DB Assignment 2: SQL on GCP

Siddhesh Karekar, karekar@usc.edu

PostgreSQL was used for this assignment.

Table Creation Queries

These queries were used to create the structure of the table, as given in the description.

```
CREATE TABLE users (
      id INTEGER PRIMARY KEY NOT NULL,
      name VARCHAR(255) NOT NULL,
      date_of_birth DATE NOT NULL
);

CREATE TABLE movies (
      id INTEGER PRIMARY KEY NOT NULL,
      name VARCHAR(255) NOT NULL,
      genre VARCHAR(255) NOT NULL,
      release_date DATE NOT NULL
);

CREATE TABLE reviews (
      user_id INTEGER NOT NULL,
      movie_id INTEGER NOT NULL,
      rating NUMERIC NOT NULL,
      comment VARCHAR(5000),
      FOREIGN KEY (user_id) REFERENCES users (id),
      FOREIGN KEY (movie_id) REFERENCES movies (id),
      PRIMARY KEY (user_id, movie_id)
);

CREATE TABLE actors (
      id INTEGER PRIMARY KEY NOT NULL,
      name VARCHAR(255) NOT NULL,
      gender VARCHAR(10) NOT NULL,
      date_of_birth DATE NOT NULL
);

CREATE TABLE lead (
      actor_id INTEGER NOT NULL,
      movie_id INTEGER NOT NULL,
      FOREIGN KEY (actor_id) REFERENCES actors (id),
      FOREIGN KEY (movie_id) REFERENCES movies (id),
      PRIMARY KEY (actor_id, movie_id)
);
```

Data Insertion Queries

These queries inserted some sample data into these tables for use by the homework queries.

```sql
-- users
INSERT INTO users VALUES (1, 'Siddhesh Karekar', '1995-07-31');
INSERT INTO users VALUES (20, 'John Doe', 'April 1, 1992');
INSERT INTO users VALUES (21, 'Jane Doe', 'April 2, 1992');
INSERT INTO users VALUES (22, 'See Kwell', 'April 30, 1990');

-- movies
INSERT INTO movies VALUES (1, 'Lawn With The Wind', 'comedy', '2003-01-05');
INSERT INTO movies VALUES (2, '2+2=U', 'comedy', '2004-11-05');
INSERT INTO movies VALUES (3, 'Notebook', 'drama', '2006-03-12');
INSERT INTO movies VALUES (4, 'First Time at the Zoo', 'comedy', '2011-10-10');
INSERT INTO movies VALUES (5, 'The light is coming', 'mystery', '2015-12-15');
INSERT INTO movies VALUES (6, 'Really old movie', 'documentary', '2001-09-25');

-- reviews
INSERT INTO reviews VALUES (20, 1, 10.0, 'Literally the best movie ever!');
INSERT INTO reviews VALUES (20, 2, 9.2, 'Would watch again.');
INSERT INTO reviews VALUES (20, 3, 5.0, 'Too sad!');
INSERT INTO reviews VALUES (20, 4, 6.32);
INSERT INTO reviews VALUES (20, 5, 10.0, 'Enlightenment is key!');

INSERT INTO reviews VALUES (21, 2, 2.7, 'Terrible!');
INSERT INTO reviews VALUES (21, 3, 8.1, 'I love this romance movie.');

INSERT INTO reviews VALUES (22, 2, 7.2, 'This movie was nothing special.');
INSERT INTO reviews VALUES (22, 3, 7.1, 'This movie was okay.');
INSERT INTO reviews VALUES (22, 6, 9.1, 'Epic movie.');

-- actors
INSERT INTO actors VALUES (1, 'Mark Clarkson', 'male', 'November 8, 1995');
INSERT INTO actors VALUES (2, 'Jack Drake', 'male', 'April 22, 1990');
INSERT INTO actors VALUES (3, 'Lava Shwava', 'female', 'July 30, 1994');
INSERT INTO actors VALUES (4, 'Margaret Tarmaset', 'female', 'December 31, 1989');
INSERT INTO actors VALUES (5, 'Julie Unroolie', 'female', 'August 3, 1961');
INSERT INTO actors VALUES (6, 'Relly Patelly', 'male', 'March 15, 1955');

-- lead
INSERT INTO lead VALUES (1, 1);
INSERT INTO lead VALUES (2, 1);

INSERT INTO lead VALUES (1, 2);
INSERT INTO lead VALUES (4, 2);
INSERT INTO lead VALUES (5, 2);

INSERT INTO lead VALUES (3, 3);
INSERT INTO lead VALUES (4, 3);

INSERT INTO lead VALUES (1, 4);
INSERT INTO lead VALUES (6, 4);
INSERT INTO lead VALUES (3, 4);
INSERT INTO lead VALUES (4, 4);
```

