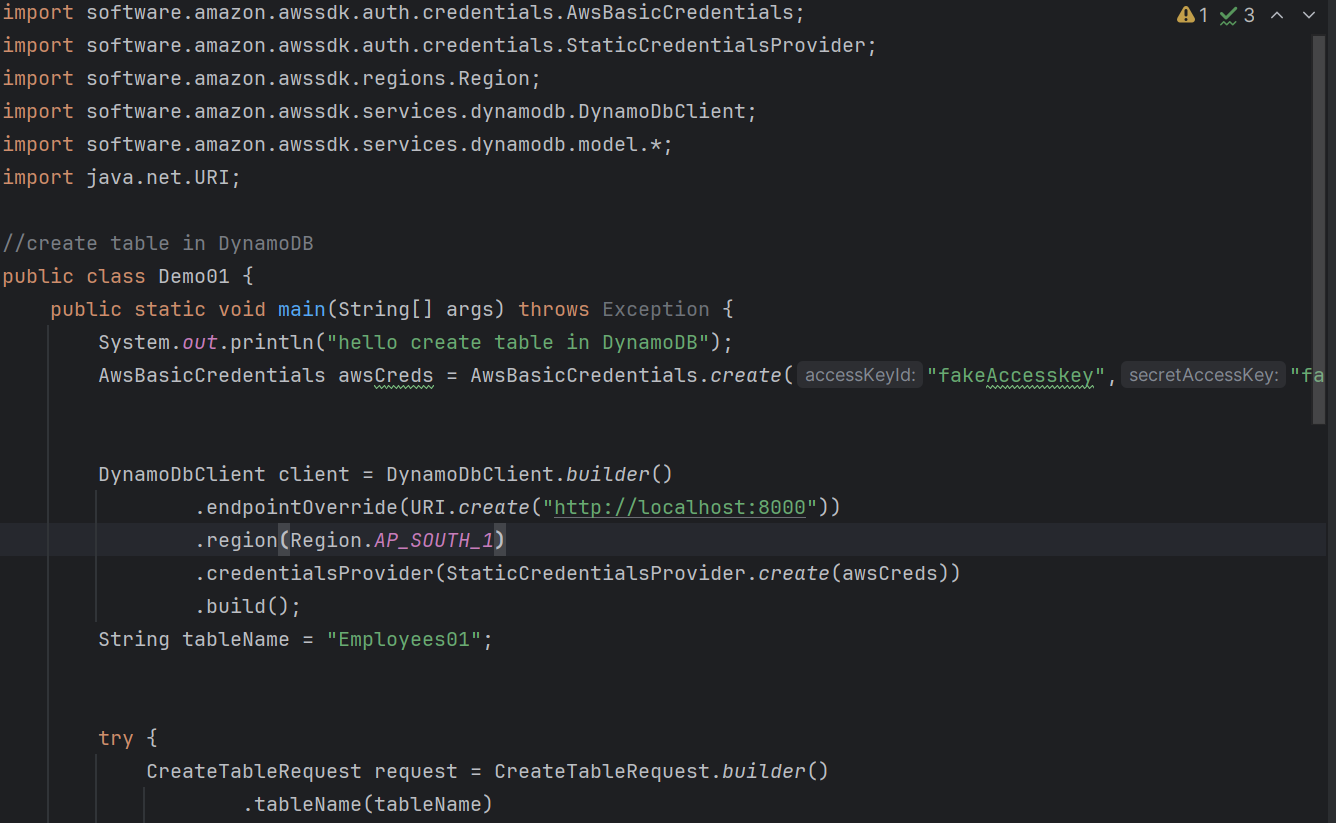
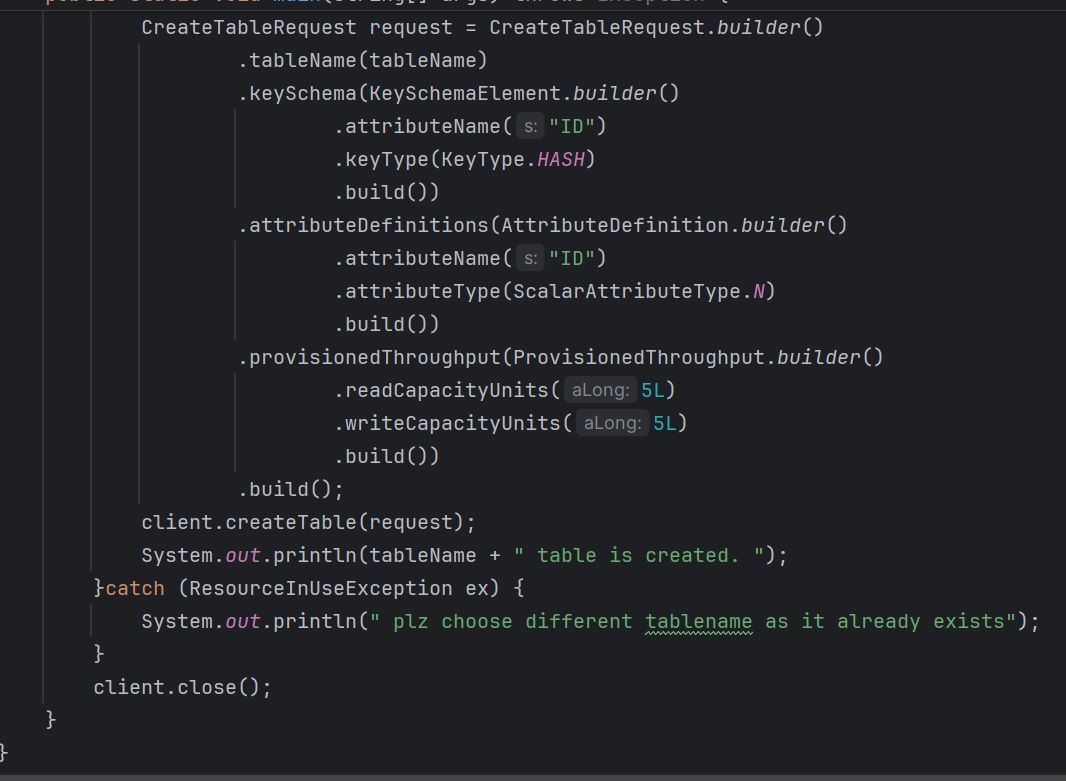
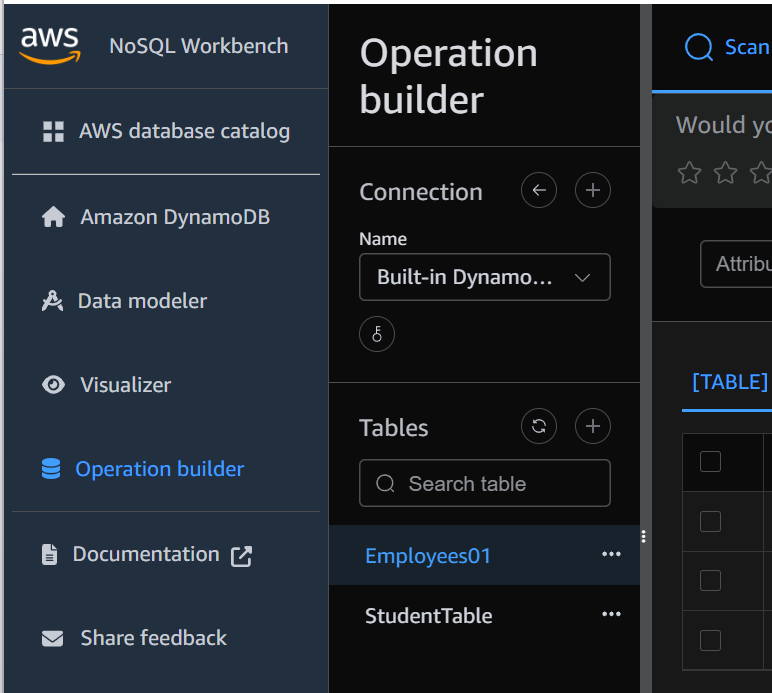
**Day 32 – September 12th**

**TASK 1 - Create a table using Java code and check if the table is created.**

****

****



**Output**:

hello create table in DynamoDB

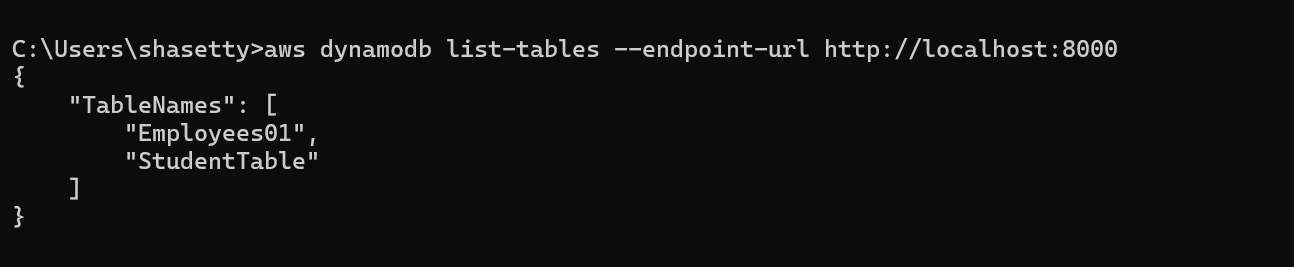
SLF4J(W): No SLF4J providers were found.

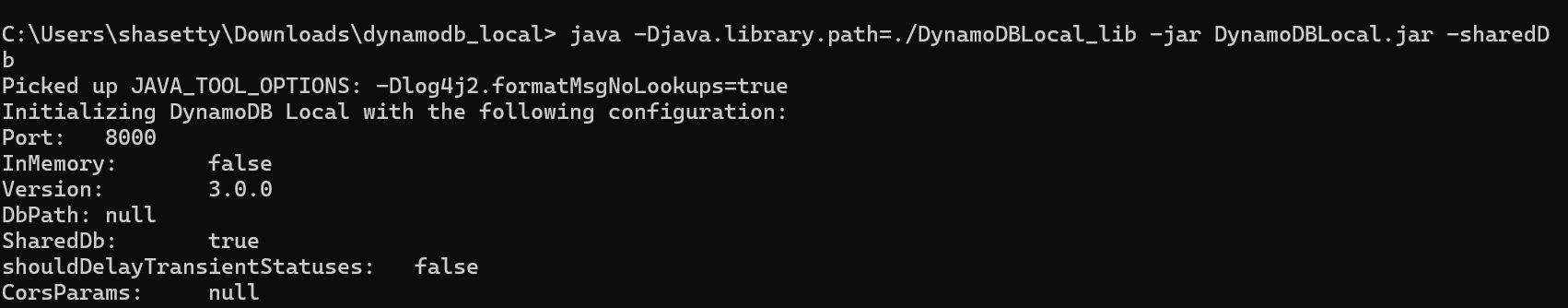
SLF4J(W): Defaulting to no-operation (NOP) logger implementation

SLF4J(W): See https://www.slf4j.org/codes.html#noProviders for further details.

