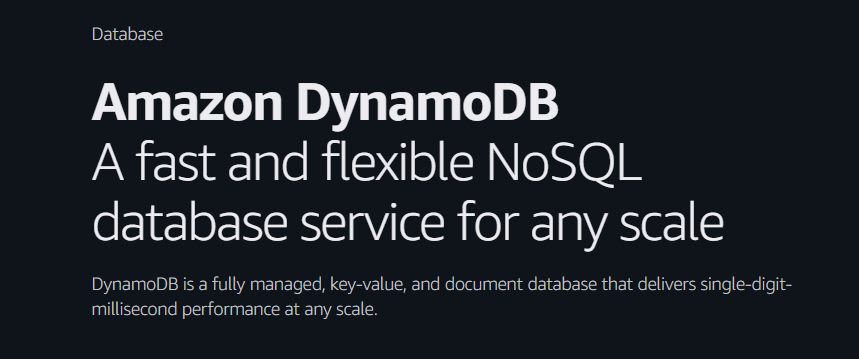
# Amazon DynamoDB



## Overview:

Amazon DynamoDB is a fully managed NoSQL database service provided by AWS, designed for fast and predictable performance with seamless scalability. It’s commonly used for applications that require low-latency data access at any scale, like gaming, ad tech, IoT, mobile apps, etc.

### Key Features of DynamoDB:

**1. Fully Managed**

* You don’t need to manage infrastructure (no servers, no updates, no patching).
* AWS handles provisioning, patching, replication, fault tolerance, backups, etc.

**2. High Performance at Scale**

* Delivers single-digit millisecond latency for reads/writes—even at millions of requests per second.
* Consistent performance whether your table has 10 items or 10 million.

**3. Automatic Scaling**

* Supports Auto Scaling in provisioned mode.
* Automatically adjusts throughput based on usage patterns, ensuring performance and cost-efficiency.

**4. Serverless + Pay-per-Use**

* You don’t provision servers—just use the table.
* With on-demand mode, you’re billed only for the actual read/write requests.

**5. Global Tables**

* Multi-region replication with active-active writes across regions.
* Great for global applications needing low-latency access everywhere.

**6. DAX (DynamoDB Accelerator)**

* In-memory cache that sits in front of DynamoDB.
* Reduces read latency even further to microseconds.
* Fully managed, compatible with DynamoDB SDKs.

**7. Fine-Grained Access Control**

* Use IAM policies + Condition keys to control access at the item or attribute level.

### Components of DynamoDB:

**1. Tables**

* A container for your data.
* You define a primary key (partition key or composite key) when creating a table.

**2. Items**

* Equivalent to a row in a relational database.
* Each item is a collection of attributes.
* Each item must have a primary key, but the rest of the attributes can vary from item to item (schema-less).

**3. Attributes**

* Equivalent to columns.
* Each attribute is a name–value pair.
* DynamoDB supports several data types:
  + Scalar types: String, Number, Boolean, Binary
  + Document types: List, Map
  + Set types: String Set, Number Set, Binary Set

**4. Primary Key**

* Uniquely identifies each item in the table.
* Two types:
  + Partition Key (simple key)
  + Partition Key + Sort Key (composite key)

**5. Secondary Indexes**

* Used to query the table in different ways, other than using the primary key.

**a. Global Secondary Index (GSI)**

* Can have different partition and sort keys than the base table.
* Can access any attribute, not just primary key.
* Can be created any time.

**b. Local Secondary Index (LSI)**

* Uses the same partition key as the base table but a different sort key.
* Can only be defined at table creation.

**6. TTL (Time to Live)**

* Automatically deletes expired items from your table.
* Helps with data retention and cost savings.

### Read and Write Capacity Modes:

DynamoDB offers two capacity modes,

*1. Provisioned Capacity:*

* You manually define Read Capacity Units (RCUs) and Write Capacity Units (WCUs).
* Ideal for predictable workloads.
* Supports Auto Scaling to adjust capacity as needed.
* You pay for:
  + RCUs: 1 RCU = 1 strongly consistent read/sec for items up to 4KB.
  + WCUs: 1 WCU = 1 write/sec for items up to 1KB.
* **Example:**
* You set 10 RCUs and 5 WCUs:
  + You can read 40KB of data/sec (10 RCUs × 4KB).
  + You can write 5KB of data/sec (5 WCUs × 1KB).

*2. On-Demand Capacity:*

* No need to provision capacity.
* DynamoDB auto-scales based on traffic.
* Ideal for unpredictable or spiky workloads.
* You only pay per read/write request.

**When to Use What?**

| **Use Case** | **Provisioned Mode** | **On-Demand Mode** |
| --- | --- | --- |
| Steady and predictable traffic | Yes | No |
| Spiky or unpredictable traffic | No | Yes |
| Cost control by limiting capacity | Yes | No |
| Minimal management effort | No | Yes |

### Primary Key:

The Primary Key is what DynamoDB uses to uniquely identify each item (row) in a table.

Unlike traditional relational databases where the primary key might be an auto-incremented ID, in DynamoDB, the Primary Key is mandatory and must be defined at the time of table creation. It determines how your data is stored and retrieved.

**There are two types of primary keys:**

#### 1. Partition Key (Simple Primary Key):

**Structure:**

* Just oneattribute (e.g., UserId, Email, etc.)

**Behaviour:**

* DynamoDB uses the Partition Key’s value to compute a hash, which determines which partition (physical storage node) your item goes to.
* Each value of the partition key must be unique in the table.

**Example:**

Suppose you're storing users:

| **Partition Key (UserId)** | **Name** | **Email** |
| --- | --- | --- |
| 101 | Ajinkya | aj@example.com |
| 102 | Sneha | sneha@example.com |

Here, UserId is the Partition Key, and every user has a unique value.

