7280 Special Topic in Database Xinglu Jiang
Github repo: https://github.com/mintso/Cassandra || Video Link: https://youtu.be/fK6NSGoEgrI

Research On Cassandra Database

I. Introduction

NoSQL database is often compared with SQL database. Significantly different from the latter, NoSQL database does not store data in relational tables, rather, it stores data in documents and provide more flexible schemes to build applications. It is widely recognized as being better in performance when it comes to large data scale and concurrent users. Its flexibility, scalability, high-performance, as well as ability to handle semi- or unstructured data make it favored by more and more applications.

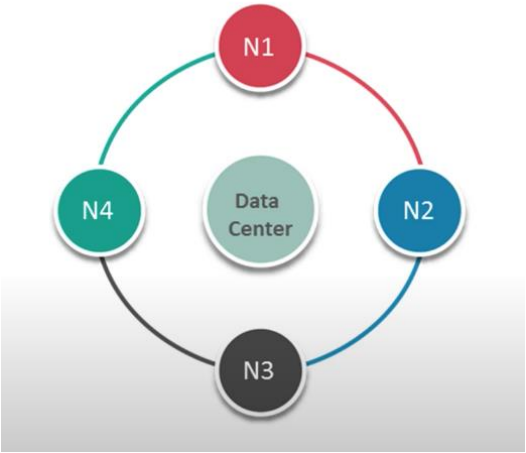
Following this light, I take Cassandra database as a cutting point and look through its features. Apache Cassandra is one of the most widely used NoSQL database. It was initially developed with intent to optimize Facebook inbox search feature and was first released in 2006. Written in Java, it supports cross-platform operating system. Cassandra is open source and distributed database; user could download and use it from its Github repository or official website. It is trusted by its linear scalability, fault-tolerance, performant, security, low latency, fast scalable, reliability, etc. Especially, according to CAP theorem, in a distributed system, it is common to see network failures and partition tolerant is crucial. Cassandra enables users to balance the level of consistency and availability. These result in it's been used by thousands of companies who have to manage large active datasets.

In this article, we will inspect on Cassandra around these following topics: architecture, data model, and applications.

II. Architecture

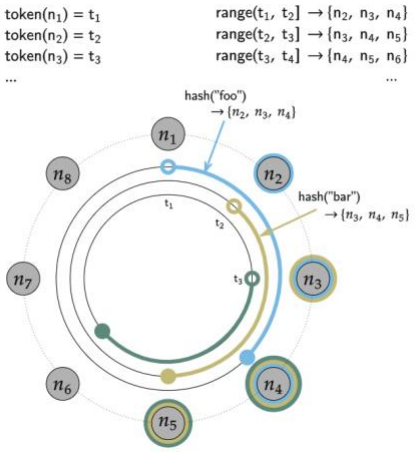
Cassandra architecture endows a database ability to scales and preforms with continuous availability. It is designed to be a best-in-class combination of meeting emerging largescale in data footprint, query volume and storage requirements.

Like demonstrated in the picture below, it has a masterless "ring" distributed architecture which is easy to scale up or down and manage. Benefiting from its built-for-scale architecture, it can handle large amount of data and allow allows large number of concurrent users/operations.



(Source: https://www.youtube.com/watch?v=ZM0wJ1suCUE)

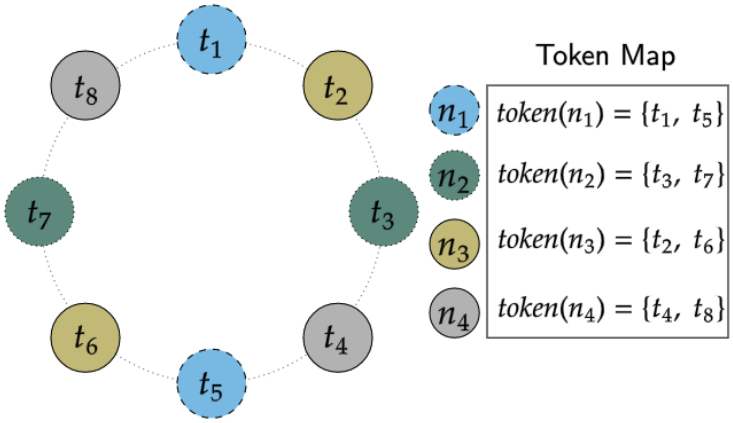
To go further, I will introduce Cassandra's dynamo. Amazon's Dynamo distributed storage key-value system contribute to lots of techniques that Cassandra depends on. In Cassandra's dynamo system, each node has request coordination over a partitioned dataset, ring membership and failure detection, and a local persistence (storage) engine[1]. Cassandra's storage engine is based on a Log Structured Merge Tree (LSM). The reason that Cassandra has horizontal scalability is due to its consistent hashing using a token ring in terms of dataset partition. Its partition is replicated to multiple physical nodes. consistent hashing means that Cassandra maps each node to one or more tokens on a continuous hash ring (as showed below). Like the Chord algorithm, the ownership is defined by hashing a key onto the ring and "walling" the ring in one direction.



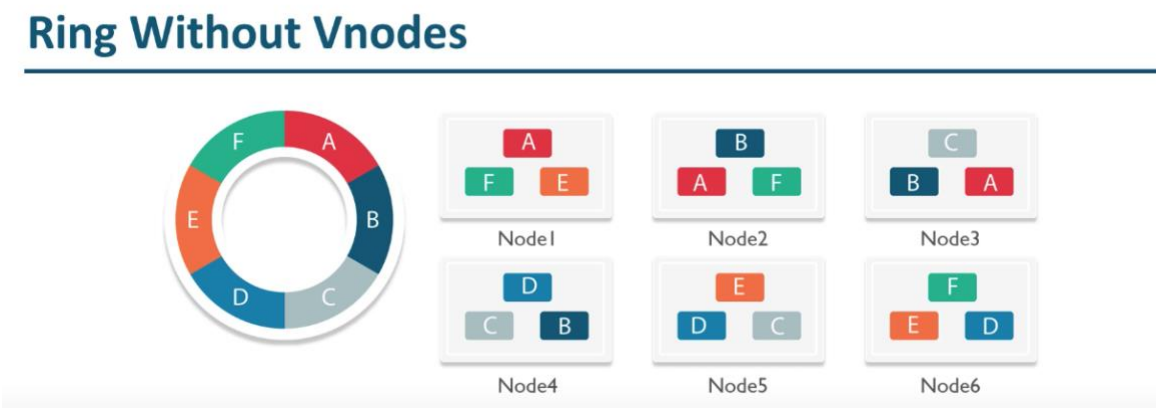
(Source: https://cassandra.apache.org/doc/latest/cassandra/architecture/dynamo.html)

