BSCCS2001: Graded Assignment with Solutions
Week 10

1. For the given statements:

   - A transaction must keep the database consistent during execution.
   - A transaction when starting to execute must see a consistent database.

   Which one of the following is the correct choice? [MCQ: 1 point]

   ◯ Both statements are correct

   ✓ First statement is wrong

   ◯ Both statements are wrong

   ◯ Second statement is wrong

   Solution:
   According to the **Consistency** property of transacti ons, a transaction must start working on a consistent database and it must keep the database consistent after it finishes execution. But a transaction while getting executed may not keep the database in a consistent state. Therefore, the first statement is wrong and option 2 is correct.

2. Which of the following methods always ensures deadlock free schedules?

   [MSQ: 2 points]

   ✓ Timestamp ordering

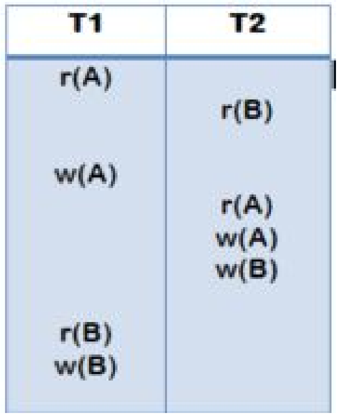
   ◯ 2-Phase Locking

   ◯ Rigorous 2-Phase locking

   ✓ Tree protocol or partial ordering

   Solution: Timestamp ordering protocol and tree protocol ensures deadlock free schedules.
   Any variation of 2-phase locking does not ensure deadlock free schedules, it only guarantees conflict serializable schedules.

Consider the following schedule **S** and answer questions 3 and 4.



T1	T2
r(A)	
	r(B)
w(A)	
	r(A)
	w(A)
	w(B)
r(B)	
w(B)	

Figure 1: S

3. Which of the following is true for schedule **S**? [MSQ: 2points]

   ○ Schedule **S** is Conflict serializable

   ○ Schedule **S** is View serializable

   ✓ Schedule **S** is not Conflict Serializable

   ✓ Schedule **S** is not View Serializable

Solution: Pairs of conflicting instructions : $w1(A) \rightarrow r2(A)$, $w1(A) \rightarrow w2(A)$, $w2(B) \rightarrow r1(B)$, $w2(B) \rightarrow w1(B)$
These conflicting instructions create a cyclic dependency of T1 on T2 and vice-versa.
Hence, the schedule is not Conflict serializable.
There are no blind writes. Hence, the given schedule is not View serializable.

4. Which of the following is correct? [MSQ: 3 points]

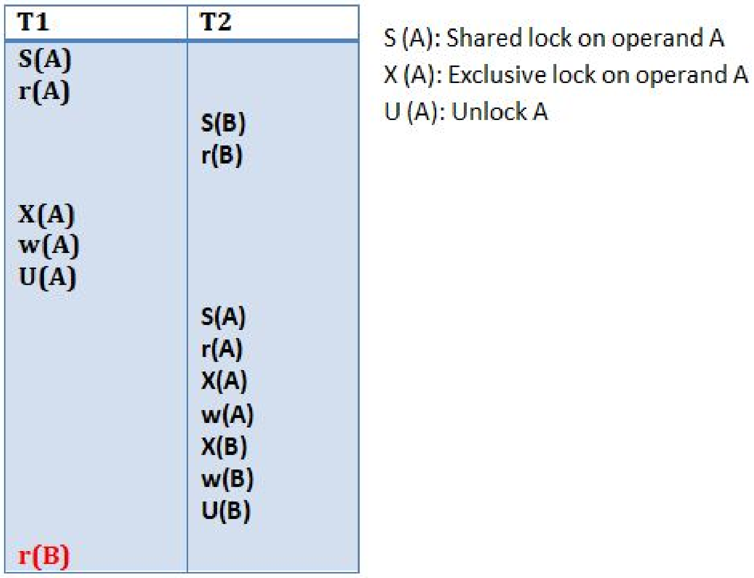
   ○ The schedule is 2-phase lockable

   ✓ The schedule is not 2-phase lockable

   ○ Schedule is Serializable

   ✓ Schedule is not Serializable

Solution: When we try to apply 2 phase locking on S.



T1	T2
S(A)	
r(A)	
	S(B)
	r(B)
X(A)	
w(A)	
U(A)	
	S(A)
	r(A)
	X(A)
	w(A)
	X(B)
	w(B)
	U(B)
r(B)	

S (A): Shared lock on operand A
X (A): Exclusive lock on operand A
U (A): Unlock A

Figure 2: S

Before executing the red colored r(B) instruction, T1 needs to have a shared lock. However, since T1 already has started unlocking, it can't gain a new lock according to 2 phase lock protocol. Thus, S is not 2 phase lockable.
Pairs of conflicting instructions : $w1(A) \rightarrow r2(A)$, $w1(A) \rightarrow w2(A)$, $w2(B) \rightarrow r1(B)$, $w2(B) \rightarrow w1(B)$
These conflicting instructions create a cyclic dependency of T1 on T2 and vice-versa. Hence, the schedule is not Conflict serializable.
There are no blind writes hence schedule is not view serializable
Hence, option 2 and 4 are correct.

