Nathan Donaldson
CS5530

Assignment #5

Problem 1. [40 pts]

- Database
  - – Employee-Table(ssn, name, salary)
    - ∗ E1(132, Smith, 20K)
    - ∗ E2(456, Kelley, 40K)
    - ∗ E3(678, Johnson, 400K)
    - ∗ E4(792, Preeston, 40K)
    - ∗ E5(865, Johnson, 60K) ...
  - – DPT-table(dnumber, dname, budget)
    - ∗ D1(1, Marketing, 1M)
    - ∗ D2(2, Engineering, 2M)
    - ∗ D3(3, R&D , 4M)
    - ∗ D4(4, HR , 1M) ...

Part 1. Consider the following schedules
1) T1.R(E1), T1.W(E1), T2.R(E2), T2.W(E2), T1.R(D1), T1.commit, T2.commit
2) T1.R(E1), T1.W(E1), T2.R(E2), T2.W(E2), T2.W(E1), T1.R(E1), T1.commit, T2.commit
3) T1.R(E where salary>40K and salary<100k), T2.Insert(into E, (999,Bob, 50k)), T2.commit, T1.R(E where salarty > 40k and salary<100k), T1.commit
4) T1.R(E where salary>40k and salary<100k), T2.Insert(into E, (999, Bob, 50k)), T1.R(E where salary>40k and salary<100k), T2.commit, T1.commit.

For each schedule, please explain if it is valid in 1) the serializable isolation level: 2) the repeatable read isolation level.

1)

T1	R(E1)	W(E1)			R(D1)	Commit	
T2			R(E2)	W(E2)			Commit

It is valid in both serializable and repeatable read isolation. The values are independent of each transaction so it did not matter.

2)

T1	R(E1)	W(E1)				R(E1)	Commit	
T2			R(E2)	W(E2)	W(E1)			Commit

- Not valid for serializable because of uncommitted data read in T1's second R(E1) with no commit after T2's W(E1).
- Not valid for repeatable read because the second R(E1) in T1 would be preforming a dirty read after W(E1) in T2 with no commit.

3)

T1	R(E where salary >40k and salary<100k)			R(E where salary>40k and salary<100k)	Commit
T2		Insert(into E, (999, Bob, 50k))	Commit		

- Not Valid for serializable because of unrepeatable read of E in T1's second read after insert of from T2.
-Not valid, unrepeatable read problem.

T1	R(E where salary>40k and salary<100k)		R(E where salary>40k and salary<100k)		Commit
T2		Insert(into E, (999, Bob, 50k))		Commit	

- Serializable, the fact that T1 commit is after T2 commit makes this serializable
- Valid repeatable read because of no dirty reads, no changes, and allowing phantoms.

Part 2: Consider the following scenario:

T1	T2

T1.Read(E1)
T1.Update(set E1.salary=E1.salary*1.10)
T1.Read(Select * from Employee)
 	
T1.insert(Into E, (999, Bob, 50k))
T1.Read(Select * from Employee)
T1.commit
	
	Begin Transaction
	
	T2.Update(set D4.budget=2M)
	T2.Delete(E5)

System Crash	

Employee		
ID	Name	Salary
(E1.ID)	(E1.name)	(E1.salary * 1.10)
999	Bob	50k
(E5.ID)	(E5.name)	(E5.salary)

