ADVANCED DATABASE MANAGEMENT SYSTEM LAB ASSIGNMENT

GROUP-(36-40)

SUBMITTED BY

SHANA PARWEEN

MCA236

TOPIC: Design database schemas and implement min 10 queries using Cassandra column based databases

Aim
To Design database schemas and implement queries using Cassandra databases

Objective(s)	
1	Study of NOSQL Cassandra.
2	Study the procedure to execute a query using Apache Cassandra.
3	Execute min 10 queries using Cassandra column based database.

1.STUDY OF NOSQL CASSANDRA

Cassandra

Apache Cassandra is an open source distributed databasemanagement system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

Features

1) Scalability:
- o Read and write throughput both increase linearly as new machines are added, with no downtime or interruption toapplications.

2) Fault-tolerant:
- o Data is automatically replicated to multiple nodes forfault-tolerance.
- o Replication across multiple data centers issupported.
- o Failed nodes can be replaced with no downtime.

3) MapReducesupport:
- o Cassandra has Hadoop integration, with MapReducesupport.

4) Query language:
- o Cassandra introduces CQL (Cassandra Query Language), a SQL-like alternative to the traditional RPCinterface.
  - Keyspace: Keyspace is the outermost container for data.It is similar to the schema in a relational database.

Basic attributes of Keyspace are:

- Replication Factor: It is the number of machines in the cluster that will receive copies of the same data
- Replica Placement Strategy: It is a strategy to place replicas in the ring
- Simple Strategy,
- Old Network TopologyStrategy
- Network TopologyStrategy

Column Families: Column family is a NoSQL object that contains columns of related data. It is a tuple (pair) that consists of a key-value pair, where the key is mapped to a value that is a set of columns. It is similar to a table in a relational database and each key-value pair being a row.
Each column is a tuple (triplet) consisting of
- o Column name
- o Value
- o Timestamp

1. STUDY THE PROCEDURE TO EXECUTE A QUERY USING APACHE CASSANDRA

Introduction

DataStax Community Edition must be installed on system before installing Cassandra. Verify the Cassandra installation using the followingcommand:

- $ Cassandra version

- If Cassandra is already installed on system, then you will get the following response:

- Connected to Test Cluster at 127.0.0.1:9042.
- [cqlsh 5.0.1 | Cassandra 2.2.4 | CQL spec 3.3.1 | Native protocol v4] Use HELP for help.
- WARNING: pyreadline dependency missing. Install to enable tab completion. From source with checksum 79e53ce7994d1628b240f09af91e1af4

Creating KEYSPACE :

Cassandra is technology that can define databases and tables to analyze structured data. The theme for structured data analysis is to store the data in a tabular manner, and pass queries to analyze it.

Create KEYSPACE Statement

- Create KEYSPACE is a statement used to create a KEYSPACE in Cassandra. A KEYSPACE in Cassandra is aKEYSPACE or a collection of tables. The syntax for this statement is as follows:
  cqlsh>CREATE KEYSPACE ABC userdb replication={ 'class':'SimpleStrategy','replication_factor':'1'};

- Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command.

- <u>The following query is executed to create a database named userdb</u>:

```
cqlsh>userdb;
Or
cqlsh> CREATE SCHEMA userdb;
```

<u>The following query is used to verify a databases list:</u>

```
cqlsh>:userdb> show tables; Improper show command. default
userdb
```

<u>For creating Table:</u>

Create Table

```
CREATE TABLE test_table (
id int, address text, name text,
PRIMARY KEY ((id))
);
```

<u>CURD using cql Updating Table:</u>

Update Table

```
insert into test_table (id, name, address) values (4, 'somnath', 'Sus');
```

<u>CURD using cql Delete Table</u>

<u>Deleting rows from Table</u>

```
delete from test_table where id =1;
```

The following queries are used to drop a database. Let us assume that the database name is userdb.

```
cqlsh:userdb> delete from Tablename where condition;
```

<u>For describing tables</u>

```
cqlsh:userdb> describe tables; show all table names
```

```
cqlsh:userdb>
```

