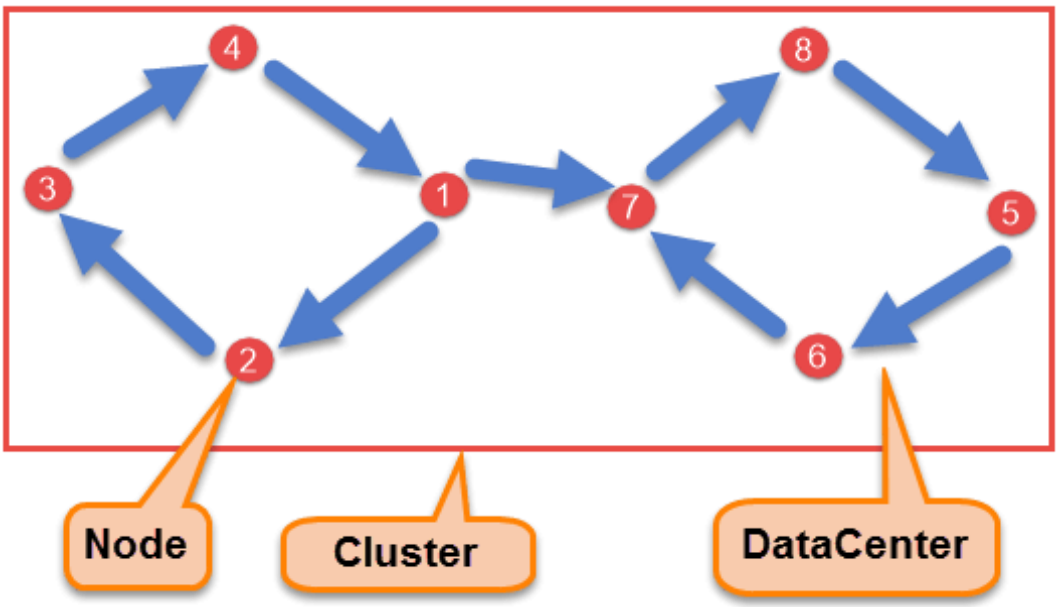
Components of Cassandra

There are following components in the Cassandra;



- **Node**

Node is the place where data is stored. It is the basic component of Cassandra.

- **Data Center**

A collection of nodes are called data center. Many nodes are categorized as a data center.

- **Cluster**

The cluster is the collection of many data centers.

- **Commit Log**

Every write operation is written to Commit Log. Commit log is used for crash recovery.

- **Mem-table**

After data written in Commit log, data is written in Mem-table. Data is written in Mem-table temporarily.

- **SSTable**

When Mem-table reaches a certain threshold, data is flushed to an SSTable disk file.

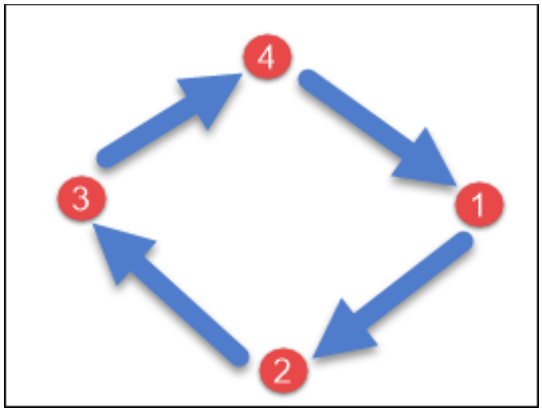
Data Replication

As hardware problem can occur or link can be down at any time during data process, a solution is required to provide a backup when the problem has occurred. So data is replicated for assuring no single point of failure.
Cassandra places replicas of data on different nodes based on these two factors.

- Where to place next replica is determined by the **Replication Strategy**.
- While the total number of replicas placed on different nodes is determined by the **Replication Factor**.

