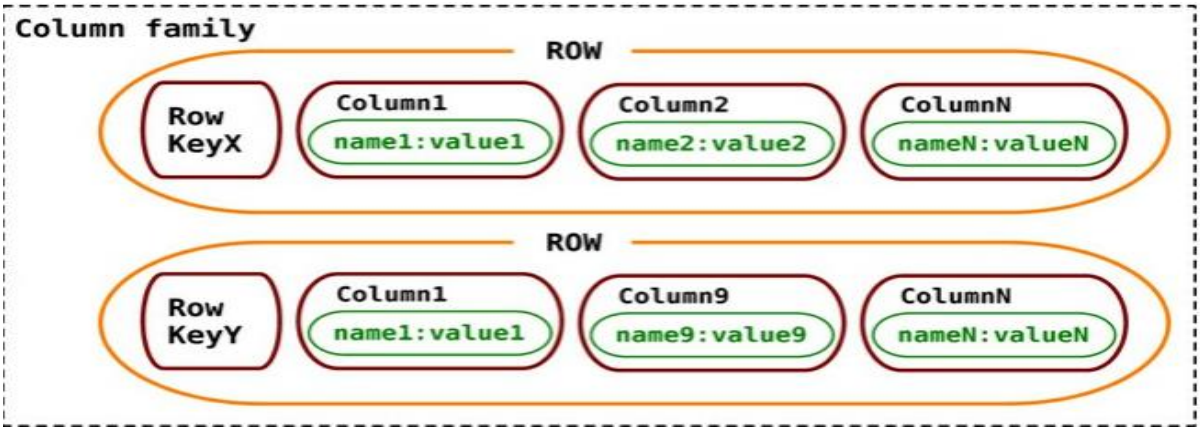
UNIT-2

Column Oriented Databases:

What Is a Column-Family Data Store, Cassandra Database: What is Cassandra, Cassandra Architecture, Cassandra Data types, Cassandra Query Language-CQL, Creating, Altering, Dropping a KeySpace, Cassandra CRUD Operations, Suitable Use Cases, and When Not to Use.

What Is a Column-Family Data Store?

- A column-family data store is a type of NOSQL database that organizes data in terms of column families, which are collections of related columns that are stored together. This type of database is also known as a column-oriented database or a wide-column store.
- In a column-family data store, each column consists of a name, a value, and a timestamp and the columns are grouped together in column families based on their relationships to each other.
- Each column family can contain any number of columns, and the data within each column family can be accessed and manipulated independently of the other column families in the database.
- This type of database is designed to handle large volumes of structured data and is well-suited for use cases such as analytics, data warehousing, and content management systems.
- It is also highly scalable and can be distributed across multiple servers or clusters, allowing for fast and efficient processing of large amounts of data.
- Examples of column-family data stores include Apache Cassandra and Apache HBase.



What is Cassandra?

- Apache Cassandra is a highly scalable, distributed, NoSQL database that is designed to handle large volumes of structured and unstructured data across multiple commodity servers or clusters.
- It was originally developed by Facebook and later released as an open-source project under the Apache Software Foundation.
- Cassandra is a column-family data store that uses a distributed architecture to provide high availability and fault tolerance.
- Cassandra's data model allows it to retrieve information very quickly. Additionally, it can perform thousands of write operations in seconds.

UNIT-2

Key features of Cassandra:

Linear scalability: Cassandra can be easily scaled by adding more nodes to the cluster, making it highly scalable and capable of handling pet bytes of data.

Data storage: Cassandra stores all types of data structures.

Efficiency: this NoSQL database system was designed to run quickly while using minimal resources.

Data transfer: Cassandra supports easy replication, which makes it effortless to transfer data between data centers.

No single point of failure: Cassandra doesn't have a single point of failure (SPOF). It makes it invulnerable to any malfunction.

Easy to use: Cassandra has a simple and intuitive API and is easy to set up and configure, making it accessible to both developers and IT administrators.

