DBMS

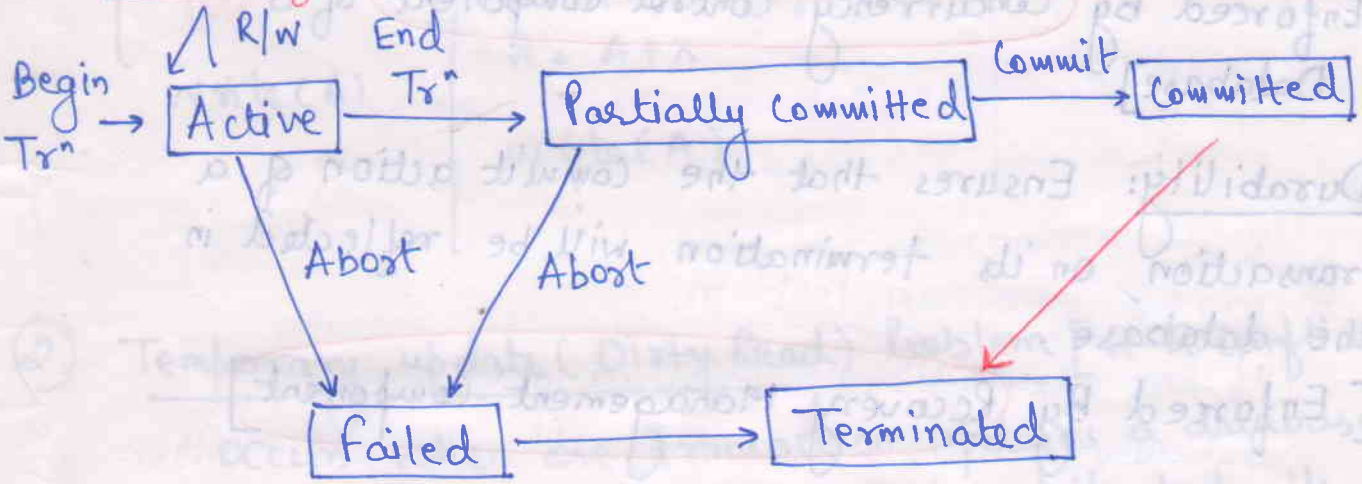
Transaction Management

A transaction is a program unit whose execution may change the contents of a database. If the database was in consistent state before a transaction, then on the completion, the database will be in consistent state.

or

Transaction is used to represent a logical unit of database processing that must be completed in its entirety to ensure correctness.

States of a Transaction:



Properties of a transaction (ACID)

① **Atomicity:** Implies that a transaction will run to completion as an indivisible unit, at the end of which either no changes have occurred to the database or database has been changed in a consistent manner.

(ALL OR NONE)

② Consistency: Implies that if the database was in a consistent state before the start of a transaction, then on termination of transaction the database will also be in a consistent state.

CONSISTENT STATE (Before transaction) —Transaction→ CONSISTENT STATE (After trⁿ)

③ Isolation: Indicates that actions performed by a transaction will be isolated or hidden from outside the transaction until the transaction terminates.

[Enforced By Concurrency Control Component of a Database]

④ Durability: Ensures that the commit action of a transaction, on its termination will be reflected in the database.

[Enforced By Recovery Management Component]

Concurrency

Concurrent execution of transactions implies that the operations from these transactions may be interleaved.

Benefits:

① Improved throughput & resource utilization.
② Helps in reducing waiting time

Problems with Concurrent Execution

① Lost update Problem [W-W Conflict]

Occurs when two transactions that accesses the same database items have their operations interleaved in a way that makes the value of the database item incorrect.

T_1	T_2
Read (A)	
A = A - 50	
update is lost	Read (A)] Reads value of 'A' before T_1 commits.
	X = A * 0.04
	A = A + X
Write (A)	
⋮	write (A)
	⋮

② Temporary update (Dirty Read) Problem [W-R Conflict]

occurs when one transaction updates a database item and then the transaction fails, but its update is read by some other transaction.