One Replication factor means that there is only a single copy of data while three replication factor means that there are three copies of the data on three different nodes.
For ensuring there is no single point of failure, **replication factor must be three.**
There are two kinds of replication strategies in Cassandra.
SimpleStrategy
SimpleStrategy is used when you have just one data center. SimpleStrategy places the first replica on the node selected by the partitioner. After that, remaining replicas are placed in clockwise direction in the Node ring.

Here is the pictorial representation of the SimpleStrategy.

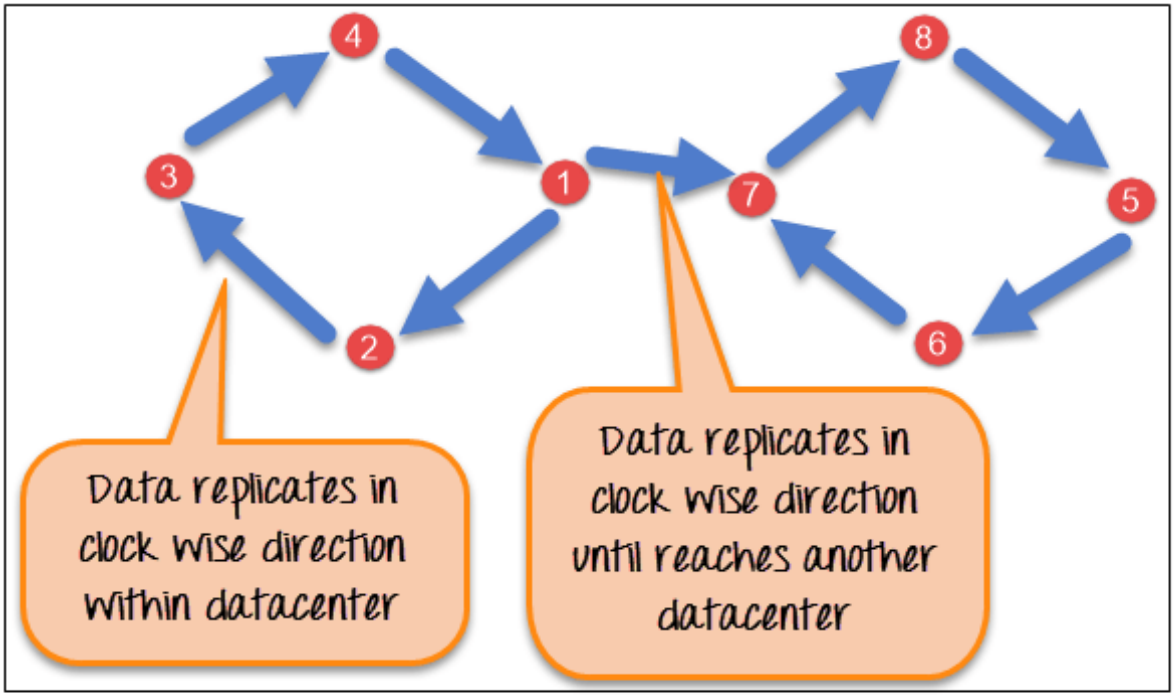


NetworkTopologyStrategy
NetworkTopologyStrategy is used when you have more than two data centers.

In NetworkTopologyStrategy, replicas are set for each data center separately. NetworkTopologyStrategy places replicas in the clockwise direction in the ring until reaches the first node in another rack.

This strategy tries to place replicas on different racks in the same data center. This is due to the reason that sometimes failure or problem can occur in the rack. Then replicas on other nodes can provide data.

Here is the pictorial representation of the Network topology strategy



Cassandra Shell Commands

Documented Shell Commands

Given below are the Cqlsh documented shell commands. These are the commands used to perform tasks such as displaying help topics, exit from cqlsh, describe, etc.

1. **HELP:** Displays help topics for all cqlsh commands.
2. **CAPTURE:** Captures the output of a command and adds it to a file.
3. **CONSISTENCY:** Shows the current consistency level, or sets a new consistency level.
4. **COPY:** Copies data to and from Cassandra.
5. **DESCRIBE:** Describes the current cluster of Cassandra and its objects.

