Contact Session 12:

Transaction Processing, Concurrency Control and Schedules

Objective:

Understanding the Transaction Processing System (TPS), characteristics of the transaction and notions underlying TPS and concurrent transaction execution, Schedules and Serializability.

Contents:

- Introduction to Transactions in DBMS
- Transaction Model
- Concurrent Transactions and Issues
- Understanding Schedules and Serializability
- Concurrency Control and Conflict Serializability
- Test for Conflict Serializability

Introduction to Transactions in DBMS

Single-User: at most one user at a time can use the system

Multiuser: many users can use the system concurrently.

Multiprogramming: allows the computer to execute multiple programs at the same time.

Interleaving: keeps the CPU busy when a process requires an input or output operation, the CPU switched to execute another process rather than remaining idle during I/O time.

Most of the theory concerning concurrency control in databases is developed in terms of interleaved concurrency

Sequential Vs Concurrent Execution

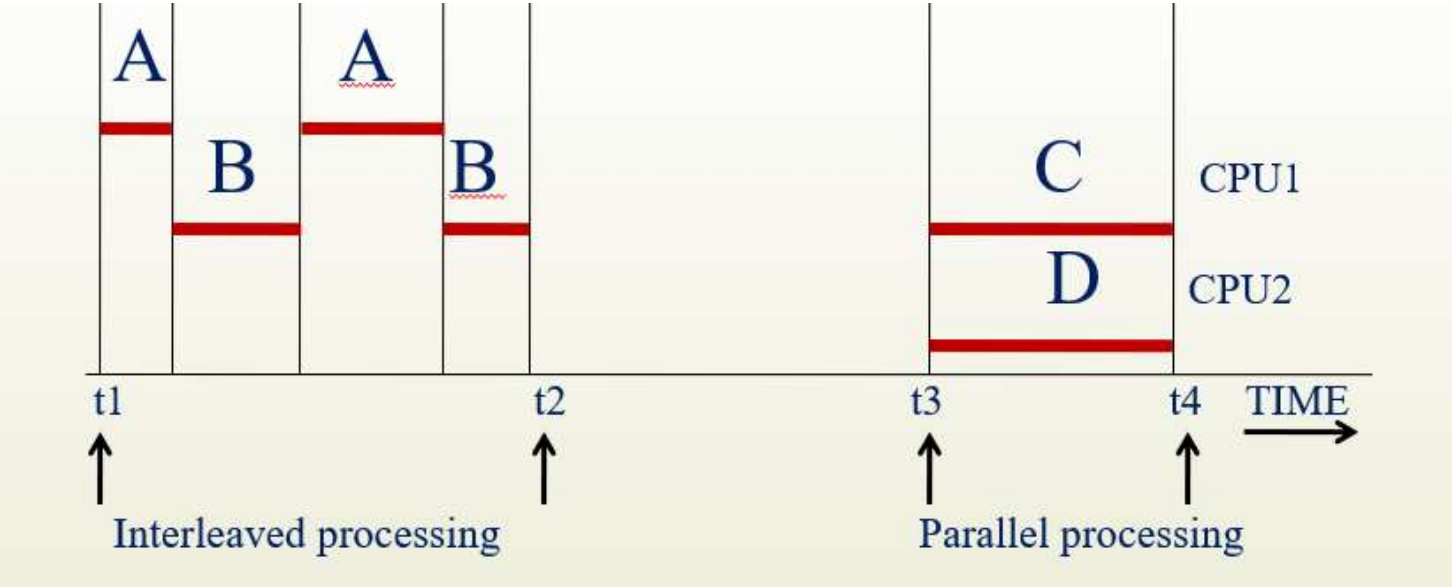
Sequential Execution

When transactions are strictly executed one after the other it is termed sequential execution

Concurrent Execution

When two or more transactions are interleaved, then we say that it is a concurrent execution

Interleaved Vs parallel processing of the concurrent tra



If the computer system has multiple hardware processors (CPUs), **parallel processing** of multiple processes is possible, as illustrated by processes C and D in the above figure.

Transaction

serves as a way to describe logical units of database processing that involve one or more database access operations.

Transaction Processing Systems:

The system with large databases and hundreds of concurrent users that are executing database transactions.

E.g., Banking, Airline reservations, credit card processing, etc.,

Operations on Transaction:

- All database access operations between **Begin Transaction** and **End Transaction** statements are considered one logical transaction.

- If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a **read-only transaction**.

Basic database access operations :

- **read_item(X) :** reads a database item X into program variable.
- **Write_item(X) :** Writes the value of program variable X into the database item X.

- **Example**:

Suppose an employee of a bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account	Y's Account
Open_Account(X)	Open_Account(Y)
Old_Balance = X.balance	Old_Balance = Y.balance
New_Balance = Old_Balance - 800	New_Balance = Old_Balance + 800
X.balance = New_Balance	Y.balance = New_Balance
Close_Account(X)	Close_Account(Y)

Transaction

A **transaction** is an atomic unit of work that is either completed in its entirety or not done at all.

- For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts.
- **The recovery manager keeps track of the following operations :**
  - **BEGIN_TRANSACTION**
  - **READ OR WRITE**
  - **END_TRANSACTION**
  - **COMMIT_TRANSACTION**
  - **ROLLBACK**

