Project - Phase 1
Group 7

Chloe Hei Yu Law - 40173275 - chloelaw75@hotmail.com
Khairo Khatib - 40071125 - khairo.khatib@gmail.com
Gasser Aly- 40135687- gasserkaly@gmail.com
Bozhidar Leshev 40105294 **-** bozhidar.leshev@gmail.com

SOEN 363 - Database Systems
Concordia University
October 30, 2022

1 Project Objectives

The objectives of phase one of the project are to help students in: (a) practicing and applying the data systems concepts, mainly modeling, storing, querying datasets, and (b) using SQL to query a real dataset such as MovieLens.

2 Loading the MovieLens Database

The MovieLens dataset is composed of information about 10,681 movies and their actors, directors, ratings and tags, all gathered from the online movie recommender service MovieLens. For this project, we will query the MovieLens dataset to extract useful information about movies.

(a)

```
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(mid),
    PRIMARY KEY (name, mid)
);

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

CREATE TABLE tag_names (
    tid integer,
```

```
    tag varchar,
    PRIMARY KEY (tid)
);

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

3 Querying the MovieLens Database

For each of the following questions, write and execute an SQL query that achieves the required task using PostgreSQL. If asked to, you have to use VIEWS in your answer, if not You may still define and use VIEWS whenever you find them suitable and report the execution time.

(a)

```
SELECT M.title
FROM actors A, movies M
WHERE A.name = 'Daniel Craig' and M.mid = A.mid
ORDER by M.title asc
```

(b)

```
SELECT A.name
FROM actors A, movies M
WHERE M.title = 'The Dark Knight' and M.mid = A.mid
ORDER by A.name asc
```

(c)

```
SELECT G.genre, COUNT(*) as N
FROM genres G, movies M
WHERE M.mid = G.mid
GROUP BY G.genre
HAVING COUNT(*)>1000
```

(d)

```
SELECT M.title, M.year, M.rating
FROM movies as M
ORDER BY
           year asc,
           rating desc;
```

(e)

```
(SELECT m.title
FROM tags AS tg, tag_names AS tgNm, movies AS m
WHERE tg.mid=M.mid and tg.tid=tgNm.tid and tgNm.tag like '%good%')
INTERSECT
(SELECT m.title
FROM tags AS tg, tag_names AS tgNm, movies AS m
WHERE tg.mid=M.mid and tg.tid=tgNm.tid and tgNm.tag like '%bad%')
```

(f.i)

```
SELECT *
FROM movies AS m
WHERE m.num_ratings = (SELECT  max(m.num_ratings)
      FROM movies AS m)
```

(f.ii)

```
SELECT *
FROM movies AS m
WHERE m.rating = (SELECT  max(m.rating)
      FROM movies AS m)
ORDER BY m.mid ASC
```

(f.iii)

```
(SELECT *
FROM movies AS m
WHERE m.num_ratings = (SELECT  max(m.num_ratings)
      FROM movies AS m))
INTERSECT
(SELECT *
FROM movies AS m
```

```
WHERE m.rating = (SELECT  max(m.rating)
      FROM movies AS m))
```

→ None fulfill this conjecture.

(f.iv)

```
SELECT *
FROM movies AS m
WHERE m.rating = (SELECT  min(m.rating)
      FROM movies AS m
      WHERE m.num_ratings!=0)
ORDER BY m.mid ASC
```

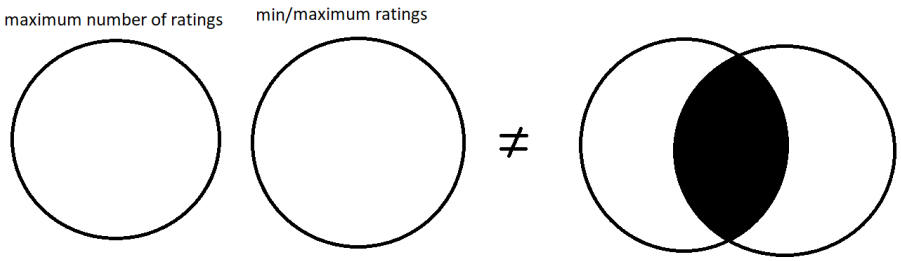
(f.v)

