Project Phase I

Submitted to: Essam Mansour

Data Systems for Software Engineers

SOEN 363– Winter 2022

Written by:

Marie-Eve Hazari, ID:40156408, email: marie.hazari@hotmail.ca
Ali Hanni, ID: 40157164, email: alihanni95@gmail.com
Beshoy Soliman, ID: 40047115, email: beshoysoliman11@gmail.com
Alireza Ziarizi ID: 40027914, email: alireza.ziaeework@gmail.com

Concordia University

Discussion on execution time (indexes and materialized view):

After adding indexes and materialized views for some critical views used throughout the project, it seems clear that the execution time is way smaller than that of the normal queries performed in part 3 of the project. Indeed, as materialized views are a 'hybrid' between views and tables, data retrieved using materialized views is physically kept in disk space. It can therefore be accessed faster without having to go through the query each time. However, a materialized view must be updated manually if data is changed in the database.
As for the indexes, many queries were much faster with them even though the difference was less significant than between materialized and normal views. Some indexes works better than other, the reason for that is because in order for index to achieve optimal a table has to have a lot of data(many rows)

Table Creation:

Please take into account that the DATE TYPE in the Table Movies for the attribute year is an INTEGER.

```
CREATE TABLE Movies (
   mid INTEGER UNIQUE,
   title VARCHAR(120),
   year INTEGER,
   rating REAL,
   num_ratings REAL,
   PRIMARY KEY (mid)
);

CREATE TABLE Actors (
   mid INTEGER,
   name VARCHAR(120),
   cast_position INTEGER,
   PRIMARY KEY (mid,name),
   FOREIGN KEY (mid) REFERENCES Movies(mid)

);

CREATE TABLE Genres (
   mid INTEGER ,
   genre VARCHAR(120),
   PRIMARY KEY (mid,genre),
   FOREIGN KEY (mid) REFERENCES Movies(mid)

);

CREATE TABLE Tags (
   mid INTEGER,
   tid INTEGER,
```

```
    PRIMARY KEY (mid,tid),
    FOREIGN KEY (mid) REFERENCES Movies(mid)

);

CREATE TABLE Tag_names (
    tid INTEGER UNIQUE,
    tag VARCHAR(120),
    PRIMARY KEY (tid),
    FOREIGN KEY (tid) REFERENCES Tags(tid)

);
```

Question 3 Querying the MovieLens Database

a)

```
SELECT movies.title
FROM Movies
JOIN Actors ON actors.mid=movies.mid
WHERE actors.name='Daniel Craig'
ORDER BY movies.title ASC;
```

b)

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

c)

```
SELECT g.genre, COUNT(g.mid) as Count
FROM genres g
GROUP BY g.genre
HAVING COUNT(g.mid) > 1000
ORDER BY Count ASC;
```

d)

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

e)

```
SELECT M.title
FROM  Movies M, Tags T1
WHERE T1.mid=M.mid AND (T1.mid IN (
                        SELECT T.mid
                        FROM Tag_names TN, Tags T
                        WHERE TN.tag LIKE 'good%' AND T.tid=TN.tid
                                GROUP BY T.mid)
```

```
            AND T1.mid IN (
                        SELECT T.mid
                        FROM Tag_names TN, Tags T
                        WHERE TN.tag LIKE 'bad%' AND T.tid=TN.tid
                        GROUP BY T.mid)
             )
GROUP BY M.mid
```

f)i

```
SELECT DISTINCT ON (M1.title) *
FROM Movies M1
WHERE M1.num_ratings = (SELECT MAX(M.num_ratings)
                                FROM Movies M )
```

f)ii

```
SELECT *
FROM Movies M1
WHERE M1.rating = (SELECT MAX(M.rating)
                          FROM Movies M )
ORDER BY M1.mid ASC
```

f)iii

```
SELECT *
FROM Movies M1
WHERE M1.rating IN (SELECT MAX(M.rating)
                          FROM Movies M )
AND M1.num_ratings IN (SELECT MAX(M.num_ratings)
                                   FROM Movies M )
```

No

f)iv

```
SELECT *
FROM Movies M1
WHERE M1.rating = (SELECT MIN(M.rating)
                          FROM Movies M )
ORDER BY M1.mid ASC
```

f)v

```
SELECT *
FROM Movies M1
WHERE M1.rating = (SELECT MIN(M.rating)
                  FROM Movies M )
AND M1.num_ratings = (SELECT MAX(M.num_ratings)
                    FROM Movies M )
```

No

f) vi.
False, it's none of them.

