# Lab 2: Working with Amazon DynamoDB Tables

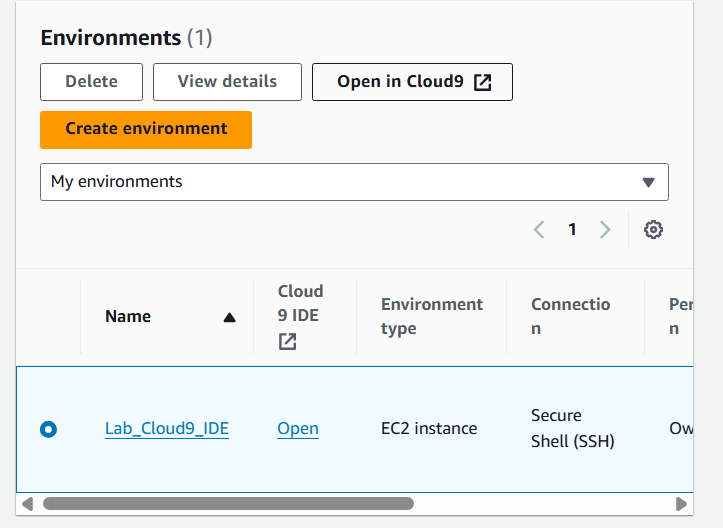
1. Determine table partition and sort keys based on data access patterns of the application.
2. Determine optimal selection of local and global secondary indexes needed to support data access patterns.
3. Create a DynamoDB table.
4. Implement a local secondary index (LSI)
5. Implement a global secondary index (GSI)

NoSQL stands for "Not Only SQL."

Key-Value Stores: Store data as key-value pairs.

