Jai Shree Ram

INDEX

AME Anurag Singhal UE: DBMS STD: ___ SEC. ___ ROLL NO

82- Conflict Serializability , Precedence Graph :-

Q:-

T1	T2	T3
R(x)		
		R(y)
		R(x)
	R(y)	
	R(z)	
		W(y)
	W(z)	
R(z)		
W(x)		
W(z)		

— Timeline.

→ First, we have to make precedence graph → (mean, just graph with edges & vertices).

Vertex → No. of Transac^n's (T1, T2 & T3).

Now

→ Edges?

Check the conflict pairs in other transactions and draw Edges.

Like → T1 → (Check in T2 & T3).

* Conflict pairs :-

R-W W-R W-W

of same variable let, (x).

* Start :-

Now, Start with first opera^n. i.e,

In T1 → R(x

& check the conflict pair of R(x) in other Transac^n's i.e. (T_2 & T_3).

→ Conflict (C.P.) pair of R(x) → w(x)

&

now, Similarly,

conflict pair of w(x) → R(x), w(x).

→ Now, do check for every opera^n's in all transac^n's & after checking cut those opera^n. & If you find any conflict pair, then draw edge acc. to vertex (Transac^n).

→

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)~~		

C.P. we find →

R(x) - w(x) ($T_3 \to T_1$)

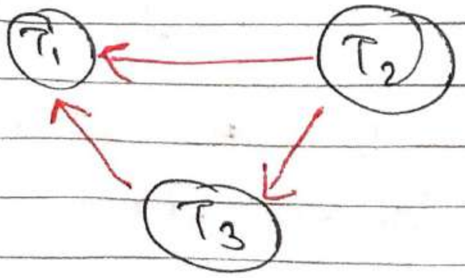
R(y) - w(y) ($T_2 \to T_3$)

R(z) - w(z) ($T_2 \to T_1$).

w(z) - R(z) ($T_2 \to T_1$)

w(z) - w(z) ($T_2 \to T_1$)

Precedence Graph:-



Now, Come on Graph :-

check that if there is any Loop / Cycle in the Graph.

How to check cycle :→ अगर हम एक किसी भी Node से चलें, तो क्या हम घूम के वापस आ सकते है। If yes → Loop / Cycle Exists.

(It is not must that the cycle must covers all the nodes. May be possible, it comes just after visiting one node). → also a cycle.

(# In our graph, there is no loop / cycle).

No Loop / Cycle → Conflict Serializable Schedule
If Loop / Cycle Exists → Not C.S.S.

→ Conflict Serializable → Serializable → Consistent
(Serial Schedule)

* Now, how to make Serializable ! →

$T_1 \to T_2 \to T_3$
$T_1 \to T_3 \to T_2$
$T_2 \to T_1 \to T_3$
✓ $\boxed{T_2 \to T_3 \to T_1}$
$T_3 \to T_1 \to T_2$
$T_3 \to T_2 \to T_1$

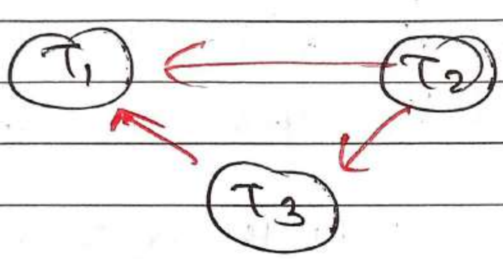
6 Cases → Now, which Case ?

Again, See the graph & check

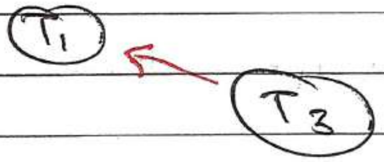
$\boxed{\text{Indegree} = 0}$

Indegree = 0, means a vertex with '0' indegree.

Mean, जहाँ vertex पर कोई भी Edge (arrow) नहीं आ रहा हो।



T_2 has, Indegree = 0,
Now, remove T_2.



∴ $\boxed{T_2 \rightarrow T_3 \rightarrow T_1}$ Ans.

T_1	T_2	T_3
	O	
		O
O		

———x———x———

83. View Serializability :→

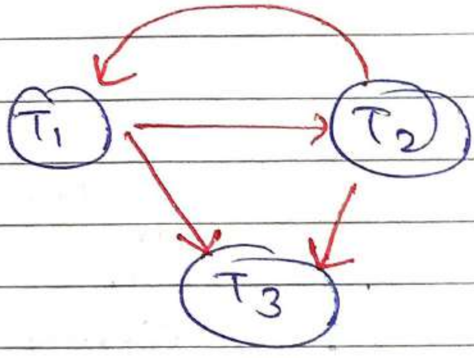
Q!- Check whether schedule is conflict Serialisable or not?

→ First make precedence graph.

Loop → No Answerable. then, view serializable answers.

S

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



→ Here, we get a loop.
So,

It is Non Conflict Serializable.

↳ Here, we can't tell that it Serializable or not.

↳ Now,
(*) To check that, we use View Serializability.

→ Now, we arrange this Table,

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

We change the posi^n →

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

↓

Now this is a serial schedule. $T_1 \to T_2 \to T_3$.

But,
We have to check whether they match each other or not.

Let, A = 100.

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

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

Now, apply this same here.

Here, finally A value - 0

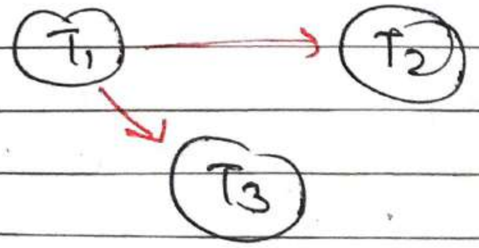
Here also finally A is '0'.

Hence, Both are Equivalent.
They are View Equivalent (≡).

Note!- bcz, finally output is given by A of T_3. So, If adjust the posⁿ of A. of T_1 & T_2. So, there is no problem.

∴

Now,



(By : Referring of only T_1)

Hence, $T_1 \rightarrow T_2 \rightarrow T_3$ (By Table).

Note!- Conflict Serializable tell that it is not Serializable (no ans.). But, View Serializable checks it & tell that it is Serializable.

(parallel schedule)

84. Concurrency Control Protocols! → (C.C.P.)

(Shared - Exclusive Locking Protocol)

⇒ C.C.P. is that how to make them serializable, or how to make them recoverable Scheduls are concurrent, but how we can make them serializable & recoverable, this comes under Concurrency Control Protocol (C.C.P.).

→ We achieve this, By using Locking Protocols.

(#) 'Shared - Exclusive locking' ! →

We use 2 locks here,

→ Shared lock (S) → if trans. locked data item in shared mode, then allowed to Read only.

$\boxed{U \rightarrow \text{means unlock}}$.

T₁	
S(A)	→ Shared lock.
R(A)	→ only Read
U (A)	→ unlock.

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

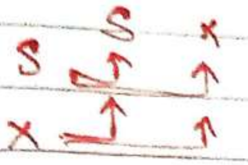
T₂	
X(A)	→ Exclusive lock.
R(A)	
W(A)	
U (A).	→ unlock.

* Compatibility Table :→

→ Request

↓ grant

	S	X
S	Yes	No
X	No	No .



Expla^n :-

S has only R(A)

& X has both R(A), W(A).

So,

⇒ S - S (No Conflict) b/w R(A)-R(A).

⇒ S - X (Conflict).

R - R

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

⇒ X - S (Conflict)

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

⇒ X - X

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

→ Conflict.

* Problems in S/X Locking :→

1.) May not sufficient to produce only Serializable Schedule.

2.) May not free from Ir-recoverability.

3.) May not free from dead lock.

4.) May not free from starvation.

85. Drawbacks in Shared | Exclusive :→ S/X Locking protocol

⇒ Locking gives us Serializable Schedule. → Consistent.

1.) May not sufficient to produce only Serializable Schedule.
→ हो भी सकता, नही भी।

Ex! →

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

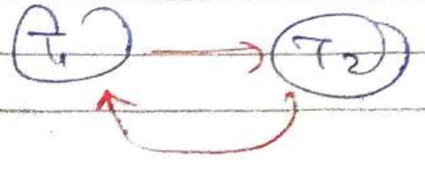
concurrent (parallel) →

~~try to~~ Convert to → T1 → T2 (or) T2 → T1

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

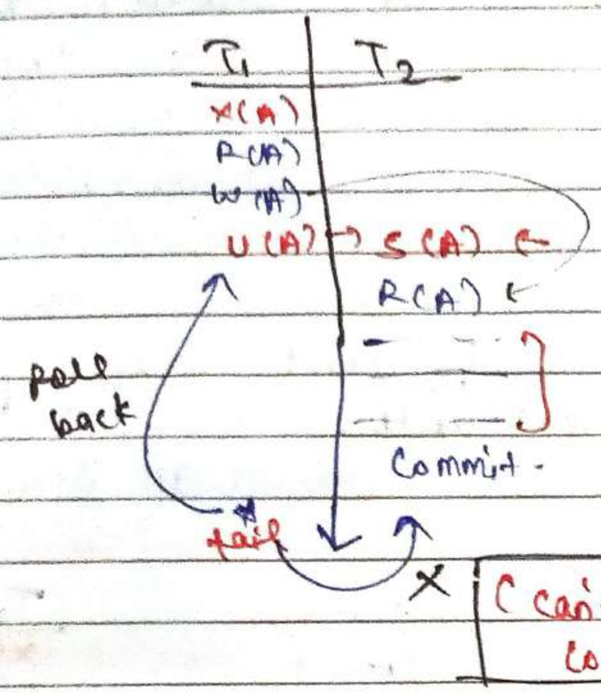
Note:→ Until, we do unlock, U(A) in T1, we don't be able to put Shared lock S(A) on T2. bcz, by Compatibility Table. [(X-S) → No], so first we unlock X(A). & then, use S(A) on T2 data.

⇒ Here, we don't get Serializable Schedule even after appling S/X locking. As u can see in Table.



2.) May not free from Ir-recoverability.

Ex!-



T1	T2
X(A)	
R(A)	
W(A)	
U(A)	S(A)
	R(A)
	—
	—
	Commit.
fail	

//Dirty Read.

X (Can't roll back, bcz committed.)

∴, It can't recover here.

3.) May not free from dead lock.

→ Dead Lock! → When 2 person wait for resources, & they both are waiting in an infinite loop, So, when we wait infinitely, it is called Dead Lock State.

G - Grant
W - Wait

Ex!-

	T1	T2	
G	X(A)		
	↓	X(B) G	← (bcz A & B are diff.)
W	X(B)	↓	
		X(A). W	

→ Here, both are in waiting bcz, They are not unlock after use. So,

→ T_1 also waits that when T_2 unlocks, he can use on X(B).

Same

→ T_2 also waits that when T_1 unlocks, he can use on X(A).

So, Both are waiting in an Infinite loop.

4.) May not free from Starvation.

[In deadlock, waiting upto Infinite time
But,
In Starvation, waiting not upto the Infinite time.]

a) [Shared के ऊपर Shared दे सकते है, without unlock
. by compatibility Table.]

Ex:-

	T_1	T_2	T_3	T_4
		S(A)		
W	X(A)			
			S(A) G	
		U(A)		
				S(A). G
			U(A)	
				U(A).

Waiting Time.

S → Shared Lock
X - Exclusive "
U - Unlock.

So, here, T_1 waits for X(A) until T_4 unlocks.
So, here Starvation also. (by Shared के ऊपर Exclusive, बिना unlock के grant नहीं होता।)

86. phase Locking (2PL) protocol in Transac^n Concurrency Control :→

⇒ 2PL (2 phase locking) is just the Extension of simple Shared/Exclusive locking.

We just do modifications in them.

2-Phase locking (2PL) :→

→ Growing phase :→ Locks are acquired & no locks are released.

→ Shrinking phase :→ locks are released & no locks are acquired.

[S/X]

T₁
X(A)
S(B)
R(A)
W(A)
R(B)
S(A)
R(C)
S(D)
R(D)
} Growing phase

U(A). ← Shrinking. } Shrinking phase

Note! When Growing phase starts, then only apply locks.

and,

When we first unlock, ie, Shrinking phase starts, then we only <u>unlock</u>. (and not able to apply any lock).

⇒ We achieve Serializabily by this, that we don't achieve in simple Shared/Exclusive protocol.
So,
we made (2-PL) protocol for this.

Ex!

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

(Growing phase.)

(Now, only unlock. ie, shrinking phase)

→ (Now, T₂ can get S(A)).

Hence, we first starts T₁, & if T₂ comes in b/w then we don't entertain him. First, we complete T₁ & then goes to T₂.

⇒ T₁ → T₂,

Serlability achieved.

↓

Consistency automatically achieved.

• Imp.

Note! → The transac^n which follow (2-PL), then it is always serializable.

Ex! → यहाँ पर भी Shared के ऊपर Shared तो दे ही सकते हैं। Means,

While T_1 is in growing phase by using S(A) [Shared Lock], then at same time T_2 also starts its growing phase by using S(A). bcz, by Compatibility Table.

Ex!

S

	T_1	T_2	
Growing	Lock S(A)		Growing
		Lock S(A)	
	Lock X(B)	.	
Lock point →	⊛		
Shrinking →	Unlock (A)		
		lock X(B)	
	Unlock(B)	⊛ →	Lock Point
		Unlock (A)	← Shrinking
		Unlock (B).	

Now,

* Serializability Schedule, How formed?

ie,

$T_1 \rightarrow T_2$ (OR)

$T_2 \rightarrow T_1$

So,

This is done by (LOCK POINT).

Commit के बाद Roll back. नहीं कर सकते।

=) [Lock Point ! → where trans. is taking the last Lock. (or) where trans is unlock first time.]

=) जिसका Lock Point पहले आ जाएगा। वो trans. पहले आ जाएगी।.

