

ADVANCED DATABASE MANAGEMENT SYSTEM LAB
ASSIGNMENT

GROUP-(36-40)

SUBMITTED BY
SIJI JOSE
TKM20MCA-2037

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 and decentralized/distributed storage system (database), for managing very large amounts of structured data spread out across the world. It provides highly available service with no single point of failure.

Listed below are some of the notable points of Apache Cassandra –

- It is scalable, fault-tolerant, and consistent.
- It is a column-oriented database.
- Its distribution design is based on Amazon's Dynamo and its data model on Google's Bigtable.
- Created at Facebook, it differs sharply from relational database management systems.
- Cassandra implements a Dynamo-style replication model with no single point of failure, but adds a more powerful "column family" data model.
- Cassandra is being used by some of the biggest companies such as Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.

Features of Cassandra

Cassandra has become so popular because of its outstanding technical features. Given below are some of the features of Cassandra:

- Elastic scalability – Cassandra is highly scalable; it allows to add more hardware to accommodate more customers and more data as per requirement.
- Always on architecture – Cassandra has no single point of failure and it is continuously available for business-critical applications that cannot afford a failure.
- Fast linear-scale performance – Cassandra is linearly scalable, i.e., it increases your throughput as you increase the number of nodes in the cluster. Therefore it maintains a quick response time.
- Flexible data storage – Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured. It can

dynamically accommodate changes to your data structures according to your needs.

- Easy data distribution – Cassandra provides the flexibility to distribute data where you need by replicating data across multiple data centers.
- Transaction support – Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID).
- Fast writes – Cassandra was designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.

Components of Cassandra

The key components of Cassandra are as follows –

- Node – It is the place where data is stored.
- Data center – It is a collection of related nodes.
- Cluster – A cluster is a component that contains one or more data centers.
- Commit log – The commit log is a crash-recovery mechanism in Cassandra. Every write operation is written to the commit log.
- Mem-table – A mem-table is a memory-resident data structure. After the commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- SSTable – It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
- Bloom filter – These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

Cassandra Query Language

Users can access Cassandra through its nodes using Cassandra Query Language (CQL). CQL treats the database (Keyspace) as a container of tables. Programmers use cqlsh: a prompt to work with CQL or separate application language drivers.

Clients approach any of the nodes for their read-write operations. That node (coordinator) plays a proxy between the client and the nodes holding the data.

1. Write Operations

Every write activity of nodes is captured by the commit logs written in the nodes. Later the data will be captured and stored in the mem-table. Whenever the mem-table is full, data will be written into the SSTable data file. All writes are automatically partitioned and replicated throughout the cluster. Cassandra periodically consolidates the SSTables, discarding unnecessary data.

2. Read Operations

During read operations, Cassandra gets values from the mem-table and checks the bloom filter to find the appropriate SSTable that holds the required data.

The data model of Cassandra is significantly different from what we normally see in an RDBMS. This chapter provides an overview of how Cassandra stores its data.

Cluster

Cassandra database is distributed over several machines that operate together. The outermost container is known as the Cluster. For failure handling, every node contains a replica, and in case of a failure, the replica takes charge. Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

Keyspace

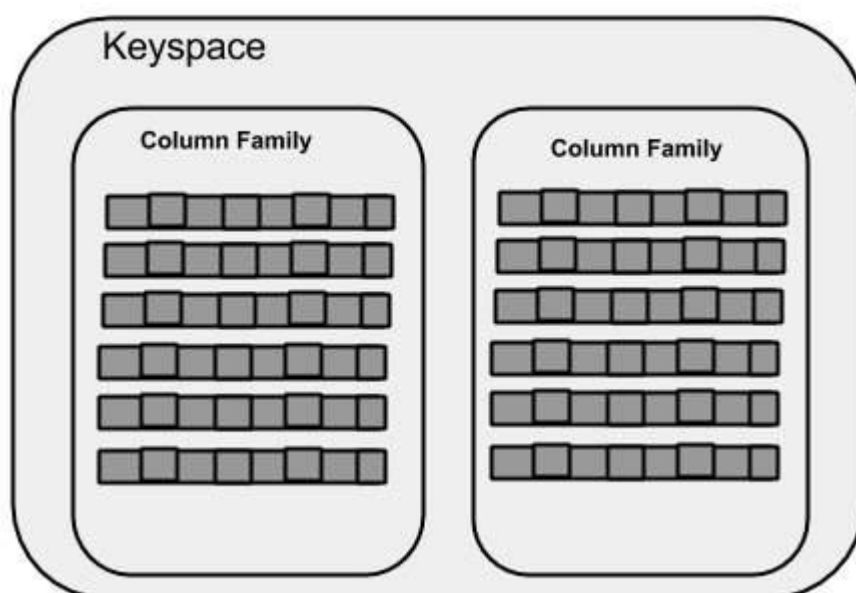
Keyspace is the outermost container for data in Cassandra. The basic attributes of a Keyspace in Cassandra are –

- Replication factor – It is the number of machines in the cluster that will receive copies of the same data.
- Replica placement strategy – It is nothing but the strategy to place replicas in the ring. We have strategies such as simple strategy (rack-aware strategy), old network topology strategy (rack-aware strategy), and network topology strategy (datacenter-shared strategy).
- Column families – Keyspace is a container for a list of one or more column families. A column family, in turn, is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

The syntax of creating a Keyspace is as follows –

```
CREATE KEYSPACE Keyspace name  
WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
```

The following illustration shows a schematic view of a Keyspace.



