GITAM
(DEEMED TO BE UNIVERSITY)
(Estd. u/s 3 of the UGC Act, 1956)

CSEN2061 – Database Management Systems

Semester – 5

Unit – 5

Transactions

- A transaction is an execution of a user program.
- DBMS has several actions or transactions.
- A user can easily understand the effect of running their program and ensure the result of concurrent execution of transactions.
- Some serial or one-at-a-time execution of the same set of transactions.
- DBMS handles concurrent execution, it is an important aspect of transaction management, and it is subject to concurrency control.

Transactions

- DBMS also handles partial transactions or transactions that are interrupted before they run to normal completion.
- It ensures that changes made by such partial transactions are not seen by other transactions.
- This is achieved on the subject of crash recovery.

Properties Property (ACID)

- **Atomic**: either execute all actions or none of them. Users should not have to worry about the effect of incomplete transactions.
- **Consistency**: The database should be consistent before and after the execution of a transaction. Each transaction is run by itself with no concurrent execution of other transactions.
- **Isolation**: multiple transactions may execute concurrency; the system guarantees that one transaction finishes before the start of another transaction. Each transaction is unaware of other transactions executing concurrently in the system.

Properties Property (ACID)

- **Durability**: after the successful completion of the transaction, the changes it has been made to the database persist, even if there are system failures.

A Simple Transaction Model

A simple bank application consisting of several accounts and a set of transactions that access and update those accounts.

- **Read(X):** it transfers the data item X from the database to a variable, also called X, in the buffer main memory belonging to the transactions that executed the read operation.
- **Write(X):** it transfers value in the variable X in the main memory buffer of the transaction that executed the write to the data item X in the database.

A Simple Transaction Model

- Let *Ti* be a transaction that transfers $50 from account A to account B. This transaction can be defined as:

T_i: read(A);
$A := A - 50$;
write(A);
read(B);
$B := B + 50$;
write(B).

- The **consistency** requirement here is that the sum of A and B be unchanged by the execution of the transaction.

