JB Sir

Ch-6 Transaction

Page :
Date : / /

Database operation Vs Transaction

Database operation → Insert, Update, delete

Transaction → Business operations
↓
Funds transfer, New Account opening

= "A transaction is a set of DML operation that performs a Bussiness unit of work."

Ex: Transfer 50$ from account A to B

1) Read (A)
2) A = A - 50
3) write (B)
4) Read (B)
5) B = B + 50
6) write (B)

DML - Data manipulation language

How transaction Begins and How it ends?

→ Transation Begins = when we execute the first DML statement after opening the session

Transaction End : Txn ends with COMMIT or ROLLBACK or DDL or DCL.

⇒ There are 2 type of txn.
(i) Read only (only select)
(ii) Read write (insert, update, delete and select are allowed)

Transaction Properties :-

A → Atomicity
C → Consistency
I → Isolation
D → Durability

Atomicity :- Either all txn (operation) Perform or not any txn Perform.

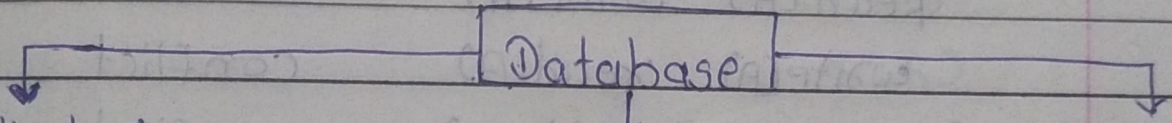
Page :
Date : / /

Consistancy :- when the txn start and end the sum of money have to be equal. should

Isolation :- Even T_1 and T_2 started concurntly the end result should be match at least one T_1 or T_2.

Durability : once txn completed the data should be permantly saved.

Concurrency and Concurrency Control :-



clint = 1
User : SCOTT
Id = 1

clint = 2
User = SCOTT
Id = 2

clint = 3
User = SCOTT
Id = 3

Insert into emp values (1, 'a')
Transaction "T1" started
Select * from emp;
1 row (1, a)
Commit;
Txn "T1" completed
Select * from emp;

Inser into emp values (2, 'b')
Transaction "T2"..
Select * from emp
1 row → (2, b)
2 row (1, a) (2, b)

Insert into emp values (3, "c")
Transaction "T3"...
→ (3, c)
3 row (1, a) (3, c)

↳ oracle use Locking to control to concurrency. (for conflicting op.)

Two actions are said to Conflict if

1.) They belong to diff. transaction
2.) Both op. refers to the same db obj.
3.) one of them is write.

Ex:

T1	T2	CONFLICT or NOT
Read(A)	Read(A)	NO conflict
Read(A)	write(A)	conflict
write(A)	read(A)	conflict
write(A)	write(A)	conflict.

Dirty Read Problem:-

Ex: T1 → Transfers 500₹ from A to B
T2 → Add 10% interest to A and B

initial value
A = 1000
B = 2000

Page :
Date : / /

Serial Execution : TI and then T2

TI	T2
A = 1000	
B = 2000	
R(A)	
W(A)	
R(B)	
W(B)	
A = 500	
B = 2500	
	R(A)
	W(A)
	R(B)
	W(B)
	A = 550
	B = 2750

Serial Execution :- T2 then TI

TI	T2
	A = 1000
	B = 2000
	R(A)
	W(A)
	R(B)
	W(B)
	A = 1100
	B = 2200

R(A)

W(A)

R(B)

W(B)

A = 600

B = 2700

Concurrent Execution :- T1 & T2

T1	T2
A = 1000	
B = 2000	
R(A)	
W(A)	
A = 500	
B = 2000	
	R(A)
	W(A)
	R(B)
	W(B)
	A = 550
	B = 2200
R(B)	
W(B)	
A = 550	
B = 2700	

↳ Concurrent Execution is not a Serializable schedule.

Unrepeatable Read [RW conflicts]

T1 = Reads 'A' two times & assigns to B and C

T2 = Increment two times value of A

A = 5

Serial Execution

T1 → T2

T1	T2
A = 5	
R(A)	
W(A)	
R(A)	
W(C)	
A = 5	
B = 5	
C = 5	W(A)
	W(A)
	A = 7
	B = 5
	C = 5

T2 → T1

T1	T2
	A = 5
	W(A)
	W(A)
	A = 7
R(A)	
W(B)	
R(A)	
W(C)	
A = 7	
B = 7	
C = 7	

Page :
Date : / /

Concurren Execution

T1 & T2

T1	T2
R(A)	
W(B)	
B=5	
	W(A)
	W(A)
	A=7
R(A)	
W(C)	
C=7	

A = 7
B = 5
C = 7

↳ same Acc. Value must be same if it differ then it called unrepetable....

Blind Write Proble (Lost update):-

Ex: Akbar [A] and Birbal [B] are 2 employee and Trans. T1 & T2 wants to equal their salaries.

T1 → Equals A & B salary to 1111 ₹

T2 → Equals A & B salary to 2222 ₹

A = 0

B = 0

Serial Execution :-

T1 → T2	T2 → T1
A = 0	A = 0
B = 0	B = 0
R(A)	R(A)
W(A)	W(A)
R(B)	R(B)
W(B)	W(B)
A = 1111	A = 2222
B = 1111	B = 2222
R(A)	R(A)
W(A)	W(A)
R(B)	R(B)
W(B)	W(B)
A = 2222	A = 1111
B = 2222	B = 1111

Page :
Date : / /

A Concurrent :-

T1	T2
A = 0	
B = 0	
R(A)	
W(A)	
A = 1111	
B = 0	
	R(B)
	W(A)
	R(B)
	W(B)
	A = 2222
	B = 2222
R(B)	
W(B)	
A = 2222	
B = 1111	

⇒ more then one updates occured it lost the privous one updats so it also called Lost update problem.