Column Family

A column family is a container for an ordered collection of rows. Each row, in turn, is an ordered collection of columns. The following table lists the points that differentiate a column family from a table of relational databases.

Relational Table	Cassandra column Family
A schema in a relational model is fixed. Once we define certain columns for a table, while inserting data, in every row all the columns must be filled at least with a null value.	In Cassandra, although the column families are defined, the columns are not. You can freely add any column to any column family at any time.

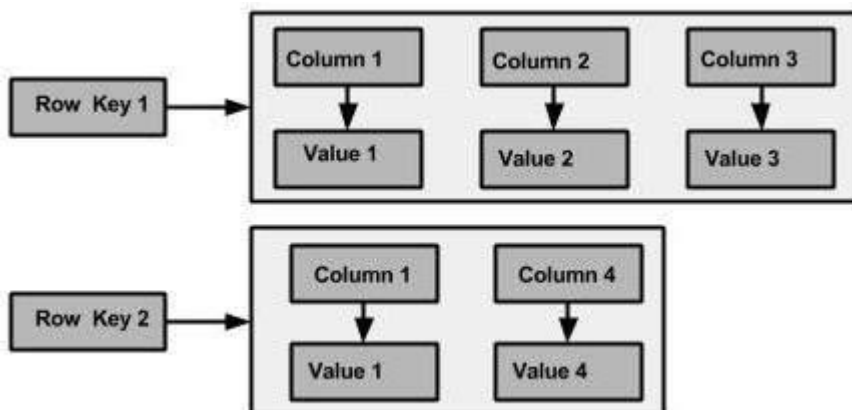
Relational tables define only columns and the user fills in the table with values.	In Cassandra, a table contains columns, or can be defined as a super column family.
--	---

A Cassandra column family has the following attributes –

- **keys_cached** – It represents the number of locations to keep cached per SSTable.
- **rows_cached** – It represents the number of rows whose entire contents will be cached in memory.
- **preload_row_cache** – It specifies whether you want to pre-populate the row cache.

Note – Unlike relational tables where a column family's schema is not fixed, Cassandra does not force individual rows to have all the columns.

The following figure shows an example of a Cassandra column family.



Column

A column is the basic data structure of Cassandra with three values, namely key or column name, value, and a time stamp. Given below is the structure of a column.

Column		
name : byte[]	value : byte[]	clock : clock[]

SuperColumn

A super column is a special column, therefore, it is also a key-value pair. But a super column stores a map of sub-columns.

Generally column families are stored on disk in individual files. Therefore, to optimize performance, it is important to keep columns that you are likely to query together in the same column family, and a super column can be helpful here. Given below is the structure of a super column.

