Concurrency control Protocols :-

Main aim to achive serializability or recoverbility

For this we use Concurrency control protocols.

To Achive this we use locking protocols.

Shared - Exclusive Protocol.

Shared Exclusive lock :-

Shared lock (S) :- if transaction locked data item in shared mode then allowed to read only.

Exclusive lock (X) :- if transaction locked data item in exclusive mode then allowed to Read & write both.

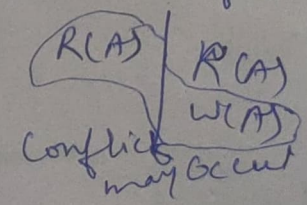
For eg :	T_1	T_2
Shared lock	- S(A)	X(A)
only Read	- R(A)	R(A)
	- U(A)	W(A)
Unlock After Completion work.		U(A)

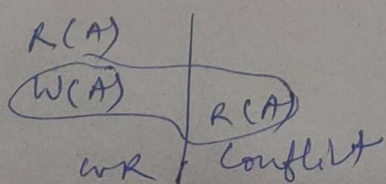
we can't take shared lock because we have to Read & write both so we used exclusive lock.

Compatibility of Locks :-

Grant \ Request	S	X
S	Yes	No
X	No	No

Grant - Shared lock on A Request R(A) — R(A) No conflict





W(A) W(A) W-W conflict.

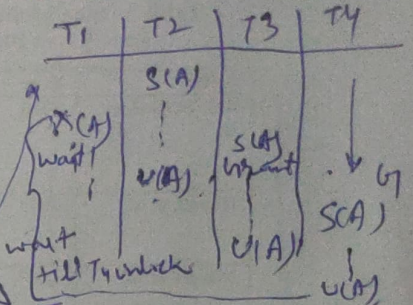
Problems in S/X locking :-

1) May not sufficient to produce only serializable schedule. — ~~Not Irrecoverable~~

2) May not free from Irrecoverability.

3) May not free from deadlock (infinite waiting)

4) May not free from starvation (finite waiting)

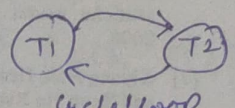


Two transactions wait for resource & entire is infinite loop.

Eg:-

T1	T2
X(A)	
R(A)	
W(A)	
U(A)	S(A) ← when T1 unlock A.
	R(A)
X(B)	U(A)
R(B)	
W(B)	
U(B)	

May not sufficient to produce serializability



cycle/loop

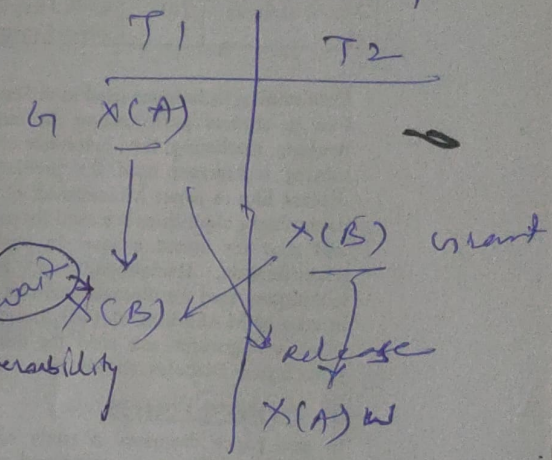
Not serializable

Eg:-

T1	T2
X(A)	
R(A)	
W(A)	
U(A)	X(A)
	R(A)
	W(A)
	Commit

Fail

May not free from Irrecoverability



2PL (2 Phase locking)

→ Growing Phase : locks are acquired and no locks are released

→ Shrinking Phase - locks are released and no locks are acquired.

S/X

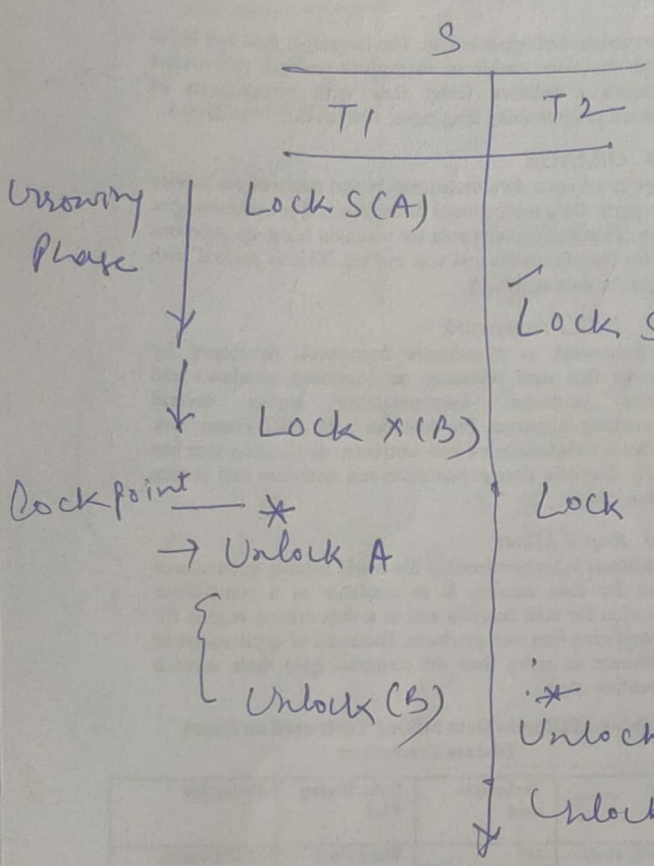
T1	
X(A)	Here we acquire locks.
S(B)	
R(A)	growing Phase
W(A)	
R(B)	
S(A)	
R(C)	
U(A) ← Shrinking Phase starts here	

By this we achive Serializability.

