CassandraDB Commands

Tushar B. Kute,
http://tusharkute.com

Shell Commands

- help
  - The HELP command displays a synopsis and a brief description of all cqlsh commands.
- capture
  - This command captures the output of a command and adds it to a file.
- CONSISTENCY
  - This command shows the current consistency level, or sets a new consistency level.

Shell Commands

- Copy
  - This command copies data to and from Cassandra to a file. Given below is an example to copy the table named emp to the file myfile.
  - COPY emp (emp_id, emp_city, emp_name, emp_phone, emp_sal) TO ‘myfile’;
- DESCRIBE
  - This command describes the current cluster of Cassandra and its objects. The variants of this command are explained below.
  - Describe cluster: This command provides information about the cluster.
  - Describe Keyspaces: This command lists all the keyspaces in a cluster.
  - Describe tables: This command lists all the tables in a keyspace.

Shell Commands

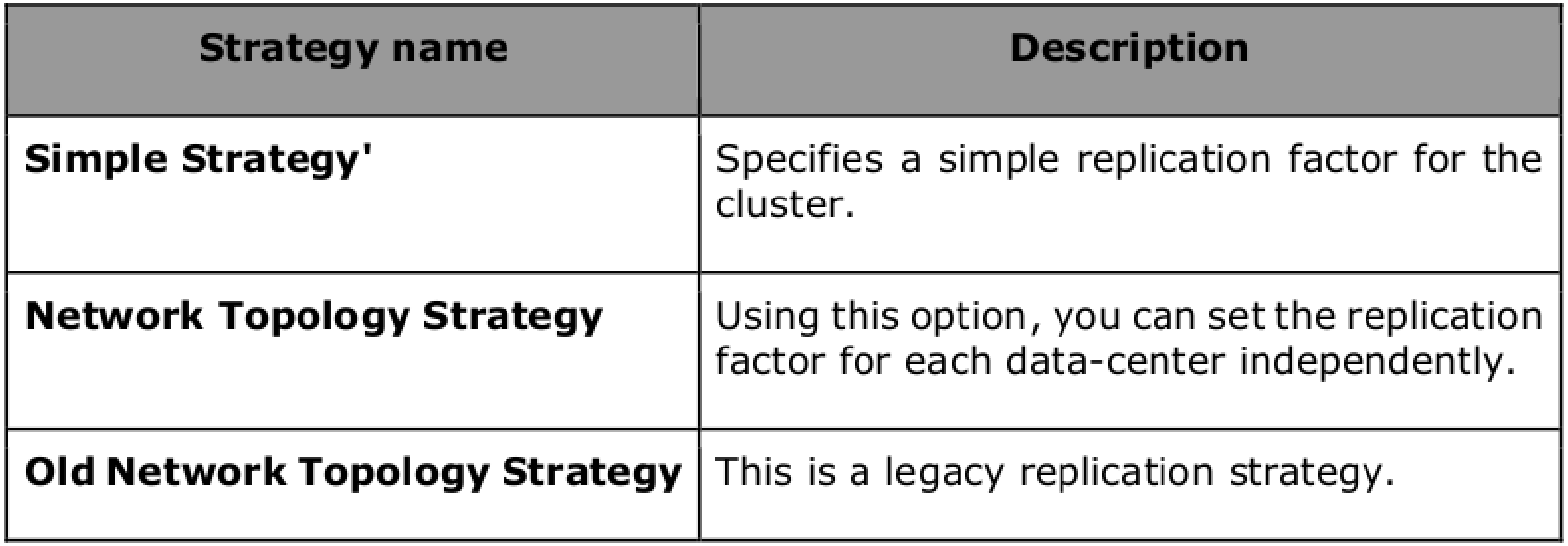
- Expand
  - This command is used to expand the output. Before using this command, you have to turn the expand command on.
- SHOW
  - This command displays the details of current cqlsh session such as Cassandra version, host, or data type assumptions.
  - `show host;`
  - `show version;`
- SOURCE
  - Using this command, you can execute the commands in a file.

KeySpace operations

- A keyspace in Cassandra is a namespace that defines data replication on nodes.
- A cluster contains one keyspace per node. Given below is the syntax for creating a keyspace using the statement CREATE KEYSPACE.
- Syntax:
  - `CREATE KEYSPACE <identifier> WITH <properties>`
  - **Example:**
  - `CREATE KEYSPACE “KeySpace Name” WITH replication = {'class': ‘Strategy name’, 'replication_factor' : ‘No.Of replicas’};`

Replication

- The replication option is to specify the Replica Placement strategy and the number of replicas wanted. The following table lists all the replica placement strategies.



Strategy name	Description
Simple Strategy'	Specifies a simple replication factor for the cluster.
Network Topology Strategy	Using this option, you can set the replication factor for each data-center independently.
Old Network Topology Strategy	This is a legacy replication strategy.

Alter Keyspace

- ALTER KEYSPACE can be used to alter properties such as the number of replicas and the durable_writes of a KeySpace.
- Syntax:
  - ALTER KEYSPACE <identifier> WITH <properties>
  - Example:
  -
```
ALTER KEYSPACE “KeySpace Name”
WITH replication = {'class': ‘Strategy name’,
'replication_factor' : ‘No.Of replicas’};
```

Drop Keyspace

- You can drop a KeySpace using the command DROP KEYSPACE. Given below is the syntax for dropping a KeySpace.
- Syntax:
  - DROP KEYSPACE <identifier>
- Example:
  - `DROP KEYSPACE tushar;`

Create table

```
CREATE TABLE tablename(
column1 name datatype PRIMARYKEY,
column2 name data type,
column3 name data type)
```

- Example:

```
CREATE TABLE emp(
emp_id int PRIMARY KEY,
emp_name text,
emp_city text,
emp_sal varint,
emp_phone varint
);
```

Alter Table

- You can alter a table using the command ALTER TABLE. Using ALTER command, you can perform the following operations:
  - Add a column
  - Drop a column
  - Update the options of a table using with keyword
- Example:

  -
    ```
    ALTER TABLE emp
    ADD emp_email text;
    ```

  -
    ```
    ALTER TABLE emp DROP emp_email;
    ```

Drop and Truncate Table

- Drop table command:
  - You can drop a table using the command Drop Table.
  - Example: DROP TABLE emp;
- Truncating a Table
  - You can truncate a table using the TRUNCATE command. When you truncate a table, all the rows of the table are deleted permanently.
  - Example: TRUNCATE student;