*2. Partition Key + Sort Key (Composite Primary Key):*

**Structure:**

* A Partition Key + a Sort Key.

**Behaviour:**

* All items with the same Partition Key are stored together in one partition.
* Items are sorted by the Sort Key within that partition.
* You can store multiple items with the same Partition Key as long as the Sort Key is different.

**Example:**

Suppose you're storing all tasks by users.

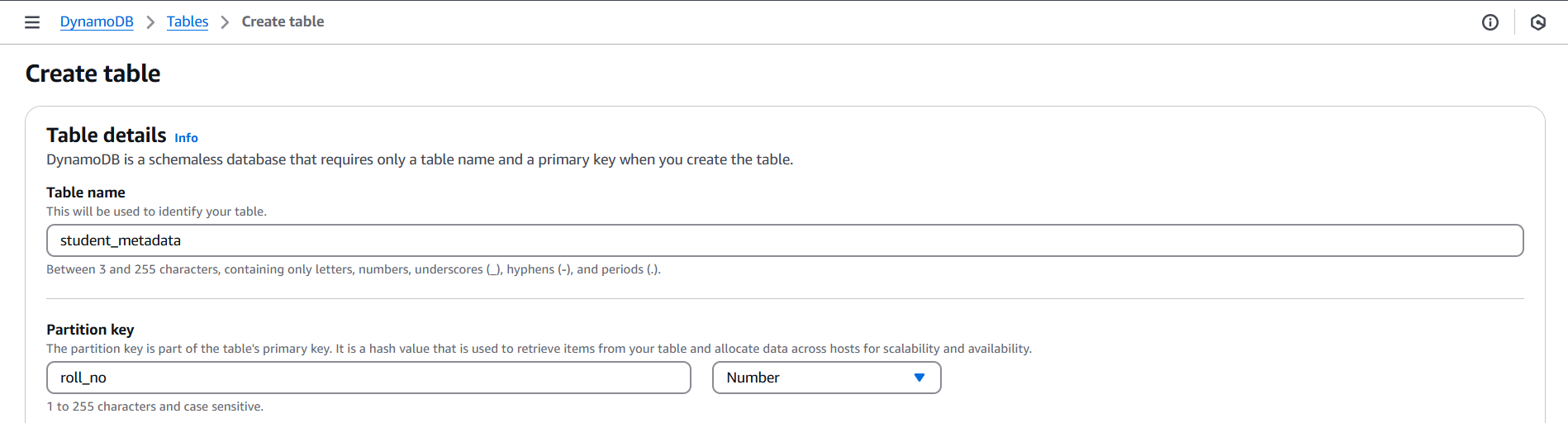
| **Partition Key (UserId)** | **Sort Key (TaskId)** | **Task Name** | **Status** |
| --- | --- | --- | --- |
| 101 | task-1 | Buy groceries | pending |
| 101 | task-2 | Study DynamoDB | done |
| 102 | task-1 | Workout | pending |

* Here, the primary key is a combination of UserId (Partition Key) and TaskId (Sort Key).
* This allows multiple tasks for the same user, uniquely identified by the combination of both.

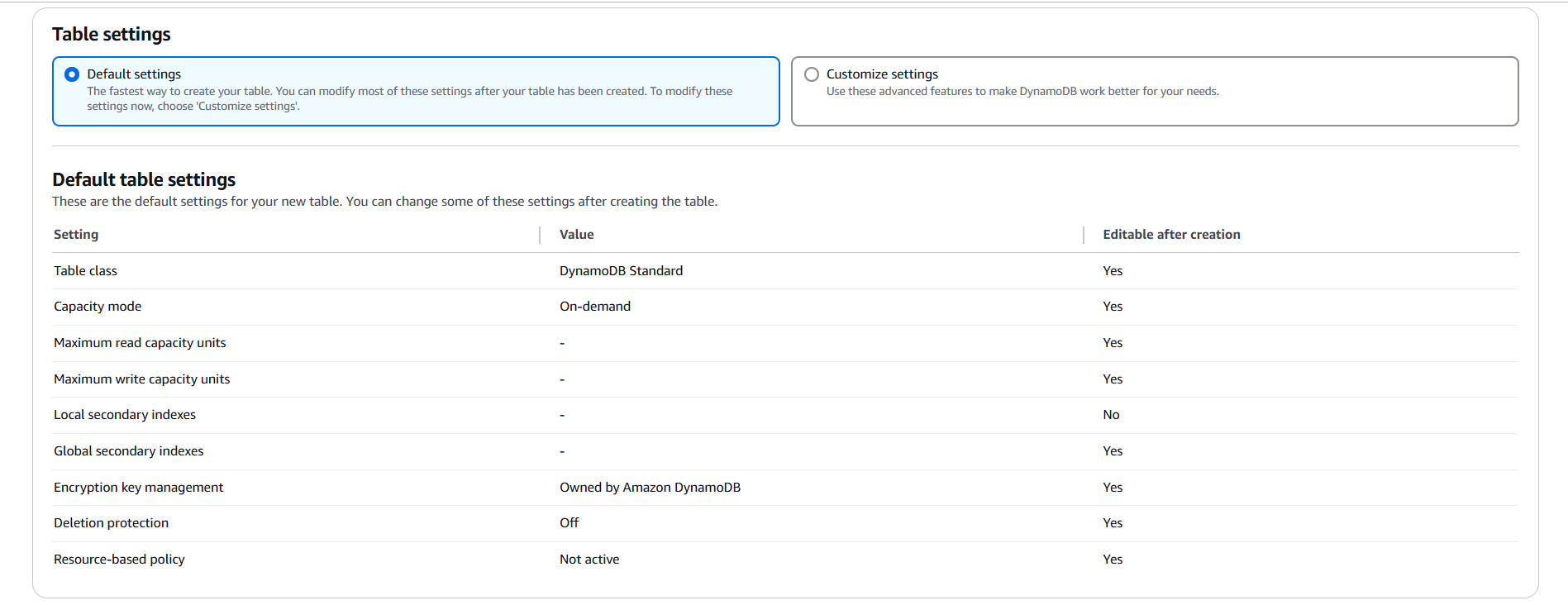
### Practical-1:

Create a table in DynamoDB with only partition key.