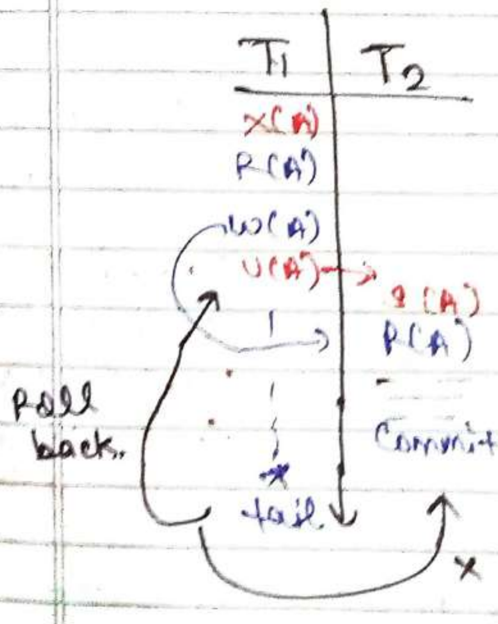
i.e, [$T_1 \rightarrow T_2$]

(87.) Drawbacks in (2-PL) protocol ! →

→ Advantage ! Always Ensures Serializability.

=) Drawbacks ! →

(1.) May not free from Ir-recoverability.



T_1	T_2
X(A)	
R(A)	
W(A)	
U(A)	
	S(A)
	R(A)
	Commit
fail.	

→ (We can't roll back it, bcz it is already commited)

Hence,
It is Ex. of Ir-recoverable Schedule.
We can't recover it (Roll back it).
(reversal).

(2.) Not free from Cascading Rollback →.

Ex:-

T1	T2	T3	T4
X(A)			
R(A)			
W(A)			
U(A)			
	S(A)		
	R(A)		
		S(A)	
		R(A)	
			S(A).
			R(A)
fail.			

//Dirty Read.!

(Roll back them all).

→ (bcz we don't do commit here, ∴ Rollback possible)

∴ Bad performance. & Cascading Rollback is possible here.

(3.) Not free from Deadlocks.

G - Grant
W - Wait.

	T1	T2	
G -	X(A)		
		X(B)	-G
W -	X(B)		
		X(A).	- W.

(किसी ने भी unlock नहीं किया अभी, ∴ दूसरे waiting में ही है)

∴ Infinite waiting here

(T1 & T2 both waiting here to complete their transac's.).

(4.) Not free from Starvation.

(wait for limited time).

In 2PL
Here, the problem of Serializability removes.

T1	T2	T3	T4
	S(A)		
W — X(A)			
		S(A)	
	U(A)		
			S(A)
		U(A)	
			U(A).

wait

88. Strict 2PL, Rigorous 2PL & Conservative 2PL Schedule :→

→ These are the Extension (advance) of basic 2PL :→

Strict 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.

Ex:-

T1	T2	T3
X(A)		
R(A)		
W(A)		
	S(A)	
unlock — U(A)	R(A)	
		S(A)
		R(A)
fail		

roll back

→ (also have to roll back T2 & T3

i.e. (Cascading roll back)

So, To stop this →.

→ (We have to unlock Exclusive lock after Commit.)

then, this problm of Cascading removes.

⇒

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

H.D.

{ bcz, after commit in T1. The T2 takes data from database (H.D) & not from W(A) of T1 so, no dirty Read. }

Hence, Here, Cascading Rollback is Removed, & It will always produce Cascadeless.

⇒ Now, ∀ Ir-recoverability :→

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

→ (Same data i.e, A)

Roll back.

→ (but we can't roll back it, bcz it is already committed.)

∴ Ir-recoverability.

→ Hence, हम इसको पहले unlock ही नहीं करेंगे।

We unlock it after commit is done.

→ (Hence, T_2 को S/X कुछ भी नही मिलेगा, जब तक T_1 Comit नही हो जाता |)

और अगर Comit होने के बाद S/X मिल रहा है (T_2 को), then No problm.

→ (Hence, Irrecoverability removes here.)

Comit
U(A) → unlock

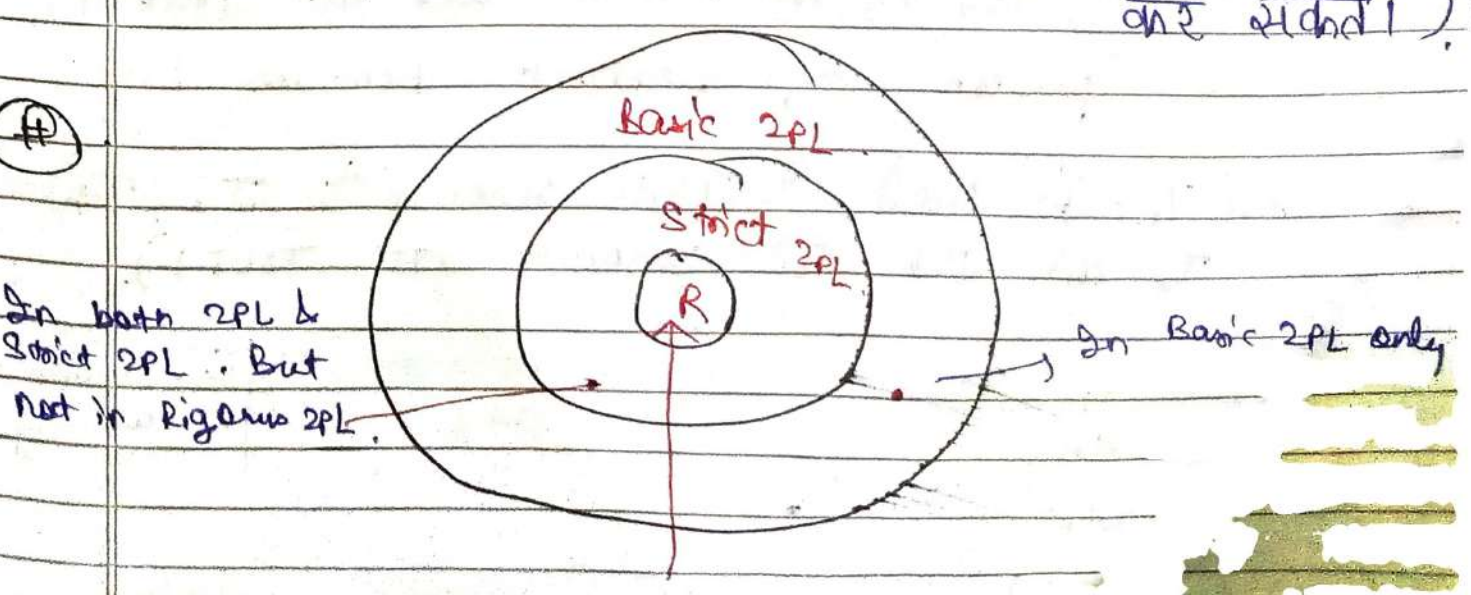
Bcz फिर वो (T_2) Database (H.D.) से ही Read करेगा |.

→ It produce Strict Recoverable Schedule.

Note!→ 2 problems removes here →
1.) Cascadeless
2.) Strict Recoverable

→ Rigorous 2PL (R), बस और Strict हो गया | Bcz इसमें S/X दोनों आ गए |.

(इसमें हम Shared Lock को भी Release नहीं कर सकते |)



- But, problm of deadlock & starvation still are there.

⇒

T_1	T_2
X(A)	
	X(B)
W - X(B)	
	X(A) - W

→ Still, Deadlock.
→ Infinite waiting.

⇒

T_1	T_2	T_3
	S(A)	
W - X(A)		
waiting		S(A)
	C	
		C

⇒ finite waiting
→ Still, Starvation.

Conservative 2PL :-

practically, It is not possible. In this, कि कोई भी Trans. start होने से पहले ही सारे Lock लेले। (A, B, C → सब पे लेले।).

⇒ इससे T_2, T_3 को एक भी Lock नहीं मिलेगा।

∴ problem of Deadlock Removes here.

→ (T_1 को पहले ही सभी resource दे दो, ताकि T_2 को एक भी resource ना मिले।).

→

T_1	T_2
G - X(A)	
	X(B) - W
G - X(B)	
	X(A) - W

G - Grant
W - waiting

∴, T_1 completes here first. so, no Infinite waiting. bcz,

(T_1 के comp. होने पर T_2 use कर होगा।)

∴,

No Deadlock here.

89. Basic Timestamp Ordering Protocol (T.S.) :→

→ unique value assign to every transaction.

→ Tells the order (when they enters into system)

Let,

10:00	10:10	10:15	→ Time
T_1	T_2	T_3	
100	200	300	→ Timestamp
↓ older (पहले आई)	↓ Younger (बाद में आई)	↳ youngest.	

T.S. (T_i)

→ Read_Ts (RTS) = last (latest) transaction no. which performed Read successfully

→ Write_Ts (WTS) = last (latest) transaction no. which performed write successfully

Ex1.

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

T.S. (T_i)

⇒ RTS (A) = 30

↳ bcz, latest Read is of T_3 & Time Stamp (T.S.) of T_3 is 30.

Ex! :

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

[ञ WT(A) = 20]

Rules :→

In Timestamp, let,

(जो Trans. पहले आई, वही पहले complete होगी).

ie,

[Serializability : older → younger.] → follows conflict serializable

→ If Read

1. Trans. T_i issues a Read (A) operaⁿ

a) If $WTS(A) > TS(T_i)$, Rollback T_i.

b.) otherwise Execute R(A) operaⁿ

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

$\left(\begin{smallmatrix}0\\200\end{smallmatrix}, \begin{smallmatrix}100\\300\end{smallmatrix}\right)$ → in next vdo use

→ Write

2. Trans. T_i issues Write (A) operaⁿ

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) operaⁿ.

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

Understanding these :→

Ex → (old) 100 | 200 (young)

T₁	T₂
R(A)	
	W(A).

✓ (NP).

(No Conflict).

T₁	T₂
W(A)	
	R(A)

✓

T₁	T₂
W(A)	
	W(A).

✓

bcz,

(old → young) as (T₁ → T₂). So,

(T₁ पहले Execute हो जाएगी, तो T₂ में कोई दिक्कत नहीं आएगी।).

Case I.

Ex': (old). 100 | 200 (young)

T₁	T₂
	R(A) -10.
(20) W(A).	
commit	

Roll back.

(✗). (Not possible).

(T₁ को होने नहीं देना।).

→ (bcz now ये T₂ में A-10, कौन सी value Read करके बैठी है। bcz, T₁ ने तो Database में 20 update कर दिया।).

Case II.

T₁	T₂
	W(A) - (20).
20 → R(A)	
commit	

Roll Back.

(Hence, ये 20 अब कहीं पे है ही नहीं। i.e, T₁ गलत Data पे काम कर गयी।).

// Dirty Read

(हम T₁ को Read नहीं करने देंगे।)

Case III.

100 T₁	200 T₂
	W(A) - 20
40 - W(A)	
commit	

Roll back.

∴, Roll back.
(∴, हम T₁ को allow नहीं करेंगे).

→ let,
(If T₂ fails, then, our updation will be lost).

|| Lost updation.

—— × —— × ——

90. Solve Ques^n on Timestamp Ordering Protocol:

We have 2 Rules →.

1.) Ist Rule is of Read → has only' Cond^n
bcz, only (R-W) conflict

2.) 2nd Rule is of Write → has 2 Cond^n's,
bcz (W-R, W-W) → 2 conflicts.

Now,
Q -

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

Roll back.

0 > 100
0 > 200
0 > 100]
0 > 100].
0 > 300
100 > 100
[300 > 200] ↳ Roll back.

100 > 300.
0 > 300.

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

=> Initially, all values are 'zero'.

(Final Table) Ans.

=> Check, for every values in the Table (Trans.) & put that values in Rules & then make the Table.

[→ for R(A) use Ist Rule of Read.
→ for W(B) use 2nd Rule of Write.]

=> first check R(A) of T_1, then
R(B) of T_2, then
W(C) of T_1, then.
R(B) of T_3, then,
R(C) of T_1, then
Here, Roll back. ← W(B) of T_2, then
W(A) of T_3

i.e, pattern wise as in Table

&

then, make the final Table of :
(RTS) & (WTS) Ans.

OR.

We also do the Ques^n direct, without using these 2 Rules with just the [older → younger] Concept & cases (we discussed in prev. video).

(Volatile – उड़नशील, परिवर्तनशील)

(91.) INDEXING :→

⇒ CPU → processor (who process).

→ Query comes to CPU. CPU process them.

⇒ Ex:-

| Select * from student where Roll no = 1; |

⇒ CPU has to execute it, But Data is in the memory.

⇒ In general architecture, we only let, 2 types of Memory →

1.) RAM – primary memory
2.) H.D. – Secondary Memory.

⇒ Ram is volatile, Ram works directly with CPU.