T_1	T_2
Read (A)	
A = A + 20	
Write (A)	→ updated value of A is read by T_2
	Read (A)
	A = A + 10
	Write (A)
ROLLBACK	
Read (B)	(T_1 fails causing Problem.

③ Unrepeatable Read [W-R Conflict]

If a transaction 'Ti' reads an item value twice and the item is changed by another transaction 'Tj' in between the two Read operation. Hence 'Ti' receives different values for its two read operation of the same item.

T_1	T_2
Read (A)	
Blw two read opr'n 'A' is updated	Read (A)
	A ← A - 100000
	Write (A)
Read (A)	
⋮	

④ Incorrect Summary Problem:

If one transaction is calculating an aggregate summary function on a no. of records, while other transaction is updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated - results in Incorrect Summary.

T_1	T_2
	Sum = 0
	Read (A)
	Sum = Sum + A
	Read (Y)
	Sum = Sum + Y [Incorrect Summary]

Schedules :

When several transactions are executing concurrently then the order of execution of various instructions is known as Schedule.

Or

A Schedule 'S' of n transactions $T_1, T_2, \ldots\ldots T_n$ is an ordering of operations of the transactions in chronological order, i.e., if operation X in T_i proceeds operation Y in T_j then X should always proceeds Y in any schedule, but the operations of T_i can be interleaved with operations of T_j.

① Serial Schedule:

↳ does not interleave the actions of any operations of different transactions.

↳ when transactions are executing serially, it always ensures a consistent state.

$T_1 \longrightarrow T_2$] T_1 is followed by T_2.

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

Q) Difference b/w Serial and Non-Serial Schedule

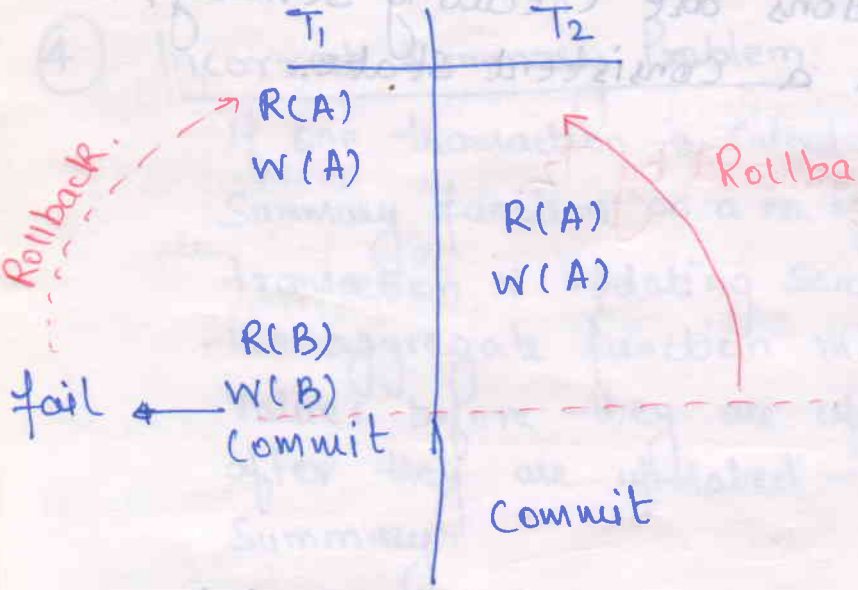
② Complete Schedule:

↳ If the last operation of each transaction is either abort or commit.

T_1	T_2
R(A)	
W(A)	R(A)
Abort	
	W(A)
	Commit

③ Recoverable Schedule:

