Assignment 2

Shreyash Annapureddy: 1072336621

Contents

ASSUMPTIONS

- The GENDER column is assumed be a lower-case character
- The NAME column in ACTORS is assumed to have capitalization where grammatically appropriate
- Favorite GENRE of a USER is assumed to be all the genres that USER has given their highest rating
- When determining the most number of ACTORS who have lead together in the most number of movies, comparing only pair-wise

QUERY 1

The query requires a check of 3 unique attributes to determine the name of the user. The movie name, user's date of birth, and rating are found in the following 3 tables:

1. USERS
2. REVIEWS
3. MOVIES

Hence it is necessary to join the 3 tables where creating a table of all the reviews made by each user and the corresponding movie. Then using the movie, birth month and rating condition we just select the names that match these conditions. This leads to the following query:

```
SELECT NAME
FROM USERS u JOIN REVIEWS r JOIN MOVIES m ON u.USER_ID = r.R_USER_ID
WHERE r.R_MOVIE_ID = m.MOVIE_ID AND MONTH(u.DATE_OF_BIRTH) = 4 AND m.MOVIE_NAME = "Notebook" AND
r.MOVIE_RATING <= 8
ORDER BY NAME DESC;
```

QUERY 2

The query requires to find a user called "John Doe" and all their favorite movies and the movie's corresponding genre. Favorite is defined as all the ratings equal to the max rating John doe has given to any number of movies.

The first step is to derive a table that lists all the movies and the corresponding genre that John Doe has reviewed which is constructed from the following tables:

1. USERS
2. REVIEWS
3. MOVIES

```
SELECT MOVIE_NAME, GENRE, MOVIE_RATING
FROM USERS u JOIN REVIEWS r JOIN MOVIES m ON u.USER_ID = r.R_USER_ID AND r.R_MOVIE_ID = m.MOVIE_ID
WHERE u.NAME = "John Doe"
```

The result from the query is:

MOVIE_NAME	GENRE	MOVIE_RATING
Star Wars: Force Awakens	Sci-Fi	7.0
Star Wars: A New Hope	Sci-Fi	6.0
Blade Runner 2049	Sci_Fi	8.0
Iron Man	Sci-Fi	8.0
Avengers: Infinity War	Sci-Fi	8.0
Notebook	Romance	6.0
Hot Fuzz	Comedy	6.0
Groundhog Day	Comedy	4.0
Rush Hour	Comedy	9.0
Happy Gilmore	Comedy	8.0
Die Hard	Action	8.0

11 rows in set (0.03 sec)

The next step is to find the rating of movies and the corresponding genre that John Doe has given a rating of 9, in this case the only movie that matches this condition is Rush Hour. Thus, find MOVIE_RATING that is equal to the max MOVIE_RATING in the derived table:

```
SELECT MOVIE_NAME, GENRE
FROM USERS u JOIN REVIEWS r JOIN MOVIES m ON u.USER_ID = r.R_USER_ID AND r.R_MOVIE_ID = m.MOVIE_ID
WHERE u.NAME = "John Doe" AND MOVIE_RATING = (SELECT MAX(MOVIE_RATING)
FROM USERS u JOIN REVIEWS r JOIN MOVIES m ON u.USER_ID = r.R_USER_ID AND r.R_MOVIE_ID = m.MOVIE_ID
WHERE u.NAME = "John Doe")
ORDER BY GENRE ASC, MOVIE_NAME ASC;
```

The solution table will be:

MOVIE_NAME	GENRE
Rush Hour	Comedy

1 row in set (0.04 sec)

QUERY 3

The first step is to derive a table that lists all the actors that are male, this is done by joining the following tables:

1. LEADS
2. ACTORS

This is accomplished with the following query:

```
SELECT LEAD_MOVIE_ID
FROM LEADS l JOIN ACTORS a ON l.LEAD_ACTOR_ID = a.ACTOR_ID WHERE a.GENDER = "m";
```

