Practical 6

This computer practical will serve as a brief introduction to Hive, Apache's data warehouse software. If you're familiar with SQL you'll notice that, unlike Pig which is loosely inspired by some SQL commands, the HiveQL language is nearly identical.

Similarly to Pig, we can start Hive by simply typing '**hive**' in the terminal. Instead of the grunt prompt, we now how a 'hive' prompt. Start the hive prompt now. You'll know you've been successful when you see the following.

```
prac@prac-VirtualBox:~$ hive
Hive Session ID = 14c68063-7fa5-46e1-bed1-a985cbab9b6a

Logging initialized using configuration in jar:file:/opt/hive/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
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.
hive>
```

Let's write our first Hive command.

```
hive> SHOW TABLES;
```

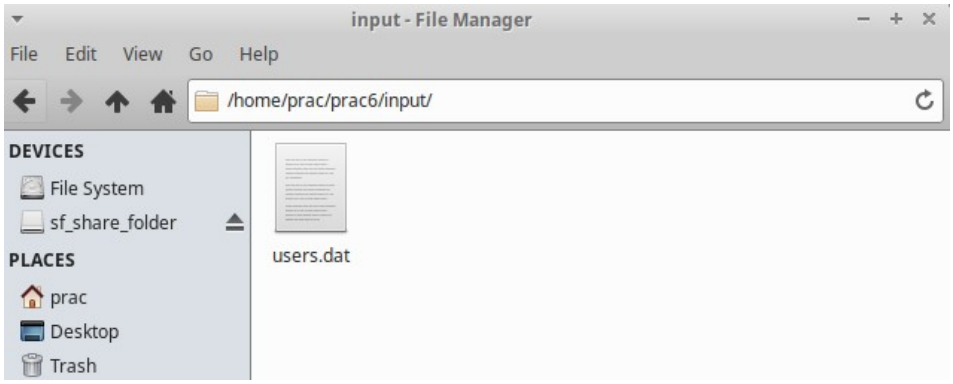
We haven't made any tables yet, so your output should look something like this:

```
hive> SHOW TABLES;
OK
Time taken: 0.035 seconds
```

As with the other languages we've used so far. You can run your queries by typing them manually into the terminal, like we just did above, or by running a script. For example, to run **script.q** we would use the following command. We don't have such a script yet, so don't run the command, it won't work.

```
$ hive -f script.q
```

Let's use the 1M movie dataset again. You might still have a copy of it, but if not, you can download again from here. In this practical we'll perform another task that we know would have been challenging to code in Hadoop MapReduce, that is, calculating the average age of users for each gender. Make a **prac6** directory with an **input** subdirectory and copy the **users.dat** file there.



To create a table containing the data, type the following into the Hive prompt, it's all one line.

```
hive> CREATE TABLE users (UserID INT, n1 STRING, Gender STRING, n2 STRING, Age INT, n3 STRING, OCCUPATION INT, n4 STRING, Zipcode STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ':';
```

```
hive> CREATE TABLE users (UserID INT, n1 STRING, Gender STRING, n2 STRING, Age I
NT, n3 STRING, OCCUPATION INT, n4 STRING, Zipcode STRING) ROW FORMAT DELIMITED F
IELDS TERMINATED BY ':';
OK
Time taken: 0.939 seconds
```

This probably looks somewhat familiar if you're used to SQL. We just created a table called **users** with all the correct fields for our **users.dat** file. However, the table itself is still empty. We load the data into our table with the following command:

```
hive> LOAD DATA LOCAL INPATH '/home/prac/prac6/input/users.dat' OVERWRITE INTO TABLE users;
```

```
hive> LOAD DATA LOCAL INPATH '/home/prac/prac6/input/users.dat' OVERWRITE INTO T
ABLE users;
Loading data to table default.users
OK
Time taken: 1.466 seconds
```

We can use the **desc** command to get an overview of a table in Hive, similarly to the **describe** command from Pig.

```
hive> DESC users;
```

```
hive> DESC users;
OK
userid                  int
n1                      string
gender                  string
n2                      string
age                     int
n3                      string
occupation              int
n4                      string
zipcode                 string
Time taken: 0.047 seconds, Fetched: 9 row(s)
```

Or the **LIMIT** clause in conjunction with the **SELECT** statement to view a subset of our data:

```
hive> SELECT * FROM users LIMIT 5;
```

```
hive> SELECT * FROM users LIMIT 5;
OK
1               F               1               10              48067
2               M               56              16              70072
3               M               25              15              55117
4               M               45              7               02460
5               M               25              20              55455
Time taken: 1.735 seconds, Fetched: 5 row(s)
```

We can add the WHERE operator to further specify the subset that we wish to view:

```
hive> SELECT * FROM users WHERE Gender == 'F' LIMIT 5;
```

```
hive> SELECT * FROM users WHERE Gender == 'F' LIMIT 5;
OK
1               F               1               10              48067
6               F               50              9               55117
10              F               35              1               95370
11              F               25              1               04093
16              F               35              0               20670
Time taken: 0.545 seconds, Fetched: 5 row(s)
```

You'll notice that in the **LOAD** command we specified the term '**LOCAL**', since we are loading data from our local OS. Try now to make a copy of your data file in HDFS, and load the data from there into your users table with the same command as above. You'll just need to exclude **LOCAL** from the command, and change the directory to point to the HDFS file.