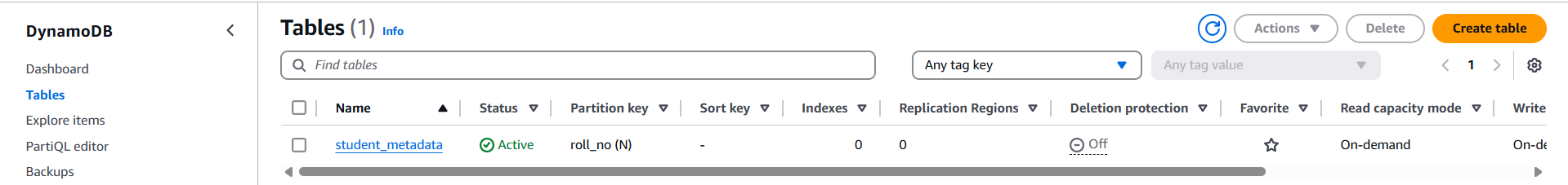
* Open DynamoDB service from search bar and click on “Create table”.
* Enter table name and partition key and keep sort key empty.



* Keep table settings as default where we can see capacity mode is “On-Demand”.



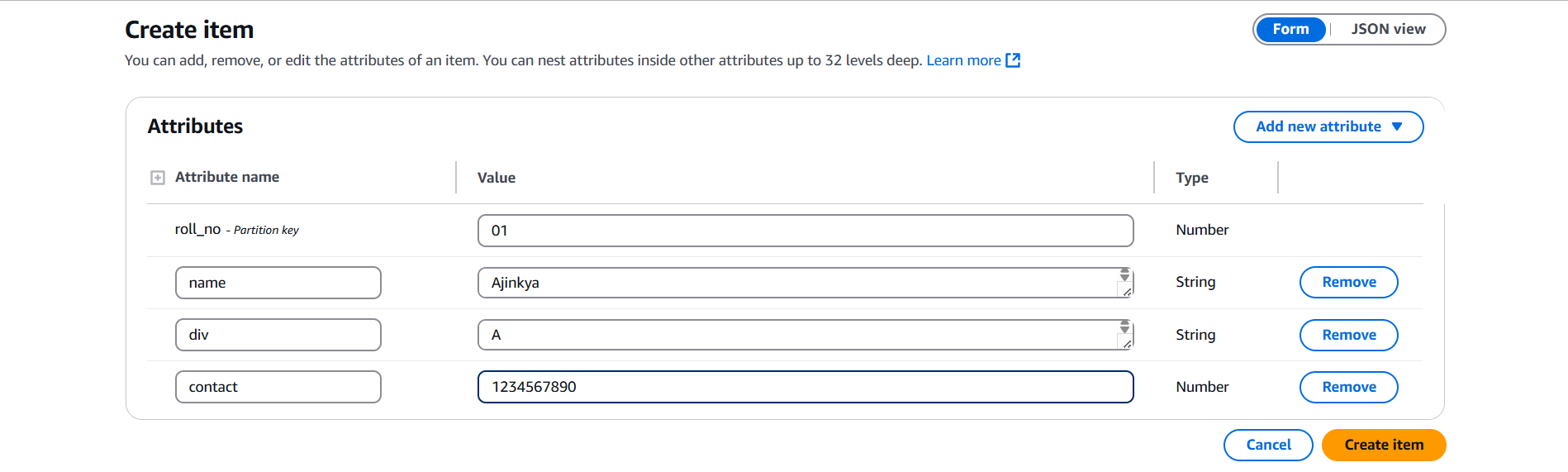
* Then click on ‘create table’ and wait for a while until the table is created.
* Once the status becomes “Active”, click on table name to open details.



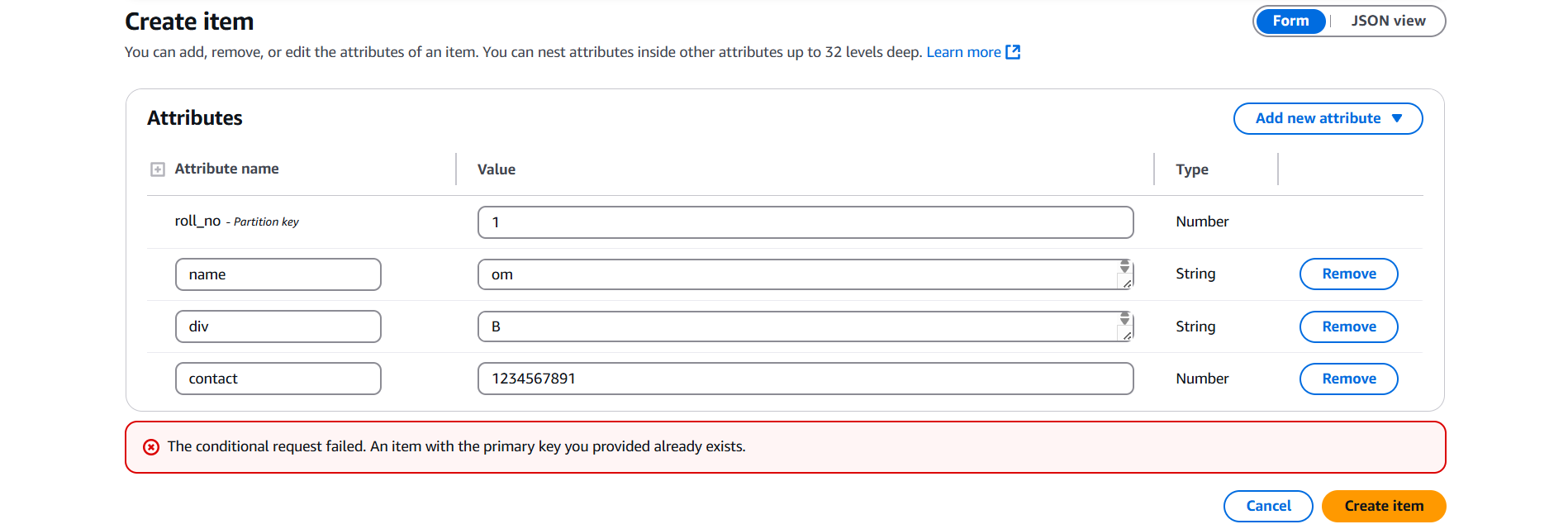
* Click on “Actions” and then “Create item”.



