

15-415 Database Applications

Homework Assignment 7

15-415 Database Applications
Carnegie Mellon University

April 17, 2006
Due: April 24, 2006 (3PM)

This assignment will test your understanding of concurrency control in database management systems. It is worth 6% of your grade. This is NOT a group assignment, and is to be done individually. The assignment is to be turned in on April 24th in class, or if you elect you use one or more late days, contact Tabreez so you can set up a time to hand in the assignment.

Problem 1: Serializability

Consider the following schedules. The actions are listed in the order they are scheduled, and prefixed with the transaction name.

1. **S1**: T1:R(X), T2:R(X), T1:W(X), T3:R(X), T2:Commit, T1:R(X), T3:W(X), T3:Commit, T1:Commit
2. **S2**: T2:R(X), T2:R(Y), T2:W(X), T2:Commit, T1:W(Y), T1:W(X), T1:R(Y), T1:Commit
3. **S3**: T3:R(X), T1:R(X), T2:R(X), T1:W(X), T1:W(Y), T3:R(Y), T2:Commit, T1:Commit, T3:Commit
4. **S4**: T1:R(X), T3:W(X), T3:Commit, T1:W(X), T1:Commit, T2:W(X), T2:R(X), T2:Commit

For each of the schedules, state if the schedule is *serial*, *conflict-serializable*, *view-serializable*. If the schedule is at least one of these, give the equivalent serial order of execution of the transactions. Please provide a brief explanation for your answer.

Problem 2: Extra Features

Consider the following (partial) schedule.

S: T1:R(X), T1:W(X), T1:R(Y), T2:R(X), T2:W(Y), T3:R(Y)

For each of the following properties, modify schedule S, by adding completion actions (i.e. commit, abort) so that the resulting schedule is complete and has the given property. If such a modification is not possible, briefly explain the reason.

- Avoids-cascading-aborts
- Avoids-cascading-aborts but is not recoverable
- Recoverable but does not avoid-cascading-aborts
- Not recoverable
- Conflict-serializable

Please give a brief explanation of why your schedule has the property, or why such a modification is not possible.

15-415 Database Applications

Problem 3: Locking Granularity

- a) We have the following hierarchy of objects to be locked in our database:

Database---v
 Table-----v
 Page-----v
 Tuple

For each of the following actions, give a sequence of lock acquisitions. Your sequence may contain S, X, IS, IX, SIX locks and references to the objects and their names where appropriate.

- Delete the record with SID = 1 in table Students (we don't need to find the tuple)
 - Get the name of the student with SID = 52 in table Students (don't do find here)
 - Change the GPA of the student with name = 'Tabreez' (assume Tabreez is unique) to 4.0 in table Students. (FIND and change involved here!)
- b) Give one example (which includes two actions and their lock acquisitions) that demonstrates the *Phantom* problem.

Problem 4: Concurrency Control Methods

Consider the following sequence of actions, listed in the order they are *submitted* to the DBMS for execution:

Sequence S: T1:R(X), T3: R(Y), T3:W(X), T3:Commit, T1:W(X), T1:W(Y) T1:Commit, T2:W(X), T2:R(X), T2:Commit

For the sequence above and each concurrency control mechanism below, list one possible sequence of actions resulting from the use of the concurrency control mechanism on the sequence (in some cases more than one possible final sequence exists depending on the way different choices are handled, e.g. time when aborted transactions are restarted and locks are requested etc.). Insert extra s-lock(x), x-lock(x) commands when a transaction locks an object. Assume timestamps are transaction numbers.

You can choose when aborted transactions are restarted. The executor will skip over actions for transactions that are currently blocked, or that have been aborted and not restarted. Once a transaction is unblocked by the algorithm or has been restarted when you restart it, the actions listed for that transaction must be executed in the order of the given sequence. Actions that the executor skipped over will have priority over those that the executor has yet to receive when a transaction is unblocked or restarted.

- Strict 2PL with timestamps used for deadlock prevention (wound-wait).
- Strict 2PL with timestamps used for deadlock prevention (wait-die).
- Strict 2PL with deadlock detection. (If there is a deadlock, just show waits-for graph and sequence preceding the deadlock).
- Timestamps concurrency control (while ensuring recoverability), using the Thomas Write Rule.
- Optimistic concurrency control with Serial Validation-Write. (just show interactions with the global objects i.e. initial reads, final writes and aborts/commits).