HIVE PROJECT

Big Data Analysis Project by Nada Inam

NADA INAM- 09826

ninam@khi.iba.edu.pk

I. Dataset Description

MovieTweetings is a dataset consisting of ratings on movies that were contained in well-structured tweets on Twitter. Since this dataset is updated regularly and is structured in different folders /latest and /snapshots, I have used the snapshot that contains 200,000 ratings. There are three files: users.dat, items.dat and ratings.dat.

1) users.dat

Contains the mapping of the users ids on their true Twitter id in the following format: userid::twitter_id. For example:

1::177651718

2) items.dat

Contains the items (i.e., movies) that were rated in the tweets, together with their genre metadata in the following format: movie_id::movie_title (movie_year)::genre|genre|genre. For example:

0110912::Pulp Fiction (1994)::Crime|Thriller

The file is UTF-8 encoded to deal with the many foreign movie titles contained in tweets.

3) ratings.dat

In this file the extracted ratings are stored in the following format: user_id::movie_id::rating::rating_timestamp. For example:

14927::0110912::9::1375657563

The ratings contained in the tweets are scaled from 0 to 10, as is the norm on the IMDb platform. To prevent information loss we have chosen to not down-scale this rating value, so all rating values of this dataset are contained in the interval [0,10].

The dataset used for this project is available on github on the following link: https://github.com/sidooms/MovieTweetings

II. Setting up Container

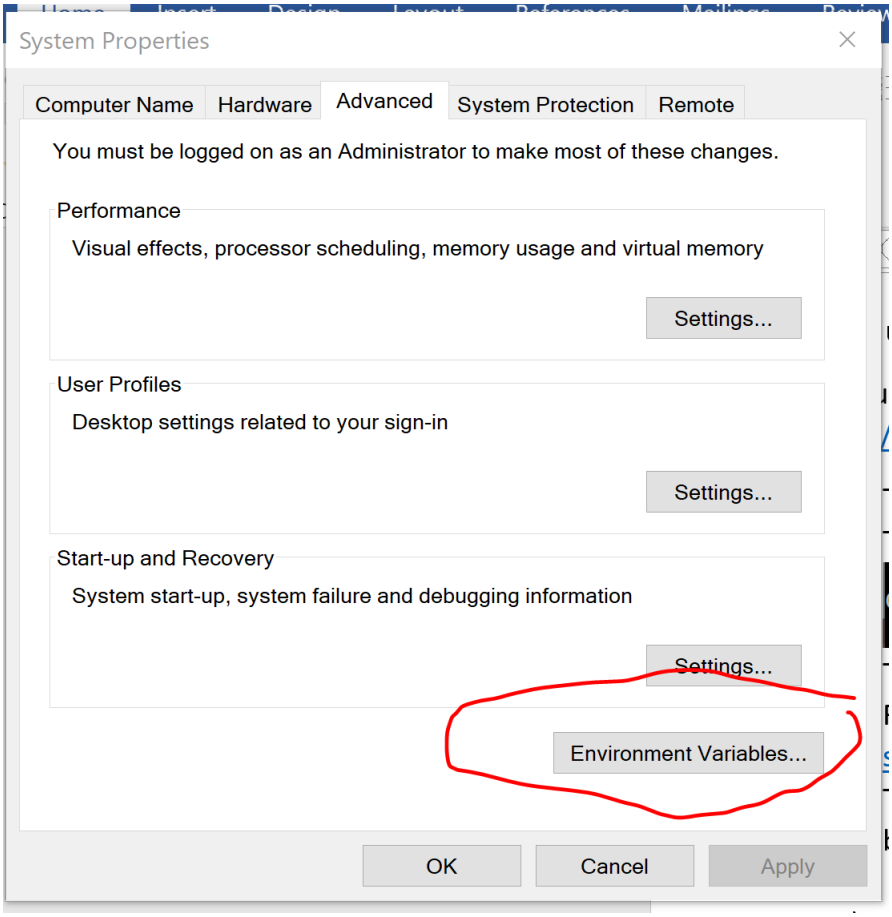
I have used the following link to set up a hive cluster: https://github.com/big-data-europe/docker-hive

1) Type command mkdir docker-hive
2) Type command cd docker-hive

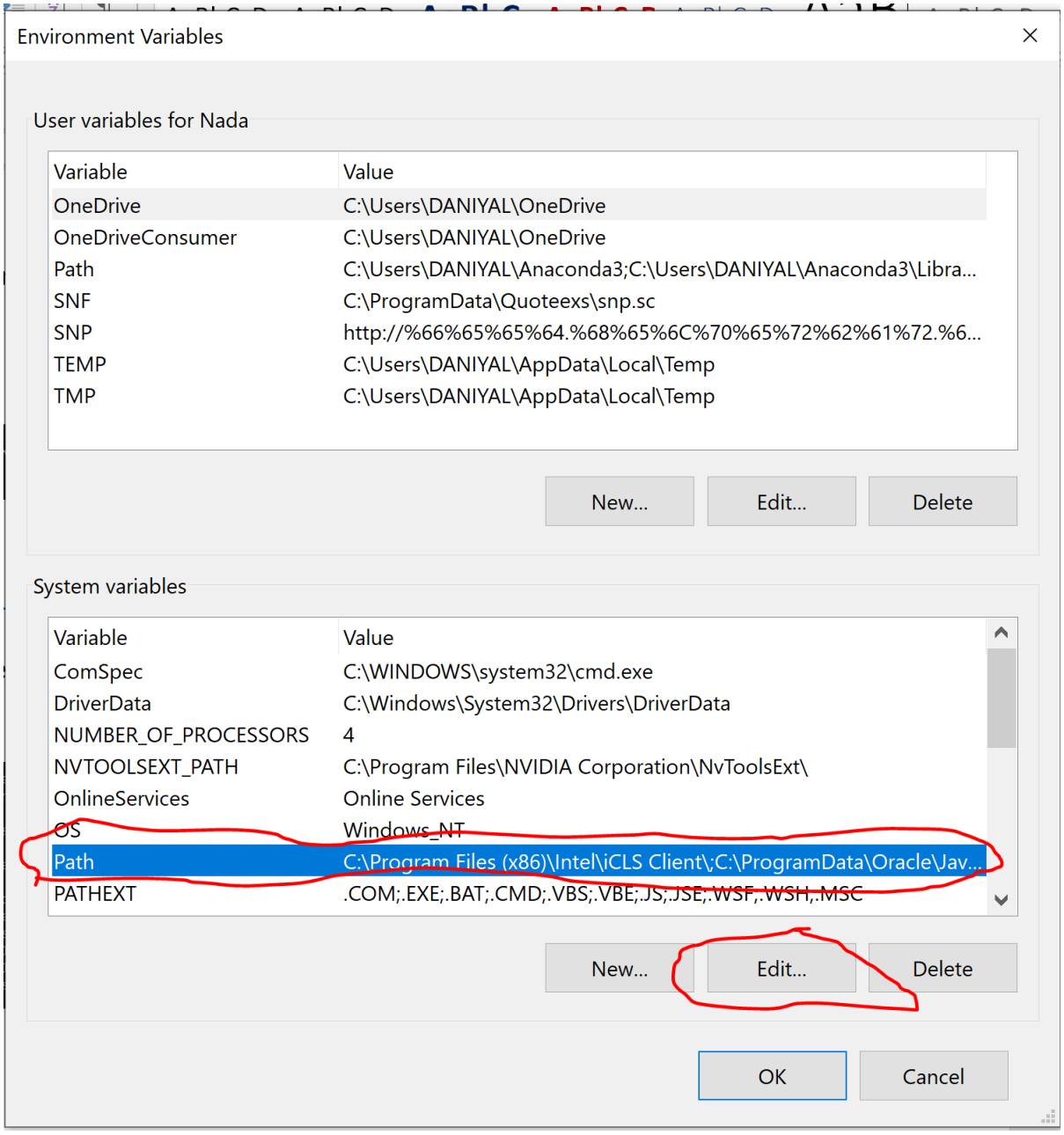
```
C:\Users\DANIYAL>cd docker-hive
```

3) Type command "git clone https://github.com/big-data-europe/docker-hive.git ."
   For this command you need to download git from this website https://git-scm.com/download/win
   Then, you need to add git to your environment variables by typing in your windows search bar "environment variables"

