Q1
0.3 Points

Answer true or false.

Q1.1
0.1 Points

If T1 wants an SIX lock on a data item Q, it *must* have an SIX lock on Q's parent in the granularity hierarchy.

- (●) True
- ( ) False

Q1.2
0.1 Points

Consider a relation R(A, B), and a tuple (10, 20) in it. The read query: "select count(*) from R where A < 80", does not conflict with a transaction that tries to update that tuple to: (90, 20).

- ( ) True
- (●) False

Q1.3
0.1 Points

If we use strict 2 phase locking, deadlocks can not happen.

- ( ) True
- (●) False

Q2

0.3 Points

For each of the following schedules, answer if they are "conflict-serializable". All read/write instructions of a transaction are shown here.

Q2.1

0.1 Points

T1	*T2*	*T3*	*T4*
read (A)			
write(A)			
	read(A)		
		read(A)	
		read(B)	
		write(B)	
			read(C)
	write(A)		
			write(C)
read(B)			
	read(B)		
read(C)			

T1	*T2*	*T3*	*T4*
	write(B)		

- (●) Conflict-serializable
- ( ) Not conflict-serializable

Q2.2

0.1 Points

T1	*T2*	*T3*	*T4*	*T5
read (A)				
		read(A)		
	read(A)			
write(A)				
		read(B)		
		write(B)		
	write(A)			
read(B)				
				read(C)
	read(B)			
				write(C)

T1	*T2*	*T3*	*T4*	*T5
read(C)				
			read(C)	
			write(C)	
	write(B)			

○ Conflict-serializable

◉ Not conflict-serializable

Q2.3

0.1 Points

T1	*T2*	*T3*	*T4*
		read(A)	
	read(A)		
read (A)			
write(A)			
		read(B)	
			read(C)
	write(A)		
read(B)			

T1	*T2*	*T3*	*T4*
	read(B)		
	write(B)		
		write(B)	
read(C)			
			write(C)

- ○ Conflict-serializable
- ◉ Not conflict-serializable

Q3

0.2 Points

The following schedule is not conflict-serializable. Explain why this schedule will not be permitted under Strict 2PL. Assume all transactions request an appropriate lock on a data item just before the first read/write instruction on that data item, and release all locks after their last instruction in the schedule -- which transactions/instructions would not have been allowed to proceed as shown?

T1	*T2*	*T3*
read (A)		
write(A)		
		read(A)

T1	*T2*	*T3*
	read(A)	
		read(B)
	write(A)	
read(B)		
	read(B)	
	write(B)	
		write(B)
read(C)		

T1 must obtain exclusive lock to write A, then T1 must release exclu sive lock to allow T2/T3 to read A. This means T1 cannot obtain an y more locks. So in order to read b and c, it must have obtained a shared lock previous to releasing exclusive A lock. Then B and C shared lock are released after reading each.
The issue:
t2 must obtain exclusive A lock to write(A). This means T3 has to h ave released its shared A lock by then, and cannot obtain any mor e locks. For t3 to read B, it must've obtained a shared lock before i t released its shared A lock. Then, it cannot later obtain an exclusiv e B lock to write(b).

Q4

0.2 Points

The following schedule is conflict-serializable. List out at least four equivalent serial schedules for this schedule. You can do this by drawing the precedence graph, and then finding sequences of

transactions where all the edges go from left to right. As an example: T1, T2, T3, T4, T5 is clearly not an equivalent serial schedule because there is an edge from T3 to T1 (due to A).

T1	*T2*	*T3*	*T4*	*T5*
read (A)				
		read(A)		
write(A)				
	read(A)			
		read(B)		
		write(B)		
read(B)				
	write(A)			
				read(C)
	read(B)			
				write(C)
read(C)				
			read(C)	
			write(C)	
	write(B)			

53124, 35124, 53142, 35142

Q5
1 Point

For the next few questions, consider the granularity hierarchy shown in the figure here.

Further assume that there are two transactions T1 and T2 that are currently holding the following locks:
T1: IS(DB), IS(R1), S(t1)
T2: IS(DB), IX(DB), SIX(R2), X(t4)

Q5.1
0.1 Points

T3 wants to update t2. Which of the following is the minimal set of lock requests it should make?
Here minimal means that T3 would lock the required items in the least restrictive lock possible. For example, if a transaction wants to take an X lock on R2, then taking a X lock on DB is more restrictive than taking an IX lock on DB and an X lock on R2 (and hence the answer would be latter).

- ○ IS(DB), IX(R1), X(t2)
- ◉ IX(DB), IX(R1), X(t2)
- ○ SIX(DB), IX(R1), X(t2)
- ○ IX(DB), X(R1)

Q5.2
0.1 Points

T3 wants to execute the following SQL query:

update R2 set R2.A = 100 where R2.B = 10;

Assume that only a few tuples in R2 satisfy the condition. Which of the following is the minimal set of lock requests T3 should make?