⇒ CPU Speed → In MIPS (Million Instn per Second)

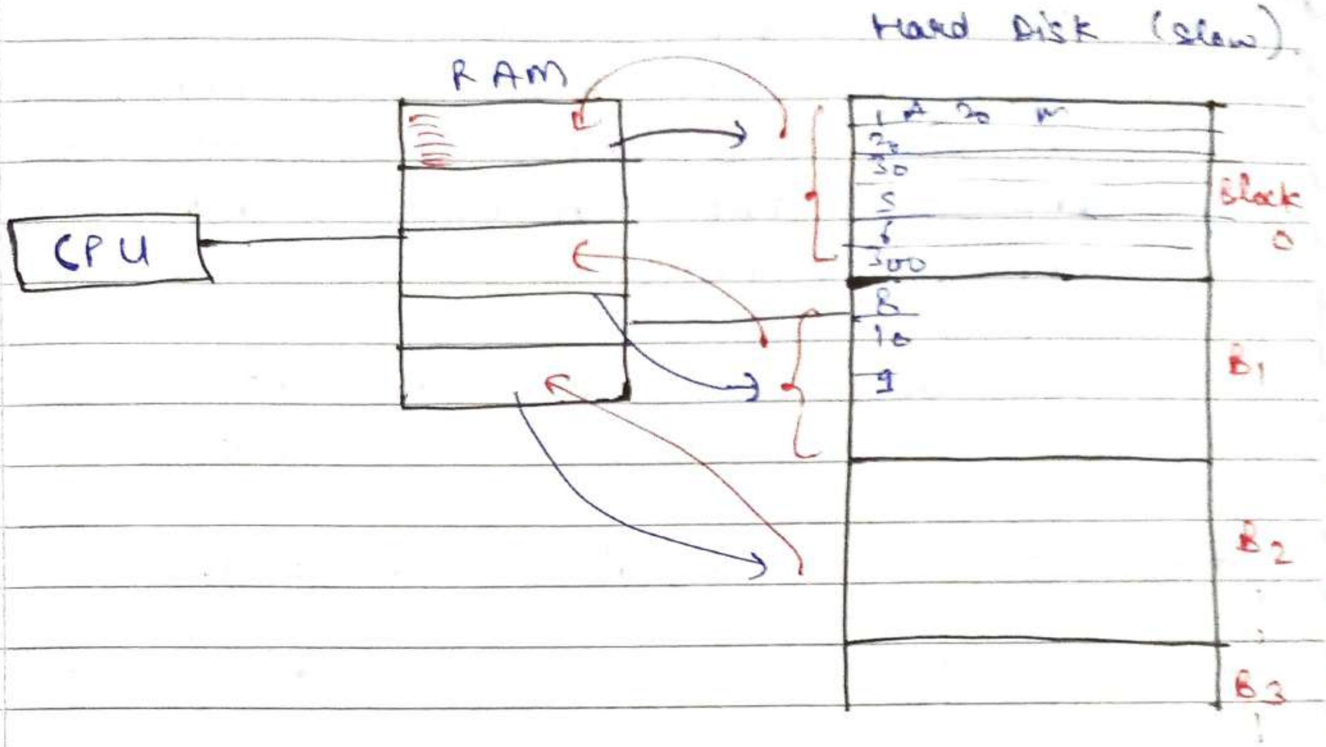
⇒ H.D. Speed ⇒ 10/20 Instruc^n per second.

∴ these both are not compatible with each other.

so,

⇒ Ram comes b/w them.

⇒ Hard speed counts in miliseconds.
CPU ——"—— nano/pico seconds.



We divide Hard Disk in Blocks.
Logical Blocks. (& not physical Blocks).

Ex:-

Logical Drives → C Drive, D Drive, — etc.

O.S. divides hard drive into fixed sized blocks & then Insert Data. (Blocks / pages)

Ex:-) If we have record of 10 k students
&
Block size is of 100 Records in Hard Disk.
so,
we need 100 Blocks in H.D. (100×100 =10k)

→ Now, data may also be stored in 2 ways -
1.) Sorted (ordered) 2.) Unsorted (unordered).

→ lets,

=> Unordered Data :→ then, we send 1-by-1 every block of H.D. into RAM. & if we get our data, then okk. Otherwise, RAM send back that block & next blocks come into RAM. & so on.

If we get data → HIT &
If we don't get → miss

(#) => So, Here, INDEXING is used.

(हम Ram में H.D. के Block का Searching time कम नहीं करते,).

=> We transfer data from H.D. to Ram. So, there is ofc transfer cost. i.e,

$\boxed{\text{I/O cost}}$ → Input / Output.

↓

हम Data को call कर रहे, इसको बोलते I/O cost.

=> & If call more blocks, then I/O cost more. & then time also increases of searching.

So,

(*) Indexing, that we have to call min^m no. of blocks. i.e, I/O cost is reduced.

Ex :→ Let, a book has 1000 pages. & we have to search a page.

Worst Case → 1000 (find in last).

Best Case → 1 (find on 1st page).

Average → 500 pages.

→ (If we use Index, then the no. of pages we shuffle are decreases.)

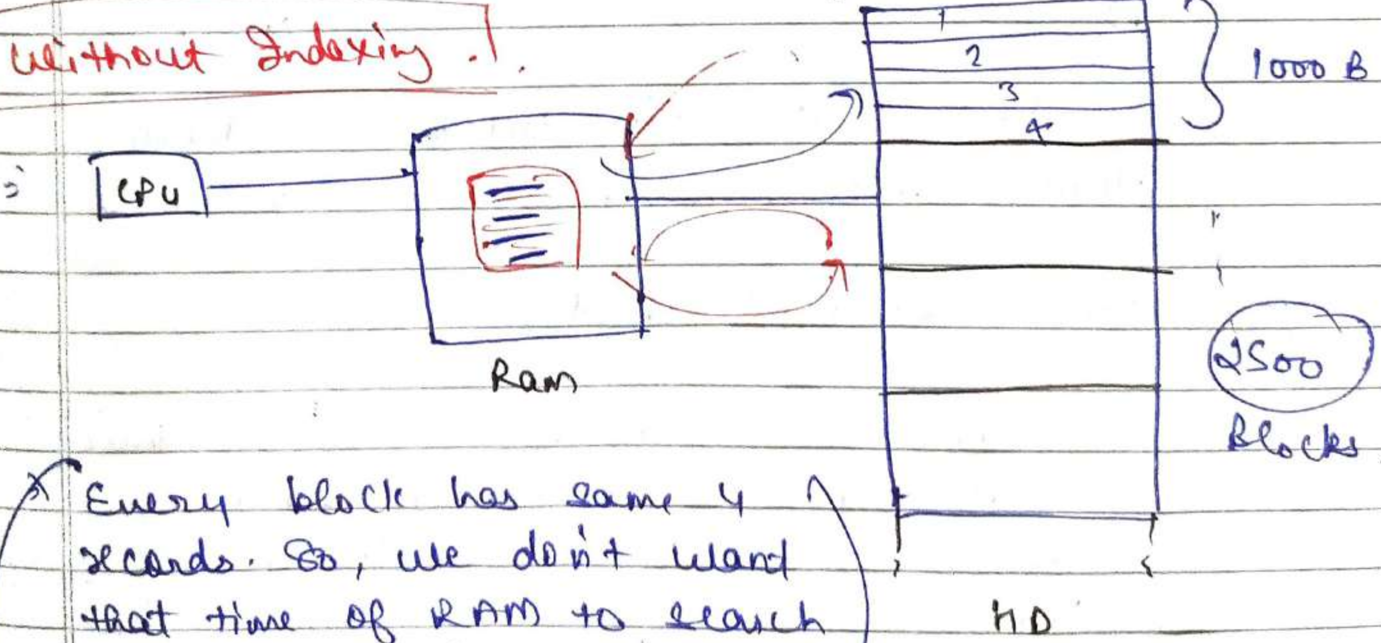
⇒ (let, Index is of 2-3 page, then just see that topic on Index & then directly go to that page).

92- Numerical Ex. on I/O Cost in Indexing :->

Q:-> Consider a Hard Disk in which Block size = 1000 Bytes, each record is of size = 250 Bytes. If total no. of records are 10,000 and the data entered in Hard Disk without any order (unordered).

→ What is avg time Complexity to search a record from HD?

[Without Indexing .]



(Every block has same 4 records. So, we don't want that time of RAM to search in a block. bcz, ie, always 4.)

⇒ No. of Records we can put in every block = $\frac{\cancel{1000}^{4}\ B}{\cancel{250 B}}$

$\boxed{= 4}$.

&

⇒ No. of Blocks Required = $\frac{\cancel{10000}^{2500}}{\cancel{4}}$

$\boxed{= 2500}$

⇒ I/O Cost :→

Best Case = 1

Worst Case = 2500 = $\boxed{N}$. Ans.

Avg Case = $\frac{2500}{2}$ ⇒ 1250 $\boxed{\frac{N}{2}}$.

$\boxed{\text{Avg Time Complexity : } O(N)}$ Ans.

& This is all for unordered Data. Hence, we used linear Search here.

Let, Ordered Data :→

then,

we use Binary search here, bcz it searches on sorted data (either ascending or descending).

& then,

Time Complexity $\boxed{: O(\log_2 N)}$ Ans

Here, ⇒ $\log_2 2500$.

⇒ $\boxed{12}$ approx ✓

Hence,
we need approx 12 blocks in ordered data.

So, this both case are <u>without Indexing</u>.

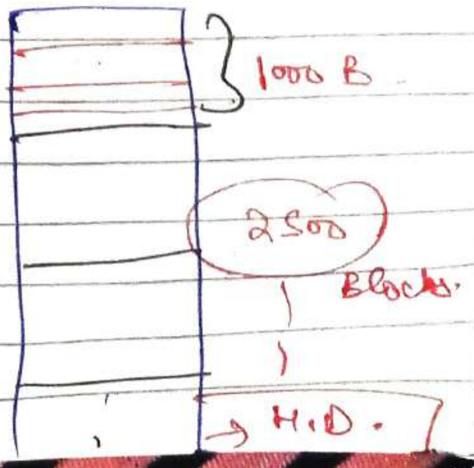
⇒ When we used Indexing, we even need less blocks than these.

[I/O cost is low.]

Q3) <u>Numerical</u> on <u>I/O Cost</u> in <u>Indexing</u> :-

Q:- Consider a H.D. in which Block size = 1000 Bytes, each record is of size = 250 Bytes. If total no. of records are 10,000 and the data entered in Hard disk without any order (Unordered Data). What is the avg time complexity to search a record from Index Table if Index Table entry = 20 B $(\underset{10B}{\text{key}} + \underset{10B}{\text{pointer}})$.

⇒ [2500 Blocks]



⇒ Index is also stored in the H.D. & when we search any data, then Index come into the Ram.

→ Then, we first search into the Index.

[Note:-] Block size in Index (page) = Block size in H.D. (page)

1000 B ↓ 1000B

⟹ ✓ understanding :- In a book, all pages are of same size whether it is Index page or anything.

(*) Index Table Entry = 20 B (key + pointer).

10B 10B.

key value → कुछ value होता जो search करते हैं (Topic name), Roll no. etc.

pointer → page. No.

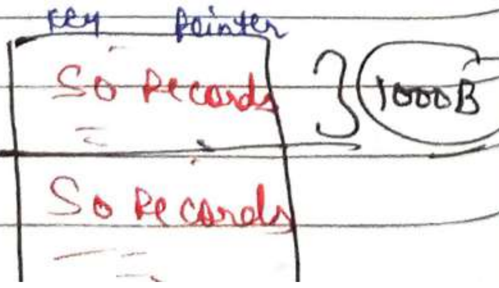
- Each block in Index Table contains Record = $\frac{\text{Block Size}}{\text{Index Entry Size}}$

$$= \frac{\overset{50}{\cancel{1000}}}{\cancel{20}}$$

$$\boxed{= 50}$$

Hence, we can put 50 records in a block of Index.

(bcz, contains only topic name ∴ ज्यादा आ जाएंगे).



⇒ How we can Enter records in an Index :→

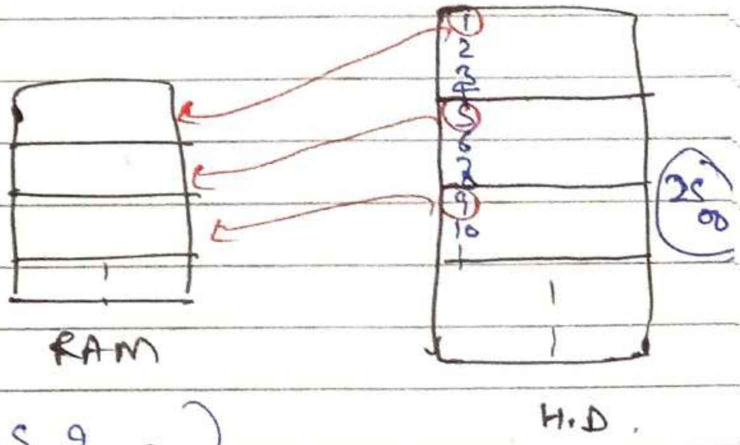
1.) **Dense** :→ used when unordered data.

Here, we have to put all the records of H.D. into the Index.

Ex:→ Here, all 10k records are to be Entered in Index.

2.) **Sparse** :→ used when data is ordered / sorted.

Here, we just Enter 1 record from the block into the Ram. Like, we entered Anchor data (leader) into the Index. (1, 5, 9 ...).



Ex:→ So, here → we just have to enter the no. of records in RAM equal to no. of blocks in H.D.

Here, 2500 records are to be Entered In Index.

(*) Now, come to ques^n.
If our data is ordered → then.
→ i.e., Sparse →

No. of ~~records~~ Blocks in Index = $\frac{\overset{50}{2500}}{50}$ $\boxed{\Rightarrow 50}$

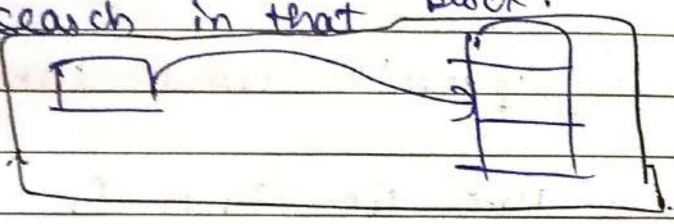
Then, search time → $\log_2 50$

$\boxed{\to 6 \text{ approx}}$.

Hence,
We have to search 6 times to find the desired page no.

&

(then that page no. takes us to the desired block in H.D & then, we just only have to search in that block.)



So

Total no. of Search = 6 + 1

= 7 ✓.

6 search in Index. &
last (1) Search in H.D..

But,
→ our data is unordered +,
Hence, Dense →.

→ No. of blocks in Index = $\frac{\text{All records}}{\text{record in one block of Index}}$

$= \frac{\cancel{10,000}^{200}}{5\cancel{0}}$

$\boxed{= 200}$.

Now,

Searching → $\log_2 200 + 1$
→ 8 + 1

$\boxed{\to 9}$ approx ✓

→ $\log_2 2^8 + 1$
→ 8 + 1 $\boxed{9}$

$\boxed{2^8 = 256}$

→ Here, we use $\lceil \log_2 200 \rceil$ also, bcz Index में तो unorderd data भी orderd/sorted होकर ही आएगा ना।.

