DynamoDB Simple

Before We Begin

AWS actually offers a local DynamoDB that you can set up and you can use the `--endpoint-url http://localhost:8000` to run the command rather than paying the money and hosting it on AWS.

Instructions

```
curl -O https://s3-us-west-2.amazonaws.com/dynamodb-local/dynamodb_local_latest.zip
```

```
unzip dynamodb_local_latest.zip
```

```
rm dynamodb_local_latest.zip
```

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
```

Key Concepts

1. Table: Table in DynamoDB is very similar to table in relational databases. Just how you group a set of related data records together that have the same structure
2. Item: This is what identifies as a single data record in the table. Every item in the table is uniquely identified by the primary key of the table. Similar to a row in a relational databases
3. Attributes: Data that are attached to a single item. A simple attribute could be the age of a user in a User table. This is similar to a column in the relational database. However, DynamoDB does not require attribute on items except the attributes that are part of the primary key

Primary Key

Every item is identified by a primary key. This must be defined at the creation of the table and must be provided when inserting a new item.

There are two types of primary key used for DynamoDB

1. Simple primary key

Simple primary jet is just similar to standard key-value stores. You have one key mapped to the value.

Simple primary key is also referred to as a partition key, the partition key you specified will be used as part of a hash calculation, which determines which partition the actual record will be stored.

2. Composite primary key

A little bit more complex, composite primary key you would specify both a partition key and a sort key. The sort key is used to sort the items within the same partition.

The partition key works the same as the simple primary key, it will be used as part of the hash calculation in order to determine which partition the record will be end up at. Then items that are part of the same partition, i.e. have the partition key value, will be ordered by the sorting key.

So it is possible a value to have same partition key, however, they cannot have the same sorting key.

Secondary indexes

The primary key uniquely identifies an item in the table. However, sometimes you have additional access patterns that requires querying with attributes that are not part of the primary key. It would be easier to query with something else.

Secondary indexes give you this ability of query with things other than the primary key.

Again two type of secondary indexes:

1. Local secondary index

Local secondary index uses the same partition key as the table but you can specify a different sorting key. This allows you to basically query but with a different sorting key

2. Global secondary index

Global secondary index on the other hand allows you to define an entirely different primary for the table, you can pick a different partition key and the sorting key.

If you make a secondary index you can pick which attributes will be copied or projected to the new index that you have created, or basically which field it will show up when you query the secondary index. At minimum if you don't specify anything then DynamoDB will output the primary key that

are used in the base table and on the secondary index.

More about attributes

Since DynamoDB is a NoSQL database, most attributes are not required for inserting for every item. If you don't specify it then DynamoDB will just not put anything for it, it is more flexible compared to SQL databases.

The only exception is that the primary key must have attribute(s) (depending on whether or not it is a simple or composite primary key).

In addition, you only need to define the primary key upfront for DynamoDB. You can add attributes in the future by just inserting them into the table, you do not need to change any schema or anything like that like SQL databases.

Attribute types

Attribute types ranges from strings, numbers, lists, maps, and sets.

The way that you specify types in AWS CLI is using short notation of types such as "S" for strings and "N" for numbers.

Querying DynamoDB

When you query the table, you will just specify the primary key. If your item contain many attributes and say you are only interested in one of the many fields, you can specify the `projection-expression` to tell which of the attribute you want to retrieve only.

Expression Attribute Names

You would need to use expression attribute names if your attribute is:

1. A reversed word
2. Your attribute contains a dot
3. Your attribute beings with a number

Expression attribute name is how you would get around those issues if your attribute fulfill one of the condition above, OR you just want a shorter querying syntax.

```
--expression-attribute-names '{
  "#a": "Age"
}'
```

In this case, `#a` is an expression attribute name that is used instead of `Age`. You can use it in pretty much any other flag for example the `projection-expression` flag if you only want to get the Age attribute.

Expression Attribute Values

Very similar to expression attribute names but instead they are used for the values used for comparison with the attribute with an item.

The syntax would be the same except the expression attribute value must start with `:` instead of `#`. In addition, they must also specify the type that they are referencing:

```
{":agelimit": {"N": 21}}
```

Condition Expressions

Condition expression can be used to conditionally carry out an operation for example, insertion or deletion.

Imagine if you are inserting into the table over an existing key, if you just straight up call the `get-item` operation then you are going to overwrite the existing item. You can prevent this by using a conditional expression to make sure before you do the operation there hasn't been a record with the same primary key that exist.

```
aws dynamodb put-item \
  --table-name User \
  --item '{
  "Username": {"S": "Xiang"},
  "Name": {"S": "Xiang Lu"},
  "Age": {"N": "33"}
  }' \
  --expression-attribute-names '{
      "#u": "Username"
  }' \
  --condition-expression "attribute_not_exists(#u)" \
  $LOCAL
```

Here is an example, which will check whether or not the primary key (Username) that we are inserting does not exist first, before we insert. If it exist then the condition will be false, and thus the request will fail. However, if the same username does not exist then the operation will succeed and be carried out.

Updating Items

There are times where you need to update an existing item, and there are couple of operations to help you do that.

1. Set: Add new attribute or modify an existing attribute
2. Remove: Delete attribute from an item
3. Add: Increment/decrement number or insert elements into a set
4. Delete: Remove items from a set

Set operation is the most commonly used one. Here is an example:

```
aws dynamodb update-item \
  --table-name User \
  --key '{
      "Username": {"S": "Ricky"}
  }' \
  --update-expression 'SET #dob = :dob' \
  --expression-attribute-names '{
      "#dob": "DateOfBirth"
  }' \
  --expression-attribute-values '{
      ":dob": {"S": "1937-10-23"}
  }' \
  $LOCAL
```

> If you include the `--return-values 'ALL_NEW'` flag it will show you the items as it exists after the operation without having you to querying it again.

Deleting items

The delete item operation is pretty simple, it just deletes the item that's associated with the primary key.

```
aws dynamodb delete-item \
    --table-name User \
    --key '{
        "Username": {"S": "Ricky"}
    }' \
    $LOCAL
```

Additionally, you can also do some conditional deletion, say you only want to delete this particular primary key if the a certain attribute matches the condition for deletion.

Revision #6
Created 2023-07-07 23:45:20 UTC by Tamarine
Updated 2023-07-09 00:14:51 UTC by Tamarine