Cassandra

Apache Cassandra is NoSQL database distributed management system which is free and open-source. Cassandra is originally developed at Facebook and it was released as open-source in google code, later apache taken as incubator project for further development. Without compromising on performance, Cassandra can provide great linear scalability and high availability.

Who can Learn ?

Software professionals who have basic RDBMS knowledge and interested to learn this NoSQL database technology can be benefitted through this tutorial with many basic operations which can be performed.

Prerequisites

Basic RDBMS knowledge is required to start with this technology. This database is being used along with big data technologies and where the high scalability and performance is needed.

Relational vs NoSQL

NoSQL databases differ from RDBMS technology in four main areas:

1. Data Model

2. Data Structure

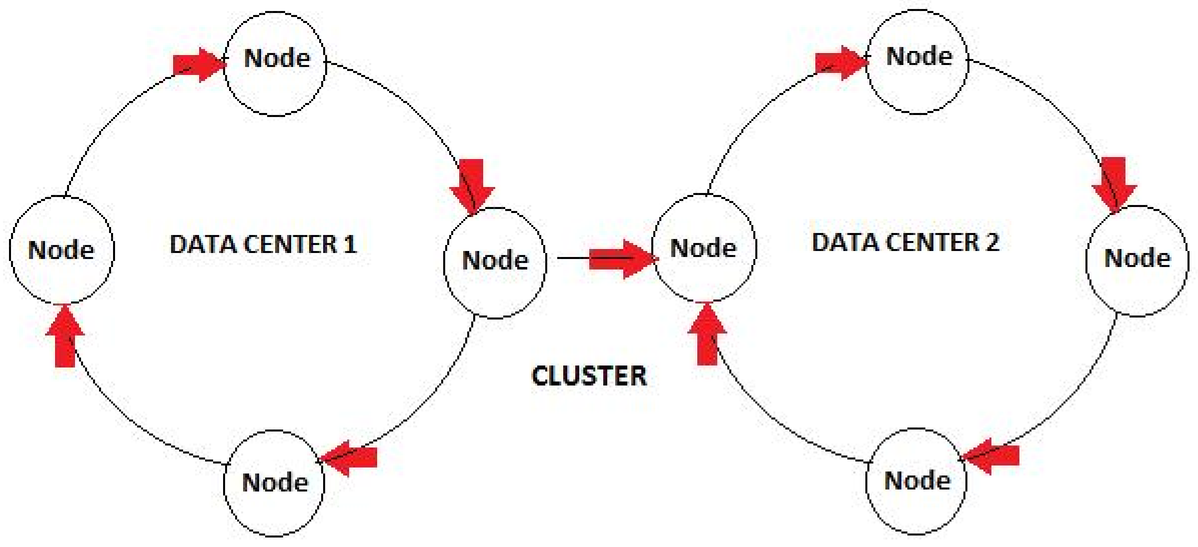
3. Scaling

4. Usage Cost

	RDBMS	NoSQL
Data Model	**RDBMS is traditional way storing the structured data in relational way. Without schema we cannot build/connect to an application.**	**NoSQL is latest technology which is designed to handle the BigData. We can build an application without having to define the schema.**
Data Structure	**Used to store the data which is structured and defined by relations.**	**Designed to handle the unstructured data (Ex: Text, Email , Image, Video).**
Scaling	**Required single server to host the entire database and Vertical scaling is costly to afford.**	**This is much cheaper to scale compared to relational database.**
Usage Cost	**Licensed and costly.**	**Open-source and ready to use.**

Cassandra – Architecture

Cassandra is a Ring based model designed for Bigdata applications, where data is distributed across all nodes in the cluster evenly using consistent hashing algorithm with no single point of failure.In Cassandra, multiple nodes that forms a cluster in a data centre which communicates with all nodes in other data centers using gossip protocol.



Each node exchange information across cluster every second and captures the write operations in a commit log in sequential manner.Thereby, data will be indexed and written to in-memory structure Memtable.One the memory is full, then the data will be written to disk in SSTable datafile. All data which is written is partitioned and replicated across the cluster.Periodical consolidation of SSTable will be done through a process called compaction.

When a request is received by a node in cluster, then it will act as a coordinator to collect the data which is distributed across other nodes and takes the responsibility of serving the requested data back to client.

