Transactions and Concurrency Control

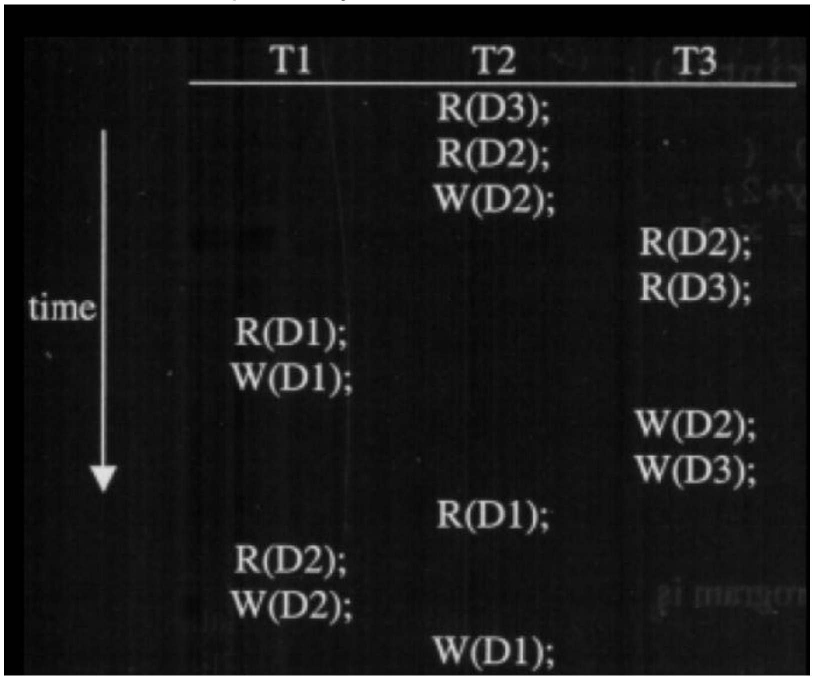
1) Which of the following scenarios may lead to an irrecoverable error in a database system?

A) A transaction writes a data item after it is read by an uncommitted transaction
B) A transaction read a data item after it is read by an uncommitted transaction
C) A transaction read a data item after it is written by an committed transaction
D) A transaction read a data item after it is written by an uncommitted transaction

Solution: (D)

A. Here if transaction writing data commits , then transaction which read the data might get phantom tuple/ Unrepeatable error. Though there is no irrecoverable error possible even in this option.
B. This is non issue. Both transaction reading data.
C. This is non issue.
D. This is dirty read. In case if transaction reading uncommitted data commits, irrecoverable error occurs of uncommitted transaction fails. So (D) is answer

2) Consider three data items D1,D2 and D3 and the following execution schedule of transactions T1,T2 and T3. In the diagram, R(D) and W(D) denote the actions reading and writing the data item D respectively.



T1	T2	T3
	R(D3);	
	R(D2);	
	W(D2);	
		R(D2);
		R(D3);
R(D1);		
W(D1);		
		W(D2);
		W(D3);
	R(D1);	
R(D2);		
W(D2);		
	W(D1);	

Which of the following statements is correct?

A) The schedule is serializable as T2; T3;T1;
B) The schedule is serializable as T2; T1;T3;
C) The schedule is serializable as T3; T2; T1;
D) The schedule is not serializable.

Solution: (D)

T1 and T2 have conflicting operations between them forming a cycle in the precedence graph. R(D2) of T2, and W(D2) of T1 (Read-Write Conflict) R(D1) of T1, and W(D1) of T2 (Read-Write Conflict) Hence in the precedence graph of the schedule there would be a cycle between T1 and T2 vertices. Therefore not a serializable schedule.

3) Consider a database with objects X and Y and assume that there are two transactions T1 and T2. Transaction T1 reads objects X and Y and then writes object X. Transaction T2 reads objects X and Y, then reads X once more, and finally writes objects X and Y (i.e. T1: R(X), R(Y), W(X); T2: R(X), R(Y), R(X), W(X), W(Y))

1. Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a write--read conflict.

2. Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a read--write conflict.

3. Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a write--write conflict.

4. For each of the three schedules, show that Strict 2PL disallows the schedule.

Solution:

1. The following schedule results in a write--read conflict:
T2:R(X), T2:R(Y), T2:W(X), T1:R(X) ...
T1:R(X) is a dirty read here.

2. The following schedule results in a read--write conflict:
T2:R(X), T1:R(X), T1:R(Y), T1:W(X), T2:R(X) ...
Now, T2 will get an unrepeatable read on X.

3. The following schedule results in a write--write conflict:
T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X), T2:R(X), T2:W(X) ...
Now, T2 has overwritten uncommitted data.

