nextwork.org

Query Data with DynamoDB

Joba Amunigun
https://www.linkedin.com/in/dvoice

Joba Amunigun
NextWork Student

Introducing Today's Project!

What is Amazon DynamoDB?

Amazon DynamoDB is a fully managed NoSQL database service by AWS. It's designed for high performance at scale, with automatic scaling, fast response times, and built-in security. Why it's useful: ⚡ Fast & Scalable: It handles large amounts of data and traffic with low latency. No server management: AWS takes care of all infrastructure, so you can focus on building. Secure & Reliable: It includes encryption, backups, and is highly available. Flexible schema: You don't need to define columns ahead of time — great for fast-moving projects. Great for real-time apps: Perfect for chat apps, gaming leaderboards, IoT, and more. In short, DynamoDB is useful when you need a fast, flexible, and reliable database that can grow with your application!

How I used Amazon DynamoDB in this project

I used Amazon DynamoDB in today's project to: Create and manage multiple tables — like Forum, Post, and Comment — each storing different but related data. Run queries using AWS CLI to find items by their keys or indexed attributes (like retrieving all comments for a post). Set up relationships between tables — for example, linking posts to forums and comments to posts. Use transactions to update multiple tables at once — like adding a comment and updating the comment count in the related forum in one step. Track counts and relationships — such as the number of posts in a forum or comments on a post. Overall, I used DynamoDB to simulate a real-world app backend with structured, related data and powerful querying — all without managing servers or worrying about performance at scale.

Joba Amunigun
NextWork Student

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

One thing I didn’t expect in this project was how easy and powerful DynamoDB transactions could be — especially updating multiple related tables at once (like adding a comment and updating a counter) in a single command. I thought it would be more complex, but the CLI made it surprisingly straightforward.

This project took me...

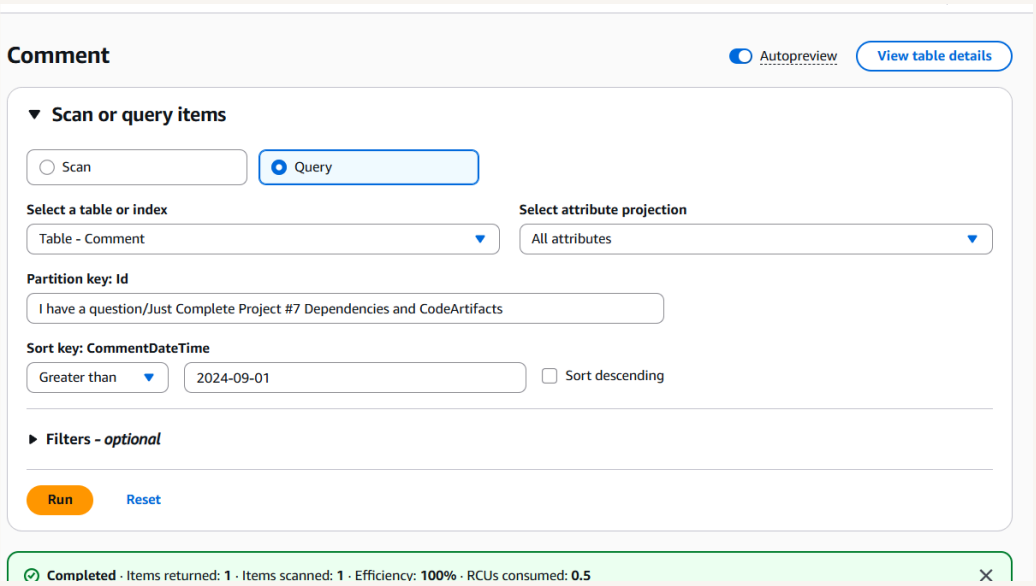
This project took me approximately 2 hours to complete, depending on how much time I spent understanding each concept and running the commands. The hands-on steps made it easier to grasp how Amazon DynamoDB works and how to use the AWS CLI effectively.

Joba Amunigun
NextWork Student

Querying DynamoDB Tables

A partition key is the primary attribute that DynamoDB uses to distribute and retrieve data efficiently across storage partitions. It acts like a filter that helps DynamoDB quickly locate items. Partition key values don't have to be unique, but in some cases—like in this table—they are, which makes retrieving individual items even faster.

