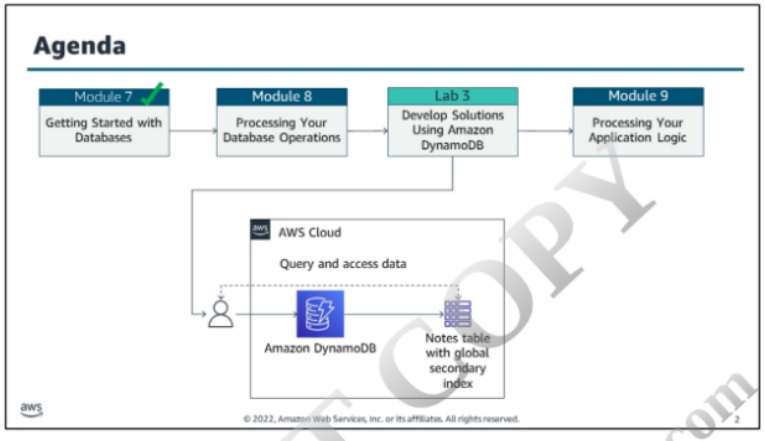
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Module objectives

By the end of this module, you will learn how to:

- Develop programs to interact with Amazon DynamoDB using AWS SDKs
- Perform CRUD operations to access tables, indexes, and data
- Describe developer best practices when accessing DynamoDB
- Review caching options for DynamoDB to improve performance

 3

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Control plane and data plane operations

Control plane operations	Data plane operations
• Create	• Put
• Update	• Update
• Delete	• Get
• Describe	• Delete
• List	

 4

This section examines control and data plane operations used while working with DynamoDB.

Control plane operations
With *control plane* operations, you can create and manage DynamoDB tables. You can use these operations to work with indexes, streams, and other objects that are dependent on tables.

The following are control plane operations:
- `CreateTable`
- `DescribeTable`
- `ListTables`
- `UpdateTable`
- `DeleteTable`

Data plane operations
With data plane operations, you can create, read, update, and delete (also called CRUD) actions on data in a table. You can also read data from a secondary index using some of the data plane operations.

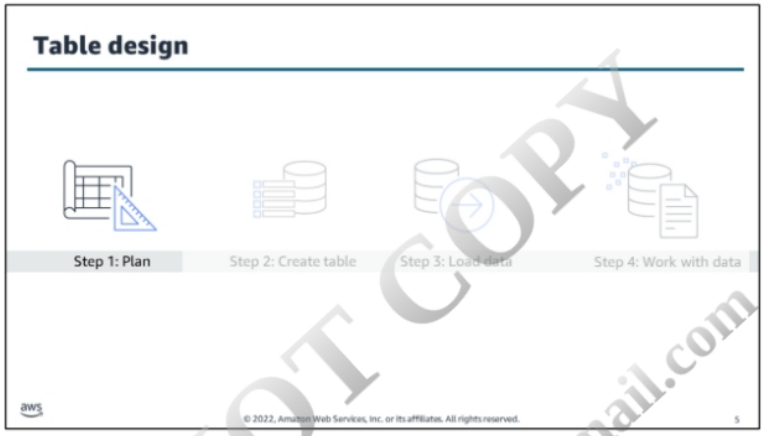
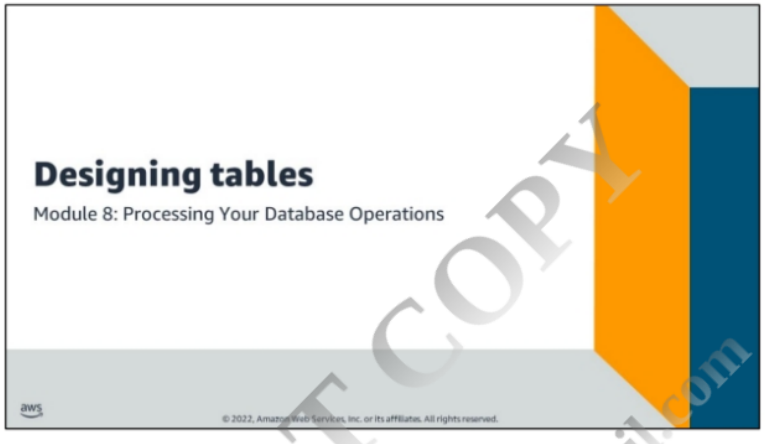


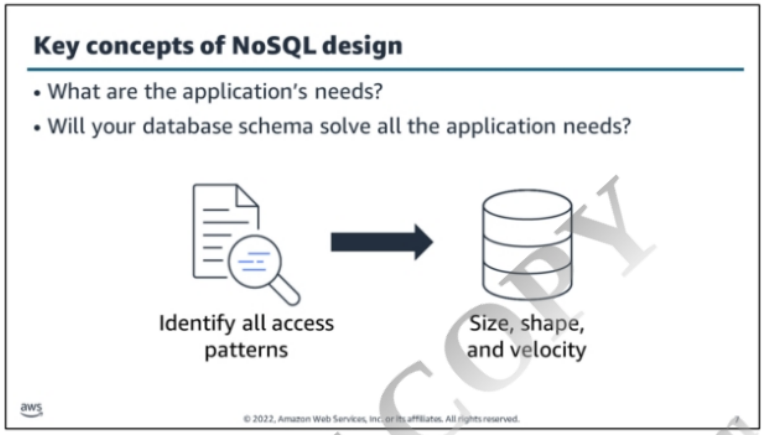
Table design starts with planning. Asking questions that help determine your application needs. Understanding your application needs helps you to design a database that is efficient to load and work with data.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 8: Processing Your Database Operations



DO NOT COPY sameersheik68@gmail.com



You have selected DynamoDB for your application. DynamoDB is a NoSQL database. Developers don't start the process by building a schema. Instead, they start by knowing the questions they need to answer. They must first identify the application's access patterns required by operations they need to support.

Understanding these three properties of your data helps identify the specific query requirements:

- **Data size** – Knowing how much data will be stored and requested at one time will help determine the most effective way to partition the data.
- **Data shape** – A NoSQL database does not reshape data when a query is processed (as an RDBMS system does). Instead, a NoSQL database organizes data so that its shape in the database corresponds with what will be queried. Data shape is a key factor in increasing speed and scalability.
- **Data velocity** – DynamoDB scales by increasing the number of physical partitions that are available to process queries and by efficiently distributing data across those partitions. Knowing in advance what the peak query loads will be might help determine how to partition data to best use read and write capacity.

Developers should not move into building tables until they have **identified all access patterns**. Understanding how your data is accessed is key to your table design.

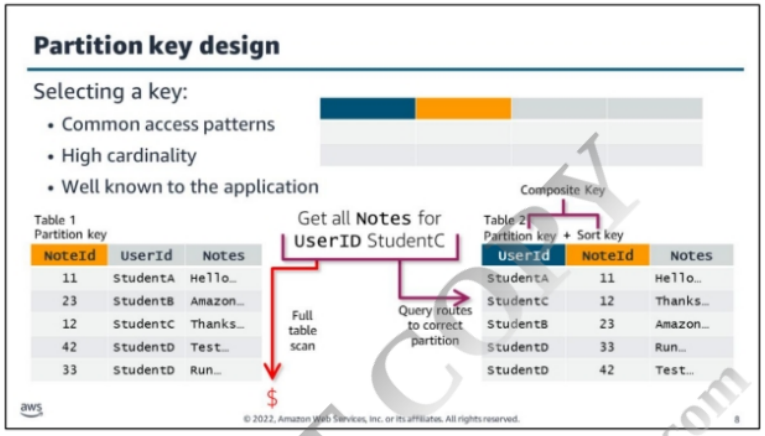
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 8: Processing Your Database Operations

For your application, the following are some of the common queries:

- List all notes for a user.
- Get a specific note for a user.
- List all user notes flagged as favorites.

© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.



Primary keys uniquely identify each item in a DynamoDB table. Using DynamoDB, you can use a partition key or a composite key (a partition key combined with a sort key) as the identifier. The partition key is a simple primary key composed of one attribute (also known as a *hash attribute*). The sort key is an attribute used to sort the order of partition keys in a composite primary key (also known as a *range attribute*).

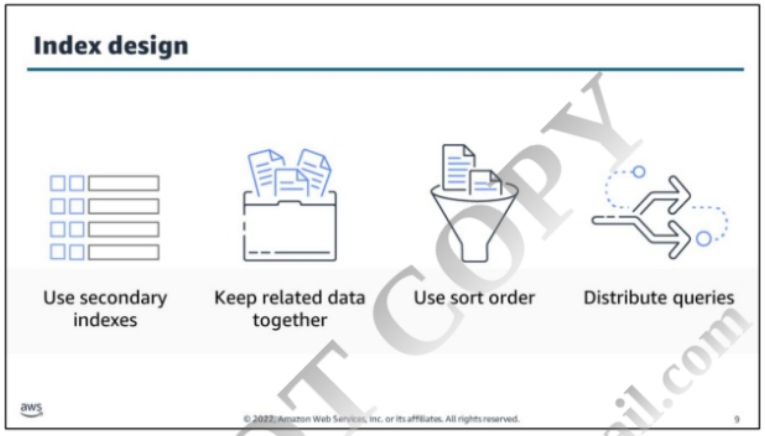
Design partition keys around common access patterns and the partition key's level of uniqueness among items in the table. This is known as *high cardinality*. In addition, a partition key is well known to your application. For example, the two tables represent the same data. However, in this instance, the partition key for Table 1 is `NoteId`. It is unique, and it does not support common access patterns. A common query "Get all `Notes` for `UserID` StudentC" will result in a potentially expensive full table Scan operation.

