Date _____ Day _____

"CONCURRENCY CONTROL"

→ Till now we already know how to check whether a schedule will maintain the consistency of DB or not using [C.S, V.S, recoverability, cascadless, strict].

→ Now we study protocols that guarantees to generate schedule which satisfy properties specially (C.S)

→ Actual problem is, different transaction trying to access data at same time.

→ Following are the concurrency control methods.

① Timestamping Protocol:

(no deadlock but can have starvation)

→ Basic idea of timestamping is to decide the order b/w the transactions before it enters into the system.

→ So that in case of conflict during execution we can resolve the conflict using ordering.

- Timestamp with transaction:

With each transaction T_i we associate a time-stamp denoted by $T.S(T_i)$,

It is the value of the system clock when a transaction enters into the system, so if a new trans T_j enters after T_i, then $T.S(T_i) < T.S(T_j)$,

~~always unique~~,

This will always be unique and will remain fixed throughout the execution.

Date _____ Day _____

→ If $T.S(T_i) < T.S(T_j)$, then in Serializability order the system ensure that in the resultant Conflict Ser— Schedule T_i will execute 1st.

• Timestamp with Data item:

→ For each data item Q, protocal mantains two timestamps. jis transaction ne sabse latest write perform kra wo system mein kab enter tha.

↳ W-timestamp (Q) is the latest timestamps of any trans that executed write(Q) succesfully

↳ R-timestamp (Q) is the latest " .

Example: If T_i transaction enters system at 12:04 pm then when it writes on Q the write(Q) = 12:04. This means the the transaction that has written on Q entered the System at 12:04 pm. And when Tj will write on Q then write(Q) = 3:05 pm.

"junior ko allowed hai, Senior ko nhi hai"!

→ T_i Request for Read(Q) (read ka sirf WTS(Q) k saath check krenge) ~~bez read read has no~~ conflict

↳ if $TS(T_i) < W.T.S(Q)$, → read permission cannot be
Senior junior granted, must roll back.

T_i	T_x	T_i ka timestamp kam hai means it is senior and the one who has currently written on Q is junior.
R(Q)	W(Q)	

↳ if $T.S(T_i) \geq W.T.S(Q)$ → permission granted
junior senior & $R.T.S(Q) = \max(RTS(Q), TS(T_i))$

Date _____ Day _____

* T_i request for write(Q)

↳ if $TS(T_i) < RTS(Q)$ → not granted, rollback
(senior, junior)

↳ if $TS(T_i) < WTS(Q)$ → not granted, roll back
(senior, junior)

↳ if $TS(T_i) \geq RTS(Q)$ → granted
$TS(T_i) \geq WTS(Q)$ & $WTS(Q) = \max(WTS(Q), TS(T_i))$
(junior, senior)

- **Strict To Algorithm:**

↳ Ensures schedules are both strict & C-S.

↳ A transaction T issues a read_item(X) or write_item(X), such that,

$TS(T) > W\text{-}TS(Q)$ has its read or write oper delayed until T' that wrote the value of X has comitted or aborted.

- **Thomas Write Rule:**

write request: if $TS(T_i) < WTS(Q)$, not granted & instead of roll back ignore the write opesation requested by Ti & continue processing.

→ but by following Thomas write it is not guaranteed to be C-S. but it will be V-S.

Date _____ Day _____

② Lock Based Protocol:

→ First obtain a lock on a data item then performed a desired operation. and then unlock it.

~~→ Modes of Lock:~~

- **Binary Locks:**

  Two states → Locked (x)
  Unlocked (x)

  Binary locking too restrictive for DB items

  ↳ before any read(x)/write(x) the transaction T must 1st acquire lock. Or else it must wait for the lock to be freed.

  ↳ After completion of operation free the lock.

- **Shared/Exclusive Or Read/Write Locks**

  ↳ Read oper on the same item are not conflicting.

  ↳ Must have the exclusive lock to write.

- **Share lock/read_lock(x):** Agr kisi ko sirf read oper perform krna hai to ye le sakta hai. Multiple transaction intending to read X can acquire this lock at same item for the X.

- **Exclusive/write_lock(x):** Agr kisi ko write or read done krna hai to wo ye lega. Or agr kisi k paas exclusive lock hai to koi doosra aake item X par shared lock bhi nhi le sakta it must wait.

Shared / excl.	Shared	excl.
Shared	T	F
excl.	F	F

Page No. ☐

Date _____ Day _____

• 2-Phase Locking:

↳ Basic 2PL:

→ This protocol requires that each trans in a sched will be two phased:

↳ Expanding / Growing phase

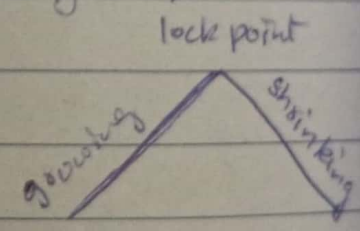
New locks can be acquired but none can be released.

