Concordia University

# Database Systems

# SOEN 363 – Fall 2022

# Project - Phase 1

Marwa Khalid (40155098) – khalidmarwa786@gmail.com

William Wells (40111253) - williamwells2013@hotmail.com

Lucas Catchlove (27145640) - lucascatchlove@gmail.com

Shawn Gorman (40157925) - gorman.shawn23@hotmail.com

Professor: Essam Mansour

Group ID: 10

Out: September 29, 2022

Due: October 30, 2022

# 1 Schema

```
CREATE TABLE movies (
   mid integer,
   title varchar,
   year integer,
   rating real,
   num_ratings integer,
   PRIMARY KEY (mid));

CREATE TABLE actors (
   mid integer,
   name varchar,
   cast_position integer,
   FOREIGN KEY (mid)
            REFERENCES movies);

CREATE TABLE genres (
   mid integer,
   genre varchar,
   FOREIGN KEY (mid)
            REFERENCES movies);

CREATE TABLE tags (
   mid integer,
   tid integer,
   FOREIGN KEY (mid)
            REFERENCES movies,
   FOREIGN KEY (tid)
            REFERENCES tag_names);

CREATE TABLE tag_names (
   tid integer,
   tag varchar,
   PRIMARY KEY (tid));
```

# 2 Querying the MovieLens Database

**A)**

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

**B)**

```
SELECT name FROM actors, movies WHERE
      title='The Dark Knight' AND
      actors.mid=movies.mid
ORDER BY name ASC;
```

**C)**

```
select genre, count(distinct title) as N
from genres g,
    movies m
where g.mid = m.mid
group by genre
having count(distinct title) > 1000
order by N;
```

**D)**

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

**E)**

```
SELECT M.title
FROM movies M
WHERE M.mid IN
(SELECT T.mid
FROM tags T
WHERE T.tid IN(SELECT TN.tid
                        FROM tag_names TN
                        WHERE TN.tag LIKE '%good%'
```

```
AND TN.tag LIKE '%bad%'))
```

**F)**

**Part I:**

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

**Part II:**

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

**Part III:**

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

INTERSECT
(SELECT *
FROM movies
where movies.rating IN
(select max(M.rating)
from movies as M);
```

Answer: No because it shows no results.

**Part 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
```

**Part 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;
```

Answer: No because it shows no results.

**Part VI:**

Our hypothesis is not true because there are no movies that have both the highest number of ratings and highest ratings. Equally, there are no movies which have the highest number of ratings and lowest ratings. Additionally, because iii and v did not return any results it means our hypothesis was wrong.

**G)**

```
SELECT M.year, M.title, M.rating
```

```
FROM movies M
WHERE (M.year, M.rating) IN
   (
      SELECT M1.year, MIN(M1.rating)
      FROM Movies M1
      WHERE M1.year >= '2005' and M1.year <= '2011' and M1.num_ratings > 0
      GROUP BY M1.year
   )
UNION
SELECT M.year, M.title, M.rating
FROM movies M
WHERE (M.year, M.rating) IN
   (
      SELECT M2.year, MAX(M2.rating)
      FROM movies M2
      WHERE M2.year >= '2005' and M2.year <= '2011' and M2.num_ratings > 0
      GROUP BY M2.year
   )
ORDER BY year ASC, rating ASC
```

**H)**

**Part I:**

```
create view high_ratings as
select distinct name
from actors a,
    movies m
where a.mid = m.mid
 and rating >= 4
order by name;

create view low_ratings as
select distinct name
from actors a,
    movies m
```

```
where a.mid = m.mid
  and rating < 4
order by name;

select count(*) low_ratings_count from low_ratings;
select count(high_ratings) high_ratings_count from high_ratings;
```

**Part II:**

```
select count(name) no_flop_count
from high_ratings
where name not in (select name from low_ratings);
```

**Part III:**

```
select a.name, count(m.title)
from actors a,
     movies m
where a.mid = m.mid
  and a.name in (select name from high_ratings)
  and a.name not in (select name from low_ratings)
group by a.name
order by count(m.title) desc
limit 10;
```

**I)**

