Note:- Here, we use a special func^n.
[group by] clause.
↓
(forms grp of dept.).

HR HR	2
MRKT MRKT	2
IT.	1

* Note:- Cond^n of group by → जो Att. का नाम हम group by के साथ use कर रहे हैं, सिर्फ वही Select ____ के साथ लिख सकते हैं।

If,
we have to write other att. in Select ____ other than the att of group by func, then we have to use aggregate func^S.

- Query:-
Select dept from emp group by dept;
→ [Output] →

HR
HR
MRKT
MRKT
IT

* Note:- Count (dept) or Count (*). → Same.

→ Query:- (aggregate func.)
Select dept, Count(*) from emp group by dept;

Output →
HR - 2
MRKT - 2
IT - 1

61. (ii) Write a Query to display all the dept names where no of Emps are less than 2.

HR 2
MRKT 2
IT 1

Note:- 'Where' clause works on complete Table. but here we group by this table. Hence, group by & where are independent. not help each other.
Hence,
we use 'having'.

Query:-

```
Select dept from Emp group by dept
having Count(*) < 2;
```

Output → IT.

Q ⇒. If they ask of Employee name in this Query, then → (Varun → output)

Query →

```
Select E-name from Emp where dept In
(Select dept from Emp group by dept
having Count(*) < 2);
```

↳ Varun.

In
HR IT
MRKT
IT

62

(viii) Write a Query to display highest Salary department wise and name of emp who is taking that Salary.

Note: A SQL Query always starts from 'from'

Select ___ from table name

Note: Select max (y) from ___ group by x;

Here, It is Right, bcz we use aggregate func 'max' with 'y' i.e. max(y) & if we don't use max, then we can only use x, bcz group by x. i (Emp).

Query :-

```
Select Ename from Emp where Salary
In (Select max (Salary) from Emp
 Group by dept );
```

Output → Ravi, Nitin, Varun ... dt ...

(Steps) →

HR HR	HR - 30,000	30k
MRKT	MRKT - 40,000	40k
MRKT	IT - 50,000	50k
IT		

Outer
10k x
20k x
30k
40k
50k

Inner
30k
40k
50k

* means All attributes.

simple in Query.

63. Use of IN and Not IN :-

1= (1,2,3,4,5) X (Wrong way).

1= 2 false. (but, Right way).

Emp

E_id	E_name	Address
1	Ravi	Chd
2	Varun	Delhi
3	Nitin	Pune
4	Robin	Bangalore
5	Ammy	Chd.

Project

E_id	P_id	P_name	Loca
1	P_1	IoT	Bangalore
5	P_2	Big Data	Delhi
3	P_3	Retail	Mumbai
4	P_4	Android	Hyderabad

Q:- Detail of Emp whose address is either Delhi or Chd or Pune ;

Query:-

```
Select * from Emp where Address = 'Delhi';
```

Output:- | 2 | Varun | Delhi |

⇒ But, we have 3 cities. So,

Query:-

```
Select * from Emp where
        Address In ('Delhi', 'Pune', 'Chd');
```

Output:

1	Ravi	Chd
2	Varun	Delhi
3	Nitin	Pune
5	Ammy	Chd.

Chd ✓
Delhi ✓
Pune ✓
~~Bangalore~~ ✗
Chd. ✓

Nested Query = Query (Query)

Now, NOT IN :- (others not included)

Query :-

```
Select * from Emp Whose
Address Not In ('Delhi', 'Pune', 'Chd');
```

Output :- | 4 | Robin | Bangalore |

Chd ✗
Delhi ✗
Pune ✗
Bangalore ✓
Chd, ✗

64- Use of IN & Not IN in Subquery :-

(Same 2 tables of before).

Query :- find the name of Emps who are working on a project ?

⇒ Suggestion :- First make Dummy Tables.

⇒ Here, we need both 2 Tables.

Both tables have common - EID
So, we compare by taking EID.

⇒ Nested → Bottom up → नीचे से ऊपर

Means पहले अंदर वाली query & then बाहर वाली

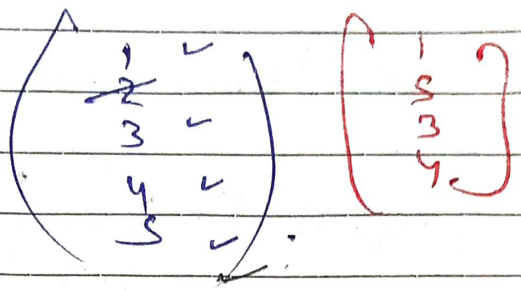
Inner Query in Nested Query - Runs 1 time.

* Query :-

```
Select Ename from Emp where Eid
In ( Select Distinct (Eid) from Project );
```

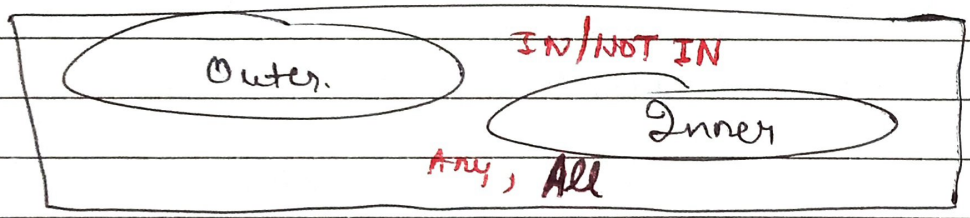
Output :- Ravi
Nitin
Robin
Ammy.

Inner Query.



65. EXIST & NOT EXIST Subqueries :-

↳ We use these in Correlated Nested Query.



⇒ In nested query. — We use IN / NOT IN.

&

In Correlated nested Query - We use EXIST / NOT EXIST.

Query → Find the detail of Emp who is working on at least one Project ?
(Same 2 Tables)

Note :- In nested Query → Inner Query executes first, & then Compare its Output with Outer Query.
In Correlated Nested Query :- the one row of Outer Query is Compared with all rows of Inner Query.
ie, Top to Down Approach.

Null → means value is not available.
Empty

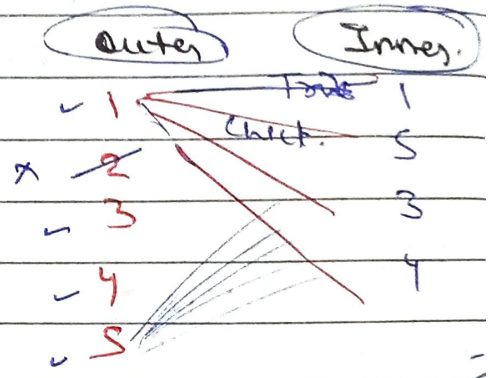
3) Query:-

```
Select * from Emp where Eid
Exists (Select Eid from Project where
        Emp.Eid = project.Eid);
```

Note:- Exist / not Exist gives True or False.

Output:-

1	Rani	Chd
3	Nitin	Pune
4	Robin	Bang.
5	Ammy	Chd.



66) Aggregate Funcs in SQL:- →

Max, Min, Count, Avg, Sum.

↳ (Total no. of values in a Table).

Emp

E_id	E_name	Dept	Salary
1	Ram	HR	10k
2	Amrit	MRKT	20k
3	Rani	HR	30k
4	Nitin	MRKT	30k
5	Varun	IT	50k
6	Sandy	Testing	Null

- Query for Max. Salary:-

→

```
Select Max(Salary) from Emp;
```

Output → 50k

⇒ If 50k repeats 2 times, then it also give output 2 times.

↳ Same for Min.

→ Select Min (Salary) from Emp;

Output: → 10k.

* COUNT:-

Count (*) means no. of rows in the Table.

→ Select Count (*) from Emp;

Output → 6 . 6 rows.

→ Select Count (Salary) from Emp;

Output → 5 (bcz, one value is Null)

* Distinct:-

→ Select Distinct (Count (Salary)) from Emp;

Output → 4 . (bcz 30k is 2 times).