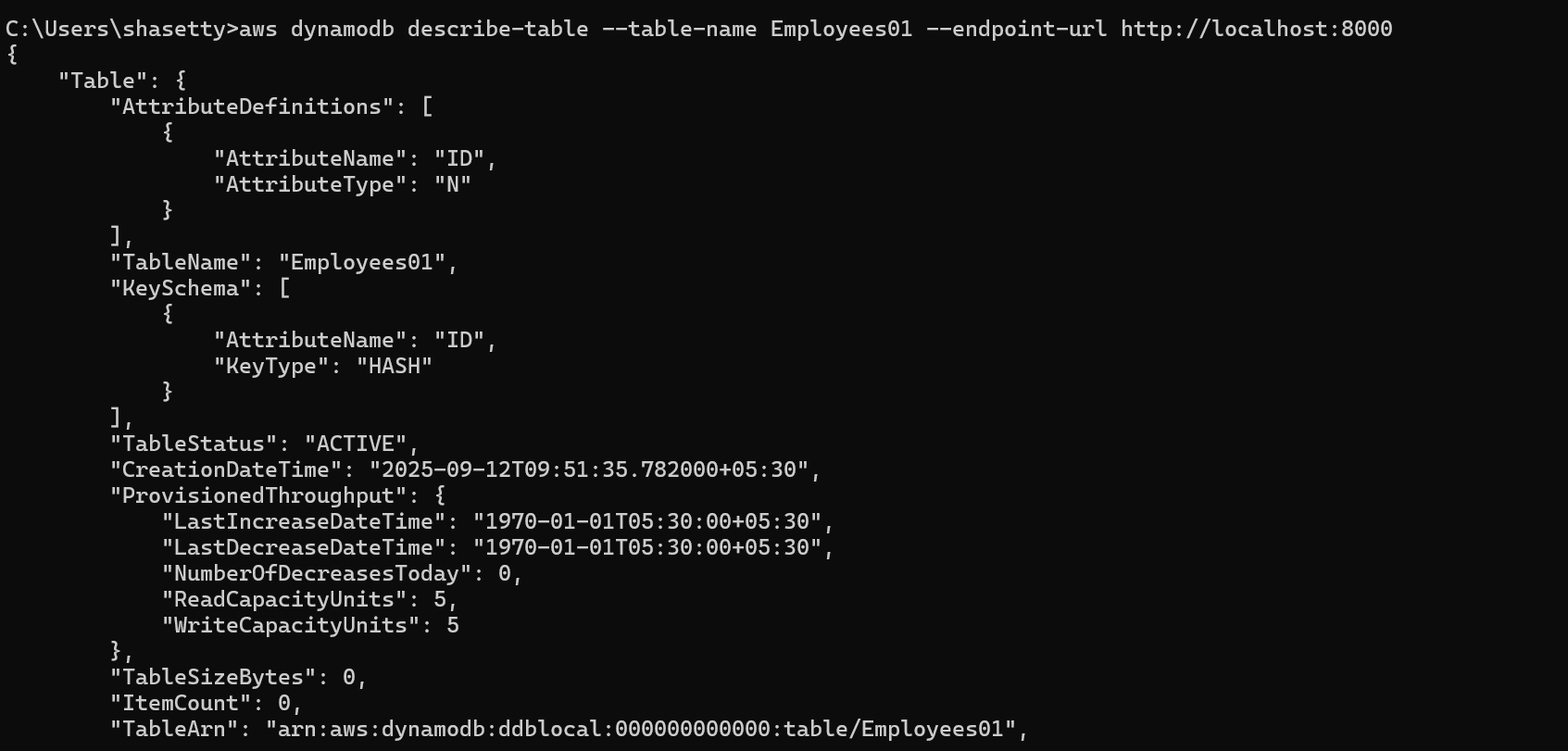
Employees01 Table is created

**TASK 2 – List of tables**

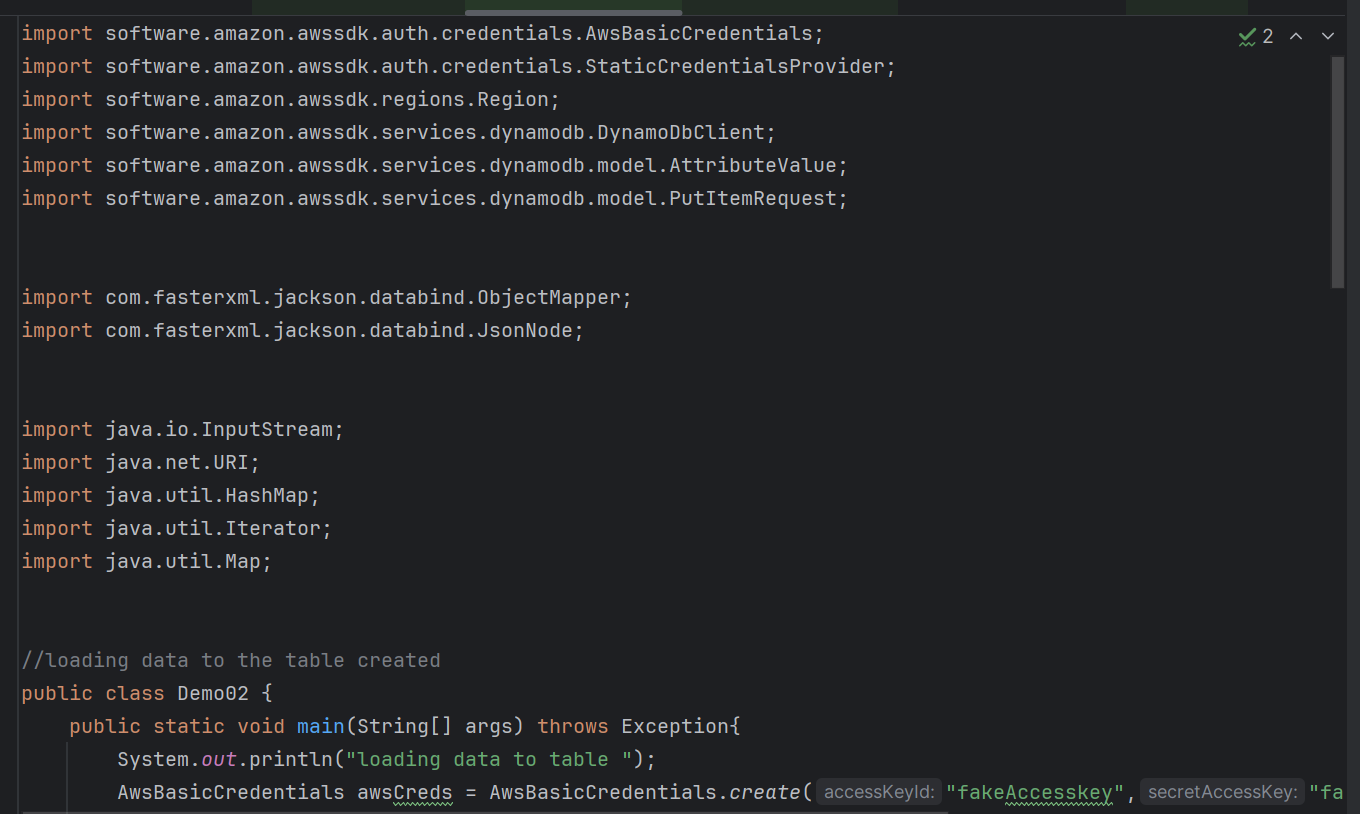
****

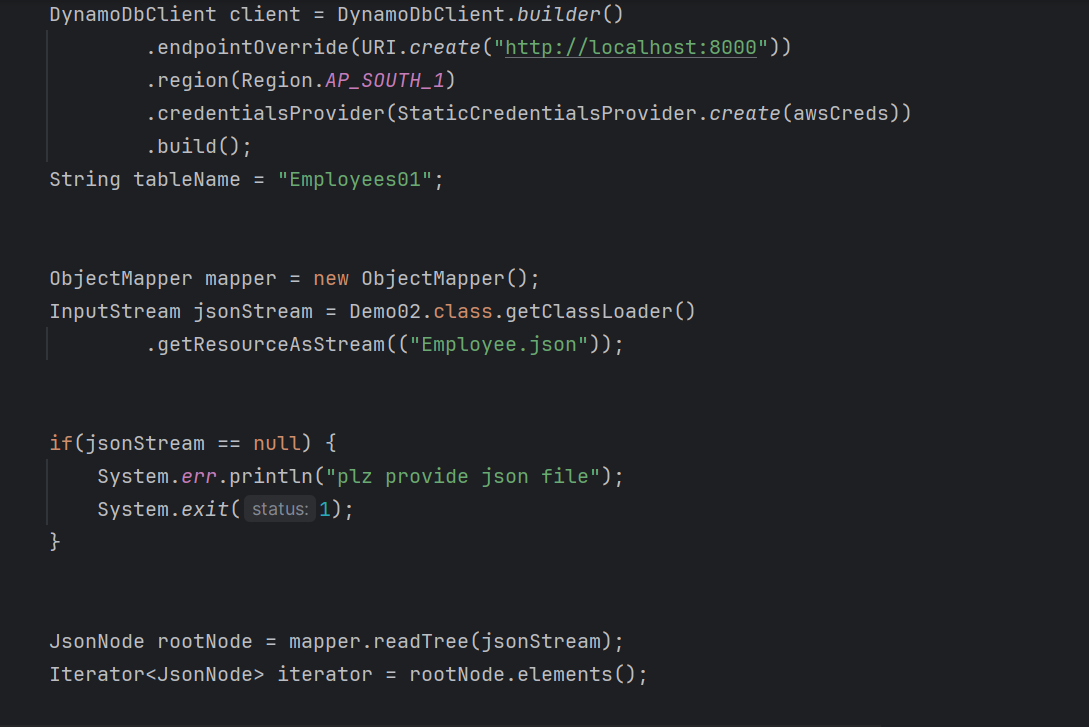
****

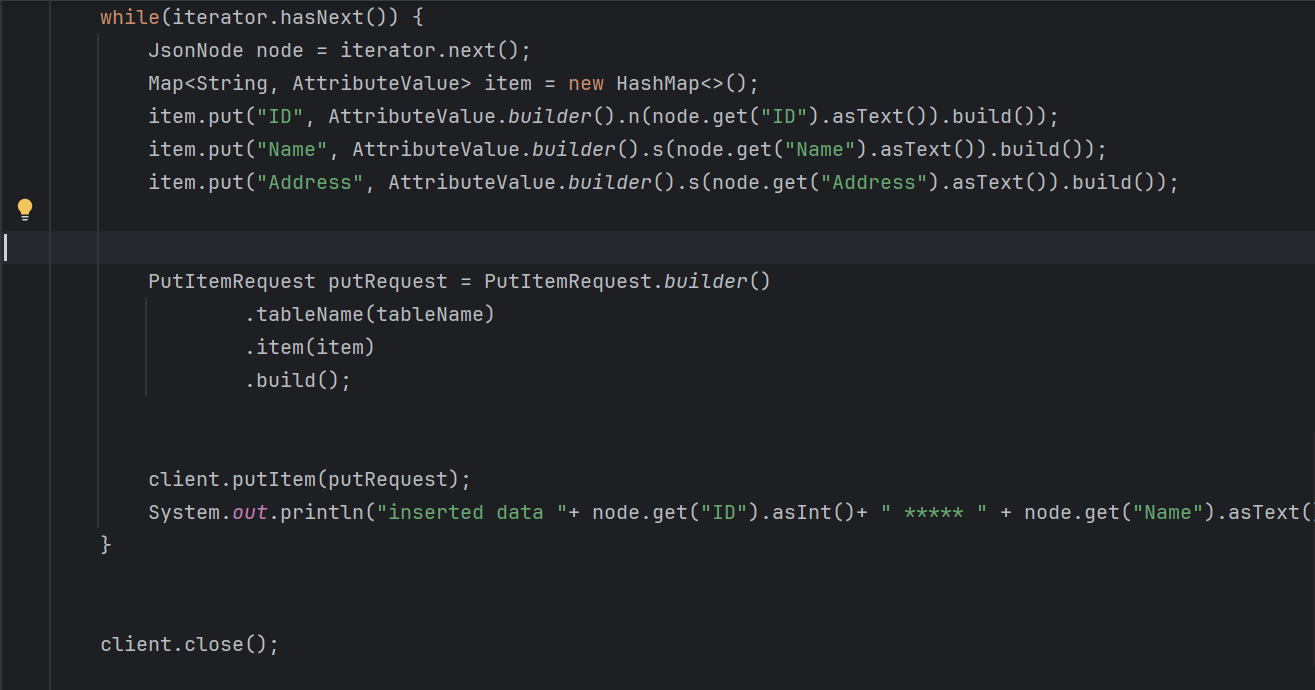
**TASK 3 – Description of the table**

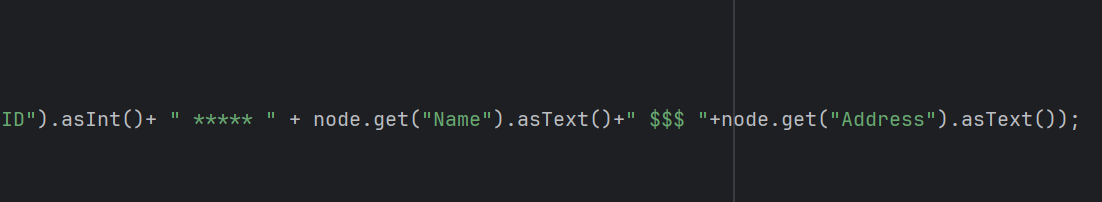
****

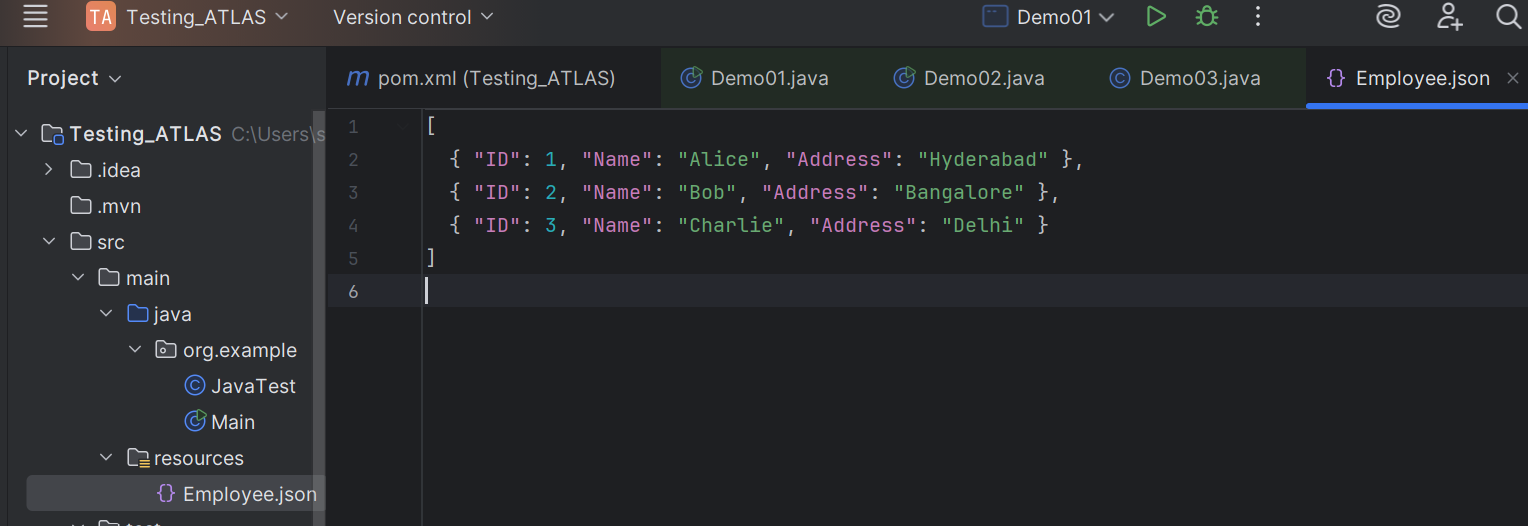
**TASK 4 – Loading/Inserting the date into the table**

