Transaction Management Exercises

1. Explain ACID properties of a transaction.

   Easy.

2. What is a serializable schedule?

   Check the definition.

3. What is a conflict serializable schedule?

   Check the definition.

4. Give an algorithm to test for conflict serializability. What is the time complexity of the algorithm?

   Hint: Construct a precedence graph and the apply the topological sorting algorithm.

5. What is a recoverable schedule?

   Check the definition.

6. What is a cascadeless schedule?

   Check the definition.

7. Show that a 2PL schedule is serializable schedule.

   Let S be a 2PL schedule for transactions $T_1, T_2, \dots, T_n$. We construct a precedence graph G for S. We claim that G has no cycle. This claim means that S is conflict serializable.

   Proof of the claim. Assume by contradiction that G has a cycle. That is, we have a cycle $T_{i_1} \to T_{i_2} \to \cdots \to T_{i_j} \to T_{i_1}$. Let t(T) denote the lock points, which is the time transaction T requests its last lock. From the precedence graph definition, $T_{i_k} \to T_{i_{k+1}}$ implies two facts: (1) T_{i_k} accessed a data x time that had already been accessed by $T_{i_{k+1}}$; and (2) the operation of T_{i_k} on x conflicts with the operation of $T_{i_{k+1}}$ on x. These two facts imply that $T_{i_{k+1}}$ held a lock on x but released its before T_{i_k} accessed x, otherwise T_{i_k} was able to access x. These further imply that T_k gain a lock on x after $T_{i_{k+1}}$ released its lock on x. Since the schedule S is a 2PL schedule, we have $t(T_k) > t(T_{k+1})$. Applying the any two adjacent transactions on the cycle $T_{i_1} \to T_{i_2} \to \cdots \to T_{i_j} \to T_{i_1}$, we have

   $$t(T_1) > t(T_2) > \cdots > t(T_j) > t(T_1).$$

   Hence, we have a contradiction. Therefore, the graph G has no cycle.

8. Give a conflict serializable schedule which cannot have an equivalent 2PL schedule.

T1	T2
W(A)	
	W(A)
	R(B)
R(B)	
	W(B)

Conflict serializable

→

T1	T2
W(A)	
R(B)	
	W(A)
	R(B)
	W(B)

Equivalent serial schedule

Need to justify the above conflict serializable schedule cannot be realized with 2PL.

9. What is deadlock? Give an algorithm to detect deadlock. What is the time complexity of the algorithm?

   Check the definition. Review wait-for graphs. Review another application of the topological sorting algorithm.

10. Show that the wait-die strategy will prevent deadlock and starvation.

    Let $T_1, T_2, \dots, T_n$ be a list of transactions. Suppose that they follow the wait-die strategy with timestamp $t(T_i) = i$. We have the following two claims.

    Claim 1: There is deadlock.

    Proof of Claim 1. We construct a wait-for graph G for $T_1, T_2, \dots, T_n$. Assume, by contradiction, that there is a deadlock. There is a cycle in G. That is, we have a cycle $T_{i_1} \to T_{i_2} \to \cdots \to T_{i_j} \to T_{i_1}$. Because all transactions follow the wait-die strategy and this strategy only allows an older transaction to wait for a younger transaction, the cycle implies

    $$t(T_1) < t(T_2) < \cdots < t(T_j) < t(T_1).$$

    Thus, we have a contradiction. Hence, there is no deadlock.

    Claim 2: There is starvation.

    Proof of Claim 2. Let $f(T_i)$ denote the time at which transaction T_i is completely executed. Following the wait-die strategy, T_1 is the oldest transaction, so it never waits for any other transaction, so it will starve and will finish at $f(T_1)$ time. For transaction T_2, it may be completed before $f(T_1)$. If not, then after $f(T_1)$, T_2 becomes the oldest transaction and it will never wait for any other transactions that have not been completed, so it will not starve. With the similar analysis, none of $T_3, \dots, T_n$ will starve.

11. Show that the wound-wait strategy will prevent deadlock and starvation.

Think about do this problem with a similar approach to Problem 10.

12. Will the timestamp-based protocol prevent deadlock and starvation? Explain.

    Pretty straightforward. Think about. 😊

13. Give a 2PL schedule which cannot have a timestamp-based schedule.

    A 2PL protocol but not timestamp

T3	T4
Lock(A)	
W(A)	
	Lock(B)
	W(B)
	Unlock(B)
Lock(B)	
R(B)	
Unlock(B)	
Unlock(A)	

    Need to justify the above schedule cannot realized with the timestamp protocol.

14. Give a timestamp-based schedule which cannot have a 2PL schedule.

    A timestamp protocol but not 2PL

T5	T6	
W(A)		
W(B)		
		← for 2PL, T5 must unlock B so that T6 lock B and then write B
	W(B)	
		← for 2PL, T5 must unlock A so that T6 can lock A and then read A
	R(A)	
		← for 2PL, T6 must unlock A so that T5 can lock A and then read A
		This contradicts to 2PL for T5
R(A)		

    Justification include.