A sort key is an optional secondary key in DynamoDB that further organizes and filters data after the partition key is applied. It allows multiple items with the same partition key to be stored in order and queried more precisely. When both a partition key and sort key are used together, they form a unique primary key that helps retrieve specific items even when they share the same partition key value.

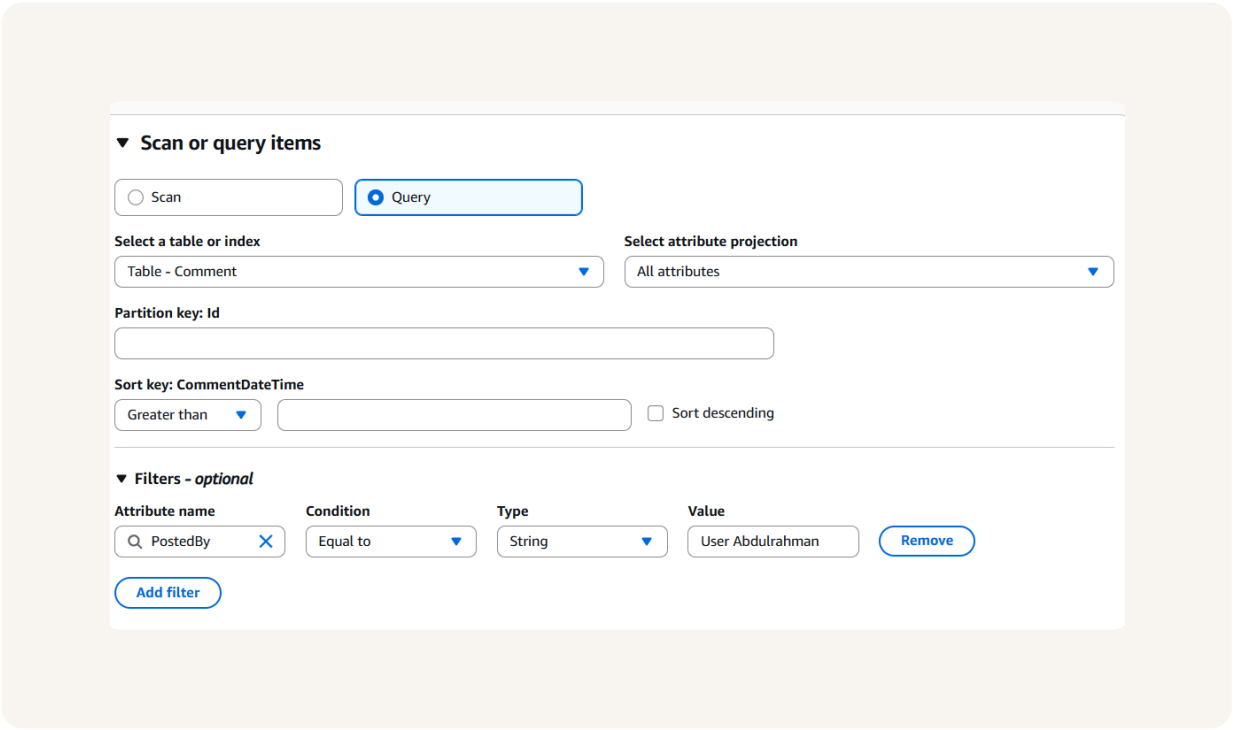

Comment
Autopreview
View table details
Scan or query items
Scan
Query
Select a table or index
Table - Comment
Select attribute projection
All attributes
Partition key: Id
I have a question/Just Complete Project #7 Dependencies and CodeArtifacts
Sort key: CommentDateTime
Greater than
2024-09-01
Sort descending
Filters - optional
Run
Reset
Completed · Items returned: 1 · Items scanned: 1 · Efficiency: 100% · RCUs consumed: 0.5

Joba Amunigun
NextWork Student

Limits of Using DynamoDB

I ran into an error when I queried for all comments posted by User Abdulrahman. This was because I didn't include the required Id (Partition key) in the query. In DynamoDB, when querying data, the partition key must always be provided — it's how the database locates and splits the data. Since I tried to filter only by the PostedBy attribute without using the partition key, the query failed. This highlights why data modeling is crucial in DynamoDB — choosing the right keys up front determines how easily and efficiently you can retrieve the data later.