6. **EXPAND:** Expands the output of a query vertically.
7. **EXIT:** Using this command, you can terminate cqlsh.
8. **PAGING:** Enables or disables query paging.
9. **SHOW:** Displays the details of current cqlsh session such as Cassandra version, host, or data type assumptions.
10. **SOURCE:** Executes a file that contains CQL statements.
11. **TRACING:** Enables or disables request tracing.

CQL Data Definition Commands

1. **CREATE KEYSPACE:** Creates a KeySpace in Cassandra.
2. **USE:** Connects to a created KeySpace.
3. **ALTER KEYSPACE:** Changes the properties of a KeySpace.
4. **DROP KEYSPACE:** Removes a KeySpace
5. **CREATE TABLE:** Creates a table in a KeySpace.
6. **ALTER TABLE:** Modifies the column properties of a table.
7. **DROP TABLE:** Removes a table.
1. **TRUNCATE:** Removes all the data from a table.
2. **CREATE INDEX:** Defines a new index on a single column of a table.
3. **DROP INDEX:** Deletes a named index.

CQL Data Manipulation Commands

1. **INSERT:** Adds columns for a row in a table.
2. **UPDATE:** Updates a column of a row.
3. **DELETE:** Deletes data from a table.
4. **BATCH:** Executes multiple DML statements at once.

CQL Clauses

1. **SELECT:** This clause reads data from a table.
2. **WHERE:** The where clause is used along with select to read a specific data.
3. **ORDERBY:** The orderby clause is used along with select to read a specific data in a specific order.

Cassandra creates a new **SSTable** when the data of a column family in Memtable is flushed to disk. **SSTable** stands for Sorted Strings Table a concept borrowed from Google BigTable which stores a set of immutable row fragments in sorted order based on row keys

The data in a SSTable is organized in six types of component files. The format of a SSTable component file is
<keyspace>-<column family>-[tmp marker]-<version>-<generation>-<component>.db

<keyspace> and **<column family>** fields represent the Keyspace and column family of the SSTable,**<version>**is an alphabetic string which represents SSTable storage format version,**<generation>**is an index number which is incremented every time a new SSTable is created for a column family and **<component>** represents the type of

information stored in the file. The optional **"tmp"** marker in the file name indicates that the file is still being created. The six SSTable components are Data, Index, Filter, Summary, CompressionInfo and Statistics.

For example, I created a column family **data** in Keyspace **usertable** using cassandra-cli and inserted 1000 rows {user0, user1,...user999} with Cassandra version 1.2.5.

