Query Processing & optimization

Query (SQL) ⇒ DBMS] Files, Tables, data sheets

select, from, where.

(DBL)

[1st From, 2nd Where, 3rd select]

Query processing is the Activity performed in extracting from the data BASE. In Query processing, it takes various steps for fetching the data from the database.

Eg: Student_Info

ID	Name	Sem
1	A	5
2	B	6
3	C	1
4	D	1

* Fetch the details of 1st Sem students.

High level language →

```
Select *
From Student_Info.
Where Sem = 1
```

← Low level Language.

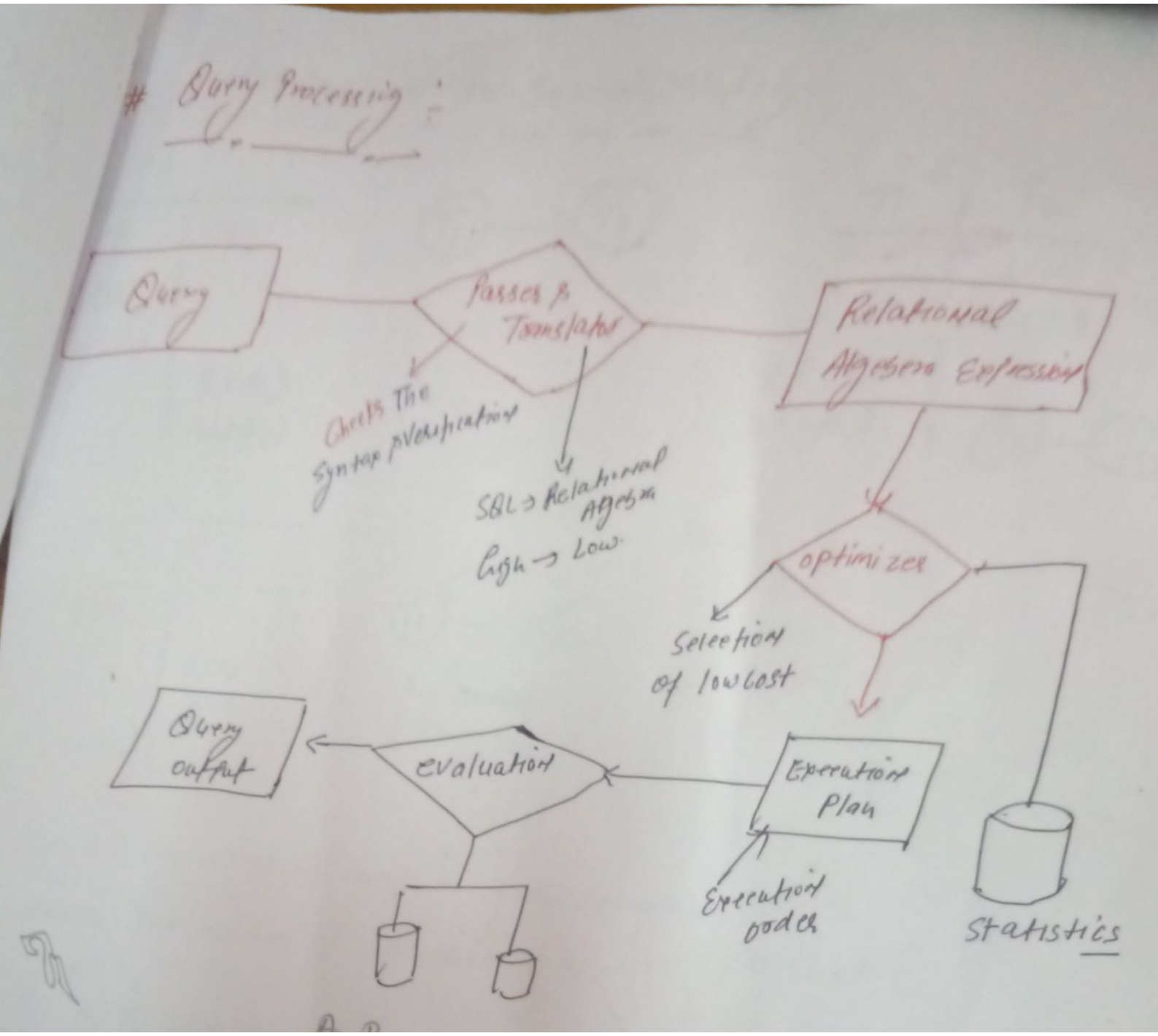
* Relational Algebra.

