Tufts University
CS 115: Database Systems
Problem Set 5
Spring 2022

Complete the following exercises to the best of your ability and submit your answers on Gradescope by 11:59 PM on Monday, May 2, 2022. ***No submissions will be accepted after that point.*** Note that this assignment is out of 75 points.

You can access Gradescope here: https://www.gradescope.com/. Click Log In and then School Credentials and scroll down to Tufts to log in using your Tufts credentials.

Format your answers as a PDF file. I recommend making a copy of this file as a Google Doc (File > Make a copy), filling in your answers, and then saving it as a PDF (File > Download > PDF document). You can also download this Google Doc as a Word document and save that as a PDF, or use other software entirely.

If you choose to, you can work with 1-2 partners on this assignment. If you worked with partners, mark in Gradescope the partners that you worked with. Make only one submission per group.

Direct any questions about the assignment to Piazza.

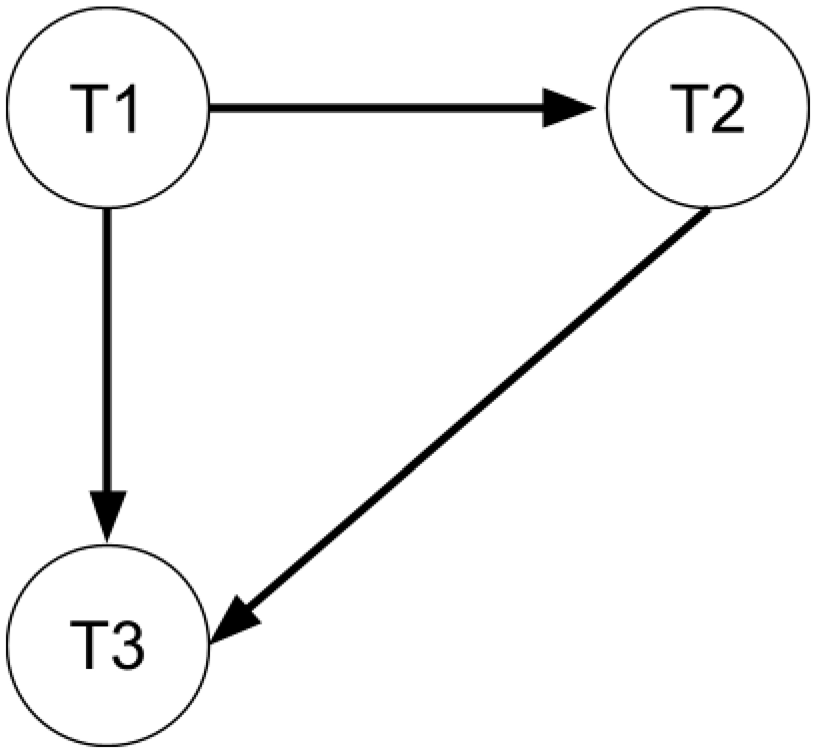
Problem 1 (10 points)

For each of the two schedules below, take the following steps:

1. Edit the provided diagram for the schedule, making whatever changes are needed to construct the schedule's precedence graph.
2. State whether the schedule is conflict serializable.
3. If the schedule is conflict serializable, state one possible equivalent serial schedule. If the schedule is not conflict serializable, explain briefly why the schedule is not equivalent to the serial schedule T1; T2; T3. Your explanation should include a discussion of why particular actions from the original schedule are not consistent with that serial schedule.

a. `w1(A); r2(A); w2(B); r3(B); w3(C); r1(C)`

Precedence graph:

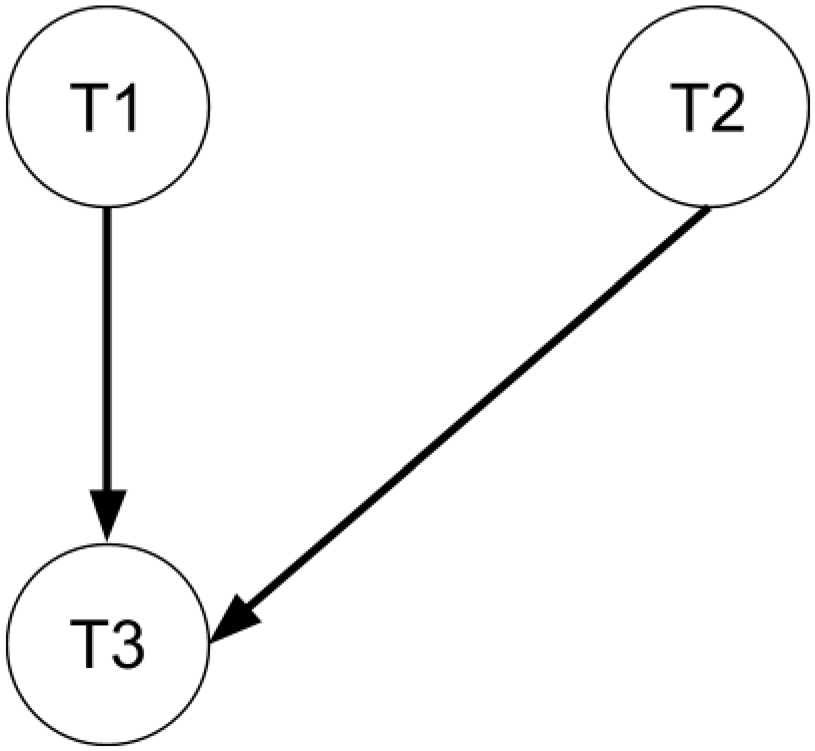


$$\begin{tikzcd} T1 \arrow[r] \arrow[d] & T2 \arrow[dl] \\ T3 & \end{tikzcd}$$

Conflict serializable (yes or no)? Yes
Equivalent serial schedule or explanation:
`w1(A); r1(C); r2(A); w2(B); r3(B); w3(C)`

b. `r1(C); w1(A); r3(A); w2(B); r3(B); w3(C)`

Precedence graph:


$$\begin{tikzcd}
T1 \arrow[d] & T2 \arrow[dl] \\
T3 &
\end{tikzcd}$$

Conflict serializable (yes or no)? Yes
Equivalent serial schedule or explanation:
`r1(C); w1(A); w2(B); r3(A); r3(B); w3(C)`

Problem 2 (15 points)

Consider these two transactions:

```
T1: r(B); r(C); r(A); w(C); c
T2: r(B); r(A); r(C); w(A); c
```

The following is the beginning of one possible schedule of these transactions, with appropriate lock instructions added:

T1	T2
sl(B)	
r(B)	
	sl(B)
	r(B)
	xl(A)
	sl(C)
	u(B)
sl(C)	
r(C)	

a. Complete this schedule (i.e., provide the remainder of the actions, including commits) in a way that follows the two-phase locking rule, as well as the rules for which operations are allowed when holding shared and exclusive locks. The completed schedule should allow both transactions to finish without being rolled back and restarted.

Notes:

- When completing the schedule table provided in the table, please put only **one** action per "line" of the table so that the order in which the actions are completed will be clear.
- Assume that shared locks can be upgraded, and that there are no update locks.
- Don't forget that the order of the operations within a given transaction cannot change. For example, regardless of how you interleave the actions from T1 and T2, T1's read of B must still come before T1's read of C, because that is the order in which those actions occur when T1 is executed on its own.

T1	T2
sl(B)	
r(B)	
	sl(B)
	r(B)
	xl(A)
	sl(C)
	u(B)
sl(C)	
r(C)	
	r(A)
	r(C)
	u(C)
	w(A)
	u(A)
	commit
sl(A)	
r(A)	
xl(C)	
w(C)	
u(A)	
u(B)	
u(C)	
commit	

b. The partial schedule that we gave you for part a does not observe rigorous locking. Explain briefly what aspect(s) of the schedule prevent it from being rigorous.

The partial schedule provided for part (a) does not observe rigorous locking because T2 acquires a shared lock for element B and then unlocks element B prior to committing. This prevents the schedule from being rigorous because rigorous locking requires all transactions to hold all locks (i.e. shared or exclusive) until committing or aborting (i.e. unlocking locked elements before committing is not allowed) and T2 did not do this.

c. Is your completed schedule recoverable? Explain briefly why or why not.

