Cassandra Activity

A. Simple strategy

1. Launch Cassandra server and open CQLSH.
2. To display all the keyspaces (databases), type:

```
describe keyspaces;
```

3. Create a keyspace named: SALES with replication using simple strategy:

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

4. To go into the SALES keyspace:

```
use SALES;
```

5. Create the following tables, just like creating tables in MySQL.

product	
prodid	int (PK)
description	text
name	text
price	float

users	
userid	int (PK)
firstname	text
lastname	text
email	text
address	text

orders	
orderid	int (PK)
orderdate	date
prodid	int
quantity	Int
userid	Int

6. Populate the tables accordingly:

users				
userid	firstname	lastname	email	address
1	Kimberly	Nicholas	KimberlyDNicholas@teleworm.us	1514 Adams Drive Houston, TX 77027
2	Joshua	Prince	JoshuaTPrince@teleworm.us	75117 Metavante Way Sioux

				Falls, SD, 57186
3	Peg	Legge	plegge@mail.com	2984 Elsie Drive Mount Vernon, SD 57363
4	George	Parker	GeorgeLParker@dayrep.com	300 My Drive New York, NY 10013
5	Shanda	Brown	ShandaJBrown@jourrapide.com	2549 Sussex Court Dublin, TX 76446
6	Charlene	Wheeler	CharleneMWheeler@armyspy.com	3846 Hillcrest Lane Irvine, CA 92614
7	Fredrick	Heyer	FredrickMHeyer@jourrapide.com	3409 Deans Lane Pleasantville, NY 10570
8	Billy	McIntire	BillyMMcIntire@rhyta.com	1275 Stanley Avenue Garden City, NY 11530
9	Jordan	Clayton	JordanLCayton@jourrapide.com	1658 Geraldine Lane New York, NY 10007
10	Nguyet	Bentz	NguyetRBentz@dayrep.com	3787 Zimmerman Lane Los Angeles, CA 90071

product			
prodid	description	name	price
800	Apple	Apple	4
801	Coconut	Coconut	5
807	Watermelon	Watermelon	7
809	Apricot	Apricot	3
810	Kiwi	Kiwi	3
811	Jackfruit	Jackfruit	9
813	Lemon	Lemon	7
814	Guava	Guava	4
815	Orange	Orange	8

orders				
orderid	orderdate	prodid	quantity	userid
1001	2022-11-01	814	5	3
1002	2022-11-15	800	10	10
1003	2022-11-15	810	3	9
1004	2022-11-15	813	10	2
1005	2022-11-16	807	2	5
1006	2022-11-16	810	5	4
1007	2022-11-17	811	2	8
1008	2022-11-17	810	2	8
1009	2022-11-17	810	2	8
1010	2022-11-18	809	3	1

7. Give the code to count the number of users in the users table.
8. Display all the orders placed between Nov. 1, 2022 to Nov. 14, 2022.
9. Display all the order id with product id 810.
10. Display maximum and minimum orders with their corresponding order id. Use alias to label the columns for maximum and minimum orders, respectively.

B. Network Strategy Topology

1. Launch CQLSH.
2. Create a keyspace that involves partitioning.
3. Let us define primary key first.

The way primary keys work in Cassandra is an important concept to grasp.

A primary key in Cassandra **consists of one or more partition keys and zero or more clustering key components**. The order of these components always puts the partition key first and then the clustering key.

Apart from making data unique, the partition key component of a primary key plays an additional significant role in the placement of the data. As a result, it improves the performance of reads and writes of data spread across multiple nodes in a cluster.

Components of primary key.

Partition Key

The primary goal of a partition key is to distribute the data evenly across a cluster and query the data efficiently.

A partition key is for data placement apart from uniquely identifying the data and is always the first value in the primary key definition.

Example:

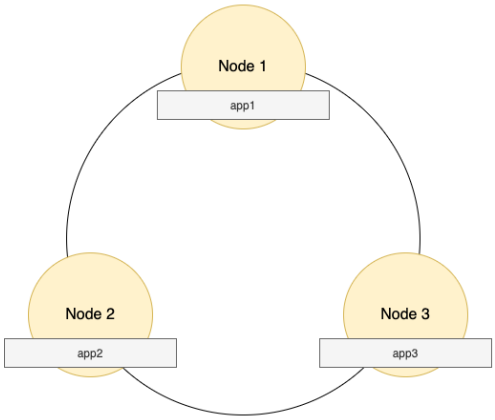
```
CREATE TABLE application_logs (
  id                    INT,
  app_name              VARCHAR,
  hostname              VARCHAR,
  log_datetime          TIMESTAMP,
  env                   VARCHAR,
  log_level             VARCHAR,
  log_message           TEXT,
  PRIMARY KEY (app_name)
);
```

Cassandra uses a consistent hashing technique to generate the hash value of the partition key (*app_name*) and assign the row data to a partition range inside a node.

