CENG352- WRITTEN ASSIGNMENT 3

Onur Can ÜNAL

2095966

1)

a) In Strict Two Phase Locking (Strict 2PL), all locks held by a transaction are released when the transaction is completed, that is after commit or rollback happens. Since all locks has been released after the commits in both transactions, T1 and T2 uses Strict 2PL. As Strict 2PL ensures conflict serializability, conflict serializability is guaranteed.

b) Yes, it is possible. For example, if the operations are in the order:

$X_1(A)R_1(A)W_1(A)\ X_2(B)R_2(B)$

Then, when Transaction 2 requested the lock for B, it holds. However, Transaction 1 needs locking B to continue. Transaction 2 cannot unlock the lock on B before getting lock on A. This causes deadlock. Under the wait-die deadlock prevention schema, Transaction 2 is aborted since wait-die is non-preemptive and Transaction 1 has started first, and so it has higher priority.

c) Cascading rollback is not possible since both transactions use Strict 2PL. In Strict 2PL, all exclusive locks are released at the end of the commit of a transaction, and this guarantees that any data that is written but uncommitted cannot be readable because it holds the lock.

d) In Two Phase Locking (2PL), all lock requests must precede all unlock requests in every transaction. Since all locks has been requested before unlocks in both transactions, T1 and T2 uses 2PL. As 2PL ensures conflict serializability, conflict serializability is guaranteed.

e) No, it is not possible. When Transaction 1 locks A, Transaction 2 requests the resource A, and it waits. Since they can not hold the different resources' locks, A and B, at the same time, deadlock is impossible.

f) Yes, it is possible. Assume that the operation order is as the following:

$X_1(A)R_1(A)W_1(A)X_1(B)U_1(A)X_2(A)X_2(B)R_2(B)R_2(A)W_2(A)$

Then, Transaction 1 aborts. However, Transaction 2 made write operations on resource A after Transaction 1 had made. Thus, when Transaction 1 aborts (rollback), Transaction 2's changes also need to be undo (rollback), which causes to cascading rollback.

2)

a)

i)

Operation	A			B			C		
	RTS	WTS	C	RTS	WTS	C	RTS	WTS	C
$r_1(A)$	1	0	True	0	0	True	0	0	True
$r_2(B)$	1	0	True	2	0	True	0	0	True
$r_3(A)$	3	0	True	2	0	True	0	0	True
$w_1(A)$ **abort**	3	0	True	2	0	True	0	0	True
$r_2(C)$	3	0	True	2	0	True	2	0	True
$w_3(B)$	3	0	True	2	3	False	2	0	True
$w_2(C)$	3	0	True	2	3	False	2	2	False
c_1 **T_1 has already aborted**	3	0	True	2	3	False	2	2	False
$r_2(A)$	3	0	True	2	3	False	2	2	False
$w_3(C)$	3	0	True	2	3	False	2	3	False
c_3	3	0	True	2	3	True	2	3	True
$w_2(B)$ **ignore (Thomas Write Rule)**	3	0	True	2	3	True	2	3	True
c_2	3	0	True	2	3	True	2	3	True

ii)

Operation	A			B			C		
	RTS	WTS	C	RTS	WTS	C	RTS	WTS	C
$r_1(A)$	2	0	True	0	0	True	0	0	True
$r_2(B)$	2	0	True	3	0	True	0	0	True
$r_3(A)$	2	0	True	3	0	True	0	0	True
$w_1(A)$	2	2	False	3	0	True	0	0	True
$r_2(C)$	2	2	False	3	0	True	3	0	True
$w_3(B)$ **abort**	2	2	False	3	0	True	3	0	True
$w_2(C)$	2	2	False	3	0	True	3	3	False
c_1	2	2	True	3	0	True	3	3	False
$r_2(A)$	3	2	True	3	0	True	3	3	False
$w_3(C)$ **T_3 has already aborted**	3	2	True	3	0	True	3	3	False
c_3 **T_3 has already aborted**	3	2	True	3	0	True	3	3	False
$w_2(B)$	3	2	True	3	3	False	3	3	False
c_2	3	2	True	3	3	True	3	3	True

b)

Commit bit is false when a transaction writes to a data, but it has not committed yet. When the commit bit is false, the other transactions that want to access (read or write) that data may have to wait to become it true. This makes Timestamp-based Scheduling with the commit bit avoiding cascading aborts, and hence this makes it recoverable.

3)