AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

ONE SHOT Revision

(Crash Course)

Unit-4

Transaction Processing Concept

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -4

Transaction Processing Concept

Lecture-1

Today's Target

- Transaction and its lifecycle
- ACID Properties
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

AKTU B.Tech 5th Sem

(BCS-501- Database Management System)

Unit-IV : Transaction Processing Concept

AKTU : Syllabus

Transaction Processing Concept: Transaction System, Testing of Serializability, Serializability of Schedules, Conflict & View Serializable Schedule, Recoverability, Recovery from Transaction Failures, Log Based Recovery, Checkpoints, Deadlock Handling. Distributed Database: Distributed Data Storage, Concurrency Control, Directory System.

Transaction

- Transaction is a set of operations which are all logically related.
- Transaction is a single logical unit of work formed by a set of operations.

Operations in Transaction-

The main operations in a transaction are-

1. Read Operation
2. Write Operation

1. Read Operation-

Read operation reads the data from the database and then stores it in the buffer in main memory.

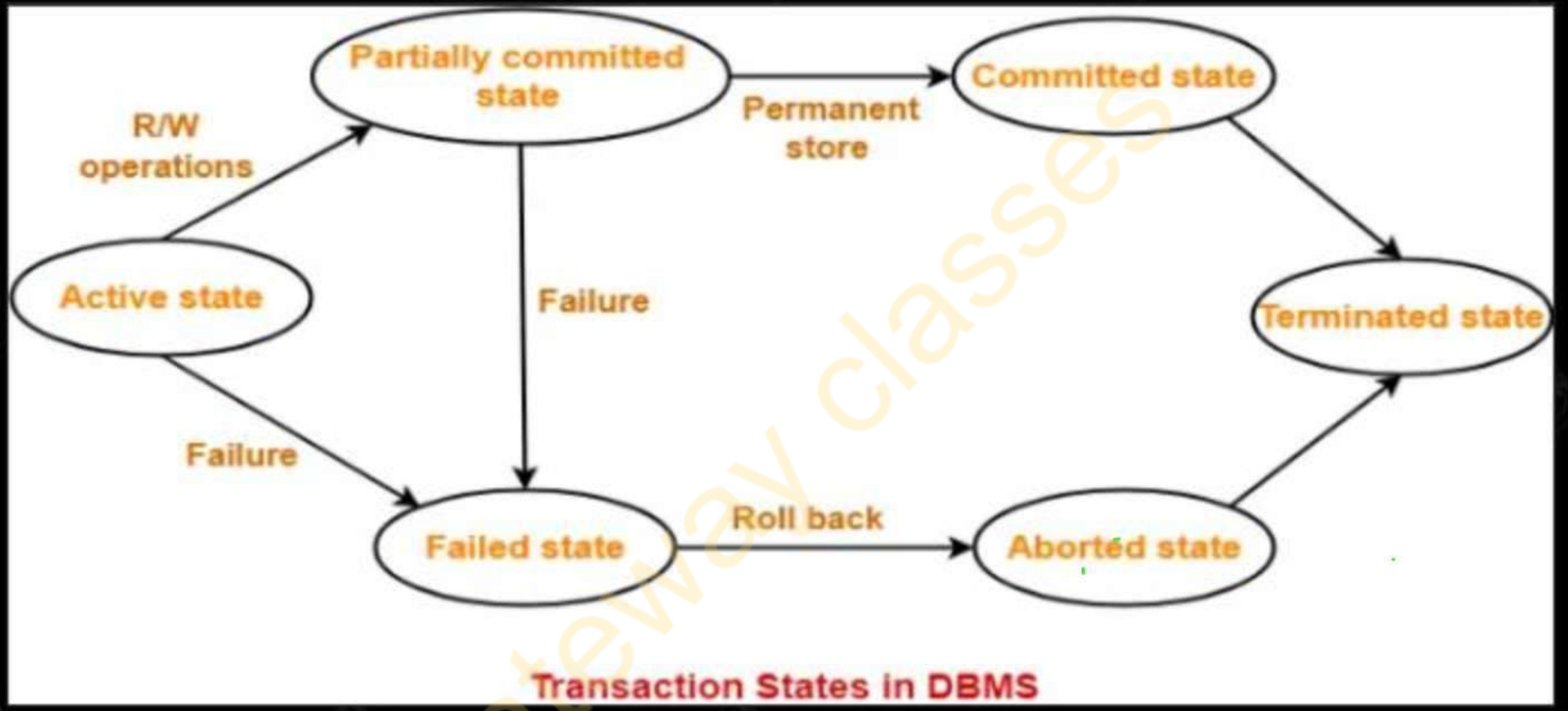
For example- Read(A) / R(A) instruction will read the value of A from the database and will store it in the buffer in main memory

2. Write Operation-

Write operation writes the updated data value back to the database from the buffer.

For example- Write(A) / W(A) will write the updated value of A from the buffer to the database.

Transaction state



Transaction States in DBMS

Transaction States-

A transaction goes through many different states throughout its life cycle.

These states are called as transaction states.

- Active state
- Partially committed state
- Committed state
- Failed state
- Aborted state
- Terminated state

1. Active State-

- This is the first state in the life cycle of a transaction.
- A transaction is called in an active state as long as its instructions are getting executed.
- All the changes made by the transaction now are stored in the buffer in main memory.

2. Partially Committed State-

- After the last instruction of transaction has executed, it enters into a partially committed state.
- After entering this state, the transaction is considered to be partially committed.
- It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.

3. Committed State-

- After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state.
- Now, the transaction is considered to be fully committed.

NOTE:

- After a transaction has entered the committed state, it is not possible to roll back the transaction.
- In other words, it is not possible to undo the changes that has been made by the transaction.
- This is because the system is updated into a new consistent state.

- The only way to undo the changes is by carrying out another transaction called as **compensating transaction** that performs the reverse operations.

4. Failed State-

- When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a failed state.

5. Aborted State-

- After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
- To undo the changes made by the transaction, it becomes necessary to roll back the transaction.
- After the transaction has rolled back completely, it enters into an aborted state

6. Terminated State-

- This is the last state in the life cycle of a transaction.
- After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end.

ACID Properties-

- It is important to ensure that the database remains consistent before and after the transaction.
- To ensure the consistency of database, certain properties are followed by all the transactions occurring in the system.
- These properties are called as ACID Properties of a transaction.

1. Atomicity-

- This property ensures that either the transaction occurs completely or it does not occur at all.
- In other words, it ensures that no transaction occurs partially.
- That is why, it is also referred to as "**All or nothing rule**".
- It is the responsibility of Transaction Control Manager to ensure atomicity of the transactions.
- **Example: Transferring $100 from Account A to Account B. If the debit from A succeeds but the credit to B fails, the entire transaction is rolled back.**

2. Consistency-

- This property ensures that integrity constraints are maintained.
- In other words, it ensures that the database remains consistent before and after the transaction.
- It is the responsibility of DBMS and application programmer to ensure consistency of the database

Example: A bank's total balance remains the same before and after a transfer, ensuring no money is lost or created

3. Isolation-

- This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
- During execution, each transaction feels as if it is getting executed alone in the system.
- A transaction does not realize that there are other transactions as well getting executed parallelly.
- Changes made by a transaction becomes visible to other transactions only after they are written in the memory.

- The resultant state of the system after executing all the transactions is same as the state that would be achieved if the transactions were executed serially one after the other.
- It is the responsibility of concurrency control manager to ensure isolation for all the transactions.

Example : Two users updating the same account balance at the same time will not interfere; one transaction will wait for the other to complete.

4. Durability-

- This property ensures that all the changes made by a transaction after its successful execution are written successfully to the disk.
- It also ensures that these changes exist permanently and are never lost even if there occurs a failure of any kind.
- It is the responsibility of recovery manager to ensure durability in the database.

Example: After transferring money, the updated balances remain saved even if the database crashes immediately after.

Q.1	What is transaction? Draw a state diagram of transaction showing it state .Explain ACID properties /transition property of transition with example	AKTU 2015-16 AKTU2016-17 AKTU 2018-19 AKTU 2019-20 AKTU 2020-21 AKTU 2022-23

lifeCycle.

Unit-3

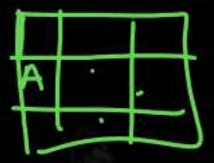
Q1 Normalisation

1NF
2NF
3NF
BCNF
4NF
5NF

Numer

then.

MV JD



* Concurrency problem

* Schedule

M.M. Imp

① lossless decomposition (7 Marks)

Key? → 2NF, 3NF

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -4

Transaction Processing Concept

Lecture-2

Today's Target

- Concurrency problem
- Schedule
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

Concurrency Problems in DBMS-

- When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems.
- Such problems are called as concurrency problems.

The concurrency problems are-

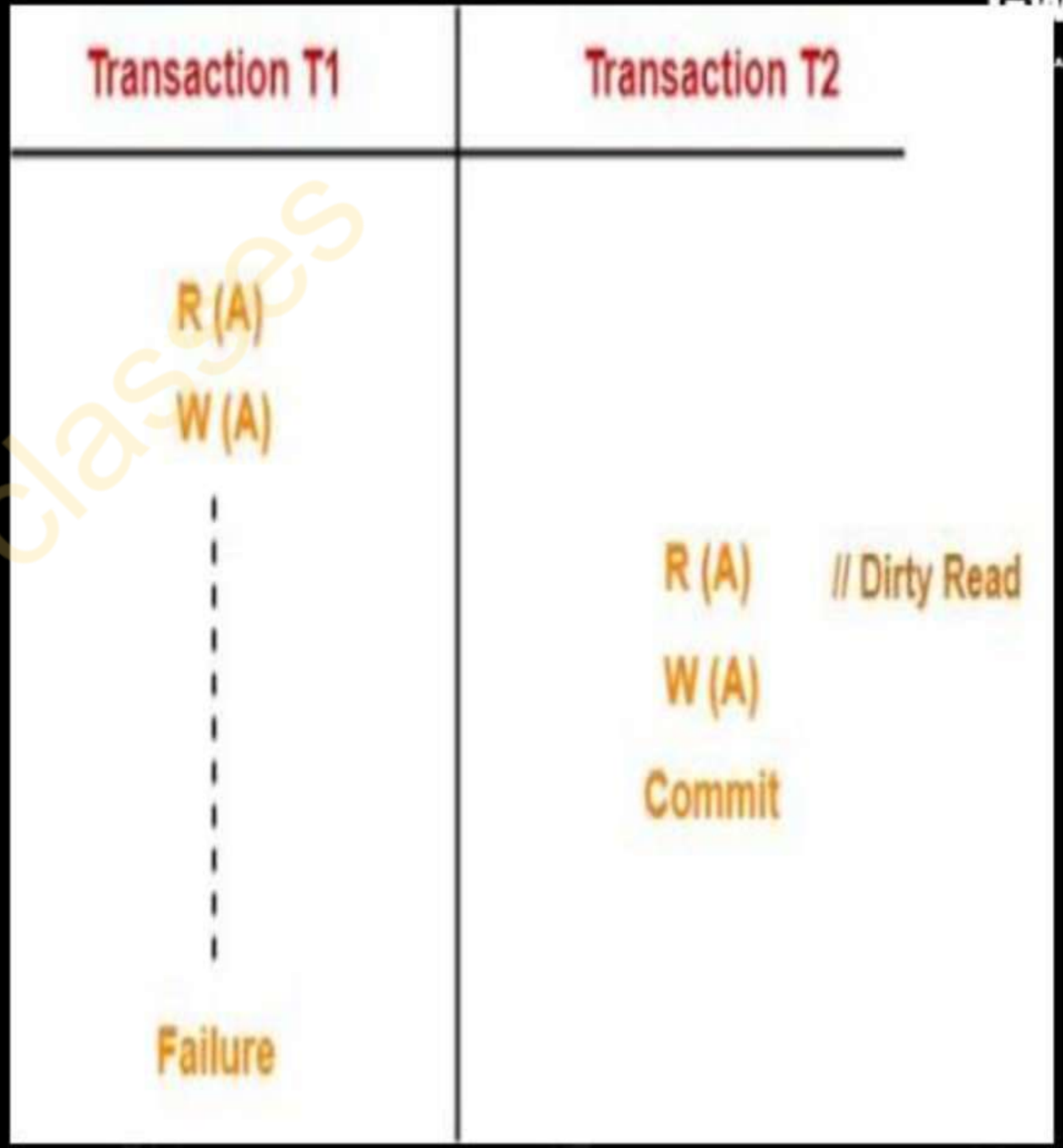
- Dirty Read Problem
- Unrepeatable Read Problem
- Lost Update Problem
- Phantom Read Problem

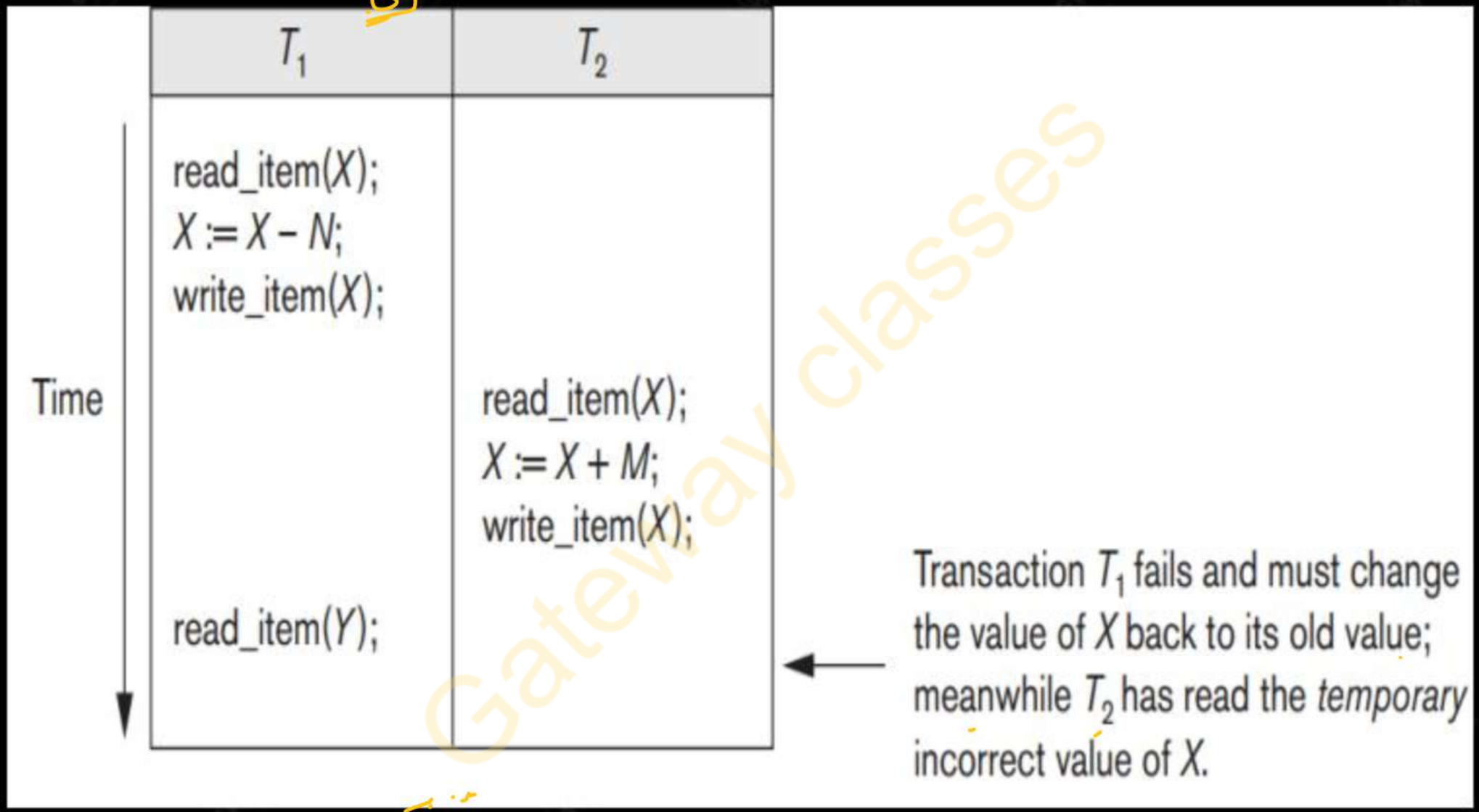
1. Dirty Read Problem(Temporary update)

- This problem occurs when one transaction updates a database item and then the transaction fails for some reason
- Meanwhile, the updated item is accessed (read) by another transaction before it is changed back (or rolled back) to its original value.
- Reading the data written by an uncommitted transaction is called as dirty read.

