BIG DATA HANDS-ON WORKSHOP

Data Manipulation with Hive and Pig

Contents

Acknowledgements

Information about the movies used in this demonstration – including titles, descriptions, actors, genres - is courtesy of Wikipedia. Movie posters are courtesy of TMDb. All other data used in this demonstration – including customers and their viewing behavior - is completely fictitious.

Introduction to Hive and Pig

In the emerging world of Big Data, data processing must be many things: fault-tolerant, massively-parallel, and linearly scalable. Central to achieving these goals is the understanding that computation is less costly to move than large volumes of data. The Hadoop ecosystem is an Open Source set of frameworks designed around this concept. Through its components, the Hadoop ecosystem enables developers to focus on solving their Big Data problems rather than developing ad hoc solutions to managing massive data.

At its core, Hadoop provides two components that are central to effectively dealing with petabyte-scale problems:

- HDFS: A distributed file system to provide fault-tolerant storage while scaling horizontally
- MapRedce: A computational framework which "moves the compute to the data" enabling incredible parallelism

The Hadoop Distributed File System (HDFS) plays a central role in storing and efficiently accessing massive amounts of data. HDFS is:

- Fault-tolerant: fault detection is rapid and recovery is automatic
- High-throughput: streaming data access and large block sizes optimizes operation on massive datasets
- Designed for data locality: enabling mapped computation is central to the design

Operating in parallel over data stored in HDFS requires a computing framework. MapReduce is the parallel data processing framework in Hadoop. Processing jobs are broken into tasks, distributed across the cluster, and run locally with the correct data. MapReduce jobs consist of two types of tasks:

- Map: filter and interpret input data, producing key-value pairs
- Reduce: summarize and aggregate the sorted map results, producing final output

Because Hadoop is largely written in Java, Java is the language of MapReduce. However, the Hadoop community understands that Java is not always the quickest or most natural way to describe data processing. As such, the ecosystem has evolved to support a wide variety of APIs and secondary languages. From scripting languages to SQL, the Hadoop ecosystem allows developers to express their data processing jobs in the language they deem most suitable. Hive and Pig are a pair of these secondary languages for interacting with data stored HDFS. Hive is a data warehousing system which exposes an SQL-like language called HiveQL. Pig is an analysis platform which provides a dataflow language called Pig Latin. In this workshop, we will cover the basics of each language.

Setup

Make sure the hands-on lab is initialized by running the following script:

```
cd /home/oracle/movie/moviework/reset
./reset_mapreduce.sh
```

Exercise 1 – Load Avro data into HDFS

Start by reviewing HDFS. You will find that its composition is similar to your local Linux file system. You will use the hadoop fs command when interacting with HDFS:

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

1. Review the commands available for the Hadoop Distributed File System:

```
cd  /home/oracle/movie/moviework/mapreduce
hadoop fs
```

2. List the contents of /user/oracle

3. Create a subdirectory called "my_stuff" in the /user/oracle folder and then ensure the directory has been created:

```
hadoop fs -mkdir /user/oracle/my_stuff
hadoop fs -ls /user/oracle
```

4. Remove the directory "my_stuff" and then ensure it has been removed:

```
hadoop fs -rm -r my_stuff
hadoop fs -ls
```

Next, 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.

5. Inspect the compressed JSON application log:

```
cd  /home/oracle/movie/moviework/mapreduce
./read_avro_file.sh
```

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

```
hadoop fs -put movieapp_3months.avro /user/oracle/moviework/applog_avro
```

7. Verify the copy by listing the directory contents in HDFS:

```
hadoop fs -ls /user/oracle/moviework/applog_avro
```

The remaining exercises operate over the data in this JSON file. Make a note of its location in HDFS and the fields in each tuple.

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

Now that you have placed the data into HDFS, you will want to start extracting information from it using an external table. An external table in hive is similar to an external table in Oracle Database 12c. It is a metadata object that is defined over a file. The metadata provides a table name, column names and types, file locations, processors, and more. Once that structure has been defined, you can query it using HiveQL.

In this exercise you will:

- Create a database to store your 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:

```
hive
```

2. Create a new hive database called **moviework.** Ensure that the database has been successfully created:

```
hive> create database moviework;
hive> show databases;
```

3. 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:

```
hive> use moviework;
```

4. 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:

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

   Notice that the schema contains the field names, data types and default values for each of the fields.

5. Create a Hive external table using that schema definition. Notice that you do not need to specify the column names or data types when defining the table. The Avro serializer-deserializer (or SERDE) will parse the schema definition to determine these values. After creating the table, review the results by selecting the first 20 rows. Go back into your Hive terminal window and run the following commands in the moviework database:

```
hive>   CREATE EXTERNAL TABLE movieapp_log_avro
  ROW FORMAT
  SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
  WITH SERDEPROPERTIES
('avro.schema.url'='hdfs://bigdatalite.localdomain/user/oracle/moviework/schemas/activity.avsc')
  STORED AS
    INPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
    OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
  LOCATION '/user/oracle/moviework/applog_avro';
hive>    SELECT * FROM movieapp_log_avro LIMIT 20;
```

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

```
hive>  SELECT MIN(time), MAX(time) FROM movieapp_log_avro;
```

Exercise 3 – Extract facts using Hive

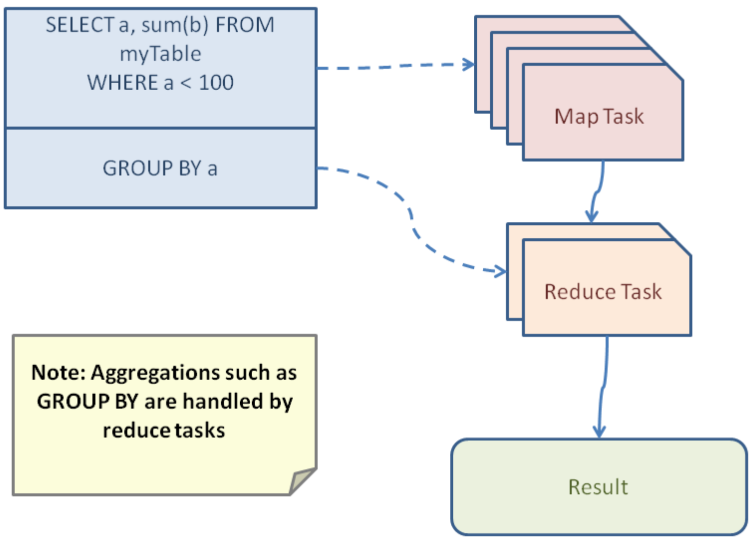
Hive allows for the manipulation of data in HDFS using a variant of SQL. This makes it excellent for transforming and consolidating data for load into a relational database. In this exercise you will use HiveQL to filter and aggregate click data to build facts about user's movie preferences. The query results will be saved in a staging table used to populate the Oracle Database.

