"> <li>• 如需詳細資訊 AWS CLI，請參閱 <a href="#">《使用 AWS CLI 用 AWS Command Line Interface 者指南》</a> AWS IAM Identity Center 中的〈配置使用〉。</li> <li>• 如需 AWS SDK、工具和 AWS API，請參閱 AWS SDK 和工具參考指南中的 <a href="#">IAM 身分中心身分驗證</a>。</li> </ul>
IAM	使用臨時登入資料來簽署對 AWS CLI、AWS SDK 或 AWS API 的程式設計要求。	遵循 <a href="#">《IAM 使用者指南》</a> 中的〈將臨時登入資料搭配 <a href="#">AWS 資源</a> 使用〉中的指示
IAM	(不建議使用) 使用長期認證簽署對 AWS CLI、AWS SDK 或 AWS API 的程式設計要求。	請依照您要使用的介面所提供的指示操作。 <ul style="list-style-type: none"> <li>• 如需相關資訊 AWS CLI，請參閱使用指南中的 <a href="#">使用 IAM 使用者登入資料進行驗證</a>。AWS Command Line Interface</li> <li>• 對於 AWS SDK 和工具，請參閱 AWS SDK 和工具參考指南中的 <a href="#">使用長期憑據進行身份驗證</a>。</li> <li>• 如需 AWS API，請參閱 <a href="#">IAM 使用者指南</a> 中的 <a href="#">管理 IAM 使用者的存取金鑰</a>。</li> </ul>



## 建立 AWS Data Pipeline 的 IAM 角色

若要使用 AWS Data Pipeline，您的 AWS 帳戶中必須有以下 IAM 角色：

- DataPipelineDefaultRole— 您的管道可以代表您執行的動作。
- DataPipelineDefaultResourceRole— AWS 管道將代表您佈建的資源。為了匯出和匯入 DynamoDB 資料，這些資源會包含 Amazon EMR 叢集以及與該叢集關聯的 Amazon EC2 執行個體。

如果您以 AWS Data Pipeline 前從未使用過，則需要創建 DataPipelineDefaultRole 和 DataPipelineDefaultResourceRole 自己。建立這些角色後，您可以隨時使用它們來匯出或匯入 DynamoDB 資料。

### Note

如果您之前曾使用 AWS Data Pipeline 控制台建立管線 DataPipelineDefaultRole，則 DataPipelineDefaultResourceRole 當時已為您建立。不需要任何進一步動作；您可以跳過本節，並使用 DynamoDB 主控台開始建立管道。如需詳細資訊，請參閱 [將資料從 DynamoDB 匯出至 Amazon S3](#) 及 [從 Amazon S3 匯入資料至 DynamoDB](#)。

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 <https://console.aws.amazon.com/iam/>。
2. 從 IAM 主控台儀表板中，按一下 Roles (角色)。
3. 按一下 Create Role (建立角色) 並執行下列作業：
  - a. 在 AWS Service (AWS 服務) 信任實體中，選擇 Data Pipeline (資料管道)。
  - b. 在 Select your use case (選取您的使用案例) 面板中，選擇 Data Pipeline (資料管道)，然後選擇 Next:Permissions (下一步：許可)。
  - c. 請注意，會自動連接 AWSDataPipelineRole 政策。選擇 Next:Review (下一步：檢閱)。
  - d. 在 Role name (角色名稱) 欄位中，輸入 DataPipelineDefaultRole 做為角色名稱，然後選擇 Create role (建立角色)。
4. 按一下 Create Role (建立角色) 並執行下列作業：

- a. 在 AWS Service (AWS 服務) 信任實體中，選擇 Data Pipeline (資料管道)。
- b. 在 Select your use case (選取您的使用案例) 面板中，選擇 EC2 Role for Data Pipeline (資料管道的 EC2 角色)，然後選擇 Next:Permissions (下一步：許可)。
- c. 請注意，會自動連接 AmazonEC2RoleForDataPipelineRole 政策。選擇 Next:Review (下一步：檢閱)。
- d. 在 Role name (角色名稱) 欄位中，輸入 DataPipelineDefaultResourceRole 做為角色名稱，然後選擇 Create role (建立角色)。

既然您已建立這些角色，就可以使用 DynamoDB 主控台開始建立管道。如需詳細資訊，請參閱 [將資料從 DynamoDB 匯出至 Amazon S3](#) 及 [從 Amazon S3 匯入資料至 DynamoDB](#)。

## 授與使用者和群組執行匯出和匯入工作的權限 AWS Identity and Access Management

如果您想要允許其他使用者、角色或群組匯出和匯入您的 DynamoDB 資料表資料，則可以建立 IAM 政策，並將它連接至您指定的使用者或群組。政策只包含執行這些任務的必要許可。

### 授予完整存取權

下列程序說明如何將受 AWS 管政

策 AmazonDynamoDBFullAccessAWSDataPipeline\_FullAccess 和 Amazon EMR 內嵌政策附加到使用者。這些受管政策可提供 DynamoDB 資源的完整存取權，並與 Amazon EMR 內嵌政策搭配使用，可讓使用者執行本文件中所述的動作。AWS Data Pipeline

#### Note

若要限制建議的許可範圍，上述的內嵌政策會強制使用標籤 dynamodbdatapipeline。如果您希望在不受限的情況下使用本文件，則可以移除建議政策的 Condition 部分。

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 <https://console.aws.amazon.com/iam/>。
2. 從 IAM 主控台儀表板中，按一下 Users (使用者)，然後選取您要修改的使用者。
3. 在 Permissions (許可) 索引標籤中，按一下 Add Policy (新增政策)。
4. 在 Attach permissions (連接許可) 面板中，按一下 Attach existing policies directly (直接連接現有政策)。

5. 同時選取 AmazonDynamoDBFullAccess 和 AWSDataPipeline\_FullAccess，然後按一下 Next:Review (下一步：檢閱)。
6. 按一下 Add Permission (新增許可)。
7. 返回 Permissions (許可) 索引標籤，按一下 Add inline policy (新增內嵌政策)。
8. 在 Create a policy (建立政策) 頁面上，按一下 JSON 索引標籤。
9. 貼上以下內容。

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

10. 按一下 Review policy (檢閱政策)。
11. 在名稱欄位上輸入 EMRforDynamoDBDataPipeline。
12. 按一下 Create Policy (建立政策)。

#### Note

您可以使用類似的程序將此受管政策連接至角色或群組，而不是使用者。

## 限制特定 DynamoDB 資料表的存取

如果您想限制存取權限，讓使用者只能匯出或匯入您的資料表子集，則需要建立自訂的 IAM 政策文件。您可以如 [授予完整存取權](#) 所述，使用此程序作為自訂政策的起點，然後修改政策，讓使用者只能使用您指定的資料表。

例如，假設您想要允許使用者只匯出和匯入 Forum、Thread 和 Reply 資料表。此程序說明如何建立自訂政策，讓使用者可以使用這些資料表，而不是其他資料表。

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 <https://console.aws.amazon.com/iam/>。
2. 從 IAM 主控台儀表板中，按一下 Policies (政策)，然後按一下 Create Policy(建立政策)。
3. 在「建立策略」面板中，移至「複製 AWS 受管理策略」，然後按一下「選取」。
4. 在「複製 AWS 受管理策略」面板中，移至 **AmazonDynamoDBFullAccess** 並按一下「選取」。
5. 在 Review Policy (檢閱政策) 面板中，執行下列操作：
  - a. 檢閱自動產生的 Policy Name (政策名稱) 和 Description (描述)。需要時，您可以修改這些值。
  - b. 在 Policy Document (政策文件) 文字方塊中，編輯政策來限制對特定資料表的存取。政策預設會允許對所有資料表執行的所有 DynamoDB 動作：

```
{
 "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...
```

若要限制政策，請先移除下行：

```
"dynamodb:*",
```

接下來，建構新的 Action，允許只存取 Forum、Thread 和 Reply 資料表：

```
{
 "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

將 us-west-2 取代為 DynamoDB 資料表所在的區域。請123456789012以您的 AWS 帳號取代。

最後，將新的 Action 新增至政策文件：

```
{
 "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. 滿意政策設定時，請按一下 Create Policy (建立政策)。

建立政策之後，即可將它連接至使用者。

1. 從 IAM 主控台儀表板中，按一下 Users (使用者)，然後選取您要修改的使用者。
2. 在 Permissions (許可) 標籤中，按一下 Attach Policy (連接政策)。
3. 在 Attach Policy (連接政策) 面板中，選取您政策的名稱，然後按一下 Attach Policy (連接政策)。

**Note**

您可以使用類似的程序將您的政策連接至角色或群組，而不是使用者。

## 將資料從 DynamoDB 匯出至 Amazon S3

本節說明如何將一或多個 DynamoDB 資料表中的資料匯出至 Amazon S3 儲存貯體。您需要先建立 Amazon S3 儲存貯體，才能執行匯出。

**Important**

如果您以 AWS Data Pipeline 前從未使用過，則必須先設定兩個 IAM 角色，然後再執行此程序。如需詳細資訊，請參閱 [建立 AWS Data Pipeline 的 IAM 角色](#)。

1. 請登入 AWS Management Console 並開啟 AWS Data Pipeline 主控台，網址為 <https://console.aws.amazon.com/datapipeline/>。
2. 如果目前 AWS 區域中尚未有任何管線，請選擇「立即開始」。  
否則，如果您已經至少有一或多個管道，則請選擇 Create new pipeline (建立新管道)。
3. 在 Create Pipeline (建立管道) 頁面上，執行下列操作：
  - a. 在 Name (名稱) 欄位中，輸入管道的名稱。例如：MyDynamoDBExportPipeline。
  - b. 針對 Source (來源) 參數，選取 Build using a template (使用範本建置)。從下拉式範本清單中，選擇 Export DynamoDB table to S3 (將 DynamoDB 資料表匯出至 S3)。
  - c. 在 Source DynamoDB table name (來源 DynamoDB 資料表名稱) 欄位中，輸入您要匯出的 DynamoDB 資料表名稱。
  - d. 在 Output S3 Folder (輸出 S3 資料夾) 文字方塊中，輸入將寫入匯出檔案的 Amazon S3 URI。例如：s3://mybucket/exports

此 URI 的格式是 `s3://bucketname/folder`，其中：

- `bucketname` 是您的 Amazon S3 儲存貯體的名稱。
- `folder` 是該儲存貯體內資料夾的名稱。如果資料夾不存在，則會自動建立資料夾。如果您未指定資料夾的名稱，則會指派其名稱，格式為 `s3://bucketname/region/tablename`。

- e. 在 S3 location for logs (日誌的 S3 位置) 文字方塊中，輸入將寫入匯出日誌檔案的 Amazon S3 URI。例如：`s3://mybucket/logs/`

S3 Log Folder (S3 日誌資料夾) 的 URI 格式與 Output S3 Folder (輸出 S3 資料夾) 相同。URI 必須解析為資料夾；日誌檔案無法寫入至 S3 儲存貯體頂層。

4. 新增金鑰為 `dynamodbdatapipeline` 和數值為 `true` 的標籤。
5. 當您滿意設定後，請按一下 `Activate` (啟用)。

現在將會建立您的管道；此程序需要幾分鐘的時間才能完成。您可以在 AWS Data Pipeline 控制台中監視進度。

匯出完成時，您可以前往 [Amazon S3 主控台](#) 來檢視匯出檔案。輸出檔案名稱為不含附檔名的識別碼值，例如此範例：`ae10f955-fb2f-4790-9b11-fbfea01a871e_000000`。此檔案的內部格式在 AWS Data Pipeline 開發人員指南中的 [檔案結構](#) 中有說明。

## 從 Amazon S3 匯入資料至 DynamoDB

本節假設您已經從 DynamoDB 資料表中匯出資料，而且已將匯出檔案寫入至 Amazon S3 儲存貯體。此檔案的內部格式在 AWS Data Pipeline 開發人員指南中的 [檔案結構](#) 中有說明。請注意，這是 DynamoDB 唯一可以使用匯入的檔案格式。AWS Data Pipeline

我們將使用來源資料表這個術語表示從中匯出資料的原始資料表，而目標資料表表示將收到已匯入資料的資料表。您可以在 Amazon S3 中匯入匯出檔案中的資料，但前提是符合下列所有項目：

- 目標資料表已存在 (匯入程序將不會建立資料表)。
- 目標資料表與來源資料表的索引鍵結構描述相同。

目標資料表不需要是空的。不過，匯入程序會取代資料表中索引鍵與匯出檔案中項目相同的任何資料項目。例如，假設您有一個包含索引鍵的 Customer 資料表 `CustomerId`，而且表格中只有三個項目 (`CustomerId1`、`2` 和 `3`)。如果您的匯出檔案也包含 `CustomerId 1`、`2` 和 `3` 的資料項目，則會將目標資料表中的項目取代為匯出檔案中的項目。如果匯出檔案也包含 `CustomerId4` 的資料項目，則該項目會新增至表格中。

目的地資料表可以位於不同的 AWS 區域。例如，假設您有一個 Customer (客戶) 資料表在美國西部 (奧勒岡) 區域，並要匯出其資料至 Amazon S3。您可以在之後將該資料匯入歐洲 (愛爾蘭) 區域中的相同 Customer (客戶) 資料表。這稱為跨區域匯出和匯入。如需 AWS 區域的清單，請前往 AWS 一般參考中的 [區域與端點](#)。



請注意，AWS Management Console 可讓您一次匯出多個來源表格。不過，您一次只能匯入一個資料表。

1. 請登入 AWS Management Console 並開啟 AWS Data Pipeline 主控台，網址為 <https://console.aws.amazon.com/datapipeline/>。
2. (選用) 如果您想要執行跨區域匯入，請前往視窗右上角，然後選擇目標區域。
3. 選擇 Create new pipeline (建立新的管道)。
4. 在 Create Pipeline (建立管道) 頁面上，執行下列操作：
  - a. 在 Name (名稱) 欄位中，輸入管道的名稱。例如：MyDynamoDBImportPipeline。
  - b. 針對 Source (來源) 參數，選取 Build using a template (使用範本建置)。從下拉式範本清單中，選擇 Import DynamoDB backup data from S3 (從 S3 匯入 DynamoDB 備份資料)。
  - c. 在 Input S3 Folder (輸入 S3 資料夾) 文字方塊中，輸入可找到匯出檔案的 Amazon S3 URI。例如：s3://mybucket/exports

此 URI 的格式是 `s3://bucketname/folder`，其中：

- `bucketname` 是您的 Amazon S3 儲存貯體的名稱。
- `folder` 是包含匯出檔案的資料夾名稱。

匯入任務預期會在指定的 Amazon S3 位置找到檔案。如需了解檔案的內部格式，請參閱《AWS Data Pipeline 開發人員指南》中的 [驗證資料匯出檔案](#)。

- d. 在 Target DynamoDB table name (目標 DynamoDB 資料表名稱) 欄位中，輸入您要將資料匯入其中的 DynamoDB 資料表名稱。
  - e. 在 S3 location for logs (日誌的 S3 位置) 文字方塊中，輸入將寫入匯入日誌檔案的 Amazon S3 URI。例如：s3://mybucket/logs/
- S3 Log Folder (S3 日誌資料夾) 的 URI 格式與 Output S3 Folder (輸出 S3 資料夾) 相同。URI 必須解析為資料夾；日誌檔案無法寫入至 S3 儲存貯體頂層。
- f. 新增金鑰為 `dynamodbdatapipeline` 和數值為 `true` 的標籤。
5. 當您滿意設定後，請按一下 Activate (啟用)。

現在將會建立您的管道；此程序需要幾分鐘的時間才能完成。匯入任務會在建立管道之後立即開始。

## 故障診斷

本節會介紹一些基本失敗模式，以及 DynamoDB 匯出的疑難排解資訊。

如果在匯出或匯入期間發生錯誤，則 AWS Data Pipeline 主控台管道狀態將會顯示為 ERROR。如果發生這種情況，請按一下故障管道的名稱，以前往其詳細資訊頁面。這將會顯示管道中所有步驟的詳細資訊，以及每個步驟的狀態。具體而言，請檢查您看到的任何執行堆疊追蹤。

最後，前往 Amazon S3 儲存貯體，並尋找任何已在該處寫入的匯出或匯入日誌檔。

下列是可能導致管道故障的一些常見問題，以及修正動作。若要診斷管道，請比較您在下面問題所看到的錯誤。

- 針對匯入，確保目標資料表已存在，而且目標資料表與來源資料表的索引鍵結構描述相同。必須符合這些條件，否則匯入會失敗。
- 確保管道具有標籤 dynamodbdatapipeline，否則 Amazon EMR API 呼叫不會成功。
- 確定已建立您指定的 Amazon S3 儲存貯體，而且您已有它的讀取和寫入許可。
- 管道可能已超過其執行逾時 (當您建立管道時，請設定此參數)。例如，您可能已設定 1 小時的執行逾時，但匯出任務所需的時間可能不僅於止。嘗試刪除後重新建立管道，但這次需要較長的執行逾時間隔。
- 如果您從 Amazon S3 儲存貯體還原，而此儲存貯體不是用來執行匯出的原始儲存貯體 (包含匯出複本)，則請更新資訊清單檔案。
- 您可能沒有執行匯出或匯入的正確許可。如需詳細資訊，請參閱 [匯出和匯入資料的必要條件](#)。
- 您可能已達到 AWS 帳戶中的資源配額，例如 Amazon EC2 執行個體的最大數量或 AWS Data Pipeline 管道數量上限。如需詳細資訊，包括如何請求提高這些配額，請參閱 AWS 一般參考中的 [AWS Service Quotas](#)。

### Note

如需管道疑難排解的詳細資訊，請前往《AWS Data Pipeline 開發人員指南》中的 [疑難排解](#)。

## 預先定義的 DynamoDB AWS Data Pipeline 和

如果您想更深入地了解 AWS Data Pipeline 作原理，我們建議您查閱 AWS Data Pipeline 開發人員指南。本指南包含建立和使用配管的自 step-by-step 學課程；您可以使用這些自學課程做為建立自己配管的起點。建議您閱讀 AWS Data Pipeline 教學，而本教學會逐步解說建立可針對您需求所自訂之匯入

和匯出管道所需的步驟。請參閱《AWS Data Pipeline 開發人員指南》中的[教學課程：使用 AWS Data Pipeline 執行 Amazon DynamoDB 匯入和匯出](#)。

AWS Data Pipeline 提供數個用於建立管道的範本；下列範本與 DynamoDB 相關。

在 DynamoDB 和 Amazon S3 之間匯出資料

#### Note

DynamoDB 主控台現在支援自己的匯出到 Amazon S3 流程，但與 AWS Data Pipeline 匯入流程不相容。如需詳細資訊，請參閱 [DynamoDB 資料匯出至 Amazon S3：運作方式](#) 和部落格文章 [《在不撰寫程式碼的情況下將 Amazon DynamoDB 資料表的資料匯出至 Amazon S3 中的資料湖》](#)。

主 AWS Data Pipeline 控制台提供兩個預先定義的範本，用於在 DynamoDB 和 Amazon S3 之間匯出資料。如需這些範本的詳細資訊，請參閱《AWS Data Pipeline 開發人員指南》的以下章節：

- [將 DynamoDB 匯出至 Amazon S3](#)
- [將 Amazon S3 匯出至 DynamoDB](#)

## 適用於 Titan 的 Amazon DynamoDB 儲存後端

適用於 Titan 的 DynamoDB 儲存後端專案已經由適用於 JanusGraph 的 Amazon DynamoDB 儲存後端取代，現在可在 [GitHub](#) 上取得。

如需 JanusGraph 的 DynamoDB 儲存後端的最新說明，請參閱 [README.md](#) 檔案。

## DynamoDB 中的保留字

下列關鍵字會保留給 DynamoDB 使用。請勿在表達式中使用任何這些單字作為屬性名稱。此表不區分大小寫。

如果需要撰寫的表達式會包含與 DynamoDB 保留字衝突的屬性名稱，您可以定義要取代保留字的表達式屬性名稱。如需詳細資訊，請參閱 [DynamoDB 中的表達式屬性名稱](#)。

```
ABORT
ABSOLUTE
```

ACTION  
ADD  
AFTER  
AGENT  
AGGREGATE  
ALL  
ALLOCATE  
ALTER  
ANALYZE  
AND  
ANY  
ARCHIVE  
ARE  
ARRAY  
AS  
ASC  
ASCII  
ASENSITIVE  
ASSERTION  
ASYMMETRIC  
AT  
ATOMIC  
ATTACH  
ATTRIBUTE  
AUTH  
AUTHORIZATION  
AUTHORIZE  
AUTO  
AVG  
BACK  
BACKUP  
BASE  
BATCH  
BEFORE  
BEGIN  
BETWEEN  
BIGINT  
BINARY  
BIT  
BLOB  
BLOCK  
BOOLEAN  
BOTH  
BREADTH

BUCKET  
BULK  
BY  
BYTE  
CALL  
CALLED  
CALLING  
CAPACITY  
CASCADE  
CASCADED  
CASE  
CAST  
CATALOG  
CHAR  
CHARACTER  
CHECK  
CLASS  
CLOB  
CLOSE  
CLUSTER  
CLUSTERED  
CLUSTERING  
CLUSTERS  
COALESCE  
COLLATE  
COLLATION  
COLLECTION  
COLUMN  
COLUMNS  
COMBINE  
COMMENT  
COMMIT  
COMPACT  
COMPILE  
COMPRESS  
CONDITION  
CONFLICT  
CONNECT  
CONNECTION  
CONSISTENCY  
CONSISTENT  
CONSTRAINT  
CONSTRAINTS  
CONSTRUCTOR

CONSUMED  
CONTINUE  
CONVERT  
COPY  
CORRESPONDING  
COUNT  
COUNTER  
CREATE  
CROSS  
CUBE  
CURRENT  
CURSOR  
CYCLE  
DATA  
DATABASE  
DATE  
DATETIME  
DAY  
DEALLOCATE  
DEC  
DECIMAL  
DECLARE  
DEFAULT  
DEFERRABLE  
DEFERRED  
DEFINE  
DEFINED  
DEFINITION  
DELETE  
DELIMITED  
DEPTH  
DEREF  
DESC  
DESCRIBE  
DESCRIPTOR  
DETACH  
DETERMINISTIC  
DIAGNOSTICS  
DIRECTORIES  
DISABLE  
DISCONNECT  
DISTINCT  
DISTRIBUTE  
DO

DOMAIN  
DOUBLE  
DROP  
DUMP  
DURATION  
DYNAMIC  
EACH  
ELEMENT  
ELSE  
ELSEIF  
EMPTY  
ENABLE  
END  
EQUAL  
EQUALS  
ERROR  
ESCAPE  
ESCAPED  
EVAL  
EVALUATE  
EXCEEDED  
EXCEPT  
EXCEPTION  
EXCEPTIONS  
EXCLUSIVE  
EXEC  
EXECUTE  
EXISTS  
EXIT  
EXPLAIN  
EXPLODE  
EXPORT  
EXPRESSION  
EXTENDED  
EXTERNAL  
EXTRACT  
FAIL  
FALSE  
FAMILY  
FETCH  
FIELDS  
FILE  
FILTER  
FILTERING

FINAL  
FINISH  
FIRST  
FIXED  
FLATTERN  
FLOAT  
FOR  
FORCE  
FOREIGN  
FORMAT  
FORWARD  
FOUND  
FREE  
FROM  
FULL  
FUNCTION  
FUNCTIONS  
GENERAL  
GENERATE  
GET  
GLOB  
GLOBAL  
GO  
GOTO  
GRANT  
GREATER  
GROUP  
GROUPING  
HANDLER  
HASH  
HAVE  
HAVING  
HEAP  
HIDDEN  
HOLD  
HOUR  
IDENTIFIED  
IDENTITY  
IF  
IGNORE  
IMMEDIATE  
IMPORT  
IN  
INCLUDING



INCLUSIVE  
INCREMENT  
INCREMENTAL  
INDEX  
INDEXED  
INDEXES  
INDICATOR  
INFINITE  
INITIALLY  
INLINE  
INNER  
INNTER  
INOUT  
INPUT  
INSENSITIVE  
INSERT  
INSTEAD  
INT  
INTEGER  
INTERSECT  
INTERVAL  
INTO  
INVALIDATE  
IS  
ISOLATION  
ITEM  
ITEMS  
ITERATE  
JOIN  
KEY  
KEYS  
LAG  
LANGUAGE  
LARGE  
LAST  
LATERAL  
LEAD  
LEADING  
LEAVE  
LEFT  
LENGTH  
LESS  
LEVEL  
LIKE

LIMIT  
LIMITED  
LINES  
LIST  
LOAD  
LOCAL  
LOCALTIME  
LOCALTIMESTAMP  
LOCATION  
LOCATOR  
LOCK  
LOCKS  
LOG  
LOGED  
LONG  
LOOP  
LOWER  
MAP  
MATCH  
MATERIALIZED  
MAX  
MAXLEN  
MEMBER  
MERGE  
METHOD  
METRICS  
MIN  
MINUS  
MINUTE  
MISSING  
MOD  
MODE  
MODIFIES  
MODIFY  
MODULE  
MONTH  
MULTI  
MULTISET  
NAME  
NAMES  
NATIONAL  
NATURAL  
NCHAR  
NCLOB

NEW  
NEXT  
NO  
NONE  
NOT  
NULL  
NULLIF  
NUMBER  
NUMERIC  
OBJECT  
OF  
OFFLINE  
OFFSET  
OLD  
ON  
ONLINE  
ONLY  
OPAQUE  
OPEN  
OPERATOR  
OPTION  
OR  
ORDER  
ORDINALITY  
OTHER  
OTHERS  
OUT  
OUTER  
OUTPUT  
OVER  
OVERLAPS  
OVERRIDE  
OWNER  
PAD  
PARALLEL  
PARAMETER  
PARAMETERS  
PARTIAL  
PARTITION  
PARTITIONED  
PARTITIONS  
PATH  
PERCENT  
PERCENTILE

PERMISSION  
PERMISSIONS  
PIPE  
PIPELINED  
PLAN  
POOL  
POSITION  
PRECISION  
PREPARE  
PRESERVE  
PRIMARY  
PRIOR  
PRIVATE  
PRIVILEGES  
PROCEDURE  
PROCESSED  
PROJECT  
PROJECTION  
PROPERTY  
PROVISIONING  
PUBLIC  
PUT  
QUERY  
QUIT  
QUORUM  
RAISE  
RANDOM  
RANGE  
RANK  
RAW  
READ  
READS  
REAL  
REBUILD  
RECORD  
RECURSIVE  
REDUCE  
REF  
REFERENCE  
REFERENCES  
REFERENCING  
REGEXP  
REGION  
REINDEX

RELATIVE  
RELEASE  
REMAINDER  
RENAME  
REPEAT  
REPLACE  
REQUEST  
RESET  
RESIGNAL  
RESOURCE  
RESPONSE  
RESTORE  
RESTRICT  
RESULT  
RETURN  
RETURNING  
RETURNS  
REVERSE  
REVOKE  
RIGHT  
ROLE  
ROLES  
ROLLBACK  
ROLLUP  
ROUTINE  
ROW  
ROWS  
RULE  
RULES  
SAMPLE  
SATISFIES  
SAVE  
SAVEPOINT  
SCAN  
SCHEMA  
SCOPE  
SCROLL  
SEARCH  
SECOND  
SECTION  
SEGMENT  
SEGMENTS  
SELECT  
SELF

SEMI  
SENSITIVE  
SEPARATE  
SEQUENCE  
SERIALIZABLE  
SESSION  
SET  
SETS  
SHARD  
SHARE  
SHARED  
SHORT  
SHOW  
SIGNAL  
SIMILAR  
SIZE  
SKEWED  
SMALLINT  
SNAPSHOT  
SOME  
SOURCE  
SPACE  
SPACES  
SPARSE  
SPECIFIC  
SPECIFICTYPE  
SPLIT  
SQL  
SQLCODE  
SQLERROR  
SQLEXCEPTION  
SQLSTATE  
SQLWARNING  
START  
STATE  
STATIC  
STATUS  
STORAGE  
STORE  
STORED  
STREAM  
STRING  
STRUCT  
STYLE

