CSCI 585 Summer 2019 HW2

Chin-Chou Ko
USC ID: 1511072586

Movie Review Application Database

A database for a movie review application consists of the following tables:

- users (**id**, name, date_of_birth).
- movies (**id**, name, genre, release_date)
- reviews (**user_id, movie_id**, rating, comment)
- actors (**id**, name, gender, date_of_birth)
- lead (**actor_id, movie_id**)

Table Creation Queries (using MYSQL):

```
CREATE DATABASE HW2;
USE HW2;

CREATE TABLE users (id int not null AUTO_INCREMENT, name VARCHAR(35), date_of_birth
date, PRIMARY KEY(id));

INSERT into users (name, date_of_birth) values ('Johnny', '1983-04-15');
INSERT into users (name, date_of_birth) values ('Apple', '1999-03-07');
INSERT into users (name, date_of_birth) values ('Mac', '1993-05-05');
INSERT into users (name, date_of_birth) values ('Jack', '1963-01-15');
INSERT into users (name, date_of_birth) values ('Chris Jackson', '1933-01-11');

SELECT * FROM users;
```

```
MySQL [HW2]> SELECT * FROM users;
+----+---------------+---------------+
| id | name          | date_of_birth |
+----+---------------+---------------+
|  1 | Johnny        | 1983-04-15    |
|  2 | Apple         | 1999-03-07    |
|  3 | Mac           | 1993-05-05    |
|  4 | Jack          | 1963-01-15    |
|  5 | Chris Jackson | 1933-01-11    |
+----+---------------+---------------+
5 rows in set (0.04 sec)
```

```
CREATE TABLE movies (id int not null AUTO_INCREMENT, name VARCHAR(35), genre
VARCHAR(35), release_date date, PRIMARY KEY(id));

INSERT into movies (name, genre, release_date) values ('POKEMON', 'animation' '2019-05-
22');
INSERT into movies (name, genre, release_date) values ('Titanic', 'romance', '1995-03-03');
```

INSERT into movies (name, genre, release_date) values ('Rush Hour', 'action', '1996-07-05');
INSERT into movies (name, genre, release_date) values ('Earth Battle', 'action', '1999-08-06');
INSERT into movies (name, genre, release_date) values ('Ninja Turtle', 'action', '1993-05-30');

SELECT * FROM movies;

```
MySQL [HW2]> SELECT * FROM movies;
+----+--------------+-----------+--------------+
| id | name         | genre     | release_date |
+----+--------------+-----------+--------------+
|  1 | POKEMON      | animation | 2019-05-22   |
|  2 | Titanic      | romance   | 1995-03-03   |
|  3 | Rush Hour    | action    | 1996-07-05   |
|  4 | Earth Battle | action    | 1999-08-06   |
|  5 | Ninja Turtle | action    | 1993-05-30   |
+----+--------------+-----------+--------------+
5 rows in set (0.04 sec)
```

CREATE TABLE reviews (user_id int not null, movie_id int not null, rating FLOAT, comment TEXT(5000), FOREIGN KEY(user_id) REFERENCES users(id), FOREIGN KEY(movie_id) REFERENCES movies(id),);

INSERT into reviews (user_id, movie_id, rating, comment) values ('1', '1', '7.0', 'just OK');
INSERT into reviews (user_id, movie_id, rating, comment) values ('2', '1', '6.5', 'meh');
INSERT into reviews (user_id, movie_id, rating, comment) values ('1', '2', '9.0', 'masterpiece');
INSERT into reviews (user_id, movie_id, rating, comment) values ('1', '3', '1.0', 'boring');
INSERT into reviews (user_id, movie_id, rating, comment) values ('1', '4', '2.0', 'sleepy');
INSERT into reviews (user_id, movie_id, rating, comment) values ('1', '5', '3.0', 'awful');
INSERT into reviews (user_id, movie_id, rating, comment) values ('4', '2', '1.0', 'for girls');
INSERT into reviews (user_id, movie_id, rating, comment) values ('4', '3', '8.0', 'awesome');
INSERT into reviews (user_id, movie_id, rating, comment) values ('4', '4', '8.0', 'not bad');
INSERT into reviews (user_id, movie_id, rating, comment) values ('5', '2', '7.5', 'great movie!');
INSERT into reviews (user_id, movie_id, rating, comment) values ('3', '3', '1.0', 'I have no idea');
INSERT into reviews (user_id, movie_id, rating, comment) values ('3', '4', '1.0', 'trash');