Super Column	
name : byte[]	cols : map<byte[], column>

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

Introduction

DataStax Community Edition must be installed on the 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 a KEYSPACE 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');
```

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

Cqlsh> Help;

Display topics

Cqlsh> Help topic name;

Help open in Browser.

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

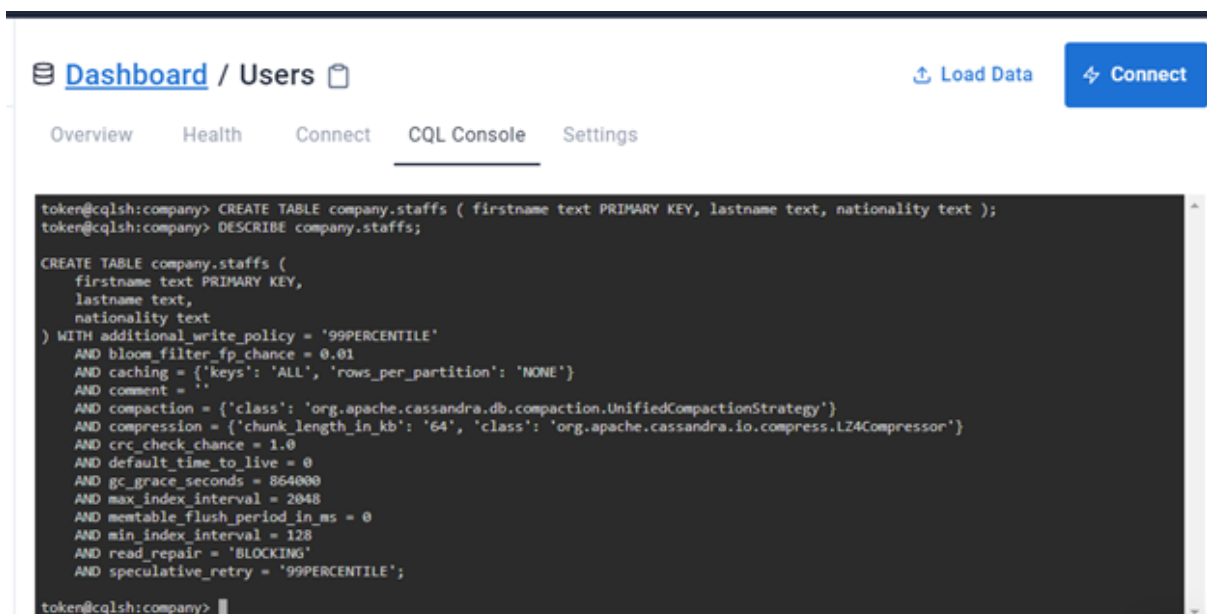
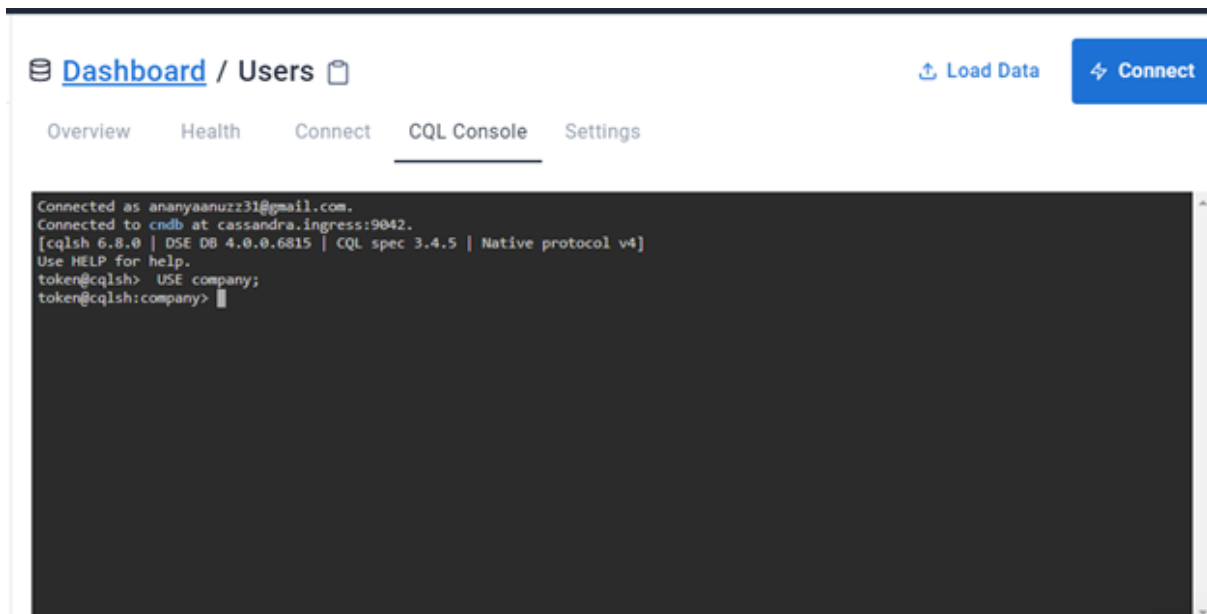
INITIAL PROCEDURE:

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”



QUERIES

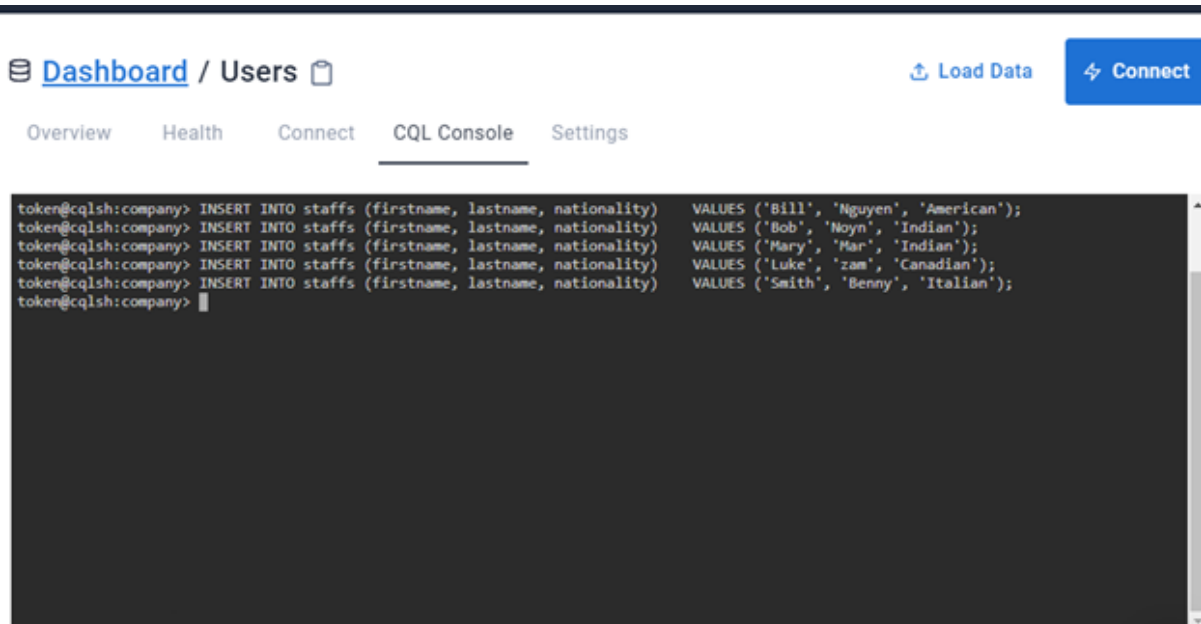
1. INSERTING DATA INTO A TABLE

Procedure:

- INSERT INTO staffs (firstname, last name, nationality) VALUES ('Bill', 'Nguyen', 'American');
- INSERT INTO staffs (firstname, last name, nationality) VALUES ('Bob', 'Noyn', 'Indian');

- INSERT INTO staffs (firstname, last name, nationality) VALUES ('Mary', 'Mar', 'Indian');
