→ Basic structure of tables in cassandra

a. In Cassandra, we have
    i. Columns
        1. Column is the smallest model in cassandra
        2. It consist name, value, timestamp, TTL(Time To Live)
        3. Timestamp is used to avoid conflict when any client makes Write operation
        4. Time To Live (TTL) is a feature that allows you to set expiration time for data in a column, when a TTL value is set for a column, the data is to be automatically deleted after the specified period
        5. TTL is used to manage the size of column family and to remove the outdated data automatically
        6. TTL value can be set at column or row level.
           If TTL value is set at column level, the value applies only to that specific column.
           If TTL value is set at row level, the value applies to all the columns in that row

```
INSERT INTO table_name(col1, col2, col3) VALUES (val1, val2, val3)
USING TTL 3600 WHERE col1=val1;
```

    ii. Rows
        1. It consist of row-key, which is also known as Primary Key and set of column consist of row-key
        2. Each row may have different column name and there are no timestamp
        3. A row must be stored on a node and cannot be separated into two or more nodes in a cluster

```
INSERT INTO table_name(col1, col2, col3) VALUES (val1, val2, val3)
USING TTL 3600;
```

    iii. Column family
        1. It is a container of set of rows
        2. A row-key in column family must be unique to identify rows
        3. The column family can be an analogy to table in relational database

    iv. Keyspace
        1. It is a namespace that defines replication factor or replication strategy
        2. Keyspace is a highest level of organization in cassandra
        3. A keyspace is analogous to schema in RDBMS
        4. It contains replication settings on how the data is replicated in the clusters. One cluster only contains one keyspace

    v. Tables

1. Introduced in 3.0+ version
2. Tables in cassandra 3.x + versions, is a collection of row-defined schema
3. A table is defined by partition key which defines how data is distributed among the cluster and how data is stored within the partition

vi.

b. asa

→ Write Operation in Cassandra

a. When you write to a node, first it will write to commit log
b. Cassandra writes to memTable in the memory table.
c. Data is written to memTable on each write request, and it writes in commit log separately
d. memTable is temporary stored in memory while commit log logs the transaction record for backup process
e. When mem Table is full, data is flushed to SSTable data file
f. asa

→ Read Operation in Cassandra

a. While reading data
b. asa

→ Commands

a. INSERT command
   i. row level

```
INSERT INTO table_name(col1, col2, col3) VALUES (val1, val2, val3)
USING TTL 3600;
```

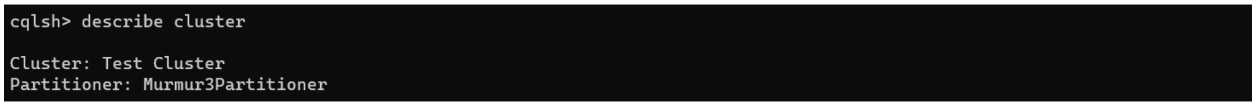
   ii. column level

```
INSERT INTO table_name(col1, col2, col3) VALUES (val1, val2, val3)
USING TTL 3600 WHERE col1=val1;
```

b.

c. describe
   i. This describes the current cluster of cassandra and information about the cluster
   ii. Example of describe cluster

```
>describe cluster
```

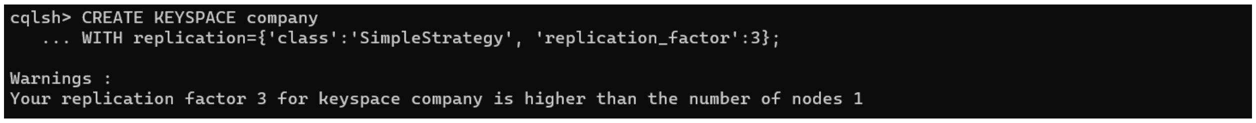


```
cqlsh> describe cluster

Cluster: Test Cluster
Partitioner: Murmur3Partitioner
```

d. CREATE KEYSPACE
   i. Creates keyspace
   ii. Example to create keyspace:

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

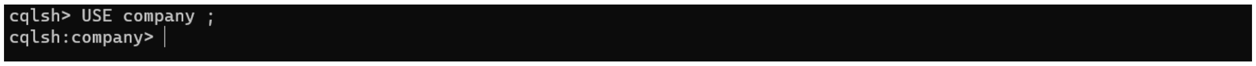


```
cqlsh> CREATE KEYSPACE company
   ... WITH replication={'class':'SimpleStrategy', 'replication_factor':3};

Warnings :
Your replication factor 3 for keyspace company is higher than the number of nodes 1
```

e. USE
    i. Example to use USE command

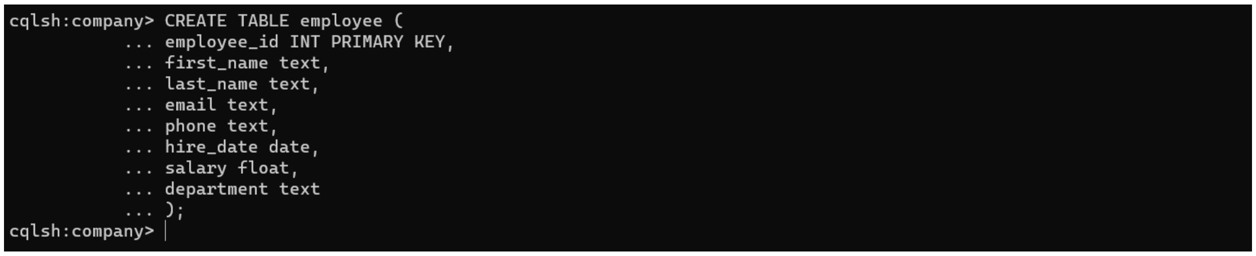
```
>USE company ;
```



```
cqlsh> USE company ;
cqlsh:company>
```

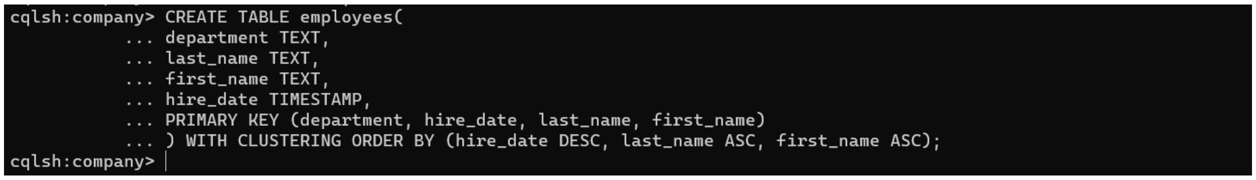
f. CREATE TABLE
    i. Example to use CREATE TABLE command

```
>CREATE TABLE employee (
employee_id INT PRIMARY KEY,
first_name text,
last_name text,
email text,
phone text,
hire_date date,
salary float,
department text
);
```



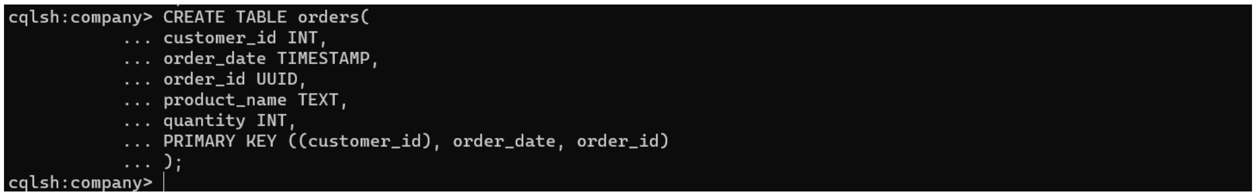
```
cqlsh:company> CREATE TABLE employee (
           ... employee_id INT PRIMARY KEY,
           ... first_name text,
           ... last_name text,
           ... email text,
           ... phone text,
           ... hire_date date,
           ... salary float,
           ... department text
           ... );
cqlsh:company>
```

```
>CREATE TABLE employees(
department TEXT,
last_name TEXT,
first_name TEXT,
hire_date TIMESTAMP,
PRIMARY KEY (department, hire_date, last_name, first_name)
) WITH CLUSTERING ORDER BY (hire_date DESC, last_name ASC, first_name
ASC);
```



```
cqlsh:company> CREATE TABLE employees(
           ... department TEXT,
           ... last_name TEXT,
           ... first_name TEXT,
           ... hire_date TIMESTAMP,
           ... PRIMARY KEY (department, hire_date, last_name, first_name)
           ... ) WITH CLUSTERING ORDER BY (hire_date DESC, last_name ASC, first_name ASC);
cqlsh:company>
```

```
>CREATE TABLE orders(
customer_id INT,
order_date TIMESTAMP,
order_id UUID,
product_name TEXT,
quantity INT,
PRIMARY KEY ((customer_id), order_date, order_id)
);
```

