Department of Computer Science & Software Engineering

SOEN 363 – Project Phase 1

Section S – Winter 2022

Data Systems for Software Engineers

Instructor: Dr. Essam Mansour

Team: Another Day, Another Data

Team Members:

Mohammad Ali Zahir – ID: 40077619 – Email: m.alizahir786@gmail.com

Dionisia Poulios - ID: 40131986 – Email: siap2468@gmail.com

Marita Brichan – ID: 40138194 – Email: maritabarikhan@hotmail.com

Sami Merhi – ID: 40136648 – Email: smerhi.sm@gmail.com

Submitted on: March 13, 2022

We certify that this submission is my original work and meets the Faculty's Expectations of Originality.

Table of Contents

1.0 Assumptions

For our project, we assumed that some data would not have more than a fix amount of characters. For example, we have title in movies, which we assume will not have more than 250 characters, hence we added the varchar(250). For the actors table, we had a naming convention problem with PG Admin, where it gave us a warning about name being a reserved SQL word, hence we changed the name attribute to actor_name. The rest of the data is shown below as follows.

Our outputs are the CSV outputs folder and the Queries, in addition to being here, are in the queries folder as well.

For convenience, we also changed the extension for the dataset files to .txt and date for the movies table in the “year-mo-dd” format.

2.0 Schema

CREATE TABLE Movies (
mid INT UNIQUE NOT NULL,
title VARCHAR(250),
year DATE,
rating REAL,
num_ratings INT,
PRIMARY KEY (mid)

```
);

CREATE TABLE Actors (
mid INT,
actor_name VARCHAR(50),
cast_position INT,
FOREIGN KEY (mid) REFERENCES Movies(mid),
PRIMARY KEY (mid, actor_name)
);

CREATE TABLE Genres (
mid INT,
genre VARCHAR(50),
FOREIGN KEY (mid) REFERENCES Movies(mid),
PRIMARY KEY (mid, genre)
);

CREATE TABLE Tag_Names (
tid INT UNIQUE NOT NULL,
tag VARCHAR(50),
PRIMARY KEY (tid)
);


CREATE TABLE Tags (
mid INT,
tid INT,
FOREIGN KEY (mid) REFERENCES Movies(mid),
```

```
FOREIGN KEY (tid) REFERENCES Tag_Names(tid),
PRIMARY KEY (mid, tid)
);
```

3.0 Queries

3a

```
SELECT M.title
FROM Movies M, Actors A
WHERE (M.mid=A.mid AND A.actor_name='Daniel Craig')
ORDER by M.title ASC
```

3b

```
SELECT A.actor_name
FROM Movies M, Actors A
WHERE (M.mid=A.mid AND M.title='The Dark Knight')
ORDER by A.actor_name ASC
```

3c

```
SELECT DISTINCT G.genre, COUNT(M.mid)
FROM Movies M, Genres G
WHERE M.mid=G.mid
```

```
GROUP BY G.genre
HAVING COUNT(M.mid) > 1000
ORDER BY COUNT(M.mid) ASC
```

3d

```
SELECT M.title, M.year, M.rating
FROM Movies M
ORDER BY M.year, M.rating DESC
```

3e

```
SELECT M.title
FROM Movies M, Tags T, Tag_Names K
WHERE (M.mid IN (SELECT M.mid
FROM Movies M, Tags T, Tag_Names K
    WHERE M.mid=T.mid and T.tid=K.tid and K.tag LIKE '%bad%'and K.tag NOT LIKE '%good%') and M.mid=T.mid and T.tid=K.tid and K.tag LIKE '%good%' and K.tag NOT LIKE '%bad%')
GROUP BY M.title HAVING COUNT(M.title) >=1;
```

3f (i)

```
select *
from movies
where movies.num_ratings IN
(select max(M.num_ratings)
from movies as M)
```

3f (ii)

```
select *
from movies
where movies.rating IN
(select max(M.rating)
from movies as M)
order by movies.mid asc
```

3f (iii)

```
create view highest_num_ratings
as
select *
from movies
where movies.num_ratings IN
(select max(M.num_ratings)
from movies as M);
```

```
create view highest_ratings
as
select *
from movies
where movies.rating IN
(select max(M.rating)
from movies as M);

select *
from highest_num_ratings
inner join highest_ratings on highest_num_ratings.mid=highest_ratings.mid;
```

The answer is hence no. Which is why we get no results here

3f (iv)

```
select *
from movies
where movies.rating IN
(select min(M.rating)
from movies as M
where M.rating is not null)
order by movies.mid asc
```

3f (v)

```
create view lowest_ratings
as
select *
from movies
where movies.rating IN
(select min(M.rating)
from movies as M
where M.rating is not null)
order by movies.mid asc;

select *
from highest_num_ratings
inner join lowest_ratings on highest_num_ratings.mid=lowest_ratings.mid;
```

Again here the answer is no, which again shows no results.

