Database Management System

Module 4: Recoverability

Salim Diwani
College of Informatics and Virtual Education
University of Dodoma

Module Objectives:

- What happens if system **fails** while a transaction is in execution? Can a **consistent** state be reached for the database**? Recoverability** attempts to answer issues in state and transaction **recovery** in the face of system failures
- Conflict serializability is a crisp concept for concurrent execution that guarantees **ACID** properties and has a simple detection algorithm. Yet only few schedules are conflict serializable in practice. There is a need to explore – **View Serializability** – a weaker system for better concurrency.

Module Outline:

- Recoverability and Isolation
- Transaction Definition in SQL
- View Serializability

What is Recovery?

- **Serializability** helps to ensure Isolation and Consistency of a schedule
- Yet, the **Atomicity** and **Consistency** may be compromised in the face of system failures
- Consider a schedule comprising a single transaction (obviously serial):

1. **Read (A)**
2. **A := A – 50**
3. **write (A)**
4. **Read (B)**
5. **B : = B + 50**
6. **write (B)**
7. **Commit // Make the changes permanent; show the result to the user**

What is Recovery?

- What if system fails after **step 3** and before **step 6**?
  - Leads to **inconsistent** state
  - Need to **rollback** update of A
  - This is known as **Recovery**

Recoverability

Irrecoverable Schedules

If in a schedule,

- A transaction performs a dirty read operation from an uncommitted transaction
- And commits before the transaction from which it has read the value
- then such a schedule is known as an **Irrecoverable Schedule**.

Irrecoverable Schedules

Example

Consider the following schedule

Transaction T1	Transaction T2
R (A)	
W (A)	
⋮	R (A) // Dirty Read
	W (A)
	Commit
Rollback	

Irrecoverable Schedule

Irrecoverable Schedules

Here,

- T2 performs a dirty read operation.
- T2 commits before T1.
- T1 fails later and roll backs.
- The value that T2 read now stands to be incorrect.
- T2 can not recover since it has already committed.

Recoverable Schedules

If in a schedule,

- A transaction performs a dirty read operation from an uncommitted transaction
- And its commit operation is delayed till the uncommitted transaction either commits or roll backs then such a schedule is known as a **Recoverable Schedule**.

Here,

- The commit operation of the transaction that performs the dirty read is delayed.
- This ensures that it still has a chance to recover if the uncommitted transaction fails later.

Recoverable Schedules

Example

Consider the following schedule

Transaction T1	Transaction T2	
R (A)		
W (A)		
	R (A)	// Dirty Read
	W (A)	
Commit		
	Commit	// Delayed

Recoverable Schedule

Recoverable Schedules

Here,

- T2 performs a dirty read operation.
- The commit operation of T2 is delayed till T1 commits or roll backs.
- T1 commits later.
- T2 is now allowed to commit.
- In case, T1 would have failed, T2 has a chance to recover by rolling back.

Irrecoverable Schedules: Example

❖ **Irrecoverable Schedule**

T1	T1's Buffer	T2	T2's Buffer	Database
				A = 5000
R (A);	A = 5000			A = 5000
A = A -1000;	A = 4000			A = 5000
W (A);	A = 4000			A = 4000
		R (A);	A = 4000	A = 4000
		A = A + 500;	A = 4500	A = 4000
		W (A);	A = 4500	A = 4500
		Commit;		
Failure Point				
Commit;				

Recoverable Schedules: Example

❖ **Recoverable Schedule with cascading rollback**

T1	T1's Buffer	T2	T2's Buffer	Database
				A = 5000
R (A);	A = 5000			A = 5000
A = A -1000;	A = 4000			A = 5000
W (A);	A = 4000			A = 4000
		R (A);		A = 4000
		A = A + 500;		A = 4000
		W (A);		A = 4500
Failure Point				
Commit;				
		Commit;		

Recoverable Schedules: Example

❖ **Recoverable Schedule without cascading rollback**

T1	T1's Buffer	T2	T2's Buffer	Database
				A = 5000
R (A);	A = 5000			A = 5000
A = A -1000;	A = 4000			A = 5000
W (A);	A = 4000			A = 4000
Commit;				
		R (A);	A = 4000	A = 4000
		A = A + 500;	A = 4500	A = 4000
		W (A);	A = 4500	4500
		Commit;		

Checking Whether a Schedule is Recoverable or Irrecoverable

Method-01:

Check whether the given schedule is conflict serializable or not.