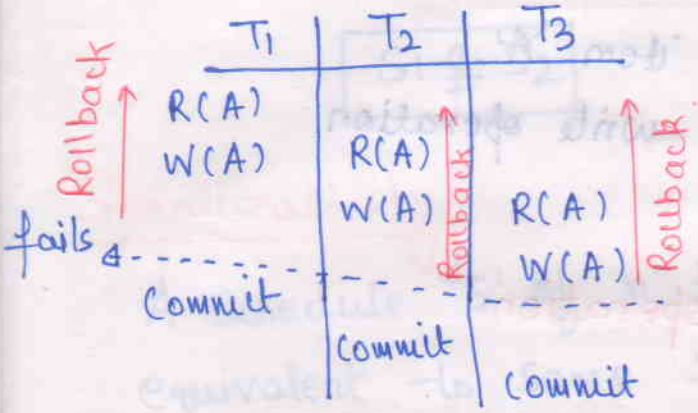
↳ Is one where for each pair of transactions (T_i, T_j) such that T_j reads a data item that was previously written by T_i, then the commit op^n of T_i should appear before commit op^n of T_j.



④ Cascadeless Schedule:

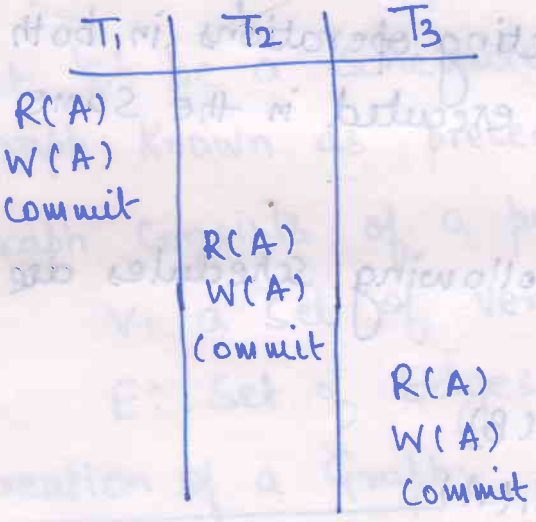
↳ If one transaction failure causes multiple transactions to rollback is called

eg:- of Cascading Rollback.



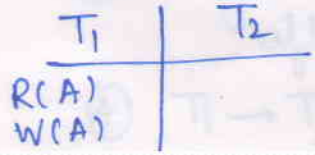
A Cascadeless Schedule is One where for each pair of transactions (T_i, T_j) such that T_j reads a data item that was written by T_i, then the commit op^n of T_i should appear before the read op^n of T_j.

eg:-



(5) Strict Schedule:

If a Value written by a transaction cannot be read or overwritten by other transaction until the transaction is either aborted or committed.



Every Strict Schedule is both Recoverable and Cascadeless.

Conflict operations:

i) Belong to different transactions.

ii) Access to same database item 'A'.

iii) Atleast one of them is a write operation.

T_1	T_2	
R(A)	W(A)	Conflict operation
W(A)	R(A)	
W(A)	W(A)	
R(A)	R(A)	Non-Conflict opr^n

EQUIVALENT SCHEDULE

(1) Conflict Equivalent:

Two schedules are said to be Conflict equivalent if all conflicting operations in both the schedules must be executed in the same order.

Ques) Check which of the following schedules are Conflict Equivalent?

S1: $R_1(A)$ $R_2(B)$ $W_1(A)$ $W_2(B)$

S2: $R_2(B)$ $R_1(A)$ $W_2(B)$ $W_1(A)$

S1

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

S2

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

$S_1 \subseteq S_2$

Ques) S1: $R_1(A)$ $W_1(A)$ $R_2(B)$ $W_2(B)$ $R_1(B)$

S2: $R_1(A)$ $W_1(A)$ $R_1(B)$ $R_2(B)$ $W_2(B)$

$S1 \not\equiv S2$

Serializability

A Schedule 'S' of n transactions is Serializable if it is equivalent to some serial schedule of the same 'n' transactions.

① **Conflict Serializable:**

If it is conflict equivalent to serial schedule.

Test for Conflict Serializability

Precedence Graph is used.

- Let 'S' be a schedule, construct a directed graph known as precedence graph.
- Graph consists of a pair of $G = (V, E)$ where

V: a set of vertices

E: Set of Edges.

Creation of a Graph:

① Create a Node for each transaction.

② A directed edge, $T_i \rightarrow T_j$, if T_j reads value of an item written by T_i.