4. Strict 2PL resolves these conflicts as follows:
(a) In S2PL, T1 could not get a shared lock on X because T2 would be holding an

exclusive lock on X. Thus, T1 would have to wait until T2 was finished.
(b) Here T1 could not get an exclusive lock on X because T2 would already be holding a shared or exclusive lock on X.
(c) Same as above.

4) Consider the following sequences of actions, listed in the order they are submitted to the DBMS:

• Sequence S1: T1:R(X), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

• Sequence S2: T1:R(X), T2:W(Y), T2:W(X), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

For each sequence and for each of the following concurrency control mechanisms, describe
how the concurrency control mechanism handles the sequence.

Assume that the timestamp of transaction Ti is i. 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.

1. Strict 2PL with timestamps used for deadlock prevention.

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

3. Conservative (and Strict, i.e., with locks held until end--of--transaction) 2PL.

Solution:

1. Assume we use Wait--Die policy.
Sequence S1: T1 acquires shared--lock on X;
When T2 asks for an exclusive lock on X, since T2 has a lower priority, it will be aborted;
T3 now gets exclusive--lock on Y;
When T1 also asks for an exclusive--lock on Y which is still held by T3, since T1 has higher priority, T1 will be blocked waiting;
T3 now finishes write, commits and releases all the locks;
T1 wakes up, acquires the lock, proceeds and finishes;
T2 now can be restarted successfully.
Sequence S2: The sequence and consequence are the same with Sequence S1,

except T2 was able to advance a little more before it gets aborted.
T1 acquires shared--lock on X;
T2 acquires exclusive--lock on Y;
T2 asks for an exclusive--lock on X, but since it has lower priority than T1, it will be aborted;
T3 acquires exclusive--lock on Y;
T1 tries to acquire exclusive lock, but since it has higher priority than T3, it is allowed to wait T3 releases the lock and commits
T1 acquire the lock on Y and proceeds

2. In deadlock detection, transactions are allowed to wait; they are not aborted until a deadlock has been detected. (Compared to prevention schema, some transactions may have been aborted prematurely.)
Sequence S1: T1 gets a shared--lock on X;
T2 blocks waiting for an exclusive--lock on X;
T3 gets an exclusive--lock on Y;
T1 blocks waiting for an exclusive--lock on Y;
T3 finishes, commits and releases locks;
T1 wakes up, gets an exclusive--lock on Y, finishes up and releases lock on X and Y;
T2 now gets both an exclusive--lock on X and Y, and proceeds to finish.
No deadlock.
Sequence S2: There is a deadlock. T1 waits for T2, while T2 waits for T1. Deadlocks are resolved either using a timeout or by maintaining a waits--for graph.

3. Sequence S1: With conservative and strict 2PL, the sequence is easy. T1 acquires lock on both X and Y, commits, releases locks; then T2; then T3.
Sequence S2: Same as Sequence S1.

5) Consider the following sequence of actions S, and answer
the following questions:

S: r1(A), r2(A), r3(B), w1(A), r2(C), r2(B), w2(B), w1(C)

1. Is the schedule S view-serializable? If so, provide a view-equivalent serial schedule

2. Is the schedule S conflict-serializable? If so, describe all the conflict-equivalent serial schedules

3. Is the schedule S a 2PL schedule (with exclusive locks)?

4. Is the schedule S a 2PL schedule (with shared and exclusive locks)?

Solution:
• every 2PL schedule is conflict-serializable,
• every conflict-serializable schedule is view-serializable,

Let us first check whether S is in 2PL with exclusive locks.

1. From the theory of serializability we know that S is also view-serializable, and the serial schedule above is view-equivalent to S

2. Given that P(S) is acyclic, the schedule is conflict- serializable (as we already knew)
• The conflict-equivalent schedules are those corresponding to the only topological order of P(S),
i.e.
– T3 T2 T1: r3(B), r2(A), r2(C), r2(B), w2(B), r1(A), w1(A), w1(C)

3.
• In order for T2 to read A, it is necessary that T1 releases the lock on A (which was obtained by T1 for reading A)
• It follows that T1 enters the shrinking phase when T2 reads A
• In order for T1 to write A, it is necessary that T1 gets again the lock
on A
• T1 cannot get the lock on A before T2 reads A
• It follows that T1 should request a lock during the shrinking phase
IMPOSSIBLE: S is not a 2PL schedule with exclusive locks

4. Yes, by anticipating the exclusive lock on B by transaction T2 and the shared lock on C by the same transaction!
sl1(A) r1(A) sl2(A) r2(A) sl3(B) r3(B) u3(B) xl2(B)
sl2(C) u2(A) xl1(A) w1(A) r2(C) r2(B) w2(B) u2(C)
u2(B) xl1(C) w1(C) u1(A) u1(C)

6) Consider the following classes of schedules: ***serializable, conflict-serializable, view serializable, recoverable, avoids-cascading-aborts, and strict***. For each of the following schedules, state which of the preceding classes it belongs to. If you cannot decide whether a schedule belongs in a certain class based on the listed actions, explain briefly.

