National University of Computer and Emerging Sciences, Lahore Campus

	Course:	**Advance Database Concepts**
	Program:	**BS (Computer Science)**
	Instructor:	**Muhammad Ishaq Raza**
	Practice Problems:	**CCT - SOLUTION**

Topic: Concurrency Control Techniques

Q1. Consider the following schedule of actions listed in the order they are submitted to the DBMS:
Schedule S: r1(X), w2(X), w2(Y), w3(Y), w1(Y), c1, c2, c3

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 wait-die policy)
b. Rigorous 2PL with timestamps used for deadlock avoidance (use wound-wait policy)
c. Rigorous 2PL with deadlock detection. (Show the wait-for-graph in case of deadlock)
d. Conservative 2PL
e. Basic Timestamp Ordering (TO)
f. Strict Timestamp Ordering (Strict TO)
g. Timestamp Ordering with Thomas's Write Rule (TWR)
h. Multi-version Timestamp Ordering

Ans:

Schedule S: r1(X), w2(X), w2(Y), w3(Y), w1(Y), c1, c2, c3

a) Rigorous 2PL with timestamps used for deadlock avoidance (**using wait-die policy**)

Transaction T_1	**Transaction T_2**	**Transaction T_3**
s1-lock(X) r1(X)		
Time	x2-lock(X)…**abort** due to T1**,** & restart with same TS	
		x3-lock(Y) w3(Y)
x1-lock(Y)…**wait** for T3 on Y		
		c3, releases all locks unlock(Y)
w1(Y)…**wake-up** **c1**, releases all locks unlock(X) unlock(Y)		
	T2 can now be restart here.	

Schedule S: r1(X), w2(X), w2(Y), w3(Y), w1(Y), c1, c2, c3

b) Rigorous 2PL with timestamps used for deadlock avoidance (**using wound-wait policy**)

Transaction T_1	**Transaction T_2**	**Transaction T_3**
s1-lock(X)		
r1(X)		
Time	x2-lock(X)…**wait** for T1 on X	
		x3-lock(Y)
		w3(Y)
x1-lock(Y)…**wound T3**		
		abort due to T1 & restart with same TS & releases all locks
		unlock(Y)
w1(Y)		
c1, releases all locks		
unlock(X)		
unlock(Y)		
	w2(X)…**wake-up**	
	x2-lock(Y)	
	w2(Y)	
	c2, releases all locks	
	unlock(X)	
	unlock(Y)	
		T3 can be **restart** here.

Schedule S: r1(X), w2(X), w2(Y), w3(Y), w1(Y), c1, c2, c3

c) Rigorous 2PL with deadlock detection. (Using wait-for-graph)

Transaction T_1	**Transaction T_2**	**Transaction T_3**
s1-lock(X)		
r1(X)		
Time	x2-lock(X)…**wait** for T1 on X	
		x3-lock(Y)
		w3(Y)
x1-lock(Y)…**wait** for T3 on Y		
		c3, releases all locks
		unlock(Y)
w1(Y)…**wake-up**		
c1, releases all locks		
unlock(X)		
unlock(Y)	w2(X)…**wake-up**	
	x2-lock(Y)	
	w2(Y)	
	c2, releases all locks	
	unlock(X)	
	unlock(Y)	

Schedule S: r1(X), w2(X), w2(Y), w3(Y), w1(Y), c1, c2, c3

d) Conservative 2PL:

Transaction T_1	Transaction T_2	Transaction T_3
s1-lock(X)		
x1-lock(Y)		
r1(X)		
w1(Y)		
c1, releases all locks		
unlock(X)		
unlock(Y)		
	x2-lock(X)	
	x2-lock(Y)	
	w2(X)	
	w2(Y)	
	c2, releases all locks	
	unlock(X)	
	unlock(Y)	
		x3-lock(Y)
		w3(Y)
		c3, releases all locks
		unlock(Y)

Time

Schedule S: r1(X), w2(X), w2(Y), w3(Y), w1(Y), c1, c2, c3

e) Basic TO:

T1	T2	T3	X RTS	X WTS	Y RTS	Y WTS
			RTS	*WTS*	*RTS*	*WTS*
			{ }	**T0**	{ }	**T0**
r1(X)			{T1}			
	w2(X)			T2		
	w2(Y)					T2
		w3(Y)				T3
w1(Y), **abort T1** & restart with new TS, as WTS(Y)>TS(T1)						
	c2					
		c3				

Timestamps and versions of Objects

Schedule S: r1(X), w2(X), w2(Y), w3(Y), w1(Y), c1, c2, c3

f) Strict TO:

Timestamps and versions of Objects

T1	T2	T3	X		Y	
			RTS	*WTS*	*RTS*	*WTS*
			{ }	**T0**	{ }	**T0**
r1(X)			{T1}			
	w2(X)			T2		
	w2(Y)					T2
		w3(Y), **delay T3** until c2 or a2, as WTS(Y)<TS(T3)				
w1(Y), **abort T1** & restart with new TS, as WTS(Y)>TS(T1)						
	c2					
		w3(Y), exe here				T3
		c3				