* SUM:-

Slect Sum (Salary) from Emp;

Output → 140 k . (Sum of all salary)

→ Select Distinct (Sum (Salary)) from Emp;

Output → 110 k (30k repeats)

(*) AVG :-

Avg (salary) = Sum (salary) / Count (salary)

= 140K / 5

= 28K

Select Avg (Salary) from Emp;

Output :- 28K.

Select Distinct (Avg (Salary)) from Emp;

Output :- 27,500.

Distinct (Avg (Salary)) = Distinct (Sum (Salary)) / Distinct (Count (Salary))

= 110K / 4

= 27,500

---

(67) Co-related Subquery in SQL :- (Synchronized Query).

- It is a subquery that uses values from Outer Query.
- Top to Down Approach :- (Outer to Inner)
- for One row of Outer Table it compare with every row of inner Table.

→ Returns True / False.

Emp

Eid	name	address
1	A	Delhi
2	B	Pune
3	A	Chd.
4	B	Delhi
5	C	pune
6	D	Mumbai
7	E	Hyd.

Dept

D-id	D-name	Eid
D_1	HR	1
D_2	IT	2
D_3	MRKT	3
D_4	Testing	4

Query:- Find all Employee detail who work in a department.

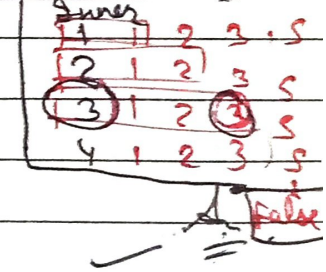
→ True होते ही हमें Employee Detail दे देगा।

→ Query :-

```
Select * from Emp where
    exists ( Select * from dept where
        dept.Eid = Emp.Eid ) ;
```

Output :→

Eid	name	Address
1	A	Delhi
2	B	Pune.
3	A	Chd
4	B	Delhi



68 DB Joins, Nested Subquery & Corelated Subquery :→

→ Nested → Bottom up. (अंदर से बाहर)
→ Corelated → Top down Approach (बाहर से अंदर)
→ Joins → Cross product + Condⁿ

Emp

E_id	name
1	A
2	B
3	C
4	D
5	E

Dept.

Dept no.	name	E_id.
D_1	IT	1
D_2	HR	2
D_3	MRKT	3

Q:→ Detail of all Emp who works in any dept.

(Here, we need both 2 Tables).

→ Nested Subquery →

```
Select * from Emp where E_id in
(Select e_id from dept);
```

Output →

1	A
2	B
3	C

→ Correlated Subquery :→

```
Select * from emp where exists
(select id from dept where
emp.e_id = dept.e_id);
```

E_id is common in both Table.

Output :→

1	A
2	B
3	C

2 rows

1 = 1 → True

2 = 1

3 = 1

2 rows

1 = 5

2 = 5

3 = 5

False

→ If 1st Table → m rows
& 2nd Table → n rows

then, Total Comparison → m × n

→ JOINS :→

Cross product + Condi".

It creates a new Table with (m × n) rows.

→ then,
check Condi" → emp.e_id = dept.e_id

Note:- Joins is faster than Related Subquery bcz it made a table of rows (m × n) at Once. While in Related, we again & again compare one by one row of Outer Table with all Rows of Inner Table. So, it takes time. △ But, Joins take more space. (bcz big Table of (m × n) Rows.).

---

69. Find Nth Highest Salary using SQL :→

Emp.

Id	Salary
1	10k
2	20k
3	20k
4	30k
5	40k
6	50k.

→ Highest Salary :-

```
Select max (Salary) from Emp;
```

Output → 50k

→ 2nd Highest Salary :-
(Nested Query).

```
Select max (Salary) from Emp where
Salary Not In (Select max (Salary) from
                Emp);
```

Output :- 40k

Now

How to recognise Correlated Subquery?
जहाँ पर outer Query का कोई ना कोई value इसने Inner use किया है।

How to find N^th Highest Salary! →

Query :-

Best method → learn it.

```
Select id, Salary from Emp e1 where
N-1 = (Select Count (Distinct Salary) from
        Emp e2 where
        e2.Salary > e1.Salary);
```

↳ Co-related Subqun

Here
we make 2 alias of Emp i.e. e1 & e2
(Dummy)

+ Ex:- on 1st case :-

E_1

ID	Salary
1	10k
2	20k
3	20k
4	30k
5	40k
6	50k.

E_2

ID	Salary	
1	10k	> 10k (false)
2	20k	> 10k count = 1
3	20k	> 10 count = 1 (distinct)
4	30k	> 10 count = 2
5	40k	> 10 c = 3
6	50k.	> 10 c = 4

⇒ Why we make 2 (E_1 & E_2) :-
bcz employee we use 2 times. (E_1, E_2).

↳ If we don't create E_1 & E_2, then
$e_2.Salary > e_1.Salary$
↳ It means, employee salary > employee salary.
↳ no mean. ✗.

↳ E_1 & E_2.

Ex:-
1st
2nd row

10k	> 20k	f
20k	> 20	f
20k	> 20	f
30k	> 20	count = 1
40k	> 20	count = 2
50k.	> 20.	count = 3

} count = 3.

Ex:-) Let we have to find 4th highest, then
N-1 ⇒ 4-1 = 3.
&

```
Select id, Salary
    3 =
```

When we get 3 as count from this Inner query, then, our output comes.

And, we get Count = 3
from the 2nd row bcz,
from 1st Row we got count = 4.
Hence,
Ans is data of 2nd Row

Output:-

ID	Salary
2	20k.

20k is our 4th highest Salary.

50k - 1st
40k - 2nd
30k - 3rd
20k - 4th.

Ex:- for 3rd highest Salary:-
N=3, N-1 = 2

Hence,
for which Row we got Count = 2,
that row will be our output.

10k > 30 F
20k > 30 F
20k > 30 F
30k > 30 F
40k > 30 T , count = 1
50k. > 30. T , count = 2

4th Row is our output

Output: 4, 30k.

30k is our 3rd highest Salary.

3 Imp. Questions on SQL basic concepts :-

Q.(E) You need to display last name of Employees who have 'A' as 2nd character in their names. Which SQL statement display the required result?

a) Select last_name from Emp where last_name like '_A%' ✓

b) — " — last name = '*A%' ✗

c) — " — name like '%A%'

d) — " — like '*A%'

Note:- Questions like, 2nd letter same, Salary be of only 5 nos, like that, based on 'Like' command.

% → any value.
_ → fixed a place for a value.

Ex:- i) %A% Anything AABA ✓ (ये भी सही है)

ii) 2nd letter → '_A%' _ (one pair is fixed)

iii) 4th character must be A → '___A%'

iv) 2nd last letter must be A → '%A_'

(2.) A command to remove relation from SQL database → (Table).

A) Delete table < table name >.
B) Drop table < ——— > 3. Ans
C) Erase table < ——— >
D) Alter table. < ——— >.

→ DDL Commands deal with Schema (Table structure).

Create, drop, Alter → DDL Commands.

→ Alter table → to change anything in table.