σ - Selection

π - Projection

$\pi_{ID, Name, Sem}(\sigma_{Sem=1}(Student\_Info))$

Query Processing :



Introduction to Serializability:

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

$T_1 \rightarrow T_2$

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

$T_1 \rightarrow T_2$

Ex

S_1	
T_1	T_2
R(A)	
	R(A)
	W(A)
W(A)	

Ex →

$T_1 \rightarrow T_2$

↓ we convert

$[\, T_1 \rightarrow T_2 \text{ OR } T_2 \rightarrow T_1 \,]$

Can we convert this schedule to Serial schedule.

We have TWO Method of Serializability → Conflict, View.

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

→

$T_1 \rightarrow T_2 \rightarrow T_3$

$T_1 \rightarrow T_3 \rightarrow T_2$

$T_2 \rightarrow T_3 \rightarrow T_1$

$T_2 \rightarrow T_1 \rightarrow T_3$

$T_3 \rightarrow T_1 \rightarrow T_2$

$T_3 \rightarrow T_2 \rightarrow T_2$

"Conflict Equivalent"

R(A) R(A) → Non conflict pair

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

S ≡ S'

S

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

→ Adjacent Non conflict pair

S1

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

↓

S0

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

S

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

Now S ≡ S'

Conflict equivalent

↓

Serializable

Ex 1

T_1	T_2
R(A)	
W(A)	
	R(A) → conflict
W(A)	No changes in position
R(B)	

Conflict Serializability :

Q. Whether a schedule is Conflict serializable or not?

T_1	T_2	T_3
R(x).		
		R(y).
		R(x)
	R(y)	
	R(z)	
		W(y)
	W(z)	
R(z)		
W(x)		
W(z)		

Conflict pair
- R → W
- W → R
- W → W

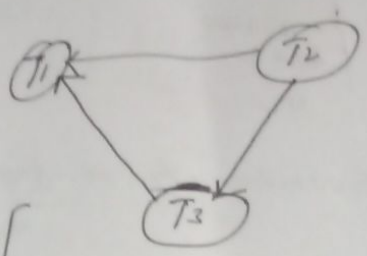
Now possibility

- $T_1 \to T_2 \to T_3$
- $T_1 \to T_3 \to T_2$
- $T_2 \to T_3 \to T_1$
- $T_2 \to T_1 \to T_3$
- $T_3 \to T_2 \to T_1$
- $T_3 \to T_1 \to T_2$

$T_2 \to T_3 \to T_1$

Check conflict pairs in other transactions and draw edges

Precedence graph :

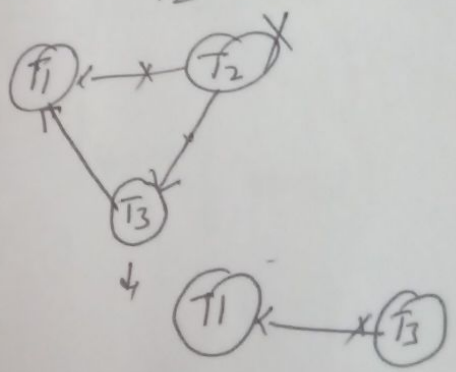


Loop / Cycle

Now here no loop and no cycle

Now this schedule Conflict Serializable & Serializable & Consistent

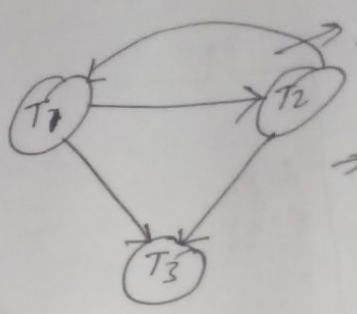
Now check Indegree = 0 of T_2



Check whether schedule is Conflict Serializable or Not ?