3f (vi)

In conclusion, it is not. There are no movies that have both the highest number of ratings and the highest ratings and there are also no movies that have both the highest number of ratings and the lowest ratings. We know this by using an inner join, or in other words revealing what is in the intersection of both sets. In both of these cases, the intersection in empty.

3g

```
SELECT M.year, M.title, M.rating
FROM Movies M
WHERE (M.year, M.rating) IN
  (
    SELECT L.year, MIN(L.rating)
    FROM Movies L
    WHERE L.year >= '2005-01-01' and L.year <= '2011-01-01' and L.num_ratings > 0
    GROUP BY L.year
  )
UNION
SELECT M.year, M.title, M.rating
FROM Movies M
WHERE (M.year, M.rating) IN
  (
    SELECT H.year, MAX(H.rating)
    FROM Movies H
    WHERE H.year >= '2005-01-01' and H.year <= '2011-01-01' and H.num_ratings > 0
    GROUP BY H.year
```

```
)
ORDER BY year ASC, rating ASC
```

3h (i)

```
CREATE VIEW high_ratings
AS
SELECT DISTINCT A.actor_name
FROM Actors A, Movies M
WHERE A.mid=M.mid and M.rating >= 4;

SELECT COUNT(H.actor_name)
FROM high_ratings H;

CREATE VIEW low_ratings
AS
SELECT DISTINCT A.actor_name
FROM Actors A, Movies M
WHERE A.mid=M.mid and M.rating < 4;

SELECT COUNT(L.actor_name)
FROM low_ratings L;
```

3h (ii)

```
SELECT COUNT (*)
   FROM high_ratings H
   LEFT JOIN low_ratings L ON H.actor_name=L.actor_name
   WHERE L.actor_name IS NULL;
```

3h (iii)

```
SELECT H.actor_name, COUNT(M.mid)
FROM high_ratings H, Movies M, Actors A
WHERE H.actor_name IN (
   SELECT H.actor_name
   FROM high_ratings H
   LEFT JOIN low_ratings L ON H.actor_name=L.actor_name
   WHERE L.actor_name IS NULL) and H.actor_name=A.actor_name and A.mid=M.mid
GROUP BY H.actor_name
ORDER BY COUNT(M.mid) DESC
LIMIT 10;
```

3i

```
SELECT A.actor_name
FROM Actors A, Actors A2, Movies M, Movies M2
WHERE A.mid=M.mid and A2.mid=M2.mid and M.mid <> M2.mid and
A.actor_name=A2.actor_name
```

```
ORDER BY age(M.year, M2.year) DESC
LIMIT 1;
```

3j1

```
CREATE OR REPLACE VIEW co_actors AS
   SELECT DISTINCT A1.actor_name
   FROM Actors A1
   WHERE A1.actor_name <> 'Annette Nicole'
   AND A1.mid IN
            (SELECT M.mid
               FROM Actors A2, Movies M
               WHERE A2.mid = M.mid
               AND A2.actor_name = 'Annette Nicole');

SELECT COUNT(*) FROM co_actors;
```

3j2

```
CREATE VIEW all_combinations AS
   SELECT C.actor_name, M.mid
   FROM co_actors C, Movies M, Actors A
```

```
   WHERE A.actor_name='Annette Nicole' and A.mid=M.mid;

SELECT COUNT(*) FROM all_combinations;
```

3j3

```
CREATE VIEW existent
AS
SELECT DISTINCT A1.actor_name, A1.mid
FROM Actors A1
        JOIN (
   SELECT A2.mid
   FROM Actors A2
   WHERE A2actor_name = 'Annette Nicole'
) AS Movies M1 ON A2.mid = M1.mid;

CREATE VIEW non_existent
AS
SELECT AC.actor_name, AC.mid
FROM all_combinations AC
WHERE AC.mid NOT IN (
   SELECT EX.mid
   FROM existent EX
   WHERE AC.actor_name = EX.actor_name
```

```
);

SELECT COUNT(*) FROM non_existent;
```

