UNIVERSITY OF WISCONSIN - MADISON
COMPUTER SCIENCES DEPARTMENT

DATABASE QUALIFYING EXAM

FALL 1995

Answer any *five* (5) of the following *six* (6) questions. Before beginning to answer a question, be sure to read it carefully. If you are confused about what the question means, state any assumptions you have made in formulating your answer. *Good luck*!

1 BITMAPPED INDICES

Consider the following two implementations of the leaf level of a B^+-tree: (a) For each value V_i in the indexed column, there is a linked list of tuple identifiers (tids), pointing to the tuples in the relation having V_i in their corresponding column (*traditional index*). (b) For each value V_i, there is a bitmap with as many bits as there are tuples in the relation; the ith bit is 1 if the ith tuple in the relation has V_i in its corresponding column (*bitmapped index*).

1. Indicate when representation (b) is to be preferred over the more traditional representation (a), and what its benefit is in those cases.

2. Describe an effective mapping between integers (bit positions) in the bitmap and the tuples (tids) indexed in the relation when the relation can hold (i) fixed-length tuples only, and (ii) variable-length tuples.

3. Consider relation $R(X, Y, Z)$ whose attributes X and Y are indexed using bitmapped indices. Describe an effective way to process the following query, where C_x and C_y are constants:

   select Z
   from R
   where $X = C_x$ **and** $Y = C_y$

4. Repeat the previous question for the following query, where C_1 and C_2 are constants:

   select Z

from R
where $X = C_1$ **or** $X = C_2$

5. Various flavors of so called *join indices* have been shown to be effective in processing queries involving joins. For example, consider the A_1=A_2 join between two relations $R_1(A_1,B_1)$ and $R_2(A_2,B_2)$. One can have a join index with A_1, or B_1, or even R_1-tids as its key; its leaves will pair up key values with tids of R_2 tuples that join with tuples of R_1 with the given key value. Explain how join indices can use the bitmapped representation.

6. Consider relation $R_0(A_0,A_1,A_2,A_3)$ and relations $R_i(A_i,B_i)$, $1 \leq i \leq 3$. For each $0 \leq i \leq 3$, column A_i is the key. Consider the following query, where C_i, $1 \leq i \leq 3$, are constants:

   select $R_0.A_0$, $R_1.A_1$, $R_2.A_2$, $R_3.A_3$
   from R_0, R_1, R_2, R_3
   where $R_0.A_1 = R_1.A_1$ **and** $R_0.A_2 = R_2.A_2$ **and** $R_0.A_3 = R_3.A_3$ **and**
   $R_1.A_1 = C_1$ **and** $R_2.A_2 = C_2$ **and** $R_3.A_3 = C_3$

   Assume that R_0 contains 100 million tuples, the selectivity of each selection is 1%, the selections are statistically independent. Specify a set of conventional and/or join indices using the bitmapped representation that can be used for a very effective way to process this query.

2 DEPENDENCIES

1. Consider a relation R with five attributes ABCDE. For each of the following instances of R, (i) does it violate the functional dependency BC $\longrightarrow$ D? (ii) does it violate the multivalued dependency BC $\rightarrow\!\!\rightarrow$ D?

   (a) { } (i.e., empty relation)
   (b) {(a,2,3,4,5), (2,a,3,5,5), (a,2,3,4,6)}
   (c) {(a,2,3,4,5), (2,a,3,4,5), (a,2,3,6,5), (a,2,3,6,6)}
   (d) {(a,2,3,4,5), (a,2,3,6,5), (a,2,3,6,6), (a,2,3,4,6)}

2. Suppose that each of the four instances for R listed above is legal, i.e., it can occur in correct, real instances. What can you say about the functional dependency A $\longrightarrow$ B?

3. Consider a relation with attributes *supplier*, *part* and *project*, denoted SPJ, with no dependencies except the following constraint: the set of relation schemes SP, PJ and JS is a lossless join decomposition of SPJ. Construct an instance of SPJ which illustrates that no two of these schemes serves as a lossless join decomposition of SPJ.

4. Can you give a sufficient condition for a relation to be in 4NF that checks only FDs?
5. Can you give a sufficient condition for a relation to be in 5NF that checks only FDs?

