Database System Implementation – CSE 507

Homework 3

Due Date: 2 April 12:00 noon

Instructions:

- All submissions must be made through usebackpack site for this course (https://www.usebackpack.com/iiitd/w2017/cse507)
- Only one submission per team would be considered and graded. It would be assumed that all members of the team have participated equally and same score would be given to all members of the team.
- Your submission should have names of all the members of your team. Only one submission should be uploaded per team.
- Question 2 is for teams of size 3 only. This question would not be graded for teams of size 2 and less.
- Any assumptions made while solving the problem should be clearly stated in the solution. Reasonable ones would be accepted and graded.
- As always correctness of the algorithm must be ensured.
- It is preferred that all algorithms are implemented in the same language.
- If you are implementing in C++ then your file extension must be .cc only. No other extension would be accepted.
- TAs would be quizzing you on your code. You must understand each and every line of your submitted code. Also the implementation specifications mentioned in the questions need to be strictly followed. Failure to adhere to these requirements would result in substantial loss of points.
- Very Important: Your code should not have a directory structure. All files (code + results + written material for questions) should be present in just one folder. Note that this is absolutely crucial for grading this assignment.

Question 1 (120 points):

For this question, you are expected to create large datasets and load it into PostgreSQL. Please install this software on your computers. This software is available as a free download at this website: http://www.postgresql.org/download/ . 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 following two attributes: (1) a_id (a random integer between 1 and 200000) 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 200000 actors.

(b) Table **Production Company:** consists of following attributes: (1) pc_id (a random integer between 1 and 50000; (2) name (a random string of length 10 characters); (3) address (a random string of length 30 characters). We have 50000 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: Take the last digit of roll numbers of two students in the team. If both are even then pc_id in this table is a random integer between 1 and 50000. If one is even and other is odd, then 90% of the pc_ids in this table are a random integer between 1 and 100 and others are distributed over 101 and 50000. 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 50000. This table has 1000000 movies. Couple of other things: (1) if the team has two students from MTech 16 batch then year of 90% of tuples is between 1990 and 2000 (otherwise uniform random); (2) If the team has atleast one BTech 3rd year student 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 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; (1) if the team has two BTech 4th year students then 90% of actor ids are within the range 1—100; (2) if the team has atleast one MTech 15 batch students then 99% of movie ids are within the range 1-10; otherwise these are 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) name in the actor table; (b) name in the movie table; (c) year in the movie; (d) m_id in the casting table; (e) a_id in the casting 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;
2. Run the explain plan for Q1,Q2,Q3 and Q4 and Submit the output.
3. Is the index on VOTES 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(*)).

Part B: Aggregate Operations

1. Consider following two SQL queries.
   - Q1: SELECT name FROM movie WHERE imdb score <= (SELECT MIN (imdb score) FROM movie WHERE year between 1990 and 1995);
   - Q2: SELECT name FROM movie WHERE imdb score <= ALL (SELECT imdb score FROM movie);
2. Run the explain plan for each Q1 and Q2 and submit the output.
3. Briefly justify the query optimizer choices (choice of query processing algorithms) in Q1 and Q2 based on parameters like query selectivity and cost models covered in class.

Part C: 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, Movie and Casting; Where m_id < 100; finally, the query outputs actor name and movie name

   Q3: Join Actor, Movie and Casting; Where year is between 1990 and 2000; finally, the query outputs actor name and movie name

   Q4: Join Movie and Production Company; where production company id is less than 50. Finally, the query outputs the movie name and production company.

Q5: Join Movie and Production Company; where imdb score is less than 1.5. Finally, the query outputs the movie name and production company.

Q6: 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 plan for each Q1, Q2, Q3, Q4, Q5 and Q6 and submit its output and also the SQL queries.
3. Briefly justify the query optimizer choices (choice of query processing algorithms, join order) in these queries based on parameters like query selectivity and cost models covered in class.

Question 2 (20 points) (For teams of size 3 only):

Consider block nested loop join algorithm over two relations R and S. Assume that R has 5 blocks and S has 4 blocks. Further assume that R is the outer relation and S is the inner relation in the join. Compute the number of page faults for block nested loop join algorithm under LRU and MRU page replacement polices. This needs to be computed for buffer sizes 4, 5, and 6. Out of these one buffer is reserved for output. You need to use the appropriate variants of block nested loop algorithm as the number of buffers are varied. For e.g., Nb-2 buffers are kept for outer block, and if the inner relation fits in main memory completely (after taking out space for min one block for outer relation and output buffer) then we put that in. Your answer should clearly state the assumptions made regarding when the page reference is counted. Reasonable ones would be accepted.