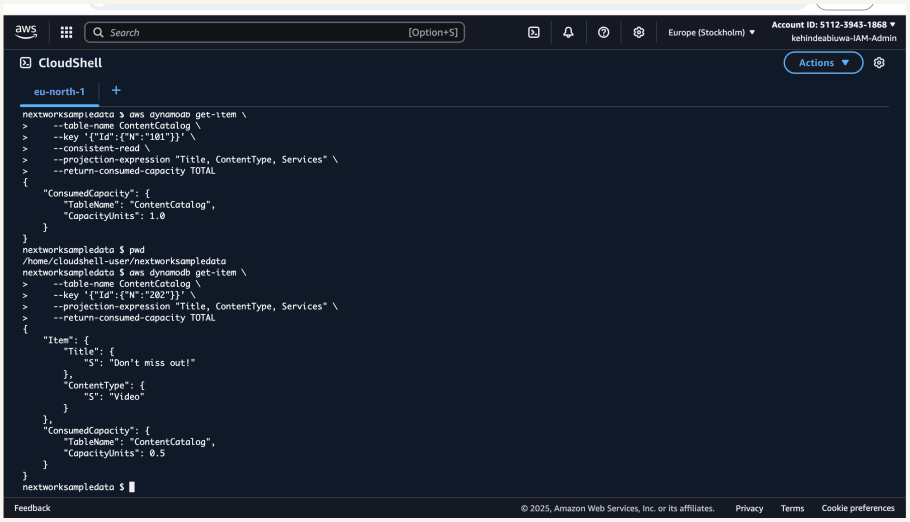
nextwork.org

Query Data with DynamoDB

Kehinde Abiuwa



Kehinde Abiuwa
NextWork Student

Introducing Today's Project!

What is Amazon DynamoDB?

Amazon DynamoDB is a fully managed NoSQL database service from AWS that stores data in key-value and document formats. It is useful because it provides: Scalability → automatically handles large amounts of data and traffic without manual intervention. High performance → delivers single-digit millisecond response times for reads and writes. Flexibility → doesn't require a fixed schema, so items in the same table can have different attributes. Fully managed operations → AWS takes care of backups, replication, security, and software patching. ACID transactions → ensures data consistency across multiple operations when needed. This makes DynamoDB ideal for applications like gaming, e-commerce, IoT, and any workload that needs fast, reliable, and scalable access to data.

How I used Amazon DynamoDB in this project

In today's project, I used Amazon DynamoDB to create multiple tables, load sample data into them using AWS CloudShell, run queries with partition and sort keys, and even perform a transaction that updated two tables at once. This helped me understand how DynamoDB manages NoSQL data, supports flexible schemas, and ensures consistency across related tables.

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

One thing I didn't expect in this project is how DynamoDB requires a partition key for every query—you can't just search the whole table like in SQL without using scans, which makes designing the right keys really important from the start.

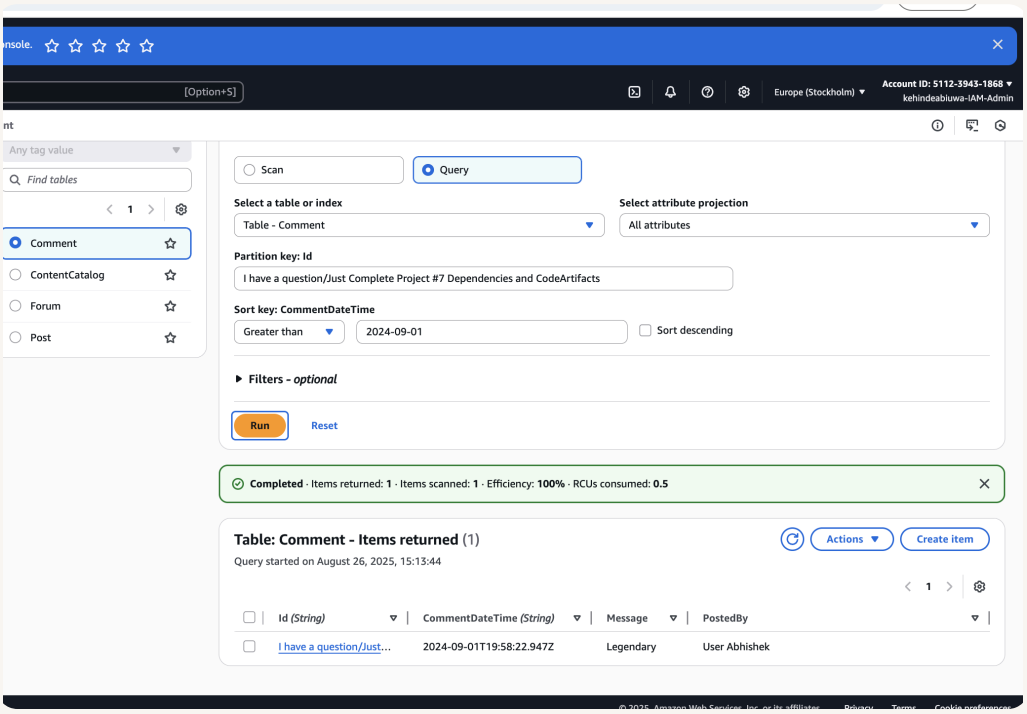
This project took me...

