

Lab 2: Working with Amazon DynamoDB Tables

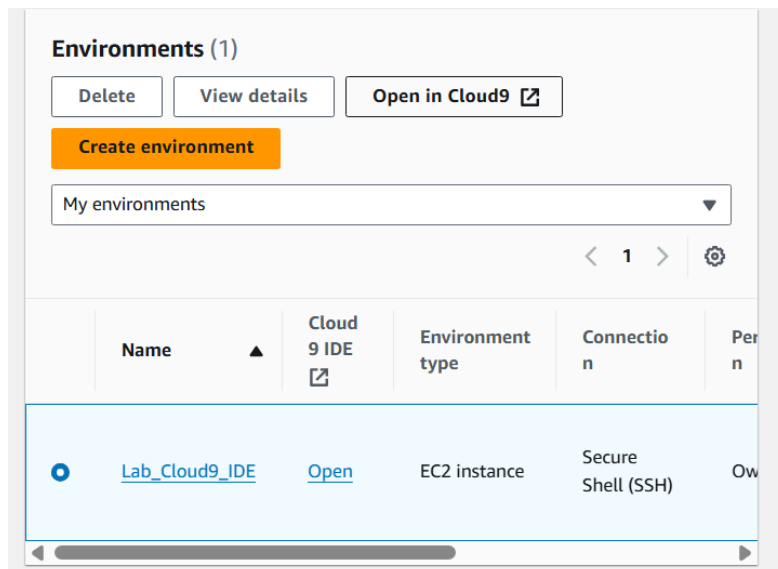
1. Determine table partition and sort keys based on data access patterns of the application.
2. Determine optimal selection of local and global secondary indexes needed to support data access patterns.
3. Create a DynamoDB table.
4. Implement a local secondary index (LSI)
5. Implement a global secondary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



3. Keep it open

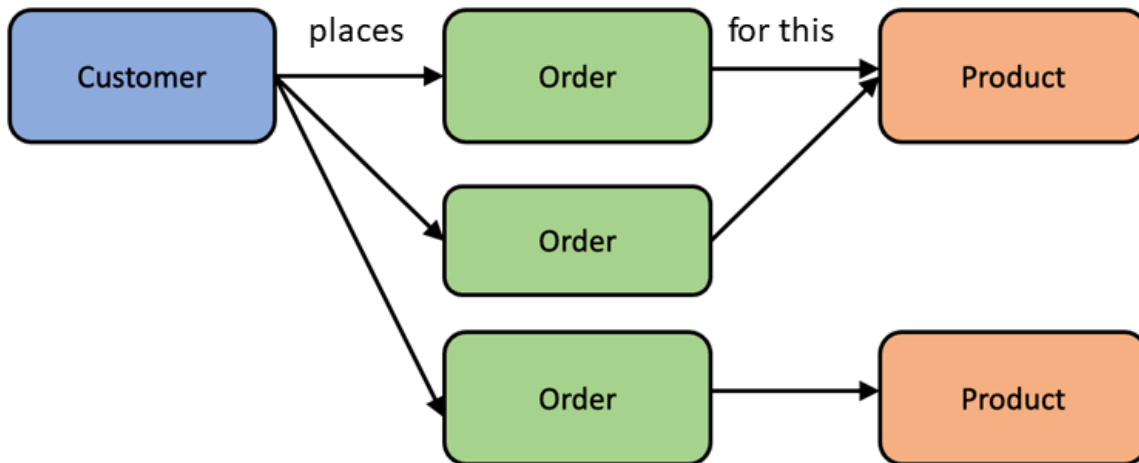
Task 1 - Identify the partition key, sort key, and any secondary indexes for the orders table

Partition Key: A unique identifier for each item. This helps distribute data across multiple partitions.

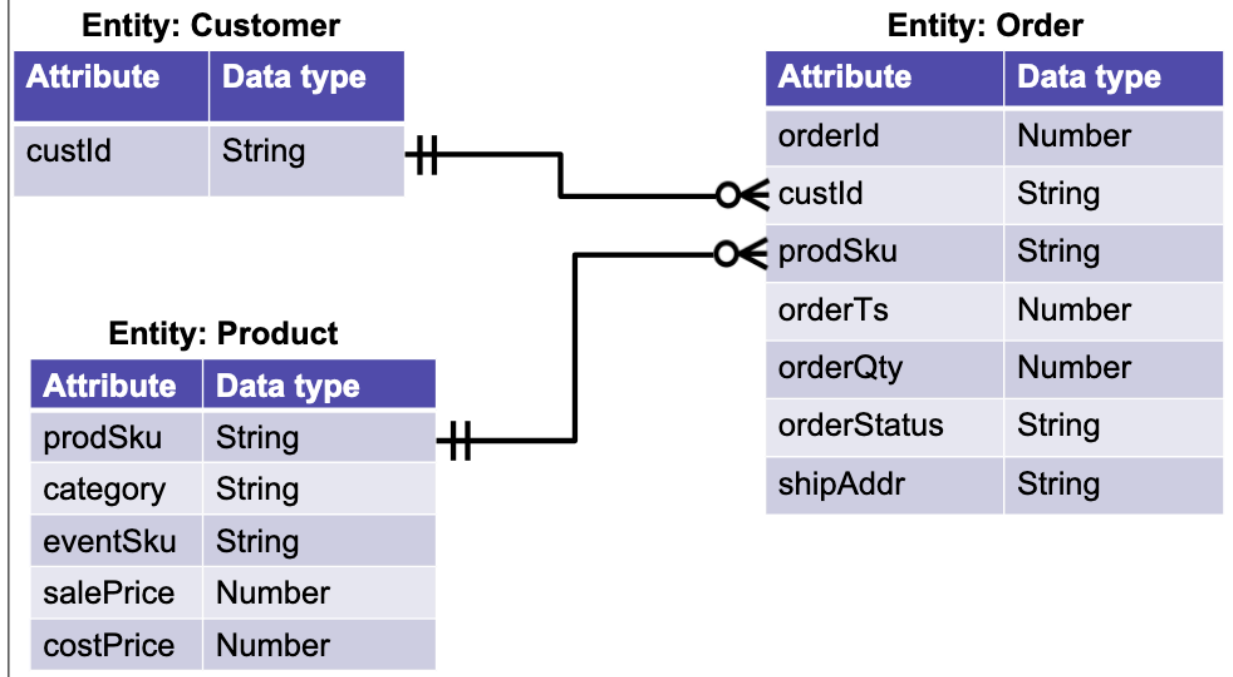
Sort Key: An additional key to sort data within a partition.