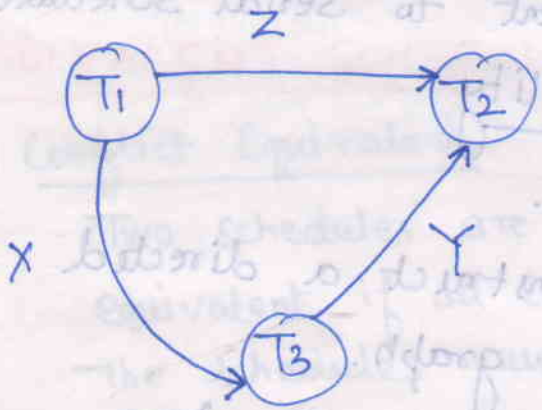
③ Edge $T_i \rightarrow T_j$, if T_j writes a value into item after it has been read by T_i.

④ $T_i \rightarrow T_j$, if T_j write after T_i write.

A Schedule is Conflict Serializable if and only if precedence graph is acyclic.

Ques)

T_1	T_2	T_3
r(x)		
	r(z)	
r(z)		
		r(x)
		r(y)
		w(x)
	r(y)	
	w(z)	
	w(y)	

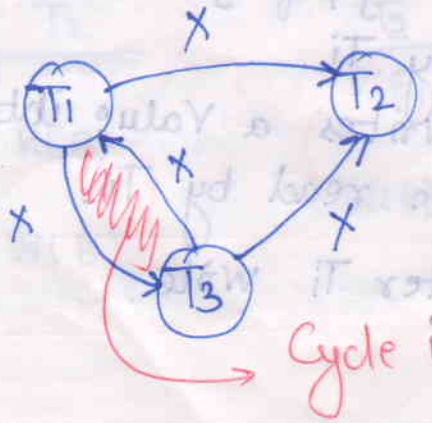


No Cycle

Conflict Serializable ✓

Ques)

T_1	T_2	T_3
r(x)		
		r(x)
		w(x)
w(x)		
	r(x)	



Conflict Serializable ✗

Ques) which of the following Schedule is Conflict schedule.

a) $r_1(x)$; $r_3(x)$; $w_1(x)$; $r_2(x)$; $w_3(x)$ → Not
b) $r_1(x)$; $r_3(x)$; $w_3(x)$; $w_1(x)$; $r_2(x)$ → Not
c) $r_3(x)$; $r_2(x)$; $w_3(x)$; $r_1(x)$; $w_1(x)$ → Yes
d) $r_3(x)$; $r_2(x)$; $r_1(x)$; $w_3(x)$; $w_1(x)$ → Not

use Precedence Graph to check.

② View Serializability:

Two Schedules S and S' are view equivalent if the following conditions are met:

i) For each data item Q, if Ti reads an initial value of Q in Schedule S, then Ti must in S' also reads an initial value of Q.

ii) If Ti executes Read Q in S, and that value was produced by Tj (if any), then Ti must in Schedule S' also reads the value of Q that was produced by Tj.

iii) For each data item Q, the transaction that perform the final write(Q) operation in Schedule S must perform the final write(Q) in Schedule S'.

A Schedule is View Serializable, if it is view equivalent to a Serial Schedule.

Note:- Every Conflict Serializable Schedule is also view Serializable but not vice-versa.

Concurrency Control

It is the process of managing simultaneous execution of transactions in a shared database, to ensure the serializability of transactions.

Purpose of Concurrency Control:

i) To enforce Isolation
ii) To preserve database consistency
iii) To resolve read-write and write-write conflicts.

Concurrency Control Techniques:

① **Lock-BASED Protocol:** A Lock guarantees exclusive use of a data item to a current transaction.
- ↳ A transaction acquires a lock prior to data access.
- ↳ Lock is released when transaction is completed.

→ Requires that all data items must be accessed in a mutually exclusive manner, i.e. when one transaction is accessing data item, no other transaction is allowed to update the data item simultaneously.

