CIS560 – Database System Concepts **Name:_____**

Homework Assignment 7 [20 points] – due November 1st (by midnight)

1. [3 points] Consider a database with objects X and Y and assume that there are two transactions T1 and T2. Transaction T1 reads objects X and Y and then writes X. Transaction T2 reads objects X and Y and then writes objects X and Y. Give three examples of schedules for the transactions T1, T2 such that:

   i. Your first schedule should contain a write-read conflict.

   ```
   The following schedule results in a write-read conflict:

   r2(X), r2(Y), w2(X), r1(X) ...

   r1(X) is a dirty read here.
   ```

   ii. Your second schedule should contain a write-write conflict.

   ```
   The following schedule results in a write-write conflict:

   r2(X), r2(Y), r1(X), r1(Y), w1(X), w2(X) ...

   Now, T2 has overwritten uncommitted data.
   ```

   iii. Your third schedule should contain a read-write conflict.

   ```
   The following schedule results in a read-write conflict:

   r2(X), r2(Y), r1(X), r1(Y), w1(X), r2(X) ...

   Now, T2 will get an unrepeatable read on X.
   ```

   In each case your schedule may contain additional conflicts, but should contain at least one conflict of the type indicated. (In particular you may give a single schedule, which illustrates all three conflicts!) In each case, indicate the conflict of the type you are illustrating.

2. [5 points] For the following schedules:

   (a) $r_2(A); r_1(C); r_2(B); w_2(B); r_3(B); r_1(A); r_3(C); w_3(C); w_1(A)$

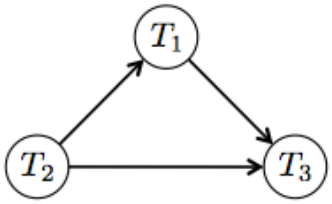
   (b) $r_2(A); r_1(C); r_2(B); r_3(B); w_2(B); r_1(A); r_3(C); w_3(C); w_1(A)$

   Answer the following questions:

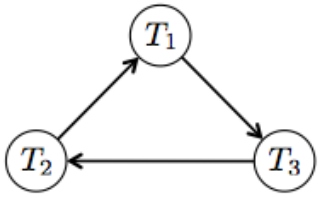
   i. What is the precedence graph for the schedule?

   ii. Is the schedule conflict-serializable? If so, what are the equivalent serial schedules?

   Schedule (a) is conflict-serializable because the precedence graph has no cycles. There is an arrow T2→T1 because of the conflict R2(A) . . . W1(A). There is an arrow T2→T3 because of the conflict W2(B)...R3(B). There is an arrow T1 → T3 because of the conflict R1(C)...W3(C). The only possible conflict-equivalent serial schedule is (T2, T1, T3).



   Schedule (b) is not conflict-serializable because the precedence graph has a cycle. There is an arrow T2→T1 because of the conflict R2(A) . . . W1(A). There is an arrow T3→T2 because of the conflict R3(B) . . . W2(B). There is an arrow T1 → T3 because of the conflict R1(C) . . . W3(C).

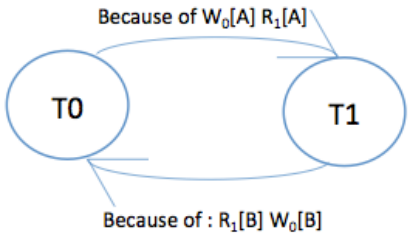


3. [6 points]

i. Consider the following two transactions and schedule (time goes from top to bottom). Is this schedule conflict-serializable? Explain why or why not.

T0	T1
$R_0(A)$	
$W_0(A)$	
	$R_1(A)$
	$R_1(B)$
	c_1
$R_0(B)$	
$W_0(B)$	
c_0	

The schedule is not conflict serializable because the precedence graph contains a cycle. The graph has an edge T0 → T1 because the schedule contains w0[A] → r1[A]. The graph has an edge T1 → T0 because the schedule contains r1[B] → wo[B].



ii. Show how 2PL can ensure a conflict-serializable schedule for the transactions above. Use the notation $L_i(A)$ to indicate that transaction i acquires the lock on element A and $U_i(A)$ to indicate that transaction i releases its lock on A.

