- **Cqlsh**

  Cassandra CQLsh stands for Cassandra CQL shell. CQLsh specifies how to use Cassandra commands. After installation, Cassandra provides a prompt Cassandra query language shell (cqlsh). It facilitates users to communicate with it.

  It comes with every Cassandra package and is located in the bin/ directory alongside the cassandra executable. Cqlsh can be used to run CQL commands on a local or remote Cassandra cluster.

  **Basic Cassandra (CQL) constructs include:**

- **Keyspace** — Similar to an RDBMS database, a keyspace is a container for application data that must have a name and a set of associated attributes. Cassandra keyspace is a SQL database.
- **Column Families/Tables** — A keyspace consists of a number of Column Families/Tables. A Cassandra column family is a SQL table.
- **Primary Key / Tables** — A Primary Key consists of a Row/Partition Key and a Cluster Key, and functions to enable users to uniquely identify internal rows of data. A Row/Partition Key determines the node on which data is stored. A Cluster Key determines the sort order of data within a particular row.

  characteristics that are associated with SQL but are not available in Cassandra (CQL) include:

  - **Arbitrary WHERE clauses** – A predicate can only contain columns specified in the Primary Key.
  - **Arbitrary ORDER BY clauses** – Order by can only be applied to a cluster column.
  - **GROUP BY** – Identical data cannot be grouped in CQL.
  - **Cassandra CQL JOIN** — Data across column families cannot be joined in CQL without third party tools.
  - **Complex transactions** — Where SQL supports complex transactions, CQL supports simple transactions.

**CQL Data Types**

Apache Cassandra CQL supports a rich set of data types, including collection types, custom types, native types, tuple types, and user-defined types:

- **collection type**s — intended to store and/or denormalize small amounts of data; supports Maps, Sets, and Lists collections
- **custom types** — a string that can be loaded by Cassandra and contains the name of Java class that extends the server side AbstractType class
- **native types** — includes Bigint, Blob, Boolean, Counter, Date, Decimal, Double, Duration, Float, Inet, Int, Smallint, Text, Time, Timestamp, Timeuuid, Tinyint, Uuid, Varchar, Varint
- **tuple types** — an ordered set of values that can functionally be considered anonymous UDT with anonymous fields, and can only be updated as a whole
- **user-defined types** — a named set consisting of named and typed fields that can be created, modified, and removed using the create_type_statement

**CQLsh provides a lot of options:**

| Options | Usage |
|---|---|
| **help** | This command is used to show help topics about the options of CQLsh commands. |
| **version** | it is used to see the version of the CQLsh you are using. |
| **color** | it is used for colored output. |
| **debug** | It shows additional debugging information. |
| **execute** | It is used to direct the shell to accept and execute a CQL command. |
| **file= "file name"** | By using this option, cassandra executes the command in the given file and exits. |
| **no-color** | It directs cassandra not to use colored output. |
| **u "username"** | Using this option, you can authenticate a user. The default user name is: cassandra. |
| **p "password"** | Using this option, you can authenticate a user with a password. The default password is: cassandra. |

- **Shell Commands/ cqlsh commands**
  - cqlsh

    Start the CQL interactive terminal.
  - Creating and using the cqlshrc file

The cqlshrc file passes default configuration information to cqlsh.

- CAPTURE

  Captures command output and appends it to a file.

- CONSISTENCY

  Shows the current consistency level, or given a level, sets it.

- COPY

  Imports and exports CSV (comma-separated values) data to and from Cassandra.

- DESCRIBE

  Provides information about the connected Apache Cassandra cluster, or about the data objects stored in the cluster.

- EXPAND

  Formats the output of a query vertically.

- EXIT

  **Terminates cqlsh.**

- **PAGING**

  Enables or disables query paging.

- **SHOW**

  Shows the Apache Cassandra version, host, or tracing information for the current cqlsh client session.

- **SOURCE**

  Executes a file containing CQL statements.

- **TRACING**

  Enables or disables request tracing.

**What is Keyspace?**

A keyspace is an object that is used to hold column families, user defined types. A keyspace is like RDBMS database which contains column families, indexes, user defined types, data center awareness, strategy used in keyspace, replication factor, etc.

**Syntax: CREATE KEYSPACE <identifier> WITH <properties>**

**Example:** CREATE KEYSPACE cdac

WITH replication = {'course':'SimpleStrategy', 'replication_factor' : 3};

**To check whether the keyspace is created or not**:

use the "DESCRIBE" command. By using this command, you can see all the keyspaces that are created.