Cassandra uses CQL as query language. CQL looks like same as SQL which eases the query language. CQL can be performed through command line utility Cqlsh.

Cassandra – Column Family

It is a NoSQL format which contains columns of key-value pair, where the key is mapped to value. Generally, In relational databases we will work with tables in which every key-value pair is a row.

In NoSQL each column is a tuple (Triplet) consists of column name , value and timestamp. whereas in RDBMS this data will be in the form of table.

Row

Row is the smallest unit that stores related data in Cassandra and individual rows constitute a column family

Row key: uniquely identifies a row in a column family.

Row: stores pairs of column keys and column values.

Column key: uniquely identifies a column value in a row.

Column value: stores one value or a collection of values.

Lets say accesing the data which is distributed across nodes would be time consuming if it is saved in table format. Also it is inefficient to read all the columns that would make a row in a relational table.

Example:

```
RDBMS: Table having the columns ID , Name , Age, Gender, City.
NoSQL: column family will be " ID , NAME , Age" , " Gender,City".
```

Now if you have query which need only the males count in a particular city , Entire table needs to be read in relational database.

In NoSQL / Distributed data store, It will access only the second chunk of standard column family as the rest of the information is not required.

Cassandra – Components

Node:

Node is a server/system where data is stored.

Data Center:

A Collection of nodes are called as Data center.Data center may be physical or virtual.Datacenters should never span physical locations.

Cluster:

Cluster contains one or more data centers which can span physical locations.

Commit log:

All database write operations are written to commitlog which can be used for crash recovery.

Mem-table:

Once after the Data written in commitlog, the same will be written in Mem-table for temporarily.

SSTable:

Once the Mem-table reached the threshold, data will be flushed to SSTable which is stored on disk sequentially and maintained for each and every Cassandra table.

Gossip:

A communication protocol to discover, share location and information about other nodes in cluster. This information will be persisted by each node in cluster to use immediately when a node restarts.

Partitioner:

Partitioner is responsible to distribute the data across the nodes in cluster and decides on which node the first copy/replica of data to be placed.

Replication factor:

It is the total number of replicas across the cluster.

Example:

```
Replication factor 1 ===> one copy of each row on one node.
Replication factor 2 ===> two copies of each row which are placed on different nodes.
```

Replica Placement Strategy:

There are 2 types of strategies.

1. Simple Strategy

when you have One Data center, this strategy place the first replica on the node selected by partitioner and remaining replicas in clockwise direction.

2. Network Topology Strategy

This is used when we have more than 2 data centers. This strategy places multiple replicas in each data center.

Cassandra – Data Model

Cassandra data model was clearly explained with detailed pictorial representations as below.

Column

Column is a Key-value pair which also contains a time-stamp along with it.

Super Column

A Column that contains one or more columns (Array of columns) can be called as Super Column.

Column Family

A Column Family that contains columns of related data. Column Families are accessed by row keys.

Super Column Family

A row which consists a container for array of super columns accessed by their names. Like Column Families, Super Column Families are accessed by row keys.

Cassandra – Data Types

In this tutorial, we will learn about the Data Types in Cassandra CQL language. DataTypes generally define the type of data a column can hold along with their own significance. CQL language supports the below list Data types:

1. Native Types

2. Collection Types

3. User-defined Types

4. tuple types

TYPE	CONSTANTS	DESCRIPTION
ascii	string	ASCII character string
bigint	integer	64-bit signed long
blob	blob	Arbitrary bytes
boolean	boolean	Either true or false
counter	integer	Counter column
date	integer, string	A date (with no corresponding time value).
decimal	integer, float	Variable-precision decimal

double	**integer float**	**64-bit IEEE-754 floating point**
float	**integer, float**	**32-bit IEEE-754 floating point**
inet	**string**	**An IP address**
int	**integer**	**32-bit signed int**
smallint	**integer**	**16-bit signed int**
text	**string**	**UTF8 encoded string**
time	**integer, string**	**A time with nanosecond precision**
timestamp	**integer, string**	**A timestamp with millisecond precision.**
timeuuid	**uuid**	**Version 1 UUID**
tinyint	**integer**	**8-bit signed int**
uuid	**uuid**	**A UUID (of any version)**
varchar	**string**	**UTF8 encoded string**
varint	**integer**	**Arbitrary-precision integer**

Collection Types

There are 3 types of collection datatypes :

1. Maps

Map is a set of Key-value pairs.Where keys are unique.