3j4

```
SELECT DISTINCT A.actor_name
FROM co_actors A
LEFT JOIN non_existent N ON A.actor_name=N.actor_name
WHERE N.actor_name IS NULL;
```

3k1

```
SELECT A2.actor_name, COUNT(DISTINCT A.actor_name)
FROM Actors A, Actors A2
WHERE A2.actor_name='Tom Cruise' and A2.mid=A.mid and A2.actor_name<>A.actor_name
GROUP BY A2.actor_name;
```

3k2

```
CREATE VIEW social_actor AS
```

```
SELECT A2.actor_name, COUNT(DISTINCT A.actor_name)
FROM Actors A, Actors A2
WHERE A2.mid=A.mid and A2.actor_name<>A.actor_name
GROUP BY A2.actor_name
ORDER BY COUNT(DISTINCT A.actor_name) DESC, A2.actor_name ASC
FETCH FIRST 1 ROWS WITH TIES;

SELECT * FROM social_actor;
```

3l

```
-- Actors that are in the movie 'Mr. & Mrs. Smith'
CREATE VIEW actors_in_movie AS
    SELECT DISTINCT A.actor_name as movie_actors
    FROM Movies M, Actors A
    WHERE M.title='Mr. & Mrs. Smith' and M.mid=A.mid;

-- Execution time: 37 ms


-- Counting the number of distinct common actors with other movies
CREATE VIEW common_actors AS
    SELECT M2.mid as movie_id, COUNT(DISTINCT A2.actor_name) as
number_of_common_actors
    FROM Movies M2, Actors A2
    WHERE actor_name IN (SELECT movie_actors FROM actors_in_movie)
                and M2.mid=A2.mid and M2.title<>'Mr. & Mrs. Smith'
    GROUP BY M2.mid;
```

```
-- Execution time: 67 ms


-- Calculation of the fraction of common actors
CREATE VIEW fraction_common_actors AS
    SELECT movie_id, ((max(number_of_common_actors) * 1.0) / (COUNT(movie_actors) * 1.0)) as fraction_actors
    FROM common_actors, actors_in_movie
    GROUP BY movie_id;

-- Execution time: 34 ms


-- Tags that are used for the movie 'Mr. & Mrs. Smith'
CREATE VIEW tags_for_movie AS
    SELECT DISTINCT T.tid as movie_tags
    FROM Movies M, Tags T
    WHERE M.title='Mr. & Mrs. Smith' and M.mid=T.mid;

-- Execution time: 38 ms


-- Counting the number of distinct common tags with other movies
CREATE VIEW common_tags AS
    SELECT M2.mid as movie_id, COUNT(DISTINCT T2.tid) as number_of_common_tags
    FROM Movies M2, Tags T2
    WHERE tid IN (
        SELECT movie_tags FROM tags_for_movie)
```

```
        and M2.mid = T2.mid and M2.title<>'Mr. & Mrs. Smith'
    GROUP BY M2.mid;

-- Execution time: 41 ms


-- Calculation of the fraction of common tags
CREATE VIEW fraction_common_tags AS
    SELECT movie_id, ((max(number_of_common_tags) * 1.0) / (COUNT(movie_tags) * 1.0)) as fraction_tags
    FROM common_tags, tags_for_movie
    GROUP BY movie_id;

-- Execution time: 33 ms


-- Genres for the movie 'Mr. & Mrs. Smith'
CREATE VIEW genres_for_movie AS
    SELECT DISTINCT G.genre as movie_genres
    FROM Movies M, Genres G
    WHERE M.title='Mr. & Mrs. Smith' and M.mid=G.mid;

-- Execution time: 34 ms


-- Counting the number of distinct common genres with other movies
CREATE VIEW common_genres AS
    SELECT M2.mid as movie_id, COUNT(DISTINCT G2.genre) as number_of_common_genres
```

```
    FROM Movies M2, Genres G2
    WHERE genre IN (
        SELECT movie_genres FROM genres_for_movie)