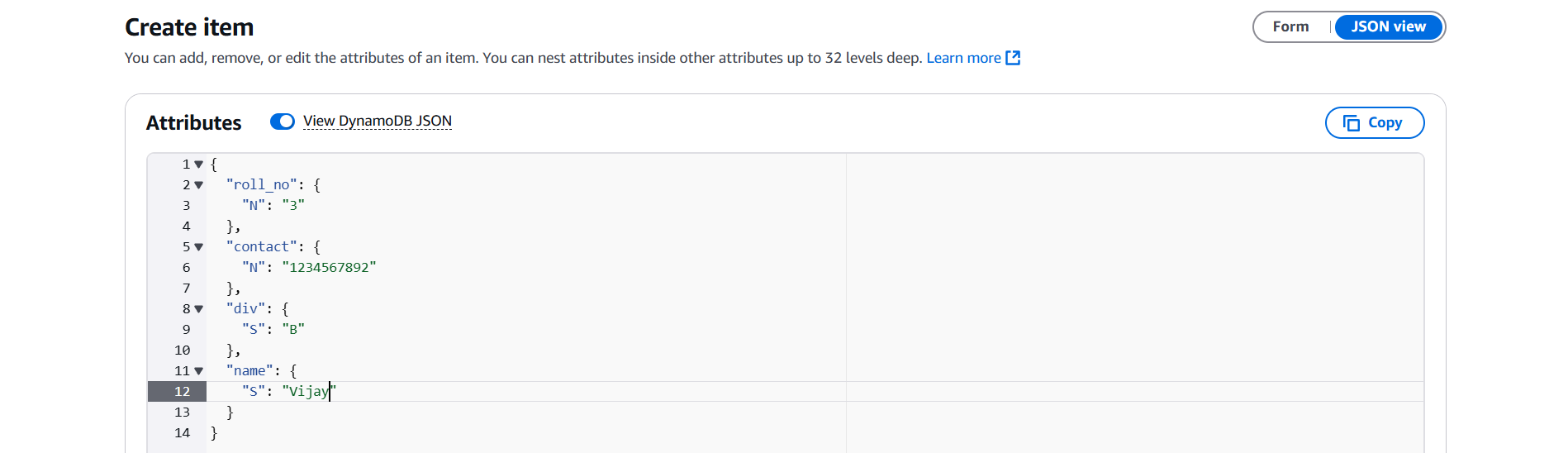
* Create an item as shown in image and note one thing by default we need to mention our first attribute (roll\_no) because its partition key which is compulsory.



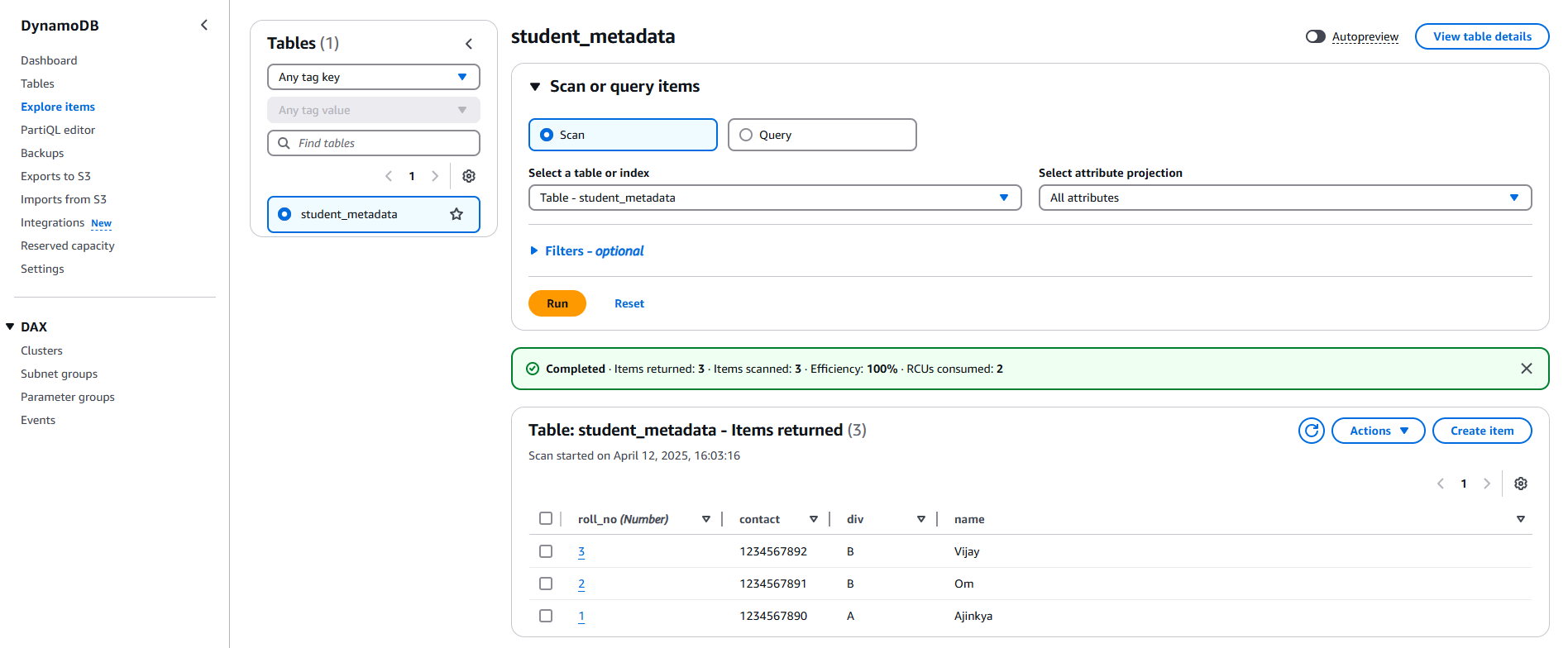
* Create a few items in this way.
* If we try to create another item with same partition key, it will return an error and the item will not be created.



