SOEN 363 - Project Phase 1
Section S

Group ID: 4
Group Name: Group B
Zain Khan - 40110693 - zmkhan025@gmail.com
Noor Hammodi - 40061760 - noor.hammodi@gmail.com
Nicolas Towa Kamgne - 40154685 - nicolastowakamgne@gmail.com
Ashiqur Rahman - 40096265 - ashlegitimate@outlook.com

Presented to Professor Essam Mansour
October 30, 2022

2 Loading the MovieLens Database

a)

The following queries are responsible for creating the tables:

```
CREATE TABLE movies(mid INT PRIMARY KEY, title VARCHAR, year INT, rating REAL,
num_ratings INT);
CREATE TABLE actors(mid INT REFERENCES movies(mid), name VARCHAR, cast_position
INT, PRIMARY KEY(mid, name));
CREATE TABLE genres(mid INT REFERENCES movies(mid), genre VARCHAR, PRIMARY
KEY(mid, genre));
CREATE TABLE tag_names(tid INT PRIMARY KEY, tag VARCHAR);
CREATE TABLE tags(mid INT REFERENCES movies(mid), tid INT REFERENCES
tag_names(tid), PRIMARY KEY(mid, tid));
```

The following query was used to populate each table:

```
copy <tableName>
from '<file directory of dataset>'
DELIMITER ' ';
```

E.g. for the 'genres' table:

```
copy genres
from 'C:\Users\user1\Desktop\phase1_dataset\genres.dat'
DELIMITER ' ';
```

3 Querying the MovieLens Database

a)

```
SELECT m.title
FROM movies m, actors a
WHERE m.mid = a.mid
AND a.name = 'Daniel Craig'
ORDER BY m.title;
```

b)

```
SELECT a.name
FROM movies m, actors a
```

```
WHERE m.title = 'The Dark Knight'
AND m.mid = a.mid
ORDER BY a.name;
```

c)

```
SELECT DISTINCT g.genre, COUNT(*) as N
FROM genres g
GROUP BY g.genre
HAVING COUNT(*) > 1000
ORDER BY N;
```

d)

```
SELECT m.title, m.year, m.rating
FROM movies m
ORDER BY m.year, m.rating DESC
```

e)

```
(SELECT m.title
FROM movies m, tag_names tn, tags t
WHERE m.mid = t.mid
AND t.tid = tn.tid
AND tn.tag LIKE '%good%')
INTERSECT
(SELECT m.title
FROM movies m, tag_names tn, tags t
WHERE m.mid = t.mid
AND t.tid = tn.tid
AND tn.tag LIKE '%bad%')
```

f)

f.i)

```
SELECT mid, title, year, num_ratings, rating
FROM movies
WHERE num_ratings IN (SELECT MAX(num_ratings)
                      FROM movies
                     );
```

f.ii) What is "include tuples that tie" ?

```
SELECT mid, title, year, num_ratings, rating
FROM movies
WHERE rating IN (SELECT MAX(rating)
                 FROM movies
```

```
                                    )
ORDER BY mid;
```
f.iii)

This would be an *intersect* between the queries f.i and f.ii - it would look like this:

```
(SELECT mid, title, year, num_ratings, rating
FROM movies
WHERE num_ratings IN (SELECT MAX(num_ratings)
                                              FROM movies
                                             ))
INTERSECT
(SELECT mid, title, year, num_ratings, rating
FROM movies
WHERE rating IN (SELECT MAX(rating)
                                    FROM movies
                                    )
ORDER BY mid);
```

- This returns nothing, as there is only 2 movie ids with the highest *num_ratings* (mid 4201 and mid 5312) which do not match the mid of the sole movie with the highest *rating* of 5 (mid 4311).

f.iv)

```
(SELECT mid, title, year, num_ratings, rating
FROM movies
WHERE rating IN (SELECT MIN(rating)
                                    FROM movies
                                    )
ORDER BY mid);
```
f.v)

This would be an *intersect* between the queries f.i and f.iv - it would look like this:

```
(SELECT mid, title, year, num_ratings, rating
FROM movies
WHERE num_ratings IN (SELECT MAX(num_ratings)
                                              FROM movies
                                             ))
INTERSECT
(SELECT mid, title, year, num_ratings, rating
FROM movies
WHERE rating IN (SELECT MIN(rating)
                                    FROM movies
```

```
                                        )
ORDER BY mid);
```