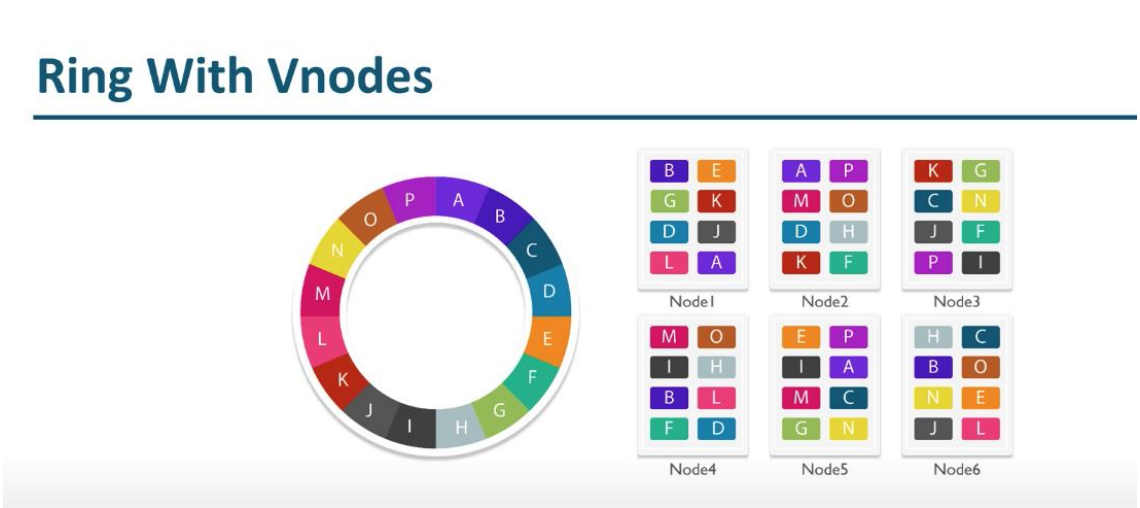
[1] Refer to Cassandra Documentation:
https://cassandra.apache.org/doc/latest/cassandra/architecture/dynamo.html

To avoid token imbalance, the usage of "virtual nodes" is proposed upon the multiple tokens per physical node. Virtual nodes enable physical node to take multiple positions in the ring. Like demonstrated in the below graph, 4 physical nodes can actually represent 8 node cluster.



(Source: https://cassandra.apache.org/doc/latest/cassandra/architecture/dynamo.html)

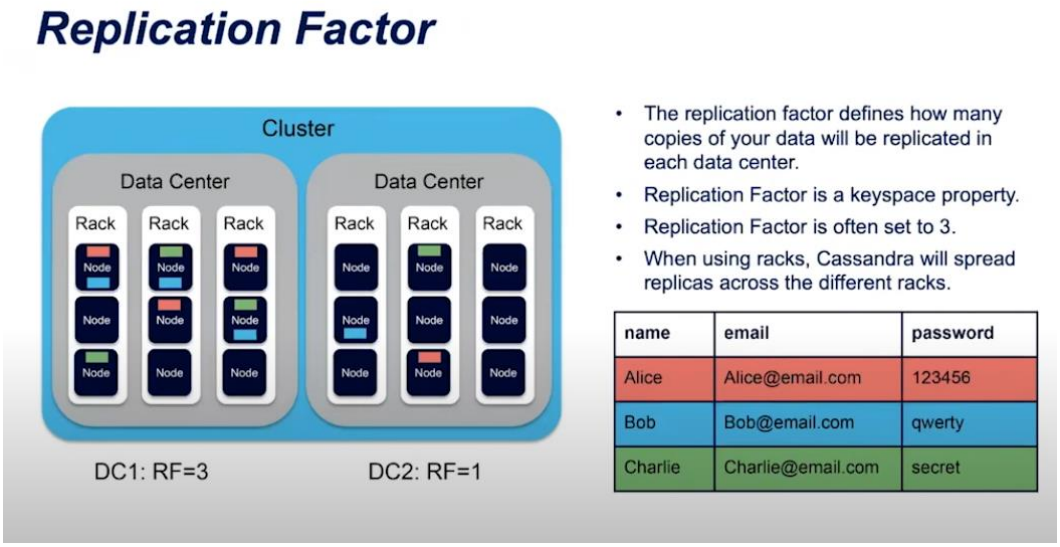


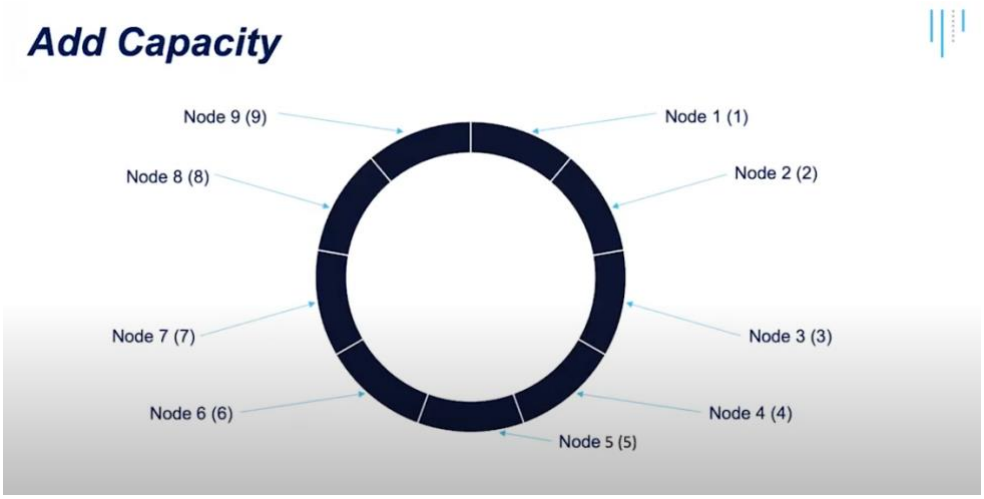


(Source: https://www.youtube.com/watch?v=ZM0wJ1suCUE)