Create index

- You can create an index in Cassandra using the command CREATE INDEX. Its syntax is as follows:
- `CREATE INDEX <identifier> ON <tablename>`
- Given below is an example to create an index to a column. Here we are creating an index to a column ‘emp_name’ in a table named emp .
- `CREATE INDEX name ON emp1 (emp_name);`

Batch statements

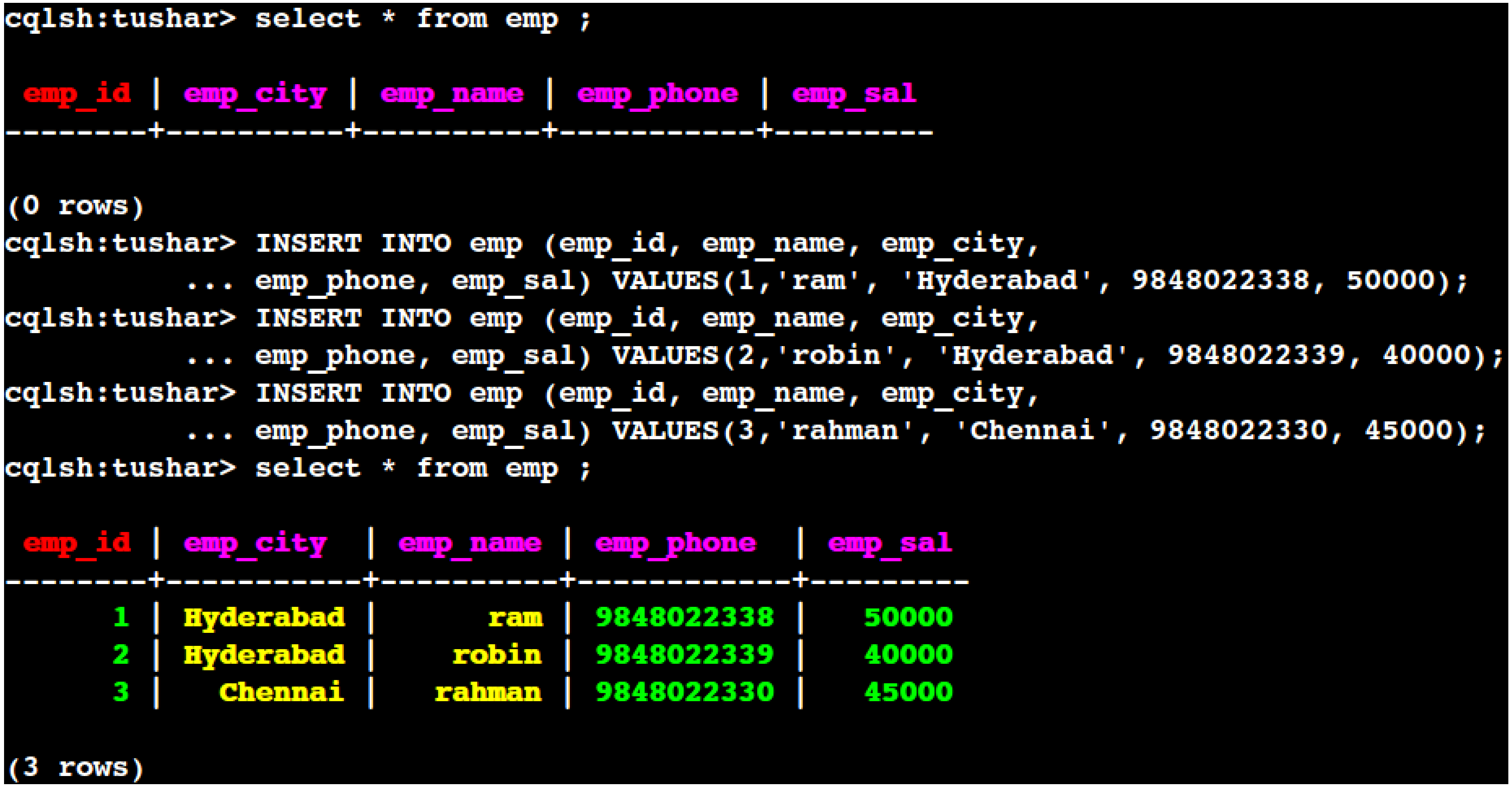
- Using BATCH, you can execute multiple modification statements (insert, update, delete) simultaneously Its syntax is as follows:

```
BEGIN BATCH
 <insert-stmt> / <update-stmt> / <delete-stmt>
APPLY BATCH
```

- Example:

```
BEGIN BATCH
... INSERT INTO emp (emp_id, emp_city, emp_name,
emp_phone, emp_sal) values ( 4,'Pune','rajeev',
9848022331, 30000);
... UPDATE emp SET emp_sal = 50000 WHERE emp_id =3;
... DELETE emp_city FROM emp WHERE emp_id = 2;
... APPLY BATCH;
```