Comparing Cassandra NOSQL Database to Relational Databases:

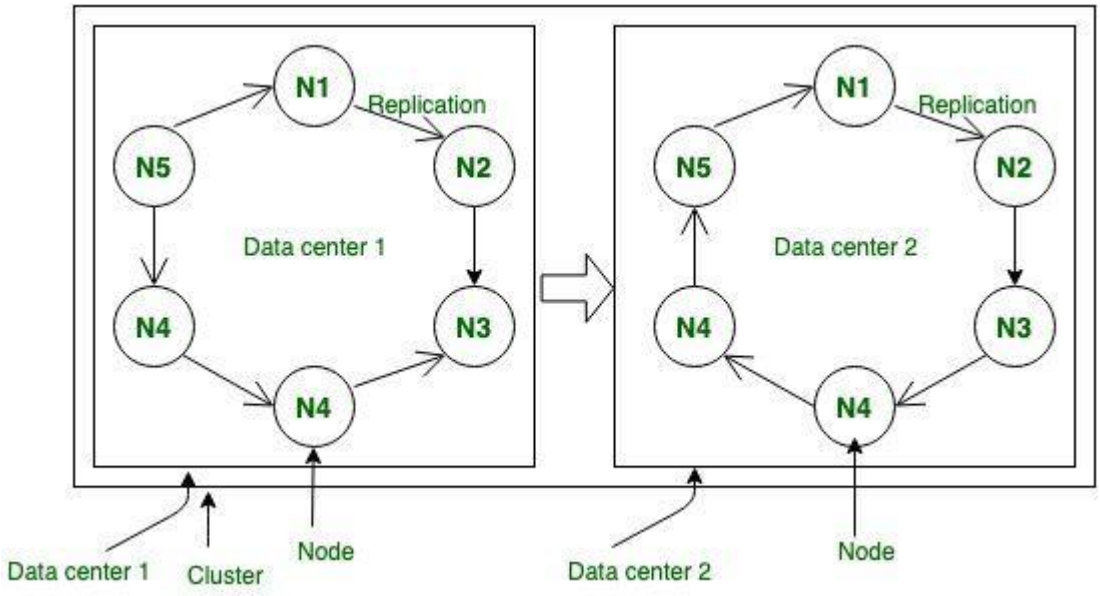
	Cassandra	Relational databases
Data model	Cassandra uses a column-family data model, where data is organized into column families that contain related columns.	Relational databases are based on a tabular data model, where data is stored in tables with fixed columns and rows.
Schema	Cassandra schema is flexible.	A relational database schema is fixed.
Querying	Cassandra databases use elementary CQL query language.	Relational databases use complex and powerful SQL language.
Scalability	Cassandra is scaled horizontally by adding additional servers.	A relational database is scaled vertically by increasing hardware.
Data storage	Structured data is stored in documents.	Data structures are stored in tables with fixed rows and columns.
Consistency	Cassandra, on the other hand, provides eventual consistency, where data is eventually consistent across all nodes, but may not be immediately consistent.	Relational databases provide strong data consistency, where data is guaranteed to be consistent across all nodes.
Data integrity	Cassandra does not have these constraints and relies on application-level logic to ensure data integrity.	Relational databases have strict rules for data integrity and enforce constraints such as unique keys and foreign keys.
Best use cases	Cassandra is used in projects that need to handle big data workloads.	Relational databases are usually meant for general public usage.

RDBMS	Cassandra
database instance	cluster
database	keyspace
table	column family
row	row
column (same for all rows)	column (can be different per row)

UNIT-2

Cassandra Architecture:

- Cassandra is a NOSQL database that is a peer-to-peer distributed database. It runs on a cluster that has homogenous nodes. It is made in such a way that it can handle large volumes of data.
- With handling this data, it should also be capable of providing a high capability. Cassandra provides high throughout when it comes to read and write operations.
- The architecture of the Cassandra cluster does not have any masters, slaves or any specific leaders. By using this way, it makes sure there is no single point of failure.



Components of Cassandra Architecture:

Cluster: A Cassandra cluster is a collection of nodes that work together to store and manage data. Each node in the cluster communicates with other nodes using a peer-to-peer protocol and is responsible for storing and managing a portion of the data.

C = DC1 + DC2 + DC3….

C: Cluster

DC1: Data Center 1 DC2: Data Center 2 DC3: Data Center 3

Node: A node is a single server in the Cassandra cluster that runs the Cassandra software. Each node is assigned a specific role in the cluster, such as coordinator node, seed node, or replica node.

Datacenter: A datacenter is a logical grouping of nodes within a Cassandra cluster that are physically located in the same geographical location. A Cassandra cluster can have multiple datacenters, each with its own set of nodes.

DC – N1 + N2 + N3 ….

