Andrew Risse HW2

REFERENCES:

Inserting multiple records https://www.electrictoolbox.com/mysql-insert-multiple-records/

Sorting increasing or decreasing https://stackoverflow.com/questions/3968135/mysql-alphabetical-order

The database used was called hw2, it follows the same structure and naming convention as described in the instructions. The database is MySQL Second Generation on the Google Cloud platform.

Commands to create database:

CREATE TABLE users (id int not null AUTO_INCREMENT, name VARCHAR(30), date_of_birth DATE, PRIMARY KEY(id));

CREATE TABLE movies (id int not null AUTO_INCREMENT, name VARCHAR(50), genre VARCHAR(20), release_date DATE, PRIMARY KEY(id));

CREATE TABLE reviews (user_id int, movie_id int, rating int, comment VARCHAR(5000), PRIMARY KEY(user_id, movie_id), FOREIGN KEY (user_id) REFERENCES users(id), FOREIGN KEY (movie_id) REFERENCES movies(id));

CREATE TABLE actors (id int not null AUTO_INCREMENT, name VARCHAR(30), gender CHAR, date_of_birth DATE, PRIMARY KEY(id));

CREATE TABLE lead (actor_id int, movie_id int, PRIMARY KEY(actor_id, movie_id), FOREIGN KEY (actor_id) REFERENCES actors(id), FOREIGN KEY (movie_id) REFERENCES movies(id));

Creation of data:

INSERT INTO users

(name, date_of_birth)

VALUES

('Andrew Risse', '1990-09-22'),

('Andrew Klavan', '1954-07-13'),

('Brandi Risse', '1934-04-24'),

('Mathew Curry', '1984-04-12'),

('Haley Curry', '1987-04-02'),

```
        ('John Doe', '1995-04-15'),
        ('Bob Axelrod', '1975-01-15'),
        ('Sandy Squirrel', '2005-04-04'),
        ('Christopher Smith', '1960-03-26'),
        ('Donald Trump', '1946-06-14'),
        ('James Mattis', '1950-09-08'),
        ('Nikki Haley', '1972-01-20'),
        ('Ben Shapiro', '1984-01-15'),
        ('Michael Knowles', '1990-03-18');

INSERT INTO movies
        (name, genre, release_date)
VALUES
        ('The Godfather', 'drama', '1972-03-24'),
        ('The Shawshank Redemption', 'drama', '1994-09-23'),
        ('Schindlers List', 'biography', '1993-09-15'),
        ('Casablanca', 'war', '1942-11-26'),
        ('Gone With The Wind', 'romance', '1940-01-17'),
        ('The Lord of the Rings The Return of the King', 'action', '2003-12-17'),
        ('Titanic', 'romance', '1997-12-19'),
        ('Gladiator', 'action', '2000-05-05'),
        ('Lone Survivor', 'war', '2013-12-25'),
        ('American Sniper', 'war', '2014-12-25'),
        ('Notebook', 'romance', '2004-06-25'),
        ('Dumb and Dumber', 'comedy', '1994-12-06'),
        ('Ace Ventura Pet Detective', 'comedy', '1994-02-04'),
        ('Fake Movie 1', 'comedy', '2005-05-18'),
        ('Fake Movie 2 The Return', 'comedy', '2007-07-04'),
```

```
        ('Fake Movie 3 When Will It End', 'comedy', '2009-09-21');


INSERT INTO actors
        (name, gender, date_of_birth)
VALUES
        ('Mark Clarkson', 'M', '1990-09-10'),
        ('Steve Matsumoto', 'M', '1990-11-17'),
        ('Daniel Dison', 'M', '1989-01-10'),
        ('Dean Risse', 'M', '1996-05-18'),
        ('Abbi Kesterson', 'F', '1993-10-02'),
        ('Julie Epsteen', 'F', '1967-02-13'),
        ('Kelly Clarkson', 'F', '1982-04-24'),
        ('Russel Crowe', 'M', '1964-04-7'),
        ('Robert Smart', 'M', '1920-06-22'),
        ('Edgar Pope', 'M', '1915-02-02'),
        ('Marie Hosel', 'F', '1910-11-16');


INSERT INTO lead
        (actor_id, movie_id)
VALUES
        (11,1),
        (9,1),
        (11,2),
        (9,2),
        (7,3),
        (11,3),
        (11,4),
        (9,4),
```

```
    (10,5),
    (1,6),
    (2,6),
    (3,7),
    (8, 8),
    (3,8),
    (8,9),
    (3,9),
    (8,10),
    (3,10),
    (4,11),
    (5, 11),
    (6, 12),
    (9,13),
    (10, 14),
    (1, 14),
    (1, 15),
    (1,16),
    (3, 14),
    (4, 14),
    (9, 14),
    (3, 15),
    (4,15),
    (1,10),
    (1,9);

INSERT INTO reviews
    (user_id, movie_id, rating, comment)
```