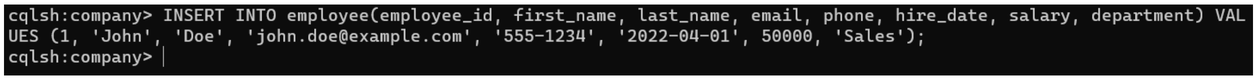


```
cqlsh:company> CREATE TABLE orders(
           ... customer_id INT,
           ... order_date TIMESTAMP,
           ... order_id UUID,
           ... product_name TEXT,
           ... quantity INT,
           ... PRIMARY KEY ((customer_id), order_date, order_id)
           ... );
cqlsh:company>
```

g. INSERT INTO

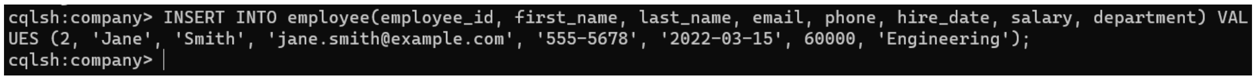
   i. Example to use INSERT INTO command

```
>INSERT INTO employee(employee_id, first_name, last_name, email,
phone, hire_date, salary, department) VALUES (1, 'John', 'Doe',
'john.doe@example.com', '555-1234', '2022-04-01', 50000, 'Sales');
```



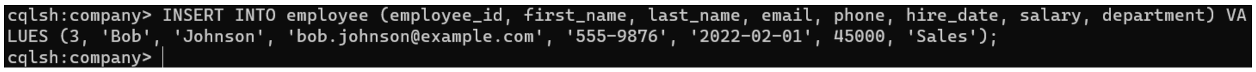
```
cqlsh:company> INSERT INTO employee(employee_id, first_name, last_name, email, phone, hire_date, salary, department) VAL
UES (1, 'John', 'Doe', 'john.doe@example.com', '555-1234', '2022-04-01', 50000, 'Sales');
cqlsh:company>
```

```
>INSERT INTO employee(employee_id, first_name, last_name, email,
phone, hire_date, salary, department) VALUES (2, 'Jane', 'Smith',
'jane.smith@example.com', '555-5678', '2022-03-15', 60000,
'Engineering');
```



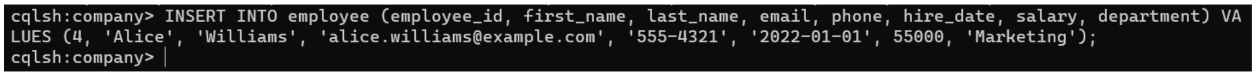
```
cqlsh:company> INSERT INTO employee(employee_id, first_name, last_name, email, phone, hire_date, salary, department) VAL
UES (2, 'Jane', 'Smith', 'jane.smith@example.com', '555-5678', '2022-03-15', 60000, 'Engineering');
cqlsh:company>
```

```
>INSERT INTO employee (employee_id, first_name, last_name, email,
phone, hire_date, salary, department) VALUES (3, 'Bob', 'Johnson',
'bob.johnson@example.com', '555-9876', '2022-02-01', 45000, 'Sales');
```