DC: Data Center

N1: Node 1 N2: Node 2 N3: Node 3

Replication: Cassandra uses replication to ensure that data is stored and available on multiple nodes in the cluster. Each piece of data is replicated across multiple nodes, with the number of replicas and the placement of those replicas determined by the replication factor and the replication strategy.

UNIT-2

Partitioning: Cassandra uses partitioning to distribute data across the nodes in the cluster. Data is partitioned into smaller pieces called partitions, and each partition is assigned to a specific node in the cluster based on a partitioning algorithm.

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.

Read and write operations: Read and write operations in Cassandra are coordinated by the coordinator node, which is responsible for routing the request to the appropriate replica node based on the partition key. If the requested data is not available on the replica node, it will be retrieved from another replica node or returned as unavailable.

Data Replication in Cassandra:

- Data replication is an essential feature of Cassandra that provides fault tolerance, high availability, and scalability to the database system.
- In Cassandra, data replication refers to the process of making copies of data across multiple nodes in a cluster. Each piece of data is replicated across multiple nodes to ensure that it is available even if some nodes in the cluster fail.
- Cassandra uses a replication factor and a replication strategy to determine how many replicas of each piece of data should be created and where those replicas should be stored.
- The replication factor is the number of copies of each piece of data that are created, and the replication strategy determines how the replicas are distributed across the nodes in the cluster.

In Cassandra, there are two types of replication strategies:

1. SimpleStrategy
2. NetworkTopologyStrategy.

1. SimpleStrategy: SimpleStrategy is the default replication strategy in Cassandra. It is designed for use in single data center deployments where all nodes are located in the same geographical region. With SimpleStrategy, replicas are placed on the next nodes in the ring, starting from the node that is responsible for the primary copy of the data.

```
CREATE KEYSPACE User_data
        WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 2};
```

2. NetworkTopologyStrategy: NetworkTopologyStrategy is designed for use in multi-data center deployments where nodes are located in different geographical regions. With NetworkTopologyStrategy, replicas are placed in different data centers, and the number of replicas in each data center is specified.

```
CREATE KEYSPACE User_data
  WITH replication = {'class': 'NetworkTopologyStrategy', 'DC1' : 2, 'DC2' : 3}
                    AND durable_writes = false;
```

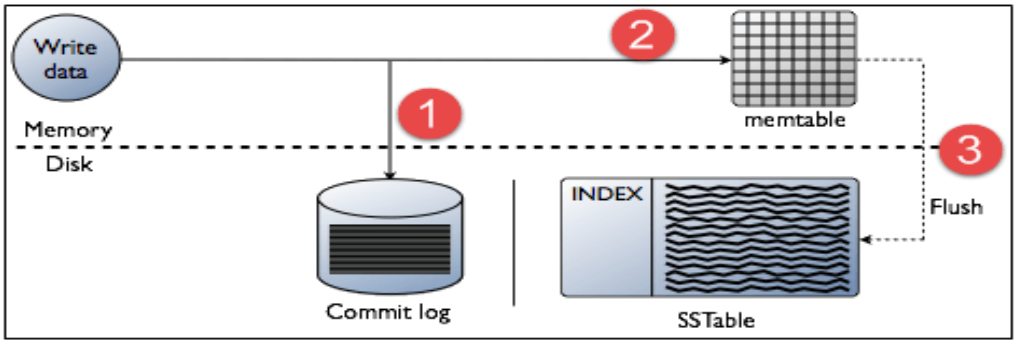
UNIT-2

Write Operation in Cassandra:

- The coordinator sends a write request to replicas. If all the replicas are up, they will receive write request regardless of their consistency level.
- Consistency level determines how many nodes will respond back with the success acknowledgment.
- The node will respond back with the success acknowledgment if data is written successfully to the commit log and memTable.

Here it is explained, how write process occurs in Cassandra,

1. When write request comes to the node, first of all, it logs in the commit log.
2. Then Cassandra writes the data in the mem-table. Data written in the mem-table on each write request also writes in commit log separately. Mem-table is a temporarily stored data in the memory while Commit log logs the transaction records for back up purposes.
3. When mem-table is full, data is flushed to the SSTable data file.



Write Operation in Cassandra

Read Operation in Cassandra:

In Read Operation there are three types of read requests that a coordinator can send to a replica. The node that accepts the write requests called coordinator for that particular operation.

Step-1: Direct Request:

In this operation coordinator node sends the read request to one of the replicas.