This project took me 40 minutes

Querying DynamoDB Tables

A partition key is the primary key attribute in a DynamoDB table that determines how data is distributed across partitions. Each item in the table must have a unique partition key value, and DynamoDB uses this value's hash to decide which partition will store the item. This ensures efficient lookups, since DynamoDB can quickly find the item by its partition key without scanning the entire table.

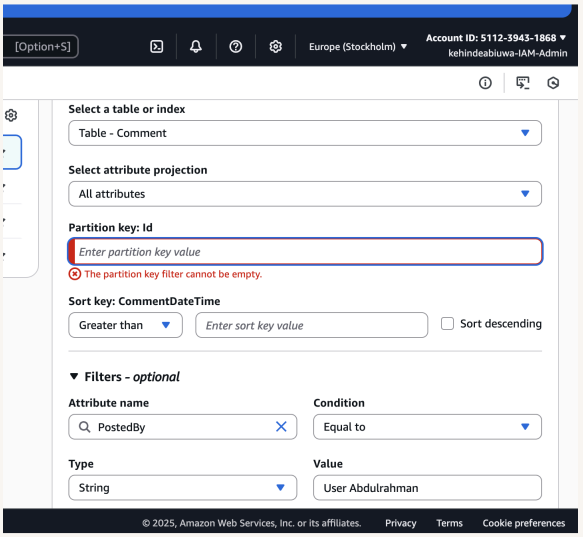
A sort key is the second part of a composite primary key in DynamoDB. It allows multiple items to share the same partition key while being uniquely identified and ordered by the sort key. This makes it possible to group related items together and query them efficiently in sorted order (e.g., by date, ID, or category).


[Option+S]
Europe (Stockholm)
Account ID: 5112-3943-1868
kehindeabiuwa-IAM-Admin
Any tag value
Find tables
Comment
ContentCatalog
Forum
Post
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
Table: Comment - Items returned (1)
Query started on August 26, 2025, 15:13:44
Actions
Create item
Id (String)
CommentDateTime (String)
Message
PostedBy
I have a question/Just...
2024-09-01T19:58:22.947Z
Legendary
User Abhishek

Limits of Using DynamoDB

I ran into an error when I queried for items in the Comment table because I didn't provide a partition key value. In DynamoDB, every query must include the partition key (in this case, Id), since it's required to locate the correct partition where the data is stored. Without it, DynamoDB cannot execute the query.

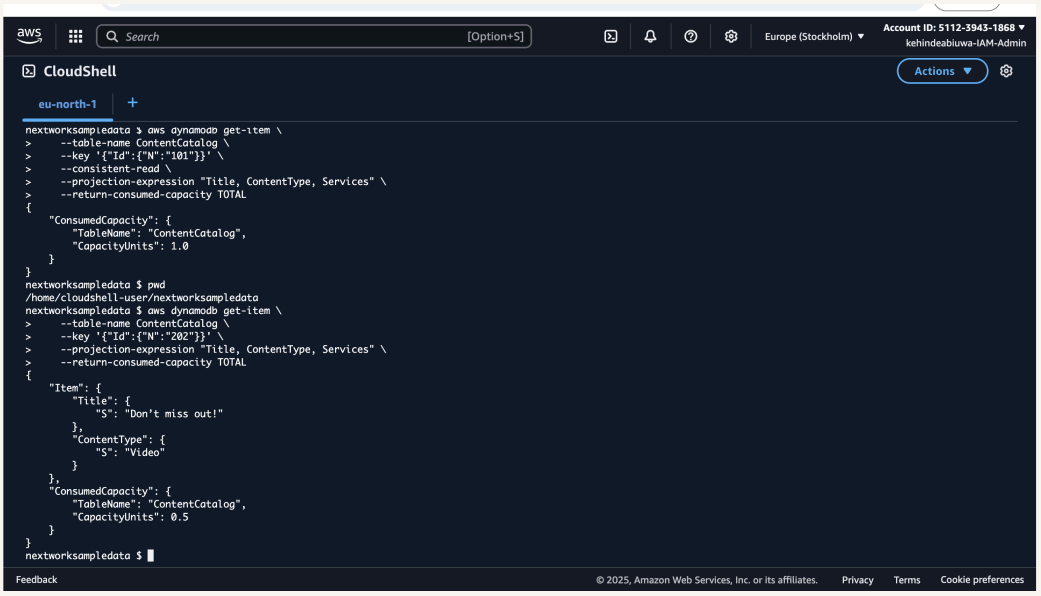
Insights we could extract from our Comment table include: The exact comments left by users (e.g., Great work., Excellent). When each comment was posted, using the CommentDateTime sort key. Which post or discussion each comment belongs to, using the Id partition key. Insights we can't easily extract from the Comment table include: How many total comments each user has made (since PostedBy isn't the partition key). Trends across all comments (e.g., average sentiment or most active users). Relationships between comments in different tables (e.g., linking comments directly to Forum or Post details) without running additional queries or joins.



