**day29\_107856406\_dsdipt\_sudipto\_21august2025**

**Employee Code:** 107856406

**Login ID:** dsdipt

**Email :** dsdipt@amazon.com

**Name:** Sudipto Das

**Date:** 21 August 2025 (Day 29)

### ***Task 001: Dynamo DB***

**What is DynamoDB?**

Amazon **DynamoDB** is a **serverless NoSQL database** provided by AWS.  
 Think of it like a **super-scalable key-value + document store** that’s fully managed (you don’t worry about servers, scaling, or patching).

**Key Features**

* **Serverless** → no need to provision servers, AWS auto-scales.
* **NoSQL** → schema-less, flexible document + key-value model.
* **High availability** → data is automatically replicated across multiple AZs.
* **Performance** → single-digit millisecond latency.
* **Integration** → works smoothly with Lambda, API Gateway, S3, etc.

**Core Concepts**

1. **Table** → container for your data (like an SQL table but schema-free).
2. **Item** → one row of data (like a record/document).
3. **Attribute** → field/column inside an item.
4. **Primary Key** → must-have for every item.  
   * Partition Key (hash key)
   * Partition Key + Sort Key (composite key)
5. **Provisioned Throughput** (R/W capacity) or **On-Demand** → how you pay for reads/writes.
6. **Indexes** →  
   * Global Secondary Index (GSI) → query with a different key.
   * Local Secondary Index (LSI) → alternate sort key but same partition key.

**CRUD in DynamoDB**

* **Create** → PutItem
* **Read** → GetItem / Query / Scan
* **Update** → UpdateItem
* **Delete** → DeleteItem

**Example Table**

Imagine a table called **Users**:

* **Partition Key**: UserId (String)
* **Attributes**: Name, Email, CreatedAt

**Example item:**

{

"UserId": "U123",

"Name": "Ankit",

"Email": "ankit@example.com",

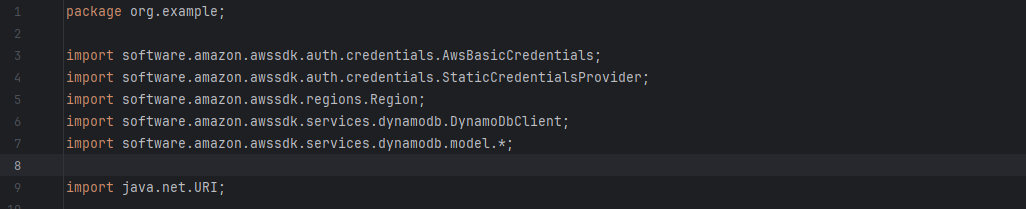
"CreatedAt": "2025-08-22"

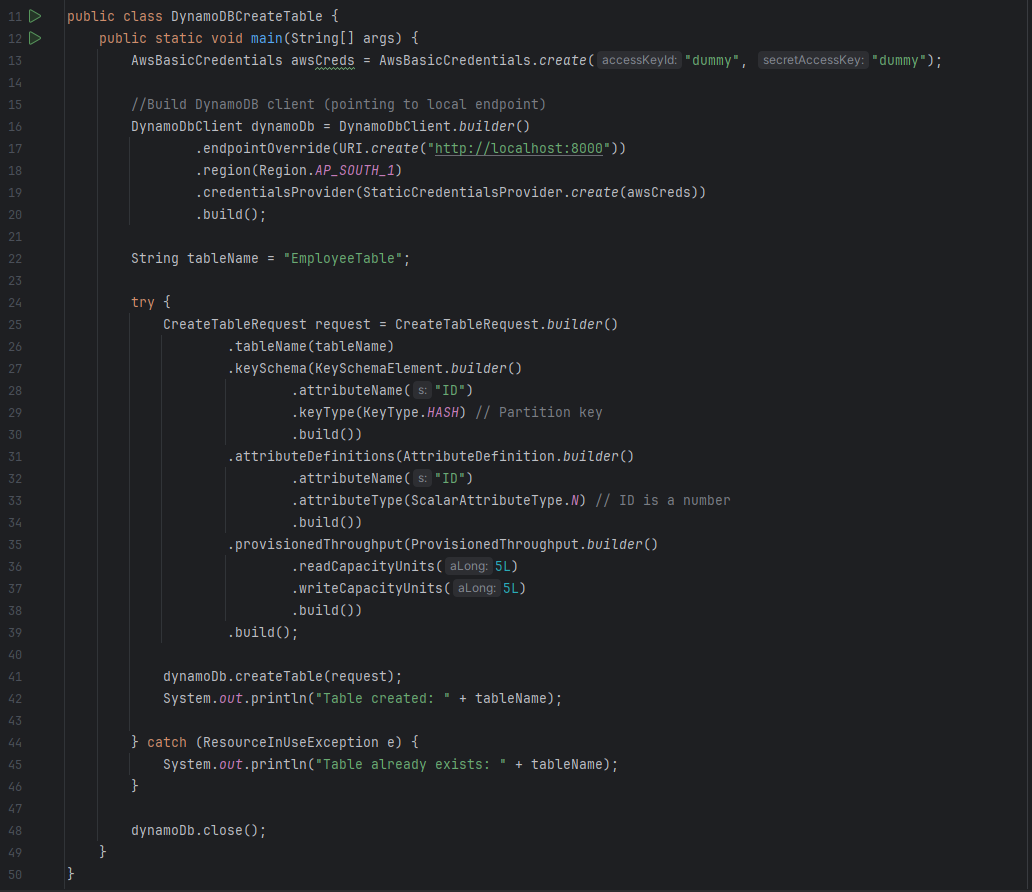
}

**SQL vs DynamoDB**

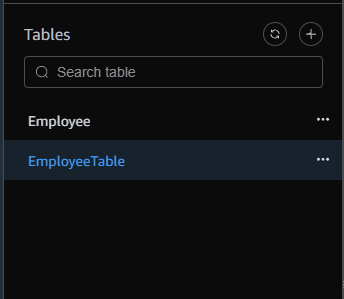
* **Schema**
  + SQL → Fixed schema (tables must follow predefined structure).
  + DynamoDB → Schema-less (flexible, you can add attributes on the fly).
* **Data Model**
  + SQL → Tables → Rows → Columns.
  + DynamoDB → Tables → Items → Attributes.
* **Primary Key**
  + SQL → Typically an ID column, sometimes composite.
  + DynamoDB → Partition key (mandatory) + optional sort key.
* **Relationships**
  + SQL → Supports JOINs (1-to-1, 1-to-many, many-to-many).
  + DynamoDB → No JOINs (you denormalize or use indexes).
* **Queries**
  + SQL → Rich querying (joins, filters, aggregations).
  + DynamoDB → Key-based queries (GetItem, Query, Scan) + Index queries.
* **Scaling**
  + SQL → Vertical scaling (add more power to one machine).
  + DynamoDB → Horizontal scaling (auto-distributes data across partitions).
* **Transactions**
  + SQL → Strong ACID transactions by default.
  + DynamoDB → Supports ACID transactions but with some limits.
* **Storage**
  + SQL → Rows stored in fixed schema format.
  + DynamoDB → Items can have flexible attributes (like JSON).
* **Use Cases**
  + SQL → Great for relational, structured data (banking, ERP, HR systems).
  + DynamoDB → Great for fast, scalable apps (IoT, gaming, e-commerce, serverless apps).

### ***Task 002: DynamoDB Table Creation***









### ***Task 003: Load and Retrieve Data from DynamoDB***

