CHAPTER 24

Advanced Application Development

Practice Exercises

24.1 Many applications need to generate sequence numbers for each transaction.

a. If a sequence counter is locked in two-phase manner, it can become a concurrency bottleneck. Explain why this may be the case.

b. Many database systems support built-in sequence counters that are not locked in two-phase manner; when a transaction requests a sequence number, the counter is locked, incremented and unlocked.

i. Explain how such counters can improve concurrency.

ii. Explain why there may be gaps in the sequence numbers belonging to the final set of committed transactions.

Answer: If two-phase locking is used on the counter, the counter must remain locked in exclusive mode until the transaction is done acquiring locks. During that time, the counter is unavailable and no concurrent transactions can be started regardless of whether they would have data conflicts with the transaction holding the counter.
If locking is done outside the scope of a two-phase locking protocol, the counter is locked only for the brief time it takes to increment the counter. SInce there is no other operation besides increment performed on the counter, this creates no problem except when a transaction aborts. During an abort, the counter *cannot* be restored to its old value but instead should remain as it stands. This means that some counter values are unused since those values were assigned to aborted transactions. To see why we cannot restore the old counter value, assume transaction T_a has counter value 100 and then T_b is given the next value, 101. If T_a aborts and the counter were restored to 100, the value 100 would be given to some other transaction T_c and then 101 would be given to

a second transaction T_d. Now we have two non-aborted transactions with the same counter value.

24.2 Suppose you are given a relation $r(a, b, c)$.

a. Give an example of a situation under which the performance of equality selection queries on attribute a can be greatly affected by how r is clustered.

b. Suppose you also had range selection queries on attribute b. Can you cluster r in such a way that the equality selection queries on $r.a$ and the range selection queries on $r.b$ can both be answered efficiently? Explain your answer.

c. If clustering as above is not possible, suggest how both types of queries can be executed efficiently by choosing appropriate indices.

Answer:

a. If r is not clustered on a, tuples of r with a particular a-value could be scattered throughout the area in which r is stored. This would make it necessary to scan all of r. Clustering on a combined with an index, would allow tuples of r with a particular a-value to be retrieved directly.

b. This is possible only if there is a special relationship between a and b such as "if tuple t_1 has an a-value less than the a-value of tuple t_2, then t_1 must have a b-value less than that of t_2". Aside from special cases, such a clustering is not possible since the sort order of the tuples on a is different from the sort order on b.

c. The physical order of the tuples cannot always be suited to both queries. If r is clustered on a and we have a B^+-tree index on b, the first query can be executed very efficiently. For the second, we can use the usual B^+-tree range-query algorithm to obtain pointers to all the tuples in the result relation, then sort those those pointers so that any particular disk block is accessed only once.

24.3 Suppose that a database application does not appear to have a single bottleneck; that is, CPU and disk utilization are both high, and all database queues are roughly balanced. Does that mean the application cannot be tuned further? Explain your answer.
Answer: it may still be tunable. Example using a materialized view may reduce both CPU and disk utilization

24.4 Suppose a system runs three types of transactions. Transactions of type A run at the rate of 50 per second, transactions of type B run at 100 per second, and transactions of type C run at 200 per second. Suppose the mix of transactions has 25 percent of type A, 25 percent of type B, and 50 percent of type C.

a. What is the average transaction throughput of the system, assuming there is no interference between the transactions?

b. What factors may result in interference between the transactions of different types, leading to the calculated throughput being incorrect?

Answer:

a. Let there be 100 transactions in the system. The given mix of transaction types would have 25 transactions each of type A and B, and 50 transactions of type C. Thus the time taken to execute transactions only of type A is 0.5 seconds and that for transactions only of type B or only of type C is 0.25 seconds. Given that the transactions do not interfere, the total time taken to execute the 100 transactions is $0.5 + 0.25 + 0.25 = 1$ second. i.e, the average overall transaction throughput is 100 *transactions per second*.

b. One of the most important causes of transaction interference is lock contention. In the previous example, assume that transactions of type A and B are update transactions, and that those of type C are queries. Due to the speed mismatch between the processor and the disk, it is possible that a transaction of type A is holding a lock on a "hot" item of data and waiting for a disk write to complete, while another transaction (possibly of type B or C) is waiting for the lock to be released by A. In this scenario some CPU cycles are wasted. Hence, the observed throughput would be lower than the calculated throughput.
Conversely, if transactions of type A and type B are disk bound, and those of type C are CPU bound, and there is no lock contention, observed throughput may even be better than calculated.
Lock contention can also lead to deadlocks, in which case some transaction(s) will have to be aborted. Transaction aborts and restarts (which may also be used by an optimistic concurrency control scheme) contribute to the observed throughput being lower than the calculated throughput.
Factors such as the limits on the sizes of data-structures and the variance in the time taken by book-keeping functions of the transaction manager may also cause a difference in the values of the observed and calculated throughput.

24.5 List some benefits and drawbacks of an anticipatory standard compared to a reactionary standard.
Answer: In the absence of an anticipatory standard it may be difficult to reconcile between the differences among products developed by various organizations. Thus it may be hard to formulate a reactionary standard without sacrificing any of the product development effort. This problem has been faced while standardizing pointer syntax and access mechanisms for the ODMG standard.

On the other hand, a reactionary standard is usually formed after extensive product usage, and hence has an advantage over an anticipatory standard - that of built-in pragmatic experience. In practice, it has been found that some anticipatory standards tend to be over-ambitious. SQL-3 is an example of a standard that is complex and has a very large number of features. Some of these features may not be implemented for a long time on any system, and some, no doubt, will be found to be inappropriate.