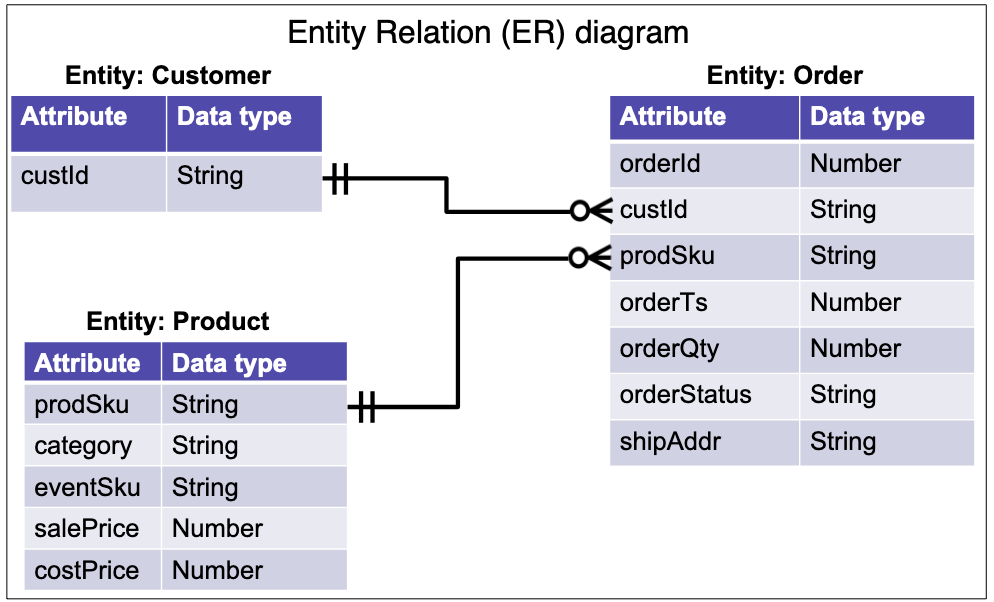
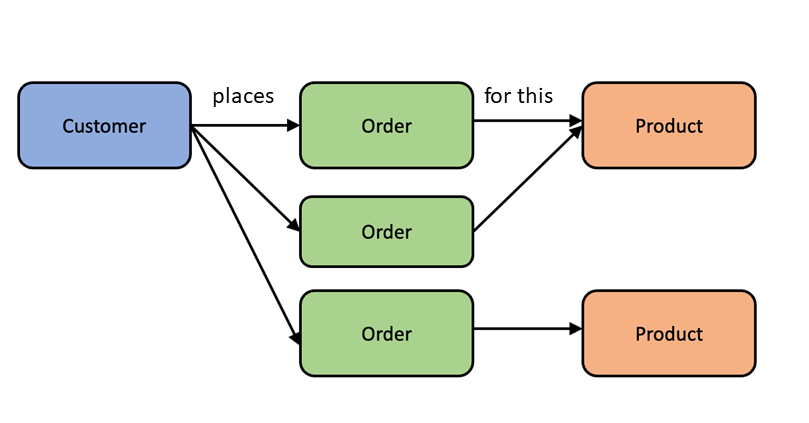
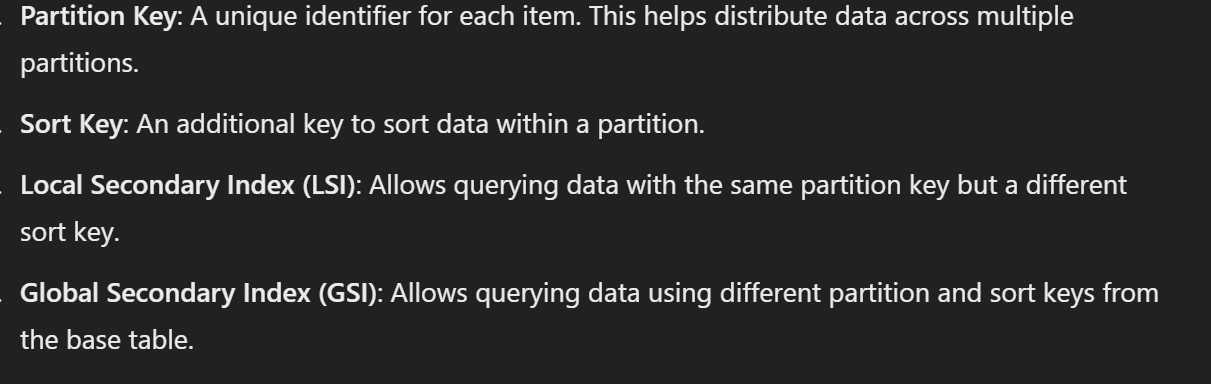
Steps-

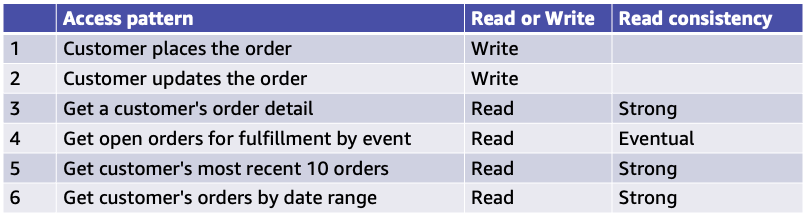
1. Open cloud9 from console
2. Open



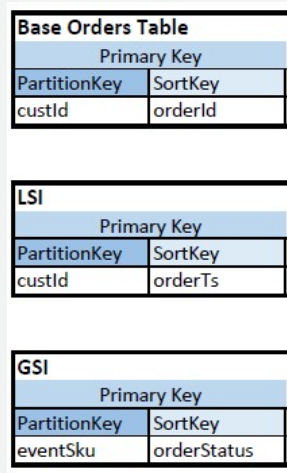
1. Keep it open

**Task 1 - Identify the partition key, sort key, and any secondary indexes for the orders table**





Choice of keys:



Base Table Design:

Partition Key: custId

Sort Key: orderId

Reasoning: Ensures unique identification of orders and efficient distribution of load across partitions.

