nextwork.org

Query Data with DynamoDB

Abdulrahman Abdulkadir

Abdulrahman Abdulkadir
NextWork Student

Introducing Today's Project!

What is Amazon DynamoDB?

Amazon DynamoDB is a fully managed NoSQL database service offered by AWS that provides fast, reliable, and scalable performance. It is designed to handle large amounts of data and traffic without requiring manual intervention to manage infrastructure. DynamoDB automatically scales up or down based on the workload, making it ideal for applications that demand consistent low-latency responses, such as real-time analytics, mobile apps, and gaming platforms. It supports both key-value and document-based data models and ensures data security through built-in encryption, fine-grained access control, and automatic backups. Its support for transactions, global tables, and event-driven triggers also makes it highly suitable for modern, distributed applications.

How I used Amazon DynamoDB in this project

In today’s project, I used Amazon DynamoDB to store and retrieve structured data for a simulated discussion forum. I created and worked with multiple tables such as `Forum`, `Post`, and `Comment`, each with specific partition and sort keys. I used the AWS CLI to run `get-item` and `query` commands to retrieve data, explored the use of projection expressions and consistent reads, and performed a transaction to ensure atomic updates across multiple tables. This helped demonstrate how DynamoDB can manage interconnected data reliably and efficiently in real-world applications.

Abdulrahman Abdulkadir
NextWork Student

One thing I didn't expect in this project was...

One thing I didn't expect in this project was how important data modeling is in DynamoDB. Unlike traditional relational databases, DynamoDB requires you to think carefully in advance about your access patterns and design your partition and sort keys around them. I realized that if the data model isn't planned properly, even simple queries—like finding all comments by a user—can become difficult or inefficient.

This project took me...

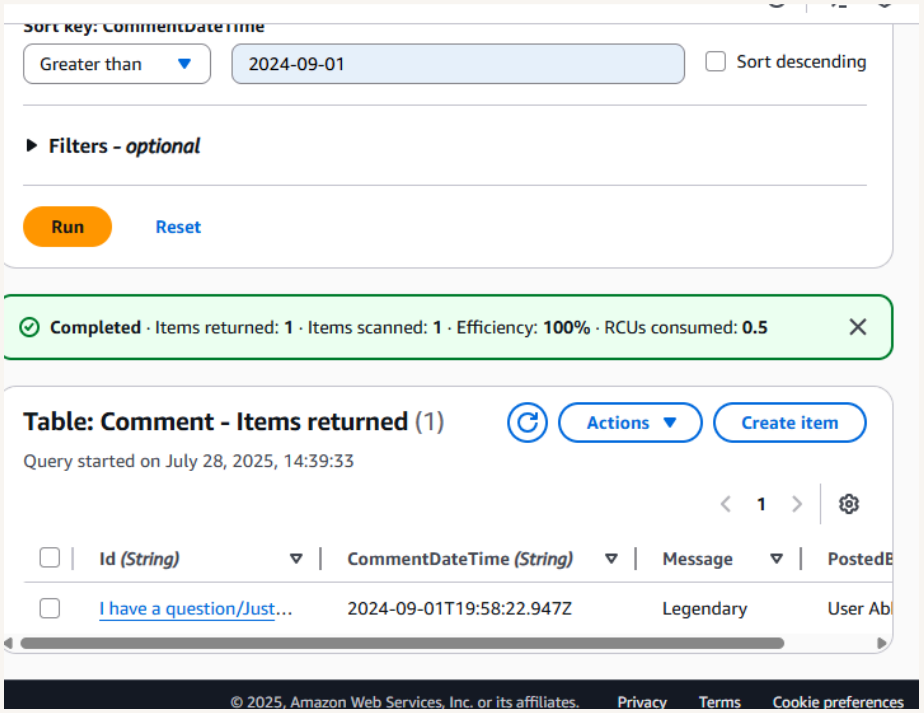
This project took me approximately 1 hour and 30 minutes to complete. It involved setting up DynamoDB tables, querying data using both the console and AWS CLI, and running a transaction, which required careful attention to detail and understanding how the data models relate to each other.