Guidlines to overcome all the 3 anomalies:-

1) Every Transsation must read to committed Data only [No Dirty Read]

2) If transaction "TI" read the committed data then no other Txn. should modify the data as long as "TI" is in progress [No unrepenteible Read]

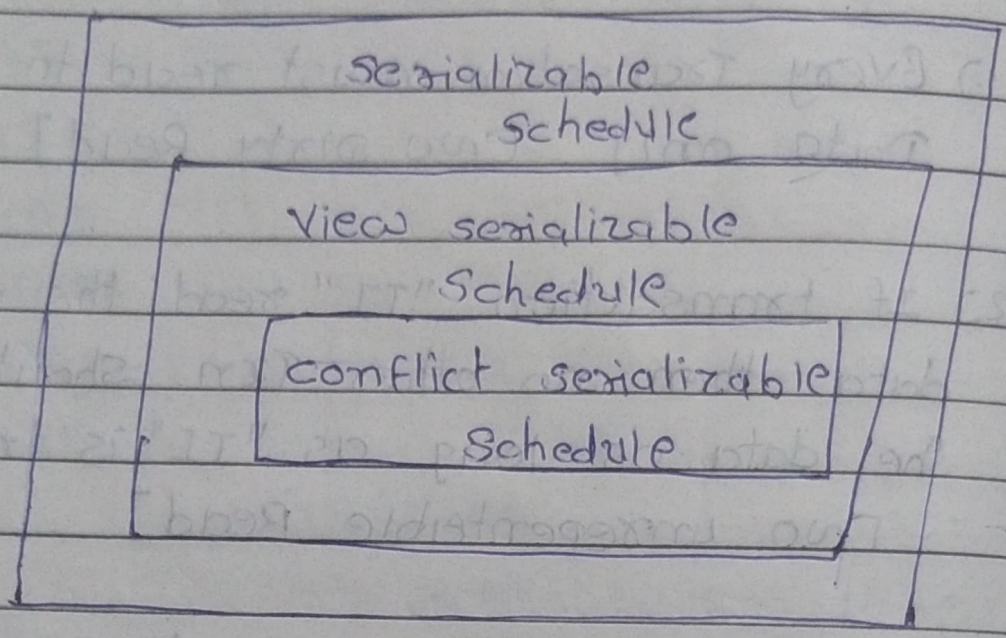
3) if Txn "TI" has modified any data, then no other Txn is allowed to modify the data again as long as "TI" is in progress [No Blind Write]

Testing for Serializabilty:-

→ A schedule 's' is said to be serializable, if and only if the net effect is identical to some complete serial schedule over 's'

→ As its difficult to verify wheter the given schedule

All schedules :-



1.) Conflict Serializability :-

(प्रीसीडन्स)

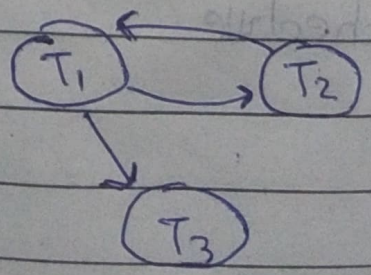
1.) draw a Precedence graph :-

(1.) A node for each commited txn in S.

(2.) An arc from T_i to T_j if an action T_i precedes and confilt with one of TJ's actions

(3) A schedule 's' is conflict-serializable, if and only if, the graph is acyclic.

T_1	T_2	T_3



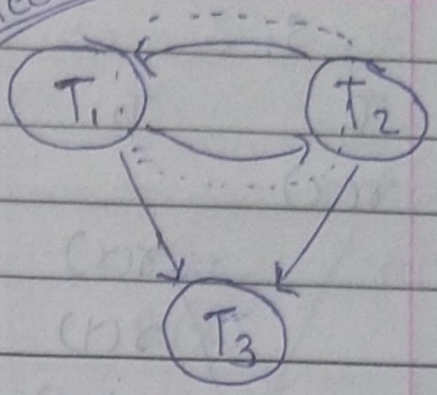
if cycle occurs :- not conflict
else :- conflict

Page :
Date : / /

Ex.

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

Tried :-



↳ comare one Txn with every other Txn for finding conflict or not.

↳ Cycle is formed ($T_1 \rightarrow T_2$) → not conflicting serializable

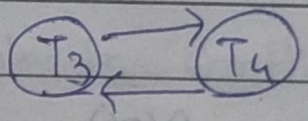
Ex:

T_3 T_4

R(Q)

W(Q)

W(Q)



not-conflict serializable

Ex:

S_1 T_1 T_2

$r_1(x)$

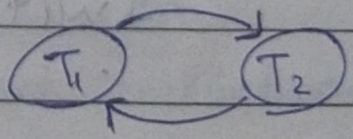
$r_1(y)$

$r_2(x)$

$r_2(y)$

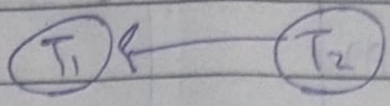
$W_2(y)$

$W_1(x)$



S_2

T_1	T_2
$r(x)$	
	$r(x)$
	$r(y)$
	$w(y)$
$r(y)$	
$w(x)$	

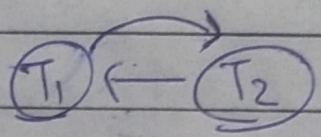


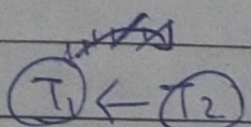
$S_1 \rightarrow$ not conflict serializable

