UNIT – III

Introduction to Cassandra

Introduction:

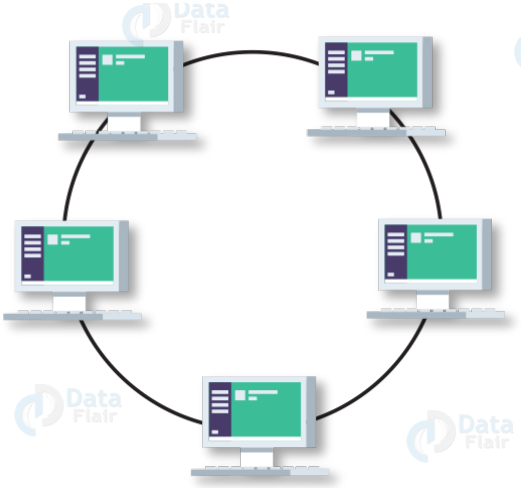
Cassandra is a popular NoSQL database. It has a wide variety of features that has made Cassandra immensely popular. Apache Cassandra was born at Facebook. It is built on top of Amazon's dynamo and Google's BigTable. Cassandra does not compromise on availability. It does not follow master slave architecture. Hence there is no question of single point failure. It is a highly scalable, high performance distributed database. It distributes and manages gigantic volumes of data across commodity servers. It is a column oriented database designed to support peer-to-peer symmetric nodes. It adheres to availability and partition tolerance properties of CAP theorem. A few companies that have deployed Cassandra and have benefitted immensely include

- Twitter
- NetFlix
- Cisco
- Adobe
- eBay

Features of Cassandra:

The following are the features of Cassandra:

- Peer-to-peer Network: Cassandra is designed to distribute and manage large data load across multiple nodes in a cluster constituted of commodity hardware. It does not follow master slave architecture which means it does not have a single point failure. A node in Cassandra is structurally identical to any other node. It has a peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster. In Cassandra, each node is independent and at the same time interconnected to other nodes. All the nodes in a cluster play the same role. Every node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster. In the case of failure of one node, Read/Write requests can be served from other nodes in the network.



- Gossip and failure detection:
  In Cassandra, the communication between nodes is often like peer-to-peer communication, where every node talks to the other. If that's the case, then all the nodes talk to one another, and there is a lot of communication happening. The Gossip Protocol is a method to resolve this communication chaos. In Cassandra, when one node talks to another, the node which is expected to respond, not only provides information about its status, but also provides information about the nodes that it had communicated with before. Through this process, there is a reduction in network log, more information is kept and efficiency of information gathering increases. The main feature of the protocol is to provide the latest information of any node respectively.
  An important feature of Gossip Protocol is Failure Detection. Basically, when two nodes communicate with one another; for instance, Node A to Node B, then Node A sends a message 'gossipdigestsynmessage', which is very similar to TCP protocol to Node B. Here, Node B, once receives the message, sends an acknowledgement message 'ack', and then Node A responds with an acknowledgement message to Node B's 'ack' message. This is known as the 3 way handshake.
  If in case, the node goes down and does not send the ack message, then it will be a mark down. Even when the nodes are down, the other nodes will be periodically pinging and that is how the failure detection happens.

- Partitioner
  A partitioner takes a call on how data is distributed on the various nodes in a cluster. It also determines the node on which the first copy of data is placed. Basically, it is a hash function to compute the token of the partition key. The partition key helps to identify a row uniquely

- Replication factor

Replication factor determines the number of copies of data that will be stored across nodes in a cluster. If one wishes to store only one copy of data of each row on one node then the replication factor is set to1. The number of copies of each row of data determines the replication factor. However, the replication factor should be more than one but less than the number of nodes in a cluster. A replication strategy is employed to determine which nodes to place data on. Two replication strategies are available:
1. Simple Strategy.
2. Network Topology Strategy.

The preferred one is Network Topology Strategy as it is simple and supports easy expansion to multiple data centers, should there be a need.

