

ADVANCED DATABASE MANAGEMENT
SYSTEM
ASSIGNMENT NO: 3

SUBMITTED BY,
GROUP II (6-10)
AMJIYAD AHSAN K
ANANTHA PADMANABHAN
ANANYA B
ANILAMOL CHACKO
ANUSHREE

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 database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

Features

1) Scalability:

- Read and write throughput both increase linearly as new machines are added, with no downtime or interruption to applications.

2) Fault-tolerant:

- Data is automatically replicated to multiple nodes for fault-tolerance.
- Replication across multiple data centers is supported.
- Failed nodes can be replaced with no downtime.

3) MapReduce support:

- Cassandra has Hadoop integration, with MapReduce support.

4) Query language:

- Cassandra introduces CQL (Cassandra Query Language), a SQL-like alternative to the traditional RPC interface.
- **Key space:** Key space is the outermost container for data. It is similar to the schema in a relational database.

Basic attributes of Key space 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 Topology Strategy
- Network Topology Strategy

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

- Column name
- Value
- 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 following command:

- **\$ 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.
- The following query is executed to create a database named userdb:

cqlsh> userdb;

Or

cqlsh> CREATE SCHEMA userdb;

The following query is used to verify a databases list:

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

userdb

For creating Table:

Create Table

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

CURD using cql Updating Table:

Update Table

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

CURD using cql Delete Table

Deleting rows from Table

```
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;
```

For describing tables

```
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 “users”

Step 2: Creating and updating a keyspace ,here keyspace is “company”

Step 3: Creating a table “staffs”

Step 4: Describe table “staffs”

 [Dashboard](#) / users 

 Load Data

 Connect

Overview

Health

Connect

CQL Console

Settings

```
Connected as anushreetp15@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> 
```

 [Dashboard](#) / Users 

 Load Data

 Connect

Overview

Health

Connect

CQL Console

Settings

```
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';

token@cqlsh:company> 
```


QUERIES

1. INSERTING DATA INTO A TABLE

Procedure:

- INSERT INTO staffs (firstname, lastname, nationality) VALUES ('Bill', 'Nguyen', 'American');
- INSERT INTO staffs (firstname, lastname, nationality) VALUES ('Bob', 'Noyn', 'Indian');
- INSERT INTO staffs (firstname, lastname, nationality) VALUES ('Mary', 'Mar', 'Indian');
- INSERT INTO staffs (firstname, lastname, nationality) VALUES ('Luke', 'zam', 'Canadian');
- INSERT INTO staffs (firstname, lastname, nationality) VALUES ('Smith', 'Benny', 'Italian');

OUTPUT

 [Dashboard](#) / [users](#) 

[Load Data](#) [Connect](#)

[Overview](#) [Health](#) [Connect](#) [CQL Console](#) [Settings](#)

```
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> INSERT INTO staffs (firstname, lastname, nationality) VALUES('Bill', 'Nguyen', 'American');
token@cqlsh:company> INSERT INTO staffs (firstname, lastname, nationality) VALUES('Bob', 'Noyn', 'Indian' );
token@cqlsh:company> INSERT INTO staffs (firstname, lastname, nationality) VALUES('Mary', 'Mar', 'Indian' );
token@cqlsh:company> INSERT INTO staffs( firstname, lastname, nationality) VALUES('Luke', 'Zam', 'Canadian' );
token@cqlsh:company> INSERT INTO staffs (firstname, lastname, nationality) VALUES('Smith', 'Benny', 'Italian' );
token@cqlsh:company> █
```

2.RETRIEVAL OF ALL DATA

Procedure:

- SELECT * FROM company.staffs;

OUTPUT

 [Dashboard](#) / [users](#) 

[Load Data](#) [Connect](#)

[Overview](#) [Health](#) [Connect](#) [CQL Console](#) [Settings](#)