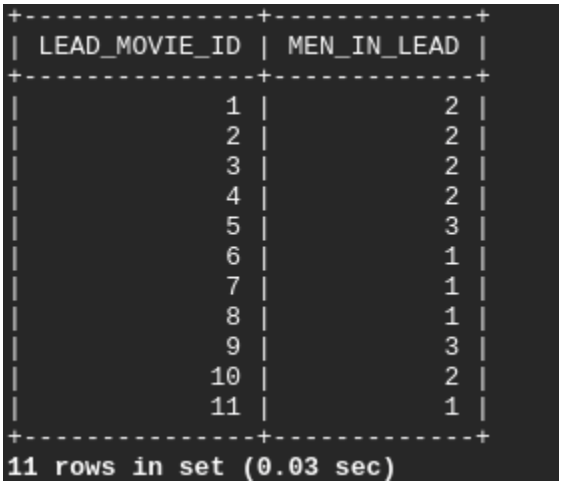
The result of the query is the following:

```
+---------------+
| LEAD_MOVIE_ID |
+---------------+
|             1 |
|             2 |
|             3 |
|             1 |
|             2 |
|             3 |
|             6 |
|             4 |
|             5 |
|             5 |
|             5 |
|             7 |
|             8 |
|             9 |
|             9 |
|            10 |
|            11 |
|             4 |
|             9 |
|            10 |
+---------------+
20 rows in set (0.04 sec)
```

The next step is to count how many MOVIE_ID are in the table for each unique MOVIE_ID. We the count for each MOVIE_ID by grouping the table by the MOVIE_ID. The query for this table is the following:

```
SELECT LEAD_MOVIE_ID, COUNT(LEAD_MOVIE_ID) MEN_IN_LEAD
FROM LEADS l JOIN ACTORS a ON l.LEAD_ACTOR_ID = a.ACTOR_ID WHERE a.GENDER = "m"
GROUP BY LEAD_MOVIE_ID;
```

The table that results from the query is:



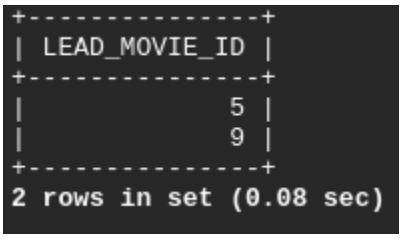
LEAD_MOVIE_ID	MEN_IN_LEAD
1	2
2	2
3	2
4	2
5	3
6	1
7	1
8	1
9	3
10	2
11	1

11 rows in set (0.03 sec)

The next and last step is to select the counted MOVIE_ID that is equal to max counted value from the same table. This leads to the final query of:

```
SELECT LEAD_MOVIE_ID
FROM LEADS l JOIN ACTORS a ON l.LEAD_ACTOR_ID = a.ACTOR_ID WHERE a.GENDER = "m"; GROUP BY
LEAD_MOVIE_ID
HAVING COUNT(LEAD_MOVIE_ID) = (SELECT MAX(MEN_IN_LEAD)
FROM (
SELECT LEAD_MOVIE_ID, COUNT(LEAD_MOVIE_ID) MEN_IN_LEAD
FROM LEADS l JOIN ACTORS a ON l.LEAD_ACTOR_ID = a.ACTOR_ID WHERE a.GENDER = "m"
GROUP BY LEAD_MOVIE_ID) as temp_table)
ORDER BY LEAD_MOVIE_ID ASC;
```

Which results in the following table:



LEAD_MOVIE_ID
5
9

2 rows in set (0.08 sec)

QUERY 4

It is necessary to determine the average of average ratings of all the films, this is done by first finding the average rating for each film individually with the following query:

```
SELECT MOVIE_NAME, AVG(MOVIE_RATING) as movie_avg
  FROM MOVIES m JOIN REVIEWS r ON m.MOVIE_ID = r.R_MOVIE_ID
  GROUP BY MOVIE_NAME
  ORDER BY movie_avg;
```

This resulting table is:

```
+--------------------------+-----------+
| MOVIE_NAME               | movie_avg |
+--------------------------+-----------+
| Star Wars: Force Awakens |   2.50000 |
| Notebook                 |   3.50000 |
| Groundhog Day            |   3.50000 |
| Die Hard                 |   5.00000 |
| Avengers: Infinity War   |   6.00000 |
| Hot Fuzz                 |   6.25000 |
| Star Wars: A New Hope    |   6.25000 |
| Blade Runner 2049        |   7.25000 |
| Happy Gilmore            |   7.50000 |
| Rush Hour                |   8.50000 |
| Iron Man                 |   9.25000 |
+--------------------------+-----------+
11 rows in set (0.04 sec)
```