SUB  
SUBMULTISET  
SUBPARTITION  
SUBSTRING  
SUBTYPE  
SUM  
SUPER  
SYMMETRIC  
SYNONYM  
SYSTEM  
TABLE  
TABLESAMPLE  
TEMP  
TEMPORARY  
TERMINATED  
TEXT  
THAN  
THEN  
THROUGHPUT  
TIME  
TIMESTAMP  
TIMEZONE  
TINYINT  
TO  
TOKEN  
TOTAL  
TOUCH  
TRAILING  
TRANSACTION  
TRANSFORM  
TRANSLATE  
TRANSLATION  
TREAT  
TRIGGER  
TRIM  
TRUE  
TRUNCATE  
TTL  
TUPLE  
TYPE  
UNDER  
UNDO  
UNION  
UNIQUE

UNIT  
UNKNOWN  
UNLOGGED  
UNNEST  
UNPROCESSED  
UNSIGNED  
UNTIL  
UPDATE  
UPPER  
URL  
USAGE  
USE  
USER  
USERS  
USING  
UUID  
VACUUM  
VALUE  
VALUED  
VALUES  
VARCHAR  
VARIABLE  
VARIANCE  
VARINT  
VARYING  
VIEW  
VIEWS  
VIRTUAL  
VOID  
WAIT  
WHEN  
WHENEVER  
WHERE  
WHILE  
WINDOW  
WITH  
WITHIN  
WITHOUT  
WORK  
WRAPPED  
WRITE  
YEAR  
ZONE



## 舊式條件式參數

本節旨在比較舊式條件式參數與 DynamoDB 中的表達式參數。

### Important

我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需更多詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。

此外，DynamoDB 不允許在單一呼叫中混用舊式條件式參數和表達式參數。例如，使用 `AttributesToGet` 和 `ConditionExpression` 來呼叫 `Query` 操作會導致錯誤。

下表顯示仍支援這些舊版參數的 DynamoDB API 操作，以及要改用的表達式參數。若您考慮更新應用程式來改用表達式參數，此資料表會有所幫助。

如果您使用此 API 操作...	採用的是這些舊式參數...	改用此表達式參數
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
	Expected	ConditionExpression

以下章節將提供舊式條件式參數的詳細資訊。

## 主題

- [AttributesToGet \(舊版\)](#)
- [AttributeUpdates \(舊版\)](#)
- [ConditionalOperator \(舊版\)](#)
- [Expected \(舊版\)](#)
- [KeyConditions \(舊版\)](#)
- [QueryFilter \(舊版\)](#)
- [ScanFilter \(舊版\)](#)
- [使用舊式參數撰寫條件](#)

## AttributesToGet (舊版)

### Note

我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需更多詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。如需取代此參數之新參數的特定資訊，請參閱 [改用 ProjectionExpression](#)。

舊版條件式參數 `AttributesToGet` 是一個陣列，包含從 DynamoDB 中擷取的一或多個屬性。如果未提供屬性名稱，則會傳回所有屬性。如果找不到請求的任一屬性，則不會在結果中顯示這些屬性。

`AttributesToGet` 可讓您擷取 List 或 Map 類型的屬性，但無法擷取 List 或 Map 中的個別元素。

請注意，`AttributesToGet` 並不會影響佈建輸送量的耗用。DynamoDB 會根據項目大小判定使用的容量單位數，而不是根據傳回給應用程式的資料量。

## 改用 ProjectionExpression - 範例

假設要從 Music 資料表擷取一個項目，但只想傳回部分屬性。您可以搭配 `AttributesToGet` 參數使用 `GetItem` 請求，如本 AWS CLI 範例所示：

```
aws dynamodb get-item \
 --table-name Music \
 --attributes-to-get '["Artist", "Genre"]' \
 --key-values '{"Artist": "The Beatles"}'
```

```
--key '{
 "Artist": {"S":"No One You Know"},
 "SongTitle": {"S":"Call Me Today"}
}'
```

您可以改用 ProjectionExpression。

```
aws dynamodb get-item \
 --table-name Music \
 --projection-expression "Artist, Genre" \
 --key '{
 "Artist": {"S":"No One You Know"},
 "SongTitle": {"S":"Call Me Today"}
 }'
```

## AttributeUpdates (舊版)

### Note

我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需更多詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。如需取代此參數之新參數的特定資訊，請參閱 [改用 UpdateExpression](#)。

在 UpdateItem 操作中，舊版條件式參數 AttributeUpdates 包含要修改的屬性名稱、每個屬性上要執行的動作，以及每個屬性的新值。如果您要更新的屬性是資料表上任何索引的索引鍵屬性，則其類型必須符合資料表描述的 AttributesDefinition 中所定義的索引鍵類型。您可以使用 UpdateItem 來更新任何非索引鍵屬性。

屬性值不可為 Null。String 和 Binary 類型屬性的長度必須大於零。Set 類型屬性不能為空白。具有空白數值的請求會遭到拒絕，並出現 ValidationException 例外狀況。

每個 AttributeUpdates 元素包含要修改的屬性名稱，並具有下列資訊：

- Value：此屬性的新值 (如適用)。
- Action：用以指定如何執行更新的數值。此動作僅適用於資料類型為 Number 或 Set 的現有屬性。請勿對其他資料類型使用 ADD。

如果在資料表中找到具有指定主索引鍵的項目，則下列各數值會執行以下動作：

- PUT：將指定屬性新增至項目。如果屬性已存在，則以新數值取代。

- DELETE：如果未指定 DELETE 的數值，則刪除屬性及其數值。指定數值的資料類型必須符合現有數值的資料類型。

如果指定一組數值，則會從舊的集合中減去該等數值。例如，如果屬性值是集合 [a,b,c]，而且 DELETE 動作指定 [a,c]，則最後屬性值為 [b]。指定空集合是一項錯誤。

- ADD：如果屬性尚未存在，則將指定數值新增至項目。如果屬性已存在，則 ADD 的行為依屬性的資料類型而定：
  - 如果現有屬性是數字，而 Value 亦為數字，則 Value 會以數學方式新增至現有屬性。如果 Value 是負數，則會從現有屬性減去。

#### Note

如果對更新前不存在的項目使用 ADD 增減數值，則 DynamoDB 會以 0 為初始數值。同樣地，如果對現有項目使用 ADD 增減更新前不存在的屬性值，則 DynamoDB 會以 0 為初始數值。例如，假設欲更新的項目不具有 itemcount 名稱的屬性，但您仍要將此屬性 ADD 數字 3。DynamoDB 會建立屬性 itemcount，將其初始數值設為 0，最後再加上 3。結果會得到數值為 3 的新 itemcount 屬性。

- 如果現有的資料類型是集合，而 Value 亦為集合，則 Value 會附加至現有集合。例如，如果屬性值是集合 [1,2]，而且 ADD 動作指定 [3]，則最後屬性值為 [1,2,3]。如果已為集合屬性指定 ADD 動作，且指定的屬性類型與現有集合類型不符，則會發生錯誤。

兩個集合必須具有相同的基本資料類型。例如，如果現有的資料類型是一組字串，Value 也必須是一組字串。

如果未在資料表中找到具有指定索引鍵的項目，則下列各數值會執行以下動作：

- PUT：讓 DynamoDB 使用指定的主索引鍵建立新項目，然後新增屬性。
- DELETE：不進行任何動作，因為無法從不存在的項目中刪除屬性。成功完成操作，但 DynamoDB 不會建立新項目。
- ADD：讓 DynamoDB 使用為屬性值提供的主索引鍵和數字 (或一組數字) 建立項目。僅允許 Number 和 Number Set 資料類型。

如果提供屬於索引鍵一部分的任何屬性，則這些屬性的資料類型必須符合資料表屬性定義中結構描述的資料類型。

## 改用 UpdateExpression - 範例

假設您想修改 Music 資料表中的一個項目。您可以搭配 AttributeUpdates 參數使用 UpdateItem 請求，如本 AWS CLI 範例所示：

```
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"}
 }
 }'
```

您可以改用 UpdateExpression。

```
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"}
 }'
```

如需有關更新屬性的詳細資訊，請參閱 [更新 DynamoDB 資料表中的項目](#)。

## ConditionalOperator (舊版)

### Note

我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需更多詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。

舊版條件式參數 `ConditionalOperator` 是邏輯運算子，可套用至 `Expected`、`QueryFilter` 或 `ScanFilter` 映射中的條件：

- AND：如果所有條件評估為 `true`，則整個映射評估為 `true`。
- OR：如果至少其中一個條件評估為 `true`，則整個映射評估為 `true`。

如果省略 `ConditionalOperator`，則會以 AND 為預設。

只有在整個映射評估為 `true` 時，操作才會成功。

#### Note

此參數不支援 List 或 Map 類型的屬性。

## Expected (舊版)

#### Note

我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需更多詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。如需取代此參數之新參數的特定資訊，請參閱 [改用 ConditionExpression](#)。

舊版條件式參數 `Expected` 是 `UpdateItem` 操作的條件式區塊。`Expected` 是屬性/條件對的映射。映射的每個元素都包含一個屬性名稱、一個比較運算子，以及一或多個數值。DynamoDB 會使用比較運算子來比較屬性與您提供的數值。對於每個 `Expected` 元素，評估結果皆為 `true` 或 `false`。

如果在 `Expected` 映射中指定一個以上元素，所有條件必須根據預設評估為 `true`。換言之，條件是以 AND 連在一起的。(您可以改用 `ConditionalOperator` 參數將條件以 OR 連在一起。若要如此，則至少其中一個條件必須評估為 `true`，而不是全部都必須評估為 `true`。)

如果 `Expected` 映射評估為 `true`，則條件式操作會成功，反之則會失敗。

`Expected` 包含下列各項：

- `AttributeValueList`：針對所提供的屬性進行評估的一或多個數值。清單中值的數目依使用的 `ComparisonOperator` 而定。

Number 類型的數值比較為數字。

大於、等於或小於的 String 數值比較是根據 UTF-8 二進位編碼的 Unicode。例如，a 大於 A，a 大於 B。

針對 Binary 類型，每當 DynamoDB 比較二進位值時，都會將二進位資料的每個位元組視為不帶正負號。

- `ComparisonOperator`：用於評估 `AttributeValueList` 中屬性的比較程式。執行比較時，DynamoDB 會使用強烈一致讀取。

可以使用下列比較運算子：

```
EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
BEGINS_WITH | IN | BETWEEN
```

以下是每個比較運算子的描述。

- `EQ`：等於。所有資料類型都支援 `EQ`，包括清單和映射。

`AttributeValueList` 僅可包含 String、Number、Binary、String Set、Number Set 或 Binary Set 類型的一個 `AttributeValue` 元素。如果項目內含的 `AttributeValue` 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，`{"S":"6"}` 不等於 `{"N":"6"}`。另外，`{"N":"6"}` 不等於 `{"NS":["6", "2", "1"]}`。

- `NE`：不等於。所有資料類型都支援 `NE`，包括清單和映射。

`AttributeValueList` 僅可包含 String、Number、Binary、String Set、Number Set 或 Binary Set 類型的一個 `AttributeValue`。如果項目內含的 `AttributeValue` 與所屬類型與請求中提供的類型不同，則數值不相符。例如，`{"S":"6"}` 不等於 `{"N":"6"}`。另外，`{"N":"6"}` 不等於 `{"NS":["6", "2", "1"]}`。

- `LE`：小於或等於。

`AttributeValueList` 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 `AttributeValue` 元素。如果項目內含的 `AttributeValue` 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，`{"S":"6"}` 不等於 `{"N":"6"}`。另外，`{"N":"6"}` 不與 `{"NS":["6", "2", "1"]}` 比較。

- `LT`：小於。

`AttributeValueList` 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 `AttributeValue`。如果項目內含的 `AttributeValue` 元素所屬類型與請求中提供的類型不

同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- GE：大於或等於。

AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- GT：大於。

AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- NOT\_NULL：屬性存在。所有資料類型都支援 NOT\_NULL，包括清單和映射。

#### Note

此運算子會測試屬性是否存在，而不是其資料類型。如果屬性 a 的資料類型為 Null，並且是以 NOT\_NULL 評估，則結果為布林值 true。此結果是因為屬性 a 存在；其資料類型與 NOT\_NULL 比較運算子無關。

- NULL：屬性不存在。所有資料類型都支援 NULL，包括清單和映射。

#### Note

此運算子會測試屬性是否不存在，而不是其資料類型。如果屬性 a 的資料類型為 Null，並且是以 NULL 評估，則結果為布林值 false。這是因為屬性 a 存在；其資料類型與 NULL 比較運算子無關。

- CONTAINS：檢查子序列，或是集合中的數值。

AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果比較的目標屬性是 String 類型，則運算子會檢查相符子字串。如果比較的目標屬性是 Binary 類型，則運算子會尋找與輸入相符的目標子序列。比較的目標屬性為集合 (SS、NS 或 BS) 時，如果找到與該集合任一成員完全相符的項目，則運算子評估為 true。



清單支援 CONTAINS：當評估 a CONTAINS b 時，a 可以是清單，但 b 不可以是集合、映射或清單。

- NOT\_CONTAINS：檢查子序列是否不存在，或者集合中的數值是否不存在。

AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果比較的目標屬性是 String，則運算子會檢查相符子字串是否不存在。如果比較的目標屬性是 Binary，則運算子會檢查與輸入相符的目標子序列是否不存在。比較的目標屬性為集合 (SS、NS 或 BS) 時，如果 does not 找到與該集合任一成員完全相符的項目，則運算子評估為 true。

清單支援 NOT\_CONTAINS：當評估 a NOT CONTAINS b 時，a 可以是清單，但 b 不可以是集合、映射或清單。

- BEGINS\_WITH：檢查字首。

AttributeValueList 僅可包含 String 或 Binary 類型 (非 Number 或集合類型) 的一個 AttributeValue。比較的目標屬性必須是 String 或 Binary 類型 (非 Number 或集合類型)。

- IN：檢查兩個集合內相符的元素。

AttributeValueList 可以包含 String、Number 或 Binary 類型 (非集合類型) 的一或多個 AttributeValue 元素。這些屬性會與項目的現有集合類型屬性進行比較。如果輸入集合的任何元素存在於項目屬性中，則表達式評估為 true。

- BETWEEN：大於或等於第一個數值，並且小於或等於第二個數值。

AttributeValueList 必須包含 String、Number 或 Binary 類型 (非集合類型) 的兩個 AttributeValue 元素。如果目標數值大於或等於第一個元素，並且小於或等於第二個元素，則目標屬性相符。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不與 {"N":"6"} 比較。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

下列參數可用來取代 AttributeValueList 和 ComparisonOperator：

- Value：DynamoDB 用來與屬性進行比較的數值。
- Exists：此為布林值，可讓 DynamoDB 在嘗試條件式操作之前評估數值：
  - 如果 Exists 為 true，則 DynamoDB 會查看該屬性值是否已存在於資料表中。如果找到該數值，則條件評估為 true，否則條件評估為 false。

- 如果 `Exists` 為 `false`，則 DynamoDB 會假設屬性值 `not` 存在於資料表中。如果該數值實際上不存在，則假設有效，條件會評估為 `true`。如果找到該值，則即使假設該值不存在，條件仍會評估為 `false`。

請注意，`Exists` 預設值為 `true`。

`Value` 和 `Exists` 參數與 `AttributeValueList` 和 `ComparisonOperator` 不相容。請注意，如果同時使用此兩組參數，DynamoDB 會傳回 `ValidationException` 例外狀況。

### Note

此參數不支援 `List` 或 `Map` 類型的屬性。

## 改用 `ConditionExpression` - 範例

假設您想只在某一條件為 `true` 的情況下修改 `Music` 資料表中的一個項目。您可以搭配 `Expected` 參數使用 `UpdateItem` 請求，如本 AWS CLI 範例所示：

```
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"}]
 }
}'
```

您可以改用 `ConditionExpression`。

```
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 (舊版)

### Note

我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需更多詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。如需取代此參數之新參數的特定資訊，請參閱 [改用 KeyConditionExpression](#)。

舊版條件式參數 KeyConditions 包含 Query 操作的選取條件。對於資料表上的查詢，您可以僅對資料表主索引鍵屬性設定條件。您必須將分割區索引鍵名稱和值提供為 EQ 條件。您可選擇性地提供指向排序索引鍵的第二個條件。

### Note

如未提供排序索引鍵條件，則會擷取所有符合分割區索引鍵的項目。如果 FilterExpression 或 QueryFilter 存在，則會在擷取項目後套用。

對於索引上的查詢，您可以僅對索引鍵屬性設定條件。您必須將索引分割區索引鍵名稱和值提供為 EQ 條件。您可選擇提供指向索引排序索引鍵的第二個條件。

每個 KeyConditions 元素包含要比較的屬性名稱，並具有下列資訊：

- AttributeValueList：針對所提供的屬性進行評估的一或多個數值。清單中值的數目依使用的 ComparisonOperator 而定。

Number 類型的數值比較為數字。

大於、等於或小於的 String 數值比較是根據 UTF-8 二進位編碼的 Unicode。例如，a 大於 A，a 大於 B。

針對 Binary，每當 DynamoDB 比較二進位值時，都會將二進位資料的每個位元組視為不帶正負號。

- ComparisonOperator：用於評估屬性的比較程式。例如：等於、大於和小於。

針對 KeyConditions，只支援下列比較運算子：

EQ | LE | LT | GE | GT | BEGINS\_WITH | BETWEEN

以下是這些比較運算子的描述。

- EQ：等於。

AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不等於 {"NS":["6", "2", "1"]}。

- LE：小於或等於。

AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- LT：小於。

AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- GE：大於或等於。

AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- GT：大於。

AttributeValueList 僅可包含 String、Number 或 Binary 類型 (非集合類型) 的一個 AttributeValue 元素。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

- BEGINS\_WITH：檢查字首。

AttributeValueList 僅可包含 String 或 Binary 類型 (非 Number 或集合類型) 的一個 AttributeValue。比較的目標屬性必須是 String 或 Binary 類型 (非 Number 或集合類型)。

- BETWEEN：大於或等於第一個數值，並且小於或等於第二個數值。

AttributeValueList 必須包含 String、Number 或 Binary 類型 (非集合類型) 的兩個 AttributeValue 元素。如果目標數值大於或等於第一個元素，並且小於或等於第二個元素，則目標屬性相符。如果項目內含的 AttributeValue 元素所屬類型與請求中提供的類型不同，則數值不相符。例如，{"S":"6"} 不與 {"N":"6"} 比較。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。

## 改用 KeyConditionExpression - 範例

假設您想從 Music 資料表擷取數個具有相同分割區索引鍵的項目。您可以搭配 KeyConditions 參數使用 Query 請求，如本 AWS CLI 範例所示：

```
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"}]
 }
 }'
```

您可以改用 KeyConditionExpression。

```
aws dynamodb query \
 --table-name Music \
 --key-condition-expression 'Artist = :Artist AND SongTitle BETWEEN :A AND :M'
```

```
--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 (舊版)

### Note

我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需更多詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。如需取代此參數之新參數的特定資訊，請參閱 [改用 FilterExpression](#)。

在 Query 操作中，舊版條件式參數 QueryFilter 是可在項目讀取後評估查詢結果並僅傳回所需值的一個條件。

此參數不支援 List 或 Map 類型的屬性。

### Note

QueryFilter 會在項目讀取後套用；篩選程序不會使用任何額外的讀取容量單位。

如果在 QueryFilter 映射中提供一個以上條件，所有條件必須根據預設評估為 true。換言之，條件是以 AND 連在一起的。(您可以改用 [ConditionalOperator \(舊版\)](#) 參數將條件以 OR 連在一起。若要如此，則至少其中一個條件必須評估為 true，而不是全部都必須評估為 true。)

請注意，QueryFilter 不允許使用索引鍵屬性。您無法在分割區索引鍵或排序索引鍵上定義篩選條件。

每個 QueryFilter 元素包含要比較的屬性名稱，並具有下列資訊：

- `AttributeValueList`：針對所提供的屬性進行評估的一或多個數值。清單中的數值數目依 `ComparisonOperator` 中指定的運算子而定。

`Number` 類型的數值比較為數字。

大於、等於或小於的 String 數值比較是以 UTF-8 二進位編碼為基礎。例如，a 大於 A，a 大於 B。

針對 Binary 類型，每當 DynamoDB 比較二進位值時，都會將二進位資料的每個位元組視為不帶正負號。

如需在 JSON 中指定資料類型的詳細資訊，請參閱 [DynamoDB 低階 API](#)。

- `ComparisonOperator`：用於評估屬性的比較程式。例如：等於、大於和小於。

可以使用下列比較運算子：

EQ | NE | LE | LT | GE | GT | NOT\_NULL | NULL | CONTAINS | NOT\_CONTAINS |  
BEGINS\_WITH | IN | BETWEEN

## 改用 FilterExpression - 範例

假設您想查詢 Music 資料表，並將條件套用至相符項目。您可以搭配 `QueryFilter` 參數使用 Query 請求，如本 AWS CLI 範例所示：

```
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"}]
 }
 }'
```

您可以改用 `FilterExpression`。

```
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 (舊版)

### Note

我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需更多詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。如需取代此參數之新參數的特定資訊，請參閱 [改用 FilterExpression](#)。

在 Scan 操作中，舊版條件式參數 ScanFilter 是可評估掃描結果並僅傳回所需值的一個條件。

### Note

此參數不支援 List 或 Map 類型的屬性。

如果在 ScanFilter 映射中指定一個以上條件，所有條件必須根據預設評估為 true。換言之，條件是以 AND 連在一起的。(您可以改用 [ConditionalOperator \(舊版\)](#) 參數將條件以 OR 連在一起。若要如此，則至少其中一個條件必須評估為 true，而不是全部都必須評估為 true。)

每個 ScanFilter 元素包含要比較的屬性名稱，並具有下列資訊：

- **AttributeValueList**：針對所提供的屬性進行評估的一或多個數值。清單中的數值數目依 **ComparisonOperator** 中指定的運算子而定。

Number 類型的數值比較為數字。

大於、等於或小於的 String 數值比較是以 UTF-8 二進位編碼為基礎。例如，a 大於 A，a 大於 B。

針對 Binary，每當 DynamoDB 比較二進位值時，都會將二進位資料的每個位元組視為不帶正負號。

如需在 JSON 中指定資料類型的詳細資訊，請參閱 [DynamoDB 低階 API](#)。

- **ComparisonOperator**：用於評估屬性的比較程式。例如：等於、大於和小於。

可以使用下列比較運算子：



EQ | NE | LE | LT | GE | GT | NOT\_NULL | NULL | CONTAINS | NOT\_CONTAINS |  
BEGINS\_WITH | IN | BETWEEN

## 改用 FilterExpression - 範例

假設您想掃描 Music 資料表，並將條件套用至相符項目。您可以搭配 ScanFilter 參數使用 Scan 請求，如本 AWS CLI 範例所示：

```
aws dynamodb scan \
 --table-name Music \
 --scan-filter '{
 "Genre":{
 "AttributeValueList":[{"S":"Rock"}],
 "ComparisonOperator": "EQ"
 }
 }'
```

您可以改用 FilterExpression。

```
aws dynamodb scan \
 --table-name Music \
 --filter-expression 'Genre = :g' \
 --expression-attribute-values '{
 ":g": {"S":"Rock"}
 }'
```

## 使用舊式參數撰寫條件

### Note

我們建議您盡可能使用新的表達式參數，而不要使用舊版參數。如需更多詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。

下節說明如何撰寫搭配舊式參數 (例如 Expected、QueryFilter、ScanFilter) 使用的條件。

**Note**

新的應用程式應改用表達式參數。如需更多詳細資訊，請參閱 [在 DynamoDB 中使用表達式](#)。

## 簡單條件

使用屬性值，您可以撰寫條件來與資料表屬性進行比較。條件一律評估為 true 或 false，並且包含下列各項：

- `ComparisonOperator`：大於、小於、等於等。
- `AttributeValueList` (選用)：要比較的屬性值。依使用中的 `ComparisonOperator` 而定，`AttributeValueList` 可能包含一兩個或更多數值，或者根本不存在。

下列各節說明各種比較運算子，並附有這些運算子在條件中的用法範例。

### 不具屬性值的比較運算子

- `NOT_NULL`：如果屬性存在，則本項為 true。
- `NULL`：如果屬性不存在，則本項為 true。

使用這些運算子來檢查屬性存在與否。因為並無可比較的數值，所以不要指定 `AttributeValueList`。

### 範例

如果 `Dimensions` 屬性存在，則下列表達式評估為 true。

```
...
 "Dimensions": {
 ComparisonOperator: "NOT_NULL"
 }
...
```

### 具有一個屬性值的比較運算子

- `EQ`：如果某屬性等於某數值，則本項為 true。

`AttributeValueList` 僅可包含 `String`、`Number`、`Binary`、`String Set`、`Number Set` 或 `Binary Set` 類型的一個數值。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。例如，字串 "3" 不等於數字 3。此外，數字 3 不等於數字集合 [3, 2, 1]。

- `NE`：如果某屬性不等於某數值，則本項為 `true`。

`AttributeValueList` 僅可包含 `String`、`Number`、`Binary`、`String Set`、`Number Set` 或 `Binary Set` 類型的一個數值。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。

- `LE`：如果某屬性小於或等於某數值，則本項為 `true`。

`AttributeValueList` 僅可包含 `String`、`Number` 或 `Binary` 類型 (非集合) 的一個數值。如果項目內含的 `AttributeValue` 所屬類型與請求中指定的類型不同，則數值不相符。

- `LT`：如果某屬性小於某數值，則本項為 `true`。

`AttributeValueList` 僅可包含 `String`、`Number` 或 `Binary` 類型 (非集合) 的一個數值。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。

- `GE`：如果某屬性大於或等於某數值，則本項為 `true`。

`AttributeValueList` 僅可包含 `String`、`Number` 或 `Binary` 類型 (非集合) 的一個數值。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。

- `GT`：如果某屬性大於某數值，則本項為 `true`。

`AttributeValueList` 僅可包含 `String`、`Number` 或 `Binary` 類型 (非集合) 的一個數值。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。

- `CONTAINS`：如果某數值在集合內，或者某數值包含另一個數值，則本項為 `true`。

`AttributeValueList` 僅可包含 `String`、`Number` 或 `Binary` 類型 (非集合) 的一個數值。如果比較的目標屬性是 `String`，則運算子會檢查相符子字串。如果比較的目標屬性是 `Binary`，則運算子會尋找與輸入相符的目標子序列。比較的目標屬性為集合時，如果找到與該集合任一成員完全相符的項目，則運算子評估為 `true`。

- `NOT_CONTAINS`：如果某數值不在集合內，或者某數值不包含另一個數值，則本項為 `true`。

`AttributeValueList` 僅可包含 `String`、`Number` 或 `Binary` 類型 (非集合) 的一個數值。如果比較的目標屬性是 `String`，則運算子會檢查相符子字串是否不存在。如果比較的目標屬性是 `Binary`，則運算子會檢查與輸入相符的目標子序列是否不存在。比較的目標屬性為集合時，如果未找到與該集合任一成員完全相符的項目，則運算子評估為 `true`。

- **BEGINS\_WITH**：如果屬性前幾個字元與提供的數值相符，則本項為 true。請勿使用此運算子來比較數字。

**AttributeValueList** 僅可包含 String 或 Binary 類型 (非 Number 或集合) 的一個數值。比較的目標屬性必須是 String 或 Binary (非 Number 或集合)。

使用這些運算子來比較屬性與數值。您必須指定包含單一數值的 **AttributeValueList**。大多數的運算子要求此數值必須為一個純量，不過 EQ 和 NE 運算子也支援集合。

## 範例

下列各表達式會評估為 true：

- 產品價格大於 100。

```
...
 "Price": {
 ComparisonOperator: "GT",
 AttributeValueList: [{"N": "100"}]
 }
...
```

- 產品類別以 Bo 開頭。

```
...
 "ProductCategory": {
 ComparisonOperator: "BEGINS_WITH",
 AttributeValueList: [{"S": "Bo"}]
 }
...
```

- 產品有紅色、綠色或黑色款式：