∴ we use $\lceil \log_2(n) \rceil$ here. ✓.

34. ✱ Types of Indexes :→

1.) Primary Index
2.) Clustered "
3.) Secondary "

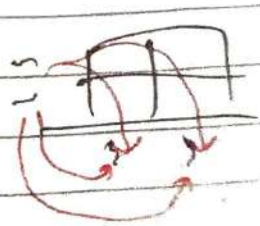
key → uniqueness,
non key → not unique,

Imp.

✱ Table :-

ordered file	primary Index	Clustered Index	~~At most 1.~~
unordered file	Secondary Index	Secondary Index	
	key	Non key.	

key	Non key
1	1
2	1
3	2
4	2
5	3
	3



✱ we all have primary Index in our Books & In last of Book → Secondary Index.

If we want I/o cost to be less, then we made Indexes.

Date ___ Page No. ___

(95.) Primary Index :→

→ In a table, if we made any of att. primary key, then By default → primary Index is operated on them Automatically.

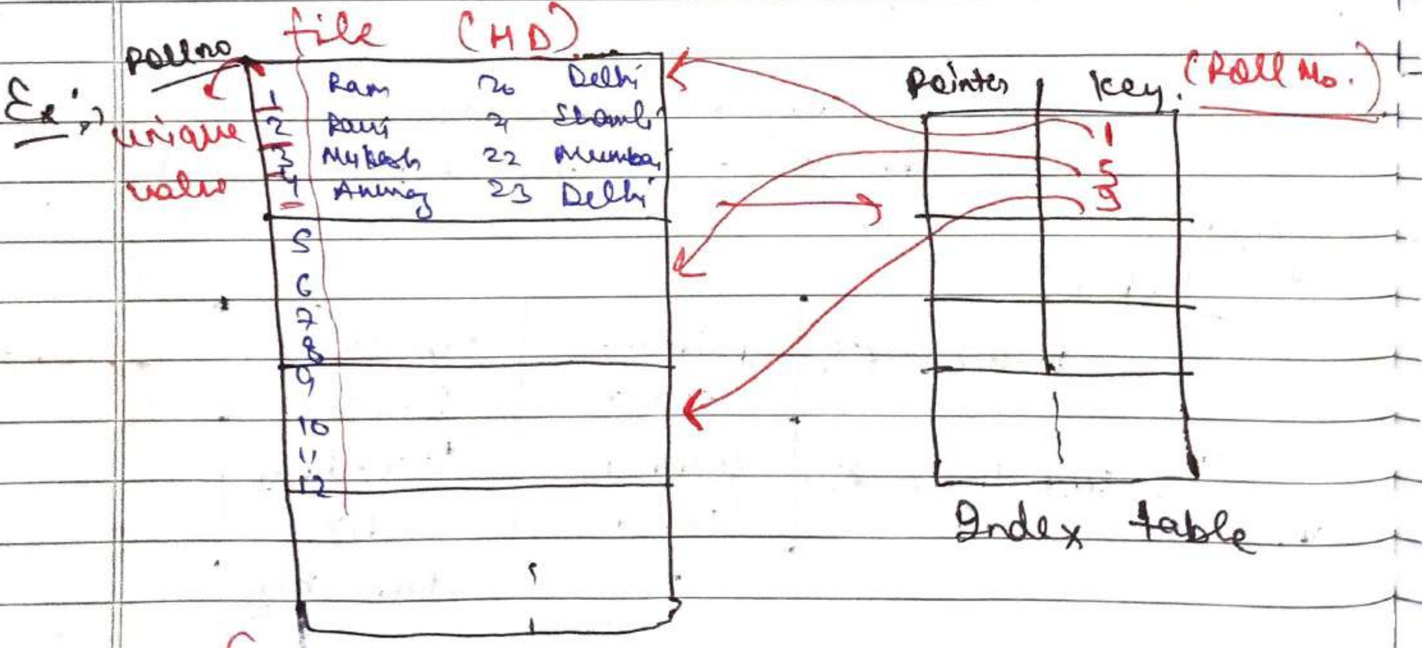
→ Advantage :-

1.) data is Ordered.

2.) & also unique. (key value is present).

Ex! In IRCTC, we just find everything, by Train No.

Ex:→



↳ Data is (Sorted + unique).
↓
So, we use Sparse here.

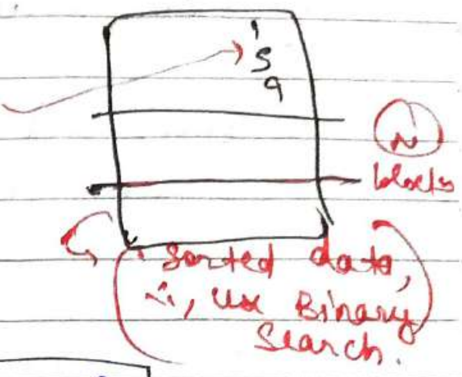
→ No. of Entries in Index Table = No. of blocks in H.D.

and, we call it Ancher (leader).
↓
(1, 5, 9, 13 .....).

$\boxed{\text{\# Primary Index is sparse.}}$

→ Ex: 1) we have to find 3,

~~then~~,
we know It is found in block of '1'.



Hence,

$$\boxed{\text{Total Search Time} = (\log_2 N + 1)}$$

Where,
N is the No. of blocks in Index Table.

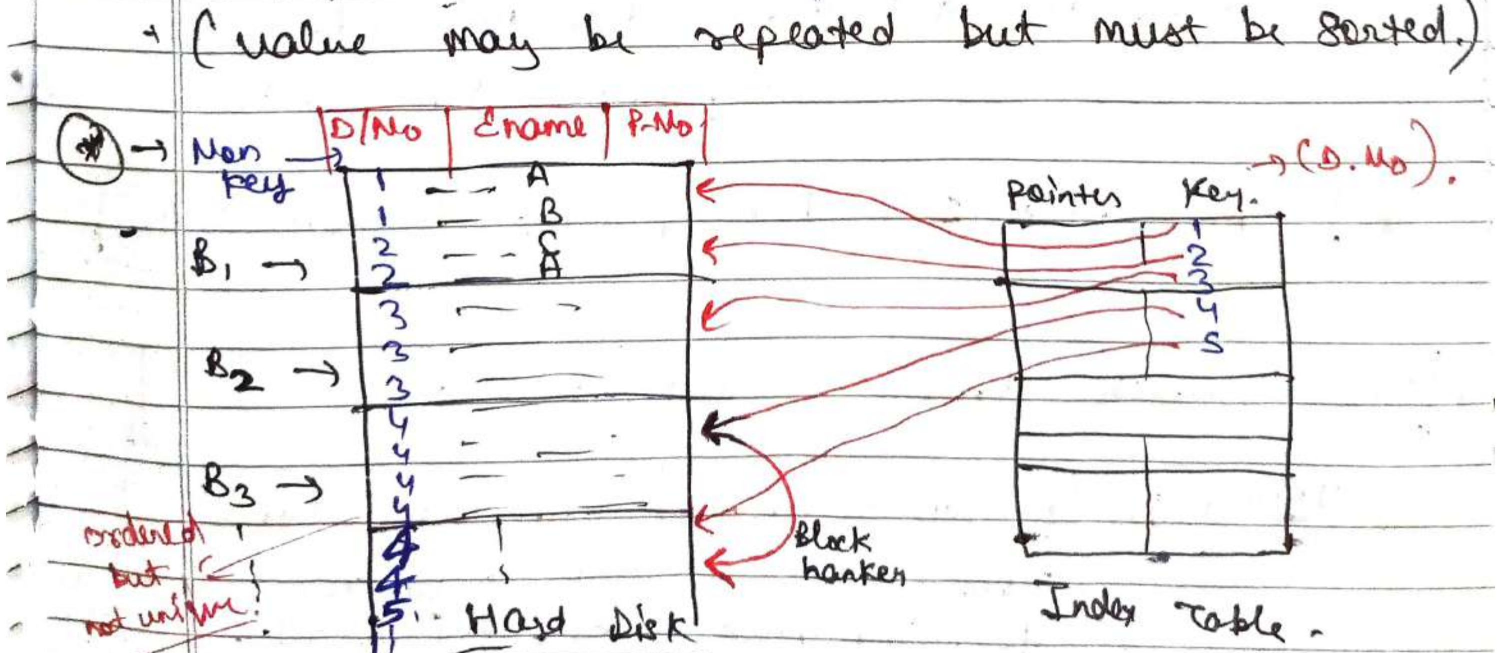
x ——— x

(36) Clustered Index! →

→ data must be ordered & with Non key, (ie, not unique).

↓ ie,
→ (value may be repeated but must be sorted.)

(*) → Non key



- If our data is multiple times (not unique) in H.D. Then, It also comes only 1 time in Index.

- (In Index, no repetition). i.e. (unique)

- Here, 4 is not only in one block. Some 4s are also in next block. So, How we point to them.
So,
Now, we use Block hanker to point to next block for same value.

- Here, Searching criteria slightly increased (↑).

- Clustered Index is also sparse, always. bcz, we don't need to make pointer for every value. ∵ Repetition - there is only 1 pointer.

- हम एक से ज्यादा primary Index & cluster Index नहीं बना सकते, bcz no need. & 1 Table has only 1 primary key ना।
So,

- At Most 1, (both primary & cluster Index)

- $\text{Total Search Time} = (\log_2 N + 1) + 1$

Where, N is No. of blocks in Index Table.

(These extras are ∵ Block hanker. for, Each block hanker, there is +1).

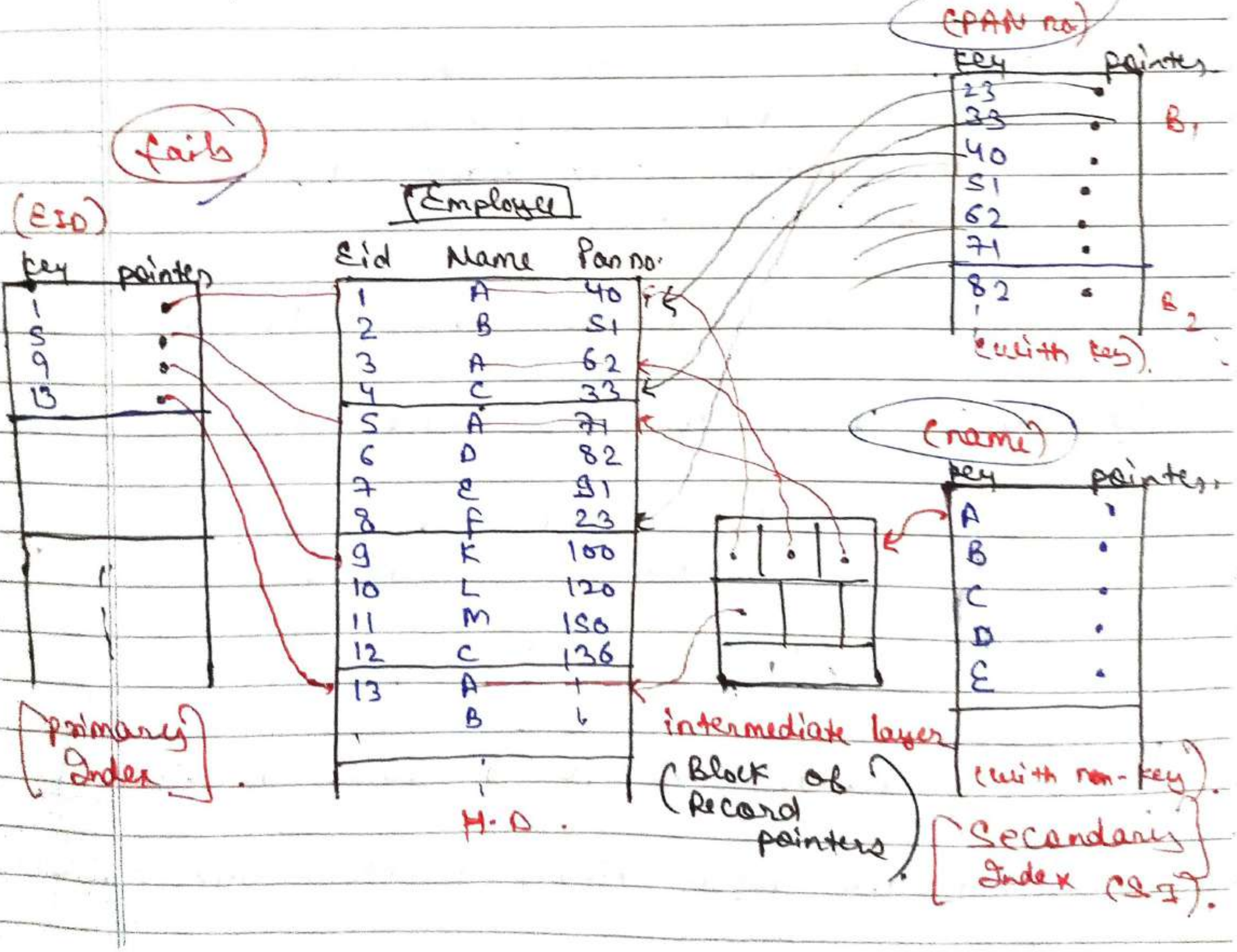
(47) Secondary Index, (Multilevel Indexing):
(S.I.).

→ (Already there is an Index in database, we have to made another Index).

→ Where use S.I.?

Unordered Data → key, Non key.

→ Why another Index?



key	pointer
1	•
5	•
9	•
13	•

Eid	Name	Pan no.
1	A	40
2	B	51
3	A	62
4	C	33
5	A	71
6	D	82
7	E	91
8	F	23
9	K	100
10	L	120
11	M	150
12	C	136
13	A	|
	B	|

key	pointer
23	•
33	•
40	•
51	•
62	•
71	•
82	•

key	pointer
A	•
B	•
C	•
D	•
E	•

→ Why Another Index?

⇒ Let, H.D. has data of Employee table & data is already sorted on basis of E-Id (primary key). → [(unique + Not Null)]

↳ bcz most of the query are on E-Id.