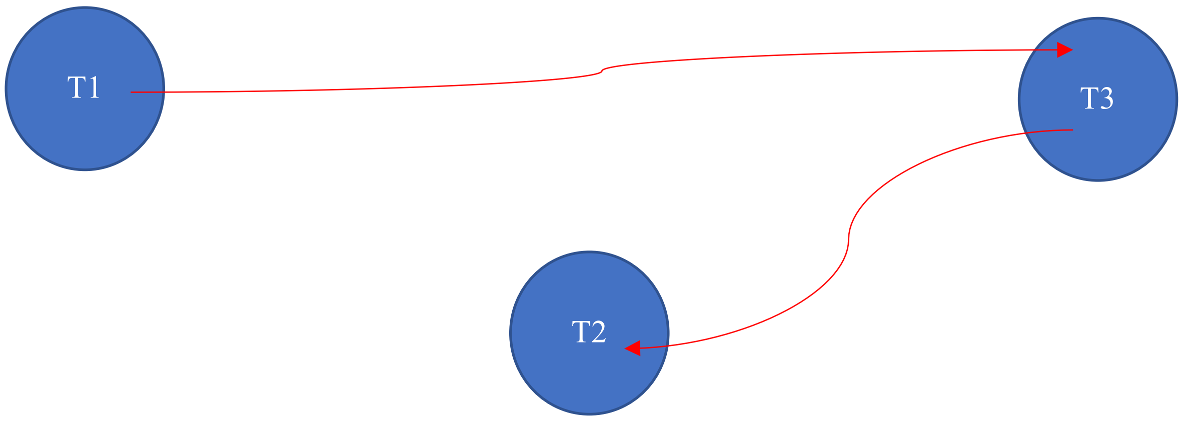
Problem 2. [60 pts]

1. In the following schedules, Ri(A) stands for a Read(A) operation by transaction i and Wi(A) stands for a Write(A) operation by transaction i. For each of the following schedules show if it is conflict-serializable and give a conflict-equivalent serial schedule if it is one. Hint: use its dependency graph.

(a) R1(A), R2(B), W3(A), R2(A), R1(B), T1.Commit, T2.Commit, T3.Commit
(b) R1(A), R2(B), W1(A), R3(C), W2(B), W3(C), R4(D), R4(A), W4(D), T1.Commit, T2.Commit, T3.Commit, T4.Commit
(c) R3(E), R1(D), W2(C), W3(A), R1(E), W4(B), R1(B), W3(E), R4(A), W4(C), T1.Commit, T2.Commit, T3.Commit, T4.Commit

a)

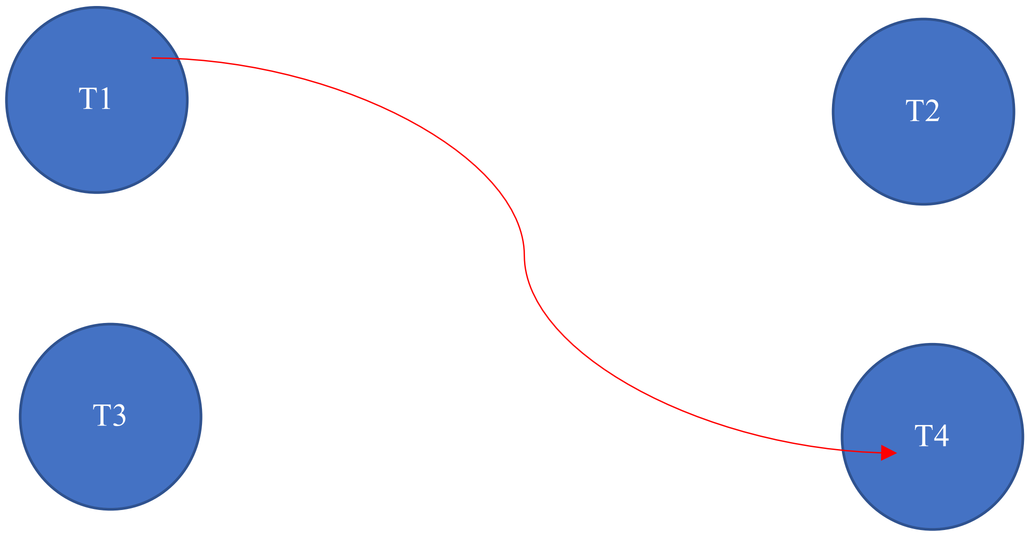
T1	R(A)				R(B)	Commit		
T2		R(B)		R(A)			Commit	
T3			W(A)					Commit



The conflict-equivalent serial schedule is T1 -> T3 -> T2

b)