```
...
 "Color": {
 ComparisonOperator: "EQ",
 AttributeValueList: [
 [{"S": "Black"}, {"S": "Red"}, {"S": "Green"}]
]
 }
...
```

**Note**

比較集合資料類型時，元素的順序無關緊要。無論在請求中指定何順序，DynamoDB 只會傳回具有相同一組數值的項目。

### 具有兩個屬性值的比較運算子

- **BETWEEN**：如果某數值介於下限和上限之間 (含端點在內)，則本項為 true。

`AttributeValueList` 必須包含 `String`、`Number` 或 `Binary` 類型 (非集合) 的兩個元素。如果目標數值大於或等於第一個元素，並且小於或等於第二個元素，則目標屬性相符。如果項目內含的數值所屬類型與請求中指定的類型不同，則數值不相符。

使用此運算子判定屬性值是否在某範圍內。`AttributeValueList` 必須包含同為 `String`、`Number` 或 `Binary` 類型的兩個純量元素。

### 範例

如果產品價格介於 100 和 200 之間，則下列表達式評估為 true。

```
...
 "Price": {
 ComparisonOperator: "BETWEEN",
 AttributeValueList: [{"N": "100"}, {"N": "200"}]
 }
...
```

### 具有 N 屬性值的比較運算子

- **IN**：如果某數值等於列舉清單中任何數值，則本項為 true。清單中僅支援純量值，不支援集合。目標屬性必須為相同的類型和確切數值才相符。

`AttributeValueList` 可以包含 `String`、`Number` 或 `Binary` 類型 (非集合) 的一或多個元素。這些屬性會與項目的現有非集合類型屬性進行比較。如果輸入集合的任何元素存在於項目屬性中，則表達式評估為 true。

`AttributeValueList` 可以包含 `String`、`Number` 或 `Binary` 類型 (非集合) 的一或多個數值。比較的目標屬性必須為相同的類型和確切數值才相符。`String` 一律不與 `String` 集合相符。

使用此運算子判定提供的數值是否在列舉清單內。您可以在 `AttributeValueList` 中指定任意數量的純量值，但所有純量值必須屬於同一資料類型。

## 範例

如果 `Id` 的數值為 201、203 或 205，則下列表達式評估為 `true`。

```
...
 "Id": {
 ComparisonOperator: "IN",
 AttributeValueList: [{"N":"201"}, {"N":"203"}, {"N":"205"}]
 }
...
```

## 使用多個條件

DynamoDB 可讓您將多個條件結合成複雜的表達式。您可以提供至少兩個表達式來執行此操作，另可選用 [ConditionalOperator \(舊版\)](#)。

根據預設，指定一個以上條件時，所有條件必須評估為 `true`，以便整個表達式評估為 `true`。換言之，會執行隱含的 AND 操作。

## 範例

如果產品為一本至少有 600 頁的書，則下列表達式評估為 `true`。這兩個條件皆須評估為 `true`，因為兩者隱含地以 AND 連在一起。

```
...
 "ProductCategory": {
 ComparisonOperator: "EQ",
 AttributeValueList: [{"S":"Book"}]
 },
 "PageCount": {
 ComparisonOperator: "GE",
 AttributeValueList: [{"N":600}]
 }
...
```

可以使用 [ConditionalOperator \(舊版\)](#) 來清楚表示會執行一個 AND 操作。下列範例與前一範例的表現方式相同。

```
...
 "ConditionalOperator" : "AND",
 "ProductCategory": {
 "ComparisonOperator": "EQ",
 "AttributeValueList": [{"N":"Book"}]
 },
 "PageCount": {
 "ComparisonOperator": "GE",
 "AttributeValueList": [{"N":600}]
 }
...

```

您也可以將 `ConditionalOperator` 設定為 `OR`，這表示其中至少一個條件必須評估為 `true`。

### 範例

如果產品為一台登山單車、屬於特定品牌，或者其價格大於 100，則下列表達式評估為 `true`。

```
...
 ConditionalOperator : "OR",
 "BicycleType": {
 "ComparisonOperator": "EQ",
 "AttributeValueList": [{"S":"Mountain"}]
 },
 "Brand": {
 "ComparisonOperator": "EQ",
 "AttributeValueList": [{"S":"Brand-Company A"}]
 },
 "Price": {
 "ComparisonOperator": "GT",
 "AttributeValueList": [{"N":"100"}]
 }
...

```

#### Note

在複雜的表達式中，會按第一到最後一個條件的順序依序處理條件。  
無法在單一表達式中同時使用 `AND` 和 `OR`。

## 其他條件式運算子

在舊版 DynamoDB 當中，Expected 參數在條件式寫入方面有不同的表現方式。Expected 映射中的每個項目顯示 DynamoDB 要檢查的屬性名稱，並具有下列資訊：

- Value：要與屬性比較的值。
- Exists：判定在嘗試操作之前是否已有此數值。

DynamoDB 持續支援 Value 和 Exists 選項；不過，這些選項僅能測試相等條件，或者是否存在某屬性。建議您改用 ComparisonOperator 和 AttributeValueList，因為這些選項可讓您建構更廣泛的條件。

### Example

DeleteItem 可以查看書籍是否已絕版，並且只在此條件為 true 時才刪除書籍。以下為使用舊式條件的 AWS CLI 範例：

```
aws dynamodb delete-item \
 --table-name ProductCatalog \
 --key '{
 "Id": {"N":"600"}
 }' \
 --expected '{
 "InPublication": {
 "Exists": true,
 "Value": {"BOOL":false}
 }
 }'
```

以下範例顯示相同項目，但不使用舊式條件：

```
aws dynamodb delete-item \
 --table-name ProductCatalog \
 --key '{
 "Id": {"N":"600"}
 }' \
 --expected '{
 "InPublication": {
 "ComparisonOperator": "EQ",
```



```
 "AttributeValueList": [{"B00L":false}]
 }
}'
```

## Example

PutItem 操作可防止覆寫已有相同主索引鍵屬性的現有項目。以下為使用舊式條件的範例：

```
aws dynamodb put-item \
 --table-name ProductCatalog \
 --item '{
 "Id": {"N":"500"},
 "Title": {"S":"Book 500 Title"}
 }' \
 --expected '{
 "Id": { "Exists": false }
 }'
```

以下範例顯示相同項目，但不使用舊式條件：

```
aws dynamodb put-item \
 --table-name ProductCatalog \
 --item '{
 "Id": {"N":"500"},
 "Title": {"S":"Book 500 Title"}
 }' \
 --expected '{
 "Id": { "ComparisonOperator": "NULL" }
 }'
```

### Note

針對 Expected 映射中的條件，請勿搭配 ComparisonOperator 和 AttributeValueList 使用舊式的 Value 和 Exists 選項。若採取此法，條件式寫入將會失敗。

## 舊版低階 API (2011-12-05)

本節記錄舊版 DynamoDB 低階 API (2011-12-05) 中可用的操作。現保留此版本的低階 API，以便與現有應用程式回溯相容。

新的應用程式應使用目前 API 版本 (2012-08-10)。如需詳細資訊，請參閱 [低階 API 參考](#)。

### Note

建議您將應用程式遷移至最新的 API 版本 (2012-08-10)，因為新的 DynamoDB 功能不會向後移植到舊版的 API。

## 主題

- [BatchGetItem](#)
- [BatchWriteItem](#)
- [CreateTable](#)
- [DeleteItem](#)
- [DeleteTable](#)
- [DescribeTables](#)
- [GetItem](#)
- [ListTables](#)
- [PutItem](#)
- [Query](#)
- [Scan](#)
- [UpdateItem](#)
- [UpdateTable](#)

## BatchGetItem

### Important

**##### API ## 2011-12-05#**

如需目前低階 API 的文件，請參閱 [Amazon DynamoDB API 參考](#)。

## 描述

BatchGetItem 操作會使用多個資料表的主索引鍵從中傳回多個項目的屬性。單一操作最多可擷取的項目數為 100。此外，擷取的項目數受到 1 MB 大小限制。如果超過回應大小限制，或者因為超出資料

表的佈建輸送量，或因為內部處理失敗而傳回部分結果，則 DynamoDB 會傳回 `UnprocessedKeys` 值，以便您重試從下一個要獲取的項目開始的操作。DynamoDB 會自動調整每個頁面傳回的項目數，以便強制執行此限制。例如，即使您要求擷取 100 個項目，但每一個別項目大小為 50 KB，系統也會傳回 20 個項目和適當的 `UnprocessedKeys` 值，以便您取得下一頁的結果。如有需要，應用程式可包含自身的邏輯，將結果頁面組合成一組。

如果因為請求中涉及的每個資料表上佈建輸送量不足而無法處理任何項目，則 DynamoDB 會傳回 `ProvisionedThroughputExceededException` 錯誤。

### Note

根據預設，`BatchGetItem` 會對請求中每個資料表執行最終一致讀取。如果想改為一致性讀取，可以在每份資料表將 `ConsistentRead` 參數設定為 `true`。

`BatchGetItem` 會平行擷取項目，將回應延遲減至最低。

設計應用程式時，請注意 DynamoDB 不保證傳回的回應中的屬性順序。在

`AttributesToGet` 中包含請求中項目的主索引鍵值，以便協助按項目剖析回應。

如果請求項目不存在，則這些項目的回應中不會傳回任何內容。請求不存在項目會使用最小讀取容量單位，具體值根據讀取類型而定。如需詳細資訊，請參閱 [DynamoDB 項目大小和格式](#)。

## 請求

### 語法

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{"RequestItems":
 {"Table1":
 {"Keys":
 [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
 {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
 {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}]},
 "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
 "Table2":
 {"Keys":
```

```

 [{"HashKeyElement": {"S": "KeyValue4"}},
 {"HashKeyElement": {"S": "KeyValue5"}}],
 "AttributesToGet": ["AttributeName4", "AttributeName5", "AttributeName6"]
 }
}

```

名稱	描述	必要
RequestItems	<p>要以主索引鍵取得的資料表名稱和對應項目的容器。在請求項目時，每個資料表名稱在每次操作僅能呼叫一次。</p> <p>類型：字串</p> <p>預設：無</p>	是
Table	<p>包含所要取得的項目的資料表名稱。該項目只是一個字串，用以指定不帶標籤的現有資料表。</p> <p>類型：字串</p> <p>預設：無</p>	是
Table:Keys	<p>定義指定資料表中項目的主索引鍵值。如需主索引鍵的詳細資訊，請參閱 <a href="#">主索引鍵</a>。</p> <p>類型：金鑰</p>	是
Table:AttributesToGet	<p>指定資料表中的屬性名稱陣列。如果未指定屬性名稱，則會傳回所有屬性。如果有部分屬性未找到，則這些屬性不會出現在結果中。</p> <p>類型：陣列</p>	否

名稱	描述	必要
Table:ConsistentRead	<p>如果設定為 true，則會發送一致性讀取，反之則會使用最終一致性。</p> <p>類型：布林值</p>	否

## 回應

### 語法

```

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"]}
 }
 }
}

```

名稱	描述
Responses	<p>資料表名稱以及資料表個別項目屬性。</p> <p>類型：映射</p>
Table	<p>包含項目的資料表名稱。該項目只是一個字串，用以指定不帶標籤的資料表。</p> <p>類型：字串</p>
Items	<p>符合操作參數的屬性名稱和值的容器。</p> <p>類型：屬性名稱映射，以及其資料類型和值。</p>
ConsumedCapacityUnits	<p>每個資料表使用的讀取容量單位數目。此值顯示套用至佈建輸送量的數字。請求不存在項目會使用最小讀取容量單位，具體值依讀取類型而定。如需詳細資訊，請參閱 <a href="#">佈建容量模式</a>。</p> <p>類型：數字</p>
UnprocessedKeys	<p>包含可能因回應大小達到限制，而未在目前回應中處理的資料表及其各自的索引鍵的陣列。UnprocessedKeys 值形式與 RequestItems 參數相同 (因此該值可直接提供給後續 BatchGetItem 操作)。如需詳細資訊，請參閱上述 RequestItems 參數。</p>

名稱	描述
	類型：陣列
UnprocessedKeys : Table: Keys	定義項目及項目相關聯屬性的主索引鍵屬性值。如需主索引鍵的詳細資訊，請參閱 <a href="#">主索引鍵</a> 。  類型：屬性名稱值組陣列。
UnprocessedKeys : Table: AttributesToGet	指定資料表中的屬性名稱。如果未指定屬性名稱，則會傳回所有屬性。如果有部分屬性未找到，則這些屬性不會出現在結果中。  類型：屬性名稱陣列。
UnprocessedKeys : Table: ConsistentRead	如果設定為 true，則指定資料表會使用一致性讀取，反之則會使用最終一致讀取。  類型：布林值。

## 特殊錯誤

錯誤	描述
ProvisionedThroughputExceededException	已超出允許的佈建輸送量上限。

## 範例

下面的例子顯示了使用該 BatchGetItem 操作的 HTTP POST 請求和響應。如需使用 AWS SDK 的範例，請參閱 [使用項目和屬性](#)。

### 請求範例

下列範例請求兩個不同資料表的屬性。

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
```

```
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0
content-length: 409

{"RequestItems":
 {"comp1":
 {"Keys":
 [{"HashKeyElement":{"S":"Casey"},"RangeKeyElement":{"N":"1319509152"}},
 {"HashKeyElement":{"S":"Dave"},"RangeKeyElement":{"N":"1319509155"}},
 {"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"1319509158"}}],
 "AttributesToGet":["user","status"]},
 "comp2":
 {"Keys":
 [{"HashKeyElement":{"S":"Julie"}}, {"HashKeyElement":{"S":"Mingus"}}],
 "AttributesToGet":["user","friends"]}
 }
}
```

## 回應範例

下列範例為回應。

```
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

**##### API ## 2011-12-05#**

如需目前低階 API 的文件，請參閱[Amazon DynamoDB API 參考](#)。

## 描述

您可利用此操作，在單一呼叫中放入與刪除多個資料表的數個項目。

您可以使用 PutItem 上傳一個項目，並且使用 DeleteItem 刪除一個項目。不過，當您想上傳或刪除大量資料，例如從 Amazon EMR (Amazon EMR) 上傳大量資料，或將資料從另一個資料庫遷移至 DynamoDB，BatchWriteItem 提供了一種高效率的替代方法。

如果您使用 Java 之類的語言，則可以使用執行緒來平行上傳項目。這增加了應用程式處理執行緒的複雜性。其他語言不支持執行緒。例如，如果使用 PHP，則必須一次上傳或刪除一個項目。在這兩種情況下，BatchWriteItem 提供一種替代方法來平行處理指定的放入與刪除操作，讓您運用強大的執行緒集區方法，卻不會增加應用程式的複雜性。

請注意，BatchWriteItem 操作中指定的每一個放入和刪除會使用相同的容量單位。不過，由於 BatchWriteItem 平行執行指定操作，您會達到更低的延遲。刪除不存在項目上的操作會使用一個寫入容量單位。如需使用的容量單位的詳細資訊，請參閱 [在 DynamoDB 中使用資料表和資料](#)。

使用 BatchWriteItem 時，請注意以下限制：

- 單次請求中的操作上限：最多可以指定總共 25 項放入或刪除操作，不過請求大小總計不得超過 1 MB (HTTP 承載)。
- BatchWriteItem 操作僅可用於放入和刪除項目。您無法使用該操作來更新現有項目。
- 不是非敗即成操作：BatchWriteItem 當中指定的個別操作為非敗即成操作，不過 BatchWriteItem 整體而言是屬於最佳操作，而不是非敗即成操作。也就是說，在 BatchWriteItem 請求中，有些操作可能成功，有些操作可能失敗。失敗的操作會在回應中的 UnprocessedItems 欄位傳回。其中有些失敗可能肇因於超出為資料表所設定的佈建輸送量，或是例如網路錯誤的暫時性失敗。您可以調查並選擇性地重新傳送請求。一般而言，您會在迴圈中呼叫 BatchWriteItem，在每次更替中檢查未處理項目，並提交一個含有這些未處理項目的新的 BatchWriteItem 請求。

- 不傳回任何項目：BatchWriteItem 是設計來有效地上傳大量資料。這不會提供 PutItem 和 DeleteItem 的一些複雜性。例如，DeleteItem 支援要求主體文中的 ReturnValues 欄位請求回應中的刪除項目。BatchWriteItem 操作不會傳回回應中的任何項目。
- 不同於 PutItem 和 DeleteItem，BatchWriteItem 不允許在操作中指定個別寫入請求的條件。
- 屬性值不可為 Null；字串和二進位類型屬性的長度必須大於零；集合類型屬性不可為空白。具有空值的請求會遭到拒絕，並出現 ValidationException。

如果符合以下任一情況，DynamoDB 會拒絕整個批次寫入操作：

- BatchWriteItem 請求中指定的一或多個資料表不存在。
- 請求中的項目上指定的主索引鍵屬性不符合對應資料表的主索引鍵結構描述。
- 嘗試在同一 BatchWriteItem 請求中的同一個項目上執行多項操作。例如，您不能在同一 BatchWriteItem 請求中放入和刪除同一個項目。
- 請求大小總計超過 1 MB 的請求大小 (HTTP 承載) 限制。
- 批次中任一個項目超過 64 KB 的項目大小限制。

## 請求

### 語法

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
 "RequestItems" : RequestItems
}

RequestItems
{
 "TableName1" : [Request, Request, ...],
 "TableName2" : [Request, Request, ...],
 ...
}

Request ::=
 PutRequest | DeleteRequest
```

```
PutRequest ::=
{
 "PutRequest" : {
 "Item" : {
 "Attribute-Name1" : Attribute-Value,
 "Attribute-Name2" : Attribute-Value,
 ...
 }
 }
}

DeleteRequest ::=
{
 "DeleteRequest" : {
 "Key" : PrimaryKey-Value
 }
}

PrimaryKey-Value ::= HashTypePK | HashAndRangeTypePK

HashTypePK ::=
{
 "HashKeyElement" : Attribute-Value
}

HashAndRangeTypePK
{
 "HashKeyElement" : Attribute-Value,
 "RangeKeyElement" : Attribute-Value,
}

Attribute-Value ::= String | Numeric | Binary | StringSet | NumericSet | BinarySet

Numeric ::=
{
 "N": "Number"
}

String ::=
{
 "S": "String"
}
```

```
Binary ::=
{
 "B": "Base64 encoded binary data"
}

StringSet ::=
{
 "SS": ["String1", "String2", ...]
}

NumberSet ::=
{
 "NS": ["Number1", "Number2", ...]
}

BinarySet ::=
{
 "BS": ["Binary1", "Binary2", ...]
}
```

要求主體中的 RequestItems JSON 物件會描述您要執行的操作。這些操作依資料表分組。您可以使用 BatchWriteItem 來更新或刪除多個資料表的數個項目。對於每個特定寫入請求，您必須確定請求類型 (PutItem、DeleteItem)，後面附上操作詳細資訊。

- 對於 PutRequest，您要提供項目，也就是屬性及其值的清單。
- 對於 DeleteRequest，您要提供主索引鍵名稱和值。

## 回應

### 語法

以下是回應中傳回的 JSON 內文語法。

```
{
 "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.

## 特殊錯誤

沒有此操作特定的錯誤。

## 範例

以下範例顯示 BatchWriteItem 操作的 HTTP POST 請求和回應。該請求指定在 Reply (回覆) 和 Thread (主題) 資料表上執行下列操作：

- 在 Reply (回覆) 資料表中放入一個項目並刪除一個項目
- 將一個項目放入 Thread (主題) 資料表

如需使用 AWS 開發套件的範例，請參閱 [使用項目和屬性](#)。

## 請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
 "RequestItems":{
 "Reply":[
 {
 "PutRequest":{
 "Item":{
 "ReplyDateTime":{
 "S":"2012-04-03T11:04:47.034Z"
 },
 "Id":{
 "S":"DynamoDB#DynamoDB Thread 5"
 }
 }
 }
 }
]
 }
}
```

```
 }
 },
 {
 "DeleteRequest":{
 "Key":{
 "HashKeyElement":{
 "S":"DynamoDB#DynamoDB Thread 4"
 },
 "RangeKeyElement":{
 "S":"oops - accidental row"
 }
 }
 }
 },
 {
 "PutRequest":{
 "Item":{
 "ForumName":{
 "S":"DynamoDB"
 },
 "Subject":{
 "S":"DynamoDB Thread 5"
 }
 }
 }
 }
],
"Thread":[
 {
 "PutRequest":{
 "Item":{
 "ForumName":{
 "S":"DynamoDB"
 },
 "Subject":{
 "S":"DynamoDB Thread 5"
 }
 }
 }
 }
]
}
```

## 回應範例

下列範例回應顯示 Thread (主題) 和 Reply (回覆) 資料表上的放入操作成功，以及 Reply (回覆) 資料表上的刪除操作失敗 (例如因超出資料表上的佈建輸送量而導致調節等原因)。在 JSON 回應中，請注意下列事項：

- Responses 物件會顯示 Thread 和 Reply 資料表兩者都成功執行放入操作而使用一個容量單位。
- UnprocessedItems 物件則會顯示 Reply 資料表失敗的刪除操作。然後，您可以發出新的 BatchWriteItem 呼叫來解決這些未處理的請求。

```
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

**##### API ## 2011-12-05#**

如需目前低階 API 的文件，請參閱[Amazon DynamoDB API 參考](#)。

## 描述

CreateTable 操作會將新的資料表新增到帳戶。

在與發出請求的 AWS 帳戶以及接收請求的 AWS 區域相關聯的表格名稱中，表格名稱必須是唯一的。每個 DynamoDB 端點都是完全獨立的。例如，如果您有兩個名為 "的資料表 MyTable，一個位於西部-2.amazonaws.com 中，另一個在 Dynamodb.us-西 1.amazonaws.com 中，它們是完全獨立的，不會共用任何資料。

CreateTable 操作會觸發非同步工作流程以開始建立資料表。DynamoDB 會立即傳回資料表狀態 (CREATING)，直到資料表處於 ACTIVE 狀態。一旦資料表處於 ACTIVE 狀態，您便可執行資料平面操作。

使用 [DescribeTables](#) 操作來檢查資料表的狀態。

## 請求

### 語法


```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "KeySchema":
 {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
 "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
 "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

名稱	描述	必要
TableName	要建立的資料表名稱。  允許的字元為 a-z、A-Z、0-9、_ (底線)、- (破折號) 和 . (點)。名稱長度可介於 3 到 255 個字元之間。	是



名稱	描述	必要
	類型：字串	
KeySchema	<p>資料表的主索引鍵 (簡單或複合) 結構。需要 <code>HashKeyElement</code> 的名稱值組，且可選用 <code>RangeKeyElement</code> 的名稱值組 (僅適用於複合主索引鍵)。如需主索引鍵的詳細資訊，請參閱 <a href="#">主索引鍵</a>。</p> <p>主索引鍵元素名稱長度可以介於 1 到 255 個字元之間，且沒有字元限制。</p> <p>的可能值 <code>AttributeType</code> 為「S」(字串)、「N」(數字) 或「B」(二進位)。</p> <p>類型：複合主索引鍵的 <code>HashKeyElement</code> 或 <code>HashKeyElement</code> 和 <code>RangeKeyElement</code> 映射。</p>	是

名稱	描述	必要
ProvisionedThroughput	<p>指定資料表的新輸送量，包括 ReadCapacityUnits 與 WriteCapacityUnits 的值。如需詳細資訊，請參閱 <a href="#">佈建容量模式</a>。</p> <div data-bbox="591 495 1029 856" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px; margin: 10px 0;"> <p> <b>Note</b></p> <p>如需目前的最大/最小值，請參閱 <a href="#">Amazon DynamoDB 中的服務、帳戶和資料表配額</a>。</p> </div> <p>類型：陣列</p>	是
ProvisionedThroughput : ReadCapacityUnits	<p>設定 DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低一致性 ReadCapacityUnits 數目。</p> <p>最終一致讀取操作比一致性讀取負擔更輕，因此每秒 50 次一致性 ReadCapacityUnits 的設定可提供每秒 100 次最終一致 ReadCapacityUnits 。</p> <p>類型：數字</p>	是

名稱	描述	必要
ProvisionedThroughput : WriteCapacityUnits	<p>設定 DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低 WriteCapacityUnits 數目。</p> <p>類型：數字</p>	是

## 回應

### 語法

```
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"
 }
}
```

名稱	描述
TableDescription	資料表屬性的容器。
CreationDateTime	<p>建立資料表時的 <a href="#">UNIX epoch 格式</a> 日期。</p> <p>類型：數字</p>
KeySchema	資料表的主索引鍵 (簡單或複合) 結構。需要 HashKeyElement 的名稱值組，且可選用

名稱	描述
	<p><code>RangeKeyElement</code> 的名稱值組 (僅適用於複合主索引鍵)。如需主索引鍵的詳細資訊，請參閱 <a href="#">主索引鍵</a>。</p> <p>類型：複合主索引鍵的 <code>HashKeyElement</code> 或 <code>HashKeyElement</code> 和 <code>RangeKeyElement</code> 映射。</p>
<code>ProvisionedThroughput</code>	<p>指定資料表的輸送量，包括 <code>ReadCapacityUnits</code> 與 <code>WriteCapacityUnits</code> 的值。請參閱 <a href="#">佈建容量模式</a>。</p> <p>類型：陣列</p>
<code>ProvisionedThroughput :ReadCapacityUnits</code>	<p>DynamoDB 與其他操作平衡負載之前，每秒所需的最低 <code>ReadCapacityUnits</code> 數目。</p> <p>類型：數字</p>
<code>ProvisionedThroughput :WriteCapacityUnits</code>	<p><code>WriteCapacityUnits</code> 與其他操作平衡負載之前，每秒所需的最低 <code>ReadCapacityUnits</code> 數目。</p> <p>類型：數字</p>
<code>TableName</code>	<p>已建立的資料表名稱。</p> <p>類型：字串</p>
<code>TableStatus</code>	<p>資料表目前的狀態 (CREATING)。一旦資料表處於 ACTIVE 狀態，您便可將資料放入其中。</p> <p>使用 <a href="#">DescribeTables</a> API 來檢查資料表狀態。</p> <p>類型：字串</p>

## 特殊錯誤

錯誤	描述
ResourceInUseException	嘗試重新建立已存在的資料表。
LimitExceededException	同步資料表請求數 (CREATING、DELETING 或 UPDATING 狀態) 超出所允許的上限。  <div data-bbox="829 548 1507 814"><p> <b>Note</b> 如需目前的最大/最小值，請參閱 <a href="#">Amazon DynamoDB 中的服務、帳戶和資料表配額</a>。</p></div>

## 範例

以下範例會使用包含一個字串和數字的複合主索引鍵建立資料表。如需使用 AWS SDK 的範例，請參閱 [在 DynamoDB 中使用資料表和資料](#)。

### 請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0

{"TableName":"comp-table",
 "KeySchema":
 {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
 "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
 "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

## 回應範例

```
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"
 }
}
```

## 相關動作

- [DescribeTables](#)
- [DeleteTable](#)

## DeleteItem

### Important

**##### API ## 2011-12-05#**

如需目前低階 API 的文件，請參閱[Amazon DynamoDB API 參考](#)。

## 描述

依主索引鍵，刪除資料表中的單一項目。您可以執行條件式刪除操作，在項目存在或者項目具有預期的屬性值時予以刪除。

### Note

如果指定 DeleteItem 而沒有屬性或值，則會刪除項目的所有屬性。

在未指定條件的情況下，DeleteItem 一律為等冪操作，在同一個項目或屬性上多次執行不會導致錯誤回應。

條件式刪除僅能用於在符合特定條件時刪除項目和屬性。如果符合條件，DynamoDB 會執行刪除。反之則不會刪除項目。

您可以在每次操作對一個屬性執行預期的條件式檢查。

## 請求

### 語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0


{"TableName":"Table1",
 "Key":
 {"HashKeyElement":{"S":"AttributeValue1"},"RangeKeyElement":
{"N":"AttributeValue2"}},
 "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3"}}},
 "ReturnValues":"ALL_OLD"}
}
```

名稱	描述	必要
TableName	包含要刪除項目的資料表名稱。  類型：字串	是
Key	定義項目的主索引鍵。如需主索引鍵的詳細資訊，請參閱 <a href="#">主索引鍵</a> 。  類型：HashKeyElement 對其值的映射和 RangeKeyElement 對其值的映射。	是

名稱	描述	必要
Expected	<p>指定條件式刪除的屬性。Expected 參數可讓您提供屬性名稱，以及 DynamoDB 是否應該檢查屬性值在刪除前是否具有特定值。</p> <p>類型：屬性名稱映射。</p>	否
Expected:Attribute Name	<p>條件式放置的屬性的名稱。</p> <p>類型：字串</p>	否