$S_2 \rightarrow$ conflict serializable

Ex: S_1

T_1	T_2
$R(x)$	
	$R(x)$
	$R(y)$
$W(x)$	
$W(y)$	
	$W(y)$



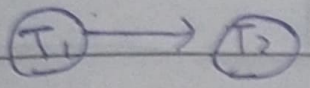


S_2

T_1	T_2
$R(x)$	
	$R(x)$
	$R(y)$
$W(x)$	
	$W(y)$

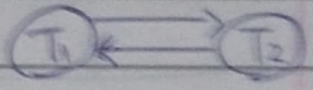
S_3

T_1	T_2
R(x)	
W(x)	
	R(x)
W(y)	
	R(y)
	W(y)



S_4

T_1	T_2
	R(x)
	R(y)
R(x)	
W(x)	
W(y)	
	W(y)



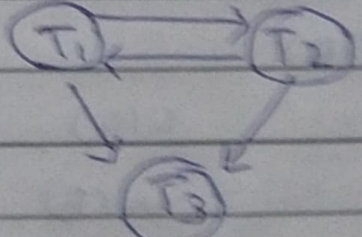
↳ S_2 and S_3 conflict Seariable

View Serializability :-

b) wheter the schedule is conflict serializable or not.

Ex

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



not-conflict serializable

↳ for view serializable create table

SNo	Database obj	Initial Read	Write-Read sequencing	Final write
1	A	T_1	—	T_3
		$T_1 \rightarrow T_3$		

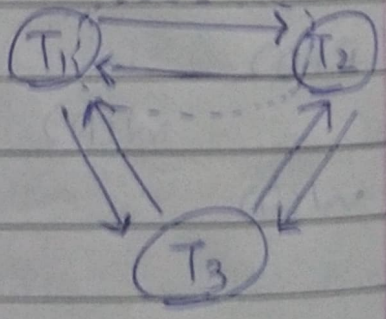
$\rightarrow$ $T_1 \rightarrow T_2 \rightarrow T_3$ (Serial schedual)

initial, middle, final

↳ view serializable above ex.

Ex:2

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



(not conflict serializable)

SNO	Db. obj.	initial Read	W-R sequencing	Final write
1.	A	T1	Yes ($T_3 \rightarrow T_2$)	T2
2.	B	T2	$T_2 \rightarrow T_1$, $T_2 \rightarrow T_3$	T2

for obj. A :- $T_1 \rightarrow T_3 \rightarrow T_2$

for obj. B :- $T_2 \rightarrow T_1 \rightarrow T_3$ or

$T_2 \rightarrow T_3 \rightarrow T_1$

↳ cheak order and match both object serial order.

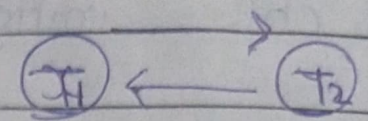
↳ if it is not matching then it is not view serializable.

↳ if it is matching then it is view serializable.

Page :
Date : / /

Ex:3

T1	T2
R(x)	
W(x)	
	R(y)
R(y)	
W(y)	
	W(y)
	R(x)
	W(x)



(not - conflict)

I	x	T1	$T_1 \rightarrow T_2$	T2
2	y	T2	$T_2 \rightarrow T_1$	T1

x : $T_1 \rightarrow T_2$

y : $T_2 \rightarrow T_1$

(not view serial)

Serial $T_1 \rightarrow T_2$

T1	T2
R(x)	
X = X - 10	
W(x)	
R(y)	
Y = Y + 10	
W(y)	
	R(y)
	Y = Y + 20
	W(y)
	R(x)
	X = X - 20
	W(x)

X = 100
Y = 100

X = 90
Y = 110

~~X = 80~~
Y = 120

X = 70
Y = 130

Serial order $T_2 \rightarrow T_1$

X = 80
Y = 120

X = 70
Y = 130

→ Last cheak Result equvivance test.
if all schedule match the result then it is serializable.