```
(SELECT *
FROM movies AS m
WHERE m.num_ratings = (SELECT  max(m.num_ratings)
      FROM movies AS m))
INTERSECT
(SELECT *
FROM movies AS m
WHERE m.rating = (SELECT  min(m.rating)
      FROM movies AS m
      WHERE m.num_ratings!=0))
```

→ None fulfill this conjecture.

(f.vi)

As shown by the previous queries, none of the max rating and min rating cases overlap with the maximum number rating. Therefore, our hypothesis is false.



(g)

```
SELECT m2.year, m2.title, m2.rating, m2.mid
FROM(
      SELECT m.year AS yr, min(m.rating) AS minR, max(m.rating) AS
maxR
      FROM movies AS m
      WHERE m.num_ratings >0 AND (m.year BETWEEN 2005 AND 2011)
      GROUP BY m.year) AS ratings_per_year, movies AS m2
WHERE m2.year = ratings_per_year.yr and (m2.rating =
ratings_per_year.minR OR m2.rating = ratings_per_year.maxR)
ORDER BY
      year ASC,
      rating ASC,
      title ASC;
```

(h.i)

```
CREATE VIEW high_ratings AS
SELECT DISTINCT actors.name, movies.rating
FROM actors, movies
WHERE movies.rating >= 4
AND movies.mid = actors.mid;

CREATE VIEW low_ratings AS
SELECT DISTINCT actors.name, movies.rating
FROM actors, movies
WHERE movies.rating < 4
AND movies.mid = actors.mid;

SELECT(
      SELECT COUNT(high_ratings.name)
      FROM high_ratings) AS high_ratings_num_rows,
      (
            SELECT COUNT(low_ratings.name)
            FROM low_ratings) AS low_ratings_num_rows;
```

(h.ii)

```
SELECT COUNT(high_ratings.name) AS num_no_flop
FROM high_ratings;
```

(h.iii)

```
SELECT high_ratings.name, COUNT(movies.title) as movies
FROM high_ratings, movies, actors
WHERE high_ratings.name = actors.name
AND actors.mid = movies.mid
GROUP BY high_ratings.name
ORDER BY movies desc
LIMIT 10
```

(i)

```
SELECT a.name
    FROM movies m, actors a
    WHERE a.mid = m.mid
    GROUP BY a.name
    ORDER BY max(m.year) - min(m.year) DESC
    LIMIT 1
```

(j.i)

```
CREATE VIEW co_actors AS
SELECT DISTINCT a.name
FROM movies m, actors a
WHERE m.mid in (SELECT m.mid
                FROM movies m , actors a
                WHERE a.name = 'Annette Nicole'
                and a.mid = m.mid)
    and m.mid = a. mid;
SELECT count(*) from co_actors
```

(j.ii)

```
CREATE VIEW all_combinations AS
SELECT c.name, m.mid
FROM movies m, co_actors c
WHERE m.mid in (SELECT m.mid
                FROM movies m , actors a
                WHERE a.name = 'Annette Nicole'
                and a.mid = m.mid);
```

```
SELECT count(*) from all_combinations
```

(j.iii)

```
CREATE VIEW non_existent AS
SELECT ac.name, ac.mid
FROM all_combinations ac
EXCEPT(SELECT DISTINCT a.name, m.mid
        FROM movies m, actors a
                WHERE m.mid in (SELECT m.mid
                                FROM movies m , actors a
                                WHERE a.name = 'Annette Nicole'
                                and a.mid = m.mid)
                and m.mid = a. mid);
SELECT count(*) FROM non_existent
```

(j.iv)

```
SELECT ca.name
FROM co_actors ca
WHERE ca.name not in (SELECT DISTINCT ne.name
        FROM non_existent ne)
    AND ca.name <> 'Annette Nicole'
```

(k-i)

```
Select a2.name, count(Distinct a1.name) -1
from actors a1,
actors a2
where
a2.name = 'Tom Cruise'
and
a1.mid in (
select Distinct movies.mid
from movies
join actors on
actors.mid = movies.mid
where actors.name = 'Tom Cruise'
)
group by a2.name
```

(k-ii)

