**CASSANDRA DEMO CODES**

**Keyspace**

Keyspace is Cassandra’s version of a database

**Create Keyspace:**

**CREATE** **KEYSPACE** Demo **WITH** replication **=** {'class': 'SimpleStrategy', 'replication\_factor' : 3};

The replication property is mandatory and must at least contains the 'class' sub-option which defines the [replication strategy](https://cassandra.apache.org/doc/latest/cql/ddl.html#replication-strategy) class to use.

Strategy type used in our example: Simple Strategy

* A simple strategy that defines a replication factor for data to be spread across the entire cluster.

| **sub-option** | **type** | **since** | **description** |
| --- | --- | --- | --- |
| 'replication\_factor' | int | all | The number of replicas to store per range |

**Use Keyspace:**

The USE statement allows to change the current keyspace. Similar to “use databse” in SQL.

**use\_statement** ::= USE [Demo](https://cassandra.apache.org/doc/latest/cql/ddl.html#grammar-token-keyspace-name)

If the above code doesn’t work use this: use Demo;

**Alter Keyspace:**

An ALTER KEYSPACE statement allows to modify the options of a keyspace:

**ALTER** **KEYSPACE** Demo **WITH** replication **=** {'class': 'SimpleStrategy', 'replication\_factor' : 4};

**Drop Keyspace:**

Dropping a keyspace can be done using the DROP KEYSPACE statement:

**DON’T ACTUALLY DROP THE KEYSPACE IN THE DEMO!! Need it to continue making edits to tables. -suggested by Bradley**

**(May be you can show how to delete and create a new one for futher parts)-by Reha**

**DROP** **KEYSPACE** Demo;

Dropping a keyspace results in the immediate, irreversible removal of that keyspace, including all the tables, UTD and functions in it, and all the data contained in those tables.

**Table**

**Create Table:** Creating a new table uses the CREATE TABLE statement:

**CREATE** **TABLE** Users (

userid int **PRIMARY** **KEY**,

name text,

occupation text

) **WITH** comment**=**'User information.';

**CREATE** **TABLE** test (

a int,

b int,

c int,

d int,

**PRIMARY** **KEY** ((a, b), c, d)

);

A CQL table has a name and is composed of a set of rows. Creating a table amounts to defining which [columns](https://cassandra.apache.org/doc/latest/cql/ddl.html#column-definition) the rows will be composed, which of those columns compose the [primary key](https://cassandra.apache.org/doc/latest/cql/ddl.html#primary-key), as well as optional [options](https://cassandra.apache.org/doc/latest/cql/ddl.html#create-table-options) for the table.

**Alter Table:**

**ALTER** **TABLE** Users **ADD** age int; (This adds column age to table)

**ALTER** **TABLE** Users

**WITH** comment **=** ‘All users information.’; (This alters the comment of the table)

The ALTER TABLE statement can:

* Add new column(s) to the table (through the ADD instruction).
* Remove column(s) from the table.
* Change some of the table options (through the WITH instruction).

**Drop Table:** Dropping a table uses the DROP TABLE statement:

**drop\_table\_statement** ::= DROP TABLE [**test**](https://cassandra.apache.org/doc/latest/cql/ddl.html#grammar-token-table-name)

If the code above doesn’t work use this: drop table test;

Dropping a table results in the immediate, irreversible removal of the table, including all data it contains.

DESCRIBE TABLES and keyspaces:

Output the names of all tables in the current keyspace, or in all keyspaces if there is no current keyspace.

DESCRIBE TABLES;

Keyspace:

Output the names of all keyspaces.

DESCRIBE KEYSPACES;

**DATA MANIPULATION:**

**INSERT**

Inserting data for a row is done using an INSERT statement:

The INSERT statement writes one or more columns for a given row in a table. Note that since a row is identified by its PRIMARY KEY, at least the columns composing it must be specified. The list of columns to insert to must be supplied when using the VALUES syntax. When using the JSON syntax, they are optional. See the section on [JSON support](https://cassandra.apache.org/doc/latest/cql/json.html#cql-json) for more detail.

**INSERT** **INTO** Users (userid, name, occupation, age)

**VALUES** (1, 'John', 'Doctor', 30) **USING** **TTL** 86400;

**INSERT** **INTO** Users (userid, name, occupation, age)

**VALUES** (3, 'Mary', 'Teacher', 35) **USING** **TTL** 86400;

**INSERT** **INTO** Users **JSON** '{"userid": 2,

"name": "Daniel",

"occupation": “Engineer”,

“age”:23}';

**SELECT**

The SELECT statements reads one or more columns for one or more rows in a table. It returns a result-set of the rows matching the request, where each row contains the values for the selection corresponding to the query. Additionally, [functions](https://cassandra.apache.org/doc/latest/cql/functions.html#cql-functions) including [aggregation](https://cassandra.apache.org/doc/latest/cql/functions.html#aggregate-functions) ones can be applied to the result.