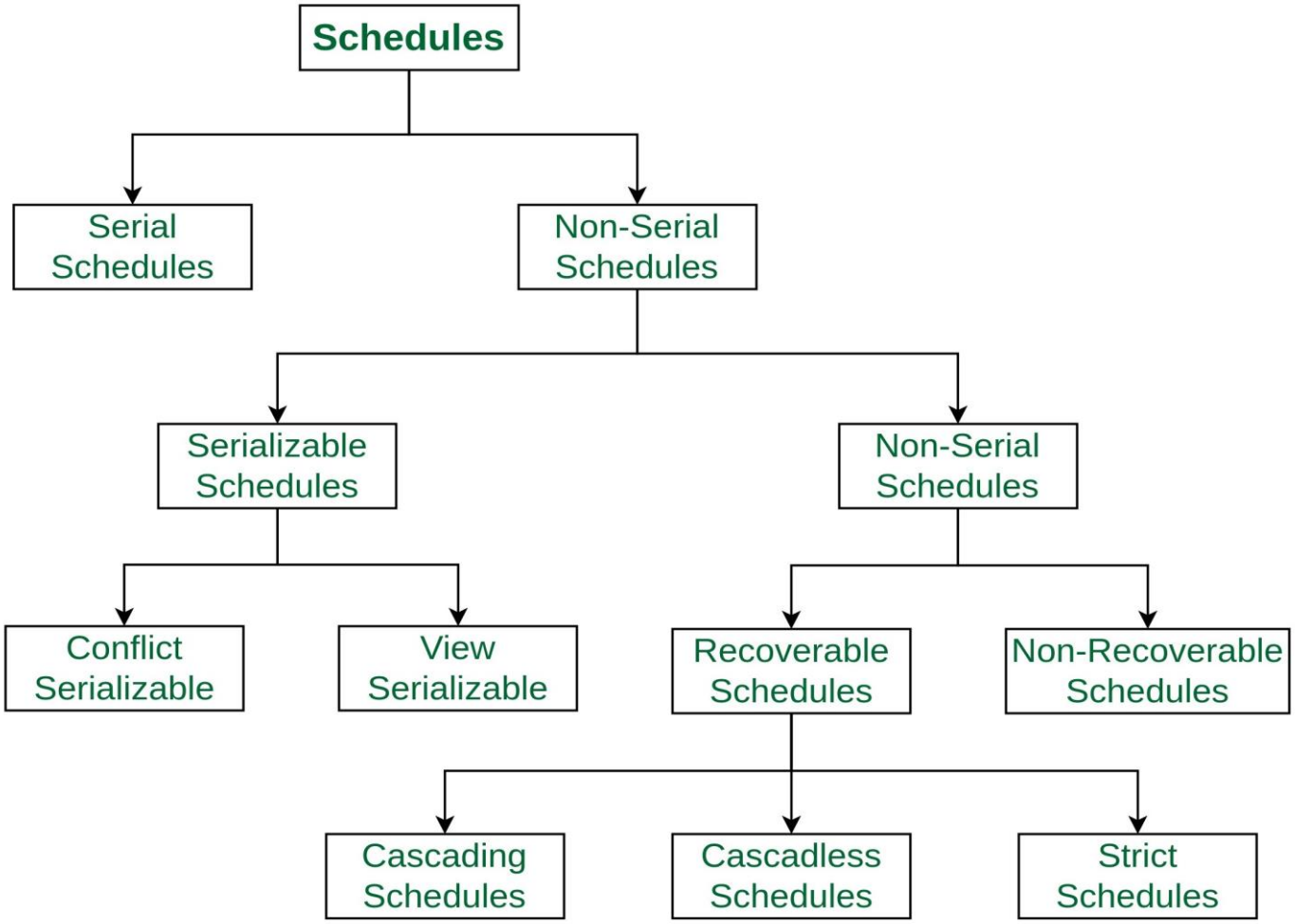
A Simple Transaction Model

- The values of accounts A and B are $1000 and $2000, respectively. Suppose write(A) is successful (A=1000-50=950) and write(B) is unsuccessful (B=2000) due to failures. The sum A + B is no longer preserved; therefore, it is inconsistent state. The system will not commit this transaction.
- if the consistency and atomicity properties are ensured for each transaction, if several transactions are executed concurrently, their operations may interleave in some undesirable way, resulting in an inconsistent state.
- Once the execution of the transaction is completed successfully, the user who initiated the transaction has been notified that the transfer of funds has taken place.

Scheduling

- A schedule defines the sequence in which various transactions read and write data items in the database.
- Sequences that indicate the chronological order in which instructions of concurrent transactions are executed.
- A schedule is a list of actions (reading, writing, aborting, or committing) from a set of transactions, and the order in which two actions of a transaction T appear in a schedule must be the same as the order in which they appear in T.

Types of Scheduling



Serial Schedule

- After the commit of one transaction begins another transaction.
- Number of possible serial schedules with n transactions is factorial n.
- No inconsistency

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit

Non-serial Schedule

- The Non-Serial schedule is a type of schedule where transactions are executed concurrently, with some overlap in time.

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

Serializable Schedules

- Serial schedules are serializable, but if steps of multiple transactions are interleaved, it is harder to determine whether a schedule is serializable.
- A schedule is serializable if it is equivalent to a serial schedule.
- It is categorized into two categories, which are below.
  - Conflict Serializability
  - View Serializability

Conflict Serializable

A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations. Two operations are said to be conflicting if all conditions are satisfied:

- They belong to different transactions
- They operate on the same data item
- At Least one of them is a write operation

Example:

T1 and T2 are two different transactions. A pair of actions of Read(R) and Write(W) operations between two different transactions. Conflict pairs are below.