****

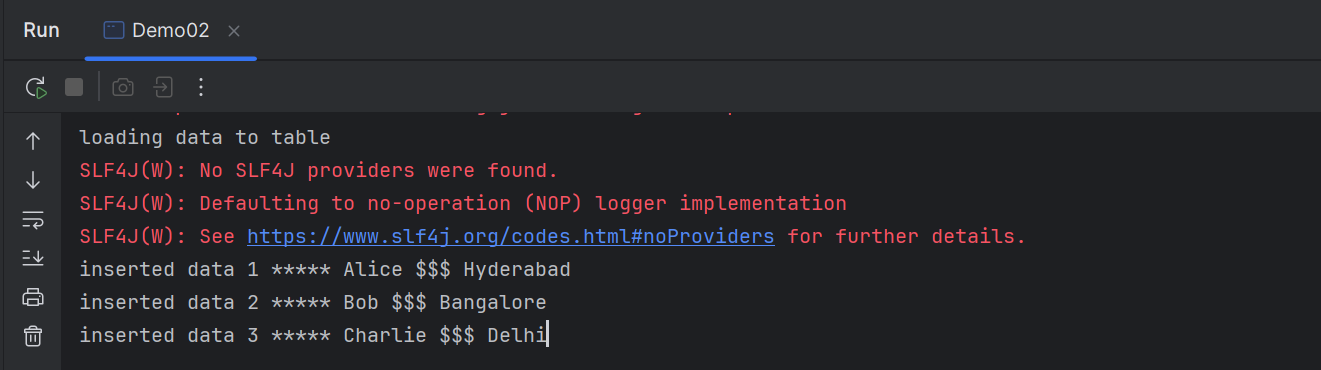
****

****

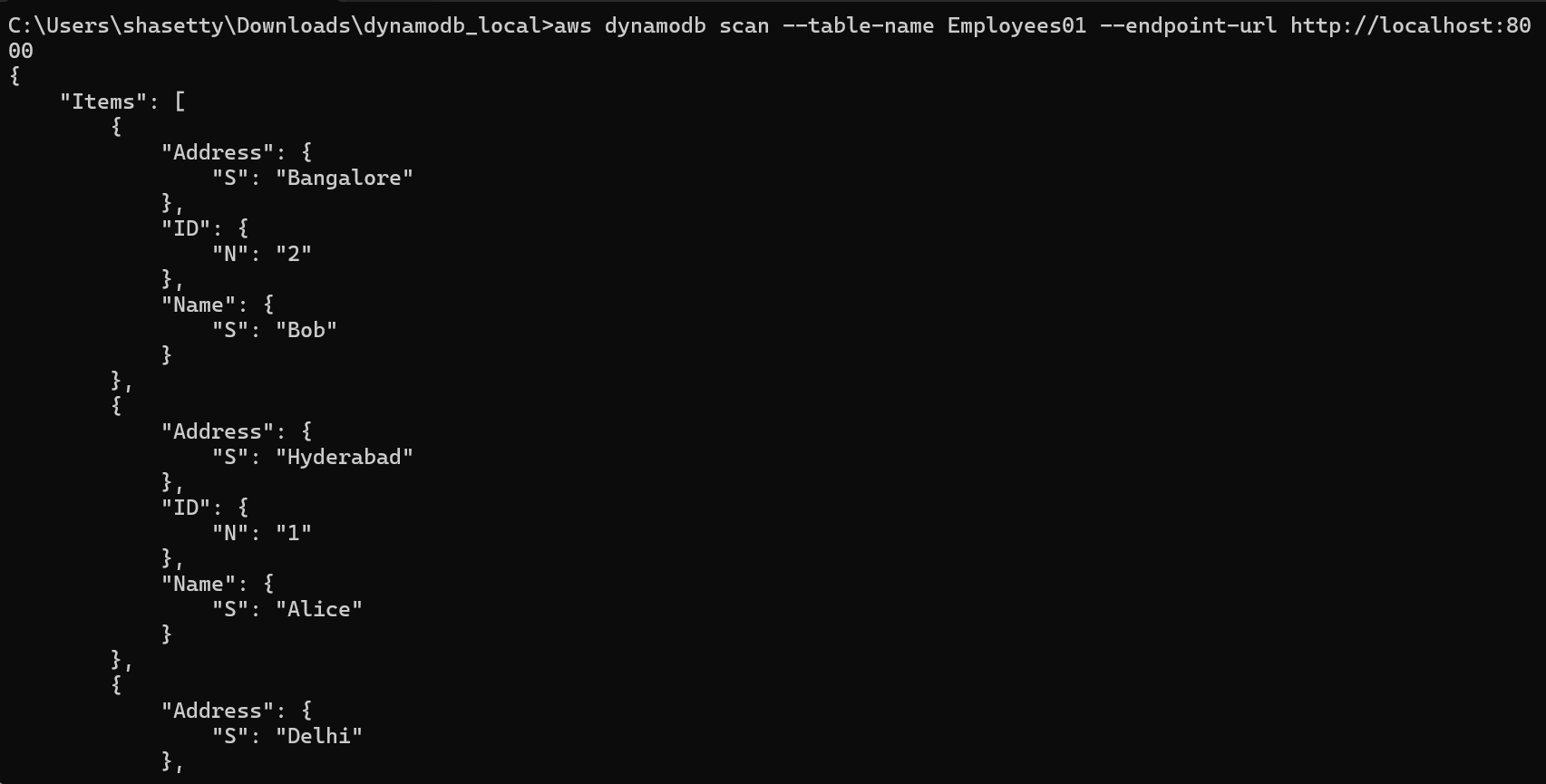
****

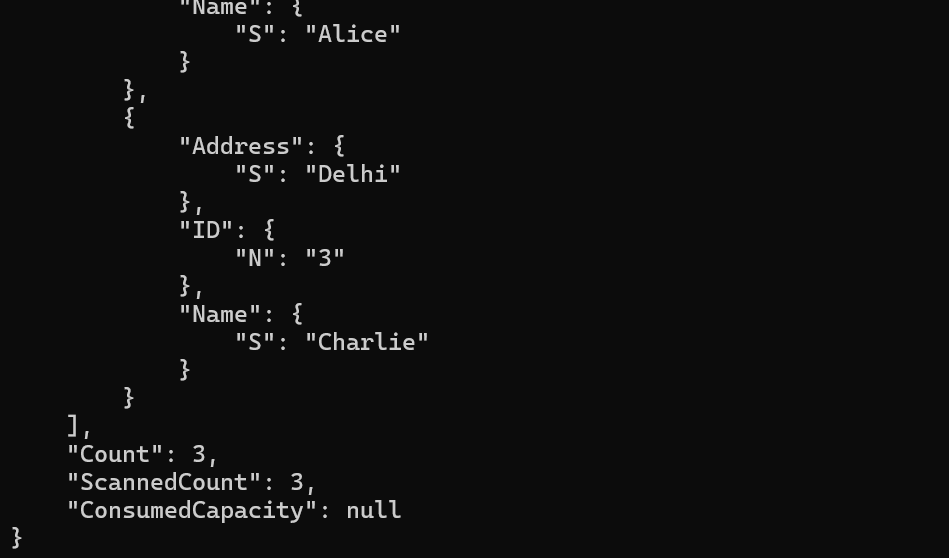
****

**Output**

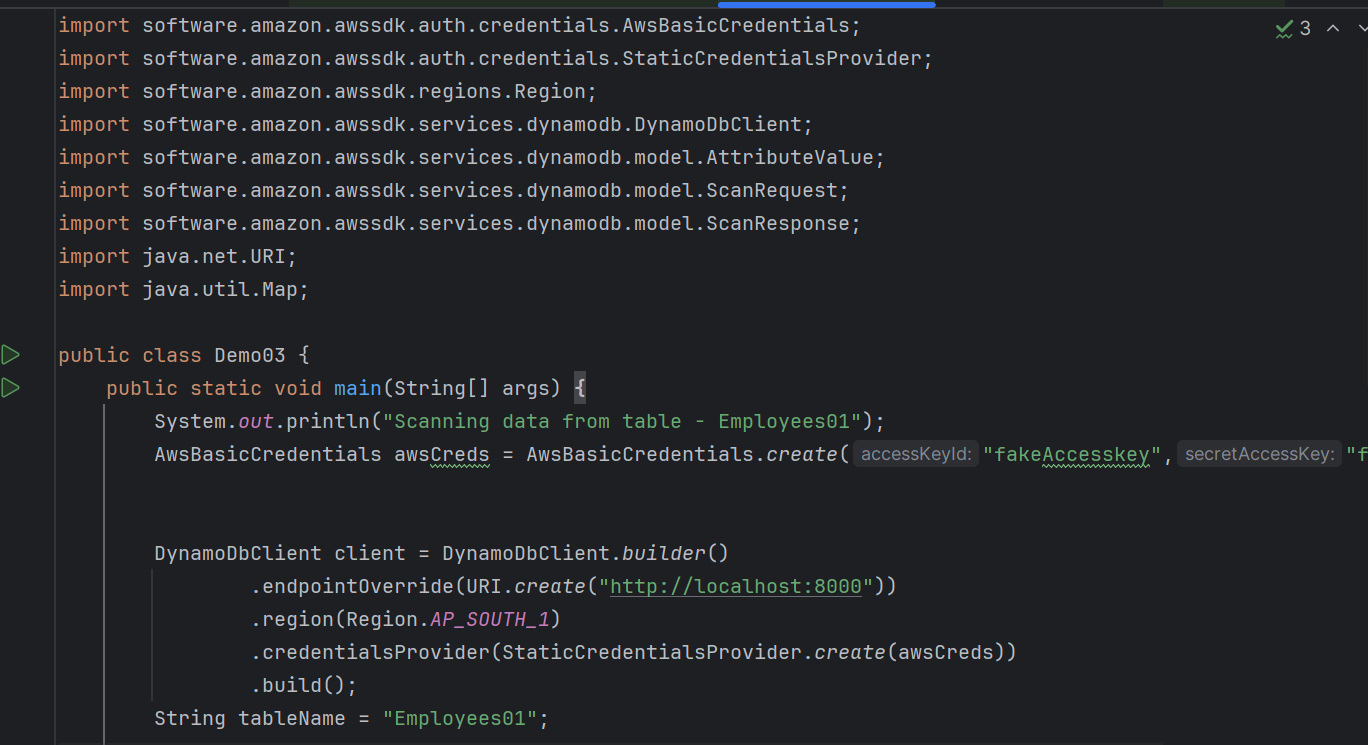
****

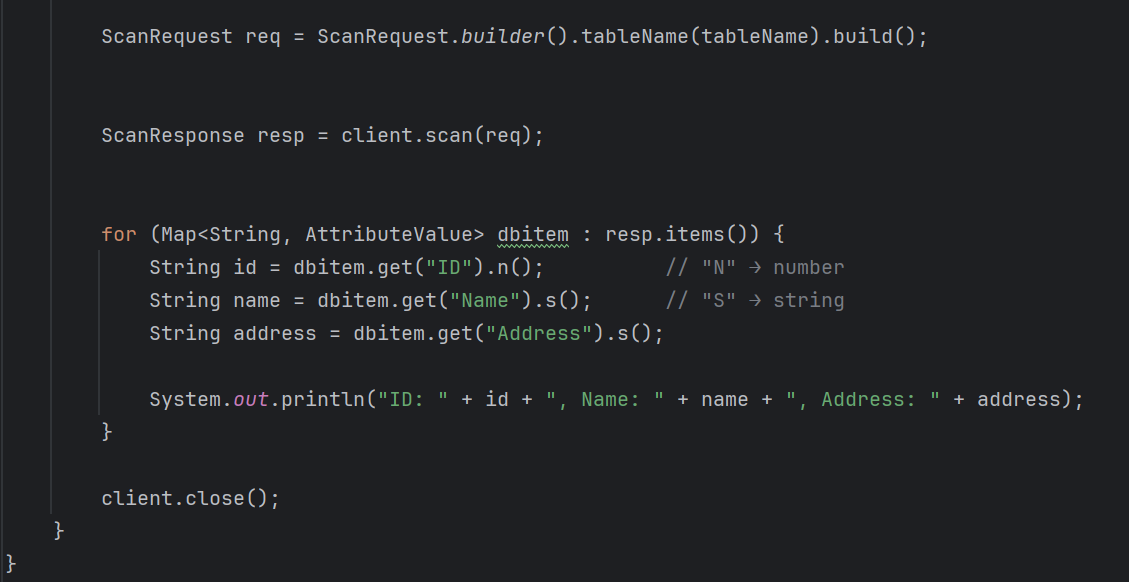
**Viewing the content**

****

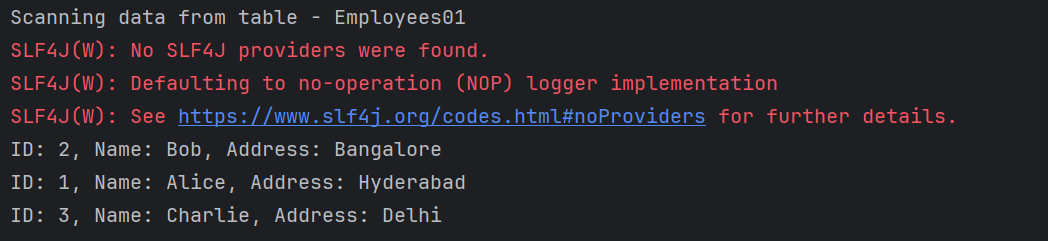
****

**TASK 5 – Scanning the table**

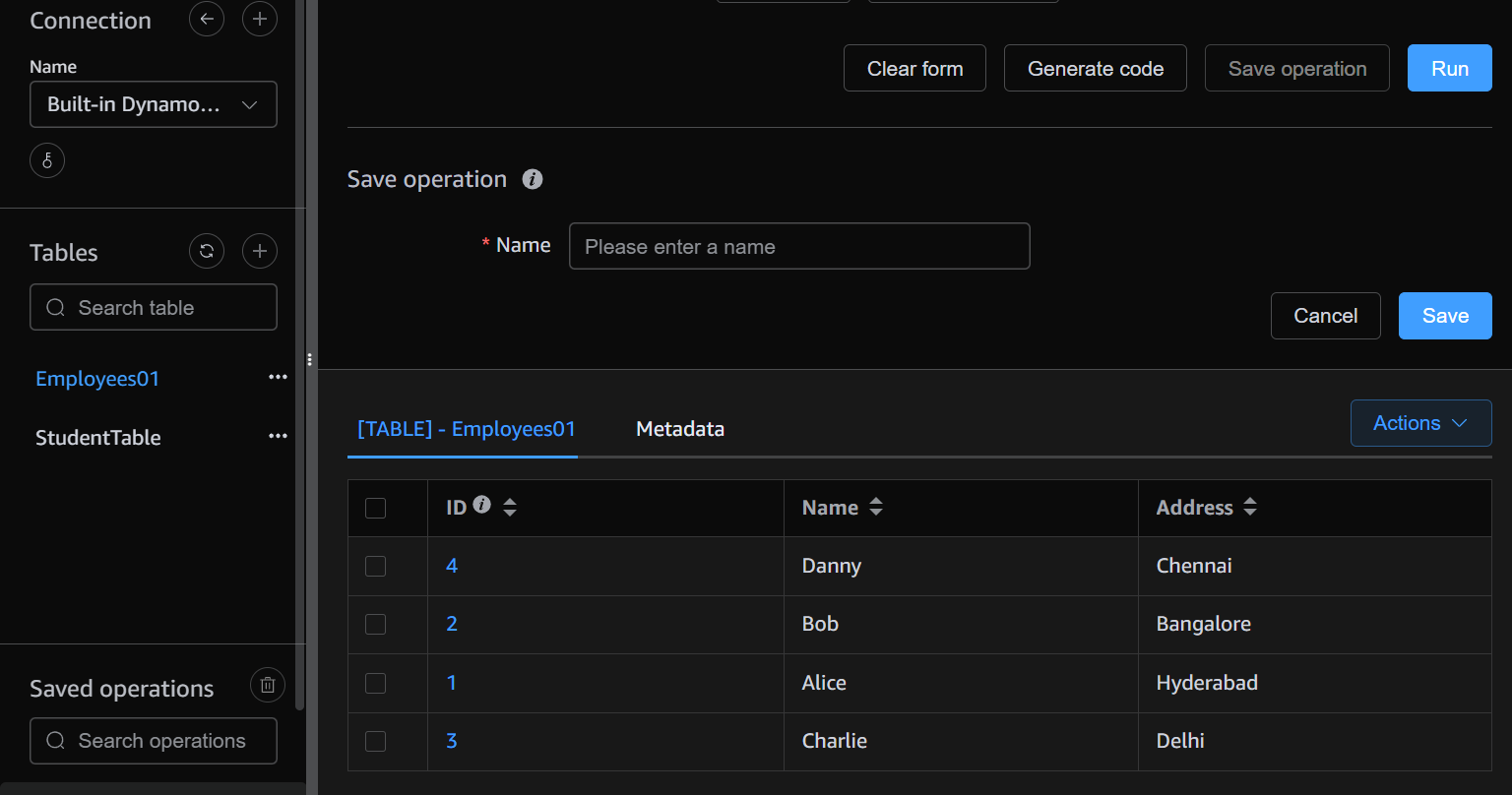
****

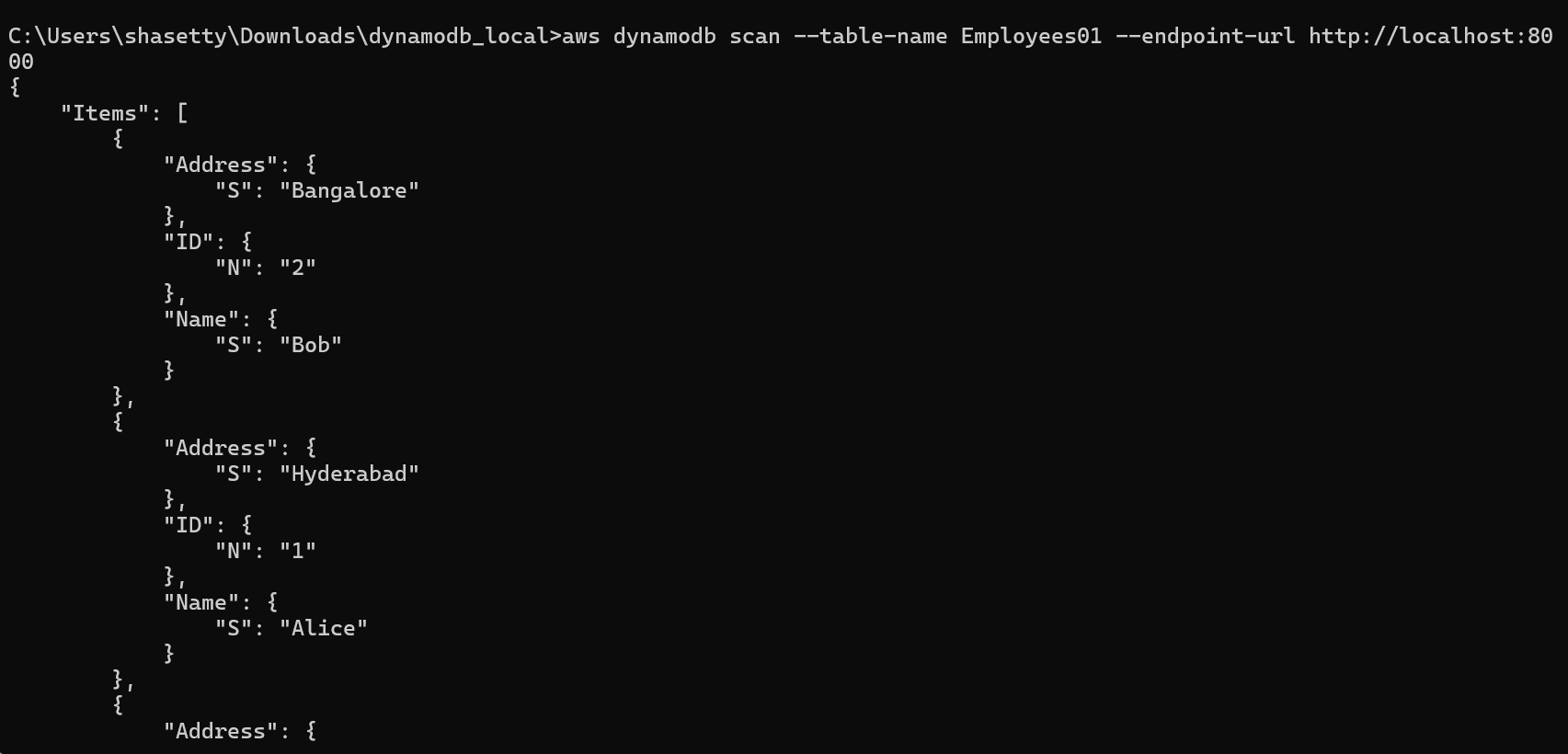
****

**Output**

****

**TASK 6 – manually inserted Items**

****

****

****

**TASK 7 – How do you lower the cost of DynomaDB.explain ways to do so**

**Choose the Right Capacity Mode**

* Use On-Demand mode if traffic is unpredictable.