- Anti-Entropy and read repair
  A cluster is made up of several nodes. Since the cluster is constituted of commodity hardware, it is prone to failure. In order to achieve fault tolerance, a given piece of data is replicated on one or more nodes. A client can connect to any node in the cluster to read data. How many nodes will be read before responding to the c1ient is based on the consistency level specified by the client. IF the client-specified consistency is not met, thread operation blocks. There is a possibility that few of the nodes may respond with an out-of-date value. In such a case, Cassandra will initiate a read repair operation to bring the replicas with stale values up to date. For repairing unread data, Cassandra uses an anti-entropy version of the gossip protocol. Anti-entropy implies comparing all the replicas of each piece of data and updating each replica to the newest version. The read repair operation is performed either before or after returning the value to the

client as per the specified consistency level.

- Writes in Cassandra:
  When a client initiates a write request it is first written to the commit log. A write is taken as successful only if it is written to the commit log. The next step is to push the write to a memory resident data structure called Memtable. A threshold value is defined in the Memtable. When the number of objects stored in the Memtable reaches a threshold the contents of Memtable are flushed to the disk in a file called SSTable (Sorted String Table). Flushing is a non-blocking operation. It is possible to have multiple Memtables For a single column family. One out of them is current and the rest are waiting to be flushed.

- Hinted handoffs:
  Cassandra works on the philosophy that it will always be available for writes. Assume that we have a cluster of three nodes - Node A, Node B, and Node C. Node C is down for some reason. We are maintaining a replication factor of 2 which implies that two copies of each row will be stored on two different nodes. The client makes a write request to Node A. Node A is the coordinator and serves as a proxy between the client and the nodes on which the replica is to be placed. The client writes Row K to Node A. Node A then writes Row K to Node B and scores a hint for Node C. The hint will have the following information:

1. Location of the node on which the replica is to be placed.
2. Version metadata.
3. The actual data.

  When Node C recovers and is back to the functional self, Node A reacts to the hint by forwarding the data to Node C.

- Tunable consistency:
  One of the features of Cassandra that has made it immensely popular is its ability to utilize tunable consistency. The database systems can go for either strong consistency or eventual consistency. Cassandra can cash in on either flavor of consistency depending on the requirements.
  a. **Strong consistency:** If we work with strong consistency, it implies that each update propagates to all locations where that piece of data resides. Let us assume a single data center setup. Strong consistency will ensure that all of the servers that should have a copy of the data, will have it, before the c1ient is acknowledged with a success. If we are wondering whether it will impact performance, yes it will. It will cost a few extra milliseconds to write to all servers.
  b. **Eventual consistency:** If we work with eventual consistency, it implies that the client is acknowledged with a success as soon as a part of the cluster acknowledges the write.

Objective: To get more details on the existing keyspaces such as keyspace name, durable writes, strategyclass, strategy options, etc.
Command:
SELECT * FROM system.schema_keyspaces;

Objective: To use the keyspace "Students", use the following syntax

Use keyspace_name

Use connects the client session to the specified
keyspace

Command:
USE Students;
Objective: To create a column family or table by the name
“student_info".

Command:

```
CREATE TABLE
Student_info ( RollNo int
PRIMARY KEY,
StudName text,
Dateof]oining
timestamp,
LastExamPercent
double
);
```

Objective: To describe the details of student_info table, we make use of the describe command.
Command:
DESCRIBE TABLE student_info;

Create, Read, Update, and Delete (CRUD) Operations

Objective: To insert data into the column family "student_info".
An insert writes one or more columns to a record in Cassandra table atomically. An insert statement does not return an output. One is not required to place values in all the columns; however it is mandatory to specify all the columns that make up the primary key. The columns that are missing do not occupy any space on disk. Internally insert and update operations are equal. However, insert does not support counters but update does.

Command:

```
BEGIN BATCH
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (l,'Michael Storm','2012-03-29', 69.6)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (2,'Sanjay','2013-02-27', 72.5)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (3,'Deepthi','2014-04-12', 81.7)
```

```
INSERT INTO student_iinfo (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (4,'Inchara','2012-05-11', 73.4)
APPLY BATCH;
```

Objective: To view the data from the table "student_info".
Command:

```
SELECT * FROM student_info;
```

The above select statement retrieves data from the "student_info" table.
Objective: To view only those records where the RollNo column either has a value 1 or 2 or 3.
Act:

```
SELECT *
FROM student_info
WHERE RollNo IN(1,2,3);
```

Note: For the above statement to execute successfully, ensure that the following criteria are satisfied:
1. Either the partition key definition includes the column that is used in the where clause i.e. search criteria.
2. OR the column being used in the where clause, that is, search criteria, has an index defined on it using the CREATE INDEX statement.