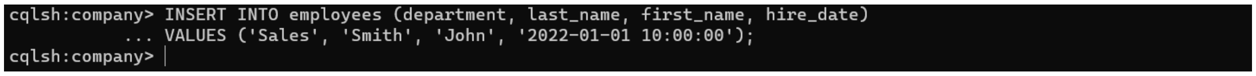
```
cqlsh:company> INSERT INTO employee (employee_id, first_name, last_name, email, phone, hire_date, salary, department) VA
LUES (3, 'Bob', 'Johnson', 'bob.johnson@example.com', '555-9876', '2022-02-01', 45000, 'Sales');
cqlsh:company>
```

```
>INSERT INTO employee (employee_id, first_name, last_name, email,
phone, hire_date, salary, department) VALUES (4, 'Alice', 'Williams',
'alice.williams@example.com', '555-4321', '2022-01-01', 55000,
'Marketing');
```



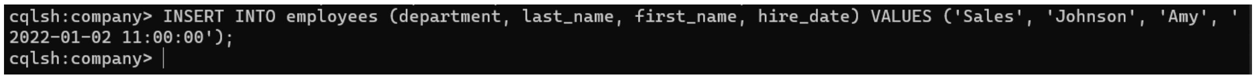
```
cqlsh:company> INSERT INTO employee (employee_id, first_name, last_name, email, phone, hire_date, salary, department) VA
LUES (4, 'Alice', 'Williams', 'alice.williams@example.com', '555-4321', '2022-01-01', 55000, 'Marketing');
cqlsh:company>
```

```
>INSERT INTO employees (department, last_name, first_name, hire_date)
VALUES ('Sales', 'Smith', 'John', '2022-01-01 10:00:00');
```

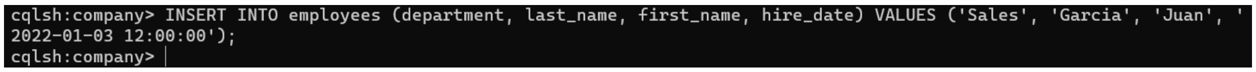


```
cqlsh:company> INSERT INTO employees (department, last_name, first_name, hire_date)
           ... VALUES ('Sales', 'Smith', 'John', '2022-01-01 10:00:00');
cqlsh:company>
```

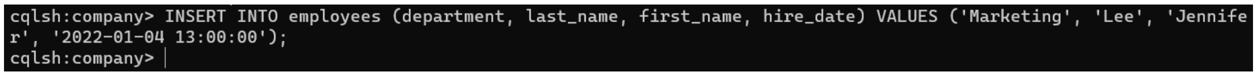
```
>INSERT INTO employees (department, last_name, first_name, hire_date)
VALUES ('Sales', 'Johnson', 'Amy', '2022-01-02 11:00:00');
```


```
cqlsh:company> INSERT INTO employees (department, last_name, first_name, hire_date) VALUES ('Sales', 'Johnson', 'Amy', '
2022-01-02 11:00:00');
cqlsh:company> |
```

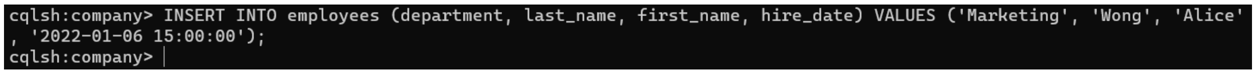
```
>INSERT INTO employees (department, last_name, first_name, hire_date)
VALUES ('Sales', 'Garcia', 'Juan', '2022-01-03 12:00:00');
```


```
cqlsh:company> INSERT INTO employees (department, last_name, first_name, hire_date) VALUES ('Sales', 'Garcia', 'Juan', '
2022-01-03 12:00:00');
cqlsh:company> |
```