↓

⇒ Find # of Employee where E-Id = ~ .

↳ then we use primary Index. (which is already there formed on basis of E-Id.).

- But, sometimes we also use name & pan no.

then, Primary Index fails. bcz, It is made on E-Id.

So, we use Secondary Index here.

(#) Case I when we also have key.

then we use PAN no. (unique + unordered).

(As u can see in Diag.)

* (#) We always use sorted values in Index.

↳ then, we apply binary search → Time Saving.

* Index is always sorted + unique.

* 2) Secondary Index on key, is always DENSE.

bcz, we don't have any anchor (leader) here, like in primary In.

→ These values are not sorted. Hence, we put them sorted in Index. & write all the records of H.D. in Index Table. So, Dense.

i.e,

[No. of records in Index = No. of records in H.D.]

→ Search time: $\log_2 N + 1$

where,
N is the no. of blocks in Index Table.

* Case II When we have Non key with unordered data.

It is the worst case.

then,
we use NAME,
↳ (neither ordered, nor key (unique)).

↳ (Secondary Index in Dbg.)

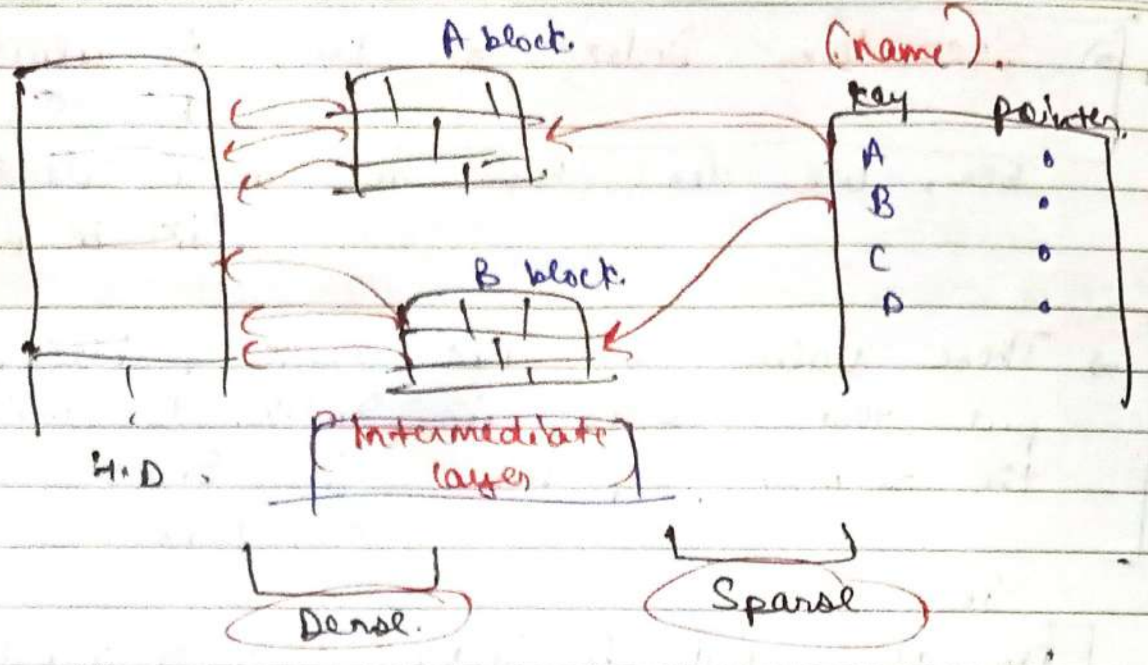
→ In Index, we don't have to take multiple values of A. (Only once).

But, A is many times in Hard disk.

So,
here we made an Intermediate layer.
(It is a block of Record pointers).

(book के पीछे Secondary Index (S.I.) मिल जाता है।)

Ex1:-)



=) Hence, It is Mix of Dense & Sparse.

so,

=) We can say it DENSE Overall.

=) Time Complexity : $\log_2 N + 1 + 1 + \ldots$

(xtra for intermediate layer)

(and this is only for 'A'. It may be more than $\lceil \log_2 N + 2 \rceil$)

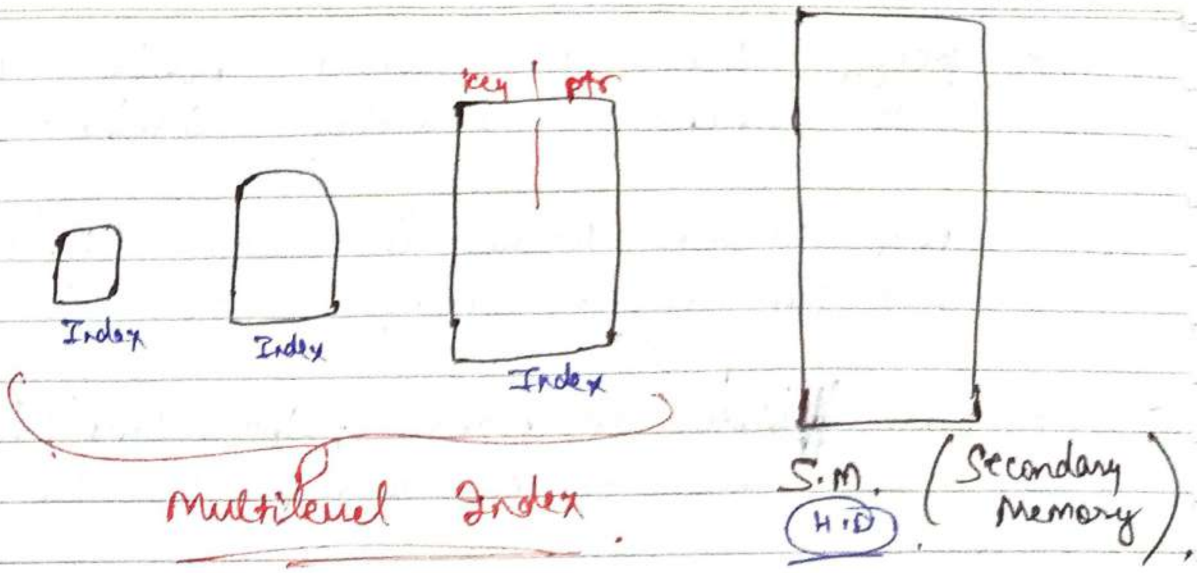
(*) Secondary Index is always dense.

(98.) Intro to B-Tree & its Structure :-

1) B-Tree (Dynamic Multilevel Index) :-

(Balanced Tree)

graph → cycle
Tree → acyclic.

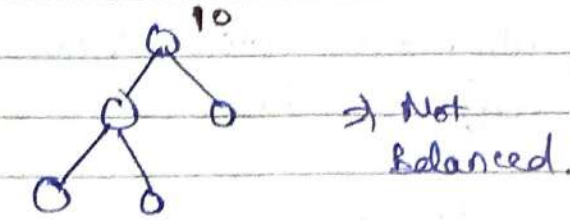


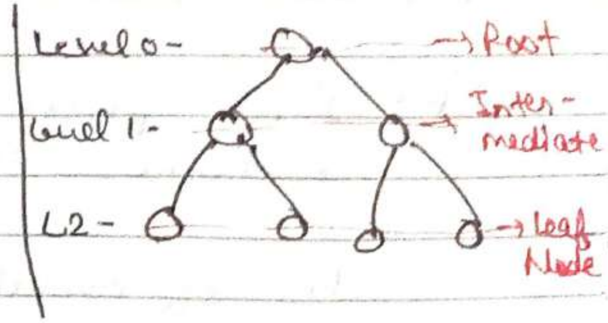
⇒ In this, It is hard to manage. bcz, If we insert data in S.M. then we have also insert in all the Indexes & Same for delete.

Balanced Tree (B-Tree! →)

→ means, all the Elements are at the Same level of leaf Node.

⇒





⇒ Block pointer (B.P.) or Tree pointer ! → when node denotes his child. Then, we use Block pointers (B.P.).

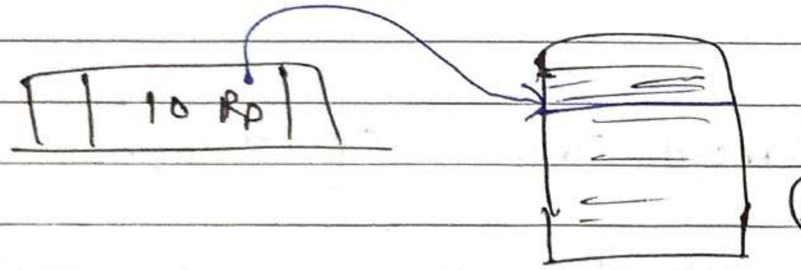
⇒ Note! → A node can contain multiple values here.

→ keys :- that on which basis we have to search. Searching criteria = key.

(key को हम Nodes के अंदर डालेंगे।)
(we can insert multiple keys in a node).

→ Data pointer (D.P.) (or) Record pointer (R.P.) :→
These are with correspond to keys.

→ This record pointers points to in the Secondary Memory (Hard Disk) where are record is present of that key.



(#) keys = Record pointers.

(#) Block pointer depends on the how many children are there of a Node.

No. of children = No. of B.P.

(#). Order = p (of a B-tree)
= Max. no. of Block pointers.

Order = p = Max. no. of children a node can have.

(#) Keys = $p-1$ ← Max
Rp = $p-1$

Min keys = $\lceil \frac{p}{2} \rceil - 1$

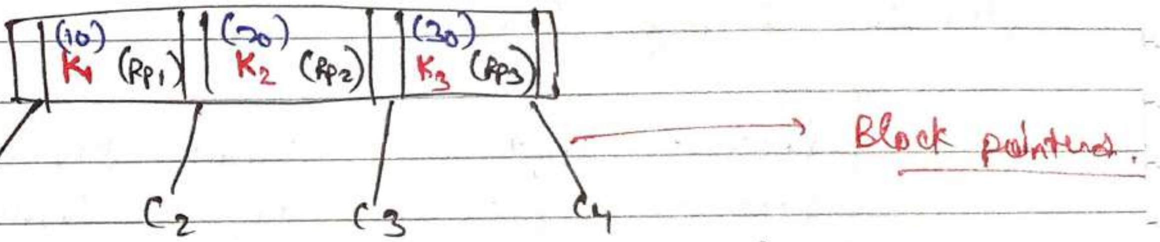
Table:-

		Root	Intermediate Node.
Children	Max	p	p
	Min	2	$\lceil \frac{p}{2} \rceil$

↳ Ceiling value

Expl:-

Let, p = 4 = order of a Tree.



(#) We always insert keys in the Sorted Order. (bcz, it follows the prop. of Binary Search Tree).

---

99. Insertion in B-Tree :-

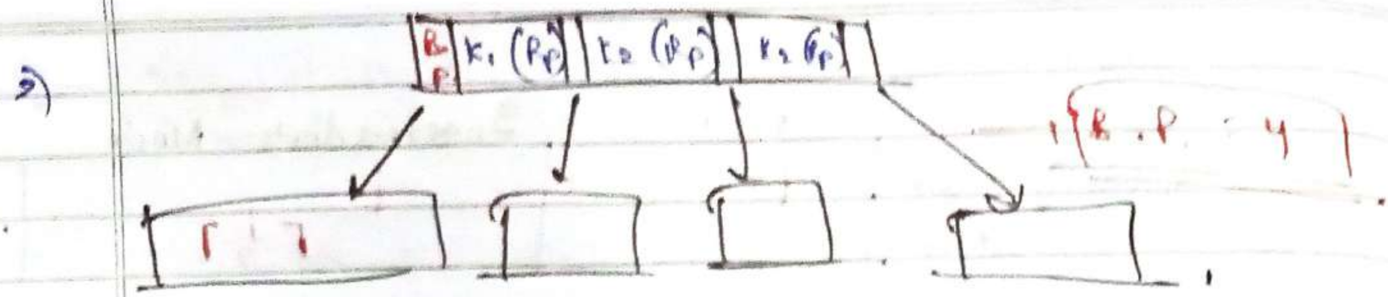
Q!- Insert the following keys into B-Tree, if order of B-tree = 4.

1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

=> Order = 4 = max. no. of children.

Max keys = p-1 = 3

Min keys = $\lceil \frac{p}{2} \rceil - 1 \Rightarrow 1$

⇒ 

⇒. We follows the prop. of Binary Search Tree in Insertion.

(#) In Binary Search Tree,

Root → Left = Small Element
Root → Right = Big Element.