The next step is to find the average of all the averages in the above table, this is done with the following query:

```
SELECT AVG(movie_avg) as total_avg
FROM(
  SELECT MOVIE_NAME, AVG(MOVIE_RATING) as movie_avg
  FROM MOVIES m JOIN REVIEWS r ON m.MOVIE_ID = r.R_MOVIE_ID
  GROUP BY MOVIE_NAME
  ORDER BY movie_avg
)as avgs;
```

Which results in the following single valued table:

```
+-------------+
| total_avg   |
+-------------+
| 5.954545454 |
+-------------+
1 row in set (0.04 sec)
```

Finally, query must select comedy movies that are released prior to 2006 and have a movie rating greater than the above total_avg value. Since a movie is reviewed by multiple people the result may return multiple entries for the same movie, to avoid this issue only distinct entries are selected.

This results in the final query:

```
SELECT DISTINCT MOVIE_NAME
FROM (MOVIES m JOIN REVIEWS r ON m.MOVIE_ID = r.R_MOVIE_ID)
WHERE m.GENRE = "Comedy" AND YEAR(m.RELEASE_DATE) < 2006
AND r.MOVIE_RATING > (SELECT AVG(movie_avg) as total_avg
FROM(
  SELECT MOVIE_NAME, AVG(MOVIE_RATING) as movie_avg
  FROM MOVIES m JOIN REVIEWS r ON m.MOVIE_ID = r.R_MOVIE_ID
  GROUP BY MOVIE_NAME
  ORDER BY movie_avg
)as avgs)
ORDER BY MOVIE_NAME ASC;
```

The result is the following table:

```
+---------------+
| MOVIE_NAME    |
+---------------+
| Groundhog Day |
| Happy Gilmore |
| Rush Hour     |
+---------------+
3 rows in set (0.04 sec)
```

QUERY 5

Like QUERY 4, the average rating score for each movie is determined with the following query:

```
SELECT R_MOVIE_ID, AVG(MOVIE_RATING) as movie_avg
FROM REVIEWS r JOIN LEADS l ON r.R_MOVIE_ID = l.LEAD_MOVIE_ID
GROUP BY R_MOVIE_ID;
```

The result of the query is:

```
+------------+-----------+
| R_MOVIE_ID | movie_avg |
+------------+-----------+
|          1 |   2.50000 |
|          2 |   6.25000 |
|          3 |   7.25000 |
|          4 |   9.25000 |
|          5 |   6.00000 |
|          6 |   3.50000 |
|          7 |   6.25000 |
|          8 |   3.50000 |
|          9 |   8.50000 |
|         10 |   7.50000 |
|         11 |   5.00000 |
+------------+-----------+
11 rows in set (0.03 sec)
```

This table does not provide information on the actors acting on the movie, hence joining this table with LEADS and ACTORS will provide this information. From this new table we query where the movie_avg is greater than 9 and the actor's name is Mark Clarkson, this has the final query of:

```
SELECT R_MOVIE_ID as Movie_ID, movie_avg
FROM(SELECT R_MOVIE_ID, AVG(MOVIE_RATING) as movie_avg
    FROM REVIEWS r JOIN LEADS l ON r.R_MOVIE_ID = l.LEAD_MOVIE_ID
    GROUP BY R_MOVIE_ID) as avgs
JOIN ACTORS a JOIN LEADS l ON avgs.R_MOVIE_ID = l.LEAD_MOVIE_ID AND a.ACTOR_ID = l.LEAD_ACTOR_ID
WHERE movie_avg > 9 AND a.ACTOR_NAME = "Mark Clarkson"
ORDER BY movie_avg DESC, R_MOVIE_ID DESC;
```

The final table result is:

```
+----------+-----------+
| Movie_ID | movie_avg |
+----------+-----------+
|        4 |   9.25000 |
+----------+-----------+
1 row in set (0.04 sec)
```

QUERY 6

The first step is to find all combinations of actors that have worked together with respect to the movie_id. This can be achieved by joining LEADS, ACTORS and MOVIES to create two of the same table. Then join the two same tables ensuring that the same two actors are not joined together and the movie_id that the actors acted in are the same. Then it is a matter of counting the pairs. It is important to note that only all pair-wise combinations of actors are being checked. This means that if a movie has three are more actors who have worked in multiple movies, then all pair-wise combinations amongst the three will be created.