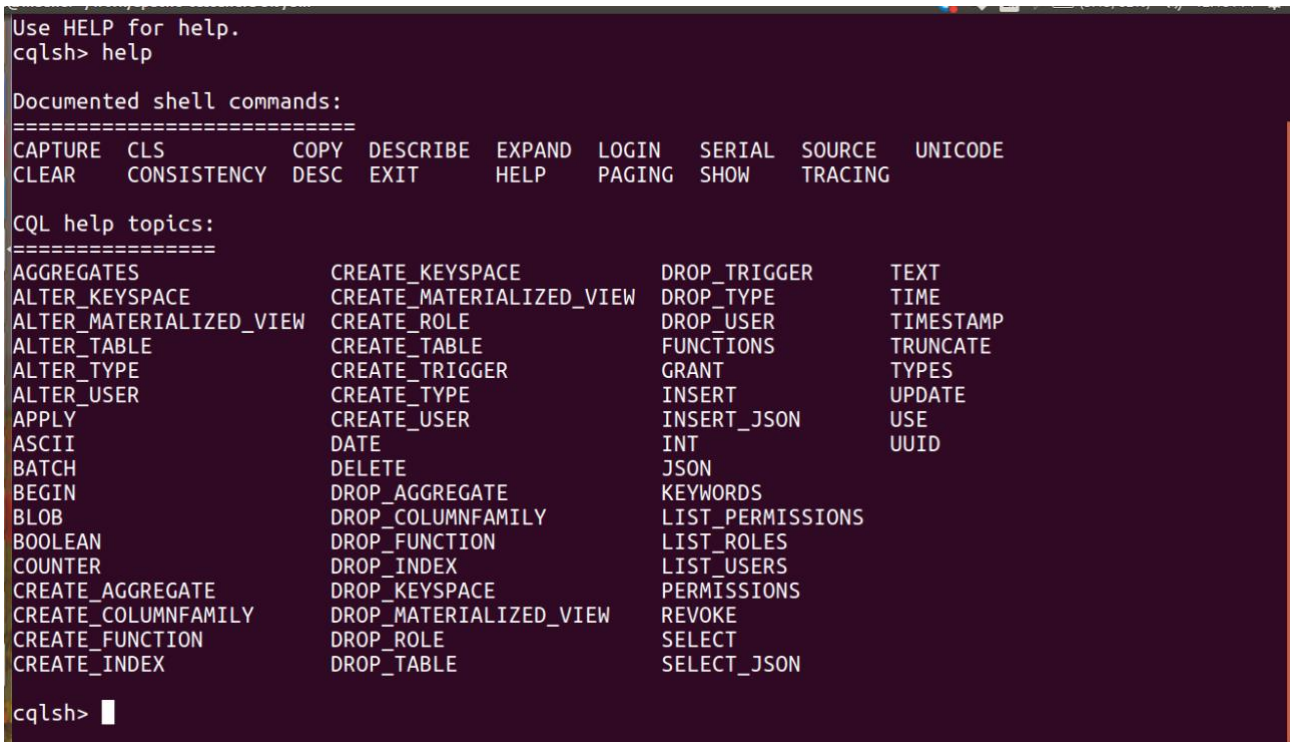
```
create keyspace usertable with placement_strategy =
'org.apache.cassandra.locator.SimpleStrategy' and strategy_options =
{replication_factor:1};
use usertable;
create column family data with comparator=UTF8Type;
```

The SSTables under cassandra/data/usertable/data directory:

```
usertable-data-ic-1-CompressionInfo.db
usertable-data-ic-1-Data.db
usertable-data-ic-1-Filter.db
usertable-data-ic-1-Index.db
usertable-data-ic-1-Statistics.db
usertable-data-ic-1-Summary.db
usertable-data-ic-1-TOC.txt
```

Documented Shell Commands

Help



Keyspace Operations

Creating a Keyspace

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>
```

```
CREATE KEYSPACE "KeySpace Name"
WITH replication = {'class': 'Strategy name', 'replication_factor' : 'No.Of replicas'};


CREATE KEYSPACE "KeySpace Name"WITH replication = {'class': 'Strategy name',
'replication_factor' : 'No.Of replicas'}
AND durable_writes = 'Boolean value';
```

The CREATE KEYSPACE statement has two properties: replication and durable_writes.

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.

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.

Example

Given below is an example of creating a KeySpace.
Here we are creating a KeySpace named bigcassandra.
We are using the first replica placement strategy, i.e.., Simple Strategy.
And we are choosing the replication factor to 3 replica.

```
cqlsh.> CREATE KEYSPACE bigcassandra WITH replication =
{'class':'SimpleStrategy', 'replication_factor' : 3};
```

Example
Given below is an example of creating a KeySpace.

Here we are creating a KeySpace named cassandraspace.
We are using the first replica placement strategy, i.e.., Simple Strategy.
And we are choosing the replication factor to 1 replica.

```
cqlsh.> CREATE KEYSPACE cassandrakeyspace
                                WITH replication = {'class':'SimpleStrategy',
                        'replication_factor' : 3};
```

Verification

You can verify whether the table is created or not using the command Describe. If you use this command over keyspaces.

```
cqlsh> describe keyspaces;

cqlsh> describe cassandraspace;
```

Durable_writes
Using this option, you can instruct Cassandra whether to use 'commitlog' for updates on the current KeySpace. This option is not mandatory and by default, it is set to true. By default, the durable_writes properties of a table is set to true, however it can be set to false. You cannot set this property to simplex strategy. Given below is the example demonstrating the usage of durable writes property

```
cqlsh> CREATE KEYSPACE test
                    ... WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy',
                       'datacenter1' : 3 }

          AND DURABLE_WRITES = false;
```

Verification
You can verify whether the durable_writes property of test KeySpace was set to false by querying the System Keyspace. This query gives you all the KeySpaces along with their properties.