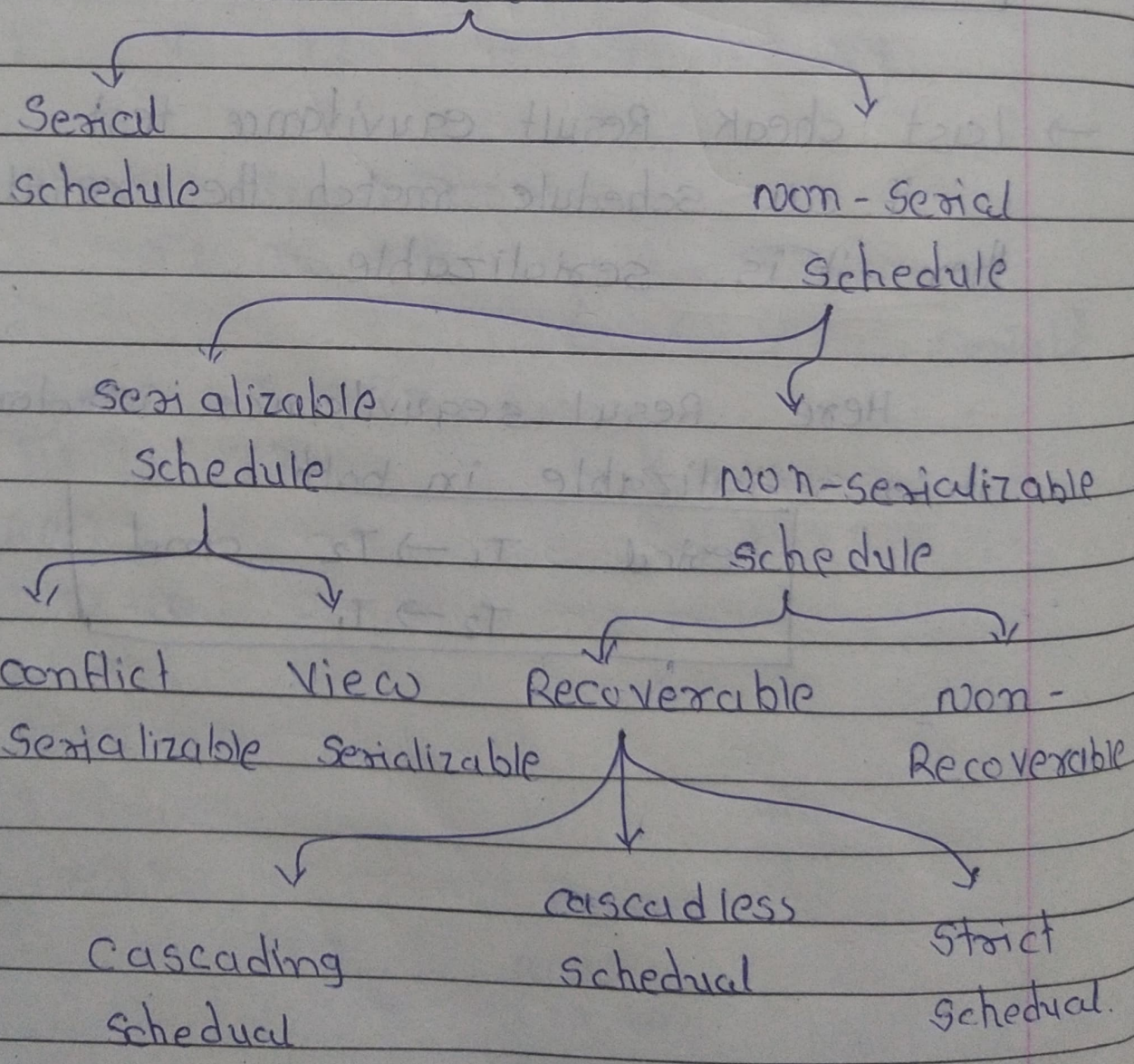
Here Resul equvilance is done and it is serializable in both

Serial $T_1 \rightarrow T_2$ and $T_2 \rightarrow T_1$

Types of Schedules :-

* Schedule :- " The order

1.) Serial schedule
2.) Conflict schedule
3.) View schedule
4.) Concurrent (non-Serieal) schedule



~~Non-Serialized~~

Non-serial schedual :-

Two types :-

1.) Recoverable :-

"schedules in which transaction commit only after all txn whose changes they read commit are called..."

Ex:	T1	T2 (Recoverable)
	R(A)	
	W(A)	
		W(A)
		R(A)
	commit	
		commit.

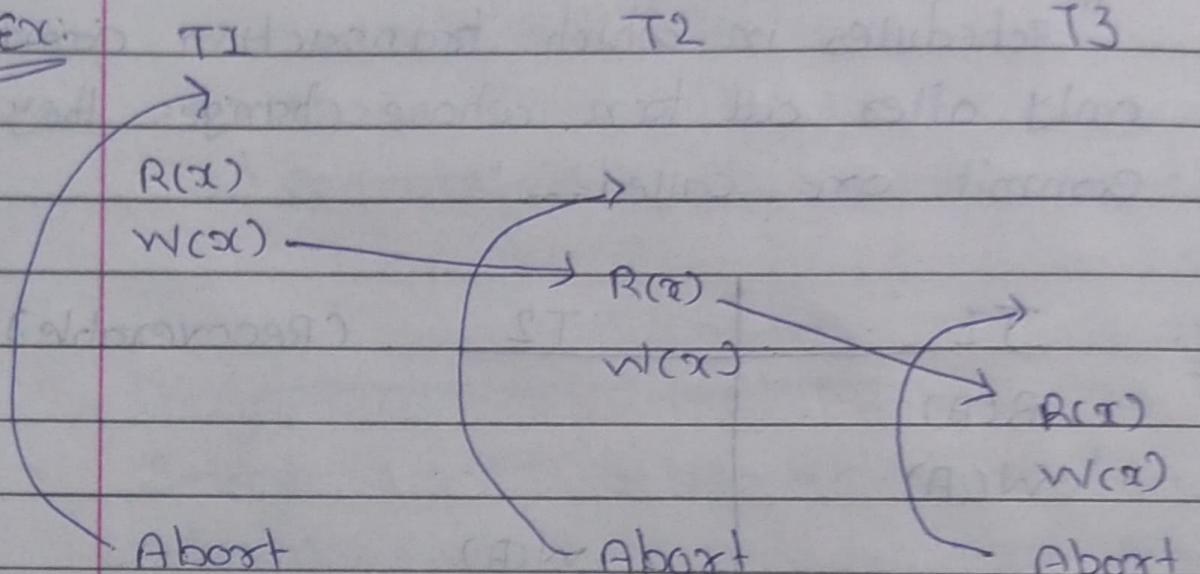
→ If only txn changes ~~the~~ and it fails then and only then it's not Recoverable.

→ If only Read op. perform and other txn changes and 1st txn fail also it called Recoverable.

Cascading schedule :- (Avoid Cascading aborts) (ACA)

"when there is a failure in one txn and this leads to the rolling back or aborting other dependent txn. it called ...."

Ex.



Cascadless schedule :-

"# schedules in which txn read values after all txn whose changes they are going to read commit are called..

⇒ other (T2) txn can't Read (R(A)) T1 txn before the Txn T1 commit

Page :
Date : / /

Ex:

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

એક વાર R(A) વાપર પહેલી other txn આ R(A) ના વાપરે. ToPa.

Strict Schedule :-

⇒ other txn are not allowed R(A) or W(A) before the commit.

↳ if there is a chang after that R(A) or W(A) not allowed. or value change not allowed until the Txn T1 commit.