Multiple solutions are possible. Two examples are shown below:

T0	T1
$L_0(A)$	
$L_0(B)$	
$R_0(A)$	
$W_0(A)$	
$U_0(A)$	
	$L_1(A)$
	$R_1(A)$
	$L_1(B)$; **DENIED…**
$R_0(B)$	
$W_0(B)$	
$U_0(B)$	
c0	…**GRANTED;** $R_1(B)$
	$U_1(A)$
	$U_1(B)$
	c1

T0	T1
$L_0(A)$	
$R_0(A)$	
$W_0(A)$	
	$L_1(A)$ **DENIED…**
$L_0(B)$	
$R_0(B)$	
$W_0(B)$	
$U_0(A)$	
$U_0(B)$	
c0	
	…**GRANTED;** $R_1(A)$
	$L_1(B)$;
	$R_1(B)$;
	$U_1(A)$
	$U_1(B)$
	c1

iii. Show how the use of locks without 2PL can lead to a schedule that is NOT conflict-serializable.

T0	T1
$L_0(A)$	
$R_0(A)$	
$W_0(A)$	
$U_0(A)$	
	$L_1(A)$
	$R_1(A)$
	$U_1(A)$
	$L_1(B)$
	$R_1(B)$
	$U_1(B)$
	c1
$L_0(B)$	
$R_0(B)$	
$W_0(B)$	
$U_0(B)$	
c0	

4. [6 points] The following schedules are presented to a timestamp-based scheduler. Assume that the read and write timestamps of each element start at 0 (RT(X) = WT(X) = 0), and the commit bits for each element are set (C(X)=1). Explain what happens as each schedule executes.

   a. st_1, st_2, st_3, $r_1(A)$, $r_2(B)$, $w_1(C)$, $r_3(B)$, $r_3(C)$, $w_2(B)$, $w_3(A)$

T_1	**T_2**	**T_3**	**A**	**B**	**C**
			RT=0 WT=0 C=1	RT=0 WT=0 C=1	RT=0 WT=0 C=1
st_1. Assume $TS(T_1) = 1$	st_2.Assume $TS(T_2)=2$	st_3. Assume $TS(T_3)=3$			
$r_1(A)$			RT=1		
	$r_2(B)$			RT=2	
$w_1(C)$					WT=1 C=0
		$r_3(B)$		RT=3	
		$r_3(C)$ **delay**			
	$w_2(B)$ $TS(T_2)<RT(B)$ **abort**				
		$w_3(A)$ **wait**			

More details about how the scheduler proceeds:

- st1, st2, st3: We assign timestamps in the order the transactions start: TS(T1) := 1, TS(T2) := 2, and TS(T3) := 3.
- r1(A): OK because element A hasn't yet been written (WT(A) = 0). Sets RT(A) := (TS(T1) =1).
- r2(B): OK because WT(B) = 0. Sets RT(B) := 2.
- w1(C): OK because C hasn't yet been read or written (RT(C) = WT(C) = 0). Sets WT(C) := 1 and C(C) := 0.
- r3(B): OK because WT(B) = 0. Sets RT(B) := 3.
- r3(C): We have (WT(C) = 1) ≤ 3, but C(C) = 0 because T1 has not yet committed or aborted. Delay T3 until C(C) = 1 or T1 aborts, then recheck timestamps and retry this action.
- w2(B): We have (RT(B) = 3) > 2, so allowing this write would cause a write too late anomaly, because T3 should have read the value about to be written by T2, which comes earlier in the serialization order. Rollback T2.
- w3(A): Wait until and unless T3 is unblocked and r3(C) above succeeds. If T3 is later aborted, then do not execute this action.

b. st_1, st_2, $r_1(A)$, $r_2(B)$, $w_2(A)$, com_2, $w_1(B)$

T_1	**T_2**	**A**	**B**
		RT=0 WT=0 C=1	RT=0 WT=0 C=1
st_1. Assume $TS(T_1) = 1$	st_2.Assume $TS(T_2)=2$		
$r_1(A)$		RT=1	
	$r_2(B)$		RT=2
	$w_2(A)$	WT=2 C=0	
	com_2	C=1	
$w_1(B)$ **abort**			

More details about how the scheduler proceeds:

- st1, st2: Let TS(T1) := 1 and TS(T2) := 2.
- r1(A): OK because WT(A) = 0. Sets RT(A) := 1.
- r2(B): OK because WT(B) = 0. Sets RT(B) := 2.
- w2(A): OK because (RT(A)=1)≤2 and (WT(A)=0)≤2. Sets WT(A):=2 and C(A):=0.
- com2: Set C(A) := 1 because T2 was the last transaction to write A, as determined by WT(A).
- w1(B): Rollback T1, because (RT(B) = 2) > 1.