```
token@cqlsh:company> INSERT INTO staffs( firstname, lastname, nationality) VALUES('Luke', 'Zam', 'Canadian' );
token@cqlsh:company> INSERT INTO staffs (firstname, lastname, nationality) VALUES('Smith', 'Benny', 'Italian' );
token@cqlsh:company>
token@cqlsh:company> SELECT * FROM company.staffs;
```

firstname	lastname	nationality
Mary	Mar	Indian
Bill	Nguyen	American
Bob	Noyu	Indian
Smith	Benny	Italian
Luke	Zam	Canadian

(5 rows)
token@cqlsh:company>

3.RETRIEVE THE DETAILS OF STAFFS USING CONDITIONS

Procedure:

- SELECT * FROM company.staffs WHERE nationality = 'Indian' ALLOW FILTERING;
- SELECT * FROM company.staffs WHERE firstname = 'Bill' ALLOW FILTERING;

OUTPUT

Dashboard / users

Load Data Connect

Overview Health Connect CQL Console Settings

Mary	Mar	Indian
Bill	Nguyen	American
Bob	Noyn	Indian
Smith	Benny	Italian
Luke	Zam	Canadian

(5 rows)

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

firstname	lastname	nationality
Mary	Mar	Indian
Bob	Noyn	Indian

(2 rows)

```
token@cqlsh:company> 
```

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

firstname	lastname	nationality
Mary	Mar	Indian
Bob	Noyn	Indian

(2 rows)

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

firstname	lastname	nationality
Smith	Benny	Italian

(1 rows)

```
token@cqlsh:company> 
```

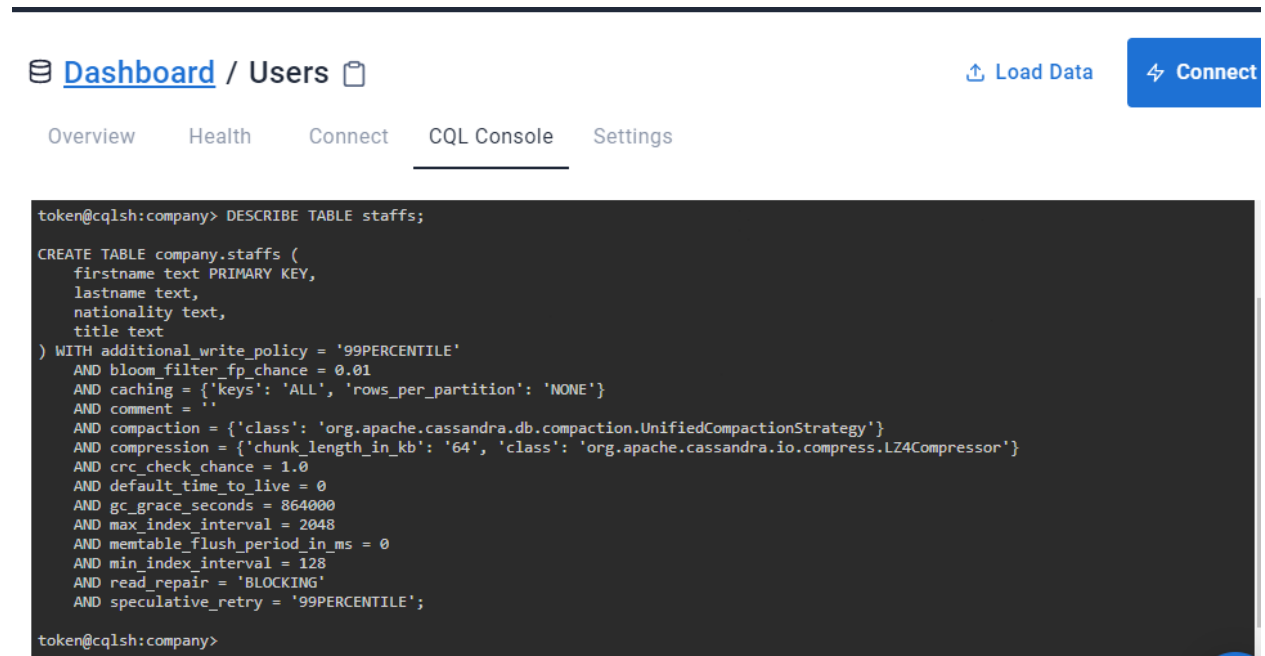
2. ALTER TABLE command

- Adding new column 'title' into the table 'staffs'

Procedure:

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

OUTPUT



The screenshot shows a web-based database interface. At the top, there's a navigation bar with 'Dashboard / Users' and a menu icon on the left, and 'Load Data' and 'Connect' buttons on the right. Below this is a sub-navigation bar with 'Overview', 'Health', 'Connect', 'CQL Console' (which is underlined), and 'Settings'. The main content area is a dark-themed terminal window. It shows the command 'token@cqlsh:company> DESCRIBE TABLE staffs;' followed by the output of the command, which is the CREATE TABLE statement for 'company.staffs'. The output includes column definitions (firstname, lastname, nationality, title) and various table options like write_policy, bloom_filter, caching, compaction, compression, and gc settings.