g)

```
SELECT M1.title, M1.year, M1.rating
FROM (SELECT M.year, MAX(M.rating) as maxrating
      FROM Movies M
      WHERE M.year>=2005 AND M.year<=2011
      GROUP BY M.year) Max,
     (SELECT M.year, MIN(M.rating) as minrating
      FROM Movies M
      WHERE M.year>=2005 AND M.year<=2011
      GROUP BY M.year) Min, Movies M1
WHERE (M1.year=Max.year AND M1.rating=Max.maxrating) OR (M1.year=Min.year AND
M1.rating=Min.minrating)
GROUP BY M1.title, M1.year, M1.rating
ORDER BY M1.year ASC, M1.rating ASC, M1.title ASC
```

h)i

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

SELECT COUNT(*) FROM high_ratings
```

———————

```
CREATE VIEW low_ratings AS
SELECT DISTINCT A.name
FROM Movies M, Actors A
WHERE M.mid=A.mid AND M.rating<4

SELECT COUNT(*) FROM low_ratings
```

h)ii

```
SELECT COUNT(*)
FROM high_ratings HR
WHERE HR.name NOT IN (SELECT *
                      FROM low_ratings)
```

h)iii

```
SELECT HA.name, COUNT(A.mid)
FROM(
    SELECT *
    FROM high_ratings HR
    WHERE HR.name NOT IN (SELECT *
                          FROM low_ratings)) HA, Actors A
```

```
WHERE HA.name=A.name
GROUP BY HA.name
ORDER BY COUNT(A.mid) DESC
LIMIT 10

i)
SELECT L.name
FROM (SELECT A.name, MAX(M.year)- MIN(M.year) as longevity
        FROM Movies M, Actors A
        WHERE M.mid=A.mid
        GROUP BY A.name) L
WHERE L.longevity = (SELECT MAX(L.longevity)
                     FROM (SELECT MAX(M.year)-MIN(M.year) as longevity
                           FROM Movies M, Actors A
                           WHERE M.mid=A.mid
                           GROUP BY A.name) L)

j)i
CREATE VIEW co_actors AS
SELECT  DISTINCT A1.name
FROM Actors A1
WHERE A1.name <>'Annette Nicole' AND A1.mid in (SELECT A.mid
                                FROM Actors A
                                WHERE A.name='Annette Nicole')

SELECT COUNT(*) FROM co_actors

j)ii
CREATE VIEW all_combinations AS
SELECT CA.name, AN.mid
FROM co_actors CA, (SELECT A.mid
                    FROM Actors A
                    WHERE A.name='Annette Nicole') AN

SELECT COUNT(*) FROM all_combinations

j)iii
CREATE VIEW non_existent AS
SELECT AC.name, AC.mid
FROM all_combinations AC
WHERE AC.mid NOT IN (SELECT A.mid
          FROM actors A
          WHERE A.name = AC.name)

j)iv
SELECT *
FROM co_actors c
WHERE c.name NOT IN (SELECT n.name FROM non_existent n)
```

k)i

```
SELECT A2.name, COUNT(DISTINCT A1.name)
FROM Actors A1, Actors A2
WHERE A1.name <>'Tom Cruise' AND A2.name = 'Tom Cruise'
AND A1.mid in (SELECT A.mid
            FROM Actors A
            WHERE A.name='Tom Cruise')
GROUP BY A2.name
```

k)ii

```
CREATE VIEW social AS
   SELECT A1.name, COUNT(DISTINCT(A2.name)) AS num_coActors
   FROM Actors A1, Actors A2
   WHERE A1.mid = A2.mid AND A1.name <> A2.name
   GROUP BY A1.name ORDER BY 2 DESC


SELECT name, num_coactors
FROM social
WHERE num_coactors >= ALL(SELECT MAX(num_coactors)
                  FROM social)
```

l)

```
Create VIEW number_of_actors as
(
select mm.mid as movieid, count(distinct a.name) as totalactors
from movies mm
      join Actors A on mm.mid = A.mid
group by mm.mid );

create View common_actors as
(
select moviex.mid as movieid1, moviey.mid as movieid2, count(distinct actorx.name) as
commonActor
from Movies moviex,
   Movies moviey,
   actors actorx,
   actors actory
where moviex.mid = actorx.mid
 and moviey.mid = actory.mid
 and actorx.name = actory.name
 and moviex.mid <> moviey.mid
GROUP BY moviex.mid, moviey.mid );


create view fraction_actors as
(
```

```
select moviex.mid                                    as movieid1,
     moviey.mid                                     as movieid2,
     (CAST(commonActor AS float) / CAST(totalactors AS float)) as fraction
from Movies moviex,
   Movies moviey,
   number_of_actors,
   common_actors
where moviex.mid = number_of_actors.movieid
 and moviex.mid = common_actors.movieid1
 and moviey.mid = common_actors.movieid2
 and moviex.mid <> moviey.mid);