Objective: Let us create another index on the "LastExamPercent" column of the "student_info" column family.
Command:

```
CREATE INDEX ON student_info(LastExamPercent);
```

Objective: To specify the number of rows rerurned in the output using limit.
Command:

```
SELECT rollno, hobbies, language, lastexampercent
FROM studencinfo LIMIT 2;
```

Objective: To use column alias for the column "language" in the "student_info" table. We would like the column heading to be "knows language".
Command :

```
SELECT rollno, language AS "knows language"
FROM student_info;
```

Objective: To update the value held in the "StudName" column of the "student_info" column family to "Dileep" for the record where the RollNo column has value = 2.
Note: An update updates one or more column values for a given row to the Cassandra table. It does not return anything.
Command:

```
UPDATE student_info SET StudName = 'Dileep' WHERE RollNo = 2;
```

Objective: Let us try updating the value of a primary key column.
Command:

```
UPDATE student_info SET rollno=6 WHERE rollno=3;
```

Note: It does not allow update to a primary key column.

Objective: To delete the column "LastExamPercent" from the "student_info" table for the record where the RollNo = 2.

Note: Delete statement removes one or more columns from one or more rows of a Cassandra table or removes entire rows if no columns are specified.
Command:
Delete LastExamPercent from student_info WHERE RollNo=2;

Objective: To delete a row (where RollNo = 2) from the table "student_info".
Command:
Delete from student_info WHERE RollNo=2;

Objective: To insert data into the column family "projeccdetails" .
Command:
BEGIN BATCH
INSERT INTO projeccdetails (projecCid,projeccname,stud_name,rating,duration)
VALUES (I,'MS data migration','Dileep',3.5,720)
INSERT INTO projeccdetails (projeccid,projeccname,stud_name,rating,duration)
VALUES (I,'MS Data Warehouse',' Dileep ',3.9,1440)
INSERT INTO projeccdetails (projecCid,projeccname,stud_name,rating,duration)
VALUES (2,'SAPReporting','Sunil',4.2,3000)
INSERT INTO projeccdetails (projecCid,projeccname,stud_name,rating,duration)
VALUES (2,'SAPBI DW,'Sunil',4,9000)
APPLY BATCH;

Objective: To view all the rows of the "projeccdetails" table.
Command:
SELECT * FROM projeccdetails;

Objective: To view rowlrecord from the "projeccdetails" table wherein the projeccid=l.
Command:
SELECT * FROM project_details WHERE projeccid=l;

Collections

Set Collection: A column of type set consists of unordered unique values. However, when the column is queried, it returns the value in sorted order. For example, for text values, it sorts in alphabetical order.

List Collection: When the order of elements matter, one should go for a list collection. For example, when you store the preferences of places to visit by a user, you would like to respect his preferences and retrieve the values in the order in which he has entered rather than in sorted order. A list also allows one to store the same value multiple times.

Map Collection: As the name implies, a map is used to map one thing to another. A map is a pair of typed values. It is used to store timestamp related information. Each element of the map is stored as a Cassandra column. Each element can be individually queried, modified, and deleted.

ALTER TABLE student_info ADD hobbies set <text>;

Objective: To alter the schema of the table "student_info" to add a list column "language".

Command:
ALTER TABLE student_info ADD language list <text>;

Objective: To update the table "studencinfo" to provide the values for "hobbies" for the student with Rollno=1.

