University of Wisconsin-Madison
Computer Sciences Department

Database Qualifying Exam
Spring 2017

INSTRUCTIONS

1. Answer each question in a separate book.
2. 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.
3. **Do not write your name on any answer book.**
4. Answer **all five (5)** 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.
5. 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: ENTITY MATCHING

Suppose we want to match tuples within a single table A (that is, find tuples in A that refer to the same real-world entity). To reduce computational costs, suppose we want to first perform sorted neighborhood-based blocking (SNB). SNB first applies a function to each tuple in table A to compute for the tuple a key (e.g., the function may return the concatenation of the last name and the birth year as the key for the tuple). Next, SNB sorts all tuples in A in decreasing order of their key. Finally, SNB goes through the sorted tuples and returns all tuple pairs that are within a k distance in the order, where k is pre-specified. For example, if $k = 2$, then SNB only returns tuple pairs whose tuples are right next to each other in the order.

(a) In practice, table A may have tens of millions of tuples, making running SNB on a single machine slow. Describe an efficient algorithm to perform SNB over table A using a cluster of machines running Hadoop. Discuss any possible problems that may arise for your algorithm.

(b) Earlier we said k is pre-specified. In practice, it is very difficult to come up with the correct k. So we may want to run SNB for a range of k, say for all k from 3 to 7, and store the result of this run. Later we may want to query the result for a particular value of k, examine how the outputs for different values of k differ, then select an appropriate k based on our examination. Describe how you can modify the algorithm in Part (a) to run SNB for not just a single value k, but a range of values for k. At the end, the algorithm's output must be such that the user can quickly obtain the output for a particular value k in the range.

2: ASSOCIATION RULE MINING

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. Let T be a set of transactions where each transaction is a set of items from I.

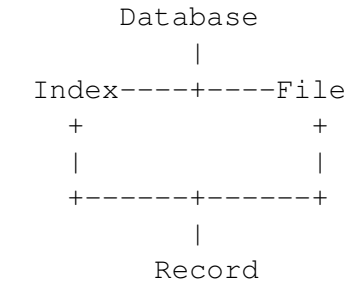
(a) Describe an algorithm that, given a support threshold, uses a Hadoop cluster to find all *frequent itemsets* from T.

(b) Let $A \rightarrow B$ and $C \rightarrow B$ be two association rules where C is the complement of A, that is, C contains all items from I that are not in A. Let T_1 and T_2 be two disjoint sets of transactions such that their union is T. Suppose that on set T_1 rule $A \rightarrow B$ has lower confidence than rule $C \rightarrow B$, and similarly on set T_2 rule $A \rightarrow B$ also has lower confidence than rule $C \rightarrow B$. Is it possible that on set T the reverse would happen, that is, rule $A \rightarrow B$ has higher confidence than rule $C \rightarrow B$? Explain your answer.

3: CONCURRENCY CONTROL

(a) Are serializable histories also recoverable? Explain your answer, and give a clear example to illustrate your position on this statement.

(b) Joe Qualtaker has successfully taken all the database classes at Wisconsin and is hired to implement the Gray et al. locking protocol for a new database engine. This engine has the following resource hierarchy.



Joe's implementation for two transactions manifests in the following lock sequence.

Transaction 1, which wants to modify a few records, first grabs an IX lock on the Database and then an X lock on the File. Thus, this transaction has locked all the paths above the X lock (on the File) in IX mode. Now, it can go and modify any record in the underlying file.

Transaction 2 wants to read a record. This transaction grabs an IS lock on the Database, then an IS lock on the Index, and finally an R lock on the Record of interest. This is what the protocol says for read paths.

Is this implementation faithful to the Gray et al.'s locking protocol?

If your answer is yes, explain why this protocol is correct. In this case, can you think of any optimizations to reduce the locking overhead? If your answer is no, explain why not, and what should be done to correct the implementation.

(c) Consider the optimistic concurrency protocol by Kung and Robinson. Is that protocol ANSI serializable? Justify your answer.

4: QUERY CONTAINMENT

In this question, we consider the following two classes of conjunctive queries defined over a schema that contains a single binary relation R:

$$P_k(x_1) : -R(x_1, x_2), R(x_2, x_3), \ldots, R(x_{k-1}, x_k).$$
$$C_k(x_1) : -R(x_1, x_2), R(x_2, x_3), \ldots, R(x_k, x_1).$$

For both classes of queries, the parameter k can take any value $k \geq 1$. The **path query** P_k outputs all the starting vertices of paths of length k, and the **cycle query** C_k outputs the starting vertices of cycles of length k.

(a) Describe the ordering of the queries in the class P_k with respect to query containment (in other words, if $q_1 \subseteq q_2$, then q_1 is "before" q_2 in the order). Is this ordering a *total ordering*? (A total ordering in our case is one where for any pair of path queries, one is *always* contained in the other.)

(b) Describe the ordering of the queries in the class C_k with respect to query containment. Is this ordering a total ordering?

5: DATALOG

Datalog is an elegant formal language, but it is often not expressive enough for many practical applications. Thus, it is typical that we extend Datalog with additional syntax and semantics. Here we will study how to extend Datalog by adding a *min* aggregator operator. We will use the following syntax:

$$C(x_1, x_2, \ldots, x_k, \min\langle z\rangle) : - \ldots.$$

We want this rule to be applied as follows: compute everything that matches the body of the rule, **group** the result by $x_1, \ldots, x_k$, and finally output the **minimum** value of z in each such group (we assume that the values can be ordered). For example, if the binary relation $E(x, y)$ denotes the edges of a directed graph, the Datalog rule $T(x, \min\langle y\rangle) : -E(x, y)$ outputs for each vertex x, the vertex y with the smallest value. To be more concrete, executing the rule over the graph $\{E(1,2), E(1,3), E(2,4), E(2,5)\}$ would produce the output $\{T(1,2), T(2,4)\}$.

Recall that we can *stratify* a Datalog program, by partitioning the predicates into equivalence classes through the precedence graph. The precedence graph has a vertex for each predicate, and an edge from R to S if there is a rule with head S and R in the body. Two predicates R, R' are in the same class (stratum) if there is a directed path from R to R' and from R' to R. In this case, we say that R, R' are *mutually recursive*.

We say that a rule is *recursive* if the body involves a predicate that is mutually recursive with the head. Suppose that we syntactically restrict the use of the *min* aggregation operator to only **non-recursive** rules. An example of such a Datalog program would be the following:

$$\begin{aligned} T(x, y) &: - E(x, y). \\ T(x, y) &: - T(x, z), E(z, y). \\ C(x, \min\langle y\rangle) &: - T(x, y). \end{aligned}$$

Explain what is the output relation C that the above Datalog program computes. Then, define a possible formal semantics for the extended Datalog with *min* aggregation operator applied only to **non-recursive** rules. How can we define a least-fixpoint model in this case?