- ○ IS(DB), S(R2), X locks on tuples that satisfy condition
- ○ IX(DB), X(R2)
- ◉ IX(DB), SIX(R2), X locks on tuples that satisfy condition
- ○ IS(DB), IX(DB), SIX(R2), X locks on tuples that satisfy condition
- ○ SIX(DB), SIX(R2), X locks on tuples that satisfy condition

Q5.3

0.1 Points

T1 wants to now update t3. Which of the following is the minimal set of new lock requests or lock upgrade requests it should make?

- ○ new lock request for X(t3)
- ◉ request to upgrade to IX(DB), then IX(R1), and then request X(t3)
- ○ request to upgrade to SIX(DB), then SIX(R1), and then request X(t3)
- ○ request to upgrade to IX(R1), and then request X(t3)

Q5.4

0.1 Points

T3 wants a SIX lock on DB. Is it compatible with the existing locks being held by the two transactions (i.e., can it be granted immediately or would T3 have to wait)?

- ○ Compatible -- T3 can be granted the lock immediately
- ◉ Not compatible -- T3 must wait
- ○ T3 will not be allowed to request the lock because it is not following multi-granularity locking rules

Q5.5

0.1 Points

T3 wants a IX lock on DB. Is it compatible with the existing locks being held by the two transactions (i.e., can it be granted immediately or would T3 have to wait)?

- ◉ Compatible -- T3 can be granted the lock immediately
- ○ Not compatible -- T3 must wait
- ○ T3 will not be allowed to request the lock because it is not following multi-granularity locking rules

Q5.6

0.1 Points

T3 wants a IS lock on R1. Is it compatible with the existing locks being held by the two transactions (i.e., can it be granted immediately or would T3 have to wait)?

- ◉ Compatible -- T3 can be granted the lock immediately
- ○ Not compatible -- T3 must wait
- ○ T3 will not be allowed to request the lock because it is not following multi-granularity locking rules

Q5.7

0.1 Points

T2 wants to upgrade its SIX lock on R2 to an X lock. Is it compatible with the existing locks being held by the two transactions (i.e., can it be granted immediately or would T2 have to wait)?

- ○ Compatible -- T2 can be granted the lock immediately
- ◉ Not compatible -- T2 must wait
- ○ T2 will not be allowed to request the lock because it is not following multi-granularity locking rules

Q5.8

0.1 Points

T1 wants to upgrade its IS lock on DB to a SIX lock. Is it compatible with the existing locks being held by the two transactions (i.e., can it be granted immediately or would T1 have to wait)?

- ○ Compatible -- T1 can be granted the lock immediately
- ◉ Not compatible -- T1 must wait
- ○ T1 will not be allowed to request the lock because it is not following multi-granularity locking rules

Q5.9

0.1 Points

T1 wants to upgrade its IS lock on DB to an IX lock. Is it compatible with the existing locks being held by the two transactions (i.e., can it be granted immediately or would T1 have to wait)?

- ◉ Compatible -- T1 can be granted the lock immediately
- ○ Not compatible -- T1 must wait
- ○ T1 will not be allowed to request the lock because it is not following multi-granularity locking rules

Q5.10

0.1 Points

T1 wants to upgrade its IS lock on R1 to an IX lock. Is it compatible with the existing locks being held by the two transactions (i.e., can it be granted immediately or would T1 have to wait)?

- ○ Compatible -- T1 can be granted the lock immediately
- ○ Not compatible -- T1 must wait
- ◉ T1 will not be allowed to request the lock because it is not following multi-granularity locking rules

Q6

1 Point

Consider the following lock table, where we are showing a few data items, the locks being currently held on them, and transactions waiting on them. Assume we are not using multi-granuarlity locking here (so only S and X locks).

Data Item	*Current Lock Holder and Lock Type*	*Transactions Waiting and Lock Type*
A	T4-S, T3-S, T6-S	T2-X
B	T5-X	T1-S
C	T2-S, T3-S, T4-S	T5-X
D	T1-X	T6-S, T4-X

Q6.1

0.1 Points

If a new transaction T7 were to request an S lock on *B*, how would the above lock table be modified? Clearly list out the modifications.

u would simply add t7-s to B queue

Q6.2

0.15 Points

If transaction T2 were to request a lock upgrade from S to X on data item C, how would the above lock table be modified? Clearly list out the modifications.

T2-x would be added to the C queue

Q6.3

0.15 Points

If transaction T1 were to be aborted, how would the above lock table be modified? Clearly list out the modifications.

T6-S would become the current lock holder for item D, B queue w ould be empty

Q6.4

0.15 Points

Is there a deadlock in the above lock table?

