Esra Şekerci

2698215

IS 503 – Assignment 4, due: 26 May 2024

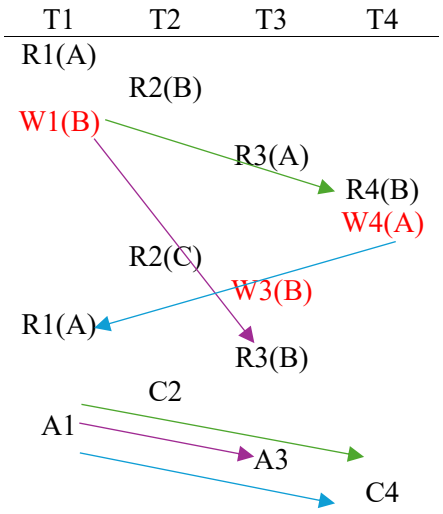
1)

Recoverable Schedule: In a recoverable schedule, no committed transaction ever needs to be rolled back because all transactions that have read data written by another transaction commit only after that other transaction has committed.

Cascadeless Schedule: In a cascadeless schedule, every transaction reads only the items that are written by committed transactions. This prevents cascading aborts.

Strict Schedule: In a strict schedule, a transaction can neither read nor write an item X until the last transaction that wrote X has committed. This ensures that no transaction can read or write dirty data.

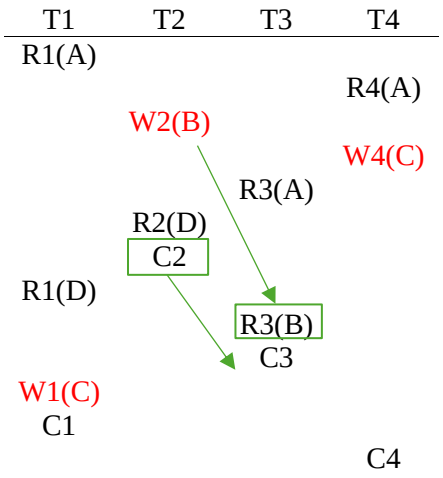
S1: R1(A); R2(B); W1(B); R3(A); R4(B); W4(A); R2(C); W3(B); R1(A); R3(B); C2; A1; A3; C4;



T4 reads from T1 and then commits, while T1 aborts. This means T4 commits based on a value written by a transaction that later aborts. This violates the condition for recoverability since T4 relies on a value from T1, which is not committed.

Therefore, S1 is **not recoverable** because T4 commits after reading a value from T1, which subsequently aborts. Since it is **not recoverable, it is also neither cascadeless nor strict.**

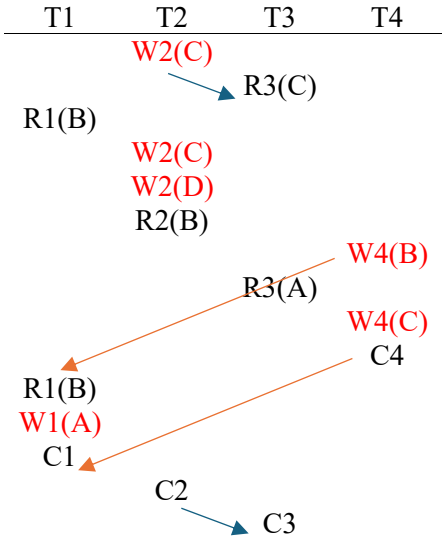
S2: R1(A); R4(A); W2(B); W4(C); R3(A); R2(D); C2; R1(D); R3(B); C3; W1(C); C1; C4;



The schedule guarantees recoverability by avoiding the need to roll back committed transactions. It ensures that transactions like T3 read data only after all preceding transactions that wrote that data have committed or aborted. For cascadelessness, it ensures that each transaction's read operation occurs only after it sees the commit or abort of preceding transactions. However, the schedule does not strictly enforce the property where transactions wait until the last transaction that wrote data has committed before reading or writing.

In conclusion, while schedule S2 is **recoverable and cascadeless**, it is **not strictly adherent** because T4 does write C after T1 has committed, which violates the strictness property.