Create VIEW number_of_tags as
(
select mm.mid as movieid, count(distinct a.tid) as totalTags
from movies mm
       join tags A on mm.mid = A.mid
group by mm.mid );


create View common_tags as
(
select moviex.mid as movieid1, moviey.mid as movieid2, count(distinct tagx.tid) as commontags
from Movies moviex,
   Movies moviey,
   tags tagx,
   tags tagsy
where moviex.mid = tagx.mid
 and moviey.mid = tagsy.mid
 and tagx.tid = tagsy.tid
 and moviex.mid <> moviey.mid
GROUP BY moviex.mid, moviey.mid );

create view fraction_tags as
(
select moviex.mid                                    as movieid1,
     moviey.mid                                     as movieid2,
     (CAST(commontags AS float) / CAST(totalTags AS float)) as fraction
from Movies moviex,
   Movies moviey,
   number_of_tags,
   common_tags
where moviex.mid = number_of_tags.movieid
 and moviex.mid = common_tags.movieid1
 and moviey.mid = number_of_tags.movieid
 and moviex.mid <> moviey.mid);
```

```
Create VIEW number_of_genres as
(
select mm.mid as movieid, count(distinct a.genre) as totalGenre
from movies mm
       join genres A on mm.mid = A.mid
group by mm.mid );

create View number_of_common_genres as
(
select moviex.mid as movieid1, moviey.mid as movieid2, count(distinct genrex.genre) as
commongenre
from Movies moviex,
   Movies moviey,
   genres genrex,
   genres genrey
where moviex.mid = genrex.mid
 and moviey.mid = genrey.mid
 and genrex.genre = genrey.genre
 and moviex.mid <> moviey.mid
GROUP BY moviex.mid, moviey.mid );


create view fraction_common_genres as
(
select moviex.mid                                    as movieid1,
     moviey.mid                                    as movieid2,
     (CAST(commongenre AS float) / CAST(totalGenre AS float)) as fraction
from Movies moviex,
   Movies moviey,
   number_of_genres,
   number_of_common_genres
where moviex.mid = number_of_genres.movieid
 AND moviex.mid = number_of_common_genres.movieid1
 AND moviey.mid = number_of_common_genres.movieid2
 AND moviex.mid <> moviey.mid);


create view year_gap as
(
select mx.mid as mx1, my.mid as mx2, 1-(ABS(CAST( (mx.year-my.year)As float))/CAST(
(mx.year)As float))
 as yeargap
from movies mx,
   movies my
where mx.mid <> my.mid);
```

```
create view rate_gap as
(
select mx.mid as mx1, my.mid as my2, 1-(ABS(mx.rating-my.rating)/mx.rating) as ratingap
from movies mx,
   movies my
where mx.mid <> my.mid
 and mx.rating <> 0
  );


CREATE VIEW similarity_of_movies AS
SELECT moviex.mid                                                    AS movieid1,
    moviey.mid                                                  AS movieid2,
    (((f1.fraction + 0 + f3.fraction + YD.yeargap + RD.ratingap) / 5.0) * 100) AS Similarity
FROM movies moviex,
   movies moviey,
   fraction_actors f1,
   fraction_common_genres f3,
   year_gap YD,
   rate_gap RD
WHERE moviex.mid = f1.movieid1
 AND moviey.mid = f1.movieid2
 AND moviex.mid = f3.movieid1
 AND moviey.mid = f3.movieid2
 AND moviex.mid = YD.mx1
 AND moviey.mid = YD.mx2
 AND moviex.mid = RD.mx1
 AND moviey.mid = RD.my2
 AND moviex.mid <> moviey.mid;


CREATE VIEW suggestion as
(
SELECT moviex.title, moviey.rating, S.Similarity
FROM similarity_of_movies S,
   movies moviex,
   movies moviey
WHERE S.movieid1 = moviex.mid
 AND moviey.title = 'Mr. & Mrs. Smith'
 AND S.movieid2 = moviey.mid
ORDER BY S.Similarity DESC
LIMIT 10);

m)
-- Find duplicates in movies table --

SELECT title, year, rating, num_ratings, COUNT(mid)
FROM movies
```

```
GROUP BY title, year, rating, num_ratings
HAVING COUNT(mid) > 1
ORDER BY title;

-- Query to display movies without duplicate --

SELECT MIN(mid) AS mid, title, year, rating, num_ratings
FROM movies
GROUP BY title, year, rating, num_ratings
ORDER BY title;

