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) $R_1(X)$, $W_2(X)$, $W_1(X)$, $R_3(X)$, $Commit_1$, $Commit_2$, $Commit_3$

Solution:

(a) **recoverable**, **cacadeless**, **not strict**.

(b) **not recoverable**, **not cascadeless**, **not strict**.

(c) **not recoverable**, **not cascadeless**. **not strict**.

(d) **not recoverable**, **not cascadeless**. **not strict**.

(e) **recoverable**, **cascadeless**. **not strict**.

(f) **recoverable**, **cascadeless**. **not strict**.

(g) **recoverable**, **not cascadeless**, **not strict**.

(h) **not recoverable**, **not cascadeless**. **not strict**.

(i) **not recoverable**, **not cascadeless**. **not strict**.

(j) **recoverable**, **cascadeless**, **strict**.

(k) **recoverable**, **cascadeless**, **strict**.

(l) **recoverable**, **not cascadeless**. **not strict**.

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) $R_1(X)$, $W_2(X)$, $W_1(X)$, $R_3(X)$, $Commit_1$, $Commit_2$, $Commit_3$

Solution:

(a) **Not 2PL, not S2PL**.

(b) **2PL, not S2PL**.

(c) **not 2PL, not S2PL**.

(d) **Not 2PL, not S2PL**.

(e) **Not 2PL, not S2PL**.

(f) **Not 2PL, not S2PL**.

(g) **Not 2PL, not S2PL**.

(h) **Not 2PL, not S2PL**.

(i) **Not 2PL, not S2PL**.

(j) **Not 2PL, not S2PL**.

(k) **Not 2PL, not S2PL**.

(l) **Not 2PL, not S2PL**.

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

Solution: Consider the schedule $W_1(x), W_2(x), Commit_2, W_3(y), Commit_3, R_1(y), Commit_1$. It is conflict equivalent to (T_3, T_1, T_2) but it is not a 2PL schedule as T_1 need to release its exclusive lock on x after $W_1(x)$ (to enable $W_2(x)$) and later request a shared lock for $R_1(y)$.

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.)

Solution:

(a) In Wait-die policy, whenever a lock request can't be granted, the requesting transaction is blocked if it has higher priority; and is aborted, otherwise.

- **Sequence S1**:
  - T_1 acquires S-lock on X for $R_1(X)$.
  - When T_2 requests for X-lock on X, T_2 is aborted since T_2 has lower priority than T_1,
  - T_3 acquires X-lock on Y for $W_3(Y)$.
  - When T_1 requests for X-lock on Y for $W_1(Y)$, T_1 is blocked since T_1 has higher priority than T_3,
  - T_3 commits and releases its X-lock on Y.
  - T_1 acquires X-lock on Y and resumes execution.
  - T_1 commits and releases its S-lock on X and X-lock on Y.
  - T_2 now can be restarted successfully.
- **Sequence S2**:
  - T_1 acquires S-lock on X for $R_1(X)$.
  - T_2 acquires X-lock on Y for $W_2(Y)$.
  - When T_2 requests for X-lock on X, T_2 is aborted since T_2 has lower priority than T_1. Thus, T_2 releases its X-lock on Y.
  - T_3 acquires X-lock on Y for $W_3(Y)$.
  - When T_1 requests for X-lock on Y, T_1 is blocked since T_1 has higher priority than T_3.
  - T_3 commits and releases its X-lock on Y.
  - T_1 now acquires X-lock on Y and resumes execution.
  - T_1 commits and releases its S-lock on X and X-lock on Y.
  - T_2 can now be restarted successfully.

(b) In Wound-wait policy, whenever a lock request can't be granted, the requesting transaction aborts the other transaction if the requesting transaction has higher priority; and gets blocked, otherwise.

- **Sequence S1**:
  - T_1 acquires S-lock on X for $R_1(X)$.
  - When T_2 requests for X-lock on X, T_2 is blocked since T_2 has lower priority than T_1.
  - T_3 acquires X-lock on Y for $W_3(Y)$.
  - When T_1 requests for X-lock on Y, T_3 is aborted since T_1 has higher priority than T_3. Thus, T_3 releases its X-lock on Y.
  - T_1 now acquires X-lock on Y.
  - T_1 commits and releases its S-lock on X and X-lock on Y.
  - T_2 now acquires X-lock on X for $W_2(X)$ and acquires X-lock on Y for $W_2(Y)$
  - T_2 commits and releases its X-locks on X & Y.
  - T_3 now can be restarted successfully.
- **Sequence S2**:
  - T_1 acquires S-lock on X for $R_1(X)$.
  - T_2 acquires X-lock on Y for $W_2(Y)$.
  - When T_2 requests for X-lock on X, T_2 is blocked since T_2 has lower priority than T_1.
  - When T_3 requests for X-lock on Y, T_3 is blocked since T_3 has lower priority than T_2.
  - When T_1 requests for X-lock on Y, T_2 is aborted since T_1 has higher priority than T_2. Thus, T_2 releases its X-lock on Y and T_1 acquires X-lock on Y for $W_1(Y)$.
  - T_1 commits and releases its S-lock on X and X-lock on Y.
  - T_2 restarts and acquires X-lock on Y for $W_2(Y)$.
  - T_2 acquires X-lock on X for $W_2(X)$.
  - T_2 commits and releases its X-locks on X & Y.
  - T_3 resumes execution and acquires X-lock on Y for $W_3(Y)$.
  - T_3 commits and releases its lock on Y.