- INSERT INTO staffs (firstname, last name, nationality) VALUES ('Luke', 'zam', 'Canadian');
- INSERT INTO staffs (firstname, last name, nationality) VALUES ('Smith', 'Benny', 'Italian');

OUTPUT



The screenshot shows a database dashboard with a navigation bar at the top 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 terminal window with the following SQL commands and their outputs:

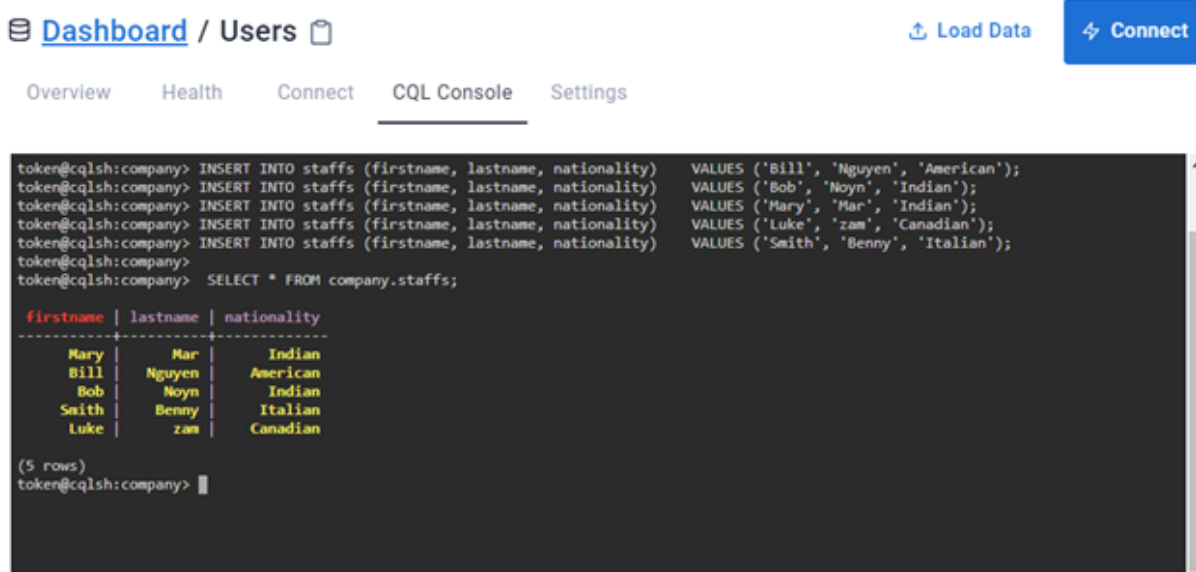
```
token@cqlsh:company> INSERT INTO staffs (firstname, lastname, nationality) VALUES ('Bill', 'Nguyen', 'American');
token@cqlsh:company> INSERT INTO staffs (firstname, lastname, nationality) VALUES ('Bob', 'Noyn', 'Indian');
token@cqlsh:company> INSERT INTO staffs (firstname, lastname, nationality) 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');
```

2. RETRIEVAL OF ALL DATA

Procedure:

SELECT * FROM company.staffs;

OUTPUT



The screenshot shows a database dashboard 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 terminal window with the following SQL commands and their output:

```
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> SELECT * FROM company.staffs;
```

firstname	lastname	nationality
Mary	Mar	Indian
Bill	Nguyen	American
Bob	Noyn	Indian
Smith	Benny	Italian
Luke	zam	Canadian

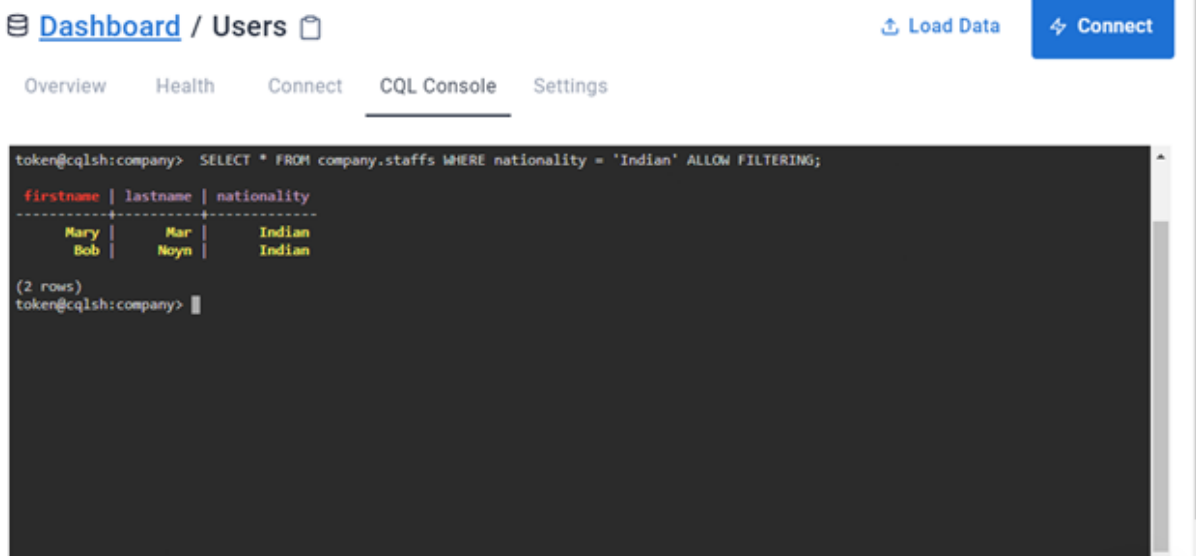
(5 rows)
token@cqlsh:company> █

3.RETRIEVE THE DETAILS OF STAFFS USING CONDITIONS

Procedure:

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

OUTPUT



The screenshot shows the same database dashboard as before, but with the 'CQL Console' tab displaying a filtered SQL query and its output:

```
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 = 'Bill' ALLOW FILTERING;
```

firstname	lastname	nationality
Bill	Nguyen	American

(1 rows)

```
token@cqlsh:company> █
```

ALTER TABLE command

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

Procedure:

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

```
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> SELECT * FROM company.staffs;

+-----+-----+-----+-----+
|firstname|lastname|nationality|title|
+-----+-----+-----+-----+
|Mary|Mar|Indian|Mrs|
|Bill|Nguyen|American|Mr|
|Bob|Noyen|Indian|Mr|
|Smith|Benny|Italian|Mr|
|Luke|zan|Canadian|Mr|
+-----+-----+-----+-----+

(5 rows)
token@cqlsh:company> █
```

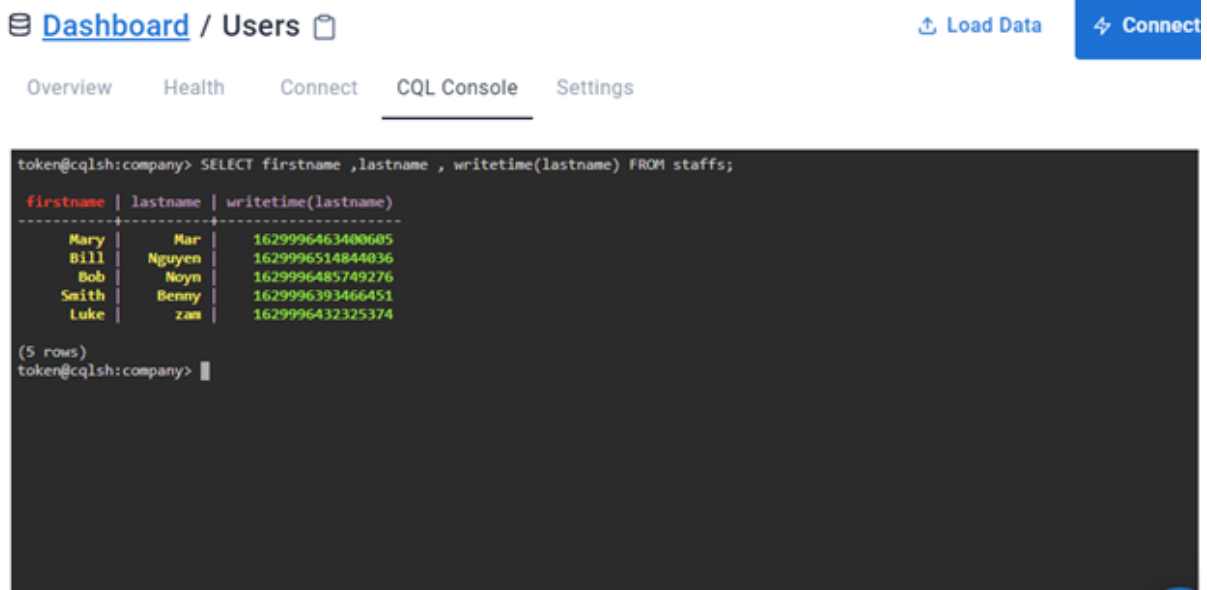
4.RETRIEVING TIMESTAMPS

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 ,last name , writetime(last name) FROM staffs;`

OUTPUT



The screenshot shows the CQL Console interface with the 'CQL Console' tab selected. The query `token@cqlsh:company> SELECT firstname ,last name , writetime(last name) FROM staffs;` has been executed. The output is a table with three columns: `firstname`, `lastname`, and `writetime(lastname)`. There are 5 rows of data.

firstname	lastname	writetime(lastname)
Mary	Mar	1629996463400605
Bill	Nguyen	1629996514844036
Bob	Noyn	1629996485749276
Smith	Benny	1629996393466451
Luke	zam	1629996432325374

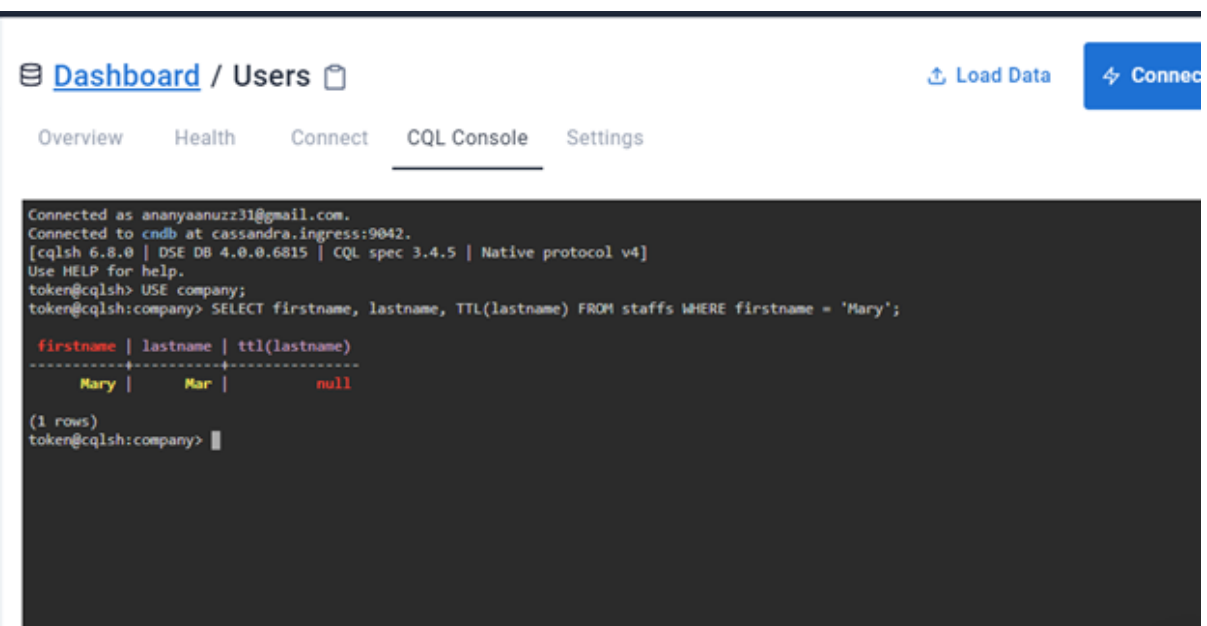
(5 rows)
token@cqlsh:company>

5. RETRIEVE THE TTL VALUE FOR ‘ Mary’s last name’

Procedure:

`SELECT firstname, last name, TTL(last name) FROM staffs WHERE firstname = 'Mary';`

OUTPUT



The screenshot shows the CQL Console interface with the 'CQL Console' tab selected. The query `token@cqlsh:company> SELECT firstname, lastname, TTL(lastname) FROM staffs WHERE firstname = 'Mary';` has been executed. The output is a table with three columns: `firstname`, `lastname`, and `ttn(lastname)`. There is 1 row of data.

firstname	lastname	ttn(lastname)
Mary	Mar	null

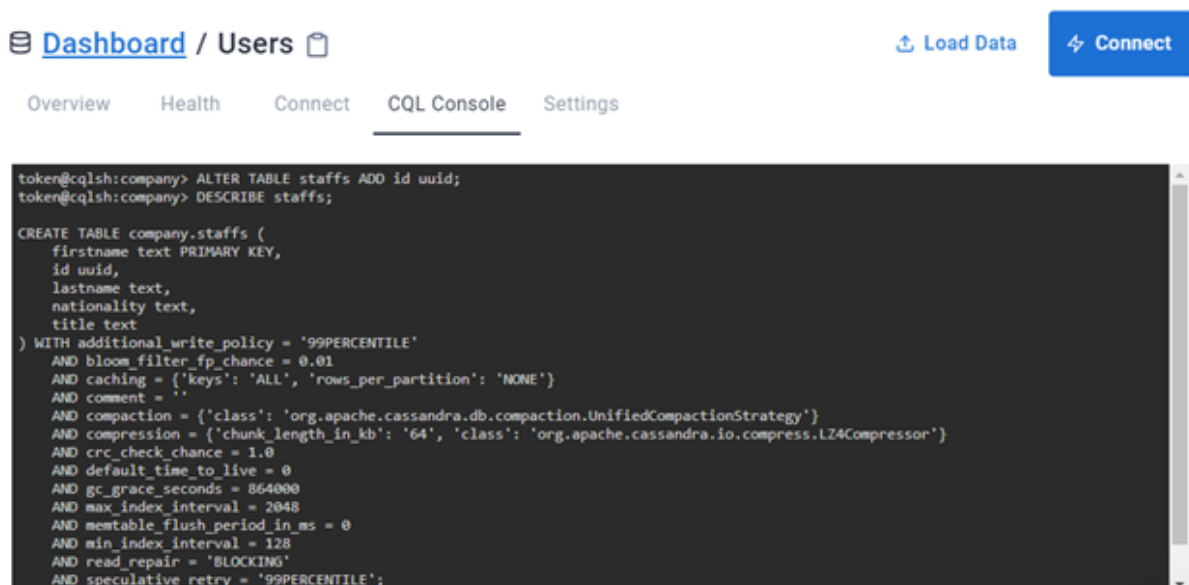
(1 rows)
token@cqlsh:company>

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 with the following content:

```
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
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';
```

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:

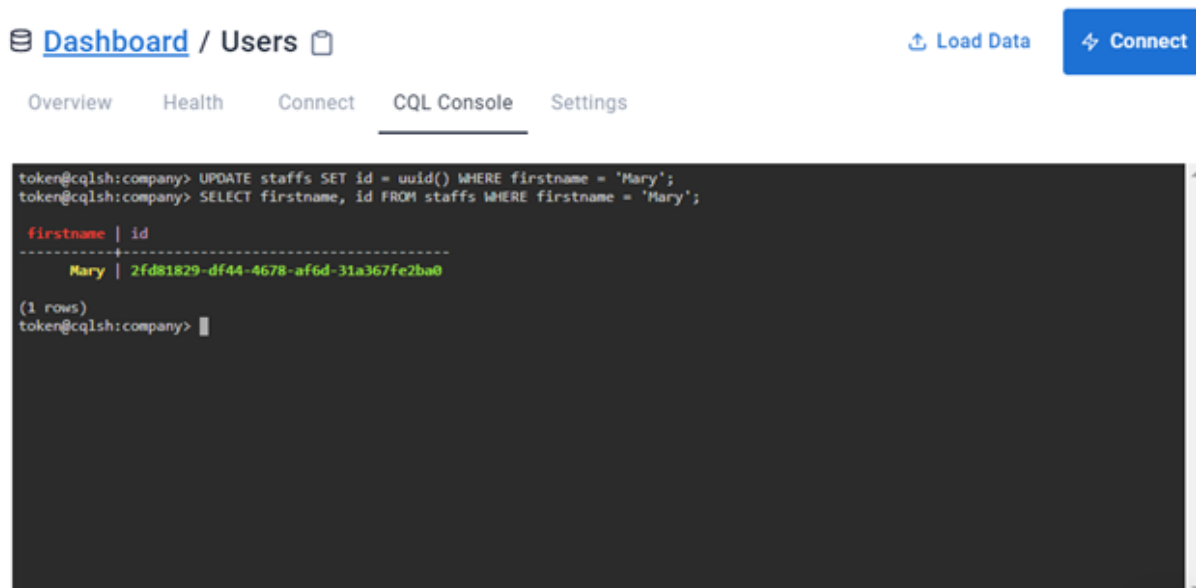
1a6300ca-0572-4736-a393-c0b7229e193e

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 web-based database 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', and 'Settings'. The 'CQL Console' tab is active, displaying a terminal window. The terminal shows the following commands and output:

```
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	2fd81829-df44-4678-af6d-31a367fe2ba0

