CS3223 Tutorial 7: Concurrency Control

Wk 10, Sem 2, 2022/23

Questions to be discussed: 3, 4 & 5.

1. (Exercise 17.2, R&G) Consider the following classes of schedules: recoverable, cascadeless, and strict. For each of the following schedules, state which of the schedule classes it belongs to.
   (a) $R_1(X)$, $R_2(X)$, $W_1(X)$, $W_2(X)$, $Commit_1$, $Commit_2$
   (b) $W_1(X)$, $R_2(Y)$, $R_1(Y)$, $R_2(X)$, $Commit_2$, $Commit_1$
   (c) $R_1(X)$, $R_2(Y)$, $W_3(X)$, $R_2(X)$, $R_1(Y)$, $Commit_1$, $Commit_2$, $Commit_3$
   (d) $R_1(X)$, $R_1(Y)$, $W_1(X)$, $R_2(Y)$, $W_3(Y)$, $W_1(X)$, $R_2(Y)$, $Commit_1$, $Commit_2$, $Commit_3$
   (e) $R_1(X)$, $W_2(X)$, $W_1(X)$, $Abort_2$, $Commit_1$
   (f) $R_1(X)$, $W_2(X)$, $W_1(X)$, $Commit_2$, $Commit_1$
   (g) $W_1(X)$, $R_2(X)$, $W_1(X)$, $Abort_2$, $Commit_1$
   (h) $W_1(X)$, $R_2(X)$, $W_1(X)$, $Commit_2$, $Commit_1$
   (i) $W_1(X)$, $R_2(X)$, $W_1(X)$, $Commit_2$, $Abort_1$
   (j) $R_2(X)$, $W_3(X)$, $Commit_3$, $W_1(Y)$, $Commit_1$, $R_2(Y)$, $W_2(Z)$, $Commit_2$
   (k) $R_1(X)$, $W_2(X)$, $Commit_2$, $W_1(X)$, $Commit_1$, $R_3(X)$, $Commit_3$
   (l) $R_1(X)$, $W_2(X)$, $W_1(X)$, $R_3(X)$, $Commit_1$, $Commit_2$, $Commit_3$

2. (Exercise 17.3, R&G) Consider the following two concurrency control protocols: 2PL and Strict 2PL. For each of the following schedules, state which of these protocols allows it, that is, allows the actions to occur in exactly the order shown. Assume that each lock acquistion step must occur right before the read/write action (e.g., it's not permissible to request for all the required locks at the beginning of the transaction).
   (a) $R_1(X)$, $R_2(X)$, $W_1(X)$, $W_2(X)$, $Commit_1$, $Commit_2$
   (b) $W_1(X)$, $R_2(Y)$, $R_1(Y)$, $R_2(X)$, $Commit_2$, $Commit_1$
   (c) $R_1(X)$, $R_2(Y)$, $W_3(X)$, $R_2(X)$, $R_1(Y)$, $Commit_1$, $Commit_2$, $Commit_3$
   (d) $R_1(X)$, $R_1(Y)$, $W_1(X)$, $R_2(Y)$, $W_3(Y)$, $W_1(X)$, $R_2(Y)$, $Commit_1$, $Commit_2$, $Commit_3$
   (e) $R_1(X)$, $W_2(X)$, $W_1(X)$, $Abort_2$, $Commit_1$
   (f) $R_1(X)$, $W_2(X)$, $W_1(X)$, $Commit_2$, $Commit_1$
   (g) $W_1(X)$, $R_2(X)$, $W_1(X)$, $Abort_2$, $Commit_1$
   (h) $W_1(X)$, $R_2(X)$, $W_1(X)$, $Commit_2$, $Commit_1$
   (i) $W_1(X)$, $R_2(X)$, $W_1(X)$, $Commit_2$, $Abort_1$
   (j) $R_2(X)$, $W_3(X)$, $Commit_3$, $W_1(Y)$, $Commit_1$, $R_2(Y)$, $W_2(Z)$, $Commit_2$
   (k) $R_1(X)$, $W_2(X)$, $Commit_2$, $W_1(X)$, $Commit_1$, $R_3(X)$, $Commit_3$
   (l) $R_1(X)$, $W_2(X)$, $W_1(X)$, $R_3(X)$, $Commit_1$, $Commit_2$, $Commit_3$

3. Give an example schedule that is conflict serializable but not a 2PL schedule.

4. (Exercise 17.4, R&G) Consider the following two sequences of actions, listed in the order they are submitted to the DBMS.

   Sequence S1: R_1(X), W_2(X), W_2(Y), W_3(Y), W_1(Y), $Commit_1$, $Commit_2$, $Commit_3$

   Sequence S2: R_1(X), W_2(Y), W_2(X), W_3(Y), W_1(Y), $Commit_1$, $Commit_2$, $Commit_3$

   For each sequence and for each of the following concurrency control mechanisms, describe how the concurrency control mechanism handles the sequence. Assume that the start timestamp of transaction T_i is i, and a transaction T_i has a higher priority than another transaction T_j if $i < j$. 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.

   (a) Strict 2PL with timestamps used for Wait-die deadlock prevention policy.

   (b) Strict 2PL with timestamps used for Wound-wait deadlock prevention policy.

   (c) Strict 2PL with deadlock detection. (Show the waits-for graph in case of deadlock.)

5. For each of the following pairs of transactions T_1 and T_2, state whether every interleaved execution of T_1 and T_2 is conflict serializable. If your answer is "No", write down an example schedule that is not conflict serializable.

   In this question, assume the following:

   1. Each transaction commits and is executed at the READ COMMITTED isolation level: short duration locks are acquired for read operations and long duration locks are acquired for write operations; transactions are allowed to acquire further locks after releasing locks.
   2. Short duration locks are released right after the associated operations are performed.
   3. Lock conversions are not used.

   (a) T_1: $R_1(x)$, $W_1(y)$, $Commit_1$
   T_2: $W_2(x)$, $Commit_2$

   (b) T_1: $R_1(x)$, $W_1(y)$, $Commit_1$
   T_2: $R_2(y)$, $W_2(x)$, $Commit_2$

   (c) T_1: $W_1(x)$, $Commit_1$
   T_2: $R_2(y)$, $W_2(x)$, $Commit_2$

   (d) T_1: $R_1(x)$, $W_1(y)$, $Commit_1$
   T_2: $R_2(x)$, $W_2(x)$, $W_2(y)$, $Commit_2$

6. For each transaction T_i in a 2PL schedule S, let $lastlocktime(T_i)$ denote the time that T_i acquires its last lock in S.

   For example, consider the following schedule S:

   $$W_2(\text{X}),\ R_1(\text{Y}),\ R_2(\text{Y}),\ R_1(\text{X}),\ Commit_1,\ Commit_2$$

   Schedule S is a 2PL schedule since it can be executed as follows:

   $$X_2(X), W_2(X), S_1(Y), R_1(Y), S_2(Y), R_2(Y), U_2(X), S_1(X), R_1(X), U_1(Y), U_1(X), Commit_1, U_2(Y), Commit_2$$

   Note that $lastlocktime(T_2) < lastlocktime(T_1)$ since T_2's last acquired lock (i.e., $S_2(Y)$) precedes T_1's (i.e., $S_1(X)$).

   Given a 2PL schedule S, let S' denote the serial schedule over the same set of transactions in S, where the transactions in S' are serialized in increasing order of their $lastlocktime()$ in S. Prove that S and S' are conflict equivalent.