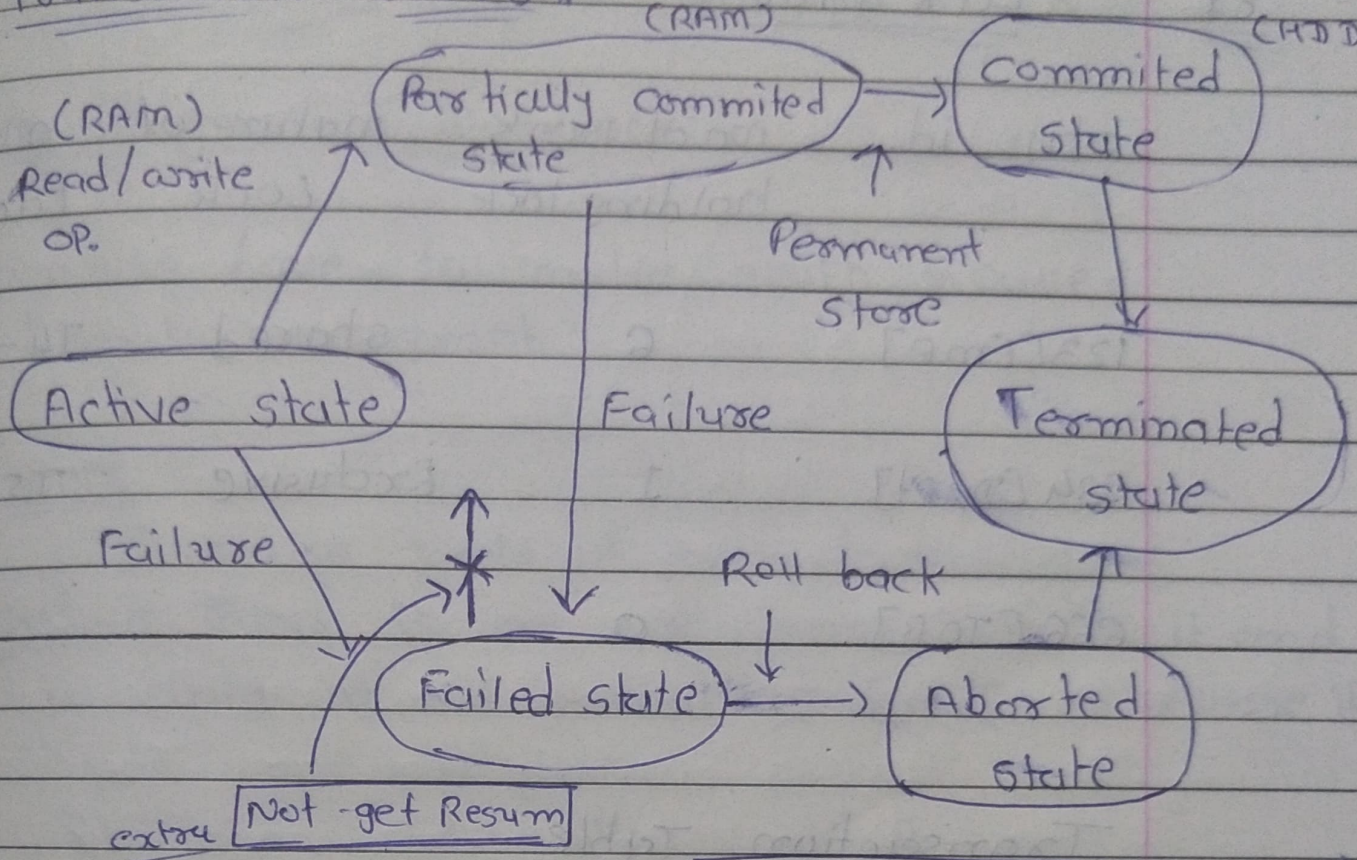
Ex:

T1	T2
R(A)	
	R(A)
W(A)	
commit	
	W(A)
	R(A)
	commit

it will be not conflict.

એક વાર Value change વાપર પહેલી R(A) કે W(A) ના વાપરી શકે commit સુધી. ToPa

Transaction States :-



Txn always Restart never Resum

Intro. to Lock Managment :-

↳ all granted by Lock manger.

1.) Shared Lock. → Read allowed

2.) Exclusive Lock. → R,W allowed

↳ some Txn T1 having shared lock than other Txn only allowed shared lock.

↳ any other case not allowed ~~Error~~ khori.

Ex: Lock Table

obj. id	no of txn's holding lock	nature of Lock	queue of Lock
123[Emp]	6	shared	T4 → T5 → T6
234 [Dept]	1	Exclusive	T5 → T6
666 [JOB]	0		

Transaction Table

Txn_ID	List of locks held by txn
T1	R[EMP], W[DEPT]
T2	R[EMP]
T3	-

S = Shared lock

X = Exclusive lock

Deadlock & Specialize Locking :-

→ Dealing with Deadlock:-

→ we have to make graph called "waits-for-graph"

Steps :-

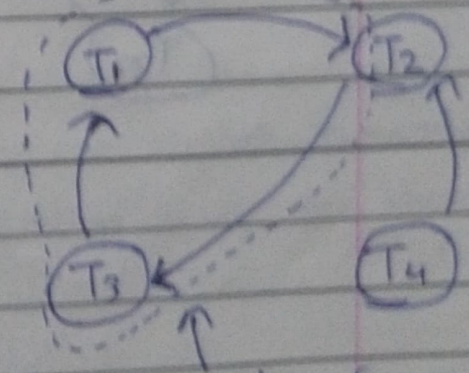
→ make node of every txn.

Imp → There is an arc from T_i to T_j. if and only if, T_i is waiting for T_j to release it.