名稱	描述	必要
Expected:Attribute Name: ExpectedA ttributeValue	<p>使用此參數來指定屬性名稱值的值是否已經存在。</p> <p>如果該項目的 Color (顏色) 屬性不存在，則下列 JSON 符號會刪除項目：</p> <pre data-bbox="594 520 1029 680">"Expected" :   {"Color":{"Exists":false}}</pre> <p>下列 JSON 符號會在刪除項目之前檢查名為 Color (顏色) 的屬性是否具有現有值 Yellow (黃色)：</p> <pre data-bbox="594 932 1029 1134">"Expected" :   {"Color":{"Exists":true}, {"Value": {"S":"Yellow"}}</pre> <p>根據預設，如果使用 Expected 參數並提供 Value，DynamoDB 會假設屬性存在，且有要取代的目前值。所以您不必指定 {"Exists":true}，因為這是暗含的。您可以將請求縮短為：</p> <pre data-bbox="594 1575 1029 1734">"Expected" :   {"Color":{"Value": {"S":"Yellow"}}</pre>	否

名稱	描述	必要
	<p> <b>Note</b></p> <p>如果指定 {"Exists":true} 而無要檢查的屬性值，則 DynamoDB 會傳回錯誤。</p>	
ReturnValues	<p>如果要在刪除前取得屬性名稱值組，請使用此參數。可能的參數值為 NONE (預設) 或 ALL_OLD。如果指定 ALL_OLD，則會傳回舊項目的內容。如果未提供此參數，或者參數為 NONE，則不會傳回任何內容。</p> <p>類型：字串</p>	否

## 回應

### 語法

```

HTTP/1.1 200 OK
x-amzn-RequestId: CS0C7TJPLR000KIRLGOHVAICUFVV4KQNS05AEMVJF66Q9ASUAAJG
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
}
```

名稱	描述
Attributes	<p>如果 ReturnValues 參數在請求中提供為 ALL_OLD，則 DynamoDB 會傳回屬性名稱值組陣列 (本質上為已刪除項目)。反之則會在回應包含空集合。</p> <p>類型：屬性名稱值組陣列。</p>
ConsumedCapacityUnits	<p>操作所使用的寫入容量單位數目。此值顯示套用至佈建輸送量的數字。刪除不存在項目上的請求會使用 1 個寫入容量單位。如需詳細資訊，請參閱 <a href="#">佈建容量模式</a>。</p> <p>類型：數字</p>

## 特殊錯誤

錯誤	描述
ConditionalCheckFailedException	條件式檢查失敗。找不到預期的屬性值。

## 範例

### 請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"comp-table",
 "Key":
 {"HashKeyElement":{"S":"Mingus"},"RangeKeyElement":{"N":"200"}},
 "Expected":
 {"status":{"Value":{"S":"shopping"}}},
 "ReturnValues":"ALL_OLD"}
```

```
}
```

## 回應範例

```
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
}
```

## 相關動作

- [PutItem](#)

## DeleteTable

### Important

**##### API ## 2011-12-05#**

如需目前低階 API 的文件，請參閱[Amazon DynamoDB API 參考](#)。

## 描述

DeleteTable 操作會刪除資料表及其所有項目。在發出 DeleteTable 請求後，指定的資料表會處於 DELETING 狀態，直到 DynamoDB 完成刪除為止。如果資料表處於 ACTIVE 狀態，則可予以刪除。如果資料表處於 CREATING 或 UPDATING 狀態，則 DynamoDB 會傳回 ResourceInUseException 錯誤。如果指定的資料表不存在，則 DynamoDB 會傳回 ResourceNotFoundException。如果資料表已處於 DELETING 狀態，則不會傳回任何錯誤。

**Note**

DynamoDB 可能會繼續接受 DELETING 狀態的資料表的資料平面操作請求 (例如 GetItem 和 PutItem) , 直到資料表刪除完成。

在與發出請求的 AWS 帳戶以及接收請求的 AWS 區域相關聯的表格中是唯一的表格 (例如 : 西部-1.amazonaws.com)。每個 DynamoDB 端點都是完全獨立的。舉例來說, 如果您有兩個名為 "的資料表MyTable, 一個位於 Dynamodb.us-西-2.amazonaws.com, 另一個在 Dynamodb.us-西-1.amazonaws.com 中, 它們是完全獨立的, 不會共用任何資料; 刪除其中一個並不會刪除另一個資料。

使用 [DescribeTables](#) 操作來檢查資料表的狀態。

**請求****語法**

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

名稱	描述	必要
TableName	要刪除的資料表的名稱。  類型 : 字串	是

**回應****語法**

```
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"
 }
}

```

名稱	描述
TableDescription	資料表屬性的容器。
CreationDateTime	建立資料表時的日期。  類型：數字
KeySchema	資料表的主索引鍵 (簡單或複合) 結構。需要 HashKeyElement 的名稱值組，且可選用 RangeKeyElement 的名稱值組 (僅適用於複合主索引鍵)。如需主索引鍵的詳細資訊，請參閱 <a href="#">主索引鍵</a> 。  類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement 映射。
ProvisionedThroughput	指定資料表的輸送量，包括 ReadCapacityUnits 與 WriteCapacityUnits 的值。請參閱 <a href="#">佈建容量模式</a> 。
ProvisionedThroughput : ReadCapacityUnits	DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低 ReadCapacityUnits 數目。

名稱	描述
	類型：數字
ProvisionedThroughput : WriteCapacityUnits	DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低 WriteCapacityUnits 數目。  類型：數字
TableName	已刪除的資料表的名稱。  類型：字串
TableStatus	資料表目前的狀態 (DELETING)。一旦資料表遭刪除，後續對資料表的請求會傳回 resource not found 。  使用 <a href="#">DescribeTables</a> 操作來檢查資料表的狀態。  類型：字串

## 特殊錯誤

錯誤	描述
ResourceInUseException	資料表處於狀態 CREATING 或 UPDATING，而且無法刪除。

## 範例

### 請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0
content-length: 40
```

```
{"TableName":"favorite-movies-table"}
```

## 回應範例

```
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"
 }
}
```

## 相關動作

- [CreateTable](#)
- [DescribeTables](#)

## DescribeTables

### Important

**##### API ## 2011-12-05#**

如需目前低階 API 的文件，請參閱[Amazon DynamoDB API 參考](#)。

## 描述

返回有關表的信息，包括表的當前狀態，主鍵模式和創建表的時間。DescribeTable 結果最終是一致的。如果您在 DescribeTable 建立表格的過程中過早使用，DynamoDB 會傳回 ResourceNotFoundException 如果您在更新資料表的過程中使用 DescribeTable 太早，新值可能無法立即使用。



## 請求

### 語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

名稱	描述	必要
TableName	要描述的資料表的名稱。  類型：字串	是

## 回應

### 語法

```
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"
 }
```

}

名稱	描述
Table	<p>正在描述的資料表的容器。</p> <p>類型：字串</p>
CreationDateTime	<p>建立資料表時的 <a href="#">UNIX epoch 格式</a> 日期。</p>
ItemCount	<p>指定資料表中的項目數。DynamoDB 大約每六個小時會更新一次此值。此值可能不會反映最近的變更。</p> <p>類型：數字</p>
KeySchema	<p>資料表的主索引鍵 (簡單或複合) 結構。需要 <code>HashKeyElement</code> 的名稱值組，且可選用 <code>RangeKeyElement</code> 的名稱值組 (僅適用於複合主索引鍵)。雜湊索引鍵大小上限為 2048 個位元組。範圍索引鍵大小上限為 1024 個位元組。此兩項限制皆為分別執行 (意即可出現雜湊 + 範圍 2048 + 1024 位元的合併索引鍵)。如需主索引鍵的詳細資訊，請參閱 <a href="#">主索引鍵</a>。</p>
ProvisionedThroughput	<p>指定資料表的輸送量，包含 <code>LastIncreaseDateTime</code> (如適用)、<code>LastDecreaseDateTime</code> (如適用)、<code>ReadCapacityUnits</code> 和 <code>WriteCapacityUnits</code> 的值。如果資料表輸送量從未增加或減少，則 DynamoDB 不會傳回這些元素的值。請參閱 <a href="#">佈建容量模式</a>。</p> <p>類型：陣列</p>
TableName	<p>所請求資料表的名稱。</p> <p>類型：字串</p>

名稱	描述
TableSizeBytes	所指定資料表的總大小 (以位元組為單位)。 DynamoDB 大約每六個小時會更新一次此值。此值可能不會反映最近的變更。  類型：數字
TableStatus	資料表目前的狀態 (CREATING、ACTIVE、DELETING 或 UPDATING)。一旦資料表處於 ACTIVE 狀態，您便可新增資料。

## 特殊錯誤

沒有此操作特定的錯誤。

## 範例

下列範例顯示 HTTP POST 要求和回應使用名為「補充資料表」資料表的 DescribeTable 作業。資料表具有複合主索引鍵。

### 請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"users"}
```

### 回應範例

```
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"
 }
}
```

## 相關動作

- [CreateTable](#)
- [DeleteTable](#)
- [ListTables](#)

## GetItem

### Important

**##### API ## 2011-12-05#**

如需目前低階 API 的文件，請參閱[Amazon DynamoDB API 參考](#)。

## 描述

GetItem 操作會傳回與主索引鍵相符的項目的一組 Attributes。如果沒有符合的項目，則 GetItem 不會傳回任何資料。

GetItem 操作預設會執行最終一致讀取。如果應用程式不接受最終一致讀取，則請使用 ConsistentRead。此操作可能比標準讀取需要更長的時間，但一定會傳回上次更新的值。如需詳細資訊，請參閱 [讀取一致性](#)。

## 請求

### 語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName": "Table1",
 "Key":
 {"HashKeyElement": {"S": "AttributeValue1"},
 "RangeKeyElement": {"N": "AttributeValue2"}
},
 "AttributesToGet": ["AttributeName3", "AttributeName4"],
 "ConsistentRead": Boolean
}
```

名稱	描述	必要
TableName	<p>包含所請求項目的資料表的名稱。</p> <p>類型：字串</p>	是
Key	<p>定義項目的主索引鍵值。如需主索引鍵的詳細資訊，請參閱<a href="#">主索引鍵</a>。</p> <p>類型：HashKeyElement 對其值的映射和 RangeKeyElement 對其值的映射。</p>	是
AttributesToGet	<p>屬性名稱陣列。如果未指定屬性名稱，則會傳回所有屬性。如果有部分屬性未找到，則這些屬性不會出現在結果中。</p> <p>類型：陣列</p>	否

名稱	描述	必要
ConsistentRead	<p>如果設定為 true，則會發送一致性讀取，反之則會使用最終一致性。</p> <p>類型：布林值</p>	否

## 回應

### 語法

```
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
}
```

名稱	描述
Item	<p>包含所請求屬性。</p> <p>類型：屬性名稱值組映射。</p>
ConsumedCapacityUnits	<p>操作所使用的讀取容量單位數目。此值顯示套用至佈建輸送量的數字。請求不存在項目會使用最小讀取容量單位，具體值依讀取類型而定。如需詳細資訊，請參閱 <a href="#">佈建容量模式</a>。</p> <p>類型：數字</p>

## 特殊錯誤

沒有此操作特定的錯誤。

## 範例

如需使用 AWS SDK 的範例，請參閱[使用項目和屬性](#)。

### 請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"comptable",
 "Key":
 {"HashKeyElement":{"S":"Julie"},
 "RangeKeyElement":{"N":"1307654345"}},
 "AttributesToGet":["status","friends"],
 "ConsistentRead":true
}
```

### 回應範例

請注意，ConsumedCapacityUnits 值為 1，因為可選參數設定ConsistentRead為true。如果針ConsistentRead對相同要求設定為 false (或未指定)，則回應最終會保持一致，且ConsumedCapacityUnits 值為 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

##### API ## 2011-12-05#

如需目前低階 API 的文件，請參閱[Amazon DynamoDB API 參考](#)。

### 描述

傳回與目前帳戶及端點相關聯之所有資料表陣列。

每個 DynamoDB 端點都是完全獨立的。例如，如果有兩個名為 MyTable 的資料表，一個在 dynamodb.us-west-2.amazonaws.com 中，另一個在 dynamodb.us-east-1.amazonaws.com 中，則兩者完全獨立，不會共用任何資料。ListTables 操作會針對接收請求的端點，傳回與發出請求之帳戶相關聯的所有資料表名稱。

### 請求

### 語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"Table1","Limit":3}
```

根據預設，ListTables 操作會針對接收請求的端點，請求與發出請求之帳戶相關聯的所有資料表名稱。

名稱	描述	必要
Limit	要傳回的資料表名稱上限數。  類型：Integer	否
ExclusiveStartTableName	清單之首的資料表名稱。如果您已執行 ListTables 操作並收	否



名稱	描述	必要
	<p>到 LastEvaluatedTableName 值，請在此處使用該值繼續清單。</p> <p>類型：字串</p>	

## 回應

### 語法

```
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"}
```

名稱	描述
TableNames	<p>在目前端點上與目前帳戶相關聯的資料表名稱。</p> <p>類型：陣列</p>
LastEvaluatedTableName	<p>目前清單中最後一個資料表的名稱，這只在帳戶和端點的部分資料表尚未傳回時才需要。如果已傳回所有資料表名稱，則回應中不存在此值。在新的請求中使用此值作為 ExclusiveStartTableName 來繼續清單，直到傳回所有資料表名稱。</p> <p>類型：字串</p>

## 特殊錯誤

沒有此操作特定的錯誤。

## 範例

下列範例顯示使用 ListTables 操作的 HTTP POST 請求和回應。

### 請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"comp2","Limit":3}
```

### 回應範例

```
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"]}
```

## 相關動作

- [DescribeTables](#)
- [CreateTable](#)
- [DeleteTable](#)

## PutItem

### Important

**##### API ## 2011-12-05#**

如需目前低階 API 的文件，請參閱 [Amazon DynamoDB API 參考](#)。

## 描述

建立新項目，或以新項目取代舊項目 (包括全部屬性)。如果具有相同主索引鍵的指定資料表中已存在某項目，則新項目會完全取代現有項目。您可以執行條件式放置 (如果具有所指定主索引鍵的項目不存在，則插入新項目)，或者在現有項目具有某些屬性值時取代項目。

屬性值不得為 Null；字串和二進位類型屬性的長度必須大於零；集合類型屬性不可為空白。具有空值的請求會遭到拒絕，並出現 `ValidationException`。

### Note

若要確保新項目不會取代現有項目，請使用條件式放置操作，針對主索引鍵屬性或屬性將 `Exists` 設定為 `false`。

如需有關使用 `PutItem` 的詳細資訊，請參閱 [使用項目和屬性](#)。

## 請求

### 語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "Item":{
 "AttributeName1":{"S":"AttributeValue1"},
 "AttributeName2":{"N":"AttributeValue2"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
 "Expected":{"AttributeName3":{"Value": {"S":"AttributeValue"}, "Exists":Boolean}},
 "ReturnValues":"ReturnValuesConstant"}
```

名稱	描述	必要
TableName	要包含項目的資料表名稱。  類型：字串	是

名稱	描述	必要
Item	<p>項目屬性映射，且必須包含定義項目的主索引鍵值。可提供其他屬性名稱值組給項目。如需主索引鍵的詳細資訊，請參閱 <a href="#">主索引鍵</a>。</p> <p>類型：屬性值的屬性名稱映射。</p>	是
Expected	<p>指定條件式放置的屬性。Expected 參數可讓您提供屬性名稱，以及 DynamoDB 是否應該檢查屬性值是否已存在；或者屬性值是否存在且在變更之前有特定值。</p> <p>類型：屬性值的屬性名稱映射，以及映射是否存在。</p>	否
Expected:Attribute Name	<p>條件式放置的屬性的名稱。</p> <p>類型：字串</p>	否

名稱	描述	必要
Expected:Attribute Name: ExpectedAttributeValue	<p>使用此參數來指定屬性名稱值的值是否已經存在。</p> <p>如果該項目的 Color (顏色) 屬性尚未存在，則下列 JSON 符號會取代項目：</p> <pre>"Expected" :   {"Color":{"Exists":false}}</pre> <p>下列 JSON 符號會在取代項目之前檢查名為 Color (顏色) 的屬性是否具有現有值 Yellow (黃色)：</p> <pre>"Expected" :   {"Color":{"Exists":true, {"Value":{"S":"Yellow"}}}}</pre> <p>根據預設，如果使用 Expected 參數並提供 Value，DynamoDB 會假設屬性存在，且有要取代的目前值。所以您不必指定 {"Exists":true}，因為這是暗含的。您可以將請求縮短為：</p> <pre>"Expected" :   {"Color":{"Value":{"S":"Yellow"}}}}</pre>	否

名稱	描述	必要
	<p> <b>Note</b></p> <p>如果指定 {"Exists":true} 而無要檢查的屬性值，則 DynamoDB 會傳回錯誤。</p>	
ReturnValues	<p>如果想在以 PutItem 請求更新屬性名稱值組之前取得屬性名稱值組，則請使用此參數。可能的參數值為 NONE (預設) 或 ALL_OLD。如果已指定 ALL_OLD，且 PutItem 覆寫屬性名稱值組，則會傳回舊項目的內容。如果未提供此參數，或者參數為 NONE，則不會傳回任何內容。</p> <p>類型：字串</p>	否

## 回應

### 語法

下列語法範例假設請求指定 ALL\_OLD 的 ReturnValues 參數，反之則回應只有 ConsumedCapacityUnits 元素。

```
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
}
```

名稱	描述
Attributes	執行放置操作之前的屬性值，但僅限於請求中指定 ReturnValues 參數為 ALL_OLD 的情況。  類型：屬性名稱值組映射。
ConsumedCapacityUnits	操作所使用的寫入容量單位數目。此值顯示套用至佈建輸送量的數字。如需詳細資訊，請參閱 <a href="#">佈建容量模式</a> 。  類型：數字

## 特殊錯誤

錯誤	描述
ConditionalCheckFailedException	條件式檢查失敗。找不到預期的屬性值。
ResourceNotFoundException	找不到指定的項目或屬性。

## 範例

如需使用 AWS SDK 的範例，請參閱 [使用項目和屬性](#)。

### 請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
```

```
"Item":
 {"time":{"N":"300"},
 "feeling":{"S":"not surprised"},
 "user":{"S":"Riley"}
 },
"Expected":
 {"feeling":{"Value":{"S":"surprised"},"Exists":true}}
"ReturnValues":"ALL_OLD"
}
```

## 回應範例

```
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
}
```

## 相關動作

- [UpdateItem](#)
- [DeleteItem](#)
- [GetItem](#)
- [BatchGetItem](#)

## Query

### Important

**##### API ## 2011-12-05#**

如需目前低階 API 的文件，請參閱[Amazon DynamoDB API 參考](#)。



## 描述

Query 作業會依主索引鍵取得一或多個項目的值及其屬性 (Query 僅適用於 hash-and-range 主索引鍵資料表)。您必須提供特定的 HashKeyValue，且能在主索引鍵的 RangeKeyValue 上使用比較運算子縮小查詢範圍。使用 ScanIndexForward 參數以範圍索引鍵取得正向或反向順序的結果。

未傳回結果的查詢會根據讀取類型使用最小讀取容量單位。

### Note

如果符合查詢參數的項目總數超過 1MB 限制，則查詢會停止，並將結果傳回給使用者，附帶 LastEvaluatedKey 用以在後續操作中繼續查詢。查詢操作不同於掃描操作，一律不會傳回空的結果集和 LastEvaluatedKey。只有在結果超過 1MB，或者您已使用 Limit 參數時，才會提供 LastEvaluatedKey。

使用 ConsistentRead 參數可設定一致性讀取的結果。

## 請求

### 語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "Limit":2,
 "ConsistentRead":true,
 "HashKeyValue":{"S":"AttributeValue1":},
 "RangeKeyCondition": {"AttributeValueList":
 [{"N":"AttributeValue2"}], "ComparisonOperator":"GT"}
 "ScanIndexForward":true,
 "ExclusiveStartKey":{
 "HashKeyElement":{"S":"AttributeName1"},
 "RangeKeyElement":{"N":"AttributeName2"}
 },
 "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

名稱	描述	必要
TableName	<p>包含請求項目的資料表的名稱。</p> <p>類型：字串</p>	是
AttributesToGet	<p>屬性名稱陣列。如果未指定屬性名稱，則會傳回所有屬性。如果有部分屬性未找到，則這些屬性不會出現在結果中。</p> <p>類型：陣列</p>	否
Limit	<p>要傳回的項目數上限 (不一定是相符項目數)。如果 DynamoDB 在查詢資料表時處理的項目數達上限，則會停止查詢，並傳回截至該時間點的相符值，附帶 LastEvaluatedKey 以套用於後續操作中繼續查詢。此外，如果在 DynamoDB 達到此限制之前結果集大小超過 1MB，則會停止查詢並傳回相符值，附帶 LastEvaluatedKey 以套用於後續操作中繼續查詢。</p> <p>類型：數字</p>	否
ConsistentRead	<p>如果設定為 true，則會發送一致性讀取，反之則會使用最終一致性。</p> <p>類型：布林值</p>	否
Count	<p>如果設定為 true，則 DynamoDB 會傳回符合查詢參數的項目總數，而不是相符項</p>	否

名稱	描述	必要
	<p>目及其屬性的清單。您可以套用 Limit 參數至僅計數查詢。</p> <p>請勿在提供 Attribute sToGet 清單之時將 Count 設定為 true，否則 DynamoDB 會傳回驗證錯誤。如需詳細資訊，請參閱 <a href="#">計算結果中的項目</a>。</p> <p>類型：布林值</p>	
HashKeyValue	<p>複合主索引鍵雜湊元件的屬性值。</p> <p>類型：字串、數字或二進位</p>	是
RangeKeyCondition	<p>屬性值容器以及用於查詢的比較運算子。查詢請求不需要 RangeKeyCondition。如果只提供 HashKeyValue，則 DynamoDB 會傳回具有指定雜湊索引鍵元素值的所有項目。</p> <p>類型：映射</p>	否

名稱	描述	必要
RangeKeyCondition : AttributeValueList	<p>查詢參數要評估的屬性值。除非已指定 BETWEEN 比較，否則 AttributeValueList 一律包含一個屬性值。對於 BETWEEN 比較，AttributeValueList 包含兩個屬性值。</p> <p>類型 : ComparisonOperator 的 AttributeValue 映射。</p>	否

名稱	描述	必要
RangeKeyCondition : ComparisonOperator	<p data-bbox="594 226 1008 352">用於評估所提供的屬性的條件，例如等於、大於。以下是查詢操作的有效比較運算子。</p> <div data-bbox="594 401 1029 1241" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px;"><p data-bbox="623 436 740 470"> Note</p><p data-bbox="672 491 992 1192">大於、等於或小於的字串值比較是根據 ASCII 字元代碼值。例如，a 大於 A，aa 大於 B。如需代碼值的清單，請參閱 <a href="http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters">http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters</a>。針對 Binary，每當 DynamoDB 比較二進位值，例如在評估查詢表達式時，都會將二進位資料的每個位元組視為不帶正負號。</p></div> <p data-bbox="594 1308 878 1341">類型：字串或二進位</p>	否

名稱	描述	必要
	<p>EQ：等於。</p> <p>對於 EQ，Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不等於 {"NS":["6", "2", "1"]}。</p>	
	<p>LE：小於或等於。</p> <p>對於 LE，Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。</p>	

名稱	描述	必要
	<p>LT：小於。</p> <p>對於 LT，Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。</p>	
	<p>GE：大於或等於。</p> <p>對於 GE，Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。</p>	

名稱	描述	必要
	<p>GT : 大於。</p> <p>對於 GT , Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。</p>	
	<p>BEGINS_WITH : 檢查字首。</p> <p>對於 BEGINS_WITH , Attribute ValueList 僅可包含 String 或 Binary 類型 (非 Number 或集合) 的一個 AttributeValue 。比較的目標屬性必須是 String 或 Binary (非 Number 或集合)。</p>	



名稱	描述	必要
	<p>BETWEEN：大於或等於第一個數值，並且小於或等於第二個數值。</p> <p>對於 BETWEEN，Attribute ValueList 必須包含 String、Number 或 Binary 類型 (非集合) 的兩個 AttributeValue 元素。如果目標數值大於或等於第一個元素，並且小於或等於第二個元素，則目標屬性相符。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不與 {"N":"6"} 比較。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]}] 比較。</p>	
ScanIndexForward	<p>指定遞增或遞減的索引周遊。DynamoDB 傳回結果會反映由範圍索引鍵決定之所請求順序：如果資料類型為數字，則會依數值順序傳回結果，反之，周遊則是根據 ASCII 字元代碼值。</p> <p>類型：布林值</p> <p>預設為 true (遞增)。</p>	否

名稱	描述	必要
ExclusiveStartKey	<p>繼續先前查詢之項目的主索引鍵。如果因為結果集大小或 Limit 參數，導致查詢操作在查詢完成前中斷，較早的查詢可能提供此值為 LastEvaluatedKey。LastEvaluatedKey 可以在新的查詢請求中傳回，以便從該時間點繼續操作。</p> <p>類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement。</p>	否

## 回應

### 語法

```

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"},
"AttributeNames":{"B":"dmFsdWU="}
}],
"LastEvaluatedKey":{"HashKeyElement":{"AttributeNames":{"S":"AttributeValue3"},
"AttributeNames":{"N":"AttributeValue4"}},
"RangeKeyElement":{"AttributeNames":{"N":"AttributeValue4"}},
"ConsumedCapacityUnits":1

```

}

名稱	描述
Items	<p>符合查詢參數的項目屬性。</p> <p>類型：屬性名稱映射，以及其資料類型和值。</p>
Count	<p>回應中的項目數。如需詳細資訊，請參閱 <a href="#">計算結果中的項目</a>。</p> <p>類型：數字</p>
LastEvaluatedKey	<p>查詢操作停止的項目的主索引鍵，包括先前的結果集。使用此值開始新操作，在新的請求中排除此值。</p> <p>當整個查詢結果集完成時 (意即操作已處理「最後一頁」)，LastEvaluatedKey 為 null。</p> <p>類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement 。</p>
ConsumedCapacityUnits	<p>操作所使用的讀取容量單位數目。此值顯示套用至佈建輸送量的數字。如需詳細資訊，請參閱 <a href="#">佈建容量模式</a>。</p> <p>類型：數字</p>

## 特殊錯誤

錯誤	描述
ResourceNotFoundException	找不到指定的資料表。

## 範例

如需使用 AWS SDK 的範例，請參閱 [DynamoDB 中的查詢操作](#)。

## 請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"John"},
 "ScanIndexForward":false,
 "ExclusiveStartKey":{"
 "HashKeyElement":{"S":"John"},
 "RangeKeyElement":{"S":"The Matrix"}
 }
}
```

## 回應範例

```
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
}
```

## 請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"Airplane"},
 "RangeKeyCondition":{"AttributeValueList":[{"N":"1980"}],"ComparisonOperator":"EQ"},
 "ScanIndexForward":false}
```

## 回應範例