```
CREATE VIEW first_movie_years AS
   SELECT name, min(year) as year FROM actors, movies WHERE
      actors.mid=movies.mid
   group by name;

CREATE VIEW last_movie_years AS
   SELECT name, max(year) as year FROM actors, movies WHERE
      actors.mid=movies.mid
   group by name;
```

```
CREATE VIEW longevity AS
   SELECT first_movie_years.name as name, (last_movie_years.year -
first_movie_years.year) as year FROM first_movie_years, last_movie_years WHERE
      first_movie_years.name=last_movie_years.name
   order by year DESC;

SELECT name FROM longevity LIMIT 1;
```

**J)**

**Part 1:**

```
create view co_actors as
select distinct a1.name
from actors a1,
    actors a2,
    movies m
where a1.mid = m.mid
 and a2.mid = m.mid
 and a2.name = 'Annette Nicole';

select count(*) co_actors_count
from co_actors;
```

**Part 2:**

```
create view all_combinations as
select ca.name, m.mid
from co_actors ca
       cross join (select m.mid
                   from movies m,
                       actors a
                   where m.mid = a.mid
                     and a.name = 'Annette Nicole') as m
where ca.name <> 'Annette Nicole';

select count(*) all_combinations_count
```

```
from all_combinations;
```

**Part 3:**

```
create view non_existent as
select name, mid
from all_combinations
where (name, mid) not in (select name, mid from actors);

select count(*) non_existent_count
from non_existent;
```

**Part 4:**

```
select name
from co_actors
where name not in (select distinct name from non_existent)
and name <> 'Annette Nicole';
```

**K)**

```
/* Count Tom Cruise's co-actors */
CREATE VIEW tom_movies AS
   SELECT DISTINCT name, actors.mid as mid FROM movies, actors WHERE
      name='Tom Cruise';

SELECT tom_movies.name as name, COUNT(DISTINCT actors)FROM tom_movies,
actors WHERE
         tom_movies.mid=actors.mid AND
         actors.name!='Tom Cruise'
group by tom_movies.name;

/* Find most social actor */
CREATE VIEW num_of_co_actors AS
   SELECT actors.name as name, COUNT(DISTINCT actors1) as co_actors FROM
actors
      INNER JOIN actors as actors1 on actors.mid = actors1.mid WHERE
         actors.name != actors1.name
```

```
    group by actors.name
    ORDER BY co_actors DESC;

SELECT * FROM num_of_co_actors WHERE
    co_actors=(SELECT MAX(num_of_co_actors1.co_actors)
            FROM num_of_co_actors as num_of_co_actors1);
```

**L)**

```
CREATE VIEW target_actors_names AS
    SELECT DISTINCT A.name as movie_actors
    FROM Movies M, Actors A
    WHERE M.title='Mr. & Mrs. Smith' and M.mid=A.mid;

CREATE VIEW common_actors AS
    SELECT M2.mid as movie_id, COUNT(DISTINCT A2.name) as
number_of_common_actors
    FROM Movies M2, Actors A2
    WHERE name IN (SELECT movie_actors FROM target_actors_names)
                and M2.mid=A2.mid and M2.title<>'Mr. & Mrs. Smith'
    GROUP BY M2.mid;

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, target_actors_names
    GROUP BY movie_id;

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;

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;

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;

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;

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;

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;
```

```
CREATE VIEW target_id(mid) AS
   SELECT MAX(movies.mid)
   FROM movies
   WHERE movies.title = 'Mr. & Mrs. Smith';

CREATE VIEW target_age(age) AS
   SELECT year
   FROM movies
   WHERE mid = (SELECT mid FROM target_id);

CREATE VIEW max_age_gap(max_gap) AS
   SELECT MAX(ABS(year - (SELECT target_age.age FROM target_age)))
   FROM movies m1
   WHERE m1.mid <> (SELECT mid FROM target_id);