Ex

T1	T2	T3	T4
S(A)			
R(A)			
	X(B)		
	W(B)		
		S(B)	
		R(B)	
			X(C)
		X(B)	
X(A)			

A = 0
B = 0
C = 0
D = 0

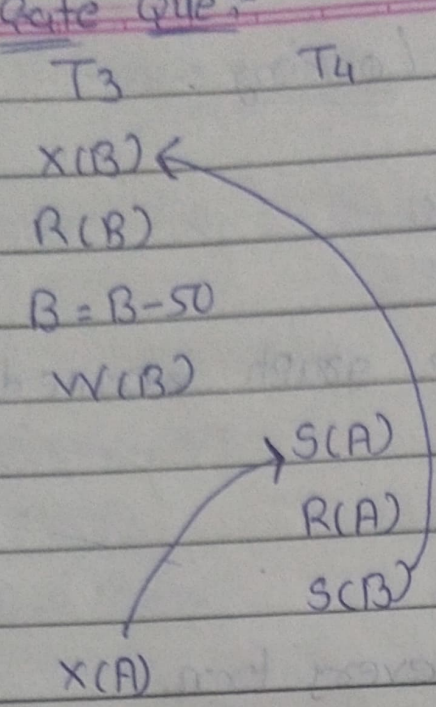


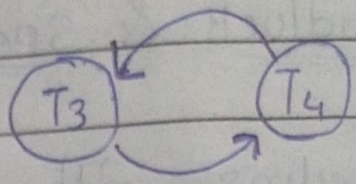
$T_1 \rightarrow T_2$
$\uparrow \quad \swarrow$
T_3

Page :
Date : / /

Rate Que :-

Ex:

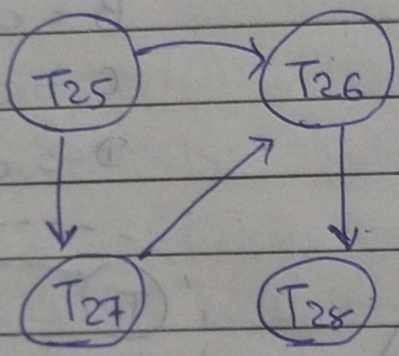




Ans:- Result in Deadlo

Ex: Txn T_{25} is waiting for txn T_{26} and T_{27}
Txn T_{27} is waiting for txn T_{26}
Txn T_{26} is waiting for txn T_{28}

Ans:



Ans: no deadlock state.

Page :
Date : / /

Deadlock Prevention :-

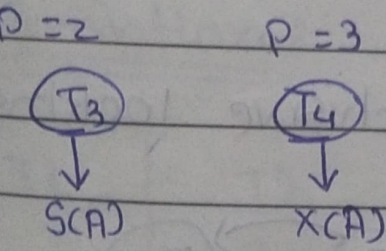
↳ we can prevent deadlock by giving each txn a Priority to each txn.

two methos availble:-

1.) wait-die :- if T_i has higher priority it is allowed to wait, otherwise, it is aborted.

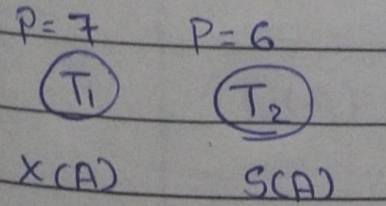
2.) wound-wait :- If T_i has higher priority abort T_j; otherwise, T_i waits.

Ex:



By using wound-wait T_4 will aborted T_3 txn. and first T_4 txn will go through

Ex:



By using wait-die T_1 will allowed to wait and if it's $P \leq 6$ it will aborted.

Starvation :- Finite time waiting.

Shared - Exclusive locking :-

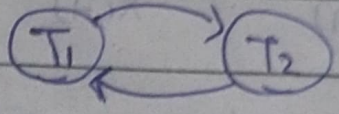
(1) shared lock (S) :- S(A) :-

(2) Exclusive lock (X) :- X(A) :-

Ex:	T1	T2
	S(A)	
	R(A)	
(unlock the lock)	U(A)	
		X(A)
		R(A)
		W(A)
		U(A)

⟹ Deadlock in shared - Exclusive locking :-

Ex:	T1	T2
	X(A)	
		X(B)
	X(B)	
		X(A)



Page :
Date : / /

Irrecoverable schedule :-

T1	T2
X(A)	
R(A)	
W(A)	
U(A)	
	X(A)
	R(A)
	W(A)
	U(A)
	commit
*	

Starvation :-

T1	T2	T3	T4
	S(A)		
X(A)	....		

		S(A)	
	U(A)	....	S(A)
		U(A)	....
			U(A)