S

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



→ Non Conflict Serializable

No ∵ View Serializability is used

S

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

ple View serializability

S		S'	
T_1	T_2	T_1	T_2
R(a)		R(a)	
W(a)		W(a)	
	R(a)	R(b)	
	W(a)	W(b)	
R(b)			R(a)
W(b)			W(b)
	R(b)		R(b)
	W(b)		W(b)

1) Inital Read Both side same transaction

S: $T_1 \leftarrow a \rightarrow T_1$, $T_1 \leftarrow b$ T_1 — S'

2) Final write

S: $T_2 \leftarrow a \rightarrow T_2$, $T_2 \leftarrow b \rightarrow T_2$ — S'

3) Intermidiate Read Same

Q1

S		
T_1	T_2	T_3
R(a)		
	W(a)	
W(a)		W(a)

→

S1		
T_1	T_2	T_3
R(a)		
W(a)		
	W(a)	
		W(a)

1) Inital Read(a) → T_1

2) final write T_3

3) Intermediate Reads NO Applicable

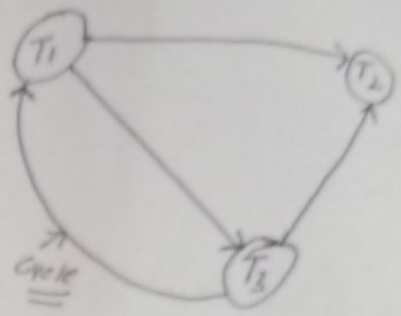
Example.

Conflict Serializability ⇒

Ex

S

T_1	T_2	T_3
R(x)		R(z)
		W(z)
	R(y)	
R(y)		
	W(y)	
		W(x)
	W(z)	
W(x)		

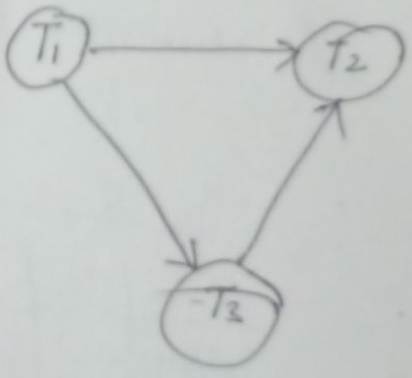


It is Not ConflictSerializable.

S2

Ex

T_1	T_2	T_3
R(x)		
	R(y)	
		R(y)
	W(y)	
W(x)		
		W(x)
	R(x)	
	W(x)	



$T_1 \rightarrow T_3 \rightarrow T_2$

Recoverable Schedules:

↓

Irrecoverable

Ex-1

S

T_1	T_2
R(A)	
A = A + 10	
W(A)	
	R(A)
	A = A - 5
	W(A)
	C.
C	

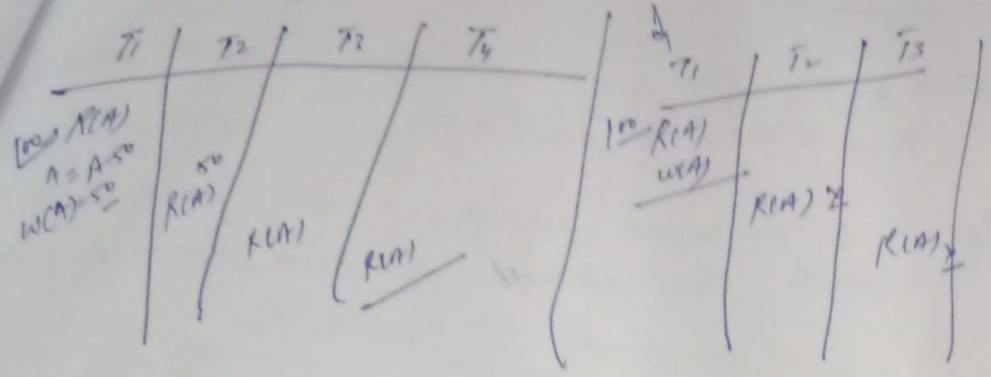
Ex-2

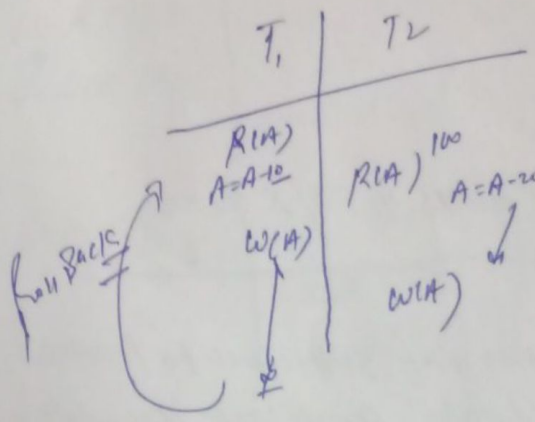
S

T_1	T_2
R(A)	
A = A - 5	
W(A) →	
	R(A) (5)
	A = A - 2
	W(A) (3)
	Commit
(4) R(B)	
: fail	

Rollback

Cascading Schedule vs Cascadless Schedule:





Concurrency control Protocol :-

Shared Exclusive locking

↳ Shared lock (S) → If transaction locked data items in shared mode then allowed to read only.

→ Exclusive lock (X) → If transaction locked data items in exclusive mode then allowed to read and write both.

Shared → T_1	Exclusive T_2
S(A)	S X(A)
R(A)	R(A)
U(A)	W(A)
	U(A)

* Problems in S/X locking

1) May not sufficient to produce only serializable schedule.

2) May not free from Irrecoverability

3) May not free from deadlock

4) May not free from starvation

grant ↓ / (Request) →	S	X
S	Yes	No
X	No	No

T_1	T_2
X(A)	
R(A)	
W(A)	
U(A)	
	S(A) ←
	R(A)
	U(A)