(c)
- **Sequence S1**:
  - T_1 acquires S-lock on X for $R_1(X)$.
  - When T_2 requests for X-lock on X, T_2 is blocked and $T_2 \rightarrow T_1$ is added to WFG.
  - T_3 acquires X-lock on Y for $W_3(Y)$.
  - When T_1 requests for X-lock on Y, T_1 is blocked and $T_1 \rightarrow T_3$ is added to WFG.
  - T_3 commits and releases its X-lock on Y; $T_1 \rightarrow T_3$ is removed from WFG.
  - T_1 acquires X-lock on Y and resumes execution.
  - T_1 commits and releases its S-lock on X and X-lock on Y; $T_2 \rightarrow T_1$ is removed from WFG.
  - T_2 now acquires exclusive-locks on X and Y and commits after its execution.
- **Sequence S2**:
  - T_1 acquires S-lock on X for $R_1(X)$.
  - T_2 acquires X-lock on Y for $W_2(Y)$.
  - When T_2 requests for X-lock on X, T_2 is blocked and $T_2 \rightarrow T_1$ is added to WFG.
  - When T_3 requests for X-lock on Y, T_3 is blocked and $T_3 \rightarrow T_2$ is added to WFG.
  - When T_1 requests for X-lock on Y, T_1 is blocked and $T_1 \rightarrow T_2$ is added to WFG.
  - A deadlock between T_1 and T_2 is detected.

- Assume that T_2 is aborted to resolve the deadlock. Thus, T_2 releases its X-lock on Y; $T_2 \rightarrow T_1$, $T_3 \rightarrow T_2$, and $T_1 \rightarrow T_2$ are removed from WFG.
- T_1 acquires X-lock on Y for $W_1(Y)$.
- T_1 commits and releases its S-lock on X and X-lock on Y.
- T_2 resumes execution and acquires X-lock on Y for $W_2(Y)$.
- T_2 acquires X-lock on X for $W_2(X)$.
- When T_3 resumes execution and requests for X-lock on Y, T_3 is blocked and $T_3 \rightarrow T_2$ is added to WFG.
- T_2 commits and releases its locks on X & Y; $T_3 \rightarrow T_2$ is removed from WFG.
- T_3 resumes execution and acquires X-lock on Y for $W_3(Y)$.
- T_3 commits and releases its X-lock on Y.

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: $R_2(x)$, $W_2(x)$, $W_2(y)$, $Commit_2$

Solution:

(a) Yes. The only conflicting pair of actions is between $R_1(x)$ and $W_2(x)$. If T_1 is granted the lock on x before T_2, then the execution is conflict equivalent to the serial schedule (T_1, T_2); otherwise, the execution is the serial schedule (T_2, T_1).

(b) No. Example:
$S_1(x), R_1(x), U_1(x), S_2(y), R_2(y), U_2(y), X_2(x), W_2(x), Commit_2, U_2(x), X_1(y) W_1(y), Commit_1, U_1(y)$.
Here, we have the conflicting action pairs $(R_1(x), W_2(x))$ and $(W_2(y), W_1(y))$.

(c) Yes. The only conflicting pair of actions is between $W_1(x)$ and $W_2(x)$. If T_1 is granted the lock on x before T_2, then the execution is conflict equivalent to the serial schedule (T_1, T_2); otherwise, the execution is the serial schedule (T_2, T_1).

(d) No. Example:
$S_1(x), R_1(x), U_1(x), S_2(x), R_2(x), U_2(x), X_2(x), W_2(x), X_2(y), W_2(y), Commit_2, U_2(x), U_2(y), X_1(y), W_1(y), Commit_1, U_1(y)$.
Here, we have the conflicting action pairs $(R_1(x), W_2(x))$ and $(W_2(y), W_1(y))$.

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.

   Solution: Consider a pair of conflicting actions $(A_i(O), A_j(O))$ in S on object O, $i \neq j$, where $A_i(O)$ precedes $A_j(O)$ in S. Since the actions are conflicting, T_i must release its lock on O before $A_j(O)$ in S. Furthermore, T_i does not acquire any other lock in S after releasing its lock on O before $A_j(O)$. Therefore, $lastlocktime(T_i) < lastlocktime(T_j)$, and T_i precedes T_j in S'. Hence, $A_i(O)$ must also precede $A_j(O)$ in S'. Therefore, S and S' are conflict equivalent.