Step-2: Digest Request:

In this operation coordinator will contact to replicas specified by the consistency level. For Example: CONSISTENCY TWO; It simply means that Any two nodes in data center will acknowledge.

Step-3: Read Repair Request:

If there is any case in which data is not consistent across the node then background Read Repair Request initiated that makes sure that the most recent data is available across the nodes.

Cassandra Data types:

- Data types can be used to define columns in Cassandra tables. Each column in a Cassandra table has a name, a data type, and an optional value. The data type of a column determines what types of values can be stored in that column.
- These data types are important and are used to reserve the memory to store the data as per the input type of data.

UNIT-2

- Basically, there are 3 types of data type in Cassandra.
  1. **Built-in Data Type**
  2. **User Defined Data Type**
  3. **Collection Data Type**

1. Built-in Data Type:

It is a pre-defined data type in Cassandra. We can directly use by simply giving data type names as per need. There are many Built-in Data types in Cassandra.

- **Boolean:** It is a data type that represents two values, true or false. So, we can use such type of data type where we need just two values.

Syntax: CREATE TABLE table_name (field _name1 Boolean, ...);

- **blob:** It is used for binary large objects such that audio, video or other multimedia and sometimes binary executable code stored as a blob.

Syntax: CREATE TABLE table_name(field_name1 blob, ...);

- **ASCII:** It is used for strings type such that words and sentences. It represents the ASCII value of the character. For example, for 'A' ASCII value is 65.

Syntax: CREATE TABLE table_name(field_name1 ascii, ...);

- **bigint:** It is used for 64 bit signed long integer. Basically it is used for high range of integer which represents a value from -(2^32) to +(2^32) .

Syntax: CREATE TABLE table_name(field_name1 bigint, ...);

- **Double:** It is used for integers which represent a 64 bit floating point. It includes a number with decimal points. for example: 5.838, 10.45 etc.

Syntax: CREATE TABLE table_name(field_name1 double, ...);

- **float:** It is used for numbers which represents a 32-bit floating point. It represents a number values with a decimal point. for example : 6.254, 5.23 etc.

Syntax: CREATE TABLE table_name(field_name1 float, ...);

- **inet:** It is used to represent an IP address that includes both numeric and characters. Basically It Represents an IP address, IPv4 or IPv6. For example, 64.233.160.0 for such type address we can use inet data type.

Syntax: CREATE TABLE table_name(field_name1 inet, ...);

- **int:** It is used to Represents 32-bit signed integers. It represents both positive and negative numbers. The range of int lies from -(2^16) to +(2^16) [only integers]

Syntax: CREATE TABLE table_name(field_name1 int, ...);

- **text:** It is used to store string type which Represents UTF8 encoded string. It encodes all valid points in Uni-code by using 1 bit to four 8 bit types in Cassandra.

Syntax: CREATE TABLE table_name(field_name1 text, ...);

- **varchar:** It is used to store arbitrary string which represents UTF8 encoded string. Example: Ashish, rana, a$#34, A67dgg...

Syntax: CREATE TABLE table_name(field_name1 varchar, ...);

- **timestamp:** It is used to represents a timestamp which is very helpful to store the exact time format value of timestamp. For example, if we want to store 15 dec 1995 at 4:00 AM then timestamp would be: 1995-12-15 04:00 +0530

Syntax: CREATE TABLE table_name(field_name1 timestamp, ...);

2. User Defined Data Type:

User Defined Data(UDT) is to Attach multiple data fields to a column. In Cassandra, UDTs play a vital role which allows group related fields (such that field 1, field 2, etc.) can be of data together and are named and type.

Syntax to define UDT:

```
CREATE TYPE UDT_name
(
 field_name 1 Data_Type 1,
 field_name 2 Data_Type 2,
 field_name 3 Data_Type 3,
 field_name 4 Data_Type 4,
 field_name 5 Data Type 5,
);
```

3. Collection Data Type:

There are 3 types of collection data type in Cassandra.

