Week -10

31/3/24

Transaction concept:

→ unit of program execution that access and updates various data items.

ex: transaction of $50 from account A → B

1) read(A)
2) A := A − 50
3) write(A)
4) read(B)
5) B := B + 50
6) write(B)

Two main issues to deal with:

→ Failure of various kinds, such as hardware failure, system crash.

→ concurrent execution: multiple transact" in same acount at same instant

Required property of ACID

1) Atomicity: means 0 or 1
either complete transact" or no transact"
(हुआ या कि A से पैसा cut गया B में reflect नहीं हुआ)

2) Consistency:
→ A+B must unchanged by execut" of transact"
→ sum of amount before transact" equal to sum of transact" after execution.

3) ISOLATION (→ being alone) : used when two/more transact happen in some a/c at some time

→ if b/w step 3 & 6 of transact", another transact" T2 ~~happens~~ is allowed to access the partially updated database it will see inconsistent.

→ every Transact" are independent.

→ Isolat" can be ensured trivially by running transact" serially.

4) DURABILITY:

→ once user get notified that transcact" completed, the update to the database by the transaction must persist even if System Crash.

→ Someone send $10

→ my balance now $60 (earlier $50)

System Crash

→ after system updated, my a/c balance must show $60.

A — ATOMicity
(All or Nothing Transcact")

C — consistency
(A+B remain constant)

I - Isolation
(Transcact"s are independent)

D — Durability
(commited data Never lost)

TRANSACTION STATE

→ Active : Initial state

→ partially committed

→ failed

→ Aborted

↳ after transact" has been rolled back (refund)

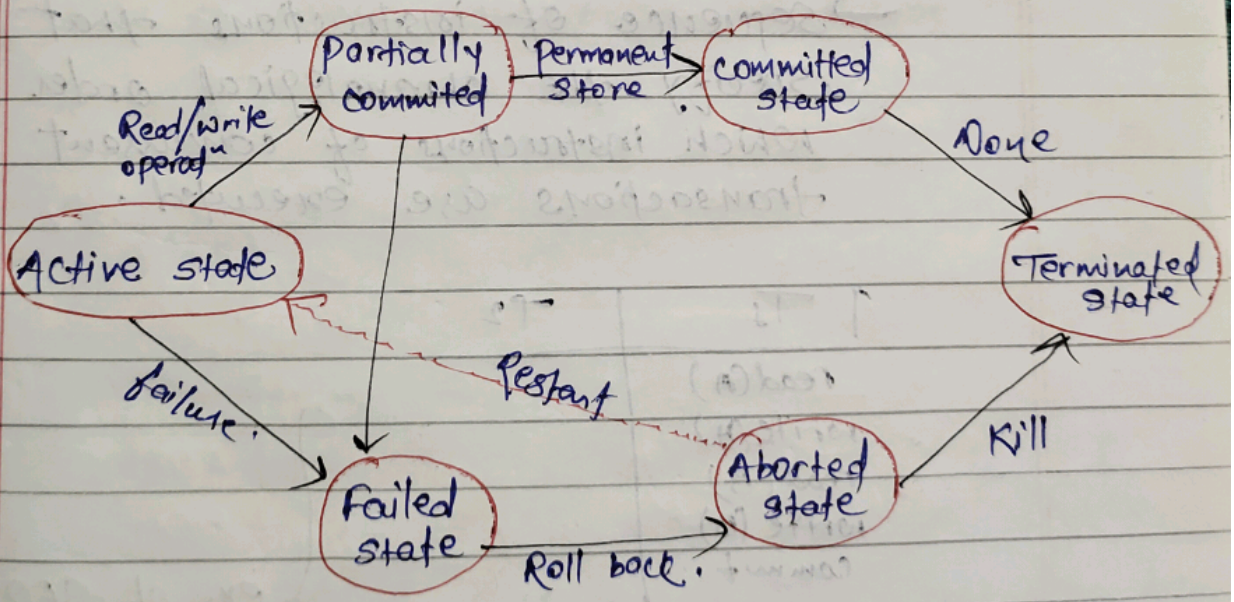
Two option after aborted :-

i) Restart the transact"

ii) Kill the transact".