S3: W2(C); R3(C); R1(B); W2(C); W2(D); R2(B); W4(B); R3(A); W4(C); C4; R1(B); W1(A); C1; C2; C3;



For recoverability, no transaction that has committed should need to be rolled back. S3 is **recoverable**. However, S3 is **not cascadeless** because T3 read data written by T2 before T2 commits. Therefore, it is **not strict** either.

2)

A schedule is said to be view serializable if it is view equivalent to a serial schedule. View equivalence ensures that the outcome of the transactions in the schedule is the same as if the transactions were executed in some serial order.

Two schedules S and S′ are view equivalent if the following three conditions hold:

1. **Same Transactions and Operations:** The same set of transactions participates in both schedules S and S′, and these schedules include the same operations of those transactions.
2. **Read-From Condition:** For any operation Ri(X) of transaction Ti in S, if the value of X read by the operation has been written by an operation Wj(X) of transaction Tj (or if it is the original value of X before the schedule started), the same condition must hold for the value of X read by operation Ri(X) of Ti in S′.
3. **Final Write Condition:** If the operation Wk(Y) of transaction Tk is the last operation to write item Y in S, then Wk(Y) of Tk must also be the last operation to write item Y in S′.

Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules. This means that the schedule can be transformed into a serial schedule by swapping non-conflicting operations while preserving the order of conflicting operations.

Any conflict serializable schedule is also view serializable, but not vice versa.

To verify conflict serializability, we need to follow these steps:

1. **Identify Conflicting Operations:** Check for read and write operations on the same data item across different transactions.
2. **Construct a Precedence Graph:** Create a directed graph where each node represents a transaction. Draw an edge from transaction Ti to transaction Tj if Ti performs a conflicting operation before Tj.
3. **Cycle Detection:** Ensure the graph has no cycles. A schedule is conflict serializable if its precedence graph is acyclic.

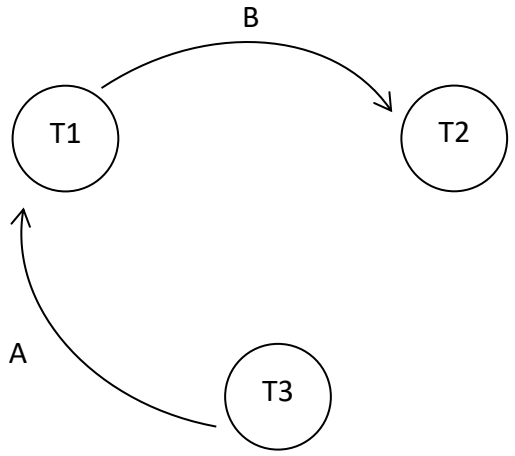
Let's first focus on determining if the schedules are conflict serializable, as it inherently implies view serializability.

S4: W1(B); W3(A); W1(D); R1(B); R2(C); R1(A); R2(B); R3(A); R2(C); R1(C); C3; C1; C2;

T1	T2	T3
W1(B)		
		W3(A)
W1(D)		
R1(B)		
	R2(C)	
R1(A)		
	R2(B)	
		R3(A)
	R2(C)	
R1(C)		
		C3
C1		
	C2	

B(T1 →T2)

A(T3→T1)



Since there are no cycles, the schedule is both conflict serializable and therefore view serializable.

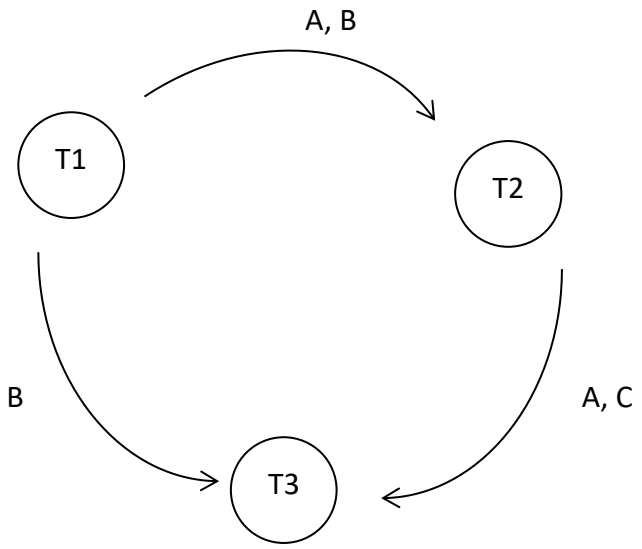
S5: R1(A); W2(A); R3(A); W1(B); R2(B); R3(B); R2(C); W3(C); R1(B); C1; C2; C3;