```
CREATE VIEW most_social_actor AS (SELECT a1.name as Actor,
COUNT(DISTINCT a2.name)-1 as co_actors
FROM actors a1, actors a2
WHERE a1.mid = a2.mid
AND a2.mid IN(SELECT a1.mid
              FROM actors a1
              )
GROUP BY a1.name
ORDER BY co_actors DESC
);
SELECT* FROM most_social_actor
```

(l)

-- Actors

```
CREATE OR REPLACE VIEW total_actors AS
    SELECT DISTINCT COUNT( DISTINCT actors_in_movie.name)*1.0 as
total
        FROM actors actors_in_movie, movies
        WHERE movies.title = 'Mr. & Mrs. Smith'
        AND actors_in_movie.mid = movies.mid;
```

-- Actors, this view will give movies mids of other films that have one or more actors that are also in Mr and Mrs Smith, ordered descending

```
CREATE OR REPLACE VIEW other_films_with_same_actors AS
    SELECT actors.mid, COUNT( actors.mid)/total_actors.total AS score
    FROM actors, movies, total_actors
    WHERE actors.name IN (SELECT DISTINCT actors_in_movie.name
        FROM actors actors_in_movie, movies
        WHERE movies.title = 'Mr. & Mrs. Smith'
        AND actors_in_movie.mid = movies.mid)
    AND movies.mid = actors.mid
    and movies.title <> 'Mr. & Mrs. Smith'
    GROUP BY actors.mid, total_actors.total
     ORDER BY COUNT(actors.mid) DESC;
```

--Tags

```
CREATE OR REPLACE VIEW total_tags AS
SELECT COUNT(tags.tid)*1.0 as total
    FROM tags
    WHERE tags.mid IN (SELECT movies.mid
        FROM movies
        WHERE movies.title = 'Mr. & Mrs. Smith');
```

-- Tags, this view will give movies mids for films with at least one of tags as mr and mrs smith, ordered descending

```
CREATE OR REPLACE VIEW other_films_with_same_tags AS
SELECT COUNT (other_films.mid)/total_tags.total AS score,
other_films.mid
FROM tags other_films, movies m, total_tags
WHERE other_films.tid IN (SELECT tags.tid
    FROM tags
    WHERE tags.mid IN (SELECT movies.mid
        FROM movies
        WHERE movies.title = 'Mr. & Mrs. Smith')
    )
AND m.mid = other_films.mid
AND m.title <> 'Mr. & Mrs. Smith'
GROUP BY other_films.mid, total_tags.total
ORDER BY COUNT (other_films.mid) DESC;
```

-- Genres

```
CREATE OR REPLACE VIEW other_films_with_same_genres AS
SELECT COUNT (other_films.mid)/total_genres.total AS score,
other_films.mid
FROM genres other_films, movies m, total_genres
WHERE other_films.genre IN (SELECT DISTINCT genres.genre
    FROM genres
    WHERE genres.mid IN (SELECT movies.mid
        FROM movies
        WHERE movies.title = 'Mr. & Mrs. Smith')
    )
AND m.mid = other_films.mid
AND m.title <> 'Mr. & Mrs. Smith'
```

```
GROUP BY other_films.mid, total_genres.total
ORDER BY COUNT (other_films.mid) DESC;
```

-- Rating Gap
-- Normalized difference of ratings of other films compared to Mr and Mrs Smith (1 means they have exact same rating, closer to 0 means larger difference in rating

```
CREATE OR REPLACE VIEW other_films_norm_ratings AS
SELECT DISTINCT (1 - (ABS(other_film.rating - MMS2.rating)/5)) AS
score, other_film.mid
FROM movies other_film, movies MMS2
WHERE MMS2.rating = (SELECT MMS.rating
    FROM movies MMS WHERE MMS.title =  'Mr. & Mrs. Smith'
    LIMIT 1
)
AND other_film.title <> 'Mr. & Mrs. Smith'
AND other_film.rating IS NOT NULL
ORDER BY  (1 - (ABS(other_film.rating - MMS2.rating)/5)) DESC;
```

-- Age Gap

```
CREATE OR REPLACE VIEW largest_age_gap AS
    SELECT  (MAX(movies.year) -MIN(movies.year))*1.0 as difference
FROM movies;
```

-- Normalized age gap between films