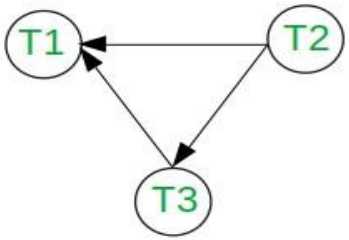
1. Read Write (RW)
2. Write Read (WR)
3. Write Write (WW)

Conflict Serializable

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

Schedule S

Precedence Graph



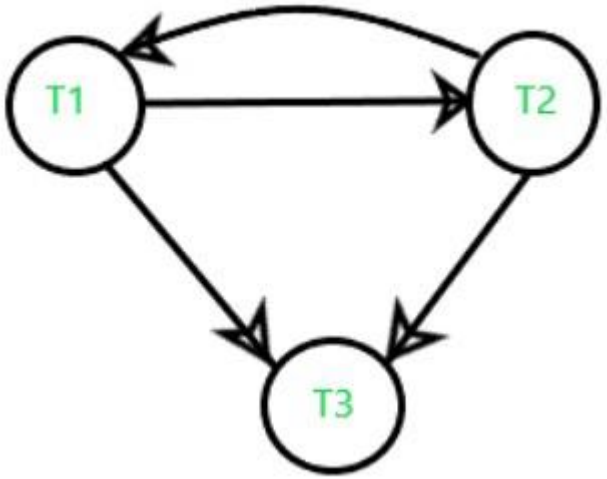
Conflict Serializable

- In transactions, node T2 has indegree 0.
- So, select T2 and remove T2 and all edges connecting from it.
- Now T3 has indegree 0. So, select T3 and remove the edge T3→T1.
- At the end, select T3. So the topological Sorting is T2, T3, T1.
- Hence, the equivalent serial schedule of the given conflict serializable schedule is T2→T3→T1, i.e., S2: R2(A) W2(A) W2(B) R3(C) W3(A) W3(C) R1(A) R2(B) W1(A) W1(B).

VIEW-SERIALIZABLE

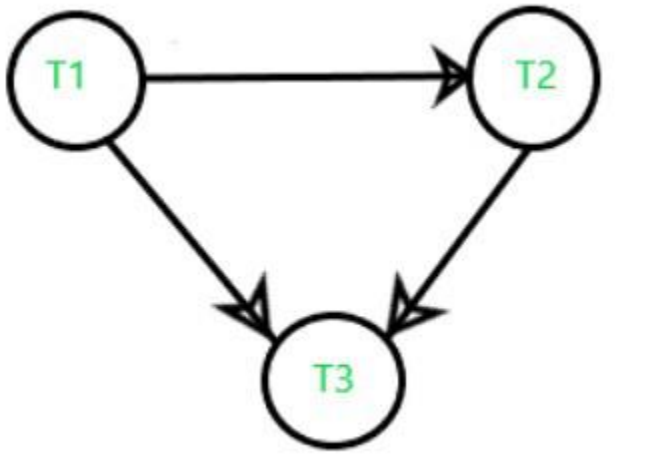
- A schedule is said to be View-Serializable if it is view equivalent to a Serial Schedule (where no interleaving of transactions is possible).

T1	T2	T3
a=100 read(a)		
	a=a-40 write(a) //60	
a=a-40 write(a) //20		
		a=a-20 write(a) //0



VIEW-SERIALIZABLE

T1	T2	T3
a=100 read(a)		
a=a-40 write(a) //60		
	a=a-40 write(a) //20	
		a=a-20 write(a) //0