```
token@cqlsh:company> DESCRIBE TABLE staffs;

CREATE TABLE company.staffs (
  firstname text PRIMARY KEY,
  lastname text,
  nationality text,
  title 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>
```

```
token@cqlsh:company> INSERT INTO staffs (firstname, lastname, nationality, title) VALUES('Bob', 'Noyn', 'Indian', 'Mr');
token@cqlsh:company> INSERT INTO staffs (firstname, lastname, nationality, title) VALUES('Mary', 'Mar', 'Indian', 'Mrs' );
token@cqlsh:company> INSERT INTO staffs( firstname, lastname, nationality, title) VALUES('Luke', 'Zam', 'Canadian', 'Mr' );
token@cqlsh:company> INSERT INTO staffs (firstname, lastname, nationality, title) VALUES('Smith', 'Benny', 'Italian', 'Mr' );
token@cqlsh:company> SELECT * FROM company.staffs;
```

firstname	lastname	nationality	title
Mary	Mar	Indian	Mrs
Bill	Nguyen	American	Mr
Bob	Noyn	Indian	Mr
Smith	Benny	Italian	Mr
Luke	Zam	Canadian	Mr

(5 rows)

```
token@cqlsh:company> 
```

4.RETRIEVING TIMESTAMPS

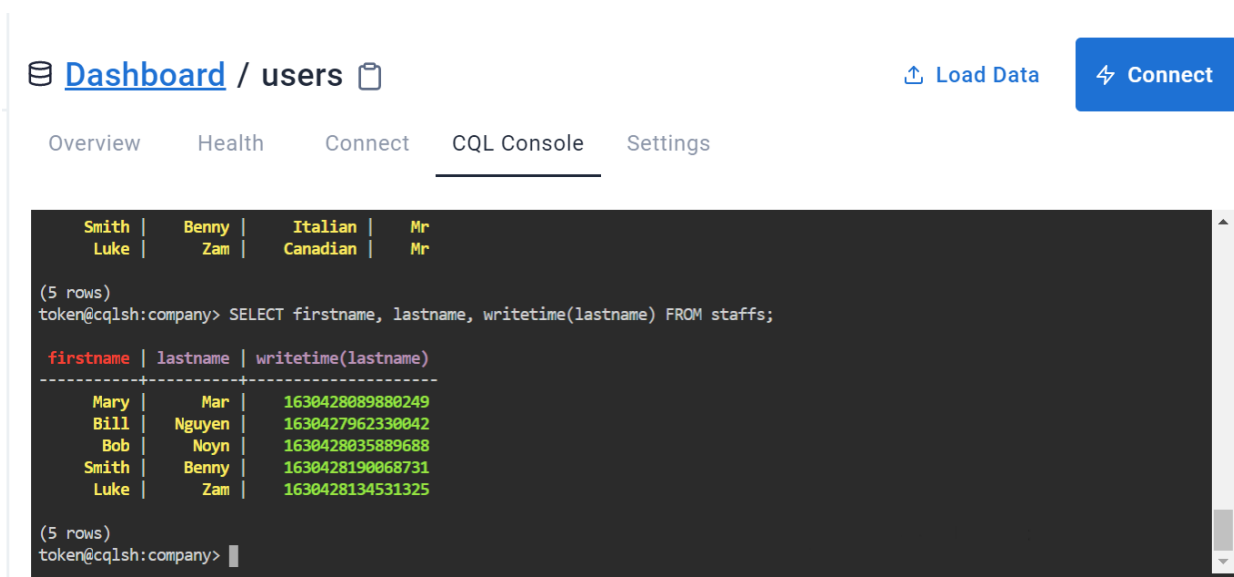
Timestamps

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



The screenshot shows a web-based CQL console interface. At the top, there's a navigation bar with 'Dashboard / users' and buttons for 'Load Data' and 'Connect'. Below this is a tabbed interface with 'Overview', 'Health', 'Connect', 'CQL Console' (selected), and 'Settings'. The main area displays a query result for the command: `token@cqlsh:company> SELECT firstname, lastname, writetime(lastname) FROM staffs;`. The result shows 5 rows of data in a table format with columns: `firstname`, `lastname`, and `writetime(lastname)`. The data rows are: Mary | Mar | 1630428089880249, Bill | Nguyen | 1630427962330042, Bob | Noyn | 1630428035889688, Smith | Benny | 1630428190068731, and Luke | Zam | 1630428134531325.

firstname	lastname	writetime(lastname)
Mary	Mar	1630428089880249
Bill	Nguyen	1630427962330042
Bob	Noyn	1630428035889688
Smith	Benny	1630428190068731
Luke	Zam	1630428134531325


5.RETRIEVE THE TTL VALUE FOR ‘ Mary’s lastname’

Procedure:

- SELECT firstname, lastname, TTL(lastname) FROM staffs WHERE firstname = 'Mary';

OUTPUT

 [Dashboard](#) / users 

 Load Data

 Connect

Overview

Health

Connect

CQL Console

Settings