For Help of any Topic

```
Cqshl> Help;
```

Display topics

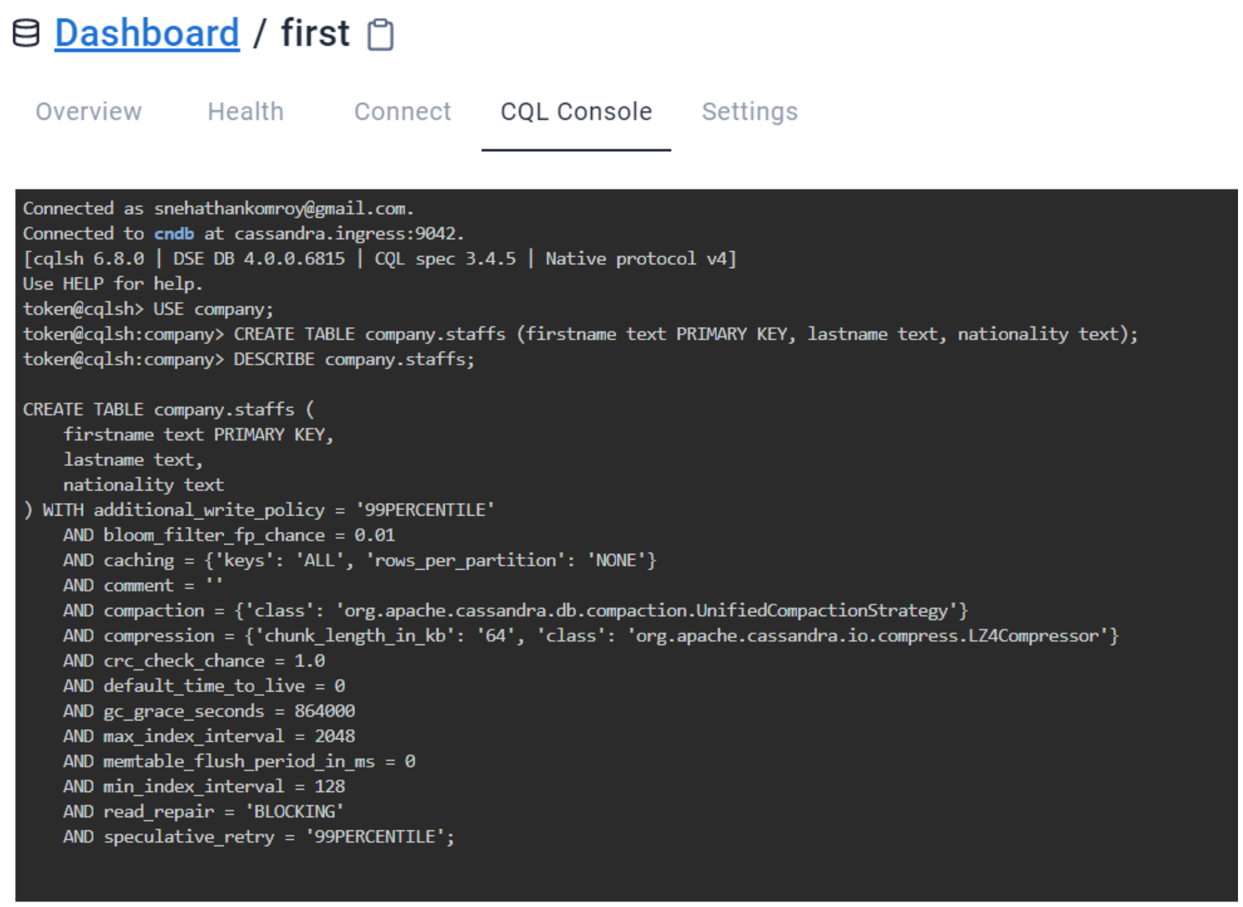
```
Cqshl> Help topic name;
```

Help open in Browser.

3.Execute minimum 10 queries using Cassandra column-based database

INITIAL PROCEDURE:

Step 1: Here we consider the database “First”
Step 2: Creating and updating a keyspace ,herekeyspace is “company”
Step 3: Creating a table “staffs”
Step 4: Describe table “staffs”



```
Connected as snehathankomroy@gmail.com.
Connected to cndb at cassandra.ingress:9042.
[cqlsh 6.8.0 | DSE DB 4.0.0.6815 | CQL spec 3.4.5 | Native protocol v4]
Use HELP for help.
token@cqlsh> USE company;
token@cqlsh:company> CREATE TABLE company.staffs (firstname text PRIMARY KEY, lastname text, nationality text);
token@cqlsh:company> DESCRIBE company.staffs;

CREATE TABLE company.staffs (
    firstname text PRIMARY KEY,
    lastname text,
    nationality text
) WITH additional_write_policy = '99PERCENTILE'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.UnifiedCompactionStrategy'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99PERCENTILE';
```