(#) Overflow होते ही, Median निकाल के break करना पड़ेगा,

→ When, n - Even ⇒ $\frac{n}{2}$ ⇒

let, 4 ⇒ ○ ○ ○ ○

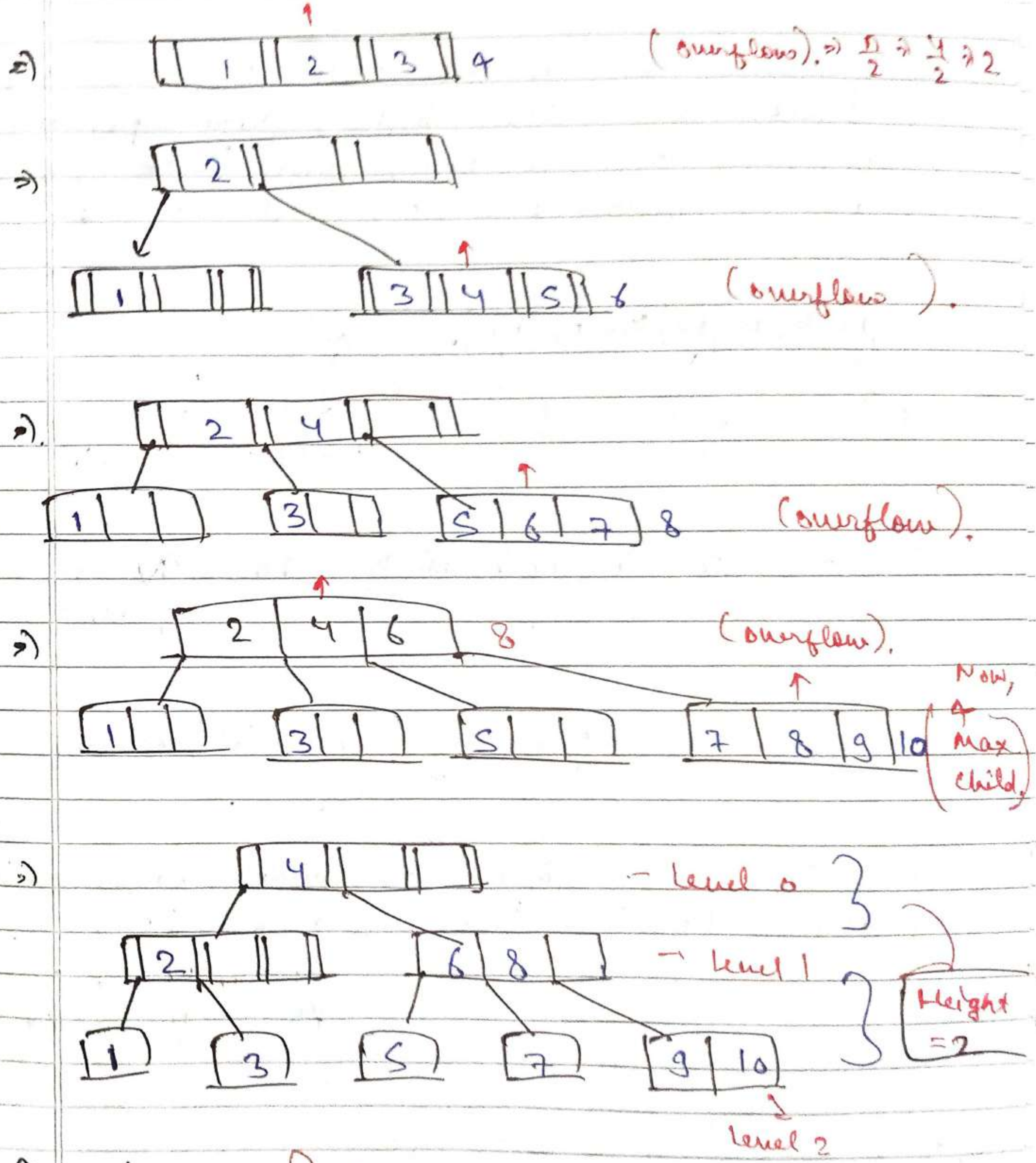
$\frac{4}{2}$ ⇒ ② Median

→ When, n - Odd ⇒ $\frac{n+1}{2}$ = Median

let, 5 ⇒ $\frac{6}{2}$ ⇒ ③ ○ ○ ○ ○ ○ Median.

⇒ and, Shift Median to upward (↑), & उसके Left & Right Elements उसके नीचे आ जाएंगे। i.e, how we break.

[illegible], 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

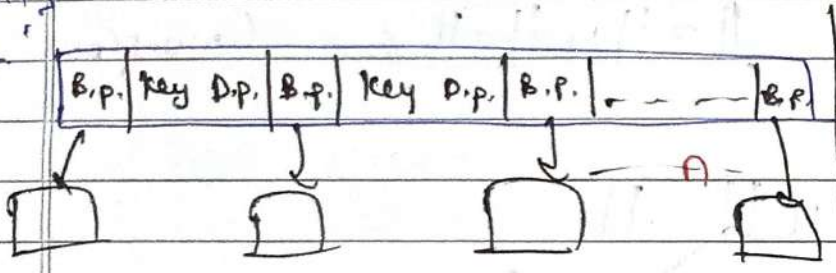


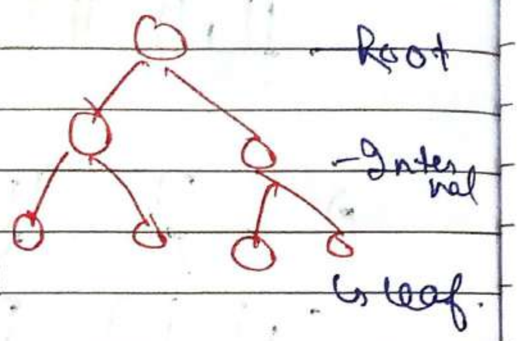
⇒) height - 2
levels - 3

100. How to find Order of B-Tree ! →

Q1- Consider a B-tree with key size = 10 bytes, block size 512 bytes, data pointer is of size 8 bytes and block pointer is 5 bytes. Find the order of B-tree?

Solⁿ :-





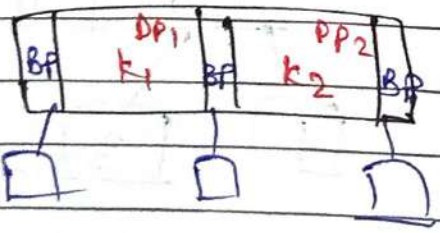
→ Let, In a node / Block has 'n' no. of Block pointers.

so,

Total size of B.P. = n × (size of 1 B.P.)

$$\boxed{= n . B_p}$$

If n B.Ps (or children a node can have) then, (n-1) keys, & Record pointer.



$$\boxed{n \times B_p + (n-1) \text{ key size} + (n-1) R_p \leq \text{Block size (or) Node size.}}$$

⇒ $n \times 5 + (n-1)(10+8) \le 512$.

⇒ $5n + 18n - 18 \le 512$

⇒ $23n \le 530$

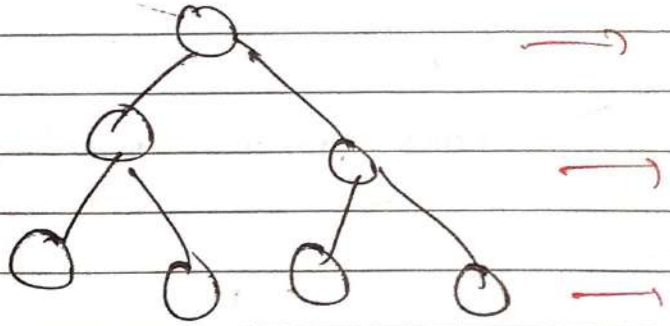
⇒ $\boxed{n \le \frac{530}{23}}$.

⇒ $n \le 23.04$

∴, $\boxed{n=23}$ Ans. — Max. 23 children.

⇒ Every B.P. represent a children.

so, $\boxed{\text{Max. Order} = 23.}$ Ans.

⇒ 

Max	Min Children
23	2
23	$\lceil \frac{23}{2} \rceil \Rightarrow \lceil 11.5 \rceil \Rightarrow \boxed{12}$.
Leaf has no children.	

Ans.

∴ Keys = Order − 1 ⇒ $\boxed{22}$ Ans.

—x— —x—

101. D/B B-Tree & B+ Tree : →

* B-Tree :→

1.) Data is stored in leaf as well as internal nodes.

2.) Searching is slower, deletion complex.

Date
Page No.

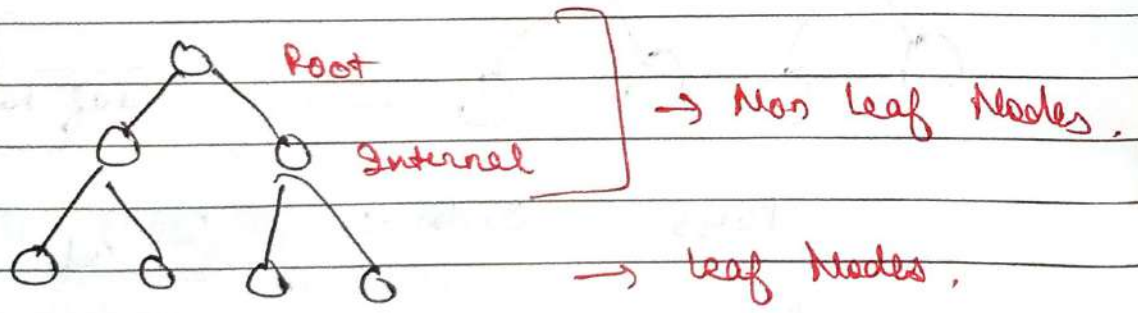
3.) No Redundant (Duplicate) search key present.
4.) leaf Nodes not linked together.

(★) B+ Tree :-

1.) Data is Stored only in leaf Nodes.
2.) Searching is faster, deletion easy. (directly from leaf Node),
3.) Redundant keys may present.
4.) Linked together like linked list.

#. We use B & B+ Tree, to put Index Record. These trees actually contain Index Record.

Index Record < key / Data pointer (Record pointer).

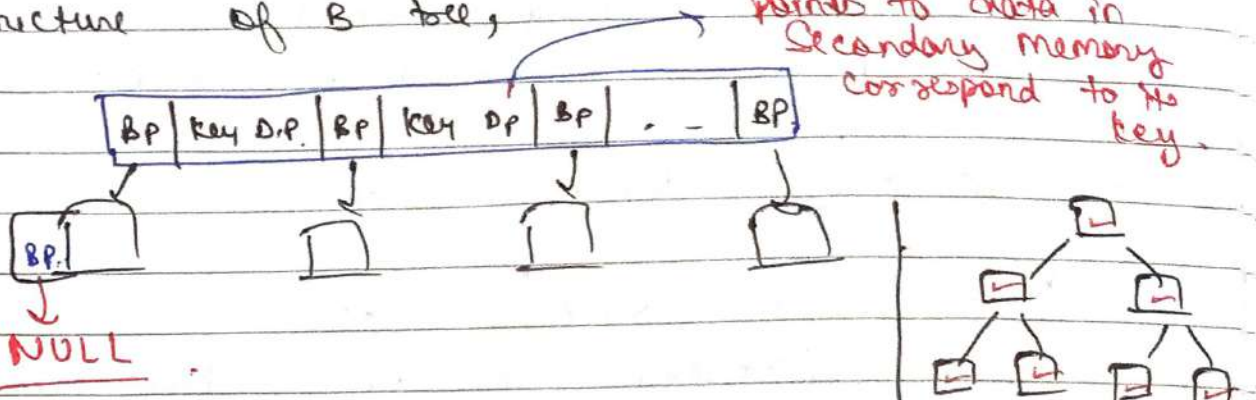
#. 

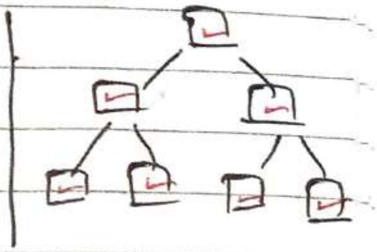
In B tree, the Structure of every node is same. (Either root, internal or leaf).

leaf Node, has no children. So, what's the role of B.P. Then, they points to NULL.

B.P - Block pointer.

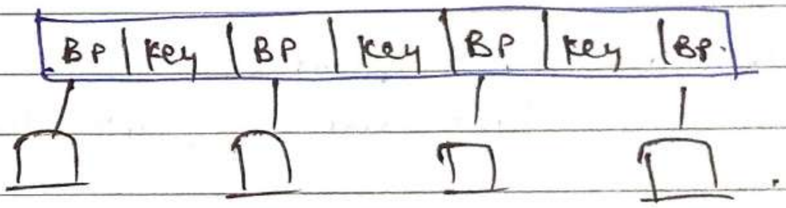
Structure of B tree,





Structure of B+ tree :→

⇒ Non leaf Structure :→ or Internal Node →.



↳ There is no Data pointer (D.P.) in Internal Node Structure / Non-leaf Node Structure.

R.P / D.P — ✗

Hence,

We have more space in internal node. Hence, we can create more children & put more keys.

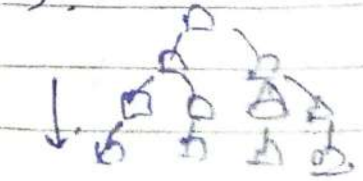
So, that's why,

B+ tree → Breathwise longer.

B tree → Depthwise longer.

↳ (bcz, less no. of children breathwise as compared to the B+ tree).

So, depth more.

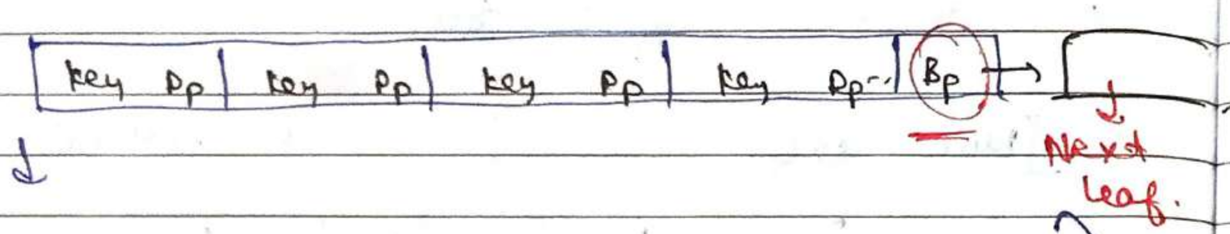


#. So, that's why searching is slower in B tree as compared to B+ tree.

#. B+ tree Structure →

Leaf Node Structure →

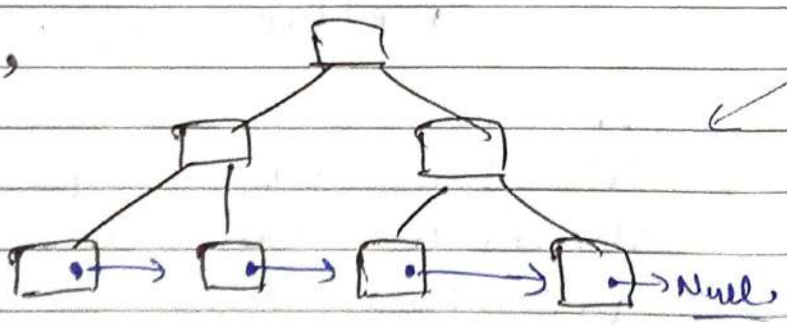
(Structure of leaf & Non leaf Nodes are different.).