Sample data:

id	app_name	env	hostname	log_datetime	log_level	log_message
1	app1	prod	host1	2021-08-15 04:05:00.000+0000	INFO	app1 INFO message
2	app2	dev	host2	2021-08-14 12:30:00.000+0000	INFO	app2 log message
3	app3	dev	host2	2021-08-13 16:45:50.000+0000	WARN	app3 log message

Let's look at possible data storage:



The above diagram is a possible scenario where the hash values of *app1*, *app2*, and *app3* resulted in each row being stored in three different nodes — *Node1*, *Node2*, and *Node3*, respectively.

All *app1* logs go to *Node1*, *app2* logs go to *Node2*, and *app3* logs go to *Node3*.
A data fetch query without a partition key in the *where* clause results in an inefficient full cluster scan.

On the other hand, with a partition key in *where* clause, Cassandra uses the consistent hashing technique to identify the exact node and the exact partition range within a node in the cluster.

Sample query for the above table:

```
select * application_logs where app_name = 'app1';
```

Composite Partition Key

If we need to combine more than one column value to form a single partition key, we use a composite partition key.

Here again, the goal of the composite partition key is for the data placement, in addition to uniquely identifying the data. As a result, the storage and retrieval of data become efficient.

Here again, the goal of the composite partition key is for the data placement, in addition to uniquely identifying the data. As a result, the storage and retrieval of data become efficient.

Here's an example of the table definition that combines the *app_name* and *env* columns to form a composite partition key:

```
CREATE TABLE application_logs (
  id                    INT,
  app_name              VARCHAR,
  hostname              VARCHAR,
  log_datetime          TIMESTAMP,
  env                   VARCHAR,
  log_level             VARCHAR,
  log_message           TEXT,
  PRIMARY KEY ((app_name, env))
);
```

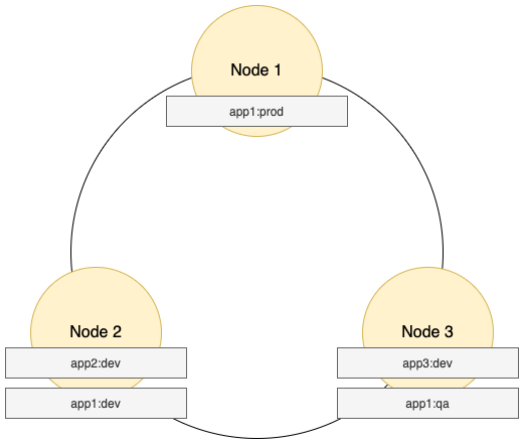
The important thing to note in the above definition is the inner parenthesis around *app_name* and *env* primary key definition. This inner parenthesis specifies that *app_name* and *env* are part of a partition key and are not clustering keys.

If we drop the inner parenthesis and have only a single parenthesis, then the *app_name* becomes the partition key, and *env* becomes the clustering key component.

Sample data

id	app_name	env	hostname	log_datetime	log_level	log_message
1	app1	prod	host1	2021-08-15 04:05:00.000+0000	INFO	app1 INFO message
2	app2	dev	host2	2021-08-14 12:30:00.000+0000	INFO	app2 log message
3	app3	dev	host2	2021-08-13 16:45:50.000+0000	WARN	app3 log message
4	app1	dev	host2	2021-08-10 11:35:50.000+0000	INFO	app1 log message
5	app1	qa	host2	2021-08-13 13:00:50.000+0000	INFO	app1 log message

Let's look at the possible data distribution of the above sample data. Please note: Cassandra generates the hash value for the *app_name* and *env* column combination:



Most importantly, **to efficiently retrieve data, the *where* clause in fetch query must contain all the composite partition keys in the same order as specified in the primary key definition**:

```
select * application_logs where app_name = 'app1' and env = 'prod';
```

Clustering Key

Partitioning is the process of identifying the partition range within a node the data is placed into. In contrast, **clustering is a storage engine process of sorting the data within a partition and is based on the columns defined as the clustering keys**.

Moreover, identification of the clustering key columns needs to be done upfront — that's because our selection of clustering key columns depends on how we want to use the data in our application.

All the data within a partition is stored in continuous storage, sorted by clustering key columns. As a result, the retrieval of the desired sorted data is very efficient.