T1	T2
X(A)	
R(A)	
W(A)	
	don't enter into T2.
S(B)	R(A) (wait)
R(B)	
U(A) — when unlock start then shared lock on T2	
U(B)	

After that we can't acquire locks

Transactions ~~schedule~~ who are using 2PL that it is always serializable. T1 → T2



Compatibility Table

Granted \ Request	S	X
S	Yes	No
X	No	No

we can provide shared lock after shared lock

we can give because this is on D.

→ Lock point - where the ~~Trans~~ transaction's first unlock starts that point is called lock point.

Problems in 2PL :- Advantage :- Always ensures serializability.

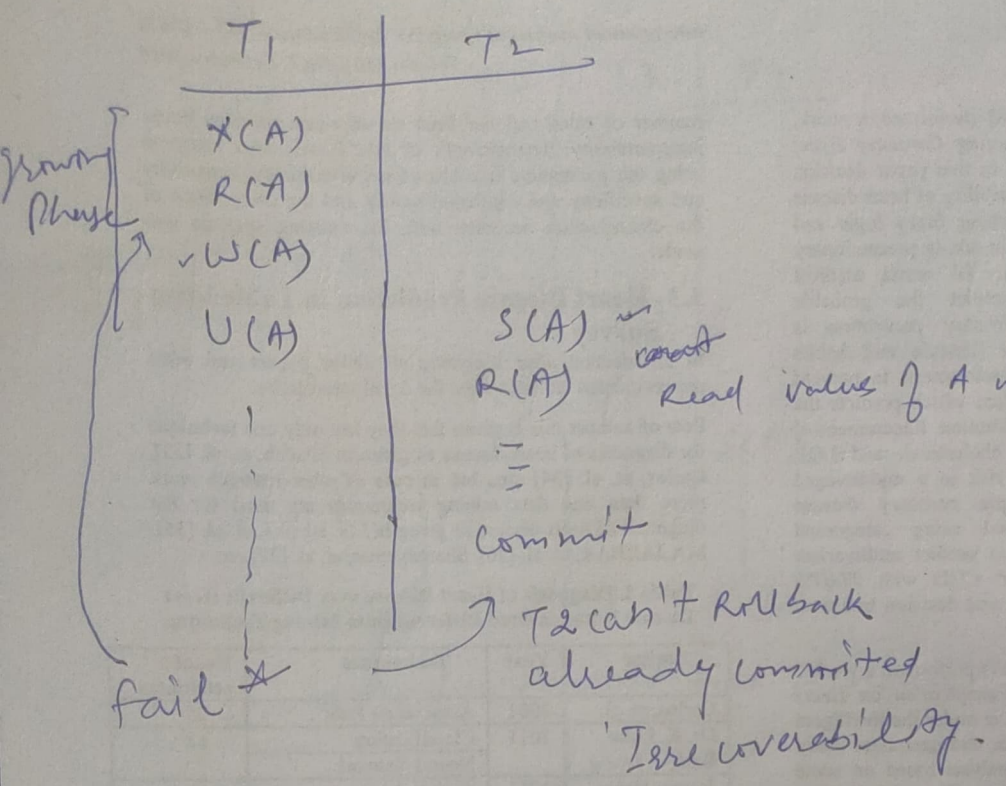
Disadvantage :- ① May not free from irrecoverability.

② Not free from deadlocks.

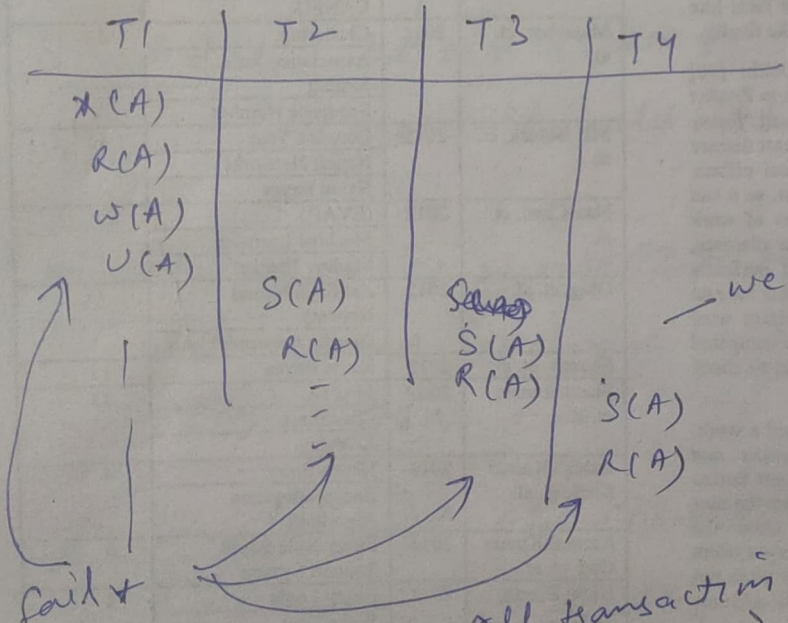
③ Not free from starvation.