T1	T2	T3
R1(A)		
	W2(A)	
		R3(A)
W1(B)		
	R2(B)	
		R3(B)
	R2(C)	
		W3(C)
R1(B)		
C1		
	C2	
		C3

A(T1→T2), A(T2→T3)

B(T1→T2), B(T1→ T3)

C(T2→T3)



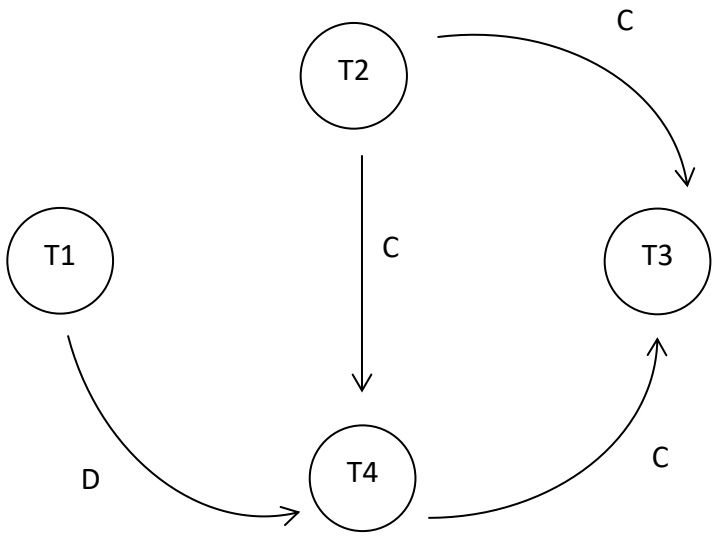
Since there are no cycles, the schedule is both conflict serializable and therefore view serializable.

S6: R3(B); W2(C); R4(B); W1(D); R2(C); R3(A); W4(D); W4(C); R3(C); R1(B); R2(B); C2; R1(B); R4(A); R3(C); R1(B); C3; C4; C1;

T1	T2	T3	T4
		R3(B)	
	W2(C)		
			R4(B)
W1(D)			
	R2(C)		
		R3(A)	
			W4(D)
			W4(C)
		R3(C)	
R1(B)			
	R2(B)		
	C2		
R1(B)			
			R4(A)
		R3(C)	
R1(B)			
		C3	
			C4
C1			

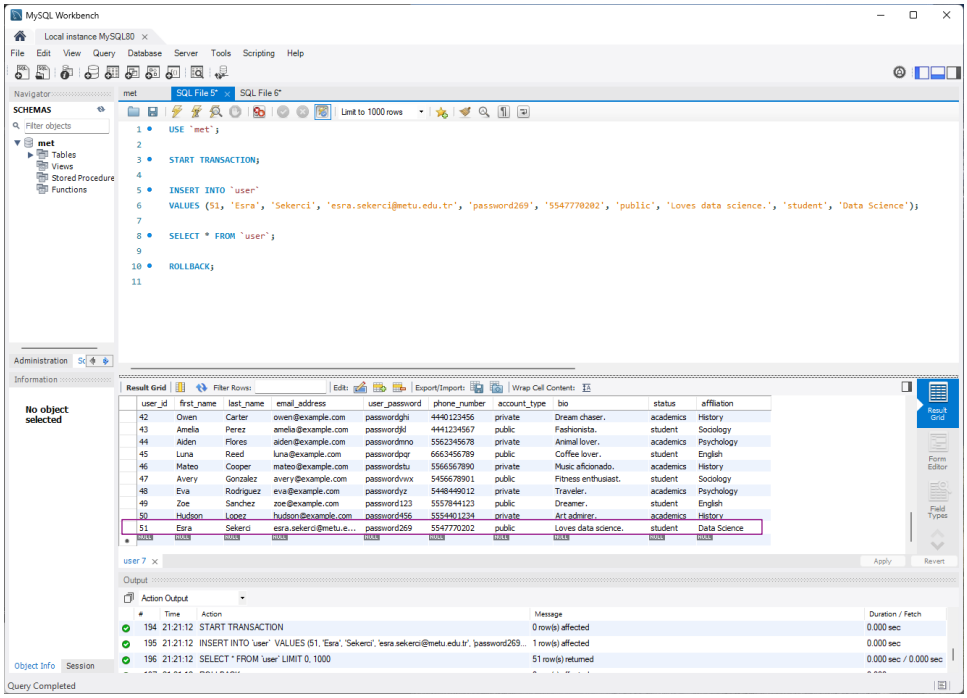
C(T2→T4), C(T2→T3), C(T4→T3)