Lock conversion upgrades must be done in this phase

↳ Shrinking phase

Existing locks can be released but none can be acquired

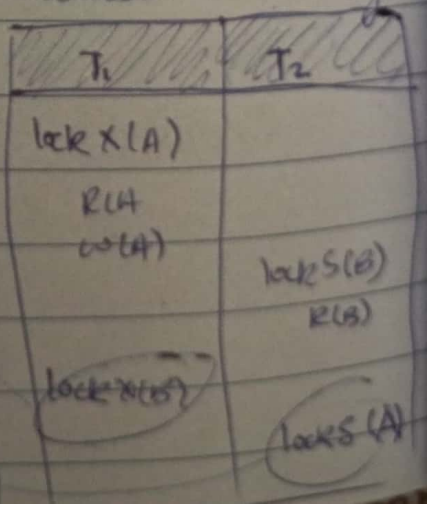
Downgrades must be done during this phase.



→ Ensures C-S / V-S, the order of serializability is the order in which transaction reach lock point.

→ May generates irrecoverable schedules & cascading rollbacks.

→ Deadlock can occur. →



Date _____ Day _____

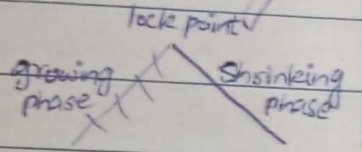
↳ Conservative (static) 2PL:

↳ Requires a transaction to lock all the items it accesses before the transaction begins

↳ Deadlock-free protocol. (kyu there is no hold)

↳ if all locks are not available then transaction must release the lock acquired so far and wait.

↳ There is no growing phase, transaction first will acquire all the locks required & then directly start from lock point.

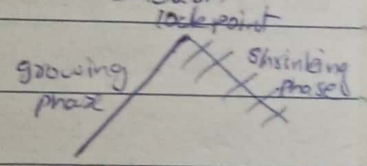


↳ C.S & V.S guaranteed

↳ But it ~~is~~ has cascadE rollbacks & is recoverable schedule.

↳ Rigorous 2PL:

→ All the locks must be held until transaction comitts



→ There is no shrinking phase

→ Te jab tak koi data item unlock nhi hoga to doosra usko ~~take~~ use nhi kr sakta to dirty read bhi nhi hoga. (→ all locks are freed.

→ Use tabhi hoga jab comitt hojayega. Or comitt k baad Dirty read nhi hot.

→ Ensures recoverablity, cascadless (as no Dirty Read)

→ C.S & V.S guaranteed.

→ Deadlock can occur.

→ inefficiency increases.

→ degree of concurrency decr.

T_1	T_2
lockX(A)	
R(A)	
W(A)	
Comitt	
	lockS(A)
	R(A)

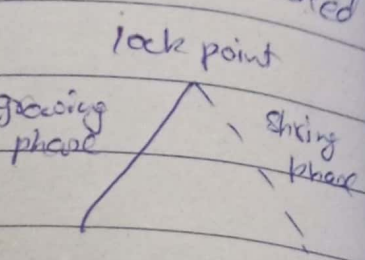
Date _____ Day _____

↳ Strict 2PL:

→ In shrinking phase unlocking of exclusive lock not allowed but shared locks can be unlocked

→ Partial shrinking phase

→ Shared ko isliye unlock krshe bcz us se dirty read ka koi issue nhi hota.



Or jab shared lock acquire krshe the to ~~esi tatra~~ iska mtlb us value par sirf read hi perform hosha tha. write hoa hi nhi tha to dirty read bhi nhi hoga.

→ Deadlock can occur!

⇒ Dealing with Deadlocks:

↳ 2PL

↳ Transaction that needs several item will lock them in that order

} both impractical

• Protocols based on timestamp:

① Wait-die

if $TS(T_i) < TS(T_j)$ (senior)

T_i waits

else (nib junior)

Abort T_i and restart it later with same timestamp.

Day

② Wound-wait :

if $TS(T_i) < TS(T_j)$
(senior)
Abort (wounds) T_j and restart it later with same ts
else (junior)
T_i wait,

• Deadlock Prevention Protocol:

↳ No Waiting Algorithm:

→ If a transaction is unable to obtain a lock, immediately abort it & restart later

↳ Cautious waiting algorithm:

→ Deadlock free
→ Reduce the no. of needless aborts/restarts.
→ If T_j is not blocked (not waiting for some other locked item), then
T_i is blocked and allowed to wait; otherwise abort T_i.
Suppose that T_i tries to lock an item X but is not able to do so bcz X is locked by T_j
Then,

Date _____ Day _____

- **Deadlock Detection:**
  - **Wait-for-Graph:**
    - ↳ One node created for each transaction that is currently executing
    - ↳ When T_i is waiting to lock an item X that is currently locked by T_j, a directed edge ($T_i \rightarrow T_j$) is created
    - ↳ When T_j releases the locks on the item T_i was waiting edge is removed.
    - ↳ "We have a state of deadlock iff the graph has a cycle."

- **Victim Selection:**
  - ↳ Deciding which trans to abort in case of deadlock
  - ↳ Select the youngest transaction.

- **Timeouts:**
  - If system waits longer than a predefined time, it aborts the transaction.

- **Starvation:**
  - Occurs if a transaction cannot proceed for an indefinite period of time while other trans continue normally

Page No. _____