Transaction Model

Transaction states

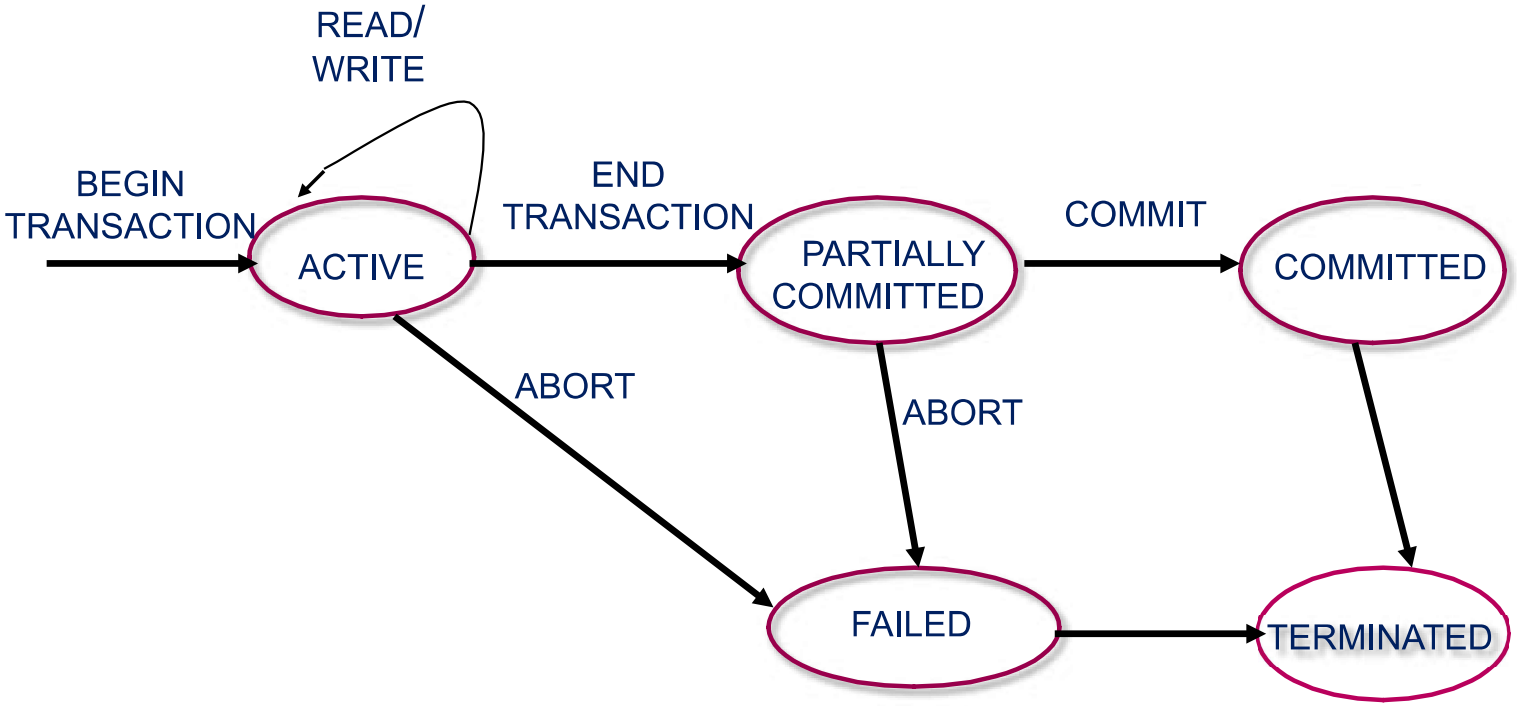


Figure 1: State transition diagram illustrating the states for transaction execution

Transaction Model

The Recovery manager keeps track of the following operations:

Begin Transaction: This is where transaction execution starts. When a transaction begins, it is in its **active state**.

Read/Write: These describe the read and write operations that are performed on database items as part of a transaction.

Commit Transaction: This indicates that the transaction has been completed successfully, allowing any changes (updates) made by the transaction to be safely committed to the database and not undone.

Partially Committed State: When the last statement is executed but the result is not written to the database, this state is achieved.

Transaction Model

Failed State: After discovering that the normal execution cannot be continued a transaction is aborted and reaches failed state. In order to ensure atomicity property, failed transactions should have no effect on the database

Rollback/ Abort: This indicates that the transaction failed and that any database modifications or effects caused by the transaction must be undone. As a result, the database must be restored to the state it was in just before the transaction started.

End Transaction: This indicates that read and write transaction activities have finished and that transaction execution has reached its completion.

- It's reached after the transaction's failure or success
- It may be required at this point to determine if the transaction's changes can be permanently applied to the database, or whether the transaction must be cancelled due to a violation of concurrency control or another reason.

Transaction Model

The Recovery manager keeps track of the following operations…

Undo: It works in a similar way as rollback, however, it only affects a single operation rather than the entire transaction.

Redo:

This specifies that specific transaction activities must be performed in order to guarantee that a committed transaction's operations have been succes sfully implemented in the database.

Force writing a log:

- Before a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk.
- This process is called force-writing the log file before committing a

Why Recovery is needed?

There are several possible reasons for a transaction to fail:

- ***A computer failure***: A ***hardware, software, or network error occurs*** in the computer system during transaction execution.
- ***A transaction or system error***: Some operations in the transaction may cause it to fail such as ***integer overflow or division by zero***. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, ***the user may interrupt the transaction*** during its execution.
- **Local errors or exception conditions Detected by the transaction.** For example, an ***insufficient account balance*** in a banking database may cause a transaction, such as a fund withdrawal from that account, to be cancelled.

Why Recovery is needed?

- ***Concurrency control enforcement***: The concurrency control method may decide to abort the transaction. Because it ***violates serializability or because several transactions are in a state of deadlock***

- ***Disk failure***: all disks or some disk blocks may lose their data because of a ***disk read/write head crash***

- ***Physical problems***: ***Disasters, theft, fire***, etc.

Therefore, the system must keep sufficient information to recover from the failure.

The System Log

The system **maintains a log** to keep track of all transaction operations that affect the values of database items. This log may be needed to recover from failures.

Types of log records :

- **[start_transaction,T] :** indicates that transaction T has started execution.
- **[write_item,T,X,old_value,new_value] :** indicates that transaction T has changed the value of database item X from old_value to new_value.
  (new_value may not be recorded)
- **[read_item,T,X]:** indicates that transaction T has read the value of database item X.
- (read_item may not be recorded)
- **[commit,T]:** transaction T has been recorded permanently.
- **[abort,T]:** indicates that transaction T has been aborted.

Transaction Properties

ACID should be enforced by the concurrency control and recovery methods of the DBMS.