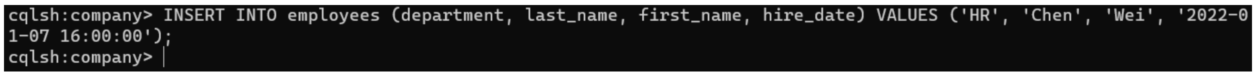
```
>INSERT INTO employees (department, last_name, first_name, hire_date)
VALUES ('Marketing', 'Lee', 'Jennifer', '2022-01-04 13:00:00');
```


```
cqlsh:company> INSERT INTO employees (department, last_name, first_name, hire_date) VALUES ('Marketing', 'Lee', 'Jennife
r', '2022-01-04 13:00:00');
cqlsh:company> |
```

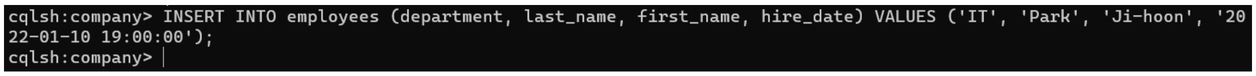
```
>INSERT INTO employees (department, last_name, first_name, hire_date)
VALUES ('Marketing', 'Wong', 'Alice', '2022-01-06 15:00:00');
```


```
cqlsh:company> INSERT INTO employees (department, last_name, first_name, hire_date) VALUES ('Marketing', 'Wong', 'Alice'
, '2022-01-06 15:00:00');
cqlsh:company> |
```

```
>INSERT INTO employees (department, last_name, first_name, hire_date)
VALUES ('HR', 'Chen', 'Wei', '2022-01-07 16:00:00');
```


```
cqlsh:company> INSERT INTO employees (department, last_name, first_name, hire_date) VALUES ('HR', 'Chen', 'Wei', '2022-0
1-07 16:00:00');
cqlsh:company> |
```

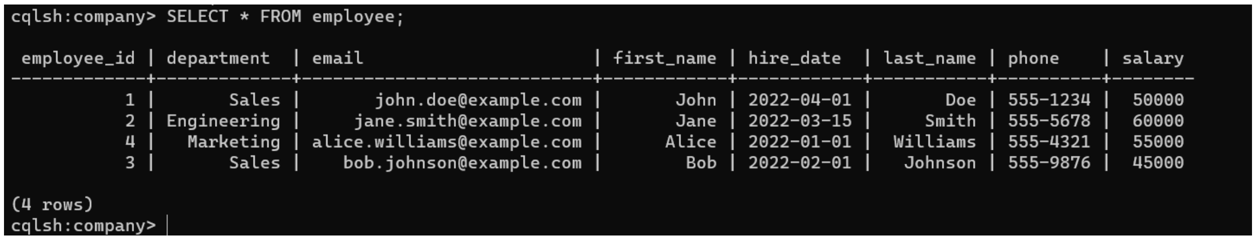
```
>INSERT INTO employees (department, last_name, first_name, hire_date)
VALUES ('IT', 'Park', 'Ji-hoon', '2022-01-10 19:00:00');
```


```
cqlsh:company> INSERT INTO employees (department, last_name, first_name, hire_date) VALUES ('IT', 'Park', 'Ji-hoon', '20
22-01-10 19:00:00');
cqlsh:company> |
```

h. SELECT command

   i. Example to use SELECT command

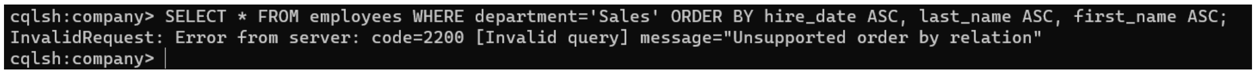
```
>SELECT * FROM employee;
```


```
cqlsh:company> SELECT * FROM employee;

 employee_id | department  | email                      | first_name | hire_date  | last_name | phone    | salary
-------------+-------------+----------------------------+------------+------------+-----------+----------+--------
           1 |       Sales |       john.doe@example.com |       John | 2022-04-01 |       Doe | 555-1234 |  50000
           2 | Engineering |     jane.smith@example.com |       Jane | 2022-03-15 |     Smith | 555-5678 |  60000
           4 |   Marketing | alice.williams@example.com |      Alice | 2022-01-01 |  Williams | 555-4321 |  55000
           3 |       Sales |    bob.johnson@example.com |        Bob | 2022-02-01 |   Johnson | 555-9876 |  45000

(4 rows)
cqlsh:company> |
```