This read is called as dirty read because-

- There is always a chance that the uncommitted transaction might roll back later.
- Thus, uncommitted transaction might make other transactions read a value that does not even exist.
- This leads to inconsistency of the database.




T_1
T_2
read_item(X);
X := X - N;
write_item(X);
Time
read_item(X);
X := X + M;
write_item(X);
read_item(Y);
Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the temporary incorrect value of X.

2. Unrepeatable Read Problem-

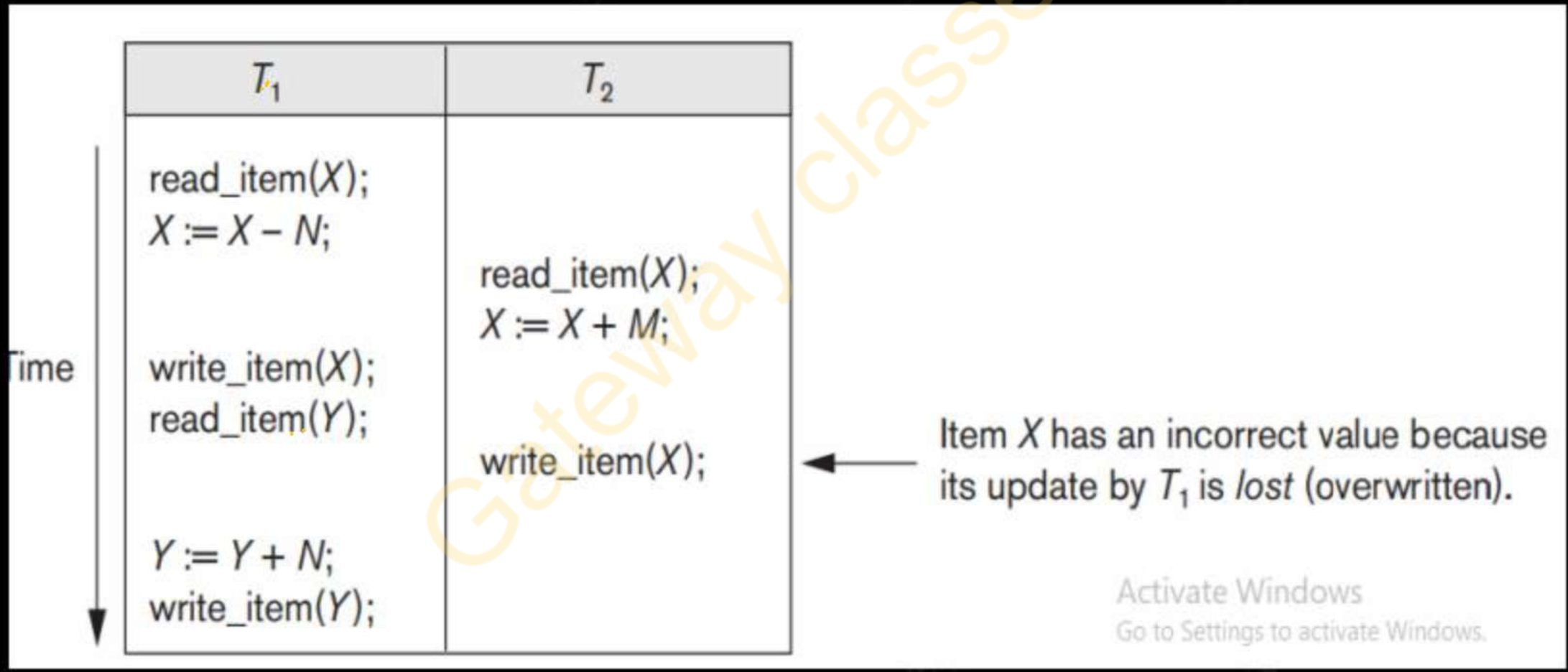
- This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.

Transaction T1	Transaction T2
R (X)	
	R (X)
W (X)	
	R (X) // Unrepeated Read

- T2 gets to read a different value of
- X in its second reading.
- T2 wonders how the value of X got changed because according to it, it is running in isolation.

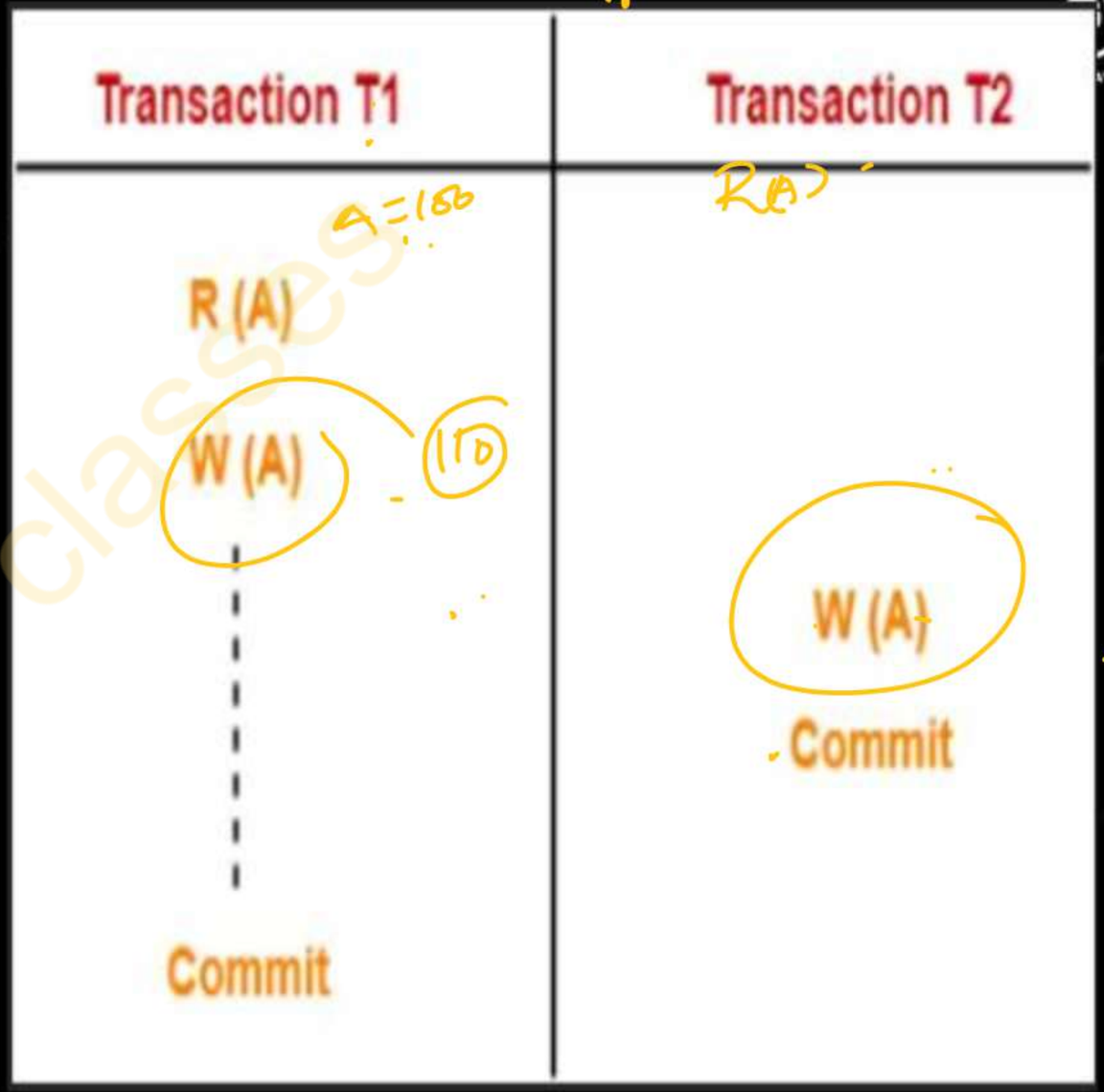
3. The Lost Update Problem.

- This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect.



This problem occurs whenever there is a **write-write conflict**.

- In write-write conflict, there are two writes one by each transaction on the same data item without any read in the middle



4. Phantom Read Problem-

This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later, it finds that the variable does not exist.

Transaction T1	Transaction T2
R (X)	
	R (X)
Delete (X)	
	Read (X)

Question: How does recovery manager ensure the atomicity and how it ensure the durability explain?

Ans. Ensuring Atomicity

Atomicity means either all parts of a transaction are executed, or none at all. The recovery manager ensures this using:

a.Undo Logging:

- Before modifying data, the original value is stored in the log.
- If a transaction fails before committing, the recovery manager uses the log to restore the original values (undo the changes).

b. Write-Ahead Logging (WAL):

- Ensures logs are written to stable storage before making changes to the database.
- This guarantees that undo information is available even if the system crashes.

c.Rollback Operations:

- If a transaction fails or is aborted, the recovery manager undoes all changes made by that transaction.

Ensuring Durability

Durability ensures that once a transaction is committed, its changes are permanently saved, even in the event of a system crash. The recovery manager ensures this using:

a. Redo Logging:

- After a transaction commits, its changes are logged to stable storage.
- If a crash occurs, the recovery manager uses the log to redo committed transactions.

Check-pointing:

- Periodically saves the database state and logs to reduce recovery time.
- After a crash, only changes after the last checkpoint need to be redone.

Stable Storage:

- Ensures committed changes are written to non-volatile storage (e.g., disk) to prevent loss due to power failure

Schedules in DBMS-

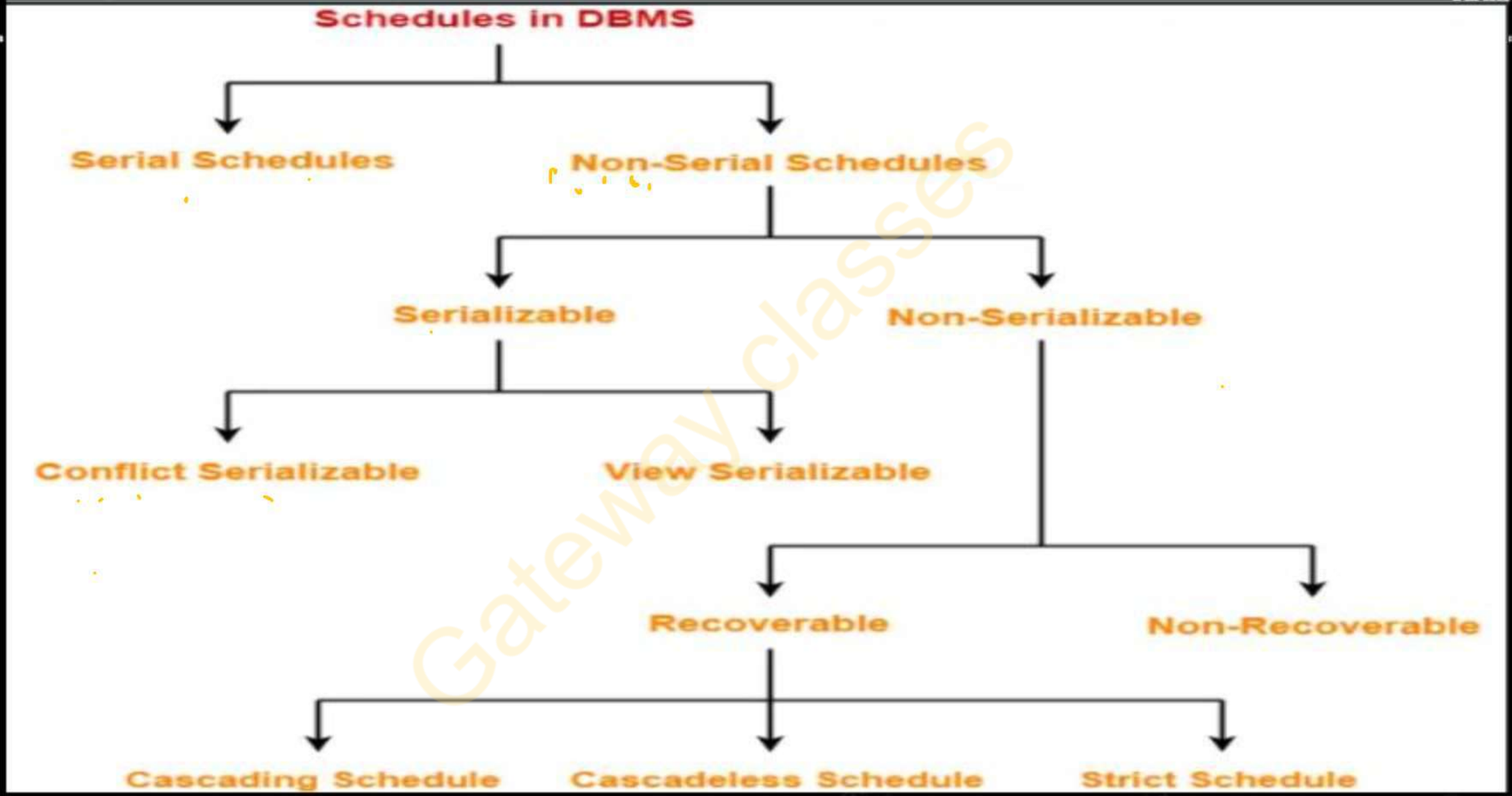
The order in which the operations of multiple transactions appear for execution is called as a schedule.

Serial Schedules-

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

Characteristics

Consistent, Recoverable, Cascade less ,Strict


Schedules in DBMS
Serial Schedules
Non-Serial Schedules
Serializable
Non-Serializable
Conflict Serializable
View Serializable
Recoverable
Non-Recoverable
Cascading Schedule
Cascadeless Schedule
Strict Schedule

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

Example of serial schedule

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

Example of serial schedule

Non-Serial Schedules-

- In non-serial schedules, Multiple transactions execute concurrently.
- Operations of all the transactions are inter leaved or mixed with each other.
- Non-serial schedules are **NOT** always-
- Consistent
- Recoverable
- Cascadeless
- Strict

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

AKTU PYQS

Q.1	Explain the concurrency problem.	AKTU 2015-16 AKTU2016-17 AKTU 2018-19 AKYI 2022-23
Q.2	What is schedule ? Explain it types in detail	AKTU 2020-21 AKTU 2018-19

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -4

Transaction Processing Concept

Lecture-3

Today's Target

- Conflict Serializability
- Numerical
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

Finding Number Of Schedules-

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

Total Number of Schedules-

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!

$3! = 3 \times 2 = 6$

Total Number of Non-Serial Schedules-

Total number of non-serial schedules

= Total number of schedules – Total number of serial schedules

T_1, T_2, T_3

$T_3 \rightarrow T_2 \rightarrow T_1$

$T_3 \rightarrow T_1 \rightarrow T_2$

$T_1 \rightarrow T_2 \rightarrow T_3$

$T_1 \rightarrow T_3 \rightarrow T_2$

$T_2 \rightarrow T_3 \rightarrow T_1$

$T_2 \rightarrow T_1 \rightarrow T_3$

Problem-

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

How many total number of schedules are possible?

How many total number of serial schedules are possible?

How many total number of non-serial schedules are possible?

$T_1 \rightarrow 2$ (Operation)
$T_2 - 3$
$T_3 - 4$

Total number of Schedule:-

$$\frac{(2+3+4)!}{2! \times 3! \times 4!} = \frac{9!}{2! \times 3! \times 4!}$$

$$\frac{9 \times 8 \times 7 \times 6 \times 5 \times 4!}{2! \times 3 \times 2 \times 1 \times 4!}$$

$$\frac{9 \times 8 \times 7 \times 5}{2 \times 1}$$

$$20 \times 63 = 1260$$ total (serial + non-serial)

Serial $= 3! = 3 \times 2 \times 1 = 6$

Non serial = Total - Serial

$1260 - 6 = 1254$

Serializability in DBMS-

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

Serializable Schedules-

- If a given non-serial schedule of 'n' transactions is equivalent to some serial schedule of 'n' transactions, then it is called as a **serializable schedule**.
- Serializable schedules behave exactly same as serial schedules.
- Consistent, **Recoverable**, **Casacadeless**, Strict
- **Types of Serializability-**

Conflict Serializability

View Serializability

Conflict Serializability-

If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a conflict serializable schedule.

Conflicting Operations-

Two operations are called as conflicting operations if all the following conditions hold true for them-

- Both the operations belong to different transactions.
- Both the operations are on the same data item.
- At least one of the two operations is a write operation.

W1 (A) and R2 (A) are called as conflicting operations.

