CSci 4707 Programming Assignment 2

Alex Lema

I. Impact of Indexes (Points: 100)

A. Print number of distinct movie ids with rating = 1

```
SELECT COUNT(*) FROM(
    SELECT DISTINCT P2.movieid, P2.rating FROM ratings P2
    WHERE P2.rating =1
    GROUP BY movieid )this_table;
```

COUNT(*)
5878

This query returned 5878 records in all cases.

B. Print number of distinct movie ids with ratings in the range 1 to 3

```
SELECT COUNT(*)
    FROM(
    SELECT DISTINCT p2.movieid, p2.rating FROM ratings P2
    WHERE 1 < p2.rating AND 3 > p2.rating
    GROUP BY p2.movieid )this_table;
```

COUNT(*)
10987

This query returned 10987 records in all cases

C. Print number of users who have rated more than 100 movies

```
SELECT COUNT(*)
    FROM(
    SELECT userid, inside_table.post_count AS sum_of FROM
        (SELECT DISTINCT p2.userid, COUNT(userid) AS post_count
        FROM ratings p2
        GROUP BY p2.userid)inside_table
    WHERE inside_table.post_count > 100
    )this_table;
```

COUNT(*)
2424

This query returned 2424 records in all cases

D. Find average movie rating per user

```
SELECT COUNT(*)
FROM(
SELECT inner_table.userid, inner_table.sum_ratings/inner_table.post_count AS
avg_rating
FROM(
SELECT DISTINCT p2.userid, COUNT(userid) AS post_count, SUM(rating) as
sum_ratings
FROM ratings p2
GROUP BY p2.userid
 ) inner_table
)this_table;
```

	COUNT(*)
▶	10183

This query returned 10183 records in all cases.

>>Based on the execution times, which field would you recommend creating an index on for the ratings table? Which of the two indexes in scenario 2 and 3 is clustered and unclustered?

The table below is the time comparison of running time in seconds of each Scenario (1,2,&3), as the assignment outline. I notice that scenario 3 is the fastest of all three. I will recommend creating an index on for the ratings table in userid.

Scenario 2 is unclustered

Scenario 3 is clustered

PART	TIME1(seconds)	RECORDS1	TIME2(seconds)	RECORD2	TIME3(seconds)	RECORD3
A	1.391	5878	1.312	5878	1.047	5878
B	1.75	10987	1.609	10987	1.25	10987
C	2.25	2424	1.844	2424	0.953	2424
D	2.359	10183	2.391	10183	1.094	10183

II. Table joins and query optimization (Points: 100)

A. Find average movie popularity per user. If you have defined primary keys on the tables, drop them before running this query. Note query execution time and report number of records returned.

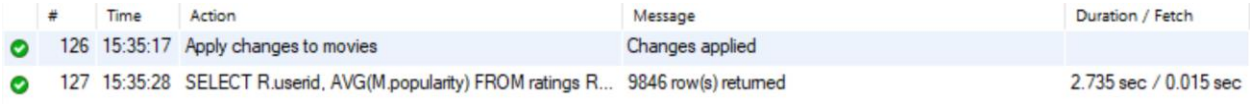
*SQL query for the three situations:

```
SELECT R.userid, AVG(M.popularity)
FROM ratings R, movies M
WHERE R.movieid = M.id
GROUP BY R.userid;
```

In this case there were not index or primary keys on the tables.
There were 9846 records returned and it took 3.093 seconds.

a. Now create primary key (id) on the movies table and run query A again. Don't forget to note the execution time.

It took 2.735 seconds for it to return 9846 records.



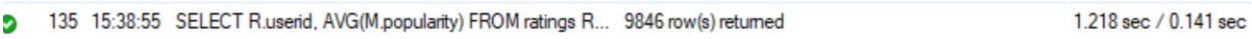
#	Time	Action	Message	Duration / Fetch
126	15:35:17	Apply changes to movies	Changes applied	
127	15:35:28	SELECT R.userid, AVG(M.popularity) FROM ratings R...	9846 row(s) returned	2.735 sec / 0.015 sec

b. Now create primary key (userid, movieid) on the ratings table and run query A. Report execution time.

With primary key (id) on the movies, and it took 0.281 seconds for it to return 9846 records.

#	Time	Action	Message	Duration / Fetch
132	15:37:46	SELECT R.userid, AVG(M.popularity) FROM ratings R...	9846 row(s) returned	0.281 sec / 2.094 sec

Without primary key (id) on the movies, and It took 1.218 seconds for it to return 9846 records.



135	15:38:55	SELECT R.userid, AVG(M.popularity) FROM ratings R...	9846 row(s) returned	1.218 sec / 0.141 sec

c. What is the performance improvement obtained after creating the two indexes as compared to running the query without indexes defined.

