COURSE : DATABASE MANAGEMENT SYSTEMS
TOPIC : SCHEDULES & SERIALIZABILITY

COURSE CODE: 23AD2102R

Session - 3

AIM OF THE SESSION

To familiarize students with the basic concept of **Transaction Management**

INSTRUCTIONAL OBJECTIVES

a) The concept of Transaction
b) Main Operations in a Transaction
c) Popular Examples of Transaction Processing
d) Transaction Processing Issues
e) Transaction States

LEARNING OUTCOMES

At the end of this session, you should be able to understand :

Transaction Processing Issues & States of Transaction Processing

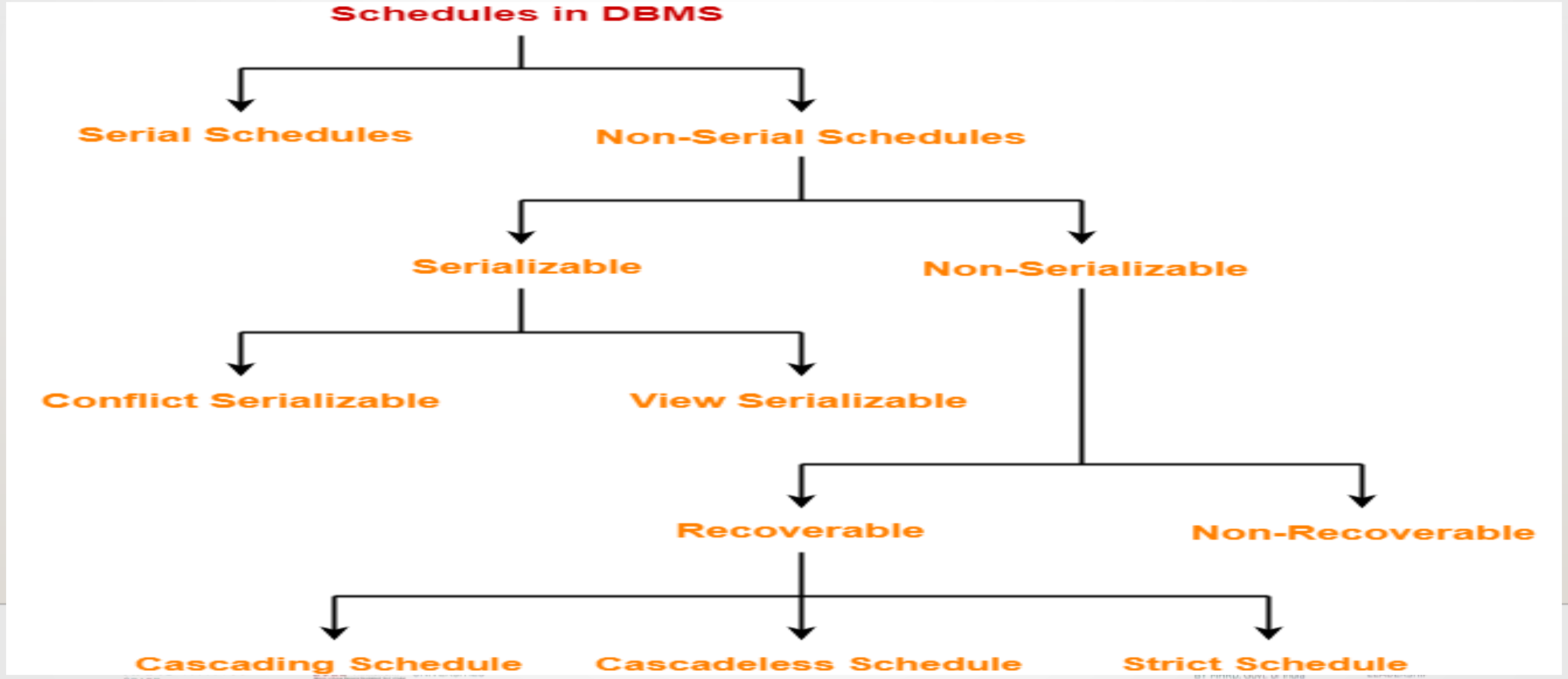
SESSION OBJECTIVE

- At the end of the Session the students will

- **Understand the concept of Schedules**
- **Serialization**

WHAT IS SCHEDULE ?

- The ***order*** in which the operations of multiple transactions appear for execution is called as a schedule.



SERIAL SCHEDULES

- All the transactions execute serially one after the other.
- When one transaction executes, no other transaction is allowed to execute.

Transaction T1	Transaction T2
R (A)	
W (A)	
R (B)	
W (B)	
Commit	
	R (A)
	W (B)
	Commit

SERIAL SCHEDULES

In the example shown in the previous slide,

- There are two transactions T1 and T2 executing serially one after the other. Transaction T1 executes first.
- After T1 completes its execution, transaction T2 executes.
- So, this schedule is an example of a ***Serial Schedule***.
- Example 02

Transaction T1	Transaction T2
	R (A)
	W (B)
	Commit
R (A)	
W (A)	
R (B)	
W (B)	
Commit	

CHARACTERISTICS OF SERIAL SCHEDULES

- *Consistent*
- *Recoverable*
- *Cascade less*
- *Strict*

NON-SERIAL SCHEDULES

- In non-serial schedules, Multiple transactions execute concurrently.
- Operations of all the transactions are inter leaved or mixed with each other.

