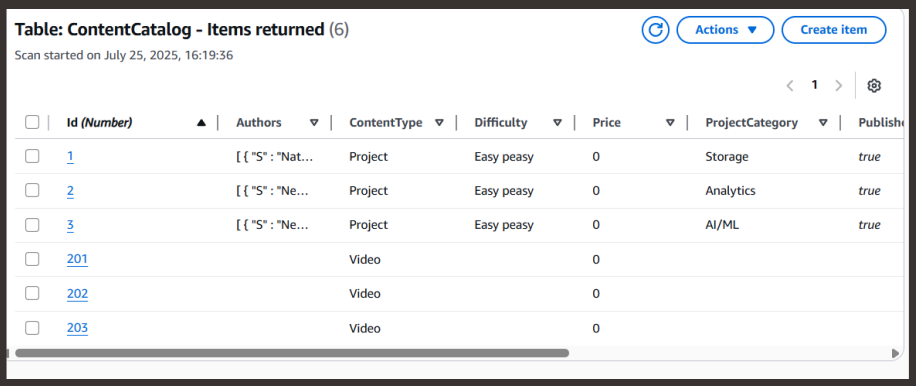
nextwork.org

Load Data into a DynamoDB Table

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 offered by AWS that provides fast, flexible, and scalable performance for applications of any size. It stores data in a key-value and document format, which allows for quick access without the need for complex joins or queries. Why is it useful? Because it's serverless and automatically scales with demand, meaning developers don’t need to manage infrastructure or worry about provisioning. It also supports automatic backups, encryption, and fine-grained access control. DynamoDB is especially useful for applications that need low-latency data access at any scale—like mobile apps, gaming backends, e-commerce platforms, and IoT systems.

How I used Amazon DynamoDB in this project

I used Amazon DynamoDB in today’s project to create and manage database tables directly from AWS CloudShell using the AWS CLI. I: Created multiple DynamoDB tables like ContentCatalog, Forum, Post, and Comment. Loaded data into each table using the batch-write-item command and verified the entries. Explored how each item could have a different set of attributes, showing DynamoDB’s flexibility. Practiced updating items (e.g., adding StudentsComplete to a project) and saw that it didn’t affect other items—unlike in a relational database. This hands-on experience helped me understand the speed, scalability, and schema flexibility DynamoDB offers, making it ideal for real-world applications.

Joba Amunigun
NextWork Student

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

ChatGPT said: One thing I didn’t expect in this project was how DynamoDB allows each item to have completely different attributes. I’m used to relational databases where every row must follow a strict schema, so it was surprising (and pretty cool) to see that I could add a new attribute like StudentsComplete to one item, and it didn’t affect the others. That level of flexibility really stood out!

This project took me...

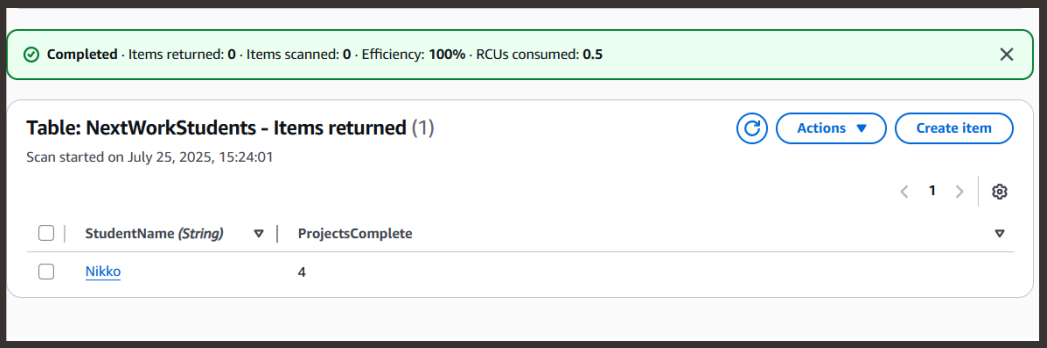
This project took me about 2 hours to complete. Most of the time went into exploring AWS CloudShell, running the CLI commands, and understanding how DynamoDB handles flexible data structures. It was a hands-on experience that helped reinforce the concepts better than just reading about them.