The query for the all valid pair-wise combinations is:

```
SELECT ACTOR_1, ACTOR_2, COUNT(*) AS NUM_MOVIES_TOGETHER
FROM (SELECT ACTORS.ACTOR_NAME AS ACTOR_1, MOVIES.MOVIE_NAME AS MOVIE_ID1 FROM LEADS
  JOIN ACTORS ON ACTORS.ACTOR_ID = LEADS.LEAD_ACTOR_ID
    JOIN MOVIES ON MOVIES.MOVIE_ID = LEADS.LEAD_MOVIE_ID) AS TEST_1,
    (SELECT ACTORS.ACTOR_NAME AS ACTOR_2, MOVIES.MOVIE_NAME AS MOVIE_ID2 FROM LEADS
  JOIN ACTORS ON ACTORS.ACTOR_ID = LEADS.LEAD_ACTOR_ID
    JOIN MOVIES ON MOVIES.MOVIE_ID = LEADS.LEAD_MOVIE_ID) AS TEST_2
WHERE MOVIE_ID1 = MOVIE_ID2 AND ACTOR_1!=ACTOR_2
GROUP BY ACTOR_1, ACTOR_2;
```

This results in the following table:

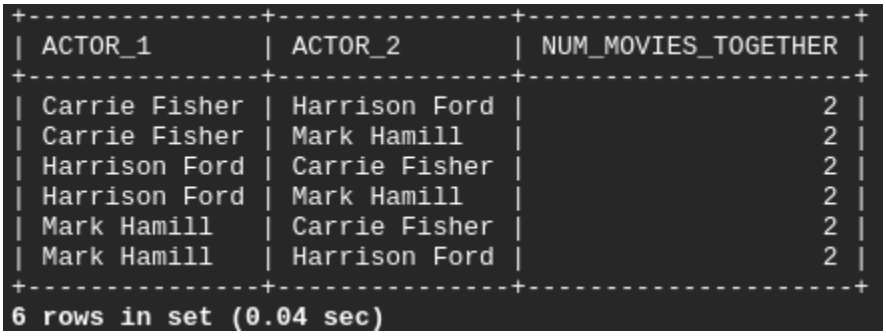
ACTOR_1	ACTOR_2	NUM_MOVIES_TOGETHER
Adam Sandler	Mark Clarkson	1
Carrie Fisher	Daisy Ridley	1
Carrie Fisher	Harrison Ford	2
Carrie Fisher	Mark Hamill	2
Chris Evans	Chris Hemsworth	1
Chris Evans	Robert Downey Jr	1
Chris Hemsworth	Chris Evans	1
Chris Hemsworth	Robert Downey Jr	1
Chris Tucker	Jackie Chan	1
Chris Tucker	Mark Clarkson	1
Daisy Ridley	Carrie Fisher	1
Daisy Ridley	Harrison Ford	1
Daisy Ridley	Mark Hamill	1
Harrison Ford	Carrie Fisher	2
Harrison Ford	Daisy Ridley	1
Harrison Ford	Mark Hamill	2
Harrison Ford	Ryan Gosling	1
Jackie Chan	Chris Tucker	1
Jackie Chan	Mark Clarkson	1
Mark Clarkson	Adam Sandler	1
Mark Clarkson	Chris Tucker	1
Mark Clarkson	Jackie Chan	1
Mark Clarkson	Robert Downey Jr	1
Mark Hamill	Carrie Fisher	2
Mark Hamill	Daisy Ridley	1
Mark Hamill	Harrison Ford	2
Robert Downey Jr	Chris Evans	1
Robert Downey Jr	Chris Hemsworth	1
Robert Downey Jr	Mark Clarkson	1
Ryan Gosling	Harrison Ford	1

30 rows in set (0.04 sec)