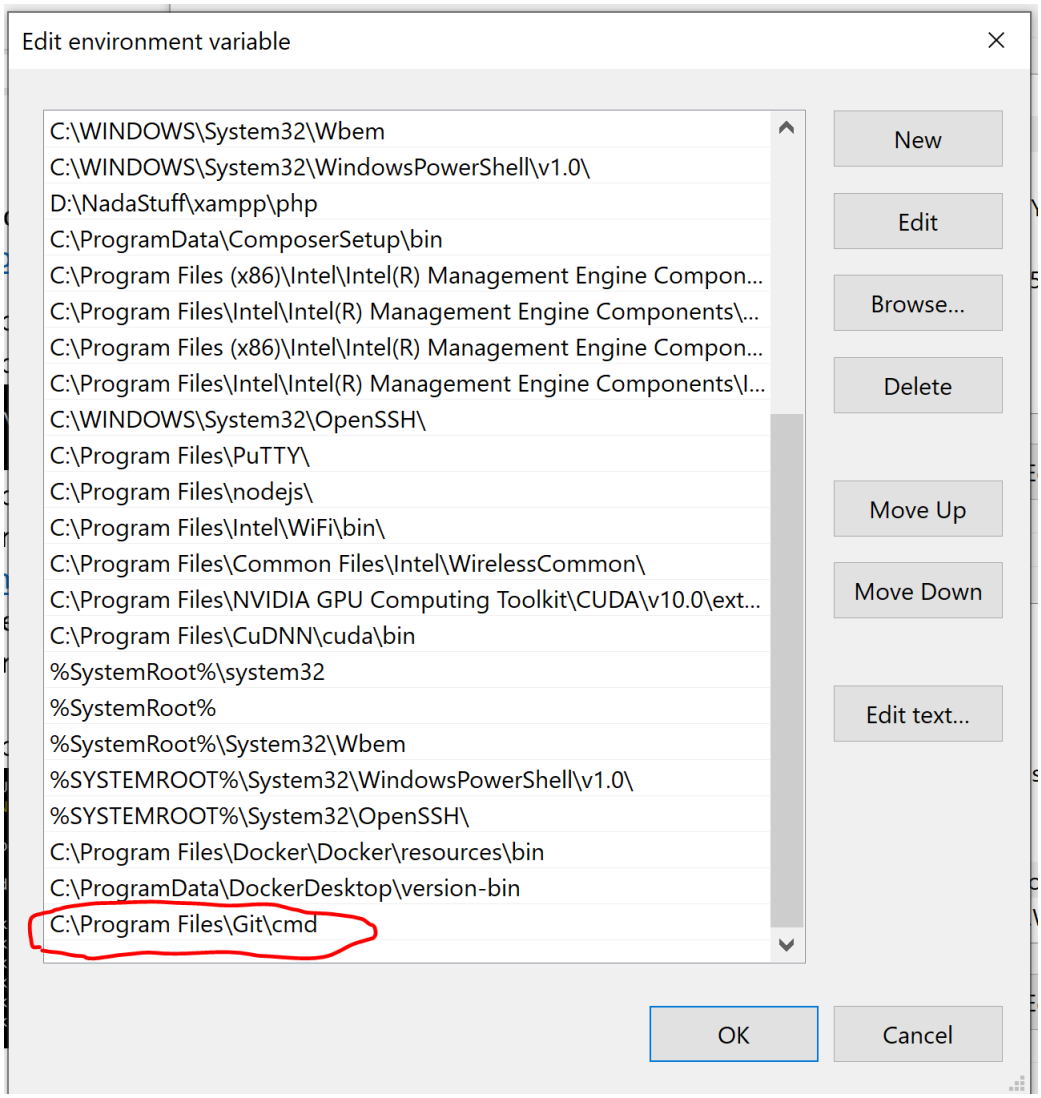
   Click on environment variables:



Click on Path and then Edit:



Finally, add the path to your Git cmd



4) Type command docker-compose up -d

```
C:\Users\DANIYAL\docker-hive>docker-compose up -d
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.

docker-hive_presto-coordinator_1 is up-to-date
docker-hive_hive-server_1 is up-to-date
docker-hive_hive-metastore_1 is up-to-date
docker-hive_namenode_1 is up-to-date
docker-hive_datanode_1 is up-to-date
docker-hive_hive-metastore-postgresql_1 is up-to-date
```

III. Loading data in container

Three files have been loaded

1) Movies: contains movie name, movie id and movie genres
2) Ratings: contains movie_id, user_id, rating and rating time stamp
3) Users: contains user id and twitter id

Before loading the files I have replaced the delimeter '::' with the delimeter ',' on all three files. The original delimiter was causing the data to be not loaded properly. The following command is use to load data in the container's local directory:

Cp <filename_withpath> <container_name>:/<path_where_files_needs_to_be_copied>

```
C:\Users\DANIYAL\docker-hive>docker cp D:\Downloads\MovieTweetings-master\MovieTweetings-master\snapshots\200K\movies.txt docker-hive_hive-se
rver_1:/opt/hive/examples/files/

C:\Users\DANIYAL\docker-hive>docker cp D:\Downloads\MovieTweetings-master\MovieTweetings-master\snapshots\200K\ratings.txt docker-hive_hive-s
erver_1:/opt/hive/examples/files/

C:\Users\DANIYAL\docker-hive>docker cp D:\Downloads\MovieTweetings-master\MovieTweetings-master\snapshots\200K\users.txt docker-hive_hive-ser
ver_1:/opt/hive/examples/files/

C:\Users\DANIYAL\docker-hive>
```

To bash into the hive server, we use the following command:

Docker exec -it <container_name> /bin/bash

After we bash into the container, we are directed to the terminal, where we can see our files have been copied to the path we defined. The file movies.txt is highlighted in the picture below.

```
C:\Users\DANIYAL\docker-hive>docker exec -it docker-hive_hive-server_1 /bin/bash
root@956bda60644a:/opt# cd hive/examples/files/
root@956bda60644a:/opt/hive/examples/files# ls
2000_cols_data.csv                  dept.txt                        kv4.txt                     smb_bucket_input.rc
3col_data.txt                       dim-data.txt                    kv5.txt                     smb_bucket_input.txt
AvroPrimitiveInList.parquet         dim_shops.txt                   kv6.txt                     smbbucket_1.rc
AvroSingleFieldGroupInList.parquet  doctors.avro                    kv7.txt                     smbbucket_1.txt
HiveGroup.parquet                   docurl.txt                      kv8.txt                     smbbucket_2.rc
HiveRequiredGroupInList.parquet     double.txt                      kv9.txt                     smbbucket_2.txt
MultiFieldGroupInList.parquet       dynamic_partition_insert.txt    leftsemijoin_mr_t1.txt      smbbucket_3.rc
NestedMap.parquet                   dynpart_test.txt                leftsemijoin_mr_t2.txt      smbbucket_3.txt
NewOptionalGroupInList.parquet      dynpartdata1.txt                lineitem.txt                smbdata.txt
NewRequiredGroupInList.parquet      dynpartdata2.txt                loc.txt                     sortdp.txt
ProxyAuth.res                       emp.txt                         location.txt                sour1.txt
SingleFieldGroupInList.parquet      emp2.txt                        lt100.sorted.txt            sour2.txt
SortCol1Col2.txt                    employee.dat                    lt100.txt                   source.txt
SortCol2Col1.txt                    employee2.dat                   lt100.txt.deflate           srcbucket0.txt
SortDescCol1Col2.txt                employee_part.txt               mapNull.txt                 srcbucket1.txt
SortDescCol2Col1.txt                empty1.txt                      map_null_schema.avro        srcbucket20.txt
StringMapOfOptionalIntArray.parquet empty2.txt                      map_null_val.avro           srcbucket21.txt
T1.txt                              encoding-utf8.txt               map_table.txt               srcbucket22.txt
T2.txt                              encoding_iso-8859-1.txt         movies.txt                  srcbucket23.txt
```

Next, we need to connect to the hive server and we are connecting on the port 10000, the command to connect to hive server through beeline is:

/opt/hive/bin/beeline -u jdbc:hive2://localhost:10000

After executing the above command, we can see we are connected to Apache Hive version 2.3.2

```
root@956bda60644a:/opt# /opt/hive/bin/beeline -u jdbc:hive2://localhost:10000
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-2.7.4/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/Sta
]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Connecting to jdbc:hive2://localhost:10000
Connected to: Apache Hive (version 2.3.2)
Driver: Hive JDBC (version 2.3.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 2.3.2 by Apache Hive
0: jdbc:hive2://localhost:10000>
```

Earlier we loaded our text files in our container's local path, now we need to

1) Create table structure
2) Load text files in the tables

The syntax to create a table in Hive is:

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name

[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]
```

Below, you can see that we have created three tables with their respective column names, row format such as specifying the delimiter by which the fields are separated and the delimiter by which the lines are terminated.