* Use Provisioned mode with Auto Scaling if traffic is predictable.
* This avoids over-provisioning and saves unused RCU/WCUs.

**Use DynamoDB Standard-IA (Infrequent Access) Storage Class**

* Move rarely accessed data into Standard-IA to save up to 60% on storage.

**Optimize Data Model & Item Size**

* Keep items small (avoid unnecessary attributes).
* Store large files (images, PDFs, logs) in S3, not DynamoDB.
* Use composite keys to reduce redundant storage.

**Optimize Queries**

* Prefer GetItem (cheapest) over Scan.
* Use Query with keys instead of scanning entire tables.
* Use ProjectionExpression to fetch only required attributes.

**Avoid Hot Partitions**

* Design high-cardinality partition keys to spread load evenly.
* Prevents throttling and avoids having to provision extra capacity.

**Add Caching Layer**

* Use DAX (DynamoDB Accelerator) or ElastiCache for frequent reads.
* Reduces read costs by serving results from memory.

**Monitor & Adjust**

* Track usage in CloudWatch metrics.
* Lower provisioned capacity if consistently underutilized.
* Use Contributor Insights to identify heavy keys.

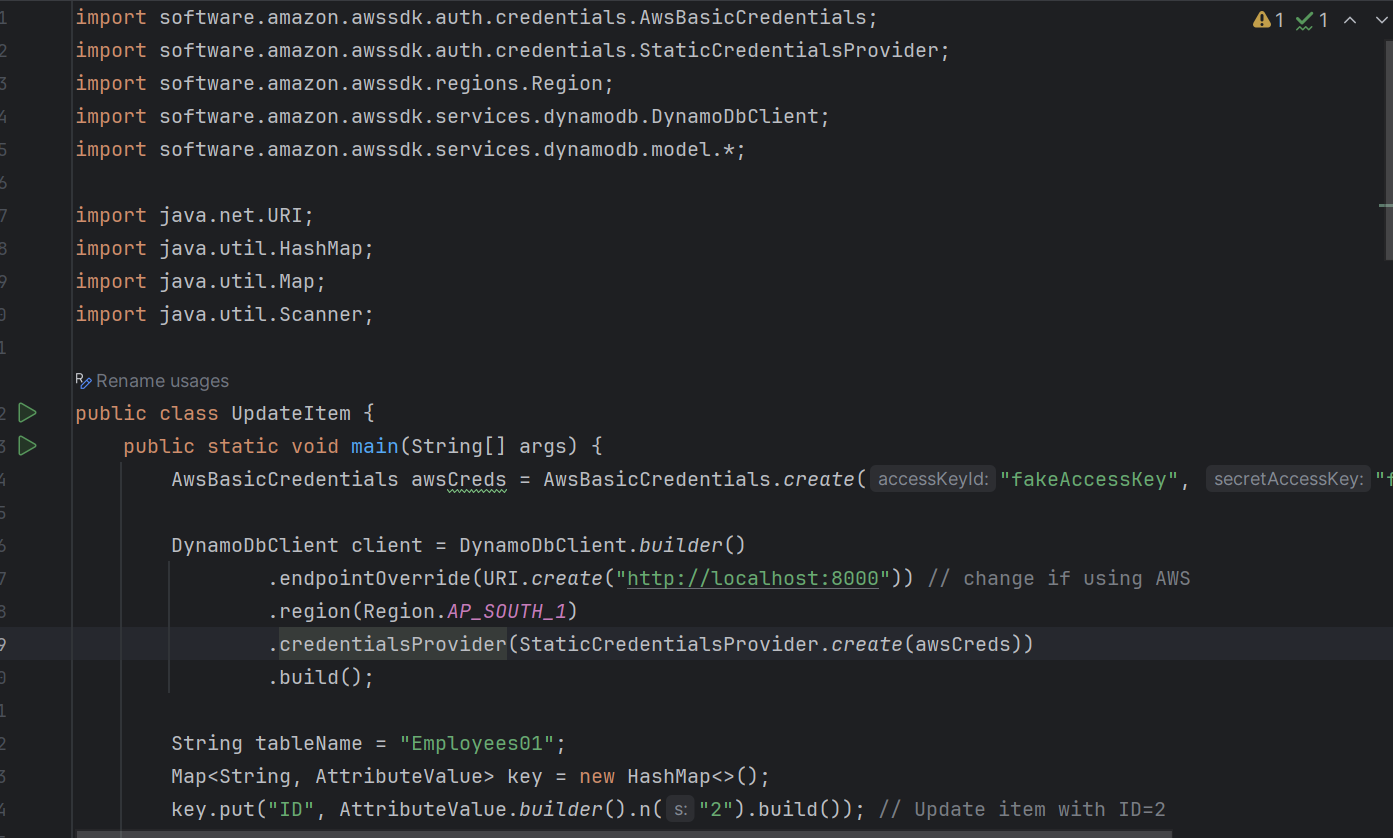
**Reserved Capacity**

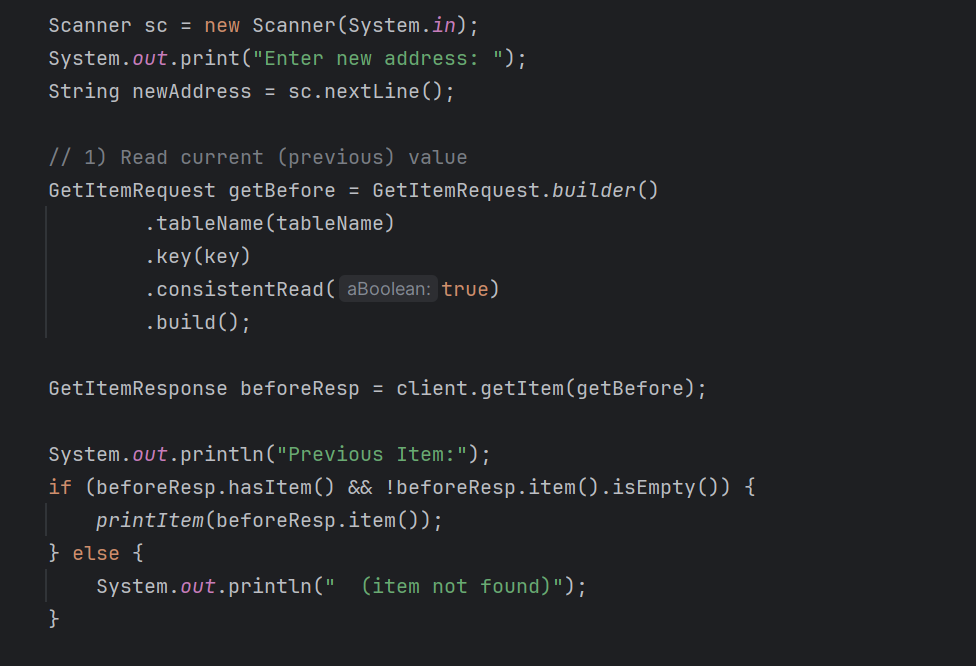
* For steady workloads, buy Reserved Capacity (1- or 3-year) → up to 75% cheaper.