NOTE: READ ON SAME DATA NEVER CONFLICT

$W_1(A)$ & $R_2(A)$ are conflicting

$R_1(A)$ & $R_2(A)$ are not conflicting

$R_2(A)$ & $R_1(B)$ Non conflicting

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

S1(NON SERIAL)	
T1	T2
R(A) W(A)	
	R(A) W(A)
R(B) W(B)	
	R(B) W(B)

S2(SERIAL)	
T1	T2
R(A) W(A) R(B) W(B)	
	R(A) W(A) R(B) W(B)

It is able to converting into serial schedule

Problem-01:

or Nonserial → Serial

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

S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$

S

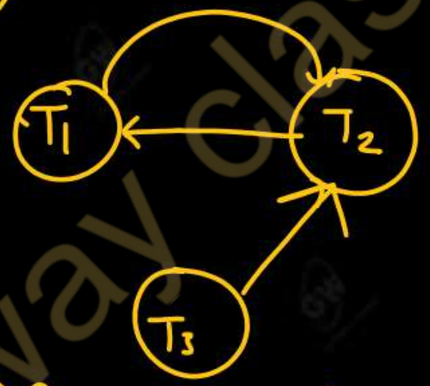
T_1	T_2	T_3
$R_1(A)$		
	$R_2(A)$	
$R_1(B)$		
	$R_2(B)$	
		$R_3(B)$
$W_1(A)$		
	$W_2(B)$	

Conflict Operation

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

$R_1(B) \rightarrow W_2(B)$ $\quad T_1 \rightarrow T_2$

$R_3(B) \rightarrow W_2(B)$ $\quad T_3 \rightarrow T_2$



Directed graph.

Cycle formed

So it Not Conflict Serializable

Problem-02:

Check whether the given schedule S is conflict serializable

T1	T2	T3	T4
	R(X)		
		W(X)	
		Commit	
W(X)			
Commit			
	W(Y)		
	R(Z)		
	Commit		
			R(X)
			R(Y)
			Commit

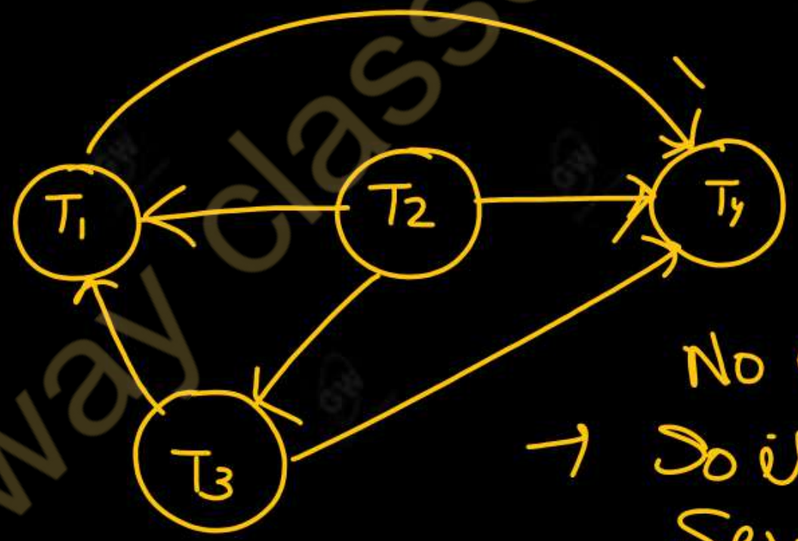
Conflict Operation

$R_2(X) \rightarrow W_3(X)$
$R_2(X) \rightarrow W_1(X)$

$W_3(X) \rightarrow W_1(X)$
$W_3(X) \rightarrow R_4(X)$
$W_1(X) \rightarrow R_4(X)$
$W_2(X) \rightarrow R_4(X)$

$R_2(X) \rightarrow W_3(X)$
$R_2(X) \rightarrow W_1(X)$
$W_3(X) \rightarrow W_1(X)$
$W_3(X) \rightarrow R_4(X)$
$W_1(X) \rightarrow R_4(X)$
$W_2(X) \rightarrow R_4(X)$

$T_2 \rightarrow T_3$
$T_2 \rightarrow T_1$
$T_3 \rightarrow T_1$
$T_3 \rightarrow T_4$
$T_1 \rightarrow T_4$
$T_2 \rightarrow T_4$



No cycle formed
→ So it is an conflict Serializable schedule
→ It is converted to Serial Schedule

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -4

Transaction Processing Concept

Lecture-4

Today's Target

- Conflict Serializability
- Numerical
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

Problem 1:

Check whether the given schedule S is conflict serializable or not. If yes, then determine all the possible serialized schedules

Or check whether a given schedule equivalent to serial schedule or not

S

T_1	T_2	T_3	T_4
	R(A)		R(A)
W(B)		R(A)	
	W(A)	R(B)	
	W(B)		

Conflicting operation

$R_4(A) \longrightarrow W_2(A)$ $T_4 \rightarrow T_2$

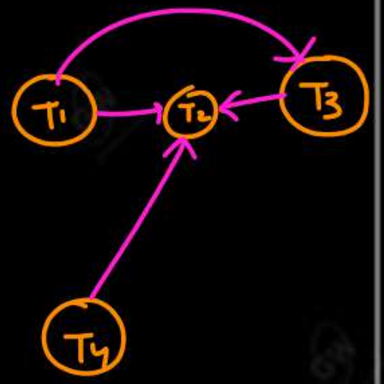
$R_3(A) \longrightarrow W_2(A)$ $T_3 \rightarrow T_2$

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

$R_3(B) \longrightarrow W_2(B)$ $T_3 \rightarrow T_2$

$W_1(B) \longrightarrow R_3(B)$ $T_1 \rightarrow T_3$

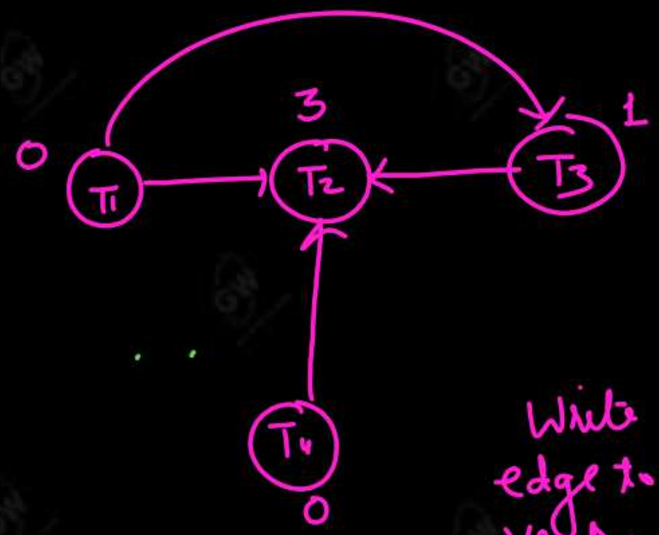
Yes, this schedul is equal to Serializable Schedul



(No cycle is formed)

→ It is conflict Serializable Schedul

→ Recoverable Schedul



With incoming edge to each vertex

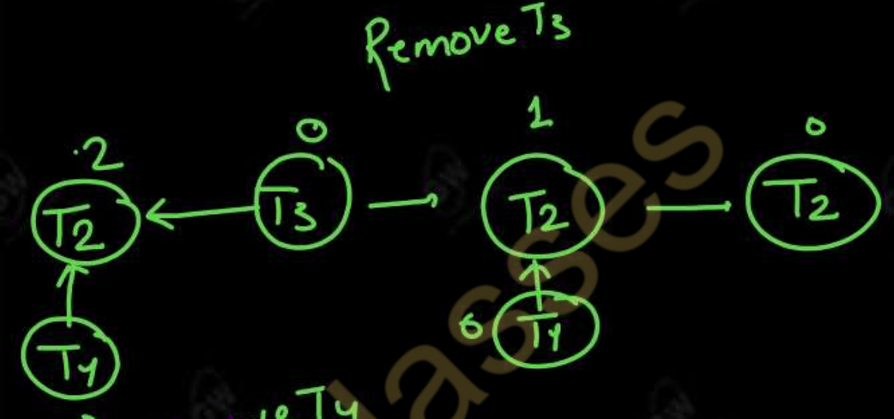
Take T_1 as starting

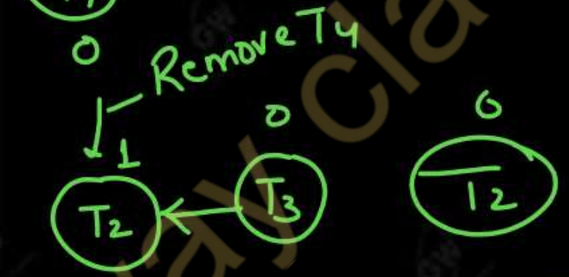
$T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$

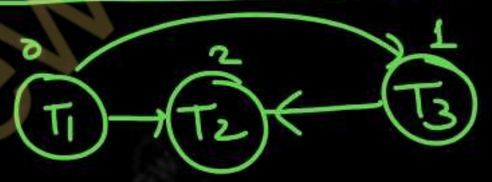
$T_1 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$

Taking T_4 as starting

$T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_3$







Problem-02:

Determine all the possible serialized schedules for the given schedule-

or Recoverable Schedu

T_1	T_2
R(A)	
A = A - 10	
	R(A)
	Temp = 0.2 x A
	W(A)
	R(B)
W(A)	
R(B)	
B = B + 10	
W(B)	
.	B = B + Temp
.	W(B)

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

Conflicting operation

$R_1(A) \rightarrow W_2(A)$ $T_1 \rightarrow T_2$

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

$W_2(A) \rightarrow W_1(A)$ $T_2 \rightarrow T_1$

$R_2(B) \rightarrow W_1(B)$ $T_2 \rightarrow T_1$

$R_1(B) \rightarrow W_2(B)$ $T_1 \rightarrow T_2$

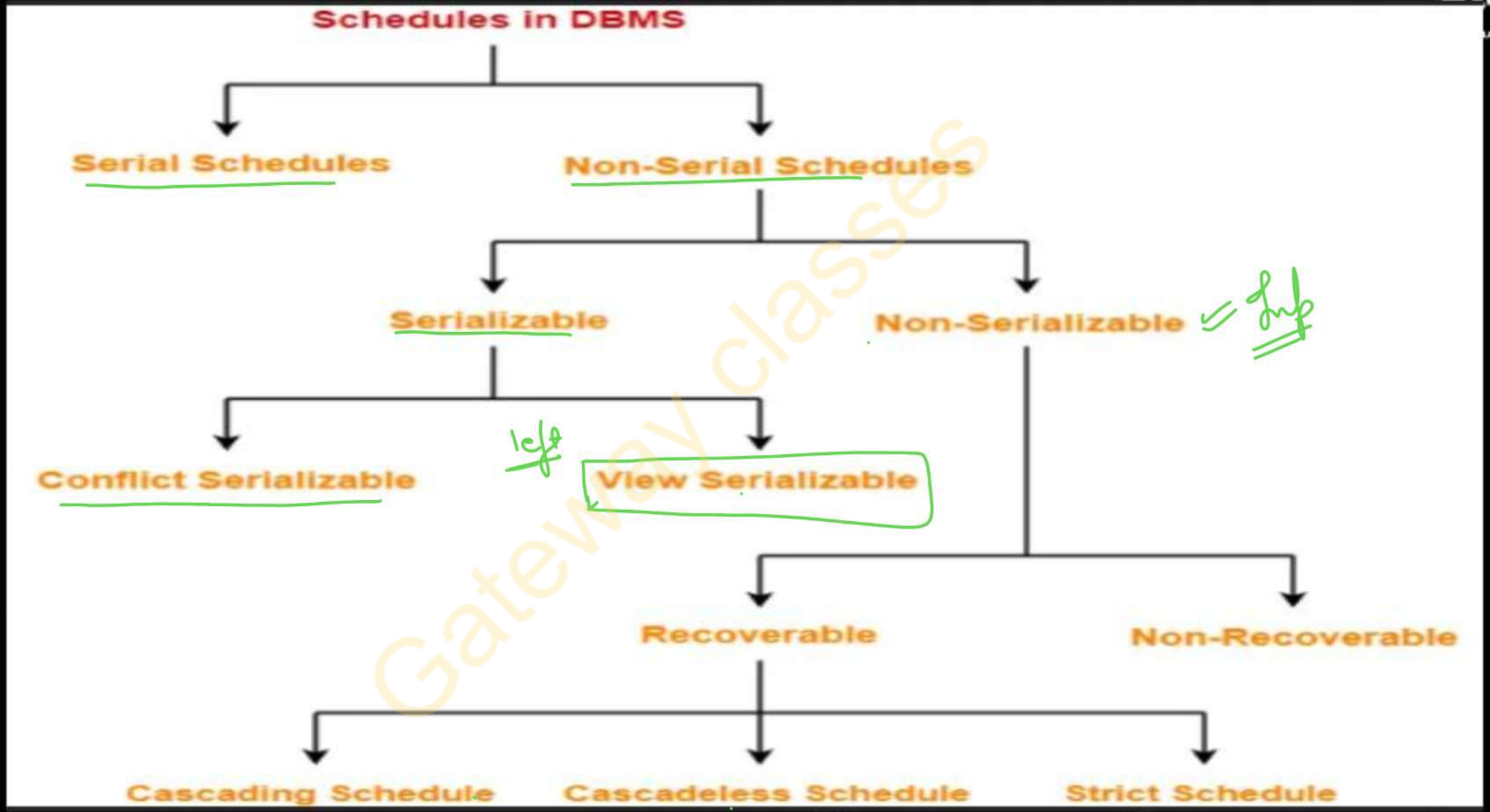
$W_1(B) \rightarrow W_2(B)$ $T_1 \rightarrow T_2$

Short
T_1 T_2
$R(A) \rightarrow W(A)$
$W(A) \rightarrow R(A) / W(A)$

Cycl is formed
So it is not Conflict
Serializable So

No Serial Schedule possible.

Recoverable:
May or may not be


Schedules in DBMS
Serial Schedules
Non-Serial Schedules
Serializable
Non-Serializable
Conflict Serializable
View Serializable
Recoverable
Non-Recoverable
Cascading Schedule
Cascadeless Schedule
Strict Schedule

Non-Serializable Schedules-

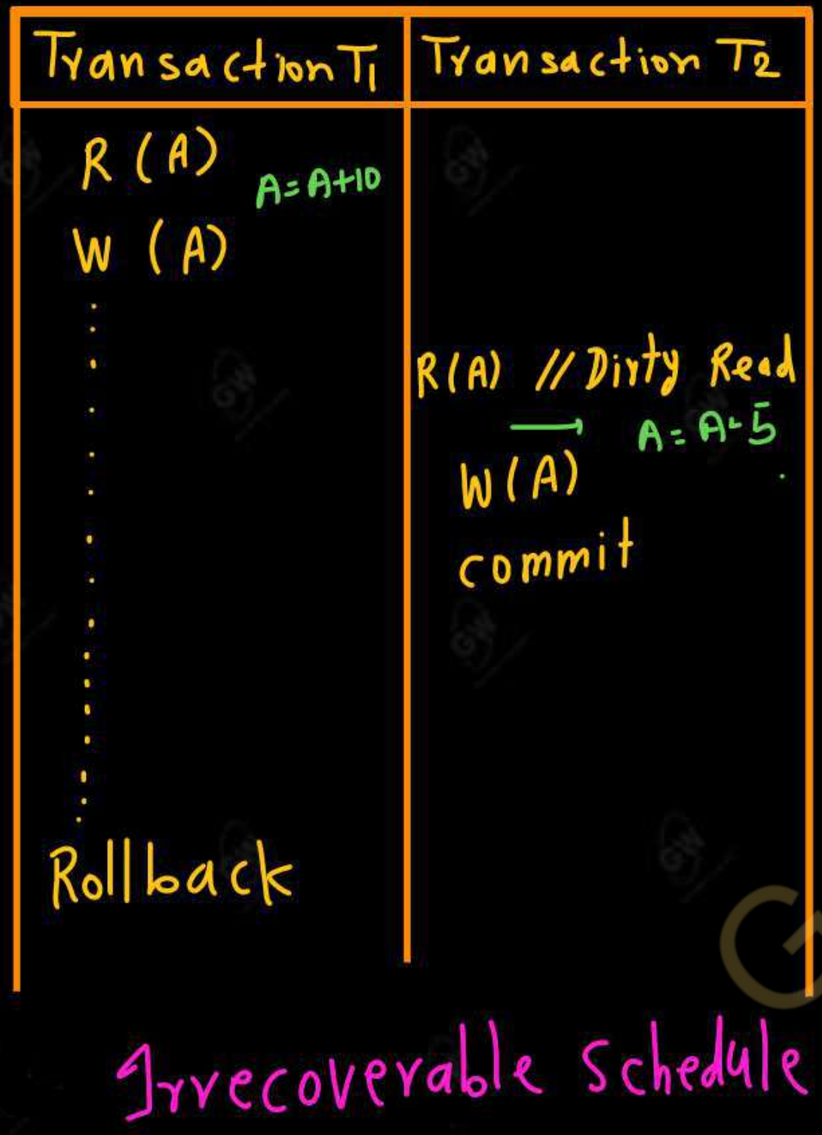
- A non-serial schedule which is not serializable is called as a non-serializable schedule.
- A non-serializable schedule is not guaranteed to produce the same effect as produced by some serial schedule on any consistent database.