Types of locks:-

i) **Shared Lock:** Data item can only be read. → 'Lock-S'

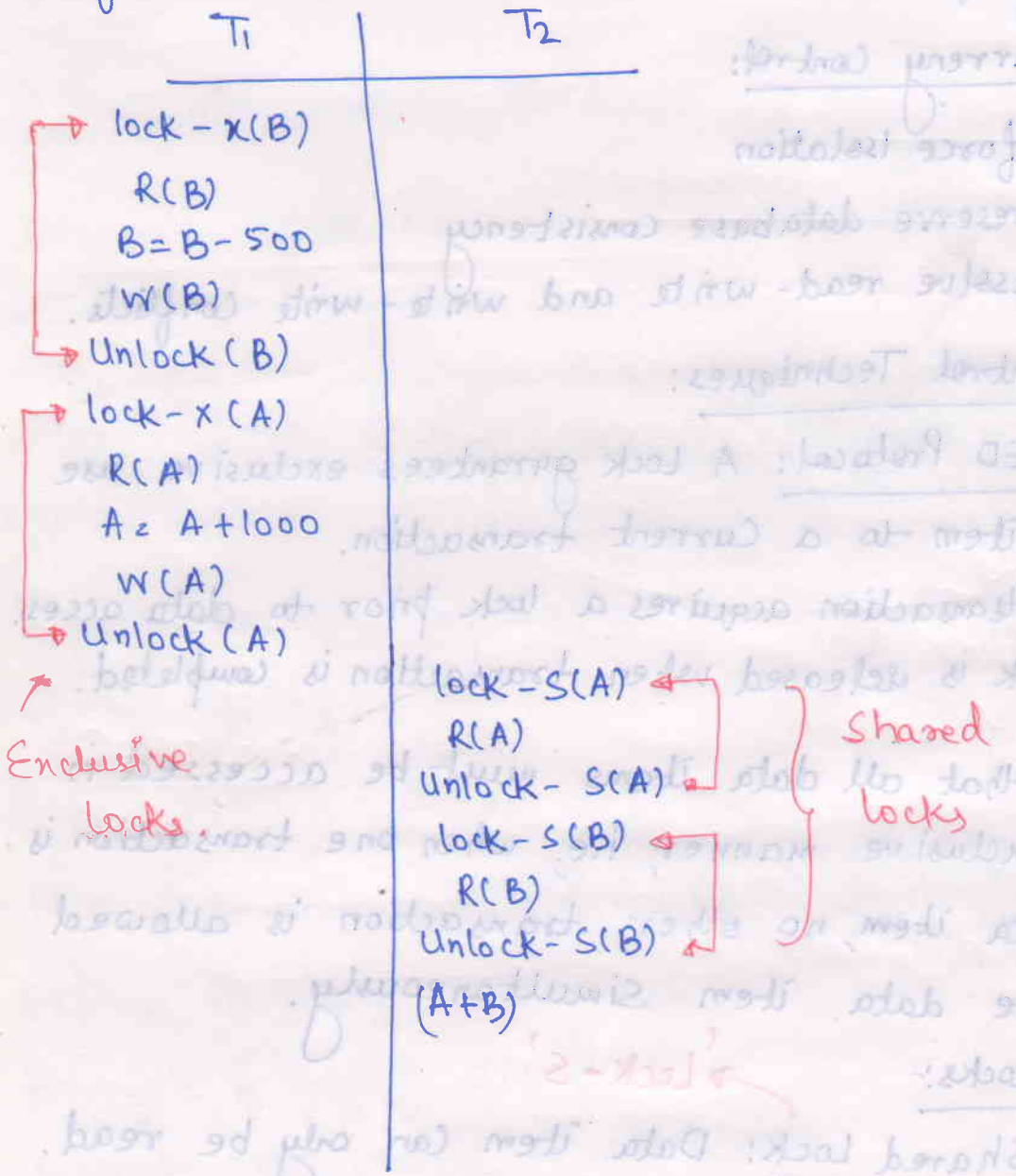
ii) **Exclusive lock:** Both read and write. → 'Lock-X'

Compatibility b/w lock modes

	S	X
S	✓T	XF
X	XF	XF

Note: Any number of transactions can hold shared locks on an item, but exclusive lock can be held only by one transaction at a time.

eg:-



Conversion of locks:

1. Upgrading: Read-lock ⟶ Write-lock.
2. Downgrading: Write-Lock ⟶ Read-Lock.

(2) Two-Phase Locking Protocol:

→ Requires Both locks and Unlocks being done in two phases.