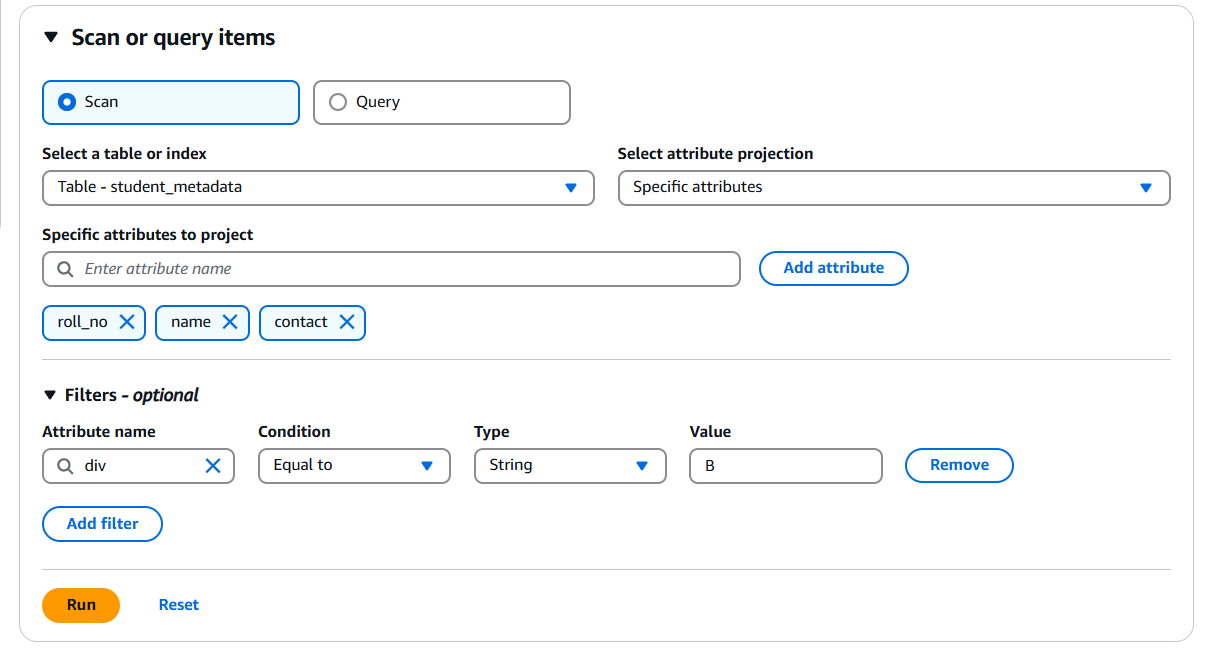
* We can create items using JSON too.

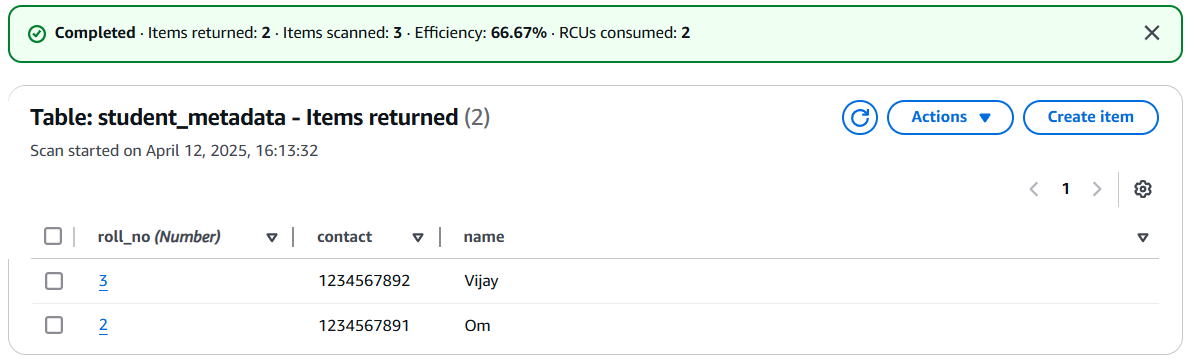


* To view the items in the table we can go to “Explore items” section from where we can see the items by refreshing the table details using the spinning arrow. While doing that, notice a thing, a notification flashes something like “**Completed** · Items returned: **3** · Items scanned: **3** · Efficiency: **100%** · RCUs consumed: **2**”.
* It states about RCUs consumed per read and items scanned.