Important note! – When we load data from the local drive, the data is copied into HDFS. When we load data from HDFS, the data will be moved from its current folder. In both scenarios, the file ends up inside HDFS in the folder **/user/hive/warehouse**. If you recall the **ls** command, you can quickly quit Hive and check the contents of the warehouse directory to see if your users file made it there.

```
prac@prac-VirtualBox:~$ hadoop fs -ls /user/hive/warehouse/users
Found 1 items
-rw-r--r--   1 prac supergroup     134368 2022-01-24 16:10 /user/hive/warehouse/
users/users.dat
```

As you might have guessed, there is a user used by **Hive** inside the Hadoop system. This user collects all data files connected to hive tables. Moving files from their original folders allows to save space and make them available to all users in the system. Created tables persist even if you restart the system. Once created the table is available until it is deleted.

So, before we try to load our HDFS files, we would delete our table first, using the **DROP** command.

```
hive> DROP TABLE users;
```

The database files will be deleted at the same time. The **ls** command returns empty on the Hive data warehouse directory.

```
prac@prac-VirtualBox:~$ hadoop fs -ls /user/hive/warehouse
prac@prac-VirtualBox:~$
```

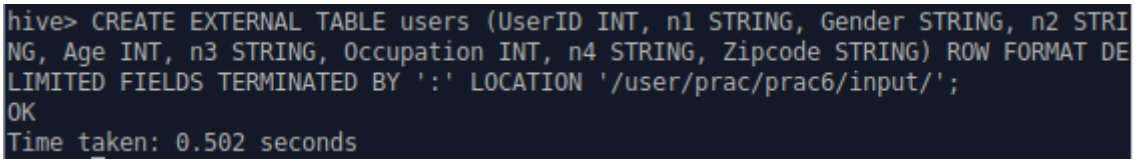
Now, let us create the directory and upload the data file onto HDFS.

```
$ hadoop fs -mkdir -p /user/prac/prac6/input
```

```
$ hadoop fs -put /home/prac/prac6/input/users.dat /user/prac/prac6/input
```

If you already have your data file in HDFS and you don't want to move it to the Hive warehouse, it's not actually necessary to LOAD the file. You can create a table **connected** to the file instead:

```
hive> CREATE EXTERNAL TABLE users (UserID INT, n1 STRING, Gender STRING, n2 STRING, Age INT, n3 STRING, Occupation INT, n4 STRING, Zipcode STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ':' LOCATION '/user/prac/prac6/input/';
```



This is called an **EXTERNAL** table because the data are not inside the Hive warehouse. Please note, that for this method we point not to the ***file*** but the ***location*** in HDFS, where the files are stored. Similarly to the case with Python streaming, MapReduce will grab and process all files it manages to find in the *input* folder. This feature is important if you get new files arriving into that folder regularly, so you don't care how many files are there – all of them will be connected to your table. You just need to be certain that all of them share the same structure.

If you plan to use several different datasets (like in this case where we have **users.dat**, **ratings.dat** and **movies.dat**), then each file should have its own folder. In which case we'd probably make a folder called **users**, a folder called **ratings** etc.

Finally, we are ready to calculate the average age of males and females in the data. Like Pig, this is much easier to do in Hive than it is in plain MapReduce with Python streaming. The command is as follows:

```
hive> SELECT Gender, AVG(Age) FROM users GROUP BY Gender;
```

Given your experience with Pig, and potentially SQL, you should have a pretty good idea of what this line is doing, but if not you can try to break it down piece by piece. This line will take a minute or so to run so don't be alarmed while it's running.

Since this command involved data processing, Hive created java code for a MapReduce program and executed it on the Hadoop system. Once again we're drawn back to the same picture: whether its Python streaming, Pig or Hive, all of the different applications create MapReduce programs and run them on Hadoop. MapReduce is the only way we have to analyse Big Data.

By the way, the result of your average age calculation should look like this:

```
hive> SELECT Gender, AVG(Age) FROM users GROUP BY Gender;
Query ID = prac_20220124163459_688e0007-03ce-4808-8bb7-d4a97a3f9d2a
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1643001686280_0001, Tracking URL = http://prac-VirtualBox:8088/proxy/application_1643001686280_0001/
Kill Command = /opt/hadoop/bin/mapred job  -kill job_1643001686280_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-01-24 16:35:09,502 Stage-1 map = 0%,  reduce = 0%
2022-01-24 16:35:13,666 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.27 sec
2022-01-24 16:35:19,830 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 2.92 sec
MapReduce Total cumulative CPU time: 2 seconds 920 msec
Ended Job = job_1643001686280_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.92 sec   HDFS Read: 150147 HDFS Write: 152 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 920 msec
OK
F       30.85956699824459
M       30.552297390902794
Time taken: 22.959 seconds, Fetched: 2 row(s)
```

Useful Resources

https://cwiki.apache.org/confluence/display/hive/languagemanual+cli