Insert statement



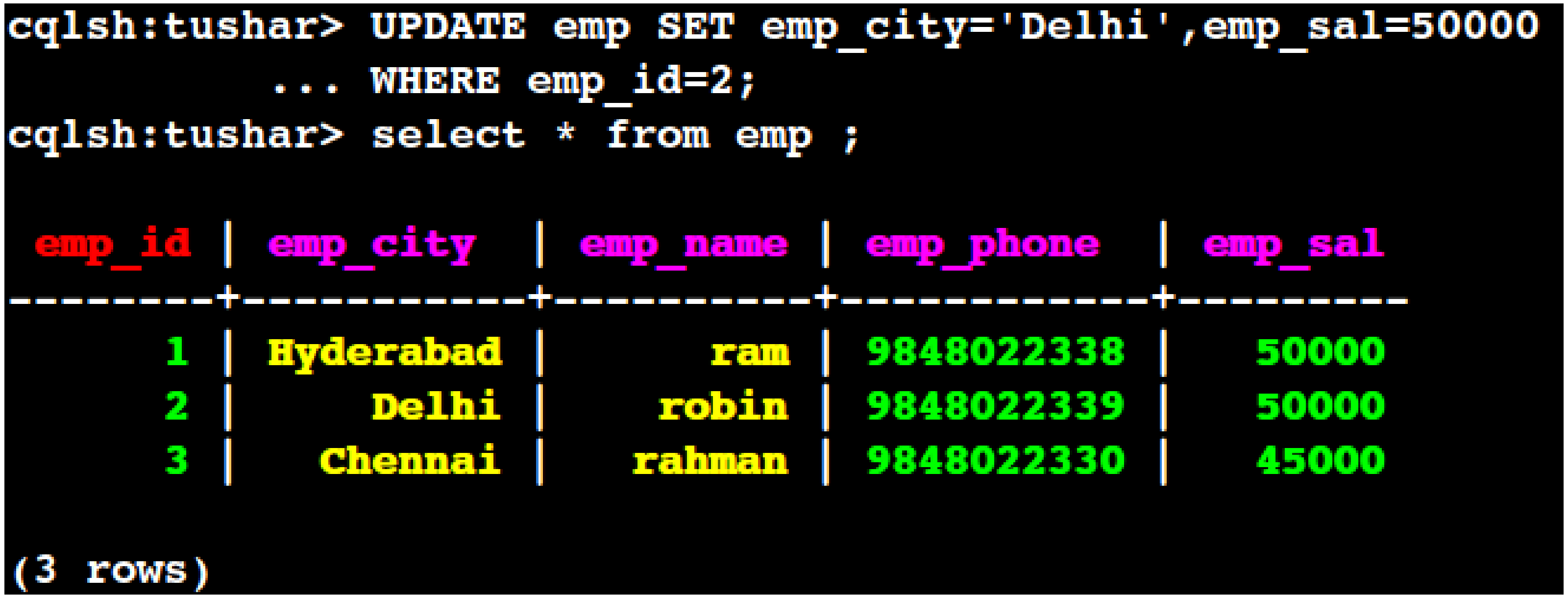
```
cqlsh:tushar> select * from emp ;

 emp_id | emp_city | emp_name | emp_phone | emp_sal
--------+----------+----------+-----------+---------

(0 rows)
cqlsh:tushar> INSERT INTO emp (emp_id, emp_name, emp_city,
          ... emp_phone, emp_sal) VALUES(1,'ram', 'Hyderabad', 9848022338, 50000);
cqlsh:tushar> INSERT INTO emp (emp_id, emp_name, emp_city,
          ... emp_phone, emp_sal) VALUES(2,'robin', 'Hyderabad', 9848022339, 40000);
cqlsh:tushar> INSERT INTO emp (emp_id, emp_name, emp_city,
          ... emp_phone, emp_sal) VALUES(3,'rahman', 'Chennai', 9848022330, 45000);
cqlsh:tushar> select * from emp ;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 | Hyderabad |    robin | 9848022339 |   40000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
```

Updating a table

- UPDATE is the command used to update data in a table. The following keywords are used while updating data in a table:
  - Where: This clause is used to select the row to be updated.
  - Set: Set the value using this keyword.
  - Must: Includes all the columns composing the primary key.
- Example:

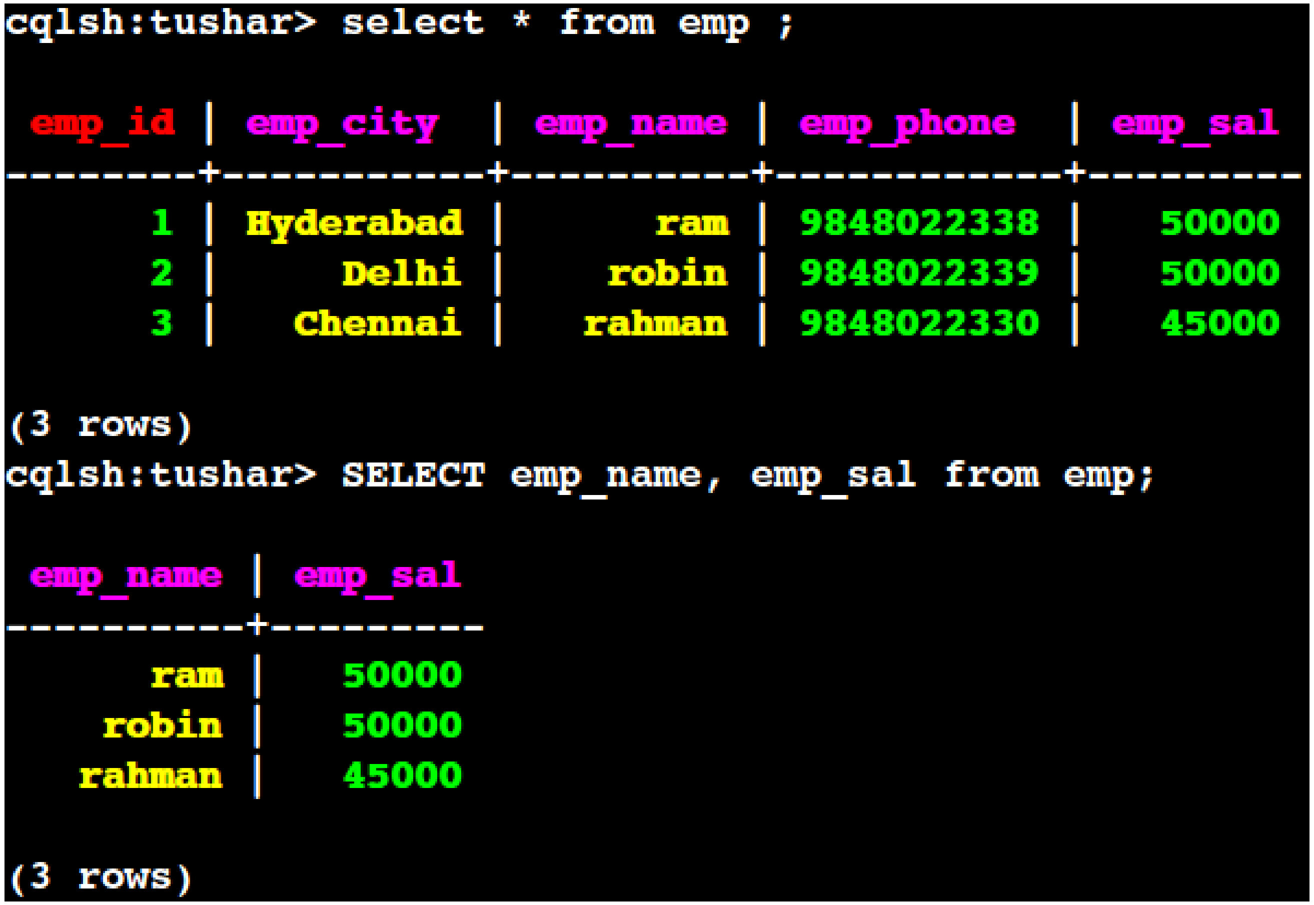
  -
    ```
    UPDATE emp SET emp_city='Delhi',
    emp_sal=50000 WHERE emp_id=2;
    ```

Updating a table



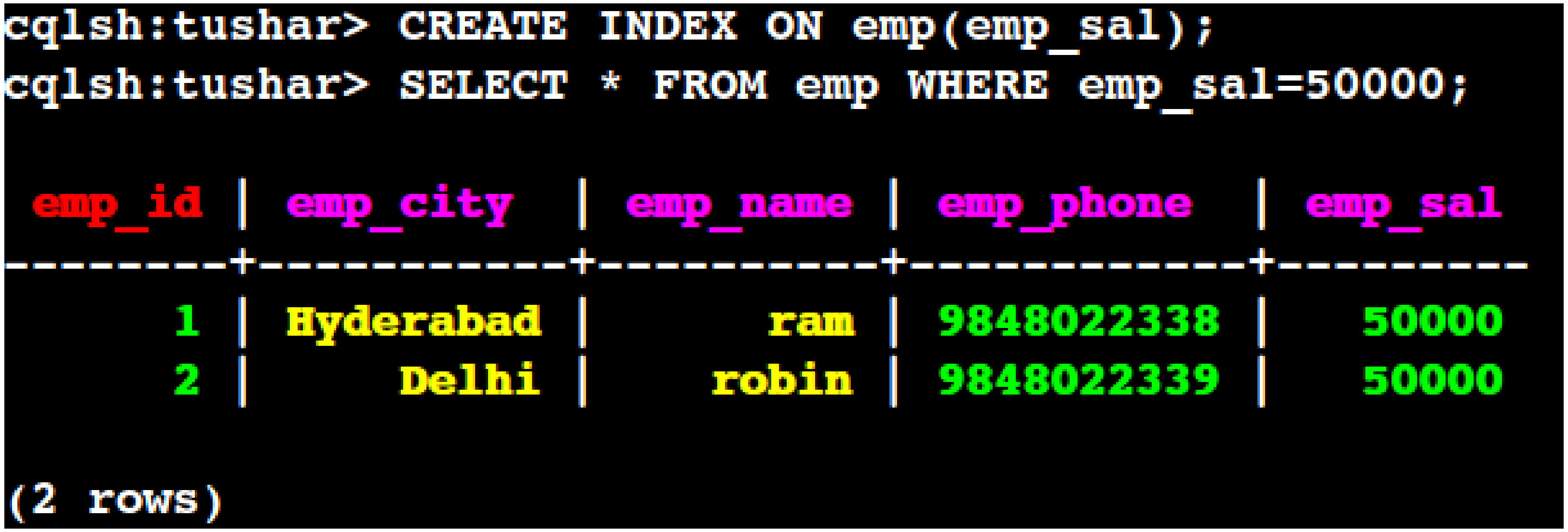
```
cqlsh:tushar> UPDATE emp SET emp_city='Delhi',emp_sal=50000
          ... WHERE emp_id=2;
cqlsh:tushar> select * from emp ;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 |     Delhi |    robin | 9848022339 |   50000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
```

Reading a data



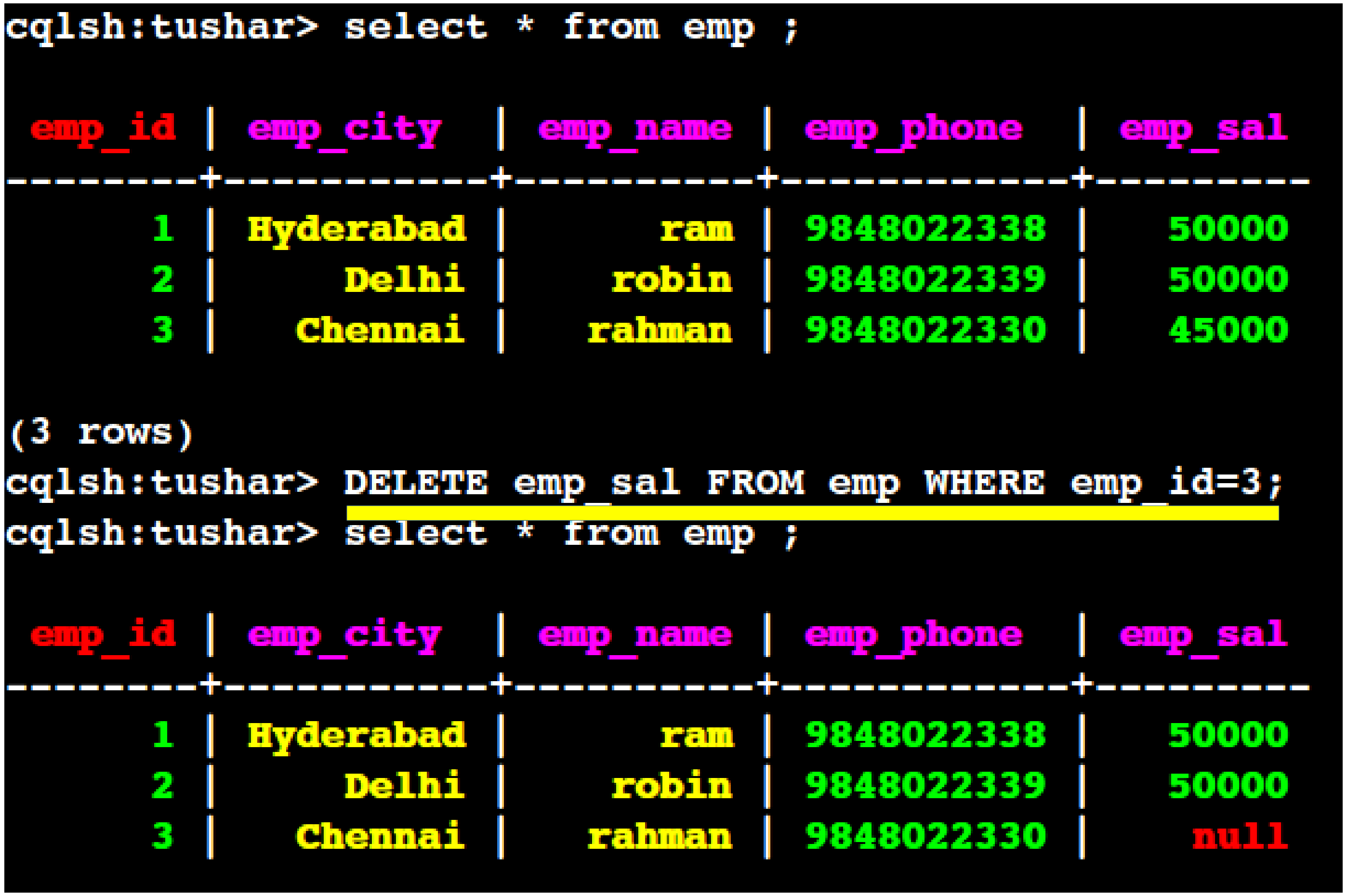
```
cqlsh:tushar> select * from emp ;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 |     Delhi |    robin | 9848022339 |   50000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
cqlsh:tushar> SELECT emp_name, emp_sal from emp;