→ Committed : after successful completion

→ ~~Tran~~ Terminated : after committed.



Concurrent Executions:

→ Multiple transactn are allowed to run concurrently in system. Advantages are :-

i) Increase processor & disc utilization
ii) Reduce av. response time.

Concurrency control schemes : mechanism to achieve isolation.

Schedule : two or more transactn together called schedule.

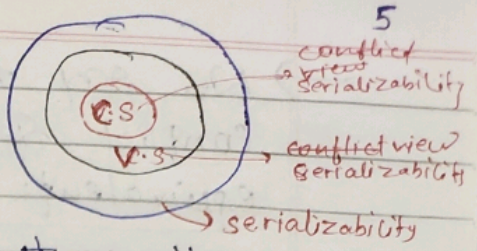
→ sequence of instructions that specify the chronological order in which instructions of concurrent transactions are executed.

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
commit	
	read(A)
	write(A)
	read(B)
	write(B)
	commit

} ex. of SERIAL Schedule

* in serial schedule if there are 'n' transactn then total n! equivalent serial schedule possible

→ Serializability
→ Conflict serializability.



Serializability: Parallel को series में convt. करना

Assumption: each transaction preserve database consistency

1) Conflict serializability
2) View serializability

} parallel को series में convt. करने के 2 method

1) CONFLICT Instruction:

→ Two transaction T_1 & T_2
→ T_1 & T_2 conflict if and only if at least one of these instructions is write

T_1	T_2
R(A)	
	R(B)

Not conflict

T_1	T_2
R(A)	
	W(B)

conflict ✓

T_1	T_2
W(A)	
	R(A)

conflict ✓

T_1	T_2
W(A)	
	W(B)

conflict ✓

CONFLICT Serializability → only if graph is acyclic

↳ concept that ensure transactions can be executed in a way that maintains the same result as if they were executed one after another.

eg)

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

≡

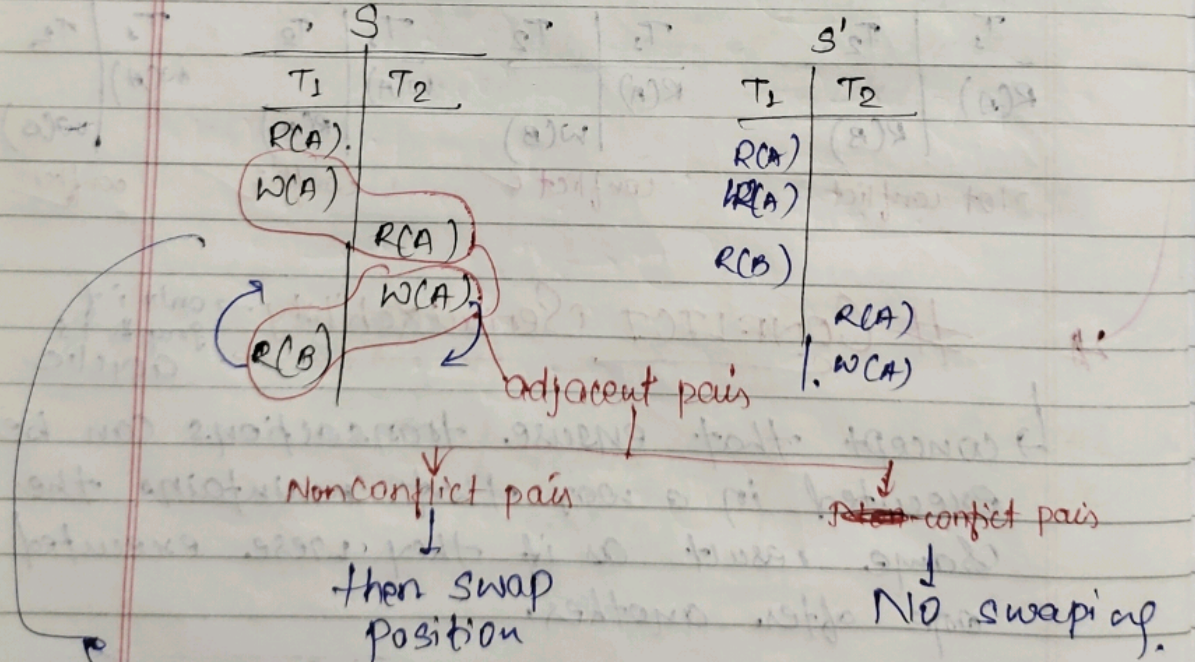
T_1	T_2
R(A)	
W(A)	
	R(B)
	W(B)