Abdulrahman Abdulkadir
NextWork Student

Querying DynamoDB Tables

A partition key is a primary key attribute in DynamoDB used to determine the partition (or storage location) of an item. It's required for every item and helps DynamoDB quickly locate and retrieve data based on that key's value. The value of the partition key is hashed internally by DynamoDB, which ensures that data is evenly distributed across storage partitions for high performance and scalability.

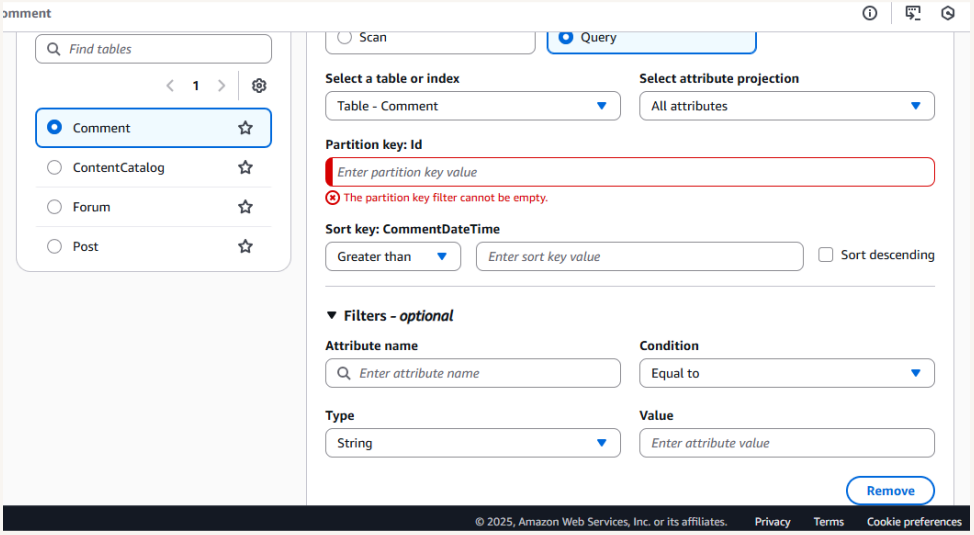
A sort key is an optional second part of the primary key in DynamoDB that allows multiple items with the same partition key to be stored and sorted by this key. It enables more complex querying and data organization within a partition, such as retrieving a range of items, filtering by sort key conditions, or ordering results by the sort key's value. This is useful when modeling one-to-many relationships in your data.


Sort key: CommentDateTime
Greater than
2024-09-01
Sort descending
Filters - optional
Run
Reset
Completed · Items returned: 1 · Items scanned: 1 · Efficiency: 100% · RCUs consumed: 0.5
Table: Comment - Items returned (1)
Actions
Create item
Query started on July 28, 2025, 14:39:33
1
Id (String)
CommentDateTime (String)
Message
I have a question/Just...
2024-09-01T19:58:22.947Z
Legendary
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Limits of Using DynamoDB

I ran into an error when I queried for all comments posted by User Abdulrahman because I didn’t include the required partition key (Id). DynamoDB needs the partition key to locate data, and filtering by another attribute like PostedBy alone isn't allowed in a query. This shows why data modeling is important. If you expect to query by attributes other than the partition key, you should plan for it—like using a Global Secondary Index. Unlike SQL, DynamoDB requires you to design your tables around how you’ll access the data.

Insights we could extract from our Comment table includes the number of comments per user, comments made on a specific post (if post IDs are used as partition keys), and activity over time when using a sort key like CommentDateTime. These are efficient because the table is designed to support queries using the defined keys. Insights we can't easily extract from the Comment table includes finding all comments by a specific user across different posts, unless that attribute is part of the key or indexed. Without a Global Secondary Index on fields like PostedBy, these kinds of queries are not efficient or even possible with the current table design.