5. For the given schedule **S**, answer the question that follows.
S: *r5(Z), w1(Y), r2(Y), w3(Y), r4(Y), w2(P), r5(P), w4(X), r1(Q), r5(X), w5(Y)*
Which of the following is the correct choice? [MCQ: 1 point]

   ◯ This schedule is not conflict serializable

   ◯ This schedule is not serializable

   ◯ This schedule is neither view nor conflict serializable

   [✓ None of the above]

Solution: The schedule is conflict serializable to the serial schedule
$T1 \rightarrow T2 \rightarrow T3 \rightarrow T4 \rightarrow T5$
A conflict serializable schedule is implicitly view serializable
Hence, option 4 is the correct choice.

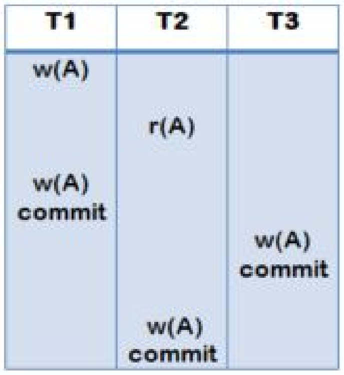
6. What is the number of serial schedules to which **S** is serializable?
S: *w1(P), r2(P), w3(P), r4(P), w5(P)*

[MCQ 2points]

- ○ 0
- [✓ 2]
- ○ 1
- ○ 3

Solution: Since the read-write relations should be maintained in a view serial schedule,
Transaction T2 must be after T1.
Transaction T4 must be after T3
Last write of attribute P must be preserved. Therefore, T5 must be the last transaction in a serial schedule and we have two possible serial schedules which are view equivalent:
$T1 \rightarrow T2 \rightarrow T3 \rightarrow T4 \rightarrow T5$
$T3 \rightarrow T4 \rightarrow T1 \rightarrow T2 \rightarrow T5$

7. For Schedule **S** shown in Figure 5, choose the correct option.



T1	T2	T3
w(A)		
	r(A)	
w(A) commit		
		w(A) commit
	w(A) commit	

Figure 3: S

[MCQ: 1 point]

- ✓ The schedule is Recoverable with cascading rollback
- ◯ The schedule is Cascadeless Recoverable
- ◯ The schedule is Strict
- ◯ The schedule is not Recoverable

Solution: Transaction T2 reads the value of A after T1 writes it but before T1 commits. Therefore, the schedule is recoverable but with cascading rollbacks.
The schedule is not strict, a strict schedule is cascadeless recoverable.
Hence, option 1 is correct.

8. Given below is a transaction that transfers an inventory of 1000 cartons from warehouse A to warehouse B. [MSQ: 2 points]

Step	Operation
1	read(A)
2	$A := A - 1000$
3	write(A)
4	read(B)
5	$B := B + 1000$
6	write(B)

Table 1: Transaction 1

Choose the correct option(s) regarding Transaction 1.

- ◯ The database must always be consistent just after Step 3 but before Step 5.
- ◯ The database must always be consistent just after Step 4 but before Step 6.
- ✓ The database may be temporarily inconsistent just after Step 3 but before Step 5.
- ✓ The database may be temporarily inconsistent just after Step 4 but before Step 6.
- ◯ The database has to be consistent throughout the transaction due to ACID properties.

Solution: Transaction 1 starts at Step 1 and ends at Step 6. According to the *consistency* property (C in ACID), the database has to be consistent before and after a transaction. It can afford to have inconsistent states in the intermediate steps. Hence, Options 3 and 4 are correct.

9. Let $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5\}$ be a set of transactions. In each of the options below, we give a table that shows how each transaction is waiting for release of locks from other transactions. Which of the following allocations is deadlock-free?

[MSQ: 3 points]

○

	T1	T2	T3	T4	T5
T1		W	W		
T2			W		
T3					W
T4	W		W		
T5				W	

○

	T1	T2	T3	T4	T5
T1		W			
T2			W		
T3				W	
T4					W
T5	W				

✓

	T1	T2	T3	T4	T5
T1		W	W	W	
T2			W		
T3					
T4			W		
T5		W	W	W	

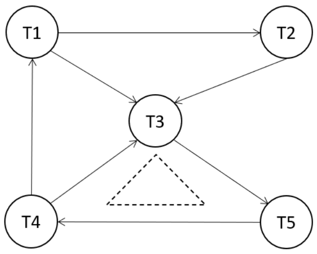
○

	T1	T2	T3	T4	T5
T1		W			
T2	W		W		
T3		W			
T4					W
T5				W	

Solution: Given below are the *wait-for* graphs corresponding to the allocation in each option.

