CS5200 Homework10

Question1

In this problem I do not show the exact operation like “X=X-N”, “Y=Y+N”, “X=X+M”

Schedule1:

Time	Transaction1	Transaction2
T1	READ(X)	
T2	WRITE(X)	
T3		READ(X)
T4		WRITE(X)
T5	READ(Y)	
T6	WRITE(Y)	

Sechdule1 is conflict serializable.

Schedule2:

Time	Transaction1	Transaction2
T1	READ(X)	
T2	WRITE(X)	
T3	READ(X)	
T4	WRITE(Y)	
T5		READ(X)
T6		WRITE(X)

Schedule2 is conflict serializable (serial schedule).

Schedule3:

Time	Transaction1	Transaction2
T1		READ(X)
T2		WRITE(X)
T3	READ(X)	
T4	WRITE(X)	
T5	READ(Y)	
T6	WRITE(Y)	

Schedule3 is conflict serializable (serial schedule).

Schedule4:

Time	Transaction1	Transaction2
T1	READ(X)	
T2	WRITE(X)	
T3	READ(Y)	
T4		READ(X)
T5		WRITE(X)
T6	WRITE(Y)	

Schedule 4 is conflict serializable.

Schedule5:

Time	Transaction1	Transaction2
T1	READ(X)	
T2	WRITE(X)	
T3		READ(X)
T4	READ(Y)	
T5		WRITE(X)
T6	WRITE(Y)	

Schedule5 is conflict serializable.

Schedule6:

Time	Transaction1	Transaction2
T1	READ(X)	
T2	WRITE(X)	
T3		READ(X)
T4	READ(Y)	
T5	WRITE(Y)	
T6		WRITE(X)

Schedule6 is conflict serializable.

Schedule7:

Time	Transaction1	Transaction2
T1	READ(X)	
T2	WRITE(X)	
T3	READ(Y)	
T4		READ(X)
T5	WRITE(Y)	
T6		WRITE(X)

Schedule7 is conflict serializable.

Schedule8:

Time	Transaction1	Transaction2
T1	READ(X)	
T2		READ(X)
T3		WRITE(X)
T4	WRITE(X)	
T5	READ(Y)	
T6	WRITE(Y)	

Schedule8 is not conflict serializable.

Schedule 9:

Time	Transaction1	Transaction2
T1	READ(X)	

T2		READ(X)
T3	WRITE(X)	
T4		WRITE(X)
T5	READ(Y)	
T6	WRITE(Y)	

Schedule9 is not conflict serializable.

Schedule10:

Time	Transaction1	Transaction2
T1		READ(X)
T2	READ(X)	
T3		WRITE(X)
T4	WRITE(X)	
T5	READ(Y)	
T6	WRITE(Y)	

Schedule10 is not conflict serializable.

Schedule11:

Time	Transaction1	Transaction2
T1	READ(X)	
T2		READ(X)
T3		WRITE(X)
T4	WRITE(X)	
T5	READ(Y)	
T6	WRITE(Y)	

Schedule11 is not conflict serializable.

Schedule12:

Time	Transaction1	Transaction2
T1	READ(X)	
T2		READ(X)
T3	WRITE(X)	
T4	READ(Y)	
T5		WRITE(X)
T6	WRITE(Y)	

Schedule12 is not conflict serializable.

Schedule13:

Time	Transaction1	Transaction2
T1	READ(X)	
T2		READ(X)
T3	WRITE(X)	
T4	READ(Y)	

T5	WRITE(Y)	
T6		WRITE(X)

Schedule13 is not conflict serializable.

Schedule14:

Time	Transaction1	Transaction2
T1		READ(X)
T2	READ(X)	
T3	WRITE(X)	
T4	READ(Y)	
T5		WRITE(X)
T6	WRITE(Y)	

Schedule14 is not conflict serializable.

Schedule15:

Time	Transaction1	Transaction2
T1		READ(X)
T2	READ(X)	
T3	WRITE(X)	
T4	READ(Y)	
T5	WRITE(Y)	
T6		WRITE(X)

Schedule15 is not conflict serializable.