T1	R(A)		W(A)							C			
T2		R(B)			W(B)						C		
T3				R(C)		W(C)						C	
T4							R(D)	R(A)	W(D)				C



The conflict-equivalent serial schedule is T1 -> T4, then T2 and T3 can run either way.

c)

T1		R(D)			R(E)		R(B)				C			
T2			W(C)									C		
T3	R(E)			W(A)				W(E)					C	
T4						W(B)			R(A)	W(C)				C

$$\begin{tikzcd}
T1 \arrow[r] & T3 \arrow[d] \\
T2 \arrow[r] & T4 \arrow[ul, bend left]
\end{tikzcd}$$

Not conflict-serializable

2. For the following 2 schedules, show if each is allowed in strict 2PL, and if not, what happens. An example is given: (the schedule will not be allowed in strict 2PL as it is). S1(A), R1(A), X2(A), W2(A), X1(B), R1(B), W1(B), T1.Commit, X2(B), W2(B), T2.Commit

T1	T2
S(A)	
R(A)	
	X(A), Blocked
X(B)	
R(B)	
W(B)	
Commit	
Release S(A)	
	W(A)
	X(B), Blocked
Release X(B)	
	W(B)
	Commit+Release its locks

OR

T1	T2
S(A)	
R(A)	
	X(A), Blocked
X(B)	
R(B)	
W(B)	
Commit	
Release S(A)	
Release X(B)	
	W(A)
	X(B)
	W(B)
	Commit+Release its locks

(a) S1(A), R1(A), S2(B), R2(B), S3(C), R3(C), X3(D), W3(D), T3.Commit, X2(C), W2(C), T2.Commit, X1(B), W1(B), T1.Commit

(b) X1(A), R1(A), X2(B), R2(B), X3(C), R3(C), S1(B), R1(B), S2(C), R2(C), S3(A), R3(A), W1(A), T1.Commit, W2(B), T2.Commit, W3(C), T3.Commit a)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T1	S(A)	R(A)											X(B)	W(B)	C
T2			S(B)	R(B)						X(C)	W(C)	C			
T3					S(C)	R(C)	X(D)	W(D)	C						

This schedule is in Strict 2PL
1 – 4: First there are shared locks on S(A), S(B), and S(C), each with a read following them. Everything is still fine.
7 – 8: An exclusive lock is put on D, which is okay because no other locks are on D.
9: T3 is committed and therefore shared lock C is released, and so is exclusive lock D.
10 – 11: A new exclusive lock is put on C, which is fine because no other locks are on C.
12: T2 is committed, therefore shared lock B is released, as well as exclusive lock C.
13: A new exclusive lock is put on B, and B has no locks so that is okay.
14: B is written to (Exclusive Lock so okay)
15: T1 is committed and shared lock A and exclusive lock B are released.

b)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T1	X(A)	R(A)					S(B)	R(B)					W(A)	C				
T2			X(B)	R(B)					S(C)	R(C)					W(B)	C		
T3					X(C)	R(C)					S(A)	R(A)					W(C)	C

This schedule is NOT in Strict 2PL
1 – 2: At the very beginning schedule there is an issue. There must be a shared lock on A before it can be read.*
3 – 4: The same for T2, it does not obtain a shared lock on B before reading. *
5 – 6: The same for T3, it does not obtain a shared lock on C before reading. *
7: There is already an exclusive lock on B, so T1 cannot open a shared lock. *
8: The read would be okay if that shared lock on b was valid; but it's not. *
9: The same goes with T2, shared lock cannot be opened on C because it already has an exclusive lock from T3. *
10: The read would be okay if that shared lock on c was valid: but it's not. *
11: The same goes with T3, shared lock cannot be opened on A because it already has an exclusive lock from T1. *
12: The read would be okay if that shared lock on A was valid, but it's not. *
13: The write on A would be okay since there is an exclusive lock on it earlier in T1.
14: After the commit, it seems the only lock that was valid was X(A), so that is released.
15: The write on B would be okay since there is an exclusive lock on it earlier in T2.
16: After the commit, it seems the only lock that was valid was X(B), so that is released.
17: The write on C would be okay since there is an exclusive lock on it earlier in T3.
18: After the commit, it seems the only lock that was valid was X(C), so that is released.