        and M2.mid = G2.mid and M2.title<>'Mr. & Mrs. Smith'
    GROUP BY M2.mid;

-- Execution time: 82 ms


-- Calculation of the fraction of common genres
CREATE VIEW fraction_common_genres AS
    SELECT movie_id, (MAX(number_of_common_genres) * 1.0 / COUNT(movie_genres) *
1.0) as fraction_genres
    FROM genres_for_movie, common_genres
    GROUP BY movie_id;

-- Execution time: 55 ms


-- Max age gap with movie 'Mr. & Mrs. Smith'
CREATE VIEW max_age_gap(max_gap) AS
    SELECT EXTRACT('year' FROM AGE(M.year, M2.year))
    FROM Movies M, Movies M2
    WHERE M.title='Mr. & Mrs. Smith' and M.mid<>M2.mid
    ORDER BY AGE(M.year, M2.year) DESC
    LIMIT 1;

-- Execution time: 53 ms
```

```
-- Release year of the movie 'Mr. & Mrs. Smith'
CREATE VIEW mr_mrs_year(release_year) AS
    SELECT DISTINCT EXTRACT('year' FROM M.year)
    FROM Movies M
    WHERE M.title='Mr. & Mrs. Smith';

-- Execution time: 36 ms


-- Normalized age gap between movies and 'Mr. & Mrs. Smith'
CREATE VIEW normalized_age AS
    SELECT M2.mid, (((SELECT max_gap FROM max_age_gap) - ABS((SELECT release_year
FROM mr_mrs_year) - EXTRACT('year' FROM M2.year)))
                    /(SELECT max_gap FROM max_age_gap)) as normalized_age_gap
    FROM Movies M, Movies M2
    WHERE M.title='Mr. & Mrs. Smith' and M.title<>M2.title;

-- Execution time: 63 ms


-- Max rating gap with the movie 'Mr. & Mrs. Smith'
CREATE VIEW max_rating_gap(rating_gap) AS
SELECT (M.rating - (SELECT MIN(M1.rating) FROM Movies M1)) as rating_gap
    FROM Movies M, Movies M2
    WHERE M.title='Mr. & Mrs. Smith' and M.mid<>M2.mid
    ORDER BY (ABS(M.rating - M2.rating)) DESC
    LIMIT 1;
```

```
-- Execution time: 31 ms



-- Rating of the movie 'Mr. & Mrs. Smith'
CREATE VIEW mr_mrs_rating(movie_rating) AS
   SELECT DISTINCT M.rating
   FROM Movies M
   WHERE M.title='Mr. & Mrs. Smith';

-- Execution time: 51 ms



-- Normalized rating gap between movies and 'Mr. & Mrs. Smith'
CREATE VIEW normalized_rating AS
   SELECT M2.mid, (((SELECT rating_gap FROM max_rating_gap) - ABS((SELECT
movie_rating FROM mr_mrs_rating) - (SELECT M2.rating)))
                  /(SELECT rating_gap FROM max_rating_gap)) as normalized_rating_gap
   FROM Movies M, Movies M2
   WHERE M.title='Mr. & Mrs. Smith' and M.mid<>M2.mid;

-- Query that calculates the similarity %, displays it with the title of the movie and its rating, and
shows the top 10 in descending order
SELECT M.title, M.rating, ROUND(CAST((((MAX(FA.fraction_actors + FT.fraction_tags +
FG.fraction_genres
                        + NA.normalized_age_gap + NR.normalized_rating_gap))/5)*100) AS
DECIMAL), 2) AS recommendation
FROM normalized_age NA
INNER JOIN fraction_common_actors FA ON FA.movie_id=NA.mid
INNER JOIN fraction_common_tags FT ON FT.movie_id=FA.movie_id
INNER JOIN fraction_common_genres FG ON FG.movie_id=FT.movie_id
```

```
INNER JOIN normalized_rating NR ON NR.mid=FG.movie_id
INNER JOIN Movies M on NA.mid = M.mid
GROUP BY M.mid
ORDER BY recommendation DESC
LIMIT 10;

-- Execution time: 61 ms
```