In Table 2, `UserId` is the partition key and `NoteId` is the sort key. The same query "Get all `Notes` for `UserID` StudentC" will result in the efficient retrieval of the same data.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

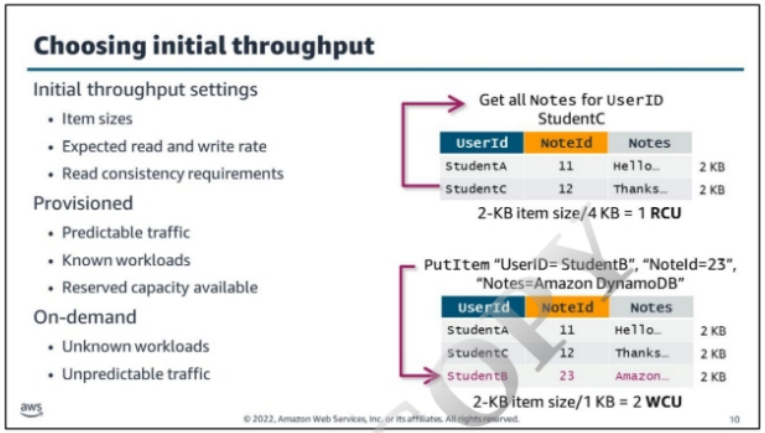
AWS Training and Certification

Module 8: Processing Your Database Operations



Index design revolves around organizing your data to enable efficient access to your application's queries. Organize data by following these principles:

- **Use secondary indexes** – By using secondary indexes, you can enable different queries than your main table can support. These queries are still fast and relatively inexpensive.
- **Keep related data together** – This principal is considered the most important factor to ensure timely responses to your queries.
- **Use sort order** – Related items can be grouped together and queried efficiently if their key design causes them to sort together.
- **Distribute queries** – It is also important that a high volume of queries not be focused on one part of the database, where they can exceed I/O capacity. Instead, design data keys to distribute traffic evenly across partitions as much as possible, avoiding hot spots.



When choosing the initial throughput capacity for your tables, consider these inputs:

- Item sizes
- Expected read and write rate to the table
- Read consistency requirements

In this example, the query that resulted in a single response item is 2 KB in size. This table would be charged 1 read capacity unit (RCU) for the operation. If the same table is used for a write operation, the table is charged 2 write capacity units (WCUs).

Capacity modes

DynamoDB offers two read/write capacity modes to support your application: on demand and provisioned. *On-demand* capacity is a flexible billing option capable of serving thousands of requests per second without capacity planning. *On-demand* mode is a good option for unknown workloads or unpredictable traffic. With provisioned mode, you specify the number of reads and writes per second that you require for your application. If you have predictable traffic, or your application has consistent traffic, provisioned mode is a good option.

Changing capacity modes

When creating your table using the AWS CLI or any AWS SDK, you are required to provision your table's capacity. You can alter the read/write capacity modes after creation through the AWS Management Console, the AWS CLI, or the AWS SDK operation.

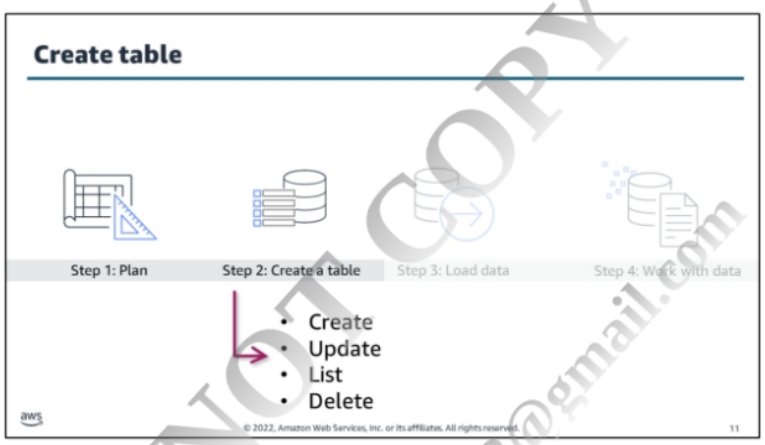
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

NoSQL table design requires that the developer start with a plan. Understand that plans change. Your application may change during development or require adjustments to write and read capacity. With DynamoDB, you can adjust to changing needs dynamically.

Note: You can switch between read and write capacity modes once every 24 hours.

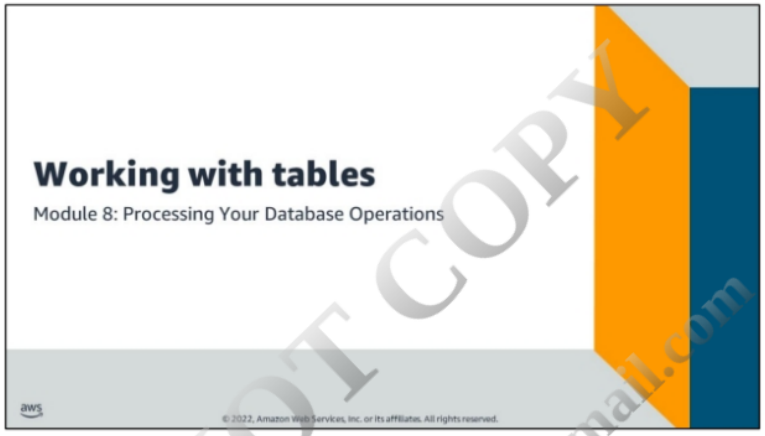
Take advantage of burst and adaptive capacity to assist with fine-tuning your application's capacity.

© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.

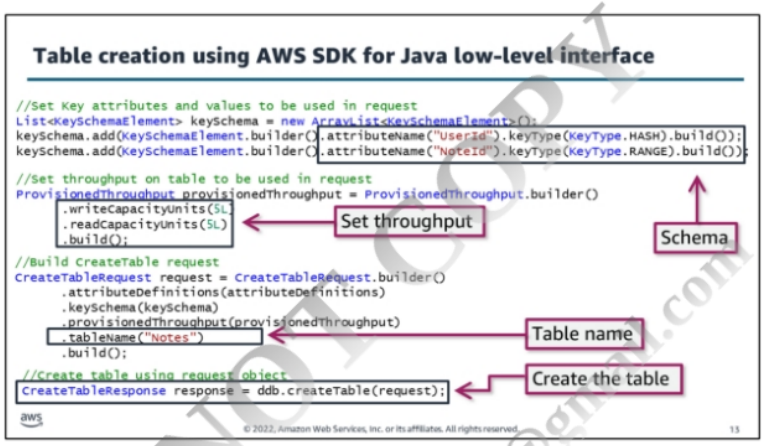


Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 8: Processing Your Database Operations



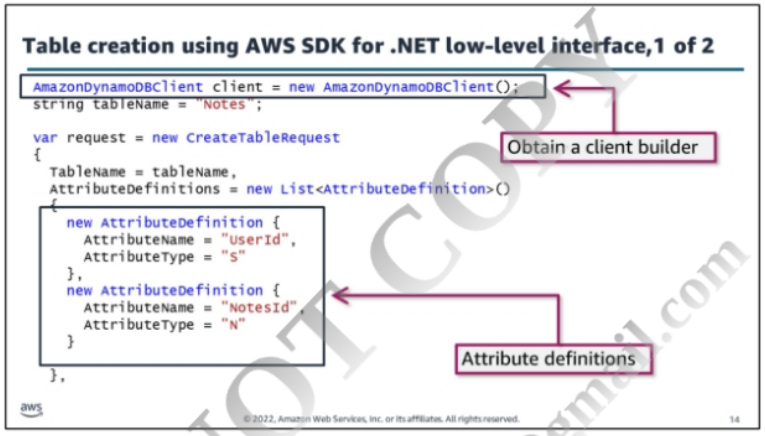
DO NOT COPY
sameersheik68@gmail.com

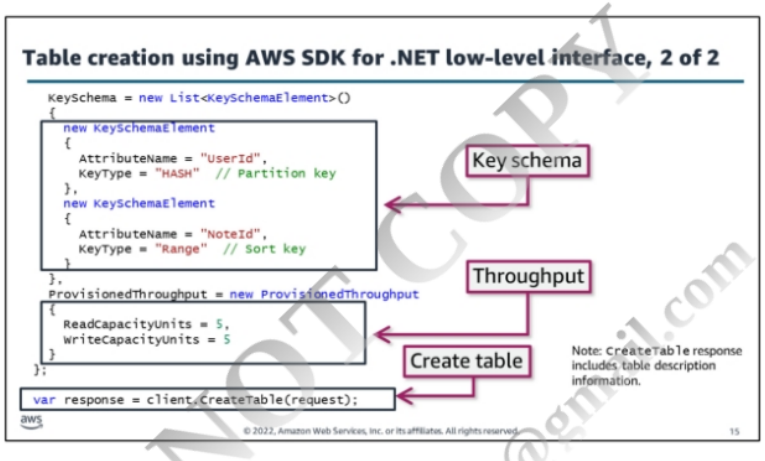


This portion of a script that defines the DynamoDB table's primary and sort key attributes and sets the attribute type. In this example, the primarykey (`UserId`) is set as a String (S) and sortKey (`NoteId`) a number (N).