Joba Amunigun
NextWork Student

Create a DynamoDB table

DynamoDB may look like it uses rows and columns, but under the hood, it's very different from a traditional relational database. Instead of being organized like a rigid spreadsheet, DynamoDB stores data as items (like individual records), and each item can have its own unique set of attributes (like fields). □ Think of it like this: In a relational database, every row must follow a strict schema — same columns, same data types. In DynamoDB, each item can have different attributes, even if they're in the same table. You could have one item with just a name, and another with name, age, and a list of projects. This flexibility is what makes DynamoDB non-relational — there's no fixed schema, and items don't need to relate to each other through joins or foreign keys like in relational systems. It's more like a collection of JSON-like objects, not structured rows in a table. You'll see how powerful this gets in the next steps ☺

An attribute is like a piece of data about an item. In this case, our item is Nikko and the attribute is the number of projects Nikko completed. Each item in DynamoDB can have multiple attributes. But, unlike relational databases where each row in a table must have the same columns, DynamoDB items can have their own unique set of attributes.


Completed · Items returned: 0 · Items scanned: 0 · Efficiency: 100% · RCUs consumed: 0.5
Table: NextWorkStudents - Items returned (1)
Scan started on July 25, 2025, 15:24:01
Actions
Create item
1
StudentName (String)
ProjectsComplete
Nikko
4

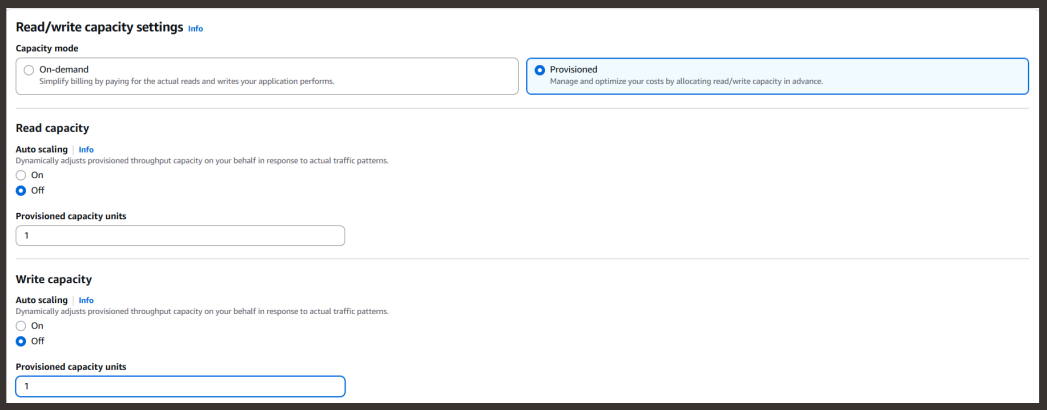

Joba Amunigun
NextWork Student

Read and Write Capacity

Read capacity units (RCUs) and write capacity units (WCUs) are ways to measure the amount of throughput capacity allocated to a DynamoDB table. RCUs represent how many "engines" DynamoDB uses to handle read operations. 1 RCU = up to 2 reads per second for strongly consistent reads of items up to 4 KB in size. If your application needs more reads per second, you'll need more RCUs. WCUs represent the power DynamoDB uses to handle write operations. 1 WCU = 1 write per second for items up to 1 KB in size. Since writes are more resource-intensive than reads, they tend to cost more. AWS charges based on the number of RCUs and WCUs your application consumes monthly, but under the Free Tier, you get 25 RCUs and 25 WCUs (enough to support up to 200 million requests per month) at no cost—just remember to delete your resources afterward to avoid unexpected charges.

ChatGPT said: Amazon DynamoDB's Free Tier covers 25 GB of data storage, along with 25 Read Capacity Units (RCUs) and 25 Write Capacity Units (WCUs). This allows you to handle up to 200 million requests per month at no cost, assuming your usage stays within these limits. The Free Tier also includes features like backups and global tables in supported regions, as long as the usage remains within the free thresholds. I turned off auto scaling because I wanted to stay within the Free Tier limits and avoid unexpected charges. Auto scaling can automatically increase RCUs and WCUs based on traffic, which is useful for production workloads but not necessary for a test or demo environment like this one.