④ Not free from Cascading Rollback.

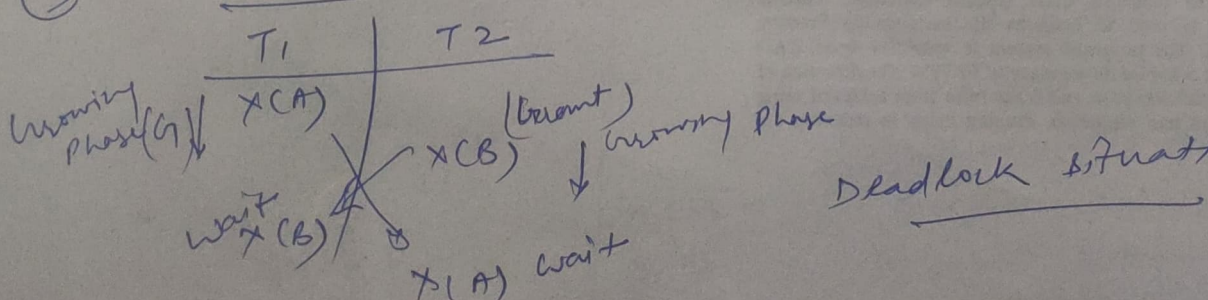
① May Not free from Irrecoverability.



② free from Cascading Rollback :-



③ Not free from Deadlocks :-



(4) Not free from Starvation:-

T1	T2	T3	T4
	S(A)		
Exclusive wait → X(A) (lock not possible)			
		S(A)	
	U(A)		
			S(A)
		U(A)	
			U(A)

T1 transaction has to wait till transaction T4 unlock the Data item.

It will take long time to wait.

Extention of 2PL

Strict 2PL : It should satisfy the basic 2PL and all exclusive locks should hold untill commit/Abort.

Rigorous 2PL :- It should satisfy the basic 2PL - all shared, exclusive locks should hold untill Commit/Abort.

This will resolve the problem of irrecoverability & Cascading Roll back.

Eg :-

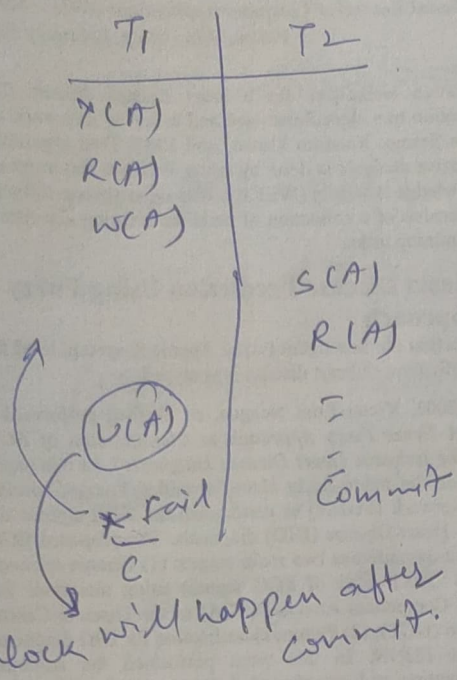
T1	T2	T3
X(A)		
R(A)		
W(A)		
U	S(A) = R(A)	
		S(A) = R(A)

T1	T2
X(A)	
R(A)	
W(A)	
Commit U(A)	S(A) R(A)

Strict 2PL says Exclusive locks should hold untill Commit/Abort.

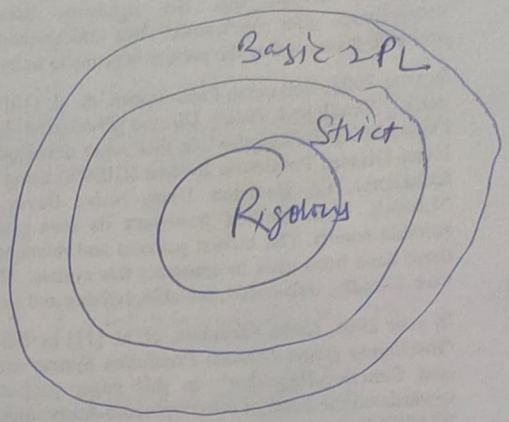
Before Unlock we have to wait Commit/Abort. the transaction