Next, the script defines the Notes table schema, primaryKey (`UserId`) as the partition key HASH, and sortKey (`NoteId`) as the RANGE.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

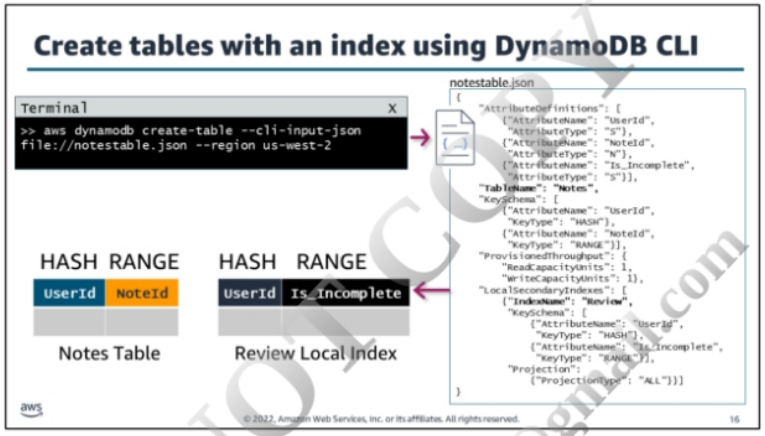




You must wait until DynamoDB creates the table and sets its status to ACTIVE. The *CreateTable* response includes the `TableDescription` property that provides the necessary table information.

You can also call the `DescribeTable` method of the client to get table information at any time.

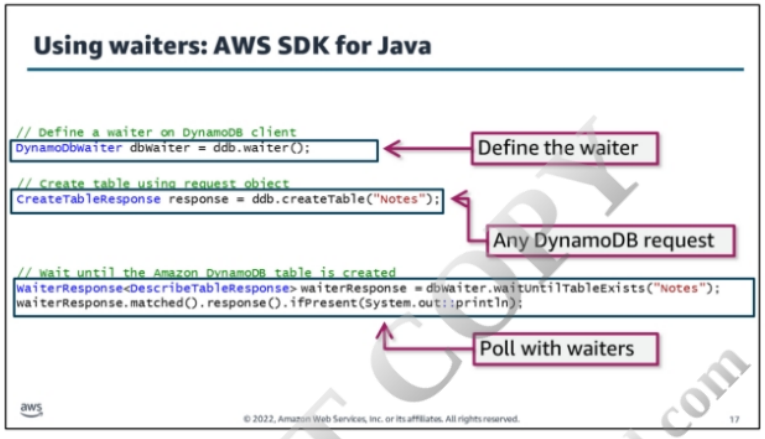
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



This example illustrates the DynamoDB `create-table` CLI, which uses the specified attributes and key schema to create a table named Notes. The command defines the attributes `UserId`, and `NoteId` for the table. Then, it assigns the partition key to the attribute `UserId (HASH)` and sort key `NoteId (RANGE)`.

Attribute types are set with the following information:
S – The attribute is of type String.
N – The attribute is of type Number.
B – The attribute is of type Binary.

Note: By design, developers do not need to declare nonkey attributes to a DynamoDB table. DynamoDB is schema-less except for the `--key-schema`.

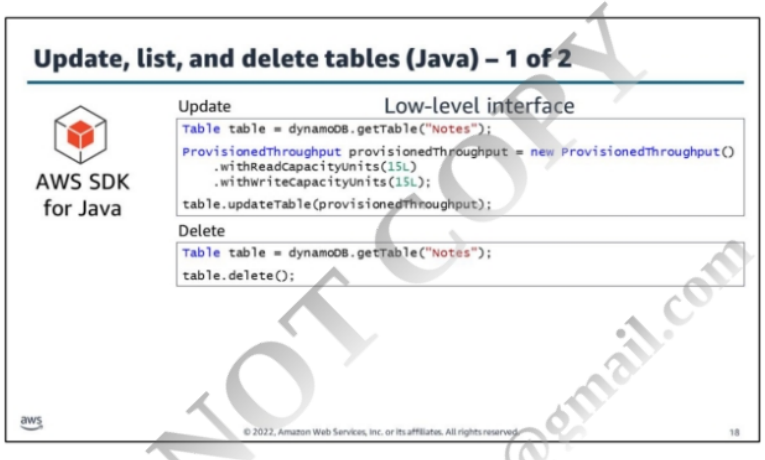


The AWS SDK for Java provides a waiters feature that you can use to wait for a resource to transition to a desired state. Without a waiter written into your code, polling for the desired state requires writing your own code that repeatedly checks the status. Though using a waiter in the DynamoDB creation process is not required, waiters make it more efficient to abstract out the polling logic into a simple API call.

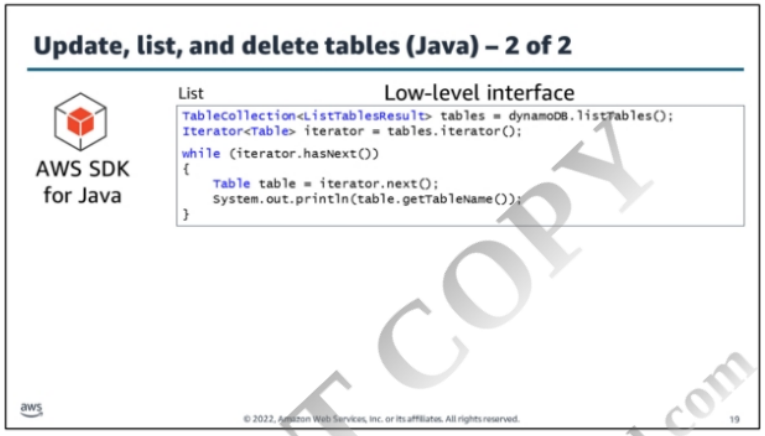
This section of code does the following:

1. Defines the waiter
2. Creates the DynamoDB table using the information from the `CreateTableRequest` object
3. Prints the status through `system.out`.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

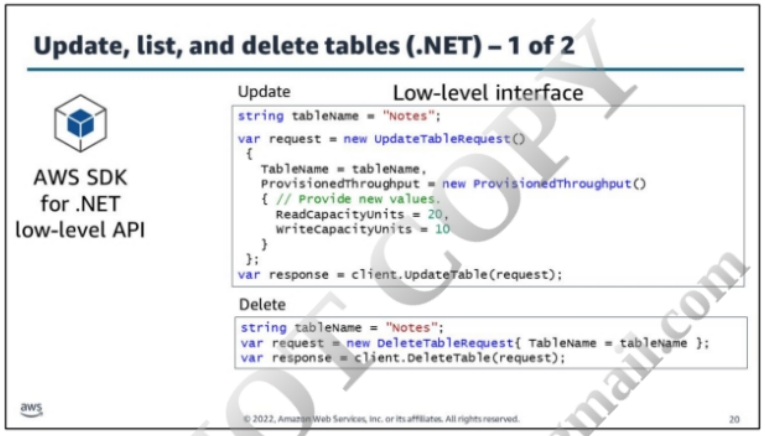


The `ListTables` operation requires no parameters.



The `ListTables` operation requires no parameters.

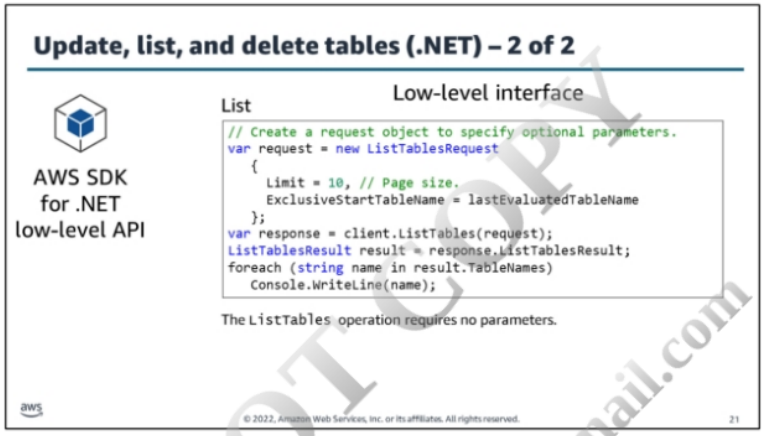
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



You can increase `ReadCapacityUnits` or `WriteCapacityUnits` as often as necessary, using the AWS Management Console or the `UpdateTable` operation. In a single call, you can increase the provisioned throughput for a table for any global secondary indexes on that table or for any combination of these. The new settings do not take effect until the `UpdateTable` operation is complete.