Joba Amunigun
NextWork Student



Joba Amunigun
NextWork Student

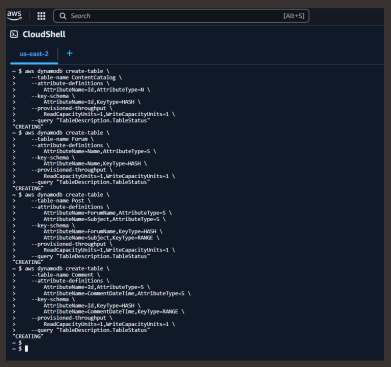
Using CLI and CloudShell

AWS CloudShell is a browser-based command-line environment built right into the AWS Management Console. It gives you a ready-to-use space where you can run code and manage AWS resources without installing anything on your computer. CloudShell comes with AWS CLI (Command Line Interface) pre-installed, so you can create, delete, and update AWS resources using simple commands instead of clicking through the console. It's perfect for both beginners and professionals — beginners can explore without setup, and experts can use it to automate tasks, write scripts, and manage cloud infrastructure more efficiently.

AWS CLI is the Command Line Interface for AWS — a powerful tool that allows you to interact with AWS services using typed commands instead of clicking through the AWS Management Console. With AWS CLI, you can create, update, and delete resources like DynamoDB tables, EC2 instances, S3 buckets, and more — all from your terminal. It's especially useful for automating tasks, writing scripts, and managing resources faster and more efficiently. Normally, you'd have to install AWS CLI on your computer, but AWS CloudShell already has it pre-installed and ready to use.

I ran a CLI command in AWS CloudShell that created four new tables in AWS DynamoDB, each with specific attributes and settings. The four tables created are: ContentCatalog Table: This table has a numeric attribute called Id. Forum Table: This table has a partition key called Name. Post Table: This table has a partition key called ForumName and a sort key called Subject. We'll dive into sort keys soon! Comment Table: This table has a partition key called Id and a sort key called CommentDateTime.

Joba Amunigun
NextWork Student



Joba Amunigun
NextWork Student

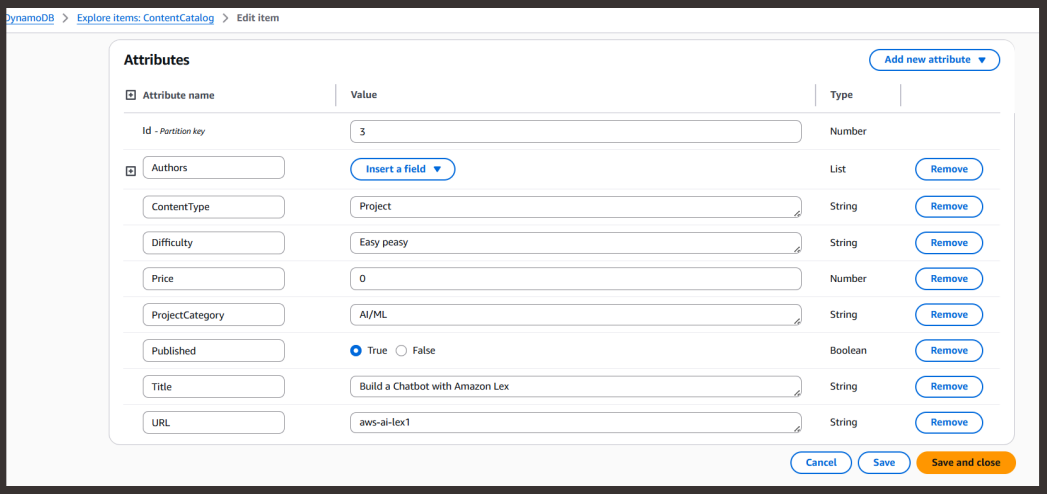
Loading Data with CLI

I ran a CLI command in AWS CloudShell that loaded data into my DynamoDB tables using the batch-write-item command. This command allowed me to insert multiple items into each table by pulling the data from local JSON files stored in my CloudShell environment. Here are the specific commands I used: aws dynamodb batch-write-item --request-items file://ContentCatalog.json aws dynamodb batch-write-item --request-items file://Forum.json aws dynamodb batch-write-item --request-items file://Post.json aws dynamodb batch-write-item --request-items file://Comment.json Each file knew exactly which DynamoDB table to load data into based on the structure inside the JSON. It was a quick and efficient way to populate my database without manually adding items one by one!

```
nextworksampledata $ aws dynamodb batch-write-item --request-items file://ContentCatalog.json
{
    "UnprocessedItems": {}
}
nextworksampledata $
nextworksampledata $ aws dynamodb batch-write-item --request-items file://Forum.json
{
    "UnprocessedItems": {}
}
nextworksampledata $
nextworksampledata $ aws dynamodb batch-write-item --request-items file://Post.json
{
    "UnprocessedItems": {}
}
nextworksampledata $
nextworksampledata $ aws dynamodb batch-write-item --request-items file://Comment.json
{
    "UnprocessedItems": {}
}
nextworksampledata $
```

Joba Amunigun
NextWork Student

Observing Item Attributes



I checked a ContentCatalog item, which had the following attributes : Id: 3 Authors: ["NextWork"] ContentType: Project Difficulty: Easy peasy Price: 0 ProjectCategory: AI/ML Published: true Title: Build a Chatbot with Amazon Lex URL: aws-ai-lex1 These attributes give us structured info about each project, such as its topic, complexity, and visibility status.