- If the given schedule is conflict serializable, then it is surely recoverable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be recoverable. Go and check using other methods.

Thumb Rules

- **All conflict serializable schedules are recoverable.**
- **All recoverable schedules may or may not be conflict serializable.**

Checking Whether a Schedule is Recoverable or Irrecoverable

Method-02:

Check if there exists any dirty read operation.

(Reading from an uncommitted transaction is called as a dirty read)

- If there does not exist any dirty read operation, then the schedule is surely recoverable. Stop and report your answer.
- If there exists any dirty read operation, then the schedule may or may not be recoverable.

If there exists a dirty read operation, then follow the following cases

Checking Whether a Schedule is Recoverable or Irrecoverable

Case-01:

If the commit operation of the transaction performing the dirty read occurs before the commit or abort operation of the transaction which updated the value, then the schedule is irrecoverable.

Checking Whether a Schedule is Recoverable or Irrecoverable

Case-02:

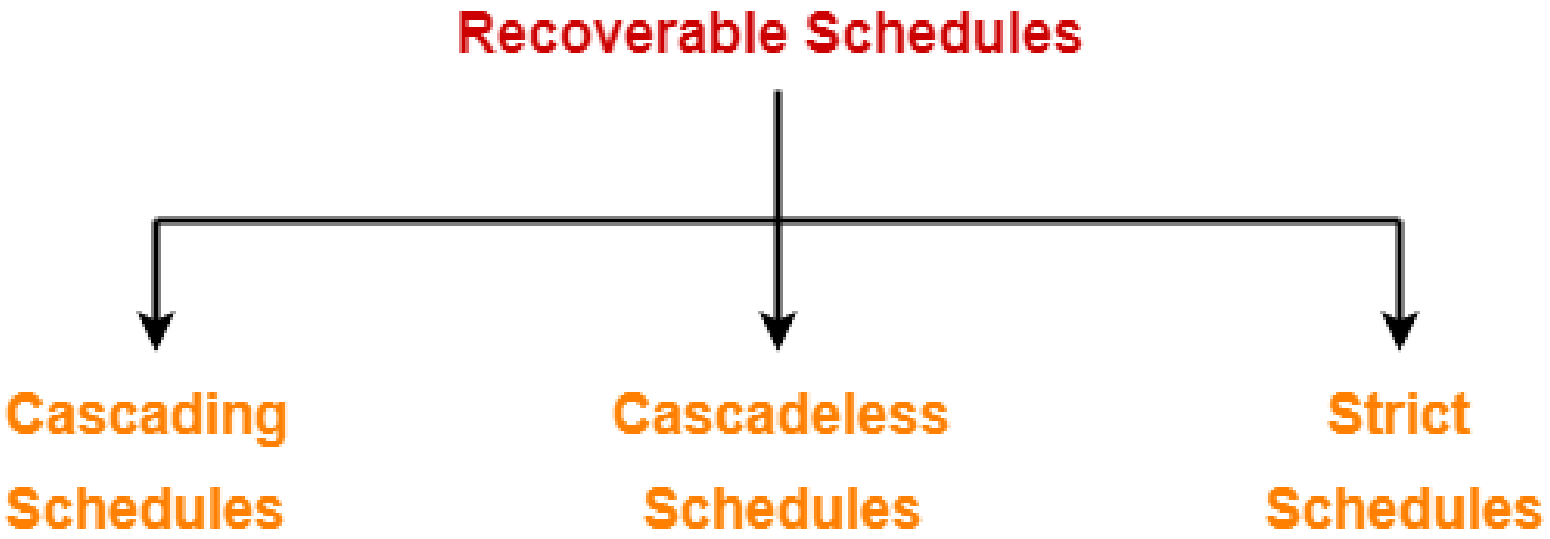
If the commit operation of the transaction performing the dirty read is delayed till the commit or abort operation of the transaction which updated the value, then the schedule is recoverable.

Thumb Rule

No dirty read means a recoverable schedule.

Types of Recoverable Schedules

A recoverable schedule may be any one of these kinds-



Cascading Schedule

- If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Schedule or Cascading Rollback or Cascading Abort.
- It simply leads to the wastage of CPU time.