Like QUERY 3, selecting the max of NUM_MOVIES_TOGETHER will result in all actors who have acted together in the greatest number of films. The query is:

```
SELECT ACTOR_1, ACTOR_2, COUNT(*) AS NUM_MOVIES_TOGETHER
FROM (SELECT ACTORS.ACTOR_NAME AS ACTOR_1, MOVIES.MOVIE_NAME AS MOVIE_ID1 FROM LEADS
  JOIN ACTORS ON ACTORS.ACTOR_ID = LEADS.LEAD_ACTOR_ID
    JOIN MOVIES ON MOVIES.MOVIE_ID = LEADS.LEAD_MOVIE_ID) AS TEST_1,
    (SELECT ACTORS.ACTOR_NAME AS ACTOR_2, MOVIES.MOVIE_NAME AS MOVIE_ID2 FROM LEADS
  JOIN ACTORS ON ACTORS.ACTOR_ID = LEADS.LEAD_ACTOR_ID
    JOIN MOVIES ON MOVIES.MOVIE_ID = LEADS.LEAD_MOVIE_ID) AS TEST_2
WHERE MOVIE_ID1 = MOVIE_ID2 AND ACTOR_1!=ACTOR_2
GROUP BY ACTOR_1, ACTOR_2
HAVING COUNT(*) = (
SELECT MAX(NUM_MOVIES_TOGETHER)
FROM(
  SELECT ACTOR_1, ACTOR_2, COUNT(*) AS NUM_MOVIES_TOGETHER
  FROM (SELECT ACTORS.ACTOR_NAME AS ACTOR_1, MOVIES.MOVIE_NAME AS MOVIE_ID1 FROM LEADS
    JOIN ACTORS ON ACTORS.ACTOR_ID = LEADS.LEAD_ACTOR_ID
      JOIN MOVIES ON MOVIES.MOVIE_ID = LEADS.LEAD_MOVIE_ID) AS TEST_1,
      (SELECT ACTORS.ACTOR_NAME AS ACTOR_2, MOVIES.MOVIE_NAME AS MOVIE_ID2 FROM LEADS
    JOIN ACTORS ON ACTORS.ACTOR_ID = LEADS.LEAD_ACTOR_ID
      JOIN MOVIES ON MOVIES.MOVIE_ID = LEADS.LEAD_MOVIE_ID) AS TEST_2
  WHERE MOVIE_ID1 = MOVIE_ID2 AND ACTOR_1!=ACTOR_2
  GROUP BY ACTOR_1, ACTOR_2
) AS TABLE_RESULT);
```

The final resulting table is:



ACTOR_1	ACTOR_2	NUM_MOVIES_TOGETHER
Carrie Fisher	Harrison Ford	2
Carrie Fisher	Mark Hamill	2
Harrison Ford	Carrie Fisher	2
Harrison Ford	Mark Hamill	2
Mark Hamill	Carrie Fisher	2
Mark Hamill	Harrison Ford	2

6 rows in set (0.04 sec)

APPENDIX A: TABLE DEFINITION

```
--Creating database
CREATE DATABASE MOVIE_REVIEW;
-- Creating multiple tables inside the database
USE MOVIE_REVIEW;

CREATE TABLE USERS (
  USER_ID            INTEGER        not null AUTO_INCREMENT UNIQUE,
  NAME               VARCHAR(35)  not null,
  DATE_OF_BIRTH   DATE             not null,
  PRIMARY KEY(USER_ID)
);
CREATE TABLE MOVIES(
  MOVIE_ID         INTEGER         not null AUTO_INCREMENT UNIQUE,
  MOVIE_NAME    VARCHAR(35)   not null,
  GENRE              VARCHAR(35)   not null,
  RELEASE_DATE    DATE             not null,
  PRIMARY KEY(MOVIE_ID)
);
CREATE TABLE REVIEWS(
  R_USER_ID              INTEGER          not null,
  R_MOVIE_ID             INTEGER          not null,
  MOVIE_RATING          DECIMAL(3,1)    not null DEFAULT 0.00,
  REVIEW_COMMENT    VARCHAR(5000)   DEFAULT null,
  -- To create a M:N relationship creating a constraint that is a primary key
  -- combination of USER and MOVIE ID
  FOREIGN KEY(R_USER_ID) REFERENCES USERS(USER_ID) on UPDATE CASCADE,
  FOREIGN KEY(R_MOVIE_ID) REFERENCES MOVIES(MOVIE_ID) on UPDATE CASCADE,
  PRIMARY KEY(R_USER_ID, R_MOVIE_ID)
);
CREATE TABLE ACTORS(
  ACTOR_ID           INTEGER         not null AUTO_INCREMENT UNIQUE,
  ACTOR_NAME      VARCHAR(35)  not null,
  GENDER             CHAR(1)         not null,
  DATE_OF_BIRTH    DATE             not null,
  PRIMARY KEY(ACTOR_ID)
);
CREATE TABLE LEADS(
  LEAD_ACTOR_ID  INTEGER not null,
  LEAD_MOVIE_ID  INTEGER not null,

  FOREIGN KEY(LEAD_ACTOR_ID) REFERENCES ACTORS(ACTOR_ID) ON UPDATE CASCADE,
  FOREIGN KEY(LEAD_MOVIE_ID) REFERENCES MOVIES(MOVIE_ID) ON UPDATE CASCADE,
  PRIMARY KEY(LEAD_ACTOR_ID, LEAD_MOVIE_ID)
);
```