ACID properties of transactions :

- **Atomicity:** a transaction is an atomic unit of processing; it is either performed entirely or not performed at all.
- **(It is the responsibility of recovery)**

- ***C*onsistency** : transfer the database from one consistent state to another consistent state [compatible or in agreement with something---standard and effect over time]
- (It is the responsibility of the applications and DBMS to maintain the constraints)

Transaction Properties

– **Isolation**: the execution of the transaction should be isolated from other transactions (Locking)

(It is the responsibility of concurrency)

* Isolation level:

-Level 0 (no dirty read)
-Level 1 (no lost update)
-Level 2 (no dirty+ no lost)
-Level 3 (level 2+repeatable reads)

– **Durability**: committed transactions must persist in the database,
– i.e. those changes must not be lost because of any failure.

(It is the responsibility of recovery)

Concurrency Control

In a database management system (DBMS), concurrency control handles simultaneous access to a database. It prevents two or more users from simultaneously accessing the same record and serializes transactions for backup and recovery.

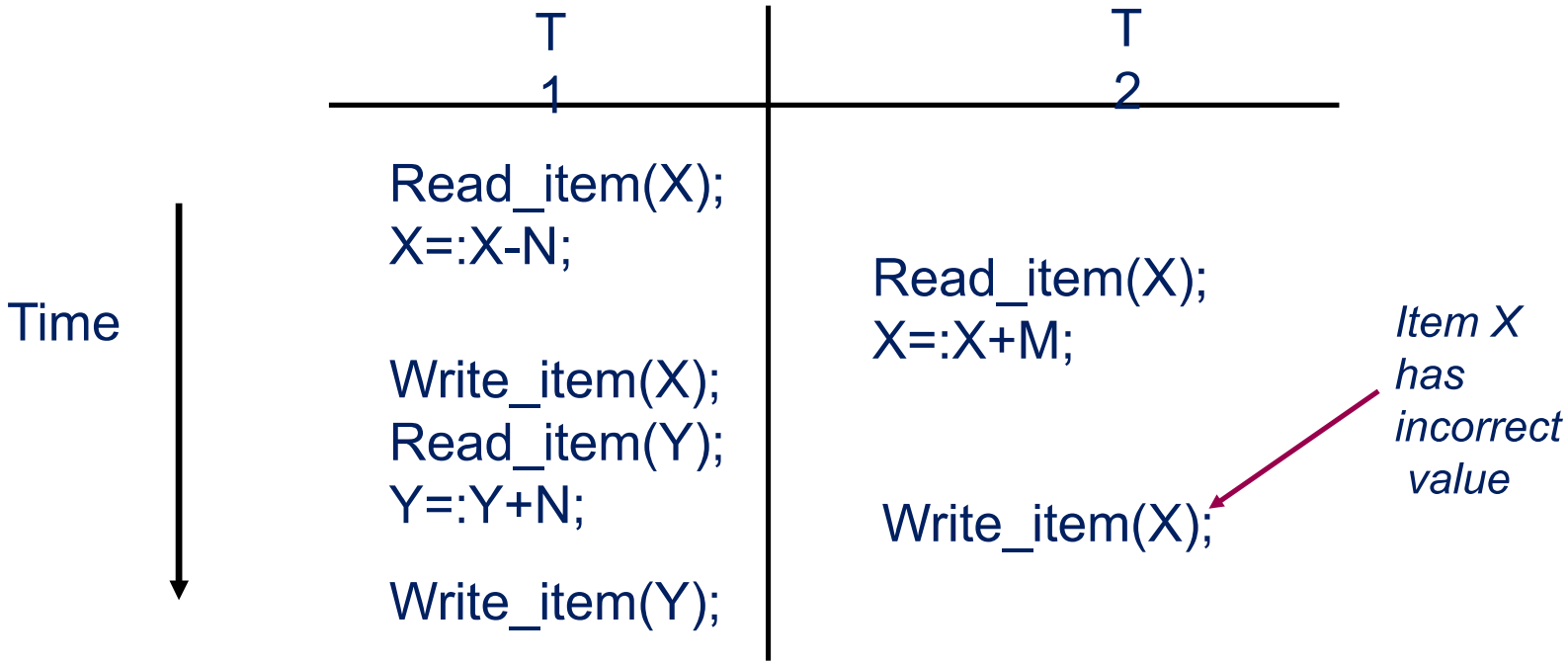
Why the concurrency control is essential?

Several problems can occur when concurrent transactions execute in an uncontrolled manner:

- Ø **The Lost Update Problem**
- Ø **The Temporary Update (or Dirty Read) Problem**
- Ø **The Incorrect Summary Problem**
- Ø **Unrepeatable Read problem**