Example

T1	T2	T3	T4
R (A)			
W (A)			
	R (A)		
	W (A)		
		R (A)	
		W (A)	
			R (A)
			W (A)
Failure			

Cascading Recoverable Schedule

Cascading Schedule

Here,

- Transaction T2 depends on transaction T1.
- Transaction T3 depends on transaction T2.
- Transaction T4 depends on transaction T3.

Cascading Schedule

In this schedule,

- The failure of transaction T1 causes the transaction T2 to rollback.
- The rollback of transaction T2 causes the transaction T3 to rollback.
- The rollback of transaction T3 causes the transaction T4 to rollback.
- Such a rollback is called as a **Cascading Rollback**.

Cascadeless Schedule

If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a Cascadeless Schedule.

In other words,

- Cascadeless schedule allows only committed read operations.
- Therefore, it avoids cascading roll back and thus saves CPU time.

Example

T1	T2	T3
R (A)		
W (A)		
Commit		
	R (A)	
	W (A)	
	Commit	
		R (A)
		W (A)
		Commit

Cascadeless Schedule

Strict Schedule

If in a schedule, a transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a **Strict Schedule**.

In other words,

- Strict schedule allows only committed read and write operations.
- Clearly, strict schedule implements more restrictions than cascadeless schedule.

Example

T1	T2
W (A)	
Commit / Rollback	
	R (A) / W (A)