Homework Queries

(each query is listed on a separate page for clarity.)

Query 1

List the name(s) of the user(s) born in April who rated at most 8 for the movie 'Notebook'. Output their names sorted in descending order.

Answer

```sql
SELECT DISTINCT u.name
FROM users u, reviews r, movies m
WHERE u.id = r.user_id
AND EXTRACT(MONTH FROM u.date_of_birth) = 04
AND r.movie_id = m.id
AND r.rating <= 8.0
AND m.name = 'Notebook'
ORDER BY name DESC;
```

Explanation

1. We join 3 tables: `users`, `reviews` and `movies` and specify the conditions given:
   - `MONTH` in `u.date_of_birth` is `04` (April)
   - `name` of the movie is `'Notebook'`, and
   - `rating` is at most (or <=) `8`.

Query 2

Find user 'John Doe''s 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) sorted them based on the movie genre then movie name in the ascending order.

Answer

```
SELECT name, genre
FROM movies
WHERE genre IN (
      SELECT m.genre
      FROM users u, reviews r, movies m
      WHERE u.id = r.user_id
      AND u.name = 'John Doe'
      AND r.movie_id = m.id
      GROUP BY m.genre
      HAVING AVG(r.rating) = (
            SELECT MAX(avg)
            FROM (
                  SELECT m.genre, AVG(r.rating)
                  FROM users u, reviews r, movies m
                  WHERE u.id = r.user_id
                  AND u.name = 'John Doe'
                  AND r.movie_id = m.id
                  GROUP BY m.genre) john_genre_rating))
ORDER BY genre ASC, name ASC;
```

Explanation

1. First, let us
   - join `users`, `reviews` and `movies`, then
   - filter `user name` to be `'John Doe'`
   - group by `movie genre`
   - and project the `genre` and `AVG(rating)`

   as follows. This lets us get an idea of how much `John` likes each `genre` on an average.

```
SELECT m.genre, AVG(r.rating)
FROM users u, reviews r, movies m
WHERE u.id = r.user_id
AND u.name = 'John Doe'
AND r.movie_id = m.id
GROUP BY m.genre;
```

2. Now, let us select the highest `rating` from the above table by wrapping the above query in `MAX`.

```
SELECT MAX(avg)
FROM (
      SELECT m.genre, AVG(r.rating)
      FROM users u, reviews r, movies m
      WHERE u.id = r.user_id
      AND u.name = 'John Doe'
      AND r.movie_id = m.id
      GROUP BY m.genre) john_genre_rating;
```

3. Now, we must repeat step 1 and use `HAVING` to retain only those `genres` which are equal to the `MAX(avg) rating`. We can remove `rating` from the projection and only keep `genre`. This leaves us a list of only `John's` favorite `genres`.

```
SELECT m.genre
FROM users u, reviews r, movies m
WHERE u.id = r.user_id
AND u.name = 'John Doe'
AND r.movie_id = m.id
GROUP BY m.genre
HAVING AVG(r.rating) = (
      SELECT MAX(avg)
      FROM (
            SELECT m.genre, AVG(r.rating)
            FROM users u, reviews r, movies m
            WHERE u.id = r.user_id
            AND u.name = 'John Doe'
            AND r.movie_id = m.id
            GROUP BY m.genre) john_genre_rating);
```

