CS 186 - Fall 2024
Exam Prep Section 8
Transactions & Concurrency

0) Transactions & Concurrency Warmup

1. Suppose a transaction T1 wants to scan a table R and update a few of its tuples. What kinds of locks should T1 have on R, the pages of R, and the updated tuples?

1. Obtain SIX on R
2. Obtain IX on Page [We don't obtain a SIX because there is already an S lock on R (from the SIX). Obtaining another S on the Page is redundant.]
3. Obtain X on Tuples being modified

2. Is an S lock compatible with an IX lock?

Suppose T1 wants an S lock on an object, O, and T2 wants an IX lock on the same object O. An S lock implies that T1 will read the entire object (all of its sub-objects). An IX lock implies that T2 will write some of the sub-objects of the object. This means that there is some sub-object of O that T1 will read and T2 will write. This is not valid, so the S and IX locks must be incompatible.

3. Consider a table which contains two pages with three tuples each, with Page 1 containing Tuples 1, 2, and 3, and Page 2 containing Tuples 4, 5, and 6.

> i. Given that a transaction T1 has an IX lock on the table, an IX lock on Page 1, and an X lock on Tuple 1, which locks could be granted to a second transaction T2 for Tuple 2?
> X or S locks
>
> ii. Given that a transaction T1 has an IS lock on the table and an S lock on Page 1, what locks could be granted to a second transaction T2 for Page 1?
> S or IS locks

1) Transactions & Concurrency

In this question, we will explore the key topics of transactions and concurrency: serializability, types of locks, two-phase locking, and deadlocks.

We will do this by actually running multiple transactions at the same time on a database, and seeing what happens. You may find it helpful (but not necessary) to draw some graphs:

- For the lock type questions, you may wish to draw a graph representing the whole database and which resources are being locked.
- For serializability questions, you may wish to draw a graph with a node for each transaction, and arrows if there are *conflicts* between transactions.
- For deadlock questions, you may wish to draw a graph with a node for each transaction, and arrows if a transaction is *waiting* for a lock held by another transaction.

We will use a database with tables A, B, C, ... and table A holds rows A1, A2, A3, ... and so on.

Consider the following sequence of operations:

Txn 1: IX-Lock(Database)	(1)
Txn 1: IX-Lock(Table A)	(2)
Txn 1: X-Lock(Row A1)	(3)
Txn 1: Write(Row A1)	(4)
Txn 1: Unlock(Row A1)	(5)
Txn 1: S-Lock(Row A2)	(6)
Txn 2: IX-Lock(Database)	(7)
Txn 2: IX-Lock(Table A)	(8)
Txn 2: X-Lock(Row A1)	(9)
Txn 2: Write(Row A1)	(10)
Txn 2: S-Lock(Row A2)	(11)
Txn 2: Read(Row A2)	(12)

1. Is Transaction 1 doing Two-Phase Locking so far?

   Answer: No; we have a lock (6) after unlock (5).

2. Is Transaction 1 doing Strict Two-Phase Locking?

   Answer: No; it's not even Two-Phase.

3. Is this schedule conflict-serializable so far? If not, what is the cycle?

   Answer: Yes, it is conflict-serializable; there is only one conflict (4 → 10), so it is serializable to a serial order of Txn 1 → Txn 2.

4. Is this schedule serial so far?

   Answer: Yes! Since all of Txn 1's operations are before Txn 2's operations, it is actually already a serial order.

Continuing with all operations so far:

Txn 2: SIX-Lock(Table B)	(13)

Txn 2: X-Lock(Row B1) (14)
Txn 2: Write(Row B1) (15)
Txn 2: Unlock(Row B1) (16)
Txn 2: Unlock(Table B) (17)
Txn 1: SIX-Lock(Table B) (18)
Txn 1: X-Lock(Row B1) (19)
Txn 1: Read(Row B1) (20)

5. Is Transaction 2 doing Two-Phase Locking so far?

   Answer: Yes. All locks (< 17) are before unlocks (17, 18).

6. Is Transaction 2 doing Strict Two-Phase Locking?

Answer: No! For strict two-phase, we must the transaction before doing any unlocks.

7. Is this schedule conflict-serializable so far? If not, what is the cycle?

   Answer: No, not anymore. We have T1 → T2 from conflict 4 → 10, and T2 → T1 from conflict 15 → 20. The cycle is T1 ↔ T2.

8. Suppose we start a new transaction, Transaction 3. What kind of locks can Transaction 3 acquire on the whole database?

   The database currently has two IX locks held by Txn 1 and Txn 2. Looking at the compatibility matrix, we see that IS and IX are the locks compatible with IX locks, so these are the locks that Txn 3 can acquire on the database.

9. Given the above answers, what kind of locks can Transaction 3 acquire on Table A?

   On the parent of Table A (the whole database), we know we can acquire IS and IX locks (from the previous question). These locks allow us to acquire S, X, IS, IX, and SIX locks below it. Which of these can we actually acquire? Table A currently has two IX locks held by Txn 1 and Txn 2, so the compatibility matrix says IS and IX locks only.

10. Given the above answers, what kind of locks can Transaction 3 acquire on Row A3?

    On the parent of Row A3 (Table A), we know we can acquire IS and IX locks (from the previous question). These locks allow us to acquire S, X, IS, IX, and SIX locks below it.
    Which of these can we actually acquire? Row A3 has no locks currently, so we can acquire any of these! But this is a leaf node and I locks are for intermediate nodes only, so in practice we can only acquire S and X locks.