SELECT * FROM movies;

```
MySQL [HW2]> SELECT * FROM reviews;
+---------+----------+--------+----------------+
| user_id | movie_id | rating | comment        |
+---------+----------+--------+----------------+
|       1 |        1 |      7 | just OK        |
|       2 |        1 |    6.5 | meh            |
|       1 |        2 |      9 | masterpiece    |
|       1 |        3 |      1 | boring         |
|       1 |        4 |      2 | sleepy         |
|       1 |        5 |      3 | awful          |
|       4 |        2 |      1 | for girls      |
|       4 |        3 |      8 | awesome        |
|       4 |        4 |      8 | not bad        |
|       5 |        2 |    7.5 | great movie!   |
|       3 |        3 |      1 | I have no idea |
|       3 |        4 |      1 | trash          |
+---------+----------+--------+----------------+
12 rows in set (0.03 sec)
```

CREATE TABLE actors (id int not null AUTO_INCREMENT, name VARCHAR(35), gender enum('M', 'F'), date_of_birth date, PRIMARY KEY(id));

INSERT into actors (name, gender, date_of_birth) values ('Will', 'M', '1950-01-05');
INSERT into actors (name, gender, date_of_birth) values ('Rose', 'F', '1980-03-02');
INSERT into actors (name, gender, date_of_birth) values ('Felice', 'F', '1998-08-01');
INSERT into actors (name, gender, date_of_birth) values ('Snorlax', 'F', '2038-09-09');

SELECT * FROM actors;

```
MySQL [HW2]> SELECT * FROM actors;
+----+---------+--------+---------------+
| id | name    | gender | date_of_birth |
+----+---------+--------+---------------+
|  1 | Will    | M      | 1950-01-05    |
|  2 | Rose    | F      | 1980-03-02    |
|  3 | Felice  | F      | 1998-08-01    |
|  4 | Snorlax | F      | 2038-09-09    |
+----+---------+--------+---------------+
4 rows in set (0.03 sec)
```

CREATE TABLE lead (actor_id int not null, movie_id int not null, FOREIGN KEY(actor_id) REFERENCES actors(id), FOREIGN KEY(movie_id) REFERENCES movies(id),);

INSERT into lead (actor_id, movie_id) values ('1', '1');
INSERT into lead (actor_id, movie_id) values ('2', '2');
INSERT into lead (actor_id, movie_id) values ('3', '2');
INSERT into lead (actor_id, movie_id) values ('4', '1');

SELECT * FROM lead;

```
MySQL [HW2]> SELECT * FROM lead;
+----------+----------+
| actor_id | movie_id |
+----------+----------+
|        1 |        1 |
|        2 |        2 |
|        3 |        2 |
|        4 |        1 |
+----------+----------+
4 rows in set (0.04 sec)
```

Questions:

1- List the movie ID(s) with most female lead sorted by movie ID(s).

Query:

```
SELECT m.id
FROM movies m JOIN
        (SELECT m.id, SUM(a. gender = 'F') AS female_qty FROM actors a JOIN lead l ON a.id
        = l.actor_id JOIN movies m ON m.id = l.movie_id GROUP BY m.id) sub
ON m.id = sub.id
WHERE sub.female_qty =
        (SELECT MAX(sub.female_qty) FROM (SELECT m.id, SUM(a. gender = 'F') AS
        female_qty FROM actors a JOIN lead l ON a.id = l.actor_id JOIN movies m ON m.id =
        l.movie_id GROUP BY m.id) sub)
ORDER BY m.id;
```

```
MySQL [HW2]> SELECT m.id
    -> FROM movies m JOIN (SELECT m.id, SUM(a. gender = 'F') AS female_qty FROM actors a JOIN lead l ON a.id = l.actor_id JOIN
    -> movies m ON m.id = l.movie_id GROUP BY m.id) sub
    ->
    -> ON m.id = sub.id
    ->
    -> WHERE sub.female_qty = (SELECT MAX(sub.female_qty) FROM (SELECT m.id, SUM(a. gender = 'F') AS female_qty FROM actors a JOIN lead l ON a.id
= l.actor_id JOIN
    -> movies m ON m.id = l.movie_id GROUP BY m.id) sub)
    ->
    -> ORDER BY m.id;
+----+
| id |
+----+
|  2 |
+----+
1 row in set (0.04 sec)
```