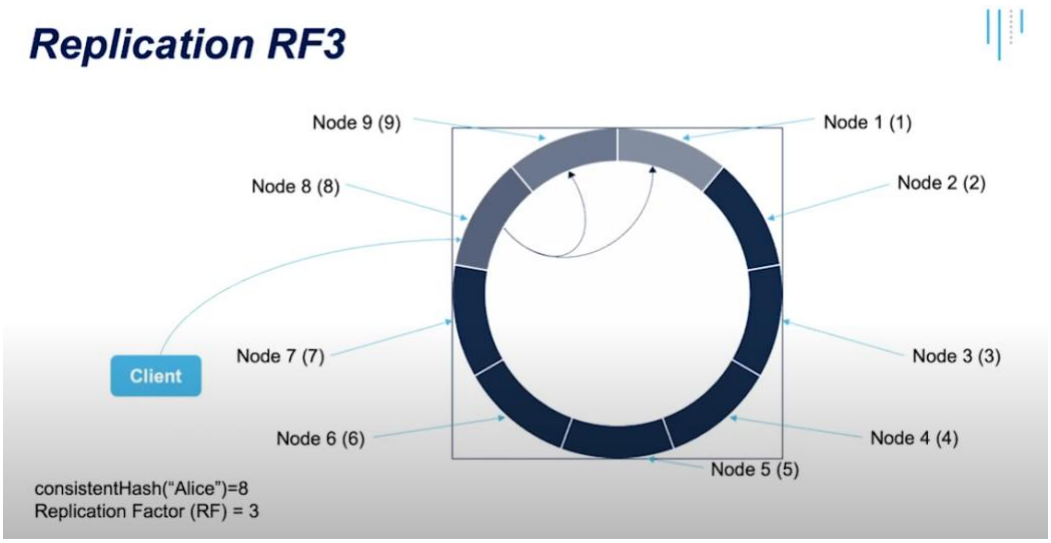
With virtual nodes, token ranges are distributed. Its calculation and assignment are automated. Rebalancing a cluster is no longer needed when nodes are added or removing. To summarize, virtual nodes improve the use of heterogeneous machines in a cluster[2].

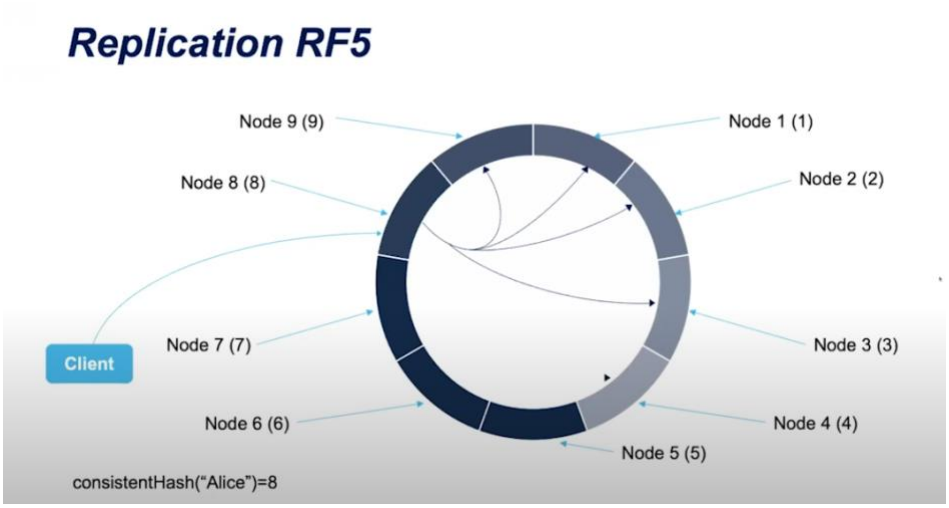
In order to achieve high availability and durability, Cassandra adopts multi-master replication, which means it replicates every partition of data to many nodes. Cassandra has two consistency levers: replication factor (RF) -- which is determined by schema – and consistency level (CL) which is chosen by the user at query time. RF means number of copies of data in each data center. Usually, Cassandra set RF at 3. Cassandra uses mutation timestamp versioning to keep data version. With tunable consistency, user could choose consistency levels.





[2] Refer to: https://www.youtube.com/watch?v=ZM0wJ1suCUE

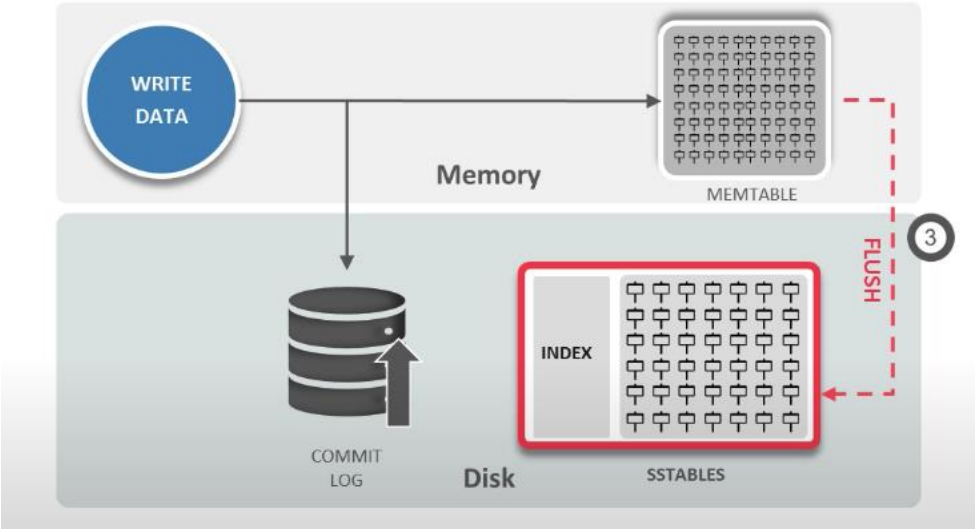




(Source: https://www.youtube.com/watch?v=0mntwB6nbc0)

Consistency Level (CL) implies that the number of replicas that must serve/ack a read or write request successfully. Without knowing the replication factor, Cassandra user can choose from the following consistency levels in reads and writes: ONE, TWO, THREE, QUORUM, ALL, LOCAL_QUORUM, EACH_QUARUM, LOCAL_ONE, and ANY. Write operations are sent to all replicas though.

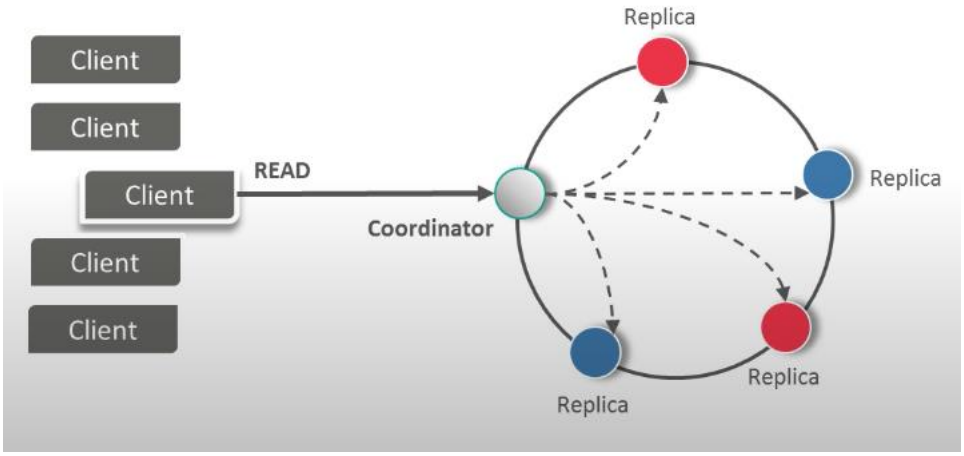
To elaborate on write request, we can refer to the below graph:

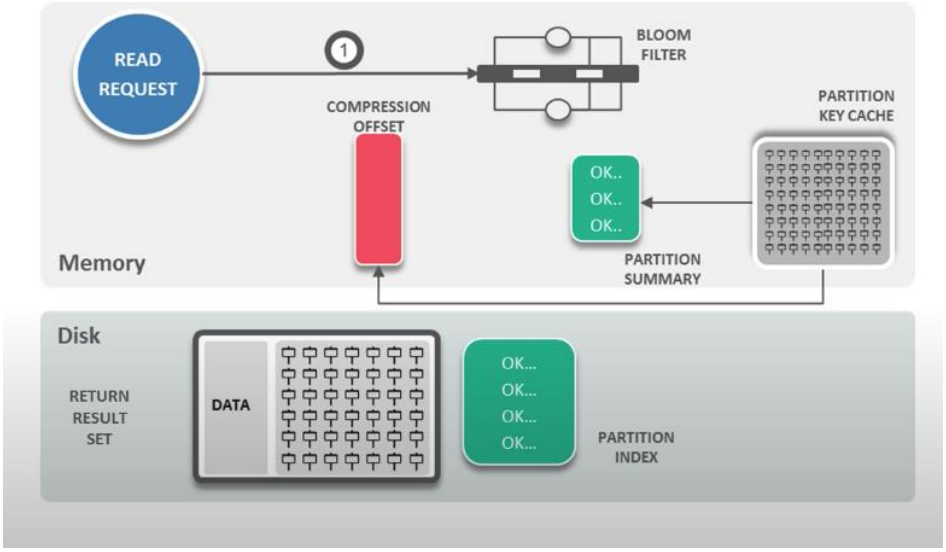


(Source: https://www.youtube.com/watch?v=ZM0wJ1suCUE)

When a node receives a write request, commit log firstly logs it. Mem-table will capture and store the data. If the mem-table is full, data will be flushed to the SSTables. Since Cassandra is masterless, a write request could arrive at any node based on user configuration. The first node that receive to write will act as function node, but it may not necessarily save the data eventually. It is decided by token range for node that the data is saved and replicated at last.

As for reading data, if the user connects to a node without the queried data, this node will function as coordinator node. The coordinator node will send request to all replica and send back to client with most recent data received from all replicas. Multiple processes could be involved as well as various memory caches so as to achieve faster read response time.





(Source: https://www.youtube.com/watch?v=ZM0wJ1suCUE)

As for language, Cassandra uses Cassandra Query Language (CQL), which is a SQL-like language, to organize data within a cluster of Cassandra nodes. Cassandra is consisted of:

1. Keyspace: defines the replication of dataset per datacenter.
2. Table: defines the typed schema for a collection of partitions.
3. Partition: defines the primary key that Cassandra must use to identify and locate the node that a row is stored.
4. Row: a collection of columns with same primary key.
5. Column: a single typed datum of a row.

Contain relations are like below:

Column → Row → Partition → Table → Keyspace

Due to performance concerns, Cassandra does not support operations that require cross partition coordination, such as distributed joins, foreign keys or referential integrity.

Based on Cassandra gossip protocol, nodes communicate state information about themselves as well as other nodes. Cassandra also has incremental scale-out ability based on data size and request rates.

As partially discussed in read and write, I will introduce more about Cassandra Storage Engine. Data will be firstly written in commit log, then to9 a memtable. This ensures unexpected shutdown won't have any influence on data durability. Memtables are in-memory structures. Each table has one active memtable. Memtables will be flushed onto disk and become SSTables, which are immutable. Therefore, Cassandra saves SSTables on disk.

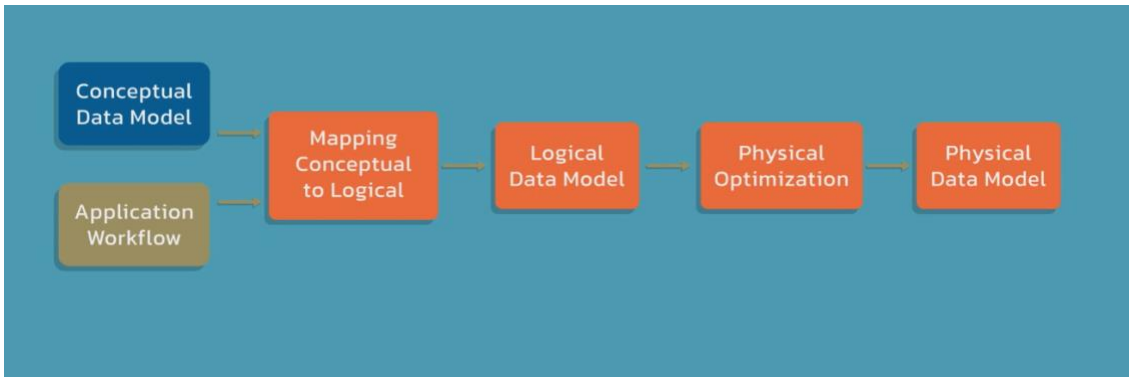
Also briefly mentioned above, CAP theorem implies that no database could have all the three guarantees simultaneously, which are consistency, availability and partition tolerance. Nevertheless, Cassandra guarantees high scalability, high availability, durability, eventual consistency, lightweight transaction with linearizable consistency, batched writes and secondary indexes[3].