Transaction T1	Transaction T2
R (A)	
W (B)	
	R (A)
R (B)	
W (B)	
Commit	
	R (B)
	Commit

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 interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially

FINDING THE NUMBER OF SCHEDULES

- Consider there are n number of transactions T1, T2, T3 …., Tn with N1, N2, N3 …., Nn number of operations respectively, then the

  Total number of possible schedules (serial + non-serial) is given by

$$\frac{(N1 + N2 + N3 + ... + Nn)!}{N1! \times N2! \times N3! \times .... \times Nn!}$$

Total Number of Serial Schedules-

Total number of serial schedules= Number of different ways of arranging n transactions = ***n!***

Total Number of Non-Serial Schedules-

Total number of non-serial schedules = Total number of schedules – Total number of serial schedules

PROBLEM

- *Consider there are three transactions with 2, 3, 4 operations respectively, find-*

1. *How many total number of schedules are possible?*
2. *How many total number of serial schedules are possible?*
3. *How many total number of non-serial schedules are possible?*

$$\text{Total number of schedules} = \frac{(2+3+4)!}{2! \times 3! \times 4!}$$

$$= 1260$$

SOLUTION

Total number of serial schedules

= Number of different ways of arranging 3 transactions

= 3!

= 6

Total number of non-serial schedules

= Total number of schedules – Total number of serial schedules

= 1260 – 6

= 1254

RECOVERABLE AND IRRECOVERABLE SCHEDULES

- Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be ***rollback.***
- But some other transaction may also have used value produced by the failed transaction. So we also have to ***rollback*** those transactions.
- ***Irrecoverable*** :The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

IRRECOVERABLE SCHEDULE

- The above table 1 shows a schedule which 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 already committed. So this type of schedule is known as irrecoverable schedule.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

RECOVERABLE SCHEDULE(CASCADE ROLLBACK)

- 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.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

RECOVERABLE SCHEDULE(CASCADELESS ROLLBACK)

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

UNDERSTANDING *BLIND WRITE*

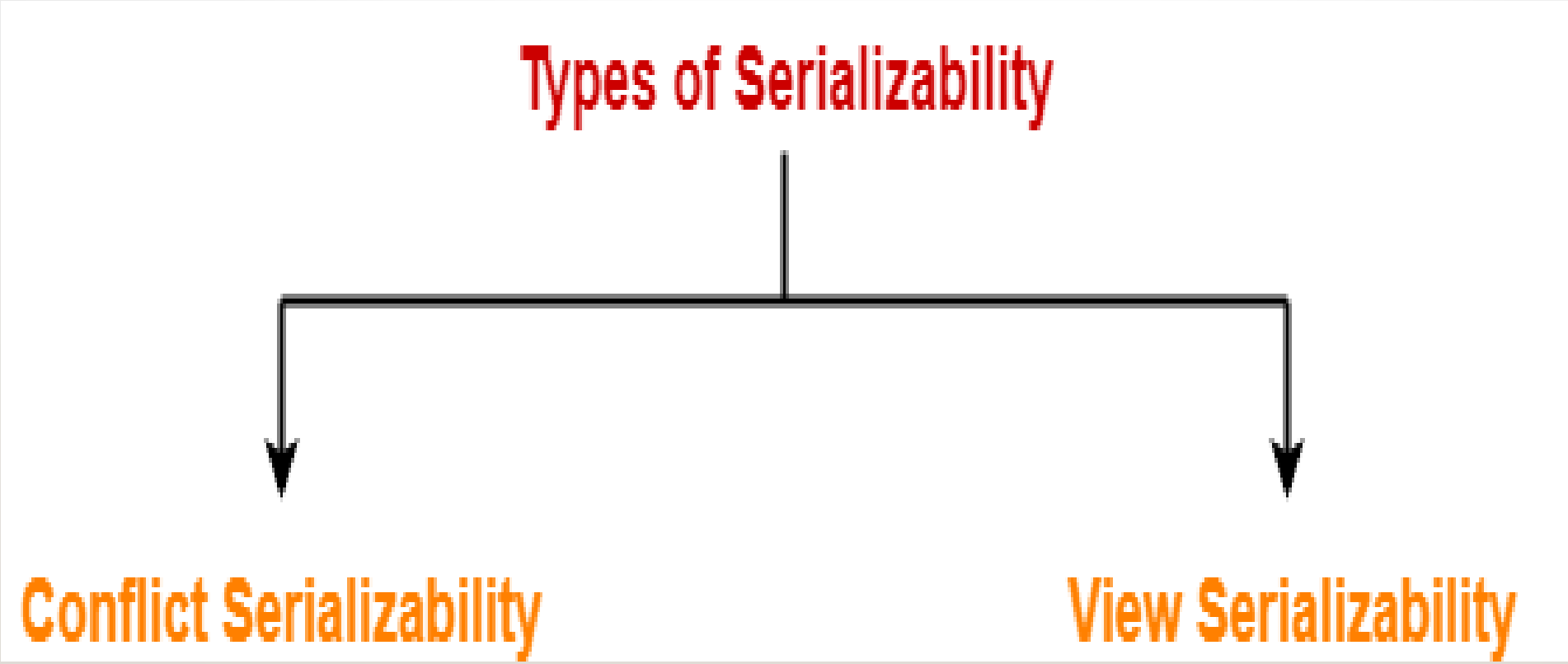
- ***Blind write*** is simply when a transaction **writes** without reading. i.e a transaction have **WRITE**(Q), but no READ(Q) before it. So, the transaction is **writing** to the database "blindly" without reading previous value.
- If there is no read that happens prior to the first write then it is said to be a *blind write*.