Example:

```
CREATE TABLE application_logs (
  id                    INT,
  app_name              VARCHAR,
  hostname              VARCHAR,
  log_datetime          TIMESTAMP,
  env                   VARCHAR,
  log_level             VARCHAR,
  log_message           TEXT,
  PRIMARY KEY ((app_name, env), hostname, log_datetime)
);
```

Sample data:

id	app_name	env	hostname	log_datetime	log_level	log_message
1	app1	prod	host1	2021-08-15 04:05:00.000+0000	INFO	app1 INFO message
2	app1	prod	host1	2021-08-15 07:15:00.000+0000	DEBUG	app1 DEBUG message
3	app2	dev	host2	2021-08-14 12:30:00.000+0000	INFO	app2 log message
4	app3	dev	host2	2021-08-13 16:45:50.000+0000	WARN	app3 log message
5	app1	dev	host2	2021-08-10 11:35:50.000+0000	INFO	app1 log message
6	app1	qa	host2	2021-08-13 13:00:50.000+0000	INFO	app1 log message

By default, the Cassandra storage engine sorts the data in ascending order of clustering key columns, but **we can control the clustering columns' sort order by using *WITH CLUSTERING ORDER BY* clause in the table definition**:

```
CREATE TABLE application_logs (
  id                    INT,
  app_name              VARCHAR,
  hostname              VARCHAR,
  log_datetime          TIMESTAMP,
  env                   VARCHAR,
  log_level             VARCHAR,
  log_message           TEXT,
  PRIMARY KEY ((app_name,env), hostname, log_datetime)
)
WITH CLUSTERING ORDER BY (hostname ASC, log_datetime DESC);
```

Sample query:

```
select * application_logs
where
app_name = 'app1' and env = 'prod'
and hostname = 'host1' and log_datetime > '2021-08-13T00:00:00';
```

4. Define a keyspace, chatters with the following definition:

```
CREATE KEYSPACE chatsandra WITH REPLICATION = {'class': 'NetworkTopologyStrategy',
'replication_factor': 3};
```

5. Define each table and populate it accordingly.

```
CREATE TABLE IF NOT EXISTS users (
  email       TEXT,
  name        TEXT,
  password    TEXT,
  user_id     UUID,
  PRIMARY KEY (( email ))
);
CREATE TABLE IF NOT EXISTS posts_by_user (
  user_id     UUID,
  post_id     TIMEUUID,
  room_id     TEXT,
  text        TEXT,
  PRIMARY KEY ((user_id), post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);

CREATE TABLE IF NOT EXISTS posts_by_room (
  room_id     TEXT,
  post_id     TIMEUUID,
  user_id     UUID,
  text        TEXT,
  PRIMARY KEY ((room_id), post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);
```

Add data to users table:

users			
email	name	password	userid
otzi@mail.com	Otzi Oney	123456	11111111-1111-1111-1111-111111111111
fred@qmail.net	Fred Fivey	qwerty	55555555-5555-5555-5555-555555555555

nina@zmail.org	Nina Niney	s3cr3t	99999999-9999-9999-9999-999999999999

Add data to posts_by_users table:

posts_by_users			
user_id	post_id	room_id	Text
11111111-1111-1111-1111-111111111111	22222222-5cff-11ec-be16-1fedb0dfd057	#hiking	I climbed Mt. Gumbo yesterday ...
11111111-1111-1111-1111-111111111111	77777777-5cff-11ec-be16-1fedb0dfd057	#running	... and Mt. Gumbo was easy!!!
55555555-5555-5555-5555-555555555555	bbbbbbbb-5cff-11ec-be16-1fedb0dfd057	#hiking	For us humans Gumbo is a tough one...!
99999999-9999-9999-9999-999999999999	cccccccc-5cff-11ec-be16-1fedb0dfd057	#running	I just love marathons.
11111111-1111-1111-1111-111111111111	eeeeeeee-5cff-11ec-be16-1fedb0dfd057	#running	Same here!
55555555-5555-5555-5555-555555555555	ffffffff-5cff-11ec-be16-1fedb0dfd057	#hiking	I have to buy new boots.

Add data to posts_by_room

posts_by_room			
User_id	Post_id	Room_id	Text
11111111-1111-1111-1111-111111111111	22222222-5cff-11ec-be16-1fedb0dfd057	#hiking	I climbed Mt. Gumbo yesterday ...
11111111-1111-1111-1111-111111111111	77777777-5cff-11ec-be16-1fedb0dfd057	#running	Who likes marathons here?
11111111-1111-1111-1111-111111111111	aaaaaaaa-5cff-11ec-be16-1fedb0dfd057	#hiking	... and Mt. Gumbo was easy!!!
55555555-5555-5555-5555-555555555555	bbbbbbbb-5cff-11ec-be16-1fedb0dfd057	#hiking	For us humans Gumbo is a tough one...!

99999999-9999-9999-9999-999999999999	cccccccc-5cff-11ec-be16-1fedb0dfd057	#running	I just love marathons.
11111111-1111-1111-1111-111111111111	eeeeeeee-5cff-11ec-be16-1fedb0dfd057	#running	Same here!
55555555-5555-5555-5555-555555555555	ffffffff-5cff-11ec-be16-1fedb0dfd057	#hiking	I have to buy new boots.

6. Display all the records from posts_by_room table.
7. Show all recors from posts_by_room table for the room id #running.
8. Display post id, user id, and text from posts_by_room with room id #hiking.
9. Display post id, room id, and text from table posts_by users with user id: 99999999-9999-9999-9999-999999999999.