CS 301 - Introduction to Database Systems
Course Project Specification Phase B (Total Weightage: 10%)

General Instructions:

- **You should submit phase B in the same team as phase A.**
- **Always make sure to run the analyze command (or Vacuum Analyze command) to get updated table statistics. Otherwise, query optimizers start to make sub optimal estimates of the query costs.**

Question 1 (100 points):

For this question, you are expected to create large datasets and load it into PostgreSQL. Also, we would be using "EXPLAIN" and "ANALYZE" statements available in PostgreSQL extensively in this homework. Please refer to their websites www.postgresql.org/docs/current/static/sql-explain.html and http://www.postgresql.org/docs/current/static/sql-analyze.html to familiarize yourselves with the syntax of these statements.

Details of datasets to be created:

Please make a note of the schema which needs to be created.

(a) Table Actor: consists of the following two attributes: (1) a_id (a random integer between 1 and 300000) and (2) name (a random string of length 15 characters). a_id is the primary key of the table. So as expected two actors should not have the same id. We have 300000 actors.

(b) Table Production Company: consists of following attributes: (1) pc_id (a random integer between 1 and 80000; (2) name (a random string of length 10 characters); (3) address (a random string of length 30 characters). We have 80000 companies.

(c) Table Movie: consists of following attributes: (1) m_id (a random integer between 1 and 1000000); (2) name (a random string of length 10 characters; (3) year (a random integer between 1900 and 2000); (4) imdb score (a random float between 1 and 5); (5) production company (foreign key referencing to pc_id of the table production company). As expected all the production companies in this table should be a valid company in the production company table. m_id is the primary key of this table. Logic behind distribution of pc_id is defined next: For teams of size-2: If both team members have even roll numbers then pc_id in this table is a random integer between 1 and 80000. If one is even and the other is odd, then 90% of the pc_ids in this table are a random integer between 1 and 500 and others are distributed over 501 and 80000. If both are odd then 99% of the pc_ids in this table are a random integer between 1 and 10 and others are distributed over 11 and 80000. This table has 1000000 movies. Couple of other things: (1) if roll number of at-least two students in the team is less than 50 then then year of 90% of tuples is between 1990 and 2000 (otherwise uniform random); (2) Additionally, if its a size-3 team then imdb score for 95% of tuples is between 1-2. (otherwise uniform random).

(d) Table Casting: consists of following attributes: (1) m_id and (2) a_id. m_id is a foreign key reference to the movie table. a_id is a foreign key referencing the actor table. m_id and a_id together form the primary key of this table. Each movie has 4 actors and all movies have actors. Couple of things regarding the distribution of values; For size-3 teams 95% of actor ids are within the range 1-1000. In case of size-2 teams (or solo), if one of the roll numbers is greater than 50 than 95% of the actor ids are within the range 1--10000. Otherwise uniform random.

Create these datasets and load it into the PostgreSQL. You would have to create files which have the required SQL insert commands for each table. These files can then be loaded into PostgreSQL. Following this you need to create indexes (using the create index command in PostgreSQL) on the following attributes: (a) a_id in the actor table; (b) m_id in the movie table; (c) imdb score in the movie table; (d) year in the movie; (e) m_id in the casting table; (f) a_id in the casting table; (g) pc_id in movie table.

Part A: Experimenting with Query Selectivity

1. Consider following range queries on VOTES and YR columns.

    Q1: SELECT name FROM movie WHERE imdb score < 2;
    Q2: SELECT name FROM movie WHERE imdb score between 1.5 and 4.5;
    Q3: SELECT name FROM movie WHERE year between 1900 and 1990;
    Q4: SELECT name FROM movie WHERE year between 1990 and 1995;
    Q5: SELECT * FROM movie WHERE pc_id < 50;
    Q6: SELECT * FROM movie WHERE pc_id > 20000;

2. Run the explain analyze command for Q1,Q2,Q3, Q4 and Q5 and Submit the output. Briefly explain the query strategy adopted by the query optimizer in each of the queries.

3. Is the index on imdb score used for Q1 and Q2 queries? Give an intuitive explanation of the observed results in terms of parameters like query selectivity (which can computed by using COUNT(*)).

4. Is the index on YR used for both Q3 and Q4 queries? Give an intuitive explanation of the observed results in terms of parameters like query selectivity (which can computed by using COUNT(*)).

5. Is the index on pc_id used for both Q5 and Q6 queries? Give an intuitive explanation of the observed results in terms of parameters like query selectivity (which can computed by using COUNT(*)).

Part B: Join Strategies

1. Consider the following SQL queries

    Q1: Join Actor, Movie and Casting; Where a_id < 50; finally, the query outputs actor name and movie name
    Q2: Join Actor and Casting; Where m_id < 50; finally, the query outputs actor name and movie name
    Q3: Join Movie and Production Company; where imdb score is less than 1.5. Finally, the query outputs the movie name and production company.
    Q4: Join Movie and Production Company; where year is between 1950 and 2000. Finally, the query outputs the movie name and production company.

2. Run the explain analyze command for each Q1, Q2, Q3 and Q4 and submit its output and also the SQL queries. Briefly explain the query strategy adopted by the query optimizer in each of the queries.

3. Briefly justify (to the extent possible) the query optimizer choices (choice of query processing algorithms) in these queries based on parameters like query selectivity and cost models covered in lecture.