- This also returns nothing, as the mids with the highest *num_ratings* does not exist among the mids with the lowest *rating* of zero; in fact the highest *num_ratings* have a 3.8 star rating and appear to be duplicate values with unique mids of the 2007 *Pirates of the Caribbean: At World's End* movie.

f.vi)

In conclusion, based on the data obtained from the MovieLens database, our hypothesis/conjecture is false.

g)

--Here, we got rid of ratings of 0 as there were plenty and we believe this was a database error since there is no other rating remotely close to 0 (e.g. 0.1), and the next smallest value was 2.1, doing so makes it that the data from 2010 and 2011 will not be shown in the tables as there were not many and were rated 0. If we wanted the full data with 0 ratings we would simply remove rating <> 0 from the query.

```
(SELECT m.title, m.year, t.minRating as movieRating
FROM (
            SELECT year, MIN(rating) AS minRating
            FROM movies
            WHERE rating <> 0 AND year >= 2005 AND year <=2011
            GROUP BY year) t JOIN movies m ON t.year = m.year AND t.minRating
= m.rating
)
UNION
(SELECT m.title, m.year, t.maxRating as movieRating
FROM (
            SELECT year, MAX(rating) AS maxRating
            FROM movies
            WHERE rating <> 0 AND year >= 2005 AND year <=2011
            GROUP BY year) t JOIN movies m ON t.year = m.year AND
t.maxRating = m.rating
)
ORDER BY year, movieRating, title;
```

h)

h.i)

```
-- Creating highRatings view
CREATE VIEW highRatings AS
```

```
(SELECT DISTINCT a.name
FROM actors a, movies m
WHERE a.mid = m.mid AND m.rating >= 4);

-- Creating lowRatings view
CREATE VIEW lowRatings AS
(SELECT DISTINCT a.name
FROM actors a, movies m
WHERE a.mid = m.mid AND m.rating < 4);

-- Printing number of rows in highRatings and lowRatings views, respectively
SELECT COUNT(*) FROM highRatings;
SELECT COUNT(*) FROM lowRatings;
```

h.ii)

```
SELECT COUNT(*) FROM
        (
                (select * from highRatings)
                EXCEPT
                (select * from lowRatings)
        ) t;
```

h.iii)

```
-- Creating noFlop view for simplicity
CREATE VIEW noFlop AS
        (
                (select * from highRatings)
                EXCEPT
                (select * from lowRatings)
        );
-- Printing top 10 'no flop' actors who have most number of movies N that he/she played
in
SELECT nf.name, COUNT(a.mid) as moviesPlayedIn
FROM noFLop nf, actors a
WHERE nf.name = a.name
GROUP BY nf.name
ORDER BY moviesPlayedIn DESC
LIMIT 10;
```

i)

```
SELECT name, MIN(year), MAX(year)
FROM movies
```

```
JOIN actors on movies.mid = actors.mid
GROUP BY (actors.name)
HAVING (MAX(year)-MIN(year)) >= ALL (SELECT MAX(year)-MIN(year)
                                     FROM movies
                                     JOIN actors on movies.mid = actors.mid
                                     GROUP BY (actors.name));
```

j)

j.1)

```
-- Creating co_actors view
CREATE VIEW co_actors AS(
        SELECT DISTINCT a.name
        FROM actors a
        WHERE a.mid IN(SELECT a.mid
                        FROM actors a
                        WHERE a.name = 'Annette Nicole'
        )
        AND a.name <> 'Anette Nicole'
);

-- Printing number of rows in co_actors view
SELECT COUNT(*) FROM co_actors;
```

j.2)

```
-- Creating all_combinations view
CREATE VIEW all_combinations AS(
        SELECT ca.name, a.mid
        FROM co_actors ca, actors a
        WHERE a.name = 'Annette Nicole');

-- Printing number of rows in all_combinations view
SELECT COUNT(*) FROM all_combinations;
```

j.3)

```
-- Creating non_existent view
CREATE VIEW non_existent AS
(
        (       SELECT ac.name, ac.mid
                FROM all_combinations ac
        )
        EXCEPT
```

```
		(		SELECT a.name, a.mid
				FROM co_actors ca, actors a
				WHERE ca.name = a.name
		)
	);
```