My completed schedule is recoverable because T1 reads element A after T2 writes to element A and T1 commits after T2. Therefore, if a crash were to happen after T2 commits but before T1 commits, then T1 would be rolled back (T2 would not because it already committed). This roll back would not cause any problems though because elements B and C were never changed by T2 (so T1 would be reading the same values for elements B and C as it did pre-crash) and the change in element A was committed by T2 before T1 read element A, so T1 would also read the same value for element A as it did pre-crash.

Problem 3 (15 points)

Consider the following partial schedule of the transactions T1, T2, and T3:

```
xl2(A); w2(A); sl1(B); r1(B); ul3(C); ...
```

Given this partial schedule, which of the following lock requests would be granted if it were the *next* (sixth) operation in the schedule, and which would be denied? Explain each answer briefly. Remember: `uli(X)` indicates Ti's request of the update lock for X.

request	***granted or denied?***	***explanation***
`ul1(A)`	Denied	T2 currently holds an exclusive lock on element A
`sl3(B)`	Granted	Only shared locks on element B are held by other txns (if any)
`ul2(B)`	Granted	Only shared locks on element B are held by other txns
`sl1(C)`	Denied	T3 currently holds the update lock on element C
`xl3(C)`	Granted	T3 currently holds the update lock on element C
`ul2(C)`	Denied	T3 currently holds the update lock on element C

Problem 4 (15 points)

For each of the two sequences of operations below, determine whether deadlock will occur under *rigorous two-phase locking*. Assume the following:

- An exclusive lock for an item is requested just before writing that item, and a shared lock for an item is requested (if needed) just before reading that item
- A transaction commits immediately after completing its final read or write
- Upgrades of shared locks are allowed
- Update locks are not used

If deadlock occurs, show the partial schedule in table form (including lock operations and any commits) up to the point of deadlock, along with the waits-for graph at the point of deadlock.

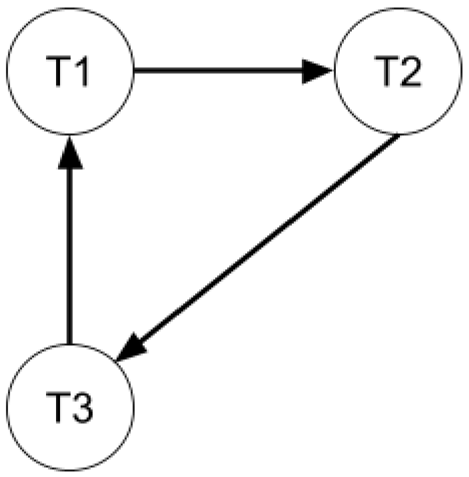
If deadlock does *not* occur, show the full schedule, including lock operations and commits.

In either case, your schedule should include any changes to the sequence of operations that occur because a transaction is forced to wait for a lock. If a transaction waits for a lock that is subsequently granted, you should include two lock requests – one for the initial request, and one at the point at which the lock is granted. If two transactions wait for a lock held by the same other transaction, and that other transaction subsequently commits, you should assume that the waiting transaction with the smaller transaction number is granted its request first.

a. `r2(C); r1(D); w1(C); r3(C); w2(C); w3(D); w1(D)`

T1	T2	T3
	sl(C)	
	r(C)	
sl(D)		
r(D)		
xl(C); wait for T2		
		sl(C)
		r(C)
	xl(C); wait for T3	
		xl(D); wait for T1

waits-for graph (if deadlock occurs):



b. `w3(A); r2(A); w2(A); r1(B); r3(B); w3(B); w1(B)`

T1	**T2**	**T3**

		xl(A)
		w(A)

	sl(A); wait for T3	

sl(B)
r(B)		

		sl(B)
		r(B)
		xl(B); wait for T1

xl(B); wait for T3		

waits-for graph (if deadlock occurs):

$$\begin{tikzcd}
T1 \arrow[d, shift right] & T2 \arrow[dl] \\
T3 \arrow[u, shift right] &
\end{tikzcd}$$

Problem 5 (20 points)

Note: we will cover the material needed for this question in class on Wednesday, April 20.