Recoverability

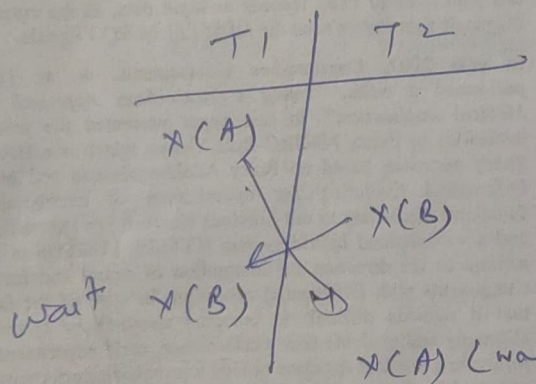


Advantages

→ Cascadeless
→ Strict Recoverable



Disadvantage — Not free from Deadlock & Starvation



(3) Conservative 2PL — Before start any transaction it will occupy all the locks that Transaction T1 wants. for this prediction Algorithm is to be used. It will solve the problem of Deadlock. But in DBMS it is not possible to predefined what locks we require in start/advance.

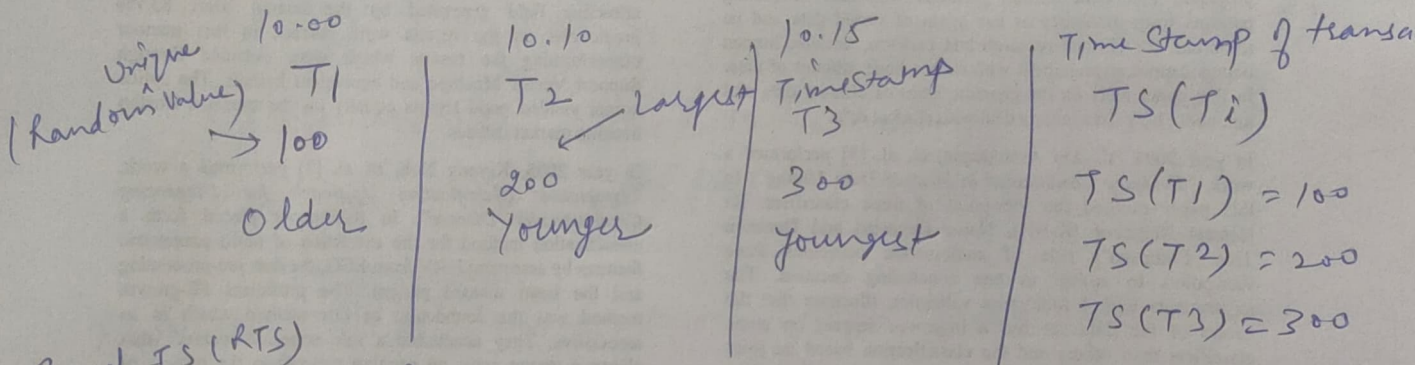
Time stamp Ordering Protocol :-

→ Time stamp is Unique value assign to every transaction

→ Tells the Order (when they enters into system)

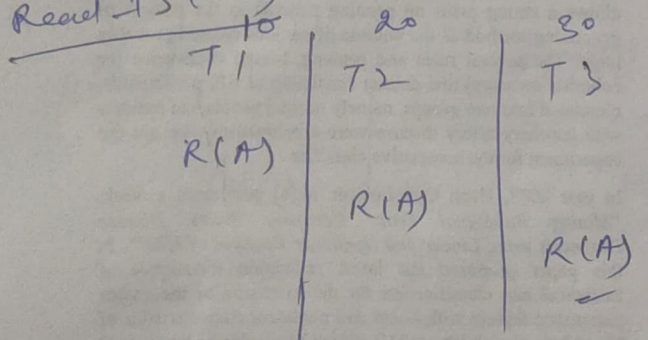
→ Read_TS (RTS) = Last (latest) transaction no. which performed Read successfully.

→ Write_TS (WTS) = Last (latest) transaction no. which performed write successfully.



Latest means last time stamp

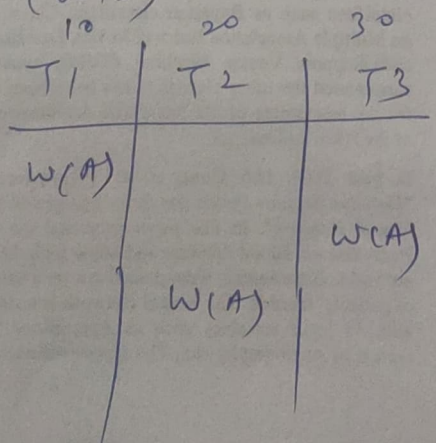
Read_TS (RTS)



RTS(A) = 30

Read Opn.

