Amazon DynamoDB

Bhakta, Tejash
Jain, Harsh Manoj

Amazon DynamoDB

- Fully managed NoSQL database offered by AWS
- Provides consistent performance at scale

Benefits:

- Flexible schema. Each row can have any number of columns
- Replicates data across your choice of AWS regions. Facilitates single-digit millisecond read and write performance
- No servers to provision or manage. No software to install, maintain, or operate

Data Model

- Table
  - Collection of items
- Items
  - No explicit limitations but cannot exceed 400 KB
- Attributes
  - Composed of attribute name and value

    Eg. “SongTitle” : “Somewhere Down The Road”

Example of Music Table[1]

Datatypes of an Attribute

- Scalar
  - Number, String, Binary, Boolean and Null

- Multi-Valued
  - String Set `"SS": ["Pop", "Jazz" ,"Rock"]`
  - Number Set `"NS": ["42.2", "-19", "7.5", "3.14"]`
  - Binary Set `"BS": ["U3Vubnk=", "UmFpbnk=", "U25vd3k="]`

- Document
  - List `"L": [ {"S": "Cookies"} , {"S": "Coffee"}, {"N", "3.14159"}]`
  - Map `"M": {"Name": {"S": "Joe"}, "Age": {"N": "35"}}`

RDBMS vs DynamoDB

Characteristics	RDBMS	DynamoDB
Optimal Workload	Ad hoc queries, data warehousing and OLAP	Web-scale applications, including social networks, gaming, and IoT
Performance	Developers and DBA optimize queries, indexes and table structure for peak performance	Implementation details are insulated from the developer and application
Connecting to the database	Application program maintains a network connection	DynamoDB is a web-service and interactions with it are stateless
Authentication	Application cannot connect to the database until it is authenticated	Every request should contain a cryptographic signature that authenticates that request

Creating a table (SQL vs DynamoDB)

SQL

```
CREATE TABLE Music (
    Artist VARCHAR(20) NOT NULL,
    SongTitle VARCHAR(30) NOT NULL,
    AlbumTitle VARCHAR(25),
    Year INT,
    Price FLOAT,
    Genre VARCHAR(10),
    Tags TEXT,
    PRIMARY KEY(Artist, SongTitle)
);
```

Creating a table in SQL[1]

DynamoDB

```
{
    TableName : "Music",
    KeySchema: [
        {
            AttributeName: "Artist",
            KeyType: "HASH", //Partition key
        },
        {
            AttributeName: "SongTitle",
            KeyType: "RANGE" //Sort key
        }
    ],
    AttributeDefinitions: [
        {
            AttributeName: "Artist",
            AttributeType: "S"
        },
        {
            AttributeName: "SongTitle",
            AttributeType: "S"
        }
    ],
}
```

Creating a table in DynamoDB[1]

Primary Key

DynamoDB

```
{
   TableName : "Music",
   KeySchema: [
      {
         AttributeName: "Artist",
         KeyType: "HASH", //Partition key
      },
      {
         AttributeName: "SongTitle",
         KeyType: "RANGE" //Sort key
      }
   ],
   AttributeDefinitions: [
      {
         AttributeName: "Artist",
         AttributeType: "S"
      },
      {
         AttributeName: "SongTitle",
         AttributeType: "S"
      }
   ],
}
```

Example of Music Table[1]

When you create a table, you must mention the primary key. Other than the primary key, the table is schemaless.

Two types:

- Partition Key
  - Composed of one attribute
  - The key's value is used as an input to an internal hash function
  - The output from the hash function determines the partition for storage
- Partition and Sort Key
  - Partition key as input and storage partition as output from the hash function
  - All items with the same partition key value are stored together, in sorted order by sort key value
  - Partition Key: "Artist" Sort Key: "SongTitle"

Query, Scan and Filters

- **Query** - Select multiple items that have the same “HASH’ key but different “RANGE” key.
  - Give me all SongTitle from “The Acme Band”
- **Scan** - Reads every item in a table or a Secondary Index.
- **Filter** - Refines the Scan/Query results

```
aws dynamodb scan \
   --table-name Music \
   --projection-expression "SongTitle" \
   --filter-expression “Genre = :gen” \
   --expression-attribute-values '{":gen":{S: “Rock””}}' \
```

Example of Music Table[1]