I checked another ContentCatalog item, which had a different set of attributes: Id: 203 ContentType: Video Price: 0 Title: Don’t miss out! URL: https://youtube.com/shorts/xDAwEu8i3BA VideoType: Shorts This Video item doesn't have attributes like Authors, Difficulty, or ProjectCategory, which were present in the Project item.

Instead, it has a VideoType attribute. This shows that each item in the table can have different attributes depending on the content type.


Joba Amunigun
NextWork Student

Benefits of DynamoDB

A benefit of DynamoDB over relational databases is flexibility, because each item in a DynamoDB table can have its own unique set of attributes. Unlike relational databases, which require every row to follow the same schema (same columns in every row), DynamoDB allows different items to store only the data that applies to them. For example, in DynamoDB, a Video item might have a VideoType attribute while a Project item has Authors and Difficulty — and that's totally fine. In a relational database, adding a new column like StudentsComplete would require every row to have a value for it, even if most items don't need it. This makes DynamoDB much more adaptable to evolving data needs. This flexibility also leads to better speed in some cases — DynamoDB uses partition keys to quickly locate items, while relational databases often need to scan entire tables, especially when not properly indexed.

Another benefit over relational databases is speed, because DynamoDB uses partition keys to quickly locate and retrieve data without scanning the entire table. This allows it to deliver low-latency responses even at massive scale. Relational databases, on the other hand, often need to scan through multiple rows or even the entire table—especially if indexes aren't optimized—which can slow down performance as the dataset grows. DynamoDB is also designed for high throughput and horizontal scaling, meaning it can handle large volumes of traffic and data more efficiently. This makes it ideal for applications that require quick access to diverse and dynamic datasets, like real-time apps, e-commerce platforms, or serverless architectures.

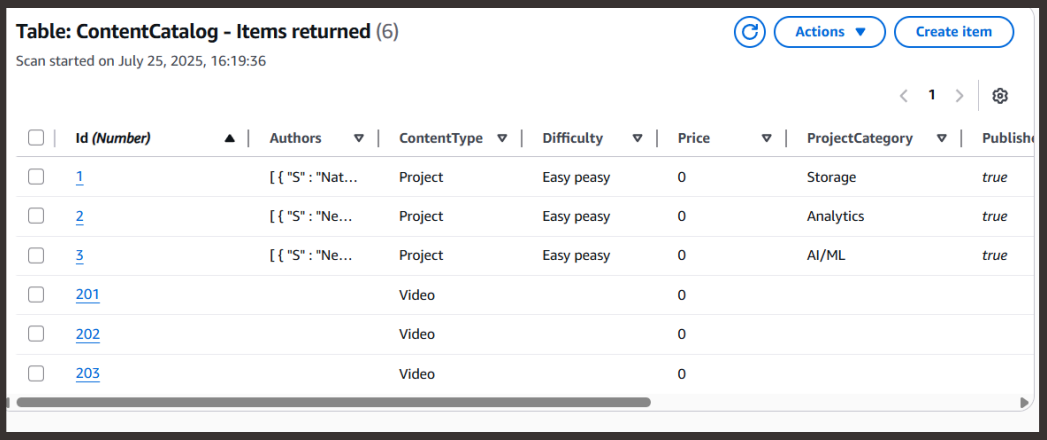


Table: ContentCatalog - Items returned (6)

Scan started on July 25, 2025, 16:19:36

Actions | Create item

1

Id *(Number)*	Authors	ContentType	Difficulty	Price	ProjectCategory	Publish
1	[{ "S" : "Nat...	Project	Easy peasy	0	Storage	*true*
2	[{ "S" : "Ne...	Project	Easy peasy	0	Analytics	*true*
3	[{ "S" : "Ne...	Project	Easy peasy	0	AI/ML	*true*
201		Video		0		
202		Video		0		
203		Video		0		

nextwork.org

The place to learn & showcase your skills

Check out nextwork.org for more projects