Introduction to Database Systems

Problem Set 4

Due: March 11, 2016 at 11:59 PM

Complete exercises 12.2, 12.6, 15.7, 16,5, and 17.4 from the book. For convenience the questions are below.

Exercise 12.2 Consider a relation R(a,b,c,d,e) containing 5,000,000 records, where each data page of the relation holds 10 records. R is organized as a sorted file with secondary indexes. Assume that $R.a$ is a candidate key for R, with values lying in the range 0 to 4,999,999, and that R is stored in $R.a$ order. For each of the following relational algebra queries, state which of the following three approaches is most likely to be the cheapest:

- Access the sorted file for R directly.
- Use a (clustered) B+ tree index on attribute $R.a$.
- Use a linear hashed index on attribute $R.a$.

1. $\sigma_{a<50{,}000}(R)$
2. $\sigma_{a=50{,}000}(R)$
3. $\sigma_{a>50{,}000 \wedge a<50{,}010}(R)$
4. $\sigma_{a\neq 50{,}000}(R)$

Exercise 12.6 Consider the two join methods described in Section 12.3.3. Assume that we join two relations R and S, and that the systems catalog contains appropriate statistics about R and S. Write formulas for the cost estimates of the index nested loops join and sort-merge join using the appropriate variables from the systems catalog in Section 12.1. For index nested loops join, consider both a B+ tree index and a hash index. (For the hash index, you can assume that you can retrieve the index page containing the rid of the matching tuple with 1.2 I/Os on average.)

Exercise 15.7 Consider the following relational schema and SQL query. The schema captures information about employees, departments, and company finances (organized on a per department basis).

Emp(*eid:* `integer`, *did:* `integer`, *sal:* `integer`, *hobby:* `char(20)`)
Dept(*did:* `integer`, *dname:* `char(20)`, *floor:* `integer`, *phone:* `char(10)`)
Finance(*did:* `integer`, *budget:* `real`, *sales:* `real`, *expenses:* `real`)

Consider the following query:

```
SELECT D.dname, F.budget
FROM   Emp E, Dept D, Finance F
WHERE  E.did=D.did AND D.did=F.did AND D.floor=1
       AND E.sal ≥ 59000 AND E.hobby = 'yodeling'
```

1. Identify a relational algebra tree (or a relational algebra expression if you prefer) that reflects the order of operations a decent query optimizer would choose.

2. List the join orders (i.e., orders in which pairs of relations can be joined to compute the query result) that a relational query optimizer will consider. (Assume that the optimizer follows the heuristic of never considering plans that require the computation of cross-products.) Briefly explain how you arrived at your list.

3. Suppose that the following additional information is available: Unclustered B+ tree indexes exist on *Emp.did*, *Emp.sal*, *Dept.floor*, *Dept.did*, and *Finance.did*. The system's statistics indicate that employee salaries range from 10,000 to 60,000, employees enjoy 200 different hobbies, and the company owns two floors in the building. There are a total of 50,000 employees and 5,000 departments (each with corresponding financial information) in the database. The DBMS used by the company has just one join method available, index nested loops.

   (a) For each of the query's base relations (Emp, Dept, and Finance) estimate the number of tuples that would be initially selected from that relation if all of the non-join predicates on that relation were applied to it before any join processing begins.

   (b) Given your answer to the preceding question, which of the join orders considered by the optimizer has the lowest estimated cost?

Exercise 16.5 Suppose that a DBMS recognizes *increment*, which increments an integer-valued object by 1, and *decrement* as actions, in addition to reads and writes. A transaction that increments an object need not know the value of the object; increment and decrement are versions of blind writes. In addition to shared and exclusive locks, two special locks are supported: An object must be locked in I mode before incrementing it and locked in D mode before decrementing it. An I lock is compatible with another I or D lock on the same object, but not with S and X locks.

1. Illustrate how the use of I and D locks can increase concurrency. (Show a schedule allowed by Strict 2PL that only uses S and X locks. Explain how the use of I and D locks can allow more actions to be interleaved, while continuing to follow Strict 2PL.)

2. Informally explain how Strict 2PL guarantees serializability even in the presence of I and D locks. (Identify which pairs of actions conflict, in the sense that their relative order can affect the result, and show that the use of S, X, I, and D locks according to Strict 2PL orders all conflicting pairs of actions to be the same as the order in some serial schedule.)

Exercise 17.4 Consider the following sequences of actions, listed in the order they are submitted to the DBMS:

- **Sequence S1:** T1:R(X), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit
- **Sequence S2:** T1:R(X), T2:W(Y), T2:W(X), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

For each sequence and for each of the following concurrency control mechanisms, describe how the concurrency control mechanism handles the sequence.

Assume that the timestamp of transaction Ti is i. For lock-based concurrency control mechanisms, add lock and unlock requests to the previous sequence of actions as per the locking protocol. The DBMS processes actions in the order shown. If a transaction is blocked, assume that all its actions are queued until it is resumed; the DBMS continues with the next action (according to the listed sequence) of an unblocked transaction.

1. Strict 2PL with timestamps used for deadlock prevention.
2. Strict 2PL with deadlock detection. (Show the waits-for graph in case of deadlock.)
3. Conservative (and Strict, i.e., with locks held until end-of-transaction) 2PL.
4. Optimistic concurrency control.
5. Timestamp concurrency control with buffering of reads and writes (to ensure recoverability) and the Thomas Write Rule.
6. Multiversion concurrency control.