Mr. Kalyan, Big Data Solution Architect,
Apache Contributor, 11+ years of IT exp, 7+ years of Big Data exp,
Cloudera CCA175 Certified Consultant, IIT Kharagpur, Gold Medalist

REAL-TIME BIGDATA HADOOP PROJECTS MATERIAL @ ORIENIT @

Mr. Kalyan

11+ years of IT exp, 7+ years of Big Data exp,
Gold Medalist, IIT Kharagpur,
Cloudera CCA 175 Hadoop and Spark Certified Consultant, Apache Contributor

(Throughout the World First Certified Cloudera CCA175 Hadoop and Spark Developer)

Kalyan Big Data Projects
Hive Project Assignment Solutions

```
DROP DATABASE IF EXISTS kalyan_projects CASCADE;

CREATE DATABASE IF NOT EXISTS kalyan_projects;

USE kalyan_projects;
```

```
DROP TABLE IF EXISTS kalyan_projects.movies;

CREATE EXTERNAL TABLE IF NOT EXISTS kalyan_projects.movies
(movieid int, title string, genres string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES ("input.regex" = "(.*)::(.*)::(.*)");

LOAD DATA LOCAL INPATH
'/home/orienit/work/kalyan_projects/input/ml-1m/movies.dat'
OVERWRITE INTO TABLE kalyan_projects.movies;

SELECT * FROM kalyan_projects.movies LIMIT 5;
```

Wrong Approach in Hive:

```
CREATE EXTERNAL TABLE IF NOT EXISTS kalyan_projects.movies
(movieid int, title string, genres string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\::'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

```
DROP TABLE IF EXISTS kalyan_projects.ratings;

CREATE EXTERNAL TABLE IF NOT EXISTS kalyan_projects.ratings
(userid int, movieid int, rating int, ts bigint)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES ("input.regex" = "(.*)::(.*)::(.*)::(.*)");

LOAD DATA LOCAL INPATH
'/home/orienit/work/kalyan_projects/input/ml-1m/ratings.dat'
OVERWRITE INTO TABLE kalyan_projects.ratings;

SELECT * FROM kalyan_projects.ratings LIMIT 5;
```

Wrong Approach in Hive:

```
CREATE EXTERNAL TABLE IF NOT EXISTS kalyan_projects.ratings
(userid int, movieid int, rating int, ts bigint)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '::'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

```
DROP TABLE IF EXISTS kalyan_projects.users;

CREATE EXTERNAL TABLE IF NOT EXISTS kalyan_projects.users
(userid int, gender string, age int, occupation string, zipcode bigint)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES ("input.regex" = "(.*)::(.*)::(.*)::(.*)::(.*)");

LOAD DATA LOCAL INPATH
'/home/orienit/work/kalyan_projects/input/ml-1m/users.dat'
OVERWRITE INTO TABLE kalyan_projects.users;

SELECT * FROM kalyan_projects.users LIMIT 5;
```

Wrong Approach in Hive:

```
CREATE EXTERNAL TABLE IF NOT EXISTS kalyan_projects.users
(userid int, gender string, age int, occupation string, zipcode bigint)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '::'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

---

1. Top 10 most viewed movies with their Titles (Ascending or Descending order)

```
CREATE VIEW IF NOT EXISTS movieidcount AS
SELECT movieid, count(movieid) as cnt
FROM kalyan_projects.ratings
GROUP BY movieid;

CREATE VIEW IF NOT EXISTS top10movieids AS
SELECT movieid, cnt FROM movieidcount
ORDER BY cnt DESC
LIMIT 10;
```

(OR)

```
CREATE VIEW IF NOT EXISTS top10movieids AS
SELECT movieid, count(movieid) as cnt
FROM kalyan_projects.ratings
GROUP BY movieid
ORDER BY cnt DESC
LIMIT 10;

DESCRIBE FORMATTED top10movieids;

SELECT * FROM top10movieids;

OK
2858 3428
260   2991
1196 2990
1210 2883
480   2672
```

2028 2653
589 2649
2571 2590
1270 2583
593 2578
Time taken: 40.653 seconds, Fetched: 10 row(s)

SELECT movies.title, top10movieids.cnt
FROM kalyan_projects.movies JOIN top10movieids
ON movies.movieid = top10movieids.movieid;

(or)

SELECT /*+ MAPJOIN(top10movieids) */ movies.title,
top10movieids.cnt
FROM kalyan_projects.movies JOIN top10movieids
ON movies.movieid = top10movieids.movieid;

OK
American Beauty (1999) 3428
Star Wars: Episode IV - A New Hope (1977) 2991
Star Wars: Episode V - The Empire Strikes Back (1980) 2990
Star Wars: Episode VI - Return of the Jedi (1983) 2883
Jurassic Park (1993) 2672
Saving Private Ryan (1998) 2653
Terminator 2: Judgment Day (1991) 2649
Matrix, The (1999) 2590
Back to the Future (1985) 2583
Silence of the Lambs, The (1991) 2578
Time taken: 59.255 seconds, Fetched: 10 row(s)

---

2. Top 20 rated movies (Condition: The movie should be rated/viewed by at least 40 users)

