University of Wisconsin-Madison
Computer Sciences Department

Database Qualifying Exam
Spring 2009

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. **Concurrency Control**

(a) In the Gray et al. "granularity of locking" paper, the authors discuss intention locking in the presence of a hierarchy. Joe QualTaker interprets the paper to mean "to get an X lock on a node in the hierarchy, get IX or X on all of its parents; to get an S lock on a node, get IS or S on one of its parents."

Consider the lock hierarchy:

$$\begin{tikzcd}
& \text{DB} \arrow[dl] \arrow[dr] & \\
\text{File} \arrow[d] & & \text{Index} \arrow[ddl] \\
\text{Page} \arrow[dr] & & \\
& \text{Record} &
\end{tikzcd}$$

Joe interprets this to mean that he can get the locks IX on DB, and X on Index (since DB has no parent, and Index has only one parent, DB, which is locked in IX mode.) What is wrong with his interpretation? (Give an example of a problem that could arise.) How should it be fixed? (Try to correct Joe's confusion so he won't get into trouble again.)

(b) Suppose T1 and T2 both hold S locks on a record, and both decide to upgrade their locks to X locks. Discuss two approaches to doing so:

i) drop the S lock, ask for an X lock
ii) while holding the S lock, request an X lock

Discuss the pros and cons of both. Note: this is not a question about intension locks.

(c) It is possible in the process of running transactions that two or more transactions could become deadlocked. In such a case the DBMS must kill one of the transactions involved in the deadlock. Propose two policies for deciding which transaction should be killed, and for each discuss the performance implications. Note: you can make up your own policy, no need to mention any that has been proposed in the literature.

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. **ORDBMSs**

Consider the following two relations in an ORDBMS that describe courses taken by students and courses taught by instructors in a specific semester:

Student (name, {takes-courses}) and *Instructor (name, {teaches-courses})*

In the Student relation, takes-courses is a set of course-ids that the student has taken. In the Instructor relation, teaches-courses is a set of course-ids that are taught by instructors. Now, consider the following query:

*Select * from Student S, Instructor I where I.teaches-courses* $\subseteq$ *S.takes-courses*

This query finds all students-instructor pairs such that the student has taken all the courses that are taught by the instructor. Such joins are called "set containment joins", and can also be useful for evaluating association rules.

(a) Design an algorithm to evaluate this operation efficiently. Your method should work when there is not enough main memory to hold either of the two relations in memory. (If you make some assumptions about how the data of the tables *Student* and *Instructor* is laid out on disk, then state those assumptions clearly.)

(b) Describe the effect of the cardinality of the nested set (takes-courses) on the performance of your algorithm.

4. **Parallel Databases**

The following question deals with how to implement SQL analytics in a parallel DBMS. If you are wondering how you missed parallel SQL analytics in the reading list, don't worry - it is nowhere in the reading list! We will describe enough about them for you to answer this question.

Suppose we have a table:

```
emp( empno,    deptno,          sal)
------------------------------------------------------------
         1      10          100,000
         2      10          100,000
         3      10          140,000
         4      10          200,000
```

```
5      20            75,000
6      20           100,000
7      20           100,000
8      20           150,000
```

We consider analytic queries that make use of DENSE_RANK() and PARTITION BY. Here is a sample:

```
SELECT empno, deptno, sal,
DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal) AS dense_rank
FROM emp
```

The output of this query on the toy example data will be:

```
empno    deptno   sal       dense_rank
1        10      100,000     1
2        10      100,000     1
3        10      140,000     2
4        10      200,000     3
5        20      75,000      1
6        20      100,000     2
7        20      100,000     2
8        20      150,000     3
```

That is, PARTITION BY functions rather like GROUP BY, except that the rows are not really combined (there is one row in the output of the PARTITION BY for each row in the input of the PARTITION BY). Then the rows are ordered within each partition, and ranked. (If you are curious about the "dense" part of the dense rank, there is also a "rank" command that leaves gaps. So, for example, where we have (1, 1, 2) with dense rank, we would have (1, 1, 3) with rank. But we are not asking about "rank" in this question!)

Suppose that you are given a shared-nothing parallel database system consisting of n nodes, each of which is a (uniprocessor) DBMS that has already implemented the analytical functions PARTITION BY, DENSE_RANK(), AVERAGE(), SUM(), and COUNT() (you probably will not need all of these.) Your task is to use these uniprocessor systems as building blocks to build a parallel solution to these queries. You can also assume that the shared-nothing parallel system provides repartitioning operations that support range, hash, and round-robin partitioning.

Assume that emp is initially hash partitioned on empno, give a parallel algorithm to answer the query. For full credit your solution should deal with situations in which the number of partitions is less than n, and also situations where the number of partitions is much greater than n.

5. **Schema Integration**

(a) Why is schema matching hard?

(b) Let S and T be the two schemas of two relational databases, respectively. In many scenarios, given an attribute *a* of schema S, a user wants to obtain a list of attributes in schema T that are likely to match *a*, ranked in the order of decreasing likelihood. This problem can potentially benefit from information retrieval (IR) techniques. For example, we can view attribute *a* as a keyword query being posed over a document collection.

If IR techniques are not appropriate in this context, then argue why. Otherwise, design an algorithm that employs IR techniques to solve the above problem. Describe clearly what your document collection is, how you process the documents (e.g., do you do stemming, stop-word removal, etc., and why?), what your keyword query is, and how you compute the similarity score between the query and the documents. Argue why the algorithm would be effective, and discuss its limitations.