→ We say that a schedule S in conflict serializable if it is conflict equivalent to a serial schedule.

R(A) R(A) } Non-conflict pairs

R(A) W(A)
W(A) R(A)
W(A) W(A) } conflict pairs

R(B) R(A)
W(B) R(A)
R(B) W(A)
W(A) W(B) } Non-conflict pairs



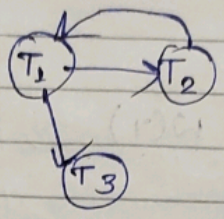
in
conflict
le.

Q

S

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

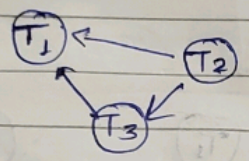
Check whether schedule is conflict serializable or not?



अगर graph में loop आ गया तो means Non-conflict serializable.

जिस जिस pair में conflict pair बना रहा है उसे → से Join कर दो

Ex



Indegree = 0

$T_2 \rightarrow T_3 \rightarrow T_1$

No loop/cycle
↓
Conflict serializable
↓
serial
↓
consistent

जब loop आ जाता है तब Serializability Check करने के लिए View Serializability use करेंगे

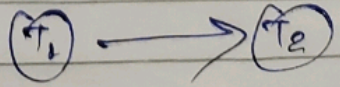
T_1	T_2	T_3
100 R(A)	A=A-40	
	W(A) — 60	
A=A-40		
20 W(A)		
		A=A-20
		W(A)
		0

≃

T_1	T_2	T_3
100 R(A)		
60 W(A)		
	20 W(A)	
		0 W(A)

final result is equal
so both relat" are equivalent

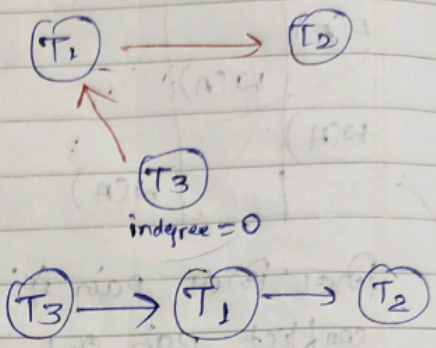
T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(A)	
W(B)	
	R(B)



No cycle → conflict serializable

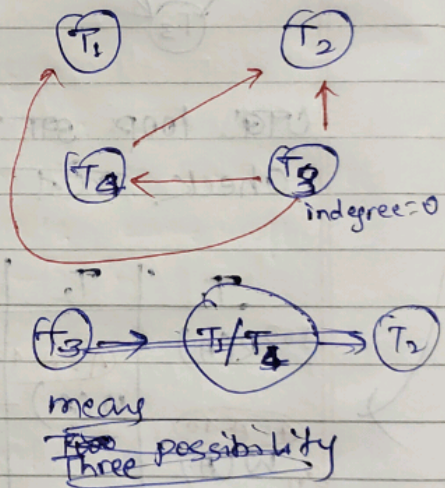
Q.

