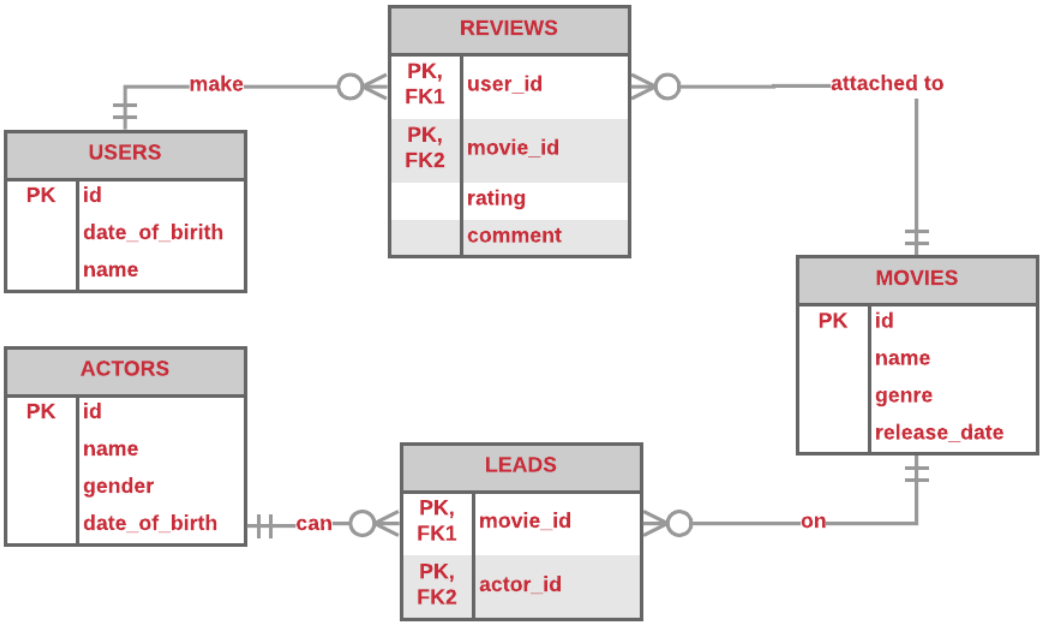
Minh Tran – CSCI 585 – Homework 1

Final Report

I. Table and model



II. Model justification

The USERS, MOVIES AND ACTORS are specified as instructed in the homework prompt. The LEADS and REVIEWS are implemented to accommodate the many to many relationships between ACTORS-MOVIES and USERS-MOVIES respectively. The following script is used to create tables and insert data:

```
# create a new database for movie review application

DROP SCHEMA IF EXISTS movRevApp;

CREATE DATABASE IF NOT EXISTS movRevApp;

USE movRevApp;
```

```
CREATE TABLE IF NOT EXISTS actors (
   id       SMALLINT UNSIGNED UNIQUE NOT NULL,
   name           VARCHAR(45) NOT NULL,
   gender         CHAR(1)     NOT NULL CHECK(gender IN ("M", "F", "U", "B")),
   date_of_birth   DATE       NOT NULL,
   PRIMARY KEY (id)
)   ENGINE=INNODB;


INSERT INTO actors
          (id, name, gender, date_of_birth)
VALUES
(1,       'PENELOPE GUINESS'   ,     'B',19530717),
(2,       'MARK CLARKSON'      ,     'M',19700628),
(3,       'ED CHASE'           ,  'U',19610722),
(4,       'JENNIFER DAVIS'     ,        'M',19480520),
(5,       'JOHNNY LOLLOBRIGIDA',          'U',19580124),
(6,       'BETTE NICHOLSON'    ,     'F',19310615),
(7,       'GRACE MOSTEL'       ,        'M',19940505),
(8,       'MATTHEW JOHANSSON'  ,          'B',19620722),
(9,       'JOE SWANK'          , 'M',19830510),
(10,    'CHRISTIAN GABLE'    ,       'B',19800811);
```

```
CREATE TABLE if not exists users (
   id           SMALLINT    UNSIGNED UNIQUE NOT NULL,
   name         VARCHAR(45) NOT NULL,
   date_of_birth DATE       NOT NULL,
   PRIMARY KEY(id)
)  ENGINE=INNODB;

INSERT INTO users
          (id, name, date_of_birth)
VALUES
(1,       'JOHN DOE'       ,      19530218),
(2,       'PATRICIA JOHNSON',       19910819),
(3,       'LINDA WILLIAMS'  ,       19810616),
(4,       'BARBARA JONES'   ,       20170813),
(5,       'ELIZABETH BROWN' ,       19320513),
(6,       'JENNIFER DAVIS'  ,       20000416),
(7,       'MARIA MILLER'    ,19700604),
(8,       'SUSAN WILSON'    ,       19420623),
(9,       'MARGARET MOORE'  ,       19870921),
(10,    'DOROTHY TAYLOR'  ,       19921215);
```

```
CREATE TABLE IF NOT EXISTS movies (
   id   SMALLINT UNSIGNED UNIQUE NOT NULL,
   name VARCHAR(45) NOT NULL,
   genre VARCHAR(20) NOT NULL,
