CSCI 6315 Applied Database Systems
ASSIGNMENT 5: Transaction Management
Due is 05/03/2020 00:00

Ulvi Bajarani
Student ID 20539914
E-mail: ulvi.bajarani01@utrgv.edu

Questions and Answers:

Problem 1. Explain the distinction between the terms *<u>serial schedule</u>* and *<u>serializable schedule</u>*.

Answer 1. While serial schedule contains the transactions that complete one after another (for example, T_1, then T_2, then T_3, etc.), the serializable schedule is the *non-serial schedule* (that contains transactions loading concurrently: this might lead to some problems.) which, however, might be transformed to the serial schedule. It should be mentioned that not all non-serial schedules are serializable.

Problem 2. Consider the following two transactions:

T_1:	read(A);
	read(B);
	if $A = 0$ then $B = B + 1$;
	write B;
T_2:	read(B);
	read(A);
	if $B = 0$ then $A = A + 1$;
	write A;

Figure 1. Two transactions T_1 and T_2

Let the consistency requirement be $A = 0 \vee B = 0$ with $A = B = 0$ the initial values.

a. Show that every serial execution involving these two transactions preserves the consistency of the database.

b. Show a concurrent execution of T_1 and T_2 that produces a nonserializable schedule.

c. Is there a concurrent execution of T_1 and T_2 that produces a serializable schedule?

Answer 2.

a. While T_1 executes first, the if branch in T_2 returns false, and vice versa. The consistency requirement is $A = 0 \vee B = 0$ with $A = B = 0$ the initial values, so one of either A or B remains 0, which allows to preserve the consistency.

b. The table below might be the solution. Swapping of the order of read(B); and write B; leads the different results :

T_1	T_2
read(A);	
	read(B);
	read(A);
read(B);	
if $A = 0$ then $B = B + 1$;	
	if $B = 0$ then $A = A + 1$;
write B;	
	write A;

c. There is no solution, because in the concurrent case, writes will not be performed. As a result, the reads in either T_2 and T_1 gets the old values, not changed

Problem 3. Consider the precedence graph in Figure 2. Is the corresponding schedule serializable?

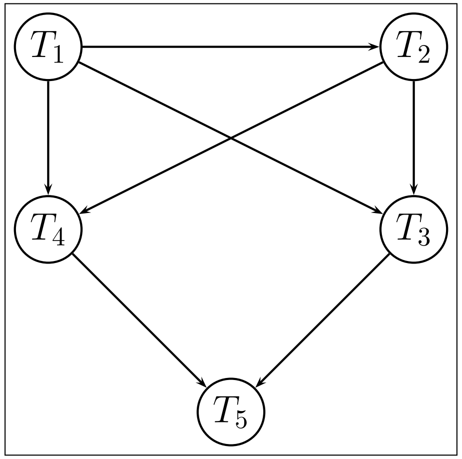

Figure 2. Precedence graph

Answer 3. The corresponding schedule is serializable, because there is no cycle between the transactions. The serial schedule is the topological sorting, for example, T_1, T_2, T_3, T_4, T_5.

Problem 4. Show that two-phase locking protocol ensures conflict serializability, and that transaction can be serialized according to their lock points.

Answer 4. If 2PL didn't ensure the conflict serializability, there would be T_0, T_1, T_2, T_3, T_4 ... T_n transactions that are non-serializable. In that case, there should be the cycle in the precedence graph, which should lead to T_0, T_1, T_2, ... T_1, T_0 precedence graph. Let α_i be the time of occured lock. If 2PL existed in the cyclic precedence graph, the locking time should be $\alpha_0 < \alpha_1 ... < \alpha_0$, which is the contradiction. Thus, 2PL cannot create non-serializable schedule. Also, $T_i \rightarrow T_j$ and $\alpha_i < \alpha_j$ means that the lock point ordering is a topological sort ordering the precedence graph, which means the serializability.

Problem 5. Consider transactions T_1 and T_2 in Figure 1. Add lock and unlock instructions to them so that they observe the two-phase locking protocol. Can the execution of these two transactions result in a deadlock?

Answer 5. The 2PL-protocol-observing case:

T_1	T_2
Lock-S(A);	
read(A);	
Lock-X(B);	
read(B);	
if $A = 0$ then $B = B + 1$;	
write B;	
Unlock-S(A);	
Unlock-X(B);	
	Lock-S(B);
	read(B);
	Lock-X(A);
	read(A);
	if $B = 0$ then $A = A + 1$;
	write A;
	Unlock-S(B);
	Unlock-X(A);

The deadlock case. Neither T_1 nor T_2 might make a progress:

T_1	T_2
Lock-S(A);	
read(A);	
	Lock-S(B);
	read(B);
Lock-X(B);	
read(B);	
	Lock-X(A);
	read(A);
if $A = 0$ then $B = B + 1$;	
write B;	
Unlock-X(B);	
Unlock-S(A);	
	if $B = 0$ then $A = A + 1$;
	write A;
	Unlock-S(B);
	Unlock-X(A);

Problem 6. Show that there are schedules that are possible under the two-phase locking protocol, but are not possible under the timestamp protocol, and vice versa.

Answer 6. the schedule with 2PL that is not possible in the Timestamp Protocol (due to step 5, where $TS(T_0) <$ W-Timestamp(B)):

T_0	T_1
Lock-S(A);	Lock-X(B);
read(A);	write B;
	unlock(B);
Lock-S(B);	
read(B);	
unlock(A);	
unlock(B);	

the schedule with Timestamp Protocol that is not possible in the 2PL Protocol, because it requires T_1 to unlock A between the steps 2 and 3, and lock B between steps 4 and 5:

T_0	T_1	T_2
write A;		write A;
	write A;	
write B;		
	write B;	

Problem 7. Explain the purpose of the checkpoint mechanism. How often should checkpoints be performed? How does the frequency of checkpoints affect

- System performance when no failure occurs
- The time it takes to recover from a system crash

- The time it takes to recover from a disk crash

Answer 7.

- Decreases the system performance (negatively);
- Decreases the time of system recovery (positively);
- Decreases the time of disk recovery (positively);

Problem 8. When the system recovers from a crash, it constructs an undo-list and a redolist. Explain why log records for transactions on the undo list must be processed in reverse order, while those log records for transactions on the redo-list are processed in a forward direction.

Answer 8. The answer is trivial: if the undo-list was processed in the forward order, it would give wrong values to the data that is updated with several transactions. The same is right for the redo-list processed in the reverse direction.