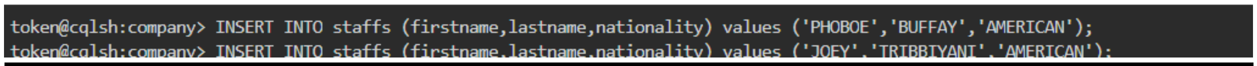
QUERIES

1.INSERTING DATA INTO A TABLE

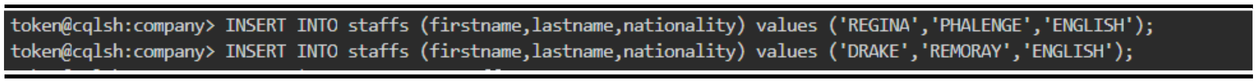
Procedure:

- INSERT INTO staffs (firstname, lastname, nationality) VALUES ('PHOBOE',’BUFFAY’,’AMERICAN’);

OUTPUT



```
token@cqlsh:company> INSERT INTO staffs (firstname,lastname,nationality) values ('PHOBOE','BUFFAY','AMERICAN');
token@cqlsh:company> INSERT INTO staffs (firstname,lastname,nationality) values ('JOEY','TRIBBIYANI','AMERICAN');
```



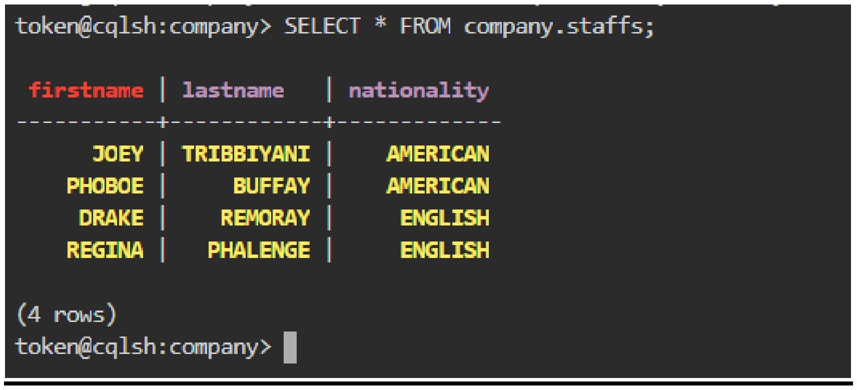
```
token@cqlsh:company> INSERT INTO staffs (firstname,lastname,nationality) values ('REGINA','PHALENGE','ENGLISH');
token@cqlsh:company> INSERT INTO staffs (firstname,lastname,nationality) values ('DRAKE','REMORAY','ENGLISH');
```

2.RETRIEVAL OF ALL DATA

Procedure:

- SELECT * FROM company.staffs;

OUTPUT



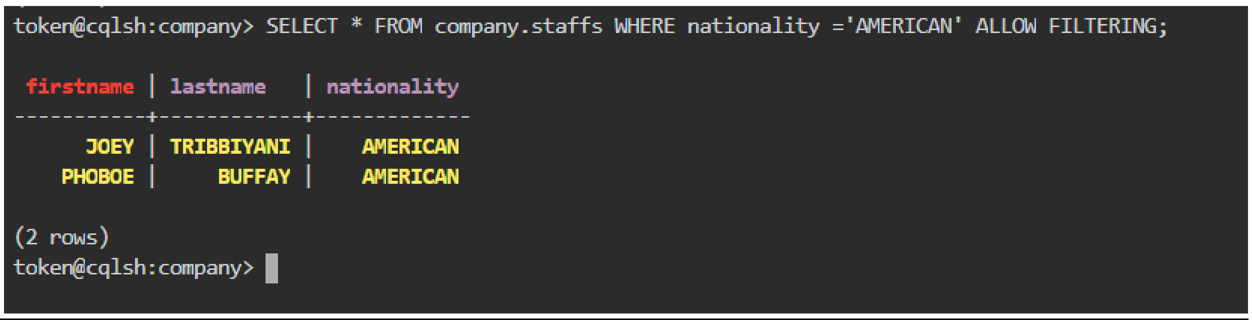
```
token@cqlsh:company> SELECT * FROM company.staffs;

 firstname | lastname   | nationality
-----------+------------+-------------
      JOEY | TRIBBIYANI |    AMERICAN
    PHOBOE |     BUFFAY |    AMERICAN
     DRAKE |    REMORAY |     ENGLISH
    REGINA |   PHALENGE |     ENGLISH

(4 rows)
token@cqlsh:company>
```