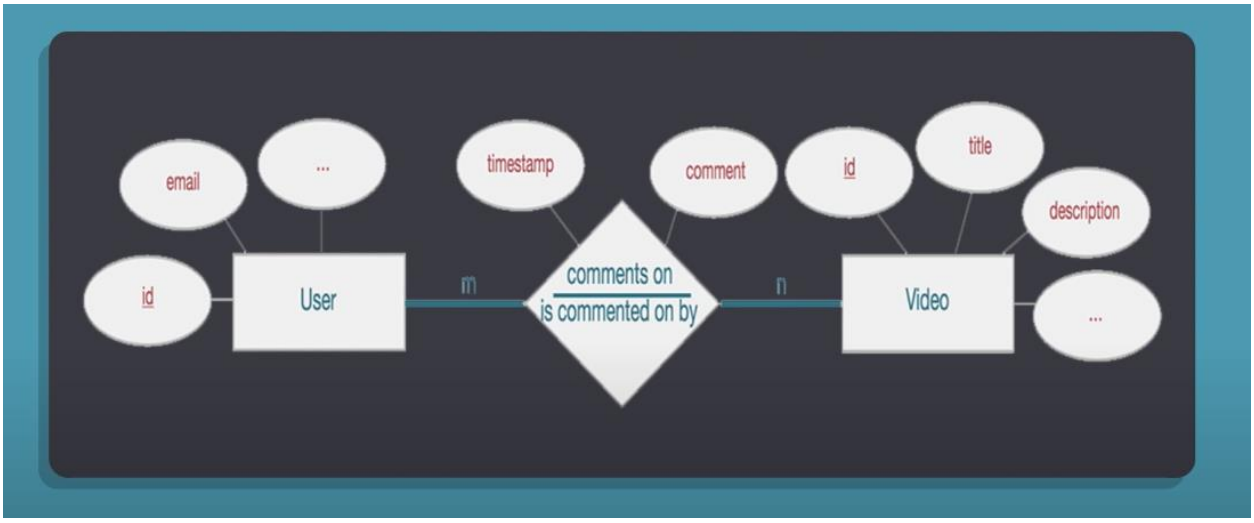
III. Data model

Data modeling is the process used to sort out entities and their relationships. Cassandra data modeling is a top-down approach of model creation, in other words, it is query-driven. In the NoSQL data modeling, the workflow is from application to models and then to data. To elaborate, the process is to analyze user behavior, identify workflows and needs, and then define queries. With the queries, table could be designed using denormalization. BATCH is used to insert or update denormalized data of multiple tables. Conceptual data models evolve into logical data model and then to physical data model.

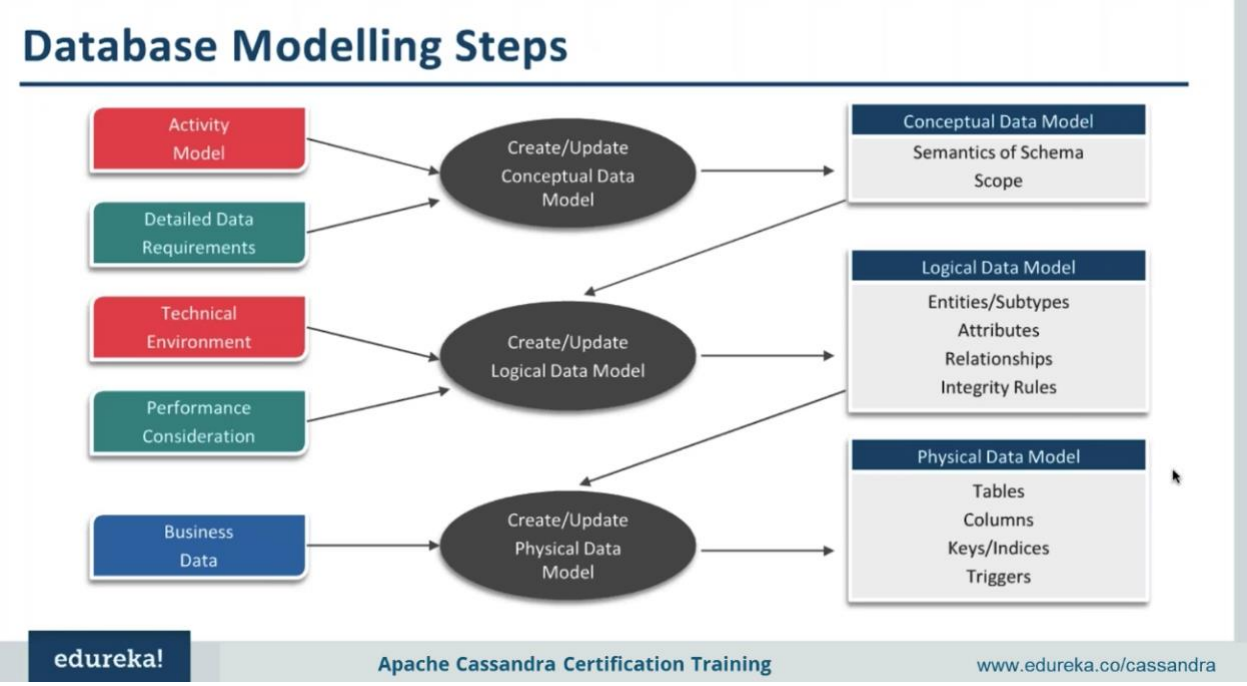
[3] Refer to: https://cassandra.apache.org/doc/latest/cassandra/architecture/guarantees.html

Relational database could join table and get data from more than one tables. On the contrary, Cassandra doesn't support joining or referential integrity across tables. There are no cascading deletes. Its table contains duplicated and denormalized data. Data is modeled around related queries. Therefore, queries need to be best designed to access a table. And a table could have more than one entity as best suits a query. These result in data duplication, to be specific, data is denormalized and duplicated to achieve read and write efficiency.



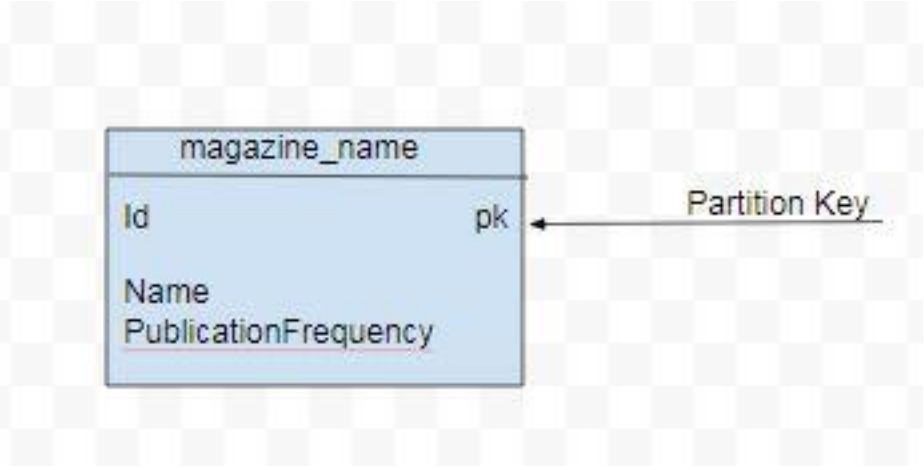


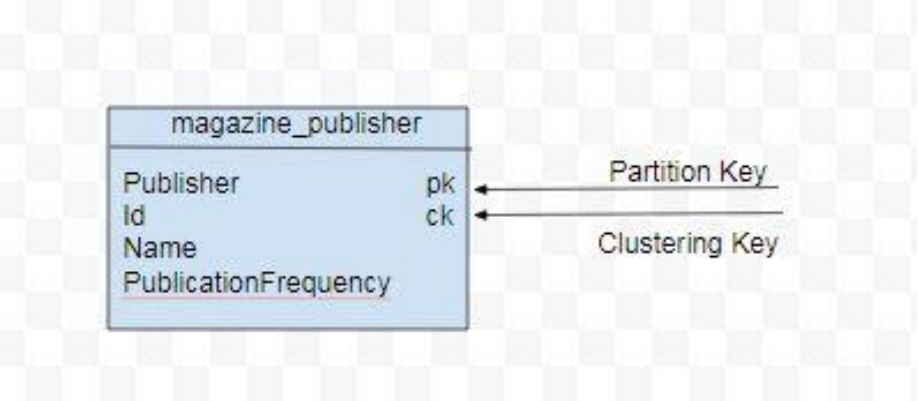
(Source: https://www.youtube.com/watch?v=_W5VvxzoS6w)