**Syntax**: describe name_keyspace

**Syntax**: describe keyspaces;  // to check all the created keyspaces

**Example:** Describe cdac;

**Using a Keyspace**

To use the created keyspace, you have to use the USE command:

**Syntax:** Use keyspace_name;

**Example:**Use cdac;

- **Creating table in Cassandra:**

```
CREATE TABLE student (
  userid text PRIMARY KEY,
  first_name text,
  last_name text,
  emails set<text>,
  top_scores list<int>,
  todo map<timestamp, text>
);


CREATE TABLE student2 (
   student_id int PRIMARY KEY,
   student_name text,
   student_city text,
   student_fees varint,
   student_phone varint
   );
```

- **Insert data into table:**

```
INSERT INTO student (userid, first_name, last_name, emails)
  VALUES('s01', 'Tarun', 'Kumar', {'f@baggins.com', 'baggins@gmail.com'});
```

- **We can input individual column data:**

  INSERT INTO student (userid, todo ) values ('E23',{'2014-10-2 12:10' : 'die' });

- **To Fetch data from the table:**

  SELECT * FROM student;

## Cassandra Alter Table

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**

  While adding column, you have to aware that the column name is not conflicting with the existing column names and that the table is not defined with compact storage option.

  **Syntax: ALTER TABLE table name**

  **ADD new column datatype;**

  **Example:**

  ALTER TABLE student

  ADD student_email text;

- **Drop a column**: drop an existing column from a table by using ALTER command.

  **Syntax:**

  ALTER table name

  DROP column name;

  **Example:**

  ALTER TABLE student

  DROP student_email;

  **If you want to drop the multiple columns, separate the columns name by ",".**

  **Example:**

  ALTER TABLE student

  DROP (student_fees, student_phone);

**Cassandra DROP table**: DROP TABLE command is used to drop a table.

**Syntax:** DROP TABLE <tablename>

**Example:** DROP TABLE student;

**Cassandra Truncate Table**

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 student;**

**Cassandra Create Index**

CREATE INDEX command is used to create an index on the column specified by the user.

**Syntax:** CREATE INDEX <identifier> ON <tablename>

- 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 name ON student (student_name);

**verify that the index is created or not, by using the creating index query once again. It will show a message that index is already created.**

**DROP Index :** DROP INDEX command is used to drop a specified index.

**Syntax:** DROP INDEX IF EXISTS keyspace_name . **field_name**

- If the index does not exist, it will return an error unless you use IF EXISTS which returns no operation.
- During index creation, you have to specify keyspace name with the index name otherwise index will be dropped from the current keyspace.

**Example:** drop index if exists cdac.student_name;

**Cassandra Batch**

BATCH is used to execute multiple modification statements (insert, update, delete) simultaneously. It is very useful when you have to update some column as well as delete some of the existing.

**Syntax:**

**BEGIN BATCH**

**<insert-stmt>/ <update-stmt>/ <delete-stmt>**

**APPLY BATCH**

Example:

**BEGIN BATCH**

INSERT INTO student2 (student_id,student_name, student_city, student_fees, student_phone) values( 4,'Varun','Delhi',50000,9848022331);

UPDATE student2 SET student_fees = 30000 WHERE student_id =4;

DELETE student_city FROM student2 WHERE student_id = 4;

**APPLY BATCH;**

student_id int PRIMARY KEY,

... student_name text,

... student_city text,

... student_fees varint,

... student_phone varint

student_id int PRIMARY KEY,

... student_name text,

... student_city text,

... student_fees varint,

... student_phone varint

**CRUD Operation**

**o Create : CREATE TABLE command is used to create a table.**

**Syntax:**

CREATE TABLE tablename(

column1 name datatype PRIMARYKEY,

column2 name data type,

column3 name data type. )

**use of primary keys:**

- Single primary key: Use the following syntax for single primary key.

  Primary key (ColumnName)

**o Read : by using select query**

**UPDATE:** UPDATE command is used to update data in a Cassandra table. If you see no result after updating the data, it means data is successfully updated otherwise an error will be

returned. 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.
- **Must:** It is used to include all the columns composing the primary key.

  **Syntax:**

  UPDATE <tablename>

  SET <column name> = <new value>

  <column name> = <value>....

  WHERE <condition>

  **Example:**

  UPDATE student2 SET student_fees=10000,student_name='Rahul'

  WHERE student_id=2;

**Delete:**

DELETE command is used to delete data from Cassandra table. You can delete the complete table or a selected row by using this command.

**Syntax:** DELETE FROM <identifier> WHERE <condition>;

- **Delete an entire row:** To delete the entire row of the student_id "3", use the following command:

**Example:** DELETE FROM student WHERE student_id=4;

- **Delete a specific column name**

  DELETE student_fees  FROM student WHERE student_id=4;

**Cassandra Collections**

Cassandra collections are used to handle tasks. You can store multiple elements in collection. There are three types of collection supported by Cassandra:

- Set : A set collection stores group of elements that returns sorted elements when querying.

  **Syntax:**

  Create table table_name

  (

  id int,

  Name text,

  Email set<text>,

```
Primary key(id)
);
```

- **List :** The list collection is used when the order of elements matters.
- **Map**

  The map collection is used to store key value pairs. It maps one thing to another.

```
CREATE TABLE student2 (
  userid text PRIMARY KEY,
  first_name text,
  last_name text,
  emails set<text>,
  top_scores list<int>,
  todo map<timestamp, text>
);
```