University of Wisconsin-Madison
Computer Sciences Department

Database Qualifying Exam
Spring 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

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. Parallel DBMS Joins

Consider a join of Customers(custKey, custName, …) and Orders(orderKey, custKey, orderName, …) on custKey. Suppose that there is one customer (the "big" customer) who has far more orders than any other customer. Finally, suppose that we are executing this join on a parallel database system with *n* nodes using a parallel hash join. Initially, Customers is partitioned on custKey and Orders is partitioned on orderKey.

(a) Joe Qualtaker is disappointed to find that the join performance is worse than he expected – in particular, it is substantially more than 1/*n**(single node time.) The problem is skew – explain what this is and why it harms parallel performance.
(b) Joe Qualtaker decides to fix the problem by buying an additional *n* nodes, so now he has a 2*n* node system. He runs the join again. Would you expect a performance improvement? Why or why not?
(c) Joe Qualtaker decides to implement a new skew-handling parallel hash join. He has an insight that he should split the tuples of Orders into two classes: those of the big customer, and all the rest. (Hint to the real qualtaker: this is actually a very good insight.) He decides to do the join by hash partitioning the non-skewed rows of Orders, just like the "normal" parallel hash join; but to leave the "skewed" rows (those of the big customer) in place. How does he need to distribute the Customers tuples for this to be a correct algorithm?
(d) Do you expect your algorithm from (c) to effectively handle the skew? Why or why not?
(e) Now consider a case where both input tables to the join are skewed. That is, we are joining R and S, and both R and S have skewed distributions on the join key. Can you think of a generalization of the algorithm from (c) that would work?

2. ARIES Recovery

In the ARIES protocol, there are three phases: ANALYSIS, REDO, and UNDO.

(a) During the Analysis phase, the dirty page table (DPT) is constructed. In the standard ARIES algorithm, the DPT that is computed during the ANALYSIS phase may be different than the DPT at the time of the crash. What is the relationship between the contents of the DPT at the end of the analysis phase and its contents at the time of the crash?

(b) Joe Gradstudentnomore's boss hates the imprecision in the ANALYSIS phase. To make Joe's boss happy, what extra information could Joe store in the log to minimize the difference between the contents of the DPT at the end of Analysis and the DPT at the time of a crash? What are the pros and cons of your solution?

(c) Joe Gradstudentnomore notices that in his implementation of ARIES the REDO phase takes a long time. In a panic, Joe calls you. To help Joe, name two optimizations that an RDBMS could do to bound the work in REDO.

(d) Joe Gradstudentnomore is given fancy new hardware, "dirt memory," that has speed and byte addressability like DRAM, a price like dirt (that is, very cheap), and dirt memory stores every page modification durably across crashes. Dirt memory is so cheap and reliable that Joe's new database appliance's memory hierarchy consists only of standard processor registers, caches, and a PBs of dirt memory. Does Joe still need recovery code to guarantee ACID properties? Which phases of ARIES does Joe need to perform? Does the WAL change?

3. Data Integration

Consider the following problem: Given *n* relational databases $D_1, D_2, ..., D_n$, build a search engine over the databases such that given a keyword query Q, the engine returns a ranked list of answers, where each answer can span multiple databases.

For example, suppose we have only two databases D_1 and D_2. Suppose further that D_1 has just one table EMPLOYEES(eid, name, dept), and that D_2 has just one table

CUSTOMER-COMPLAINTS(cid, emp_name, cust_location).

Then to find out if any customer in Madison has complained about any employee in the Electronics Department, we may pose a keyword query "madison electronics." When executed over D_1 and D_2, this query may return an answer that links a tuple (eid=12, name=David Smith, dept=electronics) in D_1 with a tuple (cid=G2, emp_name=D. Smith, cust_location=Madison, WI) in D_2. By examining this answer, we may be able to determine that employee David Smith in electronics has indeed been complained about by a customer in Madison.

a) Provide a ***formal*** definition of the above problem – that is, a definition that specifies precisely the input and the output, including the notion of scoring the answers, and anything else you feel needs specification.

b) Discuss the key challenges of the above problem and briefly discuss how you would address them.

4. Information Retrieval

Suppose you are working at Twitter and you have been charged with developing a keyword search engine for finding tweets, which as you know are limited to 140 characters or less. So, given a query such as "jobs memorial," given the current twitter traffic, it should return tweets that are related to the Steve Jobs memorial.

a) Discuss how the above problem differs from the keyword search problem over Web pages.

b) Briefly discuss a solution to the above problem. If you are using any TF/IDF-like notion, discuss how that is computed.

5. PCM Indexing

Consider the new class of Storage Class Memories (SCM), such as Phase Change Memory, which allow persistent data to byte-addressable memory, with low latency access time. So, in this environment a program can access only a few bytes of memory orders of magnitude faster than data on traditional disks, and there is little difference between sequential and random I/O. In contrast, hard disks drives provide block-access (that is, you access data on disks in page-sized chunks) and the cost of a random I/O is typically much larger than the cost of a sequential I/O.

Consider designing a high-performance B+-tree index structure for a system that uses SCM as the persistent store instead of hard disk drives. What aspects of a B+-tree index would you change, if any, for SCM compared to disk-based indices? You can assume that the disk-based indexing uses traditional 8KB page size, with the Lehman and Yao B-link protocol.

Your design should clearly specify the impact of your design on:

(a) Primary key index lookups (with the index on the primary key)
(b) Range search;
(c) Single tuple update performance;
(d) Concurrency Control