Characteristics-

- may or may not be consistent
- may or may not be recoverable

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 Schedule

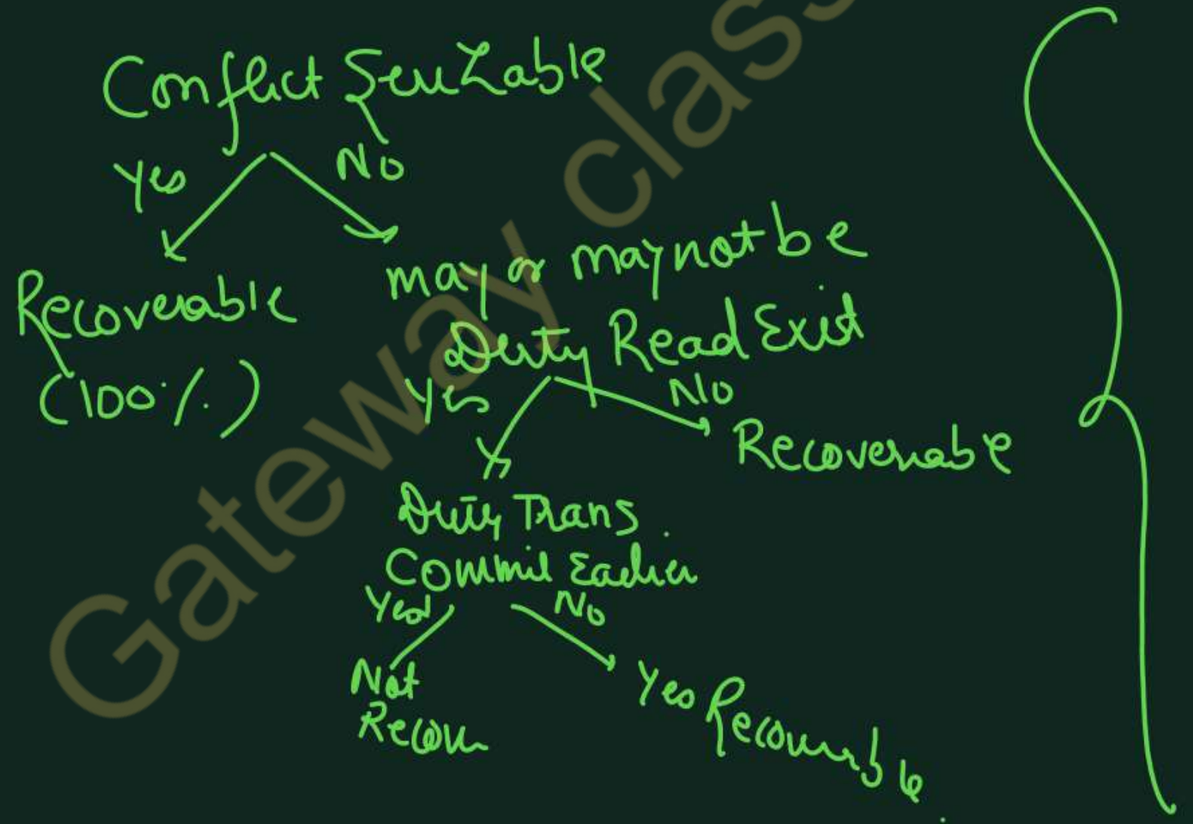
Recoverable Schedules-

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.

Note.

- 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 Schedule



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

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-

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.

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.

Types of Recoverable Schedules-

A recoverable schedule may be any one of these kinds-

1. Cascading Schedule
2. Cascade less Schedule
3. Strict Schedule

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.

NOTE-

If the transactions T2, T3 and T4 would have committed before the failure of transaction T1, then the schedule would have been irrecoverable.

Cascading Recoverable Schedule

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.

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

T_1	T_2	T_3
R(A)		
W(A)		
commit		
	R(A)	
	W(A)	
	commit	
		R(A)
		W(A)
		commit

Cascadless Schedule

Note:

- Cascadeless schedule allows only committed read operations.
- However, it allows uncommitted write operations.

T_1	T_2
R(A)	
W(A)	
	W(A) // Uncommitted Write
commit	

Cascadeless Schedule

Strict Schedule

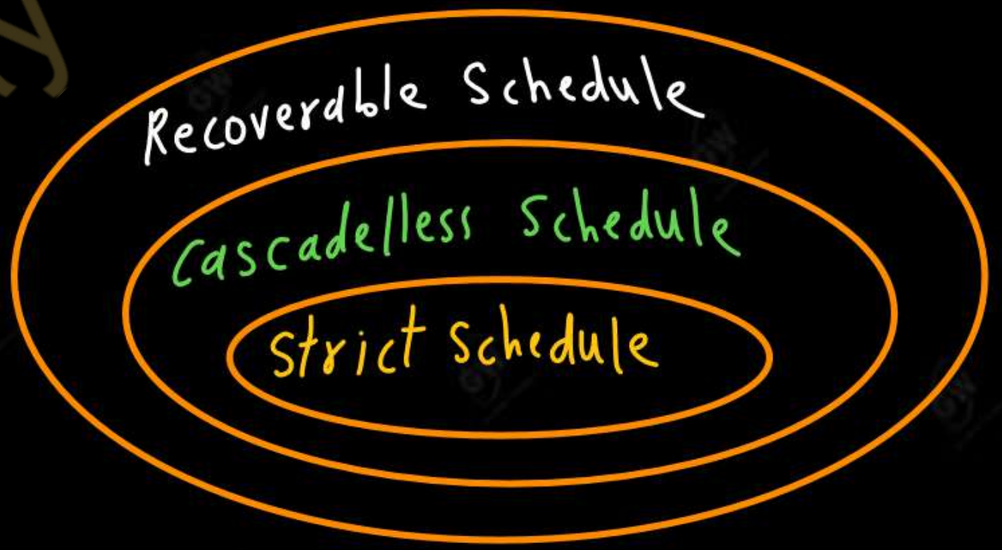
- If in a schedule, a transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a **Strict Schedule**.
- Strict schedule allows only committed read and write operations.
- Clearly, strict schedule implements more restrictions than cascadeless schedule.

T_1	T_2
W(A)	
Commit / Rollback	
	R(A) / W(A)

strict schedule

Remember-

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



AKTU PYQS

Q.1	Schedules S1 and S2given below. State whether each schedule is serializable or not. If a schedule is serializable, write down the equivalent serial schedule(s). T1: r1 (X); r1 (Z); w1 (X); T2: r2 (Z); r2 (Y); w2 (Z); w2 (Y); T3: r3 (X); r3 (Y); w3 (Y); S1: r1 (X); r2 (Z); r1 (Z); r3 (X); r3 (Y); w1 (X); w3 (Y); r2 (Y); w2 (Z); w2 (Y); S2: r1 (X); r2 (Z); r3 (X); r1 (Z); r2 (Y); r3 (Y); w1 (X); w2 (Z); w3 (Y); w2 (Y);	AKTU 2022-23/AKTU2023-24

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -4

Transaction Processing Concept

Lecture-5

Today's Target

- Numerical on recoverable(types) or irrecoverable
- Log
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

Problem-01:

Consider the transactions T1, T2, and T3 and the schedules S1 and S2 given below.

T1: r1(X); r1(Z); w1(X); w1(Z)

T2: r2(Y); r2(Z); w2(Z)

T3: r3(Y); r3(X); w3(Y)

S1: r1(X); r3(Y); r3(X); r2(Y); r2(Z); w3(Y); w2(Z); r1(Z); w1(X); w1(Z)

S2: r1(X); r3(Y); r2(Y); r3(X); r1(Z); r2(Z); w3(Y); w1(X); w2(Z); w1(Z)

S1

T_1	T_2	T_3
$r_1(X)$		
		$r_3(Y)$
		$r_3(X)$
	$r_2(Y)$	
	$r_2(Z)$	
		$W_3(Y)$
	$W_2(Z)$	
$r_1(Z)$		
$W_1(X)$		
$W_1(Z)$		

Conflicting operation

$r_3(X) \rightarrow W_1(X)$

$r_2(Y) \rightarrow W_3(Y)$

$r_2(Z) \rightarrow W_1(Z)$

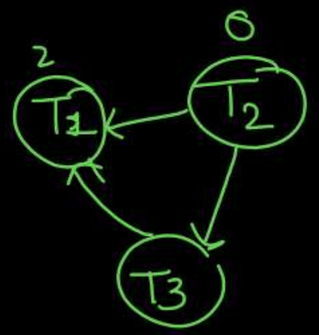
$W_2(Z) \rightarrow r_1(Z)$

$T_3 \rightarrow T_1$

$T_2 \rightarrow T_3$

$T_2 \rightarrow T_1$

$T_2 \rightarrow T_1$



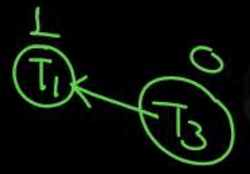
$T_2 \rightarrow T_3 \rightarrow T_1$

Serial Schedule possible

NO cycle formed

conflict Serializable

Recoverable.



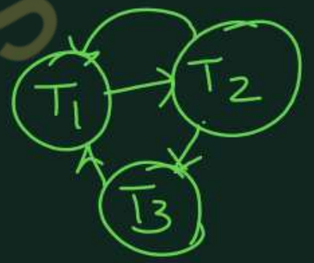
S2

T_1	T_2	T_3
$r_1(X)$		$r_3(Y)$
	$r_2(Y)$	
		$r_3(X)$
$r_1(Z)$		
	$r_2(Z)$	
		$W_3(Y)$
$W_1(X)$		
	$W_2(Z)$	
$W_1(Z)$		

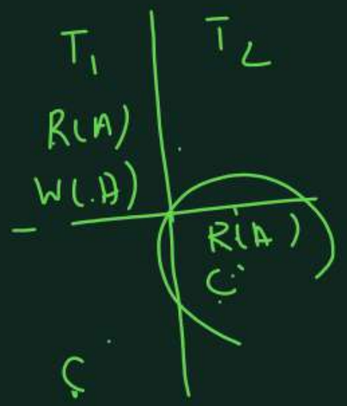
Conflicting operation

$r_2(Y) \rightarrow W_3(Y)$
$r_3(X) \rightarrow W_1(X)$
$r_1(Z) \rightarrow W_2(Z)$
$r_2(Z) \rightarrow W_1(Z)$
$W_2(Z) \rightarrow W_1(Z)$

$T_2 \rightarrow T_3$
$T_3 \rightarrow T_1$
$T_1 \rightarrow T_2$
$T_2 \rightarrow T_1$
$T_2 \rightarrow T_2$



Loop found
No conflict Serial
No Serial Schedule possible
may or may not be Recoverable



Schedule Conflict Serializable

Yes → Recoverable

NO → Check Dirty Read.

NO → Recoverable

Yes → Commit (from which we read)

before → Recoverable

After → Irrecoverable

(AKTU)

Problem 2: Consider schedules S1, S2, and S3 below. Determine whether each schedule is strict, cascade less, recoverable, or non recoverable. (Determine the strictest recoverability condition that each schedule satisfies.)

1.S1: r1 (X); r2 (Z); r1 (Z); r3 (X); r3 (Y); w1 (X); c1; w3 (Y); c3; r2 (Y); w2 (Z); w2 (Y); c2;

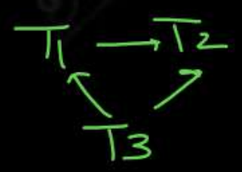
S1

T_1	T_2	T_3
$r_1(X)$	$r_2(Z)$	
$r_1(Z)$		$r_3(X)$
		$r_3(Y)$
$W_1(X)$		
C_1		
		$W_3(Y)$
		C
	$r_2(Y)$	
	$W_2(Z)$	
	$W_2(Y)$	
	C_2	

① X
② Y
③ Z

Conflicting operations

$r_1(Z) \rightarrow W_2(Z)$ $\quad T_1 \rightarrow T_2$
$r_3(X) \rightarrow W_1(X)$ $\quad T_3 \rightarrow T_1$
$r_3(Y) \rightarrow W_2(Y)$ $\quad T_3 \rightarrow T_2$
$W_3(Y) \rightarrow W_2(Y)$ $\quad T_3 \rightarrow T_2$
$W_3(Y) \rightarrow r_2(Y)$ $\quad T_3 \rightarrow T_2$



Conflict Serializable
Recoverable Schedule
Not a strict Schedule.
Not a cascadless Schedule.

2. S2: r1 (X); r2 (Z); r1 (Z); r3 (X); r3 (Y); w1 (X); w3 (Y); r2 (Y); w2 (Z); w2 (Y); c1;c2; c3;

T_1	T_2	T_3
$r_1(X)$	$r_2(Z)$	
$r_1(Z)$		$r_3(X)$
		$r_3(Y)$
$W_1(X)$		
		$W_3(Y)$
	$r_2(Y)$	
	$W_2(Z)$	
	$W_2(Y)$	
C1		
	C2	
		C3

Conflicting operations

$r_1(Z) \rightarrow W_2(Z)$ $\quad T_1 \rightarrow T_2$

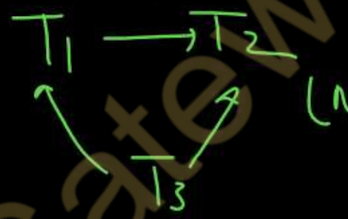
$r_3(X) \rightarrow W_1(X)$ $\quad T_3 \rightarrow T_1$

$r_3(X) \leftarrow r_2(Y)$ $\quad T_3 \rightarrow T_2$

$r_3(Y) \rightarrow W_2(Y)$ $\quad T_3 \rightarrow T_2$

$W_3(Y) \rightarrow r_2(Y)$ $\quad T_3 \rightarrow T_2$

$W_3(Y) \rightarrow W_2(Y)$



(No cycle)

Conflict Serializable

Recoverable ✓

Not Strict

Cascadeless Not

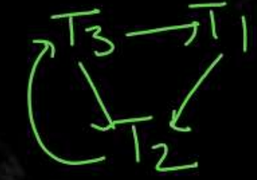
3. S3: r1 (X); r2 (Z); r3 (X); r1 (Z); r2 (Y); r3 (Y); w1 (X); c1; w2 (Z); w3 (Y); w2 (Y);c3; c2;

T_1	T_2	T_3
$r_1(x)$	$r_2(z)$	$r_3(x)$
$r_1(z)$	$r_2(y)$	$r_3(y)$
$w_1(x)$		
c_1		
	$w_2(z)$	$w_3(y)$
	$w_2(y)$	
	c_2	c_3

conflicting operation

- $r_3(x) \rightarrow w_1(x)$ ✓
- $r_1(z) \rightarrow w_2(z)$ ✓
- $r_3(y) \rightarrow w_2(y)$
- $w_3(y) \rightarrow w_2(y)$
- $r_2(y) \rightarrow w_3(y)$ ✓

$T_3 \rightarrow T_1$
$T_1 \rightarrow T_2$
$T_3 \rightarrow T_2$
$T_3 \rightarrow T_2$
$T_2 \rightarrow T_3$



Not
Conflict Serializable ✗

Recoverable

dirty Read? irrecoverable.

Log based Recovery in DBMS

- The atomicity property of DBMS states that either all the operations of transactions must be performed or none. The modifications done by an aborted transaction should not be visible to the database and the modifications done by the committed transaction should be visible.
- To achieve our goal of atomicity, the user must first output stable storage information describing the modifications, without modifying the database itself.
- This information can help us ensure that all modifications performed by committed transactions are reflected in the database. This information can also help us ensure that no modifications made by an aborted transaction persist in the database.

Log and log records

The log is a sequence of log records, recording all the updated activities in the database. In stable storage, logs for each transaction are maintained. Any operation which is performed on the database is recorded on the log.

Prior to performing any modification to the database, an updated log record is created to reflect that modification.

