

4.Using Movie rating data,Develop the Queries in hive for the following

Step1: Create a text file moviedetail in the root directory, type the data in the file. Then move the file to the HDFS directory in path user/hivedata and verify the file has been copied.

```
$vi moviedetail
```

1111008 1	1	5	Y	12-3-2016	12:12:12	1	Null	Null
1111119 2	4	1	N	11-2-2000	11:33:44	5	Null	Null
1212210 6	9	7	Y	04-2-2000	12:44:44	Null	Null	Null
2111008 1	1	5	Y	12-3-2016	12:12:12	3	Null	Null
1511119 2	4	1	N	11-2-2000	11:33:44	Null	Null	Null
1612210 6	9	7	Y	04-2-2000	12:44:44	4	Null	Null

```
$ hadoop fs -mkdir /user/cloudera/hivedata
```

```
$ hadoop fs -put moviedetail /user/cloudera/hivedata
```

```
$ hadoop fs -ls /user/cloudera/hivedata
```

Step2: Launch hive command line shell, create database moviework and create a table.

```
$ hive
```

```
$ create database moviework;
```

```
$ use moviework;
```

```
$ create table movie_details(custId int, movieId int, activity int,  
genreId int, recommended string,  
time string, rating string, price int, position int)  
row format delimited  
fields terminated by '\t'  
stored as textfile  
TBLPROPERTIES('serialization.null.format'='NULL');
```

Step3: Now load the table with the data present in  
/user/cloudera/hivedata/moviedetail using command and display the  
loaded data in table:

```
$ load data inpath '/user/cloudera/hivedata/moviedetail' into table  
movie_details;
```

```
$ select * from movie_details;
```

Step4: Queries a. List all the users who have rated the movies(Users  
who have rated at least one movie)

```
select custID from movie_details where rating is not NULL;
```

b. List all the users with max,min,average ratings they  
have given against any movie.

```
select custID,min(rating),max(rating),avg(rating)
```

```
from movie_details group by custID,movieID;
```

c. List all the Movies with max,min,average ratings  
given by any user.

```
select movieID,min(rating),max(rating),avg(rating) from movie_details  
group by custID,movieID;
```

5.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 users movie preferences. The query results will be saved in a staging table used to populate the Oracle Database.

Step1: Create a text file moviedetail in the root directory, type the data in the file. Then move the file to the HDFS directory in path user/hivedata and verify the file has been copied.

```
$vi moviedetail
```

1111008	1	1	5	Y	12-3-201612:12:12	Null	Null	Null
1111119	2	4	1	N	11-2-200011:33:44	Null	Null	Null
1212210	6	9	7	Y	04-2-200012:44:44	Null	Null	Null
2111008	1	1	5	Y	12-3-201612:12:12	Null	Null	Null
1511119	2	4	1	N	11-2-200011:33:44	Null	Null	Null
1612210	6	9	7	Y	04-2-200012:44:44	Null	Null	Null

```
$ hadoop fs -mkdir /user/cloudera/hivedata
```

```
$ hadoop fs -put moviedetail /user/cloudera/hivedata
```

```
$ hadoop fs -ls /user/cloudera/hivedata
```

Step2: Launch hive command line shell, create database moviework and create a table.

```
$ hive
```

```
$ create database moviework;
```

```
$ use moviework;
```

```
$ create table movie_details(custId int, movieId int, activity int, genreId int, recommended string,  
time string, rating int, price int, position int)  
row format delimited  
fields terminated by '\t'  
stored as textfile;
```

Step3: Now load the table with the data present in /user/cloudera/hivedata/moviedetail using command and display the loaded data in table:

```
$ load data inpath '/user/cloudera/hivedata/moviedetail' into table movie_details;
```

```
$ select * from movie_details;
```