```
SELECT * FROM system_schema.keyspaces;
```

Using a Keyspace

You can use a created KeySpace using the keyword USE. Its syntax is as follows:

```
Cqlsh> Use cassandraspace;
```

Altering a KeySpace

ALTER KEYSPACE can be used to alter properties such as the number of replicas and the durable_writes of a KeySpace. Given below is the syntax of this command.

```
ALTER KEYSPACE <identifier> WITH <properties>
```

```
ALTER KEYSPACE "KeySpace Name"
                WITH replication = {'class': 'Strategy name',
       'replication_factor' : 'No.Of replicas'};
```

The properties of ALTER KEYSPACE are same as CREATE KEYSPACE. It has two properties: replication and durable_writes.

Replication
The replication option specifies the replica placement strategy and the number of replicas wanted.

Durable_writes
Using this option, you can instruct Cassandra whether to use commitlog for updates on the c

Example
Given below is an example of altering a KeySpace.

Here we are altering a KeySpace named cassandraspace.

We are changing the replication factor from 1 to 3.

```
cqlsh.> ALTER KEYSPACE cassandraspace
                    WITH replication = {'class':'NetworkTopologyStrategy',
          'replication_factor' : 3};
```

Altering Durable_writes

You can also alter the durable_writes property of a KeySpace. Given below is the durable_writes property of the test KeySpace.

```
SELECT * FROM system_schema.keyspaces;
```

```
ALTER KEYSPACE test
          WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'datacenter1' :
3}
        AND DURABLE_WRITES = true;
```

Once again, if you verify the properties of KeySpaces, it will produce the following output.

```
SELECT * FROM system.schema_keyspaces;
```

Dropping a KeySpace

You can drop a KeySpace using the command DROP KEYSPACE. Given below is the syntax for dropping a KeySpace.

```
DROP KEYSPACE <identifier>
```

```
DROP KEYSPACE "KeySpace name"
```

Example

The following code deletes the keyspace cassandraspace.

```
Cqlsh>  DROP KEYSPACE cassandraspace;
```

Verification

Verify the keyspaces using the command Describe and check whether the table is dropped as shown below.

Since we have deleted the keyspace cassandraspace you will not find it in the keyspaces list.

Table Operations

Create a Table

You can create a table using the command CREATE TABLE. Given below is the syntax for creating a table.

```
CREATE (TABLE | COLUMNFAMILY) <tablename>
                              ('<column-definition>' , '<column-definition>')
            (WITH <option> AND <option>)
```

Defining a Column
You can define a column as shown below.

```
column name1 data type,
column name2 data type,
example:
age int,
name text
```

Primary Key
The primary key is a column that is used to uniquely identify a row. Therefore, defining a primary key is mandatory while creating a table. A primary key is made of one or more columns of a table. You can define a primary key of a table as shown below.

```
CREATE TABLE tablename(
column1 name datatype PRIMARY KEY,
column2 name data type,
column3 name data type.
)
```

Example
Given below is an example to create a table in Cassandra using cqlsh. Here we are:
Using the keyspace cassandraspace

Creating a table named emp

It will have details such as **employee name, id, city, salary, and phone number. Employee id is the primary key**.

```
cqlsh>USE cassandraspace;
cqlsh:cassandraspace> CREATE TABLE emp(
                        emp_id int PRIMARY KEY,
                        emp_name text,
                        emp_city text,
                        emp_sal varint,
                        emp_phone varint
                        );
```

Verification
The select statement will give you the schema. Verify the table using the select statement as shown below.

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

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

(0 rows)
```

Altering a Table
You can alter a table using the command ALTER TABLE. Given below is the syntax for creating a table.