APPENDIX B: TEST DATA

```
INSERT INTO MOVIES (MOVIE_NAME, GENRE, RELEASE_DATE) values ("Star Wars: Force Awakens","Sci-Fi",'2015-12-18');
INSERT INTO MOVIES (MOVIE_NAME, GENRE, RELEASE_DATE) values ("Star Wars: A New Hope","Sci-Fi",'1977-05-25');
INSERT INTO MOVIES (MOVIE_NAME, GENRE, RELEASE_DATE) values ("Blade Runner 2049","Sci_Fi",'2017-10-03');
INSERT INTO MOVIES (MOVIE_NAME, GENRE, RELEASE_DATE) values ("Iron Man","Sci-Fi",'2008-05-02');
INSERT INTO MOVIES (MOVIE_NAME, GENRE, RELEASE_DATE) values ("Avengers: Infinity War","Sci-Fi",'2018-04-27');
INSERT INTO MOVIES (MOVIE_NAME, GENRE, RELEASE_DATE) values ("Notebook","Romance",'2004-06-25');
INSERT INTO MOVIES (MOVIE_NAME, GENRE, RELEASE_DATE) values ("Hot Fuzz","Comedy",'2007-03-14');
INSERT INTO MOVIES (MOVIE_NAME, GENRE, RELEASE_DATE) values ("Groundhog Day","Comedy",'1993-02-12');
INSERT INTO MOVIES (MOVIE_NAME, GENRE, RELEASE_DATE) values ("Rush Hour","Comedy",'1998-09-18');
INSERT INTO MOVIES (MOVIE_NAME, GENRE, RELEASE_DATE) values ("Happy Gilmore","Comedy",'1996-02-16');
INSERT INTO MOVIES (MOVIE_NAME, GENRE, RELEASE_DATE) values ("Die Hard","Action",'1988-07-15');

INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Harrison Ford","m",'1942-07-13');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Mark Hamill","m",'1951-09-25');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Carrie Fisher","f",'1956-10-21');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Daisy Ridley","f",'1992-04-10');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Ryan Gosling","m",'1980-11-12');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Robert Downey Jr","m",'1965-04-04');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Chris Hemsworth","m",'1983-08-11');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Chris Evans","m",'1981-06-13');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Simon Pegg","m",'1970-02-14');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Bill Murray","m",'1950-09-21');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Jackie Chan","m",'1954-04-07');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Chris Tucker","m",'1971-08-31');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Adam Sandler","m",'1966-09-09');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Bruce Willis","m",'1955-03-19');
INSERT INTO ACTORS (ACTOR_NAME, GENDER, DATE_OF_BIRTH) values ("Mark Clarkson","m",'1974-09-07');

INSERT INTO USERS (NAME, DATE_OF_BIRTH) values ("Shreyash Annapureddy",'1994-03-16');
INSERT INTO USERS (NAME, DATE_OF_BIRTH) values ("Saivan Hamama",'1994-02-09');
INSERT INTO USERS (NAME, DATE_OF_BIRTH) values ("John Doe",'1993-04-23');
INSERT INTO USERS (NAME, DATE_OF_BIRTH) values ("Roger Federer",'1975-04-12');

INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (1, 6, 5, "Average");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (1, 8, 3, "Bad");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (1, 9, 6, "Okay");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (1, 1, 2, "Awful");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (1, 3, 1, "Why does this exist");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (1, 5, 7, "Good");
```

```
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (1, 4, 10, "Once in a generation film");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (1, 10, 6, "Okay");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (1, 11, 10, "Once in a generation film");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (1, 7, 2, "Awful");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (1, 2, 10, "Once in a generation film");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (2, 6, 2, "Awful");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (2, 8, 1, "Why does this exist");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (2, 9, 10, "Once in a generation film");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (2, 1, 0, "The Depths of Hell");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (2, 3, 10, "Once in a generation film");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (2, 5, 4, "Not Great");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (2, 4, 9, "Excellent");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (2, 10, 7, "Good");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (2, 11, 1, "Why does this exist");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (2, 7, 7, "Good");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (2, 2, 7, "Good");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (3, 6, 6, "Okay");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (3, 8, 4, "Not Great");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (3, 9, 9, "Excellent");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (3, 1, 7, "Good");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (3, 3, 8, "Great");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (3, 5, 8, "Great");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (3, 4, 8, "Great");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (3, 10, 8, "Great");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (3, 11, 8, "Great");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (3, 7, 6, "Okay");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (3, 2, 6, "Okay");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (4, 6, 1, "Why does this exist");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (4, 8, 6, "Okay");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (4, 9, 9, "Excellent");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (4, 1, 1, "Why does this exist");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (4, 3, 10, "Once in a generation film");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (4, 5, 5, "Average");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (4, 4, 10, "Once in a generation film");
```

```
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (4, 10, 9, "Excellent");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (4, 11, 1, "Why does this exist");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (4, 7, 10, "Once in a generation film");
INSERT INTO REVIEWS (R_USER_ID, R_MOVIE_ID, MOVIE_RATING, REVIEW_COMMENT) values (4, 2, 2, "Awful");


INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (1, 1);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (1, 2);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (1, 3);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (2, 1);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (2, 2);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (3, 1);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (3, 2);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (4, 1);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (5, 3);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (5, 6);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (6, 4);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (6, 5);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (7, 5);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (8, 5);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (9, 7);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (10, 8);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (11, 9);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (12, 9);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (13, 10);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (14, 11);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (15, 9);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (15, 10);
INSERT INTO LEADS (LEAD_ACTOR_ID, LEAD_MOVIE_ID) values (15, 4);
```