```
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
}
```

## 相關動作

- [Scan](#)

## Scan

### Important

**##### API ## 2011-12-05#**

如需目前低階 API 的文件，請參閱[Amazon DynamoDB API 參考](#)。

## 描述

Scan 操作會執行資料表完整掃描來傳回一或多個項目及其屬性。提供 ScanFilter 以取得更特定的結果。

### Note

如果已掃描項目總數超過 1MB 限制，則掃描會停止，並將結果傳回給使用者，附帶 LastEvaluatedKey 用以在後續操作中繼續掃描。結果還包括超出限制的項目數。掃描可能導致表內資料皆不符合篩選條件。結果集為最終一致性。

## 請求

### 語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "Limit": 2,
 "ScanFilter":{
 "AttributeName":{"AttributeValueList":
[{"S":"AttributeValue"}],"ComparisonOperator":"EQ"}
 },
 "ExclusiveStartKey":{
 "HashKeyElement":{"S":"AttributeName"},
 "RangeKeyElement":{"N":"AttributeName2"}
 },
 "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

名稱	描述	必要
TableName	包含請求項目的資料表的名稱。	是

名稱	描述	必要
	類型：字串	
AttributesToGet	屬性名稱陣列。如果未指定屬性名稱，則會傳回所有屬性。如果有部分屬性未找到，則這些屬性不會出現在結果中。  類型：陣列	否
Limit	要評估的項目數上限 (不一定是相符的項目數)。如果 DynamoDB 在處理結果時處理的項目數達上限，則會停止工作，並傳回截至該時間點的相符值，附帶 LastEvaluatedKey 以套用於後續操作中繼續擷取項目。此外，如果在 DynamoDB 達到此限制之前已掃描的資料集大小超過 1MB，則會停止掃描，並傳回截至上限的相符值，附帶 LastEvaluatedKey 以套用於後續操作中繼續掃描。  類型：數字	否

名稱	描述	必要
Count	<p>如果設定為 true，則 DynamoDB 會傳回掃描操作的項目總數，即使操作沒有指派篩選條件的相符項目也一樣。您可以套用限制參數至僅計數查詢。</p> <p>請勿在提供 Attribute sToGet 清單之時將 Count 設定為 true，否則 DynamoDB 會傳回驗證錯誤。如需詳細資訊，請參閱 <a href="#">計算結果中的項目</a>。</p> <p>類型：布林值</p>	否
ScanFilter	<p>評估掃描結果，並僅傳回所需值。多個條件會視為 AND 操作：必須符合所有條件才會納入結果中。</p> <p>類型：具有比較運算子的值的屬性名稱映射。</p>	否
ScanFilter :Attribute ValueList	<p>用來評估篩選條件掃描結果的值和條件。</p> <p>類型：Condition 的 AttributeValue 映射。</p>	否



名稱	描述	必要
ScanFilter : ComparisonOperator	<p>用於評估所提供的屬性的條件，例如等於、大於。以下是掃描操作的有效比較運算子。</p> <div data-bbox="591 396 1029 1239" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px;"><p> <b>Note</b></p><p>大於、等於或小於的字串值比較是根據 ASCII 字元代碼值。例如，a 大於 A，aa 大於 B。如需代碼值的清單，請參閱 <a href="http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters">http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters</a>。</p><p>針對 Binary，每當 DynamoDB 比較二進位值，例如在評估查詢表達式時，都會將二進位資料的每個位元組視為不帶正負號。</p></div> <p>類型：字串或二進位</p>	否

名稱	描述	必要
	<p>EQ：等於。</p> <p>對於 EQ，Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不等於 {"NS":["6", "2", "1"]}。</p>	
	<p>NE：不等於。</p> <p>對於 NE，Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不等於 {"NS":["6", "2", "1"]}。</p>	

名稱	描述	必要
	<p>LE：小於或等於。</p> <p>對於 LE，Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。</p>	
	<p>LT：小於。</p> <p>對於 LT，Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"}。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。</p>	

名稱	描述	必要
	<p>GE : 大於或等於。</p> <p>對於 GE , Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。</p>	
	<p>GT : 大於。</p> <p>對於 GT , Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不等於 {"N":"6"} 。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。</p>	
	NOT_NULL : 屬性存在。	
	NULL : 屬性不存在。	

名稱	描述	必要
	<p>CONTAINS：檢查子序列，或是集合中的數值。</p> <p>對於 CONTAINS，Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue。如果比較的目標屬性是字串，則操作會檢查相符子字串。如果比較的目標屬性是二進位，則操作會尋找與輸入相符的目標子序列。如果比較的目標屬性是集合 (SS、NS 或 BS)，則操作會檢查集合的成員 (不當作子字串)。</p>	

名稱	描述	必要
	<p>NOT_CONTAINS : 檢查子序列是否不存在，或者集合中的數值是否不存在。</p> <p>對於 NOT_CONTAINS , Attribute ValueList 僅可包含 String、Number 或 Binary 類型 (非集合) 的一個 AttributeValue 。如果比較的目標屬性是字串，則操作會檢查相符子字串是否不存在。如果比較的目標屬性是二進位，則操作會檢查與輸入相符的目標子序列是否不存在。如果比較的目標屬性是集合 (SS、NS 或 BS) ，則操作會檢查集合的成員 (不當作子字串) 是否不存在。</p>	
	<p>BEGINS_WITH : 檢查字首。</p> <p>對於 BEGINS_WITH , Attribute ValueList 僅可包含 String 或 Binary 類型 (非 Number 或集合) 的一個 AttributeValue 。比較的目標屬性必須是 String 或 Binary (非 Number 或集合)。</p>	

名稱	描述	必要
	<p>IN：檢查完全相符項目。</p> <p>對於 IN，Attribute ValueList 可以包含 String、Number 或 Binary 類型 (非集合) 的多個 AttributeValue。比較的目標屬性必須為相同的類型和確切數值才相符。String 一律不與 String 集合相符。</p>	
	<p>BETWEEN：大於或等於第一個數值，並且小於或等於第二個數值。</p> <p>對於 BETWEEN，Attribute ValueList 必須包含 String、Number 或 Binary 類型 (非集合) 的兩個 AttributeValue 元素。如果目標數值大於或等於第一個元素，並且小於或等於第二個元素，則目標屬性相符。如果項目內含的 Attribute Value 所屬類型與請求中指定的類型不同，則數值不相符。例如，{"S":"6"} 不與 {"N":"6"} 比較。另外，{"N":"6"} 不與 {"NS":["6", "2", "1"]} 比較。</p>	

名稱	描述	必要
ExclusiveStartKey	<p>繼續先前掃描之項目的主索引鍵。如果因為結果集大小或 Limit 參數，導致掃描操作在整個資料表掃描完之前中斷，較早的掃描可能提供此值。LastEvaluatedKey 可以在新的掃描請求中傳回，以便從該時間點繼續操作。</p> <p>類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement 。</p>	否

## 回應

### 語法

```

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"},
"AttributeName2":{"S":"AttributeValue2"},
"AttributeName3":{"S":"AttributeValue3"}
},{
"AttributeName1":{"S":"AttributeValue4"},
"AttributeName2":{"S":"AttributeValue5"},
"AttributeName3":{"S":"AttributeValue6"},
"AttributeName5":{"B":"dmFsdWU="}
}],
"LastEvaluatedKey":
{"HashKeyElement":{"S":"AttributeName1"},
"RangeKeyElement":{"N":"AttributeName2"}},
"ConsumedCapacityUnits":1,
"ScannedCount":2}

```



}

名稱	描述
Items	<p>符合操作參數的屬性的容器。</p> <p>類型：屬性名稱映射，以及其資料類型和值。</p>
Count	<p>回應中的項目數。如需詳細資訊，請參閱 <a href="#">計算結果中的項目</a>。</p> <p>類型：數字</p>
ScannedCount	<p>套用任何篩選條件之前的已完成掃描中項目數。ScannedCount 值很高，但 Count 結果很少或沒有，表示掃描操作不足。如需詳細資訊，請參閱 <a href="#">計算結果中的項目</a>。</p> <p>類型：數字</p>
LastEvaluatedKey	<p>停止掃描操作之項目的主索引鍵。在後續掃描操作中提供此值，以便從該時間點繼續操作。</p> <p>當整個掃描結果集完成時 (意即操作已處理「最後一頁」)，LastEvaluatedKey 為 null。</p>
ConsumedCapacityUnits	<p>操作所使用的讀取容量單位數目。此值顯示套用至佈建輸送量的數字。如需詳細資訊，請參閱 <a href="#">佈建容量模式</a>。</p> <p>類型：數字</p>

## 特殊錯誤

錯誤	描述
ResourceNotFoundException	找不到指定的資料表。

## 範例

如需使用 AWS SDK 的範例，請參閱[在 DynamoDB 中使用掃描](#)。

### 請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable","ScanFilter":{}}
```

### 回應範例

```
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}
```

## 請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0
content-length: 125

{"TableName":"comp5",
 "ScanFilter":
 {"time":
 {"AttributeValueList":[{"N":"400"}],
 "ComparisonOperator":"GT"}
 }
}
```

## 回應範例

```
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
}
```

## 請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
```

```
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Limit":2,
 "ScanFilter":
 {"time":
 {"AttributeValueList":[{"N":"400"}],
 "ComparisonOperator":"GT"}
 },
 "ExclusiveStartKey":
 {"HashKeyElement":{"S":"Fredy"},"RangeKeyElement":{"N":"2000"}}
 }
```

## 回應範例

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJJP20VALJ60TRVV4KQNS05AEMVJF66Q9ASUAAJG
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
 }
```

## 相關動作

- [Query](#)
- [BatchGetItem](#)

# UpdateItem

## ⚠ Important

##### API ## 2011-12-05#

如需目前低階 API 的文件，請參閱[Amazon DynamoDB API 參考](#)。

## 描述

編輯現有項目的屬性。您可以執行條件式更新 (在新的屬性名稱值組不存在時插入新的，或者在現有名稱值組具有某些預期的屬性值時將其取代)。

## 📘 Note

您無法使用更新主索引鍵屬性 UpdateItem。請改為刪除該項目，然後用 PutItem 來建立具有新屬性的新項目。

該 UpdateItem 操作包括一個 Action 參數，用於定義如何執行更新。您可以放入、刪除或新增屬性值。

屬性值不得為 Null；字串和二進位類型屬性的長度必須大於零；集合類型屬性不可為空白。具有空值的請求會遭到拒絕，並出現 ValidationException。

如果現有項目具有指定的主索引鍵：

- PUT：新增指定的屬性。如果屬性存在，則會以新值取代。
- DELETE：如果未指定任何值，則會移除屬性及其值。如果有指定一組值，則所指定集合中的值將從舊集合中移除。因此，如果屬性值包含 [a、b、c]，而刪除動作包含 [a、c]，則最後屬性值為 [b]。指定值類型必須符合現有值類型。指定空集合無效。
- ADD：僅對數字使用新增動作，或者在目標屬性為集合 (包括字串集合) 時使用。如果目標屬性為單一字串值或純量二進位值，則 ADD 無法運作。指定的值會新增至數值 (現有的數值進行遞增或遞減)，或者新增為字串集合中的額外值。如果有指定一組值，則會將這些值新增至現有的集合中。例如，如果原始集合為 [1、2]，而提供的值為 [3]，則在新增操作之後，集合為 [1、2、3]，而非 [4、5]。如果已為集合屬性指定 Add 動作，且指定的屬性類型與現有集合類型不符，則會發生錯誤。

如果對不存在的屬性使用 ADD，則會將屬性及其值新增至項目。

如果沒有項目符合所指定的主索引鍵：

- PUT：建立具有指定主索引鍵的新項目。然後新增指定屬性。
- DELETE：不進行任何動作。
- ADD：建立具有提供的主索引鍵和屬性值數字 (或一組數字) 的項目。不適用於字串或二進位類型。

#### Note

如果對更新前不存在的項目使用 ADD 增減數值，則 DynamoDB 會以 0 為初始值。此外，如果對更新前不存在的屬性 (但項目存在) 使用 ADD 增減數值來更新項目，則 DynamoDB 會以 0 為初始值。例如，您使用 ADD 將 +3 新增至更新前不存在的屬性。DynamoDB 使用 0 為初始值，而且更新後的值為 3。

如需使用此操作的詳細資訊，請參閱 [使用項目和屬性](#)。

## 請求

### 語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "Key":
 {"HashKeyElement":{"S":"AttributeValue1"},
 "RangeKeyElement":{"N":"AttributeValue2"}},
 "AttributeUpdates":{"AttributeName3":{"Value":
{"S":"AttributeValue3_New"},"Action":"PUT"}},
 "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3_Current"}}},
 "ReturnValues":"ReturnValuesConstant"
}
```

名稱	描述	必要
TableName	<p>包含要更新項目的資料表的名稱。</p> <p>類型：字串</p>	是
Key	<p>定義項目的主索引鍵。如需主索引鍵的詳細資訊，請參閱 <a href="#">主索引鍵</a>。</p> <p>類型：HashKeyElement 對其值的映射和 RangeKeyElement 對其值的映射。</p>	是
AttributeUpdates	<p>新值的屬性名稱映射以及更新動作。屬性名稱指定要修改的屬性，且不能包含任何主索引鍵屬性。</p> <p>類型：屬性名稱映射、值，以及屬性更新動作。</p>	
AttributeUpdates :Action	<p>指定如何執行更新。可能值：PUT (預設)、ADD 或 DELETE。語義在 UpdateItem 說明中進行了解釋。</p> <p>類型：字串</p> <p>預設：PUT</p>	否
Expected	<p>指定條件式更新的屬性。Expected 參數可讓您提供屬性名稱，以及 DynamoDB 是否應該檢查屬性值是否已存在；或者屬性值是否存在且在變更之前有特定值。</p>	否

名稱	描述	必要
	類型：屬性名稱映射。	
Expected:Attribute Name	條件式放置的屬性的名稱。 類型：字串	否



名稱	描述	必要
Expected:Attribute Name: ExpectedAttributeValue	<p>使用此參數來指定屬性名稱值的值是否已經存在。</p> <p>如果該項目的 Color (顏色) 屬性尚未存在，則下列 JSON 符號會更新項目：</p> <pre>"Expected" :   {"Color":{"Exists":false}}</pre> <p>下列 JSON 符號會在更新項目之前檢查名為 Color (顏色) 的屬性是否具有現有值 Yellow (黃色)：</p> <pre>"Expected" :   {"Color":{"Exists":true}, {"Value": {"S":"Yellow"}}</pre> <p>根據預設，如果使用 Expected 參數並提供 Value，DynamoDB 會假設屬性存在，且有要取代的目前值。所以您不必指定 {"Exists":true}，因為這是暗含的。您可以將請求縮短為：</p> <pre>"Expected" :   {"Color":{"Value": {"S":"Yellow"}}</pre>	否

名稱	描述	必要
	<p> <b>Note</b></p> <p>如果指定 {"Exists": true} 而無要檢查的屬性值，則 DynamoDB 會傳回錯誤。</p>	
ReturnValues	<p>如果想在以 UpdateItem 請求更新屬性名稱值組之前取得屬性名稱值組，則請使用此參數。可能的參數值為 NONE (預設) 或者 ALL_OLD、UPDATED_OLD、ALL_NEW 或 UPDATED_NEW。如果已指定 ALL_OLD，且 UpdateItem 覆寫屬性名稱值組，則會傳回舊項目的內容。如果未提供此參數，或者參數為 NONE，則不會傳回任何內容。如果指定 ALL_NEW，則會傳回項目新版本的所有屬性。如果指定 UPDATED_NEW，則只會傳回已更新屬性的新版本。</p> <p>類型：字串</p>	否

## 回應

### 語法

下列語法範例假設請求指定 ALL\_OLD 的 ReturnValues 參數，反之則回應只有 ConsumedCapacityUnits 元素。

```

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
}

```

名稱	描述
Attributes	<p>屬性名稱值組映射，但僅限於 ReturnValues 參數在請求中被指定為 NONE 以外選項的情況。</p> <p>類型：屬性名稱值組映射。</p>
ConsumedCapacityUnits	<p>操作所使用的寫入容量單位數目。此值顯示套用至佈建輸送量的數字。如需詳細資訊，請參閱 <a href="#">佈建容量模式</a>。</p> <p>類型：數字</p>

## 特殊錯誤

錯誤	描述
ConditionalCheckFailedException	條件式檢查失敗。屬性 ("+ name +") 值為 ("+ value +")，但預期為 ("+ expValue +")
ResourceNotFoundExceptions	找不到指定的項目或屬性。

## 範例

如需使用 AWS SDK 的範例，請參閱[使用項目和屬性](#)。

### 請求範例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Key":
 {"HashKeyElement":{"S":"Julie"},"RangeKeyElement":{"N":"1307654350"}},
 "AttributeUpdates":
 {"status":{"Value":{"S":"online"},
 "Action":"PUT"}},
 "Expected":{"status":{"Value":{"S":"offline"}}},
 "ReturnValues":"ALL_NEW"
}
```

### 回應範例

```
HTTP/1.1 200 OK
x-amzn-RequestId: 5IMH07F01Q9P7Q6QMKMMI3R3QRVV4KQNS05AEMVJF66Q9ASUAAJG
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
}
```

## 相關動作

- [PutItem](#)
- [DeleteItem](#)

# UpdateTable

## ⚠ Important

**##### API ## 2011-12-05#**

如需目前低階 API 的文件，請參閱[Amazon DynamoDB API 參考](#)。

## 描述

更新特定資料表的佈建輸送量。設定資料表的輸送量可協助您管理效能，此為 DynamoDB 佈建輸送量功能的一部分。如需詳細資訊，請參閱 [佈建容量模式](#)。

佈建輸送量數值可以根據 [Amazon DynamoDB 中的服務、帳戶和資料表配額](#) 中所列最大值和最小值來升級或降級。

資料表必須處於 ACTIVE 狀態，這項操作才會成功。UpdateTable 是非同步作業；執行作業時，資料表處於 UPDATING 狀態。資料表處於 UPDATING 狀態時，仍具有呼叫之前的佈建輸送量。新的佈建輸送量設定只有在表格返回 UpdateTable 作業後的 ACTIVE 狀態時才有效。

## 請求

### 語法

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName": "Table1",
 "ProvisionedThroughput": {"ReadCapacityUnits": 5, "WriteCapacityUnits": 15}
}
```

名稱	描述	必要
TableName	要更新的資料表名稱。  類型：字串	是

名稱	描述	必要
ProvisionedThroughput	指定資料表的新輸送量，包括 ReadCapacityUnits 與 WriteCapacityUnits 的值。請參閱 <a href="#">佈建容量模式</a> 。  類型：陣列	是
ProvisionedThroughput :ReadCapacityUnits	設定 DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低一致性 ReadCapacityUnits 數目。  最終一致讀取操作比一致性讀取負擔更輕，因此每秒 50 次一致性 ReadCapacityUnits 的設定可提供每秒 100 次最終一致 ReadCapacityUnits。  類型：數字	是
ProvisionedThroughput :WriteCapacityUnits	設定 DynamoDB 與其他操作平衡負載之前，所指定資料表每秒所需的最低 WriteCapacityUnits 數目。  類型：數字	是

## 回應

### 語法

```
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"}}
```

名稱	描述
CreationDateTime	<p>建立資料表時的日期。</p> <p>類型：數字</p>
KeySchema	<p>資料表的主索引鍵 (簡單或複合) 結構。需要 HashKeyElement 的名稱值組，且可選用 RangeKeyElement 的名稱值組 (僅適用於複合主索引鍵)。雜湊索引鍵大小上限為 2048 個位元組。範圍索引鍵大小上限為 1024 個位元組。此兩項限制皆為分別執行 (意即可出現雜湊 + 範圍 2048 + 1024 位元的合併索引鍵)。如需主索引鍵的詳細資訊，請參閱 <a href="#">主索引鍵</a>。</p> <p>類型：複合主索引鍵的 HashKeyElement 或 HashKeyElement 和 RangeKeyElement 映射。</p>
ProvisionedThroughput	<p>所指定資料表的目前輸送量設定，包括 LastIncreaseDateTime (如適用)、LastDecreaseDateTime (如適用) 的值。</p>

名稱	描述
	類型：陣列
TableName	已更新的資料表名稱。 類型：字串
TableStatus	資料表目前的狀態 (CREATING、ACTIVE、DELETING 或 UPDATING)，目前應該為 UPDATING。 使用 <a href="#">DescribeTables</a> 操作來查看資料表狀態。 類型：字串

## 特殊錯誤

錯誤	描述
ResourceNotFoundException	找不到指定的資料表。
ResourceInUseException	資料表不處於 ACTIVE 狀態。

## 範例

### 請求範例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB ## API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"comp1",
 "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```



## 回應範例

```
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"}
}
```

## 相關動作

- [CreateTable](#)
- [DescribeTables](#)
- [DeleteTable](#)

## AWS 適用於 Java 1.x 的開發套件範例

本節包含使用適用於 Java 1.x 的開發套件的 DAX 應用程式範例程式碼。

### 主題

- [搭配適用於 Java 1.x 的 AWS 開發套件使用 DAX](#)
- [修改適用於 Java 1.x 的開發套件的現有應用程式來使用 DAX](#)
- [使用適用於 Java 1.x 的開發套件查詢全域次要索引](#)

## 搭配適用於 Java 1.x 的 AWS 開發套件使用 DAX

請按照此程序操作，在 Amazon EC2 執行個體上執行 Amazon DynamoDB Accelerator (DAX) 的 Java 範例。

### Note

這些說明是針對使用適用於 Java 1.x 的 AWS 開發套件的應用程式。如需使用適用於 Java 2.x 的 AWS 開發套件的應用程式，請參閱 [Java 與 DAX](#)。

### 執行 DAX 的 Java 範例

1. 安裝 Java 開發套件 (JDK)。

```
sudo yum install -y java-devel
```

2. 下載 AWS SDK for Java (.zip 檔案)，並將其解壓縮。

```
wget http://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip
unzip aws-java-sdk.zip
```

3. 下載 DAX Java 用戶端 (.jar 檔) 最新版本。

```
wget http://dax-sdk.s3-website-us-west-2.amazonaws.com/java/DaxJavaClient-latest.jar
```

### Note

DAX SDK for Java 用戶端可在 Apache Maven 上取得。如需詳細資訊，請參閱 [使用用戶端做為 Apache Maven 依存項目](#)。

4. 設定您的 CLASSPATH 變數。在此範例中，將 *sdkVersion* 取代為 AWS SDK for Java 的實際版本號碼 (例如 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. 下載範例程式來源碼 (.zip 檔案)。

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

下載完成後，解壓縮來源檔案。

```
unzip TryDax.zip
```

6. 前往 Java 程式碼目錄並編譯程式碼，如下所示。

```
cd TryDax/java/
javac TryDax*.java
```

7. 執行程式。

```
java TryDax
```

您應該會看到類似下列的輸出。

```
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.
```

記下計時資訊：GetItem、Query 和 Scan 測試所需要的毫秒數。

8. 在先前的步驟中，您已針對 DynamoDB 端點執行程式。現在，請再次執行程式，但這一次 GetItem、Query 和 Scan 操作會由您的 DAX 叢集處理。

若要判斷您 DAX 叢集的端點，請選擇下列其中一個項目：

- 使用 DynamoDB 主控台：選擇您的 DAX 叢集。叢集端點會在主控台上顯示，如以下範例。

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- 使用 AWS CLI：輸入下列命令。

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

如下列範例所示，叢集端點會在輸出上顯示。

```
{
```

```
"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"
}
```

現在重新執行程式，但這一次，請將叢集端點做為命令列參數指定。

```
java TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

查看輸出的剩餘部分，並記下計時資訊。使用 DAX 的 `GetItem`、`Query` 和 `Scan` 已耗用時間應遠低於使用 DynamoDB 的已耗用時間。

如需此程式的詳細資訊，請參閱下列各節：

- [TryDax. 爪哇](#)
- [TryDaxHelper. 爪哇](#)
- [TryDaxTests. 爪哇](#)

## 使用用戶端做為 Apache Maven 依存項目

遵循這些步驟，在您的應用程式中將 DAX SDK for Java 用戶端做為依存項目使用。

使用用戶端作為 Maven 依存項目

1. 下載並安裝 Apache Maven。如需詳細資訊，請參閱[下載 Apache Maven](#)和[安裝 Apache Maven](#)。
2. 將用戶端 Maven 依存項目新增至您應用程式的專案物件模型 (POM) 檔案。在此範例中，將 `x.x.x.x` 取代為用戶端的實際版本號碼 (例如 `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. 爪哇

TryDax.java 檔案包含 main 方法。若您不使用任何命令列參數執行程式，它會建立 Amazon DynamoDB 用戶端，並使用該用戶端進行所有 API 操作。若您在命令列指定 DynamoDB Accelerator (DAX) 叢集端點，程式也會建立 DAX 用戶端，並用它來進行 GetItem、Query 和 Scan 操作。

您可以使用幾種方法修改程式：

- 使用 DAX 用戶端，而非 DynamoDB 用戶端。如需詳細資訊，請參閱 [Java 與 DAX](#)。
- 為測試資料表選擇不同的名稱。
- 變更 helper.writeData 參數，修改寫入的項目數目。第二個參數是分割區索引鍵的數字，第三個參數則為排序索引鍵的數字。根據預設，程式會使用 1-10 作為分割區索引鍵的值，使用 1-10 作為排序索引鍵的值，總共會將 100 個項目寫入資料表。如需詳細資訊，請參閱 [TryDaxHelper. 爪哇](#)。
- 修改 GetItem、Query 和 Scan 測試的數字，並修改其參數。
- 將包含 helper.createTable 和 helper.deleteTable 的行變更為註解 (若您不希望每次執行程式時都建立和刪除資料表的話)。

### Note

若要執行此程式，您可以設定 Maven 將適用於 Java 的 DAX 開發套件的用戶端與 AWS SDK for Java 用作依存項目。如需詳細資訊，請參閱 [使用用戶端做為 Apache Maven 依存項目](#)。或者，您必須在 classpath 中同時下載和包含 DAX Java 用戶端及 AWS SDK for Java。請參閱 [Java 與 DAX](#) 以取得設定您 CLASSPATH 變數的範例。

```
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. 爪哇

TryDaxHelper.java 檔案包含公用程式方法。

getDynamoDBClient 和 getDaxClient 方法提供 Amazon DynamoDB 和 DynamoDB Accelerator (DAX) 用戶端。針對控制平面操作 (CreateTable、DeleteTable) 及寫入操作，程式會使用 DynamoDB 用戶端。若您指定 DAX 叢集端點，主程式會建立 DAX 用戶端來執行讀取操作 (GetItem、Query、Scan)。

其他 TryDaxHelper 方法 (createTable、writeData、deleteTable) 用於設定及卸除 DynamoDB 資料表和其資料。

您可以使用幾種方法修改程式：

- 針對資料表使用不同的佈建輸送量設定。
- 修改每個寫入項目的大小 (請參閱 stringSize 方法中的 writeData 變數)。
- 修改 GetItem、Query 和 Scan 測試的數字及其參數。
- 將包含 helper.CreateTable 和 helper.DeleteTable 的行變更為註解 (若您不希望每次執行程式時都建立和刪除資料表的話)。

### Note

若要執行此程式，您可以設定 Maven 將適用於 Java 的 DAX 開發套件的用戶端與 AWS SDK for Java 用作依存項目。如需詳細資訊，請參閱 [使用用戶端做為 Apache Maven 依存項目](#)。或者，您必須在 classpath 中同時下載和包含 DAX Java 用戶端及 AWS SDK for Java。請參閱 [Java 與 DAX](#) 以取得設定您 CLASSPATH 變數的範例。

```
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. 爪哇

TryDaxTests.java 檔案包含對 Amazon DynamoDB 中測試資料表執行讀取操作的方法。這些方法皆未考慮其存取資料的方法 (使用 DynamoDB 用戶端或 DAX 用戶端)，因此不需修改應用程式邏輯。

您可以使用幾種方法修改程式：

- 修改 `queryTest` 方法，使其使用不同的 `KeyConditionExpression`。
- 將 `ScanFilter` 新增至 `scanTest` 方法，使其只傳回一部分的項目。

### Note

若要執行此程式，您可以設定 Maven 將適用於 Java 的 DAX 開發套件的用戶端與 AWS SDK for Java 用作依存項目。如需詳細資訊，請參閱 [使用用戶端做為 Apache Maven 依存項目](#)。或者，您必須在 `classpath` 中同時下載和包含 DAX Java 用戶端及 AWS SDK for Java。請參閱 [Java 與 DAX](#) 以取得設定您 `CLASSPATH` 變數的範例。

```
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));
}
}
```

## 修改適用於 Java 1.x 的開發套件的現有應用程式來使用 DAX

如果您已有使用 Amazon DynamoDB 的 Java 應用程式，則需要進行修改，使其可存取您的 DynamoDB Accelerator (DAX) 叢集。您不需要重新撰寫整個應用程式，因為 DAX Java 用戶端與 AWS SDK for Java 中所含的 DynamoDB 低階用戶端十分相似。

### Note

這些說明是指針對使用適用於 Java 1.x 的 AWS 開發套件的應用程式。如需使用適用於 Java 2.x 的 AWS 開發套件的應用程式，請參閱 [修改現有應用程式以使用 DAX](#)。

假設您有名為 Music 的 DynamoDB 資料表。資料表的分割區索引鍵為 Artist，而排序索引鍵為 SongTitle。以下程式會直接讀取 Music 資料表中的項目。

```
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());
 }
 }
}
```

若要修改程式，請將 DynamoDB 用戶端取代為 DAX 用戶端。