   release_date DATE NOT NULL,
   PRIMARY KEY (id)
)  ENGINE=INNODB;

INSERT INTO movies
        (id, name, genre, release_date)
VALUES
(1,      'ACADEMY DINOSAUR',   'Comedy'   ,19721211),
(2,      'ACE GOLDFINGER'  ,       'Thriller' ,19550709),
(3,      'ADAPTATION HOLES',     'Horror'   ,19971124),
(4,      'AFFAIR PREJUDICE',       'Adventure',19510620),
(5,      'AFRICAN EGG'     ,  'Thriller' ,20021209),
(6,      'AGENT TRUMAN'    ,       'Fantasy'  ,20050515),
(7,      'AIRPLANE SIERRA' ,       'Comedy'   ,20010729),
(8,      'AIRPORT POLLOCK' ,       'Scifi'    ,19700729),
(9,      'NOTEBOOK'        ,    'Drama'           ,20030713),
(10,   'ALADDIN CALENDAR',     'Western'  ,19990224);
```

```
CREATE TABLE IF NOT EXISTS reviews (
   user_id  SMALLINT      UNSIGNED NOT NULL,
   movie_id SMALLINT      UNSIGNED NOT NULL,
          rating   SMALLINT      NOT NULL,
   comment  VARCHAR(5000) NOT NULL,
          PRIMARY KEY (user_id, movie_id),
   INDEX (user_id, movie_id),

   FOREIGN KEY (user_id)
     REFERENCES users(id)
     ON UPDATE CASCADE ON DELETE RESTRICT,

   FOREIGN KEY (movie_id)
     REFERENCES movies(id)
     ON UPDATE CASCADE ON DELETE RESTRICT
)  ENGINE=INNODB;

INSERT INTO reviews
          (user_id, movie_id, rating, comment)
VALUES
(1       ,2,       9,       'A Epic Drama of a Feminist And a Mad Scientist who must Battle
a Teacher in The Canadian Rockies'),
```

(3 ,3, 7, 'A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China'),

(10 ,3, 0, 'A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory'),

(4 ,2, 8, 'A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank'),

(4 ,4, 5, 'A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf of Mexico'),

(1 ,9, 1, 'A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China'),

(8 ,7, 9, 'A Touching Saga of a Hunter And a Butler who must Discover a Butler in A Jet Boat'),

(8 ,9, 7, 'A Epic Tale of a Moose And a Girl who must Confront a Monkey in Ancient India'),

(5 ,8, 6, 'A Thoughtful Panorama of a Database Administrator And a Mad Scientist who must Outgun a Mad Scientist in A Jet Boat'),

(3 ,7, 6, 'A Action-Packed Tale of a Man And a Lumberjack who must Reach a Feminist in Ancient China'),

(5 ,2, 1, 'A Boring Epistle of a Butler And a Cat who must Fight a Pastry Chef in A MySQL Convention'),