Adding the clustered index of each of the constituent tables, the query execution time shortened dramatically

B. Find number of movies which have higher popularity but lower vote_average than others (This query is only on movies table)

```
SELECT COUNT(*) FROM(
SELECT COUNT(*)
FROM movies M2, movies M3
WHERE M2.popularity > M3.popularity
AND
M2.vote_average < M3.vote_average
GROUP BY M2.id)
this_table;
```

	COUNT(*)
▶	8928

This query returned 8928 rows in 87.047 seconds

✔	162	16:28:45	SELECT COUNT(*) FROM(SELECT COUNT(*) FRO...	1 row(s) returned	87.047 sec / 0.000 sec

C. Find number of movies which have equal popularity but unequal vote_average than others (This query is only on movies table)

```
SELECT COUNT(*) FROM(
SELECT COUNT(*) FROM movies M1, movies M2
WHERE M1.popularity = M2.popularity
AND M1.vote_average <> M2.vote_average
AND M1.id != M2.id
GROUP BY M1.id
)this_table;
```

	COUNT(*)
▶	9982

This query returned 9982 records in 15.125 seconds

	#	Time	Action	Message	Duration / Fetch
✔	158	16:21:23	SELECT COUNT(*) FROM(SELECT COUNT(*) FRO...	1 row(s) returned	15.125 sec / 0.000 sec

D. Find number of ratings of movies with popularity > 5 and vote_average > 5

a. First try this query with a join condition on the 2 tables (join is on the common field in the two tables)

Using primary key (id) on the movies and primary key (userid, movieid) on the ratings lets us the query below:

```
SELECT COUNT(*) FROM movies M, ratings R
WHERE M.vote_average > 5
AND M.popularity > 5
AND R.movieid = M.id;
```

	COUNT(*)
▶	211304

This query returned 211304 records in 2.187 seconds:

✔	7	16:58:23	SELECT COUNT(*) FROM movies M, ratings R WHE...	1 row(s) returned	2.187 sec / 0.000 sec

b. Next, run the query without the join condition. Note the execution time as well as number of records returned.

```
SELECT COUNT(*) FROM movies M, ratings R
WHERE M.vote_average > 5
AND M.popularity > 5;
```

	COUNT(*)
▶	3445000000

This query returned 3445000000 records in 333.015 seconds:

✔	12	17:12:53	SELECT COUNT(*) FROM movies M, ratings R WH...	1 row(s) returned	333.015 sec / 0.000 sec

c. Are number of records same in parts a and b? Why?

The number of records are not the same because in the cross product of ratings and movies, there are a significant amount of records now paired with movies that have vote-average and popularity values that are greater than 5, therefore there is no real world connection between the two entities. This condition substantially increases the number of movies x ratings tuples. Due to the nature of cross product, every rating is now paired with every movie with a vote average and popularity greater than 5.

III. Access Permissions (Points: 80)

1. Login as root user
2. Create 2 users (User1, User2) Identified by Password.
3. Grant select permission on movies table with grant option to User1.
4. Logout root.

```
mysql> SELECT user, host FROM mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| User1            | localhost |
| User2            | localhost |
```

A. Show the output of the following by logging in as User1 and User2 in the database.

a. SHOW GRANTS FOR `user1`@`localhost`

User1

```
mysql> SHOW GRANTS FOR 'User1'@'localhost';
+-----------------------------------------------------------------------+
| Grants for User1@localhost                                            |
+-----------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'User1'@'localhost'                             |
| GRANT SELECT ON `p2`.`movies` TO 'User1'@'localhost' WITH GRANT OPTION |
+-----------------------------------------------------------------------+
```

User2

```
mysql> SHOW GRANTS FOR 'User1'@'localhost';
ERROR 1044 (42000): Access denied for user 'User2'@'localhost' to database 'mysq
```

b. SELECT count(*) FROM Movies;

User1

```
mysql> SELECT count(*) FROM Movies;
+----------+
| count(*) |
+----------+
|    29991 |
+----------+
```

User2

```
mysql> SELECT count(*) FROM Movies;
ERROR 1046 (3D000): No database selected
mysql> use p2;
ERROR 1044 (42000): Access denied for user 'User2'@'localhost' to database 'p2'
```

c. SELECT count(*) FROM Ratings;

User1

```
mysql> SELECT count(*) FROM Ratings;
ERROR 1142 (42000): SELECT command denied to user 'User1'@'localhost' for table
'ratings'
```

User2

```
mysql> SELECT COUNT(*) FROM ratings;
ERROR 1046 (3D000): No database selected
mysql> use p2
ERROR 1044 (42000): Access denied for user 'User2'@'localhost' to database 'p2'
```

B. Login as User1 and Pass SELECT permission on Movies to User2. Show the outputs of 3 commands listed in A by logging in as User2.