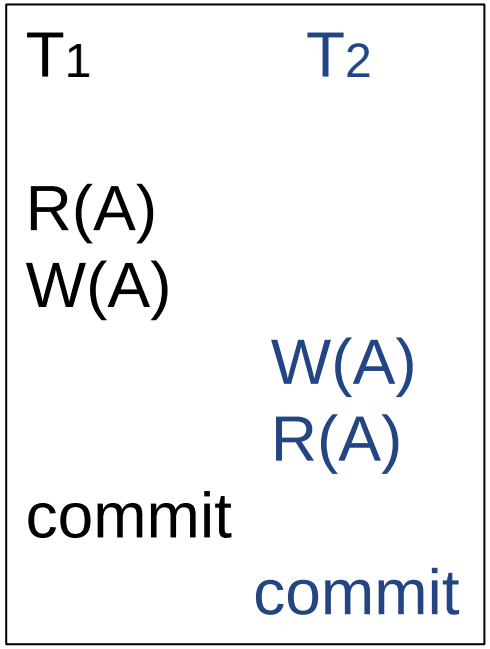
Now, the precedence graph of the table does not contain any cycle/loop, which means it is conflict serializable (equivalent to serial schedule, consistent), and the final result is the same as the first table.

NON-SERIALIZABLE SCHEDULE

- The non-serializable schedule is divided into two types: Recoverable and Non-recoverable Schedule.
- Recoverable schedules are Cascading, cascade-less, and Strict schedules.

Recoverable schedules

- Schedules in which transactions commit only after all transactions whose changes they read commit are called recoverable schedules.
- In other words, if some transaction Tj is reading value updated or written by some other transaction Ti, then the commit of Tj must occur after the commit of Ti.



This is a recoverable schedule since T1 commits before T2, which makes the value read by T2 correct.

Cascading Schedules

When there is a failure in one transaction, and this leads to the rolling back or aborting of other dependent transactions, then such scheduling is referred to as Cascading rollback or cascading abort.

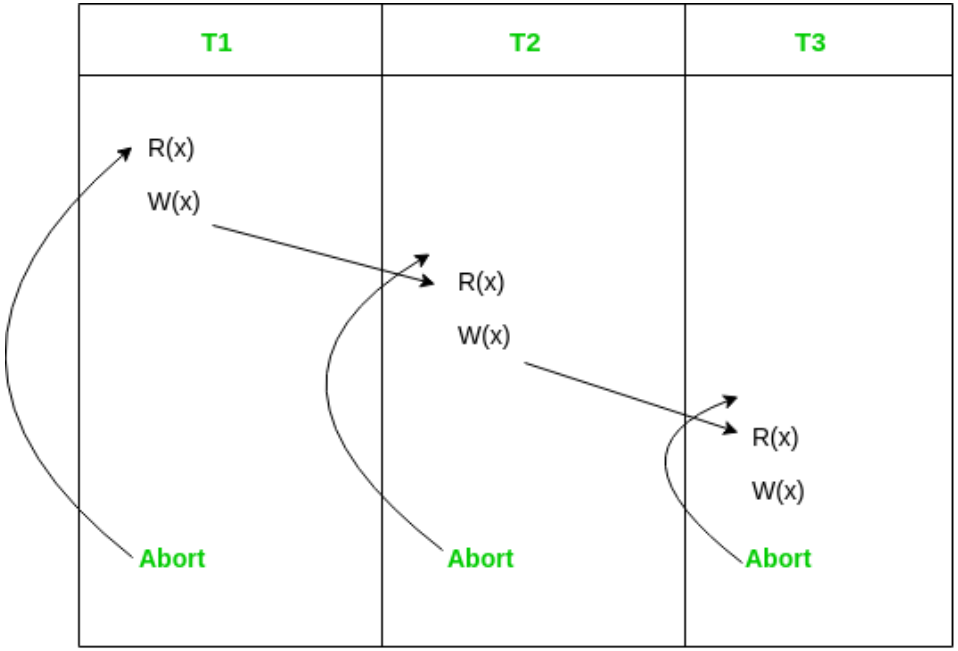
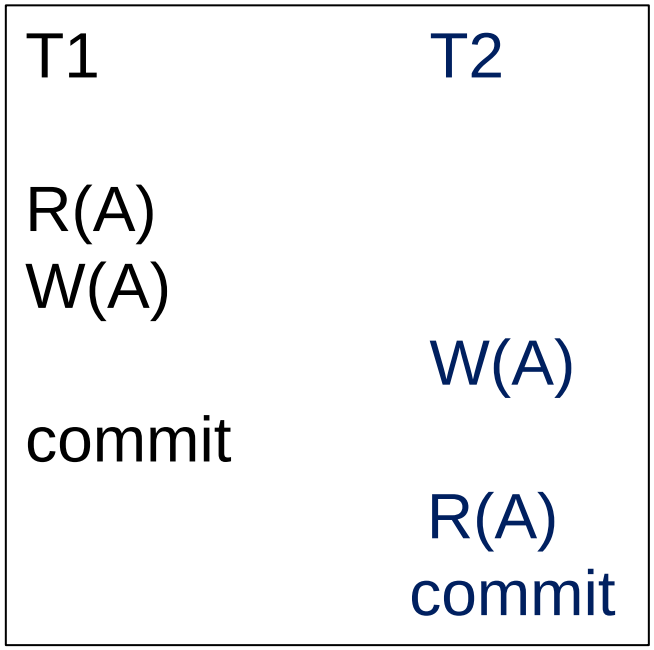