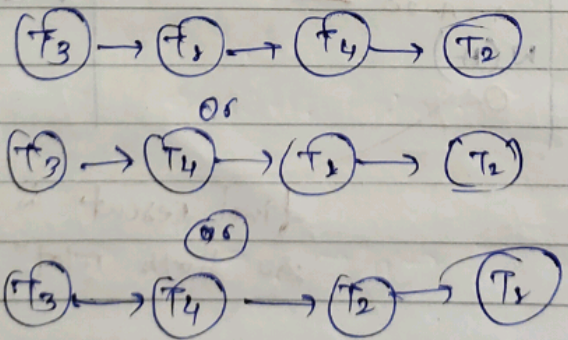
T_1	T_2	T_3
		R(Y)
R(X)		
W(X)		
	R(X)	
		W(Y)
R(Y)		
W(Y)		
	W(X)	



Q.

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





Blind write : if there is no read happens prior to first write then it is said to be blind write.

T_1	T_2	T_3	
	$R_2(x)$		→ $W_3(x)$ is blind write because there is no $R_3(x)$ before $W_3(x)$
$R_1(x)$			
		$W_3(x)$	
	$W_2(x)$		→ $W_2(x)$ is also blind write

Lec 10.3

VIEW SERALIZABLE

→ let S & S' be two schedule with same set of transaction. S & S' are view serializable if following cond" holds:

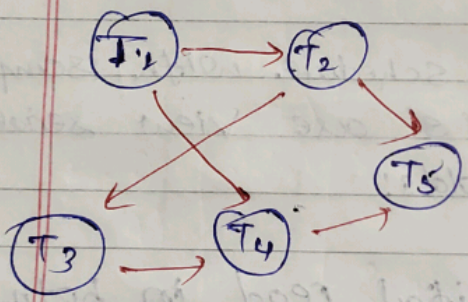
1) Initial read : initial read in both S & S' should be same.

2) Final write : final write of each in both S & S' should be same

3) Write Read pair : read perform by transact" in both S & S' correspond to same write.

PA-10
Q-4

T_1	T_2	T_3	T_4	T_5
				R(Z)
W(Y)				
	R(Y)			
		W(Y)		
			R(Y)	
	W(P)			
				R(P)
			W(X)	
R(Q)				
				R(X)
				W(Y)



→ acyclic

$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5$

possible ~~serial~~ conflict Serializ~

for view serializ~

Check : i) initial read ii) final write iii) write-read pair

i) write-Read pairs :

W-R pair → W(Y) → R(Y) ($T_1 \rightarrow T_2$)

→ W(Y) → R(Y) ($T_3 \rightarrow T_4$)

$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5$

or

$T_3 \rightarrow T_4 \rightarrow T_1 \rightarrow T_2 \rightarrow T_5$

→ 2 view serializability

RECOVERY:

→ Recoverability is not possible if a transact" T_2 reads the updated item of T_1 another transact" and perform commit before T_1

→ जिसने पहले read(A) किया वही पहले commit करे

Cascading Rollback: A single transact" leads to failure of another transact".

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

If T_1 fail then T_2 will also fail → T_3 will also fail.
T_0

Cascadeless: to achieve cascadeless recovery a transact" T_1 must be committed before any other transact" reads the data written by T_1.

Cascadeless >> Cascading

→ every cascadeless is also recoverable.

used with DML command

Transact" Control Language (TCL)

→ COMMTT → to save changes

→ ROLLBACK → to roll back the changes

→ SAVE POINT → create point within group

→ SET TRANSACT" → place a name on transact'

Q. Check whether schedule is view serializable or not

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

First read - T_2
last write - T_3

$T_2 \rightarrow T_1 \rightarrow T_3$

Here there is no write-read combinat"

lec. 10.4

LOCK-BASED-PROTOCOL:

→ lock is mechanism to control concurrent access to data item.