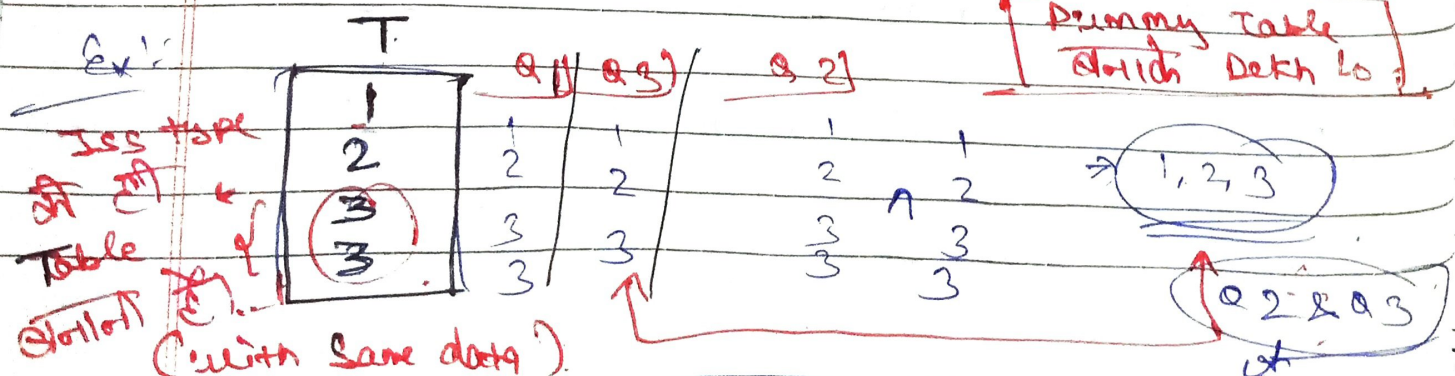
Delete table, Erase table. ≠ (Even not a command).

(3.) In the following, schema R is R(a,b).

Q1: Select * from R
Q2: (Select * from R) Intersect (Select * from R)
Q3: Select distinct * from R

a) Q1, Q2, Q3 produce same result
b) only Q1, Q2 ——— "
c) only Q2, Q3 ——— " ——— 3 Ans.
d) Q1, Q2, Q3 produce diff. result.

Ex:



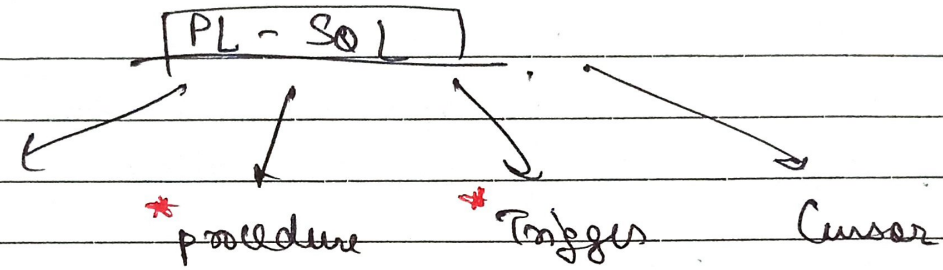
71. PL-SQL

(procedural - SQL).

⇒ SQL → Declarative in nature. (only 'What to Do').

⇒ PL-SQL → procedural (1. What to Do → 2. How to Do.)

(programming flavour वाली चीज़).



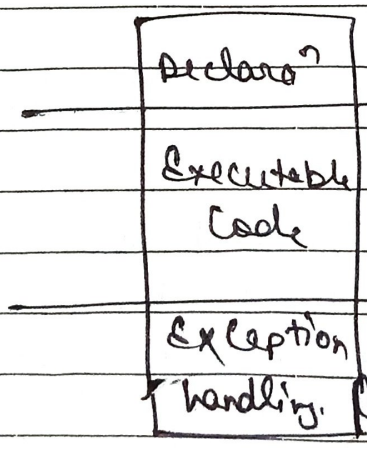
→ In SQL → (Write a query).

→ In PL-SQL → (Write a program, or Write a PL-SQL Code

(Program write करो).

⇒ Its Block Code has 3 parts: →

```
declare
a int
b int
c int
begin
a := 10;
b := 20;
c := a+b;
end;
```



↳ means (If manually, we have to raise any errors). like $\frac{x}{0}$, we define it in Excep^n handling.

in C++ { }

CPU always performs in RAM. CPU - fast.

CPU never works on Hard Disk → Hard Disk → slow

---

72. # TRANSACTION CONCURRENCY :→

Transac°: It is a set of operations used to perform a logical unit of work.

(कुछ भी हम change कर रहे, Database के अंदर, कुछ भी हम Database को Read कर रहे, that is also a part of transac°.)

Ex-1

→ When we withdraw our money from ATM, then we have to perform a ~~test~~ set of opera°, these set of opera's called as Transaction.

→ [Work - Withdraw Money.]

A transaction generally represent change in database.

Database Transactions has 2 operations :→ Read & Write. (Commit) - we also use this.

[Read is the access of Database.
Write is change in database, we made.]

→ When we read or access any data from HDD (hard drives), then it comes to RAM (where we perform opera's).

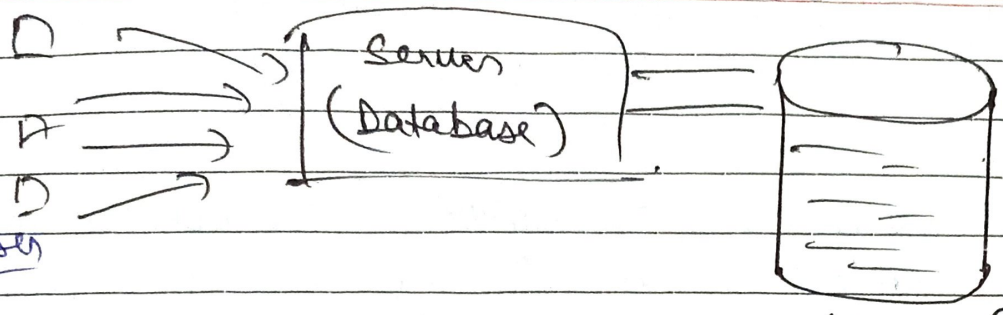
Commit → Whatever change we made, save that permanent in Hard Disk.
(Update data in HDD)

In C++ for assignment, =
In PLSQL , : =

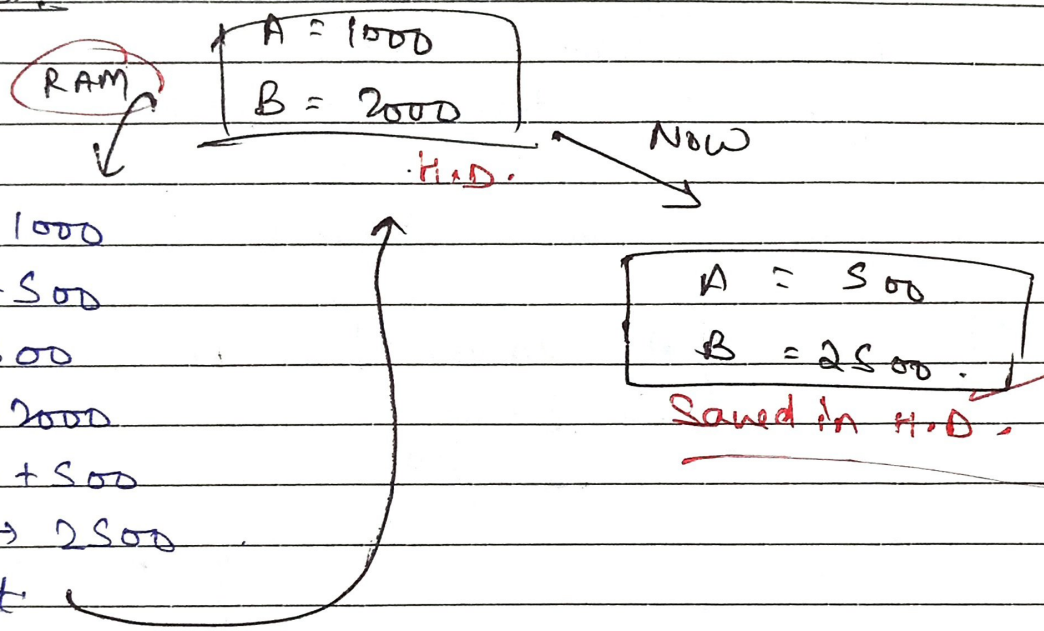
Date
Page

C++, = =
PLSQL, =

(37)



* Transfer :→



(73) ACID properties of a Transaction :→

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

} At back End

* Atomicity → Either all or None.