Local Secondary Index (LSI):

Partition Key: custId

Sort Key: orderTs (timestamp)

Projection: All remaining attributes

Purpose: Efficient querying of a customer’s orders within a specific timeframe.

Global Secondary Index (GSI):

Partition Key: eventSku

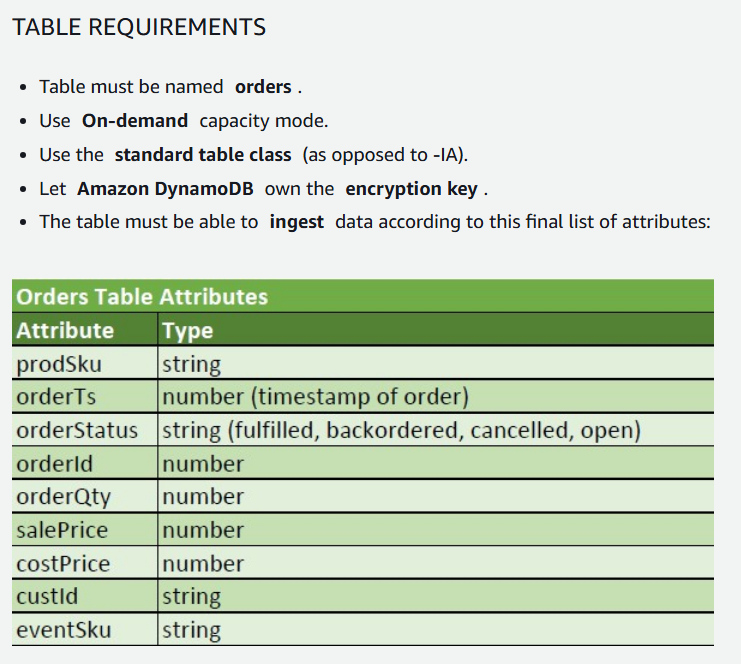
Sort Key: orderStatus

Projection: custId, orderId, prodSku, orderQty

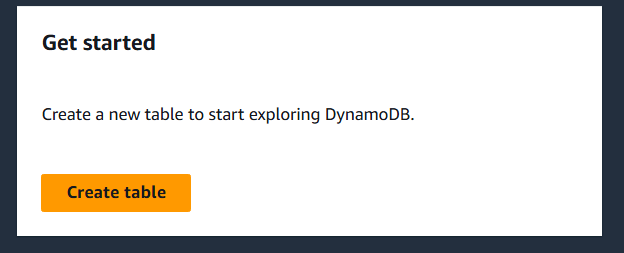
Purpose: Efficient querying of all open orders for a specific event.