The moveapp_log_avro table contains an activity column. Activity states are as follows:

1. RATE_MOVIE
2. COMPLETED_MOVIE
3. PAUSE_MOVIE
4. START_MOVIE
5. BROWSE_MOVIE
6. LIST_MOVIE
7. SEARCH_MOVIE
8. LOGIN
9. LOGOUT
10. INCOMPLETE_MOVIE
11. PURCHASE_MOVIE

Hive maps queries into MapReduce jobs, simplifying the process of querying large datasets in HDFS. HiveQL statements can be mapped to phases of the MapReduce framework. As illustrated in the following figure, selection and transformation operations occur in map tasks, while aggregation is handled by reducers. Join operations are flexible: they can be performed in the reducer or mappers depending on the size of the leftmost table.

Hive SELECT With WHERE Clause


SELECT a, sum(b) FROM myTable WHERE a < 100
GROUP BY a
Map Task
Reduce Task
Result
Note: Aggregations such as GROUP BY are handled by reduce tasks

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;
```

Select the movie ratings made by a user. And, consider the following: what if a user rates the same movie multiple times? In this scenario, you should only load the user's most recent movie rating.

In Oracle Database 12c, you can use a windowing function. However, HiveQL does not provide sophisticated analytic functions. Instead, you must use an inner join to compute the result.

Note: Joins occur **before** WHERE clauses. To restrict the output of a join, a requirement should be in the WHERE clause, otherwise it should be in the JOIN clause.

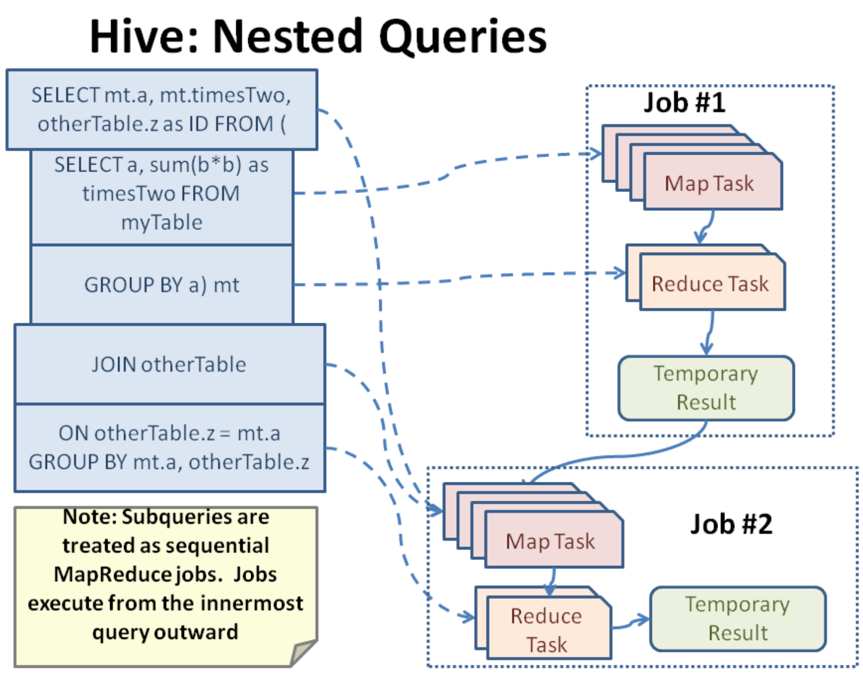
2. Write a query to select the customer ID, movie ID, recommended state and most recent rating for each movie.

```
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. First, create the staging table:

```
hive>  CREATE TABLE movieapp_log_stage (
 custId INT,
 movieId INT,
 genreId INT,
 time  STRING,
 recommended INT,
 activity INT,
 rating INT,
 sales FLOAT
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

Hive queries that involve nested queries are translated into sequential MapReduce jobs which use temporary tables to store intermediate results. The following figure illustrates how statements in a nested query are transformed into map and reduce tasks. As before, dashed lines show logical mapping and solid lines define data flow.


Hive: Nested Queries
SELECT mt.a, mt.timesTwo, otherTable.z as ID FROM (
SELECT a, sum(b*b) as timesTwo FROM myTable
GROUP BY a) mt
JOIN otherTable
ON otherTable.z = mt.a GROUP BY mt.a, otherTable.z
Note: Subqueries are treated as sequential MapReduce jobs. Jobs execute from the innermost query outward
Job #1
Map Task
Reduce Task
Temporary Result
Job #2
Map Task
Reduce Task
Temporary Result

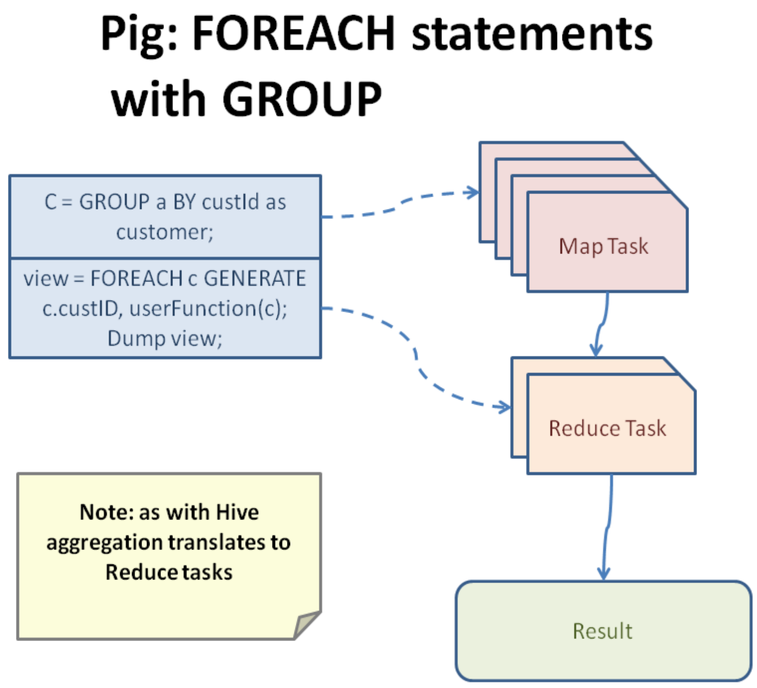
4. Next, load the results of the queries into the staging table:

```
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;
```

Optional: Extract sessions using Pig

While the SQL semantics of HiveQL are useful for aggregation and projection, some analysis is better described as the flow of data through a series of sequential operations. For these situations, Pig Latin provides a convenient way of implementing dataflows over data stored in HDFS.

Pig Latin statements are translated into a sequence of MapReduce jobs on the execution of any STORE or DUMP command. Job construction is optimized to exploit as much parallelism as possible, and much like Hive, temporary storage is used to hold intermediate results. As with Hive, aggregation occurs largely in the reduce tasks. Map tasks handle Pig's FOREACH and LOAD, and GENERATE statements. The EXPLAIN command will show the execution plan for any Pig Latin script. As of Pig 0.10, the ILLUSTRATE command will provide sample results for each stage of the execution plan.


Pig: FOREACH statements with GROUP
C = GROUP a BY custId as customer;
view = FOREACH c GENERATE c.custID, userFunction(c); Dump view;
Map Task
Reduce Task
Result
Note: as with Hive aggregation translates to Reduce tasks

In this exercise you will learn basic Pig Latin semantics and about the fundamental types in Pig Latin, Data Bags and Tuples.

1. Start the Grunt shell and execute the following statements to set up a dataflow with the clickstream data. Note: Pig Latin statements are assembled into MapReduce jobs which are launched at execution of a DUMP or STORE statement.

```
pig

REGISTER /usr/lib/pig/piggybank.jar
REGISTER /usr/lib/pig/lib/avro-1.7.4.jar
REGISTER /usr/lib/pig/lib/json-simple-1.1.jar
REGISTER /usr/lib/pig/lib/snappy-java-1.0.4.1.jar
REGISTER /usr/lib/pig/lib/jackson-core-asl-1.8.8.jar
REGISTER /usr/lib/pig/lib/jackson-mapper-asl-1.8.8.jar

applogs = LOAD '/user/oracle/moviework/applog_avro'
        USING org.apache.pig.piggybank.storage.avro.AvroStorage(
          'no_schema_check',
          'schema_file',
        'hdfs://bigdatalite.localdomain/user/oracle/moviework/schemas/activity.avsc');

DESCRIBE applogs;

log_sample = SAMPLE applogs 0.001;

DESCRIBE log_sample;

DUMP log_sample;
```

log_sample is a **bag** of **tuples**, two of the basic Pig Latin data types. Put concisely:

- 1227714 is a **field**
- (1227714,2012-09-30:22:56:03,6,,39451,6,) is a **tuple**, an ordered collection of fields
- {(1227714,2012-09-30:22:56:03,6,,39451,6,), (1070227,2012-09-30:19:09:32,8,,,,)} is a **bag**, a collection of tuples

2. Group the log_sample by movie and dump the resulting bag.

```
grunt> movie_logs = GROUP log_sample BY movieId;

grunt> dump movie_logs;
```

movie_logs is a **relation**, in this case it is an **outer bag** which contains **tuples** of (data field, bag). Operations on these **inner bags** can be parallelized using the data field member of the tuple as keys in a MapReduce job. When combined with user defined functions, Pig allows for succinct expression of complex analysis.

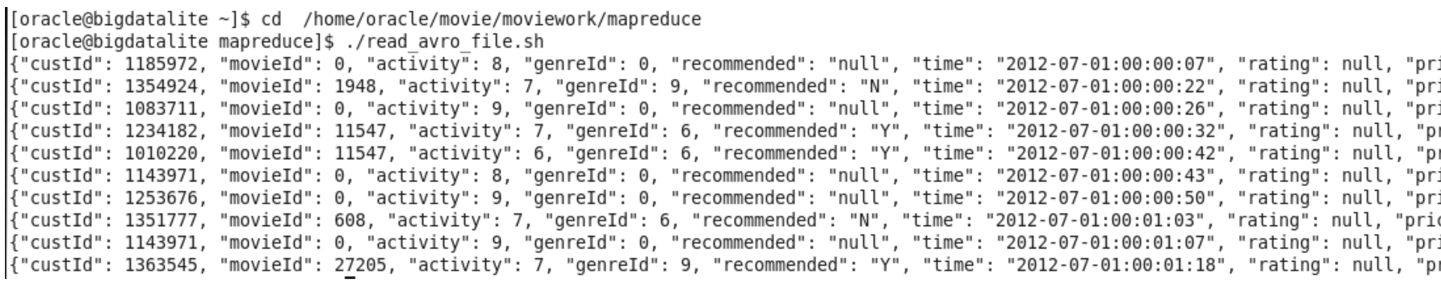
3. Add a GROUP BY statement to the sessionize.pig script to process the clickstream data into user sessions.

Solutions

Exercise 1 – Load Avro data into HDFS

1. Inspect the Avro application log:

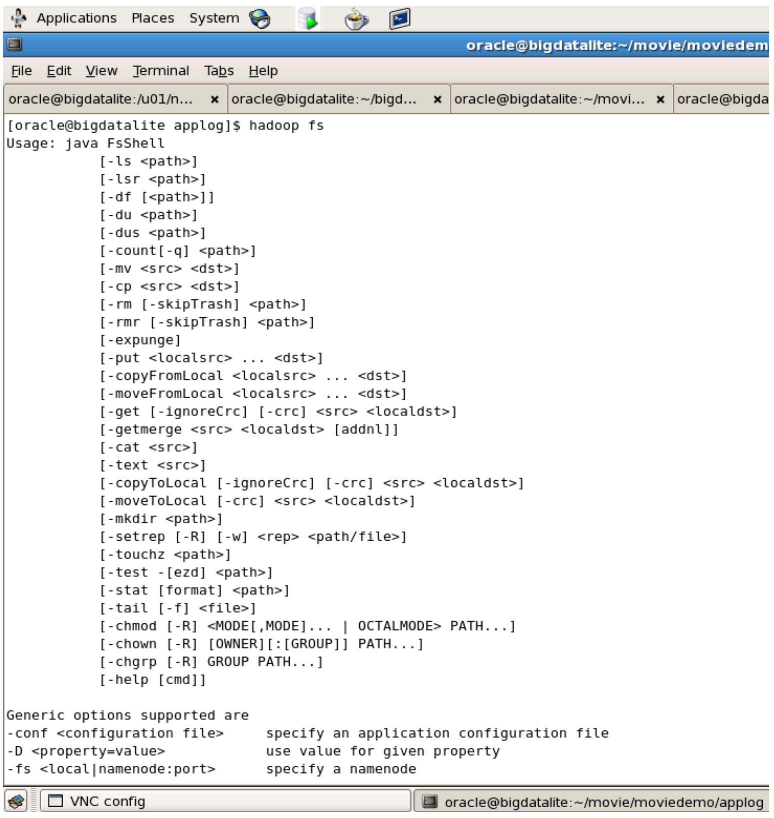
```
cd  /home/oracle/movie/moviework/mapreduce
./read_avro_file.sh
```