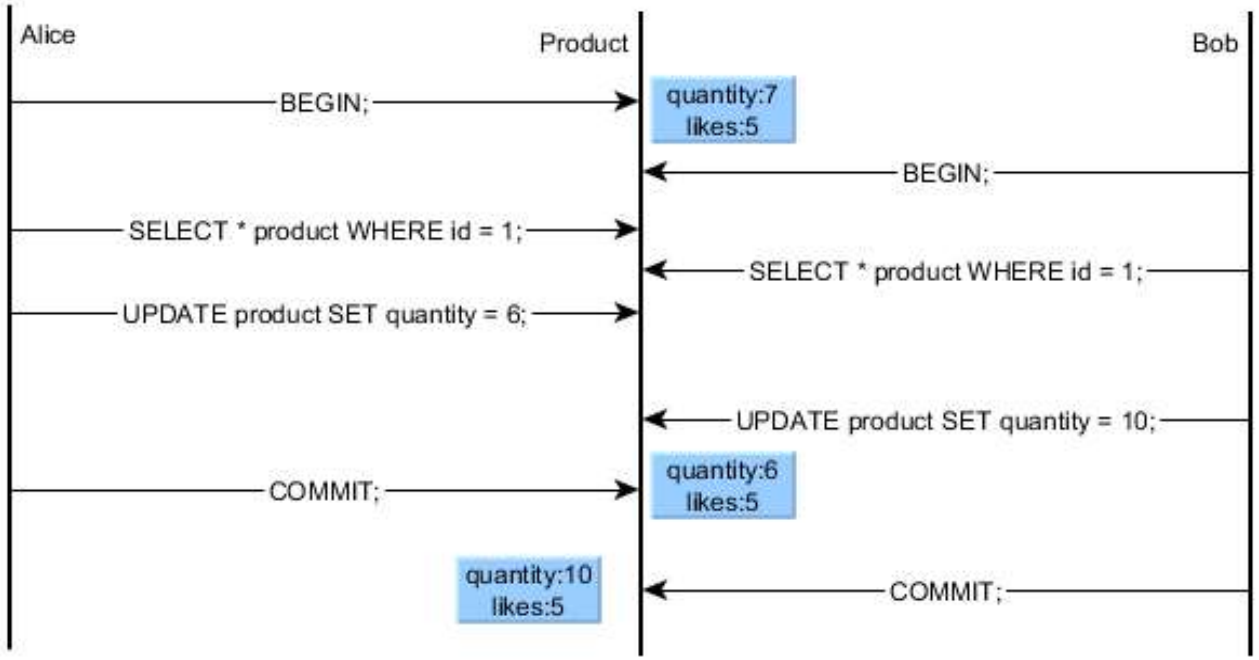
Concurrent Transactions and Issues

1. **The Lost Update Problem** : Example 1:

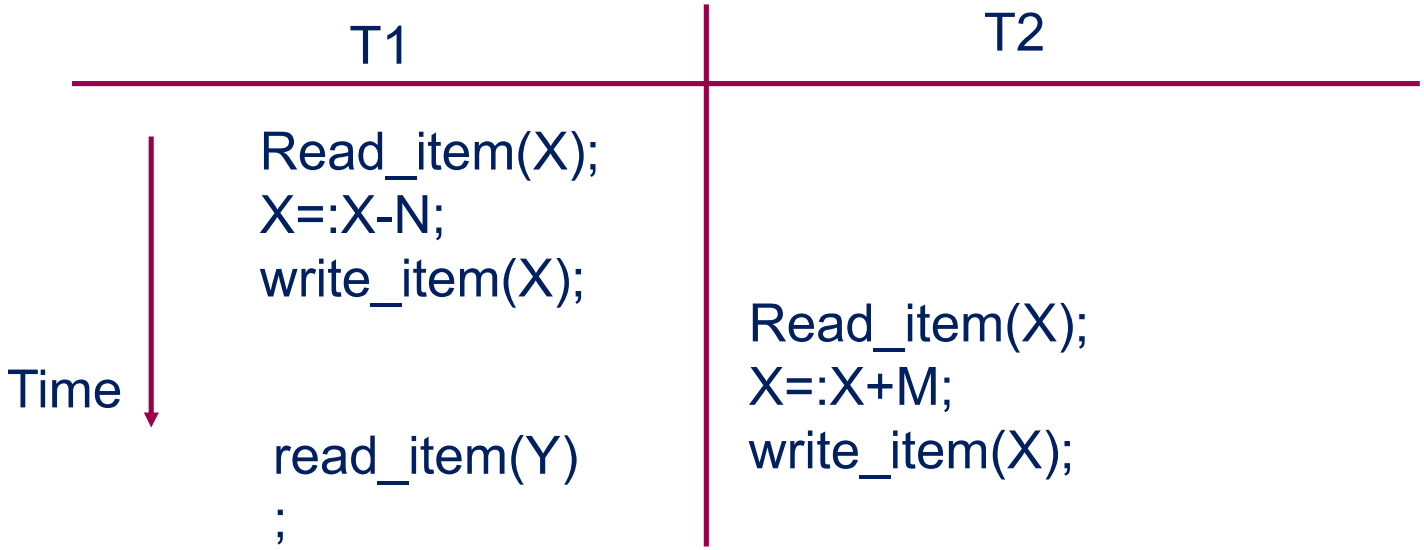


Concurrent Transactions and Issues

Example: 2 **A lost update occurs when two different transactions are trying to update the same column on the same row within a database at the same time**