1. **SET**
2. **LIST**
3. **MAP**

- **SET:** A Set is a collection data where we can store a group of elements such that a user can have multiple Email address then to store such type of data we used set collection data type which returns sorted elements when querying.
- A Set is a collection of unique values of the same data type
- **Syntax**

```
Create table keyspace-Name.Table-Name(
  field 1 data_type 1,
  /*SET keyword is used to define collection data type.*/
  field 1 set<data_type>,
  field 1 data_type 1,
  key_constraint if any
 );
```

- **LIST:** A list is a sorted collection of non-unique values where elements are ordered by their position in the list.
- A collection of values of the same data type.
- **Syntax**

```
CREATE TABLE keyspace1.Employee
 (
  E_id int,
  E_name text,

  /* list syntax to define E_email as list collection data type. */
  E_email list<text>,
  PRIMARY KEY(E_id)
 );
```

- **MAP:** A map is a collection of key-value pairs, where the keys and values can have different data types.

```
CREATE TABLE keyspace1.Activity
  (
   E_id int,
   E_name text,
   task map<timestamp, text>,
   PRIMARY KEY(E_id)
  );
```

UNIT-2

Cassandra Query Language-CQL:

- ✓ Cassandra Query Language (CQL) is a query language used to interact with the Cassandra database. CQL is a SQL-like language that is used to query, update, and manage data stored in Cassandra.
- ✓ Cassandra Database provides Cqlsh command-line interface for CQL.
- ✓ There are two data models associated with Cassandra such as keyspace and column family. CQL is effective and consistent for query processing.
- ✓ Cassandra Query Language (CQL) has commands for both Data Definition Language (DDL) and Data Manipulation Language (DML).

1. Data Definition Language (DDL) commands:

- CREATE KEYSPACE: Creates a new keyspace.
- ALTER KEYSPACE: Modifies an existing keyspace.
- DROP KEYSPACE: Deletes an existing keyspace.
- CREATE TABLE: Creates a new table in a keyspace.
- ALTER TABLE: Modifies an existing table.
- DROP TABLE: Deletes an existing table.
- CREATE INDEX: Creates an index on a column in a table.
- DROP INDEX: Deletes an index.

2. Data Manipulation Language (DML) commands:

- INSERT: Inserts a new row into a table.
- SELECT: Retrieves data from one or more rows in a table.
- UPDATE: Updates one or more columns in an existing row.
- DELETE: Deletes one or more rows from a table.

Keyspace:

- ✓ In Cassandra, a keyspace is a container for related tables, similar to a database in traditional relational database management systems. It represents a namespace that defines the scope or context of data.
- ✓ A keyspace is a top-level namespace in Cassandra. It helps to group tables that are related to each other and determine how data is distributed and replicated across the cluster.
- ✓ Keyspaces provide a way to organize data and manage it at a higher level of abstraction, which makes it easier to manage large-scale distributed databases.
- ✓ In Cassandra, "**Create Keyspace**" command is used to create keyspace.

Creating a KeySpace:

- ✓ In Cassandra, "**Create Keyspace**" command is used to create keyspace.

Syntax: **CREATE KEYSPACE <keyspace_name>**
WITH REPLICATION = {'class': '<replication_strategy>', 'replication_factor' : <num_replicas> }
[AND DURABLE_WRITES = <true_or_false>]

Where,

<**keyspace_name**>: The name of the keyspace to be created.

<replication_strategy>: The replication strategy to be used for the keyspace. The most commonly used strategies are SimpleStrategy and NetworkTopologyStrategy.

<num_replicas>: The number of replicas of the data to be stored in the cluster.

<true_or_false>: Optional parameter to enable/disable durable writes for the keyspace.

Example: creating a keyspace named "my_keyspace" using the SimpleStrategy replication strategy with a replication factor of 2:

```
CREATE KEYSPACE my_keyspace
  WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 2 };
```

Alter Keyspace:

- ✓ The "**ALTER keyspace**" command is used to alter the replication factor, strategy name and durable writes properties in created keyspace in Cassandra.

Syntax:

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

- ❖ Main points while altering Keyspace in Cassandra
- ✓ **Keyspace Name**: Keyspace name cannot be altered in Cassandra.
- ✓ **Strategy Name:** Strategy name can be altered by using a new strategy name.
- ✓ **Replication Factor:** Replication factor can be altered by using a new replication factor.
- ✓ DURABLE_WRITES: DURABLE_WRITES value can be altered by specifying its value true/false. By default, it is true. If set to false, no updates will be written to the commit log and vice versa.

Example: altering the replication strategy for a keyspace named "my_keyspace" to NetworkTopologyStrategy with a replication factor of 3:

```
ALTER KEYSPACE my_keyspace
  WITH replication = {'class': 'NetworkTopologyStrategy', 'dc1': 3};
```

Drop Keyspace:

- ✓ In Cassandra, "**DROP Keyspace**" command is used to drop keyspaces with all the data, column families, user defined types and indexes from Cassandra.