An update log record represented as: <Ti, Xj, V1, V2> has these fields:

- **Transaction identifier:** Unique Identifier of the transaction that performed the write operation.
- **Data item:** Unique identifier of the data item written.
- **Old value:** Value of data item prior to write.
- **New value:** Value of data item after write operation.

Other types of log records are:

<Ti start> : It contains information about when a transaction Ti starts.

<Ti commit> : It contains information about when a transaction Ti commits.

<Ti abort> : It contains information about when a transaction Ti aborts.

Undo and Redo Operations

Because all database modifications must be preceded by the creation of a log record, the system has available both the old value prior to the modification of the data item and new value that is to be written for data item. This allows system to perform redo and undo operations as appropriate:

Undo: using a log record sets the data item specified in log record to old value.

Redo: using a log record sets the data item specified in log record to new value.

The database can be modified using two approaches –

1. **Deferred Modification Technique:** If the transaction does not modify the database until it has partially committed, it is said to use deferred modification technique.

2. **Immediate Modification Technique:** If database modification occur while the transaction is still active, it is said to use immediate modification technique

AKTU PYQS

Q.1	**Consider the three transactions T1, T2, and T3, and the schedules S1 and S2given below. State whether each schedule is serializable or not. If a schedule is serializable, write down the equivalent serial schedule(s). T1: r1 (X); r1 (Z); w1 (X); T2: r2 (Z); r2 (Y); w2 (Z); w2 (Y); T3: r3 (X); r3 (Y); w3 (Y); S1: r1 (X); r2 (Z); r1 (Z); r3 (X); r3 (Y); w1 (X); w3 (Y); r2 (Y); w2 (Z); w2 (Y); S2: r1 (X); r2 (Z); r3 (X); r1 (Z); r2 (Y); r3 (Y); w1 (X); w2 (Z); w3 (Y); w2**	AKTU 20222-23 AKTU 2023-24
Q.2	What is log ? How log is maintained?	AKTU 2013-14 AKT7U 2014-15

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -4

Transaction Processing Concept

Lecture-6

Today's Target

- Shadow paging
- Check pointing
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

Deferred Database modification	Immediate Database modification
In a deferred update, the changes are not applied immediately to the database.	In an immediate update, the changes are applied directly to the database.
The log file contains all the changes that are to be applied to the database.	The log file contains both old as well as new values.
In this method once rollback is done all the records of log file are discarded and no changes are applied to the database.	In this method once rollback is done the old values are restored to the database using the records of the log file.
Concepts of buffering and caching are used in deferred update method.	Concept of shadow paging is used in immediate update method.

Deferred Database modification	Immediate Database modification
The major disadvantage of this method is that it requires a lot of time for recovery in case of system failure.	The major disadvantage of this method is that there are frequent I/O operations while the transaction is active.
In this method of recovery, firstly the changes carried out by a transaction on the data are done in the log file and then applied to the database on commit. Here, the maintained record gets discarded on rollback and thus, not applied to the database.	In this method of recovery, the database gets directly updated after the changes made by the transaction and the log file keeps the old and new values. In the case of rollback, these records are used to restore old values.

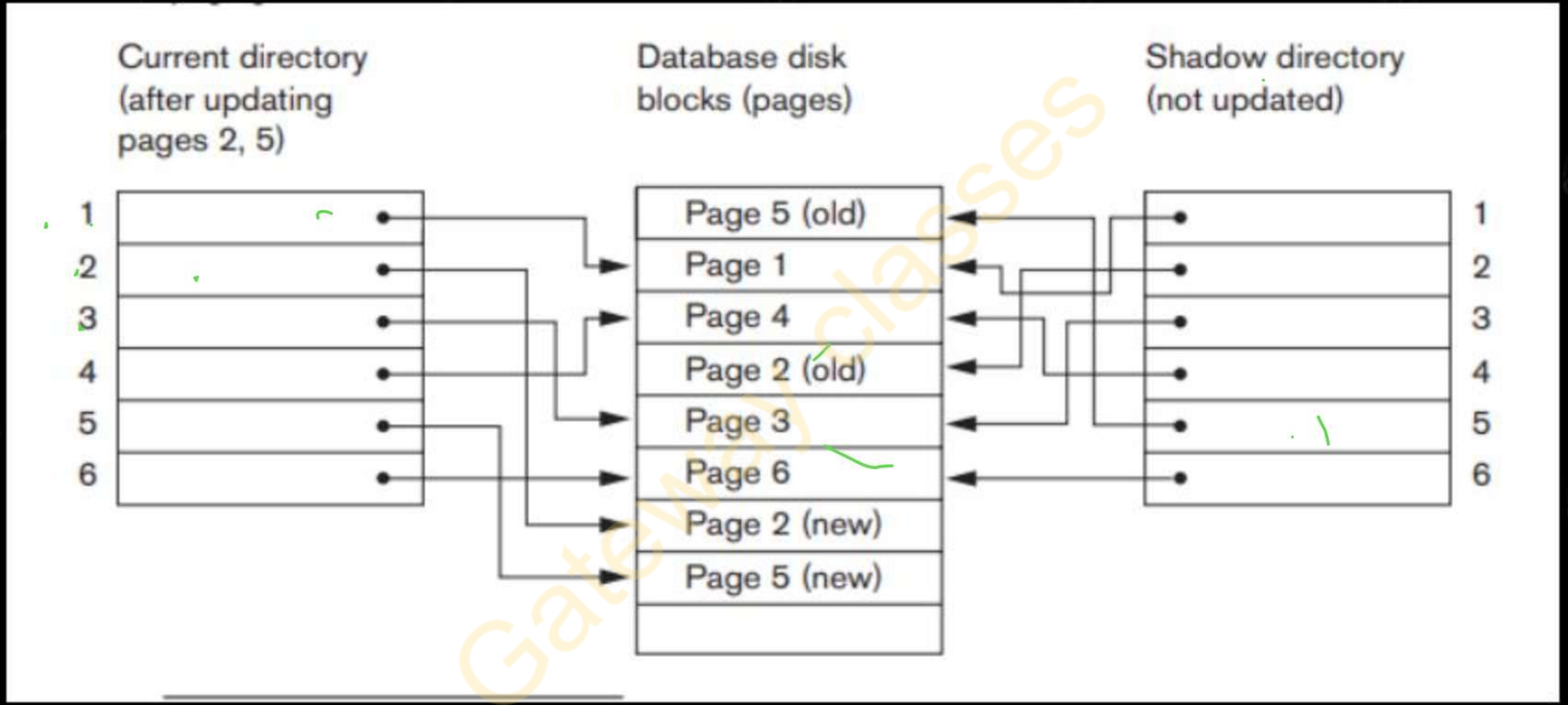
Features	Deferred Update	Immediate Update
Update timing	Updates occur after instruction execution	Updates occur during instruction execution
Processor speed	May be faster: update occurs after instruction execution, allowing for multiple updates to be performed at once	May be slower: update occurs during instruction execution, potentially causing the processor to stall
Complexity	More complex: requires additional instructions or mechanisms to handle updates after instruction execution	Less complex: updates occur immediately during instruction execution

Features	Deferred Update	Immediate Update
Consistency	May result in temporary inconsistency between data in registers and memory	Data in registers and memory are always consistent
Flexibility	May be more flexible: allows for more complex data manipulations and algorithms	Less flexible: immediate updates can limit the range of data manipulations and algorithms that can be performed

Shadow paging

- Shadow paging considers the database to be made up of a number of fixed size disk pages (or disk blocks)—say, n—for recovery purposes.
- A directory with n entries is constructed, where the ith entry points to the ith database page on disk. The directory is kept in main memory if it is not too large, and all references—reads or writes—to database pages on disk go through it.

When a transaction begins executing, the current directory—whose entries point to the most recent or current database pages on disk—is copied into a shadow directory. The shadow directory is then saved on disk while the current directory is used by the transaction


Current directory
(after updating
pages 2, 5)
Database disk
blocks (pages)
Shadow directory
(not updated)
1
2
3
4
5
6
Page 5 (old)
Page 1
Page 4
Page 2 (old)
Page 3
Page 6
Page 2 (new)
Page 5 (new)
1
2
3
4
5
6

During transaction execution, the shadow directory is never modified. When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten.

- Instead, the new page is written elsewhere—on some previously unused disk block. The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block
- For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory and the new version by the current directory.
- To recover from a failure during transaction execution, it is sufficient to free the modified database pages and to discard the current directory. The state of the database before transaction execution is available through the shadow directory, and that state is recovered by reinstating the shadow directory.

The database thus is returned to its state prior to the transaction that was executing when the crash occurred, and any modified pages are discarded.

- Committing a transaction corresponds to discarding the previous shadow directory. Since recovery involves neither undoing nor redoing data items, this technique can be categorized as a NOUNDO/NO-REDO technique for recovery

Disadvantages:

1.In shadow paging, when a page is updated, it is saved at a new location on the disk. This breaks the arrangement of related pages stored close to each other. As a result:

- Access slows down because related pages are now scattered.
- Fixing this needs extra effort and complicated methods to keep related pages together.

If the directory is large, saving the shadow directory to the disk every time a transaction commits takes a lot of time and resources. This causes delays and extra work, slowing down the system.

Checkpoints in DBMS

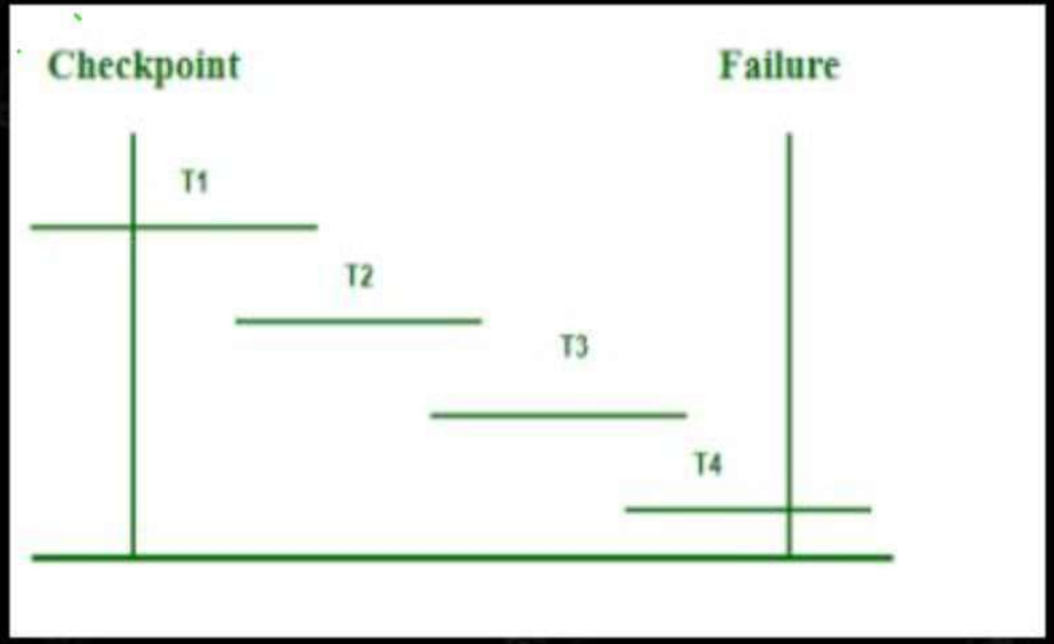
- The Checkpoint is used to declare a point before which the DBMS was in a consistent state, and all transactions were committed. During transaction execution, such checkpoints are traced.
- After execution, transaction log files will be created. Upon reaching the savepoint/checkpoint, the log file is destroyed by saving its update to the database.
- Then a new log is created with upcoming execution operations of the transaction and it will be updated until the next checkpoint and the process continues

Why do We Need Checkpoints?

- Whenever transaction logs are created in a real-time environment, it eats up lots of storage space.
- Also keeping track of every update and its maintenance may increase the physical space of the system.
- Eventually, the transaction log file may not be handled as the size keeps growing.
- This can be addressed with checkpoints
- . The methodology utilized for removing all previous transaction logs and storing them in permanent storage is called a Checkpoint

Steps to Use Checkpoints in the Database

- Write the begin_checkpoint record into a log.
- Collect checkpoint data in stable storage.
- Write the end_checkpoint record into a log.



Time

Transaction 1 (T1)	Transaction 2 (T2)	Transaction 3 (T3)	Transaction 4 (T4)
1 START			
2	START		
3	COMMIT		
4		START	
5		COMMIT	
6			START
7			FAILURE

Check

T4
1 — A = 10
2 A = 4
3 B = 6
B = 2
Fail

- The recovery system reads the logs backward from the end to the last checkpoint i.e. from T4 to T1.
- It will keep track of two lists – Undo and Redo.
- Whenever there is a log with instructions <Tn, start>and <Tn, commit> or only <Tn, commit> then it will put that transaction in Redo List. T2 and T3 contain <Tn, Start> and <Tn, Commit> whereas T1 will have only <Tn, Commit>. Here, T1, T2, and T3 are in the redo list.
- Whenever a log record with no instruction
- of commit or abort is found, that transaction is put to Undo List <Here, T4 has <Tn, Start> but no <Tn, commit> as it is an ongoing transaction. T4 will be put on the undo list.
- All the transactions in the redo list are deleted with their previous logs and then redone before saving their logs.
- All the transactions in the undo list are undone and their logs are deleted.

Types of checkpoint

1. FUZZY CHECK POINT

1. Identify Active Transactions:

During a fuzzy checkpoint, only active transactions (those still in progress at the time of the checkpoint) are logged. These transactions are noted in the log and their changes are not committed to disk yet.

F

2. Post-Failure Recovery:

- After a system failure, the recovery manager identifies the last fuzzy checkpoint and checks the transactions that were active during and after that point.
- Committed transactions before the checkpoint are already written to disk, so they don't need to be redone.

3. Undo Uncommitted Transactions:

The system undos any uncommitted transactions that were active at the time of the checkpoint (those that were not completed before the crash).

4. Redo Committed Transactions After the Checkpoint:

For committed transactions that started after the checkpoint, the system will redo them to ensure their changes are applied to the database.

5. Clean Up:

Once the undo and redo operations are complete, the logs of active transactions are deleted.

consistent recovery

1. Identify the Last Consistent Checkpoint

- The recovery manager first identifies the last consistent checkpoint in the logs. This checkpoint represents a stable and fully committed state of the database.

2.Undo Uncommitted Transactions:

- All transactions that were active (not committed) at the time of the failure, after the last checkpoint, are identified.
- The system undos these uncommitted transactions to ensure any partial changes made by them are discarded.

3. Redo Committed Transactions:

Transactions that were committed before the failure, but whose changes may not have been written to disk, are redone.

The system re-applies the changes of these committed transactions to ensure they are properly stored in the database.

4. Finalize and Clean Up:

After undoing uncommitted transactions and redoing committed ones, the recovery process is complete. The logs are cleared as the database has been restored to the state at the last consistent checkpoint

TYPES OF CHECKPOINTING

Feature	Consistent Check pointing	Fuzzy Check pointing
Scope of Logging	Logs all committed transactions up to the checkpoint and ensures no in-progress transactions are included.	Logs only active transactions at the time of the checkpoint. Committed transactions before the checkpoint are not logged.
Checkpoint Type	Creates a consistent snapshot of the entire database state	Logs the state of active transactions, without capturing the full database snapshot.

TYPES OF CHECKPOINTING

Feature	Consistent Check pointing	Fuzzy Check pointing
Performance Overhead	Higher overhead as it saves the full database state, leading to slower check pointing.	Lower overhead as only active transactions are logged, making checkpointing faster..
Frequency	Less frequent due to higher cost of taking full snapshots.	More frequent since it involves logging only active transactions, which is quicker.

TYPES OF CHECKPOINTING

Feature	Consistent Check pointing	Fuzzy Check pointing
Data Integrity	Guarantees a fully consistent database state at the time of the checkpoint..	May leave the database in an inconsistent state during checkpoint, but still provides reliable recovery for active transactions.
Storage Requirements	Requires more storage for the full database snapshot.	Requires less storage as only active transactions are logged.

AKTU PYQS		
Q.1	What is checkpoint ? Explain different types of check pointing?	AKTU 2015-16 AKTU 2017-18
Q.2	Explain the shadow paging recovery.	AKTU 2016-17

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -4

Transaction Processing Concept

Lecture-7

Today's Target

- Deadlock
- Distributed database
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

Deadlock in DBMS

In database management systems (DBMS) a deadlock occurs when two or more transactions are unable to the proceed because each transaction is waiting for the other to the release locks on resources. This situation creates a cycle of the dependencies where no transaction can continue leading to the standstill in the system. The Deadlocks can severely impact the performance and reliability of a DBMS making it crucial to the understand and manage them effectively.