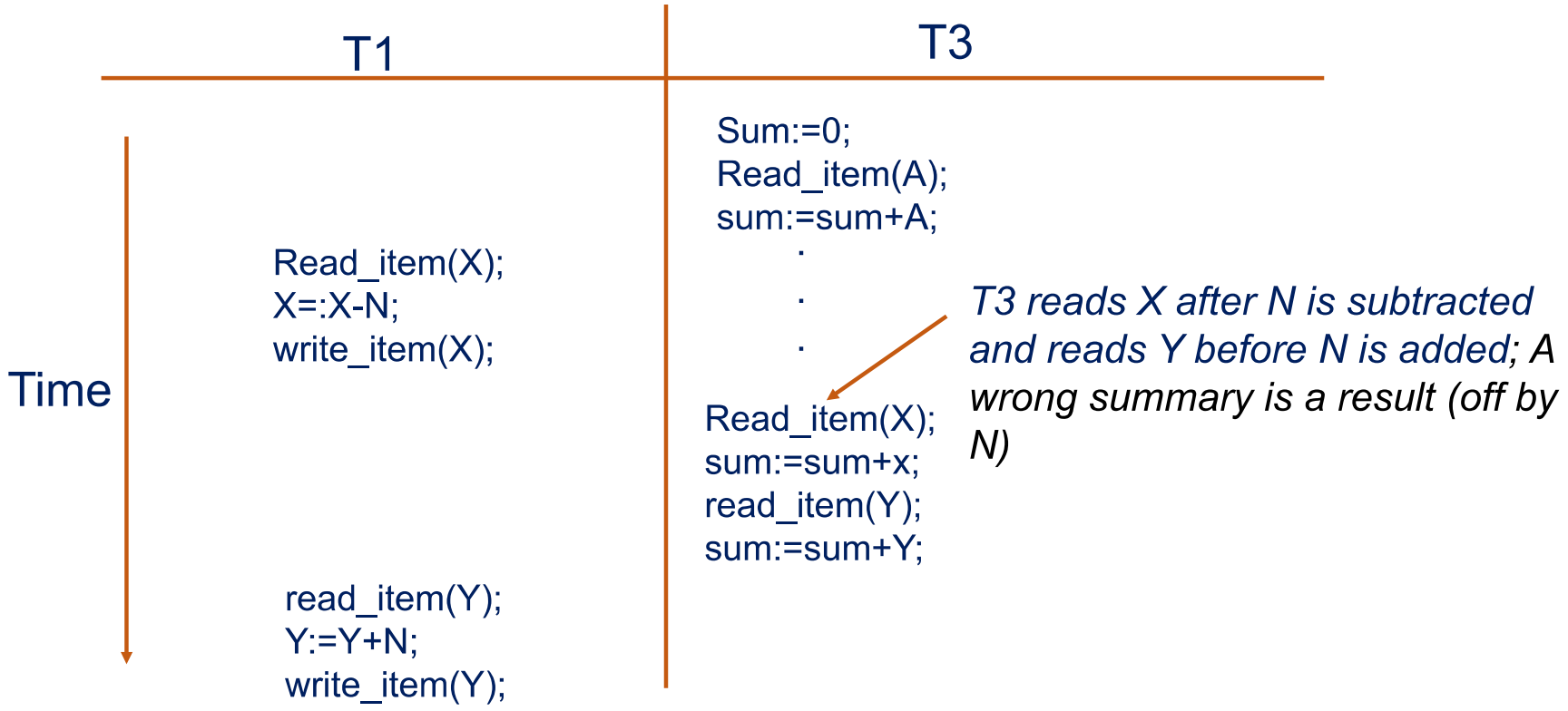


2. The temporary Update (or Dirty read) problem



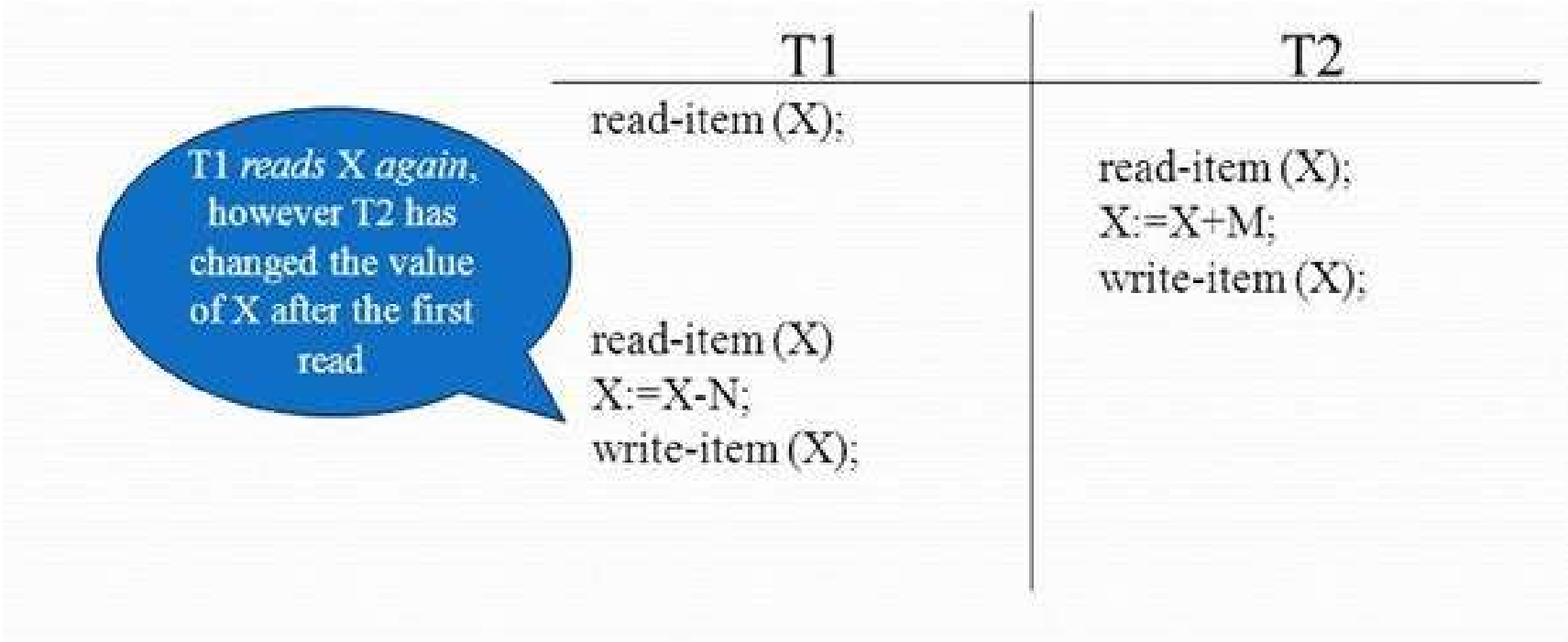
T1 fails and must rollback, meanwhile T2 has read the temporary value

3. The incorrect summary problem



4. **Unrepeatable Read problem**: **A transaction reads items twice with two different values because it was changed by another transaction between the two reads.**

*Also known as **Inconsistent Retrievals Problem that** occurs when in a transaction, two different values are read for the same database item.*



Concurrent Transactions and Issues

5. Phantom Read Problem: **The phantom read problem happens when a transaction reads a variable once but then encounters an error stating that the variable does not exist when it tries to read it again.**

T1	T2
Read (A)	
	Read (A)
Delete (A)	
	Read (A)

Table 1

In the example above Table 1, after transaction 2 reads the variable A, transaction 1 deletes the variable A without the knowledge of transaction 2. As a result, when transaction 2 attempts to read A, it is unable to do so.

Transaction Schedules & Serializability

Schedule(or History) : A series of operations from one transaction to another transaction is known as a schedule. It is used to preserve the order of the operation in each of the individual transaction.