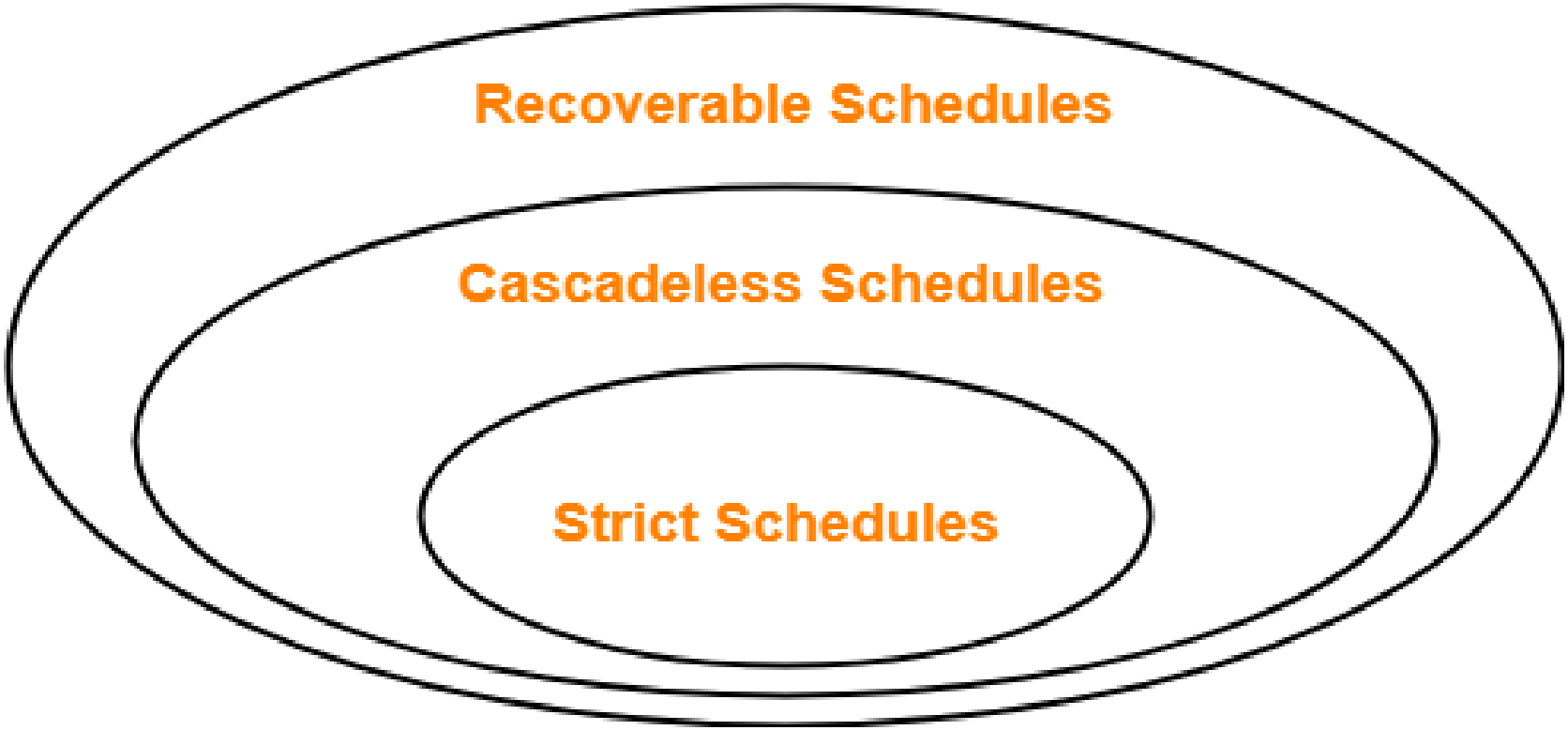
Strict Schedule

Strict Schedule

Remember

- Strict schedules are more strict than cascadeless schedules.
- All strict schedules are cascadeless schedules.
- All cascadeless schedules are not strict schedules.

Strict Schedule



Transaction Definition in SQL

- Data manipulation language must include a construct for specifying the set of actions that compromise a transaction.
- In SQL, a transaction begins implicitly
- A transaction in SQL end by:
  - **Commit work** commits current transaction and begins a new one.
  - **Rollback work** causes current transaction to abort.

Transaction Definition in SQL

- In almost all databases systems, by default, every **SQL** statement also commits implicitly if it executes successfully
  - Implicit commit can be turned off by a database directive
    - For example in JDBC**, connection.setAutoCommit(false);**

Transaction Control Language (TCL)

- The following commands are used to control transactions.
  - **COMMIT** – to save the changes
  - **ROLLBACK** – to roll back the changes
  - **SAVEPOINT** – creates point within the groups of transactions in which to ROLLBACK
  - **SET TRANSACTION** – places a name on a transaction

Transaction Control Language (TCL)

- Transactional control commands are only used with the **DML** Commands such as
  - **INSERT**, **UPDATE**, and **DELETE** only.
  - They can not be used while creating tables or dropping them because these operations are automatically committed in the database.

TCL: COMMIT Command

- The **COMMIT** is the transactional command used to save changes invoked by a transaction to the database
- The **COMMIT** saves all the transactions to the database since the last **COMMIT** or **ROLLBACK** command.
- The syntax for the COMMIT command is as follows:
  - **SQL> DELETE FROM Customers WHERE AGE = 25;**
  - **SQL> COMMIT;**

TCL: COMMIT Command

Before DELETE

SQL> SELECT * FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Salim	32	DSM	2000
2	Diwani	25	Arusha	1500
3	Amour	23	Dodoma	2000
4	John	25	Tanga	6500
5	Jacob	27	Mwanza	8500
6	Salma	22	Mtwara	4500
7	Khalil	24	Singida	10000

TCL: COMMIT Command

After DELETE

SQL> SELECT * FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Salim	32	DSM	2000
3	Amour	23	Dodoma	2000
5	Jacob	27	Mwanza	8500
6	Salma	22	Mtwara	4500
7	Khalil	24	Singida	10000

TCL: ROLLBACK Command

- The **ROLLBACK** is the command used to undo transaction that have not already been saved to the database.
- This can only be used to undo transactions since the last **COMMIT** or **ROLLBACK** command was issued
- The syntax for a **ROLLBACK** command is as follow:
  - **SQL> DELETE FROM Customers WHERE AGE = 25;**
  - **SQL> ROLLBACK;**

TCL: ROLLBACK Command

BEFORE DELETE

SQL> SELECT * FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Salim	32	DSM	2000
2	Diwani	25	Arusha	1500
3	Amour	23	Dodoma	2000
4	John	25	Tanga	6500
5	Jacob	27	Mwanza	8500
6	Salma	22	Mtwara	4500
7	Khalil	24	Singida	10000