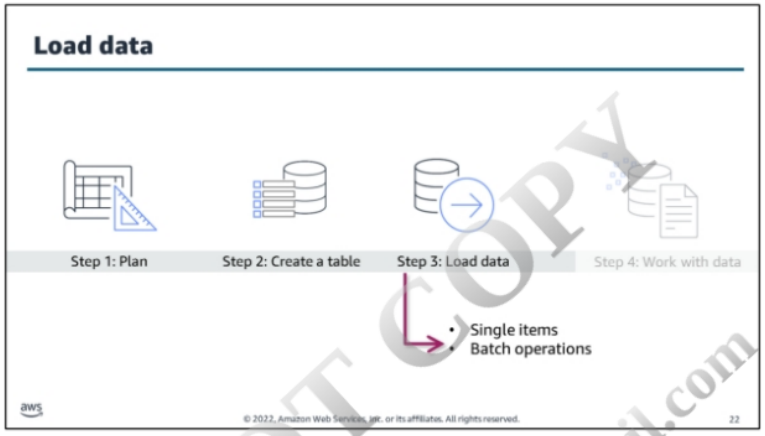
For every table and global secondary index in an `UpdateTable` operation, you can decrease `ReadCapacityUnits`, `WriteCapacityUnits`, or both. The new settings don't take effect until the `UpdateTable` operation is complete. A decrease is allowed up to four times, any time per day.

For more information on limits, see "Service, Account, and Table Quotas in Amazon DynamoDB" in the *Amazon DynamoDB Developer Guide* (https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ServiceQuotas.html).



The `ListTables` operation requires no parameters. However, you can specify optional parameters. For example, you can set the `Limit` parameter.

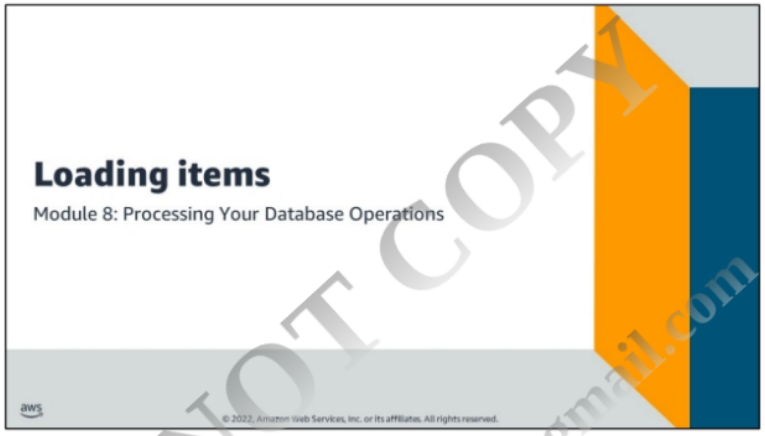
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Use data plane operations to create, read, update, and delete (CRUD) actions on data in a table.

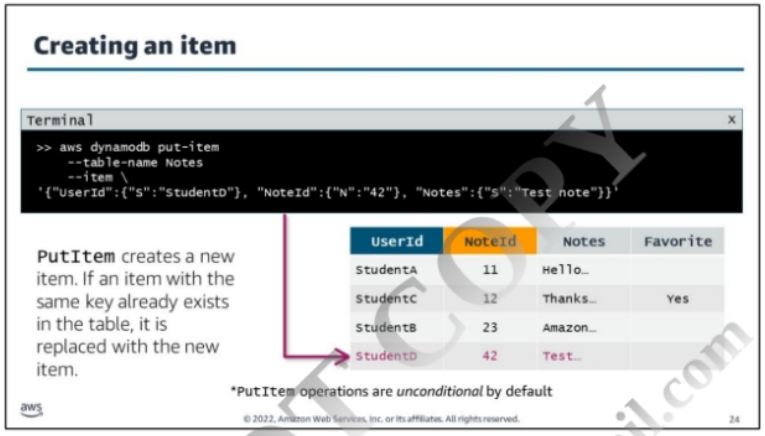
To create data, you can use the following APIs:

- `PutItem`
- `BatchWriteItem`



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 8: Processing Your Database Operations



The `PutItem` operation creates a new item or replaces an early item with a new item. If an item with the same key already exists in the table, it is replaced with the new item.

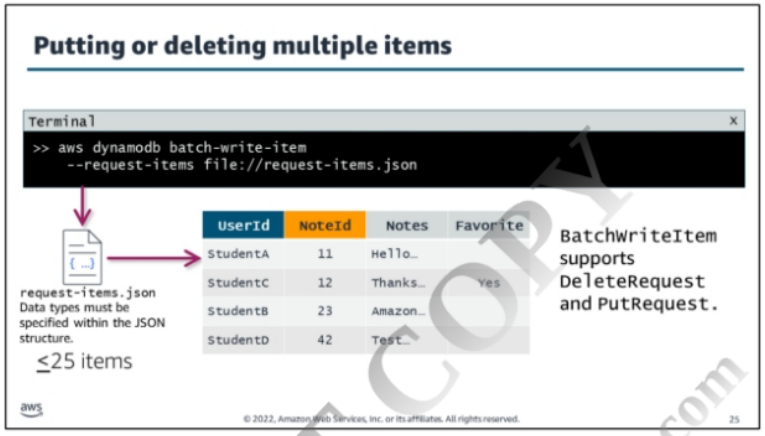
You can perform a conditional PUT operation:
- Add a new item if one with the specified primary key doesn't exist.
- Replace an existing item if it has certain attribute values.

You can return the item's attribute values in the same operation by using the `ReturnValues` parameter.

When you add an item, the primary key attributes are the only required attributes. Attribute values cannot be null.

To prevent a new item from replacing an existing item, use a conditional expression that contains the `attribute_not_exists` function with the name of the attribute being used as the partition key for the table. Because every record must contain that attribute, the `attribute_not_exists` function will succeed only if no matching item exists.

The example uses a Java method to load a new item into the Notes table.
For more information, see "PutItem" in the *Amazon DynamoDB API Reference* (https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html).



The `BatchWriteItem` operation can contain up to 25 individual `PutItem` and `DeleteItem` requests and can write up to 16 MB of data. (The maximum size of an individual item is 400 KB.) In addition, a `BatchWriteItem` operation can put or delete items in multiple tables.

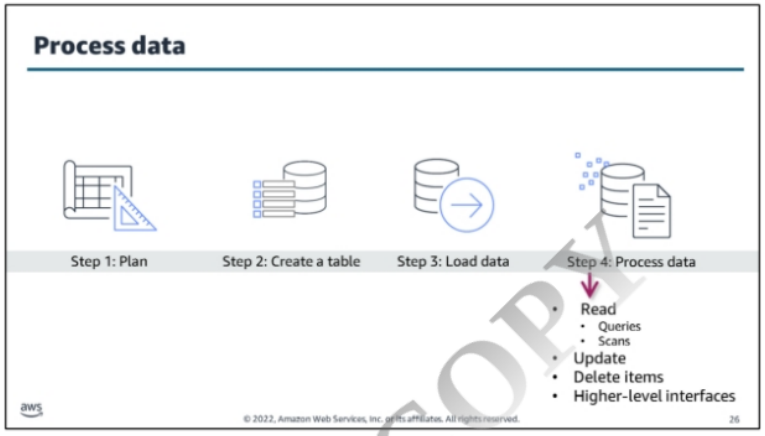
Requests must include a map of one or more table names and, for each table, a list of operations to be performed (`DeleteRequest` or `PutRequest`).
Batch operations read and write items in parallel to minimize response latencies.

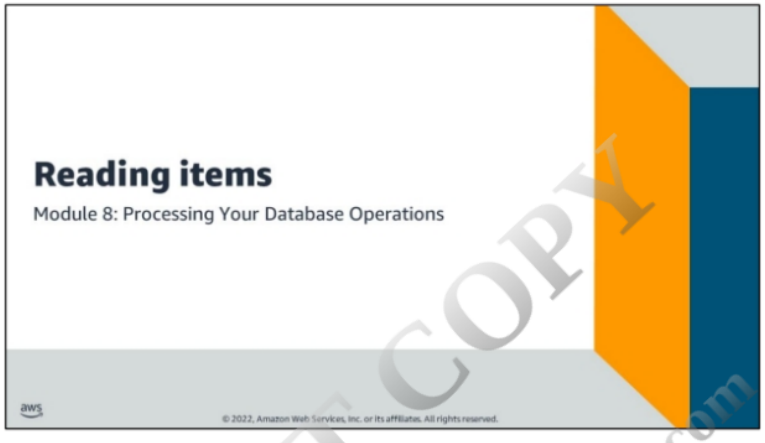
- **`BatchGetItem`** – Read up to 100 items from one or more tables up to 16 MB of data.
- **`BatchWriteItem`** – Create or delete up to 25 items in one or more tables up to 16 MB of data.

For more information on BatchWrite item JSON structure, see "BatchWriteItem" in the *Amazon DynamoDB API Reference* (https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_BatchWriteItem.html).