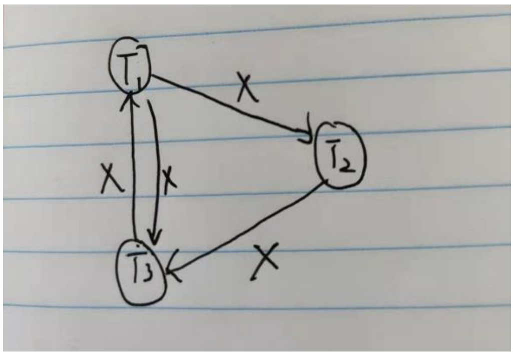
Question2

Schedule1 and schedule2 are not conflict serializable.
Schedule3 is conflict serializable.
Schedule3 can be converted into serial schedule as below:

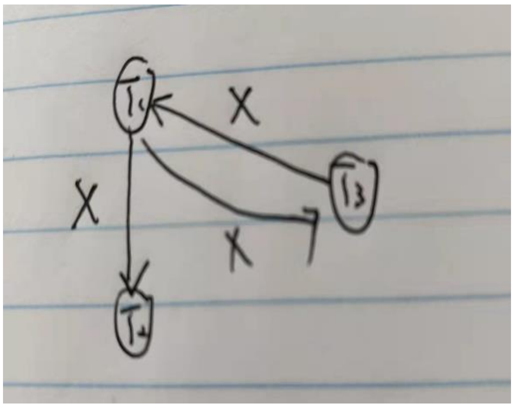
Serial schedule (converted from schedule3)
T2 Read(X)
T3 Read(X)
T3 Write(X)
T1 Read(X)
T1 Write(X)

Question3*

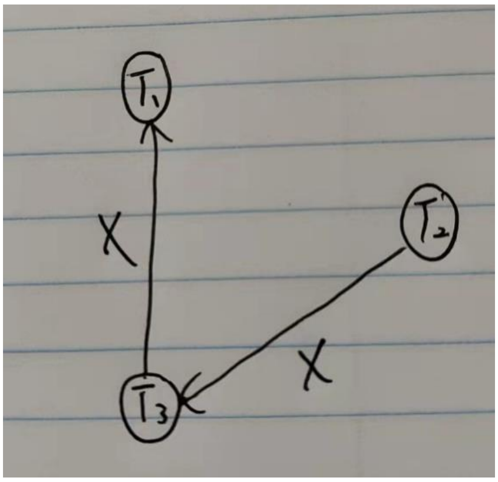
Schedule1:



Schedule2:



Schedule3:



Question4

Time	T_{17}	T_{18}
t_1	begin_transaction	
t_2	write_lock(bal_x)	begin_transaction
t_3	read(bal_x)	write_lock(bal_y)
t_4	$bal_x = bal_x - 10$	read(bal_y)
t_5	write(bal_x)	$bal_y = bal_y + 100$
t_6	write_lock(bal_y)	write(bal_y)
t_7	WAIT	write_lock(bal_x)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	⋮	WAIT
t_{11}	⋮	⋮

As we can see in the plot above, transaction T17 first begins and it applies a write_lock to lock the item bal_x, and then transaction T18 begins and it applies a write_lock to lock the item bal_y. After writing the bal_x, transaction T17 wants to apply write_lock on bal_y in the next, so it needs to wait T18 for committing/unlocking bal_y. In the same way, T18 wants to lock bal_x after writing bal_y but it needs to wait T17's committing/unlocking bal_x. And then T17 waits T18's committing/unlocking and T18 waits T17's committing/unlocking and there is no end for this waiting. It can be called deadlock.

In a word, deadlock is an impasse that may result when two (or more) transactions are each waiting for locks held by the other to be released.

Question5

I use tables to show the process of timestamp protocol (I will not show all time but some time).