4. Finally, project `name` and `genres` from the `movies` table while `genre` lies in the output of the above query, and ORDER BY `genre` then `name` as given in the question to yield the final query.

```
SELECT name, genre
FROM movies
WHERE genre IN (
      SELECT m.genre
      FROM users u, reviews r, movies m
      WHERE u.id = r.user_id
      AND u.name = 'John Doe'
      AND r.movie_id = m.id
      GROUP BY m.genre
      HAVING AVG(r.rating) = (
            SELECT MAX(avg)
            FROM (
                  SELECT m.genre, AVG(r.rating)
                  FROM users u, reviews r, movies m
                  WHERE u.id = r.user_id
                  AND u.name = 'John Doe'
                  AND r.movie_id = m.id
                  GROUP BY m.genre) john_genre_rating))
ORDER BY genre ASC, name ASC;
```

Query 3

List the movie ID(s) with most male lead. Sort the IDs in descending order.

Answer

```
SELECT l.movie_id, COUNT(l.actor_id) no_of_leads
FROM movies m, lead l, actors a
WHERE m.id = l.movie_id
AND l.actor_id = a.id
AND a.gender = 'male'
GROUP BY l.movie_id
HAVING COUNT(l.actor_id) = (SELECT MAX(no_of_leads)
      FROM (SELECT l.movie_id, COUNT(l.actor_id) no_of_leads
            FROM movies m, lead l, actors a
            WHERE m.id = l.movie_id
            AND l.actor_id = a.id
            AND a.gender = 'male'
            GROUP BY l.movie_id) movie_lead_count);
```

Explanation

1. We can first list the number of male leads per movie by joining the tables `movies`, `lead` and `actors` as follows:

```
SELECT l.movie_id, COUNT(l.actor_id) no_of_leads
FROM movies m, lead l, actors a
WHERE m.id = l.movie_id
AND l.actor_id = a.id
AND a.gender = 'male'
GROUP BY l.movie_id;
```

2. Now, we find what is the maximum possible number of male leads in the given data by using the `MAX` function:

```
SELECT MAX(no_of_leads)
FROM (SELECT l.movie_id, COUNT(l.actor_id) no_of_leads
      FROM movies m, lead l, actors a
      WHERE m.id = l.movie_id
      AND l.actor_id = a.id
      AND a.gender = 'male'
      GROUP BY l.movie_id) movie_lead_count;
```

3. Finally, we join the three original tables again, but this time add a `HAVING` clause and make sure the no_of_`leads` is equal to the maximum value obtained in Step 2, to generate our final answer:

```
SELECT l.movie_id, COUNT(l.actor_id) no_of_leads
FROM movies m, lead l, actors a
WHERE m.id = l.movie_id
AND l.actor_id = a.id
AND a.gender = 'male'
GROUP BY l.movie_id
HAVING COUNT(l.actor_id) = (SELECT MAX(no_of_leads)
      FROM (SELECT l.movie_id, COUNT(l.actor_id) no_of_leads
            FROM movies m, lead l, actors a
            WHERE m.id = l.movie_id
            AND l.actor_id = a.id
            AND a.gender = 'male'
            GROUP BY l.movie_id) movie_lead_count);
```

Query 4

List the name(s) of all comedy movie(s) that were released before 2006 and have review rating better than average rating of all movies, sorted in ascending order.

Note: 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.

Answer

```
SELECT DISTINCT m.name
FROM movies m, reviews r
WHERE m.id = r.movie_id
AND m.release_date < 'January 1, 2006'
AND m.genre = 'comedy'
AND r.rating > (
      SELECT AVG(movie_avg_rating) avg_movie_rating FROM (SELECT AVG(r.rating)
movie_avg_rating
            FROM movies m, reviews r
            WHERE m.id = r.movie_id
            GROUP BY r.movie_id) movie_avg_ratings)
ORDER BY m.name ASC;
```

Explanation

1. First, let us find the average rating for each movie by using the `AVG()` function and the `movie_id` in the `GROUP BY` clause.

```
SELECT r.movie_id, AVG(r.rating) movie_avg_rating
FROM movies m, reviews r
WHERE m.id = r.movie_id
GROUP BY r.movie_id;
```

2. Now, we must find the average of these `ratings` to find the average `rating` of all `movies`.

```
SELECT AVG(movie_avg_rating) avg_movie_rating FROM (SELECT r.movie_id, AVG(r.rating)
movie_avg_rating
      FROM movies m, reviews r
      WHERE m.id = r.movie_id
      GROUP BY r.movie_id) movie_avg_ratings;
```

3. Finally, we join the `movies` and `reviews` tables, apply the filters for `release_date` to be before 2006 and `genre` to be '`comedy`', also we only want those `movies` with `reviews` that have `ratings` better than the average rating found in the second query; finally `ORDER BY` in `ASC` `name` order.