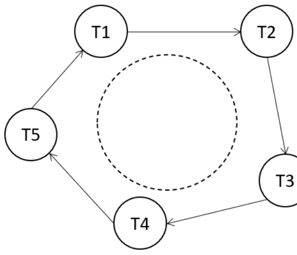
Option 1:

	T1	**T2**	**T3**	**T4**	**T5**
T1		W	W		
T2			W		
T3					W
T4	W		W		
T5				W	



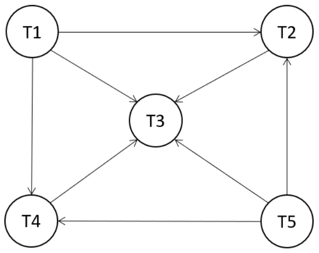
Option 2:

	T1	**T2**	**T3**	**T4**	**T5**
T1		W			
T2			W		
T3				W	
T4					W
T5	W				



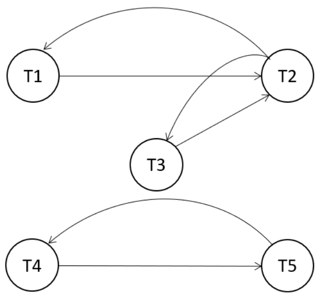
Option 3:

	T1	**T2**	**T3**	**T4**	**T5**
T1		W	W	W	
T2			W		
T3					
T4			W		
T5		W	W	W	



Option 4:

	T1	**T2**	**T3**	**T4**	**T5**
T1		W			
T2	W		W		
T3		W			
T4					W
T5				W	



Since options 1,2 and 4 have cycles in them, they indicate a deadlock. Option 3 shows an acyclic graph and hence it does not indicate a deadlock.

10. Choose the correct output obtained on running the given SQL statements on Table **Employee**.

[MCQ: 2 points]

EID	EName
E01	Arthur
E02	Raina
E03	Meena
E04	Arthur
E06	Joey

Table **Employee**

```
SQL> SAVEPOINT SP1;
SQL> UPDATE Employee SET EName='Jainie'
     WHERE EID='E06';
SQL> COMMIT;
SQL> SAVEPOINT SP2;
SQL> DELETE FROM Employee WHERE EID='E02';
SQL> SAVEPOINT SP3;
SQL> UPDATE Employee SET EName='Raina'
     WHERE EID='E04';
SQL> ROLLBACK TO SP1;
```

✓

EID	EName
E01	Arthur
E02	Raina
E03	Meena
E04	Arthur
E06	Jainie

○

EID	EName
E01	Arthur
E03	Meena
E04	Arthur
E06	Jainie

○

EID	EName
E01	Arthur
E03	Meena
E04	Raina
E06	Jainie

○

EID	EName
E01	Arthur
E02	Raina
E03	Meena
E04	Arthur
E06	Joey

Solution: Even though the name 'Joey' was updated to 'Jainie' after savepoint SP1, this modification is not rolled back because a commit has been done immediately after modifying 'Joey' to 'Jainie'. Hence, all transactions that happened after the commit are rolled back.

11. In Schedule **S**, there are 3 transactions namely T1, T2, T3. Each of these transactions will write to a file in secondary storage (hard-disk). For writing, they need access to a system component C which has a total of 'x' instances. If T1 requires 3 instances, T2 requires 5 instances, and T3 requires 3 instances of C to complete the transaction, then what should be the minimum value of 'x' so that no deadlock will occur when 2-phase locking protocol is used in the system?

[MCQ: 3 points]

- ○ 5
- ○ 3
- [✓ 9]
- ○ deadlock can not be avoided by increasing the count of a component's instance

Solution: Obviously if we have 5+3+3 = 11 instances, then deadlock can not occur. But deadlock can be avoided by using lesser number of instances.
Let's check with 5 instances.
Suppose T1 takes 2 instances simultaneously. T2 takes 3 instances, in this situation neither T1 nor T2 can complete the transaction and both will hold the instances already gained and will wait indefinitely for the leftover requirement.

To avoid this, we can use the following strategy :
Suppose T1 gains 2 out of 3 instances, T2 gains 4 out of 5 instances and T3 gains 2 out of 3 instances, i.e. there are total 8 instances. Then also deadlock is possible. If we add one more to it and make x = 9, then deadlock can be avoided. For example in the previous scenario the 9^{th} instance can satisfy either of the three transactions and thus the instances which were already held by that transaction are free since that transaction is completed. Now the other two transactions can have those free instances to complete their execution.

Therefore, answer is 9.