TCL: ROLLBACK Command

AFTER DELETE

SQL> SELECT * FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Salim	32	DSM	2000
2	Diwani	25	Arusha	1500
3	Amour	23	Dodoma	2000
4	John	25	Tanga	6500
5	Jacob	27	Mwanza	8500
6	Salma	22	Mtwara	4500
7	Khalil	24	Singida	10000

TCL: SAVEPOINT / ROLLBACK Command

- A **SAVEPOINT** is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction
- The syntax for a **SAVEPOINT** command is:
  - **SAVEPOINT SAVEPOINT_NAME;**
- This command serves only in the creation of a **SAVEPOINT** among all the transactional statements.
- The ROLLBACK command is used to undo a group of transactions
- The syntax for rolling back to a SAVEPOINT IS:
  - **ROLLBACK TO SAVEPOINT_NAME;**

EXAMPLE: SAVEPOINT

- SQL> SAVEPOINT SP1;
  - **Savepoint created.**
- SQL> DELETE FROM Customers WHERE ID = 1;
  - **1 row deleted**
- SQL> SAVEPOINT SP2;
  - **Savepoint created.**
- SQL> DELETE FROM Customers WHERE ID = 2;
  - **1 row deleted**
- SQL> SAVEPOINT SP3;
  - **Savepoint created.**
- SQL> DELETE FROM Customers WHERE ID = 3;
  - **1 row deleted**

EXAMPLE: SAVEPOINT

- SQL> SAVEPOINT SP1;
- SQL> DELETE FROM Customers WHERE ID = 1;
- SQL> SAVEPOINT SP2;
- SQL> DELETE FROM Customers WHERE ID = 2;
- SQL> SAVEPOINT SP3;
- SQL> DELETE FROM Customers WHERE ID = 3;

EXAMPLE: SAVEPOINT

- Three records deleted
- Undo the deletion of first two
- SQL> ROLLBACK TO SP2;
  - ROLLBACK complete

- SQL> SELECT * FROM Customers;

At the begining

ID	NAME	AGE	ADDRESS	SALARY
1	Salim	32	DSM	2000
2	Diwani	25	Arusha	1500
3	Amour	23	Dodoma	2000
4	John	25	Tanga	6500
5	Jacob	27	Mwanza	8500
6	Salma	22	Mtwara	4500
7	Khalil	24	Singida	10000

EXAMPLE: SAVEPOINT

- Three records deleted
- Undo the deletion of first two
- SQL> ROLLBACK TO SP2;
  - ROLLBACK complete

- SQL> SELECT * FROM Customers;

After ROLLBACK

ID	NAME	AGE	ADDRESS	SALARY
2	Diwani	25	Arusha	1500
3	Amour	23	Dodoma	2000
4	John	25	Tanga	6500
5	Jacob	27	Mwanza	8500
6	Salma	22	Mtwara	4500
7	Khalil	24	Singida	10000

TCL: RELEASE SAVEPOINT Command

- The **RELEASE SAVEPOINT** command is used to remove a SAVEPOINT that you have created
- The syntax for a **RELEASE SAVEPOINT** command is as follows.
  - **RELEASE SAVEPOINT SAVEPOINT_NAME;**
- Once a **SAVEPOINT** has been released, you can no longer use the **ROLLBACK** command to undo transactions performed since the last **SAVEPOINT**

TCL: SET TRANSACTION Command

- The **SET TRANSACTION** command can be used to initiate a database transaction
- This command is used to specify characteristics for the transaction that follows
  - For example, you can specify a transaction to be read only or read write
- The syntax for a **SET TRANSACTION** command is as follows:
  - SET TRANSACTION command is as follows:
    - **SET TRANSACTION [READ WRITE| READ ONLY];**

View Serializability

- Consider two schedules **S1** and **S2** each consisting of two transactions **T1** and **T2.**

- Schedules **S1** and **S2** are called view equivalent if the following three conditions hold true for them-

View Serializability

Condition-01:

For each data item X, if transaction T_i reads X from the database initially in schedule S1, then in schedule S2 also, T_i must perform the initial read of X from the database.

Thumb Rule

“Initial readers must be same for all the data items”.

View Serializability

Condition-02:

If transaction T_i reads a data item that has been updated by the transaction T_j in schedule S1, then in schedule S2 also, transaction T_i must read the same data item that has been updated by the transaction T_j.

Thumb Rule

"Write-read sequence must be same."