Syntax

```
ALTER (TABLE | COLUMNFAMILY) <tablename> <instruction>
```

Using ALTER command, you can perform the following operations:

Add a column
Drop a column
Update the options of a table using with keyword

Adding a Column
Using ALTER command, you can add a column to a table. While adding columns, you have to take care that the column name is not conflicting with the existing column names and that the table is not defined with compact storage option.

Given below is the syntax to add a column to a table.

```
ALTER TABLE table name
ADD newcolumn datatype;
```

Example
Given below is an example to add a column to an existing table. Here we are adding a column called emp_email of text datatype to the table named emp.

```
cqlsh:cassandraspace> ALTER TABLE emp
                                ... ADD emp_email text;
```

Verification
Use the SELECT statement to verify whether the column is added or not. Here you can observe the newly added column emp_email.

```
cqlsh:cassandraspace> select * from emp;
emp_id | emp_city | emp_email | emp_name | emp_phone | emp_sal
----------+-------------+---------------+---------------+---------------+------------
```

Dropping a Column
Using ALTER command, you can delete a column from a table. Before dropping a column from a table, check that the table is not defined with compact storage option. Given below is the syntax to delete a column from a table using ALTER command

```
ALTER table name
DROP column name;
```

Example
Given below is an example to drop a column from a table. Here we are deleting the column named emp_email.

```
cqlsh:cassandraspace> ALTER TABLE emp DROP emp_email;
```

Verification
Verify whether the column is deleted using the select statement, as shown below.

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

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



(0 rows)
```

Since emp_email column has been deleted, you cannot find it anymore.

Dropping a Table
You can drop a table using the command Drop Table. Its syntax is as follows:

Syntax

```
DROP  TABLE  <tablename>
```

Example
The following code drops an existing table from a KeySpace.

```
cqlsh:cassandraspace> DROP TABLE emp;
```

Verification
Use the Describe command to verify whether the table is deleted or not. Since the emp table has been deleted, you will not find it in the column families list.

```
cqlsh:cassandraspace> DESCRIBE COLUMNFAMILIES;


employee
```

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. Given below is the syntax of this command.

Syntax

```
TRUNCATE    <tablename>
```

Example
Let us assume there is a table called student with the following data.

s_id	s_name	s_branch	s_aggregate
1	Ram	IT	70

2	Rohan	EEE	75
3	Robin smith	M.tech	72

When you execute the select statement to get the table student, it will give you the following output.

```
cqlsh:cassandraspace> select * from student;
s_id | s_aggregate | s_branch | s_name
------+-----------------+-------------+-----------
1       | 70              | IT          | Ram
2       | 75              | EEE         | Rohan
3       | 72              | MECH        | Robin smith


(3 rows)
```

Now truncate the table using the TRUNCATE command.

```
cqlsh:cassandraspace> TRUNCATE student;
```

Verification
Verify whether the table is truncated by executing the select statement. Given below is the output of the select statement on the student table after truncating.

```
cqlsh:cassandraspace> select * from student;

s_id | s_aggregate | s_branch | s_name
------+-----------------+-------------+------------



(0 rows)
```

Creating an 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 .

```
cqlsh:cassandraspace> CREATE INDEX name ON emp1 (emp_name);
```

Dropping an Index
You can drop an index using the command DROP INDEX. Its syntax is as follows:

```
DROP INDEX <identifier>
```

Given below is an example to drop an index of a column in a table. Here we are dropping the index of the column name in the table emp.

```
cqlsh:cassandraspace> drop index name;
```

Creating Data in a Table

You can insert data into the columns of a row in a table using the command INSERT. Given below is the syntax for creating data in a table.

```
INSERT INTO <tablename>
                    (<column1 name>, <column2 name>....)
          VALUES (<value1>, <value2>....)
         USING <option>
```

Example

Let us assume there is a table called emp with columns (emp_id, emp_name, emp_city, emp_phone, emp_sal) and you have to insert the following data into the emp table.

emp_id	emp_name	emp_city	emp_phone	emp_sal1
1	Ram	Hyderabad	9849617428	50000
2	Rohan	Bangalore	9849617429	40000
3	Robin smith	Chennai	9849617430	45000

Use the commands given below to fill the table with required data.