3m

```
--if Movies has duplicates
SELECT m.title, m.year, m.rating, m.num_ratings, COUNT(*)
FROM Movies m
GROUP BY m.title, m.year, m.rating, m.num_ratings
HAVING COUNT(*) > 1;
```

	title character varying (250)	year date	rating real	num_ratings integer	count bigint
1	Revolutionary Road	2008-01-01	3.5	46044	2
2	Don't Be a Menace to South Central While Drinking Your Juice in the Hood	1996-01-01	0	0	2
3	Georgia Rule	2007-01-01	3	64056	2
4	Q	1982-01-01	0	0	2
5	Moulin Rouge!	2001-01-01	3.7	110817	2
6	East of Eden	1955-01-01	4	5957	2
7	Alien³	1992-01-01	3.1	30526	2
8	The Assassination of Jesse James by the Coward Robert Ford	2007-01-01	3.7	57534	3
9	Albino Alligator	1996-01-01	3	1198	2
10	Kiss of Death	1995-01-01	2.9	2125	2
11	The Bridges of Madison County	1995-01-01	3.6	11573	2
12	King Solomon's Mines	1950-01-01	0	0	3

Count indicates how many times the movie has been repeated

```
--View for Movies without any duplicates
CREATE VIEW movies_no_duplicates
AS
SELECT *
FROM Movies m
WHERE m.mid IN (SELECT min(m.mid)
   FROM Movies m
   GROUP BY m.title, m.year, m.rating, m.num_ratings);
```

```
--if Actors has duplicates
SELECT a.actor_name, a.cast_position, COUNT(*)
FROM Actors a, Movies m, movies_no_duplicates nm
WHERE a.mid = m.mid AND m.title = nm.title AND m.year = nm.year AND m.rating =
nm.rating AND m.num_ratings = nm.rating
GROUP BY a.actor_name, a.cast_position, nm.title, nm.year, nm.rating, nm.num_ratings
HAVING COUNT(*) > 1;
```

	actor_name character varying (50)	cast_position integer	count bigint
1	A.W. Sweatt	11	2
2	Aaron Vaughn	48	2
3	Abel Woolrich	1	2
4	Acid Drinkers	1	2
5	Adam Bolton	6	2
6	Adam Brock	12	2
7	Adolphe Menjou	3	2
8	Adrian Rosley	42	2
9	Adrianne Richards	7	3
10	Adrien Saint-Jore	16	2
11	Aimee Deshayes	14	2
12	Akira Ishida	1	2
13	Akira Kurosawa	9	2

Count indicates how many times the actor has appeared

```
--View for Actors without any duplicates
CREATE VIEW actors_no_duplicates
AS
SELECT a.mid, a.actor_name, a.cast_position
FROM Actors a
WHERE a.mid IN (SELECT min(a.mid)
FROM Actors a, movies_no_duplicates nm, Movies m
WHERE a.mid = m.mid AND m.title = nm.title AND m.year = nm.year AND m.rating = nm.rating AND m.num_ratings = nm.rating
GROUP BY a.actor_name, a.cast_position, nm.title, nm.year, nm.rating, nm.num_ratings)
```

```
AND a.actor_name IN (SELECT a.actor_name
FROM Actors a, movies_no_duplicates nm, Movies m
WHERE a.mid = m.mid AND m.title = nm.title AND m.year = nm.year AND m.rating = nm.rating AND m.num_ratings = nm.rating
GROUP BY a.actor_name, a.cast_position, nm.title, nm.year, nm.rating, nm.num_ratings);


--if Genres has duplicates
SELECT g.genre, COUNT(*)
FROM Genres g, Movies m, movies_no_duplicates nm
WHERE g.mid = m.mid AND m.title = nm.title AND m.year = nm.year AND m.rating = nm.rating AND m.num_ratings = nm.rating
GROUP BY g.genre, nm.title, nm.year, nm.rating, nm.num_ratings
HAVING COUNT(*) > 1;
```

	genre character varying (50)	count bigint
1	Action	2
2	Action	2
3	Action	2
4	Action	2
5	Action	2
6	Action	2
7	Action	2
8	Adventure	2
9	Adventure	2
10	Adventure	3
11	Adventure	2
12	Adventure	2
13	Adventure	2
14	Adventure	2