```
[oracle@bigdatalite ~]$ cd  /home/oracle/movie/moviework/mapreduce
[oracle@bigdatalite mapreduce]$ ./read_avro_file.sh
{"custId": 1185972, "movieId": 0, "activity": 8, "genreId": 0, "recommended": "null", "time": "2012-07-01:00:00:07", "rating": null, "pri
{"custId": 1354924, "movieId": 1948, "activity": 7, "genreId": 9, "recommended": "N", "time": "2012-07-01:00:00:22", "rating": null, "pri
{"custId": 1083711, "movieId": 0, "activity": 9, "genreId": 0, "recommended": "null", "time": "2012-07-01:00:00:26", "rating": null, "pri
{"custId": 1234182, "movieId": 11547, "activity": 7, "genreId": 6, "recommended": "Y", "time": "2012-07-01:00:00:32", "rating": null, "pr
{"custId": 1010220, "movieId": 11547, "activity": 6, "genreId": 6, "recommended": "Y", "time": "2012-07-01:00:00:42", "rating": null, "pr
{"custId": 1143971, "movieId": 0, "activity": 8, "genreId": 0, "recommended": "null", "time": "2012-07-01:00:00:43", "rating": null, "pri
{"custId": 1253676, "movieId": 0, "activity": 9, "genreId": 0, "recommended": "null", "time": "2012-07-01:00:00:50", "rating": null, "pri
{"custId": 1351777, "movieId": 608, "activity": 7, "genreId": 6, "recommended": "N", "time": "2012-07-01:00:01:03", "rating": null, "pric
{"custId": 1143971, "movieId": 0, "activity": 9, "genreId": 0, "recommended": "null", "time": "2012-07-01:00:01:07", "rating": null, "pri
{"custId": 1363545, "movieId": 27205, "activity": 7, "genreId": 9, "recommended": "Y", "time": "2012-07-01:00:01:18", "rating": null, "pr
```

2. Review the commands available for the Hadoop Distributed File System and copy the gzipped file into HDFS:

```
hadoop fs
hadoop fs -put movieapp_3months.avro /user/oracle/moviework/applog_avro
```

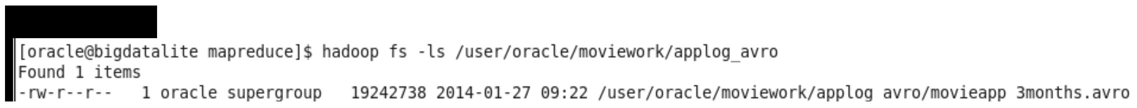

Applications Places System
oracle@bigdatalite:~/movie/moviedem
File Edit View Terminal Tabs Help
oracle@bigdatalite:/u01/n...
oracle@bigdatalite:~/bigd...
oracle@bigdatalite:~/movi...
oracle@bigda
VNC config
oracle@bigdatalite:~/movie/moviedemo/applog

```
[oracle@bigdatalite applog]$ hadoop fs
Usage: java FsShell
           [-ls <path>]
           [-lsr <path>]
           [-df [<path>]]
           [-du <path>]
           [-dus <path>]
           [-count[-q] <path>]
           [-mv <src> <dst>]
           [-cp <src> <dst>]
           [-rm [-skipTrash] <path>]
           [-rmr [-skipTrash] <path>]
           [-expunge]
           [-put <localsrc> ... <dst>]
           [-copyFromLocal <localsrc> ... <dst>]
           [-moveFromLocal <localsrc> ... <dst>]
           [-get [-ignoreCrc] [-crc] <src> <localdst>]
           [-getmerge <src> <localdst> [addnl]]
           [-cat <src>]
           [-text <src>]
           [-copyToLocal [-ignoreCrc] [-crc] <src> <localdst>]
           [-moveToLocal [-crc] <src> <localdst>]
           [-mkdir <path>]
           [-setrep [-R] [-w] <rep> <path/file>]
           [-touchz <path>]
           [-test -[ezd] <path>]
           [-stat [format] <path>]
           [-tail [-f] <file>]
           [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
           [-chown [-R] [OWNER][:[GROUP]] PATH...]
           [-chgrp [-R] GROUP PATH...]
           [-help [cmd]]

Generic options supported are
-conf <configuration file>     specify an application configuration file
-D <property=value>            use value for given property
-fs <local|namenode:port>      specify a namenode
```

3. Verify the copy by listing the directory contents in HDFS:

```
hadoop fs -ls /user/oracle/moviework/applog_avro
```



```
[oracle@bigdatalite mapreduce]$ hadoop fs -ls /user/oracle/moviework/applog_avro
Found 1 items
-rw-r--r--   1 oracle supergroup   19242738 2014-01-27 09:22 /user/oracle/moviework/applog avro/movieapp 3months.avro
```

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

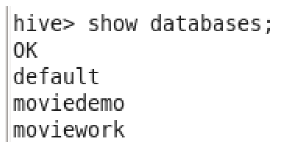
1. Enter the Hive command line by typing **hive** at the Linux prompt:

```
hive
```

2. Create a new hive database called **moviework.** Ensure that the database has been successfully created:

```
hive> create database moviework;
hive> show databases;
```

Result:

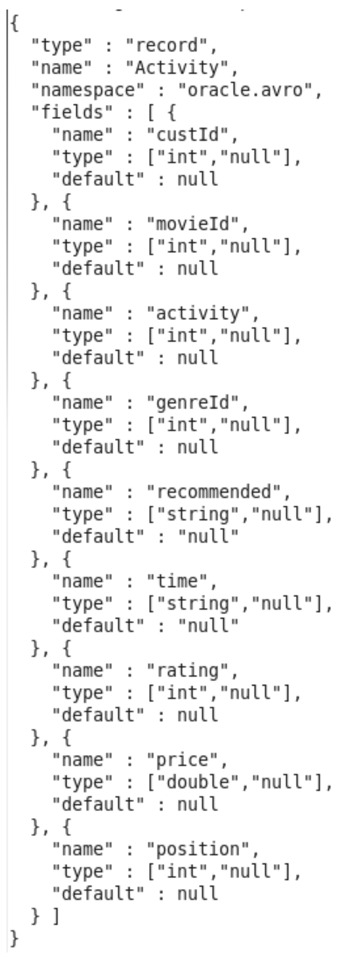