⇒ (There are no Block pointers in leaf Node Except 1 in last which points to Next leaf.

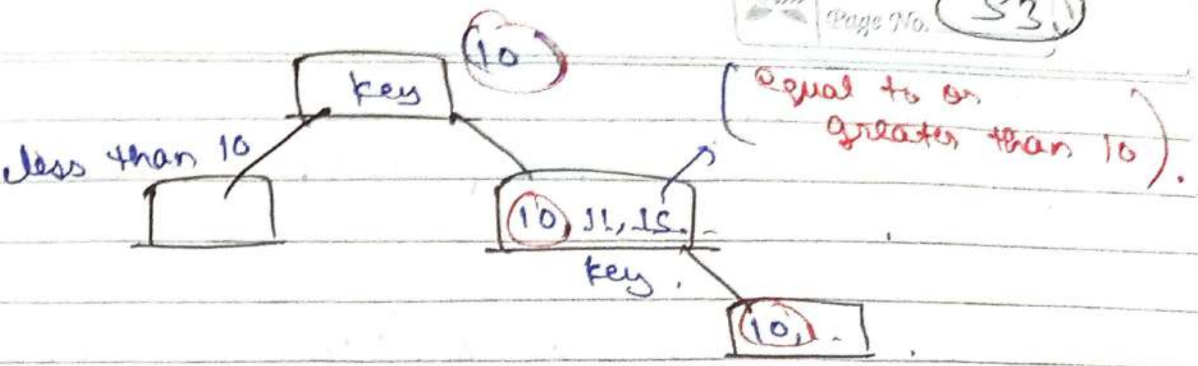
Both, B & B+ are balanced, i.e, all leaf Nodes are at Same level.

In B+ structure,



⇒ In this, less value to key is in left node & the greater value to key are in Right node. (But here, In Right side, we also have to put the key value with its greater values).

⇒ i.e,



⇒ Reason :→ bcz, we only search in leaf Node, bcz, Only leaf Node has all keys with Data pointers are present.
(D.P.)
So,
(We also have to seach the D.P. of the key)
so, that's why we also carry the key value upto leaf Node.

→ {Searching is that's why faster in B+ tree, bcz, have to search only in leaf Node.}

&

# bcz of this, Redundant keys are also present in B+ tree.

# Leaf Nodes are also linked together.

102. Quesⁿ on Order of B+ Tree :→

Q!→ Consider a B+ Tree with key size = 10 bytes, block size = 512 bytes, data pointer = 8 bytes & block pointer = 5 bytes. What is the order of leaf & non leaf node?

Solⁿ:→ Non leaf :→

BP	key	BP	key	BP	key	BP.

Leaf Node :→

key DP	key DP	key DP	B.P. →

⇒ Non-leaf :→ $\boxed{n \times B_p + (n-1) \times key \leq \text{Block size.}}$

$\rightarrow \quad 5 \times n + (n-1) 10 \leq 512$

$5n + 10n - 10 \leq 512$

$\rightarrow \quad 15n \leq 522$

$n \leq \frac{522}{15} \quad 34.8$

$\rightarrow \quad \boxed{n \leq 34.8}$

∴,

$\boxed{n = 34}$ ✓.

$\boxed{order = 34}$

[(Max BP.) or (Max children possible)].

⇒ leaf :→ $\boxed{x(\text{Key} + \text{PP}) + \text{BP} \leq \text{Block size.}}$

let, $\boxed{x \text{ pairs}}$

⇒ $x(10+8) + 5 \leq 512$

⇒ $18x \leq 507$

$x \leq \frac{507}{18} \quad \boxed{28.18}$

$\boxed{x \leq 28.18}$

∴,

$\boxed{x = 28}$ ✓ = $\boxed{order = 28}$ ✓

Note :→ Order of a leaf Node in B+ tree is the no. of (key, D.P.) pairs.

x ——— x ——— x

H.D.D. - Hard Disk Drive.

Date / / Page No. (SS)

(103.) Immediate Database Modification. →

(Log Based Recovery Methods).

⇒ ↳ Immediate means तुरंत, जल्द ही।

Ex1.) A = ~~100~~ 200
B = ~~200~~ 400

Hard Drive

T₁

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

(Transaction log.)

< T₁, Start >
< T₁, A, 100, 200 >.
old new
< T₁, B, 200, 400 >
old new
< T₁, Commit >.

Redo ✓

⇒ जैसे ही Ram के अंदर value (200) हुई, उसी Time पे ही Database में भी (A=200) कर दो। बिना Commit हुए।

⇒. i.e, At time, when we write in memory (RAM), at same time also update in the H.D. we don't wait for commit.

⇒ But, when we see in Trans. Log,

Our Recovery Manager sees the log & check whether T₁ has both Start & Commit or not. & If yes, then it REDO

→ Redo, means Saves the latest value in the Database. i.e,

→ (Recovery Manager don't see value in the H.D., it only sees in the Trans. log & check (starts & Commit) & then saves in the H.D. database. & if already saved, then Over-write & fixed them.

→ But, If

ex:-

	T_1	Transac^n log
A = ~~100~~ ~~200~~ 100	R(A)	< T_1, Start >
B = ~~200~~ ~~400~~ 200	A = A+100	< T_1, A, 100, 200 > old new
Database	W(A) - 200	
	R(B)	< T_1, B, 200, 400 >
	B = B+200	UNDO
	W(B) -400	
	fail.	

→ Here, Recovery Manager don't find commit in Trans. Log. So, he UNDO.

→ UNDO, means it Saves the old value. But, in database there are updated values now. So, Recovery Manager takes the old values from the Trans. log & saved them in the Database.

In Immediate, we store both old & new value.
but,
In Deferred, we only store new value.
(i.e, only REDO, not UNDO).

→ that's why
Immediate Database Modification is known as UNDO - REDO strategy.

Ex:- ⇒ Transacⁿ log ⇒

< T_1, start >
< T_1, A, 1000, 2000 >
< T_1, B, 5000, 6000 >
< T_1, Commit > — REDO.

< T_2, Start >
< T2, C, 700, 800 > — UNDO.
old.

———×——————×———

(104.) Quesⁿ on DBMS basic Concepts & Data Modelling :-

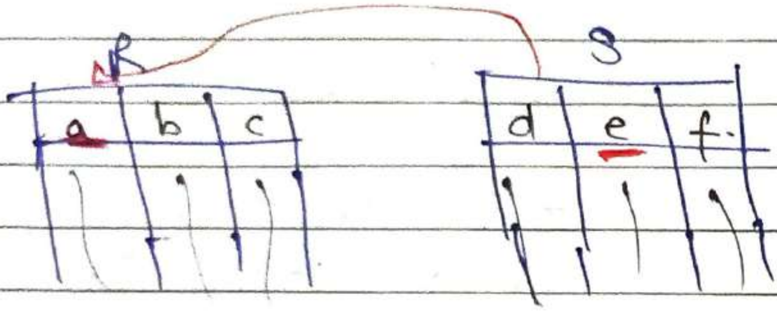
Q:- Let, R (a, b, c) and S (d, e, f) be 2 Relⁿs. 'd' is foreign key of S that refers to primary key of R. Consider 4 operaⁿs on 'R' & 'S'.

i) insert into R
ii) insert into S
iii) delete from R
iv) delete from S.

→ Which of these can violate Referential Integrity?

a) i & ii
b) i & iii
c) ii & iii → Ans.
d) i & iv.

Solⁿ :



⇒ Understand from:- कि Intelligent student कुछ भी delete कर सकता है लेकिन फिर दूसरे बच्चों को भी वो Delete करना पड़ेगा।
∵ (If they don't delete → then, violation.)

(2.) The view of total database content is ___

a) Conceptual view Ans.   b) Internal view
c) External view   d) physical view.

⇒ This is on 3 schema architecture.

* Internal view & External view,

external view is basically related to outer view.

⇒ User को Total view नहीं होता।

User has partial view (data only he need)

(Cardinal - basic, original, fundamental.)

⇒ physical view तो सबसे 3rd level view हो जाता।
(Database storage Related).

⇒ In partial case, External view is Ans.

3. In Relational Model, Cardinality is →.

a) No. of tuples b) No. of attributes.
c.) No. of Tables. d.) No. of Constraints.

⇒ If degree, then no. of attributes.

The no. of rows in a Table, called as Cardinality

4. ________ Constraint is used to maintain Consistency among tuples in two relations.
a) key b) domain.
c.) Referential Integrity Ans. d.) Entity integrity.

⇒ domain basically deals with that which type of data we put into Table. (integer, varchar, character, etc.)

⇒ Entity integrity, related with primary key.

⇒ key deals with uniqueness.

———×——— ———×———

105. Imp. Ques^n on Advance DBMS :-

(Big Data & Data Warehouse)

Q-1. What do data warehouse Support?

[✓ a)] OLAP

b) OLTP

c) OLAP & OLTP

d) operational Database.

⇒ data Warehouse, where we Integrate data at one place from all diff sources.

Ex:- Big BAZAAR, → their outlets mostly found in Every City.

& (at the end of day, all data Integrated from all cities & kept) ie, data Warehouse.

- & we store this data, to analyse on it later. So that, they can also play AD's - to diff users acc. to their data of purchasing items.
(Apply Mining algorithms on these data)

⇒ OLAP → online Analytical processing.

⇒ OLTP → online Transac^n processing.

↳ It works on Current Data.

→ (that where to Remove store, when open new store, remove or add items) — all these based on Analysis.

(2.) Hadoop is framework that works with variety of related tools. Common group include ———

A) Map reduce, Hive & Hbase ~~Ans~~

B) Map Reduce, My SQL and Google Apps.

C.) Map Reduce, Humans, Iguana

D.) Map Reduce, Heron, Trumpet.

→ Hadoop, basically work on Big Data. (It is a Tool of Big Data).

↳ Like google, Facebook. Big data deals with the multiple petabytes of the data. Now, to process this much of Data. We don't use normal tool like SQL Server, Oracle. We use Hadoop, (bcz, this data is also unstructured).

&

→ (Hadoop Ecosystem / Framework Includes many small tools like map reduce. (used to process the data, reduce the data (divide) & then works on bench processing — i.e, works on Multi-processing),

→ Hive → (If we write to SQL commands in Hadoop, then use Hive).

→ Hbase → (helps in data storage),

• also tool like Zookeeper, flum, pig, etc.

7) Google Apps, it is part of cloud.

3.) All of the following accurately describe Hadoop, Except:

A) open Source — (listed in Apache, we also install it).
B) Real time Ans.
C) Java based
d) Distributed Computing approach.

→ Hadoop, It is basically bench processing. means we first need data & then we analysis on it.

→ (In Real time, we use SPARK).

4.) Which of the following does not comes under five V's of BIG Data?

a) Volume (how much amount of data).
b) Velocity (with which velocity, data ↑).
c) Variety (Structured, unstr, Semi-structu).
d) Visualization Ans.

Note :- let, if there is 'x' amount of total data.
where, x → is all data on Earth,
then,
(80% - 90%) of x is created in last 5-6 years. mean,
Data is increasing with so much speed.

Unstructured → photos, videos,
Semi-structured → XML based data.

→ Value → (ie, value of our data).
→ Veracity → (means, trustworthyness).
↓
(how much capable our data to believe on it.)

→ 5 V's of BIG DATA :→

1.) Volume
2.) Velocity
3.) Variety
4.) Value
5.) Veracity.

→ Visualiza^n, is a part of data Analytics. where we visualise our data with help of graphs (pie chart, bar chart, - etc.)

106. Deferred Database Modification :→

→ This topic comes under Log based Recovery.
↓
(If there is any failure inside our System, then either we can recover that System or not).

→ Log, is basically a file (small sized file). in which we store our actions. that (what Trans. performs, are stored in Logs)

(deferred → delayed, postponed, belated)

→ (Like, History in our Browser) →.

When our System fails, to recover Trans:
⇒ (Either we have to Roll back them or Modify them).

– We do that, by seeing the log.

⇒ 2 Methods, by seeing the log :→.

1.) Deferred Database Modificaⁿ

2.) Immediate " "

Ex.

Database (H.D).	T_1	Transacⁿ log.

A = ~~100~~ 200
B = ~~200~~ 400
	R(A)	< T_1, Start >.

	A = A + 100	
	W(A) – 200	< T_1, A, 200 >. new value.
	R(B)	
	B = B + 200	< T_1, B, 400 >. new value.
	W(B). – 400	< T_1, Commit >.
	Commit.	Redo –

⇒ Here, It don't update in Database hand-to-hande. It updates in Database after Commit, (bcz, deferred – late, postponed).

After commit, database updated.

⇒ How we Recovery in Deferred ? Let', say system fails after Commit.
→ So, when Recovery Manager comes, it first check trans. log file. & Check (Start & Commit) in it.

Then, Recovery Manager: (REDO).

- Means, update the new values in the database.
- (Let, due to fail → our database also not there).
then,
- Recovery Manager puts the A & B value from Log in database.

A = 200
B = 400.

- (If it already in database, then overwrite हो जाएगा).

Case II

A = 100
B = 200.

T₁	Transac^n log.
	< T₁ , Start > .
W(A). — 200.	< T₁, A, 200 > .
W(B). — 400.	< T₁, B, 400 > .
* fail.	
	(Roll back.)

- Now, fails before Commit. So, now database value is same as before.
Now,
(after failure occur, when Recovery Manager opens the log file. → It sees T₁ starts but not commit. So, Here Recovery Manager don't do anything. He simply Roll back.

Means, पुरानी value को ही रखो, कुछ भी Opera^n करने की जरूरत नहीं है।

⇒ So, Deferred Modification also known as (No UNDO/REDO) Method.

Ex:-

<T1, Start>.
<T1, A, 200>
<T1, B, 400>
<T1, Commit>.
} — (REDO)

<T2, Start>.
<T2, C, 500>.
} — (No Action).

Roll back ही. होगा।

#. In Deferred, we store only new values.

