CASSANDRA - CQL COLLECTIONS

CQL provides the facility of using Collection data types. Using these Collection types, you can store multiple values in a single variable. This chapter explains how to use Collections in Cassandra.

List

List is used in the cases where

- the order of the elements is to be maintained, and
- a value is to be stored multiple times.

You can get the values of a list data type using the index of the elements in the list.

Creating a Table with List

Given below is an example to create a sample table with two columns, name and email. To store multiple emails, we are using list.

```
cqlsh:tutorialspoint> CREATE TABLE data(name text PRIMARY KEY, email list<text>);
```

Inserting Data into a List

While inserting data into the elements in a list, enter all the values separated by comma within square braces [ ] as shown below.

```
cqlsh:tutorialspoint> INSERT INTO data(name, email) VALUES ('ramu',
['abc@gmail.com','cba@yahoo.com'])
```

Updating a List

Given below is an example to update the list data type in a table called **data**. Here we are adding another email to the list.

```
cqlsh:tutorialspoint> UPDATE data
... SET email = email &plus;['xyz@tutorialspoint.com']
... where name = 'ramu';
```

Verification

If you verify the table using SELECT statement, you will get the following result:

```
cqlsh:tutorialspoint> SELECT * FROM data;

 name | email
------+-----------------------------------------------------------------
 ramu | ['abc@gmail.com', 'cba@yahoo.com', 'xyz@tutorialspoint.com']

(1 rows)
```

SET

Set is a data type that is used to store a group of elements. The elements of a set will be returned in a sorted order.

Creating a Table with Set

The following example creates a sample table with two columns, name and phone. For storing

multiple phone numbers, we are using set.

```
cqlsh:tutorialspoint> CREATE TABLE data2 (name text PRIMARY KEY, phone set<varint>);
```

Inserting Data into a Set

While inserting data into the elements in a set, enter all the values separated by comma within curly braces { } as shown below.

```
cqlsh:tutorialspoint> INSERT INTO data2(name, phone)VALUES ('rahman',
{9848022338,9848022339});
```

Updating a Set

The following code shows how to update a set in a table named data2. Here we are adding another phone number to the set.

```
cqlsh:tutorialspoint> UPDATE data2
   ... SET phone = phone + {9848022330}
   ... where name = 'rahman';
```

Verification

If you verify the table using SELECT statement, you will get the following result:

```
cqlsh:tutorialspoint> SELECT * FROM data2;

   name | phone
--------+---------------------------------------
 rahman | {9848022330, 9848022338, 9848022339}

(1 rows)
```

MAP

Map is a data type that is used to store a key-value pair of elements.

Creating a Table with Map

The following example shows how to create a sample table with two columns, name and address. For storing multiple address values, we are using map.

```
cqlsh:tutorialspoint> CREATE TABLE data3 (name text PRIMARY KEY, address
map<timestamp, text>);
```

Inserting Data into a Map

While inserting data into the elements in a map, enter all the **key : value** pairs separated by comma within curly braces { } as shown below.

```
cqlsh:tutorialspoint> INSERT INTO data3 (name, address)
   VALUES ('robin', {'home' : 'hyderabad' , 'office' : 'Delhi' } );
```

Updating a Set

The following code shows how to update the map data type in a table named data3. Here we are changing the value of the key office, that is, we are changing the office address of a person named robin.

```
cqlsh:tutorialspoint> UPDATE data3
   ... SET address = address+{'office':'mumbai'}
```

```
   ... WHERE name = 'robin';
```

Verification

If you verify the table using SELECT statement, you will get the following result:

```
cqlsh:tutorialspoint> select * from data3;

  name | address
-------+-----------------------------------------
 robin | {'home': 'hyderabad', 'office': 'mumbai'}

(1 rows)
```