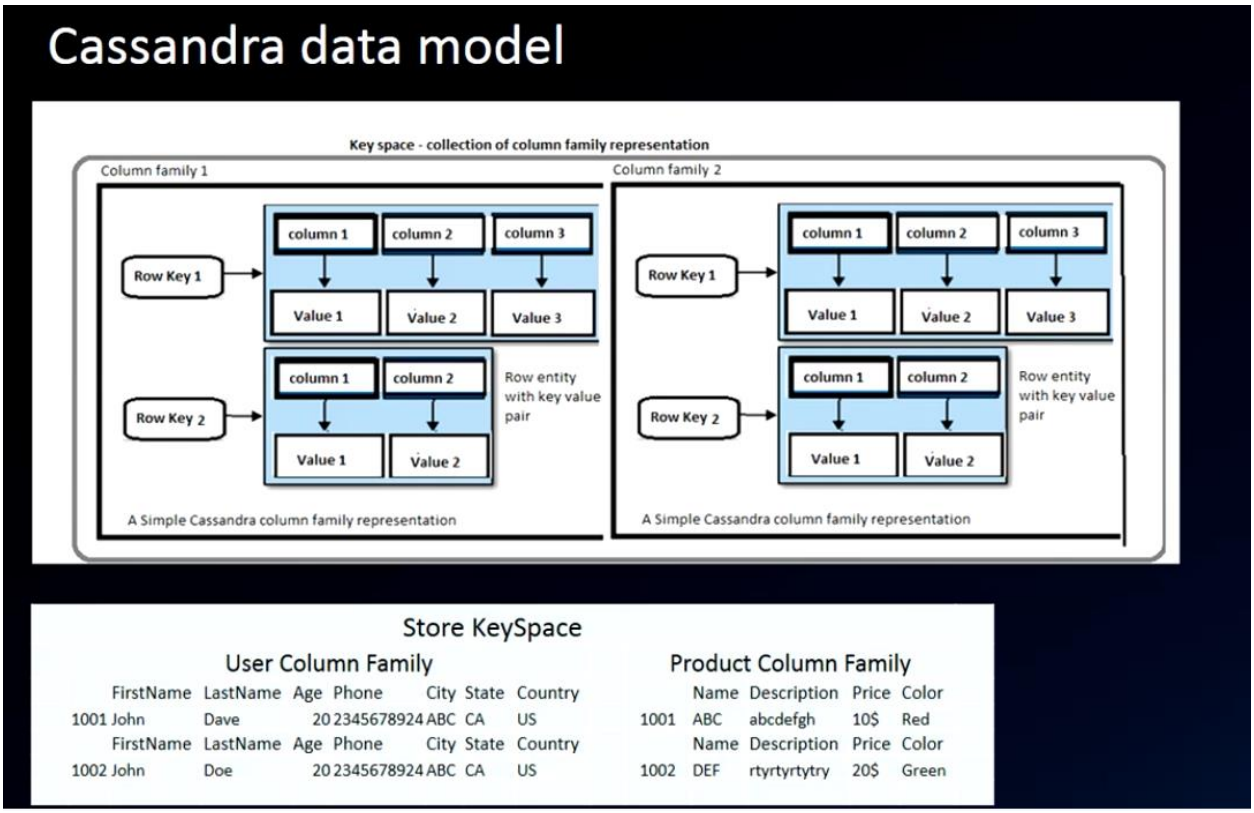
(Source: https://www.youtube.com/watch?v=eLjCtT2pBxw)

As discussed in architecture, Cassandra stores data across a cluster of nodes. It uses partition keys and a variant of consistent hashing to partition data among the nodes. Data partitioned into hash tables could be quickly looked up by partition keys. A partition key is generated from the first field of a primary key. If a table has more than one primary key, then apart from the first one, other will be used to sort within a partition rather than functioning as a partition key.





(Source: https://cassandra.apache.org/doc/latest/cassandra/data_modeling/intro.html)



(Source: https://www.youtube.com/watch?v=X-_vS8q4nu4)

IV. Applications

Cassandra has number of advantages as discussed above, thus results in its wide usage. Many well-known companies use Cassandra. For example, Reddit uses it for storage; Netflix uses it as back-end database; Cisco's WebEx uses it for storage of user feed and activity; Apple uses over 100,000 nodes; BlackRock uses it in Aladdin investment management platform; Nutanix uses it for storage of meta-data and stats, etc. Cassandra enables users to process large amount of fast-moving data in a fast, reliable, and scalable way, rendering it to be favored by companies of different background to use it for critical features.

S. No.	Company/Organization	Purpose
1.	Urban Airship	Host for 160 million users
2.	Sound Cloud	Store the dashboard of their users
3.	RockYou	Record every single click of Users in real time
4.	Reddit	Storage
5.	Rackspace	Internal Usage
6.	OpenX	Storage and replication over 130 nodes
7.	Openwave	Distributed database and storage mechanism
8.	OOyala	Built Flexible, a real-time analytics engine
9.	Netflix	Back-end database
10.	Nutanix	Storage of meta-data and stats
11.	Mahalo.com	Recording user activity logs
12.	Globo.com	Back-end database
13.	Formspring	Count responses and Storage of social graph data
14.	Digg	Storage
15.	Constant Contact	Over 200 nodes deployed in their email and social media applications
16.	Cisco's WebEx	Storage of user feed and activity
17.	CERN	Archive online DAQ system's monitoring information
18.	BlackRock	Used in Aladdin investment management platform
19.	AppScale	Back-end for Google App Engine Applications
20.	Apple	Uses 100000 nodes

(Source: https://data-flair.training/blogs/cassandra-applications/)

Next, I will briefly introduce its installation process, sample code to create database objects and CURD.

1. Installation process.

There are several options provided by Cassandra official website to install it for Linux servers. To install on Mac, I follow the following steps.