1) Exclusive (X) lock: Both read + write

2) Shared (S) lock: only read

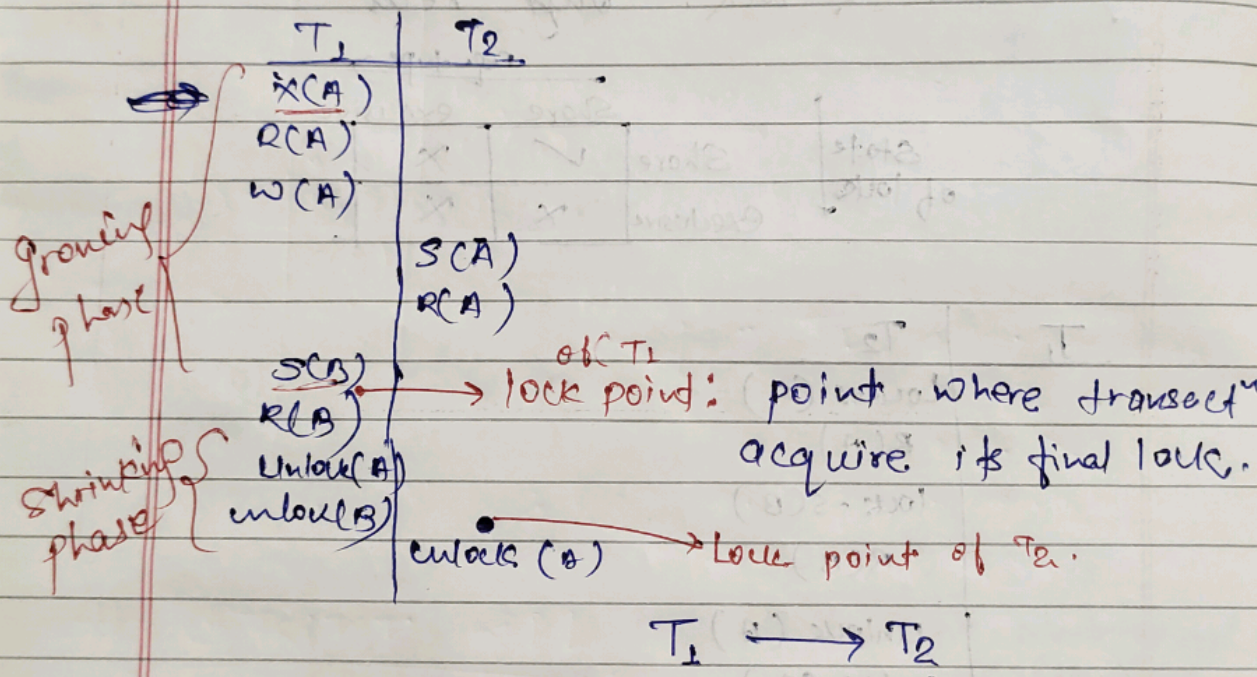
State of lock \ req. type	Share	exclusive
Share	✓	✗
exclusive	✗	✗

T_1	T_2
	lock-S(A)
	R(A)
	lock-S(B)
	R(B)
	unlock(A)
	unlock(A)
lock-X(A)	
R(A)	
W(A)	
unlock(A)	

Two-Phase Locking protocol (2PL)

1) GROWING PHASE :- locks are acquired and no locks are released.

2) SHRINKING PHASE :- Lock are released & no lock are acquired.



→ Those Schedule which follow 2PL are always conflict Serializable.

→ after shrinking phase, growing phase is not possible

→ Growing phase is compulsory but shrinking phase is optional.

DEADLOCK:

→ occurs when two/more transact^{n} are unable to proceed because each is waiting for the other to release resource.

T_1	T_2
lock-X(A)	lock-X(B)
R(A)	R(B)
W(A)	W(B)
T_1 want to update B	T_2 want to update A.

Here both T_1, T_2 are waiting for each other to give unlock this is deadlock.

→ if we donot unlock data item before requesting a lock on another data item, deadlock occurs

→ deadlock is preferable than inconsistance state

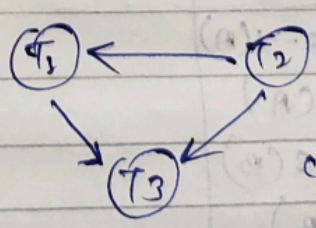
Q. Schedule S : is given:

a) ✗ All conflict serializable shedule are 2P lock (vice versa is true)

b) ✓ give S is conflict serializable

c) ✗ given S is 2P locakable

T_1	T_2	T_3
	W(P)	
W(P)		
		W(P)
	W(Q)	
W(Q)		



→ acyclic conflict serializable

$T_2 \rightarrow T_1 \rightarrow T_3$

T_1	T_2	T_3
	X (P)	
	W(P)	
	X (Q)	
	U (P	
X(P)		
W(P)		
U(P)		
		X-(P)
		W (P)
		U (P)
	W(Q)	
	U(Q)	
X(Q)		
W(Q)		
U(Q		

Growing phase

Shrink

growing

for T_1, after ~~Growing~~ Shrinking phase one more growing phase came. ✗ ✗

not follow 2PL

STRICT 2PL.