-- Printing number of rows in non_existent view
SELECT COUNT(*) FROM non_existent;

j.4)

-- Our co_actors view already does not contain Annette Nicole, it only contains her co-actors

```
(	SELECT ca.name
	FROM co_actors ca)
EXCEPT
(	SELECT DISTINCT ne.name
	FROM non_existent ne)
```

k)

k.1)

```
(SELECT a.name as Actor, COUNT(DISTINCT a1.name) as co_actors
FROM actors a, actors a1
WHERE a.name = 'Tom Cruise'
AND a.mid = a1.mid
AND a1.mid IN(SELECT a.mid
			FROM actors a
			WHERE a.name='Tom Cruise'
			)
GROUP BY a.name
);
```

k.2)

```
CREATE VIEW Actors_Social  AS
(SELECT a.name as Actor, COUNT(DISTINCT a1.name)-1 as co_actors
FROM actors a, actors a1
WHERE a.mid = a1.mid
AND a1.mid IN(SELECT a.mid
			FROM actors a
			)
GROUP BY a.name
```

```
      HAVING COUNT(DISTINCT a1.name)-1 >= ALL (SELECT COUNT(DISTINCT
      a1.name)-1 as co_actors
      FROM actors a, actors a1
      WHERE a.mid = a1.mid
      AND a1.mid IN(SELECT a.mid
                          FROM actors a
                          )
      GROUP BY a.name)
      ORDER BY a.name ASC
      );
```

l)

