TUTORIAL-II
DBMS

Exercise 1: Give an example of a transaction schedule that is conflict-serializable, but not possible under 2PL.

Answer: R1(X)R2(Y)R3(Y)W2(Y)W1(X)W3(X)R2(X)W2(X). A valid 2PL scheme cannot be scheduled with the same schedule because to execute W2(Y) the shared lock on Y by transaction 3 would have to be released but as W3(X) is also scheduled in transaction 3 after R3(Y), an exclusive lock on X would have to be acquired by transaction 3, after the release of lock on Y. This violates the 2PL protocol.

Exercise 2: The lost update anomaly is said to occur if a transaction T_j reads a data item, then another transaction T_k writes the data item (possible based on previous read), after which T_j writes the data item. The update performed by T_k has been lost, since the update done by T_j ignored the value written by T_k.
2(a): Give an example of schedule showing the lost update anomaly.
2(b) Give an example schedule to show that the lost update anomaly is possible with the read committed isolation level.
2(c) Explain why the lost update anomaly is not possible with the repeatable read isolation level.

Answer: (a) A schedule showing the Lost Update Anomaly. In the below schedule, the value written by transaction T_2 is lost because of the write of transaction T_1.

T_1	T_2
Read (A)	
	Read (A)
	Write (A)
Write (A)	

(b) Lost Update Anomaly in the Read committed Isolation level is shown below.

T_1	T_2
lock-S(A)	
Read (A)	
unlock(A)	

	lock-X(A)
	Read (A)
	Write (A)
	unlock(A)
	commit
lock-X(A)	
Write (A)	
unlock(A)	
commit	

(c) Lost update anomaly is not possible in the Repeatable read isolation level. In repeatable read isolation level, a transaction T_1 reading a data item X, holds a shared lock on X till the end. This makes it impossible for a newer transaction T_2 to write the value of X (which requires X-lock) until T_1 finishes. This forces the serialization order T_1, T_2 and thus the value written by T_2 is not lost.

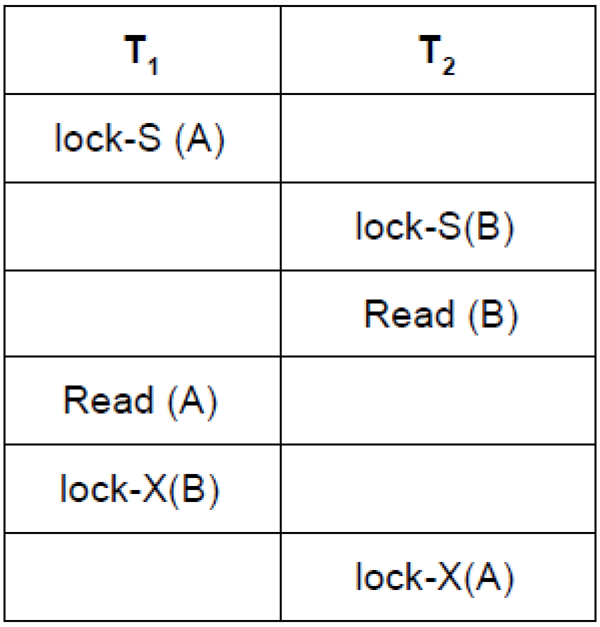
Exercise 3: Consider the following two transactions:
T1: read(A);
read(B);
If A = 0, then B = B+1;
Write(B);
T2: read(B);
read(A);
If B = 0, then A = A+1;
Write(A);
Add lock and unlock instructions to transactions T1 and T2, so that they observe the two-phase locking protocol. Can execution of these transactions result in a deadlock?

Add lock and unlock instructions to transactions T1 and T2, so that they observe the two-phase locking protocol. Can execution of these transactions result in a deadlock?

Answer. Lock and unlock instructions:
T1: lock-S(A)
read(A);
lock-X(B)
read(B);

If A = 0, then B = B+1;
Write(B);
unlock(A)
unlock(B)

T2: lock-S(B)
read(B);
lock-X(A)
read(A);
If B = 0, then A = A+1;
Write(A);
unlock(B)
unlock(A)
Execution of these transactions can result in deadlock. For example, consider the following partial schedule:



T_1	T_2
lock-S (A)	
	lock-S(B)
	Read (B)
Read (A)	
lock-X(B)	
	lock-X(A)

Exercise 4: Consider a database organized in the form of a rooted tree. Suppose that we insert a dummy vertex between each pair of vertices. Show that, if we follow the tree protocol on the new tree, we get better concurrency than if we follow the tree protocol on the original tree.

Answer. Consider two nodes A and B, where A is a parent of B. Let dummy vertex D be added between A and B. Consider a case where transaction T2 has a lock on B, and T1, which has a lock on A wishes to lock B, and T3 wishes to lock A. With the original tree, T1 cannot release the lock on A until it gets the lock on B. With the modified tree, T1 can get a lock on D, and release the lock on A, which allows T3 to proceed while T1 waits for T2. Thus, the protocol allows locks on vertices to be released earlier to other transactions, instead of holding them when waiting for a lock on a child.