3.RETRIEVE THE DETAILS OF STAFFS USING CONDITIONS

<u>Procedure:</u>

- SELECT * FROM company.staffs WHERE nationality = ‘ENGLISH' ALLOW FILTERING;

<u>OUTPUT</u>



```
token@cqlsh:company> SELECT * FROM company.staffs WHERE nationality ='AMERICAN' ALLOW FILTERING;

 firstname | lastname   | nationality
-----------+------------+-------------
      JOEY | TRIBBIYANI |    AMERICAN
    PHOBOE |     BUFFAY |    AMERICAN

(2 rows)
token@cqlsh:company> 
```

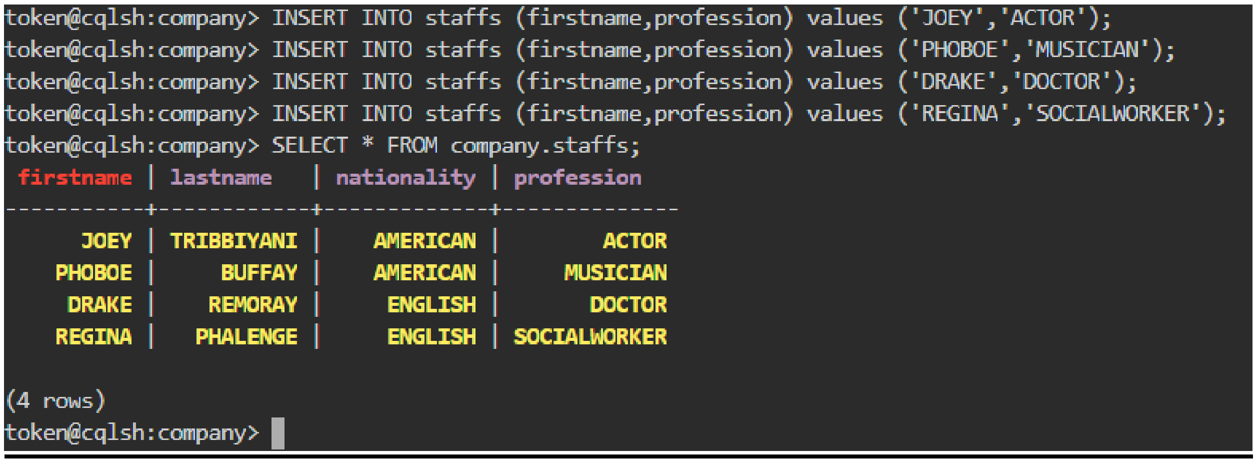
2.ALTER TABLE command

- Adding new column’profession’ into the table ‘staffs’

<u>Procedure:</u>

- ALTER TABLE staffs ADD profession text;
- DESCRIBE TABLE staffs;
- Add Data as into that column using insert command
- Then displayed the details using select command

<u>OUTPUT</u>



```
token@cqlsh:company> INSERT INTO staffs (firstname,profession) values ('JOEY','ACTOR');
token@cqlsh:company> INSERT INTO staffs (firstname,profession) values ('PHOBOE','MUSICIAN');
token@cqlsh:company> INSERT INTO staffs (firstname,profession) values ('DRAKE','DOCTOR');
token@cqlsh:company> INSERT INTO staffs (firstname,profession) values ('REGINA','SOCIALWORKER');
token@cqlsh:company> SELECT * FROM company.staffs;

 firstname | lastname   | nationality | profession
-----------+------------+-------------+--------------
      JOEY | TRIBBIYANI |    AMERICAN |        ACTOR
    PHOBOE |     BUFFAY |    AMERICAN |     MUSICIAN
     DRAKE |    REMORAY |     ENGLISH |       DOCTOR
    REGINA |   PHALENGE |     ENGLISH | SOCIALWORKER

(4 rows)
token@cqlsh:company>
```