 emp_name | emp_sal
----------+---------
      ram |   50000
    robin |   50000
   rahman |   45000

(3 rows)
```

Reading a data



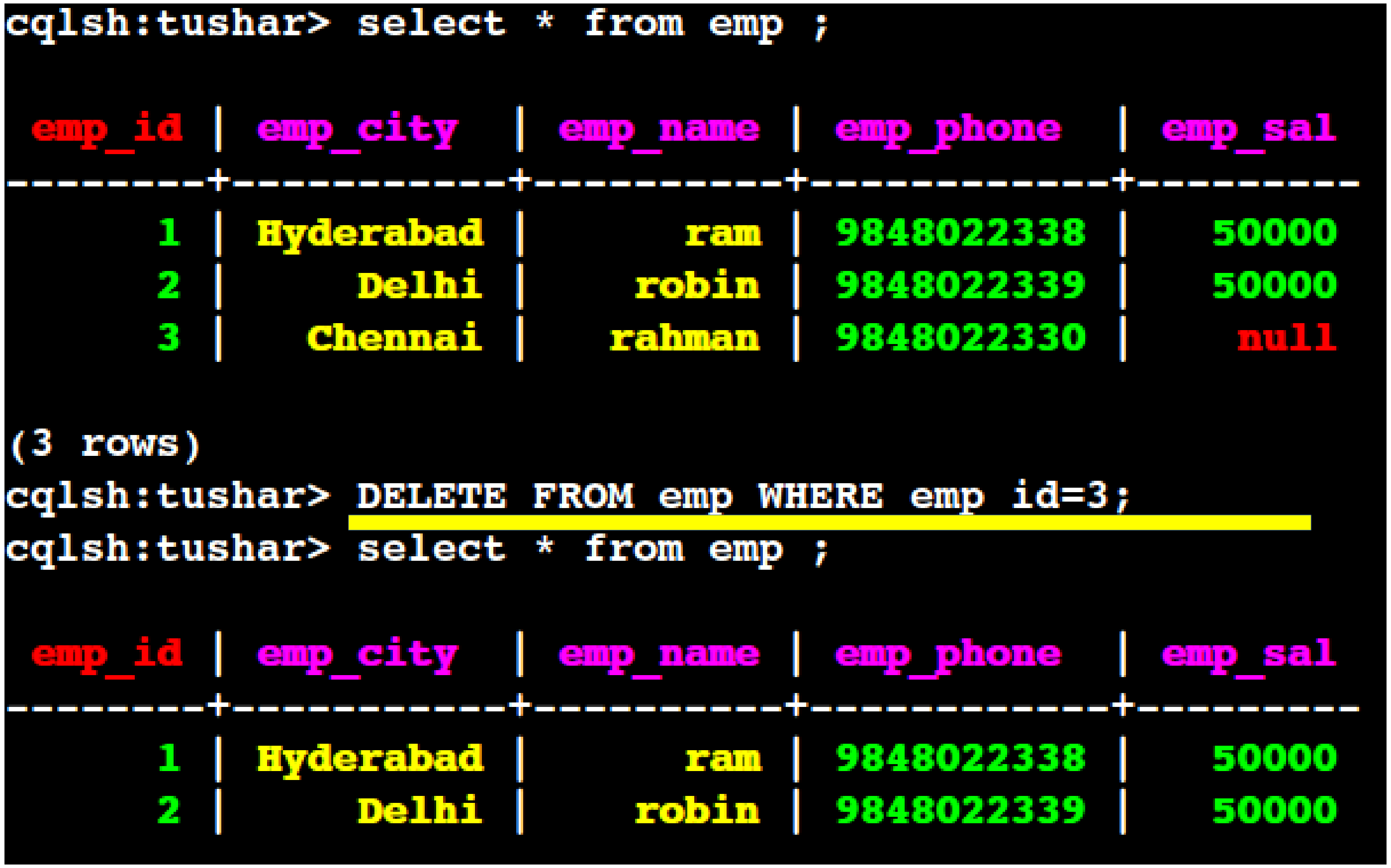
```
cqlsh:tushar> CREATE INDEX ON emp(emp_sal);
cqlsh:tushar> SELECT * FROM emp WHERE emp_sal=50000;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 |     Delhi |    robin | 9848022339 |   50000