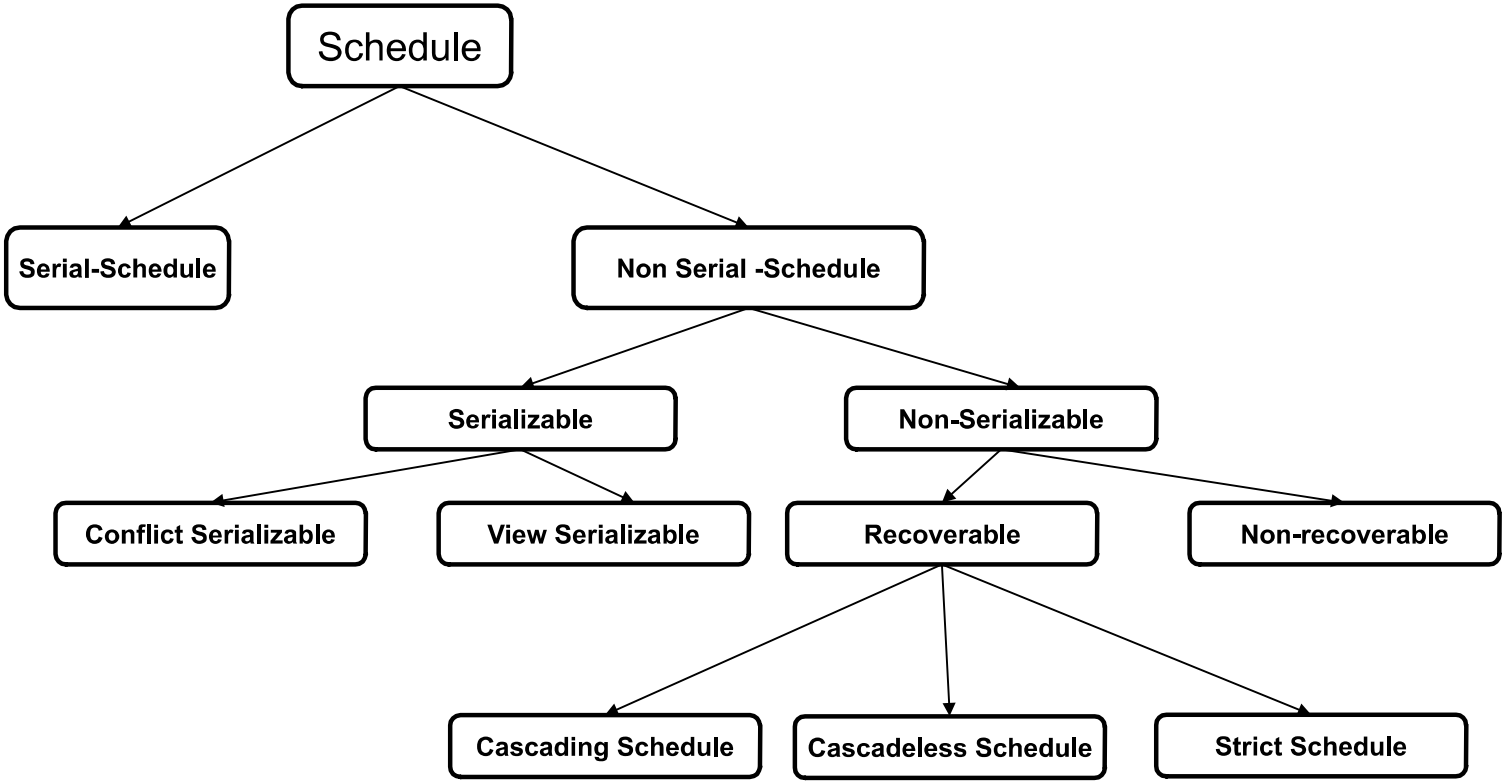
Schedule(or History) S of n transactions T1, T2,.., Tn is an ordering of operations of the transactions with the following conditions :

- for each transaction Ti that participates in S, the operations of Ti in S must appear in the same order in which they occur in Ti
- the operations from other transactions Tj can be interleaved with the operations of Ti in S.

The symbols **r,w,c,** and **a** are used for the operations **read_item, write_item,**

commit, and abort respectively.

Transaction Schedules & Serializability



Transaction Schedules & Serializability

Serial Schedule

- In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.
- All the transactions execute serially one after the other.
- When one transaction executes, no other transaction is allowed to execute.

Time

T1	T2
Read(X); X:=X-N Write(X); Read(Y); Y:=Y+N; Write(B);	
	Read(X); X:=X+M; Write(X);

Figure (a)
Schedule A

Time

T1	T2
	Read(X); X:=X+M; Write(X);
Read(X); X:=X-N Write(X); Read(Y); Y:=Y+N; Write(B);	

Figure (b)
Schedule B

Characteristics

Serial schedules are always-
- Consistent
- Recoverable
- Cascadeless
- Strict

Schedule A shows the serial schedule where T1 is followed by T2.

Schedule B shows the serial schedule where T2 is followed by T1.

Non-serial Schedule

- If interleaving of operations is allowed, then there will be a non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

Characteristics-

Non-serial schedules are **NOT** always-

- Consistent
- Recoverable
- Cascadeless
- Strict

Transaction Schedules & Serializability

Non-serial Schedule

Time

T1	T2
Read(X);	
X:=X-N	
	Read(X);
	X:=X+M;
Write(X);	
Read(Y);	
	Write(X);
Y:=Y+N;	
Write(B);	

Figure (c)
Schedule C

Time

T1	T2
Read(X);	
X:=X-N	
Write(X);	
	Read(X);
	X:=X+M;
	Write(X);
Read(Y);	
Y:=Y+N;	
Write(B);	

Figure (d)
Schedule D

Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have to interleave their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

These are of two types:

i) Conflict Serializable

ii) View Serializable

Note: A concurrent schedule whose result is the same as that of a serial schedule is called a ***concurrent serializable schedule***

i) Conflict Serializable

A schedule S is said to be conflict serializable if it is *conflict equivalent* to some serial schedule S'.