```
CREATE VIEW IF NOT EXISTS top20ratedmovieids AS
SELECT movieid, avg(rating) as avg, count(movieid) as cnt
FROM kalyan_projects.ratings
GROUP BY movieid
HAVING cnt > 40
ORDER BY avg DESC
LIMIT 20;

DESCRIBE FORMATTED top20ratedmovieids;

SELECT * FROM top20ratedmovieids;
```

```
OK
2905 4.608695652173913   69
2019 4.560509554140127   628
318  4.554557700942973   2227
858  4.524966261808367   2223
745  4.52054794520548    657
50   4.517106001121705   1783
527  4.510416666666667   2304
1148 4.507936507936508   882
922  4.491489361702127   470
1198 4.477724741447892   2514
904  4.476190476190476   1050
1178 4.473913043478261   230
260  4.453694416583082   2991
1212 4.452083333333333   480
```

750 4.4498902706656915 1367
720 4.426940639269406 438
1207 4.425646551724138 928
3435 4.415607985480944 551
912 4.412822049131217 1669
670 4.410714285714286 56
Time taken: 41.752 seconds, Fetched: 20 row(s)

SELECT movies.title, top20ratedmovieids.avg, top20ratedmovieids.cnt
FROM kalyan_projects.movies JOIN top20ratedmovieids
ON movies.movieid = top20ratedmovieids.movieid;

(or)

SELECT /*+ MAPJOIN(top20ratedmovieids) */ movies.title,
top20ratedmovieids.avg, top20ratedmovieids.cnt
FROM kalyan_projects.movies JOIN top20ratedmovieids
ON movies.movieid = top20ratedmovieids.movieid;

OK
Sanjuro (1962) 4.608695652173913 69
Seven Samurai (The Magnificent Seven) (Shichinin no samurai) (1954)
4.560509554140127 628
Shawshank Redemption, The (1994) 4.554557700942973 2227
Godfather, The (1972) 4.524966261808367 2223
Close Shave, A (1995) 4.52054794520548 657
Usual Suspects, The (1995) 4.517106001121705 1783
Schindler's List (1993) 4.510416666666667 2304
Wrong Trousers, The (1993) 4.507936507936508 882
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950) 4.491489361702127
470
Raiders of the Lost Ark (1981) 4.477724741447892 2514
Rear Window (1954) 4.476190476190476 1050

Paths of Glory (1957) 4.473913043478261 230
Star Wars: Episode IV - A New Hope (1977) 4.453694416583082 2991
Third Man, The (1949) 4.452083333333333 480
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963) 4.44989027066569151367
Wallace & Gromit: The Best of Aardman Animation (1996) 4.426940639269406 438
To Kill a Mockingbird (1962) 4.425646551724138 928
Double Indemnity (1944) 4.415607985480944 551
Casablanca (1942) 4.412822049131217 1669
World of Apu, The (Apur Sansar) (1959) 4.410714285714286 56
Time taken: 61.678 seconds, Fetched: 20 row(s)

---

3. Partition the Users data based on 3 Age Groups (Condition: group1: 1 – 20, group2: 21 – 40, group3: 41+)

```
DROP TABLE IF EXISTS kalyan_projects.usergroups;

CREATE EXTERNAL TABLE IF NOT EXISTS
kalyan_projects.usergroups (userid int, gender string, age int, occupation
int, zipcode bigint)
PARTITIONED BY (agegroup string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

FROM kalyan_projects.users
INSERT OVERWRITE TABLE kalyan_projects.usergroups
PARTITION(agegroup = 'group1') SELECT * WHERE age >= 1 AND age
<= 20
INSERT OVERWRITE TABLE kalyan_projects.usergroups
PARTITION(agegroup = 'group2') SELECT * WHERE age >= 21 AND
age <= 40
INSERT OVERWRITE TABLE kalyan_projects.usergroups
PARTITION(agegroup = 'group3') SELECT * WHERE age >= 41;
```

---

4. Find the Users whose `Occupation is Programmer or Scientist` and `Age is between 25 – 40` then Partition the Users based on `Gender and Occupation`

```
DROP TABLE IF EXISTS kalyan_projects.userbased;

CREATE EXTERNAL TABLE IF NOT EXISTS
kalyan_projects.userbased (userid int, age int, zipcode bigint)
PARTITIONED BY (gender string, occupation string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

set hive.exec.dynamic.partition.mode=nonstrict;

INSERT OVERWRITE TABLE kalyan_projects.userbased
PARTITION(gender, occupation)
SELECT userid, age, zipcode, gender,
CASE
 WHEN occupation = 12 THEN 'Programmer'
 WHEN occupation = 15 THEN 'Scientist'
END
FROM kalyan_projects.users
WHERE (occupation = 12 OR occupation = 15) AND (age >= 25 AND
age <= 40);
```