First, install homebrew (or update homebrew to newest version if already have it).

Run command:

```
brew update
```

Then use brew to install Cassandra.

Run command:

```
brew install cassandra
```

Cassandra is installed in the path /usr/local/Cellar/cassandra .

```
xinglus-mbp:~ mintsopro$ brew install cassandra
==> Downloading https://ghcr.io/v2/homebrew/core/openjdk/11/manifests/11.0.14.1
######################################################################## 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/openjdk/11/blobs/sha256:a409780
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
######################################################################## 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/cassandra/manifests/4.0.3
######################################################################## 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/cassandra/blobs/sha256:a05b39bc
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
######################################################################## 100.0%
==> Installing dependencies for cassandra: openjdk@11
==> Installing cassandra dependency: openjdk@11
==> Pouring openjdk@11--11.0.14.1.big_sur.bottle.tar.gz
🍺  /usr/local/Cellar/openjdk@11/11.0.14.1: 678 files, 298.2MB
==> Installing cassandra
==> Pouring cassandra--4.0.3.big_sur.bottle.tar.gz
==> Caveats
To restart cassandra after an upgrade:
  brew services restart cassandra
Or, if you don't want/need a background service you can just run:
  /usr/local/opt/cassandra/bin/cassandra -f
==> Summary
🍺  /usr/local/Cellar/cassandra/4.0.3: 966 files, 68.1MB
==> Running `brew cleanup cassandra`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
==> Caveats
==> cassandra
To restart cassandra after an upgrade:
  brew services restart cassandra
Or, if you don't want/need a background service you can just run:
  /usr/local/opt/cassandra/bin/cassandra -f
xinglus-mbp:~ mintsopro$
```

Then we could start Cassandra by running command:

```
brew services start cassandra
```

```
xinglus-mbp:~ mintsopro$ brew services start cassandra
==> Successfully started `cassandra` (label: homebrew.mxcl.cassandra)
xinglus-mbp:~ mintsopro$
```

Run command to login to cqsh using default credentials:

```
cqlsh
```

```
==> Successfully started `cassandra` (label: homebrew.mxcl.cassandra)
xinglus-mbp:~ mintsopro$ cqlsh
/usr/local/Cellar/cassandra/4.0.3/libexec/bin/cqlsh.py:460: DeprecationWarning: Legacy execution parameters will be r
emoved in 4.0. Consider using execution profiles.
/usr/local/Cellar/cassandra/4.0.3/libexec/bin/cqlsh.py:490: DeprecationWarning: Setting the consistency level at the
session level will be removed in 4.0. Consider using execution profiles and setting the desired consitency level to t
he EXEC_PROFILE_DEFAULT profile.
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.0.0 | Cassandra 4.0.3 | CQL spec 3.4.5 | Native protocol v5]
Use HELP for help.
cqlsh>
```

If we run command “help”, we will get a list of commands and useful topics.

```
cqlsh> help

Documented shell commands:
===========================
CAPTURE  CLS          COPY  DESCRIBE  EXPAND  LOGIN   SERIAL  SOURCE   UNICODE
CLEAR    CONSISTENCY  DESC  EXIT      HELP    PAGING  SHOW    TRACING

CQL help topics:
================
AGGREGATES               CREATE_KEYSPACE           DROP_TRIGGER      TEXT
ALTER_KEYSPACE           CREATE_MATERIALIZED_VIEW  DROP_TYPE         TIME
ALTER_MATERIALIZED_VIEW  CREATE_ROLE               DROP_USER         TIMESTAMP
ALTER_TABLE              CREATE_TABLE              FUNCTIONS         TRUNCATE
ALTER_TYPE               CREATE_TRIGGER            GRANT             TYPES
ALTER_USER               CREATE_TYPE               INSERT            UPDATE
APPLY                    CREATE_USER               INSERT_JSON       USE
ASCII                    DATE                      INT               UUID
BATCH                    DELETE                    JSON
BEGIN                    DROP_AGGREGATE            KEYWORDS
BLOB                     DROP_COLUMNFAMILY         LIST_PERMISSIONS
BOOLEAN                  DROP_FUNCTION             LIST_ROLES
COUNTER                  DROP_INDEX                LIST_USERS
CREATE_AGGREGATE         DROP_KEYSPACE             PERMISSIONS
CREATE_COLUMNFAMILY      DROP_MATERIALIZED_VIEW    REVOKE
CREATE_FUNCTION          DROP_ROLE                 SELECT
CREATE_INDEX             DROP_TABLE                SELECT_JSON

cqlsh>
```

Then we can create database objects.

2. Example to create database objects.

First we need to create a keyspace.

Given below is an example of creating a KeySpace named tutorial.

```
cqlsh.> CREATE KEYSPACE tutorial
WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 3};
```

```
cqlsh> DESCRIBE keyspaces;
```

```
cqlsh> DESCRIBE keyspaces;

system       system_distributed  system_traces  system_virtual_schema
system_auth  system_schema       system_views   tutorial

cqlsh> DESCRIBE tutorial

CREATE KEYSPACE tutorial WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '3'}  AND durable_writes = true;
```

```
cqlsh> USE tutorial;
```

Next we create a table named employee.

```
CREATE TABLE employee(
  emp_id int PRIMARY KEY,
  emp_name text,
  emp_city text,
  emp_sal varint,
  emp_phone varint
  );
```

```
select * from employee;
```

```
cqlsh> USE tutorial;
cqlsh:tutorial> CREATE TABLE employee(
            ...    emp_id int PRIMARY KEY,
            ...    emp_name text,
            ...    emp_city text,
            ...    emp_sal varint,
            ...    emp_phone varint
            ...    );
cqlsh:tutorial> select * from employee;

 emp_id | emp_city | emp_name | emp_phone | emp_sal
--------+----------+----------+-----------+---------

(0 rows)
cqlsh:tutorial>
```

3. Example to use (CRUD) the database.

CRUD stands for Create, Update, Read and Delete or Drop operations.

a) **Create**.