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

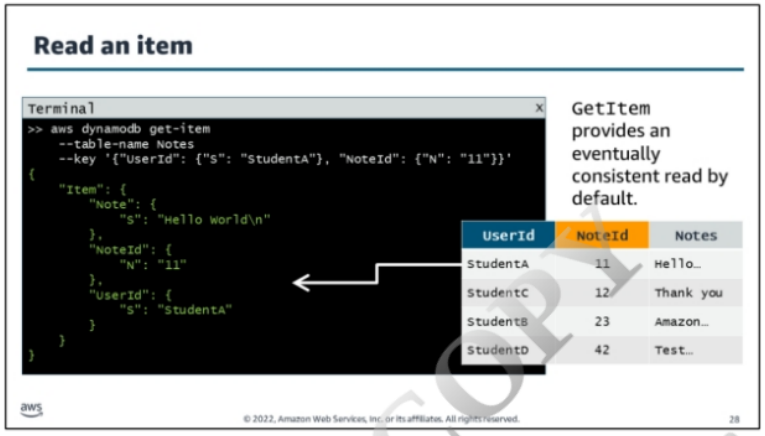
AWS Training and Certification Module 8: Processing Your Database Operations





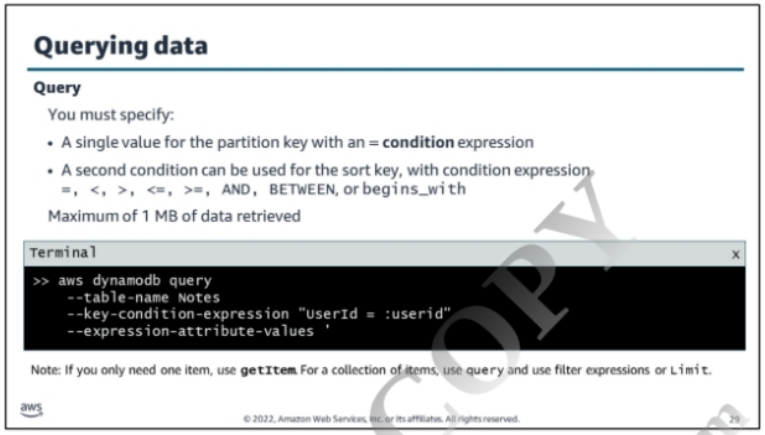
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 8: Processing Your Database Operations



To read an item from a DynamoDB table, use the `GetItem` operation. The `GetItem` operation returns a set of attributes for the item with the given primary key. If there is no matching item, `GetItem` does not return any data, and no `Item` element will be in the response.

`GetItem` provides an eventually consistent read by default. If your application requires a strongly consistent read, set `ConsistentRead` to true.

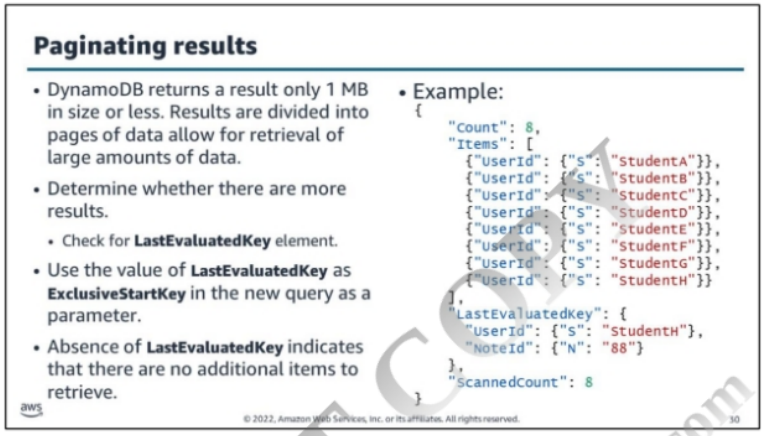


The `Query` operation in Amazon DynamoDB finds items based on primary key values. To specify the search criteria, you use a key condition expression by specifying the partition key name and value as an equality condition.

You can provide a second condition for the sort key (if present). The sort key condition must use one of the following comparison operators:
a = b: True if the attribute **a** is equal to the value **b**
a < b: True if **a** is less than b
a <= b: True if **a** is less than or equal to **b**
a > b: True if **a** is greater than **b**
a >= b: True if **a** is greater than or equal to **b**
begins_with (a, substr): True if the value of the **a** attribute begins with a particular substring.

Using a filter expression further refines a Query result by determining which query results should be returned. You can also limit the number of results that the query reads. You can combine filter expressions and limit the results. Combining the two results in a refined dataset without reading more items than needed.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

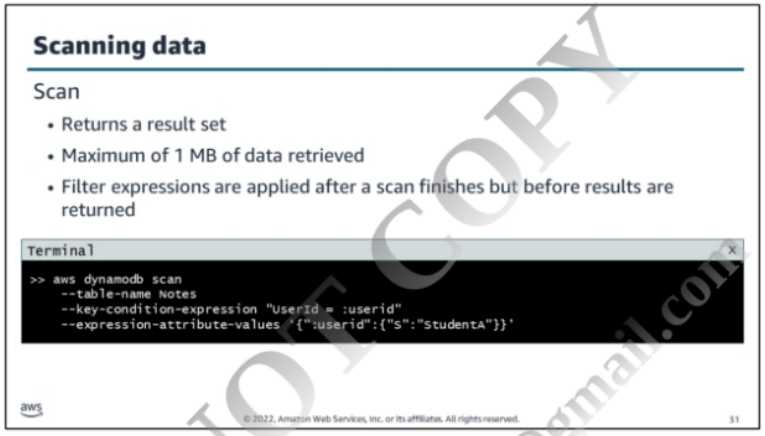


DynamoDB *paginates* the results from Query operations. By default, DynamoDB returns a result only 1 MB in size or less. Results are divided into pages. An application can process the first page of results, then the second page, and so on.

To determine whether your query results in paginated results, construct a query and check for `LastEvaluatedKey` in the results. If the result contains a `LastEvaluatedKey` element and it's non-null, construct a new `Query` request with the same parameters as the previous one. However, this time, take the `LastEvaluatedKey` value from your original query and use it as the `ExclusiveStartKey` parameter in the new `Query` request.

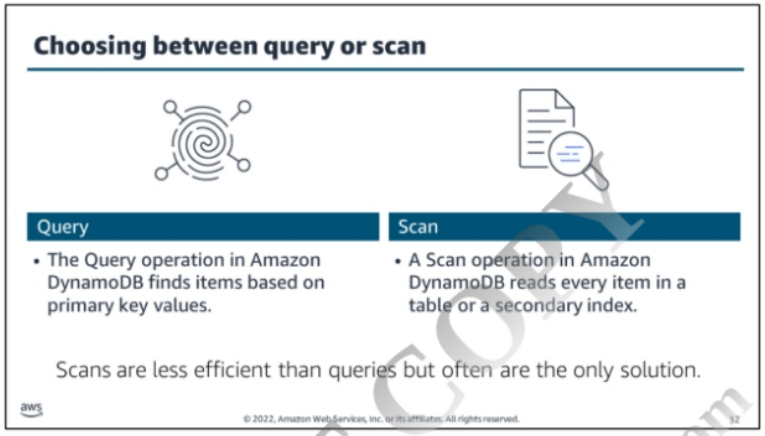
The absence of `LastEvaluatedKey` indicates that there are no more items to retrieve.

For more information, see "Paginating Table Query Results" in the *Amazon DynamoDB Developer Guide* (https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.Pagination.html).



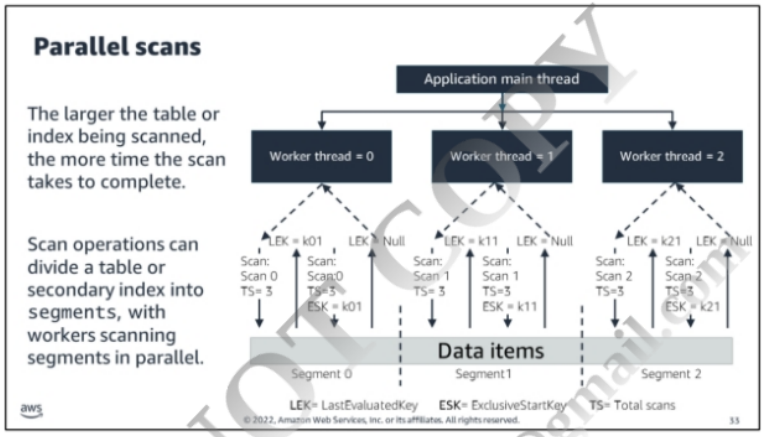
A Scan operation in Amazon DynamoDB reads every item in a table or a secondary index. A filter expression is applied after a Scan finishes but before the results are returned. Therefore, a Scan consumes the same amount of read capacity, regardless of whether a filter expression is present.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



By understanding the differences between queries and scans, developers can make efficient calls to their database. In general, scans are less efficient than queries. Avoid using a Scan operation on large tables with a filter that removes many results. As the table grows, the scan will become slower using more capacity.

Sometimes scans are the only option. With parallel table scans, you can distribute the process to keep a Scan operation from consuming an excess of resources.



The larger the table or index being scanned, the more time the scan takes to complete. In addition, a sequential scan might not always be able to use the provisioned read throughput capacity fully. Even though DynamoDB distributes a large table's data across multiple physical partitions, a Scan operation can read only one partition at a time. Consequently, the throughput of a scan is constrained by the maximum throughput of a single partition.

In this diagram, the application spawns three threads, and assigns each thread a number. (Segments are zero-based, so the first number is always 0.) Each thread issues a `Scan` request, setting Segment to its designated number and setting `TotalSegments` to 3. Each thread scans its designated segment, retrieving data 1 MB at a time, and returns the data to the application's main thread.

Note: A parallel scan with a large number of workers can easily consume all of the provisioned throughput for the table or index being scanned. If the table or index is also incurring heavy read or write activity from other applications, avoid such scans.

