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

# Database Qualifying Exam
**Fall 2012**

## 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

**You must answer four (4) of 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: HIGH PERFORMANCE CONCURRENCY CONTROL

Consider a high-performance main memory database that runs on a modern multicore machine. So, the entire database is resident in main memory and there are multiple cores that access the database using a shared-memory architecture. Design a high-performance optimistic concurrency protocol for accessing B-Trees in this environment. Also, design a high-performance pessimistic (i.e. locking-based) scheme for this environment. Then, list at least one workload for each scheme where that scheme works better on that workload as compared to the other scheme.

## 2: MODERN BUFFER MANAGER

Traditional hard disk drives are rapidly being replaced by flash storage where the cost of random access is nearly the same as that of sequential access. Assume you have a database management system that is designed purely to work on flash-only storage systems (i.e. there is no need to optimize any part of the system for rotating disks). Design a high performance buffer manager for this system.

Now, traditional buffer mangers (for rotating disk systems) generally use an LRU-based replacement policy and often scan the buffer pool to find victim pages for eviction that are sequentially laid out on disk. Traditional buffer managers also use prefetching to fetch a small set of sequential pages (e.g. 8 pages) when servicing a buffer miss on a read access. Explain how your scheme above compares to a traditional buffer manager design.

## 3: ENTITY LINKING

Let Persons(first-name, last-name, street-address, city, state, zip-code, phone) be a table with 10 million tuples. Each tuple describes a person and may contain typos, mistakes, variations, and missing data. For example, the last name "Richard" may be misspelled as "Rihard" or shortened into "Rick", or may be missing from the tuple.

Now suppose you want to find all pairs of tuples that match, that is, pairs that refer to the same real-world person. This problem is known as entity matching or record linkage, among other names, in the literature.

1. Describe an algorithm that runs on a single machine (e.g., a PC) to find all matching pairs from Table Persons. Your algorithm should try to maximize the matching accuracy and minimize the matching time.

2. Describe how you measure the matching accuracy. Give the exact definitions of the accuracy measures that you use.

3. Describe an algorithm that runs on a cluster of machines to find all matching pairs from Table Persons, in a distributed and parallel fashion.

## 4: THEORY

In this question, you will be asked to prove some facts about conjunctive queries. If you cannot prove a statement formally, don't fret too much: you will get close to full credit by identifying the key issue informally. Recall the containment problem:

Given as input two inputs q and q' in some language (relational algebra or conjunctive queries). We denote by q(I) the set of answers returned by q when applied to I. We say that q is contained in q' if for all instances I q(I) <= q'(I).

That is the answers of q are always a subset q' no matter what input database they are applied to. For example, consider q and q'

q(x) :- R(x),*S*(x) and q'(x) :- R(x)

Here, q is contained in q', and q' is not contained in q.

Let **CQ** denote the set of conjunctive queries without constants or inequalities.

a. Suppose someone gives you a function **F** that correctly decides containment, i.e., given a pair (q,q') it returns true if q is contained in q' and false otherwise. How would you use the function **F** to decide if q is equivalent to q'?

b. This question deals with containment with constraints. Suppose you have three queries q1, q2, and q3 such that q1 is contained in q2, but q1 is not contained in q3.

   Fix a relation T(x,y) and let IFD be the set of instances I such that T satisfies the functional (key) dependency x → y. Which of the statements can you conclude (and why or why not):

   (i) for all I in IFD q1(I) <= q2(I)?

   (ii) there exist an I in IFD such that q2(I) is not a subset of q3(I)?

c. For q,q' in **CQ**, recall from Aho, Sagiv, and Ullman paper that it is NP-Complete to decide whether q is contained in q' -- even if q and q' are Boolean queries (with no variables in the head of the query). One proof of this statement uses the idea of a *canonical database,* where we construct a database D from the query q such that if q' is true on D, then q is contained in q'. This suggests that answering a query on a database is NP-Complete. On the other hand, every day relational databases across the globe efficiently answer conjunctive queries (and more!). Explain this seeming contradiction.

## 5: PARALLEL RDBMS

Suppose you have been given the task of building a parallel relational DBMS, but instead of using a traditional storage manager on a shared-nothing cluster upon which to build the system, you are given a distributed key-value store on a cluster. This key-value store does what the name implies: you give it pairs (key, value), and it will store them; you can retrieve or modify or delete the value by presenting the key to the key-value store. This key-value store is distributed so any (key, value) pair can be read from any node (there is no explicit notion of the "location" of the pair in the system.) For reliability, this key value store saves three copies of each (key, value) stored in the system, and makes sure they are all stored at different nodes in a cluster. For updates it provides "eventual consistency", meaning that if no new updates arrive, eventually the three replicas will converge to the same value.

Your task in this question is to speculate on tradeoffs between a traditional parallel RDBMS (like GAMMA) and this new "parallel RDBMS on top of a key-value store." You can pick an area to focus on – e.g., query evaluation, concurrency control, etc. If you feel you need to make additional assumptions for your answer, feel free to do so, but make your assumptions explicit.

Note that this is a very open-ended question, and it is only one of five questions on this exam. So watch your time, and try to focus on the tradeoffs that best illustrate the differences in the two approaches to building a parallel relational database management system.