   Due to multiple `reviews` being present, `name` can be returned multiple times hence we use `DISTINCT`.

```
SELECT DISTINCT m.name
FROM movies m, reviews r
WHERE m.id = r.movie_id
AND m.release_date < 'January 1, 2006'
AND m.genre = 'comedy'
AND r.rating > (
      SELECT AVG(movie_avg_rating) avg_movie_rating FROM (SELECT AVG(r.rating)
movie_avg_rating
            FROM movies m, reviews r
```

```
            WHERE m.id = r.movie_id
            GROUP BY r.movie_id) movie_avg_ratings)
ORDER BY m.name ASC;
```

Query 5

List the movie ID(s) and average review(s) where the average review is higher than 9 and one of their leading actors is the actor 'Mark Clarkson'. Sort the output by average reviews and then movie IDs.

Answer

```
SELECT r.movie_id, AVG(r.rating) movie_avg_rating
FROM movies m, reviews r
WHERE m.id = r.movie_id
AND m.id IN (
      SELECT DISTINCT l.movie_id
      FROM lead l, actors a
      WHERE l.actor_id = a.id
      AND a.name = 'Mark Clarkson')
GROUP BY r.movie_id
ORDER BY movie_avg_rating, r.movie_id;
```

Explanation

1. First, let us find all `movie_ids` where `'Mark Clarkson'` is an actor. We can do this by:
   - joining `lead` and `actors` on `actor_id`,
   - retaining those rows where actor `name` is `'Mark Clarkson'`, and
   - projecting only `DISTINCT movie_ids`.

```
SELECT DISTINCT l.movie_id
FROM lead l, actors a
WHERE l.actor_id = a.id
AND a.name = 'Mark Clarkson';
```

2. Now, similar to step 1 of Query 4, we can write a query to list `movie_id` and their average `rating` sorted by `reviews` and `ids`, and then use a `HAVING` clause with `AVG(rating) > 9` as follows:

```
SELECT r.movie_id, AVG(r.rating) movie_avg_rating
FROM movies m, reviews r
WHERE m.id = r.movie_id
GROUP BY r.movie_id
HAVING AVG(r.rating) > 9
ORDER BY movie_avg_rating, r.movie_id;
```

3. Finally, we combine Step 1 and Step 2, by limiting the rows in Step 2 to contain `ids` only from the first (i.e., only those `movies` that have `'Mark Clarkson'` in the lead.) This results in the final query being

```
SELECT r.movie_id, AVG(r.rating) movie_avg_rating
FROM movies m, reviews r
WHERE m.id = r.movie_id
AND m.id IN (
      SELECT DISTINCT l.movie_id
      FROM lead l, actors a
      WHERE l.actor_id = a.id
      AND a.name = 'Mark Clarkson')
GROUP BY r.movie_id
HAVING AVG(r.rating) > 9
ORDER BY movie_avg_rating, r.movie_id;
```

Query 6

Find the actors who played the lead together the most. Display these their names and the number of times they played the lead together.

Note: The resulting table must show both actors info in the same row (Actor1 and Actor2). This might result in duplicate data where two rows might have the same actors but in different columns.