Write_TS (WTS)



WTS (A) = 20

Rules :-

(1) Transaction T_i issues a Read(A) op^n

a) if $WTS(A) > TS(T_i)$, Rollback T_i

b) Otherwise execute R(A) operation

Set $RTS(A) = Max\{RTS(A), TS(T_i)\}$

(2) Transaction T_i issues a write(A) op^n

a) If $RTS(A) > TS(T_i)$ then Rollback T_i

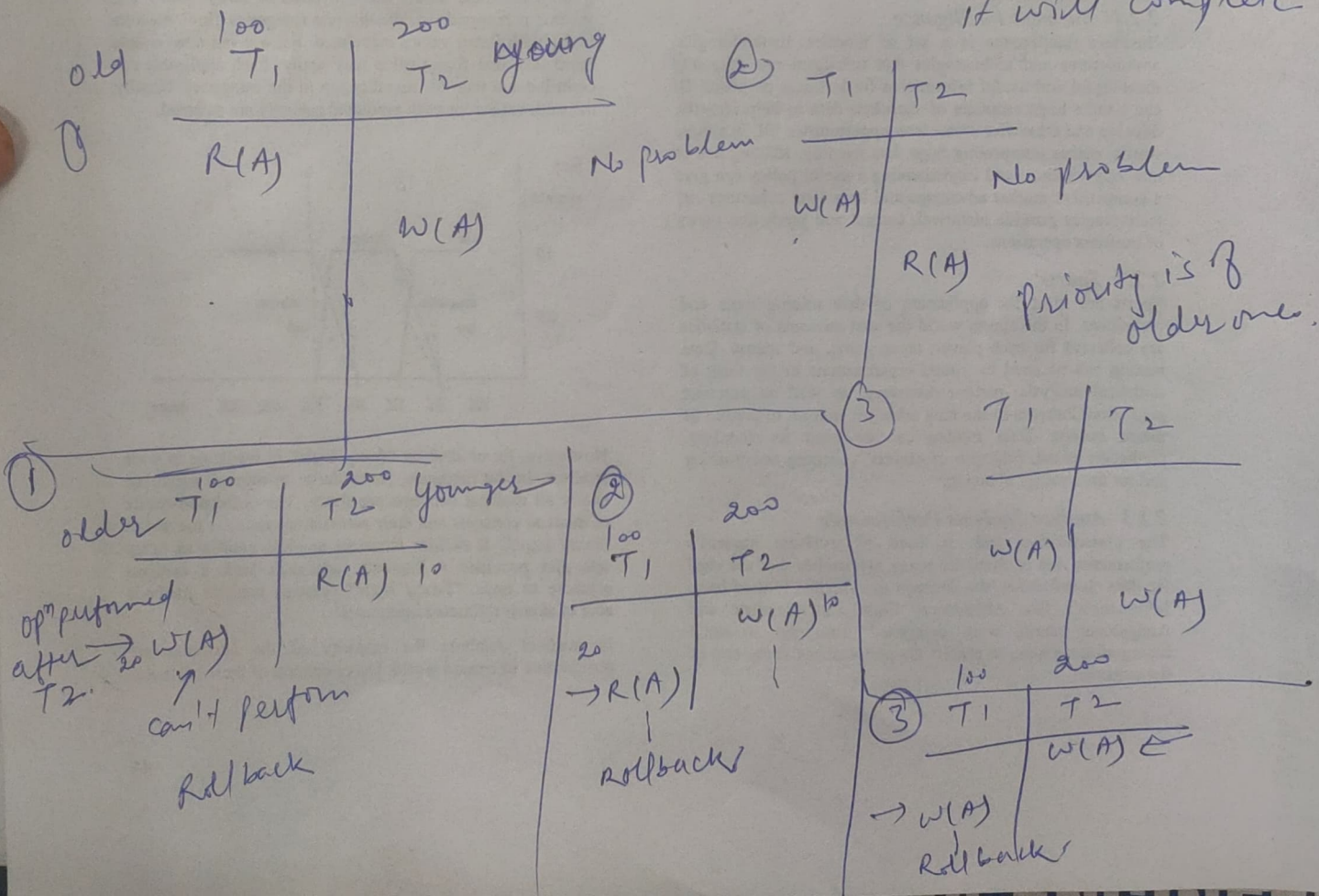
b) if $WTS(A) > TS(T_i)$ then Rollback T_i

c) Otherwise execute write(A) operation

Set $WTS(A) = TS(T_i)$

Assumption

Remember :- Older transaction comes first, then first it will complete



Q-1 Time Stamp Ordering Protocol — 1st oldest will execute

(100) oldest T1	(200) T2	(300) T3 youngest
R(A)		
	R(B)	
W(C)		
		R(B)
R(C)		
	W(B)	
		W(A)

	A	B	C
RTS	~~100~~ 0	~~200~~ ~~300~~ 0	~~0~~ 100
WTS	~~0~~ 300	0	~~0~~ 100

— Initial zero-zero.

WTS(A) TS(T_1)

Rule 1(a) 0 > -100 — false move to otherwise conditions

0 > 100 R(A) — Rule 1(a) — (b) set RTS(A) = Max(RTS(A), TS(T_1)) = Max(0, 100)

0 > 200 R(B) — Rule 1(a) — WTS(B) TS(T_2) — 0 > 200 false — RTS(B) = Max(RTS(B), TS(T_2)) = Max(0, 200) update - 200

0 > 100, 0 > 100] W(C) — Rule 2(a) > 200 (a) 0 > 100 false (b) WTS(C) > TS(T_3) 0 > 100 (c) set WTS(C) = TS(T_3) update C(0) to 100

0 > 300 R(B)

100 > 100 R(C) — false — Rule 2(a)

300 > 200] W(B) — T_2 → Rollback — Restart after T_3. Neglect all the values of T_2 transactions Restart after T_3.

