p6q8 GMB

Databases 2003

~~Answer to Question 2~~

1. *Define the ACID properties of a transaction.* *[6 marks]*

   There are four important properties of transactions that a DBMS must ensure to maintain data in the face of concurrent access and system failure: Atomicity, Consistency, Isolation and Durability (ACID).

   - **Atomicity.** Users should be able to regard the execution of a transaction as atomic: either all the actions are carried out or none are. Users should not have to worry about the effect of incomplete transactions (e.g. when a system crash occurs).
   - **Consistency.** Each transaction, run by itself, must preserve the consistency of the database. The DBMS assumes that this holds for each transaction; thus it is the responsibility of the user.
   - **Isolation.** Users should be able to understand a transaction without considering the effect of other concurrently executing transactions, even if the DBMS is interleaving their execution for performance reasons. Transactions are isolated, or protected, from the effects of concurrently scheduling other transactions.
   - **Durability.** Once the DBMS informs the user that a transaction has completed successfully, its effects should persist, even if there is a system crash before all its changes are reflected on disk.

   Atomicity and Durability are ensured by the recovery manager component of a DBMS.

2. *Define what is meant by strict two-phase locking (strict 2PL).* *[4 marks]*

   A locking protocol is a set of rules followed by each transaction (and enforced by the DBMS) to ensure that even though the actions of several transactions may be interleaved, the net effect is identical to executing the transactions in some serial order. S2PL is one particular locking protocol. First assume that we have both read and write actions. We then provide shared locks for reads, i.e. we permit concurrent reads, and exclusive locks for writes, i.e. we prevent concurrent writes. Both locks enable you to read an object, but you need an exclusive lock to update it.

   There are a number of ways of describing the S2PL protocol. Two phase locking is simply a discipline where you have to have to obtain all your locks (the 'growing stage') before releasing any (the 'shrinking' phase). On other words, once you are in this shrinking stage of releasing locks you can not acquire new locks (or upgrade locks). Strict locking handles a problem due to rollbacks. It simply enforces that a transaction holds all its locks until it commits or rolls back. Once the commit or rollback is secure, then all locks are released. In other words, the shrinking stage occurs all at once, immediately after a transaction terminates.

3. *Assume that in addition to traditional* Read *and* Write *actions a DBMS supports two new actions:* Inc *and* Dec *(both are assumed to perform blind writes).*

   Consider the following two transactions.

   $$T1 : [\mathsf{Inc}(A), \mathsf{Dec}(B), \mathsf{Read}(C)]$$
   $$T2 : [\mathsf{Inc}(B), \mathsf{Dec}(A), \mathsf{Read}(C)]$$

(a) *By considering some possible schedules of the above transactions, describe carefully the concurrency permitted by strict 2PL using just shared/read and exclusive/write locks.* *[3 marks]*

The question tells you that Inc and Dec are blind writes, and as such must require an **exclusive** lock. Imagine trying to interleave their execution: Transaction T1 acquires exclusive lock on A and then performs the Inc. Transaction T2 then acquires exclusive lock on B and performs the Inc. Now we have deadlock because T1 requires an exclusive lock on B (which is owned by T2) and T2 requires the exclusive lock on A (which is owned by T1). Hence the only valid schedules are essentially sequential, i.e. hardly any concurrency! (The observation here is that strict 2PL does not solve all our problems!)

(b) *Detail a way to gain more interleaving whilst still maintaining strict 2PL.* *[7 marks]*

In the notes it is discussed that some actions, whilst appearing to require a write/exclusive lock, in fact can safely be given a shared lock. This is because the actions **commute**. Consider the increment action. Imagine that two transactions T3 and T4 wish to perform an increment action on the same object. As increment is assumed to be a blind write, it does not matter which order we interleave these actions, the net result is the same (a double increment). Thus we can introduce a new **increment** lock that can be held by many transactions.

The clever student will notice that decrement is in fact a negative increment, and so needs no further consideration. A weaker student may well suggest that we introduce a decrement lock that can be held by many transactions.

Consider the attempt of interleaving of T1 and T2 in the previous answer. Transaction T1 acquires the increment lock on A. Transaction T2 acquires the increment lock on B. Now T1 can acquire a (shared) increment lock on B, and similarly T2 for A. So we gain more possibilities for interleaving, i.e. more concurrency.

Selecting the appropriate level of granularity of locks is an important design decision for DBMS implementors.