Answer

```
SELECT a1.name, a2.name
FROM actors a1, actors a2, (
      SELECT l1.actor_id actor1_id, l2.actor_id actor2_id, COUNT(l1.movie_id)
costar_count
      FROM lead l1, lead l2
      WHERE l1.movie_id = l2.movie_id
      AND l1.actor_id <> l2.actor_id
      GROUP BY l1.actor_id, l2.actor_id
      HAVING COUNT(l1.movie_id) = (
            SELECT MAX(costar_count)
            FROM (
                  SELECT l1.actor_id actor1_id, l2.actor_id actor2_id,
COUNT(l1.movie_id) costar_count
                  FROM lead l1, lead l2
                  WHERE l1.movie_id = l2.movie_id
                  AND l1.actor_id <> l2.actor_id
                  GROUP BY l1.actor_id, l2.actor_id) leads_costar)) leads_costar_max
WHERE a1.id = leads_costar_max.actor1_id
AND a2.id = leads_costar_max.actor2_id;
```

Explanation

1. Let us start off simple. Say we want to list all pairs of lead actors for each `movie_id`, we can join the `lead` table with itself on `movie_id` and making sure `actor_id` is unequal.

```
SELECT l1.movie_id, l1.actor_id, l2.actor_id
FROM lead l1, lead l2
WHERE l1.movie_id = l2.movie_id
AND l1.actor_id <> l2.actor_id;
```

2. We can further modify this query to `COUNT movie_ids` by having the combination `(actor1_id, actor2_id)` in the `GROUP BY` clause. This lets us see how many `movies` does a pair of `actors` star in.

```
SELECT l1.actor_id actor1_id, l2.actor_id actor2_id, COUNT(l1.movie_id) costar_count
FROM lead l1, lead l2
WHERE l1.movie_id = l2.movie_id
AND l1.actor_id <> l2.actor_id
GROUP BY l1.actor_id, l2.actor_id;
```

3. Now we can use the above query and extract the maximum number of times a lead pair co-stars.

```
SELECT MAX(costar_count)
FROM (
      SELECT l1.actor_id actor1_id, l2.actor_id actor2_id, COUNT(l1.movie_id)
costar_count
      FROM lead l1, lead l2
      WHERE l1.movie_id = l2.movie_id
```

```
      AND l1.actor_id <> l2.actor_id
      GROUP BY l1.actor_id, l2.actor_id) leads_costar;
```

4. We can modify Step 2 and add a `HAVING` clause to retain only those lead pairs with a `costar_count` equal to the `MAX(costar_count)` retrieved in Step 3.

```
SELECT l1.actor_id actor1_id, l2.actor_id actor2_id, COUNT(l1.movie_id) costar_count
FROM lead l1, lead l2
WHERE l1.movie_id = l2.movie_id
AND l1.actor_id <> l2.actor_id
GROUP BY l1.actor_id, l2.actor_id
HAVING COUNT(l1.movie_id) = (
      SELECT MAX(costar_count)
      FROM (
            SELECT l1.actor_id actor1_id, l2.actor_id actor2_id, COUNT(l1.movie_id)
costar_count
            FROM lead l1, lead l2
            WHERE l1.movie_id = l2.movie_id
            AND l1.actor_id <> l2.actor_id
            GROUP BY l1.actor_id, l2.actor_id) leads_costar);
```

5. Finally, join the above query with the `actors` table to retrieve their `names` instead of `ids` and get rid of the `costar_count` as it is not asked for in the question.

   To do this, we need to have two copies of `actors` in the `FROM` clause (aliased here as `a1` and `a2`), since each `actor_id` needs its own copy of `actors` to get the corresponding `name`.

   This results in the final query being:

```
SELECT a1.name, a2.name
FROM actors a1, actors a2, (
      SELECT l1.actor_id actor1_id, l2.actor_id actor2_id, COUNT(l1.movie_id)
costar_count
      FROM lead l1, lead l2
      WHERE l1.movie_id = l2.movie_id
      AND l1.actor_id <> l2.actor_id
      GROUP BY l1.actor_id, l2.actor_id
      HAVING COUNT(l1.movie_id) = (
            SELECT MAX(costar_count)
            FROM (
                  SELECT l1.actor_id actor1_id, l2.actor_id actor2_id,
COUNT(l1.movie_id) costar_count
                  FROM lead l1, lead l2
                  WHERE l1.movie_id = l2.movie_id
                  AND l1.actor_id <> l2.actor_id
                  GROUP BY l1.actor_id, l2.actor_id) leads_costar)) leads_costar_max
WHERE a1.id = leads_costar_max.actor1_id
AND a2.id = leads_costar_max.actor2_id;
```

This concludes the assignment.