When time = 3:

	X	Y	Z
READ		1	1
WRITE		1	

	Transaction A	Transaction B	Transaction C
TS		1	

When time = 4:

TS(C) = 4, WRITE(Y) = 1, READ(Y) = 1, so operation is accepted and executed, set READ(Y) = 4

When time = 5:

Set READ(Z) = 4

	X	Y	Z
READ		4	4
WRITE		1	

	Transaction A	Transaction B	Transaction C
TS		1	4

When time = 7:

	X	Y	Z
READ	6	4	4
WRITE	6	1	

	Transaction A	Transaction B	Transaction C
TS	6	1	4

When time = 8:

Originally, WRITE(Y) = 1, READ(Y) = 4, and TS(C) = 4, so transaction C's version of Y is not obsolete. Operation is accepted and executed. Set WRITE(Y) = TS(C).

	X	Y	Z
READ	6	4	4
WRITE	6	4	

	Transaction A	Transaction B	Transaction C
TS	6	1	4

When time = 9:

	X	Y	Z
READ	6	4	4
WRITE	6	4	4

	Transaction A	Transaction B	Transaction C
TS	6	1	4

When time = 10:

We can find that WRITE(X) = 6, and TS(B) = 1, so TS(B) < WRITE(X), so at this time transaction B need to be rolled back and restarted (using a later timestamp, i.e. 13).

	X	Y	Z
READ	6	4	4
WRITE	6	4	4

	Transaction A	Transaction B	Transaction C
TS	6	13	4

When time = 11:

TS(A) = 6, WRITE(Y) = 4, so transaction A's version of Y is not obsolete. Operation is accepted and executed. Set READ(Y) = 6.

	X	Y	Z
READ	6	6	4
WRITE	6	4	4

	Transaction A	Transaction B	Transaction C
TS	6	13	4

When time = 12:
TS(A) = 6, WRITE(Y) = 4, so set WRITE(Y) = 6.

	X	Y	Z
READ	6	6	4
WRITE	6	6	4

	Transaction A	Transaction B	Transaction C
TS	6	13	4

In conclusion, transaction B needs to be restarted at time = 10 (using BTO).

Question6
We can find that T1 commits before checkpoint, so T1 data has been flushed into disk. We do not need to do anything on operations of T1.
T4 commits after checkpoint before system crashes, so all operations of T4 in the log need to be redone. So T4 READ(D), T4 WRITE(D, 25, 15), T4 READ(A), T4 WRITE(A, 30, 20) are redone.
T2 and T3 do not commit before system crashes, so all operations of T2 and T3 are undone. So T2 READ(B), T2 WRITE(B, 12, 18), T3 WRITE(C, 30, 40), T2 READ(D), T2 WRITE(D, 15, 25) are undone.
T4 processes data item A and D, T2 processes data item B and D, but T4 commits before T2, and T2 reads (D) after T4 commits (A and D), so it is cascadeless schedule and this schedule is recoverable(D), this will not cause a rollback, so there will be no cascading rollback.

Question7
Cascading rollback is when a single transaction failure leads to a series of transaction rollbacks. Such a single transaction (which fails) must operate on the same data item with other transactions, and commit(if not fails) before other transactions read the data item.

Like the plot below, transaction 11 writes A and C before transaction 12 reads A,B and transaction 13 reads B,C. When t = 12, transaction 11 rollbacks, transaction 12 has written B but not written A, and transaction 13 has not written B,C. So when transaction 11 rollbacks, the value of A in transaction 12 and the value of B,C in transaction 13 are not correct, so transaction 12 and 13 need to be restarted. It is can be an example of cascading rollback.

Time	Transaction 11	Transaction 12	Transaction 13
t_1	Read(A)		
t_2	A = A + 100		
t_3	Read(C)		
t_4	Write(A)		
t_5	A = A - 40		
t_6		Read(B)	
t_7	Write(C)		
t_8		Read(A)	
t_9			Read(C)
t_{10}		B = B + 40	
t_{11}		Write(B)	
t_{12}	Rollback		C = C*20
t_{13}			Read(B)
t_{14}			Write(C)
t_{15}		A = A - 2	
t_{16}		Write(A)	
t_{17}		Commit	B = B - 8
t_{18}			Write(B)