Example:

```
CREATE TABLE Blog (
id int PRIMARY KEY,
data map <text, text>
);
```

2. Sets

Set is collection of unique values.

Example:

```
CREATE TABLE Blog (
ID int PRIMARY KEY,
tags set<text>
);
```

3. Lists

List is a collection of non-unique values which can be ordered with their position.

Example:

```
CREATE TABLE plays (
id text PRIMARY KEY,
Name text,
posts list<int>
)
```

User Defined Type

UDT can be created, modified and removed using below commands.

– CREATE TYPE

– ALTER TYPE

– DROP TYPE

Example:

```
CREATE TYPE Blogger ( city text, Name text, Phone int )

CREATE TYPE Blogs ( ID int, Blogname text, Address map<text, Blogger> )
```

Tuples

Tuples are a set of different types of elements.

Example:

```
CREATE TABLE Blogs (BlogId text, BlogType tuple<int, text> )
```

Cassandra – Cqlsh

CQL – Cassandra Query Language is the language to communicate with Cassandra Database.

CQLSH – This is the Command Line Utility used to execute the commands to communicate with Cassandra database.

To start the utility we need to give the command cqlsh either in linux terminal or windows command prompt. The default listen port for cqlsh is 9042.

Syntax:

```
cqlsh [options] [host [port]]
```

Example:

```
casa@casa1:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

Note: Before starting the utility, you need to check if the Cassandra Path to bin directory is configured or not.

Below is an example to start cqlsh with authentication

```
cqlsh 192.168.2.101 9042 -u admin -p admin
```

To terminate the command we need to use the semicolon where as a new line will not terminate the command rather it is used to spread the program or commands across multiple lines for more clarity and indentation.

Cassandra – Cqlsh Commands

In Cassandra, cqlsh commands can be used through CQL interactive terminal. These commands will work only within the Cqlsh shell.

SHOW VERSION – This command will show you the cqlsh, Cassandra, CQL, and native protocol versions.

Example:

```
casa@casa1:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

SHOW HOST – This command will give you the IP address and port number, cluster name.

Example:

```
cqlsh> show host
Connected to Test Cluster at 127.0.0.1:9042.
cqlsh>
```

SOURCE – This command allow you to read the contents of a file and executes CQL statements in the file.

Syntax:

```
SOURCE <string filename>
```

Example:

```
cqlsh> SOURCE '/home/cassandra/Execute.cql'
```

CAPTURE – This command will capture the command output and append to a mentioned file.

Syntax:

```
CAPTURE '<file>';
CAPTURE OFF;
CAPTURE;
```

HELP – This command will provide us the information about cqlsh commands.

TRACING – This Command Enables or disables tracing in cqlsh command prompt.

Syntax:

```
TRACING ON
TRACING OFF
```

PAGING – This command enables or disables paging, or set the page size. This will be helpful when we have large content as output.

Syntax:

```
PAGING ON
PAGING OFF
PAGING <page size in rows>
```

EXPAND – This command enables or disables vertical printing of rows. Enabling EXPAND is useful when many columns are fetched.

Syntax:

```
EXPAND ON
EXPAND OFF
```

LOGIN – This command Authenticate the Cassandra user for the current session.

Syntax:

```
LOGIN <username> [<password>]
```

EXIT – This command will end the current session and terminates the cqlsh process.

Syntax:

```
EXIT
QUIT
```

CLEAR – This command clears the console.

Syntax:

```
CLEAR
CLS
```

DESCRIBE – This command gives the description of all below Schema Elements.

Syntax:

```
DESCRIBE CLUSTER
DESCRIBE SCHEMA
DESCRIBE KEYSPACES
DESCRIBE KEYSPACE <keyspace name>
DESCRIBE TABLES
DESCRIBE TABLE <table name>
DESCRIBE INDEX <index name>
DESCRIBE MATERIALIZED VIEW <view name>
DESCRIBE TYPES
DESCRIBE TYPE <type name>
DESCRIBE FUNCTIONS
DESCRIBE FUNCTION <function name>
DESCRIBE AGGREGATES
DESCRIBE AGGREGATE <aggregate function name>
```

Example:

```
cqlsh> describe cluster
Cluster: Test Cluster
Partitioner: Murmur3Partitioner
cqlsh>
```

COPY TO – This command copies data from a table to a CSV file.