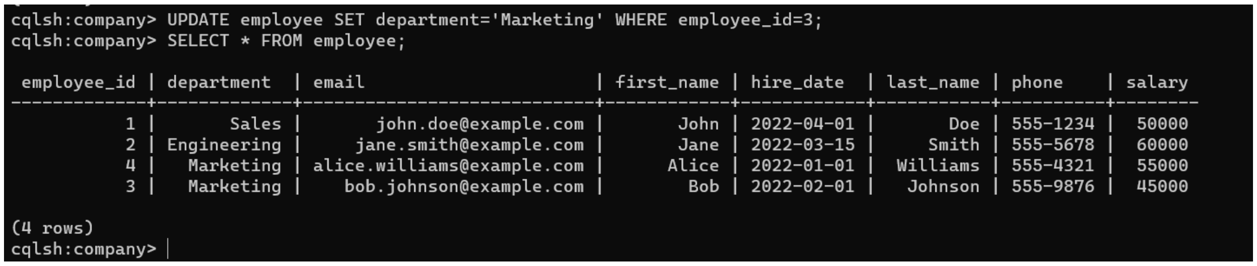
```
>SELECT * FROM employees WHERE department='Sales' ORDER BY hire_date
ASC, last_name ASC, first_name ASC;
```


```
cqlsh:company> SELECT * FROM employees WHERE department='Sales' ORDER BY hire_date ASC, last_name ASC, first_name ASC;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Unsupported order by relation"
cqlsh:company> |
```

i. UPDATE command

   i. Example of UPDATE command

```
>UPDATE employee SET department='Marketing' WHERE employee_id=3;
>SELECT * FROM employee;
```



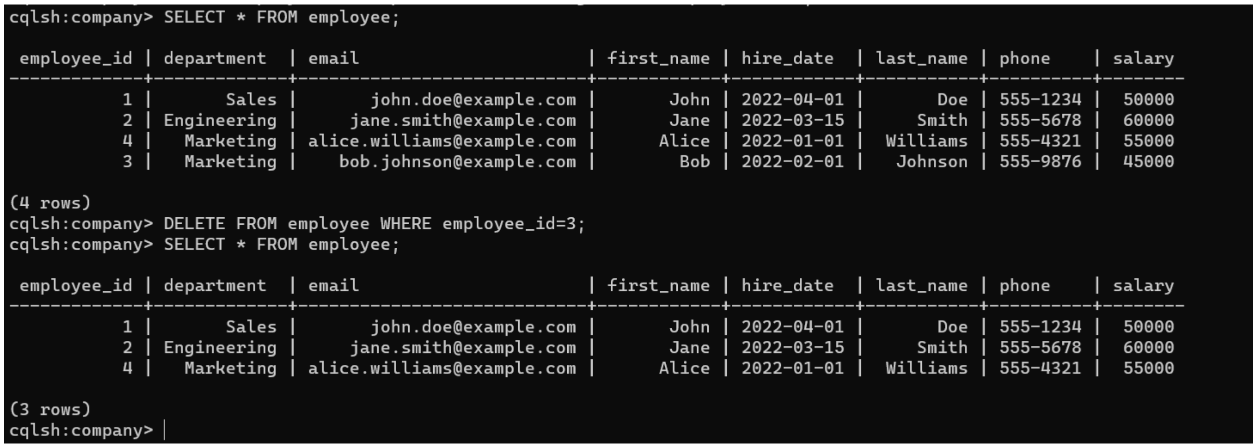
```
cqlsh:company> UPDATE employee SET department='Marketing' WHERE employee_id=3;
cqlsh:company> SELECT * FROM employee;

 employee_id | department  | email                      | first_name | hire_date  | last_name | phone    | salary
-------------+-------------+----------------------------+------------+------------+-----------+----------+--------
           1 |       Sales |       john.doe@example.com |       John | 2022-04-01 |       Doe | 555-1234 |  50000
           2 | Engineering |     jane.smith@example.com |       Jane | 2022-03-15 |     Smith | 555-5678 |  60000
           4 |   Marketing | alice.williams@example.com |      Alice | 2022-01-01 |  Williams | 555-4321 |  55000
           3 |   Marketing |    bob.johnson@example.com |        Bob | 2022-02-01 |   Johnson | 555-9876 |  45000

(4 rows)
cqlsh:company>
```