```
Beeline version 2.3.2 by Apache Hive
0: jdbc:hive2://localhost:10000> CREATE TABLE movies (movie_id STRING, movie_title STRING, movie_genres STRING) ROW FORMAT
 DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE;
No rows affected (0.18 seconds)
0: jdbc:hive2://localhost:10000> CREATE TABLE ratings (user_id STRING, movie_id STRING, rating STRING, rating_timestamp ST
RING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE;
No rows affected (0.098 seconds)
0: jdbc:hive2://localhost:10000> CREATE TABLE users (user_id STRING, twitter_id STRING) ROW FORMAT DELIMITED FIELDS TERMIN
ATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE;
No rows affected (0.123 seconds)
0: jdbc:hive2://localhost:10000>
```

After creating the table, we load the text files in each table.

The syntax to load data from local path to a table in hive is:

LOAD DATA LOCAL INPATH '<path_name_along_with_file_name>' INTO TABLE <table_name>;

```
0: jdbc:hive2://localhost:10000> LOAD DATA LOCAL INPATH '/opt/hive/examples/files/movies.txt' INTO TABLE movies;
No rows affected (1.344 seconds)
0: jdbc:hive2://localhost:10000> LOAD DATA LOCAL INPATH '/opt/hive/examples/files/ratings.txt' INTO TABLE ratings;
No rows affected (0.936 seconds)
0: jdbc:hive2://localhost:10000> LOAD DATA LOCAL INPATH '/opt/hive/examples/files/users.txt' INTO TABLE users;
```

Below, you can see we have successfully loaded movies text file into the movies table.

```
0: jdbc:hive2://localhost:10000> LOAD DATA LOCAL INPATH '/opt/hive/examples/files/movies.txt' INTO TABLE movies;
No rows affected (0.617 seconds)
0: jdbc:hive2://localhost:10000> select * from movies limit 10;
+-----------------+-----------------------------------------+-----------------------------+
| movies.movie_id |             movies.movie_title          |     movies.movie_genres     |
+-----------------+-----------------------------------------+-----------------------------+
| 0000091         | Le manoir du diable (1896)              | Short|Horror               |
| 0000628         | The Adventures of Dollie (1908)         | Action|Short               |
| 0000833         | The Country Doctor (1909)               | Short|Drama                |
| 0001223         | Frankenstein (1910)                     | Short|Horror|Sci-Fi        |
| 0001740         | The Lonedale Operator (1911)            | Short|Drama|Romance|Western |
| 0002844         | Fant?mas - ? l'ombre de la guillotine (1913) | Crime|Drama            |
| 0004936         | The Bank (1915)                         | Comedy|Short               |
| 0004972         | The Birth of a Nation (1915)            | Drama|History|Romance|War  |
| 0005078         | The Cheat (1915)                        | Drama                       |
| 0006684         | The Fireman (1916)                      | Short|Comedy               |
+-----------------+-----------------------------------------+-----------------------------+
10 rows selected (0.355 seconds)
0: jdbc:hive2://localhost:10000>
```

Below, you can see we have successfully loaded ratings text file into the ratings table.

```
0: jdbc:hive2://localhost:10000> LOAD DATA LOCAL INPATH '/opt/hive/examples/files/ratings.txt' INTO TABLE ratings;
No rows affected (0.474 seconds)
0: jdbc:hive2://localhost:10000> select * from ratings limit 10;
+-----------------+-----------------+----------------+--------------------------+
| ratings.user_id | ratings.movie_id | ratings.rating | ratings.rating_timestamp |
+-----------------+-----------------+----------------+--------------------------+
| 1               | 0068646         | 10             | 1381620027               |
| 1               | 0113277         | 10             | 1379466669               |
| 2               | 0790636         | 7              | 1389963947               |
| 2               | 0816711         | 8              | 1379963769               |
| 2               | 1091191         | 7              | 1391173869               |
| 2               | 1322269         | 7              | 1391529691               |
| 2               | 1433811         | 8              | 1380453043               |
| 2               | 1454468         | 8              | 1387016442               |
| 2               | 1535109         | 8              | 1386350135               |
| 2               | 1798709         | 10             | 1389948338               |
+-----------------+-----------------+----------------+--------------------------+
10 rows selected (0.261 seconds)
0: jdbc:hive2://localhost:10000>
```

Below, you can see we have successfully loaded users text file into the users table.

```
0: jdbc:hive2://localhost:10000> LOAD DATA LOCAL INPATH '/opt/hive/examples/files/users.txt' INTO TABLE users;
No rows affected (0.86 seconds)
0: jdbc:hive2://localhost:10000> select * from users limit 10;
+----------------+------------------+
| users.user_id  | users.twitter_id |
+----------------+------------------+
| 1              | 995885060        |
| 2              | 40501255         |
| 3              | 138805259        |
| 4              | 47317010         |
| 5              | 84541461         |
| 6              | 2278948890       |
| 7              | 1007157282       |
| 8              | 14680111         |
| 9              | 153878584        |
| 10             | 72351811         |
+----------------+------------------+
10 rows selected (0.23 seconds)
0: jdbc:hive2://localhost:10000>
```

IV. Data Analysis

1) Finding total unique movies for which user's have given their ratings: 14732 movies

Here, we use the aggregation function count() which counts the number of rows. Also, we use the function distinct() which retrieves unique values for a particular column.

```
0: jdbc:hive2://localhost:10000> select count(distinct(movies.movie_id)) from movies;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
ecution engine (i.e. spark, tez) or using Hive 1.X releases.
+--------+
|  _c0   |
+--------+
| 14732  |
+--------+
1 row selected (1.493 seconds)
```

2) Finding the total unique number of movies for which we have ratings: 14732 movies

This means we have around 14732 movies in the movies table and 14732 movies in the ratings table.

```
0: jdbc:hive2://localhost:10000> select count(distinct(ratings.movie_id)) from ratings;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different ex
ecution engine (i.e. spark, tez) or using Hive 1.X releases.
+--------+
|  _c0   |
+--------+
| 14732  |
+--------+
1 row selected (1.531 seconds)
```

3) Finding total number of ratings: 200,000 ratings

```
0: jdbc:hive2://localhost:10000> select count(*) from ratings;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
ecution engine (i.e. spark, tez) or using Hive 1.X releases.
+---------+
|   _c0   |
+---------+
| 200000  |
+---------+
1 row selected (1.571 seconds)
0: jdbc:hive2://localhost:10000>
```

4) Finding Top 10 movie ids

Here we use sum() aggregation function and cast() function to cast the rating column from string to INT. Also, we group by in order to calculate sum for each movie_id, and orderby to order the sum of ratings in a descending order.

```
0: jdbc:hive2://localhost:10000> select movie_id, sum(cast(ratings.rating as int)) sum_of_ratings from ratings group by movie_id
order by sum_of_ratings desc limit 10;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution
 engine (i.e. spark, tez) or using Hive 1.X releases.
+-----------+-----------------+
| movie_id  | sum_of_ratings  |
+-----------+-----------------+
| 1454468   | 19270           |
| 0770828   | 18063           |
| 1300854   | 16256           |
| 1670345   | 15173           |
| 0816711   | 15169           |
| 1408101   | 14181           |
| 0993846   | 13226           |
| 1535109   | 12141           |
| 1343092   | 11501           |
| 1392214   | 11241           |
+-----------+-----------------+
10 rows selected (2.889 seconds)
```

5) Finding Total number of unique users: 25011 users

```
0: jdbc:hive2://localhost:10000> select count(distinct(user_id)) from users;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider
 engine (i.e. spark, tez) or using Hive 1.X releases.
+--------+
|  _c0   |
+--------+
| 25011  |
+--------+
1 row selected (1.517 seconds)
0: jdbc:hive2://localhost:10000>
```

6) Finding the movie id with maximum number of ratings:

```
0: jdbc:hive2://localhost:10000> select movie_id, count(cast(rating as int)) as Total_ratings from ratings group by movie_id
 order by Total_ratings desc limit 1;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different exec
ution engine (i.e. spark, tez) or using Hive 1.X releases.
+-----------+---------------+
| movie_id  | total_ratings |
+-----------+---------------+
| 0770828   | 2336          |
+-----------+---------------+
1 row selected (3.637 seconds)
0: jdbc:hive2://localhost:10000>
```

7) Finding the user_id with the maximum number of ratings:

```
0: jdbc:hive2://localhost:10000> select user_id, count(cast(rating as int)) as Total_ratings from ratings group by user_id o
rder by Total_ratings desc limit 1;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different exec
ution engine (i.e. spark, tez) or using Hive 1.X releases.
+----------+---------------+
| user_id  | total_ratings  |
+----------+---------------+
| 9264     | 575            |
+----------+---------------+
1 row selected (2.884 seconds)
```

8) Finding the value of maximum rating that a movie has: 10

IMBD allows user to rate a movie on a scale of 0 to 10

```
0: jdbc:hive2://localhost:10000> select MAX(cast(ratings.rating as int)) as max_rating from ratings;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different exec
ution engine (i.e. spark, tez) or using Hive 1.X releases.
+-------------+
| max_rating  |
+-------------+
| 10          |
+-------------+
1 row selected (1.487 seconds)
```

9) Finding the value of minimum rating that a movie has: 0

```
0: jdbc:hive2://localhost:10000> select MIN(cast(ratings.rating as int)) as min_rating from ratings;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different exec
ution engine (i.e. spark, tez) or using Hive 1.X releases.
+-------------+
| min_rating  |
+-------------+
| 0           |
+-------------+
1 row selected (1.449 seconds)
```

10) Finding the id and name of the movies that have a rating greater than 8:

We would need to perform a join in order to get both movie and ratings data. For this we create a view that contains all the data of movies that have a rating greater than 8.

Below we create a view, perform left join and retrieve only those rows that have rating greater than 8.