```
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)
 ** ...
 */

 }
}
```

## 使用 DynamoDB 文件 API

AWS SDK for Java 提供 DynamoDB 的文件介面。文件 API 會作為低階 DynamoDB 用戶端的包裝函式。如需詳細資訊，請參閱[文件介面](#)。

文件介面也可以與低階 DAX 用戶端搭配使用，如以下範例所示。

```
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());
}

}
```

## DAX 非同步用戶端

AmazonDaxClient 是同步的。針對長時間執行的 DAX API 操作 (例如大型資料表 Scan)，這可能會封鎖程式執行，直到操作完成。如果程式需要在 DAX API 操作進行時執行其他工作，您可以改用 ClusterDaxAsyncClient。

下列程式示範如何使用 ClusterDaxAsyncClient 和 Java Future 來實作非封鎖解決方案。

```
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();

}
}
```

## 使用適用於 Java 1.x 的開發套件查詢全域次要索引

您可以使用 Amazon DynamoDB Accelerator (DAX) 來查詢使用 DynamoDB [程式設計介面的全域次要索引](#)。

以下範例示範如何使用 DAX 查詢在[範例：使用 AWS SDK for Java 文件 API 的全域次要索引](#)中建立的 CreateDateIndex 全域次要索引。

DAXClient 類別將與 DynamoDB 程式設計介面互動所需的用戶端物件執行個體化。

```
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);
}
}
```

您可以用下列方式查詢全域次要索引：

- 在下列範例中定義的 `QueryIndexDax` 類別使用 `queryIndex` 方法。`getDaxDocClient` 方法在 `DaxClient` 類別傳回的用戶端物件，由 `QueryIndexDax` 當做參數。
- 如果您使用 [物件持久性介面](#)，請在下列範例中定義的 `queryIndexMapper` 類別使用 `QueryIndexDax` 方法。在 `DaxClient` 類別定義之 `getDaxMapperClient` 方法傳回的用戶端物件，由 `queryIndexMapper` 當做參數。

```
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());
}
}
```

以下的類別定義代表問題表，用於 `queryIndexMapper` 方法。