APPENDIX C: FINAL QUERY SOLUTIONS

QUERY 1

```
SELECT NAME
FROM USERS u JOIN REVIEWS r JOIN MOVIES m ON u.USER_ID = r.R_USER_ID
WHERE r.R_MOVIE_ID = m.MOVIE_ID AND MONTH(u.DATE_OF_BIRTH) = 4 AND m.MOVIE_NAME = "Notebook" AND
r.MOVIE_RATING <= 8
ORDER BY NAME DESC;
```

QUERY 2

```
SELECT MOVIE_NAME, GENRE
FROM USERS u JOIN REVIEWS r JOIN MOVIES m ON u.USER_ID = r.R_USER_ID AND r.R_MOVIE_ID = m.MOVIE_ID
WHERE u.NAME = "John Doe" AND MOVIE_RATING = (SELECT MAX(MOVIE_RATING)
FROM USERS u JOIN REVIEWS r JOIN MOVIES m ON u.USER_ID = r.R_USER_ID AND r.R_MOVIE_ID = m.MOVIE_ID
WHERE u.NAME = "John Doe")
ORDER BY GENRE ASC, MOVIE_NAME ASC;
```

QUERY 3

```
SELECT LEAD_MOVIE_ID
FROM LEADS l JOIN ACTORS a ON l.LEAD_ACTOR_ID = a.ACTOR_ID WHERE a.GENDER = "m"; GROUP BY
LEAD_MOVIE_ID
HAVING COUNT(LEAD_MOVIE_ID) = (SELECT MAX(MEN_IN_LEAD)
FROM (
SELECT LEAD_MOVIE_ID, COUNT(LEAD_MOVIE_ID) MEN_IN_LEAD
FROM LEADS l JOIN ACTORS a ON l.LEAD_ACTOR_ID = a.ACTOR_ID WHERE a.GENDER = "m"
GROUP BY LEAD_MOVIE_ID) as temp_table)
ORDER BY LEAD_MOVIE_ID ASC;
```