CREATE VIEW mr_mrs_year(release_year) AS
   SELECT DISTINCT M.year
   FROM Movies M
   WHERE M.title='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) - 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;

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;
```

```
CREATE VIEW mr_mrs_rating(movie_rating) AS
   SELECT DISTINCT M.rating
   FROM Movies M
   WHERE M.title='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;

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
FULL OUTER JOIN fraction_common_actors FA ON FA.movie_id=NA.mid
FULL OUTER JOIN fraction_common_tags FT ON FT.movie_id=FA.movie_id
FULL OUTER JOIN fraction_common_genres FG ON FG.movie_id=FT.movie_id
FULL OUTER JOIN normalized_rating NR ON NR.mid=FG.movie_id
FULL OUTER JOIN Movies M on NA.mid = M.mid

GROUP BY M.mid
ORDER BY recommendation DESC
LIMIT 10;
```

**runtime: 174 ms**

**M)**

```
/* Example to check 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;

/*Example create a view */
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);
```

# 4 Performance

**Q3 - b**

```
CREATE INDEX id_mid_actors
ON actors (mid);
```

**Q3 - c**

In this case, the query relies on the columns "genre", "mid", and "title". There is no reason to create indexes as Postgres automatically creates indexes based on columns included in the primary key of a given table. In other words, the optimizations are already in place. In the case of the column "title" (not included in the primary key of the table "movies"), no index is required as the query is merely counting instances of "title", and not searching for a specific one.

**Q3 - h**

```
create index ratings_idx on movies (
   rating ASC NULLS FIRST
   );
```

An index on the column "ratings" was created, which improved the runtime of the two queries that counted the rows in "low_rating" and "high_rating". This was to be expected as "ratings" is not part of any primary key, and thus was not part of an index automatically created by Postgres.

Despite relying on the views "low_rating" and "high_rating" (both using the index "ratings_idx"), no performance improvement was observed for the query relating to the count of "no flop" actors.

A minor performance improvement was observed in the query that lists the top 10 "no flop actors".

**Q3 - i**

```
CREATE INDEX id_mid_actors
ON actors (mid);

CREATE INDEX id_year_movies
ON movies (year);
```

**Q3 - j**

Due to the fact that no query or view uses columns that are not already part of an index (created by default by Postgres), there are no indexes to create,

*BONUS*:

Due to the fact that queries on the view "non_existent" are very slow (approximately 14 seconds), a materialized view was created (despite not being in the assignment requirements) which dramatically improved runtimes of queries on that view.

The queries of subsections 3,4 in Q3-j saw a reduction in runtimes of about 14 seconds each.

note: I did not include these new runtimes in the query outputs as they are not part of the assignment requirements.

```
 create materialized view non_existent_mat as
        select name, mid
from all_combinations
where (name, mid) not in (select name, mid from actors);

refresh materialized view non_existent_mat;
```

**Q3 - k**

```
CREATE INDEX id_mid_actors
ON actors (mid);

CREATE INDEX id_name_actors
ON actors (name);
```

**Q3 - k - 2 - redone with materialized views**

```
/* Find most social actor */
CREATE MATERIALIZED VIEW num_of_co_actors AS
   SELECT actors.name as name, COUNT(DISTINCT actors1) as co_actors FROM actors
      INNER JOIN actors as actors1 on actors.mid = actors1.mid WHERE
         actors.name != actors1.name
   group by actors.name
   ORDER BY co_actors DESC;

SELECT * FROM num_of_co_actors WHERE
   co_actors=(SELECT MAX(num_of_co_actors1.co_actors)
            FROM num_of_co_actors as num_of_co_actors1);
```

**/* Query time: 128 ms */**

**Q3 - I redone with materialized views**