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 return the item in the ContentCatalog table with Id = 202, but only show the attributes Title, ContentType, and Services. In this case, it returned: Title: "Don't miss out!" ContentType: "Video" It also reported the consumed capacity units used by the request.

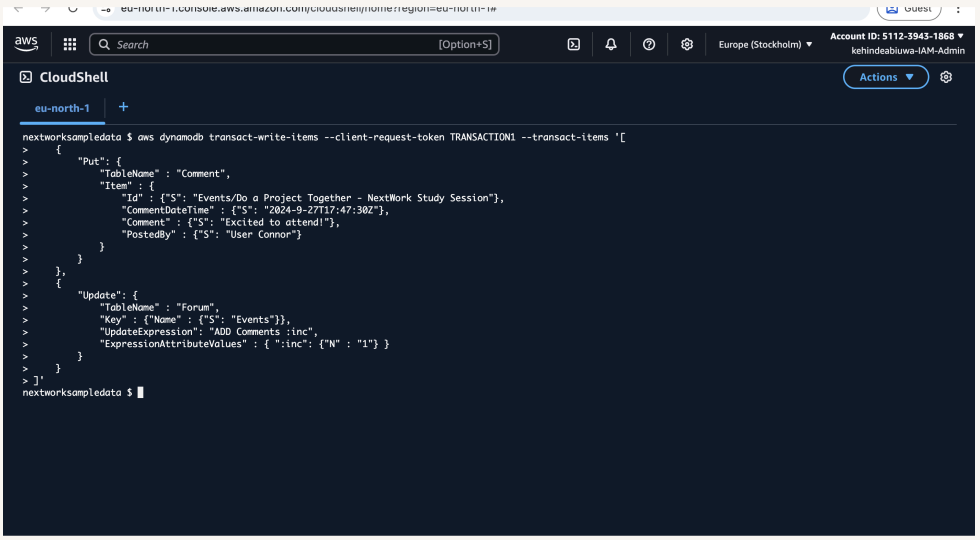
Query options I could add to my query are: --consistent-read → to ensure the query returns the most up-to-date data. --projection-expression → to return only specific attributes instead of the whole item. --filter-expression → to filter results based on non-key attributes. --expression-attribute-names → to use placeholders for attribute names (helpful if names are reserved words). --expression-attribute-values → to define placeholder values used in filter expressions. --return-consumed-capacity → to see how many read capacity units were consumed. These options make DynamoDB queries more efficient and flexible, depending on what data I need.


aws
Search
[Option+S]
Europe (Stockholm)
Account ID: 5112-3943-1868
kehindeabiuwa-IAM-Admin
CloudShell
Actions
eu-north-1
nextworksampleadata $ aws dynamodb get-item \
> --table-name ContentCatalog \
> --key '{"Id":{"N":"101"}}' \
> --consistent-read \
> --projection-expression "Title, ContentType, Services" \
> --return-consumed-capacity TOTAL
{
"ConsumedCapacity": {
"TableName": "ContentCatalog",
"CapacityUnits": 1.0
}
}
nextworksampledata $ pwd
/home/cloudshell-user/nextworksampledata
nextworksampledata $ 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
}
}
nextworksampledata $
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Transactions

A transaction is an operation that groups multiple actions together so they either all succeed or all fail as one unit. In DynamoDB, transactions (like TransactWriteItems) let you update or write to multiple items across one or more tables while maintaining consistency, isolation, and durability. This ensures that your data stays consistent, even if something goes wrong during the update process.

I ran a transaction using the aws dynamodb transact-write-items command. This transaction did two things: It updated an item in one DynamoDB table (for example, adding or changing an attribute). It updated a related item in another table at the same time. Both updates were processed together as a single atomic operation, meaning they either both succeeded or both failed, ensuring my data stayed consistent across the two tables



The place to learn & showcase your skills

Check out nextwork.org for more projects