```
0: jdbc:hive2://localhost:10000> CREATE VIEW movies_with_ratings_greater_than_8 AS Select r.movie_id, m.movie_title, r.rating,
 r.user_id from ratings r LEFT JOIN movies m ON r.movie_id = m.movie_id where cast(r.rating as int) > 8;
No rows affected (0.243 seconds)
```

On performing select over the view we have created; we can see now we have all the movies with ratings greater than 8.

```
0: jdbc:hive2://localhost:10000> Select * from movies_with_ratings_greater_than_8 limit 10;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-2.7.4/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Execution log at: /tmp/root/root_20220601181317_37627469-5830-4e15-8b97-f84fa6c90e03.log
2022-06-01 18:13:22     Starting to launch local task to process map join;      maximum memory = 477626368
2022-06-01 18:13:24     Dump the side-table for tag: 1 with group count: 14732 into file: file:/tmp/root/35ed2748-555b-4f25-8154-25763bc0b1a1/hive_2022-06-01_18-13-17_061_1908747312584907184-5
/-local-10004/HashTable-Stage-3/MapJoin-mapfile01--.hashtable
2022-06-01 18:13:24     Uploaded 1 File to: file:/tmp/root/35ed2748-555b-4f25-8154-25763bc0b1a1/hive_2022-06-01_18-13-17_061_1908747312584907184-5/-local-10004/HashTable-Stage-3/MapJoin-mapfil
e01--.hashtable (728944 bytes)
2022-06-01 18:13:24     End of local task; Time Taken: 1.798 sec.
+-----------------------------------------------+-----------------------------------------------+------------------------------------------------+-----------------------------------------------+
| movies_with_ratings_greater_than_8.movie_id   | movies_with_ratings_greater_than_8.movie_title | movies_with_ratings_greater_than_8.rating  | movies_with_ratings_greater_than_8.user_id  |
+-----------------------------------------------+-----------------------------------------------+------------------------------------------------+-----------------------------------------------+
| 0068646                                       | The Godfather (1972)                          | 10                                         | 1                                             |
| 0113277                                       | Heat (1995)                                   | 10                                         | 1                                             |
| 1798709                                       | Her (2013)                                    | 10                                         | 2                                             |
| 2024544                                       | 12 Years a Slave (2013)                       | 10                                         | 2                                             |
| 1790885                                       | Zero Dark Thirty (2012)                       | 9                                          | 4                                             |
| 0385002                                       | Hooligans (2005)                              | 10                                         | 5                                             |
| 1220198                                       | The Fourth Kind (2009)                        | 10                                         | 5                                             |
| 1462900                                       | Yi dai zong shi (2013)                        | 10                                         | 5                                             |
| 1512685                                       | Los ojos de Julia (2010)                      | 10                                         | 5                                             |
| 1631707                                       | Enter Nowhere (2011)                          | 10                                         | 5                                             |
+-----------------------------------------------+-----------------------------------------------+------------------------------------------------+-----------------------------------------------+
10 rows selected (9.236 seconds)
0: jdbc:hive2://localhost:10000>
```

Dropping the view, we just created:

```
0: jdbc:hive2://localhost:10000> DROP VIEW movies_with_ratings_greater_than_8;
No rows affected (0.112 seconds)
0: jdbc:hive2://localhost:10000>
```

Create a copy of the Table users, with the name users_twitter:

Here I use the Create Table <new_table> LIKE <original_table> which copies the entire physical structure of the original table without copying any rows, therefore this table would be empty.

In order to copy the rows we can then use INSERT INTO <new_table> SELECT * FROM <original_table>

Now, we compare the counts of both the tables and we can see the count matches, hence we successfully created a copy.

```
0: jdbc:hive2://localhost:10000> CREATE TABLE users_twitter LIKE users;
No rows affected (0.162 seconds)
0: jdbc:hive2://localhost:10000> INSERT INTO users_twitter SELECT * FROM users;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark
, tez) or using Hive 1.X releases.
No rows affected (1.988 seconds)
0: jdbc:hive2://localhost:10000> Select count(*) from users;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark
, tez) or using Hive 1.X releases.
+--------+
|  _c0   |
+--------+
| 25011  |
+--------+
1 row selected (1.555 seconds)
0: jdbc:hive2://localhost:10000> Select count(*) from users_twitter;
+--------+
|  _c0   |
+--------+
| 25011  |
+--------+
1 row selected (0.2 seconds)
0: jdbc:hive2://localhost:10000>
```

11) Rename the table users_twitter table to the new name twitter_users:

```
0: jdbc:hive2://localhost:10000> ALTER TABLE users_twitter RENAME TO twitter_users;
No rows affected (0.243 seconds)
0: jdbc:hive2://localhost:10000> SELECT * FROM twitter_users limit 1;
+-------------------------+----------------------------+
| twitter_users.user_id   | twitter_users.twitter_id   |
+-------------------------+----------------------------+
| 1                       | 995885060                  |
+-------------------------+----------------------------+
1 row selected (0.15 seconds)
```

12) Add a new column to table twitter_users names user_name:

```
0: jdbc:hive2://localhost:10000> ALTER TABLE twitter_users ADD COLUMNS (user_name STRING);
No rows affected (0.133 seconds)
0: jdbc:hive2://localhost:10000> Select * from twitter_users limit 1;
+-------------------------+----------------------------+--------------------------+
| twitter_users.user_id   | twitter_users.twitter_id   | twitter_users.user_name  |
+-------------------------+----------------------------+--------------------------+
| 1                       | 995885060                  | NULL                     |
+-------------------------+----------------------------+--------------------------+
1 row selected (0.132 seconds)
```

13) Change the name of the column user_name to twitter_username, for the table twitter_users:

```
0: jdbc:hive2://localhost:10000> ALTER TABLE twitter_users CHANGE user_name twitter_username STRING;
No rows affected (0.119 seconds)
0: jdbc:hive2://localhost:10000> Select * from twitter_users limit 1;
+-------------------------+----------------------------+--------------------------------+
| twitter_users.user_id   | twitter_users.twitter_id   | twitter_users.twitter_username |
+-------------------------+----------------------------+--------------------------------+
| 1                       | 995885060                  | NULL                           |
+-------------------------+----------------------------+--------------------------------+
1 row selected (0.166 seconds)
0: jdbc:hive2://localhost:10000>
```

14) Delete column twitter_username

In hive, there is no syntax such as drop column like in mssql/mysql etc. Therefore, to drop a column we can use REPLACE COLUMNS, this will keep only the columns specified and remove any that have not been specified between the round parenthesis.

```
0: jdbc:hive2://localhost:10000> ALTER TABLE twitter_users REPLACE COLUMNS (user_id STRING, twitter_id STRING);
No rows affected (0.084 seconds)
0: jdbc:hive2://localhost:10000> Select * from twitter_users limit 2;
+-----------------------+-------------------------+
| twitter_users.user_id | twitter_users.twitter_id |
+-----------------------+-------------------------+
| 1                     | 995885060               |
| 2                     | 40501255                |
+-----------------------+-------------------------+
2 rows selected (0.199 seconds)
0: jdbc:hive2://localhost:10000>
```

15)Show all databases

```
0: jdbc:hive2://localhost:10000> SHOW DATABASES;
+----------------+
| database_name  |
+----------------+
| default        |
+----------------+
1 row selected (0.04 seconds)
```

16)Show all Tables

```
0: jdbc:hive2://localhost:10000> SHOW TABLES;
+----------------+
|    tab_name    |
+----------------+
| movies         |
| pokes          |
| ratings        |
| twitter_users  |
| users          |
+----------------+
5 rows selected (0.099 seconds)
```

17)Drop the table named pokes:

```
0: jdbc:hive2://localhost:10000> DROP TABLE IF EXISTS pokes;
No rows affected (0.179 seconds)
0: jdbc:hive2://localhost:10000> SHOW TABLES;
+----------------+
|    tab_name    |
+----------------+
| movies         |
| ratings        |
| twitter_users  |
| users          |
+----------------+
4 rows selected (0.034 seconds)
0: jdbc:hive2://localhost:10000>
```

18) Create index on the column movie_id from the table ratings

```
0: jdbc:hive2://localhost:10000> CREATE INDEX movie_id_index ON TABLE ratings(movie_id) AS 'org.apache.hadoop.hive.ql.index.compact.
CompactIndexHandler' WITH DEFERRED REBUILD;
No rows affected (0.583 seconds)
```

19) Drop the index create

```
0: jdbc:hive2://localhost:10000> DROP INDEX movie_id_index ON ratings;
No rows affected (0.194 seconds)
0: jdbc:hive2://localhost:10000>
```

20) Find the average rating for the movie_id 1454468 round to 1 decimal place:

```
0: jdbc:hive2://localhost:10000> select round(avg(cast(rating as INT)), 2) from ratings where movie_id = '1454468';
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution en
gine (i.e. spark, tez) or using Hive 1.X releases.
+------+
| _c0  |
+------+
| 8.4  |
+------+
1 row selected (1.521 seconds)
```

Creating an index speeds up query processing time when performing a join, create index on user_id column, for both tables ratings and users and perform join. Find if there is a difference of processing speed after creating an index and before creating an index:

Below is a temporary table rating_with_user that contains all ratings information along with the user, this left join took a total of 8.684 seconds.