Assume that a database is replicated across 11 different sites.

1. If the system uses *fully distributed* locking, which of the following voting schemes (if any) would work? Assume that the system obtains the appropriate types of local locks on the copies that it is attempting to update/read. Explain briefly why each scheme would or would not work.

voting scheme	***would it work? (yes/no)***	***explanation***
1a) update 6 read 5	No	Voting requirement: $5 \overset{?}{>} 11 - 6 \rightarrow 5 \ngtr 6$ ✖ This voting scheme would not work because the voting requirement is not met
1b) update 5 read 8	No	Voting requirement: $8 \overset{?}{>} 11 - 5 \rightarrow 8 > 6$ Locking requirements : $5 \overset{?}{>} 11 / 2 \rightarrow 5 \ngtr 5.5$ ✖ $8 \overset{?}{>} 11 - 5 \rightarrow 8 > 6$ This voting scheme would not work because the first locking requirement is not met (two exclusive locks can be taken at once)
1c) update 8 read 4	Yes	Voting requirement: $4 \overset{?}{>} 11 - 8 \rightarrow 4 > 3$ Locking requirements : $8 \overset{?}{>} 11 / 2 \rightarrow 8 > 5.5$ $4 \overset{?}{>} 11 - 8 \rightarrow 4 > 3$ This voting scheme would work because the voting requirement and both locking requirements are all met
1d) update 3 read 10	No	Voting requirement: $10 \overset{?}{>} 11 - 3 \rightarrow 10 > 8$ Locking requirements : $3 \overset{?}{>} 11 / 2 \rightarrow 3 \ngtr 5.5$ ✖ $10 \overset{?}{>} 11 - 3 \rightarrow 10 > 8$ This voting scheme would not work because the first locking requirement is not met (two exclusive locks can be taken at once)

2. Now assume the system uses *primary-copy* locking. Which of the following voting schemes (if any) would work? Assume that the system obtains a primary copy lock on one of the copies that it is attempting to update/read. Explain briefly why each scheme would or would not work.

voting scheme	***would it work? (yes/no)***	***explanation***
2a) update 6 read 5	No	Voting requirement: $5 \overset{?}{>} 11 - 6 \rightarrow 5 \not> 6$ ✖ This voting scheme would not work because the voting requirement is not met
2b) update 5 read 8	Yes	Voting requirement: $8 \overset{?}{>} 11 - 5 \rightarrow 8 > 6$ This voting scheme would work because the voting requirement is met
2c) update 8 read 4	Yes	Voting requirement: $4 \overset{?}{>} 11 - 8 \rightarrow 4 > 3$ This voting scheme would work because the voting requirement is met
2d) update 3 read 10	Yes	Voting requirement: $10 \overset{?}{>} 11 - 3 \rightarrow 10 > 8$ This voting scheme would work because the voting requirement is met

3. If your workload primarily involves reads, which of the above configurations (1a, 1b, 1c, 1d, 2a, 2b, 2c, or 2d) would be the best one to choose? Explain briefly why you think it would be best, and make sure to also mention any potential drawbacks that it could have.

If the workload primarily involves reads, configuration 2c would be the best one to choose because it is the configuration which has the cheapest cost per read (4). Additionally, 2c is preferable over 1c (which also has a read cost of 4) because 2c uses primary-copy locking while 1c uses fully distributed locking; so for every read configuration 1c does, it must do 4 more internal operations (across the 11 sites) than 2c. A potential drawback of configuration 2c, however, is that if one site goes down, then all operations on items that are contained in that site (most notably primary copies of items) are blocked. This is due to configuration 2c using primary-copy locking.

4. If your workload primarily involves writes, which of the above configurations (1a, 1b, 1c, 1d, 2a, 2b, 2c, or 2d) would be the best one to choose? Explain briefly why you think it would be best, and make sure to also mention any potential drawbacks that it could have.

If the workload primarily involves writes, configuration 2d would be the best one to choose because it is the configuration which has the cheapest cost per write (3). A potential drawback of configuration 2c, however, is that if one site goes down, then all operations on items that are contained in that site (most notably primary copies of items) are blocked. This is due to configuration 2d using primary-copy locking.