Find tables
1
Comment
ContentCatalog
Forum
Post
Scan
Query
Select a table or index
Table - Comment
Select attribute projection
All attributes
Partition key: Id
Enter partition key value
The partition key filter cannot be empty.
Sort key: CommentDateTime
Greater than
Enter sort key value
Sort descending
Filters - optional
Attribute name
Enter attribute name
Condition
Equal to
Type
String
Value
Enter attribute value
Remove
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Abdulrahman Abdulkadir
NextWork Student

Running Queries with CLI

A query I ran in CloudShell was: ```bash aws dynamodb get-item \ --table-name ContentCatalog \ --key '{"Id":{"N":"202"}}' \ --projection-expression "Title, ContentType, Services" \ --return-consumed-capacity TOTAL ``` This query will retrieve the item with ID 202 from the ContentCatalog table, but only return the `Title`, `ContentType`, and `Services` attributes. It also reports how much read capacity was used, and since it uses eventual consistency by default, it consumes less capacity than a strongly consistent read.

Query options I could add to my query are: `--consistent-read`: This makes the read strongly consistent, ensuring I get the most up-to-date version of the data. It’s useful when accuracy matters, like in financial apps. `--projection-expression`: This lets me limit which attributes are returned, so I only get the specific fields I need (e.g., just the Title and ContentType), which saves resources. `--return-consumed-capacity`: This tells DynamoDB to show how much read capacity the query used. It's helpful for monitoring and optimizing table usage.

```
}
~ $
~ $ aws dynamodb get-item \
>     --table-name ContentCatalog \
>     --key '{"Id":{"N":"202"}}' \
>     --projection-expression "Title, ContentType, Services" \
>     --return-consumed-capacity TOTAL
{
    "Item": {
        "Title": {
            "S": "Don't miss out!"
        },
        "ContentType": {
            "S": "Video"
        }
    },
    "ConsumedCapacity": {
        "TableName": "ContentCatalog",
        "CapacityUnits": 0.5
    }
}
~ $
```

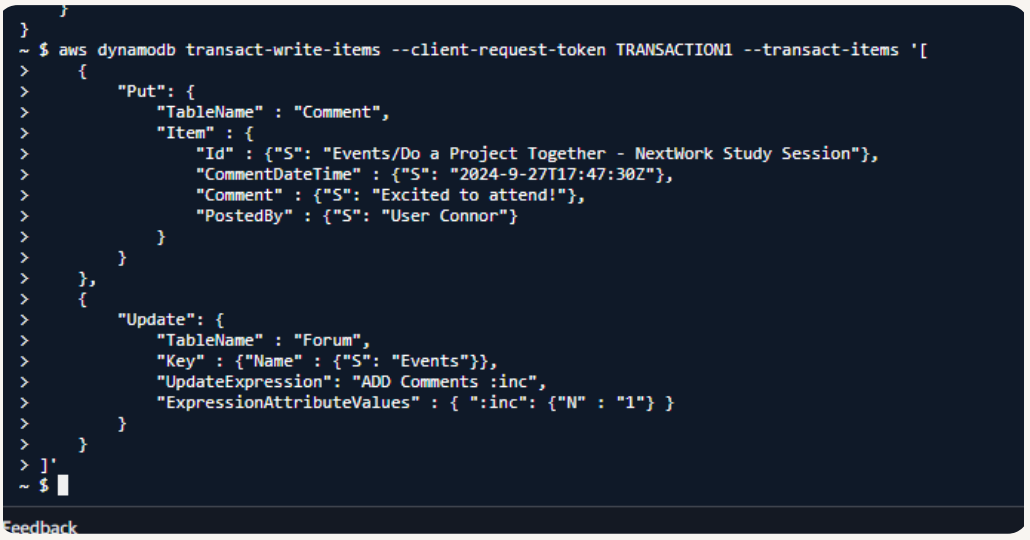
Feedback

Abdulrahman Abdulkadir
NextWork Student

Transactions

A transaction is a way to group multiple operations together so that they either all succeed or all fail as one unit. In DynamoDB, transactions help ensure data consistency across tables—for example, adding a new comment to the Comment table and updating the comment count in the Forum table at the same time. If one part of the transaction fails, none of the changes are applied.

I ran a transaction using the `aws dynamodb transact-write-items` command. This transaction did two things: it added a new comment to the `Comment` table with details like the ID, timestamp, comment text, and username; and it updated the `Forum` table by incrementing the `Comments` count for the "Events" forum by 1. Both operations were grouped into a single transaction to ensure consistency—if one failed, neither would be applied.



The place to learn & showcase your skills

Check out nextwork.org for more projects