Ex:- T1 (Transac")

R(A)
A = A - 50
W(A)
R(B)

Commit

Roll back

(उदाहरण Commit से पहले कहीं भी fail हो गया, then Roll Back)

या तो सभी opers execute होते, commit वरना पूरे से Roll back होता।

Note:- A failed Transaction cannot be Resumed. A failed Transac° will always restart.

Consistency :→

Before trans. start And, after the trans. completed,

Sum of money should be same.

Ex:

A = 2000
B = 3000
5000

A ——1000——→ B

T_1

R(A) 2000
A = A - 1000
W(A) 1000
R(B) 3000
B = B + 1000
W(B) 4000
Commit

A = 1000
B = 4000
5000

Sum is same

Isolation :→

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

parallel Transac's.

→ We try to Convert a parallel schedule into a Serial Schedule. (Conceptually)

CPU Speed is in MIPS (Million instruction per second).
Hard Disk → 10/20 instruction per second.
∴ CPU never compatible with HD for execution.

⇒ Then,
Serial Schedule is always Consistent.

# Parallel	# Schedule
T_1 ⇄ T_2	$T_1 \rightarrow T_2$
	$T_2 \rightarrow T_1$

* **Durability :→**
Whatever changes we made, they must be permanent. i.e.
(update for lifetime until we again update the data).

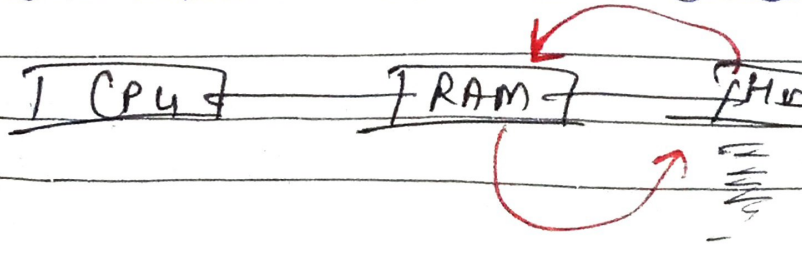
⇒ That's why, we save data in HDD for Durability.

---

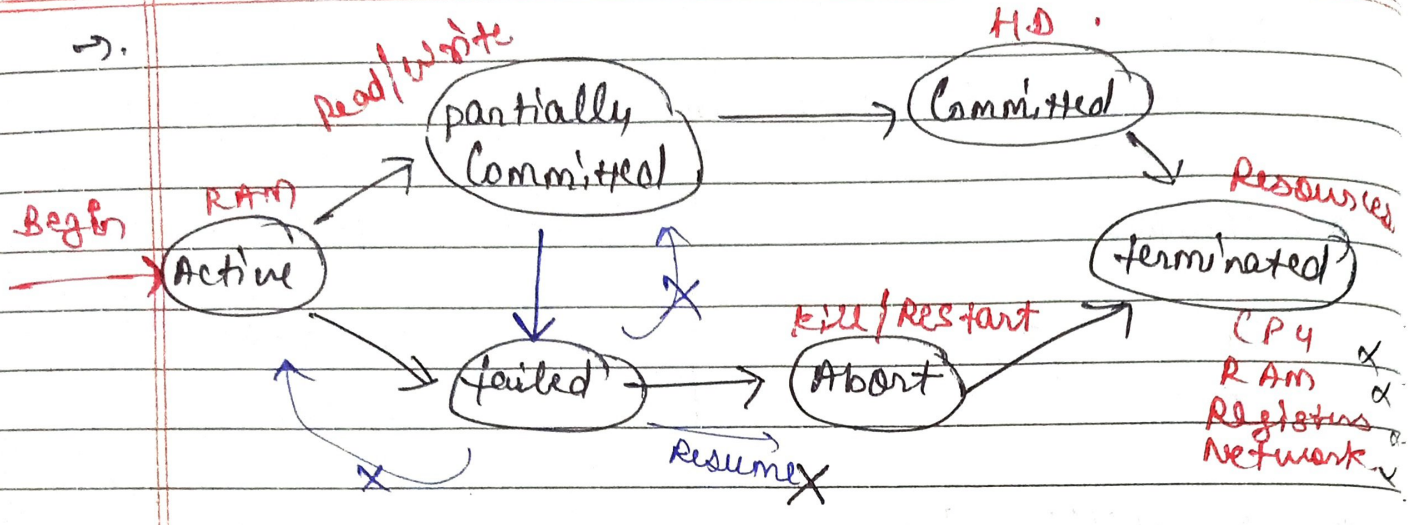
74. Transaction States :→

⇒ At now, Transac^n is in passive state. and as we start executing it, it comes into Active State.

* **In terms of O.S. :→** When we write a program in C++ & save it. Now, the program is in Hard Disk in ideal state. But when we start executing or compiling, it comes into RAM that we called ACTIVE STATE.



→.



→ partially Committed! →
All oper's are done except Commit.
ie. If N operation.
then
(N-1) is done.

All opera's are Stored in local memory/ ram/ shared memory until now.

→ Committed! -
Changes are now saved to Hard Disk.

→ terminated! →
Here, we deallocate our Resources
ie. free all Resources. bcz,
(Resources are limited).
Now, they move to anyone else.

→ failed: → power failure, switch damage, etc.
(failed either from Active or partially committed State)

→ Abort! → Rollback the opera's & Restart the opera's.

---

25. SCHEDULE! → (Serial vs parallel Schedule)

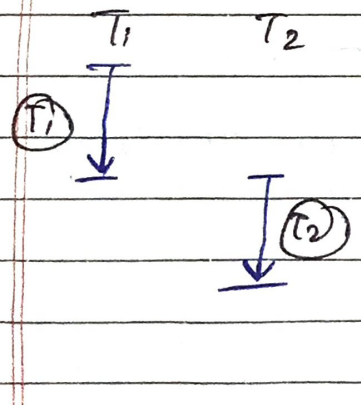
→ Schedule! → It is chronological execution sequence of multiple transactions.

T_1 T_2 T_3 — — — T_n

C, What is the sequence that these transac. are getting executed, that's called Schedule.

Serial Schedule! → Until a transac gets completed, no other trans. can interfere.

All transca's executed by a serial sequence.



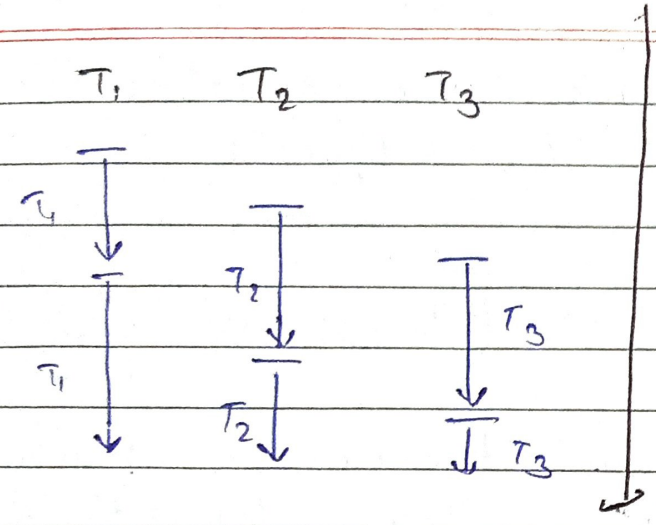
Advantage →
→ Consistent (bcz, no one can interfere).

Disadvantage →
→ When T_1 gets executed, T_2 & T_3 are in waiting.
→ Performance degrade.

Parallel Schedule! →
→ We can switch to multiple transac's at a time. i.e,
→ Multiple transac can execute at a same time.

Ex!→ Banking Online System! → Multiple users can use at a time.