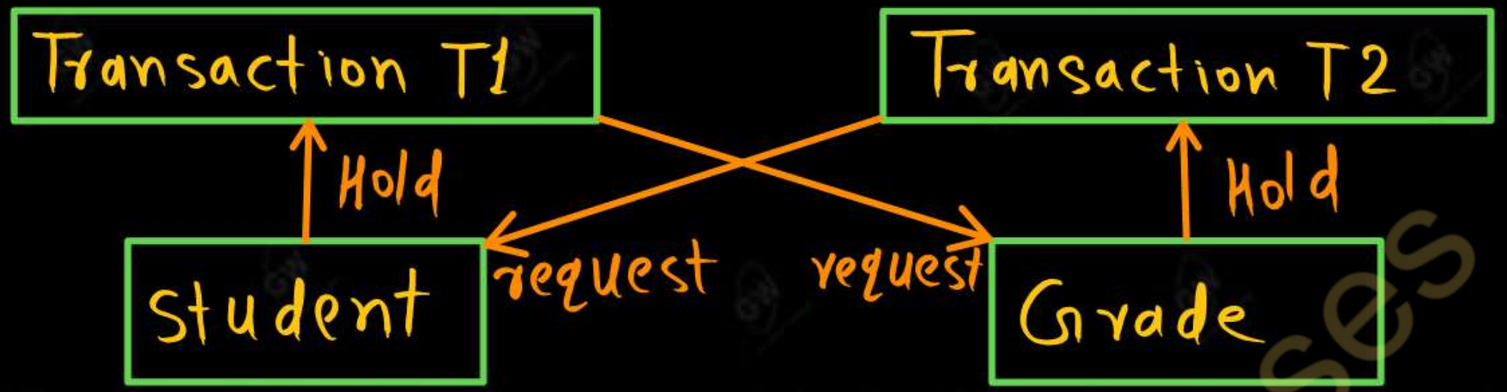
The Deadlock is a condition in a multi-user database environment where transactions are unable to the complete because they are each waiting for the resources held by other transactions. This results in a cycle of the dependencies where no transaction can proceed.

Deadlocks occur when two or more transactions wait indefinitely for resources held by each other

Characteristics of Deadlock

1. **Mutual Exclusion:** Only one transaction can hold a particular resource at a time.
2. **Hold and Wait:** The Transactions holding resources may request additional resources held by others.
3. **No Preemption:** The Resources cannot be forcibly taken from the transaction holding them.
4. **Circular Wait:** A cycle of transactions exists where each transaction is waiting for the resource held by the next transaction in the cycle.

Example – let us understand the concept of deadlock suppose, Transaction T1 holds a lock on some rows in the Students table and needs to update some rows in the Grades table. Simultaneously, Transaction T2 holds locks on those very rows (Which T1 needs to update) in the Grades table but needs to update the rows in the Student table held by Transaction T1.



Methods to handle deadlock/How it can be detected and recovered

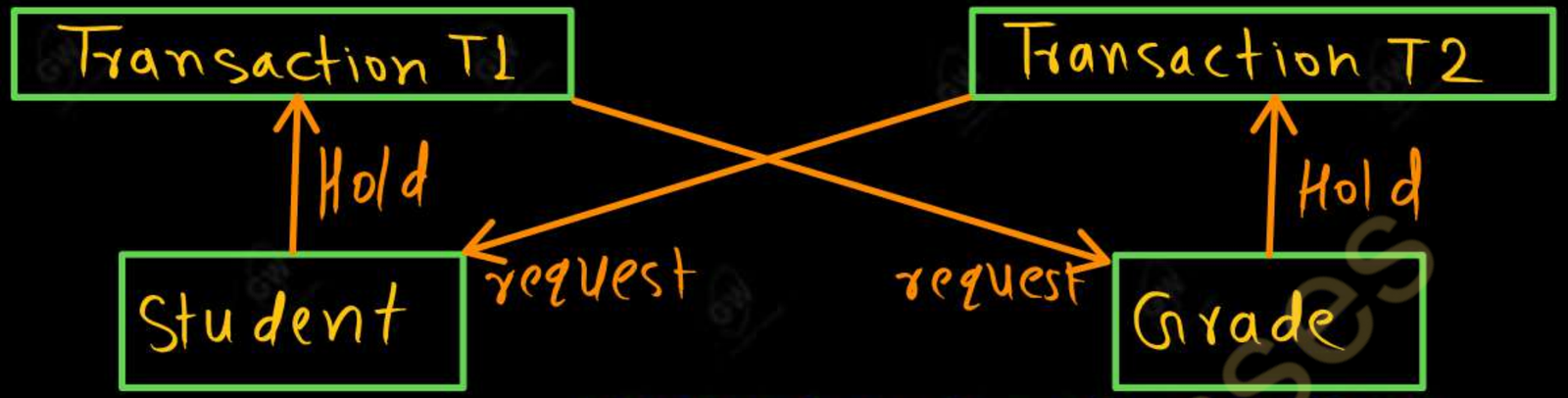
Deadlock Avoidance

Deadlock Avoidance Methods:

1. Application-Consistent Logic:

Transactions must access database tables in a consistent order.

- For example, if transactions access Students and Grades tables, they should always follow the same sequence.
- In this case, Transaction T1 waits for Transaction T2 to release the lock on Grades. Once the lock is released, T1 proceeds without conflict.



Methods to handle deadlock

Deadlock Avoidance

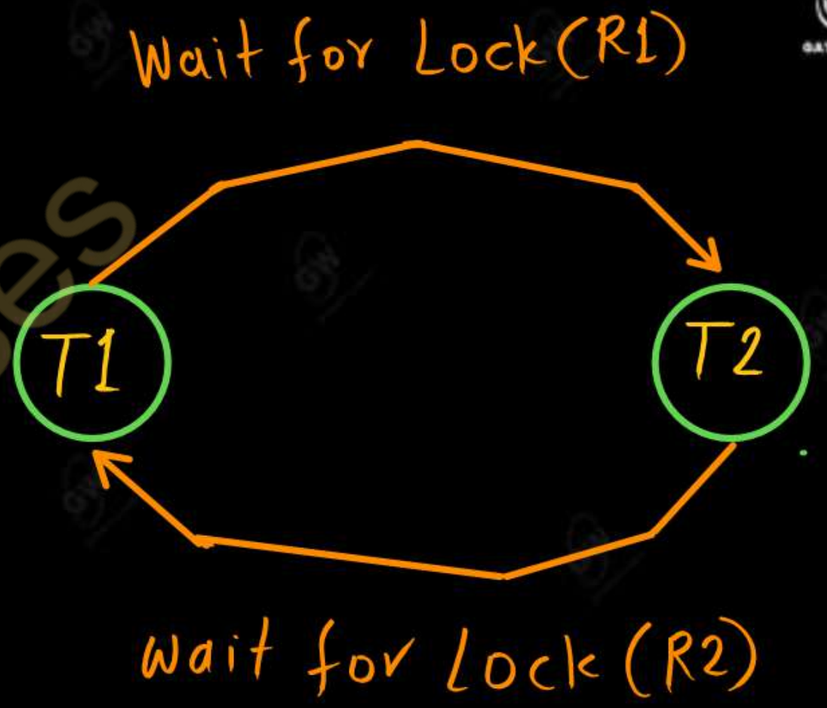
2. Row-Level Locking and READ COMMITTED Isolation Level:

- Using row-level locking ensures that only specific rows are locked, not entire tables.
- The READ COMMITTED isolation level prevents uncommitted data reads. However, this method does not completely eliminate deadlocks.

2.Deadlock Detection?

When a transaction waits indefinitely to obtain a lock, The database management system should detect whether the transaction is involved in a deadlock or not.

Wait-for-graph is one of the methods for detecting the deadlock situation. This method is suitable for smaller databases. In this method, a graph is drawn based on the transaction and its lock on the resource. If the graph created has a closed loop or a cycle, then there is a deadlock.



3. Deadlock prevention

For a large database, the deadlock prevention method is suitable. A deadlock can be prevented if the resources are allocated in such a way that a deadlock never occurs.

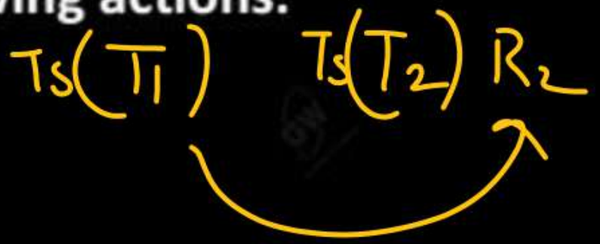
Deadlock prevention mechanism proposes two schemes:

1. **Wait-Die Scheme:**

In this scheme, If a transaction requests a resource that is locked by another transaction, then the DBMS simply checks the timestamp of both transactions and allows the older transaction to wait until the resource is available for execution.

Suppose, there are two transactions T1 and T2, and Let the timestamp of any transaction T be TS (T).

Now, If there is a lock on resources by T2 and T1 is requesting resources held by T2, then DBMS performs the following actions:



Checks if TS (T1) < TS (T2) – if T1 is the older transaction and T2 has held some resource, then it allows T1 to wait until resource is available for execution.

- That means if a younger transaction has locked some resource and an older transaction is waiting for it, then an older transaction is allowed to wait for it till it is available.
- If T1 is an older transaction and has held some resource with it and if T2 is waiting for it, then T2 is killed and restarted later with random delay but with the same timestamp. i.e. if the older transaction has held some resource and the younger transaction waits for the resource, then the younger transaction is killed and restarted with a very minute delay with the same timestamp. This scheme allows the older transaction to wait but kills the younger one.

2. Wound Wait Scheme:

In this scheme, if an older transaction requests for a resource held by a younger transaction, then an older transaction forces a younger transaction to kill the transaction and release the resource.

The younger transaction is restarted with a minute delay but with the same timestamp. If the younger transaction is requesting a resource that is held by an older one, then the younger transaction is asked to wait till the older one releases it.

Distributed Database System

A distributed database is basically a database that is not limited to one system, it is spread over different sites, i.e, on multiple computers or over a network of computers. A distributed database system is located on various sites that don't share physical components. This may be required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database.

Types:

1. **Homogeneous Database:**

   In a homogeneous database, all different sites store database identically. The operating system, database management system, and the data structures used – all are the same at all sites. Hence, they're easy to manage.

2. **Heterogeneous Database:**

In a heterogeneous distributed database, different sites can use different schema and software that can lead to problems in query processing and transactions. Also, a particular site might be completely unaware of the other sites. Different

. computers may use a different operating system, different database application. They may even use different data models for the database. Hence, translations are required for different sites to communicate.

Advantages

1.Data Accessibility

Data is distributed across multiple locations, allowing faster access for local users.

2.Improved Performance

Queries run faster since data can be processed locally at each site.

3.Reliability and Availability

If one site fails, the system can still function using data from other sites, ensuring high availability.

4.Scalability

DDBMS allows easy addition of new sites and resources without major system changes.

5.Reduced Data Redundancy

Data is fragmented and replicated intelligently, reducing unnecessary duplication.

6.Location Transparency

Users don't need to know where the data is physically stored.

7.Resource Sharing

Multiple sites can share resources like processing power and storage.

.

Disadvantages

1.Complexity

Managing distributed data across multiple sites is complex and requires advanced coordination.

2.High Cost

Implementation and maintenance of DDBMS require expensive hardware, software, and network infrastructure.

3.Data Security Issues

With data distributed across various locations, maintaining security and access control becomes challenging.

4.Data Consistency

Ensuring consistency across all sites (especially after updates) can be difficult and may require complex protocols.

Disadvantages

5.Performance Bottlenecks

Network delays or site failures can slow down system performance.

6.Difficult Backup and Recovery

Coordinating backup and recovery across multiple sites adds complexity.

7.Concurrency Control

Managing concurrent access to data at different locations can lead to conflicts.

Distributed Data Storage :

There are 2 ways in which data can be stored on different sites. These are:

1. Replication –

In this approach, the entire relationship is stored redundantly at 2 or more sites. If the entire database is available at all sites, it is a fully redundant database. Hence, in **replication, systems maintain copies of data.**

Advantageous

- As it increases the availability of data at different sites. Also, now query requests can be processed in parallel.

Disadvantages

- **Data needs to be constantly updated. Any change made at one site needs to be recorded at every site that relation is stored or else it may lead to inconsistency.**
- **This is a lot of overhead. Also, concurrency control becomes way more complex as concurrent access now needs to be checked over a number of sites.**

2. Fragmentation –

In this approach, the relations are fragmented (i.e., they're divided into smaller parts) and each of the fragments is stored in different sites where they're required. It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e, there isn't any loss of data).

Fragmentation is advantageous as it doesn't create copies of data, consistency is not a problem.

AKTU PYQS		
Q.1 	What is deadlock ? What is necessary condition for deadlock? How it can be detected an recovered	AKTU 2015-16 AKTU 2017-18
Q.2	What is distribute database management system	AKTU 2016-17/2015-16

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -4

Transaction Processing Concept

Lecture-8

Today's Target

- Fragmentation
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

Fragmentation

- Fragmentation is a process of dividing the whole or full database into various sub-tables or sub relations so that data can be stored in different systems.
- The small pieces or sub relations or subtables are called *fragments*.
- These fragments are called logical data units and are stored at various sites.
- It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e, there isn't any loss of data)

let's say, If a table T is fragmented and is divided into a number of fragments say T1, T2, T3....TN. The fragments contain sufficient information to allow the restoration of the original table T. This restoration can be done by the use of UNION or JOIN operation on various fragments.

All of these fragments are independent which means these fragments can not be derived from others. The users needn't be logically concerned about fragmentation which means they should not concerned that the data is fragmented and this is called *fragmentation Independence* or we can say *fragmentation transparency*.

Advantages :

- As the data is stored close to the usage site, the efficiency of the database system will increase
- Local query optimization methods are sufficient for some queries as the data is available locally
- In order to maintain the security and privacy of the database system, fragmentation is advantageous

Disadvantages

1. Increased Complexity in Query Processing

- Queries involving multiple fragments require coordination across different sites.
- Complex algorithms are needed to reconstruct fragmented data for global queries.

2. Higher Communication Overhead

- Accessing fragments stored on different nodes involves significant inter-node communication.
- Increases latency, especially for join operations or when data is fragmented incorrectly.

3.Difficulty in Managing Consistency

- Keeping fragmented data consistent across multiple sites is challenging.
- Changes to one fragment may require updates to dependent fragments, complicating synchronization.

4. Potential for Data Redundancy

- In some cases, fragments may overlap or be replicated unnecessarily, leading to data redundancy.

Disadvantages

5Complex Recovery and Maintenance

- **Recovery of fragmented data after failures requires accessing and synchronizing data across multiple fragments.**
- **Maintenance, such as schema changes, becomes more complicated.**

6. Security Concerns

- **Fragments stored on multiple nodes increase the surface area for potential attacks.**
- **Ensuring consistent security measures for all fragments can be challenging.**

We have three methods for data fragmenting of a table:

- **Horizontal fragmentation**
- **Vertical fragmentation**
- **Mixed or Hybrid fragmentation**

Horizontal fragmentation

- Horizontal fragmentation refers to the process of dividing a table horizontally by assigning each row (or a group of rows) of relation to one or more fragments.
- These fragments can then be assigned to different sites in the distributed system.
- Some of the rows or tuples of the table are placed in one system and the rest are placed in other systems.
- The rows that belong to the horizontal fragments are specified by a condition on one or more attributes of the relation.

In relational algebra horizontal fragmentation on table T, can be represented as follows.

$\sigma_p(T)$

where, σ is relational algebra operator for selection

p is the condition satisfied by a horizontal fragment

Note that a union operation can be performed on the fragments to construct table T. Such a fragment containing all the rows of table T is called a *complete horizontal fragment.*

For example, consider an EMPLOYEE table (T) :

Eno	Ename	Design	Salary	Dep
101	A	abc	3000	1
102	B	abc	4000	1
103	C	abc	5500	2
104	D	abc	5000	2
105	E	abc	2000	2

This EMPLOYEE table can be divided into different fragments like:

EMP 1 = $\sigma_{Dep = 1}$(EMPLOYEE) , EMP 2 = $\sigma_{Dep = 2}$(EMPLOYEE)

These two fragments are: T1 fragment of

Dep = 1

Eno	Ename	Design	Salary	Dep
101	A	abc	3000	1
102	B	abc	4000	1

T2 fragment on the basis of Dep = 2 will be :

Eno	Ename	Design	Salary	Dep
103	C	abc	5500	2
104	D	abc	5000	2
105	E	abc	2000	2

Types of Horizontal fragmentation

1.Primary Horizontal Fragmentation: It is a process of segmenting a single table in a row-wise manner using a set of conditions.