```
CREATE OR REPLACE VIEW other_films_normage AS
SELECT DISTINCT (1-(ABS(other_film.year -
MMS2.year)/largest_age_gap.difference)) AS score, other_film.mid
FROM movies MMS2, movies other_film, largest_age_gap
WHERE MMS2.year =(SELECT MMS.year
    FROM movies MMS WHERE MMS.title =  'Mr. & Mrs. Smith'
    LIMIT 1)
AND other_film.title <> 'Mr. & Mrs. Smith'
ORDER BY (1-(ABS(other_film.year -
MMS2.year)/largest_age_gap.difference)) DESC;
```

-- Top ten similar films to Mr and Mrs Smith

```
CREATE OR REPLACE VIEW top_ten_match AS
```

```
SELECT movies.title, movies.rating, 100*(f1.score + f2.score +
f3.score + g1.score + g2.score)/5 AS similarity_perc
FROM other_films_with_same_actors f1, other_films_with_same_tags f2,
other_films_with_same_genres f3,
     other_films_normage g1, other_films_norm_ratings g2, movies
WHERE f1.mid= movies.mid
AND f2.mid= movies.mid
AND f3.mid= movies.mid
AND g1.mid= movies.mid
AND g2.mid= movies.mid
ORDER BY similarity_perc DESC
LIMIT 10;

SELECT *
FROM top_ten_match;
```

(m)

-- Checking if each table has duplicates:

-- Showing that movies contains duplicates

```
CREATE VIEW movies_duplicates_check AS
SELECT year, title, COUNT(title) as count
FROM movies
GROUP BY year, title
HAVING COUNT(title) > 1
UNION
SELECT year, title, COUNT(title)
FROM movies
GROUP BY year, title
HAVING COUNT(title) = 1;
```

-- Showing that actors contains duplicates

```
CREATE VIEW actors_duplicates_check AS
SELECT name, cast_position, COUNT(name) as count
FROM actors
GROUP BY name, cast_position
HAVING COUNT(name) > 1
UNION
SELECT name, cast_position, COUNT(name) as count
```

```
FROM actors
GROUP BY name, cast_position
HAVING COUNT(name) = 1;
```

-- Showing that genres does not contain duplicates

```
CREATE VIEW genres_duplicates_check AS
SELECT genre, COUNT(genre) as count
FROM genres
GROUP BY genre
HAVING COUNT(genre) > 1
UNION
SELECT genre, COUNT(genre) as count
FROM genres
GROUP BY genre
HAVING COUNT(genre) = 1;
```

-- Showing that tags does not contain duplicates

```
CREATE VIEW tags_duplicates_check AS
SELECT tid, COUNT(tid) as count
FROM tags
GROUP BY tid
HAVING COUNT(tid) > 1
UNION
SELECT tid, COUNT(tid) as count
FROM tags
GROUP BY tid
HAVING COUNT(tid) = 1;
```

-- Showing that tag_names does not contain duplicates

```
CREATE VIEW tag_names_duplicates_check AS
SELECT tag, COUNT(tag) as count
FROM tag_names
GROUP BY tag
HAVING COUNT(tag) > 1
UNION
SELECT tag, COUNT(tag) as count
FROM tag_names
```

```
GROUP BY tag
HAVING COUNT(tag) = 1;
```

-- Therefore, here is the list of tables that contains duplicates:
-- Movies
-- Actors

-- View for movies with no duplicates

```
CREATE VIEW movies_no_duplicates AS
SELECT year, title, COUNT(title)
FROM movies
GROUP BY year, title
HAVING COUNT(title) = 1;
```

-- View for actors with no duplicates

```
CREATE VIEW actors_no_duplicates AS
SELECT name, cast_position, COUNT(name) as count
FROM actors
GROUP BY name, cast_position
HAVING COUNT(name) = 1;
```

```
SELECT *
FROM movies_no_duplicates
ORDER BY movies_no_duplicates.count DESC;
SELECT *
FROM actors_no_duplicates
ORDER BY actors_no_duplicates.count DESC;
```

4 Performance

In this section, we will be exploring indexes and materialized views. Check this link for more information.

Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. You need to study the list of queries and check if you can use indexes to accelerate the query processing time. Some of these indexes will be of benefit in the case of bigger datasets. Thus, you may need to discuss these situations.

(a)