100 > 300, 0 > 300] W(A)

(2)

Multiple Granularity Protocol :-

Granularity means hierarchy of the level or different levels of data.

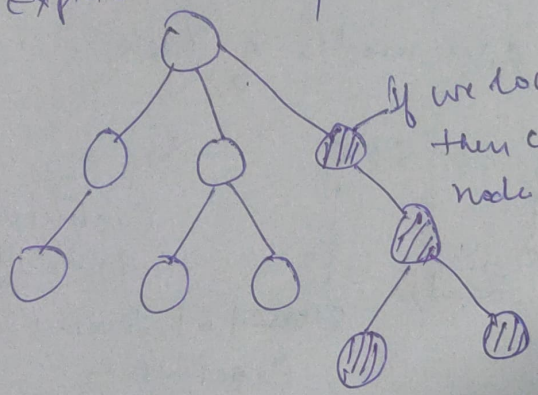
Hierarchy of Database :-

1. Database
2. Tables
3. Attribute
4. Tuples

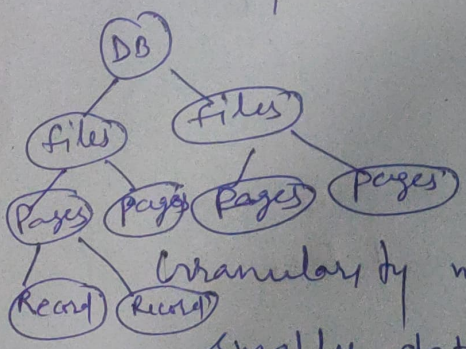
How to ~~Con~~maintain concurrency control with the help of Granularity.

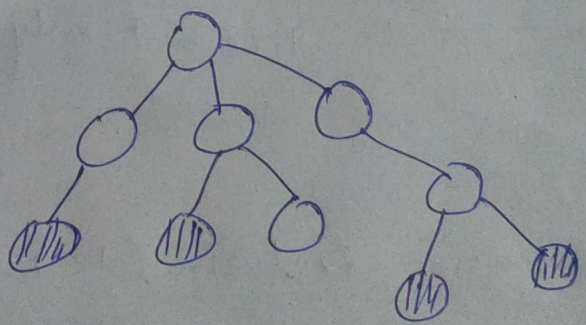
Lock Concept :- Two types of Lock :- Explicit & Implicit

Explicit locking



Implicit Locking :- we lock only leaf nodes in Explicit locking.



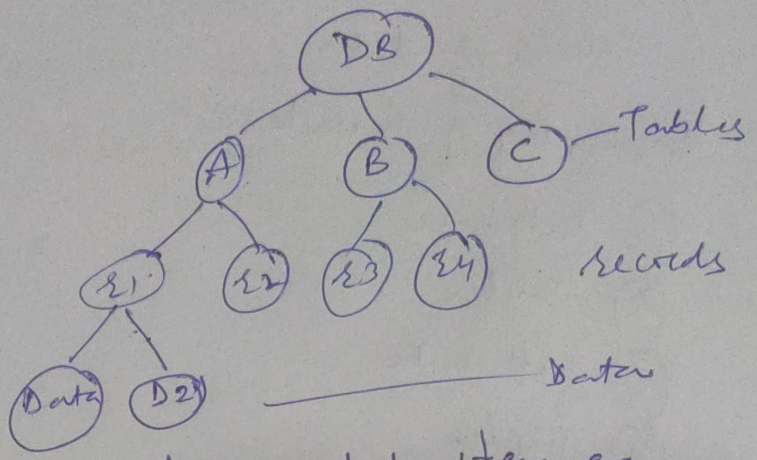


Granularity means different level of data where smaller data items are listed inside bigger.

We use this granularity to ensure concurrency.

We have various levels of data sizes.

→ Data items of different size. We represent this in the form of tree structure.



Course Granularity :- refers to large data item eg. entire relation or a database.

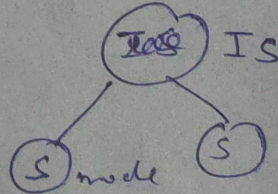
Fine Granularity :- Refers to a small data items eg - tuples or attributes.

Locks Modes :- Total five modes of lock.

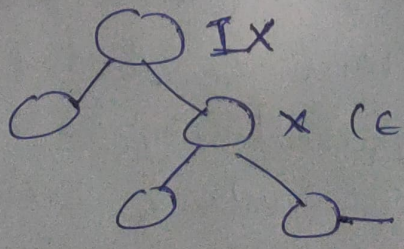
IS | IX | S | SIX | X

IS — Intension share
IX — Intension Exclusive
S — Share (Read)
SIX — Shared & Intension Exclusive
X — Exclusive (Read/write)

(1) Intension share (IS) :- explicit locking at lower level of tree but only with shared lock.