-- Create View for movies without duplicates --

CREATE VIEW movies_no_duplicates AS
   SELECT MIN(mid) AS mid, title, year, rating, num_ratings
   FROM movies
   GROUP BY title, year, rating, num_ratings;



-- Display actors and duplicate movies they played in --

SELECT name, cast_position, title AS movie_title, t1.count AS count
FROM (
   SELECT
        name, cast_position, title,
        COUNT(movies.mid)
   FROM actors
   JOIN movies ON actors.mid = movies.mid
   GROUP BY name, cast_position, title
   HAVING COUNT(movies.mid) > 1) AS t1
ORDER BY name, cast_position;

-- Create View for actors without duplicates --

CREATE VIEW actors_no_duplicates AS
   SELECT actors.*
   FROM actors JOIN movies_no_duplicates mnd ON actors.mid = mnd.mid

-- Display genres and duplicate movies associated with them --

SELECT genre, title as movie_title, COUNT(movies.mid)
FROM genres
JOIN movies ON genres.mid = movies.mid
GROUP BY genre, title
HAVING COUNT(movies.mid) > 1
ORDER BY genre, movies.title
```

```
-- Create view for genres without duplicates --

CREATE VIEW genres_no_duplicates AS
   SELECT genres.mid, genre
   FROM genres
   JOIN movies_no_duplicates mnd ON genres.mid = mnd.mid
   ORDER BY  mid, genre

-- Query that shows that no duplicates in tag_names (returns nothing) --

SELECT tid, tag, COUNT(tid)
FROM tag_names
GROUP BY tid, tag
HAVING COUNT(tid) > 1

-- Query to show duplicates for tags table --

SELECT tags.tid, tag, title as movie_title, COUNT(movies.mid) AS count
FROM tags
JOIN tag_names tn on tags.tid = tn.tid
JOIN movies ON movies.mid = tags.mid
GROUP BY tags.tid, tag, title
HAVING COUNT(movies.mid) > 1

-- Create view for tags without duplicates --

CREATE VIEW tags_no_duplicates AS
   SELECT tags.mid, tags.tid
   FROM tags
   JOIN movies_no_duplicates mnd on tags.mid = mnd.mid
   ORDER BY tags.mid, tags.tid
```

Question 4 Performance

a)

```
CREATE INDEX movies_index_mid ON movies (mid);
CREATE INDEX movies_index_year ON movies (year);
CREATE INDEX movies_index_title ON movies (title);
CREATE INDEX movies_index_rating ON movies (rating);
CREATE INDEX movies_index_num_ratings ON movies (num_ratings);
CREATE INDEX actors_index ON actors (mid, name);
CREATE INDEX genres_index ON genres (mid, genre);
CREATE INDEX tag_names_index ON tag_names (tid);
```

b)i

```
CREATE MATERIALIZED VIEW mat_social AS
   SELECT A1.name, COUNT(DISTINCT(A2.name)) AS num_coActors
   FROM Actors A1, Actors A2
   WHERE A1.mid = A2.mid AND A1.name <> A2.name
```

```
   GROUP BY A1.name ORDER BY 2 DES


SELECT name, num_coactors
FROM mat_social
WHERE num_coactors >= ALL(SELECT MAX(num_coactors)
                    FROM mat_social)


b)ii
CREATE MATERIALIZED VIEW mat_similarity_of_movies AS
SELECT moviex.mid AS movieid1 , moviey.mid AS movieid2 ,( ((f1.fraction + 0 + f3.fraction +
YD.yeargap + RD.ratingap)/5.0) * 100) AS Similarity
FROM movies moviex,movies moviey,fraction_actors f1, fraction_common_genres f3,year_gap YD
,rate_gap RD
WHERE moviex.mid = f1.movieid1 AND moviey.mid = f1.movieid2   AND moviex.mid =
f3.movieid1 AND moviey.mid = f3.movieid2 AND moviex.mid = YD.mx1 AND moviey.mid =
YD.mx2 AND moviex.mid = RD.mx1 AND moviey.mid = RD.my2 AND moviex.mid <>
moviey.mid;

CREATE MATERIALIZED VIEW mat_suggestion AS

  SELECT moviex.title, moviey.rating, S.Similarity
  FROM mat_similarity_of_movies S,
     movies moviex,
     movies moviey
  WHERE S.movieid1 = moviex.mid
   AND moviey.title = 'Mr. & Mrs. Smith'
   AND S.movieid2 = moviey.mid
  ORDER BY S.Similarity DESC
  LIMIT 10
```