- (●) There is a deadlock; killing T1 would resolve it.
- ( ) There is a deadlock; killing T3 and T4 would resolve it.
- ( ) There is a deadlock; killing T1 alone wouldn't resolve it but killing T1 and T2 would.
- ( ) There is no deadlock.

Q6.5

0.15 Points

Assume we were using graph-based protocol with order A, B, C, D. Explain why the above lock table would not be permitted under that protocol.

t2 acquires c, but then wants to obtain lock for a . That order does not work because t2 can only acquire locks in order abcd

Q6.6

0.15 Points

Assume the transactions arrived into the system in the order T1, T2, .., T6 (i.e., T1 is older than T2, etc). Explain why the above lock table would not be permitted under the "wound-wait" scheme.

T1 will force T2 to abort if it needs a lock T2 has, and T2 will wait f or T1. for A, T2 is in queue and would force T4/5/6 to abort and w ait for T2. But T2 is waiting for T4/5/6

Q6.7

0.15 Points

Assume the transactions arrived into the system in the order T1, T2, .., T6 (i.e., T1 is older than T2, etc). Explain why the above lock table would not be permitted under the "wait-die" scheme.

T1 will wait for T2. T2 will not wait for T1. t4/6 are in queue for D, bu t T1 currently has. T1 is older than T4, so T4/6 would simply abort.

Q7

0.25 Points

What is a “dirty read” ? Which desirable property of transactions is compromised if dirty reads are allowed (even if serializability is guaranteed) ?

dirty read: if one transaction performs write, but is not committed, another transaction reads (this read is dirty). it is not cascade-less recoverable. you must abort all dirty reads

Q8

0.25 Points

Say there is a B+-tree primary index on a relation R. Since an insert or a delete into R changes the index as well, we must take locks on the index during updates. Briefly explain what kind of locks you would take on the index pages when doing an insert or a delete into the relation. Remember a page split or merge may involve the ancestors and the siblings of a node.

I would take a SIX lock for ancestor/siblings, because we need S t o read before deciding what to update. To actually update, the leaf must be on X, and the parent should be IX as well. Requiring S and IX equates to SIX

Q9

0.25 Points

The main difference between strict 2PL and rigorous 2PL is that -- for the latter, the serializability order is the same as the commit order (i.e., if you were to draw a precedence graph and find the serial schedule -- it would be the same as the order in which the transactions committed. Show through an example how this is not true for Strict 2PL. That is: provide an example schedule that follows Strict 2PL, but the serializability order is different from the commit order.

t2 s lock(a), t2 read(a)
t2 unlock(A)
T1 x-lock(a),
T1 read/write(a),
T1 commit(a)
T1 unlock(a)

T2 commit
commit order: t1 -> t2
serializability order: t2 -> t1

Q10
0.25 Points

Clearly and concisely explain at least 2 different schemes for dealing with “deadlocks” (just names of the techniques are not sufficient, a brief explanation must be provided as well).

Timeout based: transaction waits a certain time for a lock and abor ts if it doesn't get hit by then.
wait-die scheme: using timestamps, assuming t1 is older than t2, T1 will wait for T2 if T2 needs a lock T1 currently has, but not the othe r way around.

Assignment 12: Part 1

● UNGRADED

15 HOURS, 16 MINUTES LATE

STUDENT
Forest Li Wu

TOTAL POINTS
- / 4 pts

QUESTION 1

(no title) 0.3 pts

1.1 (no title) 0.1 pts

1.2 (no title) 0.1 pts

1.3 (no title) 0.1 pts

QUESTION 2

(no title) 0.3 pts

2.1 (no title) 0.1 pts

2.2 (no title) 0.1 pts

2.3 (no title) 0.1 pts

QUESTION 3

(no title) 0.2 pts

QUESTION 4

(no title) 0.2 pts

QUESTION 5

(no title) 1 pt

5.1 (no title) 0.1 pts

5.2 (no title) 0.1 pts

5.3 (no title) 0.1 pts

5.4 (no title) 0.1 pts

5.5 (no title) 0.1 pts

5.6 (no title) 0.1 pts

5.7 (no title) 0.1 pts

5.8 (no title) 0.1 pts

5.9 (no title) 0.1 pts

5.10 (no title) 0.1 pts

QUESTION 6

(no title) 1 pt

6.1 (no title) 0.1 pts

6.2 (no title) 0.15 pts

6.3 (no title) 0.15 pts

6.4 (no title) 0.15 pts

6.5 (no title) 0.15 pts

6.6 (no title) 0.15 pts

6.7 (no title) 0.15 pts

QUESTION 7

(no title) 0.25 pts

QUESTION 8

(no title) 0.25 pts

QUESTION 9

(no title) 0.25 pts

QUESTION 10

(no title) 0.25 pts