```
CREATE INDEX idx_movies ON movies (mid);
CREATE INDEX idx_actors ON actors (mid, name);
CREATE INDEX idx_actors ON actors (name);
CREATE INDEX idx_tags ON tags (mid, tid);
CREATE INDEX idx_genres ON genres (mid, genre);
CREATE INDEX idx_tag_names ON tag_names (tid);
```

(b)

After creating the indexes we notice an improvement in the query execution time, for example:

- Question 3-k-1 takes around **200ms** before indexes and that improves to around **100ms** after creating the indexes.
- Question 3-k-2 takes around **30 SECONDS** to query all fields on the created view. After indexes the average is around **13 SECONDS.**
- Question 3-i takes around **500ms** to execute before indexes. After indexes that is reduced to around **300ms.**
- Question 3-g takes around **160ms** before indexing, after it improves to around **80ms.**
- Other improvements can be seen on other queries as well with varying degrees.

For questions 3-k-2 and 3-l we asked you to use views. However, we are now interested in exploring the performance between views and materialized views. for this reason:

(b.a)

We notice that querying the materialized view takes around 90ms compared to the normal view which takes around 13 SECONDS. This makes sense because we know that on a normal view we compute the result every time we query the view, but on a materialized view the data is actually stored and we do not need to compute it on the fly which makes materialized view significantly faster. However, we have to remember that we need to refresh materialized views to update the data inside it.

```
CREATE MATERIALIZED VIEW IF NOT EXISTS materialized_most_social_actor
AS (SELECT a1.name as Actor, COUNT(DISTINCT a2.name)-1 as co_actors
FROM actors a1, actors a2
WHERE a1.mid = a2.mid
AND a2.mid IN(SELECT a1.mid
              FROM actors a1
              )
```

```
GROUP BY a1.name
ORDER BY co_actors DESC
);

SELECT* FROMmaterialized_most_social_actor // this takes ~90ms
```

(b.b)
Find the implementation below. We notice that querying materialized views is a lot faster than normal views. This is consistent with what we learned because normal views are always computed on the spot when they are queried and the data is not stored explicitly in contrast, materialized views actually compute and store the data in the database so they are a lot faster to query. Keep in mind that materialized views must be updated (refreshed) either manually or by setting a periodic job to update or some other setting (such as update the materialized view on insertions ...etc)

In this question querying materialized views took around **70ms** while querying normal views took about **10 SECONDS**

-- Actors

```
CREATE MATERIALIZED VIEW IF NOT EXISTS materialized_total_actors AS
    SELECT DISTINCT COUNT( DISTINCT actors_in_movie.name)*1.0 as
total
        FROM actors actors_in_movie, movies
        WHERE movies.title = 'Mr. & Mrs. Smith'
        AND actors_in_movie.mid = movies.mid;
```

-- Actors, this view will give movies mids of other films that have one or more actors that are also in Mr and Mrs Smith, ordered descending

```
CREATE MATERIALIZED VIEW IF NOT EXISTS
materialized_other_films_with_same_actors AS
    SELECT actors.mid, COUNT(
actors.mid)/materialized_total_actors.total AS score
    FROM actors, movies, materialized_total_actors
    WHERE actors.name IN (SELECT DISTINCT actors_in_movie.name
        FROM actors actors_in_movie, movies
        WHERE movies.title = 'Mr. & Mrs. Smith'
        AND actors_in_movie.mid = movies.mid)
    AND movies.mid = actors.mid
```

```
    and movies.title <> 'Mr. & Mrs. Smith'
    GROUP BY actors.mid, materialized_total_actors.total
     ORDER BY COUNT(actors.mid) DESC;
```

-- Tags

```
CREATE MATERIALIZED VIEW IF NOT EXISTS materialized_total_tags AS
SELECT COUNT(tags.tid)*1.0 as total
    FROM tags
    WHERE tags.mid IN (SELECT movies.mid
        FROM movies
        WHERE movies.title = 'Mr. & Mrs. Smith');
```

-- Tags, this view will give movies mids for films with at least one of tags as mr and mrs smith, ordered descending