```
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;
}
}
```

# DynamoDB 的文件歷史記錄

下表說明 2018 年 7 月 3 日後每個《DynamoDB 開發人員指南》版本的重要變更。如需有關此文件更新的通知，您可以訂閱 RSS 摘要 (位於本頁面左上角)。

變更	描述	日期
<a href="#">重新架構並整合 DynamoDB 監控和記錄文件</a>	DynamoDB 中監視和記錄的新結構包括三個簡潔章節，說明指標、記錄操作和參與者深入解析。	2024年5月3日
<a href="#">重新架構並整合 DynamoDB 容量模式文件</a>	DynamoDB 指南現在包含一個新章節，其中包含有關 DynamoDB 容量模式的所有資訊-隨需和佈建。透過此更新，「變更讀取/寫入容量模式時的考量」主題已移至「最佳作法」一章中。此主題現已重新命名為切換容量模式時的考量事項，並包含切換容量模式時最佳作法的詳細資訊。此外，該指南現在還提供了一個新章節，其中包括有關 DynamoDB 讀取和寫入的所有資訊，以及讀取和寫入操作的容量單位消耗。如需詳細資訊，請參閱 <a href="#">DynamoDB 輸送量容量</a> 、 <a href="#">切換容量模式時的考量</a> 以及 <a href="#">DynamoDB 讀取和寫入</a> 。	2024年5月1日
<a href="#">隨需請求的最大數量</a>	您現在可以指定個別資料表、索引或兩者都可以執行的隨選要求數目上限。指定最大隨選輸送量將有助於保持表格層級使用量和成本限制，並防止不	2024年5月1日

慎使用的資源激增。如需詳細資訊，請參閱[隨選資料表的最大輸送量](#)。

### [NoSQL 工作台操作生成器的改進](#)

NoSQL 工作台現在包括對暗模式的本機支持。改進了操作生成器中的表格和項目操作。項目結果和操作生成器請求信息以 JSON 格式提供。如需詳細資訊，請參閱 [NoSQL 工作台作業建構器](#)。

2024年4月24日

### [適用於亞馬遜資源的以資源為基礎的政策](#)

DynamoDB 現在可針對資料表、索引和串流支援以資源為基礎的政策。以資源為基礎的策略可讓您定義存取權限，方法是指定每個資源的存取權限，以及允許這些資源對每個資源執行的動作。如需詳細資訊，請參閱針對 [DynamoDB 使用以資源為基礎的政策](#)。

2024年3月20日

### [DynamoDB 策更新](#)

已新增AmazonDynamoDBReadOnlyAccess 受管理原則的新權限 dynamodb:GetResourcePolicy 。此權限可讓您存取附加至 DynamoDB 資源的讀取以資源為基礎的原則。如需詳細資訊，請參閱[AWS 受管原則：AmazonDynamoDB ReadOnlyAccess](#)。

2024年3月20日



## [AWS PrivateLink 適用於 Amazon DynamoDB](#)

Amazon DynamoDB 現在支援。AWS PrivateLink 您可以使用介面 VPC 端點和私有 AWS PrivateLink IP 位址，簡化虛擬私有雲 (VPC)、DynamoDB 和內部部署資料中心之間的私有網路連線。如需詳細資訊，請參閱[AWS PrivateLink 如需 DynamoDB 的資訊](#)。

2024年3月19日

## [編程與 JavaScript 指南](#)

Amazon DynamoDB 供的程式設計指南。AWS SDK for JavaScript 瞭解抽象層 AWS SDK for JavaScript、設定連線、處理錯誤、定義重試原則、管理保持活動狀態等。如需詳細資訊，請參閱[使用 JavaScript](#)。

2024年3月6日

## [編程與 AWS SDK for Java 2.x 指南](#)

建立新的程式設計指南，深入了解高階、低階和文件介面、HTTP 用戶端及其組態、錯誤處理，並解決您在使用 Java 2.x 版 SDK 時應考量的最常見組態設定。如需詳細資訊，請參閱[使用](#). AWS SDK for Java 2.x

2024年3月5日

## [使用 NoSQL 工作台複製資料表](#)

允許開發人員使用 NoSQL 工作台在開發環境和區域之間複製或複製表格 (DynamoDB 本機和 DynamoDB 網頁版)。如需詳細資訊，請參閱 < [使用 NoSQL 工作台複製資料表](#) >。

2024年2月26日

[使用 Python 指南進行編程](#)

創建了一個新的指南，深入了解高級和低級庫，並解決了在使用 Python SDK 時應考慮的最常見配置設置。如需詳細資訊，請參閱[使用 Python 程式設計](#)。

2024年1月5日

[存活時間 \( TTL \) 主題重寫](#)

完全重寫了指南的 TTL 部分。新指南會在過程中提供程式 ready-to-use 碼片段，協助您開始使用 TTL。目前提供的程式碼片段是在 Python 和 JavaScript 中。如需詳細資訊，請參閱[TTL](#)。

2023 年 12 月 20 日

[瞭解 AWS 帳單和用量報告的最佳做法](#)

新增一個新區段，說明 DynamoDB 中各種用量類型和這些用量類型的費用。如需詳細資訊，請參閱[帳單和用量報告](#)。

2023 年 12 月 15 日

[Amazon DynamoDB 零 ETL 與 Amazon 服務整合 OpenSearch](#)

Amazon DynamoDB 現在支援與 Amazon OpenSearch 服務的零 ETL 整合，可讓您透過自動複寫和轉換 DynamoDB 資料執行搜尋，而無需自訂程式碼或基礎設施。如需詳細資訊，請參閱[DynamoDB 零 ETL 與 Amazon 服務整合](#)。  
OpenSearch

2023 年 11 月 28 日

[從關聯式資料庫移轉至 DynamoDB](#)

建立[移轉指南](#)，以協助處理事務的使用者如何從關聯式資料庫移轉至 DynamoDB。

2023 年 11 月 27 日

<a href="#">使用 NoSQL Workbench 產生範例資料</a>	適用於 Amazon DynamoDB 的 NoSQL Workbench 現在支援直接從 <a href="#">範例資料模型範本</a> 建立資料模型，以協助您為工作負載設計資料結構描述。在 DynamoDB 上建置應用程式時，您可以使用此功能熟悉 NoSQL 資料建模最佳實務。	2023 年 9 月 28 日
<a href="#">增量匯出至 S3</a>	您現在可以匯出以小幅增量插入、更新或刪除的資料。透過 <a href="#">增量匯出</a> ，您可以在 AWS 管理主控台、API 呼叫或 AWS 命令列介面中按幾下滑鼠，將變更的資料匯出範圍從幾 MB 到 TB。	2023 年 9 月 26 日
<a href="#">適用於 DynamoDB 的資料建模</a>	您現在可以使用 DynamoDB 範例進一步了解 <a href="#">資料模型</a> ，這些範例著重於特定使用案例、其存取模式，以及實現這些存取模式的 step-by-step 指引。	2023 年 7 月 14 日
<a href="#">故障診斷一節</a>	您現在可以找到 DynamoDB 資料表中可能發生的延遲和限流問題的 <a href="#">疑難排解內容</a> 。	2023 年 3 月 13 日
<a href="#">Amazon DynamoDB 的刪除保護</a>	現在，所有 AWS 區域的 Amazon DynamoDB 資料表都可使用刪除保護功能。DynamoDB 現在可讓您在執行常態資料表管理操作時，保護資料表免遭意外刪除。	2023 年 3 月 8 日

<a href="#">AWS CloudFormation 全域表格中的 KDS 支援</a>	適用於 DynamoDB 的 Amazon Kinesis Data Streams 現在支援 AWS CloudFormation DynamoDB 全域表，這表示您可以使用範本在 DynamoDB 全域表上啟用串流至 Amazon Kinesis 資料串流。CloudFormation	2023 年 2 月 15 日
<a href="#">DynamoDB 本機版支援每個交易 100 個動作</a>	您現在可以在 DynamoDB 本機版的單一交易中執行多達 100 個動作。	2023 年 2 月 9 日
<a href="#">使用 DynamoDB Well-Architected Lens 來最佳化您的 DynamoDB 工作負載</a>	您現在可以使用 <a href="#">DynamoDB Well-Architected Lens</a> ，其收集了用於設計架構良好 DynamoDB 工作負載的設計原則和指引。	2023 年 2 月 3 日
<a href="#">PartiQL GovCloud 的可用性</a>	<a href="#">PartiQL 域 — 現在 AWS GovCloud (美國東部) 和 (美國西部) 支援 Amazon DynamoDB 的 SQL 相容查詢語言</a> 。AWS GovCloud	2022 年 12 月 21 日
<a href="#">NoSQL Workbench 和 DynamoDB 本機版的單一安裝套件</a>	<a href="#">DynamoDB 專用 NoSQL Workbench</a> 現在包含引導式 <a href="#">DynamoDB 本機版</a> 安裝程序，可簡化 DynamoDB 本機版開發環境的設定。	2022 年 12 月 6 日
<a href="#">從 S3 大量匯入</a>	Amazon DynamoDB 現在 <a href="#">支援從 Amazon S3 大量匯入資料</a> ，讓您可以更輕鬆地將資料遷移並載入新的 DynamoDB 資料表。	2022 年 8 月 18 日

<a href="#">與 Service Quotas 加強整合</a>	<a href="#">Service Quotas</a> 現在可讓您主動管理您的帳戶和資料表配額。您可以檢視目前的值、設定配額使用率超過可設定閾值時發出警示等。	2022 年 6 月 15 日
<a href="#">NoSQL Workbench 新增資料表和 GSI 支援</a>	您現在可以使用 NoSQL 工作台進行資料表和全域次要索引 (GSI) <a href="#">控制平面作業</a> ，例如 CreateTable UpdateTable、和。 DeleteTable	2022 年 6 月 2 日
<a href="#">標準的不常存取資料表類別現已在中國推出</a>	Amazon DynamoDB 標準–不常存取資料表類別在中國地區推出。針對儲存不常存取的資料的資料表使用此全新資料表類別，藉此降低 <a href="#">高達 60% 的 DynamoDB 成本</a> 。	2022 年 4 月 18 日
<a href="#">增加預設服務配額和資料表管理操作</a>	<a href="#">DynamoDB 增加了每個帳戶和區域的資料表數目預設配額</a> ，從 256 個增加到 2,500 個資料表，並將並行資料表管理操作的數量從 50 個增加到 500 個。	2022 年 3 月 9 日
<a href="#">適用於 DynamoDB 的 PartiQL 的項目限制 (選用)</a>	DynamoDB 可在每個請求中，透過選用參數，針對 DynamoDB 操作 <a href="#">限制 PartiQL 中處理的項目數</a> 。	2022 年 3 月 8 日

<a href="#">AWS Backup 在中國 (北京和寧夏) 地區提供整合服務</a>	在中國 (北京與寧夏) 區域， <a href="#">AWS Backup</a> 現已與 DynamoDB 整合。透過增強的備份功能 (例如跨帳戶和跨區域備份) AWS Backup，您可以更輕鬆地滿足合規性和業務持續性需求。	2022 年 1 月 26 日
<a href="#">透過 PartiQL API 呼叫的輸送容量資訊</a>	DynamoDB 可以傳回 <a href="#">PartiQL API</a> 呼叫所耗用的輸送容量資訊，以協助您最佳化查詢和輸送量成本。	2022 年 1 月 18 日
<a href="#">AWS Backup 整合</a>	DynamoDB 現在可透過 <a href="#">AWS Backup</a> 中的強化備份功能 (例如跨帳戶和跨區域備份)，協助您更容易符合合規和商業持續性要求。	2021 年 11 月 24 日
<a href="#">CSV 格式的 NoSQL Workbench 匯入/匯出資料集</a>	<a href="#">適用於 Amazon DynamoDB 的 NoSQL Workbench</a> 現在可以匯入和自動填入範例資料，以協助建置和視覺化資料模型。	2021 年 10 月 11 日
<a href="#">篩選和擷取 Amazon DynamoDB 串流資料平面活動 AWS CloudTrail</a>	Amazon DynamoDB 現在為您提供了更精細的稽核日誌記錄控制，方法是讓您能夠在 <a href="#">AWS CloudTrail</a> 中篩選 <a href="#">Streams 資料平面 API 活動</a> 。	2021 年 9 月 22 日
<a href="#">已更新主控台</a>	現在 <a href="#">DynamoDB 主控台</a> 是您的預設主控台，可協助您更輕鬆地管理資料、簡化常見任務，並讓您更快地存取資源和功能。	2021 年 8 月 25 日

<a href="#">適用於 Java 2.x 的 DAX SDK 現已推出</a>	適用於 Java 2.x 的 <a href="#">DynamoDB 加速器 (DAX) 開發套件</a> 現已推出，並且與適用於 Java 2.x 的 AWS 開發套件相容。您可以從最新功能中受益，包括非封鎖輸入/輸出 (I/O)。	2021 年 7 月 29 日
<a href="#">NoSQL Workbench 功能更新，包括控制平面操作</a>	<a href="#">Amazon DynamoDB 專用 NoSQL Workbench</a> 現在可以協助您更輕鬆地執行頻繁的操作以修改和存取資料表資料。	2021 年 7 月 28 日
<a href="#">DynamoDB 全域資料表現已在亞太區域提供</a>	<a href="#">DynamoDB 全域資料表</a> 現已在亞太區域 (大阪) 提供。跨越您選擇的 22 個 AWS 區域自動複製 DynamoDB 資料表。	2021 年 7 月 28 日
<a href="#">DAX 現已在中國提供</a>	<a href="#">DynamoDB Accelerator (DAX)</a> 現已在由 Sinnet 營運的中國 (北京) 區域提供。	2021 年 7 月 28 日
<a href="#">DAX 傳輸中加密</a>	<a href="#">DynamoDB Accelerator (DAX)</a> 現在支援在應用程式和 DAX 叢集之間以及 DAX 叢集內的節點之間傳輸資料時進行加密。	2021 年 7 月 24 日
<a href="#">CloudFormation 和 CloudTrail 整合</a>	與 <a href="#">CloudFormation 資料平面記錄</a> 整合 AWS CloudFormation 並增強安全性。	2021 年 6 月 18 日
<a href="#">CloudFormation 現在支援全域表</a>	<a href="#">Amazon DynamoDB 全域表</a> 現在支援 <a href="#">AWS CloudFormation</a> ，這表示您可以建立全域表，並使用範本管理其設定。	2021 年 5 月 14 日

### [適用於 Java 2.x 的 Amazon DynamoDB 本地支援](#)

現在您可以使用[適用於 Java 2.x 的 AWS SDK](#) 搭配 Amazon DynamoDB 的可下載版本 [DynamoDB Local](#)。藉助 DynamoDB Local，您可以使用在本地開發環境中執行的 DynamoDB 版本來開發和測試應用程式，而不會產生任何額外成本。

2021 年 5 月 3 日

### [NoSQL 工作台現在支持 AWS CloudFormation](#)

[適用於 Amazon DynamoDB 的 NoSQL 工作台](#) 現在支援 [AWS CloudFormation](#)，因此您可以使用範本來管理和修改 DynamoDB 資料模型。CloudFormation 此外，您現在可以在 NoSQL Workbench 中配置資料表容量設定。

2021 年 4 月 22 日

### [與 AWS Amplify 現在的功能整合](#)

[AWS Amplify](#) 現在可以在單個部署中協調多個 DynamoDB 全域次要索引更新。

2021 年 4 月 20 日

### [AWS CloudTrail 記錄 Amazon DynamoDB 資料平面 API](#)

您現在可以使用 [AWS CloudTrail 記錄 Amazon DynamoDB Streams 資料平面 API 活動](#)，並監視和調查 DynamoDB 資料表中的項目層級變更。

2021 年 4 月 20 日



[亞馬遜運 Amazon DynamoDB 資料串流現在支援 AWS CloudFormation](#)

[Amazon Kinesis Data Streams](#) 現在支援 AWS CloudFormation，這表示您可以透過範本啟用串流至 DynamoDB 表上的 Amazon Kinesis 資料串流。CloudFormation 藉由將 DynamoDB 資料變更串流至 Kinesis 資料串流，您可以使用 Amazon Kinesis 服務建置進階串流應用程式。

2021 年 4 月 12 日

[Amazon Keyspaces 現在提供符合 FIPS 140-2 標準的端點](#)

[Amazon Keyspaces \(適用於 Apache Cassandra\)](#) 現已提供符合聯邦資訊處理標準 (FIPS) 140-2 的端點，可幫助您更輕鬆地執行高度監管的工作負載。FIPS 140-2 是美國和加拿大政府標準，指定密碼模組的安全要求以保護敏感資訊。

2021 年 4 月 8 日

[適用於 DAX 的 Amazon EC2 T3 執行個體](#)

DAX 目前支援 [Amazon EC2 T3 執行個體類型](#)，此類執行個體可提供基本水準的 CPU 效能，並可在需要時大幅突破基本水準。

2021 年 2 月 15 日

[Amazon DynamoDB 專用 NoSQL Workbench 支援 PartiQL](#)

您現在可以使用 [DynamoDB 專用 NoSQL Workbench](#) 為 DynamoDB 建置 [PartiQL](#) 陳述式。

2020 年 12 月 4 日

[DynamoDB 專用 PartiQL](#)

您現在可以將 [PartiQL 用於 DynamoDB](#) (SQL 相容的查詢語言)，與 DynamoDB 表互動，並使用、和適用於 PartiQL 的 DynamoDB API 來執行隨機操作查詢。AWS Management Console AWS Command Line Interface

2020 年 11 月 23 日

[Amazon DynamoDB 專用 Amazon Kinesis Data Streams](#)

您現在可以將 [Amazon DynamoDB 專用 Amazon Kinesis Data Streams](#) 與 DynamoDB 資料表搭配使用，以便擷取項目層級的變更，並將變更複寫至 Kinesis 資料串流。

2020 年 11 月 23 日

[DynamoDB 資料表匯出](#)

您現在可以將 [DynamoDB 表匯出到 Amazon S3](#)，讓您能夠使用 Athena 和 Lake Formation 等服務對資料執行分析和複雜查詢。AWS Glue

2020 年 11 月 9 日

[空值支援](#)

DynamoDB 現在支援 DynamoDB 資料表中的非索引鍵字串和二進位屬性的空值。空值支援功能讓您能夠更靈活地將屬性運用於更廣泛的使用案例，無需在傳送至 DynamoDB 之前轉換這類屬性。清單、映射及設定資料類型也支援空白字串和二進位數值。

2020 年 5 月 18 日

[Amazon DynamoDB 專用  
NoSQL Workbench 支援 Linux](#)

Amazon DynamoDB 專用 NoSQL Workbench 現在也支援 [Linux Ubuntu、Fedora 及 Debian](#)。

2020 年 5 月 4 日

[CloudWatch 針對 DynamoDB  
的貢獻者深入解析 — GA](#)

[CloudWatchDynamoDB 的參與者深入解析](#) 正式推出。CloudWatchDynamoDB 的參與者深入解析是一種診斷工具，可提供 DynamoDB 表格流量趨勢的 at-a-glance 檢視，並協助您識別資料表最常存取的金鑰 (也稱為快速鍵)。

2020 年 4 月 2 日

[升級全域資料表](#)

現在，您只要在 DynamoDB 主控台中按幾下滑鼠，即可將全域資料表從 2017 年 11 月 29 日版本更新至 [全域資料表的最新版本 \(2019 年 11 月 21 日\)](#)。藉由升級全域表的版本，您可以將現有表格延伸到其他 AWS 區域，輕鬆提高 DynamoDB 表的可用性，而不需要重建資料表。

2020 年 3 月 16 日

[Amazon DynamoDB 專用  
NoSQL Workbench : GA 版](#)

[Amazon DynamoDB 專用 NoSQL Workbench](#) 現已正式推出。使用 NoSQL Workbench 設計、建立、查詢及管理 DynamoDB 資料表。

2020 年 3 月 2 日

[DAX 快取叢集指標](#)

DAX 支援新 [CloudWatch 指標](#)，讓您更了解 DAX 叢集的效能。

2020 年 2 月 6 日

## [CloudWatch DynamoDB 的貢獻者深入解析 — 預覽](#)

[CloudWatchDynamoDB 的參與者深入解析](#)是一種診斷工具，可提供 DynamoDB 表格流量趨勢的 at-a-glance 檢視，並協助您識別資料表最常存取的金鑰 (也稱為快速鍵)。

2019 年 11 月 26 日

## [支援不平衡工作負載的調適型容量](#)

Amazon DynamoDB 的自適應容量功能現在可藉由自動隔離經常存取的項目，來更妥善地處理不平衡的工作負載。如果應用程式會導致一或多個項目的流量過大，DynamoDB 則會重新平衡您的分割區，讓經常存取的項目不會駐留在同一分割區上。

2019 年 11 月 26 日

## [支援客戶受管金鑰](#)

DynamoDB 現在支援客戶受管金鑰，這代表您可以完全控制如何加密與管理 DynamoDB 資料的安全性。

2019 年 11 月 25 日

## [支援 DynamoDB 本機版 \(可下載版本\) 的 NoSQL Workbench](#)

NoSQL Workbench 現在支援連接至 [DynamoDB 本機版 \(可下載版本\)](#)，以設計、建立、查詢和管理 DynamoDB 資料表。

2019 年 11 月 8 日

## [NoSQL Workbench : 預覽版](#)

這是 DynamoDB 專用 NoSQL Workbench 的初始版本。使用 NoSQL Workbench 設計、建立、查詢及管理 DynamoDB 資料表。如需詳細資訊，請參閱 [Amazon DynamoDB 專用 NoSQL Workbench \(預覽版\)](#)。

2019 年 9 月 16 日

## [DAX 新增支援使用 Python 和 NET 的交易操作](#)

DAX 支援以 Go、Java、NET、Node.js 和 Python 編寫之應用程式的 TransactWriteItems 與 TransactGetItems API。如需詳細資訊，請參閱[使用 DAX 進行記憶體內部加速](#)。

2019 年 2 月 14 日

## [Amazon DynamoDB 本機版 \(可下載版本\) 更新](#)

DynamoDB 本機版 (可下載版本) 現在支援交易 API、隨需讀取/寫入容量、讀取和寫入操作容量報告，以及 20 個全域次要索引。如需詳細資訊，請參閱[可下載的 DynamoDB 和 DynamoDB Web 服務之間的差異](#)。

2019 年 2 月 4 日

## [Amazon DynamoDB 隨需功能](#)

DynamoDB 隨需功能是彈性的計費選項，可在沒有容量規劃的情況下，每秒處理數千筆請求。DynamoDB 隨需提供讀取和寫入請求的 pay-per-request 定價，因此您只需按使用量付費。如需詳細資訊，請參閱[DynamoDB 輸送量容量](#)。

2018 年 11 月 28 日

## [Amazon DynamoDB Transactions](#)

DynamoDB 交易可對資料表內部和跨表格的多個項目進行協調 all-or-nothing 變更，從而在 DynamoDB 中提供原子性、一致性、隔離性和持久性 (ACID)。如需詳細資訊，請參閱[Amazon DynamoDB Transactions](#)。

2018 年 11 月 27 日

## [Amazon DynamoDB 會將客戶的所有靜態資料加密](#)

每當資料儲存到持久性的媒體時，DynamoDB 的靜態加密功能，就會透過保護加密資料表中的資料 (包括其主索引鍵、本機及全域次要索引、串流、全域資料表、備份和 DAX 叢集)，來提供多一層的資料保護機制。如需詳細資訊，請參閱 [Amazon DynamoDB 靜態加密](#)。

2018 年 11 月 15 日

## [運用全新的 Docker 影像來更輕鬆地使用 Amazon DynamoDB 本機版](#)

現在，使用 DynamoDB 本機版 (也就是可下載的 DynamoDB 版本) 變得更加容易，可讓您使用全新的 DynamoDB 本機 Docker 影像，來開發和測試 DynamoDB 應用程式。如需詳細資訊，請參閱 [DynamoDB \(可下載版本\)](#) 和 [Docker](#)。

2018 年 8 月 22 日

## [DynamoDB Accelerator \(DAX\) 新增對靜態加密的支援](#)

DynamoDB Accelerator (DAX) 現在支援新 DAX 叢集的靜態加密功能，可協助您在注重安全的應用中，加快從 Amazon DynamoDB 資料表讀取資料的作業，這類應用必須配合嚴格的合規與法規要求。如需詳細資訊，請參閱 [DAX 靜態加密](#)。

2018 年 8 月 9 日

## [DynamoDB point-in-time 復原 \(PITR\) 新增對還原已刪除資料表的支援](#)

如果您刪除已啟用 point-in-time 復原的表格，系統備份會自動建立並保留 35 天 (無需額外費用)。如需詳細資訊，請參閱 [在您開始使用時間點復原之前](#)。

2018 年 8 月 7 日

[現在可以透過 RSS 獲得更新](#)

您現在可以訂閱 [RSS 訊息](#) (在此頁面的左上角)，以接收《Amazon DynamoDB 開發人員指南》的更新通知。

2018 年 7 月 3 日

## 舊版更新

下表說明 2018 年 7 月 3 日前《DynamoDB 開發人員指南》的重要變更。

變更	描述	變更日期
取得 DAX 支援	您現在能夠對以 Go 程式語言撰寫之應用程式中的 Amazon DynamoDB 資料表啟用微秒讀取效能，方法是使用新 DynamoDB Accelerator (DAX) SDK for Go。如需詳細資訊，請參閱 <a href="#">適用於 Go 的 DAX 開發套件</a> 。	2018 年 6 月 26 日
DynamoDB 發行 SLA	DynamoDB 已發行公有可用性 SLA。如需詳細資訊，請參閱 <a href="#">Amazon DynamoDB 服務水準協議</a> 。	2018 年 6 月 19 日
DynamoDB 持續備份和時間點復原 (PITR)	Point-in-time 復原有助於保護您的 Amazon DynamoDB 資料表免於意外寫入或刪除操作。有了時間點復原，就無需為建立、維護或排程隨需備份而煩惱。例如，假設測試指令碼意外寫入至生產 DynamoDB 資料表。透過 point-in-time 復原，您可以將該資料表還原到過去 35 天內的任何時間點。DynamoDB 維護您資料表的	2018 年 4 月 25 日

變更	描述	變更日期
	增量備份。如需詳細資訊，請參閱 <a href="#">適用於 DynamoDB 的 Point-in-time 復原</a> 。	
適用於 DynamoDB 的靜態加密	新 DynamoDB 資料表可用的 DynamoDB 靜態加密可透過使用在 AWS Key Management Service 中存放之 AWS 受管的加密金鑰，來協助確保 Amazon DynamoDB 資料表中應用程式資料的安全性。如需詳細資訊，請參閱 <a href="#">DynamoDB 靜態加密</a> 。	2018 年 2 月 8 日
DynamoDB 備份和還原	隨需備份可讓您建立 DynamoDB 資料表的完整備份以進行資料封存，協助您符合公司與政府的法規要求。您可以備份資料量從幾 MB 到數百 TB 不等的資料表，而且不會影響您生產應用程式的效能與可用性。如需詳細資訊，請參閱 <a href="#">使用 DynamoDB 的隨需備份與還原</a> 。	2017 年 11 月 29 日



變更	描述	變更日期
DynamoDB 全域資料表	Global Tables 建立於 DynamoDB 全域佈局的基礎之上，提供您全受管、多區域以及多個作用中的資料庫，為大幅度擴展的全域應用程式提供快速、本機、讀取與寫的效能。全域表會在您選擇的區域中自動複寫 Amazon DynamoDB 資料表。AWS 如需詳細資訊，請參閱 <a href="#">全域資料表：DynamoDB 的多區域複寫</a> 。	2017 年 11 月 29 日
DAX 的 Node.js 支援	Node.js 開發人員可以使用適用於 Node.js 的 DAX 用戶端，來利用 DynamoDB Accelerator (DAX)。如需詳細資訊，請參閱 <a href="#">使用 DynamoDB Accelerator (DAX) 的記憶體內加速</a> 。	2017 年 10 月 5 日
DynamoDB 的 VPC 端點	DynamoDB 端點可讓您 Amazon VPC 中的 Amazon EC2 執行個體存取 DynamoDB，而不需暴露於公有網際網路中。您的 VPC 與 DynamoDB 之間的網路流量都會在 Amazon 網路的範圍內。如需詳細資訊，請參閱 <a href="#">使用 Amazon VPC 端點來存取 DynamoDB</a> 。	2017 年 8 月 16 日

變更	描述	變更日期
DynamoDB Auto Scaling 功能	DynamoDB Auto Scaling 功能讓您不必手動定義或調整佈建輸送量設定。相反地，DynamoDB Auto Scaling 功能會根據實際的流量模式，動態調整讀寫容量。這可讓資料表或全域次要索引增加其佈建的讀取與寫入容量，以處理突然增加的流量，而不需調節。當工作負載降低時，DynamoDB Auto Scaling 功能就會降低佈建的容量。如需詳細資訊，請參閱 <a href="#">使用 DynamoDB Auto Scaling 功能自動管理輸送容量</a> 。	2017 年 6 月 14 日
DynamoDB Accelerator (DAX)	DynamoDB Accelerator (DAX) 是適用於 DynamoDB 的全受管、高可用性記憶體內快取，即使每秒數百萬個請求也能將時間從毫秒縮短到微秒，提供高達 10 倍的效能改進。如需詳細資訊，請參閱 <a href="#">使用 DynamoDB Accelerator (DAX) 的記憶體內加速</a> 。	2017 年 4 月 19 日
DynamoDB 現在支援使用存留時間 (TTL) 的項目自動過期	Amazon DynamoDB 存留時間 (TTL) 讓您可自動刪除資料表的過期項目，無額外成本。如需詳細資訊，請參閱 <a href="#">生存時間 (TTL)</a> 。	2017 年 2 月 27 日

變更	描述	變更日期
DynamoDB 現在支援成本分配標籤	您現在可以對您的 Amazon DynamoDB 資料表新增標籤，以改善用量分類及讓成本報告更精細。如需詳細資訊，請參閱 <a href="#">將標籤新增至資源</a> 。	2017 年 1 月 19 日
新 DynamoDB DescribeLimits API	DescribeLimits API 會傳回區域中您 AWS 帳戶目前佈建的容量限制，包括整個區域和您在該地區建立的任何 DynamoDB 表。它可讓您判斷目前的帳戶等級限制，以便與目前使用的佈建容量比較，讓您在達到限制前有充分的時間申請提高限制容量。如需詳細資訊 <a href="#">Amazon DynamoDB 中的服務、帳戶和資料表配額</a> ，請參閱 Amazon DynamoDB 參考 <a href="#">DescribeLimits</a> 中的和。	2016 年 3 月 1 日

變更	描述	變更日期
DynamoDB 主控台更新與新的主索引鍵屬性術語	<p>DynamoDB 管理主控台經過重新設計，更直觀、更易於操作。在這次更新中，我們也會推出新的主索引鍵屬性術語：</p> <ul style="list-style-type: none"><li>• Partition Key (分割區索引鍵)：也稱為雜湊屬性。</li><li>• Sort Key (排序索引鍵)：也稱為範圍屬性。</li></ul> <p>唯一變更的只有名稱，功能仍保持不變。</p> <p>當您建立資料表或次要索引時，可以選擇簡易主索引鍵 (僅分割區索引鍵) 或複合主索引鍵 (分割區索引鍵和排序索引鍵)。DynamoDB 文件已更新以反映這些變更。</p>	2015 年 11 月 12 日

變更	描述	變更日期
適用於 Titan 的 Amazon DynamoDB 儲存後端	<p>適用於 Titan 的 DynamoDB 儲存後端是在 Amazon DynamoDB 上層實作的 Titan 圖形資料庫儲存後端。使用適用於 Titan 的 DynamoDB 儲存後端時，您的資料受 DynamoDB 所保護，該服務在所有 Amazon 具高可用性的資料中心上執行。該外掛程式可供 Titan 0.4.4 版 (主要針對與現有應用程式的相容性) 和 Titan 0.5.4 版 (新的應用程式建議使用) 使用。如同適用於 Titan 的其他儲存後端，此外掛程式也支援 Tinkerpop 堆疊 (2.4 和 2.5 版)，包括 Blueprints API 和 Gremlin shell。如需詳細資訊，請參閱 <a href="#">適用於 Titan 的 Amazon DynamoDB 儲存後端</a>。</p>	2015 年 8 月 20 日

變更	描述	變更日期
DynamoDB Streams、跨區域複寫和使用強烈一致讀取的掃描	<p>DynamoDB Streams 可擷取任何 DynamoDB 資料表中依時間順序排序的項目層級修改，並將此資訊存放於日誌中長達 24 小時。應用程式可以存取此日誌，並近乎即時地檢視資料項目修改前後的顯示內容。如需詳細資訊，請參閱 <a href="#">DynamoDB Streams 的變更資料擷取</a> 及 <a href="#">《DynamoDB Streams API 參考》</a>。</p> <p>DynamoDB 跨區域複寫是一種用戶端解決方案，可以近乎即時地跨不同區 AWS 域維護相同的 DynamoDB 表複本。您可以使用跨區域複寫備份 DynamoDB 資料表，或在使用者分散於不同地理位置的情況下提供資料的低延遲存取。</p> <p>根據預設，DynamoDB Scan 操作會使用最終一致讀取。您可以將 <code>ConsistentRead</code> 參數設為 <code>true</code>，改用強烈一致讀取。如需詳細資訊，請參閱 <a href="#">《Amazon DynamoDB API 參考》</a> 中的 <a href="#">掃描的讀取一致性</a> 及 <a href="#">掃描</a>。</p>	2015 年 7 月 16 日

變更	描述	變更日期
AWS CloudTrail 對 Amazon DynamoDB 支援	現在已與 CloudTrail 主控台或 DynamoDB API 發出的 API 呼叫，並在記錄檔中追蹤這些呼叫。如需詳細資訊，請參閱 <a href="#">使用 AWS CloudTrail 記錄 DynamoDB 操作</a> 及《 <a href="#">AWS CloudTrail 使用者指南</a> 》。	2015 年 5 月 28 日
改善 Query 表達式的支援	此版本對 KeyConditionExpression API 新增了 Query 參數。Query 會使用主索引鍵值從資料表或索引讀取項目。KeyConditionExpression 參數是識別主索引鍵名稱的字串，也是要套用到索引鍵值的條件；Query 只會擷取那些符合表達式的項目。KeyConditionExpression 的語法與 DynamoDB 中其他表達式參數的語法類似，可讓您在表達式中定義名稱和值的替換變數。如需詳細資訊，請參閱 <a href="#">DynamoDB 中的查詢操作</a> 。	2015 年 4 月 27 日

變更	描述	變更日期
條件式寫入的新比較函數	<p>在 DynamoDB 中，Condition Expression 參數可決定 PutItem、UpdateItem 或 DeleteItem 是否成功：只有當條件評估為 true 時才會寫入項目。此版本新增了兩個函數 attribute_type 和 size，配合 Condition Expression 一起使用。這些函數可讓您根據資料表中屬性的資料類型或大小，執行條件式寫入。如需詳細資訊，請參閱 <a href="#">條件表達式</a>。</p>	2015 年 4 月 27 日
次要索引的掃描 API	<p>在 DynamoDB 中，Scan 操作會讀取資料表中的所有項目，套用使用者定義的篩選條件，再將所選資料項目傳回給應用程式。這項功能現在同樣可用於次要索引。若要掃描本機次要索引或全域次要索引，您要指定索引名稱及其父資料表的名稱。根據預設，索引 Scan 會傳回索引中的所有資料，您可使用篩選條件表達式縮減傳回給應用程式的結果。如需詳細資訊，請參閱 <a href="#">在 DynamoDB 中使用掃描</a>。</p>	2015 年 2 月 10 日



變更	描述	變更日期
全域次要索引的線上操作	<p>線上索引可讓您對現有資料表新增或移除全域次要索引。</p> <p>使用線上索引時，您不需要在建立資料表時定義資料表的所有索引，而可隨時新增索引。同樣地，如果您認為不再需要某個索引，就可以隨時予以移除。線上索引不是阻擋式操作，在索引新增或移除時，仍可讀取和寫入資料表。如需詳細資訊，請參閱 <a href="#">管理全域次要索引</a>。</p>	2015 年 1 月 27 日
JSON 的文件模型支援	<p>DynamoDB 可讓您使用完整的文件模型支援，存放及擷取文件。新的資料類型與 JSON 標準完全相容，可讓您將文件元素彼此巢狀嵌入。您可以使用文件路徑取消參考運算子讀取及寫入個別元素，而不必擷取整份文件。此版本也會推出新的表達式參數，用以在讀取或寫入資料項目時指定投影、條件和更新動作。若要進一步了解 JSON 文件模型支援，請參閱 <a href="#">資料類型</a> 和 <a href="#">在 DynamoDB 中使用表達式</a>。</p>	2014 年 10 月 7 日

變更	描述	變更日期
彈性調整規模	您可為資料表及全域次要索引增加任意數量的佈建讀取輸送容量和佈建寫入輸送容量，前提是您未超過每份資料表及每個帳戶的限制。如需詳細資訊，請參閱 <a href="#">Amazon DynamoDB 中的服務、帳戶和資料表配額</a> 。	2014 年 10 月 7 日
更大的項目大小	DynamoDB 的項目大小上限已從 64 KB 提高至 400 KB。如需詳細資訊，請參閱 <a href="#">Amazon DynamoDB 中的服務、帳戶和資料表配額</a> 。	2014 年 10 月 7 日
改善條件表達式	DynamoDB 擴展了條件表達式可用的運算子，讓您能更靈活地進行條件式放置、更新和刪除。新提供的運算子可讓您檢查屬性是否存在、是否大於或小於特定值、是否介於兩值之間、是否以特定字元開頭等。DynamoDB 也提供選用的 OR 運算子來評估多個條件。根據預設，表達式中的多項條件使用 AND 連在一起，所以只有當表達式的所有條件都為 true 時，表達式才為 true。如果您改為指定 OR，當一或多項條件為 true 時，表達式即為 true。如需詳細資訊，請參閱 <a href="#">使用項目和屬性</a> 。	2014 年 4 月 24 日

變更	描述	變更日期
查詢篩選條件	<p>DynamoDB Query API 支援新的 QueryFilter 選項。根據預設，Query 會尋找符合特定分割區索引鍵值和選用排序索引鍵條件的項目。Query 篩選條件會將條件表達式套用到其他非索引鍵屬性，如果 Query 篩選條件存在，則會在 Query 結果傳回應用程式前，捨棄不符合篩選條件的項目。如需詳細資訊，請參閱 <a href="#">DynamoDB 中的查詢操作</a>。</p>	2014 年 4 月 24 日
資料匯出和匯入 AWS Management Console	<p>DynamoDB 主控台已強化，可簡化 DynamoDB 資料表的資料匯出與匯入。只需按幾下滑鼠，您就可以設定協調工作流程，並設定 Amazon 彈性 MapReduce 叢集將資料從 DynamoDB 表複製到 Amazon S3 儲存貯體，反之亦然。</p> <p>AWS Data Pipeline 您只可執行一次匯出或匯入，或者設定每日匯出任務。您甚至可以執行跨區域匯出和匯入，將某個區域中的表格中的 DynamoDB 資料複製到另一個 AWS 區域的表格。AWS 如需詳細資訊，請參閱 <a href="#">使用以下方式匯出和匯入 DynamoDB 資料 AWS Data Pipeline</a>。</p>	2014 年 3 月 6 日

變更	描述	變更日期
重新整理較上層 API 的文件	<p>現已可輕易取得下列 API 的資訊：</p> <ul style="list-style-type: none"><li>• Java : DynamoDBMapper</li><li>• .NET : 文件模型和物件持久性模型</li></ul> <p>這些較上層的 API 現在都記錄在這裡：<a href="#">適用於 DynamoDB 的更高階程式設計界面</a>。</p>	2014 年 1 月 20 日
全域次要索引	<p>DynamoDB 新增了全域次要索引的支援。如同本機次要索引，您使用資料表中的其他索引鍵來定義全域次要索引，再對索引發出 Query 請求。但與本機次要索引不同的是，全域次要索引的分割區索引鍵不必和資料表的分割區索引鍵一樣，而可以是該資料表中任何純量屬性。排序索引鍵為選用，而且也可以是任何純量資料表屬性。全域次要索引也有自己的佈建輸送量設定，而且與父資料表的佈建輸送量設定無關。如需詳細資訊，請參閱<a href="#">使用次要索引來改善資料存取</a>及<a href="#">在 DynamoDB 中使用全域次要索引</a>。</p>	2013 年 12 月 12 日

變更	描述	變更日期
精細定義存取控制	<p>DynamoDB 新增了精細定義存取控制的支援。這項功能可讓客戶指定哪些委託人 (使用者、群組或角色) 可以存取 DynamoDB 資料表或次要索引中的個別項目和屬性。應用程式也可充分利用 Web 聯合身分，將使用者身分驗證的任務交給第三方身分提供者，例如 Facebook、Google 或 Login with Amazon。如此，應用程式 (包括行動應用程式) 即可處理非常大量的使用者，同時確保除非使用者獲得授權，否則均無法存取 DynamoDB 資料項目。如需詳細資訊，請參閱 <a href="#">使用 IAM 政策條件進行精細定義存取控制</a>。</p>	2013 年 10 月 29 日
4 KB 讀取容量單位大小	<p>讀取容量單位大小已從 1 KB 提高至 4 KB。這項強化功能可以減少許多應用程式所需的佈建讀取容量單位數目。例如，在這個版本以前，讀取 10 KB 的項目會使用 10 個讀取容量單位；而現在同樣讀取 10 KB 只會使用 3 個單位 (10 KB / 4 KB，四捨五入到下一個 4 KB 內)。如需詳細資訊，請參閱 <a href="#">輸送量容量</a>。</p>	2013 年 5 月 14 日

變更	描述	變更日期
平行掃描	DynamoDB 新增了對平行掃描操作的支援。應用程式現在已可將資料表分割成多個邏輯區段，然後同時掃描所有區段。這項功能可減少完成掃描所需的時間，並充分利用資料表的佈建讀取容量。如需詳細資訊，請參閱 <a href="#">在 DynamoDB 中使用掃描</a> 。	2013 年 5 月 14 日
本機次要索引	DynamoDB 新增了對本機次要索引的支援。您可以對非索引鍵屬性定義排序索引鍵索引，然後在 Query 請求中使用這些索引。使用本機次要索引時，應用程式可跨多維度有效率地擷取資料項目。如需詳細資訊，請參閱 <a href="#">本機次要索引</a> 。	2013 年 4 月 18 日
新的 API 版本	DynamoDB 在此版本中推出新的 API 版本 (2012-08-10)。並仍支援舊版 API (2011-12-05)，以與現有應用程式回溯相容。新的應用程式應使用新的 API 版本 2012-08-10。建議您將現有的應用程式遷移至最新的 API 版本 (2012-08-10)，因為新的 DynamoDB 功能 (例如本機次要索引) 不會向後移植到舊版的 API。如需 API 版本 2012-08-10 的詳細資訊，請參閱 <a href="#">《Amazon DynamoDB API 參考》</a> 。	2013 年 4 月 18 日

變更	描述	變更日期
IAM 政策變數支援	<p>IAM 存取政策語言現在支援變數。當政策接受評估時，任何政策變數都會取代為已驗證使用者工作階段中內容資訊提供的值。您可使用政策變數定義一般用途政策，並不需明確列出政策的所有元件。如需政策變數的詳細資訊，請參閱AWS Identity and Access Management 使用 IAM 指南中的<a href="#">政策變數</a>。</p> <p>如需 DynamoDB 的政策變數範例，請參閱 <a href="#">Amazon DynamoDB 的 Identity and Access Management</a>。</p>	2013 年 4 月 4 日
PHP 代碼示例更新了 AWS SDK for PHP 版本 2	<p>的版本 2 現 AWS SDK for PHP 在可供使用。《Amazon DynamoDB 開發人員指南》中的 PHP 程式碼範例已更新，可使用這項新的開發套件。如需開發套件第 2 版的詳細資訊，請參閱 <a href="#">AWS SDK for PHP</a>。</p>	2013 年 1 月 23 日
新的端點	<p>DynamoDB 會擴展至 AWS GovCloud (美國西部) 區域。如需目前的服務端點和協定清單，請參閱<a href="#">區域與端點</a>。</p>	2012 年 12 月 3 日
新的端點	<p>DynamoDB 擴展至南美洲 (聖保羅) 區域。如需目前的支援端點清單，請參閱<a href="#">區域與端點</a>。</p>	2012 年 12 月 3 日

變更	描述	變更日期
新的端點	DynamoDB 擴展至亞太區域 (雪梨)。如需目前的支援端點清單，請參閱 <a href="#">區域與端點</a> 。	2012 年 11 月 13 日
DynamoDB 實作 CRC32 檢查總和的支援、支援強烈一致批次擷取，及移除對並行資料表更新的限制。	<ul style="list-style-type: none"> <li>• DynamoDB 會計算 HTTP 承載的 CRC32 檢查總和，並在新的標頭 <code>x-amz-crc32</code> 中傳回此檢查總和。如需詳細資訊，請參閱 <a href="#">DynamoDB 低階 API</a>。</li> <li>• 根據預設，BatchGetItem API 執行的讀取操作為最終一致。ConsistentRead 中的新 BatchGetItem 參數可讓您為請求中的任何資料表改選擇強式讀取一致性。如需詳細資訊，請參閱 <a href="#">描述</a>。</li> <li>• 此版本會在同時更新許多資料表時，移除一些限制。一次可更新的資料表總數仍為 10，但這些資料表現在可以是 CREATING、UPDATING 或 DELETING 狀態的任意組合。此外，不再有任何增加或減少或表格的ReadCapacityUnits最小數量。WriteCapacityUnits如需詳細資訊，請參閱 <a href="#">Amazon DynamoDB 中的服務、帳戶和資料表配額</a>。</li> </ul>	2012 年 11 月 2 日



變更	描述	變更日期
最佳實務文件	《Amazon DynamoDB 開發人員指南》提供使用資料表和項目的最佳實務，以及查詢和掃描操作的建議。	2012 年 9 月 28 日

變更	描述	變更日期
二進位資料類型的支援	<p>除數字和字串類型外，DynamoDB 現在也支援二進位資料類型。</p> <p>在此版本以前，若要存放二進位資料，您要將二進位資料轉換成字串格式存放在 DynamoDB 中。除了用戶端必要的轉換工作，轉換通常會增加資料項目的大小，而需要更多儲存體和可能的額外佈建輸送容量。</p> <p>使用二進位類型屬性，您現在已可存放任何二進位資料，例如壓縮的資料、加密的資料和影像。如需詳細資訊，請參閱 <a href="#">資料類型</a>。有關使用 AWS SDK 處理二進制類型數據的工作示例，請參閱以下各節：</p> <ul style="list-style-type: none"><li>• <a href="#">範例：使用 AWS SDK for Java Document API 處理二進位類型屬性</a></li><li>• <a href="#">範例：使用 AWS SDK for .NET 低階 API 處理二進位類型屬性</a></li></ul> <p>對於 AWS SDK 中添加的二進制數據類型支持，您將需要下載最新的 SDK，您可能還需要更新任何現有的應用程序。如需下載 AWS 開發套件的資訊，請參閱「<a href="#">.NET 程式碼範例</a>」。</p>	2012 年 8 月 21 日

變更	描述	變更日期
<p>您可使用 DynamoDB 主控台更新及複製 DynamoDB 資料表項目</p>	<p>DynamoDB 使用者現在除了可新增及刪除項目之外，還可以使用 DynamoDB 主控台更新及複製資料表項目。這項新功能簡化了透過主控台變更個別項目的程序。</p>	<p>2012 年 8 月 14 日</p>
<p>DynamoDB 降低了最低資料表輸送量需求</p>	<p>DynamoDB 現在支援較低的最低資料表輸送量需求，具體為 1 個寫入容量單位和 1 個讀取容量單位。如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的 <a href="#">Amazon DynamoDB 中的服務、帳戶和資料表配額</a> 主題。</p>	<p>2012 年 8 月 9 日</p>
<p>Signature 第 4 版支援</p>	<p>DynamoDB 現在支援 Signature 第 4 版驗證請求。</p>	<p>2012 年 7 月 5 日</p>
<p>DynamoDB 主控台內的 Table Explorer 支援</p>	<p>DynamoDB 主控台現在支援 Table Explorer，讓您可瀏覽及查詢資料表中的資料。您也可以插入新的項目或刪除現有的項目。已針對這些功能更新 <a href="#">在 DynamoDB 中建立資料表，以及載入程式碼範例的資料</a> 和 <a href="#">使用主控台</a> 章節。</p>	<p>2012 年 5 月 22 日</p>
<p>新的端點</p>	<p>DynamoDB 可用性因美國西部 (加利佛尼亞北部) 區域、美國西部 (奧勒岡) 區域和亞太區域 (新加坡) 中的新端點而得以擴展。</p> <p>如需目前的支援端點清單，請前往 <a href="#">區域與端點</a>。</p>	<p>2012 年 4 月 24 日</p>

變更	描述	變更日期
BatchWriteItem API 支援	DynamoDB 現已支援批次寫入 API，讓您在單一 API 呼叫中對一或多個資料表，存放與刪除多個項目。如需 DynamoDB 批次寫入 API 的詳細資訊，請參閱「 <a href="#">BatchWriteItem</a> 」。  如需使用 AWS SDK 處理項目和使用批次寫入功能的相關資訊，請參閱 <a href="#">使用項目和屬性和.NET 程式碼範例</a> 。	2012 年 4 月 19 日
記錄更多錯誤碼	如需詳細資訊，請參閱 <a href="#">使用 DynamoDB 時發生錯誤</a> 。	2012 年 4 月 5 日
新的端點	DynamoDB 擴展至亞太區域 (東京)。如需目前的支援端點清單，請參閱 <a href="#">區域與端點</a> 。	2012 年 2 月 29 日
新增了 ReturnedItemCount 指標	新的量度提供 DynamoDB 的查詢或掃描作業回應中傳回的項目數 ReturnedItemCount，可透過以下方式進行監視。CloudWatch	2012 年 2 月 24 日
新增了增加數值的範例	DynamoDB 支援增加和減少現有的數值。這些範例會在下列主題的「更新項目」章節中示範新增至現有數值：  <a href="#">使用項目：Java</a> 。  <a href="#">處理項目：.NET</a> 。	2012 年 1 月 25 日
初始產品版本	DynamoDB 作為 Beta 版新服務推出。	2012 年 1 月 18 日

# 舊版 DynamoDB 能

下列主題是 DynamoDB 仍支援的舊版功能。這些功能沒有積極開發。

## 主題

- [全域表格版本 \(舊版\)](#)

## 全域表格版本 (舊版)

### Important

本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用[全域表版本 2019.11.21 \(目前版本\)](#)，因為它比 2017.11.29 (舊版) 提供更大的彈性、更高的效率以及更少的寫入容量。

若要判斷您使用的是哪個版本，請參閱[判斷您正在使用的全域資料表版本](#)。若要將現有全域資料表從 2017.11.29 版更新至 2019.11.21 版 (目前)，請參閱[升級全域資料表](#)。

## 主題

- [全域資料表：運作方式](#)
- [管理全域資料表的最佳實務和要求](#)
- [建立全域資料表](#)
- [監控全域資料表](#)
- [對全域資料表使用 IAM](#)

## 全域資料表：運作方式

### Important

本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用[全域表版本 2019.11.21 \(目前版本\)](#)，因為它比 2017.11.29 (舊版) 提供更大的彈性、更高的效率以及更少的寫入容量。

若要判斷您使用的是哪個版本，請參閱[判斷您正在使用的全域資料表版本](#)。若要將現有全域資料表從 2017.11.29 版更新至 2019.11.21 版 (目前)，請參閱[升級全域資料表](#)。

以下各節可協助您了解 Amazon DynamoDB 中全域資料表的概念和行為。

## 2017.11.29 版 (舊版) 的全域資料表概念

全域資料表是一或多個複本列表的集合，全由單個 AWS 帳戶擁有。

複本列表 (簡稱複本) 是單一 DynamoDB 資料表，可作為全域資料表的一部分運作。每個複本會存放相同的資料項目集。任何特定全域資料表的每個 AWS 區域只能有一個複本列表。

以下是如何建立全域資料表的概念性概觀。

1. 在啟用 DynamoDB Streams 的情況下，在 AWS 區域建立一般 DynamoDB 資料表。
2. 針對要複寫資料的每個其他區域重複步驟 1。
3. 根據您建立的資料表定義 DynamoDB 全域資料表。

AWS Management Console 會自動執行這些任務，因此您可以更快速、更輕鬆地建立全域資料表。如需詳細資訊，請參閱 [建立全域資料表](#)。

產生的 DynamoDB 全域資料表由多個複本列表組成，DynamoDB 會將其視為單一單位 (每個區域一個)。每個複本都有相同的資料表名稱和相同的主索引鍵結構描述。當應用程式將資料寫入某個區域的複本列表時，DynamoDB 會自動將寫入傳播到另一個 AWS 區域的其他複本列表。

### Important

為了讓資料表資料保持同步，全域資料表會自動為每個項目建立下列屬性：

- `aws:rep:deleting`
- `aws:rep:updatetime`
- `aws:rep:updateregion`

請勿修改這些屬性或建立具有相同名稱的屬性。

您可以將複本列表新增到全域資料表，以便在其他區域中使用該複本列表。(若要執行這項操作，全域資料表必須是空的。換句話說，任何複本列表中都不得包含任何資料。)

您也可以從全域資料表中移除複本列表。如果您執行這項操作，則資料表與全域資料表的關聯會完全取消。這個新的獨立資料表不再與全域資料表互動，而且資料不再傳播至全域資料表或從全域資料表傳播。

### Warning

請注意，移除複本不是自動的。為了確保一致的行為和已知狀態，您可能需要考慮事先將應用程式寫入流量從要移除的複本轉移。移除後，請等候所有複本區域端點將複本顯示為已解除關聯，再將其作為自己的隔離區域表進行進一步的寫入操作。

## 一般任務

全域資料表的一般任務如下。

您可以刪除全域資料表的複本資料表，方式與一般資料表相同。這將停止複寫到該區域，並刪除保留在該區域中的資料表複本。您無法中斷複寫，並讓資料表的複本以獨立實體的形式存在。

### Note

在來源資料表用於啟動新區域後至少 24 小時之後，您才能刪除來源資料表。若您嘗試刪除，很快便會收到錯誤。

如果應用程式大約在同一時間更新不同區域中的相同項目，則可能會發生衝突。為了協助確保最終一致性，DynamoDB 全域資料表會在並行更新間使用「最後寫入者獲勝」方法，在並行更新間使用最後寫入者。所有的複本都會同意最新的更新，並朝它們都具有相同資料的狀態收斂。

### Note

有幾種方式可以避免衝突，包括：

- 使用 IAM 政策，只允許寫入一個區域中的資料表。
- 使用 IAM 政策，將使用者路由到一個區域，並將另一個區域保持為閒置待命狀態，或者交替將奇數使用者路由到一個區域，甚至將使用者路由到另一個區域。
- 避免使用非等冪更新，例如書籤 = 書籤 + 1，以支援靜態更新，例如書籤 = 25。

## 監控全域資料表

您可以使 CloudWatch 用觀察度量標準 `ReplicationLatency`。此量度會追蹤一個複本表格的 DynamoDB 串流中出現更新項目，以及該項目出現在全域表格的另一個複本之間的經過時間。

ReplicationLatency以毫秒為單位表示，並針對每個來源-區域和目的地-區域配對發出。這是全域表 v2 提供的唯一 CloudWatch 測量結果。

您觀察到的延遲取決於所選擇區域之間的距離以及其他變數。區域的 0.5 到 2.5 秒範圍內的延遲在同一地理區域內可能很常見。

## 存留時間 (TTL)

您可以使用存留時間 (TTL) 來指定屬性名稱，其值表示項目的到期時間。這個值被指定為自 Unix 紀元開始以來的秒數。

使用全域資料表舊版本時，TTL 刪除作業不會自動跨其他複本複寫。當透過 TTL 規則刪除項目時，會執行該工作，而不會耗用寫入單位。

請注意，如果來源和目標資料表的佈建寫入容量非常低，這可能會造成限流，因為 TTL 刪除需要寫入容量。

## 使用全域資料表的串流和交易

每個全域資料表都會根據其所有寫入作業產生獨立的串流，無論這些寫入的起始點為何。您可以選擇在一個區域或所有區域中獨立使用此 DynamoDB 串流。

如果您想要處理本機寫入，但不要複製寫入，可以將自己的區域屬性新增至每個項目。然後，您可以使用 Lambda 事件篩選條件，只叫用 Lambda 在本機區域中進行寫入。

交易操作僅在最初進行寫入的區域內提供 ACID (不可分割性、一致性，隔離性和耐久性) 保證。全域資料表不支援跨區域交易。

例如，如果您在美國東部 (俄亥俄) 和美國西部 (奧勒岡) 區域有一個包含複本的全域表格，並在美國東部 (俄亥俄) 區域執行 TransactWriteItems 作業，則可能會在美國西部 (奧勒岡) 區域觀察到部分完成的交易，當變更複寫時。只有當變更已在來源區域遞交的情況下，這些變更才會複寫至其他區域。

### Note

- 全域資料表透過直接更新 DynamoDB 來「寫入」DynamoDB 加速器。因此，DAX 不會意識到它持有過時的資料。只有在快取的 TTL 到期時，才會重新整理 DAX 快取。
- 全域資料表上的標籤不會自動傳播。



## 讀取和寫入輸送量

全域資料表會以下列方式管理讀取和寫入輸送量。

- 跨區域的所有表格執行個體的寫入容量必須相同。
- 使用版本 2019.11.21 (目前) 時，如果資料表設定為支援自動擴展資源或處於隨需模式，則寫入容量會自動保持同步。每個區域目前佈建的寫入容量將在這些同步的自動擴展資源設定中獨立增加和減少。如果將資料表置於隨需模式，則該模式將同步至其他複本。
- 區域之間的讀取容量可能會有所不同，因為讀取可能不相等。將全域複本新增至資料表時，會傳播來源區域的容量。建立之後，您可以調整一個複本的讀取容量，而這個新設定不會傳輸到另一端。

## 一致性和衝突解決

對任何複本列表中任何項目所做的任何變更都會複寫到相同全域資料表中的所有其他複本。在全域資料表中，新寫入的項目通常會在幾秒鐘內傳播到所有複本列表。

使用全域資料表，每個複本列表會存放相同的資料項目集。DynamoDB 不支援僅對某些項目進行部分複寫。

應用程式可以讀取資料和將資料寫入至任何複本列表。DynamoDB 支援跨區域的最終一致讀取，但不支援跨區域的高度一致性讀取。如果應用程式只使用最終一致讀取，並且只對一個 AWS 區域發出讀取，則該應用程式不需任何修改即可運作。不過，如果您的應用程式需要強式一致讀取，則必須在相同區域中執行所有高度一致性讀取和寫入。否則，如果您對某個區域進行寫入並從另一個區域進行讀取，則讀取回應可能包含過時資料，這些資料不會反映最近在另一個區域中完成的寫入結果。

如果應用程式大約在同一時間更新不同區域中的相同項目，則可能會發生衝突。為了協助確保最終一致性，DynamoDB 全域資料表會在並行更新間使用最後寫入者獲勝核對機制，其中 DynamoDB 會在並行更新間盡最大努力判斷最後寫入者。有了這個衝突解決機制，所有的複本都會同意最新的更新，並朝它們都具有相同資料的狀態收斂。

## 可用性與持久性

如果單一 AWS 區域變得孤立或降級，您的應用程式可以重新導向到不同的區域，並對不同的複本列表執行讀取和寫入。您可以套用自訂商業邏輯來決定何時將請求重新導向至其他區域。

如果區域變得孤立或降級，DynamoDB 會追蹤已執行但尚未傳播到所有複本列表的任何寫入。當區域重新回到線上的狀態時，DynamoDB 會繼續將該區域的任何擱置寫入傳播到其他區域的複本列表中。其也會繼續將寫入從其他複本列表傳播到目前已重回到線上狀態的區域。無論區域隔離多長時間，所有之前成功的寫入都將最終傳播。

## 管理全域資料表的最佳實務和要求

### Important

本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用 [全域表版本 2019.11.21 \(目前版本\)](#)，因為它比 2017.11.29 (舊版) 提供更大的彈性、更高的效率以及更少的寫入容量。

若要判斷您使用的是哪個版本，請參閱 [判斷您正在使用的全域資料表版本](#)。若要將現有全域資料表從 2017.11.29 版更新至 2019.11.21 版 (目前)，請參閱 [升級全域資料表](#)。

您可以使用 Amazon DynamoDB 全域表格，跨 AWS 區域複寫表格資料。全域資料表中的複本列表和次要索引必須具有相同的寫入容量設定，才能確保資料能正確複寫。

### 主題

- [全域資料表版本](#)
- [新增複本資料表的要求](#)
- [管理容量的最佳實務和要求](#)

## 全域資料表版本

有兩個版本的 DynamoDB 全域資料表可供使用：[全域表版本 2019.11.21 \(目前版本\)](#) 和 [全域表格版本 \(舊版\)](#)。客戶應盡可能使用全域表版本 2019.11.21 (目前版本)，因為它比 2017.11.29 (舊版) 提供更大的彈性、更高的效率以及更少的寫入容量。

若要判斷您使用的是哪個版本，請參閱 [判斷您正在使用的全域資料表版本](#)。若要將現有全域資料表從 2017.11.29 版更新至 2019.11.21 版 (目前)，請參閱 [升級全域資料表](#)。

## 新增複本資料表的要求

若想將新的複本列表新增至全域資料表，下列每一個條件都必須成立：

- 資料表必須具備與所有其他複本相同的分割區索引鍵。
- 資料表必須具備指定的寫入容量管理設定。
- 資料表必須具備與所有其他複本相同的名稱。
- 資料表必須啟用 DynamoDB Streams，並且串流同時包含項目的新映像和舊映像。
- 全域資料表中的新或現有複本列表都不能包含任何資料。

如果指定全域次要索引，則也必須符合下列條件：

- 全域次要索引必須具有相同的名稱。
- 全域次要索引必須具有同樣的分割區索引鍵和排序索引鍵 (若有)。

### Important

應在全域資料表的所有複本列表和相符的次要索引之間一致地設定寫入容量設定。若要更新全域資料表的寫入容量設定，強烈建議使用 DynamoDB 主控台或 UpdateGlobalTableSettings API 操作。UpdateGlobalTableSettings 會自動將寫入容量設定的變更套用至所有複本列表和全域資料表中相符的次要索引。如果使用 UpdateTable、RegisterScalableTarget 或 PutScalingPolicy 操作，您應該將變更套用至每個複本列表和個別相符的次要索引。如需詳細資訊，請參閱 [Amazon DynamoDB API 參考 UpdateGlobalTableSettings](#) 中的。

我們強烈建議您啟用自動調整規模來管理佈建的寫入容量設定。如果您偏好手動管理寫入容量設定，則應為所有複本列表佈建相同的複寫寫入容量單位。此外，您還要為全域資料表中的相符次要索引佈建相同的複寫寫入容量單位。

您還必須擁有適當的 AWS Identity and Access Management (IAM) 許可。如需詳細資訊，請參閱 [對全域資料表使用 IAM](#)。

## 管理容量的最佳實務和要求

在 DynamoDB 中管理複本列表的容量設定時，請考慮下列事項。

### 使用 DynamoDB Auto Scaling

使用 DynamoDB Auto Scaling 功能管理輸送容量設定的建議方式，適用於使用佈建模式的複本列表。DynamoDB 自動擴展功能會根據實際應用程式工作負載，自動調整每個複本列表的讀取容量單位 (RCU) 和寫入容量單位 (WCU)。如需詳細資訊，請參閱 [使用 DynamoDB Auto Scaling 功能自動管理輸送容量](#)。

如果您使用建立複本表格 AWS Management Console，依預設會為每個複本表格啟用 auto 調整規模，並使用預設的 auto 調整設定來管理讀取容量單位和寫入容量單位。

透過 DynamoDB 主控台或使用 UpdateGlobalTableSettings 呼叫對複本列表或次要索引的自動調整規模設定所做的變更，會自動套用至所有複本列表，以及全域資料表中的相符次要索引。這些變更會覆寫任何現有的自動調整規模設定。這可確保佈建的寫入容量設定在全域資料表中的

複本列表和次要索引之間保持一致。如果使用 UpdateTable、RegisterScalableTarget 或 PutScalingPolicy 呼叫，您應該將變更個別套用至每個複本列表，以及相符的次要索引。

### Note

如果自動調整規模無法滿足應用程式的容量變更 (無法預測的工作負載)，或者您不想進行其設定 (最小、最大或使用率閾值的目標設定)，則可以使用隨需模式來管理全域資料表的容量。如需詳細資訊，請參閱 [隨需模式](#)。

如果您在全域資料表上啟用隨需模式，則複寫寫入請求單元 (rWCU) 的耗用方式會與 rWCU 的佈建方式一致。例如，如果您對在另外兩個區域中複寫的本機資料表執行 10 次寫入，則會耗用 60 個寫入要求單位 ( $10 + 10 + 10 = 30$ ； $30 \times 2 = 60$ )。耗用的 60 個寫入請求單位包括全域資料表版本 2017.11.29 (舊版) 所耗用的額外寫入，以更新 `aws:rep:deleting`、`aws:rep:updatetime` 和 `aws:rep:updateregion` 屬性。

## 手動管理容量

如果您決定不使用 DynamoDB Auto Scaling 功能，則必須在每個複本列表和次要索引上手動設定讀取容量和寫入容量設定。

每個複本列表上佈建的複寫寫入容量單位 (rWCU) 應設定為將跨所有區域的應用程式寫入所需的 rWCU 總數乘以二。這可容納在本地區域發生的應用程式寫入，以及來自其他區域的複寫應用程式寫入。例如，假設您預期在俄亥俄州的複本列表每秒 5 次寫入，在維吉尼亞北部複本列表每秒 5 次寫入。在此情況下，您應該為每個複本表格佈建 20 個 RWCU ( $5 + 5 = 10$ ； $10 \times 2 = 20$ )。

若要更新全域資料表的寫入容量設定，強烈建議使用 DynamoDB 主控台或 UpdateGlobalTableSettings API 操作。UpdateGlobalTableSettings 會自動將寫入容量設定的變更套用至所有複本列表和全域資料表中相符的次要索引。如果使用 UpdateTable、RegisterScalableTarget 或 PutScalingPolicy 操作，您應該將變更套用至每個複本列表和個別相符的次要索引。如需詳細資訊，請參閱 [《Amazon DynamoDB API 參考》](#)。

### Note

若要更新 DynamoDB 中全域資料表的設定 (UpdateGlobalTableSettings)，您必須擁有 `dynamodb:UpdateGlobalTable`、`dynamodb:DescribeLimits`、`application-autoscaling:DeleteScalingPolicy` 以及 `application-autoscaling:DeregisterScalableTarget` 許可。如需更多詳細資訊，請參閱 [對全域資料表使用 IAM](#)。

## 建立全域資料表

### ⚠ Important

本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用 [全域表版本 2019.11.21 \(目前版本\)](#)，因為它比 2017.11.29 (舊版) 提供更大的彈性、更高的效率以及更少的寫入容量。

若要判斷您使用的是哪個版本，請參閱 [判斷您正在使用的全域資料表版本](#)。若要將現有全域資料表從 2017.11.29 版更新至 2019.11.21 版 (目前)，請參閱 [升級全域資料表](#)。

本節說明如何使用 Amazon DynamoDB 主控台或 AWS Command Line Interface (AWS CLI) 建立全域資料表。

### 主題

- [建立全域資料表 \(主控台\)](#)
- [建立全域資料表 \(AWS CLI\)](#)

### 建立全域資料表 (主控台)

請遵循下列步驟，使用主控台建立全域資料表。下列範例使用美國及歐洲的複本列表，建立全域資料表。

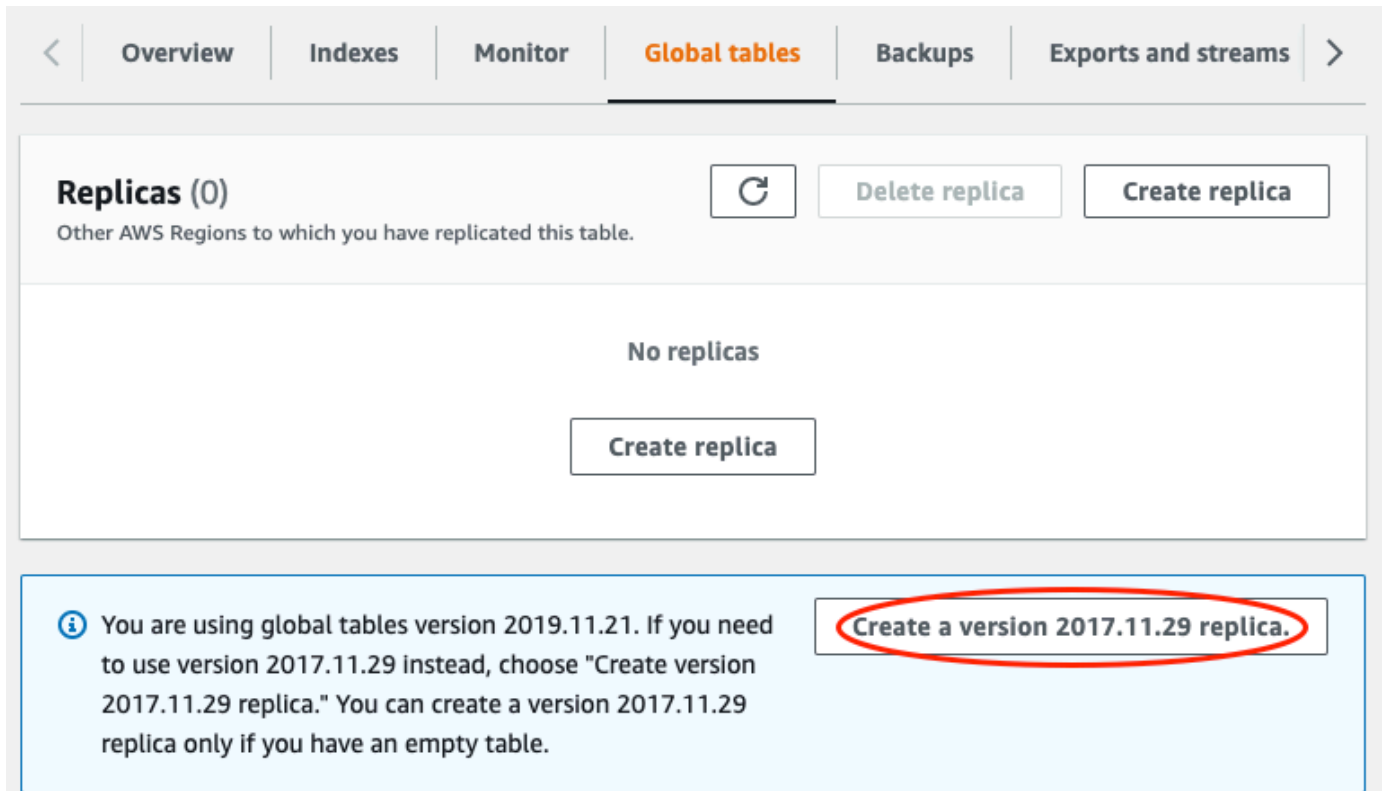
1. 在 <https://console.aws.amazon.com/dynamodb/home> 開啟 DynamoDB 主控台。針對此範例，請選擇 us-east-2 (美國東部俄亥俄州) 區域。
2. 在主控台左側的導覽窗格中，選擇 Tables (資料表)。
3. 選擇 Create Table (建立資料表)。

對於 Table name (資料表名稱)，請輸入 **Music**。

針對 Primary key (主索引鍵)，輸入 **Artist**。選擇 Add sort key (新增排序索引鍵)，然後輸入 **SongTitle**。( **Artist** 和 **SongTitle** 都必須是字串。)

若要建立資料表，請選擇 Create (建立)。此資料表會做為新全域資料表中的第一個複本資料表。這是您稍後新增其他複本資料表的原型。

4. 選擇 Global Tables (全域資料表) 索引標籤，然後選擇 Create a version 2017.11.29 (Legacy) replica (建立版本 2017.11.29 (舊版) 複本)。



The screenshot shows the AWS Management Console interface for Global Tables. At the top, there are navigation tabs: Overview, Indexes, Monitor, Global tables (selected), Backups, and Exports and streams. Below the tabs, there's a section for 'Replicas (0)' with a subtitle 'Other AWS Regions to which you have replicated this table.' This section contains a refresh icon, 'Delete replica', and 'Create replica' buttons. A large 'No replicas' message is centered, with a 'Create replica' button below it. At the bottom, a blue information box contains a message: '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.' A red circle highlights the 'Create a version 2017.11.29 replica.' button within this message.

5. 從 Available replication Regions (可用的複製區域) 下拉式清單，選擇 US West (Oregon) (美國西部 (奧勒岡))。

主控台會檢查，確認所選取的區域中沒有同名的資料表。若有同名的資料表，您必須先刪除現有的資料表，才可在該區域中建立新的複本資料表。

6. 選擇 Create replica (建立複本)。這會啟動在美國西部 (奧勒岡) 建立資料表的程序。

所選資料表的 Global Table (全域資料表) 標籤 (及所有其他複本資料表)，會顯示該資料表已複寫到多個區域。

7. 您現在可以新增其他區域，複寫及同步美國與歐洲的全域資料表。若要執行此作業，請重複步驟 5，但這次請改為指定 Europe (Frankfurt) (歐洲 (法蘭克福))，而非 US West (Oregon) (美國西部 (奧勒岡))。
8. 您應該仍會在美國東部 (俄亥俄) 區域使用 AWS Management Console。選取左側導覽功能表中的 Items (項目)，選取 Music (音樂) 資料表，然後選擇 Create Item (建立項目)。
  - a. 針對 Artist (藝人)，輸入 **item\_1**。
  - b. 針對 SongTitle，請輸入 **Song Value 1**。
  - c. 若要寫入該項目，請選擇 Create item (建立項目)。



- 稍待片刻之後，該項目將會複寫到您全域資料表中的所有三個區域。若要確認，請在主控台中移至右上角的區域選擇器，然後選擇 Europe (Frankfurt) (歐洲 (法蘭克福))。歐洲 (法蘭克福) 中的 Music 資料表此時應已包含新的項目。
- 重複步驟 9，然後選擇 US West (Oregon) (美國西部 (奧勒岡)) 以驗證該區域中的複寫。

## 建立全域資料表 (AWS CLI)

請遵循下列步驟，使用 AWS CLI 建立全域資料表 Music。以下範例會建立全域資料表，並在美國及歐洲皆擁有複本資料表。

- 在美國東部 (俄亥俄) 建立新的資料表 (Music)，並啟用 DynamoDB Streams (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
```