Syntax:

```
COPY <table name> [(<column>, ...)] TO <file name> WITH <copy option> [AND <copy option> ...]
```

Note:

If we need specific columns , then we need to specify the column list, Otherwise it will copy all columns from the table to the CSV file.

Cassandra – Create KeySpace

KeySpace in NoSQL database is just like a schema in regular RDBMS concept, Anyhow it does not have any concrete structure. In NoSQL database, there will be one keyspace per application.

A Keyspace contains column families or super columns. Each super column contains one or more column family, each column family contains at least one column.

Syntax:

```
CREATE  KEYSPACE | SCHEMA  IF NOT EXISTS keyspace_name
WITH REPLICATION = map
AND DURABLE_WRITES =  true | false
```

Replication factor:

It is the total number of replicas across the cluster.

Example:

```
Replication factor 1 ===> one copy of each row on one node.
Replication factor 2 ===> two copies of each row which are placed on different nodes.
```

Replica Placement Strategy:

There are two types of strategies.

1. Simple Strategy

when you have One Data center, this strategy place the first replica on the node selected by partitioner and remaining replicas in clockwise direction.

Example:

Let us create a KeySpace i2Tutorials

```
CREATE KEYSPACE i2Tutorials
WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
```

Note: We used replication factor 3 which is ideal.

2. Network Topology Strategy

This is used when we have more than 2 data centers. This strategy places multiple replicas in each data.

Example:

```
CREATE KEYSPACE i2Tutorials
WITH replication = {'class': 'NetworkTopologyStrategy', 'DC1' : 1, 'DC2' : 3}
AND durable_writes = false;
```

Lets Create Keyspace called i2Tutorials.

```
cqlsh> CREATE KEYSPACE i2Tutorials
WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
Cqlsh> Describe keyspaces;
i2tutorials  system_schema system_auth system system_distributed system_traces
```

Validate the KeySpace creation

```
cqlsh:system_schema> SELECT * FROM system_schema.keyspaces;
keyspace_name      | durable_writes | replication
--------------------+----------------+---------------------------------------
I2Tutorials | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor':'3'}
system_auth | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor':'1'}
system_schema | True | {'class': 'org.apache.cassandra.locator.LocalStrategy'}
system_distributed | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor':'3'}
system | True |                 {'class': 'org.apache.cassandra.locator.LocalStrategy'}
system_traces | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor':'2'}
```

Cassandra – Alter KeySpace

Alter Keyspace command allows you to alter the replicaton factor, strategy name and durable writes attributes. Below the syntax for Alter KeySpace command.

Syntax:

```
Alter Keyspace KeyspaceName with replication=
{'class':'StrategyName', 'replication_factor': no of replications on different nodes}
with DURABLE_WRITES=true/false
```

Note: Using Alter command Keyspace Name cannot be changed in cassandra.

By default, DURABLE_WRITES value is true. If set to false, no updates will be written to the commit log.

Example:

```
cqlsh> ALTER KEYSPACE i2Tutorials
WITH replication = {'class': 'NetworkTopologyStrategy', 'replication_factor' : 3};
Cqlsh> Describe keyspaces;
i2tutorials  system_schema system_auth  system system_distributed system_traces
```

Validate the ALTER KeySpace

```
cqlsh:system_schema> SELECT * FROM system_schema.keyspaces;
keyspace_name      | durable_writes | replication
--------------------+----------------+----------------------------------------------
I2Tutorials | True | {'class': 'org.apache.cassandra.locator.NetworkTopologyStrategy', 'replication_factor':'3'}
system_auth | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor':'1'}
system_schema | True | {'class': 'org.apache.cassandra.locator.LocalStrategy'}
system_distributed | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor':'3'}
system | True | {'class': 'org.apache.cassandra.locator.LocalStrategy'}
system_traces | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor':'2'}
```

Cassandra – Drop KeySpace

In this tutorial, we will learn about the DROP Keyspace command which allows you to drop the keyspace which includes all the data, column families,UTD,indexes from cassandra.