3 PARALLEL DBMS

Over the past several years, vendors such as Sequent, Sun, and HP have found it difficult to effectively scale their pure shared-memory systems beyond about 10-20 processors, especially for database applications. Consequently they are turning to clusters of shared-memory computers. Basically this corresponds to a shared-nothing architecture in which each node, instead of being a single processor, consists of a small (e.g. 4-16 processor) shared-memory multiprocessor.

1. Assume that you work for Informix. Their parallel database product (PDB) employs Gamma-style parallelism and assumes a pure-shared nothing architecture.
   (a) Discuss what changes would you make to take advantage of the hybrid architecture described above. Consider query processing, optimization, concurrency control, and recovery issues.
   (b) How is your recommended course of action affected by the number of processors in each shared-memory unit?
   (c) How is your recommended course of action affected by the speed of the interconnection network and the primitives provided for communications?
2. Assume that you work for Oracle on their parallel database product. This product has been designed for a shared-memory architecture. All key data structures (such as the buffer pool, lock table, log tail, ...) reside in shared memory. You are responsible for adopting the system to work on the same cluster of shared-memory multiprocessors as your colleague at Informix. The future of Oracle depends on how you architect this evolution.
   (a) Discuss what changes would you make to take advantage of the hybrid architecture described above. Consider query processing, optimization, concurrency control, and recovery issues.
   (b) How is your recommended course of action affected by the number of processors in each shared-memory unit?
   (c) How is your recommended course of action affected by the speed of the interconnection network and the primitives provided for communications?
3. Which company do you recommend buying stock in??

4 SPATIAL ACCESS METHODS

With the increasing interest in geographic information systems, R-Trees and their variants have become very popular spatial indexing methods.

1. Describe the R-tree indexing structure
2. Give a simple data set with 4-5 rectangles and show the corresponding R-tree.
3. What criteria is used to select which leaf node to insert a rectangle into?
4. Is there generally more than one node to choose from?
5. Since R-trees are derived from B-trees, how can this be? What accounts for the difference?
6. What criteria do the authors propose to distinguish between good and bad splitting algorithms? Can you suggest another criteria?
7. Explain the "quadratic cost" algorithm for splitting a node into two nodes.

5 DEEP QUERY PLANS

A "deep" query plan is one in which for every join, one input is a stored table (not the result of another join). In a "left-deep" plan, the inner input of each join is a stored table, and in a "right-deep" plan, the outer input of each join is a stored table. A query plan which is not deep is called "bushy".

1. Many if not most systems implement only left-deep plans. Give a justification for this design decision.
2. In a system that supports only hash join, what is the maximum number of distinct left-deep trees for a query of n relations? Explain how you derive your answer.
3. Give a scenario in which a bushy tree might outperform a left-deep tree.
4. In a system with only sort-merge joins and no indices, how does the left-deep plan ((A equijoin B) equijoin C) differ in cost from the right-deep plan (A equijoin (B equijoin C))? What about in a system with only nested-loop joins and no indices? Assume no temps are stored for the nest-loop joins. (You need not give cost equations for these plans unless it helps you. All that's required is a discussion of how the queries are processed and how this affects their relative costs.)

6 POINTER JOINS

Object-Relational Database Systems support references as a data type. It is often claimed that using references instead of key-foreign key relationships will give much higher performance for joins. This question asks you to explore this issue.

1. Suppose we have the DDL

   ```
   create table R(
       rkey: integer;
       A: REFERENCE to S;
   );

   create table S(
       skey: integer;
       otherdata: integer;
   );
   ```

   Now consider the query:

   ```
   select R.rkey, r.A.skey
   from R;
   ```

   What algorithm would you suggest to evaluate this pointer join? How would you expect it to perform compared to a straight relational join of the same two relations? For the straight relational join, you may assume that the DDL is

   ```
   create table R(
       rkey: integer;
       A: integer; // stores a foreign key from S
   );

   create table S(
       skey: integer;
       otherdata: integer;
   );
   ```

2. Suppose now we have

```
create table R(
    rkey: integer;
    A: SET OF REFERENCE to S;
);

create table S(
    skey: integer;
    otherdata: integer;
);
```

That is, now we have a many-to-one relationship from R to S. Consider the query:

```
select R.rkey, r.{A}.skey
from R;
```

where now "r.{A}.skey" means to follow *every pointer* in the set of references in the set r.{A}. What algorithm would you use to evaluate this join? How does this compare to the equivalent join in a straight relational framework? (Here state any assumptions you make about how the tables are defined for the straight relational equivalent.)