- 在美國東部 (維吉尼亞北部) 建立相同的 Music 資料表。

```
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. 建立全域資料表 (Music) , 其中包含 us-east-2 和 us-east-1 區域中的複本列表。

```
aws dynamodb create-global-table \
 --global-table-name Music \
 --replication-group RegionName=us-east-2 RegionName=us-east-1 \
 --region us-east-2
```

#### Note

全域資料表名稱 (Music) 必須與每個複本列表 (Music) 的名稱相符。如需詳細資訊，請參閱 [管理全域資料表的最佳實務和要求](#)。

4. 在歐洲 (愛爾蘭) 中建立另一個資料表，其設定與您在步驟 1 和步驟 2 中建立的資料表設定相同。

```
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
```

完成此步驟後，將新資料表新增至 Music 全域資料表。

```
aws dynamodb update-global-table \
 --global-table-name Music \
 --replica-updates 'Create={RegionName=eu-west-1}' \
 --region us-east-2
```

5. 若要確認複寫正常運作，請將項目新增到美國東部 (俄亥俄) 中的 Music 資料表。

```
aws dynamodb put-item \
 --table-name Music \
 --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
 --region us-east-2
```



6. 稍待幾秒鐘，然後檢查該項目是否已成功複寫到美國東部 (維吉尼亞北部) 與歐洲 (愛爾蘭)。

```
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
```

## 監控全域資料表

### Important

本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用 [全域表版本 2019.11.21 \(目前版本\)](#)，因為它比 2017.11.29 (舊版) 提供更大的彈性、更高的效率以及更少的寫入容量。

若要判斷您使用的是哪個版本，請參閱 [判斷您正在使用的全域資料表版本](#)。若要將現有全域資料表從 2017.11.29 版更新至 2019.11.21 版 (目前)，請參閱 [升級全域資料表](#)。

您可以使 CloudWatch 用 Amazon 監控全域表的行為和效能。Amazon DynamoDB 為全域資料表中的每個複本發佈 ReplicationLatency 和 PendingReplicationCount 指標。

- **ReplicationLatency**：某個更新項目出現在 DynamoDB 串流的複本列表，以及當該項目出現在全域資料表中的另一個複本經過的時間。ReplicationLatency 會以毫秒表示，並會針對每對來源及目標區域配對發送。

ReplicationLatency 在正常操作期間應該很穩定。ReplicationLatency 值上升可能表示某個複本的更新未及時散佈到其他複本資料表。一段時間後，這會造成其他複本資料表落後，因為他們不再一致地收到更新。在此情況下，您應該確認每個複本資料表的讀取容量單位 (RCU) 和寫入容量單位 (WCU) 皆相同。此外，選擇 WCU 設定時應遵循 [全域資料表版本](#) 中的建議。

如果某個 AWS 區域遭到降級，而您在該區域有複本列表，ReplicationLatency 就會增加。這種情況下，您可以暫時將應用程式的讀取和寫入活動重新導向至不同的 AWS 區域。

- **PendingReplicationCount**：寫入複本列表，但尚未寫入全域資料表中另一個複本的項目更新數目。PendingReplicationCount 會以項目數量表示，並會針對每對來源及目標區域配對發送。

正常的操作期間，PendingReplicationCount 的值應非常低。如果 PendingReplicationCount 大幅增加，請調查複本列表的佈建寫入容量設定是否足以滿足您目前的工作負載。

如果某個 AWS 區域遭到降級，而您在該區域有複本列表，PendingReplicationCount 就會增加。這種情況下，您可以暫時將應用程式的讀取和寫入活動重新導向至不同的 AWS 區域。

如需更多詳細資訊，請參閱 [DynamoDB 指標和維度](#)。

## 對全域資料表使用 IAM

### Important

本文件適用於全域資料表 2017.11.29 版 (舊版)，新的全域資料表應避免使用此文件。客戶應盡可能使用 [全域表版本 2019.11.21 \(目前版本\)](#)，因為它比 2017.11.29 (舊版) 提供更大的彈性、更高的效率以及更少的寫入容量。

若要判斷您使用的是哪個版本，請參閱 [判斷您正在使用的全域資料表版本](#)。若要將現有全域資料表從 2017.11.29 版更新至 2019.11.21 版 (目前)，請參閱 [升級全域資料表](#)。

當您第一次建立全域資料表時，Amazon DynamoDB 會自動為您建立 AWS Identity and Access Management (IAM) 服務連結角色。此角色名為 [AWSServiceRoleForDynamoDBReplication](#)，可讓 DynamoDB 代您管理全域資料表的跨區域複寫。請勿刪除此服務連結角色。若您刪除，則所有全域資料表將無法再運作。

如需服務連結角色的詳細資訊，請參閱 IAM 使用者指南中的 [使用服務連結角色](#)。

若要在 DynamoDB 中建立和維護全域資料表，您必須具備 dynamodb:CreateGlobalTable 許可才能存取下列各項：

- 您要新增的複本列表。
- 已經屬於全域資料表的每個現有複本。
- 全域資料表本身。

若要更新 DynamoDB 中全域資料表的設定 (UpdateGlobalTableSettings)，您必須擁有 dynamodb:UpdateGlobalTable、dynamodb:DescribeLimits、application-autoscaling:DeleteScalingPolicy 以及 application-autoscaling:DeregisterScalableTarget 許可。

在更新現有的擴展政策時需要使用 application-autoscaling:DeleteScalingPolicy 和 application-autoscaling:DeregisterScalableTarget 許可。如此一來，全域資料表服務可以在將新政策連接至資料表或次要索引之前移除舊的擴展政策。

如果您使用 IAM 政策來管理對複本列表的存取，則應將相同的政策套用至該全域資料表中的所有其他複本。此實務可協助您在所有複本列表中維持一致的許可模型。

透過在全域資料表中的所有複本上使用相同的 IAM 政策，您也可以避免將非預期的讀取和寫入存取權授予全域資料表資料。例如，假設使用者只能存取全域資料表中的一個複本。如果該使用者可以寫入此複本，則 DynamoDB 會將寫入傳播到所有其他複本列表。實際上，使用者可以 (間接) 寫入全域資料表中的所有其他複本。在所有複本列表上使用一致的 IAM 政策，即可避免此情況。

### 範例：允許動 CreateGlobalTable 作

在將複本新增至全域資料表之前，您必須擁有全域資料表及其每個複本列表的 dynamodb:CreateGlobalTable 許可。

下列 IAM 政策會授予許可，允許在所有資料表上執行 CreateGlobalTable 動作。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": ["dynamodb:CreateGlobalTable"],
 "Resource": "*"
 }
]
}
```

### 範例：允許應用程式 UpdateGlobalTable 自動調度資源：和應用程式自動調度資源：DeleteScalingPolicy 動作 DescribeLimits DeregisterScalableTarget

若要更新 DynamoDB 中全域資料表的設定 (UpdateGlobalTableSettings)，您必須擁有 dynamodb:UpdateGlobalTable、dynamodb:DescribeLimits、application-

autoscaling:DeleteScalingPolicy 以及 application-autoscaling:DeregisterScalableTarget 許可。

下列 IAM 政策會授予許可，允許在所有資料表上執行 UpdateGlobalTableSettings 動作。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "dynamodb:UpdateGlobalTable",
 "dynamodb:DescribeLimits",
 "application-autoscaling:DeleteScalingPolicy",
 "application-autoscaling:DeregisterScalableTarget"
],
 "Resource": "*"
 }
]
}
```

**範例：**僅允許特定區域允許複本的特定全域表格名稱 CreateGlobalTable 執行動作

下列 IAM 政策授予許可，允許 CreateGlobalTable 動作來建立在兩個區域中具有複本且名為 Customers 的全域資料表。

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

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。