Explanation:

First, find the quantity of female lead from each movie, I use SUM function here to calculate it: SUM(a. gender = 'F'). In my test case, movie 1's female lead is 1, movie 2's female lead is 2. Then, choose the MAX quantity of female lead. So the most female lead in my test case is 2. Finally, order by movie id and printout the movie id, so my output is id 2.

2- Find user ‘Jack’ favorite type of movie genre(s) based on his movie review ratings. List the name(s) and genre(s) of all the movie(s) under this/these movie genre(s).

Query:

```
SELECT DISTINCT m.name, m.genre
FROM movies m JOIN
        (SELECT r.rating, m.genre FROM reviews r JOIN movies m ON r.movie_id = m.id JOIN
        users u ON r.user_id = u.id WHERE u.name = ‘Jack’) sub
ON m.genre = sub.genre
WHERE sub.rating =
        (SELECT MAX(sub.rating) FROM(SELECT r.rating, m.genre FROM reviews r JOIN
        movies m ON r.movie_id = m.id JOIN users u ON r.user_id = u.id WHERE u.name =
        ‘Jack’) sub);
```

```
MySQL [HW2]> SELECT DISTINCT m.name, m.genre FROM movies m JOIN  (SELECT r.rating, m.genre FROM reviews r JOIN movies m ON r.movie_id = m.id JOINu
sers u ON r.user_id = u.id WHERE u.name = 'Jack') sub ON m.genre = sub.genre WHERE sub.rating =  (SELECT MAX(sub.rating) FROM(SELECT r.rating, m.
genre FROM reviews r JOIN movies m ON r.movie_id = m.id JOIN users u ON r.user_id = u.id WHERE u.name = 'Jack') sub);
+-------------+--------+
| name        | genre  |
+-------------+--------+
| Rush Hour   | action |
| Earth Battle | action |
| Ninja Turtle | action |
+-------------+--------+
3 rows in set (0.04 sec)
```

Explanation:

First, find Jack’s highest movie rating and return that movie’s genre. In my test case, Jack’s favorite movies are Rush Hour and Earth Battle, both their rating are 8.0, the genres are action. So next, based on Jack’s favorite movie’s genre(action), I list all the movie names (Rush Hour, Earth Battle and Ninja Turtle) and genres from this genre(action).

3- List the name(s) of the user(s) born in January who rated at least 6 for the movie 'Titanic'.

Query:

```
SELECT DISTINCT u.name
FROM users u JOIN reviews r ON u.id = r.user_id JOIN movies m ON r.movie_id = m.id
WHERE MONTH(date_of_birth) = 1
        AND m.name = ‘Titanic’
        AND rating >= 6;
```

```
MySQL [HW2]> SELECT DISTINCT u.name FROM users u JOIN reviews r ON u.id = r.user
_id JOIN movies m ON r.movie_id = m.id WHERE MONTH(date_of_birth)= 1  AND m.name
 = 'Titanic'  AND rating >= 6;
+---------------+
| name          |
+---------------+
| Chris Jackson |
+---------------+
1 row in set (0.03 sec)
```

Explanation:

This question is straightforward. I join three tables: users, movies and reviews. Then set three WHERE clause: MONTH(date_of_birth) = 1, m.name = 'Titanic' and rating >= 6. In my test case, it only show Chris Jackson.

4- List the movie name(s) not reviewed by Chris Jackson.