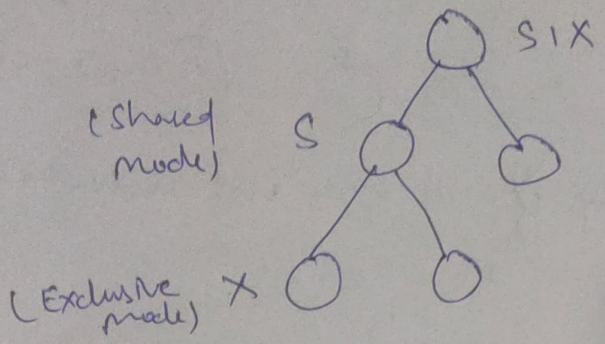


(2). Intension Exclusive (IX) - explicit locking at lower level with exclusive or shared lock.



(3) Shared & Intension Exclusive (SIX) –

Subtree routed by that node is locked explicitly in shared mode & explicit locking is done at lower level with exclusive mode.

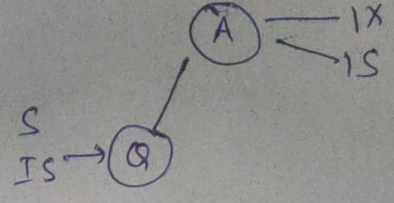


Compatibility Matrix :-

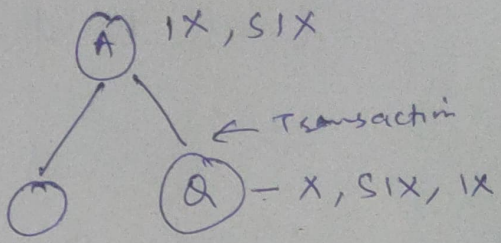
	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	X
IX	✓	✓	X	X	X
S	✓	X	✓	X	X
SIX	✓	X	X	X	X
X	X	X	X	X	X

Rules :- So a transaction Ti that attempts to lock a node Q must follow these rules :-

(1) must observe the lock compatibility
(2) Root of the tree must be locked first, and it can lock in any mode.
(3) A node Q can be locked by Ti in S OR IS mode only if the parent of Q is current locked by Ti in either IX OR IS mode.

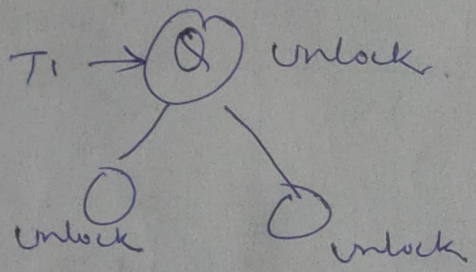


④ A node Q can be locked by T_i in X, SIX or IX mode only if the parent of Q is currently locked by T_i in either IX or SIX mode.



⑤ T_i can lock a node only if it has not unlocked any node so far.

⑥ T_i can unlock a node Q only if none of the child of Q is currently locked by T_i.



Locking: Root to leaf

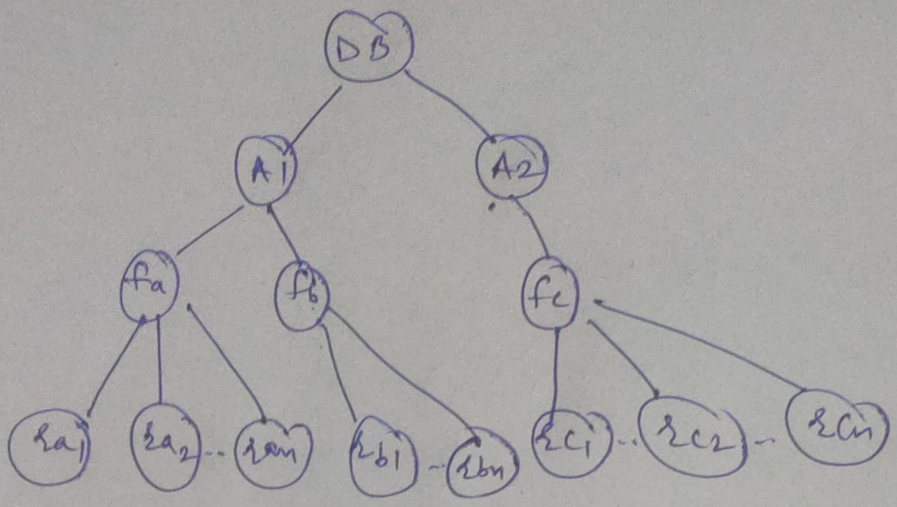
Unlocking: Leaf to Root

Thomas write Rule:-

T_1	T_2
	W(Q)
Reject * W(Q) only reject	

Q = ~~10~~ ~~20~~ 30

Ex:1



a) T_1 reads r_{a_2} (DB, A1, fa → IS || r_{a_2} → S)