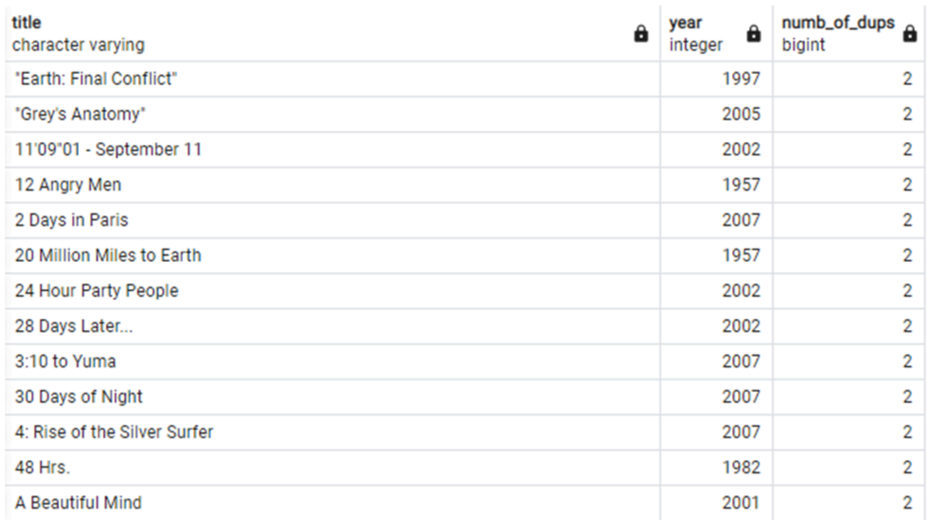
```
--Actors
CREATE VIEW commonActorsDenominator AS
  SELECT DISTINCT COUNT(DISTINCT a.name)*1.0 as denominator
  FROM actors a, movies m
  WHERE m.title = 'Mr. & Mrs. Smith'
  AND a.mid = m.mid;

CREATE VIEW fractionCommonActors AS
  SELECT a.mid, COUNT(a.mid)/cad.denominator AS fraction
  FROM actors a, movies m, commonActorsDenominator cad
  WHERE m.mid = a.mid
  AND m.title <> 'Mr. & Mrs. Smith'
  AND a.name IN (SELECT DISTINCT a.name
                      FROM actors a, movies m
                      WHERE m.title = 'Mr. & Mrs. Smith'
                      AND a.mid = m.mid)
  GROUP BY a.mid, cad.denominator
  ORDER BY COUNT(a.mid) DESC;
```

```
--Tags
CREATE VIEW commonTagsDenominator AS
        SELECT COUNT(t.tid)*1.0 as denominator
         FROM tags t
        WHERE t.mid IN (SELECT m.mid
                        FROM movies m
                        WHERE m.title = 'Mr. & Mrs. Smith');

CREATE VIEW fractionCommonTags AS
        SELECT COUNT (t.mid)/ctd.denominator AS fraction, t.mid
        FROM tags t, movies m, commonTagsDenominator ctd
        WHERE m.mid = t.mid
        AND m.title <> 'Mr. & Mrs. Smith'
        AND t.tid IN (SELECT t1.tid
                      FROM tags t1
                      WHERE t1.mid IN (SELECT m.mid
                                       FROM movies m
                                       WHERE m.title = 'Mr. & Mrs. Smith')
                      )
        GROUP BY t.mid, ctd.denominator
        ORDER BY COUNT (t.mid) DESC;



--Genres
CREATE VIEW commongenresdenominator AS
        SELECT COUNT( DISTINCT g.genre)*1.0 as denominator
        FROM genres g
        WHERE g.mid IN (SELECT DISTINCT m.mid
                        FROM movies m
                        WHERE m.title = 'Mr. & Mrs. Smith');

CREATE VIEW fractionCommonGenres AS
        SELECT COUNT (g.mid)/cad.denominator AS fraction, g.mid
        FROM genres g, movies m, commonactorsdenominator cad
        WHERE m.mid = g.mid
        AND m.title <> 'Mr. & Mrs. Smith'
        AND g.genre IN (SELECT DISTINCT g.genre
                        FROM genres g
                        WHERE g.mid IN (SELECT m.mid
                                                FROM movies m
                                                WHERE m.title = 'Mr. & Mrs. Smith')
                                    )
        GROUP BY g.mid, cad.denominator
        ORDER BY COUNT (g.mid) DESC;

--Age Gap
CREATE VIEW ageGap AS
SELECT DISTINCT ABS(m1.year - m2.year) AS gap, m1.mid
FROM movies m1, movies m2
WHERE m1.title <> 'Mr. & Mrs. Smith'
AND m2.year = (SELECT m.year
                        FROM movies m WHERE m.title =  'Mr. & Mrs. Smith'
                        LIMIT 1)
ORDER BY ABS(m1.year - m2.year) DESC;


--Rating Gap
CREATE VIEW ratingGap AS
        SELECT DISTINCT ABS(m1.rating - m2.rating) AS gap, m1.mid
        FROM movies m1, movies m2
        WHERE m1.title <> 'Mr. & Mrs. Smith'
        AND m1.rating IS NOT NULL
        AND m2.rating = (SELECT m.rating
                                FROM movies m
                                WHERE m.title = 'Mr. & Mrs. Smith'
                                LIMIT 1
                            )
        ORDER BY  ABS(m1.rating - m2.rating) DESC;
```

```
--sim(X,Y) Equation
CREATE VIEW toptenmovies AS
SELECT m.title, m.rating, 100*(fraction1.fraction + fraction2.fraction + fraction3.fraction +
gap1.gap + gap2.gap)/5 AS SimXY
FROM fractionCommonActors fraction1, fractionCommonTags fraction2,
fractionCommonGenres fraction3,
    ageGap gap1, ratingGap gap2, movies m
WHERE fraction1.mid= m.mid AND fraction2.mid= m.mid AND fraction3.mid= m.mid AND
gap1.mid= m.mid AND gap2.mid= m.mid
ORDER BY SimXY DESC LIMIT 10;

--Displaying the top ten movies
SELECT * from toptenmovies;
```

m)