```
hive> show databases;
OK
default
moviedemo
moviework
```

3. Review the schema definition for the avro file and then define a table using that schema file:

```
hadoop fs -cat moviework/schemas/activity.avsc:
```

Result:



```
{
  "type" : "record",
  "name" : "Activity",
  "namespace" : "oracle.avro",
  "fields" : [ {
    "name" : "custId",
    "type" : ["int","null"],
    "default" : null
  }, {
    "name" : "movieId",
    "type" : ["int","null"],
    "default" : null
  }, {
    "name" : "activity",
    "type" : ["int","null"],
    "default" : null
  }, {
    "name" : "genreId",
    "type" : ["int","null"],
    "default" : null
  }, {
    "name" : "recommended",
    "type" : ["string","null"],
    "default" : "null"
  }, {
    "name" : "time",
    "type" : ["string","null"],
    "default" : "null"
  }, {
    "name" : "rating",
    "type" : ["int","null"],
    "default" : null
  }, {
    "name" : "price",
    "type" : ["double","null"],
    "default" : null
  }, {
    "name" : "position",
    "type" : ["int","null"],
    "default" : null
  } ]
}
```

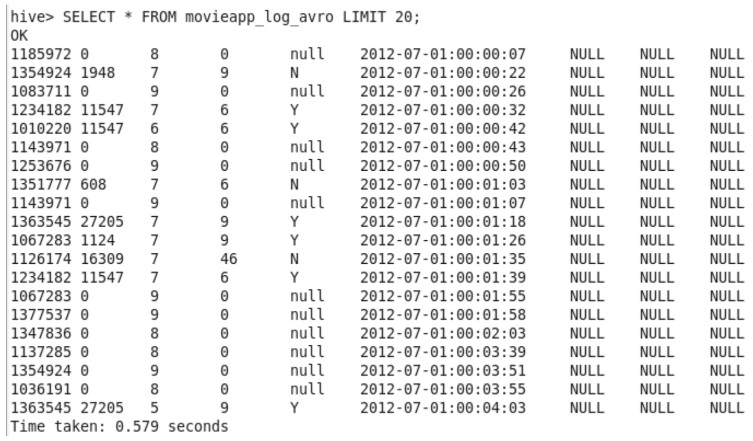
4. To create a table in a database, you can either fully qualify the table name (i.e. prepend the database to the name of the table) or you can designate that you want all DDL and DML operations to apply to a specific database. For simplicity, you will apply subsequent operations to the moviework database:

```
hive> use moviework;
```

Now create the external table **movieapp_log_avro** that uses the activit Avro schema. Select the first 20 rows:

```
hive>  CREATE EXTERNAL TABLE movieapp_log_avro
  ROW FORMAT
  SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
  WITH SERDEPROPERTIES
('avro.schema.url'='hdfs://bigdatalite.localdomain/user/oracle/moviework/schemas/activity.avsc')
  STORED AS
    INPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
    OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
  LOCATION '/user/oracle/moviework/applog_avro';

SELECT * FROM movieapp_log_avro LIMIT 20;
```



```
hive> SELECT * FROM movieapp_log_avro LIMIT 20;
OK
1185972 0       8       0       null    2012-07-01:00:00:07     NULL    NULL    NULL
1354924 1948    7       9       N       2012-07-01:00:00:22     NULL    NULL    NULL
1083711 0       9       0       null    2012-07-01:00:00:26     NULL    NULL    NULL
1234182 11547   7       6       Y       2012-07-01:00:00:32     NULL    NULL    NULL
1010220 11547   6       6       Y       2012-07-01:00:00:42     NULL    NULL    NULL
1143971 0       8       0       null    2012-07-01:00:00:43     NULL    NULL    NULL
1253676 0       9       0       null    2012-07-01:00:00:50     NULL    NULL    NULL
1351777 608     7       6       N       2012-07-01:00:01:03     NULL    NULL    NULL
1143971 0       9       0       null    2012-07-01:00:01:07     NULL    NULL    NULL
1363545 27205   7       9       Y       2012-07-01:00:01:18     NULL    NULL    NULL
1067283 1124    7       9       Y       2012-07-01:00:01:26     NULL    NULL    NULL
1126174 16309   7       46      N       2012-07-01:00:01:35     NULL    NULL    NULL
1234182 11547   7       6       Y       2012-07-01:00:01:39     NULL    NULL    NULL
1067283 0       9       0       null    2012-07-01:00:01:55     NULL    NULL    NULL
1377537 0       9       0       null    2012-07-01:00:01:58     NULL    NULL    NULL
1347836 0       8       0       null    2012-07-01:00:02:03     NULL    NULL    NULL
1137285 0       8       0       null    2012-07-01:00:03:39     NULL    NULL    NULL
1354924 0       9       0       null    2012-07-01:00:03:51     NULL    NULL    NULL
1036191 0       8       0       null    2012-07-01:00:03:55     NULL    NULL    NULL
1363545 27205   5       9       Y       2012-07-01:00:04:03     NULL    NULL    NULL
Time taken: 0.579 seconds
```

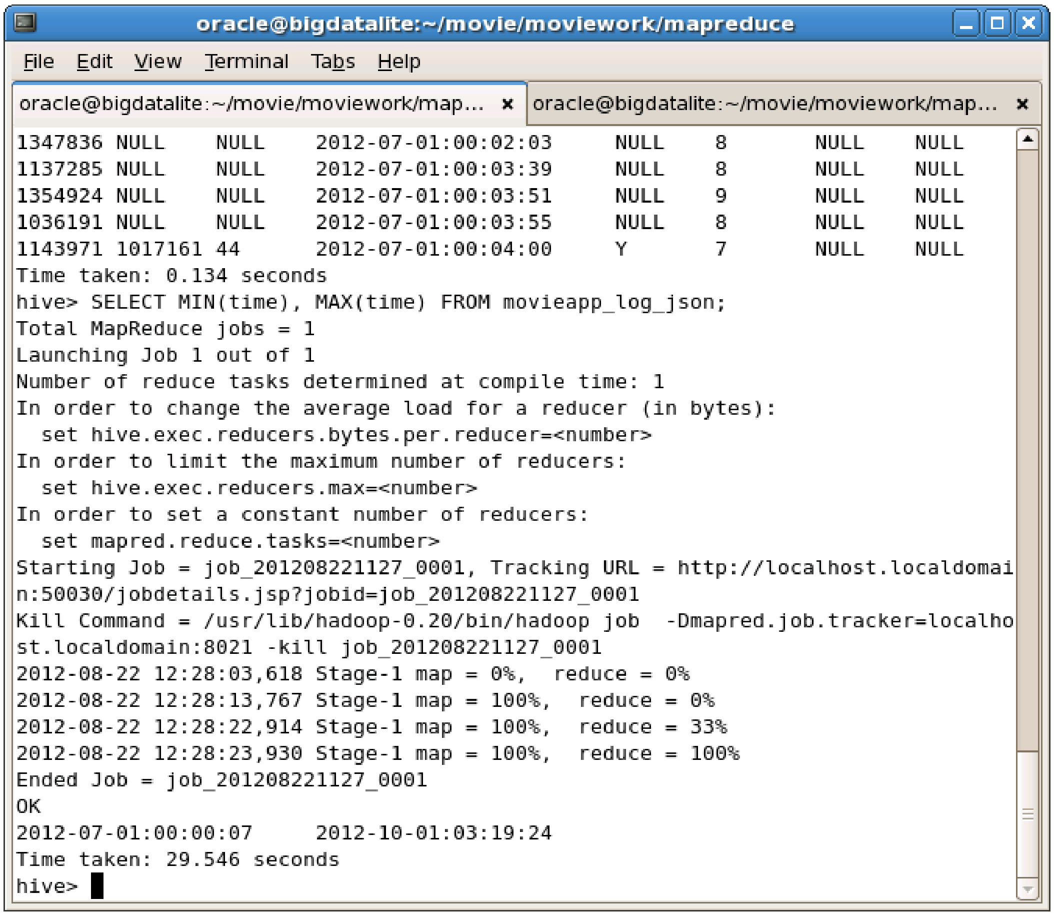
5. Write a query in the Hive command line that returns the first 5 rows from the table. After reviewing the results, drop the table:

```
hive> SELECT * FROM movieapp_log_avro LIMIT 5;
hive> drop table movieapp_log_avro;
```

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

```
hive>  SELECT MIN(time), MAX(time) FROM movieapp_log_avro
```

Result:

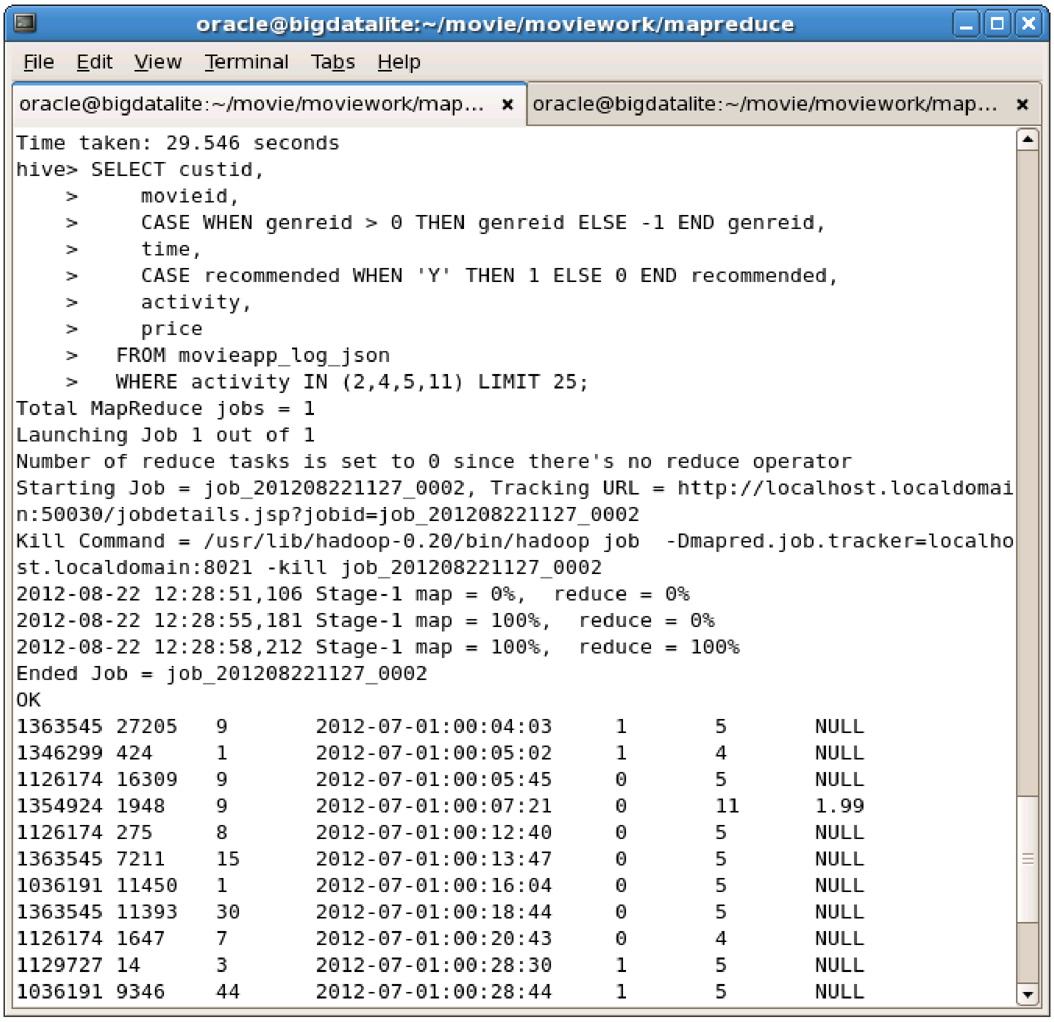

```
1347836 NULL    NULL    2012-07-01:00:02:03     NULL    8       NULL    NULL
1137285 NULL    NULL    2012-07-01:00:03:39     NULL    8       NULL    NULL
1354924 NULL    NULL    2012-07-01:00:03:51     NULL    9       NULL    NULL
1036191 NULL    NULL    2012-07-01:00:03:55     NULL    8       NULL    NULL
1143971 1017161 44      2012-07-01:00:04:00     Y       7       NULL    NULL
Time taken: 0.134 seconds
hive> SELECT MIN(time), MAX(time) FROM movieapp_log_json;
Total MapReduce 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 mapred.reduce.tasks=<number>
Starting Job = job_201208221127_0001, Tracking URL = http://localhost.localdomai
n:50030/jobdetails.jsp?jobid=job_201208221127_0001
Kill Command = /usr/lib/hadoop-0.20/bin/hadoop job  -Dmapred.job.tracker=localho
st.localdomain:8021 -kill job_201208221127_0001
2012-08-22 12:28:03,618 Stage-1 map = 0%,  reduce = 0%
2012-08-22 12:28:13,767 Stage-1 map = 100%,  reduce = 0%
2012-08-22 12:28:22,914 Stage-1 map = 100%,  reduce = 33%
2012-08-22 12:28:23,930 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201208221127_0001
OK
2012-07-01:00:00:07     2012-10-01:03:19:24
Time taken: 29.546 seconds
hive> 
```

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;
```