Syntax:

```
DROP KEYSPACE [ IF EXISTS ] keyspace_name
```

Example:

```
DROP KEYSPACE i2Tutorials;
```

Validate the Drop KeySpace.

```
Cqlsh> Describe keyspaces;
system_schema system_auth  system system_distributed system_traces
```

Cassandra – Create Table

In this tutorial, we will learn about the CREATE Table command in Cassandra which allows us to create new table using cqlsh by providing the valid column names, data types.

Syntax:

```
CREATE TABLE keyspace_name.table_name
( column_definition, column_definition, ...)
WITH property AND property ...
```

Example:

```
CREATE TABLE LearningStore.users (
user_id int PRIMARY KEY,
created_date timestamp,
first_name text,
last_name text,
email text
);
```

In the above example, LearningStore is keyspace and user_id is where the primary key which consists of only one column.

Compounded PrimaryKey:

This consists more than one column as primary key. Cassandra treats the first column as partition key.

```
CREATE TABLE LearningStore.users (
user_idint PRIMARY KEY,
created_date timestamp,
first_name text,
last_name text,
email text,
PRIMARY KEY (user_id, first_name)
);
```

Column definition Syntax:

```
column_namecql_type
| column_namecql_type PRIMARY KEY
| PRIMARY KEY ( partition_key )
| column_namecollection_type
```

Example:

```
CREATE TABLE LearningStore.users (
user_idint PRIMARY KEY,
created_date timestamp,
first_name text,
last_name text,
emails set<text>,
lessons list<int>,
tobecompleted map<timestamp, text>
);
```

Validate the Table Creation

Make sure that you are in LearningStoreKeyspace before giving the below command.

```
cqlsh:LearningStore> select * from users;
```

Cassandra – Alter Table

Using Alter Table command you can change the data type of column, can add or remove the columns to existing table and change table properties too.

Syntax:

```
ALTER TABLE keyspace_name.table_name
ALTER column_name TYPE cql_type
| ( ADD column_namecql_type )
| ( DROP column_name )
| ( RENAME column_name TO column_name )
| ( WITH property AND property ... )
```

Example:

```
CREATE TABLE LearningStore.users (
user_id int PRIMARY KEY,
created_date timestamp,
first_name text,
last_name text,
email text
);
```

Change the type of column:

```
ALTER TABLE users ALTER Created_date TYPE date;
```

Adding a new column:

```
ALTER TABLE users ADD City varchar;
```

Removing the column:

```
ALTER TABLE users DROP City;
```

Cassandra – Drop Table

Drop table can be used to drop the table.

Syntax:

```
DROP TABLE <tablename>
```

Example:

Before giving this command , you should check that if you are in proper keyspace and you are dropping the exact table what you want.

```
cqlsh> DROP TABLE users;
```

Validate

If the table get deleted , then you should not be able to see the table name after giving the below command in the respective KeySpace.

```
cqlsh>Describe columnfamilies;
```

Cassandra – Truncate Table

In this tutorial, we will learn about TRUNCATE Table command in Cassandra which will allow to truncate all the data from a table. The Date removed is irreversible.

Note: Truncating a table triggers an automatic snapshot which backs up the data only, not the schema.

Syntax:

```
TRUNCATE keyspace_name.table_name;
```

Example:

```
cqlsh> Truncate users;
```

Cassandra – Insert Data

Insert command allows us to creat or insert the data records into the columns. Here it is not required to define all columns and all those missing columns will get no space on disk.So if columns Exists, it is updated.

Syntax:

```
INSERT INTO keyspace_name.table_name
( column_name, column_name...)
VALUES ( value, value ... )
USING option AND option
```

Example:

```
INSERT INTO users (user_id, first_name, last_name, email)
VALUES('1002', 'cassandra', 'julie', 'julie@email.com');
```

Validate the data:

```
Cqlsh> select * from users;
```

Cassandra – Update Data

Update Command allows us to update the one or more columns values in cassandra table. To update the multiple columns , we need to seperate the name-value pairs using commas.

Syntax:

```
UPDATE keyspace_name.table_name
USING option AND option
SET assignment, assignment, ...
WHERE row_specification
```

Example:

```
UPDATE users SET city = 'Banglore' WHERE userID = '1020'
```

Cassandra – Delete Data

In this tutorial, we will learn about the DELETE command in Cassandra which allows to remove one or more columns data in a table and also it removes entire row if no column is specified.