Local Secondary Index (LSI): Allows querying data with the same partition key but a different sort key.

Global Secondary Index (GSI): Allows querying data using different partition and sort keys from the base table.



Entity Relation (ER) diagram



	Access pattern	Read or Write	Read consistency
1	Customer places the order	Write	
2	Customer updates the order	Write	
3	Get a customer's order detail	Read	Strong
4	Get open orders for fulfillment by event	Read	Eventual
5	Get customer's most recent 10 orders	Read	Strong
6	Get customer's orders by date range	Read	Strong

Choice of keys:

Base Orders Table	
Primary Key	
PartitionKey	SortKey
custId	orderId

LSI	
Primary Key	
PartitionKey	SortKey
custId	orderTs

GSI	
Primary Key	
PartitionKey	SortKey
eventSku	orderStatus

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

TABLE REQUIREMENTS

- Table must be named **orders** .
- Use **On-demand** capacity mode.
- Use the **standard table class** (as opposed to -IA).
- Let **Amazon DynamoDB** own the **encryption key** .
- The table must be able to **ingest** data according to this final list of attributes:

Orders Table Attributes	
Attribute	Type
prodSku	string
orderTs	number (timestamp of order)
orderStatus	string (fulfilled, backordered, cancelled, open)
orderId	number
orderQty	number
salePrice	number
costPrice	number
custId	string
eventSku	string

2.2 open dynamoDB in conole, create table

Get started

Create a new table to start exploring DynamoDB.

Create table

2.3 create table with 2.1 details

Create table

Table details [Info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

orders

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

custId

String ▼

1 to 255 characters and case sensitive.

Sort key - *optional*

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

orderId

String ▼

1 to 255 characters and case sensitive.


LSI-

Secondary indexes Info						Delete		Create local index		Create global index	
<input type="checkbox"/>	Name	Type	Partition key	Sort key	Projected a						
<input type="checkbox"/>	orderTs-index	Local	-	orderTs (Number)	All						

GSI-

eventSku-orderStatus-index	Global	eventSku (String)	orderStatus (String)	Include: custId, orderId,
--	--------	-------------------	----------------------	---------------------------

2.4 create tablew

 **Task complete:** You created a **DynamoDB table** with the keys you designated in the previous task along with **secondary indexes** needed to efficiently support the access patterns.

Task 3 - Validate orders table and indexes configuration

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

```
~/environment $ cd ~/environment/labFiles
~/environment/labFiles (main) $ python VerifyDynamoDB.py
```

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.


```

Verifying table and collecting information from orders
Loading the data... this may take a minute or two...
All data successfully loaded
Total time to load data: 49.317 sec

Testing access patterns 1 and 2 - adding or updating a new order for customer "LL4H5L"
Total time for initial PUT of new order: 0.06 sec
Test of access patterns 1 and 2 PASSED

-----

Testing access pattern 3 - retrieving order just created for customer "LL4H5L" using a get_item call since custID is a key
Count is 1
ScannedCount is 1
ConsumedCapacity is 0.5
Total time for GET of new order: 0.073 sec
Test of access pattern 3 PASSED

-----

Testing access pattern 4 -- retrieving all open orders for event "DMEL" using the GSI named eventSku-orderStatus-index
Count is 82
ScannedCount is 82
ConsumedCapacity is 1.5
Total time for QUERY of event DMEL: 0.084 sec
Test of access pattern 4 PASSED

-----

Testing access pattern 5 -- retrieving customer "MYTES0" 10 most recent orders using the LSI named orderTs-index
Count is 10
ScannedCount is 10
ConsumedCapacity is 0.5
Total time for QUERY for first 10 of customer MYTES0: 0.067 sec
Test of access pattern 5 PASSED

-----

Testing access pattern 6 -- retrieving orders for customer "MYTES0" orders within a given date range using the LSI named orderTs-index
Count is 3211
ScannedCount is 3211
ConsumedCapacity is 48.5
Total time for QUERY using LSI with data range of customer MYTES0: 0.622 sec
Test of access pattern 6 PASSED

```

note-

Count - The number of items that were returned to meet the access pattern request

ScannedCount - The number of items that were scanned to find the requested items. There are just under 75,000 items in the table when it is fully loaded.

ConsumedCapacity - The number of read capacity units (RCUs) consumed to complete the request. This is a key value that is associated with table capacity and pricing. The lower the number, the more efficiently the request was handled.