We can see the repeated values here

```
--View for Genres without any duplicates
CREATE VIEW genres_no_duplicates
AS
SELECT g.mid, g.genre
FROM Genres g
WHERE g.mid IN (SELECT min(g1.mid)
FROM Genres g1, movies_no_duplicates nm, Movies m
WHERE g1.mid = m.mid AND m.title = nm.title AND m.year = nm.year AND m.rating = nm.rating AND m.num_ratings = nm.rating
GROUP BY g1.genre, nm.title, nm.year, nm.rating, nm.num_ratings)
AND g.genre IN (SELECT g1.genre
FROM Genres g1, movies_no_duplicates nm, Movies m
WHERE g1.mid = m.mid AND m.title = nm.title AND m.year = nm.year AND m.rating = nm.rating AND m.num_ratings = nm.rating
GROUP BY g1.genre, nm.title, nm.year, nm.rating, nm.num_ratings);
```

```
--if Tag_names has duplicates
SELECT tn.tag, tn.tid, COUNT(*)
FROM Tag_names tn
GROUP BY tn.tag, tn.tid
HAVING COUNT(*) > 1;
```

tag character varying (50)	tid [PK] integer	count bigint

No duplicates for tag names

```
--View for Tag_names without any duplicates
CREATE VIEW tag_names_no_duplicates
AS
SELECT *
FROM Tag_names tn
WHERE tn.tid IN (SELECT min(tn1.tid)
   FROM Tag_names tn1
   GROUP BY tn1.tid, tn1.tag);

--if Tags has duplicates
--assuming that there are no duplicates in tag_names which was actually proven by the previous query for our current dataset
SELECT t.tid, COUNT(*)
FROM Tags t, Tag_names tn, Movies m, movies_no_duplicates nm
WHERE t.mid = m.mid AND m.title = nm.title AND m.year = nm.year AND m.rating = nm.rating AND m.num_ratings = nm.rating
AND t.tid = tn.tid
GROUP BY t.tid, nm.title, nm.year, nm.rating, nm.num_ratings
HAVING COUNT(*) > 1;
```

	tid integer	count bigint
1	12	2
2	12	2
3	38	2
4	56	2
5	125	2
6	136	2
7	186	2
8	203	2
9	583	2

Duplicates here again

--View for Tags without any duplicates

CREATE VIEW Tags_no_duplicates

AS

SELECT t.mid, t.tid

FROM Tags t

WHERE t.mid IN (SELECT min(t1.mid)

FROM Tags t1, movies_no_duplicates nm, Movies m

WHERE t1.mid = m.mid AND m.title = nm.title AND m.year = nm.year AND m.rating = nm.rating AND m.num_ratings = nm.rating

GROUP BY t1.tid, nm.title, nm.year, nm.rating, nm.num_ratings) AND t.tid IN (SELECT min(t1.tid)

FROM Tags t1, movies_no_duplicates nm, Movies m

WHERE t1.mid = m.mid AND m.title = nm.title AND m.year = nm.year AND m.rating = nm.rating AND m.num_ratings = nm.rating

GROUP BY t1.tid, nm.title, nm.year, nm.rating, nm.num_ratings);

4a1

CREATE MATERIALIZED VIEW IF NOT EXISTS view_movies AS SELECT * FROM Movies;

CREATE INDEX movie_index ON view_movies(mid);

CREATE MATERIALIZED VIEW IF NOT EXISTS view_actors AS SELECT * FROM Actors;

CREATE INDEX actor_index ON view_actors(mid, actor_name);

CREATE MATERIALIZED VIEW IF NOT EXISTS view_genres AS SELECT * FROM Genres;

CREATE INDEX genre_index ON view_genres(mid, genre);

CREATE MATERIALIZED VIEW IF NOT EXISTS view_tag_names AS SELECT * FROM Tag_Names;

CREATE INDEX tag_name_index ON view_tag_names(tid);

CREATE MATERIALIZED VIEW IF NOT EXISTS view_tags AS SELECT * FROM Tags;

CREATE INDEX tag_index ON view_tags(mid, tid);

--- QUERY EXECUTION TIMES ---

-- view_movies: 56 ms

-- movie_index: 52 ms

-- view_actors : 397 ms

-- actor_index : 558 ms

-- view_genres : 54 ms

-- genre_index : 66 ms

-- view_tag_names : 65 ms

-- tag_name_index : 55 ms

-- view_tags : 80 ms

-- tag_index : 70 ms

4a2

```
CREATE MATERIALIZED VIEW IF NOT EXISTS view_social_actor AS
SELECT A2.actor_name, COUNT(DISTINCT A.actor_name)
FROM Actors A, Actors A2
WHERE A2.mid=A.mid and A2.actor_name<>A.actor_name
GROUP BY A2.actor_name
ORDER BY COUNT(DISTINCT A.actor_name) DESC, A2.actor_name ASC
FETCH FIRST 1 ROWS WITH TIES;

SELECT * FROM view_social_actor;
```