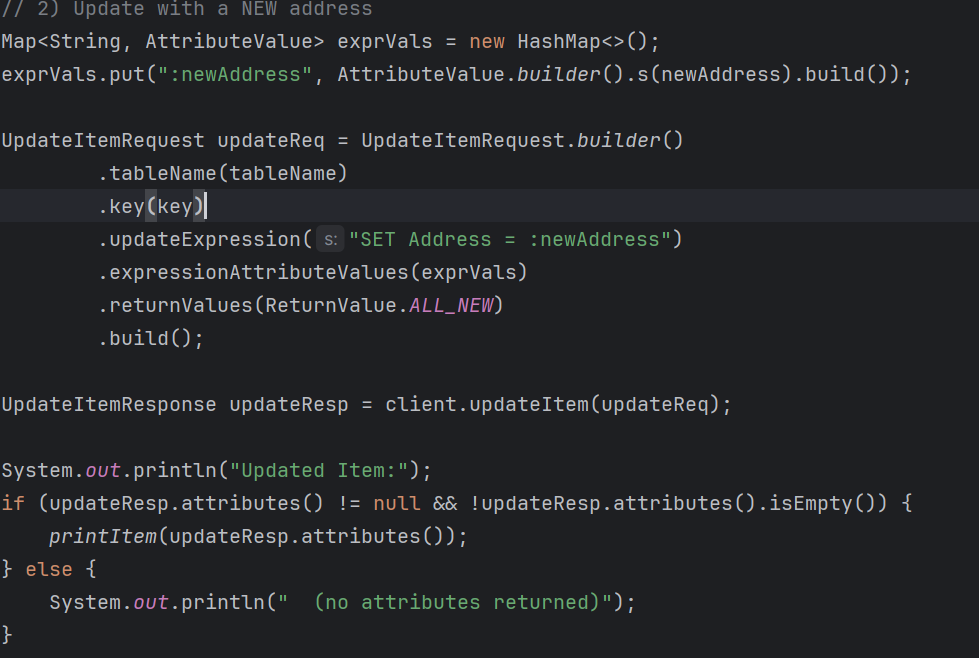
**Use Free Tier (if eligible)**

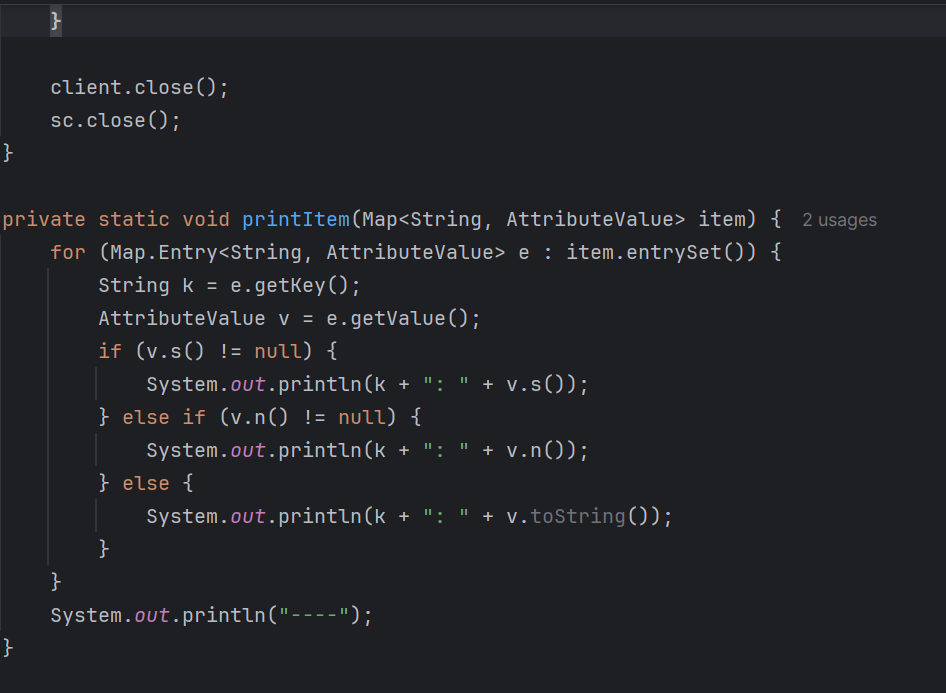
* 25 GB storage + 25 RCUs & 25 WCUs free each month.