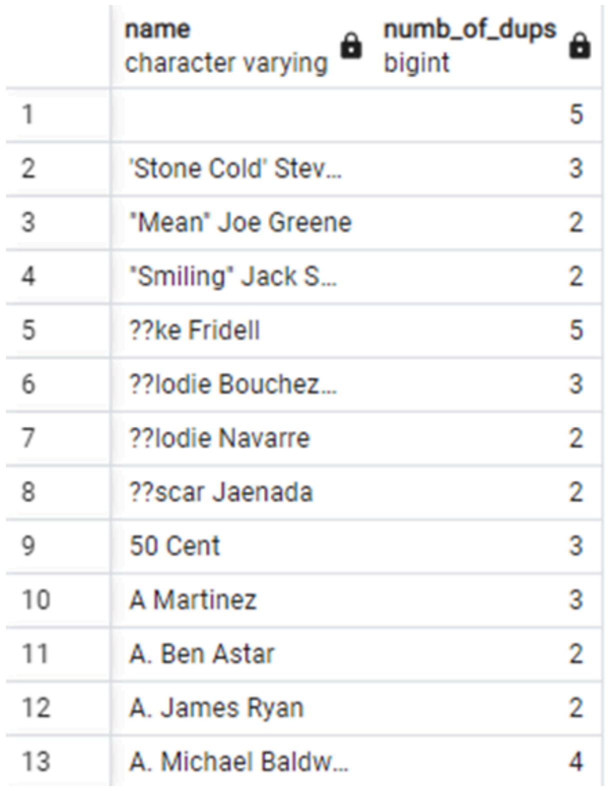
```
SELECT title, year, COUNT(*) as numb_of_dups
FROM movies
GROUP BY title, year
HAVING COUNT(*) > 1
ORDER by title ASC;
```



title character varying	year integer	numb_of_dups bigint
"Earth: Final Conflict"	1997	2
"Grey's Anatomy"	2005	2
11'09"01 - September 11	2002	2
12 Angry Men	1957	2
2 Days in Paris	2007	2
20 Million Miles to Earth	1957	2
24 Hour Party People	2002	2
28 Days Later...	2002	2
3:10 to Yuma	2007	2
30 Days of Night	2007	2
4: Rise of the Silver Surfer	2007	2
48 Hrs.	1982	2
A Beautiful Mind	2001	2

--this is a list of all duplicated movies

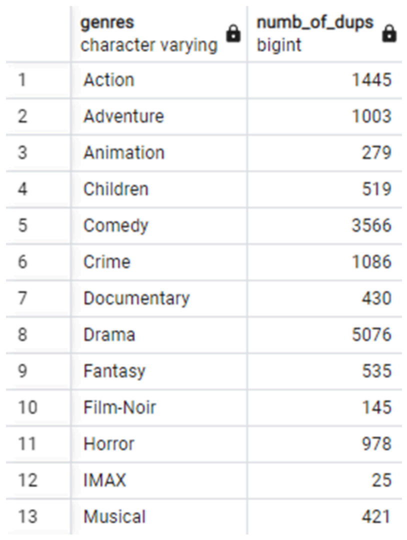
```
SELECT  name, COUNT(*) as numb_of_dups
FROM actors
GROUP BY  name
HAVING COUNT(*) > 1
ORDER by name ASC

CREATE VIEW actors_without_dups AS
SELECT DISTINCT name
FROM actors
GROUP BY mid, name
ORDER by name ASC;
```



	name character varying	numb_of_dups bigint
1		5
2	'Stone Cold' Stev...	3
3	"Mean" Joe Greene	2
4	"Smiling" Jack S...	2
5	??ke Fridell	5
6	??lodie Bouchez...	3
7	??lodie Navarre	2
8	??scar Jaenada	2
9	50 Cent	3
10	A Martinez	3
11	A. Ben Astar	2
12	A. James Ryan	2
13	A. Michael Baldw...	4

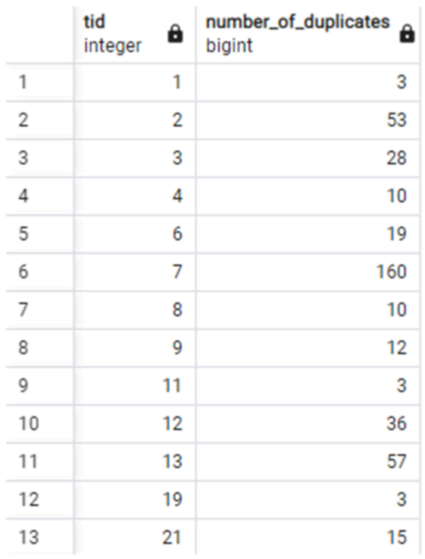
```
--this is a list of all duplicates actors

SELECT genres, COUNT(*) as numb_of_dups
FROM genres
GROUP BY genres
HAVING COUNT(*) > 1
ORDER by genres ASC
```

```
CREATE VIEW genres_without_dups AS
SELECT DISTINCT genres
FROM genres
GROUP BY mid, genres
ORDER by genres ASC;
```