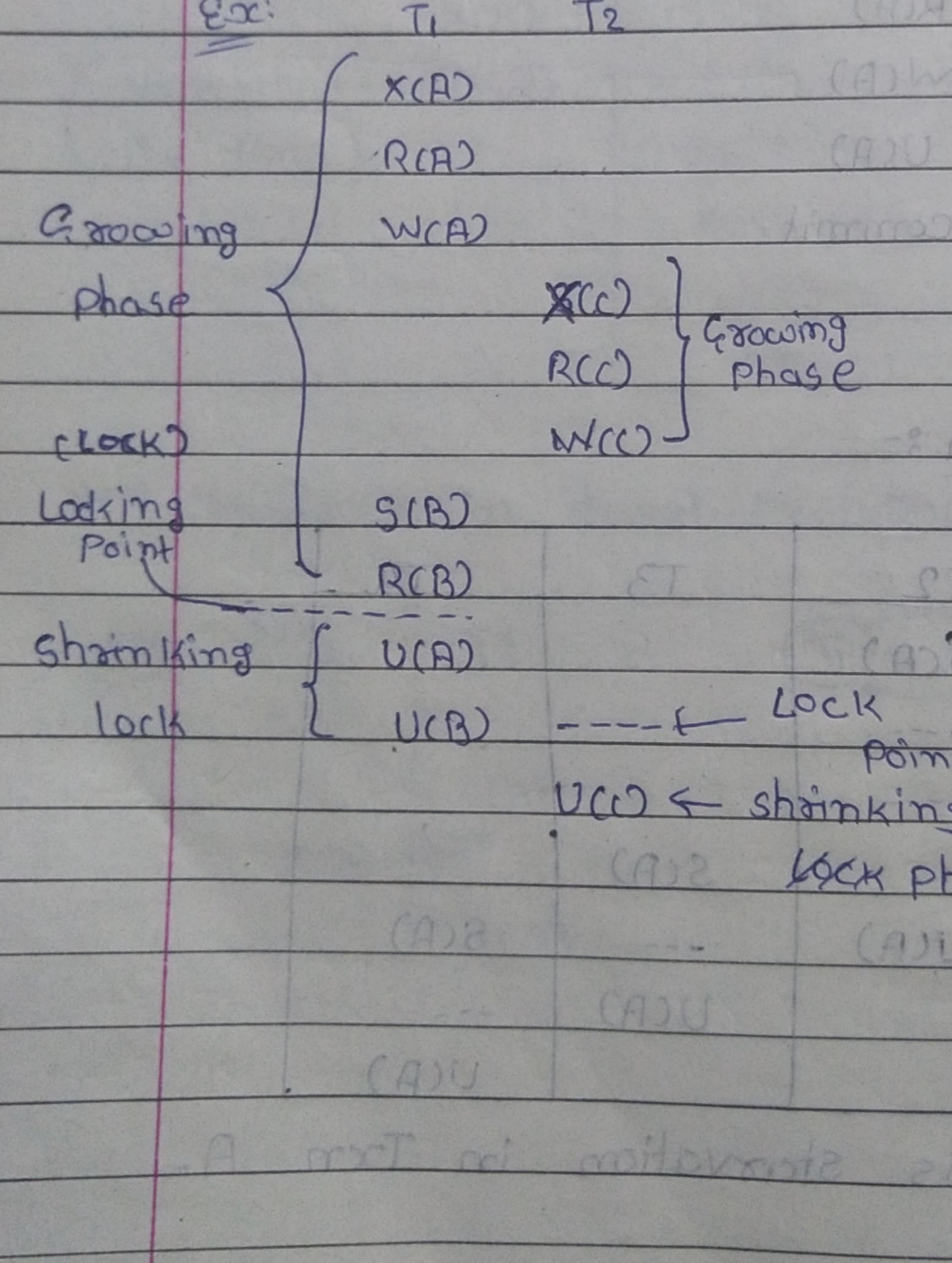
There is starvation in Txn A.

2-Phase locking :-

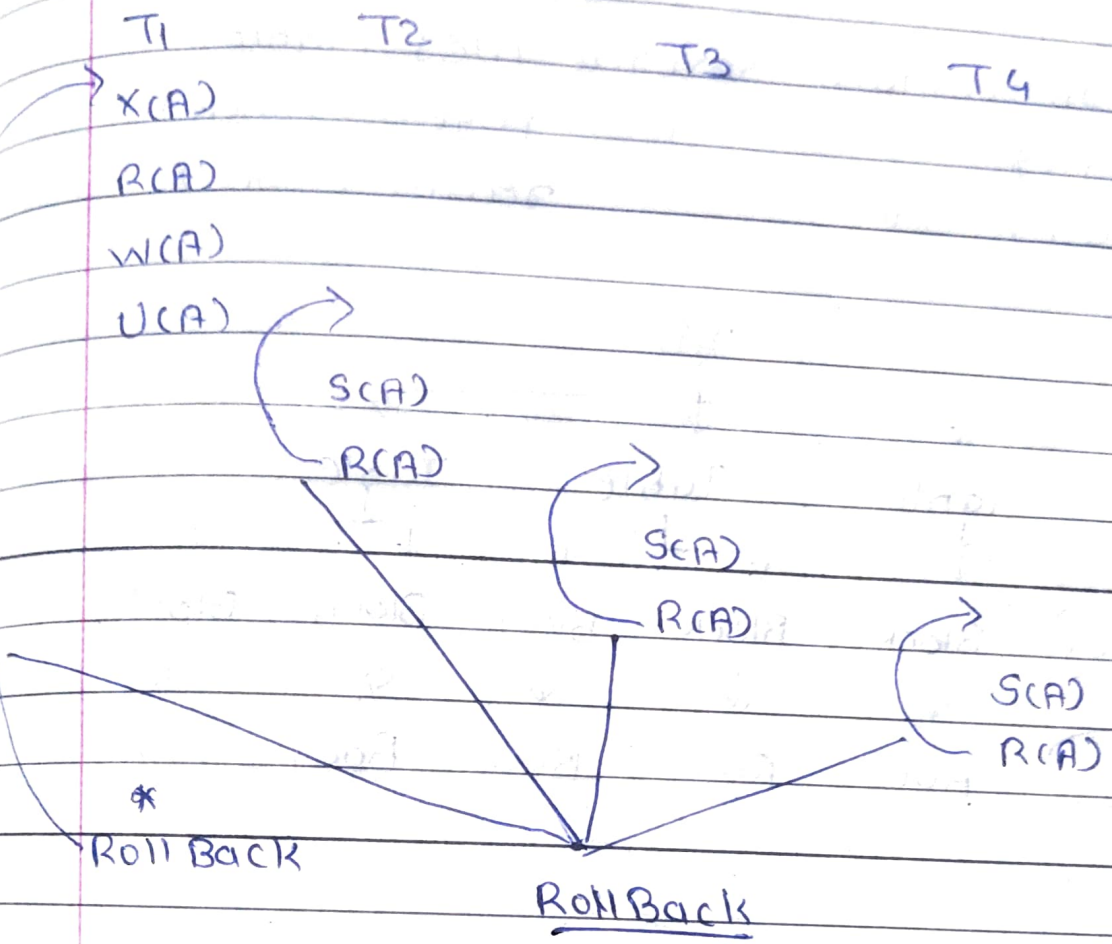
Growing lock : lock are acquired and no locks are released.