**TASK 8 – Update Item**

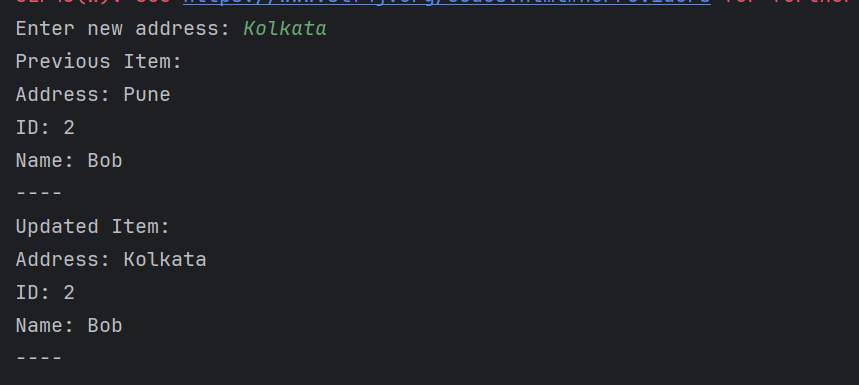
****

****

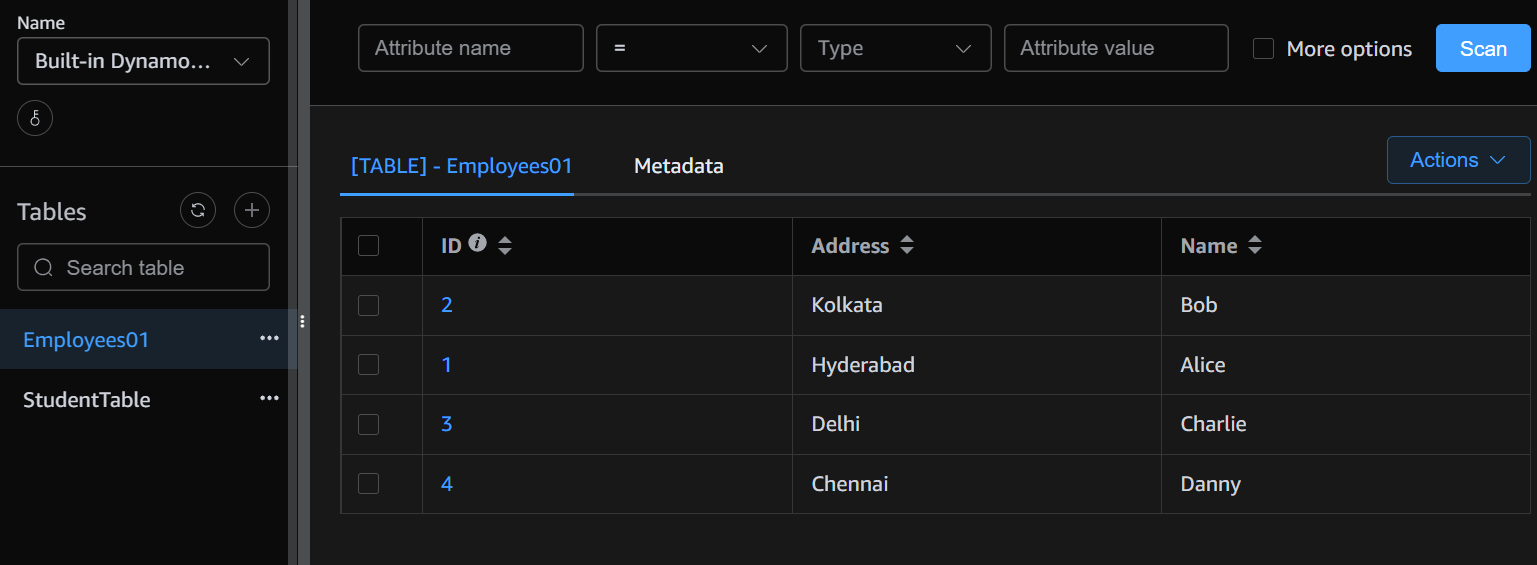
****

****

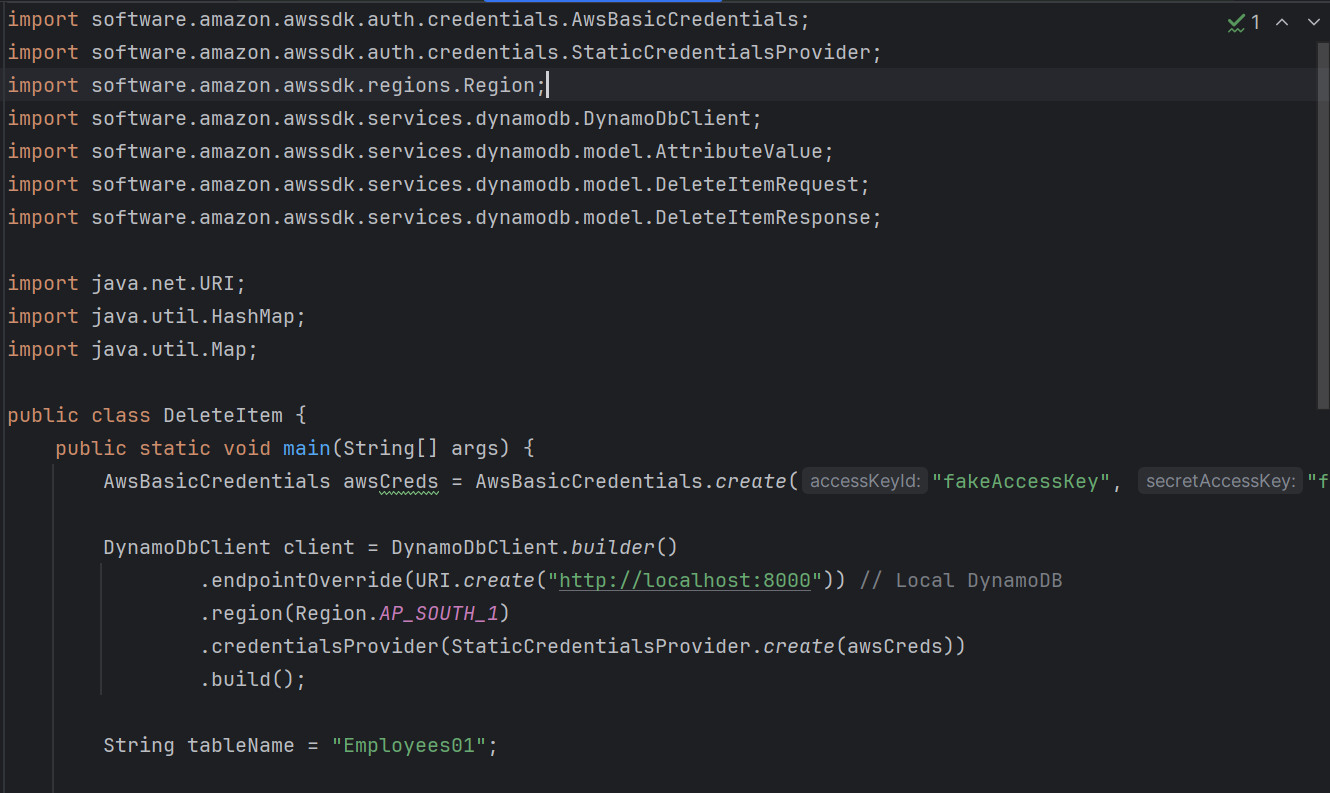
**Output**

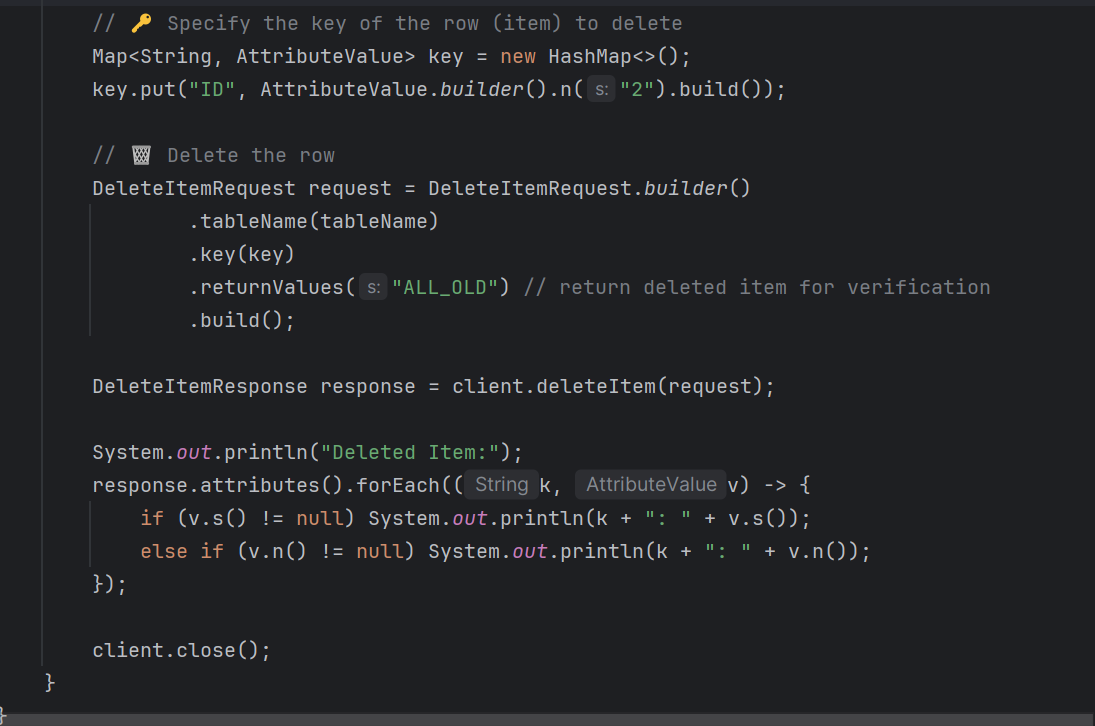
****

**Updated in NOSQL Workbench**

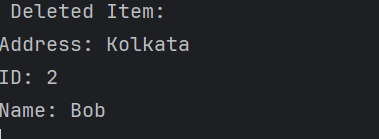
****

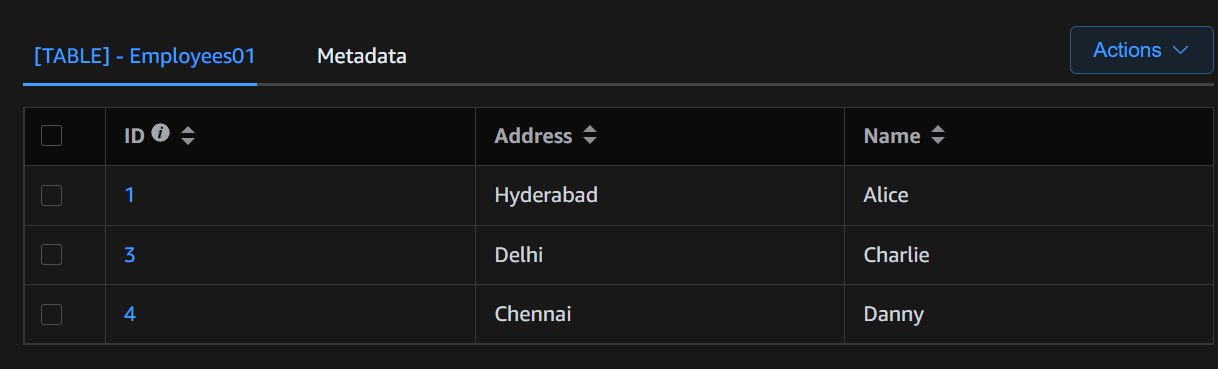
**TASK 9 – Delete Item from the table**

****

****

**Output**

****

****

**HOME TASK1 - What are the features of DynamoDB?**

**Fully Managed**

* AWS handles infrastructure, scaling, backups, and replication.
* No need to manage servers or clusters manually.

**Fast and Consistent Performance**

* Single-digit millisecond response times at any scale.
* Option for eventual consistency or strongly consistent reads.

**Scalable and Elastic**

* Automatically scales throughput and storage.
* Supports millions of requests per second.

**Flexible Data Model**

* Key-Value and Document database.
* Supports partition key + sort key (composite keys).
* Schema-less: you can store different attributes for different items.

**Capacity Modes**

* Provisioned mode with Auto Scaling.
* On-Demand mode for unpredictable workloads.

**Global Tables**

* Multi-Region, multi-master replication.
* Enables low-latency global applications.

**High Availability and Durability**

* Data is automatically replicated across multiple AZs (Availability Zones).

**Built-in Security**

* Supports IAM policies, encryption at rest (KMS), and in-transit (TLS).
* Fine-grained access control.

**Backup and Restore**

* On-demand and continuous backups.
* Point-in-time recovery (PITR) for the last 35 days**.**

**Streams and Event-Driven Integration**

* DynamoDB Streams capture table activity.
* Can trigger AWS Lambda for serverless event processing.

**DAX (DynamoDB Accelerator)**

* In-memory caching service.
* Improves read performance to microseconds**.**

**Cost-Effective Options**

* Standard vs. Standard-IA (Infrequent Access) table class.
* Reserved capacity for long-term savings.

**HOME TASK 2 - What are the advantages and disadvantages of Dynamodb**

**Advantages of DynamoDB**

1. **Fully Managed**
   * No need to handle server provisioning, patching, or scaling. AWS manages everything.
2. **High Availability & Durability**
   * Data is automatically replicated across multiple Availability Zones in a region.
3. **Scalability**
   * Handles millions of requests per second with automatic scaling (both storage and throughput).
4. **Flexible Data Model**
   * Supports key–value and document-based storage (JSON-like structure).

**Disadvantages of DynamoDB**

1. **Pricing Complexity**
   * Can get expensive if read/write traffic is unpredictable.
   * Pay separately for storage, throughput, backups, and streams.
2. **Limited Query Flexibility**
   * No joins, group by, or complex aggregations like SQL databases.
   * Queries are limited to **primary key (Partition Key + Sort Key)** or **indexes**.
3. **Index Management Overhead**
   * Global Secondary Indexes (GSIs) and Local Secondary Indexes (LSIs) increase cost and must be designed carefully.
4. **Eventual Consistency** (for some reads)
   * By default, strongly consistent reads are not guaranteed unless explicitly requested.
5. **Data Size Limits**
   * Each item has a **400 KB limit**, which can be restrictive for large objects.
6. **Hot Partition Issues**
   * Poor partition key design can cause uneven traffic (some partitions overloaded, others idle).

**HOMETASK 3 - Where do we use dynamoDB(the uses cases of DynamoDb)?**

**Use Cases of DynamoDB**

1. **Real-time applications requiring low latency**
   * Gaming leaderboards
   * Real-time bidding platforms
   * Session management for web/mobile apps
2. **Applications with variable or high traffic**
   * E-commerce shopping carts
   * Flash sales or promotional events
   * Social media feeds
3. **Serverless and microservices architectures**
   * Storing state for AWS Lambda functions
   * Event-driven microservices
   * IoT device data ingestion
4. **Flexible schema applications**
   * Content management systems
   * User profiles with optional fields
   * Product catalogs with varying attributes
5. **High availability and global distribution**
   * Multi-region financial transaction history
   * Global leaderboards or worldwide apps
6. **Queueing or caching patterns**
   * Message queues or event queues
   * Caching frequently accessed data with DAX
   * Change data capture using DynamoDB Streams
7. **Security and compliance needs**
   * Storing sensitive logs securely
   * Compliant audit trails for healthcare, finance, or regulatory apps

**HOMETASK 4 – What is DynamoDBMapper?**

DynamoDBMapper is a high-level Java library provided by AWS to simplify working with Amazon DynamoDB. It acts as an Object-Relational Mapping (ORM) tool, but for DynamoDB (a NoSQL database), allowing you to interact with tables using Java objects instead of writing low-level API calls.

* Maps **Java classes** to DynamoDB **tables**.
* Maps **class fields** to DynamoDB **attributes**.
* Simplifies **CRUD operations** (Create, Read, Update, Delete) and **queries**.