```
(5 rows)
Mary | Mar | 1630428089880249
Bill | Nguyen | 1630427962330042
Bob | Noyn | 1630428035889688
Smith | Benny | 1630428190068731
Luke | Zam | 1630428134531325

token@cqlsh:company> SELECT firstname, lastname, TTL(lastname) FROM staffs WHERE firstname = 'Mary';

firstname | lastname | ttl(lastname)
-----+-----+-----
Mary | Mar | null

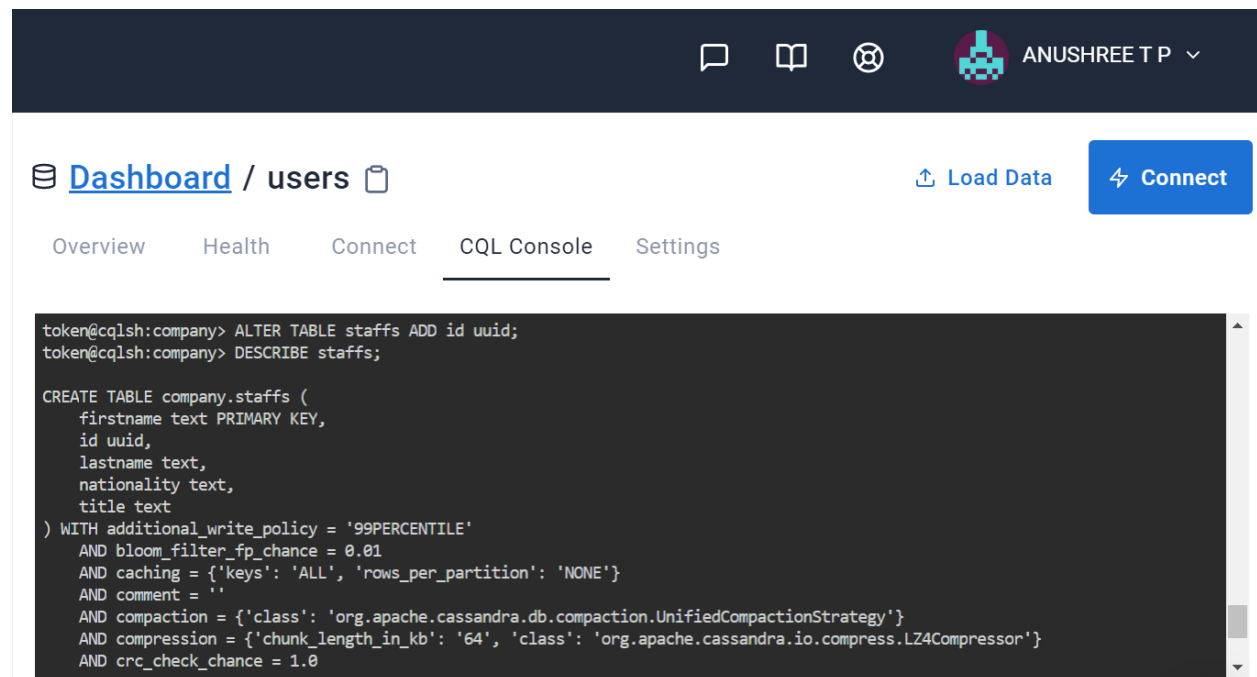
(1 rows)
token@cqlsh:company> 
```

6. QUERY TO ADD THE IDENTIFIER USING A UUID

Procedure:

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

OUTPUT



The screenshot shows the CQL Console interface. At the top, there's a dark blue header with icons for chat, book, user, and a profile icon labeled 'ANUSHREE T P'. Below the header, there's a navigation bar with 'Dashboard / users' and buttons for 'Load Data' and 'Connect'. The main content area has tabs for 'Overview', 'Health', 'Connect', 'CQL Console' (which is active), and 'Settings'. The CQL Console shows the following queries and output:

```
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,
  title 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
```

7.QUERY TO INSERT AN ID FOR MARY 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 = 'Mary';
- SELECT firstname, id FROM staffs WHERE firstname = 'Mary';

OUTPUT



The screenshot shows a database interface with a top navigation bar containing 'Dashboard / users', 'Load Data', and 'Connect' buttons. Below the navigation bar are tabs for 'Overview', 'Health', 'Connect', 'CQL Console', and 'Settings'. The 'CQL Console' tab is active, displaying a series of SQL queries and their results. The queries executed are:
1. A series of ALTER TABLE statements for the 'staffs' table:
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';
2. UPDATE staffs SET id = uuid() WHERE firstname = 'Mary';
3. SELECT firstname, id FROM staffs WHERE firstname = 'Mary';
The output of the third query is displayed in a table with two columns: 'firstname' and 'id'. The table contains one row with the values 'Mary' and '1fb100af-9f5e-403a-9f35-e494a9c12381'. Below the table, it indicates '(1 rows)' and shows the prompt 'token@cqlsh:company>'.

```
Dashboard / users Load Data Connect

Overview Health Connect CQL Console Settings

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

+-----+-----+
|firstname|id      |
+-----+-----+
| Mary    | 1fb100af-9f5e-403a-9f35-e494a9c12381 |
+-----+-----+

(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.

Procedure:

- ALTER TABLE staffs ADD emails set<text>;
- UPDATE staffs SET emails = { 'mary@example.com' } WHERE firstname = 'Mary';
- SELECT emails FROM staffs WHERE firstname = 'Mary';

OUTPUT

 [Dashboard](#) / users 

 Load Data

 Connect

Overview Health Connect CQL Console Settings

```
firstname | id
-----+-----
Mary | 1fb100af-9f5e-403a-9f35-e494a9c12381

(1 rows)
token@cqlsh:company> ALTER TABLE staffs ADD emails set<text>;
token@cqlsh:company> UPDATE staffs SET emails = { 'mary@example.com' } WHERE firstname = 'Mary';
token@cqlsh:company> SELECT emails FROM staffs WHERE firstname = 'Mary';

emails
-----
{'mary@example.com'}

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

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

Procedure:

- UPDATE staffs SET emails = emails + { 'mary.mcdonald.AZ@gmail.com' } WHERE firstname = 'Mary';
- SELECT emails FROM staffs WHERE firstname = 'Mary';

OUTPUT

 [Dashboard](#) / users 

 Load Data

 Connect

Overview

Health

Connect

CQL Console

Settings

```
token@cqlsh:company> SELECT emails FROM staffs WHERE firstname = 'Mary';

emails
-----
{'mary@example.com'}

(1 rows)
token@cqlsh:company> UPDATE staffs SET emails = emails + { 'mary.mar29@gmail.com' } WHERE firstname = 'Mary';
token@cqlsh:company> SELECT emails FROM staffs WHERE firstname = 'Mary';

emails
-----
{'mary.mar29@gmail.com', 'mary@example.com'}

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

10.QUERY TO MODIFY OUR STAFFS TABLE TO ADD A LIST OF PHONE NUMBERS AND ADD A PHONE NUMBER FOR MARY 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 phone_numbers list<text>;
- UPDATE staffs SET phone_numbers = ['1-800-999-9999']
WHERE firstname = 'Mary';
- SELECT phone_numbers FROM staffs WHERE firstname =
'Mary';

OUTPUT

 [Dashboard](#) / users 

 Load Data

 Connect

Overview Health Connect CQL Console Settings

```
emails
-----
{'mary.mar29@gmail.com', 'mary@example.com'}

(1 rows)
token@cqlsh:company> ALTER TABLE staffs ADD phone_numbers list<text>;
token@cqlsh:company> UPDATE staffs SET phone_numbers = [ '1-800-999-9999' ]
... WHERE firstname = 'Mary';
token@cqlsh:company> SELECT phone_numbers FROM staffs WHERE firstname = 'Mary';

phone_numbers
-----
[ '1-800-999-9999' ]

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