(107) Like Command in SQL :-

⇒ We use Like Command, generally to search the Data.

Q:- 1.) Find Employee detail whose name starting with 'A'.

2.) Find Emp. detail whose name ending with 'n'.

3.) whose name contains 'ee'.

4.) whose name contain 'a' in 2nd place.

5.) whose name contain 'a' in 2nd place, & name should contain total five Characters.

% → any value (कुछ भी chars. हो)
↳ चाहे '0' character हो तो, कोई problem नहीं ।.

Emp.

ID	Name
1.	Varun
2.	Arun
3.	Karuna
4.	Amrit
5.	Ranjeet
6.	Ajeet

%	→ any value & length.
_	→ reserved for a value.

1.) Select * from Emp where name like 'A%' ;
Output → Arun, Amrit, Ajeet

2.) —"— like '%n' ;
Output → Varun, Arun.

3.) —"— like '%ee%' ;
output → Ranjeet, Ajeet

Note:→ If Name → EEan, Arunee, — any thing
↳ then, these also comes, bcz sirf 'ee' we need.

% → also '0' character include.

4.) —"— Like '_a%' ;
output → Varun, Karuna, Ranjeet.

5.) —"— like '_a___' ;
Output → Varun.

In C++ ⇒ =
Here ⇒ :=

|| → or

Date / /
Page No.

108. Basic PL-SQL programming with Execution :→

→ Program 1: Find the Sum of 2 numbers.

Declarative part:

```
⇒ declare
  a int ;
   b int ;
   c int ;
```

Executable part:

```
  begin
  a := &a;                         // value given by user.
                                   (to take input from user).
  b := &b;
   c := a+b;
 dbms_output.put_line ('Sum of a and b ='||c);
 end;
                                   // cout in C++
                                   // } in C++
```

(#) program 2: Greatest of 2 numbers →

```
→ declare
  a int;
   b int;
   begin
   a := &a;
   b := &b;
  if (a>b)
  then
 dbms_output.put_line ('a is greater');    //a
 Else
 dbms_output.put_line ('b is greater');    //b.
```

SQL Line में हमें value पहले ही देनी पड़ती है इसमें बाद में user value नही दे सकता।

```
end if ;
end ;
```

⇒ (both code works fine). ✓

(109.) PL-SQL :- (while, for loop) :-

program 3:- For Loop :-

```
declare
a number (2) ;
begin
for a in 0..10          // = 0 to 10..
loop                    // By default,
                        increment of 1 by 1.
dbms_output.put_line (a) ;
end loop ;
end ;
```

program 4:- While loop :-

```
declare                 // print from a to b.
a int ;
b int ;
begin
a := 0 ;
b := &b ;
while a < b
loop
a := a+1 ;
dbms_output.put_line (a) ;
end loop ;
end ;
```

→ Output → 1 to 10

then,
→ output → 0 to 9
(Now, first print then increment)
→ Output → 10

* In oracle sometimes there shows error in output:- then write.

=> Set Serveroutput on → // but in Live SQL It is By default.

Code

=> (both code works fine). ✓

110. Single Row & Multi Row functions in SQL :-

=> Single Row:-
If our func. is applicable on single row, and only apply on single row. & gives an single output corresponding to that row. → then, It is Single Row funcs.

=> Multi-Row:- func.
func. that apply on more than one rows, & gives an single output corresponding to all these rows.

=> Round	(Round-off value)
=> Mod.	(gives remainder after divison).
=> lower	(Convert a string into lower letters).
=> InitCap.	(Capital the Initial letter).
=> Concat	(Add 2 string & make 1).
=> LPAD/ RPAD	(for left & Right padding).
=> NVL, NVL2	(to take care of NULL values).

SQL functions -

- Single-Row func^n.
  - Number func.
    - Round
    - Trunc
    - Mod
  - Character functions
    - Case Manipulation
      - lower
      - upper
      - initcap.
    - Character manipulation.
      - Concat
      - Substr
      - length
      - Instr
      - LPAD/RPAD
      - TRim
      - Replace.
  - Date function
    - Months_between
    - Add-Months
    - NEXT-DAY
    - LAST-DAY
    - ROUND
    - TRUNC
  - General functions
    - NVL
    - NVL 2
    - NULL IF
    - COAL ESCE
    - CASE
    - DECODE
  - Data-Type Conversion.
    - TO-CHAR
    - TO-NUMBER
    - TO-DATE
- Multi-Row func^n.
  - Aggregate.
    - SUM
    - MAX
    - MIN
    - COUNT
    - AVG

(III) Character functions in SQL with Execution :→

→ [Character functions]

Case function Manipulation

Character Manipulation.

→ lower
→ upper
→ Init Cap.

→ Concat ('Varun', 'singla') Varun Singla
→ Substr ('Varun', 2, 4) arun
→ Instr ('Varun', 'u') 4th
→ length ('Varun') 5
→ LPAD ('varun', 10, '*') *****Varun
(10 - length)
→ RPAD — " —
→ TRIM ('V' from 'Varun') Arun
→ REPLACE ('Varun', 'V', 'T') Tarun.

(#) A N U R A G
1 2 3 4 5 6

(#) Execution :→ If we want to implement these funcs, so, first we have to make schema (table).

→ [Output Table :→] select * from emp;

ID	NAME
1	Crate Smashers
2	Varun Singla.

=>
```
Create table emp
(
id int,
name varchar2 (20)
);
```
Table Created.

2 Rows
```
insert into emp values (1, 'Gate Smashers');
insert into emp values (2, 'Varun Singla');
```
2 Rows.

Select 1 by 1 Each Row, & See the output

```
Select * from emp;
Select lower (name) from emp;
Select upper (name) from emp;
Select initcap (name) from emp;
Select Concat (id, name) from emp;
Select SUBSTR (name, 2, 5), INSTR (name, 'v')
                                    from emp;
Select length (name) from emp;
Select lpad (name, 15, '*') from emp;
Select rpad (name, 15, '*') from emp;
Select trim ('v' from name) from emp;
Select replace (name, 'v', 't') from emp;
```

Output :-

REPLACE (NAME, 'V', 'T')
Gate Smashers
tarun Singla.

Output :-

LPAD (NAME, 15, '*')
**GATE_SMASHERS
***Varun_Singla.

Here, It counts (_) Space also.

Output :-

SUBSTR (NAME, 2, 5)	INSTR (NAME, 'v')
ate S	0
arun	1

(It don't count (_) here).

ie,

G	A	T	E		S	M	A	S	H	E	R	S
1	2	3	4		5	6	7	8	9	10	11	12

→ ate S

(112) View in Database!→

(Oracle, SQL Server Views)

• What is view in Database ?

→ Virtual Table!→

(The Table we create,
Create Table xyz

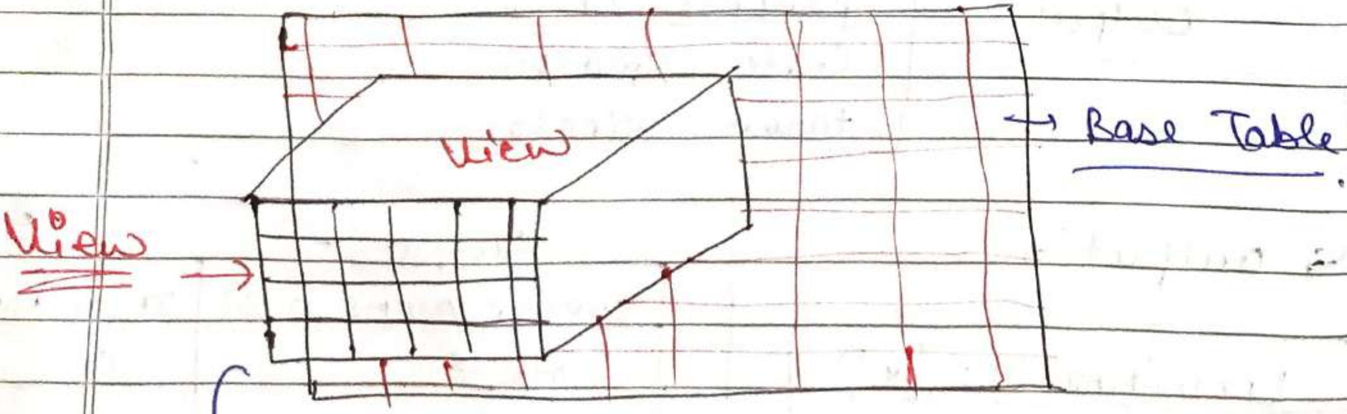
It takes physical space in memory (H.D.)).

⇒ But, view is the virtual Table.
It looks like a Table, but it is not actually a table. It don't take any space.

⇒ View is the result set of a stored query

[Query!→]

→ Create view V1 as Select id from Student;



This view has no physical Existence, we don't store it anywhere.

(#) The Execute code of this query, Stores after compile. &

after compile, when we write
Select * from V1
then,
it shows the data like a table from the view

(#) So, actually we just store this little query, rather than Result. We don't Store Result.
That's why It's a Virtual Table.

⇒ Read - only (vs) updatable Views ! →

⇒ If we made any changes in Base Table & that colm is also in our view, then, obviously that also changes.

⇒ Same, If we delete from Base Table. Then also deletes from View.

(#) ⇒ But, if we change anything in View :→

and we want that changes to execute also in the Base Table, then updatable Views. &
If we Shut down the (Insert, Delete & update) (DDL commands) on view, i.e, we disable these commands. So, that it can't Operate on View. Then, we made Read- Only Views, for it.

(Snapshot - जल्दी से photo लेना |.)

2). Materialised View :-

- type of updated version.

(If our data is on the remote server, & I want a copy of that on my local server/machine. ie, I want a snapshot of that remote data on my local Server. So, that is called Materialised View.

→ (This takes space, but takes less Space as comparatively to that data),

(#) We can't apply any DDL command (Alter, etc.) On View.
But, apply only DML Commands if we make the updatable view.

(#) We can insert the data of more than 1 table, in a View.
ie.
(View can also take data from Multiple Tables.)

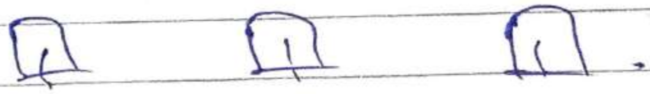
(#) We also can take any particular row from Table to make View. like,
~~~~~ where address = 'Delhi';
So, all Delhi students come into the view.

(#) Advantages of View :-

1.) To restrict the Data Access.
(Like, we don't give the access of original Table to our user, we just give View to them) of Some data.
~~~~~

2.) To make complex queries Easy.

Ex.1) Like, we some data from 3 tables.



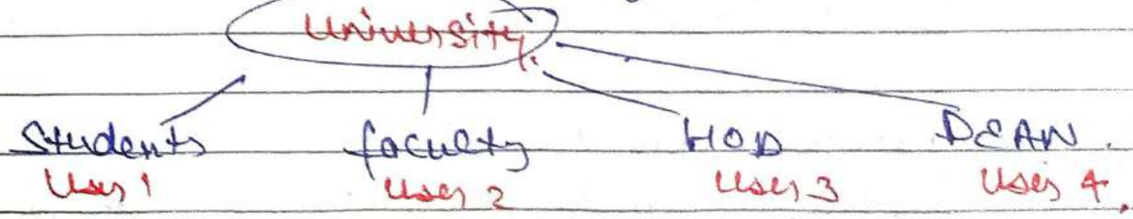
then,
By default, we apply Join or Nested Query.

So,
Its better to just make the view from these Tables. Rather, then to write Complex Query

3.) To provide data Independence,
↓
(we just a give a particular access to a user of a Table, rather than giving the full database access).

4.) To present diff. views of the same data.

Ex.2



(They all have diff. privileges).
So,
→ we give diff. view to all of them of the same Data (Table).

x ——— x

SQL CHEAT SHEET :→

Examples →

1.) Select all rows from table with filter applied

→ Select * FROM tbl where Col 1 > 5;

2.) Select first 10 rows for 2 columns

Select Col1, Col2 forom tbl LimiT 10;

3.) Select all rows with multiple filters applied

→ Select * from tbl where Col 1 > 5 AND Col 2 < 2;

4.) Select all rows from col1 and col2 ordering by col1

→ Select col1, col2 from tbl order By 1;

5.) Return Count of rows in table

→ Select Count (*) from tbl ;

6.) Return Sum Of Col1

→ Select SUM (col1) from tbl ;

7.) Return max value from Col1

→ Select MAX (col1) from tbl ;

8.) Computes Summary Statistics by grouping col2

→ Select AVG (col1) from table GROUP BY col2;

9.) Combine data from 2 tables using a left Join

→

```
Select * from tbl1 As t1
LEFT JOIN tbl2 As t2 ON t2.col1 = t1.col1
;
```

10.) Aggregate & filter Results

→

```
SELECT
        col1,
        AVG (col2) = AVG (col3) AS total
FROM tbl
GROUP BY col1
HAVING total > 2 .
```

11.) Implementation of CASE statement →

→

```
Select col1,
CASE
    when col1 > 10 THEN 'more than 10',
    when col1 < 10 THEN 'less than 10'
    ELSE '10'
END AS New_ColumnName
FROM tbl ;
```

(*) ORDER OF EXECUTION →

FROM
WHERE
GROUP BY
HAVING
SELECT
ORDER BY
LIMIT

↓

8. **Create**

```
CREATE DATABASE Mydatabase ;

CREATE INDEX IndexName
ON TableName (col1);

CREATE TABLE OurTable (
        id int,
        name varchar (12)
);
```

- **UPDATE TABLE**

```
UPDATE OusTable
SET col1 = 56
where col2 = 'something';
```

- **DELETE**

```
DROP DATABASE OurDatabase ;

DROP TABLE Ous table;
```

- **DELETE Records**

```
→ Delete from Ous Table
Where col1 = 'something';
```

Add / Remove Column

```
Alter Table Ourtable
ADD cols int ;
```

```
ALTER TABLE OurTable
DROP Column cols;
```