HW 2

TABLE CREATION QUERY

```
CREATE TABLE users (
id INTEGER,
name VARCHAR(30) NOT NULL,
date_of_birth DATE NOT NULL,
PRIMARY KEY(id));

CREATE TABLE movies (
id INTEGER,
name VARCHAR(30) NOT NULL,
genre VARCHAR(20) NOT NULL,
release_date DATE NOT NULL,
PRIMARY KEY(id));

CREATE TABLE reviews (
user_id INTEGER,
movie_id INTEGER,
rating DECIMAL(4,2) NOT NULL,
comment VARCHAR(5000) NOT NULL,
PRIMARY KEY(user_id, movie_id),
FOREIGN KEY(user_id) REFERENCES users(id) ON UPDATE CASCADE,
FOREIGN KEY(movie_id) REFERENCES movies(id) ON UPDATE CASCADE );

CREATE TABLE actors (
id INTEGER,
name VARCHAR(30) NOT NULL,
gender CHAR(1) NOT NULL,
date_of_birth DATE NOT NULL,
PRIMARY KEY(id));

CREATE TABLE lead (
actor_id INTEGER,
movie_id INTEGER,
PRIMARY KEY(actor_id, movie_id),
FOREIGN KEY(actor_id) REFERENCES actors(id) ON UPDATE CASCADE,
FOREIGN KEY(movie_id) REFERENCES movies(id) ON UPDATE CASCADE );
```

DATA

1.users

```
+-------+----------------+---------------+
| id    | name           | date_of_birth |
+-------+----------------+---------------+
| 11111 | Pranali Kanere | 1995-11-11    |
| 12232 | Mark Clarkson  | 2000-02-01    |
| 12345 | John Doe       | 1995-04-11    |
| 12631 | Alex Mary      | 1990-11-29    |
| 13652 | Natasha K      | 1998-04-19    |
| 21532 | Zack           | 2005-04-01    |
| 34521 | Joe Ryan       | 2000-04-30    |
+-------+----------------+---------------+
```

2.movies

```
+-----+----------------------+----------+--------------+
| id  | name                 | genre    | release_date |
+-----+----------------------+----------+--------------+
| 111 | Pirates of Caribbean | Action   | 2007-10-10   |
| 112 | Dark Knight          | Thriller | 2010-01-01   |
| 113 | Notebook             | Comedy   | 2000-11-11   |
| 114 | Men in Black         | Comedy   | 1996-03-05   |
| 115 | Avengers             | Sci-fi   | 2016-10-10   |
+-----+----------------------+----------+--------------+
```

3.reviews

```
+---------+----------+--------+-----------------+
| user_id | movie_id | rating | comment         |
+---------+----------+--------+-----------------+
|   11111 |      111 |   9.50 | Awesome movie   |
|   11111 |      113 |   6.50 | Awesome movie   |
|   11111 |      114 |  10.00 | Could be better |
|   12232 |      112 |   6.00 | Boring          |
|   12345 |      113 |   8.00 | Loves the songs |
|   13652 |      113 |   7.90 | cool            |
|   21532 |      113 |   2.40 | Not good        |
|   34521 |      115 |  10.00 | OMG great       |
+---------+----------+--------+-----------------+
```

4. Actors

```
+------+----------------+--------+---------------+
| id   | name           | gender | date_of_birth |
+------+----------------+--------+---------------+
| 1111 | Orlando Bloom  | M      | 1970-04-12    |
| 1112 | Johnny Depp    | M      | 1962-02-18    |
| 1113 | Christian Bale | M      | 1960-02-16    |
| 1114 | Tom Hardy      | M      | 1950-03-13    |
| 1115 | Mark Clarkson  | F      | 1974-07-05    |
| 1116 | Brad Garrett    | M      | 1959-01-22     |
+------+-----------------+--------+----------------+
```

5. lead

```
+----------+----------+
| actor_id | movie_id |
+----------+----------+
|     1111 |      111 |
|     1112 |      111 |
|     1115 |      111 |
|     1116 |      111 |
|     1113 |      112 |
|     1114 |      113 |
|     1115 |      114 |
|     1111 |      115 |
|     1116 |      115 |
+----------+----------+
```

GIVEN QUERIES

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 name
FROM users
WHERE id
IN ( SELECT r.user_id
     FROM  movies m, reviews r
     WHERE m.name="Notebook"
     AND  m.id=r.movie_id
     AND  r.rating<=8)
AND month(date_of_birth)=4;
```

```
+-----------+
| name      |
+-----------+
| John Doe  |
| Natasha K |
| Zack      |
+-----------+
```

EXPLANATION : The query inside the IN operator returns list of user id of all users who rated movie “Notebook” at most 8. Outer query returns name of all users with id in the value list and “april” as birth month. month() function returns month value from date_of_birth column (ie. 4 for april).

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