D(T1→T4)

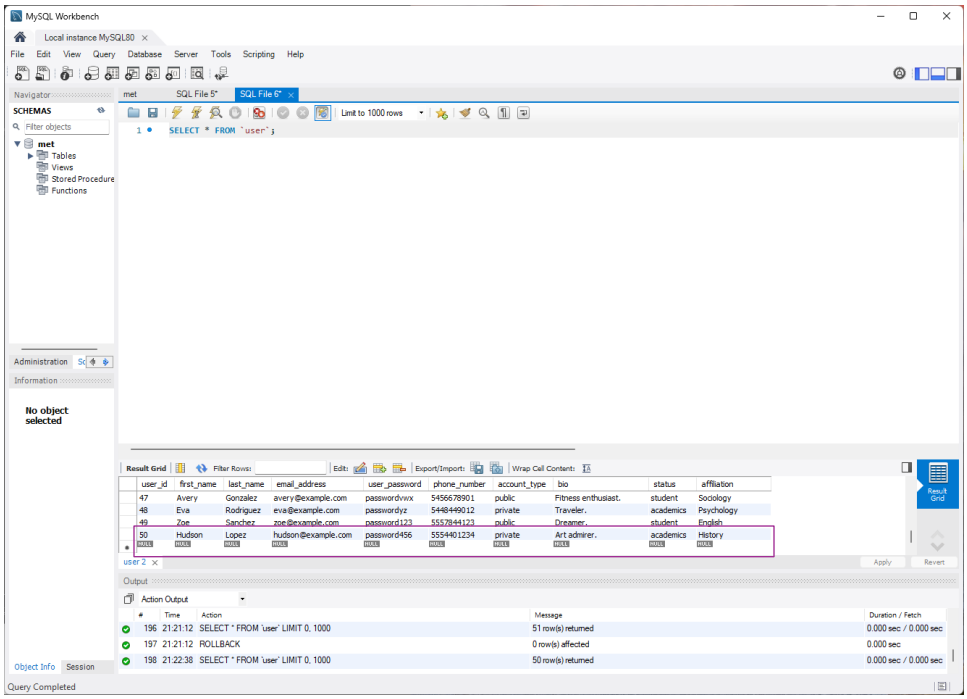


Since there are no cycles, the schedule is both conflict serializable and therefore view serializable.

3)



Even though the transaction is rolled back, we still see the new row in the result set because the rollback has not yet taken effect.



Here, new SQL tab showing the Users table without the new row, highlighting the difference.

Transactions: A transaction is a sequence of one or more SQL operations treated as a single unit. Transactions ensure data integrity and consistency. They follow the ACID properties (Atomicity, Consistency, Isolation, Durability).

- **Atomicity:** Ensures all operations within a transaction are completed successfully. If any operation fails, the transaction is aborted.
- **Consistency:** Ensures the database remains in a consistent state before and after the transaction.
- **Isolation:** Ensures that transactions are executed in isolation, meaning the intermediate state of a transaction is not visible to other transactions.
- **Durability:** Ensures that once a transaction is committed, it is permanently recorded in the database.

Commit: The commit statement is used to save all changes made during the transaction to the database. Once committed, the changes are permanent and visible to other transactions.

Rollback: The rollback statement undoes all changes made during the transaction. It reverts the database to the state before the transaction began, ensuring no partial changes are saved.