```
0: jdbc:hive2://localhost:10000> CREATE TEMPORARY TABLE ratings_with_user AS Select r.rating, r.movie_id, c.user_id from ratings r l
eft join users c ON r.user_id = c.user_id;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution en
gine (i.e. spark, tez) or using Hive 1.X releases.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-2.7.4/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBin
der.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Execution log at: /tmp/root/root_20220601200836_8d17ea77-7a37-4654-adc1-94b80ba7aadf.log
2022-06-01 20:08:41     Starting to launch local task to process map join;      maximum memory = 477626368
2022-06-01 20:08:43     Dump the side-table for tag: 1 with group count: 25011 into file: file:/tmp/root/35ed2748-555b-4f25-8154-257
63bc0b1a1/hive_2022-06-01_20-08-36_857_3680510304880631418-5/-local-10004/HashTable-Stage-4/MapJoin-mapfile21--.hashtable
2022-06-01 20:08:43     Uploaded 1 File to: file:/tmp/root/35ed2748-555b-4f25-8154-25763bc0b1a1/hive_2022-06-01_20-08-36_857_3680510
304880631418-5/-local-10004/HashTable-Stage-4/MapJoin-mapfile21--.hashtable (567160 bytes)
2022-06-01 20:08:43     End of local task; Time Taken: 1.708 sec.
No rows affected (8.684 seconds)
```

Below is a temporary table rating_with_user_2 that contains all ratings information along with the user, after index has been created over user_id for both tables, this left join took a total of 8.564 seconds.

This means that after creating index on a column, query processing time does decrease.

```
0: jdbc:hive2://localhost:10000> CREATE INDEX user_id_index ON TABLE ratings(user_id) AS 'org.apache.hadoop.hive.ql.index.compact.Co
mpactIndexHandler' WITH DEFERRED REBUILD;
No rows affected (0.528 seconds)
0: jdbc:hive2://localhost:10000> CREATE INDEX user_id_index ON TABLE users(user_id) AS 'org.apache.hadoop.hive.ql.index.compact.Comp
actIndexHandler' WITH DEFERRED REBUILD;
No rows affected (0.294 seconds)
0: jdbc:hive2://localhost:10000> CREATE TEMPORARY TABLE ratings_with_user_2 AS Select r.rating, r.movie_id, c.user_id from ratings r
 left join users c ON r.user_id = c.user_id;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution en
gine (i.e. spark, tez) or using Hive 1.X releases.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-2.7.4/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBin
der.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Execution log at: /tmp/root/root_20220601201016_0c499de7-d0fa-4f3d-be42-c87838b95791.log
2022-06-01 20:10:21     Starting to launch local task to process map join;      maximum memory = 477626368
2022-06-01 20:10:22     Dump the side-table for tag: 1 with group count: 25011 into file: file:/tmp/root/35ed2748-555b-4f25-8154-257
63bc0b1a1/hive_2022-06-01_20-10-16_214_1450672744608293078-5/-local-10004/HashTable-Stage-4/MapJoin-mapfile31--.hashtable
2022-06-01 20:10:22     Uploaded 1 File to: file:/tmp/root/35ed2748-555b-4f25-8154-25763bc0b1a1/hive_2022-06-01_20-10-16_214_1450672
744608293078-5/-local-10004/HashTable-Stage-4/MapJoin-mapfile31--.hashtable (567160 bytes)
2022-06-01 20:10:22     End of local task; Time Taken: 1.763 sec.
No rows affected (8.564 seconds)
0: jdbc:hive2://localhost:10000>
```

21)Calculate average rating per user:

```
0: jdbc:hive2://localhost:10000> Select user_id, round(avg(cast(rating as int)),2) Average_rating from ratings_with_user_2 group by
user_id limit 10;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution en
gine (i.e. spark, tez) or using Hive 1.X releases.
+----------+-----------------+
| user_id  | average_rating  |
+----------+-----------------+
| 1        | 10.0            |
| 10       | 6.7             |
| 100      | 8.0             |
| 1000     | 8.5             |
| 10000    | 9.0             |
| 10001    | 8.0             |
| 10002    | 6.67            |
| 10003    | 10.0            |
| 10004    | 7.0             |
| 10005    | 7.67            |
+----------+-----------------+
10 rows selected (1.443 seconds)
0: jdbc:hive2://localhost:10000>
```

22)Calculate average rating per movie:

For this, first we need to perform a join between movies and ratings and store the rows in a table. Next, we can calculate the average rating using AVG() aggregate function and group by over movie_id

```
0: jdbc:hive2://localhost:10000> CREATE TEMPORARY TABLE ratings_with_movies AS Select r.rating, r.movie_id, c.movie_title from ratin
gs r left join movies c ON r.movie_id = c.movie_id;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution en
gine (i.e. spark, tez) or using Hive 1.X releases.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-2.7.4/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBin
der.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Execution log at: /tmp/root/root_20220601204610_77ecea7d-7a87-4691-8142-ad4814b02d05.log
2022-06-01 20:46:15     Starting to launch local task to process map join;      maximum memory = 477626368
2022-06-01 20:46:17     Dump the side-table for tag: 1 with group count: 14732 into file: file:/tmp/root/35ed2748-555b-4f25-8154-257
63bc0b1a1/hive_2022-06-01_20-46-10_781_7917927722610296124-5/-local-10004/HashTable-Stage-4/MapJoin-mapfile41--.hashtable
2022-06-01 20:46:17     Uploaded 1 File to: file:/tmp/root/35ed2748-555b-4f25-8154-25763bc0b1a1/hive_2022-06-01_20-46-10_781_7917927
722610296124-5/-local-10004/HashTable-Stage-4/MapJoin-mapfile41--.hashtable (728944 bytes)
2022-06-01 20:46:17     End of local task; Time Taken: 1.691 sec.
No rows affected (9.77 seconds)
```

```
0: jdbc:hive2://localhost:10000> Select movie_title, round(avg(cast(rating as int)),2) Average_rating from ratings_with_movies group
 by movie_title limit 10;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution en
gine (i.e. spark, tez) or using Hive 1.X releases.
+-------------------------------------------+-----------------+
|                movie_title                | average_rating  |
+-------------------------------------------+-----------------+
| $ (1971)                                  | 8.0             |
| $5 a Day (2008)                           | 7.0             |
| $50K and a Call Girl: A Love Story (2014) | 5.5             |
| $ellebrity (2012)                         | 4.0             |
| 'Breaker' Morant (1980)                   | 8.5             |
| 'Crocodile' Dundee II (1988)              | 7.0             |
| 'Hukkunud Alpinisti' hotell (1979)        | 10.0            |
| 'I Know Where I'm Going!' (1945)          | 9.0             |
| 'Neitzsche' Ate Here (2013)               | 10.0            |
| *batteries not included (1987)            | 6.67            |
+-------------------------------------------+-----------------+
10 rows selected (1.496 seconds)
0: jdbc:hive2://localhost:10000>
```

23) Create table with users and the number of movies for which they have given their ratings:

```
0: jdbc:hive2://localhost:10000> Create temporary table user_rating_info AS Select user_id, count(cast(rating as INT)) Total_ratings from
 ratings group by user_id;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine
(i.e. spark, tez) or using Hive 1.X releases.
No rows affected (1.632 seconds)
0: jdbc:hive2://localhost:10000> Select * from user_rating_info limit 2;
+--------------------------+-------------------------------+
| user_rating_info.user_id  | user_rating_info.total_ratings  |
+--------------------------+-------------------------------+
| 1                        | 2                             |
| 10                       | 10                            |
+--------------------------+-------------------------------+
2 rows selected (0.198 seconds)
```

24) Show the columns and data types of the table:

```
0: jdbc:hive2://localhost:10000> describe user_rating_info;
+-----------------+------------+----------+
|    col_name     | data_type  | comment  |
+-----------------+------------+----------+
| user_id         | string     |          |
| total_ratings   | bigint     |          |
+-----------------+------------+----------+
2 rows selected (0.025 seconds)
```

25) Calculate total number of users with a minimum rating of 50 and calculate total number of users with a minimum rating of 10:

```
0: jdbc:hive2://localhost:10000> Select count(user_id) from user_rating_info where total_ratings <= 50;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine
(i.e. spark, tez) or using Hive 1.X releases.
+--------+
|  _c0   |
+--------+
| 24344  |
+--------+
1 row selected (1.493 seconds)
0: jdbc:hive2://localhost:10000> Select count(user_id) from user_rating_info where total_ratings <= 10;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine
(i.e. spark, tez) or using Hive 1.X releases.
+--------+
|  _c0   |
+--------+
| 20524  |
+--------+
1 row selected (1.429 seconds)
0: jdbc:hive2://localhost:10000>
```

26) Find total number of action movies:

There are two ways to match a substring, one is using LIKE and other is using LOCATE() function.