**SELECT** name, occupation **FROM** **users** **WHERE** userid **IN** (1);

**SELECT** **JSON** name, occupation **FROM** **users** **WHERE** userid **=** 2;

**SELECT** name **AS** user\_name, occupation **AS** user\_occupation **FROM** **users**;

**SELECT** **COUNT** (**\***) **AS** user\_count **FROM** **users**;

**UPDATE:**

Updating a row is done using an UPDATE statement:

The UPDATE statement writes one or more columns for a given row in a table. The [**where\_clause**](https://cassandra.apache.org/doc/latest/cql/dml.html#grammar-token-where-clause) is used to select the row to update and must include all columns composing the PRIMARY KEY. Non primary key columns are then set using the SET keyword.

**UPDATE** Users **USING** **TTL** 400

**SET** name **=** 'Johnsnow',

age **=** 32

**WHERE** userid **=** 1;

**DELETE:**

Deleting rows or parts of rows uses the DELETE statement:

The DELETE statement deletes columns and rows. If column names are provided directly after the DELETE keyword, only those columns are deleted from the row indicated by the WHERE clause. Otherwise, whole rows are removed.

The WHERE clause specifies which rows are to be deleted. Multiple rows may be deleted with one statement by using an IN operator. A range of rows may be deleted using an inequality operator (such as >=).

**DELETE** age **FROM** **Users**

**WHERE** userid **IN** (3);

**BATCH:**

Multiple INSERT, UPDATE and DELETE can be executed in a single statement by grouping them through a BATCH statement:

The BATCH statement group multiple modification statements (insertions/updates and deletions) into a single statement. It serves several purposes:

* It saves network round-trips between the client and the server (and sometimes between the server coordinator and the replicas) when batching multiple updates.
* All updates in a BATCH belonging to a given partition key are performed in isolation.
* By default, all operations in the batch are performed as *logged*, to ensure all mutations eventually complete (or none will). See the notes on [UNLOGGED batches](https://cassandra.apache.org/doc/latest/cql/dml.html#unlogged-batches) for more details.

**BEGIN** **BATCH**

**INSERT** **INTO** **Users** (userid, **occupation**, name) **VALUES** (4, 'Plumber', 'Jack');

**UPDATE** **Users** **SET** **age=** 32 **WHERE** userid **=** 3;

**INSERT** **INTO** **Users** (userid, **name**) **VALUES** (5,’Rose');

**DELETE** occupation **FROM** **Users** **WHERE** userid **=** 3;

**APPLY** **BATCH**;

**SECONDARY INDEXES:**

The CREATE INDEX statement is used to create a new (automatic) secondary index for a given (existing) column in a given table. A name for the index itself can be specified before the ON keyword, if desired. If data already exists for the column, it will be indexed asynchronously. After the index is created, new data for the column is indexed automatically at insertion time

**CREATE** **INDEX** userIndex **ON** Users (**occupation**);

**DROP INDEX:**

Dropping a secondary index uses the DROP INDEX statement:

DROP INDEX userIndex;

## **Aggregate functions**

#### **Count**

The count function can be used to count the rows returned by a query. Example:

**SELECT** **COUNT** (**\***) **FROM** Users;

It also can be used to count the non null value of a given column:

**SELECT** **COUNT** (age) **FROM** Users;

#### **Max and Min**

The max and min functions can be used to compute the maximum and the minimum value returned by a query for a given column. For instance:

**SELECT** MIN (age), MAX (age) **FROM** Users;

#### **Sum**

The sum function can be used to sum up all the values returned by a query for a given column. For instance:

**SELECT** SUM (age) **FROM** Users;

#### **Avg**

The avg function can be used to compute the average of all the values returned by a query for a given column. For instance:

**SELECT** AVG (age) **FROM** Users;

CREATE MATERIALIZED VIEW

Materialized Views are essentially standard CQL tables that are maintained automatically by the Cassandra server – as opposed to needing to manually write to many denormalized tables containing the same data

CREATE MATERIALIZED VIEW usersbyage AS

SELECT **\*** FROM users

WHERE age IS NOT NULL AND name IS NOT NULL

PRIMARY KEY (userid,age)

WITH comment**=**'All users information.';

To view the created view:

Select \* from usersbyage;

## ALTER MATERIALIZED VIEW

ALTER MATERIALIZED VIEW usersbyage WITH comment=’All users information’ AND bloom\_filter\_fp\_chance = 0.02;

## DROP MATERIALIZED VIEW[¶](https://cassandra.apache.org/doc/latest/cql/mvs.html#drop-materialized-view)

Example: DROP MATERIALIZED VIEW usersbyage;