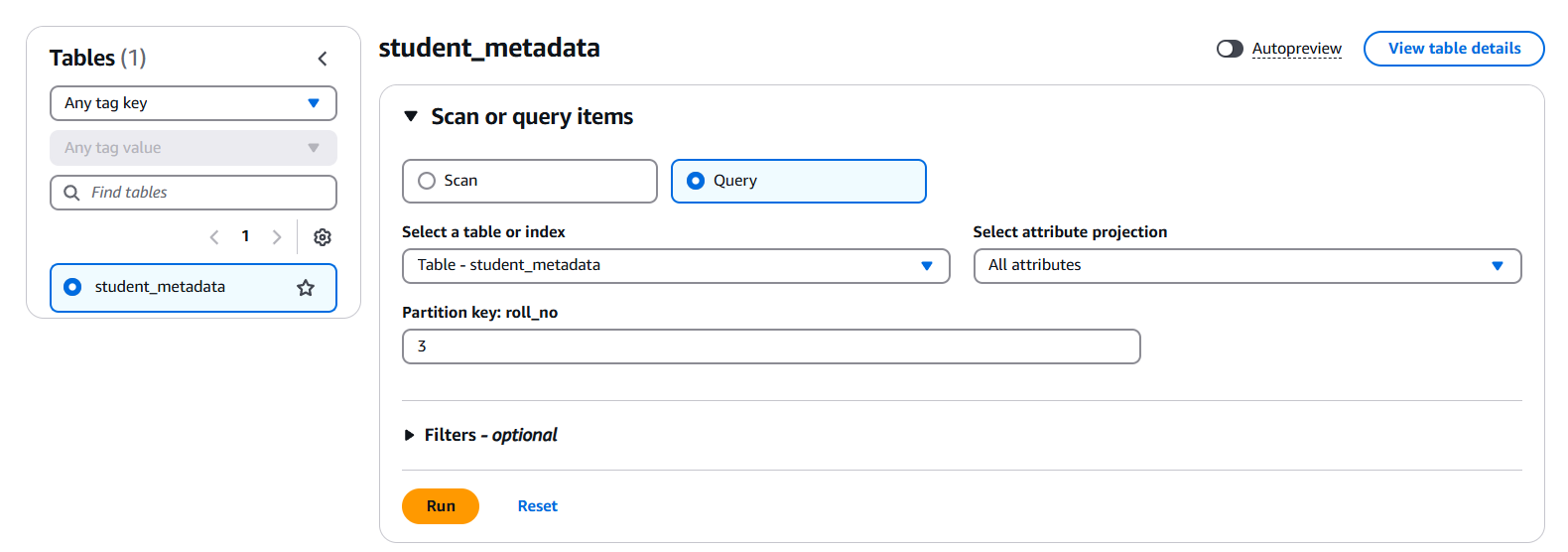


* The same thing of scanning items can be done with/without filters using the “Scan” option available there.
* We will scan for specific attributes (roll\_no, contact and name) with filter where div is B so it will return (by scanning) item details only with those attributes and “div” attribute won’t be displayed.

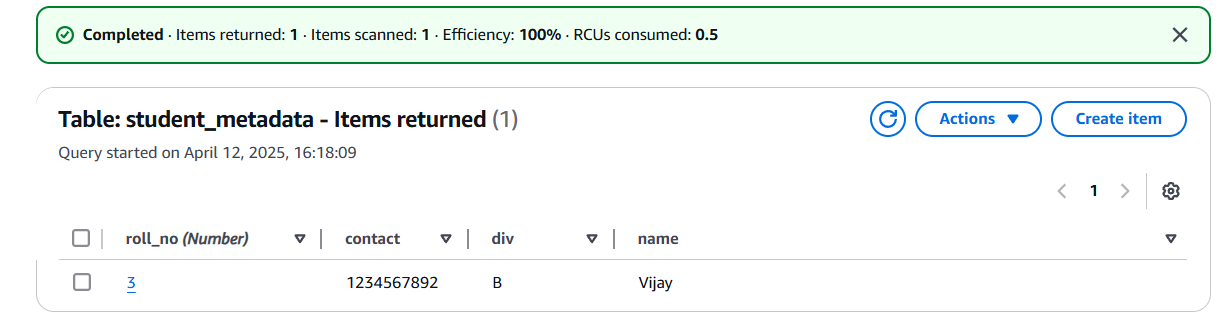




* We can search for items using query section and entering the value of partition key.



* It will return the item that has the key.



