CHAPTER 25

# Advanced Application Development

Solutions for the Practice Exercises of Chapter 25

## Practice Exercises

**25.1**

**Answer:**

- Postgresql: The *EXPLAIN* command lets us see what query plan the system creates for any query. The numbers that are currently quoted by *EXPLAIN* are:
  a. Estimated startup cost
  b. Estimated total cost
  c. Estimated number of rows output by this plan node
  d. Estimated average width of rows
- SQL: There is a Microsoft tool called *SQLProfiler*. The data are logged, and then the performance can be monitored. The following performance counters can be logged.
  a. Memory
  b. Physical disk
  c. Process
  d. Processor
  e. SQLServer:Access Methods
  f. SQLServer:Buffer Manager
  g. SQLServer:Cache Manager
  h. SQLServer:Databases

i. SQLServer:General Statistics
j. SQLServer:Latches
k. SQLServer:Locks
l. SQLServer:Memory Manager
m. SQLServer:SQL Statistics
n. SQLServer:SQL Settable

**25.2**

**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 $T_a$ aborted 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.

**25.3**

**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.

**25.4**

**Answer:**

a. Index update can be expensive, with potentially 1 I/O per record inserted, which could take 10 msec with magnetic disks. It is often cheaper to perform a bottom up build to recreate the index if there are a large enough number of inserts.

b. If the inserts are provided as a batch to the database system, for example via bulk-load utilities, the database can sort the records in index order, and then perform a merge with the leaf level of the $B^+$-tree. This would greatly reduce the I/O overhead.

c. With an LSM tree, there wouldn't be any significant performance penalty since the LSM tree is already designed to reduce the number of I/O operations required for insertion. The underlying techniques are similar to the solution for part (b), since they are based on merging of leaf-level entries of different $B^+$-trees.

**25.5**

**Answer:**

It may still be tunable at a higher level. For example, by creating an index or a materialized view, which may reduce both CPU and disk utilization significantly. Further, caching may be done above the database layer, to reduce the load on the database.

**25.6**

**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 bookkeeping functions of the transaction manager may also cause a difference in the values of the observed and calculated throughput.

**25.7**

**Answer:**
FILL IN

**25.8**

**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.

**25.9**

**Answer:**
This can be done using referrals. For example an organization may maintain its information about departments either by geography (i.e. all departments in a site of the the organization) or by structure (i.e. information about a department from all sites). These two hierarchies can be maintained by defining two

different schemas with department information at a site as the base information. The entries in the two hierarchies will refer to the base information entry using referrals.