University of Wisconsin-Madison
Computer Sciences Department

Database Qualifying Exam
Fall 2013

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. Join Algorithm

Consider the equijoin of two relations R and S, as in the SQL query:
Select *
From R, S
Where R.A = S.B

Furthermore, suppose that R is sorted on the A attribute, and S is sorted on the B attribute. You may find that that some of the following is under-specified; please explicitly list any assumptions you think you need to make in your answer.

a. Give the merge-join algorithm for such a problem (the equijoin of two relations sorted on their join attribute.)

b. While the merge algorithm in part a. is generally fast, it is somehow not satisfying when there are large "gaps" in the matching tuples. So, for example, suppose the ith tuple in R is r_i, and the jth tuple in S is s_j. Suppose that ri does not match s_(j+1), and furthermore r_(i+1) does not match any tuple in S until s_k, where k is substantially larger than j+1. Then the merge join from part a will compare r_(i+1) with many many S tuples with no output. It would be nice if the algorithm could skip to the next match. Fortunately, there is a data structure that lets you do this: a B+ tree index. So suppose there are indexes on R.A and S.B, modify your algorithm from part a. to "skip" to the next match. Note that the opportunities for skipping can be symmetric and your algorithm should handle this.

c. How would you expect the performance of your new algorithm to compare with the performance of hybrid hash join?

d. Now suppose that S has no index on it. You would now have the opportunity to first build an index on S and then do your new skipping merge join. How would you expect this index-building algorithm to compare with hybrid hash join?

2. Decision Support

Consider a table Sales(A,B,C,P), where A, B, and C are dimensions and P is a measure.

(a) What is the full "cube" of the sales table with "sum" as the aggregating operator? That is, what are the aggregates that will be computed for this cube?

(b) In general would it be most efficient to compute the cube "top down" (starting with the most highly aggregated aggregate) or "bottom-up" (starting with the least highly aggregated aggregate)? Why? Note: "Group By A" is more highly aggregated than "Group By AB."

(c) Suppose that there are n_A distinct values in column A, n_B distinct values in column B, and n_C distinct values in column C. What is the largest possible number of rows in the cube of Sales? (Equivalently, what is the largest number of non-null entries in the cube?) Does this require any bound on n, the number of tuples in the table? What other conditions can you impose on the values in the tuples in the table so that this bound is reached?

3. Concurrency Control

(a) In optimistic concurrency control as proposed by Kung and Robinson, one of the conditions that guarantees serializability can be stated as: if TN(Ti) < TN(Tj), then Write-Phase(Ti) ends before Write-Phase(Tj) begins and Write-Set(Ti) and Read-Set(Tj) do not overlap. Suppose that during the execution of a set of transactions T1, T2, …, Tn, each pair of transactions Ti and Tj in this set satisfy the condition. Explain why this schedule is serializable (it is not sufficient to just cite Kung and Robinson).

(b) What is the relationship between the set of schedules allowed by Kung and Robinson and the set of schedules allowed at degree 3 consistency as defined by Gray?

(c) Is non-strict 2PL identical to any of the levels of consistency defined by Gray et al.? Explain your answer.

4. Schema Integration

Let S be a relational database with two tables. The first table is HOUSES, with attributes **location**, **price**, and **agent-id**. This table has two tuples:

("Atlanta, GA"; 360,000; 32)
("Raleigh, NC"; 430,000; 15).

The second table is AGENTS, with attributes **id**, **name**, **city**, **state**, and **fee-rate**. This table has two tuples:

(32; "Mike Brown"; Athens; GA; 0.03)
(15; "Jean Laup"; Raleigh; NC; 0.04)

Let T be another relational database with a single table LISTINGS. This table has attributes **area**, **list-price**, **agent-address**, and **agent-name**. It has two tuples:

("Denver, CO"; 550,000; "Boulder, CO"; "Laura Smith")
("Atlanta, GA"; 370,800; "Athens, GA"; "Mike Brown")

a) Suppose we want to copy all data from database S to database T. Write a single SQL query that when executed over database S would transform all data of S into the format of T. That is, the query would create tuples for table LISTINGS of T from the data in S.

b) In practice, writing such SQL queries to copy data from one database to another is very time consuming. To save time, a user can employ a schema matching tool (such as those described in the Rahm/Bernstein paper) to find semantic matches between S and T. Examples of such matches are: **location** = **area** and **name** = **agent-name**. The user then employs a tool such as Clio (described in the Rahm/Bernstein paper) to elaborate these matches into SQL queries (that can then be executed to copy data).

Using these tools, can the above process be completely automated? If yes, why? If not, why not? In that latter case, discuss at which points in the process the user must be involved, what the user must do, and why.

c) Suppose the user has copied data from database S to database T, and has also copied data from database T to another database U. While doing this, the user has established that attribute x of S matches y of T, and attribute y of T matches z of U. Can the user conclude that attribute x of S matches z of U? If yes, why? If not, why not?