4.RETRIEVING TIMESTAMPS

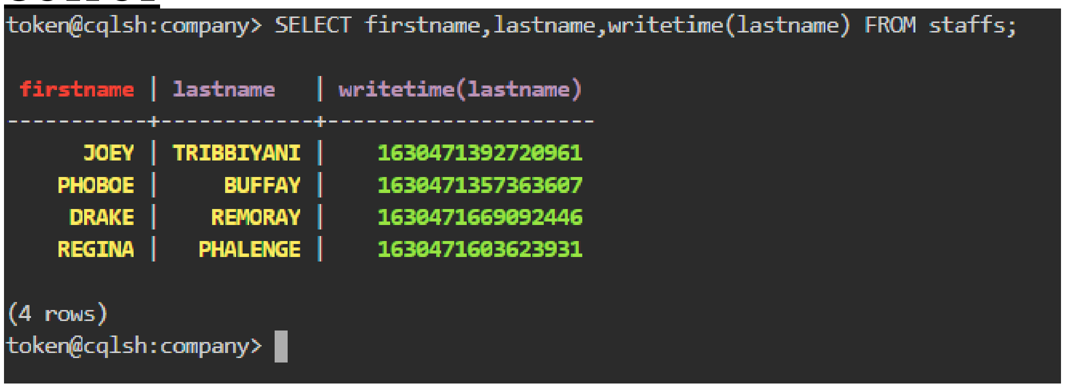
Timestamps

Each time you write data into Cassandra, a timestamp is generated for each column value that is updated. Internally, Cassandra uses these timestamps for resolving any conflicting changes that are made to the same value.

Procedure:

- SELECT firstname ,lastname , writetime(lastname) FROM staffs;

OUTPUT



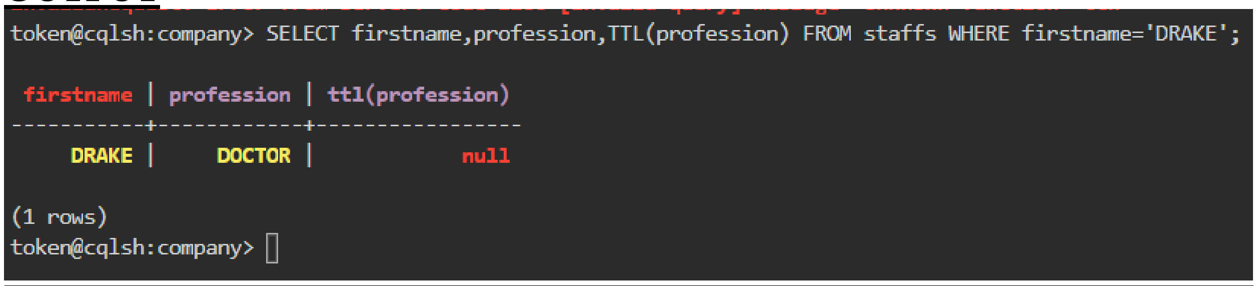
```
token@cqlsh:company> SELECT firstname,lastname,writetime(lastname) FROM staffs;

 firstname | lastname   | writetime(lastname)
-----------+------------+---------------------
      JOEY | TRIBBIYANI |    1630471392720961
    PHOBOE |     BUFFAY |    1630471357363607
     DRAKE |    REMORAY |    1630471669092446
    REGINA |   PHALENGE |    1630471603623931

(4 rows)
token@cqlsh:company>
```

5.RETRIEVE THE TTL VALUE FOR ‘DRAKE’S PROFESSION

<u>Procedure:</u>

- SELECT firstname, profession, TTL(profession) FROM staffs WHERE firstname = ‘DRAKE';