→ It assumes that a transaction can only be in one of the following two phases:

i) Growing (Expanding Phase): In this phase new locks on items can be acquired but none can be released. The point at which the transaction has obtained its final lock is called LOCK POINT.

ii) Shrinking Phase: Existing locks can be released but no new locks can be acquired. The transaction enters the shrinking phase as soon as it releases the first lock after crossing lock Point.

Enforces Serializability but may reduce concurrency due to the following reasons:

i) Holding lock un-necessarily.
ii) Locking too early.
iii) Penalty to other transactions.

Variations of 2PL locking protocol:

① Conservation 2PL (Static PL)

↳ Requires the transaction to obtain all the locks before it starts and release all locks after it commits.

→ Avoids Cascading Rollback.
→ Deadlock free protocol.

② Strict 2PL:

↳ Requires that a Transaction 'T' does not release any of its exclusive locks until transaction commits.

→ Helps in creating Cascadeless schedule.
↳ It means Cascading rollback can be prevented.

③ Rigrous 2PL: In this a Transaction does not release any of its locks (exclusive / shared) until the transaction commits or aborts.

→ Avoids Cascading Rollback.

Deadlock May occur.

DEADLOCK

A System is in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set.

eg:- T_1: access data items X and Y.

T_2: access data items Y and X.

If T_1 has not unlocked data item Y, T_2 Cannot begin;
If T_2 has not unlocked data item X T_1 Cannot Continue. So T_1 and T_2 each wait for other to unlock the required data item.

Techniques to Control Deadlock:

① **Deadlock Prevention:** This protocol ensures that the system will never enter into a deadlock state. Deadlocks can be prevented by preventing atleast one of the four cond^n:

(a) Mutual Exclusion
(b) Hold and Wait
(c) No Preemption
(d) Circular Wait

Other techniques used in Deadlock Prevention are the following:

(i) Use of timestamps

↳ (a) Wait-Die Scheme:

Assuming that T_i requests a data item currently held by T_j

if $Ts(T_i) < Ts(T_j)$ [T_i older than T_j], then T_i is allowed to wait otherwise if (T_i younger than T_j) aborts T_i (T_i dies) and restart it later with same timestamps.

(b) Wound-Wait Scheme:

if $Ts(T_i) < Ts(T_j)$ [T_i older than T_j], then abort T_j (T_i wounds T_j) and restart it with same timestamp, otherwise if (T_i younger than T_j) T_i is allowed to wait

ii) Time out-Based Schemes:

↳ Based on Lock-timeouts.

→ A transaction that has requested a lock waits for at most a specified amount of time. If the lock has not been granted within that time, transaction is said to timeout, and it rolls itself back and restarts.

Starvation:

A transaction is starved if it cannot proceed for an indefinite period of time while other transactions in the system continue normally.

→ Starvation can also occur if algorithm dealing with deadlock selects the same transaction as victim repeatedly, thus causing it to abort and never finish execution.

② Deadlock Detection:

Using directed Graph, called a Wait-for Graph.

$G_1 = (V, E)$ → Set of Edges

↓

Set of vertices

$T_i \longrightarrow T_j$ (directed edge)

↳ implies that T_i is waiting for T_j to release a data item that it needs.

[Deadlock occurs if there is a cycle in Wait-for Graph]

③ Deadlock Avoidance: Employs an algorithm to access the possibility that deadlock could occur and acting accordingly.

eg: Banker's Algo.

Recovery from Deadlock:

(i) Selection of a victim

(ii) Rollback

Total rollback Partial rollback

(iii) Starvation

TIME STAMPING METHOD

A timestamp is a tag that can be attached to any transaction or any data item, which denotes specific time on which the transaction or data item had been activated in any way.

