**Assignment 7**

**Data Manipulation with Hive and Pig**

CIS: Big Data Solutions for Business

Karishma Borse

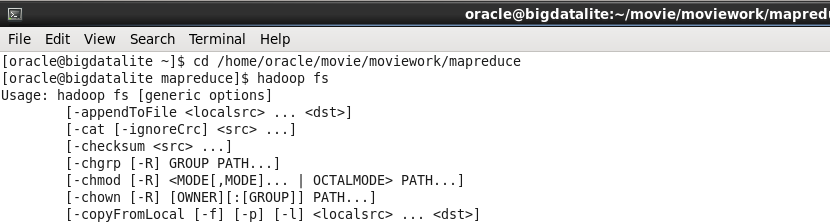
## Exercise 1 – Loading Avro data into HDFS

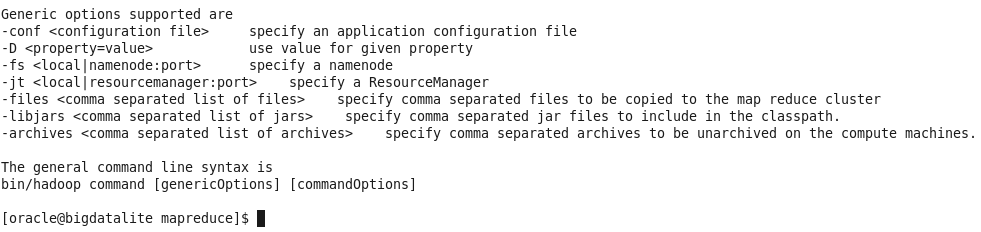
Run the below commands from the /home/oracle/movie/moviework/mapreduce directory.

#1. See the commands available for the Hadoop Distributed File System using following command

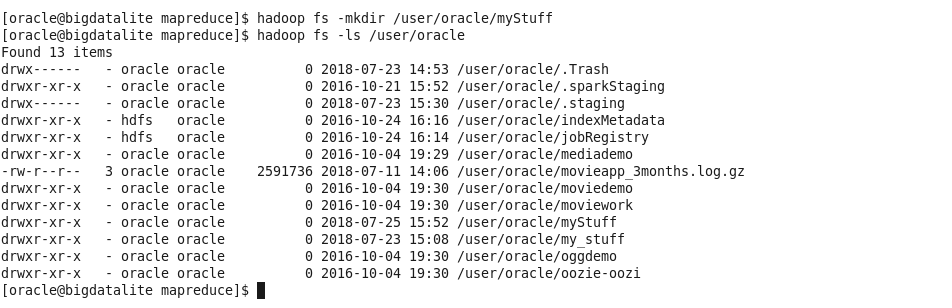
cd /home/oracle/movie/moviework/mapreduce

hadoop fs

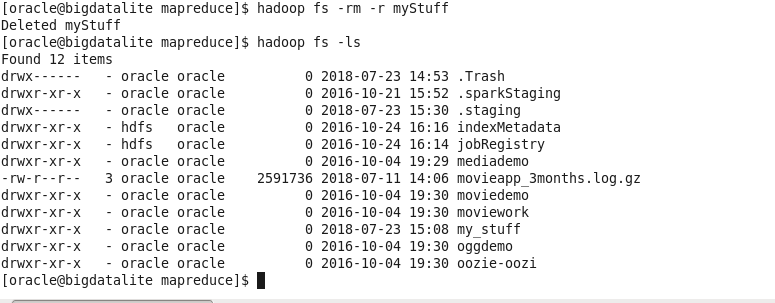




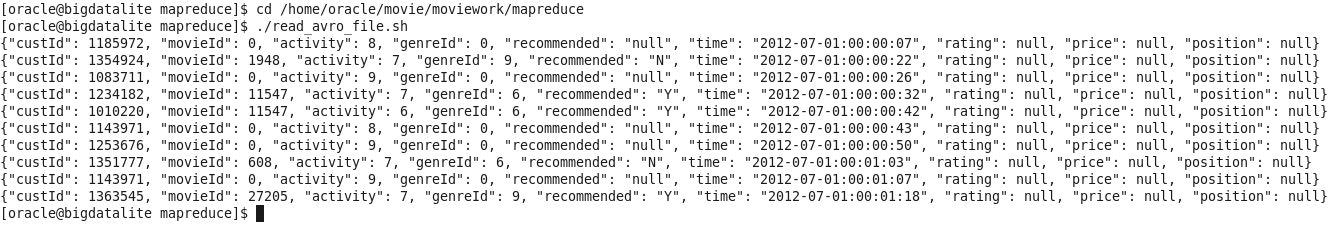
#2. Create a subdirectory by name “my\_stuff” in the /user/oracle folder and then ensure the directory has been created:



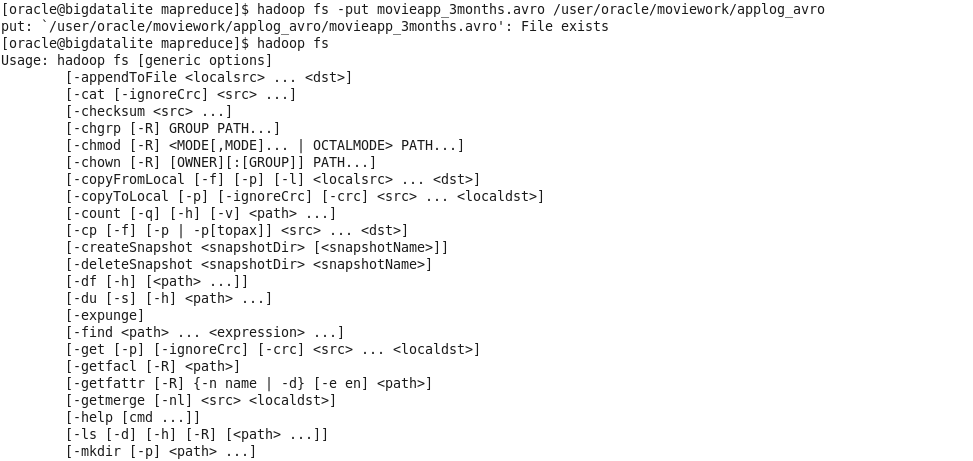
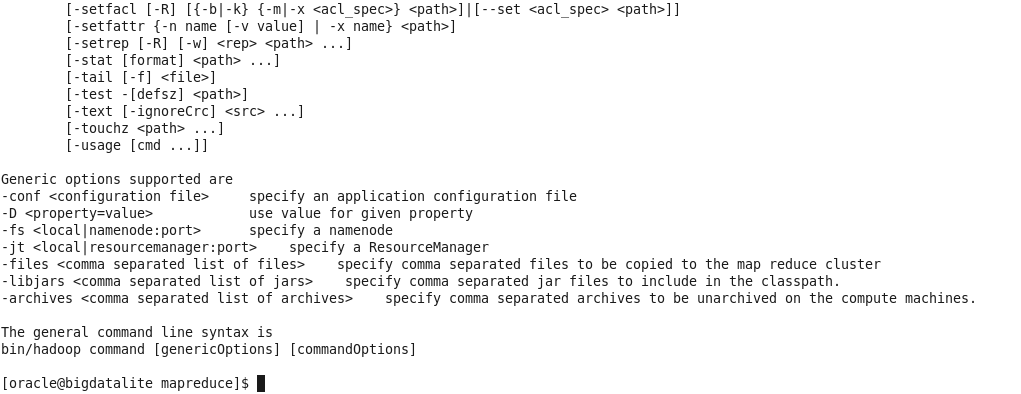
#3. Remove the directory “my\_stuff” and then confirm that it has been removed:



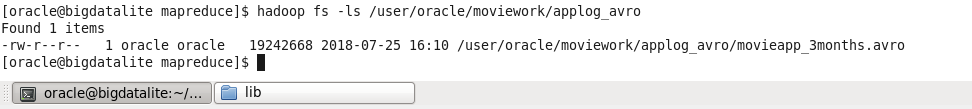
Then, load a file into HDFS from the local file system. Specifically you will load an Avro log file that tracked activity in an on-line movie application. The Avro data represents individual clicks from an online movie rental site. You will use the basic “put” commands for moving data into Hadoop Distributed File System.