--- QUERY EXECUTION TIME ---

-- view_social_actor: 12118 ms

-- select execution time : 30 ms

4b1

```
EXPLAIN ANALYZE SELECT * FROM view_movies;
EXPLAIN ANALYZE SELECT * FROM Movies;
```

-- With materialized view index - Execution time : 0.710ms

-- with no index - Execution time :0.720ms

EXPLAIN ANALYZE SELECT * FROM view_actors;

EXPLAIN ANALYZE SELECT * FROM Actors;

-- With materialized view index - Execution time : 15.501ms

-- with no index - Execution time :15.774ms

EXPLAIN ANALYZE SELECT * FROM view_genres;

EXPLAIN ANALYZE SELECT * FROM Genres;

-- With materialized view index - Execution time : 1.454ms

-- with no index - Execution time :1.538ms

EXPLAIN ANALYZE SELECT * FROM view_tag_names;

EXPLAIN ANALYZE SELECT * FROM Tag_Names;

-- With materialized view index - Execution time : 0.916ms

-- with no index - Execution time :0.941ms

EXPLAIN ANALYZE SELECT * FROM view_tags;

EXPLAIN ANALYZE SELECT * FROM Tags;

-- With materialized view index - Execution time : 3.696ms

-- with no index - Execution time :4.069ms

4b2

```
-- Actors that are in the movie 'Mr. & Mrs. Smith'
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_actors_in_movie AS
   SELECT DISTINCT A.actor_name as movie_actors
   FROM Movies M, Actors A
   WHERE M.title='Mr. & Mrs. Smith' and M.mid=A.mid;

-- Execution time: 26 ms

-- Counting the number of distinct common actors with other movies
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_common_actors AS
   SELECT M2.mid as movie_id, COUNT(DISTINCT A2.actor_name) as
number_of_common_actors
   FROM Movies M2, Actors A2
   WHERE actor_name IN (SELECT movie_actors FROM mw_actors_in_movie)
            and M2.mid=A2.mid and M2.title<>'Mr. & Mrs. Smith'
   GROUP BY M2.mid;

-- Execution time: 53 ms

-- Calculation of the fraction of common actors
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_fraction_common_actors AS
   SELECT movie_id, ((max(number_of_common_actors) * 1.0) / (COUNT(movie_actors) *
1.0)) as fraction_actors
   FROM mw_common_actors, mw_actors_in_movie
   GROUP BY movie_id;

-- Execution time: 31 ms
```

```
-- Tags that are used for the movie 'Mr. & Mrs. Smith'
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_tags_for_movie AS
    SELECT DISTINCT T.tid as movie_tags
    FROM Movies M, Tags T
    WHERE M.title='Mr. & Mrs. Smith' and M.mid=T.mid;

-- Execution time: 31 ms

-- Counting the number of distinct common tags with other movies
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_common_tags AS
    SELECT M2.mid as movie_id, COUNT(DISTINCT T2.tid) as number_of_common_tags
    FROM Movies M2, Tags T2
    WHERE tid IN (
        SELECT movie_tags FROM mw_tags_for_movie)
        and M2.mid = T2.mid and M2.title<>'Mr. & Mrs. Smith'
    GROUP BY M2.mid;

-- Execution time: 32 ms

-- Calculation of the fraction of common tags
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_fraction_common_tags AS
    SELECT movie_id, ((max(number_of_common_tags) * 1.0) / (COUNT(movie_tags) * 1.0))
as fraction_tags
    FROM mw_common_tags, mw_tags_for_movie
    GROUP BY movie_id;

-- Execution time: 31 ms
```

```
-- Genres for the movie 'Mr. & Mrs. Smith'
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_genres_for_movie AS
    SELECT DISTINCT G.genre as movie_genres
    FROM Movies M, Genres G
    WHERE M.title='Mr. & Mrs. Smith' and M.mid=G.mid;

-- Execution time: 28 ms

-- Counting the number of distinct common genres with other movies
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_common_genres AS