Throughput → No. of transac'ns executed / time.



Throughput ∝ performance.

* Advantage :→
→ performance Increased. i.e, (Throughput is high.)

* Disadvantage :→
→ probln may occur. (Inconsistent)

(*) Nowadays,
(Parallel Schedule is more performed.)
↳ bcz, good performance in less time.

(*) Types of problems in Concurrency :→

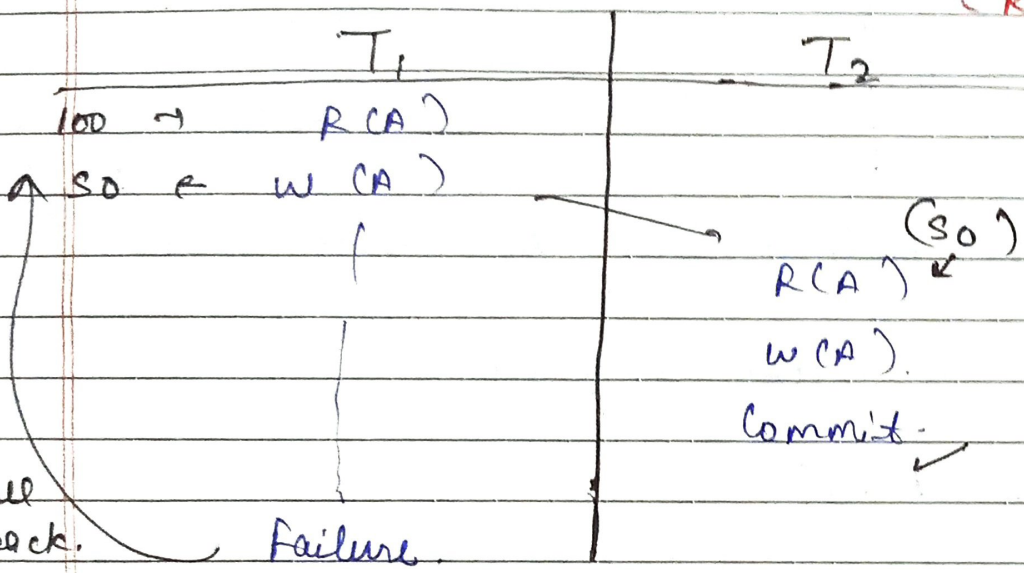
→ Concurrency, means when multiple trans. executed at a same time i.e, Parallel schedule.

→ (Mean, No problem is in Serial Schedule. There are Problems only in Parallel Schedule)

1) Dirty Read
2) Incorrect Summary

3.] Lost update
4.] Unrepeatable Read
5.] phantom Read.

1.] Dirty Read :→ Or Uncommited Read or RAW.
(Read after write)



Hence, when T_1 gets failed. Then, how T_2 can use the A = 50. So, Dirty Read.

2.] Incorrect Summary problm :→ (T_1)
It occurs mostly when a transac^n start & T_2 comes & start performing its aggregate func^s.
(Sum, average)
then,
we get incorrect value of Sum, average etc.

3.] Lost Update :→

⇒ जो पहले (T_1) ने changes किये / उस changes को (T_2) ने ओर update कर दिया / & पुराने वाले changes हमेशे lost हो गए / i.e, update किया हुआ lost हो गया /

Ex:→ Cid-1 T_1 | T_2 Cid-1
5 likes | 5 likes

⇒ & we finally get 5 likes on the same product instead of 10 likes.

Phantom - Glimpse, Shadow.

4.) Unrepeatable Read :→

T_1	T_2
R(x) → 100	
	R(x) → 100.
So ← W(x)	
	R(x) → 50.

// Unrepeated Read.

⇒ T_2 got diff. values on diff. Read. (100 & 50).

So, It is also a problem.

5.) Phantom Read :→

T_1	T_2
R(x)	
10 ←	
	R(x) → 10.
Delete (x)	
(0) ✗	→ Read(x). ('It also tries to Delete '0')

(7) Read - Write conflict (Problem) (OR) Unrepeatable Read Problem :→

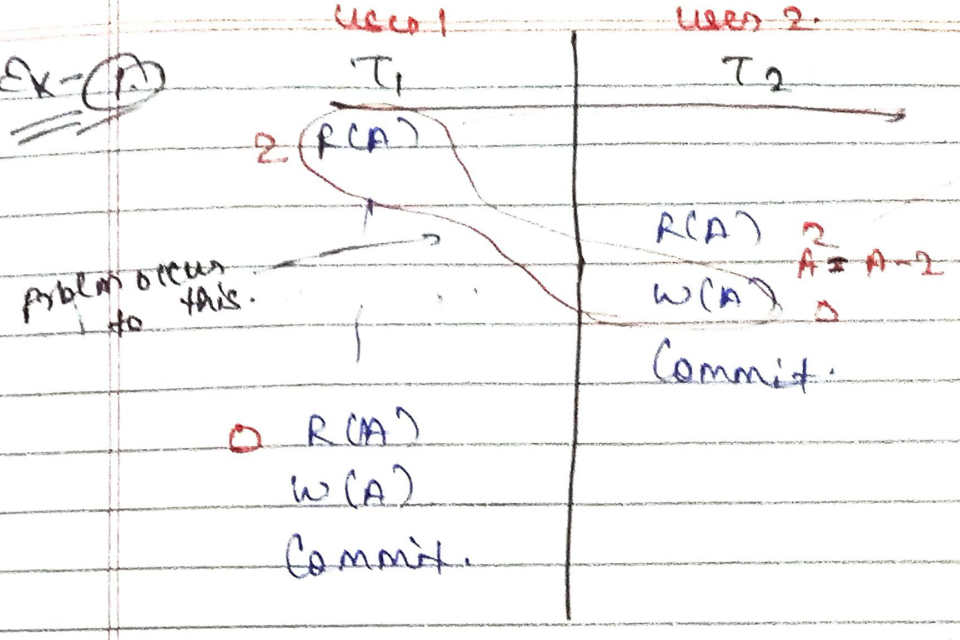
⇒ 4 Cases are there →

Same Data.

R (A)	R (A)
R (A)	W (A)
W (A)	R (A)
W (A)	W (A)

Problem.

Ex-(1)



(*) Ex - IRCTC

⇒ Let user 2 reserve both the seats. Then,

$A = A - 2 \quad = 0$.

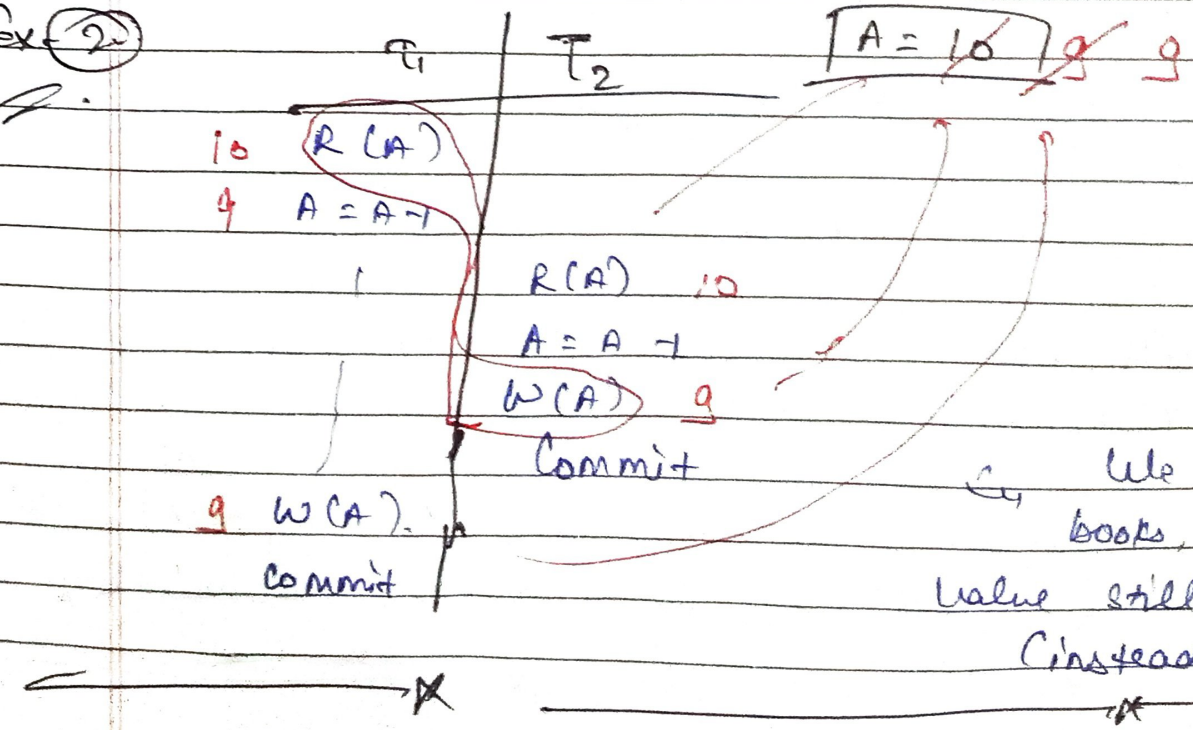
→ & user 1 remained in thinking to reserve or not. & when he sees again,