↳ follow 2PL and all exclusive lock should unlock after commit operation.

T_1	T_2
lock-X(A)	
W(A)	
lock-S(B)	
R(B)	
commit	
unlock(A)	
	lock-X(A)
	W(A)
	commit
	unlock(A)
Unlock(B)	

Grow

Shrink

grow

shrink

⟶ Yes it 2PL + Strict 2PL.

Rigorous 2PL.

↳ follow 2PL
↳ all lock (exclusive / shared) should unlock after commit.

lock manager:

↳ process to which a transact^n sends lock and unlock request.
↳ reply a lock request by sending message asking transact^n to roll back, (if deadlock happ)

Deadlock Timestamp:

→ each transact" that enters the system is assigned a unique time stamp, typically based on order they arrive.
  (1) System clock
  (2) logical counter (1, 2, 3 ...)

Wait - Die Scheme

↳ non-preemptive technique

↳ older transact" may wait for younger one to release data (older means small timestamp)

ex Suppose T_5, T_{10}, T_{15} have time stamp 5, 10, 15 resp.
T_5 → older one (small timestamp)

then,

→ if T_5 request a data heald by T_{10} then T_5 will wait.

→ if T_{15} request data held by T_{10} then T_{15} will be killed

Wound - wait scheme

↳ preemptive technique

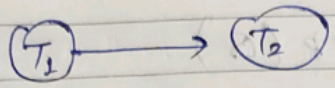
→ younger transact" may wait for older one

in last ex:

→ if T_5 request a data held by T_{10} then T_{10} will be wounded.

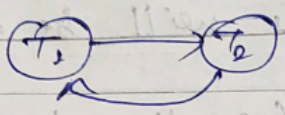
→ if T_{15} request data held by T_{10}, then T_{15} will 'wait'.

Deadlock detection



means T_1 is waiting for T_2 to release data.

→ System is in deadlock if cycle occurs.



Both are waiting for each other to release data.

For deadlock Recovery:

Some transact" have to rolled back to break deadlock.

this is done Through:

Time stamp - Based Protocol.

→ W-timestamp(Q) ← largest time taken by any transact" to execute W(Q) sucessfuly.

→ R-timestamp(Q) ← largest — — — — — — — — — — to execute R(Q)

Case I: If T_1 issues a read operat" R(Q) then:

$TS(T_1) \leq$ W-timestamp(Q)

then T_1 will be rejected.

$TS(T_1) \geq$ W-timestamp(Q)

↳ then T_1 will execute

Case-II T_1 issues $W(Q)$ then:

→ If $TS(T_1) < R\text{-timestamp}(Q)$
↳ T_1 will be rejected

→ If $TS(T_1) < W\text{-timestamp}(Q)$
↳ T_1 will be rejected

→ If $TS(T_1) \geq R/W\text{-timestamp}(Q)$
↳ T_1 will be executed

Q) S: $R_1(A), R_2(B), W_2(A), R_1(A)$

TS for T_1 & T_2 are 1, 2 respectively.

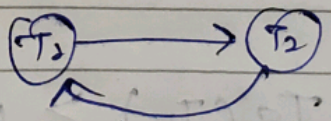
a) T_1 need to roll back $TS(T_1) = 1$
b) T_2 need to roll back. $TS(T_2) = 2$

$R_2(A) = 0$

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

$W_2(A) = 2$
$1 < 2$
roll back.

	A	B
W-TS	2	
R TS	1	1



deadlock occurs one transject need to roll back: we have to find which one to roll back.