Exercise 5: Show by example that there are schedules possible under the tree protocol that are not possible under the two-phase locking protocol, and vice versa.

Answer. Consider a tree-structured database graph (A -> B -> C), where A is the root node,

and A is the parent of B. B is the parent of node C.
Schedule possible under tree protocol but not under 2PL.

T_1	**T_2**
Lock (A)	
Lock (B)	
Unlock (A)	
	Lock (A)
Lock (C)	
Unlock (B)	
	Lock (B)
	Unlock (A)
	Unlock (B)
Unlock (C)	

Schedule possible under 2PL but not tree protocol:

T_1	**T_2**
Lock (A)	
	Lock (B)
Lock (C)	
	Unlock (B)
Unlock (A)	
Unlock (C)	

Exercise 6 Consider the following tree and locking sequences:

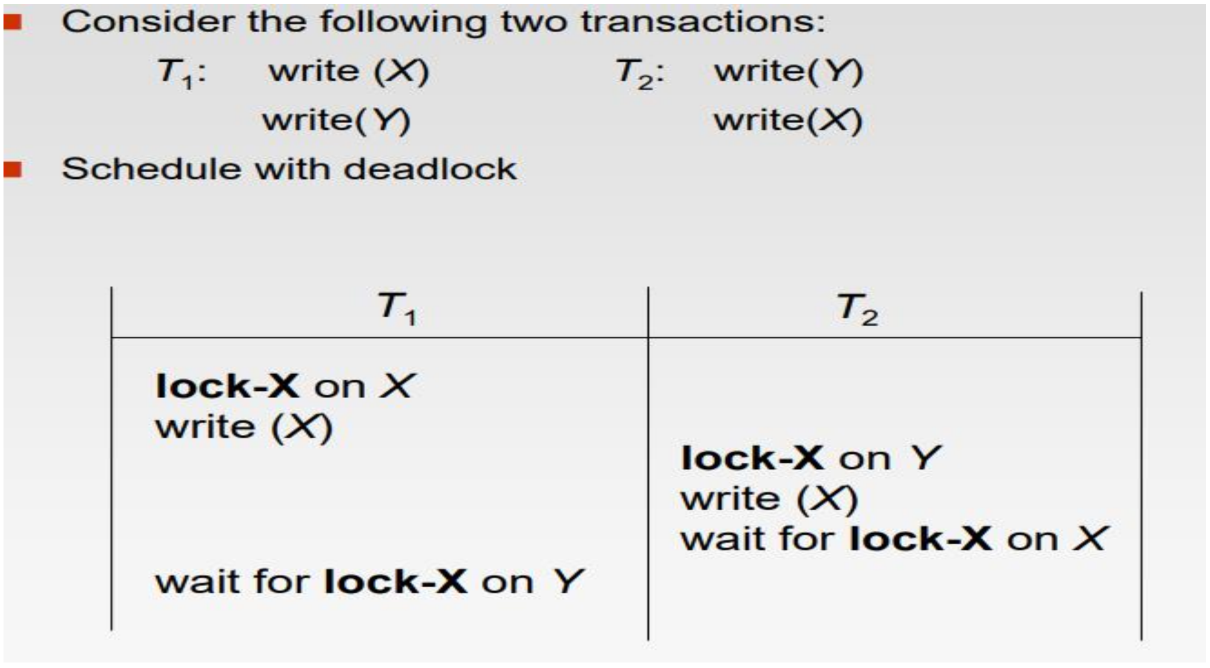
$$\begin{tikzcd}
 & & A \arrow[dl] \arrow[dr] & & \\
 & B \arrow[dl] \arrow[d] \arrow[dr] & & C \arrow[dr] & \\
D & E \arrow[dl] \arrow[dr] & F & & G \\
H & & I & &
\end{tikzcd}$$

(1) Lock-X(A), Lock-X(B), Lock-X(D), unlock(D), Lock-X(F), unlock(F), unlock(B), unlock(A)
(2) Lock –X(A), Lock-X(E), Lock-X(H), unlock(H), unlock(E), unlock(A)
(3) Lock –X(B), Lock-X(F) ,Lock-X(E) ,unlock(F) ,unlock(E) ,unlock (B)

Which of the above is (are) the valid sequence(s) for tree based protocol?

Answer 1 and 3

Exercise 7: Give a schedule which is conflict serializable but not dead lock free?



- Consider the following two transactions:

  T$_1$: write (*X*) write(*Y*) *T*$_2$: write(*Y*) write(*X*)

- Schedule with deadlock

T$_1$	*T*$_2$
lock-X on *X*	
write (*X*)	
	lock-X on *Y*
	write (*X*)
	wait for **lock-X** on *X*
wait for **lock-X** on *Y*	