    SELECT M2.mid as movie_id, COUNT(DISTINCT G2.genre) as
number_of_common_genres
    FROM Movies M2, Genres G2
    WHERE genre IN (
        SELECT movie_genres FROM mw_genres_for_movie)
        and M2.mid = G2.mid and M2.title<>'Mr. & Mrs. Smith'
    GROUP BY M2.mid;

-- Execution time: 68 ms

-- Calculation of the fraction of common genres
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_fraction_common_genres AS
    SELECT movie_id, (MAX(number_of_common_genres) * 1.0 / COUNT(movie_genres) *
1.0) as fraction_genres
    FROM mw_genres_for_movie, mw_common_genres
    GROUP BY movie_id;

-- Execution time: 43 ms
```

```
-- Max age gap with movie 'Mr. & Mrs. Smith'
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_max_age_gap(max_gap) AS
    SELECT EXTRACT('year' FROM AGE(M.year, M2.year))
    FROM Movies M, Movies M2
    WHERE M.title='Mr. & Mrs. Smith' and M.mid<>M2.mid
    ORDER BY AGE(M.year, M2.year) DESC
    LIMIT 1;

-- Execution time: 46 ms

-- Release year of the movie 'Mr. & Mrs. Smith'
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_mr_mrs_year(release_year) AS
    SELECT DISTINCT EXTRACT('year' FROM M.year)
    FROM Movies M
    WHERE M.title='Mr. & Mrs. Smith';

-- Execution time: 29 ms

-- Normalized age gap between movies and 'Mr. & Mrs. Smith'
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_normalized_age AS
    SELECT M2.mid, (((SELECT max_gap FROM mw_max_age_gap) - ABS((SELECT
release_year FROM mw_mr_mrs_year) - EXTRACT('year' FROM M2.year)))
                    /(SELECT max_gap FROM mw_max_age_gap)) as normalized_age_gap
    FROM Movies M, Movies M2
    WHERE M.title='Mr. & Mrs. Smith' and M.title<>M2.title;

-- Execution time: 59 ms
```

```
-- Max rating gap with the movie 'Mr. & Mrs. Smith'
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_max_rating_gap(rating_gap) AS
SELECT (M.rating - (SELECT MIN(M1.rating) FROM Movies M1)) as rating_gap
   FROM Movies M, Movies M2
   WHERE M.title='Mr. & Mrs. Smith' and M.mid<>M2.mid
   ORDER BY (ABS(M.rating - M2.rating)) DESC
   LIMIT 1;

-- Execution time: 33 ms


-- Rating of the movie 'Mr. & Mrs. Smith'
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_mr_mrs_rating(movie_rating) AS
   SELECT DISTINCT M.rating
   FROM Movies M
   WHERE M.title='Mr. & Mrs. Smith';

-- Execution time: 29 ms


-- Normalized rating gap between movies and 'Mr. & Mrs. Smith'
CREATE MATERIALIZED VIEW IF NOT EXISTS mw_normalized_rating AS
   SELECT M2.mid, (((SELECT rating_gap FROM mw_max_rating_gap) - ABS((SELECT
movie_rating FROM mw_mr_mrs_rating) - (SELECT M2.rating)))
                  /(SELECT rating_gap FROM mw_max_rating_gap)) as normalized_rating_gap
   FROM Movies M, Movies M2
   WHERE M.title='Mr. & Mrs. Smith' and M.mid<>M2.mid;

-- Execution time: 45 ms
```

```
-- Query that calculates the similarity %, displays it with the title of the movie and its rating, and shows the top 10 in descending order

SELECT M.title, M.rating, ROUND(CAST((((MAX(FA.fraction_actors + FT.fraction_tags + FG.fraction_genres

                + NA.normalized_age_gap + NR.normalized_rating_gap))/5)*100) AS DECIMAL), 2) AS recommendation

FROM mw_normalized_age NA

INNER JOIN mw_fraction_common_actors FA ON FA.movie_id=NA.mid

INNER JOIN mw_fraction_common_tags FT ON FT.movie_id=FA.movie_id

INNER JOIN mw_fraction_common_genres FG ON FG.movie_id=FT.movie_id

INNER JOIN mw_normalized_rating NR ON NR.mid=FG.movie_id

INNER JOIN Movies M on NA.mid = M.mid

GROUP BY M.mid

ORDER BY recommendation DESC

LIMIT 10;


-- Execution time: 44 ms
```