Result:

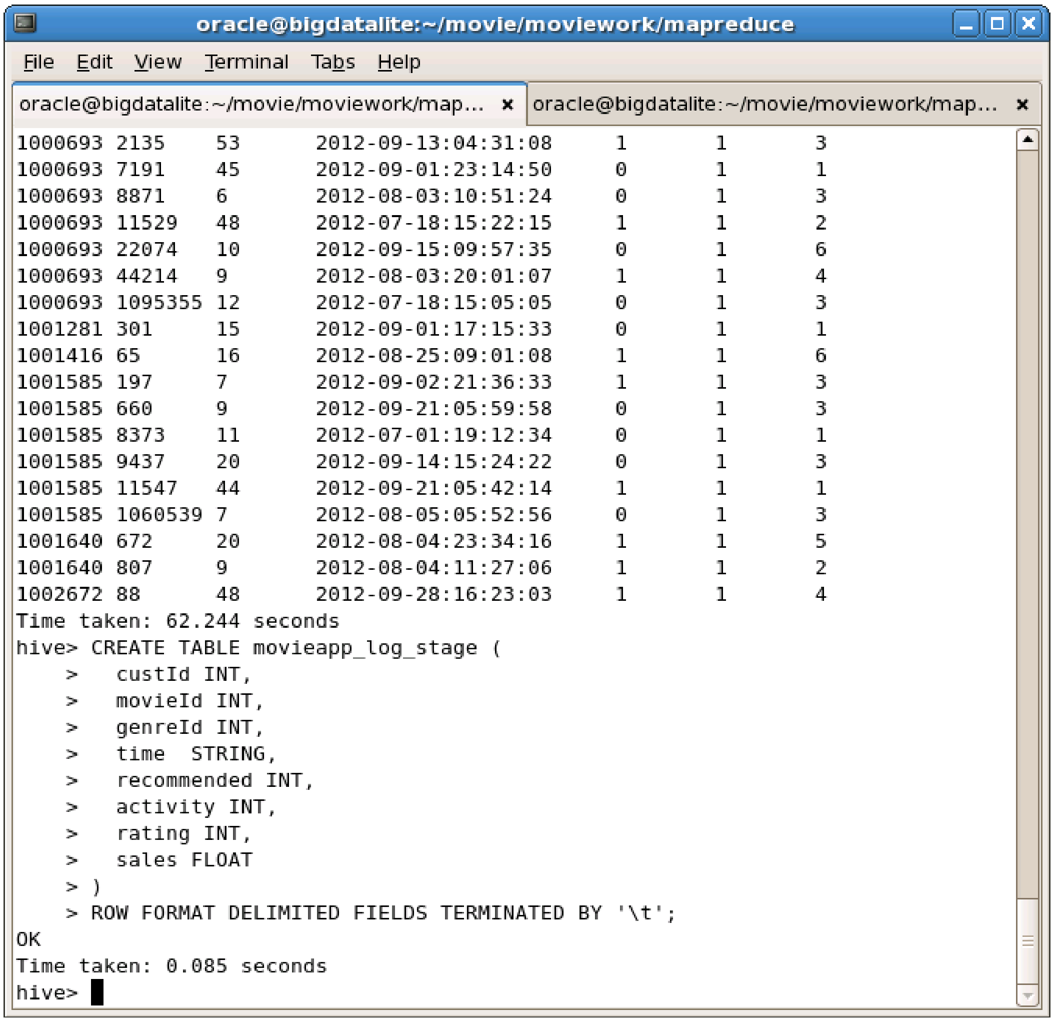

oracle@bigdatalite:~/movie/moviework/mapreduce
File Edit View Terminal Tabs Help
oracle@bigdatalite:~/movie/moviework/map...
oracle@bigdatalite:~/movie/moviework/map...

```
Time taken: 29.546 seconds
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_json
    >   WHERE activity IN (2,4,5,11) LIMIT 25;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201208221127_0002, Tracking URL = http://localhost.localdomai
n:50030/jobdetails.jsp?jobid=job_201208221127_0002
Kill Command = /usr/lib/hadoop-0.20/bin/hadoop job  -Dmapred.job.tracker=localho
st.localdomain:8021 -kill job_201208221127_0002
2012-08-22 12:28:51,106 Stage-1 map = 0%,  reduce = 0%
2012-08-22 12:28:55,181 Stage-1 map = 100%,  reduce = 0%
2012-08-22 12:28:58,212 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201208221127_0002
OK
1363545 27205   9       2012-07-01:00:04:03     1       5       NULL
1346299 424     1       2012-07-01:00:05:02     1       4       NULL
1126174 16309   9       2012-07-01:00:05:45     0       5       NULL
1354924 1948    9       2012-07-01:00:07:21     0       11      1.99
1126174 275     8       2012-07-01:00:12:40     0       5       NULL
1363545 7211    15      2012-07-01:00:13:47     0       5       NULL
1036191 11450   1       2012-07-01:00:16:04     0       5       NULL
1363545 11393   30      2012-07-01:00:18:44     0       5       NULL
1126174 1647    7       2012-07-01:00:20:43     0       4       NULL
1129727 14      3       2012-07-01:00:28:30     1       5       NULL
1036191 9346    44      2012-07-01:00:28:44     1       5       NULL
```

