CSci 4707 Programming Assignment 2 Solutions (Total 350)

The database schema as well as commands to load the data have been provided in **schema.sql**. There are two tables: **Ratings** with 1 million records, and **Movies** with 9,997 records. You are expected to turn in one file (PDF, txt or doc) with SQL queries along with answers to the questions below.

This assignment tests the following:

I. Impact of Indexes (100)

This task involves running queries and noting the query execution time. Commands to note execution time have been provided in **schema.sql**. You do not need to turn in SQL outputs if the output is anything besides count, instead write the number of records returned by each query.

Write the SQL query for the following and then run these queries on the Ratings table using the different scenarios given below

A. Print number of distinct movie ids with rating = 1 **(10)**
   select count(distinct movieid) from ratings where rating =1;
   5878

B. Print number of distinct movie ids with ratings in the range 1 to 3 **(10)**
   select count(distinct movieid) from ratings where rating between 1 and 3;
   16137

C. Print number of users who have rated more than 100 movies **(10)**
   select count(userid) from (select userid, count(rating) from ratings group by userid having count(rating) > 100) a;
   2424

D. Find average movie rating per user **(10)**
   select userid, avg(rating) from ratings group by userid;
   10183 rows returned

Scenario 1: Run the above queries (A-D) on the given table as is (without defining any key or index). Note execution time and number of records returned by the query.

Scenario 2: Create an index on **rating**. Now run the queries again and note execution time. **(25)**

create index movie_rating on ratings (rating); **(15);** Correct time measure **(10)**

Scenario 3: Define (userid, movieid) as the primary key. Run the queries again and note execution time. **(25)**

alter table ratings add primary key(userid,movieid); **(15)**; Correct time measure **(10)**

Based on the execution times, which field would you recommend creating an index on for ratings table? Which of the two indexes in scenario 2 and 3 is clustered and unclustered? **(10)**

II. Table joins and query optimization (100)

This task involves joining the two tables and then performing some select operations. Write the SQL query, run it and report query execution time and number of records returned:

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. **(10 for query)**
select userid, avg(popularity) from movies m join ratings r on r.movieid = m.id group by userid;
9846 rows returned

   a. Now create primary key (id) on the **movies** table and run query A again. Don't forget to note the execution time. **(20: query + time)**
   alter table movies add primary key(id);

   b. Now create primary key (userid, movieid) on the **ratings** table and run query A. Report execution time. **(20: query + time)**
   alter table ratings add primary key(userid,movieid);

   c. What is the performance improvement obtained after creating the two indexes as compared to running the query without indexes defined. **(5)**

B. Find number of movies which have higher **popularity** but lower **vote_average** than others (This query is only on movies table) **(10)**
select count(distinct m1.id) from movies m1 join movies m2 on m1.popularity > m2.popularity and m1.vote_average < m2.vote_average;
8928

C. Find number of movies which have equal **popularity** but unequal **vote_average** than others (This query is only on movies table) **(10)**
select count(distinct m1.id) from movies m1 join movies m2 on m1.popularity = m2.popularity and m1.vote_average != m2.vote_average;
9982

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) **(10)**
   select count(*) from ratings r, movies m where r.movieid = m.id and popularity > 5 and vote_average > 5;
   211304

b. Next, run the query without the join condition. Note the execution time as well as number of records returned. **(10)**
select count(*) from ratings r, movies m where popularity > 5 and vote_average > 5;
3445000000

c. Are number of records same in parts a and b? Why? **(5)**
Not the same because b doesn't have a join condition, hence will create a Cartesian product of all ratings and movies that match the given condition

III. Access Permissions (80)

Perform the following operations first:

1. Login as root user

2. Create 2 users (User1, User2) Identified by Password **(10)**
CREATE USER 'User1'@'localhost' IDENTIFIED BY 'password';
CREATE USER 'User2'@'localhost' IDENTIFIED BY 'password';

3. Grant select permission on movies table with grant option to User1. **(10)**
GRANT SELECT ON P2.movies TO 'User1'@'localhost' WITH GRANT OPTION;

4. Logout root.

A. Show the output of the following by logging in as User1 and User2 in the database. (15)
   a. SHOW GRANTS (5)
   GRANT USAGE ON *.* TO 'User1'@'localhost' |
   GRANT SELECT ON `P2`.`movies` TO 'User1'@'localhost' WITH GRANT OPTION

   b. SELECT count(*) FROM Movies; (2.5 + 2.5)
   9997

   c. SELECT count(*) FROM Ratings; (2.5 + 2.5)
   ERROR 1142 (42000): SELECT command denied to user 'User1'@'localhost' for table 'ratings'

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. (10 query + 10: 5+2.5+2.5 = 20 total)
GRANT SELECT ON P2.movies TO 'User2'@'localhost';
   a. GRANT USAGE ON *.* TO 'User2''@'localhost'