Now, Seats becomes '0'.

So,

Now, user 1 have to be Roll back. (Abort)

Ex-(2)

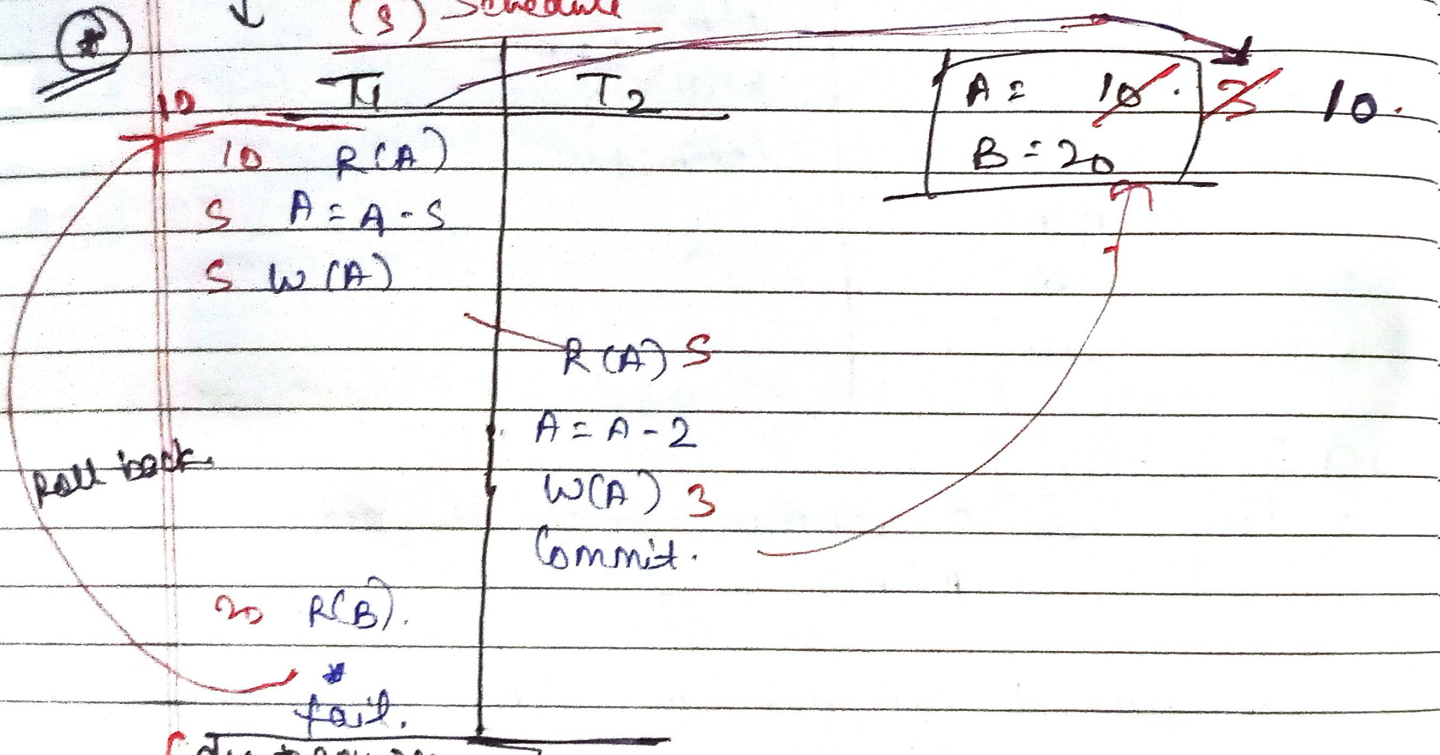


Date :
Page :

78. Irrecoverable Vs Recoverable Schedules in Transactions :-

↓

(1) Schedule



↳ Now, (T_1 Rolls back due to Atomicity property).

↓

(Either all or None)

i.e.

On Roll back, everything happens in T_1 is gone. So,

Again Now, A is 10.

i.e.

But, here T_2 also done something & that is Lost Now.

⇒ T_2 Change is lost. We can't recover it now. ∴ It is Irrecoverable Schedule.

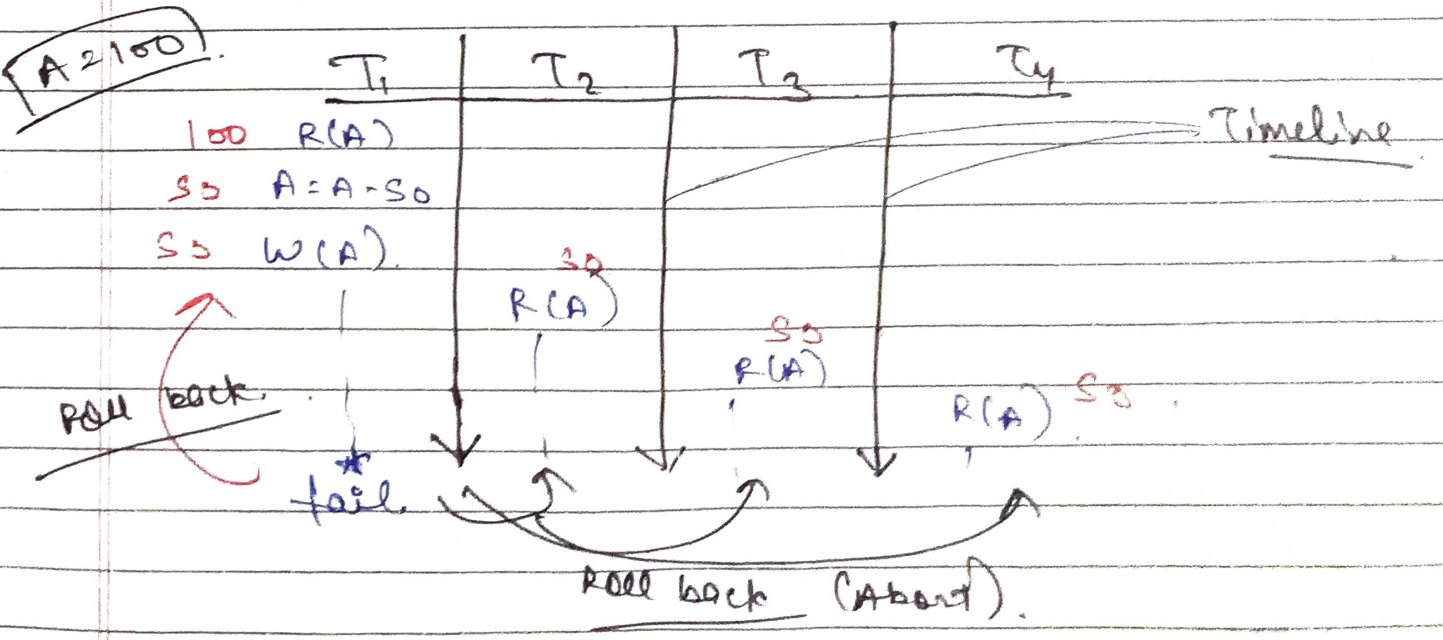
Cascade → झरना

79. Cascading (Vs) Cascadeless Schedule :→

↳ In Recoverability, these 5 are Important :→

1.) Recoverable
2.) Non-recoverable
3.) Cascadeless
4.) Cascading
5.) Strict Recoverable.