Syntax: **DROP KEYSPACE IF EXISTS <keyspace_name>;**

<keyspace_name>: The name of the keyspace to be dropped.

IF EXISTS: Optional clause to check if the keyspace exists before dropping it.

Example: dropping a keyspace named "my_keyspace":

DROP KEYSPACE IF EXISTS my_keyspace;

UNIT-2

Create Table in CQL:

- ✓ In Cassandra, **CREATE TABLE** command is used to create a table. Here, column family is used to store data just like table in RDBMS.
- ✓ **Syntax:**

```
CREATE TABLE <table_name> (
    <column_name> <data_type> PRIMARY KEY,
    <column_name> <data_type>,
    <column_name> <data_type>,
    ...
) [ WITH <option_name> = <option_value> ... ];
```

Where,

<table_name>: The name of the table to be created.

<column_name>: The name of a column in the table.

<data_type>: The data type of a column.

PRIMARY KEY: Indicates that the column is a primary key for the table.

WITH <option_name> = <option_value> ...: Optional clause for defining additional table properties, such as clustering order or compaction settings.

Example: creating a table named "users" with columns for a user's first name, last name, and email address:

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    first_name TEXT,
    last_name TEXT,
    email TEXT
);
```

Alter Table in CQL:

- ✓ ALTER TABLE command is used to alter the table after creating it. You can use the **ALTER** command to perform two types of operations:
  - Add a column
  - Drop a column

Syntax:

- ✓ ALTER (TABLE | COLUMNFAMILY) <tablename> <instruction>

(Or)

```
ALTER TABLE <table_name>
    ADD <new_column_name> <data_type>
    [ WITH <option_name> = <option_value> ]
    | ALTER <column_name> TYPE <new_data_type>
    | ALTER <column_name> WITH <option_name> = <option_value>
    | RENAME <column_name> TO <new_column_name>
    | DROP <column_name>;
```

<table_name>: The name of the table to be altered.

<new_column_name>: The name of the new column to be added to the table.

<data_type>: The data type of the new column.

<column_name>: The name of the column to be altered, renamed or dropped.

<new_data_type>: The new data type for the existing column.

<option_name>: The name of the option to be changed.

<option_value>: The new value for the option.

1. Adding a Column: **ALTER TABLE table name ADD** new **column** datatype;

   Example: altering a table named "users" by adding a new column "phone_number":

   - ✓ **ALTER TABLE users ADD phone_number TEXT;**

   Example: rename the column "email" to "email_address":

   - ✓ **ALTER TABLE users RENAME email TO email_address;**

1. Dropping a Column: **ALTER table name DROP column name**;

   Example: to drop a column named student_email from a table named student.

   - ✓ **ALTER TABLE student DROP student_email;**

DROP table in CQL:

- ✓ DROP TABLE command is used to drop a table.

Syntax: **DROP TABLE <table_name>;**

<table_name>: The name of the table to be dropped.

Example: dropping a table named "users":

DROP TABLE users;

Truncate Table in CQL:

- ✓ **TRUNCATE** command is used to truncate a table. If you truncate a table, all the rows of the table are deleted permanently.

Syntax: **TRUNCATE** <tablename>

Example: TRUNCATE users;

CREATE INDEX: Creates an index on a column in a table.

Syntax: CREATE INDEX <index_name> ON <table_name> (<column_name>);

<**index_name**>**:** The name of the index to be created.

<**table_name**>**:** The name of the table on which the index will be created.

<**column_name**>**:** The name of the column on which the index will be created.

Rules for creating Index

- ✓ The index cannot be created on primary key as a primary key is already indexed.
- ✓ In Cassandra, Indexes on collections are not supported.
- ✓ Without indexing on the column, Cassandra can't filter that column unless it is a primary key.

Example: **CREATE INDEX email_index ON users (email);**

DROP Index in CQL:

- ✓ DROP INDEX command is used to drop a specified index.

Syntax: DROP INDEX <index_name>;

Example: dropping an index named "email_index":

- ✓ **DROP INDEX email_index;**

Cassandra CRUD Operations:

- ✓ CRUD operations in Cassandra Query Language (CQL) are used for creating, reading, updating, and deleting data in tables.

Create Data:

- ✓ INSERT command is used to insert data into the columns of the table.

Syntax: INSERT INTO <table_name> (<column1>, <column2>, ...) VALUES (<value1>, <value2>, ..);

<table_name>: The name of the table.

<column1>, <column2>, ...: The names of the columns.

<value1>, <value2>, ...: The values to be inserted into the columns.

Example: Insert data in user table.

- ✓ **INSERT INTO users (id, name, age) VALUES (1, 'John', 30);**
- ✓ **INSERT INTO users (id, name, age) VALUES (2, 'James', 25);**

2. **By using JSON we can also insert data.**

Syntax:

```
INSERT INTO table_name JSON '{"field_name1":"field_value1",
                              "field_name2":"field_value2",
                              "field_name3":"field_value3",
...}'
```

Example: INSERT INTO users JSON '("id" :"107", "name" : "Ashish", "age" : "30")' ;

READ Data:

- ✓ SELECT command is used to read data from Cassandra table. You can use this command to read a whole table, a single column, a particular cell etc.

Syntax: SELECT <column1>, <column2>, ... FROM <table_name> WHERE <condition>;

<column1>, <column2>,: The names of the columns to retrieve data from.

<table_name>: The name of the table.

<condition>: The condition to filter the results.

Example: SELECT name, age FROM users WHERE id = 1;

- ✓ This command will retrieve the "name" and "age" columns from the "users" table where the "id" value is 1.

Update Data:

- ✓ UPDATE command is used to update data in a Cassandra table.
- ✓ While updating data in Cassandra table, the following keywords are commonly used:
- ✓ **Where:** The WHERE clause is used to select the row that you want to update.
- ✓ **Set:** The SET clause is used to set the value.

Syntax:

UPDATE <table_name> SET <column1> = <value1>, <column2> = <value2>, ... WHERE <condition>;

<table_name>: The name of the table.

<column1>, <column2>, ...: The names of the columns to be updated.

<value1>, <value2>, ...: The new values for the columns.

<condition>: The condition to filter the rows to be updated.

Example: UPDATE users SET age = 31 WHERE id = 1;

- ✓ This command will update the "age" column of the row with the "id" value of 1 in the "users" table to 31.

DELETE Data:

- ✓ When user want to delete some existing data based on some condition then we can perform 'DELETE' data manipulation command in Cassandra.
- ✓ DELETE command is used to delete data from Cassandra table.

Syntax: DELETE FROM <table_name> WHERE <condition>;

<table_name>: The name of the table.

<condition>: The condition to filter the rows to be deleted.

Example: DELETE FROM users WHERE id = 1;

This command will delete the row with the "id" value of 1 from the "users" table.

Suitable Use Cases:

Big Data: Cassandra can handle massive amounts of data and is highly scalable, making it an excellent choice for big data applications.
Real-Time Analytics: Cassandra can store and process large volumes of data in real-time, making it ideal for real-time analytics use cases.
IoT Applications: With its ability to handle large volumes of data from a variety of sources, Cassandra is well-suited for IoT applications.
Social Media: Cassandra can handle the high volume of data generated by social media platforms and can provide fast, real-time access to this data.
E-commerce: Cassandra is ideal for e-commerce applications, where the ability to store and process large volumes of data in real-time is critical.
Financial Services: Cassandra's high availability and fault-tolerance make it a suitable choice for financial services applications, where data accuracy and availability are crucial.
Healthcare: Cassandra's ability to handle large volumes of data makes it a good choice for healthcare applications, where there is a lot of data generated by electronic health records and other sources.
Gaming: Cassandra's ability to handle large volumes of data in real-time makes it an excellent choice for gaming applications, where data accuracy and speed are essential for a positive user experience.

When Not to Use:

- ✓ It does not assist ACID and properties of the relational database.
- ✓ Because it can manage huge amounts of data and multiple requests.
- ✓ Data in it has been designed on every side of queries and it is not structured which can derive similar information which has been reserved for multiple times.
- ✓ Cassandra can reserve the extensive amount of data that we can accomplish in JVM memory management topic.
- ✓ It does not provide the join or subquery assistance.
- ✓ It does not assist the aggregates.
- ✓ Cassandra has been improved from the beginning of the rapid writing and reading which can receive the short end of the stick so that its promises to be moderate.
- ✓ Ultimately it requires official documentation from Apache so we have to focus for it between third party companies.