**Task 2 - Create the orders DynamoDB table and any secondary indexes**

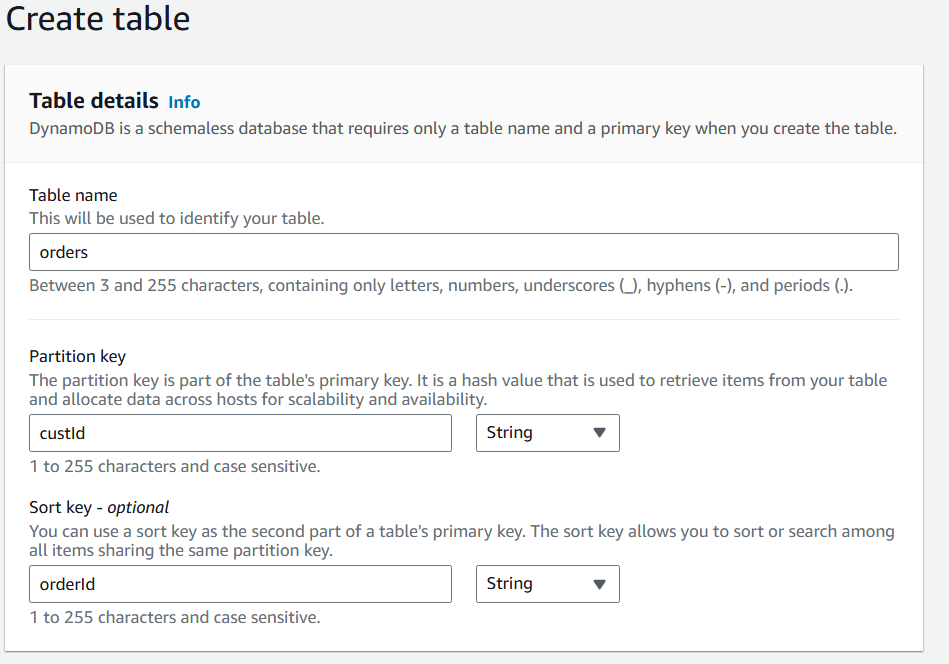
2.1 note this for table



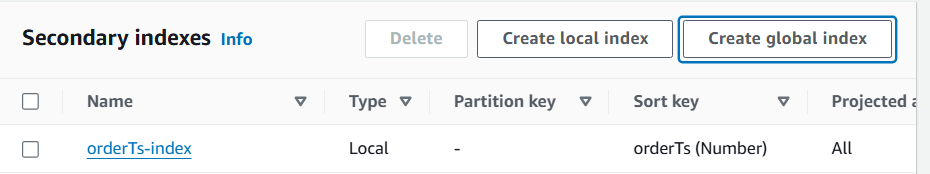
2.2 open dynamoDB in conole, create table



2.3 create table with 2.1 details



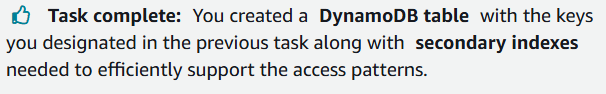
LSI-



GSI-



2.4 create tablew



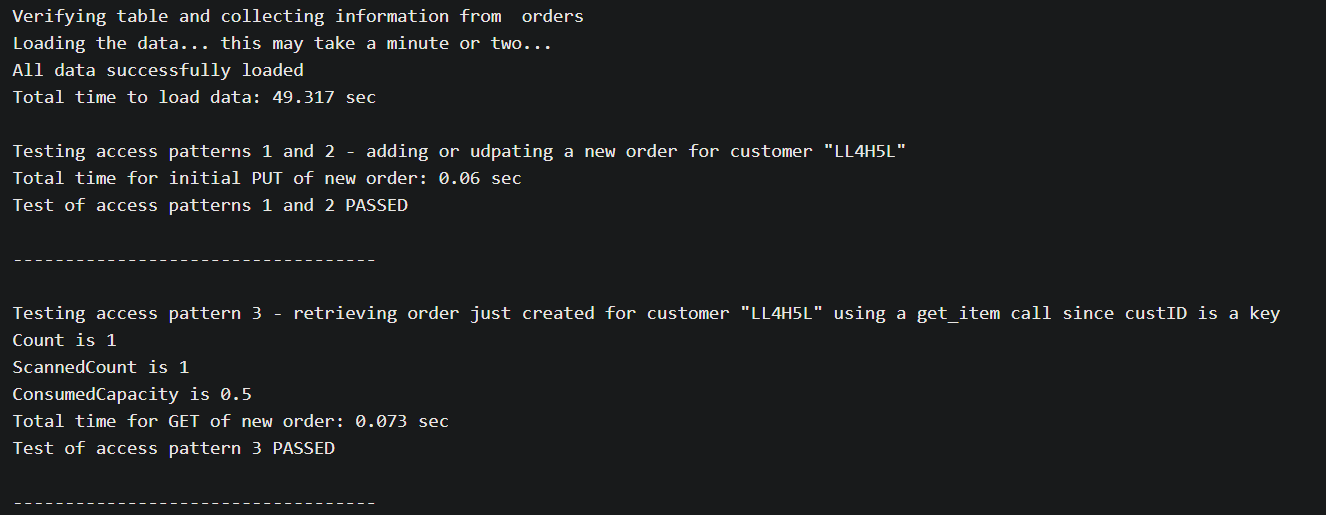
**Task 3 - Validate orders table and indexes configuration**