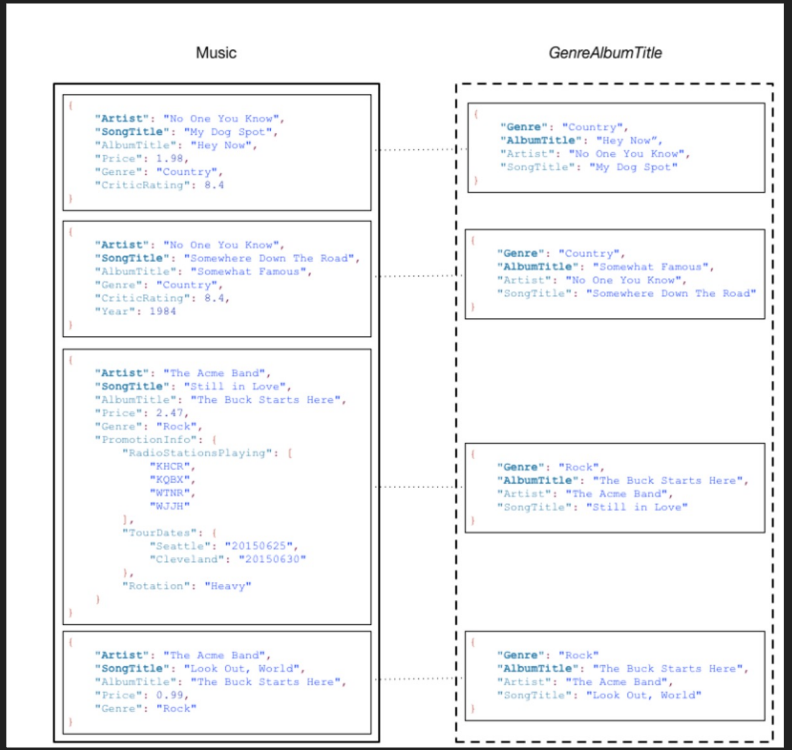
Example of Music Table[1]

Secondary Indexes

- Specify alternate key structures which can be used in Query or Scan operations
- Fast way of accessing a non-key attribute
- Two types of secondary indexes
  - Local Secondary Index
  - Global Secondary Index
- Quota of 20 Global Secondary Indexes and 5 local secondary indexes

Properties of Secondary Indexes

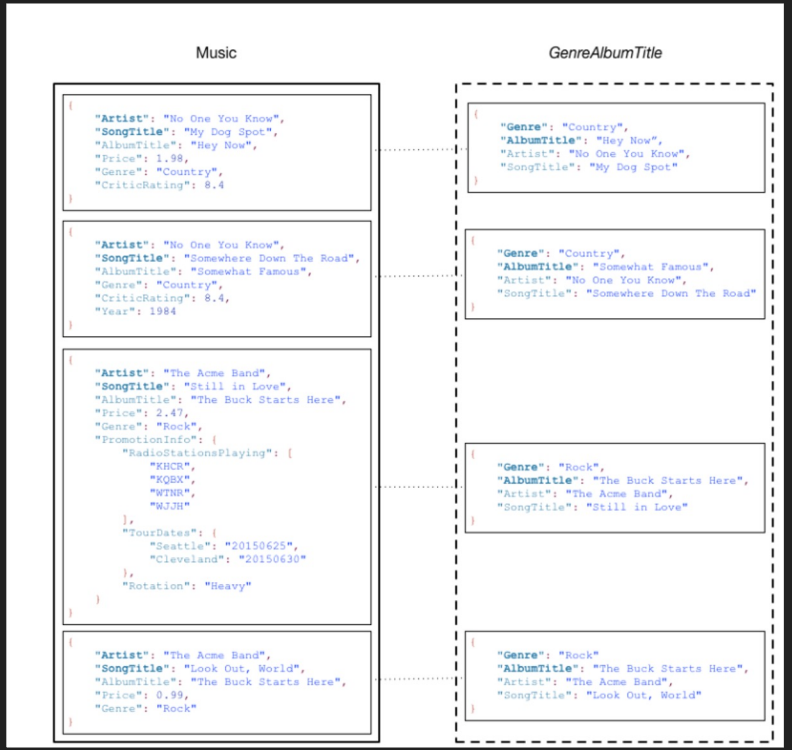
- Every index belongs to a table (base table).
- DynamoDB maintains the indexes automatically.
- When creating an index, the user specifies which attributes to be copied or projected from the base table.
  - KEYS_ONLY
  - INCLUDE
  - ALL



Example of Global Secondary Index[1]

Global Secondary Index

- Primary Key can either be simple or composite
- Partition Key and Sort Key (if any) can be any base table attributes.
- Can add this index either at creation or at a later stage
- Example Query:
  - All Albums from “Rock” Genre.
  - All songs from “Rock” Genre and “The Buck Starts Here”



Example of Global Secondary Index[1]

Local Secondary Index

- Same HASH key(Partition key) as the base table but different RANGE(Sort) key.
- Can be specified only at Table creation.
- Example Query
  - Get the year and genre for a particular album.

```
aws dynamodb query \
    --table-name Music \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v_artist and AlbumTitle = :v_title" \
    --expression-attribute-values  '{":v_artist":{"S":"Acme Band"},":v_title":{"S":"Songs About Life"} }'
```

Query on LSI[1]