$T_1 \to T_2$ or $T_2 \to T_1$

T_1	T_2
↳ → X(A)	
	X(B)
X(B) → ↳	R(B)
	W(B)
W X(B)	U(B)
(wait) ↳	
	X(A) → wait

2-Phase Locking

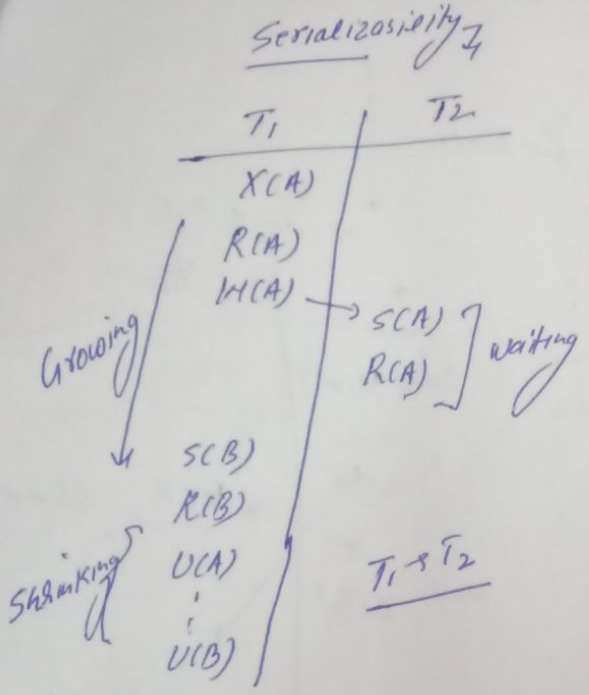
→ Growing phase

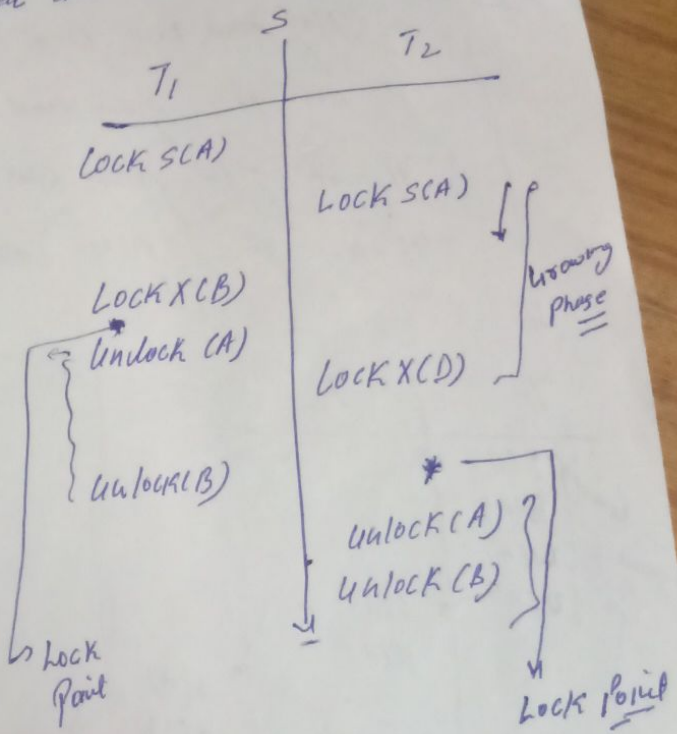
→ Shrinking

2-Phase Locking (2PL)