```
0: jdbc:hive2://localhost:10000> select count(distinct(movie_id)) from movies where movie_genres like '%Action%';
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
execution engine (i.e. spark, tez) or using Hive 1.X releases.
+-------+
|  _c0  |
+-------+
| 2199  |
+-------+
1 row selected (1.416 seconds)
0: jdbc:hive2://localhost:10000> select count(distinct(movie_id)) from movies where locate('Action', movie_genres)>0;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
execution engine (i.e. spark, tez) or using Hive 1.X releases.
+-------+
|  _c0  |
+-------+
| 2199  |
+-------+
1 row selected (1.396 seconds)
0: jdbc:hive2://localhost:10000>
```

27) Find total number of movies with genre both action and Thriller:

```
0: jdbc:hive2://localhost:10000> select count(distinct(movie_id)) from movies where locate('Action', movie_genres)>0 and
 locate('Thriller', movie_genres)>0;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
execution engine (i.e. spark, tez) or using Hive 1.X releases.
+------+
| _c0  |
+------+
| 987  |
+------+
1 row selected (1.422 seconds)
0: jdbc:hive2://localhost:10000>
```

28) Find the total rating and average rating for movies which have genre as both action and thriller:

Here I am using left join to gather both ratings information and movies, then in the where clause I am filtering out rows to get only those movies & ratings for which the movie genre is both action and thriller (using locate() function).

I am using count() and avg() functions to calculate total and average rating respectively. Then I use a group by to gather these stats.

I used the round() function which returns a decimal place to any desired decimal place we specify, here I used 2.

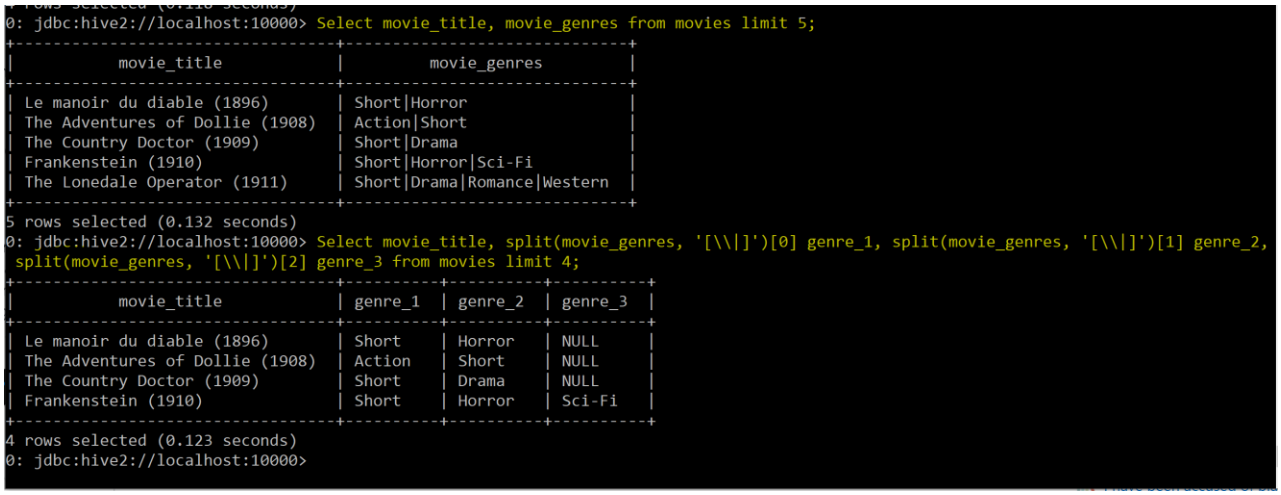
```
0: jdbc:hive2://localhost:10000> select m.movie_title, r.movie_id, count(r.rating) Total_ratings, round(AVG(cast(r.rating as INT)),2) Average_
rating from ratings r left join movies m ON r.movie_id = m.movie_id where locate('Action', m.movie_genres)>0 and locate('Thriller', m.movie_ge
nres)>0 group by m.movie_title, r.movie_id limit 10;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e.
 spark, tez) or using Hive 1.X releases.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-2.7.4/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Execution log at: /tmp/root/root_20220602105031_d9bac134-2c2f-4d1d-8d4d-63f139ac8074.log
2022-06-02 10:50:36     Starting to launch local task to process map join;      maximum memory = 477626368
2022-06-02 10:50:37     Dump the side-table for tag: 1 with group count: 14732 into file: file:/tmp/root/35ed2748-555b-4f25-8154-25763bc0b1a1/
hive_2022-06-02_10-50-31_692_2239503900129721460-5/-local-10005/HashTable-Stage-2/MapJoin-mapfile81--.hashtable
2022-06-02 10:50:38     Uploaded 1 File to: file:/tmp/root/35ed2748-555b-4f25-8154-25763bc0b1a1/hive_2022-06-02_10-50-31_692_22395039001297214
60-5/-local-10005/HashTable-Stage-2/MapJoin-mapfile81--.hashtable (989283 bytes)
2022-06-02 10:50:38     End of local task; Time Taken: 1.824 sec.
+-------------------------------------+-------------+----------------+-----------------+
|            m.movie_title            | r.movie_id  | total_ratings  | average_rating  |
+-------------------------------------+-------------+----------------+-----------------+
| 12 Rounds (2009)                    | 1160368     | 8              | 5.63            |
| 13 Rue Madeleine (1947)             | 0038279     | 2              | 10.0            |
| 15 Minutes (2001)                   | 0179626     | 6              | 6.67            |
| 16 Blocks (2006)                    | 0450232     | 16             | 6.38            |
| 2 Fast 2 Furious (2003)             | 0322259     | 32             | 6.72            |
| 2 Guns (2013)                       | 1272878     | 691            | 7.09            |
| 3000 Miles to Graceland (2001)      | 0233142     | 2              | 6.5             |
| 40 Days and Nights (2012)           | 2439946     | 3              | 3.0             |
| 48 Hrs. (1982)                      | 0083511     | 7              | 7.86            |
| 4: Rise of the Silver Surfer (2007) | 0486576     | 31             | 5.32            |
+-------------------------------------+-------------+----------------+-----------------+
10 rows selected (8.191 seconds)
0: jdbc:hive2://localhost:10000>
```

29)Split the column genre into different columns, with each genre in a single column:

The first query return the movie_titles and movie_genres. We can see the movie_genre is a concatenated string which can be further broken into substrings each containing a single genre, this can help us in further analyzing the category.

I used the split() function to extract a substring from the movie_genres string. The split function takes in the string value, and the delimeter on which you want the string to be broken. It further takes a value in [] square parenthesis which defines which position word would you like to be extracted.

If we have a string “Nada|Inam|09826” and I write [0] then it would extract the 1st word (which is before the parenthesis, that is Nada. For the substring 09826 we would need to write [2], as string position start from 0.