Conflict Operations:

For transactions T1 & T2 the order of read operation on any data element does not matter.

{T1R(Q), T2R(Q)} or {T2R(Q), T1R(Q)} does not matter.

The result is the same and ***does not lead to any conflict***.

Conflicting Operations:

Here, Q is the data element.

But {T1R(Q), T2W(Q)} is *not* same as {T2W(Q), T1R(Q)}

If **Ii** and **Ij** are the operations (instructions) of two different transactions on the same data item, and at least one of these instructions is a WRITE operation then we say that Ii and Ij ***are conflict operations***.

Here **I** stands for instruction and **i** and **i** are transactions

Transaction Schedules & Serializability

Hence it is evident that if ***we swap non-conflicting operations of a concurrent schedule, it will not affect the final result***.

Example

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

(S_1)

Concurrent schedule with T_1 & T_2 accessing A, B (data item)

T_1	T_2
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)
W(B)	
	R(B)
	W(B)

(S_2)

Swap W(A) in T_2 with R(B) in T_1 (because they are non conflicting)

T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
W(B)	
	W(A)
	R(B)
	W(B)

(S_3)

Swap R(A) of T_2 with R(B) of T_1 and W(A) in T_2 with W(B) in T_1 (Since non conflicting)

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

(S_4)

Swap R(A) of T_2 with W(B) of T_1

Now, the final schedule (S_4 is a serial schedule

Conflict Equivalent Schedules

We say S and S' are conflict equivalent if a schedule S can be transformed into a schedule S' by swapping non-conflicting instructions.

Furthermore, if a schedule S is ***conflict equivalent*** to a serial schedule, it is said to be ***conflict serializable***.

In the preceding example***, S4 is a serial schedule that is conflict equivalent to S1.***

As a result, S1 is a conflict serializable schedule

T_1	T_2
R(θ)	
	W(θ)
W(θ)	

Now, in this schedule, we can not perform any swap between instructions of T_1 and T_2 Hence, it is ***not conflict serializable***

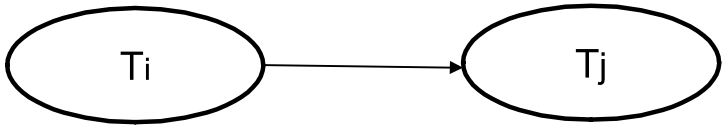
Test for Conflict Serializability

- Assume a schedule S.
- Let's create a graph called a **precedence graph/serialization graph** for S.
- G = (V, E) is a pair in this graph, where V is a collection of vertices and E is a set of edges.
- All of the transactions in the schedule are represented by a set of vertices.

- All edges **Ti ->Tj** that satisfy one of the three conditions are included in the set of edges:

i. Create a node Ti → Tj if Ti executes write (Q) before Tj executes read (Q).
ii. Create a node Ti → Tj if Ti executes read (Q) before Tj executes write (Q).
iii. Create a node Ti → Tj if Ti executes write (Q) before Tj executes write (Q).

Precedence Graph for Schedule S:



- If a precedence graph contains a single edge Ti → Tj, then all the instructions of Ti are executed before the first instruction of Tj is executed.
- If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

Transaction Schedules & Serializability

Example 1

Sh1: r1(J); r2(K); w2(J); r3(L); w3(K); w4(L);

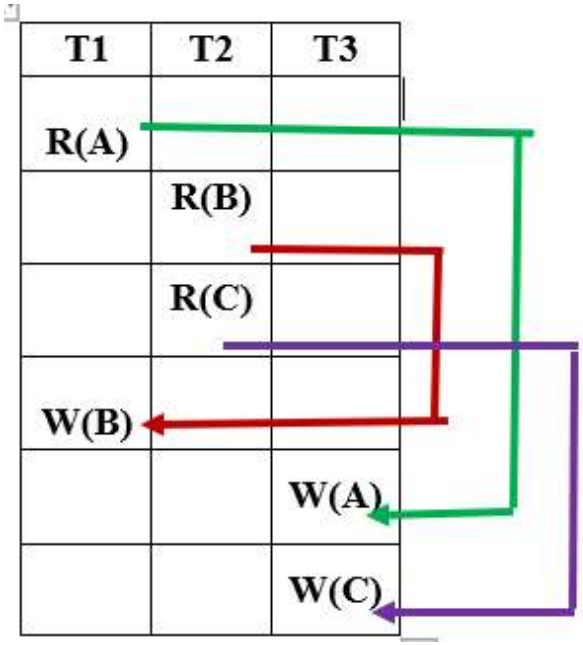
Cycle formed in the precedence graph in the given schedule Sh1.

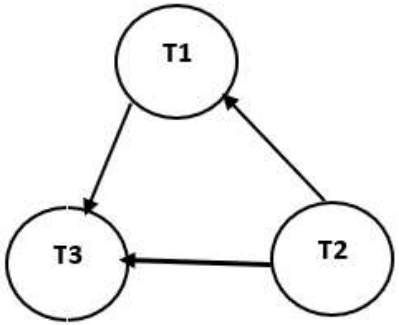
Therefore, it is "not conflict serializable".

Transaction Schedules & Serializability

Example 2

Sh2: r1(A); r2(B); r2(C); w1(B); w3(A); w3(C);





There is no cycle formed in the precedence graph in the given schedule Sh2.

Therefore, it is "Conflict serializable".

Transaction Schedules & Serializability

Conflict equivalent:

ü If the order of any two conflicting operations is the same in both schedules,

they are said to be conflict equivalent.

ü In the conflict equivalent, one can be transformed into another by swapping

non- conflicting operations.

ü Two schedules are said to be conflict equivalent if and only if:

- They contain the same set of transactions.
- If each pair of conflict operations are ordered in the same way.

Transaction Schedules & Serializability

ü In the following example,

S2 is conflict equivalent to S1

(S1 can be converted to S2 by swapping non-conflicting operations).