```
cqlsh:cassandraspace> INSERT INTO emp (emp_id, emp_name, emp_city,
emp_phone, emp_sal) VALUES(1,'Ram', 'Hyderabad', 9849617428, 50000);

cqlsh:cassandraspace> INSERT INTO emp (emp_id, emp_name, emp_city,
emp_phone, emp_sal) VALUES(2,'Rohan', 'Hyderabad', 9849617429, 40000);

cqlsh:cassandraaspace> INSERT INTO emp (emp_id, emp_name, emp_city,
emp_phone, emp_sal) VALUES(3,'Robin smith', 'Chennai', 9849617430, 45000);
```

Verification

After inserting data, use SELECT statement to verify whether the data has been inserted or not. If you verify the emp table using SELECT statement, it will give you the following output.

```
cqlsh:cassandraspace> SELECT * FROM emp;
emp_id | emp_city| emp_name | emp_phone | emp_sal
----------+------------+----------------+----------------+------------
1         | Hyderabad | Ram            | 9849617428 | 50000
2         | Hyderabad | Rohan          | 9849617429  | 40000
3         |     Chennai | Robin smith  | 9849617430 | 45000
```

```
(3 rows)
```

Here you can observe the table has populated with the data we inserted.

Updating Data in 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.

While updating rows, if a given row is unavailable, then UPDATE creates a fresh row. Given below is the syntax of UPDATE command:

```
UPDATE <tablename>
          SET <column name> = <new value>
               <column name> = <value>....
 WHERE <condition>
```

Example
Assume there is a table named emp. This table stores the details of employees of a certain company, and it has the following details:

emp_id	emp_name	emp_city	emp_phone	emp_sal1
1	Ram	Hyderabad	9849617428	50000
2	Rohan	Bangalore	9849617429	40000
3	Robin smith	Chennai	9849617430	45000

Let us now update emp_city of robin to Delhi, and his salary to 50000. Given below is the query to perform the required updates.

```
cqlsh:cassandraspace> UPDATE emp SET emp_city='Delhi',emp_sal=50000
                                          WHERE emp_id=2;
```

Verification
Use SELECT statement to verify whether the data has been updated or not. If you verify the emp table using SELECT statement, it will produce the following output.

```
cqlsh:cassandraspace> SELECT * FROM emp;
emp_id | emp_city| emp_name | emp_phone | emp_sal
```

```
----------+------------+----------------+-----------------+------------
1         | Hyderabad | Ram             | 9849617428 | 50000
2         | Delhi         | Rohan         | 9849617429  | 40000
3         |     Chennai | Robin smith  | 9849617430 | 45000


(3 rows)
```

Here you can observe the table data has got updated.

Reading Data using Select Clause

SELECT clause is used to read data from a table in Cassandra. Using this clause, you can read a whole table, a single column, or a particular cell. Given below is the syntax of SELECT clause.

```
SELECT FROM <tablename>
```

Example

Assume there is a table in the keyspace named emp with the following details:

emp_id	emp_name	emp_city	emp_phone	emp_sal1
1	Ram	Hyderabad	9849617428	50000
2	Rohan	Bangalore	9849617429	40000
3	Robin smith	Chennai	9849617430	45000
4	Ranga	Pune	9849617120	30000

The following example shows how to read a whole table using SELECT clause. Here we are reading a table called emp.

```
cqlsh:cassandraspace> SELECT * FROM emp;
emp_id | emp_city| emp_name | emp_phone | emp_sal
----------+------------+----------------+-----------------+------------
1         | Hyderabad  | Ram             | 9849617428 | 50000
2         | Delhi          | Rohan          | 9849617429  | 40000
3         | Chennai      | Robin smith  | 9849617430 | 45000
4         | Pune          | Ranga          | 984961720  | 30000


(3 rows)
```

Reading Required Columns

The following example shows how to read a particular column in a table.