(2 rows)
```

Deleting a data



```
cqlsh:tushar> select * from emp ;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 |     Delhi |    robin | 9848022339 |   50000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
cqlsh:tushar> DELETE emp_sal FROM emp WHERE emp_id=3;
cqlsh:tushar> select * from emp ;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 |     Delhi |    robin | 9848022339 |   50000
      3 |   Chennai |   rahman | 9848022330 |    null
```

Deleting a data

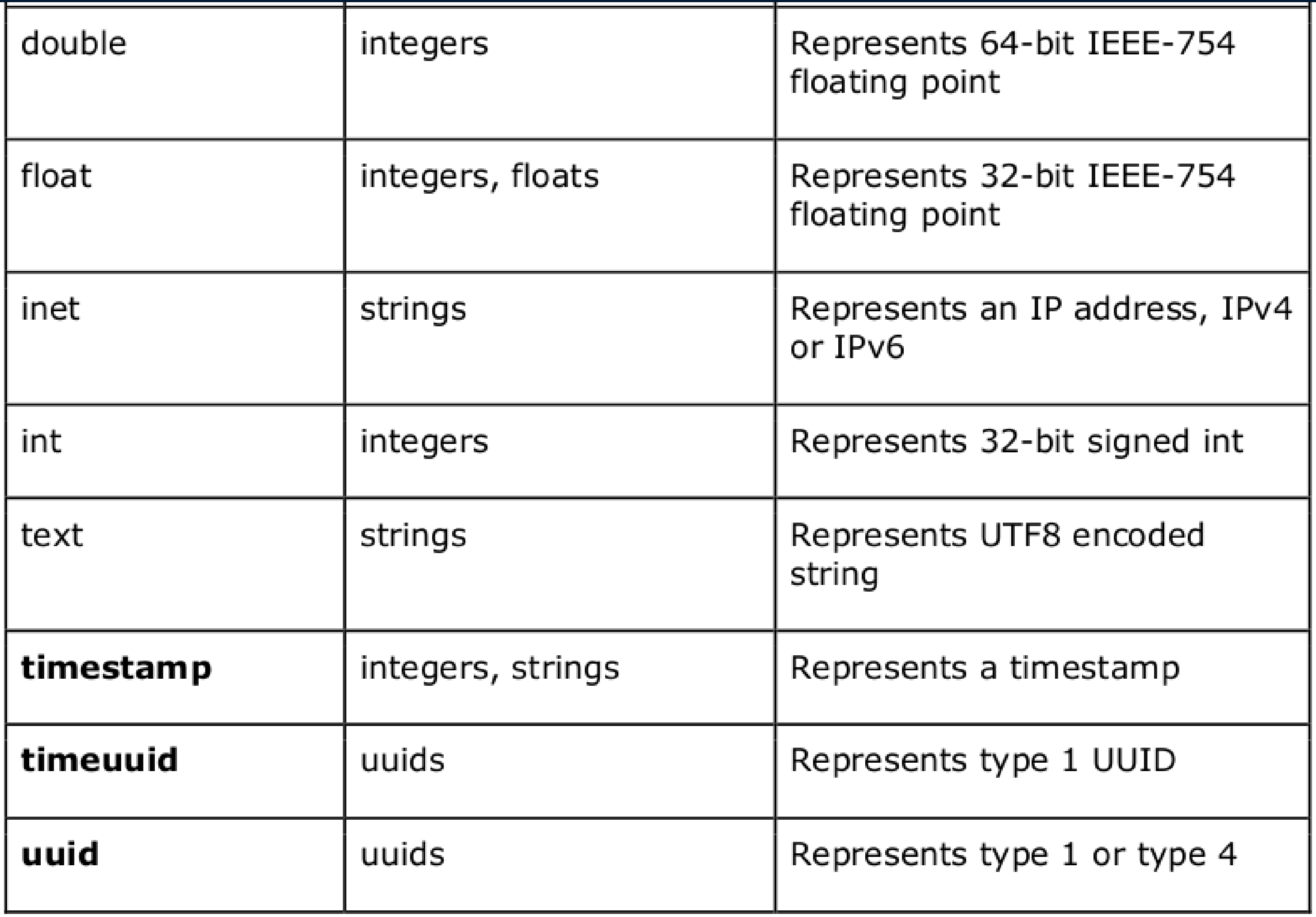


```
cqlsh:tushar> select * from emp ;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 |     Delhi |    robin | 9848022339 |   50000
      3 |   Chennai |   rahman | 9848022330 |    null