	genres character varying	numb_of_dups bigint
1	Action	1445
2	Adventure	1003
3	Animation	279
4	Children	519
5	Comedy	3566
6	Crime	1086
7	Documentary	430
8	Drama	5076
9	Fantasy	535
10	Film-Noir	145
11	Horror	978
12	IMAX	25
13	Musical	421

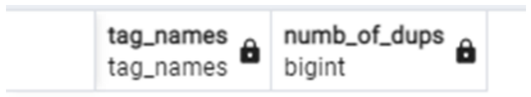
```
--this is a list of all duplicates genres


SELECT t.tid, COUNT(*) as numb_of_dups
FROM Tags t, Movies m
WHERE t.mid = m.mid
GROUP BY t.tid
HAVING COUNT(*) > 1
ORDER by tid ASC

CREATE VIEW tags_without_dups AS
SELECT DISTINCT tid
FROM tags
GROUP BY mid, tid
ORDER by tid ASC;
```



	tid integer	number_of_duplicates bigint
1	1	3
2	2	53
3	3	28
4	4	10
5	6	19
6	7	160
7	8	10
8	9	12
9	11	3
10	12	36
11	13	57
12	19	3
13	21	15

--this is a list of all movies ids that are duplicated in tags

```
SELECT tag_names, COUNT(*) as numb_of_dups
FROM tag_names
GROUP BY tag_names
HAVING COUNT(*) > 1
ORDER by tag_names ASC;

CREATE VIEW tag_names_without_dups AS
SELECT DISTINCT tag_names
FROM tag_names
GROUP BY tid
ORDER by tag_names ASC;
```



	tag_names tag_names	numb_of_dups bigint

--this is a list of all tagids and tag_names that are duplicated in tag_names

4 Performance

a) Since question h, j and l is one of the questions with the most execution time. We decided to create indexes to enhance their performance.

```
create index index_actors_name
ON actors (name);
create index index_actors_mid
ON actors (mid);
Create index index_movies_title
ON movies(title);
```

b)

	Without indexes	With indexes
For question h)	Query complete 00:00:00.240	Query complete 00:00:00.233
For j2) 3)	Query complete 00:00:00.125 Query complete 00:00:00.131	Query complete 00:00:00.045 Query complete 00:00:00.050
For l)	Query complete 00:00:02.868	Query complete 00:00:02.860

c)

```
CREATE MATERIALIZED VIEW Actors_Social1 AS
SELECT a.name as Actor, COUNT(DISTINCT a1.name)-1 as co_actors
FROM actors a, actors a1
WHERE a.mid = a1.mid
AND a1.mid IN(SELECT a.mid
FROM actors a
)
```

```
GROUP BY a.name
HAVING COUNT(DISTINCT a1.name)-1 = (
SELECT COUNT(DISTINCT a1.name)-1 as co_actors
FROM actors a, actors a1
WHERE a.mid = a1.mid
AND a1.mid IN(SELECT a.mid
FROM actors a
)
GROUP BY a.name
ORDER BY co_actors DESC
LIMIT 1
);
```