**Object Mapping**

* Annotate a Java class with @DynamoDBTable(tableName="TableName").
* Annotate fields with @DynamoDBHashKey, @DynamoDBRangeKey, @DynamoDBAttribute.

**CRUD Operations**

* mapper.save(object) → inserts or updates an item
* mapper.load(Class, key) → retrieves an item by key
* mapper.delete(object) → deletes an item

**Querying & Scanning**

* Supports DynamoDBQueryExpression and DynamoDBScanExpression for advanced queries.

**Automatic Conversion**

* Converts Java types to DynamoDB attribute types (String, Number, Boolean, List, Map, etc.).

**HOMETASK 5 - What are projections in DynamoDB?**

A projection specifies which attributes are copied from a table into a secondary index (GSI or LSI) or returned in a query.

* Helps optimize queries by only retrieving the data you need.
* Reduces read capacity usage and latency.

**HOMETASK 6 - How can you say DynamoDB prevents Dataloss?**

**Multi-AZ Replication**

* DynamoDB automatically replicates data across **three Availability Zones (AZs)** in an AWS region.
* This ensures that if one AZ fails, the data is still available in the other AZs.
* **Effect:** Prevents data loss due to hardware or AZ failures.

**2. Durability**

* All writes are **synchronously written** to multiple storage nodes across AZs.
* DynamoDB uses **SSD-backed storage** with **automatic replication**.
* **Effect:** Guarantees that once a write is acknowledged, it will not be lost.

**3. Point-in-Time Recovery (PITR)**

* Optional feature to enable **continuous backups**.
* You can restore the table to any point in the past **up to 35 days**.
* **Effect:** Protects against accidental deletes or updates.

**4. On-Demand Backups**

* You can create **full table backups** manually or via automation.
* **Effect:** Provides recovery options in case of corruption or accidental deletion.

**5. Automatic Error Handling**

* DynamoDB handles **node failures, network issues, and hardware errors** transparently.
* Internal replication and recovery mechanisms make these **failures invisible to the user**.

**6. Strong Consistency Option**

* You can enable **strongly consistent reads**.
* Ensures that the read reflects the **most recent write**, avoiding lost or outdated data being read.

**HOMETASK 7 - What is in-place atomic update means ..? do you think your dynamoDb supports it?**

* **In-place update:** You update an **existing attribute of an item** without overwriting the whole item.
* **Atomic:** The update is **all-or-nothing** — no other operation can interfere.

Example: Incrementing a counter, appending to a list, or updating a single field without touching other fields.

Yes, **DynamoDB supports in-place atomic updates** using **UpdateItem** operation

**HOME TASK 8 - What are Streams in DynamoDb?**

**DynamoDB Streams** is a feature that **captures a time-ordered sequence of changes** (insert, update, delete) made to items in a DynamoDB table.

* Think of it like a **change log** or **event stream** for your table.
* Each change appears as a **stream record** for processing.

**Key Features**

1. **Time-ordered events** – records are delivered in the order changes occur.
2. **Retention** – stream records are stored for **24 hours**.
3. **Event types captured**:
   * INSERT → new item added
   * MODIFY → existing item updated
   * REMOVE → item deleted
4. **Optional payloads**:
   * KEYS\_ONLY – only primary key attributes
   * NEW\_IMAGE – entire new item
   * OLD\_IMAGE – entire old item before modification
   * NEW\_AND\_OLD\_IMAGES – both old and new item

**How Streams Are Used**

* **Trigger AWS Lambda functions** automatically when data changes.
* **Replicate data** to other databases or systems (ETL, analytics).
* **Audit trails** – track who changed what and when.
* **Event-driven microservices** – react to table changes in real time.

**HOME TASK 9 - What are DynamoDB Pricing tiers?**

**1. On-Demand Capacity Mode**

* **Pay-per-request**: You are charged for **actual reads and writes**.
* No need to specify capacity upfront.
* **Best for:** unpredictable or variable workloads.

**Pricing Example:**

* Write request unit (WRU): ~$1.25 per million writes
* Read request unit (RRU): ~$0.25 per million reads

**Pros:**

* No capacity planning
* Auto-scales instantly

**Cons:**

* Can be more expensive for steady, predictable workloads

**2. Provisioned Capacity Mode**

* You **pre-define read and write capacity units** (RCUs and WCUs).
* **Pay for provisioned capacity**, whether you use it or not.
* **Best for:** steady, predictable workloads.

**Pricing Example:**

* Write capacity unit (WCU): ~$0.00065 per WCU-hour
* Read capacity unit (RCU): ~$0.00013 per RCU-hour

**Optional Auto Scaling:**

* DynamoDB can **adjust provisioned capacity** automatically based on traffic.

**3. Additional Costs**

1. **Storage**
   * Charged per GB per month for data stored in DynamoDB tables.
2. **DynamoDB Streams**
   * Charged based on read request units for accessing stream records.
3. **Global Tables (multi-region replication)**
   * Extra cost for replicated writes across regions.
4. **DAX (DynamoDB Accelerator / caching)**
   * Separate cost for provisioned cluster nodes.
5. **On-demand backup & PITR**
   * Charged per GB of data backed up.

**HOME TASK 10 - Do you have any max limit for Item Size in DynamoDb? If so what is it?**

Yes! DynamoDB **does have a maximum item size limit**. Here’s the breakdown:

**Maximum Item Size**

* **Limit:** **400 KB per item** (includes **attribute names + values** + any metadata).
* This applies to **each individual item**, whether it’s a single record in the table or an item in a secondary index.

**HOMETASK 11 - At a max of how many GSI’s Global Secondary Indexes can you create in a table?**

**Maximum GSIs per Table**

* Default limit: 20 Global Secondary Indexes per table
* This is the maximum number of **GSIs** you can create on a single table.

Note: This is separate from Local Secondary Indexes (LSIs), which have their own limit (5 per table).

**HOMETASK 12 - What is DynamoDb Accelerator?**

**Definition**

DynamoDB Accelerator (DAX) is a fully managed, in-memory caching service for DynamoDB.

* **Purpose:** Improve read performance of DynamoDB tables.
* Provides microsecond latency for read-heavy workloads instead of the usual millisecond latency.
* Fully compatible with DynamoDB API, so you don’t need to change your application logic.

**Key Features**

1. **In-Memory Cache**
   * Stores frequently accessed items in memory for ultra-fast reads.
2. **Fully Managed**
   * AWS handles cluster provisioning, patching, and scaling.
3. **Write-Through Caching**
   * Writes go to DynamoDB and the cache automatically.
4. **Transparent Integration**
   * Minimal changes needed in your application; just point your client to the DAX cluster.
5. **Scalable**
   * Can scale horizontally by adding nodes to the DAX cluster.

**Benefits**

* **Faster Reads:** Microsecond response times for repeated queries.
* **Reduced Load on DynamoDB:** Offloads read traffic, reducing consumed Read Capacity Units (RCUs).
* **Seamless Integration:** Works with existing DynamoDB tables and APIs.
* **Cost-Effective:** Lower DynamoDB read costs for read-heavy workloads.

**HOMETASK 13 - What are DynamoDB Global tables?**

**Definition**

DynamoDB Global Tables are multi-region, fully replicated DynamoDB tables.

* They allow automatic replication of table data across two or more AWS regions.
* Provides low-latency, high-availability access for users worldwide.
* Changes made in one region are eventually replicated to other regions automatically.

**2. Key Features**

1. **Multi-region replication**
   * Each region has a **read/write copy** of the table.
2. **Fully managed**
   * AWS handles replication, conflict resolution, and consistency.
3. **High availability**
   * Data remains available even if one region fails.
4. **Global low-latency access**
   * Users access the nearest region for faster performance.
5. **Conflict resolution**
   * DynamoDB uses **last-writer-wins (timestamp-based)** to resolve write conflicts.

**HOME TASK 14 - What are indexes and Secondary indexes in DynamoDB?**

* An index is a data structure that allows you to query a table efficiently using an alternate key rather than the primary key.
* Think of it as a **shortcut** to access items quickly based on attributes other than the main primary key.

**Primary Types of Indexes in DynamoDB**

DynamoDB supports **two types of secondary indexes**:

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

* Definition: An index that shares the same partition key as the base table but has a different sort key.
* **Key points:**
  + Must be created **when the table is created** (cannot add later).
  + Maximum 5 LSIs per table.
  + Allows querying items with different sort keys without scanning the entire table.
* **Use case:** Query items by a different sort key within the same partition.

**Example:**  
Table Orders has CustomerID as partition key and OrderID as sort key.

* LSI: Partition key = CustomerID, Sort key = OrderDate → lets you query all orders for a customer sorted by date.

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

* Definition: An index that can have a different partition key and sort key from the base table.
* **Key points:**
  + Can be added or deleted anytime.
  + Maximum 20 GSIs per table.
  + Can span all partitions, not just a single one.
* **Use case:** Query items based on attributes not part of the main primary key.

**Example:**  
Table Orders has OrderID as primary key.

* GSI: Partition key = CustomerID, Sort key = OrderDate → lets you query all orders by customer efficiently.

**HOME TASK 15 - What are Hot Keys and Hot Partitions?**

**Hot Key**

* A hot key is a partition key that is accessed far more frequently than others.
* When many requests target a single key, it can overwhelm the underlying partition.
* **Effect:** High read/write traffic on one key can cause throttling or performance degradation.

**Hot Partition**

* DynamoDB partitions table data across multiple physical partitions.
* Each partition has a throughput limit (RCUs and WCUs).
* A hot partition occurs when too much traffic is concentrated on a single partition, exceeding its capacity.

**Effect:**

* Some requests may be throttled even if the overall table capacity is sufficient.

**HOME TASK 16 - What are Table level operations and Item level operations in DynamoDB?**

**1. Table-Level Operations**

These operations affect the entire table or its structure, not individual items.

**Examples**

1. **CreateTable** – Create a new DynamoDB table.
2. **DeleteTable** – Delete an existing table.
3. **UpdateTable** – Change table settings like provisioned throughput, indexes, or TTL.
4. **DescribeTable** – Get table metadata (schema, keys, capacity, etc.).
5. **ListTables** – List all tables in your account/region.

**Key Points**

* Operates on the table as a whole.
* Used mainly for administrative tasks.
* May affect availability while updating table settings.

**2. Item-Level Operations**

These operations affect individual items (rows) in a table.

**Examples**

1. **PutItem** – Add a new item or replace an existing item.
2. **GetItem** – Retrieve an item by its primary key.
3. **UpdateItem** – Update attributes of an existing item (supports atomic updates).
4. **DeleteItem** – Remove an item by its primary key.
5. **BatchGetItem / BatchWriteItem** – Perform multiple item operations in a single request.

**Key Points**

* Operates at the row/item level, not the table.
* Commonly used for CRUD operations in applications.
* Can leverage conditional writes for concurrency control.