j. DELETE command
    i. Example of DELETE command

```
>SELECT * FROM employee;
>DELETE FROM employee WHERE employee_id=3;
>SELECT * FROM employee;
```



```
cqlsh:company> SELECT * FROM employee;

 employee_id | department  | email                      | first_name | hire_date  | last_name | phone    | salary
-------------+-------------+----------------------------+------------+------------+-----------+----------+--------
           1 |       Sales |       john.doe@example.com |       John | 2022-04-01 |       Doe | 555-1234 |  50000
           2 | Engineering |     jane.smith@example.com |       Jane | 2022-03-15 |     Smith | 555-5678 |  60000
           4 |   Marketing | alice.williams@example.com |      Alice | 2022-01-01 |  Williams | 555-4321 |  55000
           3 |   Marketing |    bob.johnson@example.com |        Bob | 2022-02-01 |   Johnson | 555-9876 |  45000

(4 rows)
cqlsh:company> DELETE FROM employee WHERE employee_id=3;
cqlsh:company> SELECT * FROM employee;

 employee_id | department  | email                      | first_name | hire_date  | last_name | phone    | salary
-------------+-------------+----------------------------+------------+------------+-----------+----------+--------
           1 |       Sales |       john.doe@example.com |       John | 2022-04-01 |       Doe | 555-1234 |  50000
           2 | Engineering |     jane.smith@example.com |       Jane | 2022-03-15 |     Smith | 555-5678 |  60000
           4 |   Marketing | alice.williams@example.com |      Alice | 2022-01-01 |  Williams | 555-4321 |  55000

(3 rows)
cqlsh:company>
```

k.
l. asa

→ List of commands that cassandra does not support

a. Does not support Aggregation queries like min, max, sum
b. Does not support Group by queries
c. Does not support Joins
d. Does not support Logical OR
e. Does not support Wildcards
f. Does not support Unions, intersections
g. Table column cannot be filtered without creating index
h. Greater than/less than queries only supported on clustering columns. This is the reason why cassandra not suitable for analytical purpose as there are so many limitations
i. asa

→ Terminologies:

a. Node:
    i. It is the basic infrastructure

ii. It is a fully functional machine which connects to other nodes in a cluster with high network speed
iii. The fully functional machines can be servers or EC2 instance or any virtual machine
iv. All the nodes are connected/organized in ring network topology
v. Every node works independently and have same role in the topology
vi. This arrangement of node is called p2p (peer-to-peer) structure
vii. Each node accepts Read/Write requests
viii. Each node is responsible for storing a portion of data and each node communicates with each other to ensure data consistency
ix. The number of nodes can be increased/decreased in a cluster to handle volume of traffic
x. A node stores information of location of data center
xi. It keeps data like keyspace, table, schema

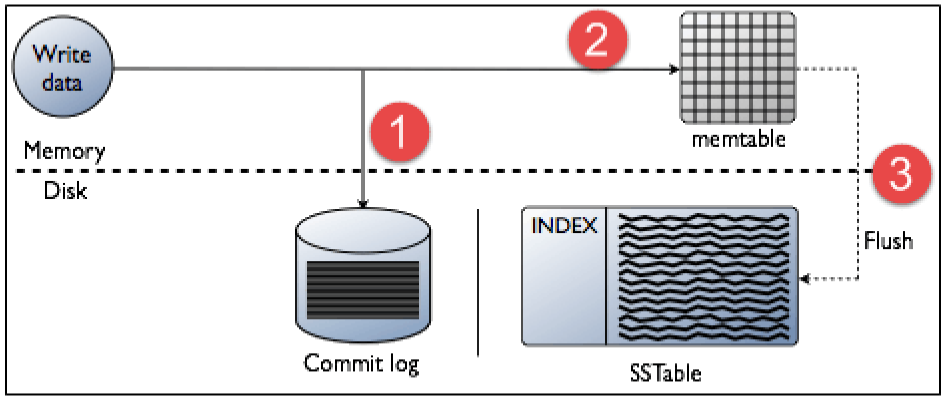
b. Racks
    i. Rack is a logical group of nodes within the topological ring. We can say, Rack is a collection of servers
    ii. A database uses a rack to ensure that replicas are distributed among different logical nodes. As a result, it sends operation not only to one node, but to multiple nodes