a.

```
mysql> SHOW GRANTS FOR 'User2'@'localhost';
+-----------------------------------------------------+
| Grants for User2@localhost                          |
+-----------------------------------------------------+
| GRANT USAGE ON *.* TO 'User2'@'localhost'           |
| GRANT SELECT ON `p2`.`movies` TO 'User2'@'localhost' |
+-----------------------------------------------------+
2 rows in set (0.00 sec)

mysql> SHOW GRANTS FOR 'User1'@'localhost';
ERROR 1044 (42000): Access denied for user 'User2'@'localhost' to database 'mysq
```

b.

```
mysql> SELECT COUNT(*) FROM Movies;
+----------+
| COUNT(*) |
+----------+
|    29991 |
+----------+
```

c.

```
mysql> SELECT COUNT(*) FROM Ratings;
ERROR 1142 (42000): SELECT command denied to user 'User2'@'localhost' for table
'ratings'
```

C. Login as root and Revoke access of movies table from User1. Show outputs of 3 commands listed in A by logging in as User1 and User2.

a. REVOKE SELECT, GRANT OPTION ON P2.movies from 'User'@'localhost'

User1

```
mysql> SHOW GRANTS FOR 'User1'@'localhost';
+-----------------------------------------
| Grants for User1@localhost
+-----------------------------------------
| GRANT USAGE ON *.* TO 'User1'@'localhost'
```

User2

```
mysql> SHOW GRANTS FOR 'User2'@'localhost';
+-----------------------------------------------------+
| Grants for User2@localhost                          |
+-----------------------------------------------------+
| GRANT USAGE ON *.* TO 'User2'@'localhost'           |
| GRANT SELECT ON `p2`.`movies` TO 'User2'@'localhost' |
+-----------------------------------------------------+
```

b.

User1

```
mysql> SELECT COUNT(*) FROM Movies;
ERROR 1046 (3D000): No database selected
mysql> use p2
ERROR 1044 (42000): Access denied for user 'User1'@'localhost' to database 'p2'
```

User2

```
mysql> SELECT COUNT(*) FROM Movies;
+----------+
| COUNT(*) |
+----------+
|    29991 |
+----------+
```

c.

User1

```
mysql> use p2
ERROR 1044 (42000): Access denied for user 'User1'@'localhost' to database 'p2'
mysql> SELECT COUNT(*) FROM Ratings;
ERROR 1046 (3D000): No database selected
```

User2

```
mysql> SELECT COUNT(*) FROM Ratings;
ERROR 1142 (42000): SELECT command denied to user 'User2'@'localhost' for table
'ratings'
```

Submit the grant and revoke commands you used to give and revoke access from the user:

GRANT and REVOKE commands:

- GRANT SELECT ON P2.Movies TO 'User1'@'localhost' WITH GRANT OPTION;
- GRANT SELECT ON P2.movies TO 'User2'@'localhost'
- REVOKE SELECT, GRANT OPTION ON P2.movies FROM 'User1'@'localhost';

IV. Transactions

This task involves looking at the lifecycle of a transaction. Perform the following operations:

A. Find the number of entries in the ratings table that have rating = 5. Report the SQL query and output

Entries with rating of 5: 144849

SQL query: Select * from ratings where rating = 5;

```
Select * from ratings where rating = 5;
```


```
+--------+---------+--------+------------+
144849 rows in set (1.07 sec)
```

B. Execute the following command to prevent the default autocommit operation: SET autocommit = 0

Execution es done.

```
mysql> SET autocommit = 0;
Query OK, 0 rows affected (0.00 sec)
```

C. Start Transaction; update the ratings of movies with rating 5 to rating 5.5; Rollback Transaction; Now find the number of entries in the ratings table that have rating = 5.5. Report the output.

Using P2

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> update ratings p2 set p2.rating =5.5 Where P2.rating = 5;
Query OK, 144849 rows affected (5.00 sec)
Rows matched: 144849  Changed: 144849  Warnings: 0
```

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> select COUNT(*) from ratings p2 where p2.rating = 5.5;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.77 sec)
```

D. Start Transaction; update the ratings of movies with rating 5 to rating 5.5; Commit Transaction; Now find the number of entries in the ratings table that have rating = 5.5. Report the output.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> update ratings p2 set p2.rating =5.5 Where P2.rating = 5;
Query OK, 144849 rows affected (5.00 sec)
Rows matched: 144849  Changed: 144849  Warnings: 0
```

```
mysql> COMMIT;
Query OK, 0 rows affected (0.06 sec)

mysql> select COUNT(*) from ratings p2 where p2.rating = 5.5;
+----------+
| COUNT(*) |
+----------+
|   144849 |
+----------+
1 row in set (0.78 sec)

mysql>
```

number of entries = 144849

E. Explain the difference in output observed in parts C and D

In the transaction in part c, the changes are rolled back instead of the commit, which means that the changes are not written to the database but are rolled back to the value they were before.

If there were no ratings with a value of 5.5 in the original sample that there would be no ratings with a value of 5.5 in the final result so that there was change that will occur in the situation in part c but not in the situation in part d.

After execution of COMMIT statement, the transaction can not be ROLLBACK. Once ROLLBACK is executed database reaches its previous state.