QUERY 4

```
SELECT DISTINCT MOVIE_NAME
FROM (MOVIES m JOIN REVIEWS r ON m.MOVIE_ID = r.R_MOVIE_ID)
WHERE m.GENRE = "Comedy" AND YEAR(m.RELEASE_DATE) < 2006
AND r.MOVIE_RATING > (SELECT AVG(movie_avg) as total_avg
FROM(
  SELECT MOVIE_NAME, AVG(MOVIE_RATING) as movie_avg
  FROM MOVIES m JOIN REVIEWS r ON m.MOVIE_ID = r.R_MOVIE_ID
  GROUP BY MOVIE_NAME
  ORDER BY movie_avg
)as avgs)
ORDER BY MOVIE_NAME ASC;
```

QUERY 5

```
SELECT R_MOVIE_ID as Movie_ID, movie_avg
FROM(SELECT R_MOVIE_ID, AVG(MOVIE_RATING) as movie_avg
    FROM REVIEWS r JOIN LEADS l ON r.R_MOVIE_ID = l.LEAD_MOVIE_ID
    GROUP BY R_MOVIE_ID) as avgs
JOIN ACTORS a JOIN LEADS l ON avgs.R_MOVIE_ID = l.LEAD_MOVIE_ID AND a.ACTOR_ID = l.LEAD_ACTOR_ID
WHERE movie_avg > 9 AND a.ACTOR_NAME = "Mark Clarkson"
ORDER BY movie_avg DESC, R_MOVIE_ID DESC;
```

QUERY 6

```
SELECT ACTOR_1, ACTOR_2, COUNT(*) AS NUM_MOVIES_TOGETHER
FROM (SELECT ACTORS.ACTOR_NAME AS ACTOR_1, MOVIES.MOVIE_NAME AS MOVIE_ID1 FROM LEADS
 JOIN ACTORS ON ACTORS.ACTOR_ID = LEADS.LEAD_ACTOR_ID
  JOIN MOVIES ON MOVIES.MOVIE_ID = LEADS.LEAD_MOVIE_ID) AS TEST_1,
  (SELECT ACTORS.ACTOR_NAME AS ACTOR_2, MOVIES.MOVIE_NAME AS MOVIE_ID2 FROM LEADS
 JOIN ACTORS ON ACTORS.ACTOR_ID = LEADS.LEAD_ACTOR_ID
  JOIN MOVIES ON MOVIES.MOVIE_ID = LEADS.LEAD_MOVIE_ID) AS TEST_2
WHERE MOVIE_ID1 = MOVIE_ID2 AND ACTOR_1!=ACTOR_2
GROUP BY ACTOR_1, ACTOR_2
HAVING COUNT(*) = (
SELECT MAX(NUM_MOVIES_TOGETHER)
FROM(
 SELECT ACTOR_1, ACTOR_2, COUNT(*) AS NUM_MOVIES_TOGETHER
 FROM (SELECT ACTORS.ACTOR_NAME AS ACTOR_1, MOVIES.MOVIE_NAME AS MOVIE_ID1 FROM LEADS
  JOIN ACTORS ON ACTORS.ACTOR_ID = LEADS.LEAD_ACTOR_ID
   JOIN MOVIES ON MOVIES.MOVIE_ID = LEADS.LEAD_MOVIE_ID) AS TEST_1,
   (SELECT ACTORS.ACTOR_NAME AS ACTOR_2, MOVIES.MOVIE_NAME AS MOVIE_ID2 FROM LEADS
  JOIN ACTORS ON ACTORS.ACTOR_ID = LEADS.LEAD_ACTOR_ID
   JOIN MOVIES ON MOVIES.MOVIE_ID = LEADS.LEAD_MOVIE_ID) AS TEST_2
 WHERE MOVIE_ID1 = MOVIE_ID2 AND ACTOR_1!=ACTOR_2
 GROUP BY ACTOR_1, ACTOR_2
) AS TABLE_RESULT);
```