```
0: jdbc:hive2://localhost:10000> Select movie_title, movie_genres from movies limit 5;
+---------------------------------+-----------------------------+
|           movie_title           |         movie_genres        |
+---------------------------------+-----------------------------+
| Le manoir du diable (1896)      | Short|Horror                |
| The Adventures of Dollie (1908) | Action|Short                |
| The Country Doctor (1909)       | Short|Drama                 |
| Frankenstein (1910)             | Short|Horror|Sci-Fi          |
| The Lonedale Operator (1911)    | Short|Drama|Romance|Western  |
+---------------------------------+-----------------------------+
5 rows selected (0.132 seconds)
0: jdbc:hive2://localhost:10000> Select movie_title, split(movie_genres, '[\\|]')[0] genre_1, split(movie_genres, '[\\|]')[1] genre_2,
 split(movie_genres, '[\\|]')[2] genre_3 from movies limit 4;
+---------------------------------+----------+----------+----------+
|           movie_title           | genre_1  | genre_2  | genre_3  |
+---------------------------------+----------+----------+----------+
| Le manoir du diable (1896)      | Short    | Horror   | NULL     |
| The Adventures of Dollie (1908) | Action   | Short    | NULL     |
| The Country Doctor (1909)       | Short    | Drama    | NULL     |
| Frankenstein (1910)             | Short    | Horror   | Sci-Fi   |
+---------------------------------+----------+----------+----------+
4 rows selected (0.123 seconds)
0: jdbc:hive2://localhost:10000>
```

V. Multi-Node (multi-container)

Listing all available networks using the command docker network ls. We plan to connect a container to the network names docker-hive_default and make it communicate with other containers

```
C:\Users\DANIYAL\docker-hive>docker network ls
NETWORK ID     NAME                   DRIVER    SCOPE
4ea53886e636   bridge                 bridge    local
5a080aeecc7d   docker-hive_default    bridge    local
3168e77d1a89   docker_gwbridge        bridge    local
059faddfb012   host                   host      local
nebir80i0cmb   ingress                overlay   swarm
86ecd436454b   manager1               bridge    local
04ac6a343846   mongodb_default        bridge    local
f4fb74adf4fd   none                   null      local
b79db990faea   worker1                bridge    local
cd4143a2fa08   worker2                bridge    local
11b5aa2d1975   worker3                bridge    local
```

Using the command docker inpect docker-hive_default we can see all the containers that are connected to this network. There are a total of 6 nodes connected to the network.

```
C:\Users\DANIYAL\docker-hive>docker inspect docker-hive_default
[
    {
        "Name": "docker-hive_default",
        "Id": "5a080aeecc7d292778783059c883d472b568ce4c6aa1f4e6b68673dd4bfa9a8d",
        "Created": "2022-05-31T12:56:00.0667327Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.24.0.0/16",
                    "Gateway": "172.24.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": true,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "0cfba8ba6e614bc0c11ddafd4da173a13282885857dcc9731498597f5cb89229": {
                "Name": "docker-hive_presto-coordinator_1",
                "EndpointID": "ffef24709fbe314086a9ab1909a742f9dcea87fdd1d91e8f6052d8a19a6b827e",
                "MacAddress": "02:42:ac:18:00:05",
                "IPv4Address": "172.24.0.5/16",
                "IPv6Address": ""
            },
            "42daad4916e04418aa53f89d8d53552d1c4214482717bcb84d4c5bc77b2cf74a": {
                "Name": "docker-hive_datanode_1",
            },
            "42daad4916e04418aa53f89d8d53552d1c4214482717bcb84d4c5bc77b2cf74a": {
                "Name": "docker-hive_datanode_1",
                "EndpointID": "5c48ded6eb4a354e5d6df2daf279d862b1445af6e0ca371dc21469124e240718",
                "MacAddress": "02:42:ac:18:00:07",
                "IPv4Address": "172.24.0.7/16",
                "IPv6Address": ""
            },
            "700e2df0954186f42628e46306b3127e5395182ab8d2aaaeb2d3eeaced06e0be": {
                "Name": "docker-hive_hive-metastore-postgresql_1",
                "EndpointID": "f9b9454e64a8402aba8516ea9231e21ec2b489794b422ce5a5b92a7e9a769dcd",
                "MacAddress": "02:42:ac:18:00:06",
                "IPv4Address": "172.24.0.6/16",
                "IPv6Address": ""
            },
            "74adaf8a49e25769cd2c1215942049e6cfb25d2f6afd9fa8fa33ec2afb7af514": {
                "Name": "docker-hive_hive-metastore_1",
                "EndpointID": "8ccd1210a6fc0eb7744c0c70890e1ff1291a9c107141b937f65fd085b3e54742",
                "MacAddress": "02:42:ac:18:00:02",
                "IPv4Address": "172.24.0.2/16",
                "IPv6Address": ""
            },
            "956bda60644a441957f56f7b147ae5b0300f6b5b5346a33bf9e6ede135811281": {
                "Name": "docker-hive_hive-server_1",
                "EndpointID": "c5fece72755ec9f21993525cac189e52d75232f04bea98ef52f229270cc04cf2",
                "MacAddress": "02:42:ac:18:00:03",
                "IPv4Address": "172.24.0.3/16",
                "IPv6Address": ""
            },
            "9f491e10b73ae8d4927e02b7eee07d9566a2c26e80c22550b256c2b0b20b93c9": {
                "Name": "docker-hive_namenode_1",
                "EndpointID": "1553b25da1363f8b46660f55a6b77c88518c5e055baf240b20b01d0c31bf426f",
                "MacAddress": "02:42:ac:18:00:04",
                "IPv4Address": "172.24.0.4/16",
                "IPv6Address": ""
            }
        },
```

Next, we create a container which will act as a datanode with the name docker-hive_datenode_2. I mistakenly created a node with the name docker-hive_namenode_2, but I will rename it.

We use the command docker network connect docker-hive_default docker-hive_namenode_2, this command will connect the container we just created to the docker-hive_default network.

```
C:\Users\DANIYAL\docker-hive>docker run -dit --name docker-hive_namenode_2 alpine
aa3e854df01bf7b97cccd0c13c2213c83d34e36e2fc3d04ed5ffe9131d4c72fb

C:\Users\DANIYAL\docker-hive>docker network connect docker-hive_default docker-hive_namenode_2
```

When we use the command docker inspect docker-hive_default we can see that now the network has 7 containers connected instead of 6 like we previously saw.

```
C:\Users\DANIYAL\docker-hive>docker inspect docker-hive_default
[
    {
        "Name": "docker-hive_default",
        "Id": "5a080aeecc7d292778783059c883d472b568ce4c6aa1f4e6b68673dd4bfa9a8d",
        "Created": "2022-05-31T12:56:00.0667327Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.24.0.0/16",
                    "Gateway": "172.24.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": true,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "0cfba8ba6e614bc0c11ddafd4da173a13282885857dcc9731498597f5cb89229": {
                "Name": "docker-hive_presto-coordinator_1",
                "EndpointID": "ffef24709fbe314086a9ab1909a742f9dcea87fdd1d91e8f6052d8a19a6b827e",
                "MacAddress": "02:42:ac:18:00:05",
                "IPv4Address": "172.24.0.5/16",
                "IPv6Address": ""
            },
            "42daad4916e04418aa53f89d8d53552d1c4214482717bcb84d4c5bc77b2cf74a": {
                "Name": "docker-hive_datanode_1",
```

The new container docker-hive_namenode_2 can now be seen.


```
        "700e2df0954186f42628e46306b3127e5395182ab8d2aaaeb2d3eeaced06e0be": {
            "Name": "docker-hive_hive-metastore-postgresql_1",
            "EndpointID": "f9b9454e64a8402aba8516ea9231e21ec2b489794b422ce5a5b92a7e9a769dcd",
            "MacAddress": "02:42:ac:18:00:06",
            "IPv4Address": "172.24.0.6/16",
            "IPv6Address": ""
        },
        "74adaf8a49e25769cd2c1215942049e6cfb25d2f6afd9fa8fa33ec2afb7af514": {
            "Name": "docker-hive_hive-metastore_1",
            "EndpointID": "8ccd1210a6fc0eb7744c0c70890e1ff1291a9c107141b937f65fd085b3e54742",
            "MacAddress": "02:42:ac:18:00:02",
            "IPv4Address": "172.24.0.2/16",
            "IPv6Address": ""
        },
        "956bda60644a441957f56f7b147ae5b0300f6b5b5346a33bf9e6ede135811281": {
            "Name": "docker-hive_hive-server_1",
            "EndpointID": "c5fece72755ec9f21993525cac189e52d75232f04bea98ef52f229270cc04cf2",
            "MacAddress": "02:42:ac:18:00:03",
            "IPv4Address": "172.24.0.3/16",
            "IPv6Address": ""
        },
        "9f491e10b73ae8d4927e02b7eee07d9566a2c26e80c22550b256c2b0b20b93c9": {
            "Name": "docker-hive_namenode_1",
            "EndpointID": "1553b25da1363f8b46660f55a6b77c88518c5e055baf240b20b01d0c31bf426f",
            "MacAddress": "02:42:ac:18:00:04",
            "IPv4Address": "172.24.0.4/16",
            "IPv6Address": ""
        },
        "aa3e854df01bf7b97cccd0c13c2213c83d34e36e2fc3d04ed5ffe9131d4c72fb": {
            "Name": "docker-hive_namenode_2",
            "EndpointID": "b8cf2aa34ee4f7bf3160e2b9cd82e598b5c264ffcf5543d734bb46f4bb63b1bf",
            "MacAddress": "02:42:ac:18:00:08",
            "IPv4Address": "172.24.0.8/16",
            "IPv6Address": ""
        }
    },
    "Options": {},
```


We will rename the container docker-hive_namenode_2 to docker-hive_datanode_2

```
C:\Users\DANIYAL\docker-hive>docker rename docker-hive_namenode_2 docker-hive_datanode_2
```

When we inspect the docker-hive_default network we can see the name of the container has been automatically renamed here as well.