Shrinking lock : locks are released and no locks are acquired.

Ex:



Cascading RollBack :-



1) Strict 2PL :

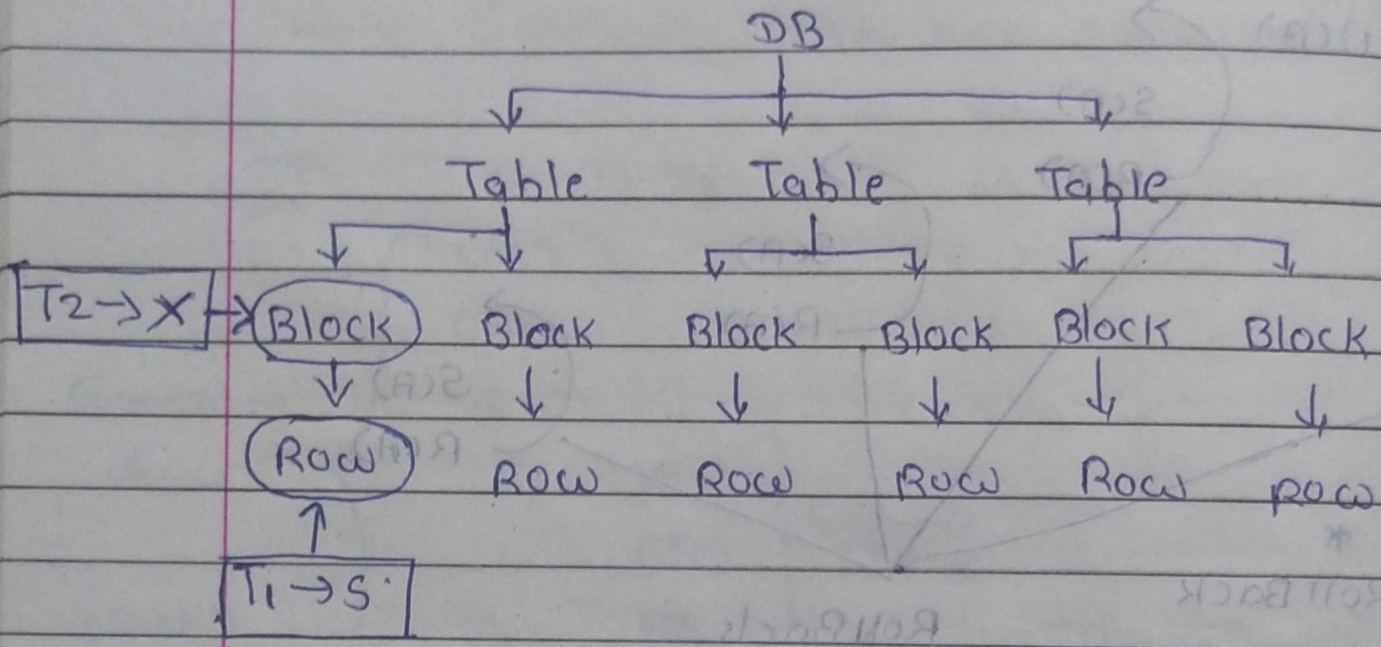
→ Satisfy 2PL and X (exclusive) lock hold until commit / abort.

2) Rigorous 2PL :-

→ satisfy 2PL and all S (shared) and X (exclusive) locks hold until commit / abort.

Multiple - Granularity Locking :-

→ DB contain several files (table).
→ files " " pages (block)
→ Pages " " records (row)



→ Shared lock (S) } Locks used to lock the
Exclusive lock (X) } object (Row)

→ Intention shared (IS) } Locks used to lock
Intention Exclusive (IX) } ancestors (Table, Block)
→ IS conflict with X locks
→ IX conflict with S and X locks.

→ Lock must be acquired from Root-to-leaf
→ Lock must be released from Leaf-to-Root

Log based recovery: (deferred database modification)

Syntax:

<Transaction no., object, new value>

	T1	Log
A=100	R(A)	
B=200	A=A+100	<T1, START>
	W(A)	<T1, A, 200>
	R(B)	<T1, B, 400>
	B=B+200	<T1, COMMIT>
	W(B)	
A=200	Commit	Redo operation
B=400	* (failed)	

Log based recovery: (Immediate database modification)

Log

<T1, START>

<T1, A, 100, 200>

<T1, B, 200, 400>

<T1, COMMIT>

→ If txn is not commited at it failed then we will do UNDO operation otw. we do REDO operation.

Timestamp Protocol :-

Before that some knows. :-

TS = Time stamp
TS(TI) = T_I's Time stamp

→ whenever any txn come into System The unique value assigned to it.

RTS = Read time stamp (Last txn TS)
WTS = write time stamp (Last txn TS)

Time stamp Protocol Rules :-

1) R(A)

A) WTS(A) > TS(T_i) then roll back T_i.
B) else execute RTS(A) = max {RTS(A), TS(T_i)}

2) W(A)

A) RTS(A) > TS(T_i)
B) WTS(A) > TS(T_i)

} Rollback T_i if and only if both true