Timestamps

If a transaction Ti has been assigned timestamp Ts(Ti), and a new transaction Tj enter the system, then Ts(Ti) < Ts(Tj). Methods for implementing this scheme is as follows:

a) System clock as timestamp.

b) Logical counter that is incremented after a new timestamp has been assigned.

To implement the scheme, each data item 'Q' has two timestamp values:

i) W-timestamp(Q): Largest timestamp of any transaction that executed write(Q) successfully.

ii) R-timestamp(Q): largest timestamp of any transaction that executed read(Q) successfully.

TIMESTAMP-ORDERING Protocol

It ensures that any conflicting read & write operations are executed in timestamp order.

(1.) T_i issues read(Q)

↳ if $TS(T_i) < W\text{-timestamp}(Q)$; Transaction T_i is an older transaction than the last transaction that wrote the value of Q. [Request will fail]

↳ if $TS(T_i) \geq W(Q)$ T_i is allowed to read the updated value of Q. [Request will succeed]

(2) T_i issues write(Q)

↳ if $TS(T_i) \geq W(Q)$ and $TS(T_i) \geq R(Q)$; T_i is allowed to write the value of Q and $TS(T_i)$ becomes the current value of W(Q)

↳ if $TS(T_i) < R(Q)$; Younger transaction is already using current value of Q. So updation is not allowed.

↳ If $R(Q) \leq Ts(T_i) < W(Q)$; means that Younger transaction has already updated the value of Q, and the value that T_i is writting must be based on an obsolete value of Q and hence 'T_i' is not allowed to modify Q.

Thomas' Write Rule

Suppose that Transaction 'T_i' issues Write(Q)

i) If $Ts(T_i) < R(Q)$, then value of Q that T_i is producing was previously needed, and it had been assumed that the value would never be produced. Hence System rejects the write operation and rolls T_i back.

ii) If $Ts(T_i) < W(Q)$, then T_i is attempting to write an obsolete value of Q, Hence this write opr^n can be ignored.

iii) Otherwise the System executes write operation and sets $W(Q)$ to $Ts(T_i)$.

Failure classification:

↳ i) Transaction Failure → logical Error
                         → System Error

ii) System Crash

iii) Disk Failure

iv) Exception Condition.

Storage Structure:

Classification

↳ (i) Volatile Storage (Eg:- Main/Cache Memory)

↳ Fast access

↳ direct access to data

(ii) Non-Volatile Storage

eg:- (disks and magnetic tapes)

↳ Slower than volatile storage

(iii) Stable Storage

↳ Info^n is never lost

↳ theoretically never

Cannot be guaranteed.

Recovery Schemes:

① LOG BASED Recovery:-

→ Log is the most commonly used structure for recording database modification. Update log has the following fields:-

ⓐ # Transaction Identifier

ⓑ # Data item identifier

ⓒ # Old Value (Prior to write)

ⓓ # New Value (After write)

(*) Log is written before any update is made to the database. This is called as the Write-ahead log strategy.

According to this strategy, transaction is not allowed to modify the physical database until the UNDO portion of the log is written to stable storage.

→ In effect, both the undo and redo portion of the log will be written to the stable storage before a transaction commit.

Operations in Recovery Procedure:

(i) Undo(T_i) → Restore old values.

(ii) Redo(T_i) → updates by New values.

Eg:- of a log Record

<T_i start>

<T_i, X_j, V_1, V_2> (V_2: New Value, V_1: old value, X_j: Write on X_j)

<T_i commit>

<T_i Abort>.

Approaches in Transaction Recovery Procedure:

① Deferred Database Modification:-

→ In this transaction operations do not immediately update the physical database. Instead Only transaction log is updated.

→ Database is physically updated only after the transaction reaches its commit point, using info" from transaction log.

→ Also Known as NO-UNDO/REDO AlGO.

② Immediate Database Modification:

→ Database is immediately updated by the transaction operations during the execution of transaction, even before it reaches Commit point.