→ Growing phase :- locks are acquired and no locks are released

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





$T_1 \rightarrow T_2$

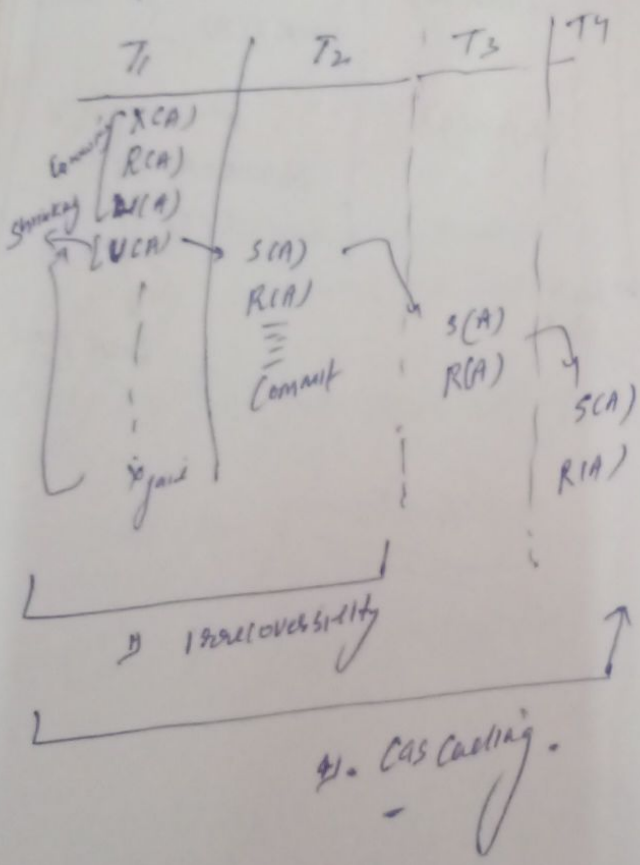
Advantages

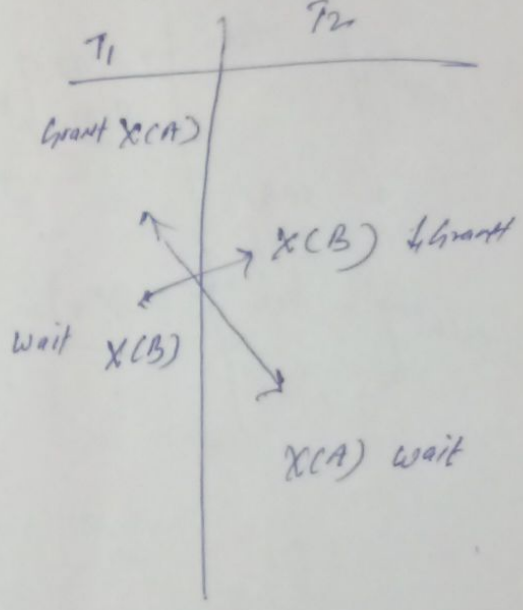
2PL (2Phase locking)

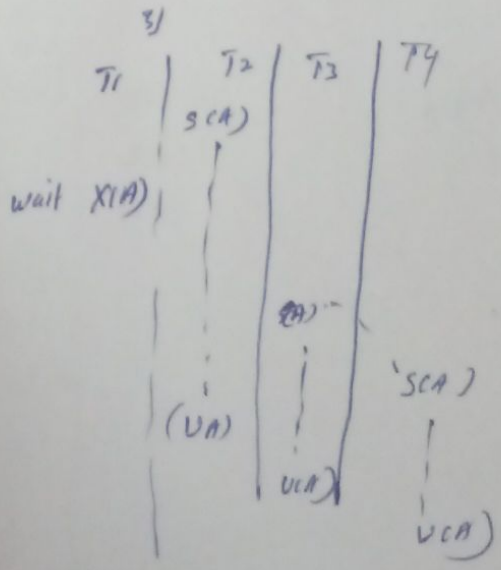
Advantages: Always ensure Serializability

Drawbacks:

1) May not free from irrecoverability
2) not free from dead locks
3) not free from Starvation
4) not free from Cascading rollbacks.

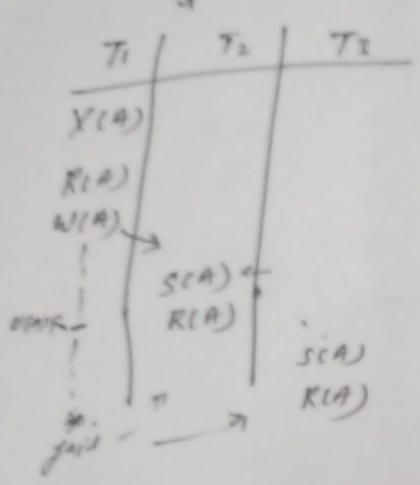


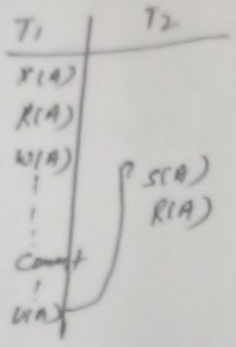




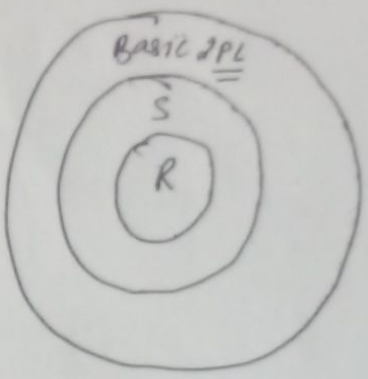
ict 2PL :- It should satisfy the Basic 2PL and all exclusive locks should hold until Commit / Abort.