-It seems the last six were the only valid actions. Those actions were the exclusive locks on A, B, and C with transactions T1, T2, and T3 respectively. There was one write on each of those transactions for those values. So 3 locks, 3 writes, and 3 commits were done in this schedule.

3. For the following schedules, show what happens in each case. An example of non-strict 2PL is given (either is fine; in fact, there could be even more. But showing one is enough): S1(A), R1(A), X1(B), X2(A), W2(A), R1(B), W1(B), T1.Commit, X2(B), W2(B), T2.Commit

T1	T2
S(A)	
R(A)	
X(B)	
	X(A), Blocked
Release S(A)	
	W(A)
R(B)	
W(B)	
Release X(B)	
Commit	
	X(B)
	W(B)
	Release X(A), Release X(B)
	Commit

OR

T1	T2
S(A)	
R(A)	
X(B)	
Release S(A)	
	X(A)
	W(A)
R(B)	
W(B)	
Release X(B)	
Commit	
	X(B)
	Release X(A)
	W(B)
	Release X(B)
	Commit

(a) X1(B), W1(B), X2(A), W2(A), S2(B), R2(B), S1(A), R1(A), T1.Commit, T2.Commit. Using strict 2PL
(b) X1(B), W1(B), X2(A), W2(A), S2(B), R2(B), S1(A), R1(A), T1.Commit, T2.Commit. Using non-strict 2PL
(c) X1(B), W1(B), S1(A), S2(B), R2(B), R1(A), X2(A), W2(A), T1.Commit, T2.Commit. Using non-strict 2PL

a)

	1	2	3	4	5	6	7	8	9	10
T1	X(B)	W(B)					S(A)	R(A)	C	
T2			X(A)	W(A)	S(B)	R(B)				C

For strict 2PL this is not valid.
1 – 2: The exclusive lock and write on B is fine on T1
3 – 4: The exclusive lock and write on A is fine on T2
5 – 6: The shared lock on B is NOT fine on T2 because T1 already has a lock on B. *
7 – 8: The shared lock on A is NOT fine on T1 because T2 already has a lock on A. *
9 – 10: Both T1 and T2 are committed, release locks exclusive locks from B and A.

- Only two writes are done in this schedule, The write on B in T1 and the write on A in T2.

b)

	1	2	3	4	5	6	7	8	9	10
T1	X(B)	W(B)					S(A)	R(A)	C	
T2			X(A)	W(A)	S(B)	R(B)				C

Not valid for Non-Strict 2PL
1 – 2: Exclusive lock and write on T1 with value B is okay.
3 – 4: Exclusive lock and write on T2 with value A is okay.
5 – 6: Lock is release from X(B) on T1 and S(B) lock is gained on T2.
7 – 8: Since X(B) released its lock on B So that T2 could take B, it can no longer gain locks until it commits. *
9 – 10: Commits are done on T1 and T2

- T1 successfully does a write on B, but does not do a read on A due to Non-Strict 2PL not allowing gaining of locks after release (released B). T2 was able to do both a write on A and a read on B, but at the cost of losing read for A on T1.

c)

	1	2	3	4	5	6	7	8	9	10
T1	X(B)	W(B)	S(A)			R(A)			C	
T2				S(B)	R(B)		X(A)	W(A)		C

Valid for Non-Strict 2PL
1 – 2: Exclusive lock and write on B is okay for T1.
3: Shared lock gain on A is fine on T1.
4 – 5: Exclusive lock on B is released on T1, and shared lock on B in T2 is gained and read is done.
6: Read on A is fine on T1
7 – 8: Shared lock on A in T1 is released and exclusive lock is gained on A in T2. A write is also done on A in T2.
9 – 10: T1 and T2 commit

- Due to the allowing of releasing locks, T1 is allowed to release locks as time goes on so that T2 can use them as well. If there would have been an attempt to gain a new lock after the start of releasing, it would have not been valid.