Figure - Cascading Abort

Cascade-less Schedule

- Transactions read values only after all transactions whose changes they will read commit are called cascade-less schedules.
- It avoids that a single transaction abort leads to a series of transaction rollbacks.
- In other words, if some transaction Tj wants to read value updated or written by some other transaction Ti, then the commit of Tj must read it after the commit of Ti.



This schedule is cascade-less. Since the updated value of A is read by T2 only after the updating transaction, i.e., T1 commits.

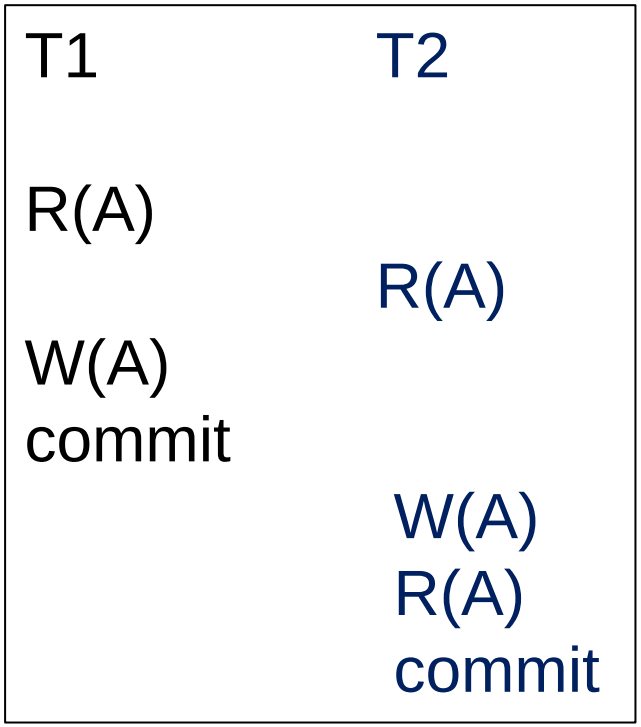
Cascade-less Schedule

It is a recoverable schedule, but it does not avoid cascading aborts. It can be seen that if T1 aborts, T2 will have to be aborted, too, to maintain the schedule's correctness, as T2 has already read the uncommitted value written by T1.

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
abort	
	abort

Strict Schedule

- A schedule is strict if any two transactions Ti, Tj, if a write operation of Ti precedes a conflicting operation of Tj (either read or write), then the commit or abort event of Ti also precedes that conflicting operation of Tj.
- In other words, Tj can read or write Ti updated or written values only after Ti commits/aborts.
- *Example*: Consider the schedule involving two transactions, T1 and T2.



Non-Recoverable Schedule

- *Example*: Consider the following schedule involving two transactions, T1 and T2.
- T2 read the value of A written by T1 and committed. T1 later aborted. Therefore, the value read by T2 is wrong, but since T2 committed, this schedule is non-recoverable.