```
C:\Users\DANIYAL\docker-hive>docker inspect docker-hive_default
[
    {
        "Name": "docker-hive_default",
        "Id": "5a080aeecc7d292778783059c883d472b568ce4c6aa1f4e6b68673dd4bfa9a8d",
        "Created": "2022-05-31T12:56:00.0667327Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.24.0.0/16",
                    "Gateway": "172.24.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": true,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "0cfba8ba6e614bc0c11ddafd4da173a13282885857dcc9731498597f5cb89229": {
                "Name": "docker-hive_presto-coordinator_1",
                "EndpointID": "ffef24709fbe314086a9ab1909a742f9dcea87fdd1d91e8f6052d8a19a6b827e",
                "MacAddress": "02:42:ac:18:00:05",
                "IPv4Address": "172.24.0.5/16",
                "IPv6Address": ""
            },
            "42daad4916e04418aa53f89d8d53552d1c4214482717bcb84d4c5bc77b2cf74a": {
                "Name": "docker-hive_datanode_1",
                "Name": "docker-hive_hive-metastore_1",
                "EndpointID": "8ccd1210a6fc0eb7744c0c70890e1ff1291a9c107141b937f65fd085b3e54742",
                "MacAddress": "02:42:ac:18:00:02",
                "IPv4Address": "172.24.0.2/16",
                "IPv6Address": ""
            },
            "956bda60644a441957f56f7b147ae5b0300f6b5b5346a33bf9e6ede135811281": {
                "Name": "docker-hive_hive-server_1",
                "EndpointID": "c5fece72755ec9f21993525cac189e52d75232f04bea98ef52f229270cc04cf2",
                "MacAddress": "02:42:ac:18:00:03",
                "IPv4Address": "172.24.0.3/16",
                "IPv6Address": ""
            },
            "9f491e10b73ae8d4927e02b7eee07d9566a2c26e80c22550b256c2b0b20b93c9": {
                "Name": "docker-hive_namenode_1",
                "EndpointID": "1553b25da1363f8b46660f55a6b77c88518c5e055baf240b20b01d0c31bf426f",
                "MacAddress": "02:42:ac:18:00:04",
                "IPv4Address": "172.24.0.4/16",
                "IPv6Address": ""
            },
            "aa3e854df01bf7b97cccd0c13c2213c83d34e36e2fc3d04ed5ffe9131d4c72fb": {
                "Name": "docker-hive_datanode_2",
                "EndpointID": "b8cf2aa34ee4f7bf3160e2b9cd82e598b5c264ffcf5543d734bb46f4bb63b1bf",
                "MacAddress": "02:42:ac:18:00:08",
                "IPv4Address": "172.24.0.8/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {
            "com.docker.compose.network": "default",
            "com.docker.compose.project": "docker-hive",
            "com.docker.compose.version": "1.29.2"
        }
    }
]
```

Now, we bash into the new container we just created and we will ping other containers from it.

The command used to bash is docker exec -it docker-hive_Datanode_2 /bin/sh

We then run apk update and apk add iputils to install all libraries that are needed to ping.

```
C:\Users\DANIYAL\docker-hive>docker exec -it docker-hive_datanode_2 /bin/sh
/ # apk update
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/community/x86_64/APKINDEX.tar.gz
v3.16.0-57-gefb53c0ec1 [https://dl-cdn.alpinelinux.org/alpine/v3.16/main]
v3.16.0-65-ga9b6c9d003 [https://dl-cdn.alpinelinux.org/alpine/v3.16/community]
OK: 17018 distinct packages available
/ # apk add iputils
(1/2) Installing libcap (2.64-r0)
(2/2) Installing iputils (20211215-r0)
Executing busybox-1.35.0-r13.trigger
OK: 6 MiB in 16 packages
/ # ping 172.24.0.2
PING 172.24.0.2 (172.24.0.2) 56(84) bytes of data.
64 bytes from 172.24.0.2: icmp_seq=1 ttl=64 time=1.08 ms
64 bytes from 172.24.0.2: icmp_seq=2 ttl=64 time=0.056 ms
64 bytes from 172.24.0.2: icmp_seq=3 ttl=64 time=0.087 ms
64 bytes from 172.24.0.2: icmp_seq=4 ttl=64 time=0.088 ms
64 bytes from 172.24.0.2: icmp_seq=5 ttl=64 time=0.181 ms
64 bytes from 172.24.0.2: icmp_seq=6 ttl=64 time=0.107 ms
64 bytes from 172.24.0.2: icmp_seq=7 ttl=64 time=0.115 ms
64 bytes from 172.24.0.2: icmp_seq=8 ttl=64 time=0.124 ms
64 bytes from 172.24.0.2: icmp_seq=9 ttl=64 time=0.148 ms
64 bytes from 172.24.0.2: icmp_seq=10 ttl=64 time=0.145 ms
```

We can see the newly created, and connected container can now communicate with other containers. Here it is communicating with docker-hive_namenode_1

```
C:\Users\DANIYAL\docker-hive>docker exec -it docker-hive_datanode_2 /bin/sh
/ # ping 172.24.0.4
PING 172.24.0.4 (172.24.0.4) 56(84) bytes of data.
64 bytes from 172.24.0.4: icmp_seq=1 ttl=64 time=36.6 ms
64 bytes from 172.24.0.4: icmp_seq=2 ttl=64 time=0.394 ms
64 bytes from 172.24.0.4: icmp_seq=3 ttl=64 time=0.201 ms
64 bytes from 172.24.0.4: icmp_seq=4 ttl=64 time=0.061 ms
64 bytes from 172.24.0.4: icmp_seq=5 ttl=64 time=0.118 ms
64 bytes from 172.24.0.4: icmp_seq=6 ttl=64 time=0.068 ms
64 bytes from 172.24.0.4: icmp_seq=7 ttl=64 time=0.343 ms
64 bytes from 172.24.0.4: icmp_seq=8 ttl=64 time=0.062 ms
64 bytes from 172.24.0.4: icmp_seq=9 ttl=64 time=0.118 ms
64 bytes from 172.24.0.4: icmp_seq=10 ttl=64 time=0.149 ms
64 bytes from 172.24.0.4: icmp_seq=11 ttl=64 time=0.138 ms
64 bytes from 172.24.0.4: icmp_seq=12 ttl=64 time=0.172 ms
64 bytes from 172.24.0.4: icmp_seq=13 ttl=64 time=0.149 ms
```


```
},
"9f491e10b73ae8d4927e02b7eee07d9566a2c26e80c22550b256c2b0b20b93c9": {
    "Name": "docker-hive_namenode_1",
    "EndpointID": "1553b25da1363f8b46660f55a6b77c88518c5e055baf240b20b01d0c31bf426f",
    "MacAddress": "02:42:ac:18:00:04",
    "IPv4Address": "172.24.0.4/16",
    "IPv6Address": ""
}
```

We can see the newly created, and connected container can now communicate with other containers. Here it is communicating with docker-hive_hive_server_1

```
C:\Users\DANIYAL\docker-hive>docker exec -it docker-hive_datanode_2 /bin/sh
/ # ping 172.24.0.3
PING 172.24.0.3 (172.24.0.3) 56(84) bytes of data.
64 bytes from 172.24.0.3: icmp_seq=1 ttl=64 time=0.601 ms
64 bytes from 172.24.0.3: icmp_seq=2 ttl=64 time=0.143 ms
64 bytes from 172.24.0.3: icmp_seq=3 ttl=64 time=0.066 ms
64 bytes from 172.24.0.3: icmp_seq=4 ttl=64 time=0.105 ms
64 bytes from 172.24.0.3: icmp_seq=5 ttl=64 time=0.093 ms
64 bytes from 172.24.0.3: icmp_seq=6 ttl=64 time=0.078 ms
64 bytes from 172.24.0.3: icmp_seq=7 ttl=64 time=0.103 ms
```


```
"956bda60644a441957f56f7b147ae5b0300f6b5b5346a33bf9e6ede135811281": {
    "Name": "docker-hive_hive-server_1",
    "EndpointID": "c5fece72755ec9f21993525cac189e52d75232f04bea98ef52f229270cc04cf2",
    "MacAddress": "02:42:ac:18:00:03",
    "IPv4Address": "172.24.0.3/16",
    "IPv6Address": ""
},
```

VI. Conclusion

HiveQL is a language similar to SQL, it is very easy to use and is faster compared to SQL. I was very excited to try Hive as I use SQL and PostgreSQL on a daily basis and the time joins take without indexing is long. However, in my opinion, in hive queries were running smoothly and returning results quicker than I expected even without indexing. Also, I never load data files directly into mssql server, sql (using workbench) and on Greenplum (Dbeaver user interface) as they take a bit of time and is not very reliant, therefore I prefer using ETL tools, but with Hive it is much easier and quicker to load data from local path into a table.