(1 rows)
token@cqlsh:company> █

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

SET

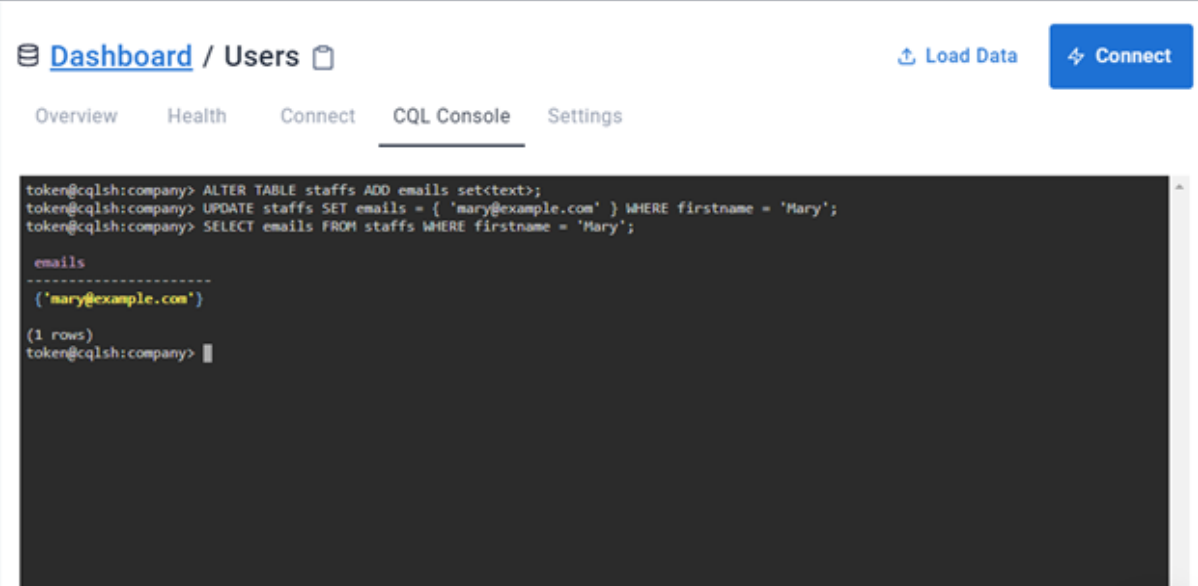
The set data type stores a collection of elements. The elements are unordered, but cqlsh returns the elements in sorted order. For example, text values are returned in alphabetical order. One advantage of using set is the ability to insert additional items without having to read the contents first.

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



The screenshot shows a database dashboard with a 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 terminal window with the following content:

```
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';

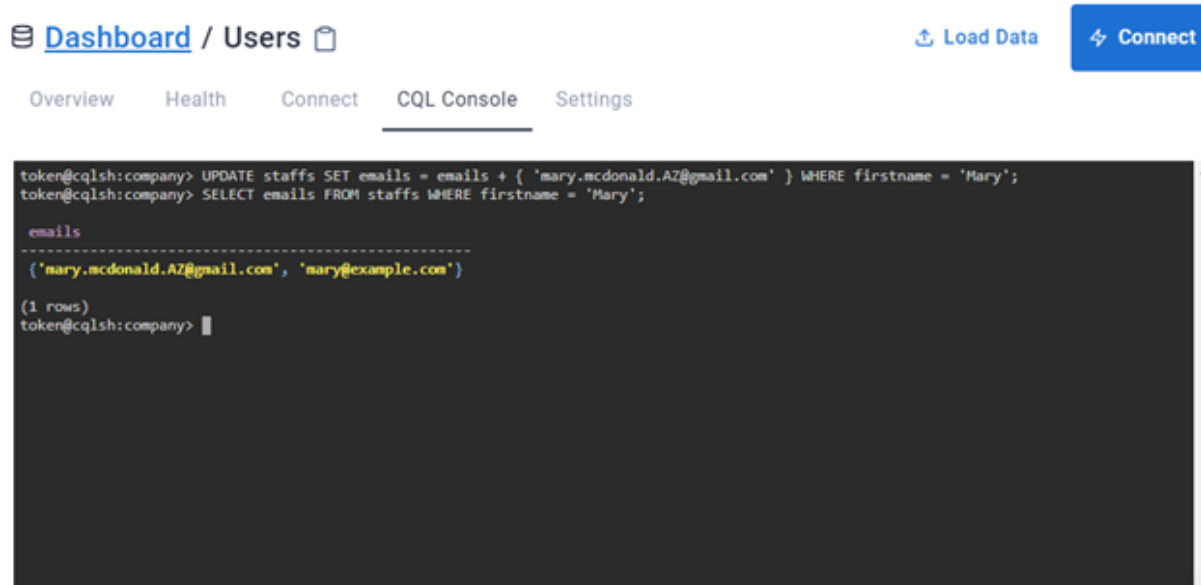
emails
-----
('mary@example.com')
(1 rows)
token@cqlsh:company> █
```

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



```
token@cqlsh:company> UPDATE staffs SET emails = emails + { 'mary.mcdonald.AZ@gmail.com' } WHERE firstname = 'Mary';
token@cqlsh:company> SELECT emails FROM staffs WHERE firstname = 'Mary';

emails
-----
('mary.mcdonald.AZ@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



The screenshot shows a web-based interface for a database console. At the top, there is a navigation bar with a hamburger menu icon, the text "Dashboard / Users", and a clipboard icon. To the right of this bar are two buttons: "Load Data" with an upload icon and a partially visible "Connect" button with a double arrow icon. Below the navigation bar is a tabbed interface with four tabs: "Overview", "Health", "Connect", and "CQL Console" (which is currently selected and underlined), and "Settings". The main area of the interface is a dark-themed terminal window. It contains the following text:

```
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';
```

 Below the commands, the output of the last query is displayed:

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

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

 At the bottom right of the interface, there is a watermark that says "Activate Windows".