Query:

```
SELECT name
FROM movies m
WHERE m.id NOT IN
        (SELECT m.id FROM movies m JOIN reviews r ON m.id = r.movie_id JOIN users u ON
        r.user_id = u.id WHERE u.name = 'Chris Jackson')
```

```
MySQL [HW2]> SELECT name FROM movies m WHERE m.id NOT IN (SELECT m.id FROM movies m JOIN reviews r ON m.id = r.movie_id JOIN users u ON r.user_id=
 u.id WHERE u.name = 'Chris Jackson');
+--------------+
| name         |
+--------------+
| POKEMON      |
| Rush Hour    |
| Earth Battle |
| Ninja Turtle |
+--------------+
4 rows in set (0.04 sec)
```

Explanation:

This question I use NOT IN operator. First, find the movie which Chris Jackson reviews. Then remove that movie, using NOT IN operator. In my case, Chris Jackson reviews Titanic, so the output will not show Titanic.

5- For all pairs of reviewers such that both reviewers gave a rating to the same movie, return the names of both reviewers. Eliminate duplicates, don't pair reviewers with themselves, and include each pair only once. For each pair, return the names in the pair in alphabetical order.

Query:

```
SELECT DISTINCT u1.name, u2.name
FROM reviews r1, reviews r2, users u1, users u2
WHERE r1.movie_id = r2.movie_id
AND u1.id = r1.user_id
AND u2.id = r2.user_id
AND u1.name < u2.name
ORDER BY u1.name, u2.name;
```

```
MySQL [HW2]> SELECT DISTINCT u1.name, u2.name
    -> FROM reviews r1, reviews r2, users u1, users u2
    -> WHERE r1.movie_id = r2.movie_id
    -> AND u1.id = r1.user_id
    -> AND u2.id = r2.user_id
    -> AND u1.name < u2.name
    -> ORDER BY u1.name, u2.name;
+---------------+---------+
| name          | name    |
+---------------+---------+
| Apple         | Johnny  |
| Chris Jackson | Jack    |
| Chris Jackson | Johnny  |
| Jack          | Johnny  |
| Jack          | Mac     |
| Johnny        | Mac     |
+---------------+---------+
6 rows in set (0.03 sec)
```

Explanation:

This question we need to find the pairs review to the same movie without duplicates, so first I will find two distinct users from reviews and users tables. Then in my WHERE clause, I set r1.movie_id = r2.movie_id, and connect users from users and reviews tables by id. Finally, since we need to return the pair in alphabetical order, so I set u1.name < u2.name.

6- List the name(s) of all action movie(s) that were released before 2007 and have review rating less than average rating of all movies, sorted in ascending order.

a. Note that you should compute the average of movie average ratings, not the average of all ratings. E.g. movie A got reviews 10, 10, and 10, and movie B got just one 6, the result should be ((10 + 10 + 10) / 3 + 6) / 2 = 8, instead of (10 + 10 + 10 + 6) / 4 = 9.

Query:

```
SELECT name
FROM movies m JOIN reviews r ON m.id = r.movie_id
WHERE genre = 'action'
        AND YEAR(release_date) < 2007
        AND rating < (SELECT AVG(sub.each_average) FROM (SELECT AVG(rating) AS
        each_average FROM reviews GROUP BY movie_id) sub)
GROUP BY name
ORDER BY name ASC;
```

```
MySQL [HW2]> SELECT name FROM movies m JOIN reviews r ON m.id = r.movie_id
 WHERE genre = 'action'  AND YEAR(release_date) < 2007  AND rating < (SELE
CT AVG(sub.each_average) FROM (SELECT AVG(rating) AS each_average FROM rev
iews GROUP BY movie_id) sub) GROUP BY name ORDER BY name ASC;
+--------------+
| name         |
+--------------+
| Earth Battle |
| Ninja Turtle |
| Rush Hour    |
+--------------+
3 rows in set (0.04 sec)
```

Explanation:

First, we need to calculate the average of average ratings. To do that, I set a subquery: SELECT AVG(sub.each_average) FROM (SELECT AVG(rating) AS each_average FROM reviews GROUP BY movie_id) sub). The rest of parts are easier. I set the WHERE clause: genre = ‘action’ AND YEAR(release_date) < 2007. Finally, GROUP/ORDER BY movie name. In my test case, the result is Earth Battle, Ninja Turtle and Rush Hour.