→ (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 remains here.)

comit
|
u(A) → unlock

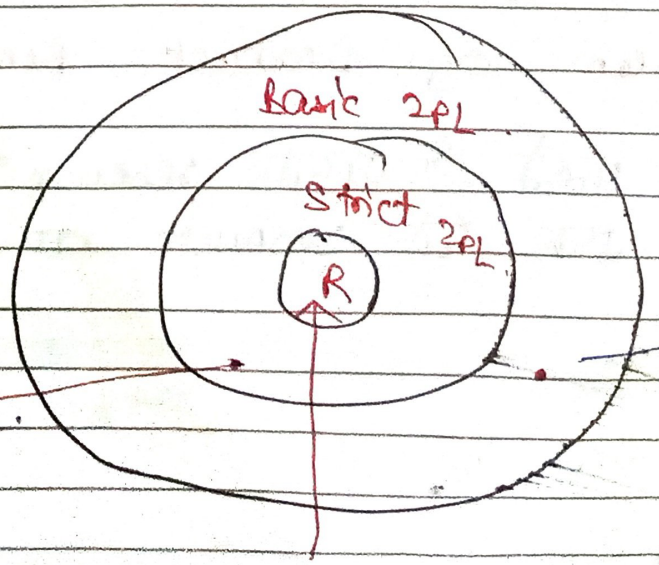
फिर वो (T_2) Database (H.D.) से ही Read करेगा।

→ It produce Strict Recoverable Schedule.

Note:- 2 problms remains here →
1.) Cascadeless
2.) Strict Recoverable

→ Rigorous 2PL (R), बस और Strict हो गया। BC, इसमें S/X दोनों आ गए।
(इसमें हम Shared lock को भी Release नहीं कर सकते।)

(H)



In both 2PL & Strict 2PL. But not in Rigorous 2PL.

In Basic 2PL only

Date / /
Page No.

- 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)		
		S(A)
	C	
		C

waiting

⇒ 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 ये लेले।).

⇒ इससे T_2, T_3 को एक भी lock नहीं मिलेगा।
i, problem of Deadlock Removes here.

- (T_1 को पहले ही सारी resource दे दो, बाकी 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 लेगा।)

∴

No Deadlock here.

Q9. 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
↓	↓	↳ youngest.
older (पहले आई)	younger (बाद में आई)	

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

Ex:→

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

T.S. (T_i)

$\boxed{\Rightarrow RTS(A) = 30}$

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

Ex:-

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

$\boxed{\text{So } WT(A) = 20}$

* (#) Rules :→

In Timestamp, let,

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

i.e.

$\boxed{\text{Serializability : Older} \rightarrow \text{Younger}}$ → follows Conflict Serializable

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

(0, 100; 200, 300) → in next udo 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)

T_1	T_2
R(A)	
	W(A).

✓ (NP).
(No Conflict).

T_1	T_2
W(A)	
	R(A)

✓

T_1	T_2
W(A)	
	W(A).

✓

bcz,
(old → young) as ($T_1 \rightarrow T_2$). So,
(T_1 पहले execute हो जाएगी, तो T_2 में कोई दिक्कत नहीं आएगी।)

Case I

Ex:- (old) 100 | 200 (young)

T_1	T_2
	R(A) -10.
20 W(A).	
commit	
	(X)

Roll back.
(Not possible)
(T_1 को होने नहीं देना।)

→ (bcz, now ये T_2 में A-10, कोन-सी value Read करके रोकी है। bcz T_1 ने तो Database में 20 update कर दिया।)

Case II

T_1	T_2
	W(A) - (20).
20 → R(A)	
commit	

Roll back.

(Hence, ये 20 अब कहीं पे है ही नहीं। i.e, T_1 गलत Data पे काम कर गयी।).

|| Dirty Read

(हम T_1 को read नहीं करने देंगे।)

Date ___/___/___
Page No. _____

Case III.

100 T_1	200 T_2
	W(A) = 20
40 = W(A)	
!	
Commit	
	*

Roll Back.

i.e., Roll back.
(i.e., हम T_1 को allow नहीं करेंगे).

⇒ (If T_2 fails, then, our updation will be lost).