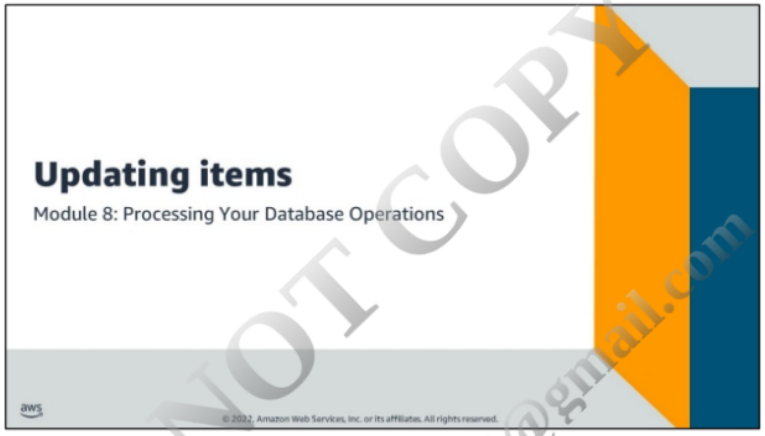
To control the amount of data returned per request, use the `Limit` parameter. This helps prevent situations in which one worker consumes all the provisioned throughput at the expense of all other workers.

For more information, see "Parallel Scan" in the *Amazon DynamoDB Developer Guide* (https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Scan.html#Scan.ParallelScan).

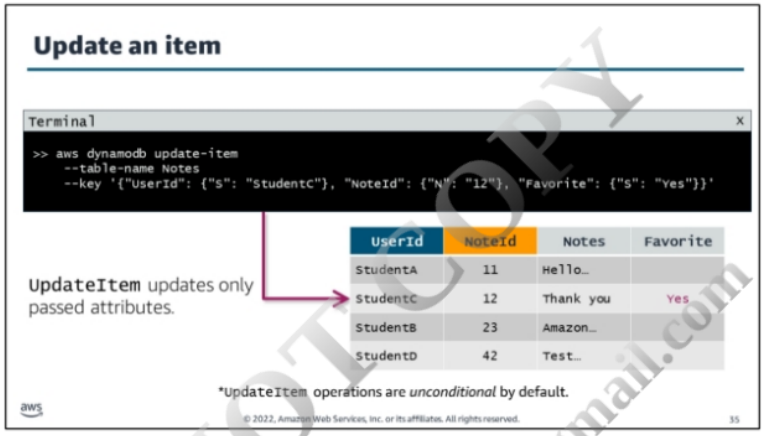
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification

Module 8: Processing Your Database Operations



© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.



The `updateItem` method of a table object can update existing attribute values, add new attributes, or delete attributes from an existing item.

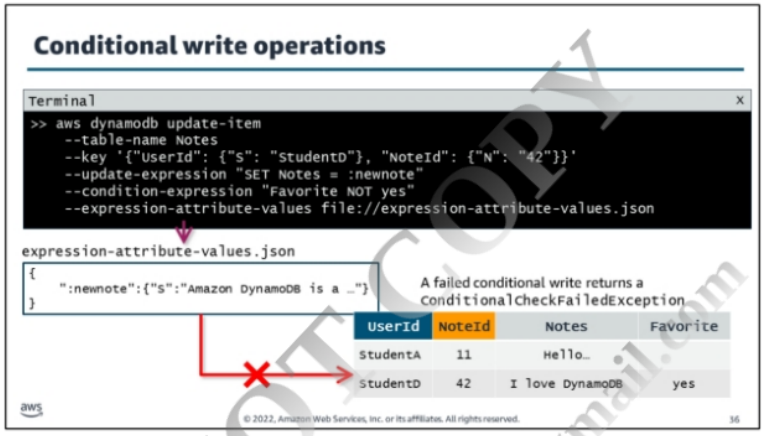
If an item with the specified key does not exist, `UpdateItem` creates a new item. Otherwise, it modifies an existing item's attributes. You can update multiple attribute values.

DynamoDB considers the size of the item as it appears before and after the update. The provisioned throughput consumed reflects the larger of these item sizes. Even if you update only a subset of the item attributes, `UpdateItem` will consume the full amount of provisioned throughput. The provisioned throughput can be the larger of the *before* and *after* item sizes.

Update conditionally, as DynamoDB returns an exception if a match is not correct.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 8: Processing Your Database Operations



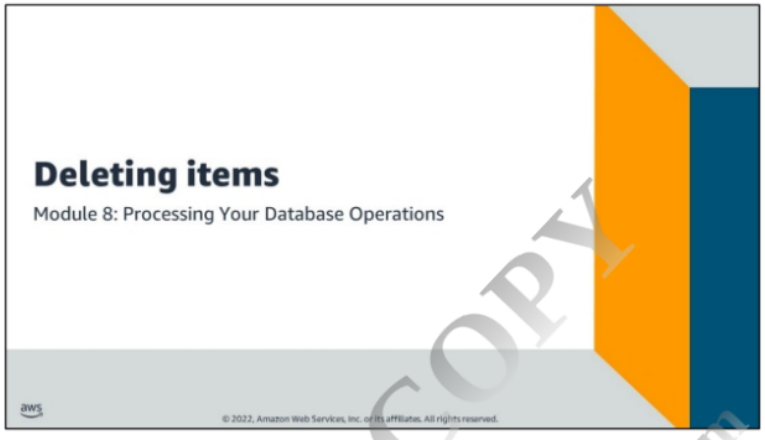
By default, the DynamoDB write operations (`PutItem`, `UpdateItem`, `DeleteItem`) are *unconditional*. Conditional writes succeed only if item attributes meet one or more expected conditions.

This `Conditional Write` operation shows how a developer could prevent a note update if the note has been flagged as favorite. The `UpdateItem` conditional expression checks whether the `Favorite` attribute is not set to yes. Because the Favorite attribute for the item does not meet the expected conditions, the operation fails.

Note: If a `ConditionExpression` evaluates to false during a conditional write, DynamoDB still consumes write capacity from the table.

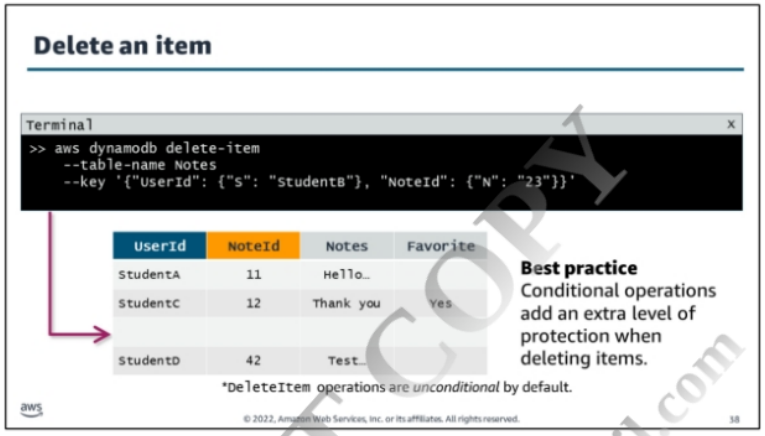
- If the item does not currently exist in the table, DynamoDB consumes one write capacity unit.
- If the item does exist, the number of write capacity units consumed depends on the size of the item. For example, a failed conditional write of a 1-KB item would consume one write capacity unit. If the item is twice that size, the failed conditional write consumes two write capacity units.

A failed conditional write returns a `ConditionalCheckFailedException`. When this occurs, you don't receive any information in the response about the write capacity that was consumed. However, you can view the `ConsumedWriteCapacityUnits` metric for the table in Amazon CloudWatch.



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 8: Processing Your Database Operations



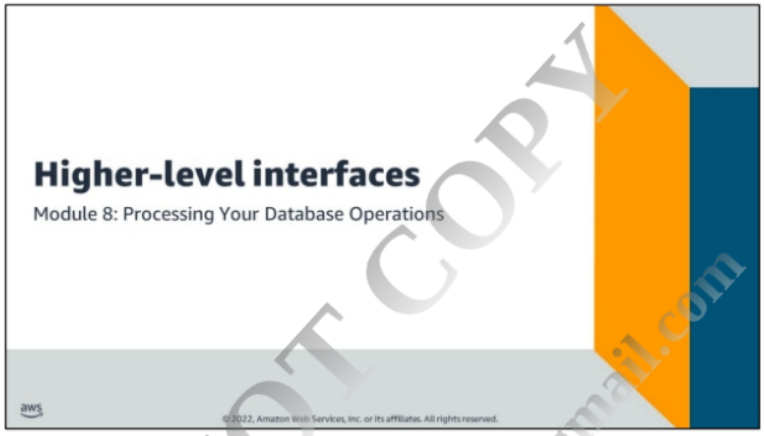
Delete Item

Deletes a single item in a table by primary key. You can perform a conditional delete operation that deletes the item if it exists, or if it has an expected attribute value.

In addition to deleting an item, you can also return the item's attribute values in the same operation, using the ReturnValues parameter.