A company maintains a Customer table with the following schema:

Customer(CustomerID, Name, Region, Email)

Partitioning Criteria:

The table is partitioned based on the Region attribute.

Fragment 1: Customers from Region A.

Fragment 2: Customers from Region B.

Fragment 1

(Region A):

Rows for CustomerID 1 and 3.

(1, Alice, Region A, alice@example.com)

(3, Charlie, Region A, charlie@example.com)

Fragment 2

(Region B):

Rows for CustomerID 2 and 4.

(2, Bob, Region B, bob@example.com)

(4, David, Region B, david@example.com)

CustomerID	Name	Region	Email
1	Alice	Region A	alice@example.com
2	Bob	Region B	bob@example.com
3	Charlie	Region A	charlie@example.com
4	David	Region B	david@example.com

$F1 \leftarrow \sigma_{Region = 'Region A'}(Customer)$

$F2 \leftarrow \sigma_{Region = 'Region B'}(Customer)$

2. Derived Horizontal Partitioning

Fragmentation that is being derived from primary relation.

A company has two tables: Customer and Orders.

Customer(CustomerID, Name, Region)

Orders(OrderID, CustomerID, Amount)

Partitioning Criteria:

The Customer table is partitioned by Region (using Primary Horizontal Partitioning).

The Orders table is partitioned based on the CustomerID in each fragment of the Customer table.

Resulting Partitions:

Customer Table Fragments:

Fragment 1 (Region A): Customers 1 and 3.

Fragment 2 (Region B): Customers 2 and 4.

Derived Orders Table Fragments:

Fragment 1 (Region A): Orders 101 and 103 (from Customers 1 and 3).

(101, 1, 500)

(103, 3, 700)

F1

Oid	Cid	Amt
101	1	500
103	3	700

Fragment 2 (Region B): Orders 102 and 104 (from Customers 2 and 4).

(102, 2, 300) (104, 4, 200)

F2

102	2	300
104	4	200

CustomerID	Name	Region
1	Alice	Region A
2	Bob	Region B
3	Charlie	Region A
4	David	Region B

Orders Table:

OrderID	CustomerID	Amount
101	1	500
102	2	300
103	3	700
104	4	200

Vertical fragmentation

- Vertical fragmentation refers to the process of decomposing a table vertically by attributes or columns.
- In this fragmentation, some of the attributes are stored in one system and the rest are stored in other systems.
- This is because each site may not need all columns of a table. In order to take care of restoration, each fragment must contain the primary key field(s) in a table.
- The fragmentation should be in such a manner that we can rebuild a table from the fragment by taking the natural JOIN operation and to make it possible we need to include a special attribute called ***Tuple-id*** to the schema

. For this purpose, a user can use any super key. And by this, the tuples or rows can be linked together. The projection is as follows:

$\pi_{a1, a2,..., an}(T)$

where, π is relational algebra operator

a1…., an are the attributes of T

T is the table (relation)

Eno	Ename	Design	Salary	Dep
101	A	abc	3000	1
102	B	abc	4000	1
103	C	abc	5500	2
104	D	abc	5000	2
105	E	abc	2000	2

Eno	Ename	Design	Tuple_id
101	A	abc	1
102	B	abc	2
103	C	abc	3
104	D	abc	4
105	E	abc	5

for the EMPLOYEE table we have T1

Eno	Ename	Design	Salary	Dep
101	A	abc	3000	1
102	B	abc	4000	1
103	C	abc	5500	2
104	D	abc	5000	2
105	E	abc	2000	2

For the second. sub table of relation after vertical fragmentation

Salary	Dep	Tuple_id
3000	1	1
4000	2	2
5500	3	3
5000	1	4
2000	4	5

This is T2 and to get back to the original T, we join these two fragments T1 and T2 as $\pi_{EMPLOYEE}$ (T1 ⋈ T2)

Employee(Name, Bdate, Address, and Sex, Salary, Super_ssn, and Dno, SSN)

L1 = {Ssn, Name, Bdate, Address, Sex}

and

L2 = {Ssn, Salary, Super_ssn, Dno}

$L_1 \leftarrow \pi_{SSN, Name, Bdate, Address, Sex}$ (Employee)

$L_2 \leftarrow \pi_{SSN, Salary, Super\_ssn, Dno}$ (Employee)

- . A vertical fragment on a relation R can be
- specified by a πLi (R) operation in the relational algebra. A set of vertical fragments whose projection lists L1, L2, ..., Ln include all the attributes in R but share only the primary key attribute of R is called acomplete vertical fragmentation of R
- L1 U L2 U ... U Ln = ATTRS(R). ■ Li ∩ Lj = PK(R) for any i ≠ j, where ATTRS(R) is the set of attributes of R and PK(R) is the primary key of R.

3. Hybrid Horizontal Partitioning

- Hybrid fragmentation combines both horizontal fragmentation (dividing rows based on conditions) and vertical fragmentation (dividing columns into subsets) to create more flexible and efficient data distribution schemes.
- This method allows for a two-step fragmentation process:
- Horizontal Fragmentation: Rows of the table are divided based on a specified condition.
- Vertical Fragmentation: Columns of each horizontally fragmented subset are divided into smaller subsets.

Customer(CustomerID, Name, Region, Address, AccountBalance, TransactionHistory)

Step 1: Horizontal Fragmentation

Partition rows based on the Region attribute.

Fragment 1: Customers in Region A. (CustomerID: 1, 3, 5, ...)

Fragment 2: Customers in Region B (CustomerID: 2, 4, 6, ...)

Step 2: Vertical Fragmentation

Divide the columns of each horizontal fragment into subsets.

Fragment 1A (Region A, Personal Information):

(CustomerID, Name, Address)

Fragment 1B (Region A, Business Data):

(CustomerID, AccountBalance, TransactionHistory)

Fragment 2A (Region B, Personal Information):

(CustomerID, Name, Address)

Fragment 2B (Region B, Business Data):

(CustomerID, AccountBalance, TransactionHistory)

Advantages of Hybrid Fragmentation:

1. **Optimized Data Access:**
   - Ensures that only the necessary rows and columns are sent to queries, reducing communication overhead.
2. **Better Locality:**
   - Frequently accessed data can be stored closer to users or applications.

3. Improved Query Performance:

- Queries that target specific rows or columns are processed faster since only relevant fragments are accessed.

4.Flexibility:

- Can handle complex queries by combining the benefits of horizontal and vertical fragmentation.

Disadvantages of Hybrid Fragmentation:

1.High Complexity:

Designing and maintaining hybrid fragmentation requires in-depth analysis of query patterns and data usage.

2.Increased Maintenance Overhead:

Changes in schema or query patterns may require re-fragmentation, which is resource-intensive.

3.Synchronization Challenges:

Updating fragmented data across multiple nodes requires coordination to maintain consistency.

GW
GATEWAY CLASSES

AKTU PYQS		
Q.1	What is fragmentation Or Explain different types of fragmentation Or Explain vertical and horizontal Fragmenation	AKTU 2015-16 AKTU2016-17

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -4

Transaction Processing Concept

Lecture-9

Today's Target

- Replication
- Directory System
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

Types of Data Replication

1.Transactional Replication:

- In Transactional replication users receive full initial copies of the database and then receive updates as data changes.
- Data is copied in real-time from the publisher to the receiving database(subscriber) in the same order as they occur with the publisher therefore in this type of replication, **transactional consistency is guaranteed.**
- Transactional replication is typically used in server-to-server environments.

It does not simply copy the data changes, but rather consistently and accurately replicates each change.

Imagine a retail company with two servers:

Publisher: Central database in the headquarters that records all sales transactions.

Subscriber: Regional office database needing real-time sales updates.

When a sale happens at headquarters (e.g., a customer buys an item), the sale record is immediately replicated to the regional office database in the exact sequence. This ensures both databases stay synchronized, and any queries at the regional office reflect up-to-date transaction data.

2.Snapshot Replication:

- Snapshot replication distributes data exactly as it appears at a specific moment in time and the does not monitor for updates to the data.
- The entire snapshot is generated and sent to Users. Snapshot replication is generally used when data changes are infrequent.
- It is a bit slower than transactional because on each attempt it moves multiple records from one end to the other end.

- Snapshot replication is a good way to perform initial synchronization between the publisher and the subscriber.

3.Merge Replication:

- Data from two or more databases is combined into a single database. Merge replication is the most complex type of replication because it allows both publisher and subscriber to independently make changes to the database.
- Merge replication is typically used in server-to-client environments.
- It allows changes to be sent from one publisher to multiple subscribers.

Replication Schemes

1. Full Replication:

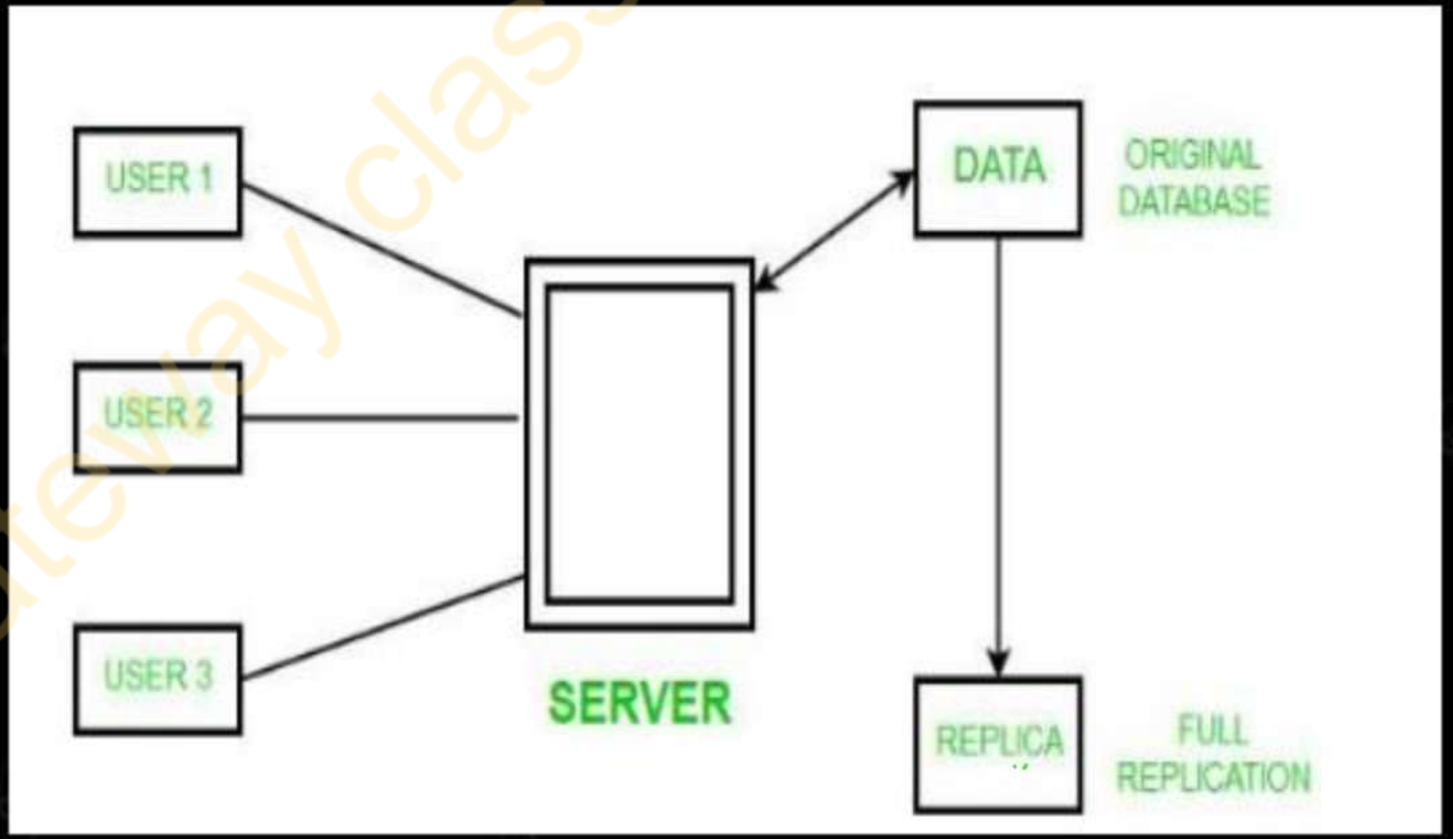
- The most extreme case is replication of the whole database at every site in the distributed system.
- This will improve the availability of the system because the system can continue to operate as long as at least one site is up.

Advantages of full replication:

- High Availability of Data.
- Improves the performance for retrieval of global queries as the result can be obtained locally from any of the local site.
- Faster execution of Queries.

Disadvantages of full replication:

- Concurrency is difficult to achieve in full replication.
- Slow update process as a single update must be performed at different databases to keep the copies consistent.



2. No replication

No replication means, each fragment is stored exactly at one site.

Advantages of No replication:

- Concurrency has been minimized as only one site to be updated
- Only one site hence easy to recover data.

Disadvantages of No replication:

- Poor availability of data as centralized server only has data.
- Slow down query execution as multiple clients accessing same server.

3.Partial replication:

- Partial replication means, some fragments are replicated whereas others are not.
- Only a subset of the database is replicated at each site.
- This reduces storage costs but requires careful planning to ensure data consistency.

Advantages of partial replication:

- Number of replicas created for a fragment directly depends upon the importance of data in that fragment.
- Optimized architecture give advantages of both full replication and no replication scheme

Features of data replication are:

- **Increased Availability:** Data replication can improve availability by providing multiple copies of the same data in different locations, which reduces the risk of data unavailability due to network or hardware failures.
- **Improved Performance:** Replicated data can be accessed more quickly since it is available in multiple locations, which can help to reduce network latency and improve query performance.
- **Enhanced Scalability:** Replication can improve scalability by distributing data across multiple nodes, which allows for increased processing power and improved performance.

Improved Fault Tolerance: By storing data redundantly in multiple locations, replication can improve fault tolerance by ensuring that data remains available even if a node or network fails.

- Improved Data Locality: Replication can improve data locality by storing data close to the applications or users that need it, which can help to reduce network traffic and improve performance.
- Simplified Backup and Recovery: Replication can simplify backup and recovery processes by providing multiple copies of the same data in different locations, which reduces the risk of data loss due to hardware or software failures.

ADVANTAGES

- Improved performance, as data can be read from a local copy of the data instead of a remote one.
- Increased data availability, as copies of the data can be used in case of a failure of the primary database.
- Improved scalability, as the load on the primary database can be reduced by reading data from the replicas.

Aspect	Replication Transparency	Fragmentation Transparency
Definition	Hides the existence of multiple copies of the same data (replicas).	Hides the division of data into fragments across multiple locations.
Purpose	Ensures users interact with data as if only one copy exists.	Ensures users access data without knowing its fragmented structure.
Data Distribution	Manages copies of the same data in multiple locations.	Manages partitions of data distributed across locations.
System Responsibility	Handles consistency and synchronization of replicas.	Handles reconstruction and retrieval of fragmented data.
Example	Updating a record automatically updates all its replicas.	Querying customer data retrieves it seamlessly, regardless of its fragments.

Directory System in DBMS

- A Directory System in a database refers to a structure or component that stores metadata, which is information about the data in the database.
- It acts as a reference to describe the organization, relationships, and constraints of the data. Essentially, it helps the database management system (DBMS) manage and access information efficiently and securely.

Key Components of a Directory System:

1.Data Dictionary:

- A central part of the directory system, the data dictionary stores metadata about the structure of the database (like tables, views, columns, and relationships).
- It is automatically managed by the DBMS and provides a way to look up details about the database schema.

2.System Catalog:

- The system catalog stores more comprehensive metadata about the entire database, including tables, indexes, stored procedures, and users.
- It also stores information on constraints, triggers, and security-related details like user permissions and roles.

Functions of a Directory System:

1.Schema Definition:

Describes the structure of the data, such as the tables, columns, data types, and relationships.

Example: It defines that Customer table contains Customer_ID, Name, Address, and Email columns, with Customer_ID being the primary key.

2. Data Integrity:

Ensures that data follows the rules defined by the schema (e.g., constraints like primary keys, foreign keys, and check constraints).

Example: The Order table may have a foreign key constraint linking Product_ID to the Products table, ensuring referential integrity.