d)

```
--Actors
CREATE MATERIALIZED  VIEW commonActorsDenominator AS
   SELECT DISTINCT COUNT(DISTINCT a.name)*1.0 as denominator
   FROM actors a, movies m
   WHERE m.title = 'Mr. & Mrs. Smith'
   AND a.mid = m.mid;

CREATE MATERIALIZED  VIEW fractionCommonActors AS
   SELECT a.mid, COUNT(a.mid)/cad.denominator AS fraction
   FROM actors a, movies m, commonActorsDenominator cad
   WHERE m.mid = a.mid
   AND m.title <> 'Mr. & Mrs. Smith'
   AND a.name IN (SELECT DISTINCT a.name
                                    FROM actors a, movies m
                                    WHERE m.title = 'Mr. & Mrs. Smith'
                                    AND a.mid = m.mid)
   GROUP BY a.mid, cad.denominator
   ORDER BY COUNT(a.mid) DESC;
```

```
--Tags
CREATE MATERIALIZED  VIEW commonTagsDenominator AS
        SELECT COUNT(t.tid)*1.0 as denominator
         FROM tags t
        WHERE t.mid IN (SELECT m.mid
                        FROM movies m
                        WHERE m.title = 'Mr. & Mrs. Smith');

CREATE MATERIALIZED VIEW fractionCommonTags AS
        SELECT COUNT (t.mid)/ctd.denominator AS fraction, t.mid
        FROM tags t, movies m, commonTagsDenominator ctd
        WHERE m.mid = t.mid
        AND m.title <> 'Mr. & Mrs. Smith'
        AND t.tid IN (SELECT t1.tid
                      FROM tags t1
                      WHERE t1.mid IN (SELECT m.mid
                                       FROM movies m
                                       WHERE m.title = 'Mr. & Mrs. Smith')
                     )
        GROUP BY t.mid, ctd.denominator
        ORDER BY COUNT (t.mid) DESC;



--Genres
CREATE MATERIALIZED VIEW commongenresdenominator AS
        SELECT COUNT( DISTINCT g.genre)*1.0 as denominator
        FROM genres g
        WHERE g.mid IN (SELECT DISTINCT m.mid
                        FROM movies m
                        WHERE m.title = 'Mr. & Mrs. Smith');

CREATE MATERIALIZED VIEW fractionCommonGenres AS
        SELECT COUNT (g.mid)/cad.denominator AS fraction, g.mid
        FROM genres g, movies m, commonactorsdenominator cad
        WHERE m.mid = g.mid
        AND m.title <> 'Mr. & Mrs. Smith'
        AND g.genre IN (SELECT DISTINCT g.genre
                        FROM genres g
                        WHERE g.mid IN (SELECT m.mid
                                                FROM movies m
                                                WHERE m.title = 'Mr. & Mrs. Smith')
                                    )
        GROUP BY g.mid, cad.denominator
        ORDER BY COUNT (g.mid) DESC;

--Age Gap
CREATE MATERIALIZED VIEW ageGap AS
SELECT DISTINCT ABS(m1.year - m2.year) AS gap, m1.mid
FROM movies m1, movies m2
WHERE m1.title <> 'Mr. & Mrs. Smith'
AND m2.year = (SELECT m.year
                        FROM movies m WHERE m.title =  'Mr. & Mrs. Smith'
                        LIMIT 1)
ORDER BY ABS(m1.year - m2.year) DESC;


--Rating Gap
CREATE MATERIALIZED VIEW ratingGap AS
        SELECT DISTINCT ABS(m1.rating - m2.rating) AS gap, m1.mid
        FROM movies m1, movies m2
        WHERE m1.title <> 'Mr. & Mrs. Smith'
        AND m1.rating IS NOT NULL
        AND m2.rating = (SELECT m.rating
                                FROM movies m
                                WHERE m.title = 'Mr. & Mrs. Smith'
                                LIMIT 1
                            )
        ORDER BY  ABS(m1.rating - m2.rating) DESC;
```

```
--sim(X,Y) Equation
CREATE MATERIALIZED VIEW toptenmovies AS
SELECT m.title, m.rating, 100*(fraction1.fraction + fraction2.fraction + fraction3.fraction +
gap1.gap + gap2.gap)/5 AS SimXY
FROM fractionCommonActors fraction1, fractionCommonTags fraction2,
fractionCommonGenres fraction3,
    ageGap gap1, ratingGap gap2, movies m
WHERE fraction1.mid= m.mid AND fraction2.mid= m.mid AND fraction3.mid= m.mid AND
gap1.mid= m.mid AND gap2.mid= m.mid
ORDER BY SimXY DESC LIMIT 10;

--Displaying the top ten movies
SELECT * from toptenmovies;
```