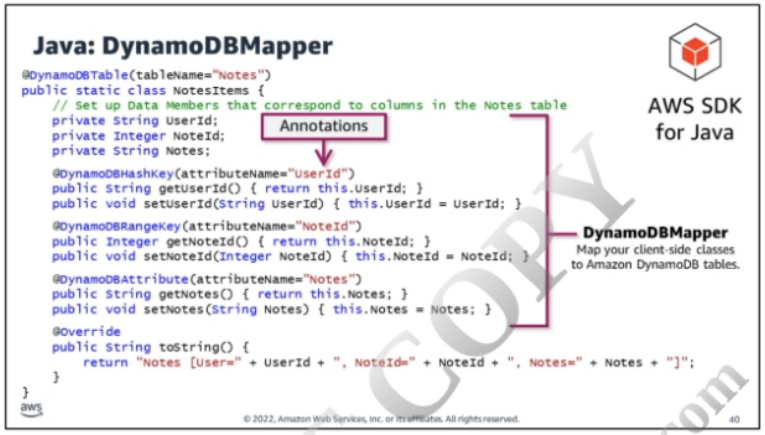
Unless you specify conditions, the DeleteItem is an idempotent operation; running it multiple times on the same item or attribute does *not* result in an error response.

Conditional deletes are useful for deleting items only if specific conditions are met. If those conditions are met, DynamoDB performs the delete. Otherwise, the item is not deleted.



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 8: Processing Your Database Operations

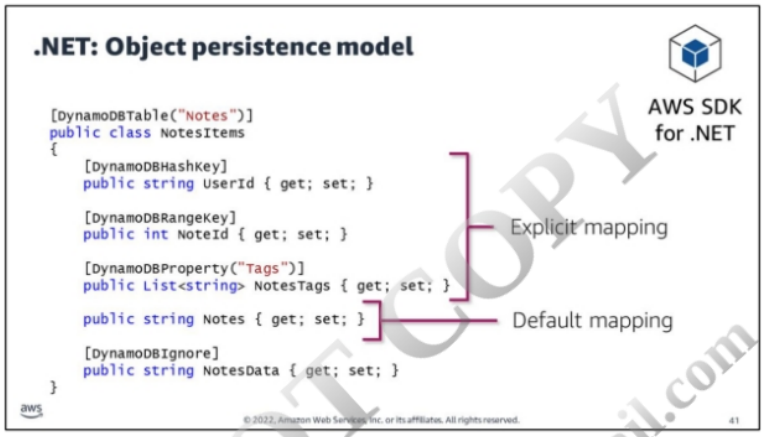


In this code example, the `@DynamoDBTable` annotation maps the `NotesItems` class to the Notes table. You can store individual class instances as items in the table. In the class definition, the `@DynamoDBHashKey` annotation maps the Id property to the primary key.

By default, the class properties map to the attributes of the same name in the table. The properties `UserId`, `NoteId`, and `Notes` map to the attributes of the same name in the table.

The `@DynamoDBAttribute` annotation is optional when the name of the DynamoDB attribute matches the name of the property declared in the class. When they differ, use this annotation with the `attributeName()` parameter to specify which DynamoDB attribute this property corresponds to.

Note: You cannot use the DynamoDBMapper class to create, update, or delete tables. To perform those tasks, use the low-level SDK for Java interface instead. For more information, see "Working with DynamoDB Tables in Java" in the *Amazon DynamoDB Developer Guide* (https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/JavaDocumentAPIWorkingWithTables.html).



The AWS SDK for .NET provides an object persistence model that you can use to map your client-side classes to Amazon DynamoDB tables. Each object instance then maps to an item in the corresponding tables.

To save your client-side objects to the tables, the object persistence model provides the DynamoDBContext class, an entry point to DynamoDB. This class provides you a connection to DynamoDB. You can then access tables, perform various CRUD operations, and run queries.

In this C# code example, you have a `NotesItems` class with `UserId`, `NoteId`, tags, and `NotesData` properties. You can map the `NotesItems` class to the `Notes` table by adding the attributes defined by the object persistence model.

The object persistence model supports both the explicit and default mapping between class properties and table attributes.

Explicit mapping
To map a property to a primary key, you must use the `DynamoDBHashKey` and `DynamoDBRangeKey` object persistence model attributes. For the nonprimary key attributes, if a property name in your class and the corresponding table attribute to which you want to map it are not the same, you must define the mapping. You can do this explicitly adding the `DynamoDBProperty` attribute.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 8: Processing Your Database Operations

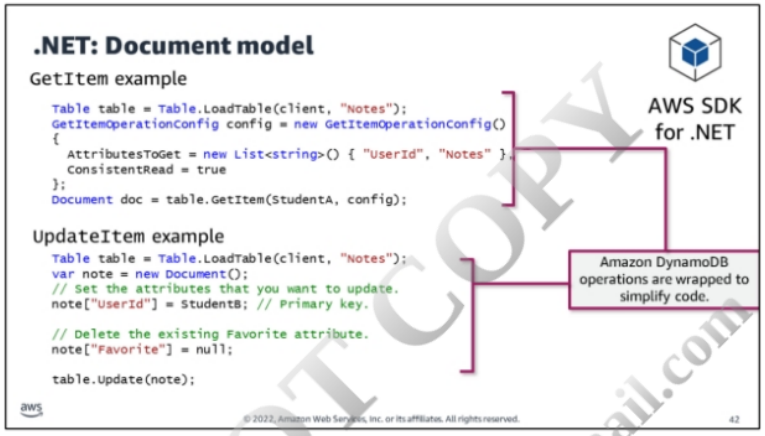
The `UserId` property maps to the primary key with the same name, and the Notes property maps to the `Notes` attribute in the Notes table.

Default mapping
By default, the object persistence model maps the class properties to the attributes with the same name in the table. The properties `NoteId` and `Notes` map to the attributes with the same name in the Notes table.

Identify class properties you do not want to map by adding the `DynamoDBIgnore` attribute.

The object persistence model provides a set of attributes to map client-side classes to tables, and properties/fields to table attributes.

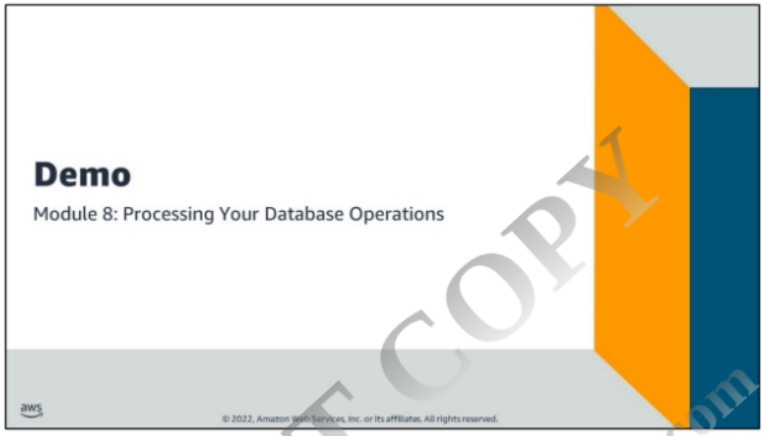
© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.

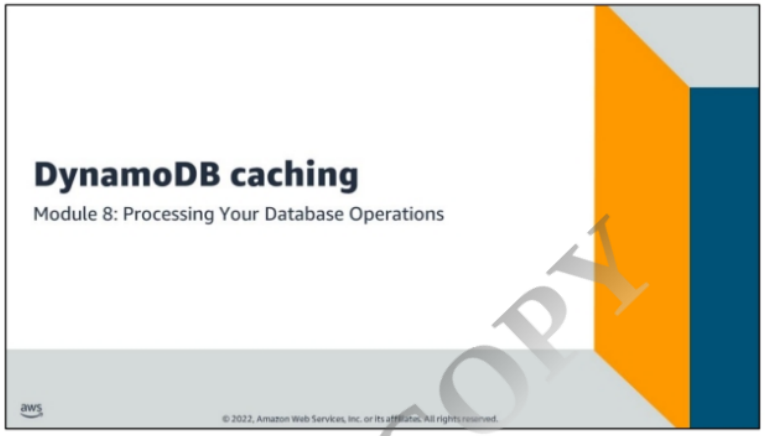


The AWS SDK for .NET provides document model classes that wrap some of the low-level Amazon DynamoDB operations, further simplifying your coding. In the document model, the primary classes are Table and Document. The Table class provides CRUD data operation methods such as `PutItem`, `GetItem`, and `DeleteItem`. It also provides the Query and the Scan methods. The Document class represents a single item in a table.

You cannot use the document model classes to create, update, and delete tables. However, the document model does support most common data operations.

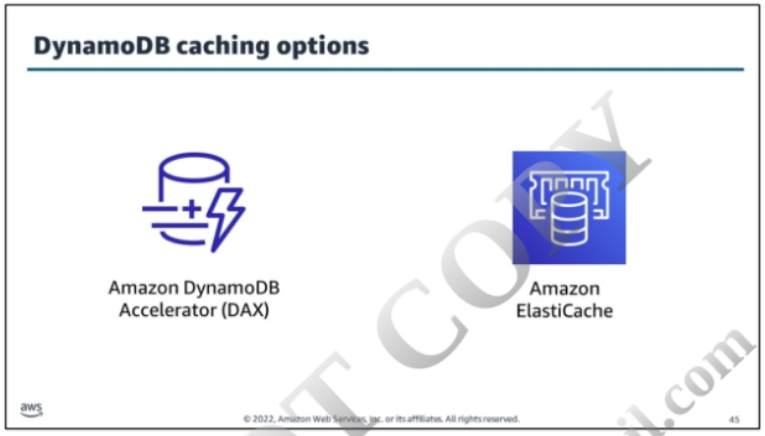
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.





Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

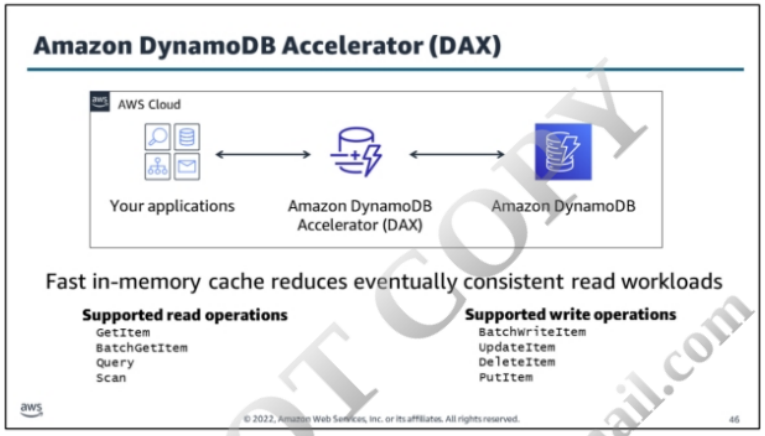
AWS Training and Certification Module 8: Processing Your Database Operations



Amazon DynamoDB is designed for scale and performance. In most cases, the DynamoDB response times can be measured in single-digit milliseconds. However, certain use cases require response times in microseconds. For these use cases, Amazon DynamoDB Accelerator (DAX) delivers fast response times for accessing eventually consistent data.

Amazon ElastiCache is a web service that you can use to deploy and run Memcached or Redis protocol-compliant server nodes in the cloud. Amazon ElastiCache improves the performance of web applications. You can retrieve information from a fast, managed, in-memory system instead of relying entirely on slower disk-based databases.

This lesson focuses on DAX.



DAX is a DynamoDB-compatible caching service that you can use to benefit from fast in-memory performance for demanding applications. DAX addresses three core scenarios:

- As an in-memory cache, DAX reduces the response times of eventually consistent read workloads by an order of magnitude, from single-digit milliseconds to microseconds.
- DAX reduces operational and application complexity by providing a managed service that is API-compatible with Amazon DynamoDB. DAX requires only minimal functional changes to use with an existing application.
- For read-heavy or bursty workloads, DAX provides increased throughput and potential operational cost savings by reducing the need to over-provision read capacity units. This is especially beneficial for applications that require repeated reads for individual keys.

DAX can respond to the following Read API calls:

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

DAX API write operations are considered "write-through":

- `BatchWriteItem`
- `UpdateItem`
- `DeleteItem`
- `PutItem`

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 8: Processing Your Database Operations

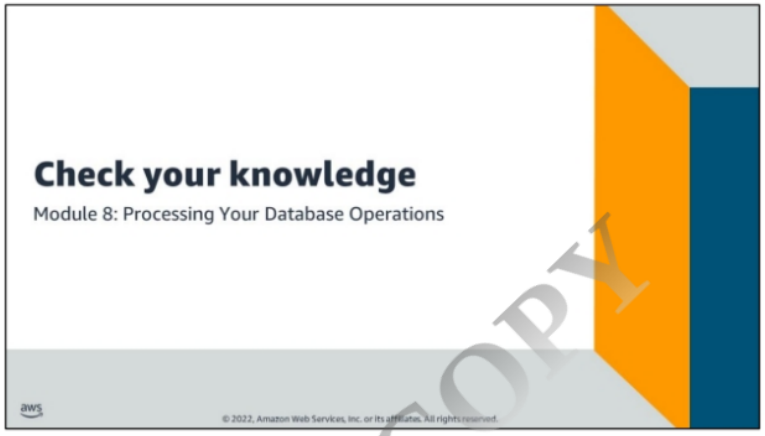
Use cases for DAX:

- Applications that require the fastest possible response time for reads
- Applications that read a small number of items more frequently than others
- Applications that are read-intensive but are also cost-sensitive
- Applications that require repeated reads against a large set of data

DAX is *not* ideal for the following use cases:

- Applications that require strongly consistent reads
- Applications that do not require microsecond response times for reads
- Applications that are write-intensive or that do not perform much read activity
- Applications that are already using a different caching solution with DynamoDB

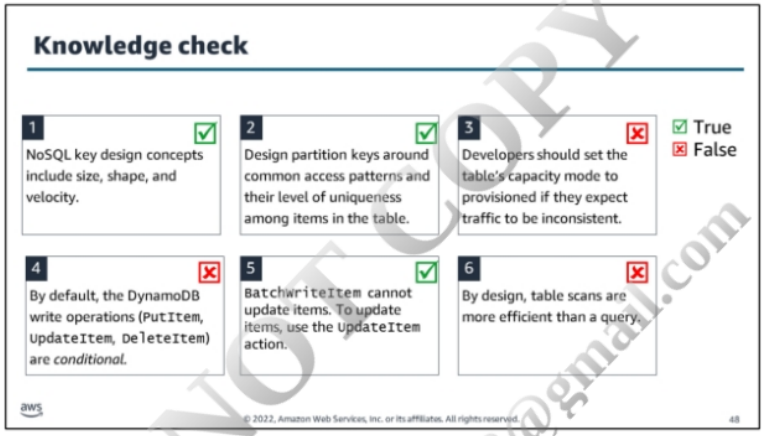
© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

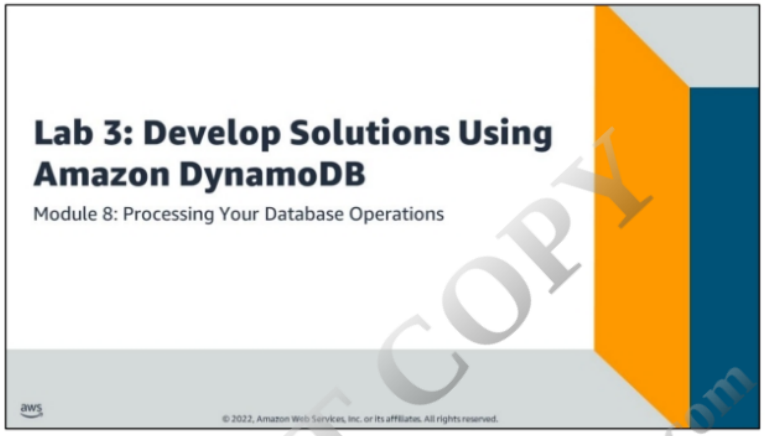
AWS Training and Certification

Module 8: Processing Your Database Operations

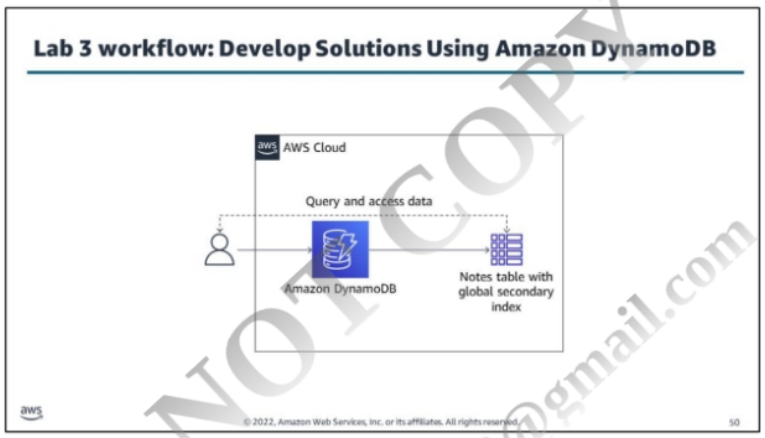


1. (True)
2. (True)
3. (False) On-demand mode is best suited for inconsistent traffic.
4. (False) By default, the DynamoDB write operations (PutItem, UpdateItem, DeleteItem) are *unconditional.*
5. (True)
6. (False) Table scans are less efficient than queries and should be used sparingly.

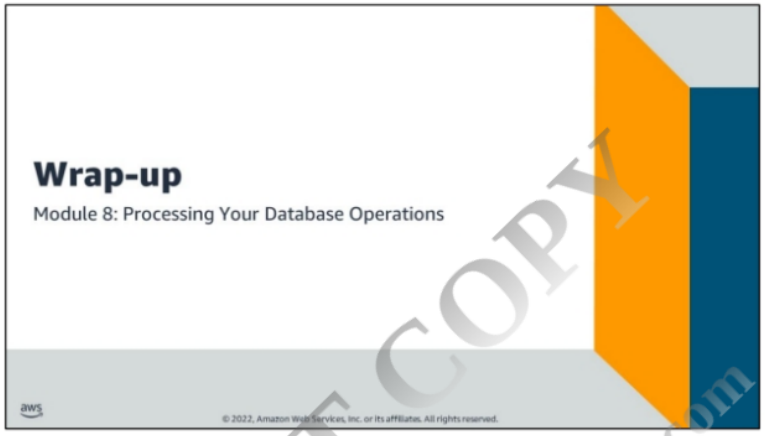
© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



DO NOT COPY
sameersheik68@gmail.com



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification
Module 8: Processing Your Database Operations