Step4: create an external table(example: movieapp\_log). An external table is created for the conversion text

file to binary file which can then be sent to HDFS through the operating system. Avro system is used for

conversion of text file to binary file.

```
$ CREATE EXTERNAL TABLE movieapp_logs  
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'  
STORED AS INPUTFORMAT 'org.apache.hadoop.hive ql.io.avro.AvroContainerInputFormat'  
OUTPUTFORMAT 'org.apache.hadoop.hive ql.io.avro.AvroContainerOutputFormat'  
tblproperties ('avro.schema.literal'='{  
"name": "my_record",
```

```

"type": "record",
"fields": [
  {"name": "custId", "type": "int"}, {"name": "movieId", "type": "int"},
  {"name": "activity", "type": "int"},
  {"name": "genreId", "type": "int"}, {"name": "recommended", "type": "string"},
  {"name": "time", "type": "string"},
  {"name": "rating", "type": ["int", "null"], "default": "null"},
  {"name": "price", "type": ["int", "null"], "default": "null"},
  {"name": "position", "type": ["int", "null"], "default": "null"} ]});

```

Step5: Now insert the details of internally created table(movie\_details) using insert overwrite command to the external table and view the data

```
$ insert overwrite table movieapp_logs select * from movie_details;
```

```
$ select * from movieapp_logs;
```

Step6: Till now we have created the tables and loaded the data into the tables. Now we are going to start

the query.

Query 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 10 rows.

```
$ SELECT custid,movieid,CASE WHEN genereid> 0 THEN genereid ELSE -1 END genereid,time,CASE  
recommended WHEN 'Y'
```

```
THEN 1 ELSE 0 END recommended,activity,price FROM movieapp_logs WHERE activity IN (2,4,5,11)  
LIMIT 10;
```

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

each movie.

```
$ SELECT m1.custid,m1.movieid,CASE WHEN m1.genereid > 0 THEN m1.genereid ELSE -1 END  
genereid,m1.time,CASE m1.recommended WHEN 'Y' THEN 1 ELSE 0 END
```

```
recommended,m1.activity,m1.rating FROM movieapp_logs m1 JOIN(SELECT custid,movieid,  
CASE WHEN genereid> 0 THEN genereid ELSE -1 END genereid,MAX(time) max_time,activity FROM  
movieapp_logs
```

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

```
AND m1.genereid = m2.genereid AND m1.time = m2.max_time AND m1.activity = 1 AND m2.activity =  
1 ) LIMIT 15;
```

Query 3: find the minimum and maximum time periods that are available in the log file

```
$ SELECT MIN(time), MAX(time) FROM movieapp_logs;
```

6.The moveapp\_log\_json table contains an activity column. Activity states are as follows:

RATE\_MOVIE

COMPLETED\_MOVIE

PAUSE\_MOVIE

START\_MOVIE

BROWSE\_MOVIE

LIST\_MOVIE

SEARCH\_MOVIE

LOGIN

LOGOUT

INCOMPLETE\_MOVIE.

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

2.Next, load the results of the queries into the staging table.

Query1: Load the results of the previous two queries into a staging table.First, create the staging table:

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

Query 2:Next, load the results of the queries into the staging table:

```
$ INSERT OVERWRITE TABLE movieapp_logs_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_logs WHERE activity IN (2,4,5,11)

UNION ALL SELECT m1.custid,m1.movieid,CASE WHEN m1.generelId > 0 THEN m1.generelId ELSE -1
END generelId,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_logs m1 JOIN (SELECT custid,movieid,CASE WHEN generelId>
0 THEN

generelId ELSE -1 END generelId,MAX(time) max_time,activity FROM movieapp_logs GROUP BY custid,
movieid, generelId,

activity) m2 ON ( m1.custid = m2.custid AND m1.movieid = m2.movieid AND m1.generelId =
m2.generelId

AND m1.time = m2.max_time AND m1.activity = 1 AND m2.activity = 1 ) union_result;

```

Query 3: View the staging table.

```
$ select * from movieapp_logs_stage;
```