11. Given the above answers, what kind of locks can Transaction 3 acquire on Table B?

    On the parent of Table B (the whole database), we know we can acquire IS and IX locks (from the previous question). These locks allow us to acquire S, X, IS, IX, and SIX locks below it. Which of these can we actually acquire? Table B currently has an SIX lock, so the compatibility matrix says IS locks only.

12. Given the above answers, what kind of locks can Transaction 3 acquire on Row B2?

    On the parent of Row B2 (Table B), we know we can acquire IS locks (from the previous question). These locks allow us to acquire S and IS locks below it.
    Which of these can we actually acquire? Row B2 has no locks currently, so we can acquire any of these! But this is a leaf node and I locks are for intermediate nodes only, so in practice we can only acquire the S lock.

13. What kind of locks can Transaction 1 acquire on Row B2?

    Transaction 1 currently holds an SIX lock on the parent of Row B2 (Table B), which allows it to acquire X and IX locks below it.
    Which of these can it actually acquire? Row B2 has no locks, so it can acquire either of them;

however, I locks are for intermediate nodes only, so it can just acquire the X lock.

Txn 2: IX-Lock(Table B)	(21)
Txn 3: IX-Lock(Database)	(22)
Txn 3: X-Lock(Table C)	(23)
Txn 3: IX-Lock(Table A)	(24)
Txn 3: X-Lock(Row A1)	(25)
Txn 1: IX-Lock(Table C)	(26)

14. We have now entered a deadlock. What is the waits-for cycle between the transactions?

    T1 (26) waits on T3 (23).
    T3 (25) waits on T2 (9).
    T2 (21) waits on T1 (18).

15. We can end this deadlock by aborting the youngest transaction. Which transaction do we abort?

    The youngest transaction is T3, so we abort that one.

    Alternatively, we could have avoided this deadlock in the first place by using *wound-wait* or *wait-die*. Recall from lecture that these methods cause transactions to sometimes abort, according to a priority order, when they try to acquire locks.

    Let's say that the priority of the transaction is its number (Txn 1 is highest priority).

16. If we were using *wound-wait*, what is the first operation in this sequence that would cause a trans action to get aborted, and which transaction gets aborted?
    In *wound-wait*, the only waiting that happens is lower priority waiting for higher priority. If a higher priority transaction tries to wait, it will just abort ("wound") the transaction it is waiting on. At (21), T2 can wait on T1. At (25), T3 can wait on T2. But at (26), T1 will not wait on T3; it will instead abort T3.

17. If we were using *wait-die*, what is the first operation in this sequence that would cause a transaction to get aborted, and which transaction gets aborted?

    In *wait-die*, the only waiting that happens is higher priority waiting for lower priority. If a lower priority transaction tries to wait, it will just abort itself ("die") instead.
    At (21), T2 will not wait on T1, since T2 is lower priority. Thus, T2 will abort.

2) Miscellaneous Transactions & Concurrency

Consider a database with objects X and Y and two transactions. Transaction 1 reads X and Y and then writes X and Y. Transaction 2 reads and writes X then reads and writes Y.

1. Create a schedule for these transactions that is not serializable. Explain why your schedule is not serializable.

   Here is an example of a schedule that is not serializable.

Transaction 1	Transaction 2
Read(X)	
Read(Y)	
	Read(X)
	Write(X)
	Read(Y)
Write(X)	
Write(Y)	
	Write(Y)

In this example, T1 reads X before T2 writes X. However, T1 writes X after T2 reads/writes it. The schedule is thus not serializable since there is no equivalent serial schedule that would have the same result (neither T1-T2 or T2-T1).

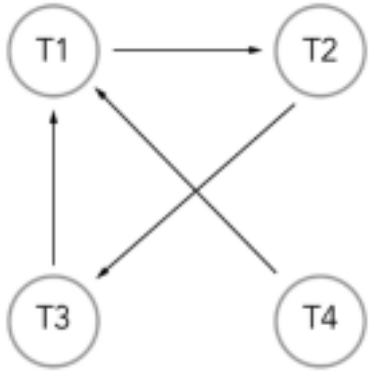
2. Would your schedule be allowed under strict two-phase locking? Why or why not?

No, because strict 2PL ensures serializability. Keep in mind that strict 2PL only allows releasing locks at the end of a transaction. In the example schedule shown above, when Transaction 2 attempts to acquire an exclusive lock to write X, it will have to wait for Transaction 1 to release its lock on X, which will not happen until Transaction 1 commits. This will never happen, so this schedule is not possible under strict 2PL.

Now consider the following schedule.

	1	2	3	4	5	6	7	8
T1				X(B)	X(A)			S(D)
T2		X(D)	S(E)					
T3	X(E)						S(B)	
T4						X(A)		

3. Draw the waits-for graph for this schedule.



4. Are there any transactions involved in deadlock?

Yes, Transactions 1, 2, and 3 are deadlocked.

5. Next, assume that T1 priority > T2 > T3 > T4. You are the database administrator and one of your

users purchases an exclusive plan to ensure that their transaction, Transaction 2, runs to completion. Assuming the same schedule, what deadlock avoidance policy would you choose to make sure Transaction 2 commits?

Choose wait-die. Under wait-die, T3 and T4 abort after steps 6 and 7 because they are attempting to acquire a lock held by a transaction with higher priority. Afterwards, both T1 and T2 run to completion.

However, under wound-wait, T3 will be killed by T2 at step 3, and T2 will be killed by T1 at step 8.

Since we want to make sure Transaction 2 commits, we should choose wait-die.