T1	T2	T3
	$R_2(X)$	
$R_1(X)$		
		$W_3(X)$
	$W_2(X)$	

- In the above example, **W3(X)** is a ead before write [i.e there is no R3(X) before W3(X)]
- **W2(X)** is not a blind write, as a read happens before write [R2(X) before W2(X)]

SERIALIZABILITY IN DBMS-

- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.



CONFLICT SERIALIZABILITY

- A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
- The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

Conflicting Operations

The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

NON-CONFLICT

- Swapping is possible only if S1 and S2 are logically equal.
- Here, S1 = S2. That means it is non-conflict.

1. T1: Read(A) T2: Read(A)

T1	T2
Read(A)	
	Read(A)

Schedule S1

Swapped

T1	T2
	Read(A)
Read(A)	

Schedule S2

CONFLICT

- Here, S1 ≠ S2. That means it is conflict.

2. T1: Read(A) T2: Write(A)

T1	T2
Read(A)	
	Write(A)

Schedule S1

Swapped

T1	T2
	Write(A)
Read(A)	

Schedule S2

CONFLICT EQUIVALENT

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

1. They contain the same set of the transaction.
2. If each pair of conflict operations are ordered in the same way.

- In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations.

CONFLICT EQUIVALENT

- In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations).
- 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.

Non-serial schedule

T1	T2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Schedule S1

Serial Schedule

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S2

CONFLICT EQUIVALENT

After swapping of non-conflict operations, the schedule S1 becomes:

T1	T2
Read(A) Write(A) Read(B) Write(B)	Read(A) Write(A) Read(B) Write(B)

VIEW SERIALIZABILITY

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

- *If a schedule is conflict serializable, then it will be view serializable.*
- *The view serializable which does not conflict serializable , contains blind writes.*
- ***View Equivalent :***
- Two schedules S1 and S2 are said to be view equivalent if they satisfy the following THREE conditions:

  a) Initial Read

  b) Updated Read

  c) Final Write

VIEW SERIALIZABILITY-INITIAL READ

- An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	Write(A)

Schedule S1

T1	T2
Read(A)	Write(A)

Schedule S2

- Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

VIEW SERIALIZABILITY-UPDATED READ

- In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)

Schedule S1

T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

Schedule S2

- Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

VIEW SERIALIZABILITY -FINAL WRITE

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.

SUMMARY OF SCHEDULES

- *Serial schedules are always consistent.*
- *Non-serial schedules are not always consistent.*
- *Non-serial schedules may be serializable or non-serializable*

SELF-ASSESSMENT QUESTIONS

1. A True Transaction updates the Database

(a) True
(b) False
(c) Can't Say
(d) None of the Above

2. At the end of every transaction, every transaction concludes with

(a) Read ( ) Operation
(b) Commit ( ) Operation
(c) Roll Back ( ) Operation
(d) None of the Above

TERMINAL QUESTIONS

1. Discuss about the Concept of Transaction Processing
2. List out the Steps of Operations in a Transaction Processing
3. Why a simple Operation on a database can not be a Transaction ?
4. Discuss about various Issues during a Transaction ?
5. Draw the diagram of Transaction States and Explain ?

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books:

1. "Database Management Systems" by Raghu Ramakrishnan and Johannes Gehrke - This is a comprehensive textbook that covers both theoretical and practical aspects of DBMS, including data modeling, relational algebra, SQL, query optimization, transaction management, and distributed databases.

2. "Fundamentals of Database Systems" by Ramez Elmasri and Shamkant Navathe - This is another popular textbook that covers the fundamental concepts of DBMS, including database design, normalization, SQL programming, transaction management, and concurrency control.

Sites and Web links:

1. https://www.geeksforgeeks.org/shadow-paging-dbms/
2. https://www.tutorialspoint.com/what-is-shadow-paging-in-dbms
3. https://whatisdbms.com/shadow-paging-in-dbms/

THANK YOU

(DEEMED TO BE UNIVERSITY)

Team – DBMS