⇒). Cascading :→ Means Due to occurence of one event, multiple Events are automatically occuring.



↳ Here, let T_1 fails due to any reason. Then, it roll backs automatically due to Atomicity & again (A is 100). But, now T_2, T_3, T_4 has (A → 50). So, they are working on wrong data. So, now, We also forcefully Roll back (Abort) the T_2, T_3 & T_4.

So, This is Cascading.
(If T_1 fails, then we also have to roll back T_2, T_3 & T_4).

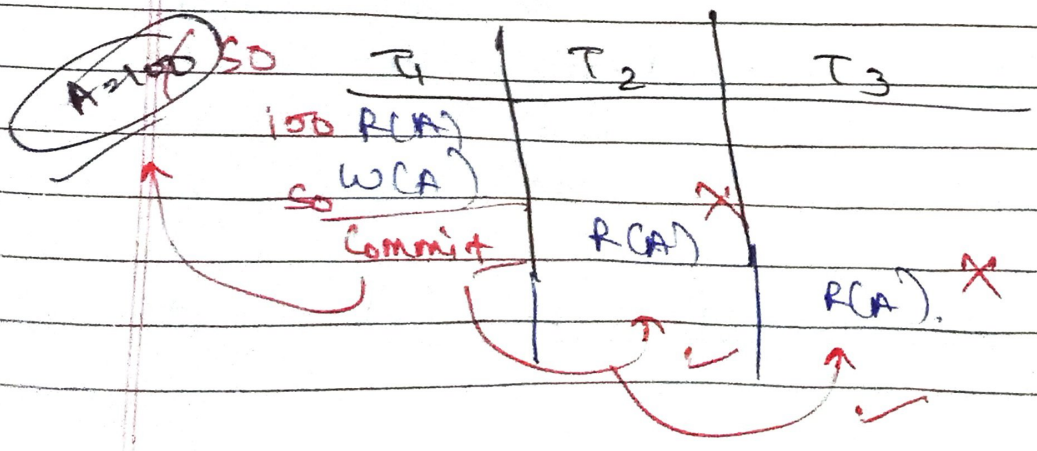
→ Here,
CPU utilisa^n gone waste by (T_2, T_3 & T_4).
∴
performance is bad/poor.

Ex:- If a Intelligent student (S_1) removes a answer. Then, other 3 students (S_2, S_3 & S_4) also cuts that answer. Bcz. it is wrong. So, their work has gone waste. Cascading.

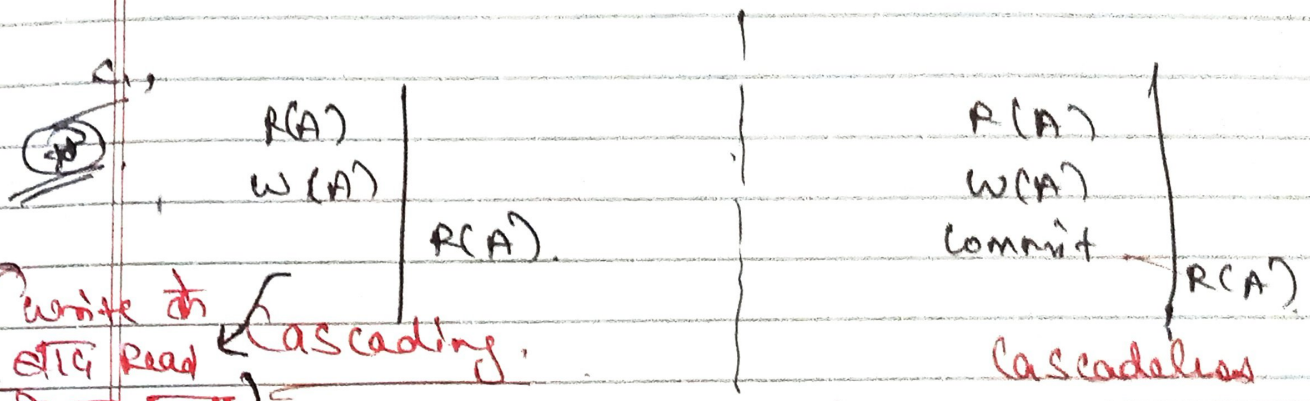
* Cascadeless :→

→ How to remove the problem of Cascading?

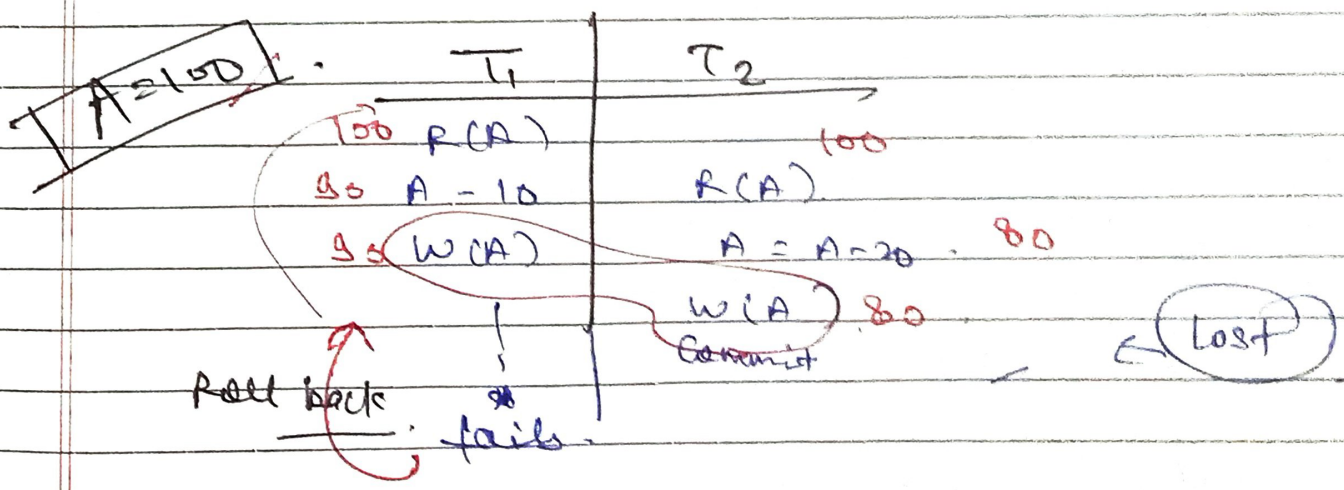
→ Sol^n:- T_2 & T_3 can't Read the (A) value from T_1 until A value gets Committed or Roll back in T_1.
&
(Commit → होने के बाद तो कोई Problem ही नहीं है।)



→ i.e, Don't allow Read in T_2 & T_3.
So,
It automatically becomes Cascadeless.