(1 ,5, 1, 'A Fanciful Saga of a Hunter And a Pastry Chef who must Vanquish a Boy in Australia'),

```
(9      ,7,     9,      'A Action-Packed Drama of a Dentist And a Crocodile who must
Battle a Feminist in The Canadian Rockies'),
(6      ,6,     6,      'A Emotional Drama of a A Shark And a Database Administrator
who must Vanquish a Pioneer in Soviet Georgia'),
(5      ,1,     10,     'A Brilliant Drama of a Cat And a Mad Scientist who must Battle a
Feminist in A MySQL Convention'),
(6      ,9,     6,      'A Fast-Paced Drama of a Robot And a Composer who must Battle
a Astronaut in New Orleans'),
(4      ,8,     1,      'A Fast-Paced Character Study of a Composer And a Dog who
must Outgun a Boat in An Abandoned Fun House'),
(4      ,1,     9,      'A Thoughtful Drama of a Composer And a Feminist who must
Meet a Secret Agent in The Canadian Rockies'),
(8      ,2,     1,      'A Emotional Display of a Pioneer And a Technical Writer who
must Battle a Man in A Baloon');



CREATE TABLE IF NOT EXISTS leads (
   actor_id SMALLINT UNSIGNED NOT NULL,
   movie_id SMALLINT UNSIGNED NOT NULL,

        PRIMARY KEY (actor_id, movie_id),
   INDEX (actor_id, movie_id),
```

```
   FOREIGN KEY (actor_id)
    REFERENCES actors(id)
    ON UPDATE CASCADE ON DELETE RESTRICT,

   FOREIGN KEY (movie_id)
    REFERENCES movies(id)
    ON UPDATE CASCADE ON DELETE RESTRICT
)  ENGINE=INNODB;

INSERT INTO leads
        (actor_id, movie_id)
VALUES
(4 ,9),
(9 ,8),
(9 ,4),
(6 ,8),
(10,7),
(7 ,1),
(1 ,2),
(9 ,1),
(1 ,10),
(1 ,1),
(7 ,9),
```

```
(10,6),

(6 ,7),

(3 ,10),

(9 ,5),

(9 ,2),

(7 ,7),

(1 ,7),

(1 ,8),

(6 ,5),

(8 ,6),

(2 ,1),

(1 ,3),

(5, 9),

(4, 1);
```

III. Questions and answers:

ALL WORKING SCRIPT IS BOLDED IN RED (SHADED GREY) → WORKING PERFECTLY IN MYSQL BENCHMARK BUT FOR SOME REASONS IS NOT WORKING IN GOOGLE CLOUD SHELL

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.

```
SELECT users.name
        FROM users, reviews, movies
        WHERE MONTH(users.date_of_birth) = 4
                AND users.id = reviews.user_id
                AND movies.name = "NOTEBOOK"
                AND movies.id = reviews.movie_id
                AND reviews.rating <= 8
        ORDER BY users.name DESC;
```

This query works because in the WHERE clause:

- USERS table is connected to REVIEWS table via user_id
- MOVIES table is connected to REVIEWS table via movie_id
- Reviews.rating <= 8 (rate at most 8)
- Movies.name = “Notebook” (movie name is “Notebook”
- Month(users.date_of_birth) = 4 means user was born in April

2. Find user 'John Doe''s favorite type of movie genre(s) based on his movie reviews 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.

Part 1: find user "JOHN DOE"'s favorite type of movie genres based on his movie reviews ratings:

I Assume that his favorite type of movie genre is the genre that has highest average rating based on his review. I break this problem down into smaller parts. First, the query to select all reviews made by John Doe:

```
SELECT users.name, movies.genre AS genre, reviews.rating AS RATING
        FROM users, movies ,reviews
        WHERE movies.id = reviews.movie_id
                        AND users.id = reviews.user_id
       AND users.name = "JOHN DOE"
ORDER BY genre;
```

Then I calculate average of all ratings of reviews made by JOHN DOE:

```
SELECT AVG(RATING) AS AVG_RATING, genre
FROM (SELECT users.name, movies.genre AS genre, reviews.rating AS RATING
        FROM users, movies ,reviews
        WHERE movies.id = reviews.movie_id
                        AND users.id = reviews.user_id
       AND users.name = "JOHN DOE") AS Q2_1
GROUP BY genre;
```

From this, I get the favorite type by getting the maximum of average rating of each genre rated by JOHN DOE:

```
SELECT name, genre, MAX(AVG_RATING)
        FROM
        (SELECT AVG(RATING) AS AVG_RATING, genre, name
                FROM
    (SELECT users.name AS name, movies.genre AS genre, reviews.rating AS RATING
                        FROM users, movies ,reviews
                                WHERE movies.id = reviews.movie_id
```

```
                    AND   users.id = reviews.user_id
                    AND   users.name = "JOHN DOE") AS Q2_1
        GROUP BY genre) AS Q2_2;
```