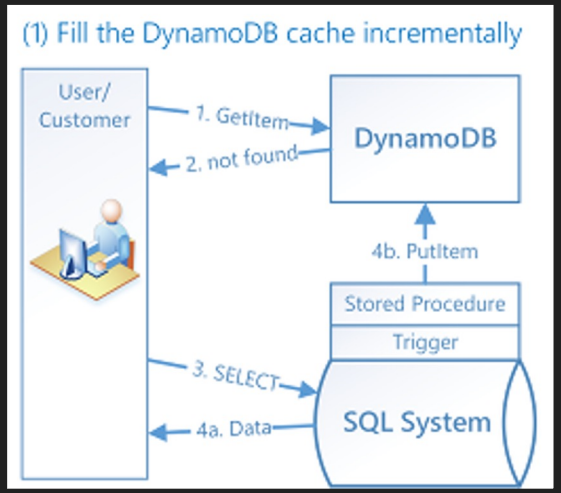
Create Table

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
AttributeName=Artist,AttributeType=S \
AttributeName=SongTitle,AttributeType=S \

AttributeName=AlbumTitle,AttributeType=S  \
    --key-schema
AttributeName=Artist,KeyType=HASH \

AttributeName=SongTitle,KeyType=RANGE \
     --local-secondary-indexes \
       "[{"IndexName": "AlbumTitleIndex",
       "KeySchema":
[{"AttributeName":"Artist","KeyType":"HASH"},

{"AttributeName":"AlbumTitle","KeyType":"RANGE"}],
       "Projection":{"ProjectionType":"INCLUDE",
"NonKeyAttributes":[”Genre", "Year"]}}]"
```

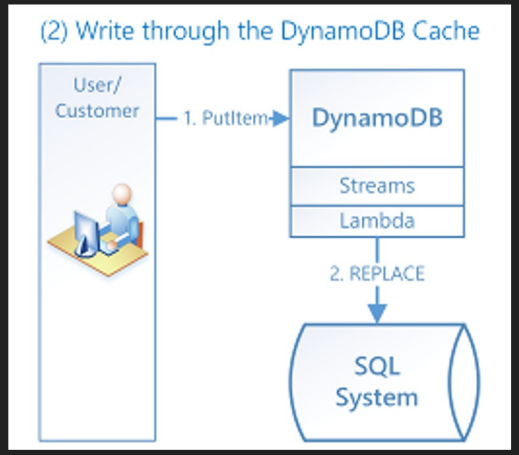
Define LSI on creation of the table[1]

Hybrid DynamoDB–RDBMS

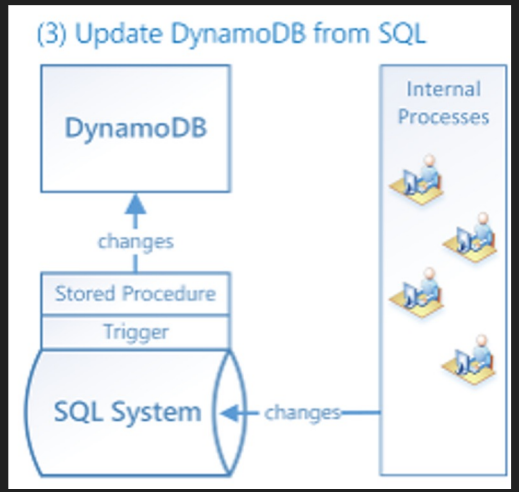
- Sometimes organizations might not find it beneficial to migrate from RDBMS to DynamoDB
- You can setup a hybrid system where DynamoDB creates a materialized view of the data stored in the RDBMS and handles high-traffic requests against this view



- When an item is queried, look for it in DynamoDB
- If it is not there, look for it in the SQL System and load it into DynamoDB



- When a customer changes a value in the DynamoDB, a lambda function is triggered to write the data to the SQL system



- When a value is changed in the SQL system, a stored procedure is triggered to update the value in the DynamoDB materialized view

Hybrid DynamoDB-RDBMS[1]

PartiQL

- PartiQL is a SQL-Compatible query language for DynamoDB
- You can select, insert, update and delete data. You can interact with DynamoDB tables and run ad hoc queries

Create an item in the Music table

```
aws dynamodb execute-statement --statement "INSERT INTO Music  VALUE  {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

Retrieve an item from the Music table

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
```

Performing Transactions with PartiQL

You cannot mix both read and write statements in one transaction. You can either have only read or only write operations

Save the following json

```
[
   {
      "Statement": "EXISTS(SELECT * FROM Music where Artist='No One You Know' and SongTitle='Call Me Today' and Awards is  MISSING)"
   },
   {
      "Statement": "INSERT INTO Music value {'Artist':'?','SongTitle':'?'}",
      "Parameters": [{"S": "Acme Band"}, {"S": "Best Song"}]
   },
   {
      "Statement": "UPDATE Music SET AwardsWon=1 SET AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and SongTitle='PartiQL Rocks'"
   }
]
```

Execute the transaction

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

Performing transactions with PartiQL[1]

References

[1] https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

- Core Components
- Creating a table
- Working with Indexes
- Hybrid DynamoDB-RDBMS
- Performing transactions with PartiQL

[2] https://www.dynamodbguide.com/