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;
```

Result:



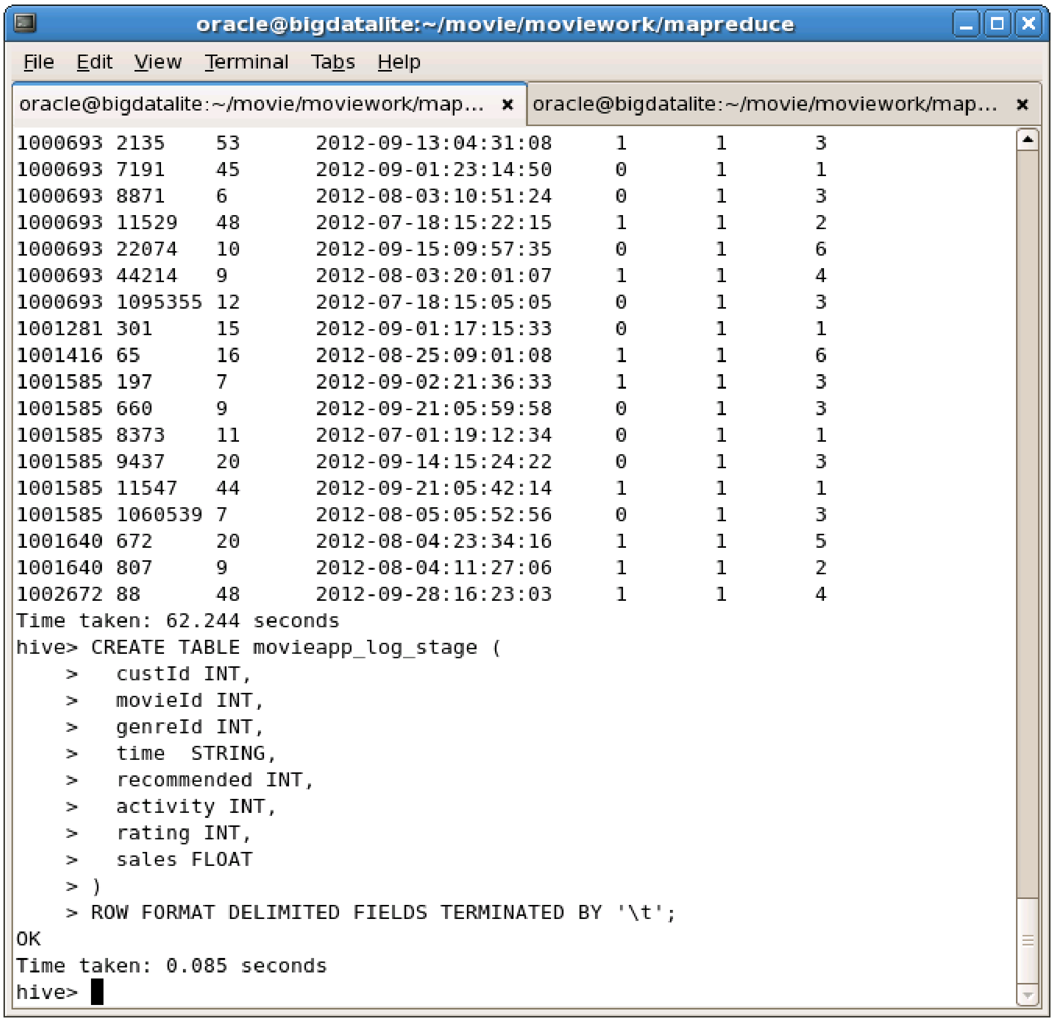
oracle@bigdatalite:~/movie/moviework/mapreduce

```
1000693 2135    53      2012-09-13:04:31:08     1       1       3
1000693 7191    45      2012-09-01:23:14:50     0       1       1
1000693 8871    6       2012-08-03:10:51:24     0       1       3
1000693 11529   48      2012-07-18:15:22:15     1       1       2
1000693 22074   10      2012-09-15:09:57:35     0       1       6
1000693 44214   9       2012-08-03:20:01:07     1       1       4
1000693 1095355 12      2012-07-18:15:05:05     0       1       3
1001281 301     15      2012-09-01:17:15:33     0       1       1
1001416 65      16      2012-08-25:09:01:08     1       1       6
1001585 197     7       2012-09-02:21:36:33     1       1       3
1001585 660     9       2012-09-21:05:59:58     0       1       3
1001585 8373    11      2012-07-01:19:12:34     0       1       1
1001585 9437    20      2012-09-14:15:24:22     0       1       3
1001585 11547   44      2012-09-21:05:42:14     1       1       1
1001585 1060539 7       2012-08-05:05:52:56     0       1       3
1001640 672     20      2012-08-04:23:34:16     1       1       5
1001640 807     9       2012-08-04:11:27:06     1       1       2
1002672 88      48      2012-09-28:16:23:03     1       1       4
Time taken: 62.244 seconds
hive> CREATE TABLE movieapp_log_stage (
    >   custId INT,
    >   movieId INT,
    >   genreId INT,
    >   time  STRING,
    >   recommended INT,
    >   activity INT,
    >   rating INT,
    >   sales FLOAT
    > )
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.085 seconds
hive>
```

3. Load the results of the previous two queries into a staging table. First, create the staging table:

```
hive>  CREATE TABLE movieapp_log_stage (
  custId INT,
  movieId INT,
  genreId INT,
  time  STRING,
  recommended INT,
  activity INT,
  rating INT,
  sales FLOAT
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

Result:



oracle@bigdatalite:~/movie/moviework/mapreduce

File Edit View Terminal Tabs Help
oracle@bigdatalite:~/movie/moviework/map...
oracle@bigdatalite:~/movie/moviework/map...

```
1000693 2135    53      2012-09-13:04:31:08     1       1       3
1000693 7191    45      2012-09-01:23:14:50     0       1       1
1000693 8871    6       2012-08-03:10:51:24     0       1       3
1000693 11529   48      2012-07-18:15:22:15     1       1       2
1000693 22074   10      2012-09-15:09:57:35     0       1       6
1000693 44214   9       2012-08-03:20:01:07     1       1       4
1000693 1095355 12      2012-07-18:15:05:05     0       1       3
1001281 301     15      2012-09-01:17:15:33     0       1       1
1001416 65      16      2012-08-25:09:01:08     1       1       6
1001585 197     7       2012-09-02:21:36:33     1       1       3
1001585 660     9       2012-09-21:05:59:58     0       1       3
1001585 8373    11      2012-07-01:19:12:34     0       1       1
1001585 9437    20      2012-09-14:15:24:22     0       1       3
1001585 11547   44      2012-09-21:05:42:14     1       1       1
1001585 1060539 7       2012-08-05:05:52:56     0       1       3
1001640 672     20      2012-08-04:23:34:16     1       1       5
1001640 807     9       2012-08-04:11:27:06     1       1       2
1002672 88      48      2012-09-28:16:23:03     1       1       4
Time taken: 62.244 seconds
hive> CREATE TABLE movieapp_log_stage (
    >   custId INT,
    >   movieId INT,
    >   genreId INT,
    >   time  STRING,
    >   recommended INT,
    >   activity INT,
    >   rating INT,
    >   sales FLOAT
    > )
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.085 seconds
hive> 
```

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;
```