Insights we could extract from our Comment table includes individual comments tied to specific Id and CommentDateTime, such as who posted them (PostedBy) and the message content (Message). We can also retrieve specific items quickly if we know their partition key (Id), which helps with pinpointing exact feedback or interactions. Insights we can't easily extract from the Comment table includes grouped or filtered data based on attributes like PostedBy, because queries in DynamoDB require the partition key to be used. This makes it difficult to retrieve all comments by a single user unless that user was part of the partition key structure. To support such queries, we'd need to rethink our data modeling — perhaps by creating a Global Secondary Index (GSI) that supports filtering by attributes like PostedBy.


Scan or query items
Scan
Query
Select a table or index
Table - Comment
Select attribute projection
All attributes
Partition key: Id
Sort key: CommentDateTime
Greater than
Sort descending
Filters - optional
Attribute name
Condition
Type
Value
PostedBy
Equal to
String
User Abdulrahman
Remove
Add filter

Joba Amunigun
NextWork Student

Running Queries with CLI

A query I ran in CloudShell was: aws dynamodb get-item \ --table-name ContentCatalog \ --key '{"Id":{"N":"202"}}' \ --projection-expression "Title, ContentType, Services" \ --return-consumed-capacity TOTAL This query will retrieve only the Title, ContentType, and Services attributes from the item with the Id of 202 in the ContentCatalog table. It uses an eventually consistent read, which is the default in DynamoDB, and it also tells us how much read capacity was consumed during the request.

Query options I could add to my query are: --consistent-read: This ensures the read is strongly consistent, meaning the data returned is guaranteed to be the most recent version. It's useful when accuracy is critical, like in financial or real-time apps. --projection-expression: This lets me specify only the attributes I want to retrieve from the item. It helps reduce data transfer and improves performance by narrowing the result. --return-consumed-capacity: This tells DynamoDB to include information about how much read capacity was consumed by the request. It's helpful for monitoring and optimizing usage costs.

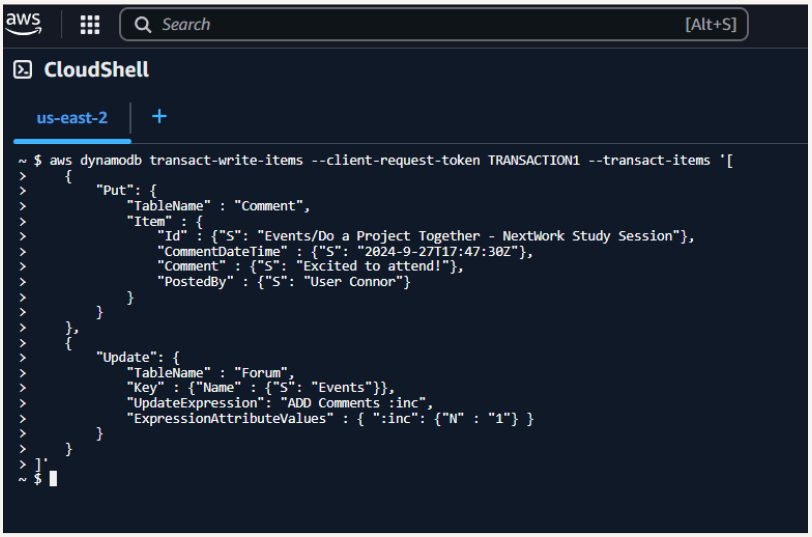
Joba Amunigun
NextWork Student

Joba Amunigun
NextWork Student

Transactions

A transaction is a group of one or more operations in a database that are processed together as a single unit — either all succeed or none do. In DynamoDB, a transaction ensures that when you're updating related items (like adding a comment and updating the comment count in another table), both actions happen together. If one fails, the other is rolled back automatically — no partial updates, no messy data. It's like buying something online: your card gets charged only if the item is in stock. No item, no charge.

I ran a transaction using the AWS CLI with the "transact-write-items" command. This transaction did two things: It added a new comment to the Comment table. It updated the comment count in the related Forum item in the Forum table. Both actions were combined in a single command so they would only succeed if both operations worked — ensuring the data stays consistent and accurate.



The place to learn & showcase your skills

Check out nextwork.org for more projects