Syntax:

```
DELETE column_name, ... | ( column_name term )
FROM keyspace_name.table_name
USING TIMESTAMP integer
WHERE row_specification
```

Example:

```
DELETE email FROM users WHERE user_id = '1020';
```

Validate Data:

```
Select * from users;
```

Cassandra – Select data

In this tutorial, we will learn about the SELECT command in Cassandra which is used to retrieve the data from the cassandra table. We can perform various projections using the SELECT statement.

Syntax:

```
SELECT select_expression
FROM keyspace_name.table_name
WHERE relation AND relation ...
ORDER BY ( clustering_column ( ASC | DESC )...)
LIMIT n
ALLOW FILTERING
```

Select Expression will be given as input to the select statement, where the output depends on the select expression.

```
SELECT * from users;
Select count(*) from users;
```

Cassandra – Create Index

Create Index command allows to create new index on the specified column for a table. Cassandra indexes the data during the execution of command and also the new data that is being inserted once after the creation of index.

Syntax:

```
CREATE CUSTOM INDEX index_name
ON keyspace_name.table_name ( column_name )
(USING class_name) (WITH OPTIONS = map)
```

Example:

```
CREATE TABLE LearningStore.users (
user_idint PRIMARY KEY,
created_date timestamp,
first_name text,
last_name text,
email text
);
CREATE INDEX email_idx ON LearningStore.users(email);
CREATE INDEX ON myschema.users (zip);
```

Cassandra – Drop Index

A DROP INDEX command allows us to drop the existing index. If the index was not given a name during creation, the index name is <table_name>_<column_name>_idx.

Syntax:

```
DROP INDEX name
```

Example:

```
DROP INDEX email_idx;
```

Cassandra – Collection Types

Collections provide the simplest way to handle multiple tasks.The maximum size of an item in collection is 64K. Also it is advisable to keep collections small to avoid delay during query execution.Even though if you insert more than 64K data that will result into dataloss, because only 64K is queryable.

SET Type

Set Type will help us the situations when we need to include multiple emails.

Example:

Create a table

```
CREATE TABLE users (
user_id text PRIMARY KEY,
first_name text,
last_name text,
emails set<text>
);
```

Insert data into the set. Set values must be unique.

```
INSERT INTO users (user_id, first_name, last_name, emails)
VALUES('1020', 'cassandra', 'julie', {'c@emails.com', 'julie@email.com'});
```

If you want to add one more email

```
UPDATE users SET emails = emails + {'casa@email.com'} WHERE user_id = '1020';
```

List Type

List Type can be used when data elements required in an order.It will return the values as per the index value in the list.

Example:

Add the list of technologies known by users to a table by adding a column technologies of the list type to the users table.

```
ALTER TABLE users ADD technologies list<text>;
```

Use the UPDATE command to insert values into the list.

```
UPDATE users SET technologies = ['java', 'C++'] WHERE user_id = '1020';
```

MAP

Map type is used to store key value pairs which are mapped.Each element of the map can be modified, replace, delete and query.

Example:

Let us add a column which is map type to show events.

```
ALTER TABLE users ADD todo map<DATE, text>;
```

Using the INSERT or UPDATE command lets add the map data

```
UPDATE users
SET todo =
{
```

```
'2016-08-22' : 'Seminar Event',
'2016-10-10' : 'Auditing'
}
WHERE user_id = '1020';
```

Cassandra – Batch

In tutorial, we will learn about the Batch command in Cassandra which allows us to write multiple DML statements in a single shot. DML statements include the Insert, Update, Delete commands.

DML statements:

INSERT

UPDATE

DELETE

Syntax:

```
BEGIN ( UNLOGGED | COUNTER ) BATCH
USING TIMESTAMP timestamp
dml_statement;
dml_statement;
...
APPLY BATCH;
```

Example:

```
BEGIN BATCH
INSERT INTO users (user_id,first_name, second_name) VALUES ('1001', 'Cassandra', 'julie')
UPDATE users SET city = 'Banglore' WHERE userID = '1020'
DELETE first_name FROM users WHERE userID = '1234'
APPLY BATCH;
```