#4. Inspect the compressed JSON application log

#5. Review the commands available for the Hadoop Distributed File System and copy the Avro file into HDFS:



#6. Verify the copy by listing the directory contents in HDFS:

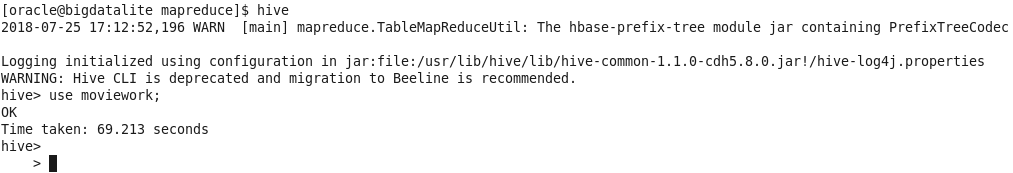


## 

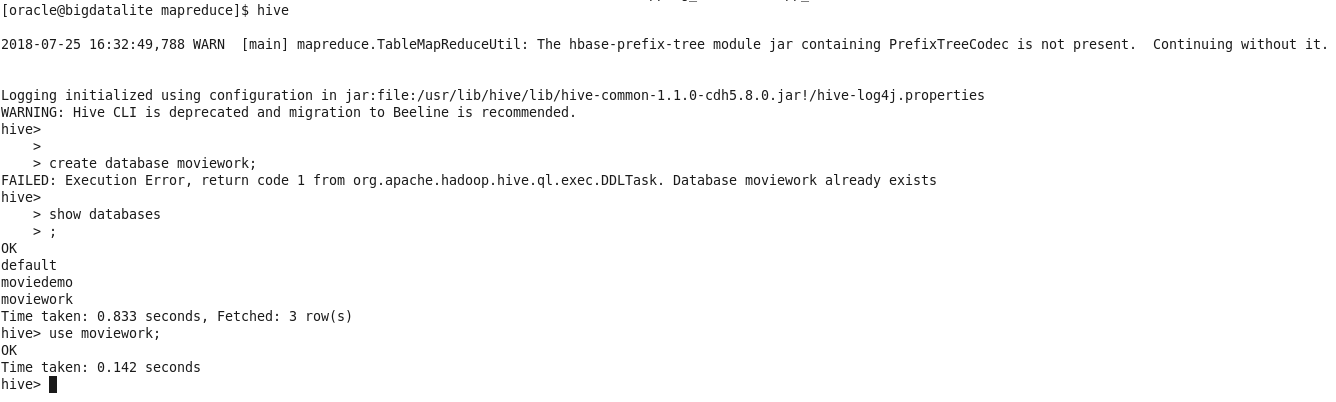
## Exercise 2 – Define an external Hive table and review the results

* Create a database to store hive tables.
* Review the Avro schema for the data file that contains the movie activity.
* Create an external table that parses the Avro fields and maps them to the columns in the table.
* Select the min and max time periods contained table using HiveQL.

#1. Enter the Hive command line by typing hive at the Linux prompt:



#2. Create a new hive database called “moviework”. Ensure that the database has been successfully created

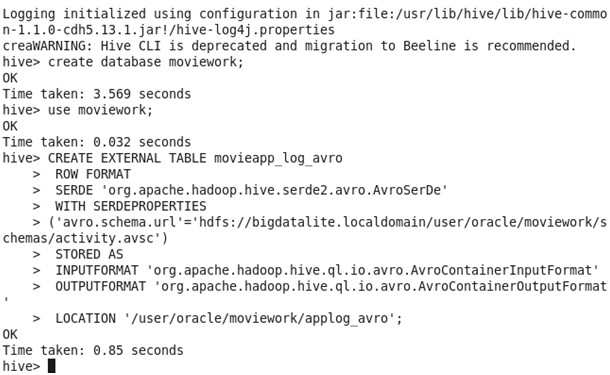


#3. Review the schema definition for the AVRO file and then define a table using that schema. Using the below command:

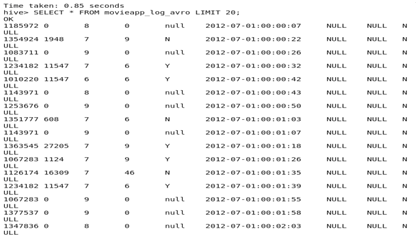
**hadoop fs -cat moviework/schemas/activity.avsc**



#4. To create a table in a database, you can either fully qualify the table name (i.e. prepend the database to the name of the table) or you can designate that you want all DDL and DML operations to apply to a specific database. For simplicity, you will apply subsequent operations to the moviework database:



5. Review the schema for the Avro file. This schema definition has already been saved in HDFS in the /user/oracle/moviework/schemas/ directory. Create a new Terminal window and type the following command at the Linux prompt to review the schema definition:



Also perform the following command

hive> select count(\*) FROM movieapp\_log\_avro LIMIT 20;

Query ID = oracle\_20180723152727\_d8a1fa8e-9a56-4b19-a3fc-eca59ce6a55d

Total jobs = 1

Launching Job 1 out of 1

Number of reduce tasks determined at compile time: 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\_1532366204583\_0001, Tracking URL = <http://bigdatalite.localdomain:8088/proxy/application_1532366204583_0001/>

Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job\_1532366204583\_0001

Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1

2018-07-23 15:29:21,158 Stage-1 map = 0%, reduce = 0%

2018-07-23 15:30:09,450 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.41 sec

2018-07-23 15:30:41,438 Stage-1 map = 100%, reduce = 67%, Cumulative CPU 6.89 sec

2018-07-23 15:30:47,930 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 9.43 sec

MapReduce Total cumulative CPU time: 9 seconds 430 msec

Ended Job = job\_1532366204583\_0001

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 9.43 sec HDFS Read: 12170 HDFS Write: 2 SUCCESS

Total MapReduce CPU Time Spent: 9 seconds 430 msec

OK

0

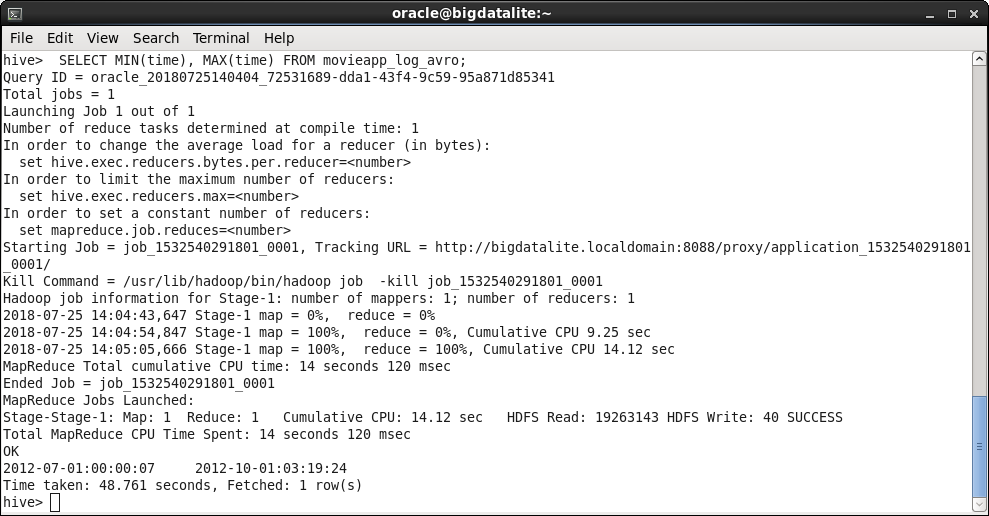
Time taken: 180.218 seconds, Fetched: 1 row(s)

hive>

#6. HiveQL supports many standard SQL operations. Find the min and max time periods that are available in the log file:

Using the below command:

hive> SELECT MIN(time), MAX(time) FROM movieapp\_log\_avro;

****

## Exercise 3 – Extract facts using Hive

#1. Write a query to select only those clicks which correspond to starting, browsing, completing, or purchasing movies. Use a CASE statement to transform the RECOMMENDED column into integers where ‘Y’ is 1 and ‘N’ is 0. Also, ensure GENREID is not null. Only include the first 25 rows:

hive> SELECT custid, movieid,

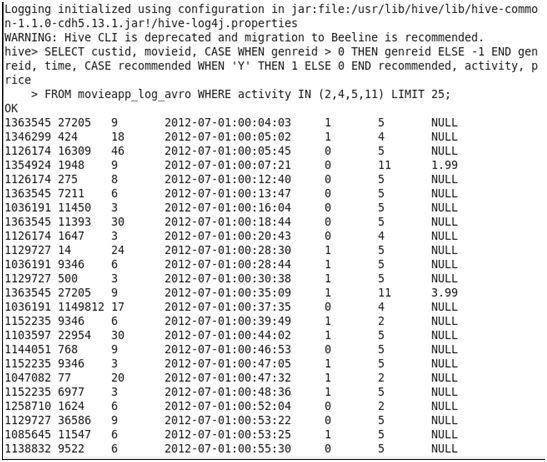
CASE WHEN genreid > 0 THEN genreid ELSE -1 END genreid, time,

CASE recommended WHEN 'Y'

THEN 1 ELSE 0 END recommended, activity, price FROM movieapp\_log\_avro

WHERE activity IN (2,4,5,11) LIMIT 25;

**Results:**

****

#2. Write a query to select the customer ID, movie ID, recommended state and most recent rating for each movie.