(3 rows)
cqlsh:tushar> DELETE FROM emp WHERE emp id=3;
cqlsh:tushar> select * from emp ;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 |     Delhi |    robin | 9848022339 |   50000
```

CQL Data types

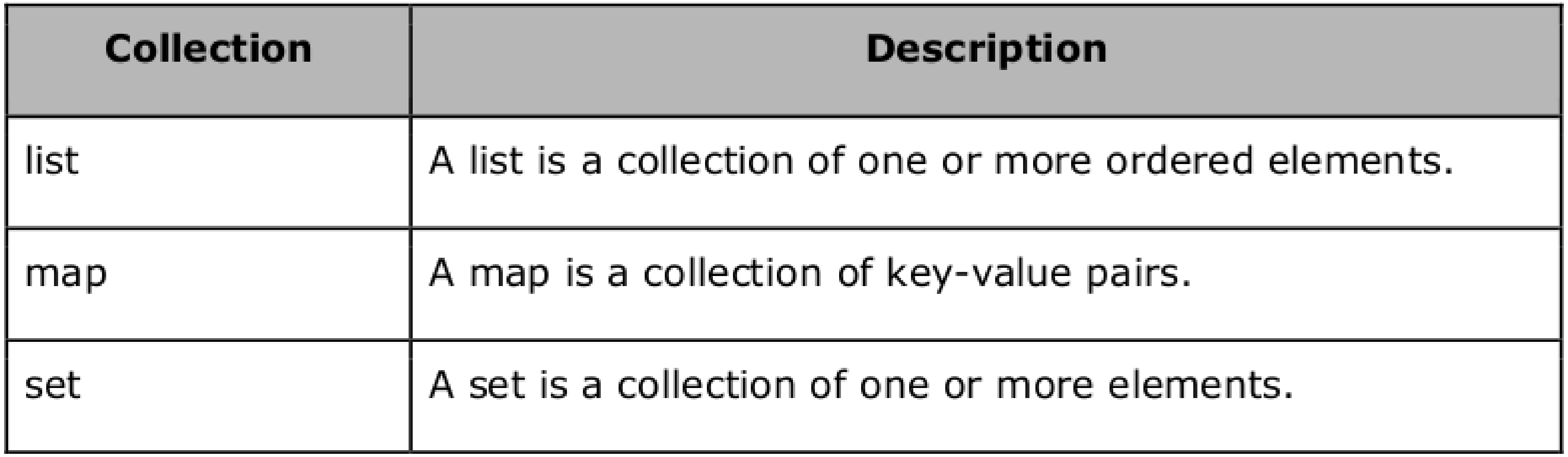
Data Type	Constants	Description
ascii	strings	Represents ASCII character string
bigint	integers	Represents 64-bit signed long
blob	blobs	Represents arbitrary bytes
Boolean	booleans	Represents true or false
counter	integers	Represents counter column
decimal	integers, floats	Represents variable-precision decimal

CQL Data types



double	integers	Represents 64-bit IEEE-754 floating point
float	integers, floats	Represents 32-bit IEEE-754 floating point
inet	strings	Represents an IP address, IPv4 or IPv6
int	integers	Represents 32-bit signed int
text	strings	Represents UTF8 encoded string
timestamp	integers, strings	Represents a timestamp
timeuuid	uuids	Represents type 1 UUID
uuid	uuids	Represents type 1 or type 4

CQL Collection types



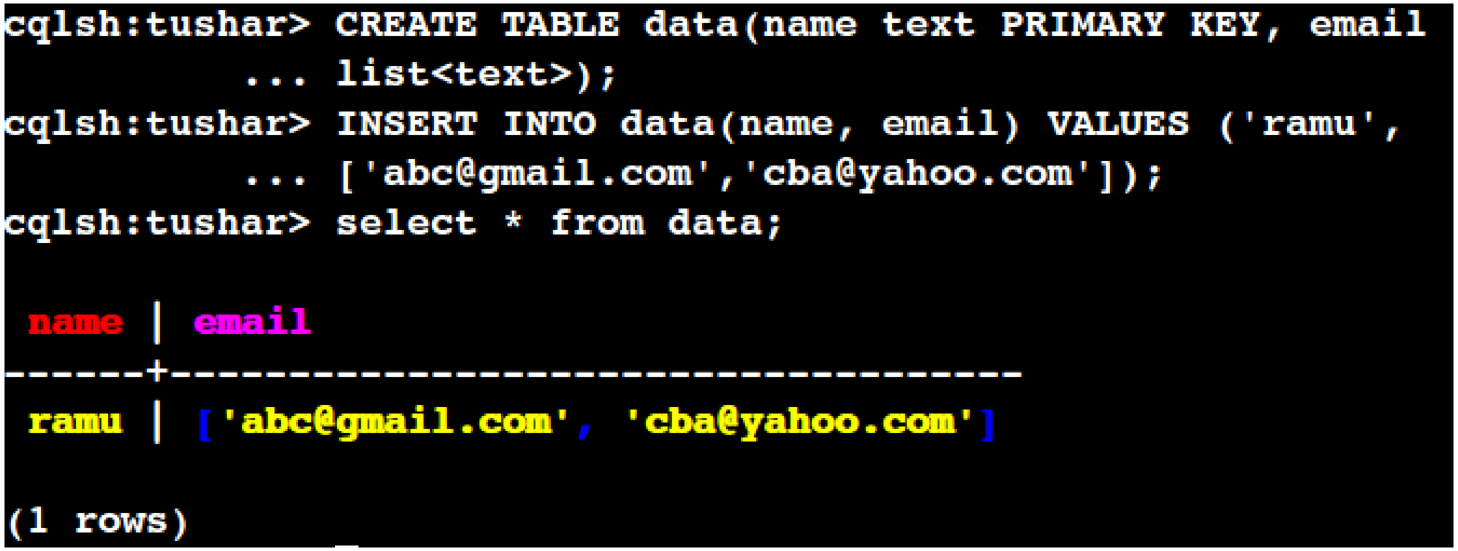
Collection	Description
list	A list is a collection of one or more ordered elements.
map	A map is a collection of key-value pairs.
set	A set is a collection of one or more elements.

CQL Collections

- CQL provides the facility of using Collection data types.
- Using these Collection types, you can store multiple values in a single variable.
  - List
  - Map
  - Set

Lists

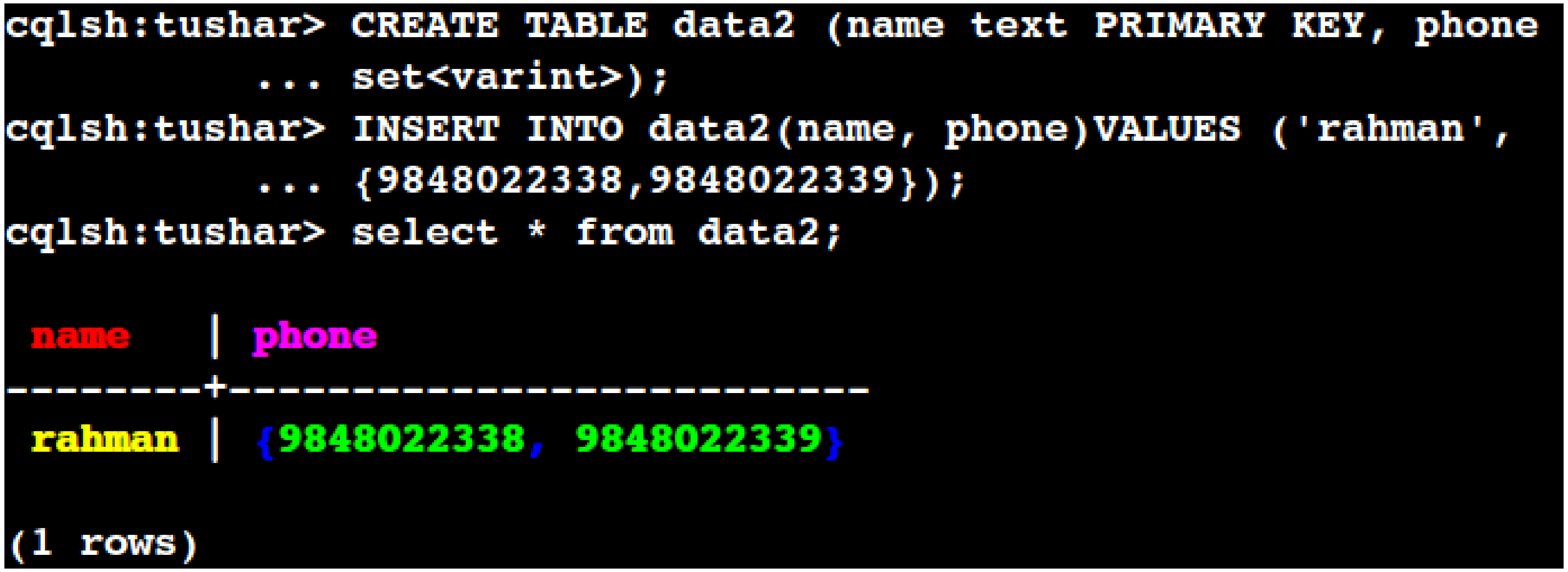
- List is used in the cases where
  - the order of the elements is to be maintained, and
  - a value is to be stored multiple times.
- You can get the values of a list data type using the index of the elements in the list.