<u>OUTPUT</u>



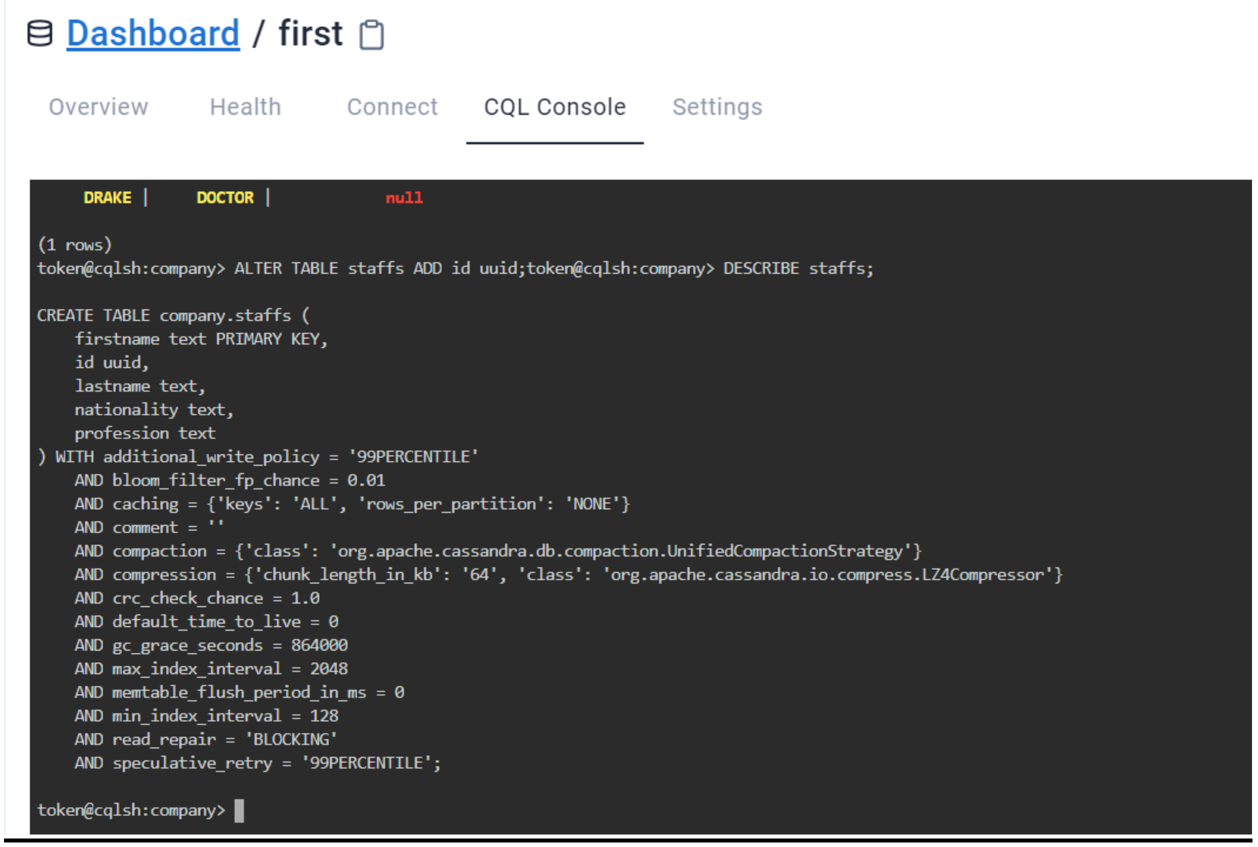
```
token@cqlsh:company> SELECT firstname,profession,TTL(profession) FROM staffs WHERE firstname='DRAKE';

 firstname | profession | ttl(profession)
-----------+------------+-----------------
     DRAKE |     DOCTOR |            null

(1 rows)
token@cqlsh:company>
```

6. QUERY TO ADD THE IDENTIFIER USING A UUID

<u>Procedure:</u>

- ALTER TABLE staffs ADD id uuid;
- DESCRIBE staffs;

OUTPUT



```
     DRAKE |      DOCTOR |            null

(1 rows)
token@cqlsh:company> ALTER TABLE staffs ADD id uuid;token@cqlsh:company> DESCRIBE staffs;

CREATE TABLE company.staffs (
    firstname text PRIMARY KEY,
    id uuid,
    lastname text,
    nationality text,
    profession text
) WITH additional_write_policy = '99PERCENTILE'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.UnifiedCompactionStrategy'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99PERCENTILE';

token@cqlsh:company> ▌
```

7.QUERY TO INSERT AN ID FOR JOEY USING UUID() FUNCTION AND THEN VIEW THE RESULTS

uuid
A universally unique identifier (UUID) is a 128-bit value in which the bits conform to one of several types, of which the most commonly used are known as Type 1 and Type 4. The CQL uuid type is a Type 4 UUID, which is based entirely on random numbers. UUIDs are typically represented as dash-separated sequences of hex digits. For example:

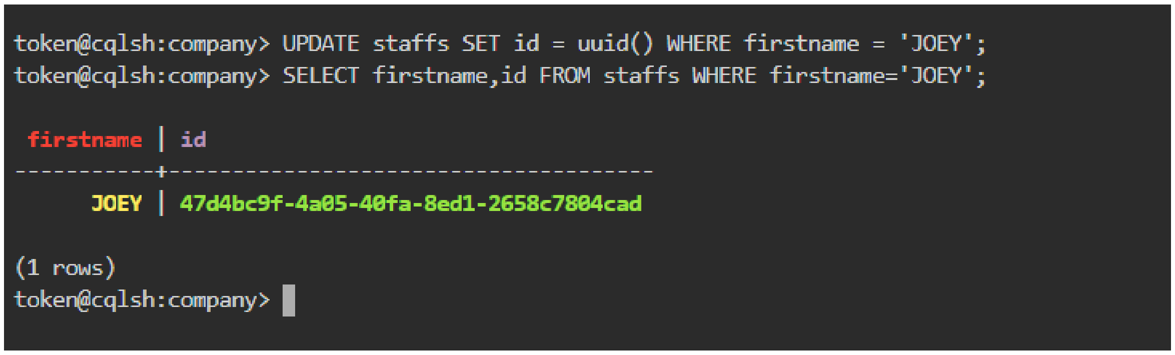
1a6300ca-0572-4736-a393-c0b7229e193e

The uuid type is often used as a surrogate key, either by itself or in combination with other values.

Procedure:

- UPDATE staffs SET id = uuid() WHERE firstname = ‘JOEY';
- SELECT firstname, id FROM staffs WHERE firstname=‘JOEY';

OUTPUT



```
token@cqlsh:company> UPDATE staffs SET id = uuid() WHERE firstname = 'JOEY';
token@cqlsh:company> SELECT firstname,id FROM staffs WHERE firstname='JOEY';

 firstname | id
-----------+--------------------------------------
      JOEY | 47d4bc9f-4a05-40fa-8ed1-2658c7804cad

(1 rows)
token@cqlsh:company>
```

8.QUERY TO MODIFY OUR STAFFS TABLE TO ADD A SET OF EMAIL ADDRESSES

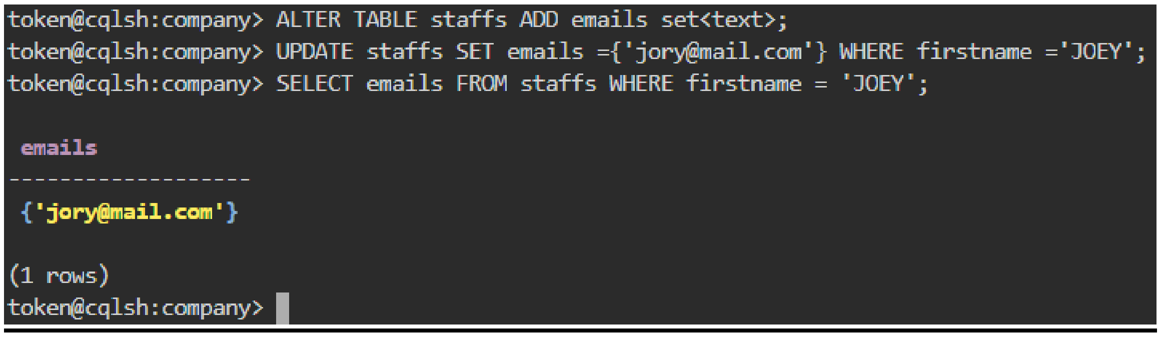
set

The set data type stores a collection of elements. The elements are unordered, but cqlsh returns the elements in sorted order. For example, text values are returned in alphabetical order. One advantage of using set is the ability to insert additional items without having to read the contents first.