Command:
UPDATE studenCinfo
SET hobbies = hobbies + {'Chess, Table Tennis'}
WHERE RollNo=l;

Objective: To update values in the list column, "language" of the table "student_info",
Command:
UPDATE studencinfo
SET language = language + ['Hindi, English']
WHERE RolINo=l;

TIME TO LIVE (TTL)
Data in a column, other than a counter column, can have an optional expiration period called TTL (time to live). The client request may specify a TTL value for the data. The TTL is specified in seconds.
CREATE TABLE userlogin userid int primary key, password text ;

INSERTINTO userlogin (userid, password)
VALUES (l,'infy') USING TTL 30;

INSERT INTO userlogin (userid, password) I --- VALUES (1, 'infy') USING TTL 30;

SELECTTTL (password)
FROM userlogin
WHERE userid=1;

SELECT TTL (password) from userlogin mwhere userid=1;

ALTER COMMANDS

Let us look at a few Alter commands to bring about changes to the structure of the table / column family.

l. Create a table "sample" with columns "sample_id" and "sample_name".

```
CREATE TABLE sample(
sample_id text,
sample_name text,
PRIMARY KEY (sample_id);
```

2. Insert a record into the table "sample".

```
INSERT INTO sample( sample_id, sample_name) VALUES ('S101', 'Big Data');
```

3. View the records of the table "sample".

```
SELECT * FROM sample;
```

Alter Table to Change the Data Type of a Column

1. Alter the schema of the table "sample". Change the data type of the column "sample_id" to integer from text.

```
ALTER TABLE sample
ALTER sample_id TYPE int;
```

2. After the data type of the column "sample_id" is changed from text to integer, try inserting a record as follows and observe the error message:

```
INSERT INTO sample(sample_id, sample_name) VALUES( 'SI02', 'Big Data');
```

3. Try inserting a record as given below into the table "sample".

```
INSERT INTO sample(sample_id, sample_name) VALUES( 102, 'Big Data');
```

4. Alter the data type of the "sample_id" column to varchar from integer.

```
ALTER TABLE sample ALTER sample_id TYPE varchar;
```

5. Check the records after the data type of "sample_id" has been changed to varchar from integer.

```
select * from sample;
```

Alter table to Delete a Column

1. Drop the column "sample_id" from the table "sample".

```
ALTER TABLE sample DROP sample_id;
```

Note: The request to drop the "sample_id" column from the table "sample" does not succeed as it is the primary key column.

2. Drop the column "sample_name" from the table "sample".

```
ALTERTABLE sample
DROP sample_name;
```

Drop a Table

1. Drop the column familytable "sample".

```
DROP columnfamily sample;
```

The above request succeeds. The table/column family no longer exists in the keyspace.

2. Confirm the non-existence of the table "sample" in the keyspace by giving the following command:

```
describe table sample;
Column family 'sample' not found
```

Drop a Database

1. Drop the keyspace "students".

DROP keyspace students;
2. Confirm the non-existence of the keyspace "students" by issuing the following command:
describe keyspace students;
Keyspace 'students' not found.

IMPORT and EXPORT :
Export to CSV:

Objective: Export the contents of the table/column family "elearninglists" present in the "students" databaseto a CSV file (d:\elearninglists.csv).
Act:
Step 1: Check the records of the table "elearninglists" present in the "students" database.
SELECT * FROM elearninglists;

```
cqlsh:students> select * from elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            3 |      1003 |       Seema |    Hadoop Sqoop
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(4 rows)
```

Step 2: Execute the below command at the cqlsh prompt:
COPYe1earninglists (id, course_order, course_id, courseowner, tide) TO 'd:\elearninglisrs.csv';
***Step* 3:** Check the existence of the "elearninglists.csv" file in "0:\". Given below is the content of the "d:\elearninglists.csv" file.