```
SET @var:=(SELECT MAX(avg_rating)
           FROM (SELECT m.genre AS fav_gen, AVG(r.rating) AS
     avg_rating
                FROM movies m , reviews r, users u
                WHERE u.name LIKE "John Doe"
                AND r.user_id =u.id
                AND m.id=r.movie_id
                GROUP BY fav_gen)
           AS T2);

SELECT mo.genre, mo.name
FROM movies mo JOIN (SELECT fav_gen
                     FROM (SELECT m.genre AS fav_gen,
                          AVG(r.rating) AS avg_rating
                          FROM movies m , reviews r, users u
                          WHERE u.name LIKE "John Doe"
                          AND r.user_id =u.id
                          AND m.id=r.movie_id
                          GROUP BY fav_gen)
                     AS T1
                     WHERE avg_rating = @var) G
ON mo.genre=G. fav_gen
ORDER BY mo.genre, mo.name;
```

```
+--------+--------------+
| genre  | name         |
+--------+--------------+
| Comedy | Men in Black |
| Comedy | Notebook     |
+--------+--------------+
```

EXPLANATION : The first query calculates the maximum of average ratings of all movies watched by “John Doe” grouped by genre. The result is maximum rating by John Doe for a genre which is stored in temporary variable @var. Second query returns the genres with ratings equal to maximum rating(ie. @var) and is matched with movie name of that genre by join.

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

```
SET @var:=(SELECT MAX(count)
          FROM (SELECT COUNT(m.id) AS count
                FROM movies m, lead l, actors a
                WHERE m.id=l.movie_id
                AND l.actor_id=a.id
                AND a.gender="M"
                GROUP BY m.id)
          AS T);

SELECT m.id, COUNT(m.id) AS count
FROM movies m, lead l, actors a
WHERE m.id=l.movie_id
AND l.actor_id=a.id
AND a.gender="M"
GROUP BY m.id
HAVING count = @var
ORDER BY m.id;
```

```
+-----+-------+
| id  | count |
+-----+-------+
| 111 |     3 |
+-----+-------+
```

EXPLANATION - The query performs inner join operation on movies, lead, actors table. COUNT() aggregation function returns count of male actors grouped by movie id. The outer query returns the maximum count. This is stored in temporary variable @var. Second query returns ids of movies with count equal to @var(ie. maximum count).

Assumption : "gender" is CHAR(1) with domain values("M","F").

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.

```
SELECT name
FROM movies
WHERE id IN (SELECT movie_id
             FROM reviews
             WHERE rating >= (SELECT AVG(avg_rating)
                              FROM (SELECT AVG(rating) AS avg_rating
                                FROM reviews GROUP BY movie_id) AS
                           R))
AND genre LIKE "Comedy"
AND YEAR(release_date)<2006
ORDER BY name;
+--------------+
| name         |
+--------------+
| Men in Black |
+--------------+
```

EXPLANATION - Inner query computes the average of average ratings of all movies. It returns a list of movie id which are matched with genre=Comedy and release date before 2006.

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

```
SELECT AVG(rating) AS average, movie_id
FROM reviews
GROUP BY movie_id
HAVING average>9
AND movie_id IN (SELECT movie_id
                 FROM lead, actors
                 WHERE actor_id = id
                 AND name="Mark Clarkson")
ORDER BY average,movie_id;
```

```
+-----------+----------+
| average   | movie_id |
+-----------+----------+
|  9.500000 |      111 |
| 10.000000 |      114 |
+-----------+----------+
```

EXPLANATION - The query first finds average ratings of all movies using aggregate function AVG() grouped by their movie id then selecting only those whose average value is greater than 9 and one of actor is “Mark Clarkson” using having clause.

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.

```
SET @var = (SELECT MAX(count)
           FROM (SELECT COUNT(a.actor_id) as count
                FROM lead a JOIN lead b
                ON a.movie_id=b.movie_id
                AND a.actor_id!=b.actor_id
                GROUP BY a.actor_id,b.actor_id)
           AS T);

SELECT (SELECT name FROM actors WHERE a.actor_id=id) AS Actor1,
      (SELECT name FROM actors WHERE b.actor_id=id) AS Actor2,
      COUNT(a.actor_id) AS count
FROM lead a JOIN lead b
ON a.movie_id=b.movie_id
AND a.actor_id!=b.actor_id
GROUP BY a.actor_id,b.actor_id
HAVING count = @var;

+---------------+---------------+-------+
| Actor1        | Actor2        | count |
+---------------+---------------+-------+
| Orlando Bloom | Brad Garrett  |     2 |
| Brad Garrett  | Orlando Bloom |     2 |
+---------------+---------------+-------+
```

EXPLANATION - The query performs inner join on lead table with itself. It ensures that a row is not joined with itself by a.actor_id!=b.actor_id . The first query stores the maximum count of paired actors in @var variable. The second query find names of paired actors with count equals @var (ie. maximum count).