GRANT SELECT ON `P2`.`movies` TO 'User2''@'localhost' WITH GRANT OPTION

b. 9997
c. 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. (10 + 15)
REVOKE SELECT, GRANT OPTION ON P2.movies FROM 'User1'@'localhost';
User1:
   a. GRANT USAGE ON *.* TO 'User1'@'localhost'
   b. ERROR 1142 (42000): SELECT command denied to user 'User1'@'localhost' for table 'movies'
   c. ERROR 1142 (42000): SELECT command denied to user 'User1'@'localhost' for table ratings

User2:
   a. GRANT USAGE ON *.* TO 'User2''@'localhost'
      GRANT SELECT ON `P2`.`movies` TO 'User2''@'localhost' WITH GRANT OPTION
   b. 9997
   c. ERROR 1142 (42000): SELECT command denied to user 'User2'@'localhost' for table ratings

IV. Transactions (70)

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 (10)
select count(*) from ratings where rating=5;
144849

B. Execute the following command to prevent the default autocommit operation: (5)

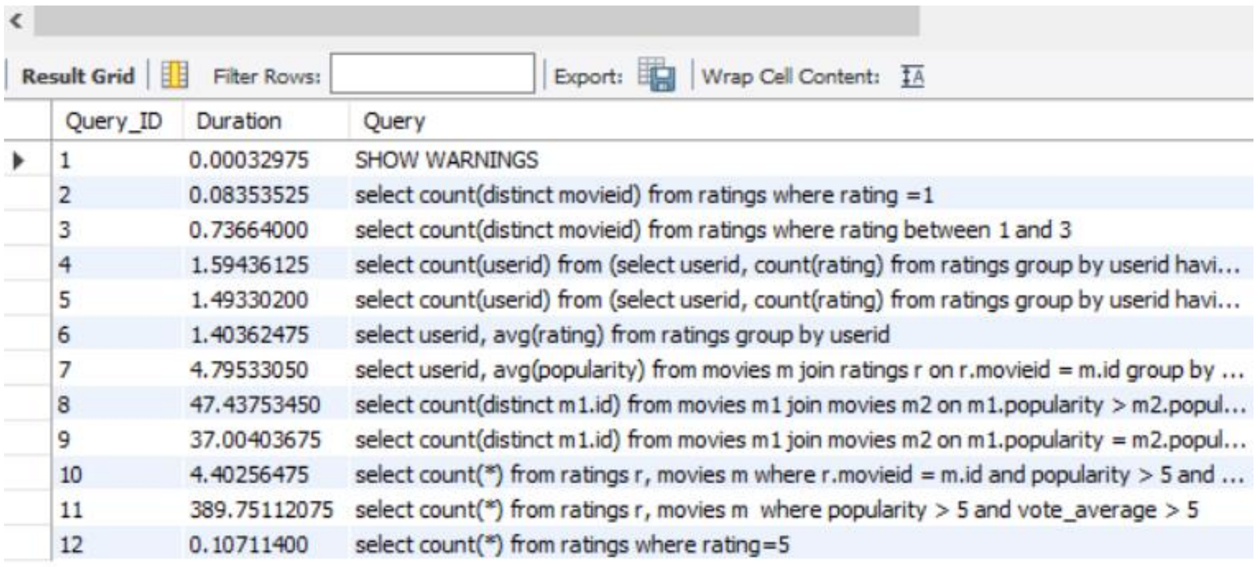
```
SET autocommit = 0
```

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. (20)
Update ratings set rating = 5.5 where rating = 5;

select count(*) from ratings where rating=5.5;
0

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. (20)
select count(*) from ratings where rating=5.5;
144849

E. Explain the difference in output observed in parts C and D (15)


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Query_ID	Duration	Query
1	0.00032975	SHOW WARNINGS
2	0.08353525	select count(distinct movieid) from ratings where rating =1
3	0.73664000	select count(distinct movieid) from ratings where rating between 1 and 3
4	1.59436125	select count(userid) from (select userid, count(rating) from ratings group by userid havi...
5	1.49330200	select count(userid) from (select userid, count(rating) from ratings group by userid havi...
6	1.40362475	select userid, avg(rating) from ratings group by userid
7	4.79533050	select userid, avg(popularity) from movies m join ratings r on r.movieid = m.id group by ...
8	47.43753450	select count(distinct m1.id) from movies m1 join movies m2 on m1.popularity > m2.popul...
9	37.00403675	select count(distinct m1.id) from movies m1 join movies m2 on m1.popularity = m2.popul...
10	4.40256475	select count(*) from ratings r, movies m where r.movieid = m.id and popularity > 5 and ...
11	389.75112075	select count(*) from ratings r, movies m where popularity > 5 and vote_average > 5
12	0.10711400	select count(*) from ratings where rating=5