	A	B	C	D	E
1	101	1	1001	Subhashini	NoSQL Cassandra
2	101	2	1002	Seema	NoSQL MongoDB
3	101	3	1003	Seema	Hadoop Sqoop
4	101	4	1004	Subhashini	Hadoop Flume

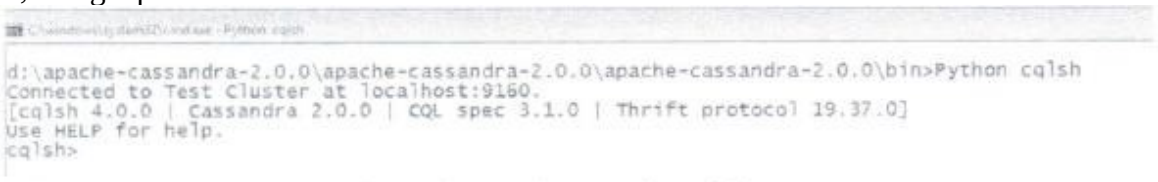
CQL Data Types:

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	Explanation
Int	32 big signed integer
Bigint	64 bit signed long
Double	64-bit IEEE-754 floating point
Float	32-bit IEEE-754 floating point
Boolean	True or False
Blob	Arbitrary bytes, expressed in hexadecimal
Counter	Distributed counter value
Decimal	Variable-precision integer
List	A collection of one or more ordered elements
Map	A JSON style array of elements
Set	A collection of one or more elements
Timestamp	Date plus time

Cassandra Query Language Shell (CQLSH)

Logging into cqlsh**:** The below screenshot depicts the cqlsh command prompt after logging in, using cqlsh succeeds.



The help command assists the user to work with the CQ shell

Keyspaces

A keyspace is a container to hold application data. It is comparable to a relational database. It is used to group column families together. Typically, a cluster has one keyspace per application. Replication is controlled on a per keyspace basis. Therefore, data that has different replication requirement should reside on different keyspaces. When one creates a keyspace, it is required to specify a strategy class. There are two choices available with us. Either we can specify a "Simple Strategy" or a "Network Topology Strategy" class. While using Cassandra for evaluation purpose, go with "Simple Strategy" class and for production usage, work with the “Network Topology Strategy" class.

Objective: To describe all the existing keyspaces
Command:
DESCRIBE KEYSPACES;

The replication factor stated above in the syntax for creating keyspace is related to the number of copies of keyspace data that is housed in a cluster.

Objective: To get more details on the existing keyspaces such as keyspace name, durable writes, strategyclass, strategy options, etc.

Command:

SELECT * FROM system.schema_keyspaces;

Objective: To use the keyspace "Students", use the following syntax

Use keyspace_name

Use connects the client session to the specified keyspace

Command:

USE Students;

Objective: To create a column family or table by the name "student_info".

Command:

CREATE TABLE Student_info (RollNo int PRIMARY KEY,
StudName text, Dateof]oining timestamp, LastExamPercent double
);

Objective: To describe the details of student_info table, we make use of the describe command.

Command:

DESCRIBE TABLE student_info;

Objective: To insert data into the column family "student_info

BEGIN BATCH
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (l,'Michael Storm','2012-03-29', 69.6)
APPLY BATCH;

Objective: To view the data from the table "student_info".

Command:

SELECT * FROM student_info

Objective: To view only those records where the RollNo column either has a value 1 or 2 or 3. Act:

SELECT * FROM student_info WHERE RollNo IN(1,2,3);

Objective: Let us create another index on the "LastExamPercent" column of the "student_info" column family.

Command:

CREATE INDEX ON student_info(LastExamPercent);

Objective: To specify the number of rows rerurned in the output using limit.

Command:

SELECT rollno, hobbies, language, lastexampercent FROM studencinfo LIMIT 2;

Objective: To use column alias for the column "language" in the "student_info" table. We would like the column heading to be "knows language".

Command :

SELECT rollno, language AS "knows language" FROM student_info;

Objective: Let us try updating the value of a primary key column.

Command:

UPDATE student_info SET rollno=6 WHERE rollno=3;

Objective: To delete the column "LastExamPercent" from the "student_info" table for the record where the RollNo = 2.

Command:

Delete LastExamPercent from student_info WHERE RollNo=2;

Objective: To delete a row (where RollNo = 2) from the table "student_info".

Command:

Delete from student_info WHERE RollNo=2;

Objective .To update the value held in the "StudName" column of the "student_info" column family to "Dileep" for the record where the RollNo column has value = 2.

Note: An update updates one or more column values for a given row to the Cassandra table. It does not return anything.

Command: UPDATE student_info SET StudName = 'Dileep' WHERE RollNo = 2;