VALUES

(1,1,10, 'fantastic!'),

(2,10,9, 'loved it!'),

(3,9,10, 'perfection'),

(4,9,10, 'yes'),

(1,2,7, 'pretty good'),

(1,3, 4, 'alright'),

(1,4, 6, 'liked it'),

(2,5, 4, 'I guess'),

(2,9, 7, 'go see it'),

(2,4, 5, 'middle of the road'),

(7,10, 10, 'wow'),

(6,3,7, 'sad but good'),

(6,4,7, 'classic!'),

(6,9,10, 'incredible!'),

(6,10,10, 'amazing!'),

(6,5,1, 'boring!'),

(6,7,2, 'overated'),

(6,11,1, 'wife won’t stop making me watch this'),

(6,12,5, 'alright'),

(6,13,6, 'decent'),

(6,14,4, 'OK'),

(2,12,9, 'hilarious!'),

(7,12,10, 'very funny!'),

(3,13,9, 'laughed a lot!'),

(11,13,9, 'lots of laughs'),

(4,12,8, 'loved it'),

(5,13,9, 'stomach hurts from laughing'),

(7,14,1, 'dumb'),

(8,14,2, 'stupid'),

(9,14,3, 'lame!'),

(10,15,3, 'waste of time'),

(12,15,1, 'pathetic'),

(13,15,1, 'horrendous'),

(14,16,2, 'idiotic'),

(4,16,2, 'useless'),

(2,11,8, 'wonderful'),

(4,11,9, 'touching'),

(8,11,7, 'better than most'),

(9,11,2, 'made me want to puke'),

(5,11,1, 'gross'),

(5,16,2, 'junk');

Question 1:

```
SELECT name
FROM users JOIN
    (SELECT user_id
    FROM reviews
    WHERE reviews.movie_id = (SELECT id FROM movies WHERE name =
    'Notebook') AND reviews.rating <= 8) AS subTable ON user_id=users.id
WHERE MONTH(users.date_of_birth) = 4
ORDER BY name DESC;
```

This query uses subqueries to find the movie ID for “Notebook”, then find the reviews for that movie that are less than or equal to 8 and finds the corresponding user IDs. It then joins

users with the resulting table on user IDs. Finally, names are selected from the resulting table whose birthday is in April and the results are listed in descending order by name.

Question 2:

```
SELECT name, genre
FROM movies NATURAL JOIN (SELECT  genre, AVG(rating) AS genreAverage
        FROM movies JOIN reviews ON reviews.movie_id = movies.id
        WHERE reviews.user_id = (SELECT id FROM users WHERE users.name =
        'John Doe')
        GROUP BY genre) AS avgRatingTable
ORDER BY genreAverage DESC, name;
```

This query first selects all of John Doe's reviews and groups them by genre, then it finds the overall genre average ratings and joins that to the movie table so that all movies now include a genreAverage column. Name and genre are then selected from this table and ordered by highest genre rating first, then movie name in ascending order.