```
cqlsh:cassandraspace> SELECT emp_name, emp_sal from emp;

emp_name | emp_sal
----------+---------
```

```
Ram      | 50000
Robin    | 50000
Ranga    | 30000
Robin smith | 50000



(4 rows)
```

Where Clause
Using WHERE clause, you can put a constraint on the required columns. Its

syntax
is as follows:

```
SELECT FROM <table name> WHERE <condition>;
```

Note: A WHERE clause can be used only on the columns that are a part of primary

key or have a secondary index on them.
In the following example, we are reading the details of an employee whose salary is 50000. First of all, set secondary index to the column emp_sal.

```
cqlsh:cassandraspace> CREATE INDEX ON emp(emp_sal);

cqlsh:cassandraspace> SELECT * FROM emp WHERE emp_sal=50000;

emp_id | emp_city| emp_name | emp_phone| emp_sal
--------+-----------+----------+------------+---------
1          | Hyderabad | Ram          | 9848022338 | 50000
2          | null           | Rohan        | 9848022339 | 50000
3          | Chennai     | Robin smith| 9848022330 | 50000
```

Deleting Data from a Table
You can delete data from a table using the command DELETE. Its syntax is as follows:

```
DELETE FROM <identifier> WHERE <condition>;
```

Example
Let us assume there is a table in Cassandra called emp having the following data:

emp_id	emp_name	emp_city	emp_phone	emp_sal1
1	Ram	Hyderabad	9849617428	50000
2	Rohan	Bangalore	9849617429	40000
3	Robin smith	Chennai	9849617430	45000

The following statement deletes the emp_sal column of last row:

```
cqlsh:cassandraspace> DELETE emp_sal FROM emp WHERE emp_id=3;
```

Verification
Use SELECT statement to verify whether the data has been deleted or not. If you verify the emp table using SELECT, it will produce the following output:

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

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


(3 rows)
```

Since we have deleted the salary of Rahman, you will observe a null value in place of salary.

Deleting an Entire Row
The following command deletes an entire row from a table.

```
cqlsh:cassandraspace> DELETE FROM emp WHERE emp_id=3;
```

Verification
Use SELECT statement to verify whether the data has been deleted or not. If you verify the emp table using SELECT, it will produce the following output:

```
cqlsh:cassandraspace> select * from emp;
emp_id | emp_city| emp_name | emp_phone| emp_sal
----------+-------------+---------------+----------------+------------
1 | Hyderabad | Ram    | 9848022338 | 50000
2 | Delhi         | Rohan | 9848022339 | 50000|


(2 rows)
```

Since we have deleted the last row, there are only two rows left in the table.

CQL DATATYPES
CQL provides a rich set of built-in data types, including collection types. Along with these data types, users can also create their own custom data types. The following table provides a list of built-in data types available in CQL.

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
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

Collection Types
Cassandra Query Language also provides a collection data types. The following table provides a list of Collections available in CQL.

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.

User-defined datatypes: Cqlsh provides users a facility of creating their own data types. Given below are the commands used while dealing with user defined datatypes.

CREATE TYPE: Creates a user-defined datatype.
ALTER TYPE: Modifies a user-defined datatype.
DROP TYPE: Drops a user-defined datatype.
DESCRIBE TYPE: Describes a user-defined datatype.
DESCRIBE TYPES: Describes user-defined datatypes.

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:cassandraspace> 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:cassandraspace> 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:cassandraspace> UPDATE data
                                ... SET email = email +['xyz@cassandraspace.com']
                                ... where name = 'ramu';
```

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

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

name | email
------+-----------------------------------------------------------
ramu | ['abc@gmail.com', 'cba@yahoo.com', 'xyz@cassandraspace.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:cassandraspace> 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:cassandraspace> 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:cassandraspace> 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:cassandraspace> 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:cassandraspace> 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:cassandraspace> 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:cassandraspace> 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:cassandraspace> select * from data3;

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


(1 rows)
```