View Serializability

Condition-03:

For each data item X, if X has been updated at last by transaction T_i in schedule S1, then in schedule S2 also, X must be updated at last by transaction T_i.

Thumb Rule

"Final writers must be same for all the data items".

Checking Whether a Schedule is View Serializable Or Not

Method-01:

Check whether the given schedule is conflict serializable or not.

- If the given schedule is conflict serializable, then it is surely view serializable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be view serializable. Go and check using other methods.

Thumb Rules

- **All conflict serializable schedules are view serializable.**
- **All view serializable schedules may or may not be conflict**

Checking Whether a Schedule is View Serializable Or Not

Method-02:

Check if there exists any blind write operation.

(Writing without reading is called as a blind write).

If there does not exist any blind write, then the schedule is surely not view serializable. Stop and report your answer.

If there exists any blind write, then the schedule may or may not be view serializable. Go and check using other methods.

Thumb Rule

No blind write means not a view serializable schedule.

Checking Whether a Schedule is View Serializable Or Not

Method-03:

In this method, try finding a view equivalent serial schedule.

- By using the above three conditions, write all the dependencies.
- Then, draw a graph using those dependencies.
- If there exists no cycle in the graph, then the schedule is view serializable otherwise not.

Problem-01:

Check whether the schedule is view serializable or not?

T1	T2	T3	T4
R (A)			
	R (A)		
		R (A)	
			R (A)
W (B)			
	W (B)		
		W (B)	
			W (B)

Solution

We know, if a schedule is conflict serializable, then it is surely view serializable.

So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

Solution

Step-01:

List all the conflicting operations and determine the dependency between the transactions-

$W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$

$W_1(B)$, $W_3(B)$ $(T_1 \rightarrow T_3)$

$W_1(B)$, $W_4(B)$ $(T_1 \rightarrow T_4)$

$W_2(B)$, $W_3(B)$ $(T_2 \rightarrow T_3)$

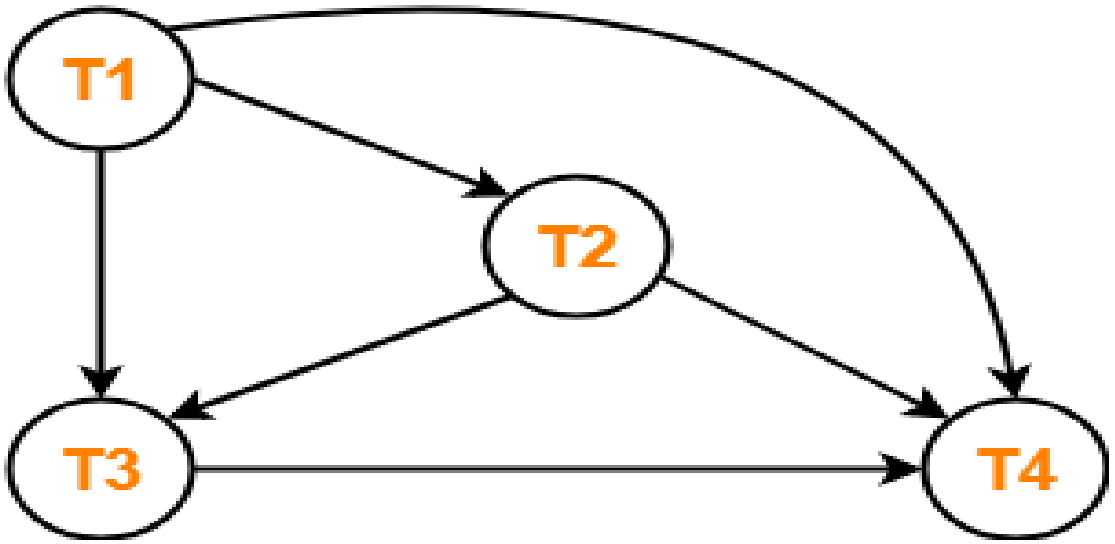
$W_2(B)$, $W_4(B)$ $(T_2 \rightarrow T_4)$

$W_3(B)$, $W_4(B)$ $(T_3 \rightarrow T_4)$

Solution

Step-02:

Draw the precedence graph-



Solution

- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.
- Thus, we conclude that the given schedule is also view serializable.