The actions are listed in the order they are scheduled and prefixed with the transaction name. If a commit or abort is not shown, the schedule is incomplete; assume that abort or commit must follow all the listed actions.

1. T1: R(X), T2: R(X), T1: W(X), T2: W(X)
2. T1: W(X), T2: R(Y), T1: R(Y), T2: R(X)

3. T1: R(X), T2: R(Y), T3: W(X), T2: R(X), T1: R(Y)
4. T1: R(X), T1: R(Y), T1: W(X), T2: R(Y), T3: W(Y), T1: W(X), T2: R(Y)
5. T1: R(X), T2: W(X), T1: W(X), T2: Abort, T1: Commit
6. T1: R(X), T2: W(X), T1: W(X), T2: Commit, T1: Commit
7. T1: W(X), T2: R(X), T1: W(X), T2: Abort, T1: Commit
8. T1: W(X), T2: R(X), T1: W(X), T2: Commit, T1: Commit
9. T1: W(X), T2: R(X), T1: W(X), T2: Commit, T1: Abort
10. T2: R(X), T3: W(X), T3: Commit, T1: W(Y), T1: Commit, T2: R(Y), T2: W(Z), T2: Commit
11. T1: R(X), T2: W(X), T2: Commit, T1: W(X), T1: Commit, T3: R(X), T3: Commit
12. T1: R(X), T2: W(X), T1: W(X), T3: R(X), T1: Commit, T2: Commit, T3: Commit

Solution:

1. Serizability (or view) cannot be decided but NOT conflict serizability. It is recoverable and avoid cascading aborts; NOT strict.

2. It is serializable, conflict-serializable, and view-serializable regardless which action (commit or abort) follows It is NOT avoid cascading aborts, NOT strict; We can not decide whether it's recoverable or not, since the abort/commit sequence of these two transactions are not specified.

3. It is the same with 2.

4. Serizability (or view) cannot be decided but NOT conflict serizability. It is NOT avoid cascading aborts, NOT strict; We can not decide whether it's recoverable or not, since the abort/commit sequence of these transactions are not specified.

5. It is serializable, conflict-serializable, and view-serializable; It is recoverable and avoid cascading aborts; it is NOT strict.

6. It is NOT serializable, NOT view-serializable, NOT conflict-serializable; it is recoverable and avoid cascading aborts; It is NOT strict.

7. It belongs to all classes

8. It is serializable, NOT view-serializable, NOT conflict-serializable; It is NOT recoverable, therefore NOT avoid cascading aborts, NOT strict.

9. It is serializable, view-serializable, and conflict-serializable; It is NOT recoverable, therefore NOT avoid cascading aborts, NOT strict.

10. It belongs to all above classes.

11. It is NOT serializable and NOT view-serializable, NOT conflict-serializable; it is recoverable, avoid cascading aborts and strict.

12. It is NOT serializable and NOT view-serializable, NOT conflict-serializable; it is recoverable, but NOT avoid cascading aborts, NOT strict.

7) Consider the following sequence S of actions, and answer these questions:

S: r2(A), r3(B), w1(A), r2(C), r2(D), w1(D)

1. Is the schedule S view-serializable? If so, provide a view-equivalent serial schedule

2. What is the precedence graph associated to S? Is the schedule S conflict-serializable? If so, describe all the conflict-equivalent serial schedules

3. Is the schedule S a 2PL schedule (with exclusive locks)?

Solution:

3. S can give rise to the following 2PL schedule with the commands for exclusive locks:

l2(A), r2(A), l3(B), r3(B), u3(B), l2(C), l2(D), u2(A), l1(A), w1(A), r2(C), u2(C), r2(D), u2(D), l1(D), w1(D), u1(A), u1(D)

2. Given that S is 2PL, S is also conflict-serializable. All conflict-equivalent schedules are those corresponding to possible topological order of P(S), i.e.
– T2 T1 T3: r2(A), r2(C), r2(D), w1(A), w1(D), r3(B)
– T3 T2 T1: r3(B), r2(A), r2(C), r2(D), w1(A), w1(D)
– T2 T3 T1: r2(A), r2(C), r2(D), r3(B), w1(A), w1(D)

1. Given that S is conflict-serializable, it is also view-serializable and all schedules above are also view-equivalent to S