Assumption 1: The problem wants highest rated genres first, not genres sorted alphabetically. Movies within the genre list are to be listed in ascending order.

Assumption 2: If two genres have the same average rating, movies from both genres will be in same group sorted in ascending order.

Question 3:

```
SELECT movie_id, COUNT(*)
FROM lead JOIN (SELECT id
        FROM actors
        WHERE actors.gender = 'M') AS males ON id = lead.actor_id
GROUP BY movie_id
ORDER BY movie_ID DESC;
```

This query finds all the male actors and JOINs that result to the table “lead” creating a table that has only movies with male leads. It then groups by movie_id and counts all the repeating movie_ids then orders them in descending order.

Question 4:

```
SELECT DISTINCT name
FROM reviews JOIN
        (SELECT name, id
        FROM movies
          WHERE movies.genre = 'comedy ' AND YEAR(movies.release_date)
        <2006) AS comedyList ON id = reviews.movie_id
WHERE rating >
        (SELECT AVG(individualAverage) AS totalAverage FROM
        (SELECT movie_id, AVG(rating) AS individualAverage
        FROM reviews
        GROUP BY movie_id) AS table1)
ORDER BY name;
```

This query finds the average rating for each individual movie, then computes the average of all movie average ratings. It then finds all comedies released before 2006 and JOINs that with the “reviews” table. Distinct names are then selected from the resulting table where the movie’s rating is greater than the average. The results are then ordered in ascending order by name.

Question 5:

```
SELECT movie_id, individualAverage
FROM lead NATURAL JOIN(SELECT movie_id, AVG(rating) AS individualAverage
        FROM reviews
        GROUP BY movie_id) AS movieAvgTable
```

```
WHERE movieAvgTable.individualAverage > 9 AND lead.actor_id =
                (SELECT id FROM actors
                WHERE actors.name = 'Mark Clarkson')
ORDER BY individualAverage DESC;
```

This query first calculates the average review rating for each movie, it then joins this table with the table “lead”. Next it finds those movies with ratings greater than 9 that have Mark Clarkson as an actor and lists the movie id as well as average. Results are then sorted by individual average.

Assumption 1: The assignment says to sort by average reviews, then movie IDs. Each movie only has 1 average review rating, so they cannot be sorted by both of these. They are either ordered by rating, or ID since each movie will only be listed once and I chose to sort by average rating descending.

Question 6:

```
SELECT  Actor1, name AS Actor2, count
FROM actors JOIN (SELECT name AS Actor1, ActorID1, ActorID2, count  FROM  actors
        JOIN  (SELECT ACTORID1.actor_id AS ActorID1, ACTORID2.actor_id
        AS ActorID2, COUNT(*) AS count FROM lead ACTORID1 JOIN lead
        ACTORID2 ON
        ACTORID1.movie_id = ACTORID2.movie_id   WHERE
        ACTORID1.movie_id =
        ACTORID2.movie_id AND ACTORID1.actor_id != ACTORID2.actor_id
        GROUP BY
        ACTORID1.actor_id, ACTORID2.actor_id) AS subTable ON ActorID1 =
        actors.id) AS
        bigTable ON ActorID2 = actors.id
ORDER BY count DESC;
```

This query recursively joins “lead” tables so that lead actors are paired with other lead actors they worked with on the same movie. It then groups those by actor ids and counts them up. This actor id pair table is then joined to the actor table on the first actor id column, then joined to the

actor table again on the second actor id column. The names of actors are then selected along with the count of how many times they worked together. The table is then ordered by count descending to find the actors who played lead together the most. Note- I attempted to clean up the code to be easier to read multiple times, but I think some weird Word formatting problem was causing the code to break. If I leave it as it is above, it works. I added some color coding to help decipher the nested SELECT queries.