Rigorous 2PL: It should satisfy the basic 2PL and all shared & exclusive locks should hold until Commit / Abort





Now its cascadless strict Recoverable



Time stamp ordering protocol :

→ Unique value assign to every transaction
→ Tells the order (when they enter into system)
→ Read - Ts (RTS) = Last (latest) transaction which performed Read successfully
→ Write - Ts (WTS) = Last (latest) transaction no. which performed write successfully

10:00	10:10	10:15	Ts(Tc)
T_1	T_2	T_3	
100	200	300	
Order	younger	youngest	

10	20	30
T_1	T_2	T_3
R(A)		
	R(A)	
		R(A)

RTS = 30

10	20	30
T_1	T_2	T_3
W(A)		
		W(A)
	W(A)	

→ WTS = 20

Rules

1) Transaction T_i issues a Read(A) operation
 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 Write(A) operation
 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 $WTS(A) = TS(T_i)$

Notes which Transaction come first, complete ஆகும்

Case 1

100 T_1	200 T_2 (Young)
R(a)	
	W(a)

2

T_1	T_2
W(A)	
	R(A)

(ok)

3

T_1	T_2
W(A)	
	W(A)

(1)

100 T_1	200 T_2
	R(A) →10
20→ W(A)	
RollBack	

Not Allowed

100 T_1	200 T_2
	W(A) →20
20→ R(A)	
RollBack	

Not Allowed

100 T_1	200 T_2
	W(A)
W(A)	

Numerical problem

oldest T_1 100	200 T_2	Youngest 300 T_3
R(A)		
		R(B)
W(C)		
R(C)		

Numerical Problems

Ex

oldest T_1 100	T_2 200	Youngest T_3 300
R(A)		
	R(B)	
W(C)		R(B)
R(C)	W(B)	
	Rollback	W(A)

Rules

1) Transaction T_i issues a read(A) operation

a) If WTS(A) > TS(T_i), Roll Back T_i

b) Otherwise R(A) operation is executed

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

2) Transaction T_i issues write(A) operation

a) If RTS(A) > TS(T_i) Then Roll Back T_i

b) If WTS(A) > TS(T_i) Then Roll Back T_i

c) Otherwise execute write(A) operation

Set WTS(A) = TS(T_i)

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

Log Based Recovery :- (Deferred Database)

(i)

A=100
B=200
DB

A=200
B=400

T_1
R(A)
A = A+100
W(A) 200
R(B)
B = B+200
W(B) 400
Commit

<T1 start>
<T1, A, 200> NEW
<T1, B, 400>
<T1 commit>

Redo ←

<T1, start>
<T2, A, 200>
<T1, B, 400>
<T1, commit>
<T2, start>
<T2, C, 500>

Immed

100
200
DB

(ii)

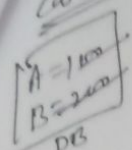
[A=100
B=200]

T1
R(A)
A=A+100
W(A)
R(B)
B=B+200
W(B)
*fail

<T1, start>
<T1, A, 200>
<T1, B, 400>
~~Commit~~
RollBack
No Undo/Redo

Immediate Database modification (Log Based Recovery)

Case-1



T_1
R(A)
A = A+100
W(A) 200
R(B)
B = B+200
W(B) 400
Commit

log T_1

<T_1, start>
<T_1, A, 100, 200> (Old → New)
<T_1, B, 200, 400>
<T_1, commit>

Redo

log T_1

<T_1, start>
<T_1, A, 1000, 200>
<T_1, B, 500, 600>
Redo <T_1, commit>
<T_2, start>
<T_2, C, 700, 800>
↑ Undo

Case-2

T_1
R(A)
A = A+100
W(A)
R(B)
B = B+200
W(B)
↓
(fail)

log T_1

<T_1 start>
<T_1, A, 100, 200> (Old → New)
<T_1 B, 200, 400>

Undo | Redo