Part 2: 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.

```
SELECT movies.name, movies.genre
        FROM
  movies, (SELECT name, genre, MAX(AVG_RATING)
                    FROM
                    (SELECT AVG(RATING) AS AVG_RATING, genre, name
                            FROM
          (SELECT users.name AS name, movies.genre AS genre, reviews.rating AS RATING
                                    FROM users, movies ,reviews
                                    WHERE movies.id = reviews.movie_id
                                            AND   users.id = reviews.user_id
                                            AND   users.name = "JOHN DOE") AS Q2_1
                            GROUP BY genre) AS Q2_2) AS userFAV
  WHERE movies.genre = userFAV.genre
  ORDER BY movies.genre, movies.name ASC;
```

3. List the movie id(s) with most male lead. Sort the ids in descending order.

First I list the all movie ID with corresponding number of male leads:

```
SELECT leads.movie_id, COUNT(leads.actor_id) as numMaleLead
        FROM leads, actors
   WHERE leads.actor_id = actors.id
                AND actors.gender = "M"
        GROUP BY leads.movie_id
        ORDER BY numMaleLead DESC;
```

Then I select a movie with the most male leads:

```
SELECT max(numMaleLead), movieid
FROM (SELECT leads.movie_id AS movieid, COUNT(leads.actor_id) as numMaleLead
        FROM leads, actors
   WHERE leads.actor_id = actors.id
                AND actors.gender = "M"
        GROUP BY leads.movie_id
        ORDER BY numMaleLead DESC) as Q3_1;
```

Finally, I select all movies with the most number of male lead

```
SELECT movieid, numMaleLead
        FROM
        (SELECT leads.movie_id AS movieid, COUNT(leads.actor_id) as numMaleLead
                FROM leads, actors
                WHERE leads.actor_id = actors.id
                        AND actors.gender = "M"
                GROUP BY leads.movie_id
                ORDER BY numMaleLead DESC) as Q3_1
        WHERE numMaleLead = (SELECT MAX(numMaleLead) FROM (SELECT leads.movie_id AS
movieid, COUNT(leads.actor_id) as numMaleLead

        FROM leads, actors
```

```
WHERE leads.actor_id = actors.id

        AND actors.gender = "M"

GROUP BY leads.movie_id

ORDER BY numMaleLead DESC) as Q3_1)
ORDER BY movieid DESC;
```

4. List the name(s) of all comedy movie(s) that were released before 2006 and have reviews rating better than average rating of all movies, sorted in ascending order. Note that you should compute the average of movie average ratings, not the average of all ratings. E.g. movie A got reviewss 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.

First, the below query select all comedy released before 2006:

```
SELECT movies.name, movies.release_date, reviews.rating
        FROM movies, reviews
        WHERE YEAR(movies.release_date)<2006
                AND movies.genre = "COMEDY"
    AND movies.id = reviews.movie_id
        ORDER BY movies.name;
```

Then I compute the average rating for each movie:

```
SELECT reviews.movie_id, AVG(reviews.rating) AS avgRating
        FROM reviews
  GROUP BY reviews.movie_id;
```

Then I calculate average rating for all comedy movies before 2006:

```
SELECT movieid, name, AVG(rating) AS avgRating
        FROM (SELECT reviews.movie_id AS movieid, movies.name AS name,
movies.release_date as release_date, reviews.rating as rating
                        FROM movies, reviews
                        WHERE YEAR(movies.release_date)<2006
                                AND movies.genre = "COMEDY"
                                AND movies.id = reviews.movie_id
                        ORDER BY movies.name) AS Q4_1
  GROUP BY movieid
ORDER BY movieid;
```

Then I calculate average rating of all average rating:

```
SELECT AVG(avgRating)
FROM (SELECT reviews.movie_id, AVG(reviews.rating) AS avgRating
```

```
        FROM reviews
    GROUP BY reviews.movie_id) AS Q4_2;
```

