**ASSIGNMENT NO. 9**

**Aim**

Design database schemas and implement min 10 queries using DynamoDB

**Objective**

Understand, design and implement queries in Dynamodb (key value store) .

**Theory**

* Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database, so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.
* With DynamoDB, you can create database tables that can store and retrieve any amount of data, and serve any level of request traffic. You can scale up or scale down your tables' throughput capacity without downtime or performance degradation, and use the AWS Management Console to monitor resource utilization and performance metrics.
* DynamoDB automatically spreads the data and traffic for your tables over a sufficient number of servers to handle your throughput and storage requirements, while maintaining consistent and fast performance. All of your data is stored on solid state disks (SSDs) and automatically replicated across multiple Availability Zones in an AWS region, providing built-in high availability and data durability.

Dynamodb Data Model

* [Tables, Items, and Attributes](http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DataModel.html)
* [Primary Key](http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DataModel.html)
* [Secondary Indexes](http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DataModel.html)
* [DynamoDB Data Types](http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DataModel.html)
* [Item Distribution](http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DataModel.html)

Table, Items and Attributes

* A *table* is a collection of *items* and each item is a collection of *attributes*.
* Each attribute in an item is a name-value pair
* An attribute can be a scalar (single-valued), a JSON document, or a set

Example of Item

{ Id = 101

ProductName = "Book 101 Title"

ISBN = "111-1111111111"

Authors = [ "Author 1", "Author 2" ]

Price = -2

Dimensions = "8.5 x 11.0 x 0.5"

PageCount = 500

InPublication = 1

ProductCategory = "Book"

}

Primary Key

DynamoDB supports two different kinds of primary keys:

* **Partition Key** – A simple primary key, composed of one attribute known as the *partition key*. DynamoDB uses the partition key's value as input to an internal hash function; the output from the hash function determines the partition where the item will be stored. No two items in a table can have the same partition key value.
* **Partition Key and Sort Key** – A composite primary key, composed of two attributes. The first attribute is the *partition key*, and the second attribute is the *sort key*. DynamoDB uses the partition key value as input to an internal hash function; the output from the hash function determines the partition where the item will be stored. All items with the same partition key are stored together, in sorted order by sort key value. It is possible for two items to have the same partition key value, but those two items must have different sort key values.

Secondary Indexes

* Query the data in the table using an alternate key, in addition to queries against the primary key.

DynamoDB supports two kinds of secondary indexes:

* **Global secondary index** – an index with a partition key and sort key that can be different from those on the table.
* **Local secondary index** – an index that has the same partition key as the table, but a different sort key.

**DynamoDB Data Types**

* **Scalar types** – Number, String, Binary, Boolean, and Null.
* **Document types** – List and Map.
* **Set types** – String Set, Number Set, and Binary Set.

Item Distribution: Partition Key

* To write an item to the table, DynamoDB uses the value of the partition key as input to an internal hash function. The output value from the hash function determines the partition in which the item will be stored.
* To read an item from the table, you must specify the partition key value for the item. DynamoDB uses this value as input to its hash function, yielding the partition in which the item can be found.

Item Distribution: Partition Key and sort key

* To write an item to the table, DynamoDB calculates the hash value of the partition key to determine which partition should contain the item. In that partition, there could be several items with the same partition key value, so DynamoDB stores the item among the others with the same partition key, in ascending order by sort key.
* To read an item from the table, you must specify its partition key value and sort key value. DynamoDB calculates the partition key's hash value, yielding the partition in which the item can be found.

Table Operations

* CreateTable
* DeleteTable
* UpdateTable
* ListTable

Item Operations

* BatchGetItem
* BatchWriteItem
* DeleteItem
* GetItem
* PutItem
* UpdateItem
* Query
* Scan

**1. Create Table**

var params = {

TableName : "emp",

KeySchema : [

{AttributeName: "empid", KeyType: "HASH" },

{AttributeName: "ename", KeyType: "RANGE" }

],

AttributeDefinitions: [

{AttributeName: "empid", AttributeType: "S" },

{AttributeName: "ename", AttributeType:"S" }

],

ProvisionedThroughput: {

ReadCapacityUnits: 1,

WriteCapacityUnits: 1

}

};

dynamodb.createTable(params, function(err, data)

{

if(err)

console.log(JSON.stringify(err, null, 2));

else

console.log(JSON.stringify(data, null, 2));

});

**2. DescribeTable**

var params = {

TableName : "emp",

};

dynamodb.describeTable(params, function(err, data)

{

if(err)

console.log(JSON.stringify(err, null, 2));

else

console.log(JSON.stringify(data, null, 2));

});

**3. Sacn table**

var params = {

TableName: "emp",

};

dynamodb.scan(params, function(err, data) {

if (err)

console.log(JSON.stringify(err, null, 2));

else

console.log(JSON.stringify(data, null, 2));

});

**4. Put Item**

var params = {

TableName: "emp",

Item : {

"empid":"s1002",

"ename":"GMW"

}

};

dynamodb.putItem(params, function(err, data) {

if (err)

console.log(JSON.stringify(err, null, 2));

else

console.log(JSON.stringify(data, null, 2));

});

**5 Get Item**

var params = {

TableName : "emp",

Key:{

"empid":"s1003",

"ename":"ask",

}

};

dynamodb.getItem(params, function(err, data)

{

if(err)

console.log(JSON.stringify(err, null, 2));

else

console.log(JSON.stringify(data, null, 2));

});

**6 Projection Expression**

var params = {

TableName : "emp",

Key:{

"empid":"s1003",

"ename":"ask",

},

ProjectionExpression: "salary"

};

dynamodb.getItem(params, function(err, data)

{

if(err)

console.log(JSON.stringify(err, null, 2));

else

console.log(JSON.stringify(data, null, 2));

});

**7 Delete Item**

var params = {

TableName : "emp",

Key:{

"empid":"s1003",

"ename":"ask",

}

};

dynamodb.deleteItem(params, function(err, data)

{

if(err)

console.log(JSON.stringify(err, null, 2));

else

console.log(JSON.stringify(data, null, 2));

});

**8 Conditional Delete**

var params = {

TableName : "emp",

Key:{

"empid":"s1004",

"ename":"pvg",

},

ConditionExpression:"salary = :esal",

ExpressionAttributeValues:{":esal":20000}

};

dynamodb.deleteItem(params, function(err, data)

{

if(err)

console.log(JSON.stringify(err, null, 2));

else

console.log(JSON.stringify(data, null, 2));

});

**9 Delete Table**

var params = {

TableName : "emp",

};

dynamodb.deleteTable(params, function(err, data)

{

if(err)

console.log(JSON.stringify(err, null, 2));

else

console.log(JSON.stringify(data, null, 2));

});

**10 Update Item**

var params = {

TableName : "emp",

Key:{

"empid":"s1003",

"ename":"ask"

},

UpdateExpression:"SET salary = :esal",

ExpressionAttributeValues:{

":esal":40000

},

ReturnValues:"ALL\_NEW"

};

dynamodb.updateItem(params, function(err, data)

{

if(err)

console.log(JSON.stringify(err, null, 2));

else

console.log(JSON.stringify(data, null, 2));

});

**11 BatchGetItem**

var params = {

RequestItems: {

"emp": {

Keys: [

{

"empid": "1001",

"ename": "YNG"

},

{

"empid": "1002",

"ename": "GMW"

}

],

ProjectionExpression:"empid, salary"

}

}

};

dynamodb.batchGetItem(params, function (err, data) {

if (err)

console.log(JSON.stringify(err, null, 2));

else

console.log(JSON.stringify(data, null, 2));

});