**University of Wisconsin-Madison**
**Computer Sciences Department**

# Database Qualifying Exam
# Fall 2010

## 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** 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. 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: Datalog

Consider the following Datalog program:

```
t(X,Y) :- c(X,Y).
t(X,Y) :- a(X,W), t(W,Z), b(Z,Y).
```

We are omitting the facts for the EDB predicates **a**, **b**, and **c**.

a) Give the magic sets rewriting of this program for the query: `t(1,Y)`?

b) In addition to Magic Sets, people talk about Naive vs. Semi-Naïve evaluation as options for evaluating recursive Datalog programs. If we begin with Naive evaluation of the program above without the Magic Sets rewriting, we can make the evaluation more efficient by changing from the Naive evaluation of the program without the Magic Sets rewriting to the Semi-Naive evaluation of the program without the Magic Sets rewriting. We can also make the evaluation more efficient by moving from the Naive evaluation of the original program to the Naive evaluation of the program with the Magic Sets rewriting.

   Describe a data set (that is, values for **a**, **b**, and **c**) such that moving from the Naive evaluation of the original program to the Semi-Naive evaluation of the original program is expected to improve efficiency more than moving from the Naive evaluation of the original program to the Naive evaluation of the Magic Sets rewriting of the original program.

## 2: Buffer Management

a) For each pair of replacement policies (x,y) below, give two examples: (1) an example of a single buffer pool and sequence of accesses where there are fewer page faults using x than using y and (2) an example where there are fewer faults using y than using x.

a. LRU, MRU
b. LRU, LRU-2

LRU is Least Recently Used. MRU is Most Recently Used. LRU says that we should evict the page that was least recently used according to the most recent access. In contrast, LRU-2 says to evict the page that was least recently used according to the second most recent access, i.e., the access before the most recent.

b) In a paper in 1986, Gray and Putzulo formulated their famous five-minute rule:

*"Pages referenced every five minutes should be memory resident."*

In this question, your goal is to derive the formula behind this rule.

Suppose that a disk costs x dollars (with controller, etc) and delivers y accesses (pages) per second. Consider a workload that requires n access per second. To accommodate this workload one would need n/y disks at a total cost of x * n / y.

i. If a memory resident page P saves z access per second. Thus, if we choose to store 10z in memory, then we only need to buy enough disks to service n – 10z requests. How much money does keeping P in memory save? You may assume that you are able to purchase fractional disks.

ii. Suppose that memory costs m to hold a page in memory. Thus, for a cost of 2m we would save 2z access per second. Using this information, compute the "breakeven point"? That is, when does it save money to keep a page in memory as a function of z? Explain informally how one would use this equation to derive the five-minute rule.

iii. For a fixed disk, the page size is an important quantity in the above calculation. Describe how the above calculation changes as one varies the page size.

## 3: Hierarchical Optimistic Concurrency Control

Consider Kung and Robinson's optimistic concurrency control method. As presented in the paper, there is no notion of hierarchy of lockable objects like the one presented in the Gray "granularity of locks" paper. Your task in this question is to extend Kung and Robinson to accommodate such a hierarchy.

a) In Kung and Robinson's approach, an object is added to a transaction's read or write set when it is read or written. In your new hierarchy-aware approach, the read and write sets should contain objects of different levels of the hierarchy (e.g., the read or write set could contain records, or pages, or files.) Propose a way to maintain these read and write sets (that is, how do you decide whether to include records or pages? What exactly do you put in these sets? How do you maintain these read and write sets as transactions access more and more objects?)

b) Modify Kung and Robinson's rules for serial validation so that they will work with these new multi-granularity read and write sets.

c) Do you think the new approach will outperform the original approach? Why or why not? (Note: we are not looking for formulas here, just intuitive arguments.)

## 4. Join Algorithms

Consider two equijoin algorithms for joining tables R and S: the GRACE hash join, and sort-merge join. Furthermore, suppose that R and S can be partitioned in one pass, and that R and S can be sorted in two passes.

a) Give cost formulas from the Shapiro paper for these two algorithms.

b) Now focus on I/O. We want you to be more precise than Shapiro. Some of the I/Os will be random. Some will be sequential. Assume that you have only one disk, and write and explain the formulas for the number of random and sequential I/Os done by the two algorithms. Note that some reads that would otherwise be sequential will be random because they are interspersed with other I/Os from other parts of the algorithm. You can assume that the answer tuples generated are not stored to disk (so writes of the answer will not affect the random/sequential nature of I/Os during the join.)

c) Now consider a variant of sort-merge join, optimized as follows. Instead of generating the complete sorted versions of R and S, do the following:

1. generate initial sorted runs of R
2. generate initial sorted runs of S
3. simultaneously (merge the runs of R, merge the runs of S, generate answer tuples).

In slightly more detail, the intent of step 3 is that you never generate a fully sorted R or S; rather, instead of the final merge that would generate the fully sort R or S, you do the join. Do you think this last step (step 3) is possible? Why or why not? Assuming it is possible, what would be the number of random and sequential reads for this optimized sort-merge algorithm?

**5: String Matching**

This question is about string matching, a ubiquitous operation in many data integration applications.

a) Edit distance is a popular string distance score. The edit distance d(x,y) computes the minimal cost of transforming a string x into string y, by deleting a character, inserting a character, or substituting a character with another. The cost of each operation is 1. For example, d(madison,maddisom) = 2.
Briefly describe an efficient solution to compute the edit distance between two given strings x and y. You don't have to fully describe the algorithm; sketching out the key ideas is sufficient. What is the time and space complexity of your algorithm?

b) Two popular string similarity measures are the Overlap measure and the Jaccard measure. To compute these measures for two strings x and y, first we tokenize the

strings, say by generating all 2-grams (sequences of 2 consecutive characters) of the strings. Thus, string "madison" is tokenized into the set of 2-grams {"ma", "ad", "di", "is","so", "on"}.

Let Bx and By be the set of 2-grams generated for strings x and y, respectively. We then define

- the Overlap measure $O(x,y) = \mid Bx \cap By \mid$ and
- the Jaccard measure $J(x,y) = \dfrac{\mid Bx \cap By \mid}{\mid Bx \cup By \mid}$

where |S| is the number of elements in set S. Prove for t in the range [0,1]:

$$J(x,y) \geq t \Leftrightarrow O(x,y) \geq \frac{t}{1+t}(\mid Bx \mid + \mid By \mid)$$