→ In Case of transaction Abort before it reaches Commit point, a ROLLBACK or Undo operation needs to be done to restore the database to Consistent State.

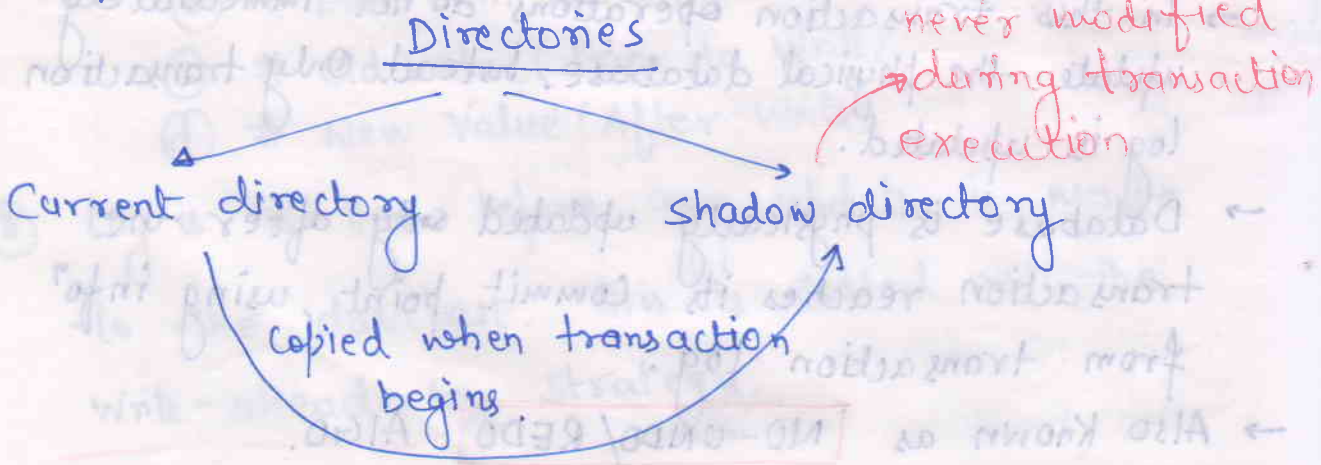
→ Also Known as UNDO/NO-REDO AlGO.

② Shadow Paging Recovery Scheme:

→ does not require the use of a log in a single-user env.

→ Considers that the database is made up of a no. of fixed-size disk pages (or disk blocks) - say, (n) - for recovery purpose.

→ A directory with n-entries is Constructed, where the ith entry points to the ith database page on disk.



When a write operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten.

→ 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.

→ Recovery is done by restoring the shadow directory.

(3) Recovery with Concurrent Transactions

Four ways:

i) Interaction with concurrency control:
   - ↳ rollback failed transaction
   - ↳ UNDO all update operation.

ii) Transaction Rollback:
   - ↳ rollback using log

iii) CheckPoints:
   - ↳ Use of checkpoint to reduce no. of log records.

iv) Restart Recovery:

Assignment

Q1) Consider following two transactions:

T_{31} :	T_{32} :
r(A);	r(B);
r(B);	r(A);
if A=0 then B:=B+1;	if B=0 then A:=A+1;
W(B)	W(A)

Add lock and Unlock instructions to T_{31} and T_{32}, so that they observe two-phase locking protocol.

Q2) Differentiate b/w Explicit and Implicit lock.

Q3) Explain the importance of "write to log" before changing db values.

Q4) Explain transaction Roll Back.

Q5) Discuss No waiting, Cautious Waiting and timeout protocols for deadlock prevention.

Q6) How a non-Serializable Schedule can be changed into an Serializable Schedule.

Q7) Difference b/w log-based recovery and Shadow Paging scheme.

Q8) Explain Write-ahead logging.

Q9) How does 2PL protocol achieves Serializability.

Q10) Explain Partially Committed state of a transaction.

Q11) Benefits of strict 2PL?

Q12) Explain the need of checkpoints.

XXXXXXX

XXXXXXX