```
CREATE MATERIALIZED VIEW IF NOT EXISTS
materialized_other_films_with_same_tags AS
SELECT COUNT (other_films.mid)/materialized_total_tags.total AS
score, other_films.mid
FROM tags other_films, movies m, materialized_total_tags
WHERE other_films.tid IN (SELECT tags.tid
    FROM tags
    WHERE tags.mid IN (SELECT movies.mid
        FROM movies
        WHERE movies.title = 'Mr. & Mrs. Smith')
    )
AND m.mid = other_films.mid
AND m.title <> 'Mr. & Mrs. Smith'
GROUP BY other_films.mid, materialized_total_tags.total
ORDER BY COUNT (other_films.mid) DESC;
```

-- Genres

```
CREATE MATERIALIZED VIEW IF NOT EXISTS materialized_total_genres AS
SELECT COUNT( DISTINCT genres.genre)*1.0 as total
    FROM genres
    WHERE genres.mid IN (SELECT DISTINCT movies.mid
        FROM movies
        WHERE movies.title = 'Mr. & Mrs. Smith');
```

-- Genres, finds movies mid for films with at least one of the same tags as mr and mrs smith, films ordered from highest number of same tags to lowest

```
CREATE MATERIALIZED VIEW IF NOT EXISTS
materialized_other_films_with_same_genres AS
SELECT COUNT (other_films.mid)/materialized_total_genres.total AS
score, other_films.mid
FROM genres other_films, movies m, materialized_total_genres
WHERE other_films.genre IN (SELECT DISTINCT genres.genre
    FROM genres
    WHERE genres.mid IN (SELECT movies.mid
        FROM movies
        WHERE movies.title = 'Mr. & Mrs. Smith')
    )
AND m.mid = other_films.mid
AND m.title <> 'Mr. & Mrs. Smith'
GROUP BY other_films.mid, materialized_total_genres.total
ORDER BY COUNT (other_films.mid) DESC;
```

-- Rating Gap
-- Normalized difference of ratings of other films compared to Mr and Mrs Smith (1 means they have exact same rating, closer to 0 means larger difference in rating

```
CREATE MATERIALIZED VIEW IF NOT EXISTS
materialized_other_films_norm_ratings AS
SELECT DISTINCT (1 - (ABS(other_film.rating - MMS2.rating)/5)) AS
score, other_film.mid
FROM movies other_film, movies MMS2
WHERE MMS2.rating = (SELECT MMS.rating
    FROM movies MMS WHERE MMS.title =  'Mr. & Mrs. Smith'
    LIMIT 1
)
AND other_film.title <> 'Mr. & Mrs. Smith'
AND other_film.rating IS NOT NULL
ORDER BY  (1 - (ABS(other_film.rating - MMS2.rating)/5)) DESC;
```

-- Age Gap

```
CREATE MATERIALIZED VIEW IF NOT EXISTS materialized_largest_age_gap
AS
    SELECT  (MAX(movies.year) -MIN(movies.year))*1.0 as difference
```

```
FROM movies;
```

-- Normalized age gap between films

```
CREATE MATERIALIZED VIEW IF NOT EXISTS
materialized_other_films_normage AS
SELECT DISTINCT (1-(ABS(other_film.year -
MMS2.year)/materialized_largest_age_gap.difference)) AS score,
other_film.mid
FROM movies MMS2, movies other_film, materialized_largest_age_gap
WHERE MMS2.year =(SELECT MMS.year
    FROM movies MMS WHERE MMS.title =  'Mr. & Mrs. Smith'
    LIMIT 1)
AND other_film.title <> 'Mr. & Mrs. Smith'
ORDER BY (1-(ABS(other_film.year -
MMS2.year)/materialized_largest_age_gap.difference)) DESC;
```

-- Top ten similar films to Mr and Mrs Smith

```
CREATE MATERIALIZED VIEW IF NOT EXISTS materialized_top_ten_match AS
SELECT movies.title, movies.rating, 100*(f1.score + f2.score +
f3.score + g1.score + g2.score)/5 AS similarity_perc
FROM materialized_other_films_with_same_actors f1,
materialized_other_films_with_same_tags f2,
materialized_other_films_with_same_genres f3,
     materialized_other_films_normage g1,
materialized_other_films_norm_ratings g2, movies
WHERE f1.mid= movies.mid
AND f2.mid= movies.mid
AND f3.mid= movies.mid
AND g1.mid= movies.mid
AND g2.mid= movies.mid
ORDER BY similarity_perc DESC
LIMIT 10;

SELECT *
FROM materialized_top_ten_match;  -- this takes ~70ms
```