GENERAL INSTRUCTIONS

Answer each question in a separate book.

Indicate on the cover of each book the area of the exam, your code number, and the question answered in that book. On one of your books list the numbers of all the questions answered. Return all answer books in the folder provided. Additional answer books are available if needed.

Do not write your name on any answer book.

SPECIFIC INSTRUCTIONS

Answer all four (4) questions. Before beginning to answer a question make sure that you read it carefully. If you are confused about what the question means, state any assumptions that you have made in formulating your answer. Good luck!

The grade you will receive for each question will depend on both the correctness of your answer and the quality of the writing of your answer.

Policy on misprints and ambiguities:

The Exam Committee tries to proofread the exam as carefully as possible. Nevertheless, the exam sometimes contains misprints and ambiguities. If you are convinced a problem has been stated incorrectly, mention this to the proctor. If necessary, the proctor can contact a representative of the area to resolve problems during the first hour of the exam. In any case, you should indicate your interpretation of the problem in your written answer. Your interpretation should be such that the problem is nontrivial.
1. You are given a relation Friends(A, B) that describes the friendship relationship in a social network (A is a friend of B). You can assume that the relation is symmetric, in other words if Friends(A,B) then also Friends(B,A). We want to compute all pairs of people P(X,Y) that are connected through a chain of friendships: this is expressed with the following Datalog program:

\[
P(x,y) \leftarrow \text{Friends}(x,y).
\]
\[
P(x,y) \leftarrow \text{Friends}(x,z), P(z,y).
\]

a) Describe the algorithm for the naive and semi-naive evaluation of the above Datalog program.

b) Since the relation Friends is huge, we want to use MapReduce to speed up the computation. Discuss how one could implement semi-naive evaluation in MapReduce. You should briefly sketch the function of the mappers and reducers, and also how you would check the termination of the evaluation.

c) What are the limitations of using MapReduce for computing the above query? How would you modify MapReduce in order to overcome these limitations?

2. Modern main-memory computing systems are a combination of processors that are connected with fast buses. The inter-processor communication component is starting to become increasingly important for performance.

Consider a four socket server configuration and two “network topologies.” One, called “square,” which has each socket arranged at each vertex of a square, and thus each socket is connected to two other sockets. The other configuration called a “fully-connected mesh”, which is the square configuration with additional links along the diagonals. As you can imagine, in the square configuration moving data from the sockets that are diagonally opposite takes twice as long compared to moving data between two processors on the same edge of the square.

Further, assume that each processor has associated with it some local memory, and that the amount of memory attached to each processor is identical.

When running a database management system in such multi-socket systems, one will distribute/partition the data across all the memory available in the system. For simplicity, assume that each table is uniformly and randomly partitioned across all the sockets in the system.

a) Under this circumstance can you design an equijoin algorithm that is efficient from the communication perspective for the “square” topology, and another algorithm for the “mesh” topology? What are the pros and cons of your approach over using a standard method in which one re-partitions both tables and then joins each partition pair (locally at each socket)?
b) Next consider an 8 processor configuration, which for the square topology generalizes to a ring with each socket having two links, and the mesh topology generalizing to a fully-connected “clique” configuration with 8 sockets, in which each pair of sockets are connected by a bus. Do your two algorithms above generalize to (communication) efficient algorithms with the 8 socket case? If not, what would you change for the 8 socket case?

3. Concurrency Control
(a) In Optimistic Concurrency Control as described by Kung and Robinson, transactions record a “start_tn” that is the value of the “CurrentTN” counter when they start, and are assigned a transaction number (“tn”) that is the CurrentTN at the time that they successfully validate. Joe Qualtaker makes a mistake. He correctly records the “start_tn” as the value of the CurrentTN at the time a transaction starts, but also assigns this value as the “tn.” That is, at transaction startup, he executes:

```
<
  start_tn = CurrentTN;
  tn = CurrentTN++
>
```

Discuss the impact of this mistake. How does it change the correctness, performance, and allowable schedules of Optimistic Concurrency Control?

(b) Jane Qualtaker is implementing multiple granularity locking. She has interpreted the rule for multi-path hierarchies as “before getting an S or IS lock on an object, get an S or IS lock on one of its parents; before getting an X or IX lock on an object, get an X or IX lock on all of its parents.” (Ignore SIX locks for this question.)

Suppose we implement her approach on the following hierarchy:

```
     DB
      |    
     File  
      |     
     Index
     Page  
      |     
     Record
```

Then suppose transaction T1 gets an IX on DB, IX on File, and X on Page. This satisfies her condition for locking in hierarchies. Describe another transaction T2 that could follow T1 and create a conflict that should have been prevented by locking. Can you modify her definition to fix this problem?
4. Consider two relational tables A and B, each consisting of millions of tuples. Each tuple describes a person (with attributes such as first name, last name, phone, city, state, zip code, etc.). Our goal is to match A and B, that is, to find tuple pairs (x,y) such that x is in A, and y is in B, and x and y refer to the same real-world person.

To do this, suppose we will manually write matching rules. Each rule R(x,y) has the form t_1(x,y) AND t_2(x,y) AND ... AND t_n(x,y) => match, where each term t_i(x,y) has the form \[f_i(x,y) \text{ op}_i v_i\].

Here x and y are tuples from tables A and B, respectively. f_i is a similarity function being applied to an attribute value of x and an attribute value of y, op_i is a comparison operator such as =, >, <, etc., and v_i is a value. Any similarity function such as edit distance, Jaccard measure, TF/IDF, etc. can be used in the rule. For example, the following rule

\[
R(x,y) = [edit\_distance(x.last\_name,y.last\_name) < 3] \\
\quad \text{AND} [exact\_match(x.phone,y.phone) = 1] \Rightarrow \text{match}
\]

states that if the edit distance of the two last names is less than 3 and the phones match exactly, then the two tuples match.

Suppose we have written a set of such rules. Describe an algorithm to execute the rules over the two tables A and B as fast as possible, on a single machine. Here you can assume that the two tables A and B are stored as two files on that machine.