Schedule S: r1(X), w2(X), w2(Y), w3(Y), w1(Y), c1, c2, c3

g) Timestamp Ordering with TWR:

			Timestamps and versions of Objects			
T1	**T2**	**T3**	**X**		**Y**	
			RTS	*WTS*	*RTS*	*WTS*
			{ }	**T0**	{ }	**T0**
r1(X)			{T1}			
	w2(X)			T2		
	w2(Y)					T2
		w3(Y)				T3
w1(Y), <u>**ignore**</u> this write operation & continue.						
C1						
	c2					
		c3				

Schedule S: r1(X), w2(X), w2(Y), w3(Y), w1(Y), c1, c2, c3

h) Multi-version Timestamp Ordering:

			Timestamps and versions of Objects			
T1	**T2**	**T3**	**X**		**Y**	
			RTS	*WTS*	*RTS*	*WTS*
			{T0}	**T0**	{T0}	**T0**
r1(X)			{T1}			
	w2(X)		{T2}	T2		
	w2(Y)				{T2}	T2
		w3(Y)			{T3}	T2 T3
w1(Y), **abort T1** & restart with new TS, as RTS(Y)>TS(T1) (otherwise late write operation would invalidate a read operation of younger.)						
	c2			**T2**		**T2** T3
		c3				**T3**

Q2. Consider the following schedule: indicate if it is valid according to 2PL (two-phase locking); Explain how. Also identify the type of 2PL (basic, conservative, strict, rigorous) satisfied by the schedule, explain how. Also check if it successfully completes or may result in a deadlock state, and how.

S: sl1(A); r1(A); xl2(B); r2(B); w2(B); xl1(B); c2; ul2(B); w1(B); c1; ul1(A); ul1(B);

Note: sl= Shared Lock, xl= Exclusive Lock

Ans:

The schedule is valid and implements rigorous 2PL as all locks are held until after commit. Note that T1 was forced to wait for an exclusive lock on B until T2 committed. The schedule would successfully complete.

It is also conflict-serializable and view-serializable.

Q3. Consider the following schedule:

S: $R_1(X)$, $W_2(Y)$, $R_2(X)$, $W_1(Y)$, C_1, C_2

State which of the following concurrency control protocols allows it, that is allows the actions to occur in exactly the order shown: Basic 2PL, Strict 2PL, Rigorous 2PL. Add appropriate lock before each read/write operation. Please provide a brief explanation for your answer...If YES, show where the lock requests could have happened; If NO, explain briefly

Ans:
The schedule will be permitted by Basic 2PL, but not by Strict/Rigorous 2PL. This is because when T1 tries to request an exclusive lock to write Y, T2 has not committed yet, and therefore has not released its exclusive lock on Y. It will be permitted by Basic 2PL because then T2 can release the lock on Y immediately after it acquires a shared lock for X (prior to reading X). Then T1 will be able to acquire its exclusive lock on Y.
It is also conflict-serializable and view-serializable.

Q4. Consider the following schedule:

S: r1(z);r1(y);w1(y); r2(y);r2(z); r3(x);w3(x); w2(y);w2(z); r1(x); r3(y);w3(y); r1(x).

a. Apply the rigorous 2PL to the above schedule and determine whether the protocol will allow the execution of the schedule. Add appropriate lock before each read/write operation. Indicate what lock requests are denied and whether deadlock occurs, explain how. Show the wait-for graph. If a deadlock occurs pick a younger transaction to abort and then continue. Indicate what transactions would finish execution.

b. Apply the basic timestamp-ordering (assume T1 < T2 < T3) to the above schedule and determine whether the protocol will allow the execution of the schedule. Indicate what transactions would finish execution.

Ans: Do it yourself.

Q5. For the schedule **S:** r2(X), w3(X), c3, w1(Y), r2(Y), r2(Z), c2, r1(Z), c1. Show that schedule S will be accepted/rejected in exactly the order shown by the below protocols. Provide proper reasons and show your work.

a. Basic 2PL (add locks to the transactions)

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

c. Strict Timestamp Ordering (Assume T1 < T2 < T3)

d. Optimistic Concurrency Control

Ans:

a. Basic 2PL (Reject);

Here xl3(X) cannot acquire due to sl2(X).

1- ABORT T3 for Deadlock Avoidance using wait-die scheme

2- WAIT T3 for sl2(X) to release for Deadlock Avoidance using wound-wait scheme

3- WAIT T3 for sl2(X) to release for Deadlock Detection (use wait-for-graph, in case of deadlock)

b. Basic TO (ACCEPT);

- w3(X) allow as T3 is younger than RTS of X (i.e. TS(T3) > RTS(X))

- r2(Y) allow as T2 is younger than WTS of Y (i.e. TS(T2) > WTS(Y))

c. Strict TO (REJECT);

- r2(Y) delay until c1/a1; as T2 is younger than WTS of Y (i.e. TS(T2) > WTS(Y))

d. Optimistic (ACCEPT- incase T3 delay, Otherwise REJECT – incase T3 abort);

- T3 Forward Validation fails due to item X conflict with T2; so it may be delay or abort T3/T2.

- But the validation of T1 & T2 is successful, if T3 abort/delay until T2 commit.