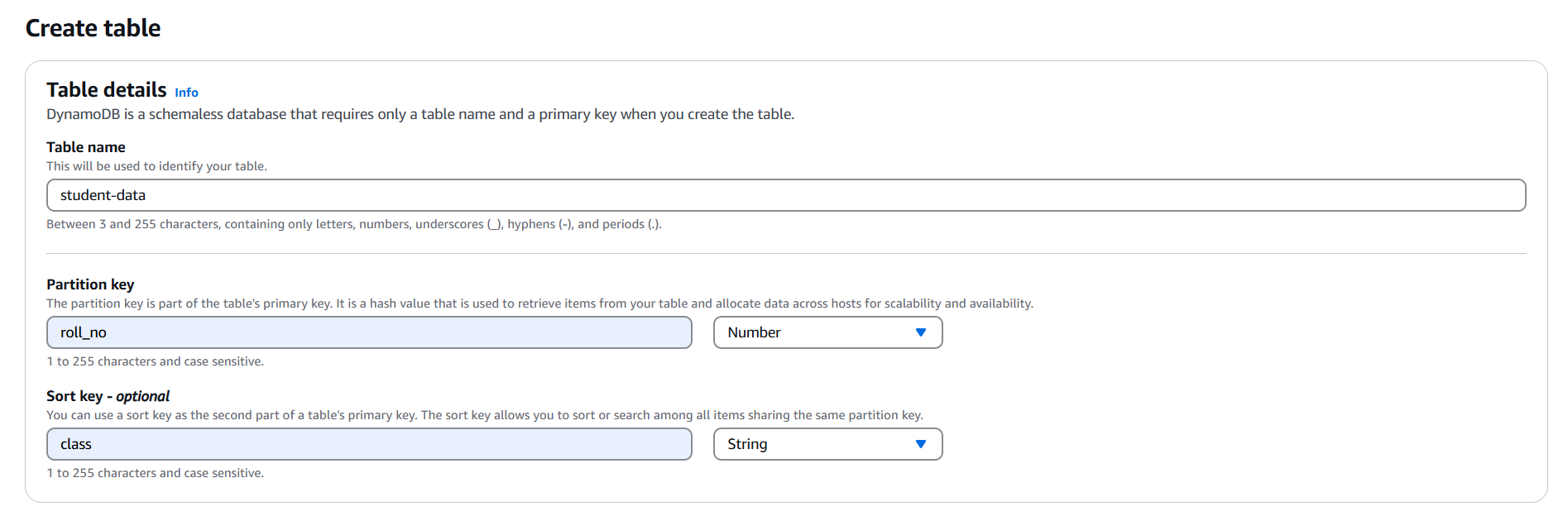
* We can run traditional queries in PartiQL editor to search for items too.

### Practical-2:

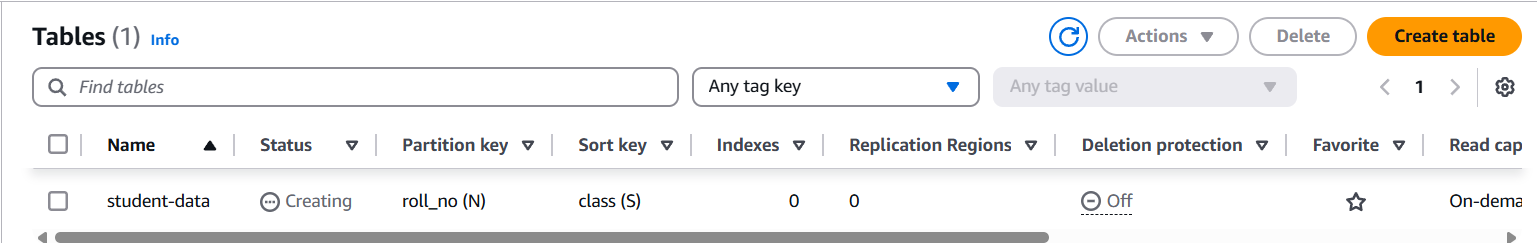
Create a table in DynamoDB with composite key (partition key + sort key).

It will allow to create items with same partition key but sort key and partition key cannot be same for both the items.

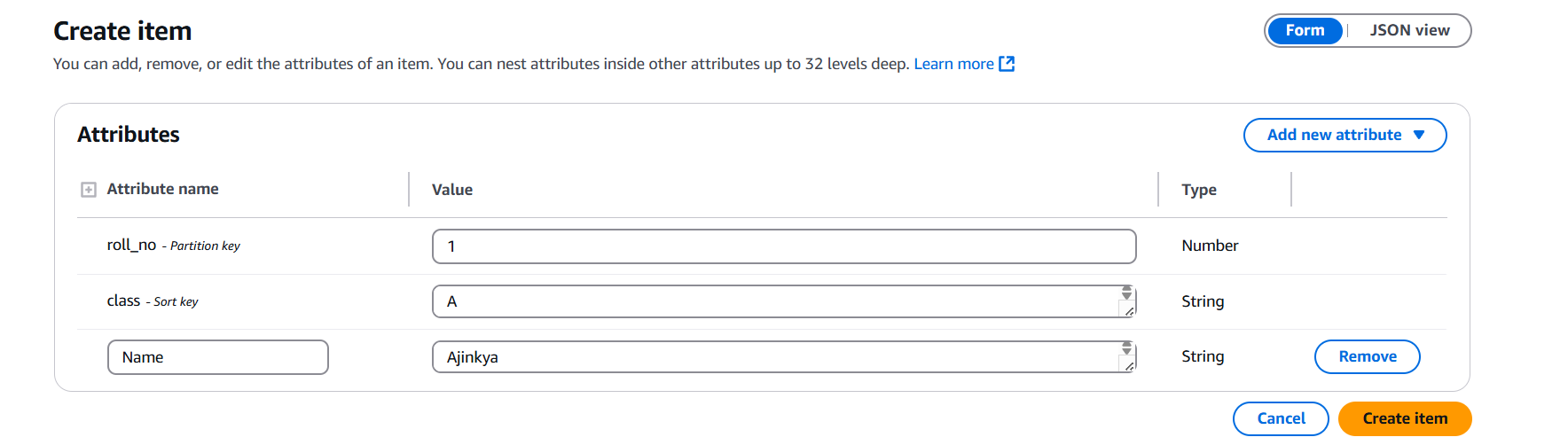
* This time while creating table, enter both partition key and sort key and create the table.



