Practice Questions

Q1.

Q. No 5: Consider the following schedule of actions: [10]

S: r1(X), r2(Z), w1(Z), r3(X), r3(Y), w1(X), w3(Y), r2(Y), c3, c2, c1, w4(Z), w4(Y), c4.

For each of the following concurrency control mechanisms, describe how the concurrency control mechanism handles the schedule. Assume that the timestamp of transaction *Ti* is *i*. For lock-based concurrency control mechanisms, add lock and unlock requests to the above schedule 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 schedule) of an unblocked transaction.

a. Rigorous 2PL with timestamps used for deadlock detection (Use wait-for-graph to deal with deadlock)

b. Basic Timestamp Ordering (Assume T1 < T2 < T3)

Solution:

Ans: a)

T_1	T_2	T_3	T_4
s1-lock(X) r1(X)			
	s2-lock(Z) r2(Z)		
x1-lock(Z) ...wait for T2			
		s3-lock(X) r3(X) s3-lock(Y) r3(Y) x3-lock(Y) ...upgrade-lock w3(Y)	
	s2-lock(Y) ...wait for T3		
		c3, release all locks. unlock3(X) unlock3(Y)	
	r2(Y) ...wake-up c2, release all locks. unlock2(Z) unlock2(Y)		
w1(Z) ...wake-up x1-lock(X) ...upgrade-lock w1(X) c1, release all locks. unlock1(Z) unlock1(X)			
			x4-lock(Z) w4(Z) x4-lock(Y) w4(Y) c4, release locks. unlock4(Z) unlock4(Y)

Wait-for-graph:

b)

T1	T2	T3	T4	X		Y		Z	
				RTS	*WTS*	*RTS*	*WTS*	*RTS*	*WTS*
				T0	{ }	T0	{ }	T0	{ }
r1(X)				{T1}					
	r2(Z)							{T2}	
w1(Z)									
abort T1 as RTS(Z)>TS(T1)									
				{T3}					
		r3(X)				{T3}			
		r3(Y)					T3		
		w3(Y)							
	r2(Y)								
	abort T2 as WTS(Y)>TS(T2)								
		c3							
			w4(Z)						T4
			w4(Y)				T4		
			c4						
restart T1 *with new TS.*									
	Restart T2 *with new TS.*								

Q2:

Q. No 4: Consider the following schedule: [5]

S: r1(X), r2(Z), w1(Z), r3(X), r3(Y), w1(X), w3(Y), r4(Y), w4(Z), w4(Y).

Draw the serializability (precedence) graph for this schedule. State whether this schedule is conflict-serializable (correct) or not. If the schedule is conflict-serializable, write down the equivalent serial schedule(s) otherwise explain why it is not. *Also state whether this schedule is view-serializable or not.*

Solution:

Ans:

Ans: It is conflict-serializable and equivalent serial schedules are T2→T3→T1→T4 and T3→T2→T1→T4. It is also view-serializable.

Q4. *(15 points)* Consider the following schedule of actions, listed in the order they are submitted to the DBMS:

S3: $r_1(A); r_2(B); w_3(B); w_1(C); r_2(C); r_2(D); r_3(C); c_3; c_2; w_1(D); c_1.$

For each of the following concurrency control mechanisms, describe how the concurrency control mechanism handles the schedule. Assume that the timestamp of transaction *Ti* is *i*. For lock-based concurrency control mechanisms, add lock and unlock requests to the above schedule 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 schedule) of an unblocked transaction.

a. Rigorous 2PL with timestamps used for deadlock avoidance (Use wound-wait policy)
b. Strict Timestamp Ordering (Assume T1 < T2 < T3)
c. Optimistic concurrency control technique (Use defer the validation until a later time when the conflicting transactions have finished.)

a) Rigorous 2PL with deadlock avoidance (Using wound-wait policy)

Time ↓

Transaction T_1	Transaction T_2	Transaction T_3
s1-lock(A)		
r1(A)		
	s2-lock(B)	
	r2(B)	
		x3-lock(B) .. wait for T2 on B
x1-lock(C)		
w1(C)		
	s2-lock(C)...wait for T1 on C	
x1-lock(D)		
w1(D)		
c1, releases all locks		
unlock(A)		
unlock(C)		
unlock(D)		
	r2(C)...wake-up	
	s2-lock(D)	
	r2(D)	
	c2, releases all locks	
	unlock(B)	
	unlock(C)	
	unlock(D)	
		w3(B) ...wake-up
		s3-lock(C)
		r3(C)
		c3, releases all locks
		unlock(C)
		unlock(B)

b) Strict TO:

T1	T2	T3	*Timestamps and versions of Objects*							
			A		B		C		D	
			RTS	*WTS*	*RTS*	*WTS*	*RTS*	*WTS*	*RTS*	*WTS*
			{ }	**T0**	{ }	**T0**	{ }	**T0**	{ }	**T0**
r1(A)			{T1}							
	r2(B)				{T2}					
		w3(B)				T3				
w1(C)								T1		
	r2(C) **wait for T1 to c/a**									
		r3(C) **wait for T1 to c/a**								
w1(D)										T1
c1										
	r2(C) ...wake-up						{T2}			
		r3(C) ...wake-up					{T3}			
	r2(D)								{T2}	
	c2									
		c3								