hive> SELECT m1.custid, m1.movieid,

CASE WHEN m1.genreid > 0 THEN m1.genreid ELSE -1 END genreid, m1.time,

CASE m1.recommended

WHEN 'Y' THEN 1 ELSE 0 END recommended, m1.activity, m1.rating FROM movieapp\_log\_avro m1 JOIN (SELECT custid, movieid, CASE WHEN genreid > 0

THEN genreid ELSE -1 END

genreid, MAX(time) max\_time, activity FROM movieapp\_log\_avro

GROUP BY custid, movieid, genreid, activity ) m2 ON ( m1.custid = m2.custid AND m1.movieid = m2.movieid

AND m1.genreid = m2.genreid

AND m1.time = m2.max\_time

AND m1.activity = 1 AND m2.activity = 1 )

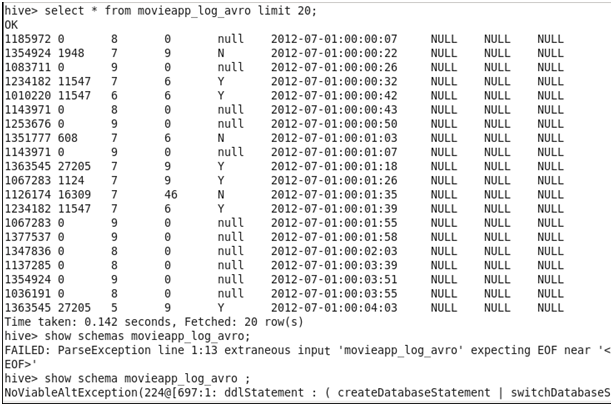
LIMIT 25;



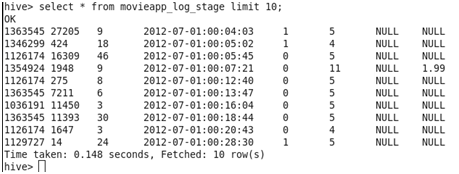
#3. Load the results of the previous two queries into a staging table.

Load the Results:

Result screen # 1



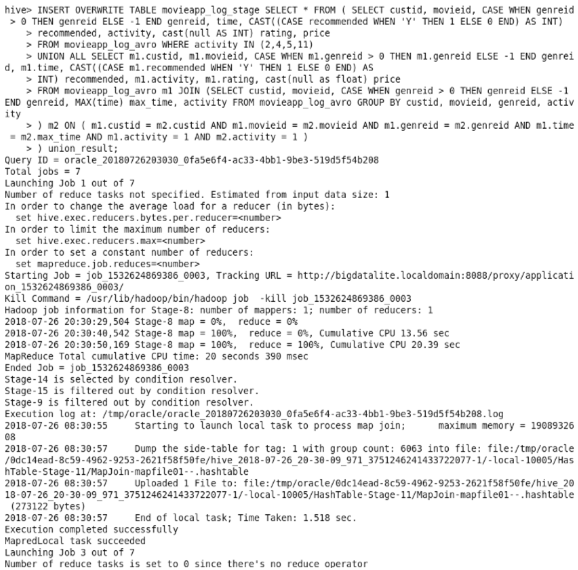
Result screen # 2



#4. Next, load the results of the queries into the staging table

INSERT OVERWRITE TABLE movieapp\_log\_stage SELECT \* FROM ( SELECT custid, movieid, CASE WHEN genreid > 0 THEN genreid ELSE -1 END genreid, time, CAST((CASE recommended WHEN 'Y' THEN 1 ELSE 0 END) AS INT) recommended, activity, cast(null AS INT) rating, price FROM movieapp\_log\_avro WHERE activity IN (2,4,5,11) UNION ALL SELECT m1.custid, m1.movieid, CASE WHEN m1.genreid > 0 THEN m1.genreid ELSE -1 END genreid, m1.time, CAST((CASE m1.recommended WHEN 'Y' THEN 1 ELSE 0 END) AS INT) recommended, m1.activity, m1.rating, cast(null as float) price FROM movieapp\_log\_avro m1 JOIN (SELECT custid, movieid, CASE WHEN genreid > 0 THEN genreid ELSE -1 END genreid, MAX(time) max\_time, activity FROM movieapp\_log\_avro GROUP BY custid, movieid, genreid, activity ) m2 ON ( m1.custid = m2.custid AND m1.movieid = m2.movieid AND m1.genreid = m2.genreid AND m1.time = m2.max\_time AND m1.activity = 1 AND m2.activity = 1 ) ) union\_result;

**Result screen: -**

****

All the steps followed and completed successfully.