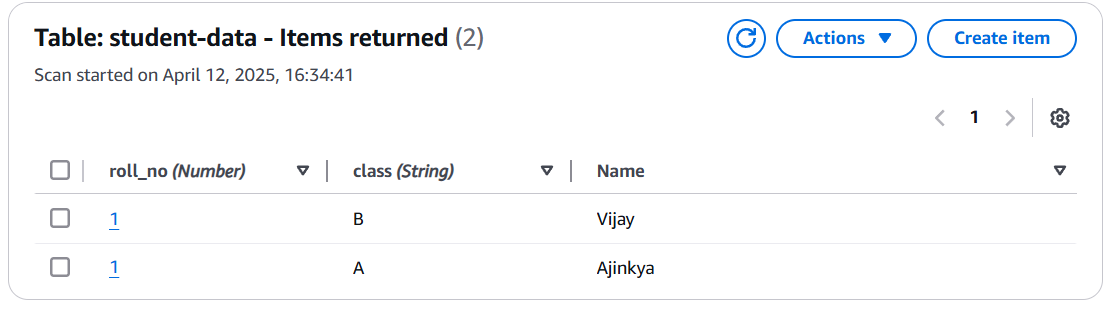
* On table list view als owe can see that sort key of string type (class (S) in my case) is mentioned.



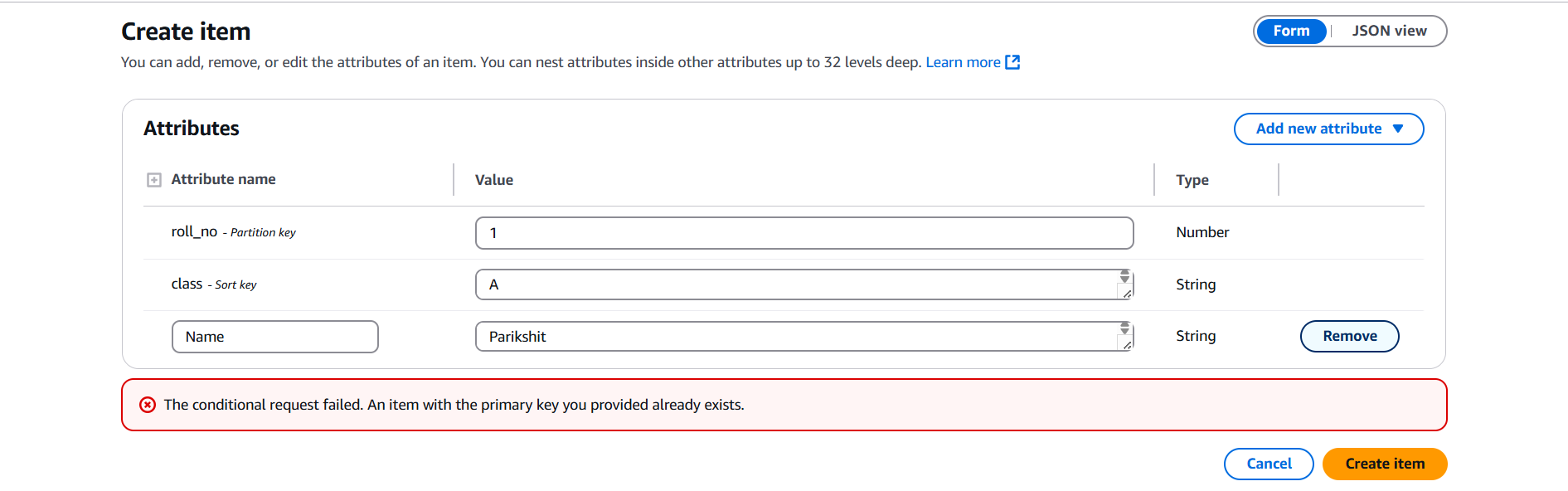
* Create an item and this time while creating the item two attributes will be visible by default as partition key and sort key.

****

* Try to create another item with same partition key but different sort key and it will get created.

****

* But if we now try to create item with both the values same then it will generate an error and item won’t be created.



### Use Cases of DynamoDB:

**1. User Profile Stores:**

* **Use Case**: Store user data like name, email, preferences, settings.
* **Why DynamoDB?**

Fast lookups, flexible schema, scalable for millions of users.

* **Example**: A social media app storing user profiles with UserId as the partition key.

**2. Session Management:**

* **Use Case**: Track active user sessions, login status, tokens.

**Why DynamoDB?**

Low-latency, TTL (Time-to-Live) support to auto-expire sessions.

* **Example**: A mobile app storing user sessions with TTL set to auto-delete after 1 hour.

**3. E-Commerce Product Catalog:**

* **Use Case**: Store and retrieve product info like price, stock, category.

**Why DynamoDB?**

Handles millions of products with fast filtering and access.

* **Example**: Category as Partition Key, ProductId as Sort Key to query all products in a category.

**4. Shopping Cart / Order History:**

* **Use Case**: Manage users' cart items or past orders.

**Why DynamoDB?**

Composite keys help store multiple orders/carts per user.

* **Example**: UserId as Partition Key, OrderId as Sort Key.

**5. Real-Time Leaderboards / Gaming:**

* **Use Case**: Track players’ scores, rankings, match history.

**Why DynamoDB?**

Fast reads/writes, sort by score or time.

* **Example**: GameId as Partition Key, PlayerScore as Sort Key to rank players.

and many more.

Amazon DynamoDB is a fully managed NoSQL database service offered by AWS that provides consistent performance, seamless scalability, and serverless architecture. In this document, we explored DynamoDB’s key features like auto-scaling, on-demand capacity, global tables, and fine-grained access control, all of which make it suitable for high-performance applications.

We examined the main components such as tables, items, attributes, and the importance of primary keys—both simple (partition key) and composite (partition + sort key). Through hands-on practicals, we created tables using both types of keys and understood how they control item uniqueness and querying behaviour.

Additionally, we explored DynamoDB's two capacity modes; Provisioned and On-Demand, highlighting when to use each based-on workload patterns

This practical and conceptual understanding builds a strong foundation for leveraging DynamoDB in real-world cloud-based applications.

♣♣♣