```
create materialized view mat_target_actors_names AS
   SELECT DISTINCT A.name as movie_actors
   FROM Movies M, Actors A
   WHERE M.title='Mr. & Mrs. Smith' and M.mid=A.mid;

create materialized view mat_common_actors AS
   SELECT M2.mid as movie_id, COUNT(DISTINCT A2.name) as number_of_common_actors
   FROM Movies M2, Actors A2
   WHERE name IN (SELECT movie_actors FROM mat_target_actors_names)
            and M2.mid=A2.mid and M2.title<>'Mr. & Mrs. Smith'
   GROUP BY M2.mid;

create materialized view mat_fraction_common_actors AS
   SELECT movie_id, ((max(number_of_common_actors) * 1.0) / (COUNT(movie_actors) * 1.0))
as fraction_actors
   FROM mat_common_actors, mat_target_actors_names
   GROUP BY movie_id;

create materialized view mat_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;

create materialized view mat_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 mat_tags_for_movie)
      and M2.mid = T2.mid and M2.title<>'Mr. & Mrs. Smith'
   GROUP BY M2.mid;

create materialized view mat_fraction_common_tags AS
   SELECT movie_id, ((max(number_of_common_tags) * 1.0) / (COUNT(movie_tags) * 1.0)) as
fraction_tags
   FROM mat_common_tags, mat_tags_for_movie
   GROUP BY movie_id;

create materialized view mat_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;

create materialized view mat_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 mat_genres_for_movie)
        and M2.mid = G2.mid and M2.title<>'Mr. & Mrs. Smith'
    GROUP BY M2.mid;

create materialized view mat_fraction_common_genres AS
    SELECT movie_id, (MAX(number_of_common_genres) * 1.0 / COUNT(movie_genres) * 1.0)
as fraction_genres
    FROM mat_genres_for_movie, mat_common_genres
    GROUP BY movie_id;

create materialized view mat_target_id(mid) AS
    SELECT MAX(movies.mid)
    FROM movies
    WHERE movies.title = 'Mr. & Mrs. Smith';

create materialized view mat_target_age(age) AS
    SELECT year
    FROM movies
    WHERE mid = (SELECT mid FROM mat_target_id);

create materialized view mat_max_age_gap(max_gap) AS
    SELECT MAX(ABS(year - (SELECT mat_target_age.age FROM mat_target_age)))
    FROM movies m1
    WHERE m1.mid <> (SELECT mid FROM mat_target_id);

create materialized view mat_mr_mrs_year(release_year) AS
    SELECT DISTINCT M.year
    FROM Movies M
    WHERE M.title='Mr. & Mrs. Smith';

create materialized view mat_normalized_age AS
    SELECT M2.mid, (((SELECT max_gap FROM mat_max_age_gap) - ABS((SELECT
release_year FROM mat_mr_mrs_year) - M2.year))
                    /(SELECT max_gap FROM mat_max_age_gap)) as normalized_age_gap
    FROM Movies M, Movies M2
    WHERE M.title='Mr. & Mrs. Smith' and M.title<>M2.title;
```

```
create materialized view mat_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;

create materialized view mat_mr_mrs_rating(movie_rating) AS
   SELECT DISTINCT M.rating
   FROM Movies M
   WHERE M.title='Mr. & Mrs. Smith';

create materialized view mat_normalized_rating AS
   SELECT M2.mid, (((SELECT rating_gap FROM mat_max_rating_gap) - ABS((SELECT
movie_rating FROM mat_mr_mrs_rating) - (SELECT M2.rating)))
                  /(SELECT rating_gap FROM mat_max_rating_gap)) as normalized_rating_gap
   FROM Movies M, Movies M2
   WHERE M.title='Mr. & Mrs. Smith' and M.mid<>M2.mid;

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 mat_normalized_age NA
FULL OUTER JOIN mat_fraction_common_actors FA ON FA.movie_id=NA.mid
FULL OUTER JOIN mat_fraction_common_tags FT ON FT.movie_id=FA.movie_id
FULL OUTER JOIN mat_fraction_common_genres FG ON FG.movie_id=FT.movie_id
FULL OUTER JOIN mat_normalized_rating NR ON NR.mid=FG.movie_id
FULL OUTER JOIN Movies M on NA.mid = M.mid

GROUP BY M.mid
ORDER BY recommendation DESC
```

**runtime: 160 ms**