<u>Procedure:</u>

- ALTER TABLE staffs ADD emails set<text>;
- UPDATE staffs SET emails = { 'jory@mail.com' } WHERE firstname = ‘JOEY';
- SELECT emails FROM staffs WHERE firstname = 'JOEY';

OUTPUT



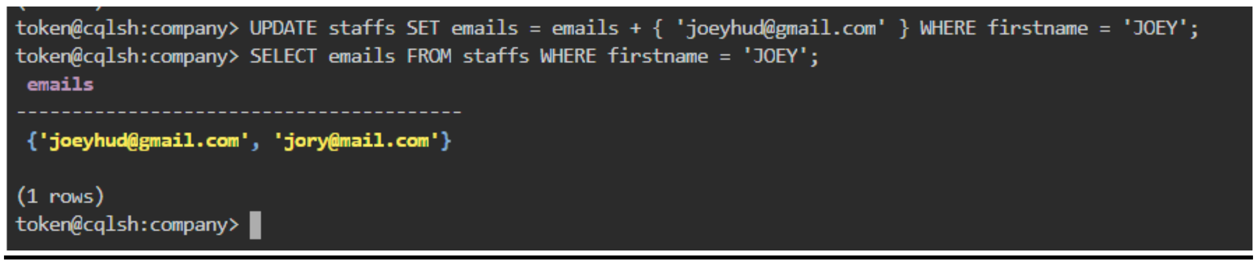
```
token@cqlsh:company> ALTER TABLE staffs ADD emails set<text>;
token@cqlsh:company> UPDATE staffs SET emails ={'jory@mail.com'} WHERE firstname ='JOEY';
token@cqlsh:company> SELECT emails FROM staffs WHERE firstname = 'JOEY';

 emails
-------------------

 {'jory@mail.com'}

(1 rows)
token@cqlsh:company>
```

9.QUERY TO ADD ANOTHER EMAIL ADDRESS WITHOUT REPLACING THE WHOLE SET BY USING CONCATENATION

<u>Procedure:</u>

- UPDATE staffs SET emails = emails + {‘joeyhud@gmail.com' } WHERE firstname = 'JOEY';
- SELECT emails FROM staffs WHERE firstname = 'JOEY';

<u>OUTPUT</u>



```
token@cqlsh:company> UPDATE staffs SET emails = emails + { 'joeyhud@gmail.com' } WHERE firstname = 'JOEY';
token@cqlsh:company> SELECT emails FROM staffs WHERE firstname = 'JOEY';
 emails
----------------------------------------
 {'joeyhud@gmail.com', 'jory@mail.com'}

(1 rows)
token@cqlsh:company>
```

10.*QUERY TO MODIFY OUR STAFFS TABLE TO ADD A LIST OF PINCODE AND ADD A PINCODE FOR REGINA AND CHECK THAT IT WAS ADDED SUCCESSFULLY*

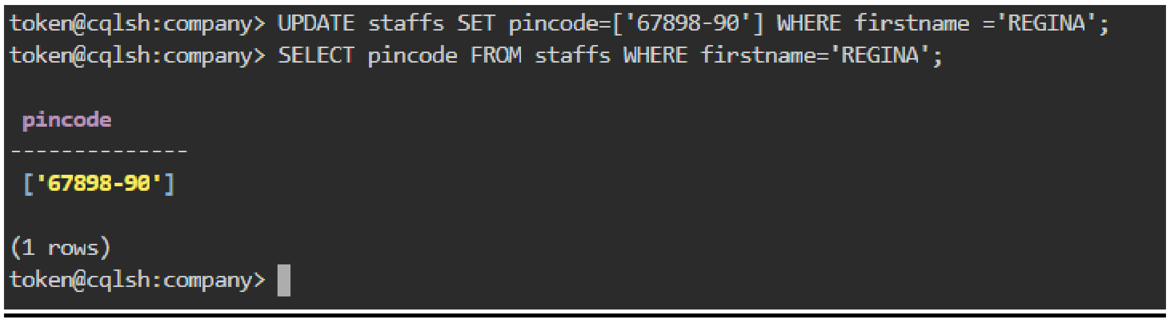
List

The list data type contains an ordered list of elements. By default, the values are stored in order of insertion.

Procedure:

- ALTER TABLE staffs ADD pincode list<text>;
- UPDATE staffs SET pincode = ['67898-90'] WHERE firstname = ‘REGINA';
- SELECT pincode FROM staffs WHERE firstname = ‘REGINA';

OUTPUT



```
token@cqlsh:company> UPDATE staffs SET pincode=['67898-90'] WHERE firstname ='REGINA';
token@cqlsh:company> SELECT pincode FROM staffs WHERE firstname='REGINA';

 pincode
--------------
 ['67898-90']

(1 rows)
token@cqlsh:company>
```