Finally, the question is answered by the following query

```
SELECT name
        FROM
   (SELECT movieid, name, AVG(rating) AS avgRating
                FROM
      (SELECT reviews.movie_id AS movieid, movies.name AS name, movies.release_date as
release_date, reviews.rating as rating
                        FROM movies, reviews
                        WHERE YEAR(movies.release_date)<2006
                                AND movies.genre = "COMEDY"
                                AND movies.id = reviews.movie_id
                        ORDER BY movies.name) AS Q4_1
                GROUP BY movieid
                ORDER BY movieid) as Q4_3
        WHERE avgRating> (SELECT AVG(avgRating)
                FROM
     (SELECT reviews.movie_id, AVG(reviews.rating) AS avgRating
                        FROM reviews
                GROUP BY reviews.movie_id) AS Q4_2)
        ORDER BY avgRating ASC;
```

5. List the movie id(s) and average reviews(s) where the average reviews higher than 9 and one of their leading actors is the actors 'Mark Clarkson'. Sort the output by average reviews and then movie ids.

First, I calculate average rating of all movies:

```
SELECT movieid, name, AVG(rating) AS avgRating
        FROM (SELECT reviews.movie_id AS movieid, movies.name AS name,
movies.release_date as release_date, reviews.rating as rating
                        FROM movies, reviews
                        WHERE movies.id = reviews.movie_id) AS Q4_1
   GROUP BY movieid
ORDER BY movieid;
```

Then the question is answered by the following query:

```
SELECT movieid, name, avgRating
        FROM
   (SELECT movieid, name, AVG(rating) AS avgRating
                FROM
     (SELECT reviews.movie_id AS movieid, movies.name AS name, movies.release_date as
release_date, reviews.rating as rating
                        FROM movies, reviews, actors, leads
                        WHERE movies.id = reviews.movie_id
                                AND movies.id = leads.movie_id
                                AND leads.actor_id = actors.id
                                AND actors.name = "MARK CLARKSON"
                        ORDER BY movies.name) AS Q5_1
                GROUP BY movieid
                ORDER BY movieid) AS Q5_2
        WHERE avgRating> 9
        ORDER BY avgRating, movieid ASC;
```

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 (actors1 and actors2). This might result in duplicate data where two rows might have the same actors but in different columns. Here is an example of two actors that played the lead in two movies together.

The following query works because I create 2 duplicate tables of LEADS as r1 and r2. Then I perform inner join between the two tables with the criteria of same movie_id and r1.actor_id>r2.actor_id (to prevent counting the shared_movie twice than reality.

Then the second part of the query is to extract the name of the corresponding actors with those ID.

```
SELECT actors1.name, actors2.name, shared_movie
        FROM actors actors1,actors actors2,
  (SELECT r1.actor_id as actors_a, r2.actor_id as actors_b, count(r1.movie_id) as shared_movie
                FROM leads r1 inner join leads r2 on r1.movie_id = r2.movie_id and r1.actor_id > r2.actor_id
                GROUP BY r1.actor_id, r2.actor_id
                ORDER BY shared_movie desc) as Q6_1
  WHERE shared_movie =
                (SELECT max(shared_movie)
                        FROM
       (SELECT r1.actor_id as actors_a, r2.actor_id as actors_b, count(r1.movie_id) as shared_movie
                                FROM leads r1 inner join leads r2 on r1.movie_id = r2.movie_id and r1.actor_id > r2.actor_id
                                GROUP BY r1.actor_id, r2.actor_id
                                ORDER BY shared_movie desc) as Q6_2)
                AND actors1.id = actors_a
     AND actors2.id = actors_b;
```

IV. Query results:

Below are the screen shot of results by running the script on MYSQL:

1. Question 1:

	name
▶	JENNIFER DAVIS

2. Question 2:

	name	genre	MAX(avg_rating)
▶	JOHN DOE	Thriller	5.0000

	name	genre
▶	ACE GOLDFINGER	Thriller
	AFRICAN EGG	Thriller

3. Question 3:

	movieid	numMaleLead
▶	1	4

4. Question 4:

	name
▶	AIRPLANE SIERRA
	ACADEMY DINOSAUR

5. Question 5:

	movieid	name	avgrating
▶	1	ACADEMY DINOSAUR	9.5000

6. Question 6:

	name	name	shared_movie
▶	JOE SWANK	PENELOPE GUINESS	3