Command INSERT is used to insert data into a certain table. We run the below commands to insert into table “employee” that we just created.

```
INSERT INTO employee (emp_id, emp_name, emp_city,
  emp_phone, emp_sal) VALUES(1,'ram', 'Hyderabad', 9848022338, 50000);

INSERT INTO employee (emp_id, emp_name, emp_city,
  emp_phone, emp_sal) VALUES(2,'robin', 'Hyderabad', 9848022339, 40000);

INSERT INTO employee (emp_id, emp_name, emp_city,
  emp_phone, emp_sal) VALUES(3,'rahman', 'Chennai', 9848022330, 45000);
```

We can verify the data by running:

```
SELECT * FROM employee;
```

```
cqlsh:tutorial> INSERT INTO employee (emp_id, emp_name, emp_city,
            ...    emp_phone, emp_sal) VALUES(1,'ram', 'Hyderabad', 9848022338, 50000);
cqlsh:tutorial> INSERT INTO employee (emp_id, emp_name, emp_city,
            ...    emp_phone, emp_sal) VALUES(2,'robin', 'Hyderabad', 9848022339, 40000);
cqlsh:tutorial> INSERT INTO employee (emp_id, emp_name, emp_city,
            ...    emp_phone, emp_sal) VALUES(3,'rahman', 'Chennai', 9848022330, 45000);
cqlsh:tutorial> SELECT * FROM employee
            ...
            ... ;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 | Hyderabad |    robin | 9848022339 |   40000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
cqlsh:tutorial>
```

b) Update.

Command UPDATE is used to update data in a certain table. For example, we can update the city of employee with id 1 to “Sanjose”, and his salary to 100000:

```
UPDATE employee SET emp_city='Sanjose',emp_sal=100000
  WHERE emp_id=1;
```

Run select command again and we can verify that the data has been updated.

```
 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 | Hyderabad |    robin | 9848022339 |   40000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
cqlsh:tutorial> UPDATE employee SET emp_city='Sanjose',emp_sal=100000
           ...    WHERE emp_id=1;
cqlsh:tutorial> select * from employee;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 |   Sanjose |      ram | 9848022338 |  100000
      2 | Hyderabad |    robin | 9848022339 |   40000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
cqlsh:tutorial>
```

c) Read.

SELECT is used to read data. As showed above, we select a certain table and get all information about the table by running:

```
SELECT * FROM employee;
```

We can also select particular columns in the table:

```
SELECT emp_name, emp_city from employee;
```

```
           ...    WHERE emp_id=1;
cqlsh:tutorial> select * from employee;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 |   Sanjose |      ram | 9848022338 |  100000
      2 | Hyderabad |    robin | 9848022339 |   40000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
cqlsh:tutorial> SELECT emp_name, emp_city from employee;

 emp_name | emp_city
----------+-----------
      ram |   Sanjose
    robin | Hyderabad
   rahman |   Chennai

(3 rows)
cqlsh:tutorial>
```

We can also use “WHERE” clause to select specific data that meet our requirements.

SELECT * FROM employee WHERE emp_sal=10000;

```
(3 rows)
cqlsh:tutorial> SELECT * FROM employee WHERE emp_sal=10000;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and t
hus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
cqlsh:tutorial> SELECT * FROM employee WHERE emp_sal=100000   ALLOW FILTERING ;

 emp_id | emp_city | emp_name | emp_phone  | emp_sal
--------+----------+----------+------------+--------
      1 |  Sanjose |      ram | 9848022338 |  100000

(1 rows)
cqlsh:tutorial>
```

d) Drop / Delete.

DELETE command is used to delete certain data from our table. We can add "WHERE" clause to delete certain data that is qualified.

DELETE emp_sal from employee where emp_id = 1;

```
cqlsh:tutorial> DELETE emp_sal from employee where emp_id = 1;
cqlsh:tutorial> select * from employee;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 |   Sanjose |      ram | 9848022338 |    null
      2 | Hyderabad |    robin | 9848022339 |   40000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
cqlsh:tutorial>
```

Now we can see the data we delete become "null".

DROP:

Command DROP can be used to drop a Keyspace, a table or an index.
Command "DROP INDEX" is used to drop an index.

First, we create an index:

Create index emp_index on tutorial.employee(emp_sal);

Drop index emp_index;

DROP TABLE employee;

DROP KEYSPACE tutorial;

```
cqlsh:tutorial>
cqlsh:tutorial> Create index emp_index on tutorial.employee(emp_sal);

cqlsh:tutorial>
cqlsh:tutorial> Drop index emp_index;
cqlsh:tutorial> DROP TABLE employee;
cqlsh:tutorial> Describe tutorial;

CREATE KEYSPACE tutorial WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '3'}  AND durable_writes = true;
cqlsh:tutorial> select * from employee;
InvalidRequest: Error from server: code=2200 [Invalid query] message="table employee does not exist"
cqlsh:tutorial> DROP KEYSPACE tutorial;
cqlsh:tutorial> describe KEYSPACES;

system       system_distributed  system_traces  system_virtual_schema
system_auth  system_schema       system_views

cqlsh:tutorial>
```

4. Exit Cassandra.

We could run this command to exit Cassandra:

```
Exit
```

We could stop Cassandra using:

```
brew services stop cassandra
```

```
cqlsh:tutorial> exit
xinglus-mbp:~ mintsopro$ brew services stop cassandra
Stopping `cassandra`... (might take a while)
==> Successfully stopped `cassandra` (label: homebrew.mxcl.cassandra)
xinglus-mbp:~ mintsopro$
```

References:

Cassandra official website:
https://cassandra.apache.org/_/index.html
https://cassandra.apache.org/doc/latest/cassandra/architecture/index.html

Wikipedia:
https://en.wikipedia.org/wiki/Apache_Cassandra

https://en.wikipedia.org/wiki/NoSQL

NoSQL databases:
https://www.couchbase.com/resources/why-nosql
https://aws.amazon.com/nosql/

Cassandra Applications | Why Cassandra Is So Popular?
https://data-flair.training/blogs/cassandra-applications/

YouTube Videos:
Cassandra Architecture: Understanding, Scaling and Optimizing - Ben Bromhead:
https://www.youtube.com/watch?v=0mntwB6nbc0

Cassandra Architecture | Apache Cassandra Tutorial | Apache Cassandra Training | Edureka
https://www.youtube.com/watch?v=ZM0wJ1suCUE

Lesson 3: Cassandra - Cassandra Data Model
https://www.youtube.com/watch?v=X-_vS8q4nu4

05 | Intro to Cassandra - The Art of Data Modeling
https://www.youtube.com/watch?v=_W5VvxzoS6w

Cassandra Data Modeling | Introduction to Cassandra Data Model | Apache Cassandra Training | Edureka
https://www.youtube.com/watch?v=eLjCtT2pBxw

How to install Cassandra on MacOS
https://www.javatpoint.com/how-to-install-cassandra-on-mac

Create database in Cassandra
https://www.geeksforgeeks.org/create-database-in-cassandra/

Cassandra Crud Operation – Create, Update, Read & Delete
https://data-flair.training/blogs/cassandra-crud-operation/

04 | Intro to Cassandra - Create a Table
https://www.youtube.com/watch?v=zrJlojbxeXo&list=PL2g2h-wyI4SqCdxdiyi8enEyWvACcUa9R&index=4

Cassandra Tutorial
https://www.tutorialspoint.com/cassandra/index.htm