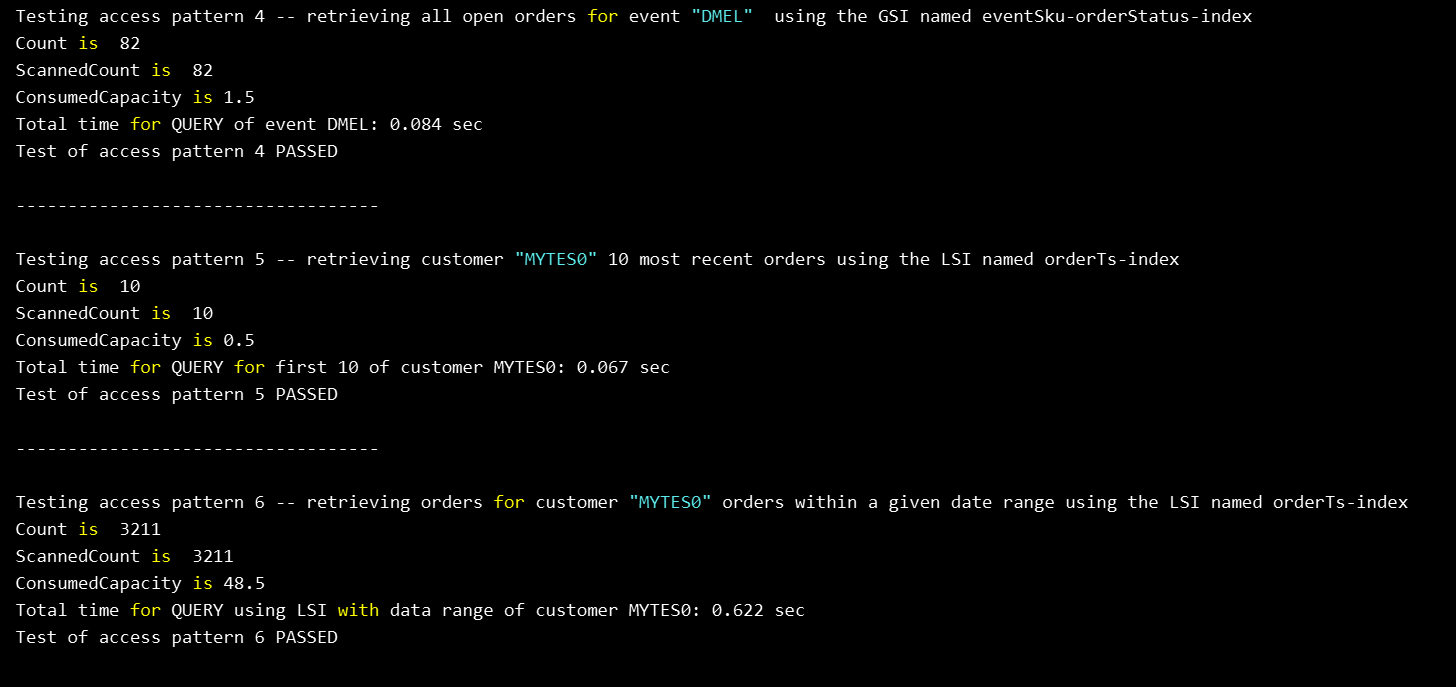
3.1 Run this in cloud9 terminal (precreated .py file to check tables)



3.2 output

1. The first part of the script attempts to load sample data into the orders table.
2. After that, It will verify all 6 access patterns.





note-