T1	T2
Read(X);	
Write(X);	
	Read(X);
	Write(X);
Read(Y);	
Write(Y);	
	Read(Y);
	Write(Y);

Fig 1 Schedule S1

T1	T2
Read(X);	
Write(X);	
Read(Y);	
Write(Y);	
	Read(X);
	Write(X);
	Read(Y);
	Write(Y);

Fig 2 Schedule S2

T1	T2
Read(X);	
Write(X);	
Read(Y);	
Write(Y);	
	Read(X);
	Write(X);
	Read(Y);
	Write(Y);

Fig 3 After swapping of non-conflict operations the Schedule S1 becomes

- Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2.
- Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.
- After swapping of non-conflict operations the schedule S1 (fig3), since S1 is conflict serializable)

Transaction Schedules & Serializability

Result Equivalent:

When two schedules produce the same final state of the database, they are said to be ***result equivalent.***

Note:

- Being serializable is not the same as being serial
- Being serializable implies that the schedule is the correct schedule.
  - It will leave the database in a consistent state.
  - The interleaving is appropriate and will result in a state as if the transactions were serially executed, yet will achieve efficiency due to concurrent execution.

ii) View Serializable

A schedule is view serializable if it is ***view equivalent*** to a serial schedule.

View Equivalent Schedules

Two schedules S and S' (where the same set of transactions participate in both schedules) are said to be view equivalent if the following ***three conditions are met***:

a) For each data item Q, if the transaction Ti reads the initial value of Q in S, then transaction Ti must in schedule S', also read the initial value of Q.

b) For each data item Q, if transaction Ti executes read (Q) in S, and the value produced by transaction Tj (if any) then transaction Ti must in schedule S also read the value of Q that was produced by transaction Tj .

c) For each data item Q, the transaction if any that performs the final write(Q) operation in schedule S must perform the final write(Q) operation in schedule S'

Now, we say that a schedule S is view serializable if it is view equivalent to a serial schedule

Transaction Schedules & Serializability

Non-Serial
S1

T1	T2
Read(X);	
Write(X);	
	Read(X);
	Write(X);
Read(Y);	
Write(Y);	
	Read(Y);
	Write(Y);

Serial
S2

T1	T2
Read(X);	
Write(X);	
	Read(X);
	Write(X);
Read(Y);	
Write(Y);	
	Read(Y);
	Write(Y);

We can say that given schedule s1 is view serializable if we can prove that they are *view equivalent*.

Note: Every conflict serializable schedule is view serializable. But not all view serializable schedules are conflict serializable.

Non-Serializability in DBMS

A non-serial schedule, which is not serializable is called as non-serializable schedule.

Non-serializable schedules may/may not be consistent or recoverable.

Non-serializable schedule is divided into types:

i. **Recoverable schedule**
ii. **Non-recoverable schedule**

Recoverable Schedule

Due to a software issue, system crash, or hardware malfunction, a transaction may not finish completely. The unsuccessful transaction must be rolled back in this scenario. However, the value generated by the unsuccessful transaction may have been utilized by another transaction. As a result, we must also roll back those transactions.

Transaction Schedules & Serialization

Non-recoverable/ Irrecoverable Schedule

T1	T1's Buffer space	T2	T2's Buffer space	Database
ead(X) ;	X = 5000			X = 5000
= X - 250 ;	X = 4750			X = 5000
rite (X) ;	X = 4750			X = 4750
		Read (X) ;	X = 4750	X = 4750
		A = A + 1000 ;	X = 5750	X = 4750
		Write (X) ;	X = 5750	X = 5750
		Commit ;		
ilure Point				
mmit ;				

Table 1

The above Table 1 shows a schedule that has two transactions.

T1 reads and writes the value of A and that value is read and written by T2.

T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1.

T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it is already committed. So this type of schedule is known as ***"non-recoverable/irrecoverable"*** schedule.

Irrecoverable schedule: The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

Recoverable with Cascade Rollback

T1	T1's Buffer space	T2	T2's Buffer space	Database
(X) ;	X = 5000			X = 5000
- 250 ;	X = 4750			X = 5000
(X) ;	X = 4750			X = 4750
		Read (X) ;	X = 4750	X = 4750
		A = A + 1000 ;	X = 5750	X = 4750
		Write (X) ;	X = 5750	X = 5750
e Point				
nit ;				
		Commit ;		

Table 2

The above Table 2 shows a schedule with two transactions.

Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1.

T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is ***recoverable with cascade rollback***.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

Cascadeless Recoverable Schedule

ead(X) ;	X = 5000			X = 5000
= X - 250 ;	X = 4750			X = 5000
rite (X) ;	X = 4750			X = 4750
ommit ;		Read (X) ;	X = 4750	X = 4750
		A = A + 1000 ;	X = 5750	X = 4750
		Write (X) ;	X = 5750	X = 5750
		Commit ;		

Table 3

The above Table 3 shows a schedule with two transactions.

Transaction T1 reads and write A and commits, and that value is read and written by T2.

So this is a ***cascade less recoverable schedule***.

Strict Recoverable Schedule

When a transaction is not allowed to access or write data until the last transaction that has written it is committed or aborted, the schedule is referred to as Strict Schedule.

Here, transaction T2 reads/writes the written value of transaction T1 only after the transaction T1 commits. Hence, the schedule is a ***strict schedule.***

Table 4

Let's have two transactions T1 and T2 in Table 4.

The write operation of transaction T1 precedes the read or write operation of transaction T2, so the commit or abort operation of transaction T1 should also precede the read or write of T2.

The Strict schedule allows only committed read and write operations.

This schedule implements more restrictions than cascadeless schedule.

SUMMARY

- **Transactions in DBMS**
- **Concurrent Transactions and Issues**
- **Schedules in DBMS**
  - Types of schedules
    - **Serial Schedule**
    - **Non-serial Schedule**
      - **Serializable**
        - Conflict Serializable & Test for Conflict Serializability
        - View Serializable
      - **Non-serializable**
        - **Non-Recoverable Schedule**
        - **Recoverable Schedule**
          - Cascading Schedule
          - Cascadeless Schedule
          - Strict Schedule