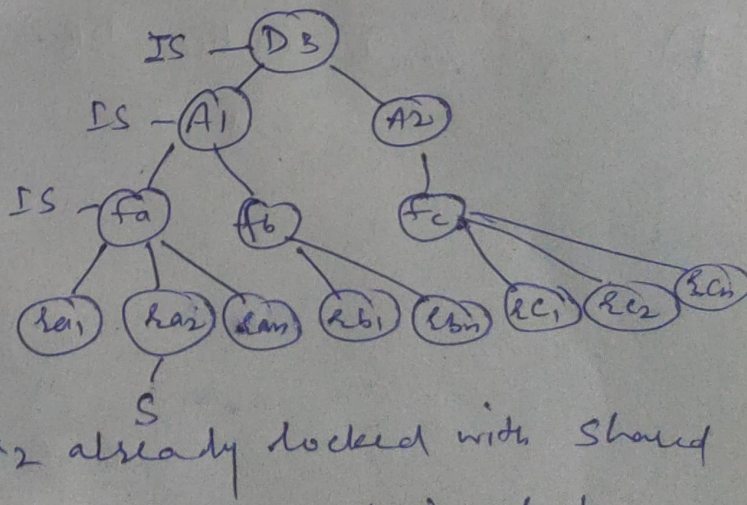
b) T_2 modifies r_{a_2} ✗ Not allocated exclusive

b) T_2 modifies r_{a_3}

c) T_3 reads all records on fa. → check for compatible → feasible table.

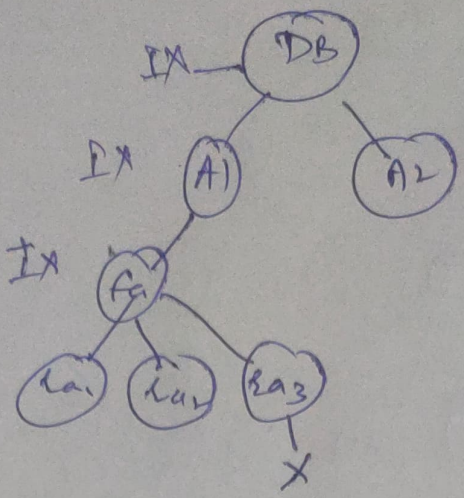
d) T_4 reads entire DB. → feasible

(1) T_1 reads r_{a_2} - r_{a_2} wants to read - To obtain a shared lock. But we know it travels from Root to Node. It starts from DB.

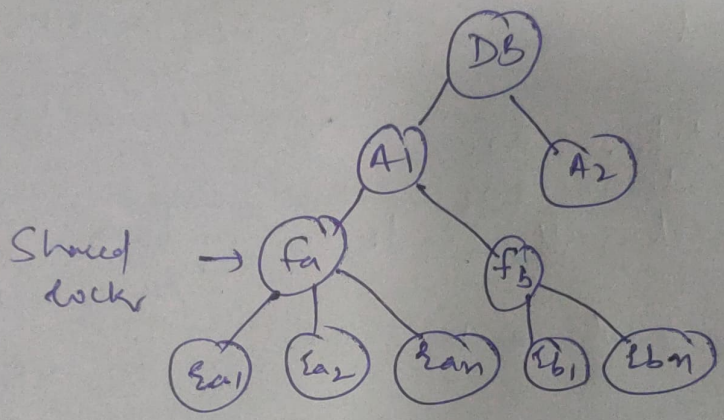


(2) T_2 modifies r_{a_2} - r_{a_2} already locked with shared lock. Modifies r_{a_2} means exclusive lock. Exclusive lock will not compatible with shared lock. It would n't be obtained. Transaction will rollback.

(3) Assumes r_{a_3} - write r_{a_3} - exclusive lock. check at upper level.. compatibility at upper level. IS is not compatible with X. IX lock is compatible

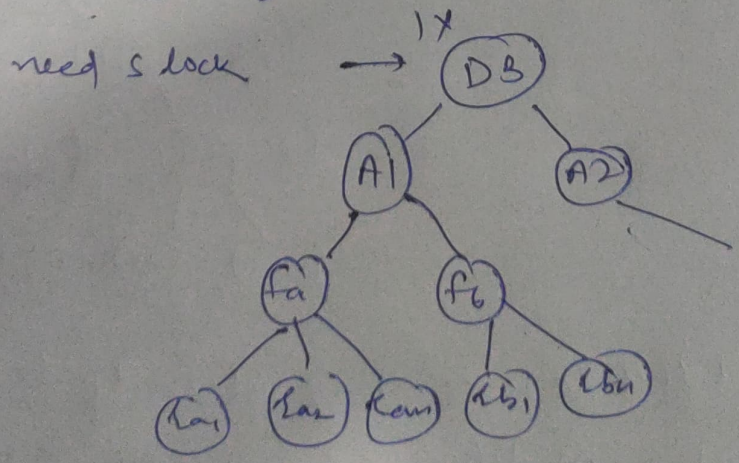


(c) T_3 reads all records on f_a.



Shared lock is not compatible with IX.
Shared lock can not be granted here because it is already granted IX lock.

(d) T_4 reads entire DB.



Shared lock is not compatible with IX.
We can not granted shared lock on DB.