Problem-02:

Check whether the given schedule S is view serializable or not-

T1	T2	T3
R (A)		
	R (A)	
		W (A)
W (A)		

Solution

We know, if a schedule is conflict serializable, then it is surely view serializable.

So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

Solution

Step-01:

List all the conflicting operations and determine the dependency between the transactions-

$R_1(A)$, $W_3(A)$ $(T_1 \rightarrow T_3)$

$R_2(A)$, $W_3(A)$ $(T_2 \rightarrow T_3)$

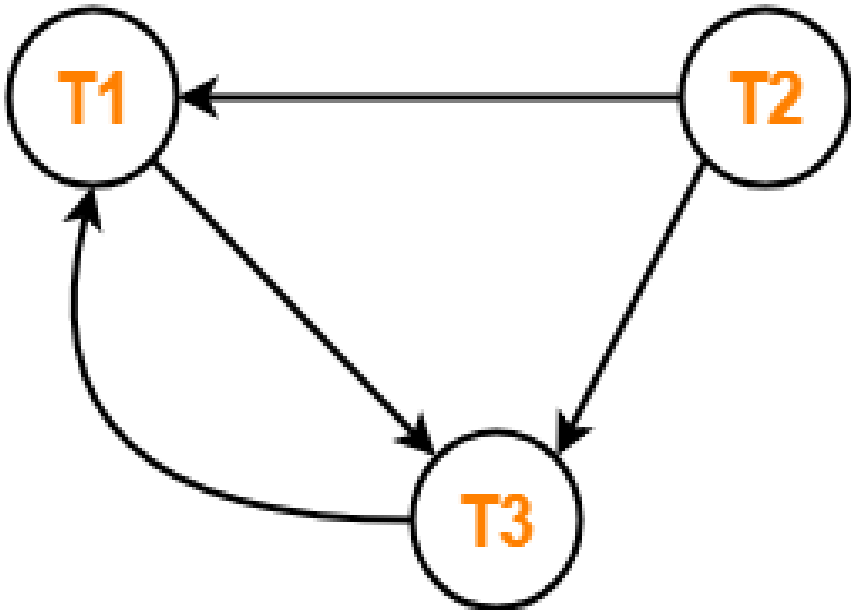
$R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$

$W_3(A)$, $W_1(A)$ $(T_3 \rightarrow T_1)$

Solution

Step-02:

Draw the precedence graph-



Solution

- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
- To check whether S is view serializable or not, let us use another method.
- Let us check for blind writes.

Solution

Checking for Blind Writes

There exists a blind write $W_3(A)$ in the given schedule S.

Therefore, the given schedule S may or may not be view serializable.

Now,

To check whether S is view serializable or not, let us use another method.

Let us derive the dependencies and then draw a dependency graph.

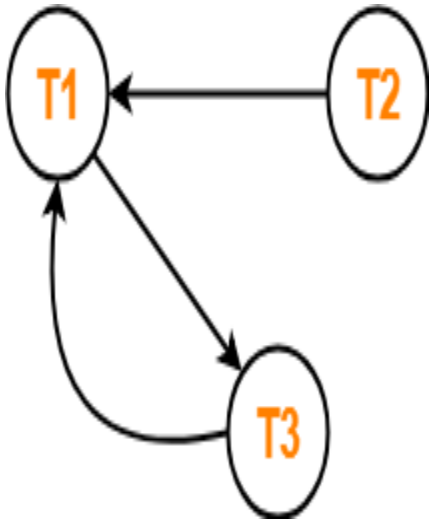
Solution

Drawing a Dependency Graph

- T1 firstly reads A and T3 firstly updates A.
- So, T1 must execute before T3.
- Thus, we get the dependency **T1 → T3**.
- Final updation on A is made by the transaction T1.
- So, T1 must execute after all other transactions.
- Thus, we get the dependency **(T2, T3) → T1**.
- There exists no write-read sequence.

Solution

Now, let us draw a dependency graph using these dependencies-



- Clearly, there exists a cycle in the dependency graph.
- Thus, we conclude that the given schedule S is not view serializable.

Quiz

- Check whether the schedule is conflict serialazable and view serializable or not?
  - S: R1(A); R2(A); R3(A); R4(A); W1(B); W2(B); W3(B); W4(B)

Than you...!