HW5 Solution [1]

April 18, 2018

[1] CS 5530 Database Systems; Spring 2018 Instructor: Feifei Li, University of Utah

Problem 1.

Part 1.

1) It is serializable (i.e. valid in the serializable isolation level). Its effect is the same as T_1 then T_2. It is also repeatable read, since there are no dirty reads, and no items have been read multiple times in the same transaction (i.e., no need to check the 2nd condition for repeatable read).

2) It is NOT serializable. Its effect is not the same to either T_1 then T_2 or T_2 then T_1. In essence, T_1 needs to read the value of E1 written by T_2. But if we were to execute T_2 first then T_1, T_1 will have to read the value of E1 written by itself.

It is also clearly NOT reapeatable read, since it has dirty read (the second T1.R(E1)) and item read multiple times could have changed value (E1 read twice by T1, and in-between has been updated by T2).

3) It is NOT serializable, but it is repeatable read. It is not serializable since (a) it is not the same to T_1 then T_2 (in this case, both select statements in T_1 will not be able to read the inserted record by T_2); (b) and it is not the same to T_2 then T_1 (in this case, both select statements in T_1 will have to read the inserted record by T_2).

It is repeatable read for two reasons. First, there is no dirty read by T_1 (since the second select statement from T_1 will read the newly inserted record by T_2, but it's been committed already).

Secondly, all items read by T_1 **MULTIPLE TIMES** from the two select statements **DO NOT** change values. Note that the newly inserted record by T2 is only read ONCE by T1.

4) It is NOT serializable for the same reason as stated in 3).

It is also NOT repeatable read, because now T_1 has a dirty read (T_2 only commits after the second select statement in T_1).

Part 2.

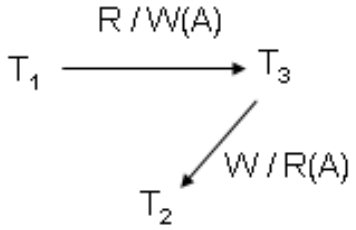
The final state of table E is as follows:

- E1(132, Smith, 22K)
- E2(456, Kelley, 40K)
- E3(678, Johnson, 400K)
- E4(792, Preeston, 40K)
- E5(865, Johnson, 60K)
- E6(999, Bob, 50K) ...

Problem 2.

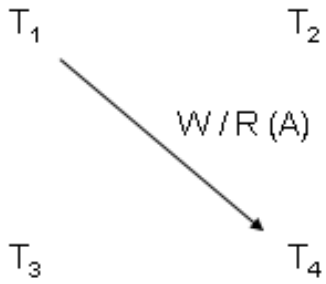
- **(a)** yes. since there are no cycles as shown in Figure 1(a). The serial schedule is $< T_1, T_3, T_2 >$.
- **(b)** yes. since there are no cycles in Figure 1(b). The serial schedule is $< T_1, T_2, T_3, T_4 >$.
- **(c)** no. since is a cycle in Figure 1(c).

T_1	T_2	T_3
R(A)		
	R(B)	
		W(A)
	R(A)	
R(B)		



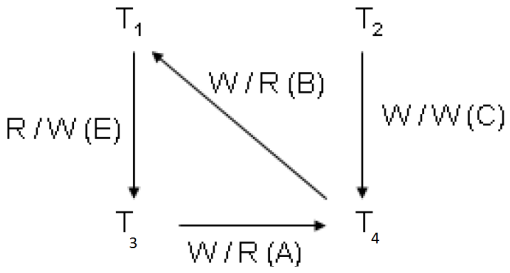
(a) 1.1(a) Given schedule and conflict graph

T_1	T_2	T_3	T_4
R(A)			
	R(B)		
W(A)			
		R(C)	
	W(B)		
		W(C)	
			R(D)
			R(A)
			W(D)



(b) 1.1(b) Given schedule and conflict graph

T_1	T_2	T_3	T_4
		R(E)	
R(D)			
	W(C)		
		W(A)	
R(E)			
			W(B)
R(B)			
		W(E)	
			R(A)
			W(C)



(c) 1.1(c) Given schedule and conflict graph

2.2 (a) this schedule is allowed.

T_1	T_2	T_3
S(A)		
R(A)		
	S(B)	
	R(B)	
		S(C)
		R(C)
		X(D)
		W(D)
		Commit
		Release X(D)
		Release S(C)
	X(C)	
	W(C)	
	Commit	
	Release X(C)	
	Release S(B)	
X(B)		
W(B)		
Commit		
Release X(B)		
Release S(A)		

Figure 1: 1.2(a) Schedule with locking

(b) A schedule with locks is shown in Figure 2. The transactions go into a deadlock.

T_1	T_2	T_3
X(A)		
R(A)		
	X(B)	
	R(B)	
		X(C)
		R(C)
S(B), Blocked		
	S(C), Blocked	
		S(A), Blocked
Deadlock	Deadlock	Deadlock

Figure 2: 1.2(b) Schedule with locking

2.3 (a) this schedule is not allowed.

T_1	T_2
X(B)	
W(B)	
	X(A)
	W(A)
	S(B), Blocked
S(A), Blocked	
Deadlock	Deadlock

Figure 3: 1.3(a) Schedule with locking

(b) no difference, same as above. since no one can release lock early to get rid of the deadlock, as they both still need to acquire lock(s) down the road.

T_1	T_2
X(B)	
W(B)	
	X(A)
	W(A)
	S(B), Blocked
S(A), Blocked	
Deadlock	Deadlock

Figure 4: 1.3(b) Schedule with locking

(c) This schedule is allowed in non-strict 2PL. It is shown in Figure 2.

T_1	T_2
X(B)	
W(B)	
S(A)	
Release X(B)	
	S(B)
	R(B)
R(A)	
Release S(A)	
	X(A)
	Release S(B)
	W(A)
	Relase X(A)
Commit	
	Commit

Figure 5: 1.3(c) Schedule with locking