→ But, there is still W-W (Write-write) problem in Cascadeless. bcz,
(Read allow नहीं है, write तो कर सकता है।)



→ Now, when T_1 fails. (A becomes 100 again)
&
The work of T_2 automatically gets lost.
(i.e, Write-Write problem (or) lost updation problem
bcz T_2 value of W(A) → 80 तो कहीं पे Reflect होना ही नहीं। ∴ finally A = 100

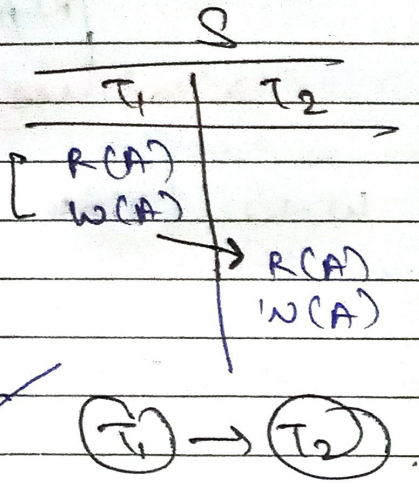
⇒ Strict Recoverable tells that we also can't write along with Read.

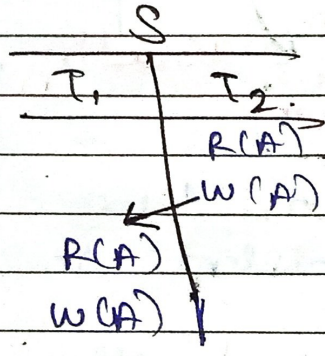
SERIALIZABILITY :→

→ (Serializability means that a Schedule has ability to become a Serializable.)

Mean,

Schedule - Collection of transaction ($T_1, T_2, \ldots T_n$)





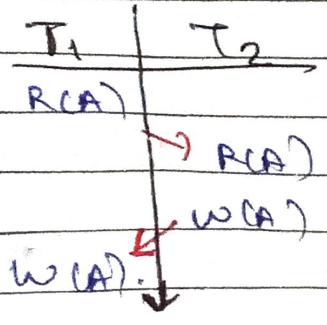
$T_1 \rightarrow T_2$

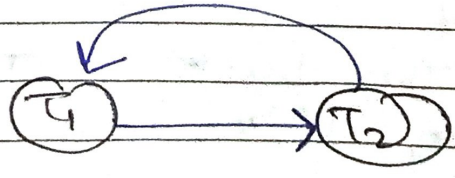
$T_1 \leftarrow T_2$

By seeing these, we can tell that they are already in serial Schedule.

(तो, Serial Schedule तो ये already है, इसको कुछ नहीं कर सकते।)

→ We want, Parallel Schedule →





→ Convert it to Serial Schedule →

2 ways :-

1) $T_1 \longrightarrow T_2$

(OR)

2) $T_2 \longrightarrow T_1$

⇒ To check Serializable? Check that if there Exists an Serial Schedule Equivalent to Parallel Schedule, or not. This Concept is known as Serializability.

(#) Serializability (2 method)

- Conflict Serializable
- View Serializable

→ We check that a parallel Schedule can convert to a Serial Schedule or not. → Serializable (clone of 11 schedule).

(#) Let, a Schedule has 3 transac^n's.

S

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

parallel Schedule →

⇒ Serial Schedule

6 ways !

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

⇒ If we able to convert that parallel schedule in any of the 6 forms of serial schedule, then we can say that it is a Serializable.

⇒ Why we get 6 ~~forms~~ ! →

we have 3 values (T_1, T_2 & T_3).

So,

$3! = 6$ ways.

---

(1) Conflict Equivalent Schedules ! →

R(A)	R(A)	} Non - Conflict pair.
R(A)	W(A)	} Conflict pairs
W(A)	R(A)	
W(A)	W(A)	
R(B)	R(A)	} Non - Conflict pair.
W(B)	R(A)	
R(B)	W(A)	
W(A)	W(B)	

(2) How to Convert : →

If we have adjacent non - conflict pair, then swap their pair.

Ex :

T_1	T_2
	R(A)
R(B)	

⇒

T_1	T_2
R(B)	
	R(A)

Q:- To check Conflict Equivalent :-→

$S \equiv S'$ → check

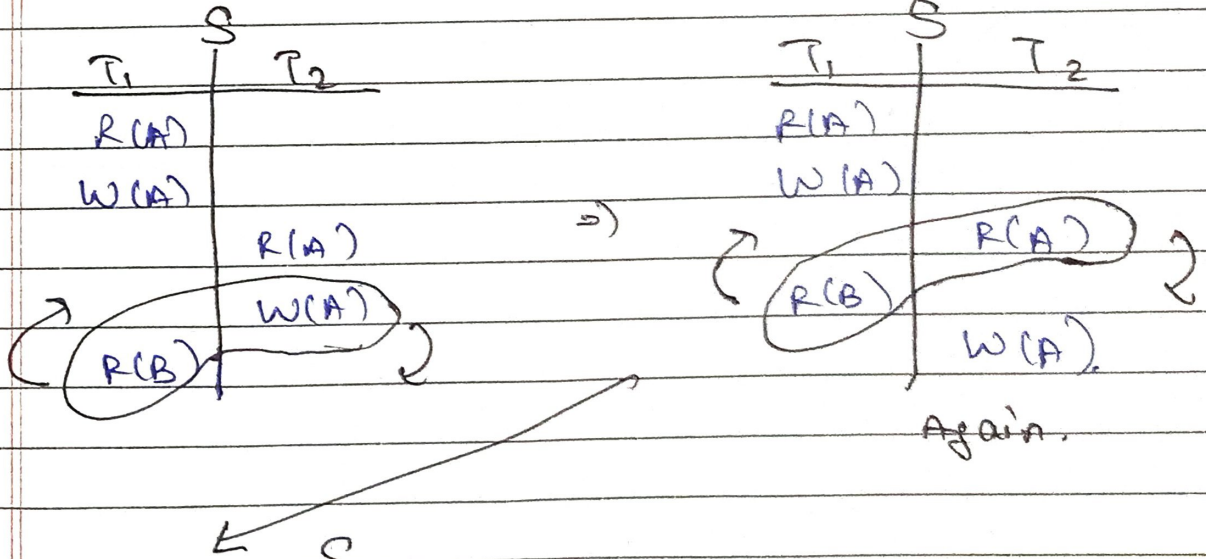
S

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

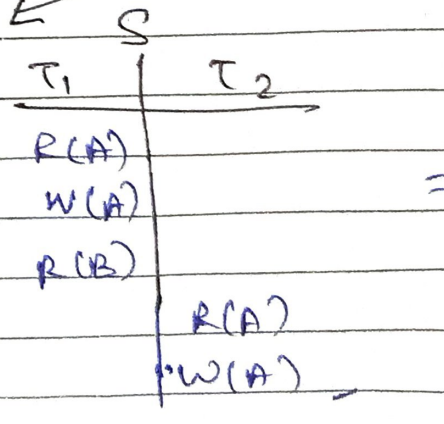
S'

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

Sol":- So, In S, we have adjacent non-conflict pair, so swap them.



Again.



Now Serial ← Schedule.

$= S'$

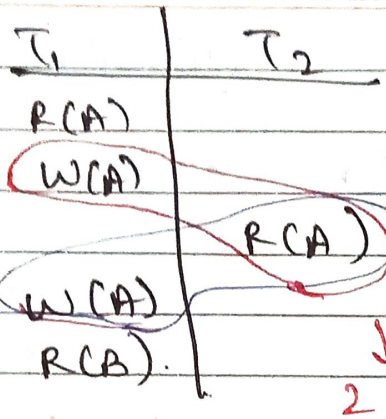
Hence,

$S \equiv S'$

Hence.

S & S' are conflict Equivalent Schedule.

Ex:-



2 adjacent pairs, But they are conflict pairs. i.e, no change in positions.

Note:-> $S \xrightarrow{CE} S' \longrightarrow$ Serializable i.e, (Serial Schedule)

(CE: conflict equivalent)

(i - 81) videos end here.