c. Data Centers
    i. A data center is a logical set of racks
    ii. A data center should contain at least one rack
    iii. Data center is used to provide fault tolerance, disaster recovery, low latency, access of data to users in different locations
    iv. Replication strategy is used to replicate data across data centers to ensure high availability of data
    v. While writing any operation, data is replicated to the number of nodes in the same data center and to specified number of nodes in different data centers. This way, it provides fault tolerance at data center level

d. Cluster
    i. It is a collection of nodes which work together to store and manage data
    ii. A cluster can have one or more data centers within it

e. Commit log
    i. When data is written to cassandra, the data is stored in memory cache which is mem-Table, and also append to commit log
    ii.

f. memTable
    i. mem-Table is an in-memory cache that holds data before data is flushed to SSTable

g. SSTable
    i. SSTable is a file that stores set of immutable data rows or key-values stored by row key
    ii. 

h. Seed Node
    i. A seed node is a node which is used to bootstrap a new node into the cluster
    ii. A new node is added to the cluster, it needs to discover the topology of the cluster and connect to other nodes in the cluster, this process is called bootstrapping. It typically needs list of initial nodes to connect
    iii. It plays an important role in bootstrapping by providing important information to new nodes to join a cluster and start participating in the cluster. Without seed node, new node may not be able to discover other nodes in the cluster
    iv. A seed node is designated in the configuration of the cluster to provide a list of initial nodes to new nodes joining the cluster. A new node contacts seed node to get the list of other nodes in the cluster

i. Gossip Protocol
    i. It is used for p2p (peer-to-peer) communication
    ii. Gossip informs a node about their state
    iii. This protocol is used to communicate cluster state information between nodes and cluster
    iv. Each node in cluster periodically sends gossip message to few other nodes in the cluster which contains information about itself and other nodes
    v. When a node receives a gossip message, it updates its view of cluster based on the information in the message
    vi. If a node receives gossip message about new node, it contacts new node to learn about it and add it to the view of cluster
    vii. If a node receives gossip that a node has failed, it marks the node down and propagate this information to the other nodes in the cluster and start raising the flag of severity level
    viii. The node in cluster gossip with each other and exchange information about states and update their view until they all have same view

j. Partition

    i. Partitioning is a way of distributing the data across the nodes
    ii. Each table has primary key and the primary key is divided into partition key and clustering column
    iii. Clustering column is optional and there is no unique constraint for any of the key
    iv. The partition key is used to index the data
    v. All the rows which share the common partition key, make single data partition which is basic unit of partitioning storage and retrieval of data
    vi. Partitioning is a technique to distribute data across multiple nodes in a cluster. The data is divided into partitions based on the partition key which is a part of the primary key of a table
    vii. Each partition is stored on a specific node in a cluster based on a hashing algorithm
    viii. When a query is made to retrieve data from partition, cassandra routes the query to the node that owns the partition
    ix. Partition key is converted into a token by the partitioner. These token values can range from -2^{63} to $+2^{63}-1$
    x.

k. Replication
    i. Data in each keyspace is replicated with replication factor
    ii. Replication is process of copying data across multiple nodes in a cluster to ensure data is available even if one node fails
    iii. There are many replication strategies
        1. Simple strategy
            a. This replicates the data in the node across the same cluster
        2. Network topology strategy
            a. Data is replicated across the nodes in different data centers

l. asa

→