```
cqlsh:tushar> CREATE TABLE data(name text PRIMARY KEY, email
          ... list<text>);
cqlsh:tushar> INSERT INTO data(name, email) VALUES ('ramu',
          ... ['abc@gmail.com','cba@yahoo.com']);
cqlsh:tushar> select * from data;

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

(1 rows)
```

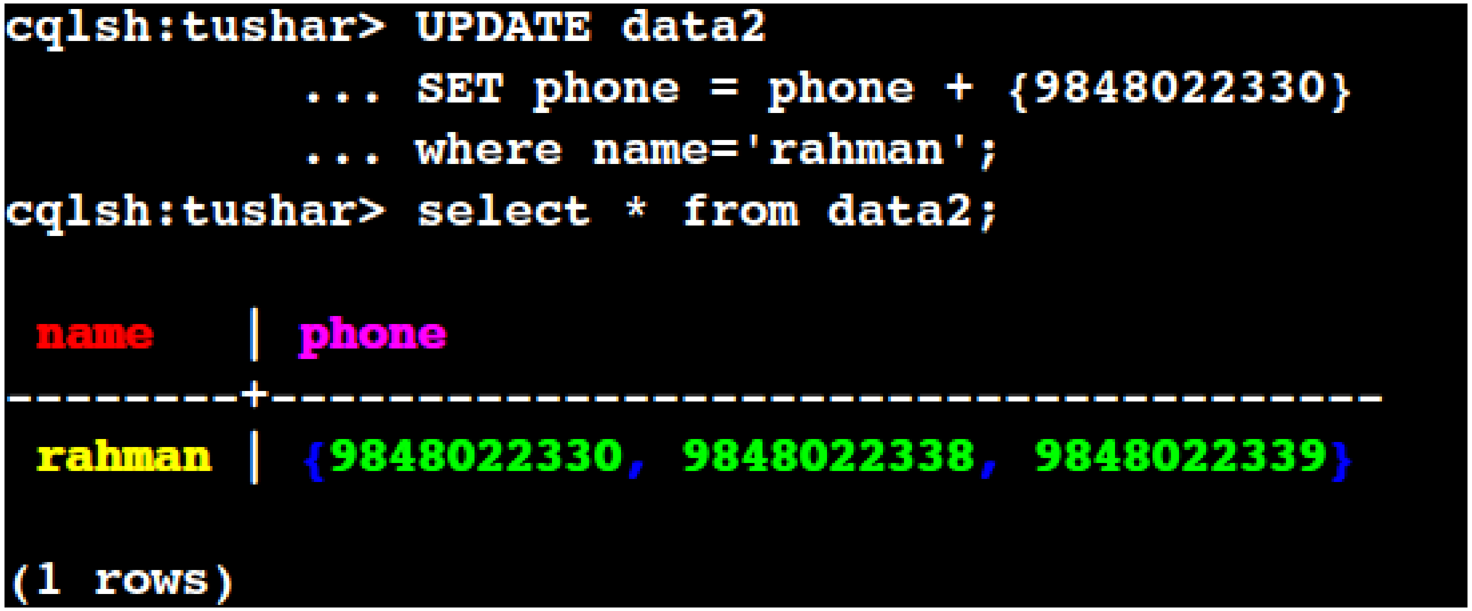
Set

- 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.



```
cqlsh:tushar> CREATE TABLE data2 (name text PRIMARY KEY, phone
          ... set<varint>);
cqlsh:tushar> INSERT INTO data2(name, phone)VALUES ('rahman',
          ... {9848022338,9848022339});
cqlsh:tushar> select * from data2;

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

(1 rows)
```

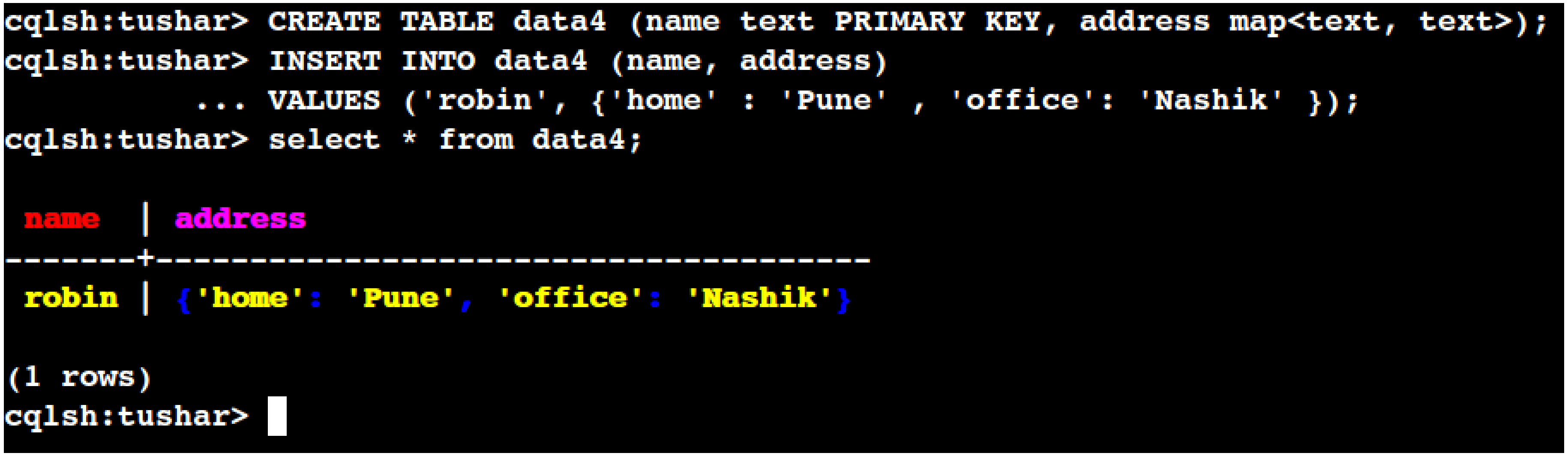
Set operations



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



```
cqlsh:tushar> CREATE TABLE data4 (name text PRIMARY KEY, address map<text, text>);
cqlsh:tushar> INSERT INTO data4 (name, address)
          ... VALUES ('robin', {'home' : 'Pune' , 'office': 'Nashik' });
cqlsh:tushar> select * from data4;

 name  | address
-------+-------------------------------------
 robin | {'home': 'Pune', 'office': 'Nashik'}

(1 rows)
cqlsh:tushar>
```

Thank you

Web Resources
http://mitu.co.in
http://tusharkute.com

Blogs
http://digitallocha.blogspot.in
http://kyamputar.blogspot.in

tushar@tusharkute.com