3.User Management:

Manages access control by defining who can access or modify specific data.

Example: A user might have read-only access to a Products table but full access to an Orders table.

4.Query Optimization:

Uses metadata to optimize the execution of queries. Information about indexes and data distribution helps the DBMS choose the most efficient query plan.

Example: If an index exists on Customer_ID in the Customer table, the system will use that index to speed up a search query that filters by Customer_ID.

Consider an online bookstore database with the following schema:

Tables:

Books: Contains information about the books.

Columns: Book_ID (PK), Title, Author, Price

Customers: Contains customer information.

Columns: Customer_ID (PK), Name, Email

Orders: Stores order details.

Columns: Order_ID (PK), Customer_ID (FK), Order_Date

Order_Items: Contains the books ordered in each order.Columns: Order_Item_ID (PK), Order_ID (FK), Book_ID (FK), Quantity

Directory System Metadata:

The directory system (or data dictionary) would store metadata like:

Table Definitions:

- Books: The columns are Book_ID, Title, Author, and Price.
- Customers: The columns are Customer_ID, Name, and Email etc
- Orders: The column are Orderid, Customerid Order_Date
- Orderitems: Orderitemid, Orderid, Bookid, Quantity

Constraints:

- Primary Key on Book_ID in the Books table.
- Foreign Key on Customer_ID in the Orders table (referencing Customers.Customer_ID).

Indexes:

- Index on Title in the Books table to speed up search queries for books by title.
- Index on Customer_ID in the Customers table to speed up queries by customer.

User Roles and Permissions:

- **Admin:** Full access to all tables and schemas.
- **Sales:** Read-only access to Books and Orders tables, and write access to Orders.
- **Customer Support:** Read access to Books and Customers, but no access to Order

Example Query:

If a user runs the query:

SELECT Name, Email FROM Customers WHERE Customer_ID = 100;

The directory system ensures the following:

It verifies that Customer_ID is valid in the Customers table (using metadata).

It checks for any access restrictions for the user querying the database (e.g., read access).

It uses the metadata to retrieve the correct data, ensuring that only the Name and Email columns are returned (since the query specifies these columns).

Conclusion:

Directory Access Protocol

The Directory Access Protocol (DAP) is a set of standards used to interact with directory services, which are specialized databases designed to store and manage information about users, devices, applications, and network resources in a structured and hierarchical manner.

Purpose of Directory Access Protocol

- To enable clients to query, retrieve, and manage data stored in directory services.
- To provide a standardized interface for accessing directory information, irrespective of the underlying system.

Directory Access Protocols

1.X.500 DAP (Directory Access Protocol):

- Part of the X.500 standard developed by ITU (International Telecommunication Union).
- Operates over the OSI (Open Systems Interconnection) model.
- Provides a comprehensive, feature-rich mechanism for directory access but is considered heavyweight and complex.

AKTU PYQS		
Q.1	Explain the directory system in detail	AKTU 2019-20
Q.2	What is distributed database? List advantages and disadvantages of data replication and data fragmentation	AKTU 2019-20
Q.3	Write the difference between replication transparency and fragmentation transparency	AKTU 2015-16

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -4

Transaction Processing Concept

Lecture-10

Today's Target

- Deadlock recovery
- Some miscellaneous question
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

Deadlock Recovery

1.Killing The Process:

- Killing all the processes involved in the deadlock.
- Killing process one by one.
- After killing each process check for deadlock again and keep repeating the process till the system recovers from deadlock.
- Killing all the processes one by one helps a system to break circular wait conditions.

2.Resource Preemption:

- Resources are preempted from the processes involved in the deadlock, and preempted resources are allocated to other processes so that there is a possibility of recovering the system from the deadlock.
- In this case, the system goes into starvation.

3.Concurrency Control:

- Concurrency control mechanisms are used to prevent data inconsistencies in systems with multiple concurrent processes. (transaction)
- These mechanisms ensure that concurrent processes (transaction) do not access the same data at the same time, which can lead to inconsistencies and errors.
- Deadlocks can occur in concurrent systems when two or more processes (transaction) are blocked, waiting for each other to release the resources they need.
- This can result in a system-wide stall, where no process can make progress.

Concurrency control mechanisms can help prevent deadlocks by managing access to shared resources and ensuring that concurrent processes do not interfere with each other.

E.g. Whenever there is process write Write transaction so that you will not confuse.)

Advantages of Deadlock Detection and Recovery

- **Improved System Stability:** Deadlocks can cause system-wide stalls, and detecting and resolving deadlocks can help to improve the stability of the system.
- **Better Resource Utilization:** By detecting and resolving deadlocks, the operating system can ensure that resources are efficiently utilized and that the system remains responsive to user requests.
- **Better System Design:** Deadlock detection and recovery algorithms can provide insight into the behavior of the system and the relationships between processes and resources, helping to inform and improve the design of the system.

Disadvantages of Deadlock Detection and Recovery

- **Performance Overhead:** Deadlock detection and recovery algorithms can introduce a significant overhead in terms of performance, as the system must regularly check for deadlocks and take appropriate action to resolve them.
- **Complexity:** Deadlock detection and recovery algorithms can be complex to implement, especially if they use advanced techniques such as the Resource Allocation Graph or Timestamping.
- **False Positives and Negatives:** Deadlock detection algorithms are not perfect and may produce false positives or negatives, indicating the presence of deadlocks when they do not exist or failing to detect deadlocks that do exist.
- **Risk of Data Loss**: In some cases, recovery algorithms may require rolling back the state of one or more processes, leading to data loss or corruption.

Q.1 Discuss the immediate update recovery technique in both single-user and multiuser environments (AKTU 2022-23)

- In a multi-user environment, multiple transactions are executed concurrently, and the immediate update recovery technique ensures that updates are applied directly to the database as they occur while maintaining system consistency and recoverability.
- Logs maintain both before-image (value before the update) and after-image (value after the update)
- .Logs are written to stable storage before the actual database update (Write-Ahead Logging - WAL).
- Concurrency is managed to ensure multiple transactions do not interfere with one another. This involves Transactions acquire locks on data items to prevent conflicts. Two-phase locking (2PL) is often used to ensure serializability. (Unit-5)

Recovery in Multi-User Immediate Update:

The system must handle failures while multiple transactions are active:

1.Commit and Abort States:

- If a **transaction commits**, its changes must **persist in the database.**
- If a **transaction aborts or the system crashes**, its **changes are undone.**

2.Undo and Redo Operations:

- **Undo:** Reverse uncommitted changes using the before-image log.
- **Redo:** Reapply changes of committed transactions using the after-image log

Write-Ahead Logging (WAL):

To ensure consistency:

- Log entries are written before the database is updated.
- This guarantees that, during recovery, all necessary information is available to undo or redo changes.

Single user environment

In a single-user environment, where only one transaction executes at a time, the immediate update recovery technique applies changes to the database as soon as a transaction issues an update operation.

1. Immediate Update Mechanism:

1.1.Direct Application of Changes: When a transaction performs an update, the change is immediately applied to the database without waiting for the transaction to reach its commit point.

1.2.Logging: Each update operation is logged with both the before-image (old value) and after-image (new value). This log is crucial for recovery purposes.

2. Recovery Process:

In the event of a system failure, the recovery process involves:

1.Identifying Transaction States:

- **Committed Transactions** that reached their commit point before the failure.
- **Active (Uncommitted) Transactions:** Transactions that were in progress and had not committed at the time of failure.

Undo Operations:

For active transactions, the system reverses their effects using the before-image from the logs, restoring the database to its state before these transactions began.

Redo Operations:

For committed transactions, the system reapplies their operations using the after-image from the logs to ensure all changes are accurately reflected in the database.

. Write-Ahead Logging (WAL):

- To maintain data integrity, the system employs the **Write-Ahead Logging (WAL) protocol,** which ensures that:
- **Log records** are **written to stable storage** before **any corresponding changes** are applied to the **database.**
- This guarantees that, in the **event of a failure**, the system can **use the logs to accurately** perform **undo and redo operations**

Advantages:

- **Immediate Reflectance of Changes:**

Updates are instantly visible in the database, which can be beneficial for applications requiring real-time data accuracy.

- **Simplified Concurrency Management:**

In a single-user environment, there's no need for complex concurrency control mechanisms, simplifying the implementation of immediate updates.

Disadvantages:

1. **Increased Risk During Failures**: Since changes are applied immediately, a system failure can leave the database in an inconsistent state, necessitating a robust recovery mechanism.
2. **Overhead of Logging:** Maintaining detailed logs for every update operation introduces additional storage and processing overhead.

Usefulness of ACID properties

1. Atomicity

Definition: Ensures that a transaction is all-or-nothing; it either completes fully or not at all.

Usefulness:

- **Data Integrity:** Prevents partial updates, ensuring that incomplete transactions don't corrupt the database.
- **Error Handling:** Simplifies recovery from failures, as the system can roll back incomplete transactions to maintain a consistent state.

Example: In a banking system, transferring funds from Account A to Account B involves debiting A and crediting B. Atomicity ensures that both operations succeed or neither does, preventing discrepancies.

2. Consistency

Definition: Ensures that a transaction brings the database from one valid state to another, maintaining predefined rules and constraints.

Usefulness:

1.Data Validity: Guarantees that all data written to the database adheres to integrity constraints, such as unique keys and foreign key relationships.

2.Application Integrity: Ensures that business rules are consistently enforced across transactions.

Example: In an inventory system, if a rule states that the total quantity of items must be non-negative, consistency ensures that transactions violating this rule are aborted.

3. Isolation

Definition: Ensures that concurrently executing transactions do not interfere with each other, maintaining the illusion that each transaction operates in isolation.

Usefulness:

Concurrency Control: Prevents issues like dirty reads, non-repeatable reads, and phantom reads, ensuring accurate results in concurrent environments.

Predictability: Allows developers to reason about transactions without considering concurrent operations, simplifying application logic.

Example: In an online booking system, isolation ensures that two users booking the last available seat simultaneously do not both succeed, preventing overbooking.

4. Durability

Definition: Ensures that once a transaction is committed, its results are permanent, even in the event of a system failure.

Usefulness:

Reliability: Guarantees that committed data is not lost, providing confidence to users that their actions are preserved.

System Recovery: Facilitates recovery processes by ensuring that the system can restore committed data after crashes.

Example: After a user submits an order in an e-commerce platform, durability ensures that the order details remain saved, even if the system crashes immediately afterward.

AKTU PYQS

Q.1	What is deadlock prevention scheme?	AKTU 2015-16 AKTU 2017-18
Q.2	What is deadlock detection and recovery explain in detail. Or Explain the procedure of the deadlock detection and recovery	AKTU2015-16 AKTU 2018-19 AKTU 2022-23
Q.3.	Explain the usefulness of each ACID PROPERTY	AKTU 2022-23 AKTU 2021-22
Q.4	Discuss the immediate update recovery technique in both single-user and multiuser environments. What are the advantages and disadvantages of immediate update.	AKTU2022-23

AKTU

B.Tech III-Year

5th Semester

CS IT & CS Allied

GATEWAY CLASSES

DBMS: Database Management System

UNIT -4

Transaction Processing Concept

Lecture-11

Today's Target

- View Serializability-
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E.)

View Serializability-

If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.

View Equivalent Schedules

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

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.

Initial readers must be same for all the data items"

Condition02:

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

Write-read sequence must be same.

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.

"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

Intial read a and b t1 in s1 and s2

Final write in a and b t2(s1 and s2)

S1	
T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

Other than Intial Read

T_1

T_2

S2	
T1	T2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

So, these schedule are view Equivalent

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.

Method-03:

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

Identify Dependencies:

Read-after-Write (RAW): If Transaction Ti writes a value and Transaction Tj later reads it.

Write-after-Read (WAR): If Transaction Ti reads a value and Transaction Tj later writes it.

Write-after-Write (WAW): If TransactionTi writes a value and Transaction Tj later writes it on the same data.

Construct the Precedence Graph (Dependency Graph):

Represent each transaction as a node.

For each dependency (RAW, WAR, WAW), draw a directed edge from Ti to Tj

- Use any cycle detection algorithm (e.g., DFS).
- If the graph contains no cycle, the schedule is view serializable.
- If there is a cycle, it is not view serializable.

Problem-01

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

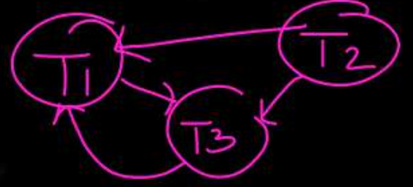
RAW
WAR
WAW

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

Conflicting operation

$R_1(A) \rightarrow W_3(A)$
$R_2(A) \rightarrow W_1(A)$
$R_2(A) \rightarrow W_3(A)$
$W_3(A) \rightarrow W_1(A)$

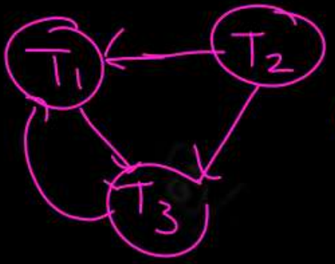
$T_1 \rightarrow T_3$
$T_2 \rightarrow T_1$
$T_2 \rightarrow T_3$
$T_3 \rightarrow T_1$



Cycle is formed
No Conflict Serializable

T_3 $W_3(A)$ - Blind Write present may or may not view

$T_1 \rightarrow T_3$ RAW
$T_2 \rightarrow T_3$ RAW
$T_2 \rightarrow T_1$ RAW
$T_3 \rightarrow T_1$ WAW



Cycle formed
Not View Serializable

Problem-02

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

T1	T2
R (A)	
A = A + 10	
	R (A)
	A = A + 10
W (A)	
	W (A)
R (B)	
B = B + 20	
	R (B)
	B = B x 1.1
W (B)	
	W (B)

Conflicting operations

$R_1(A) \rightarrow W_2(A)$: $T_1 \rightarrow T_2$

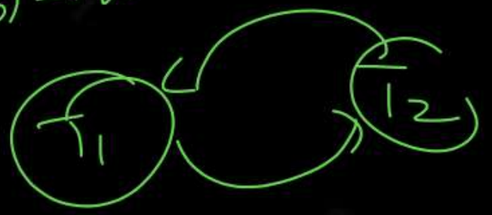
$R_2(A) \rightarrow W_1(A)$: $T_2 \rightarrow T_1$

$W_1(A) \rightarrow W_2(A)$: $T_1 \rightarrow T_2$

$R_1(B) \rightarrow W_2(B)$: $T_1 \rightarrow T_2$

$R_2(B) \rightarrow W_1(B)$: $T_2 \rightarrow T_1$

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



Cycle formed

No conflict Serializable

No Blind Write,

So definitely, not view Serializable

Problem-03

Check whether the given schedule S is view serializable or not. If yes, then give the serial schedule.

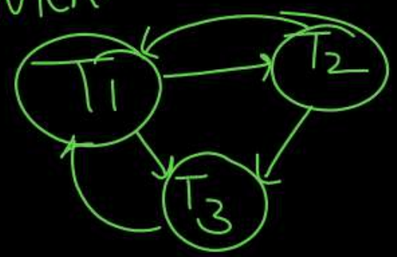
S : $R_1(A)$, $W_2(A)$, $R_3(A)$, $W_1(A)$, $W_3(A)$

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

Conflicting operations

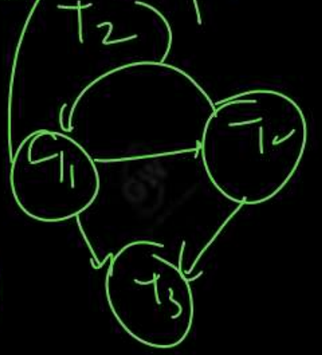
$R_1(A) \rightarrow W_2(A)$
$R_1(A) \rightarrow W_3(A)$
$W_2(A) \rightarrow W_1(A)$
$W_2(A) \rightarrow W_3(A)$
$W_2(A) \rightarrow R_3(A)$
$R_3(A) \rightarrow W_1(A)$
$W_1(A) \rightarrow W_3(A)$

$T_1 \rightarrow T_2$
$T_1 \rightarrow T_3$
$T_2 \rightarrow T_1$
$T_2 \rightarrow T_3$
$T_2 \rightarrow T_3$
$T_3 \rightarrow T_1$
$T_1 \rightarrow T_3$



Cycle formed NO conflict serializable

CBlid Write present



(not view serializable)

$T_1 \rightarrow T_2$ (RAW)
$T_1 \rightarrow T_3$ (RAW)
$T_2 \rightarrow T_3$ (WAR) (WAW)
$T_2 \rightarrow T_1$ (WAR)