Lost updation.

---

90. Solve Ques on Timestamp Ordering Protocol:

We have 2 Rules →

1.) Ist Rule is of Read → has only 1 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) T_1 oldest	T_2 (200)	T_3 (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) ✓

(Trans.)

⇒ Check, for every values in the Table & put that values in Rules & then make the Table.

- for R(A) use 1st 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) ✓

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 (Works 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 Instr. per Second)

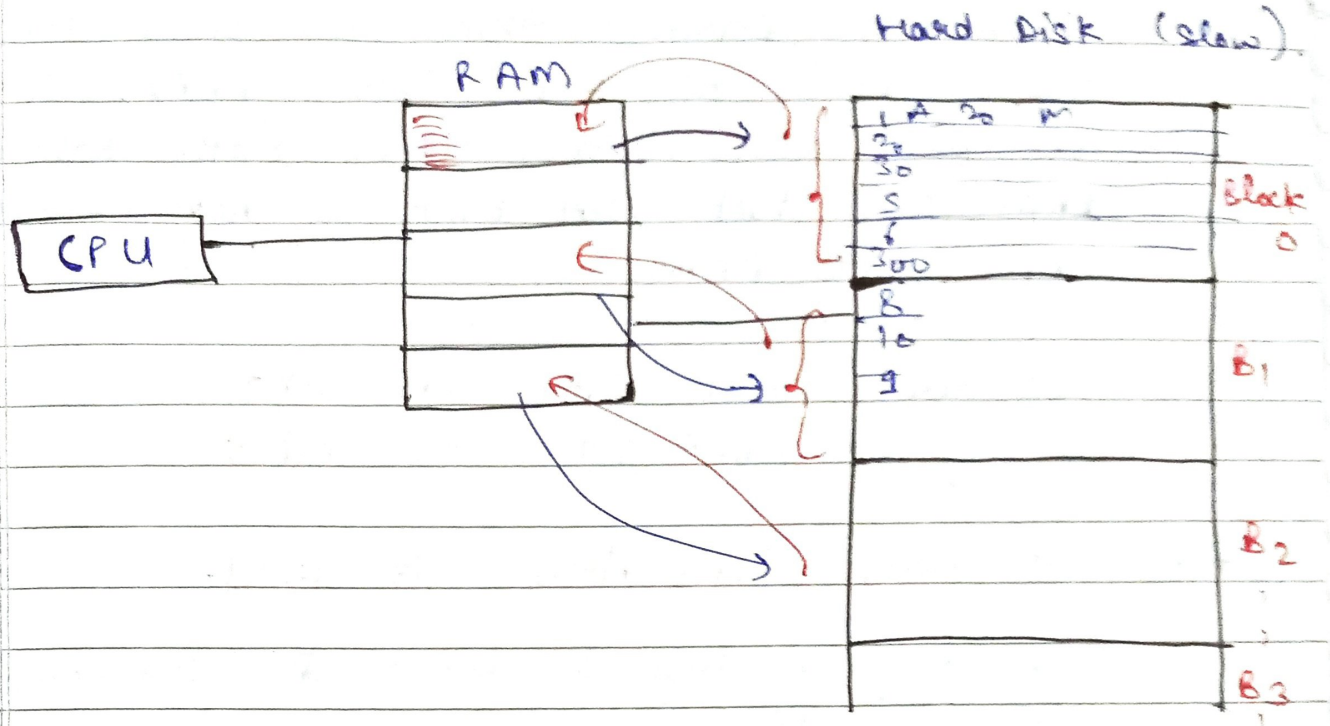
⇒ H.D. Speed ⇒ 10/20 Instruc^n per second

∴ these both are not compatible with each other.

So,

⇒ Ram comes b/w them.

⇒ Hard speed counts in milliseconds.
CPU ——— " ——— nano/picoseconds



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

Date / /
Page No.

=> Unordered Data! → then, we send 1-by-1 every block of H.D. into RAM. & if we get our data, then ok. 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.

(ईस Ram में H.D. के Block को Searching time को नहीं करते,).

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

[I/O Cost]
→ Input/ out-put.

↓

इस 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).

Block/pages

Average → 500 pages.

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

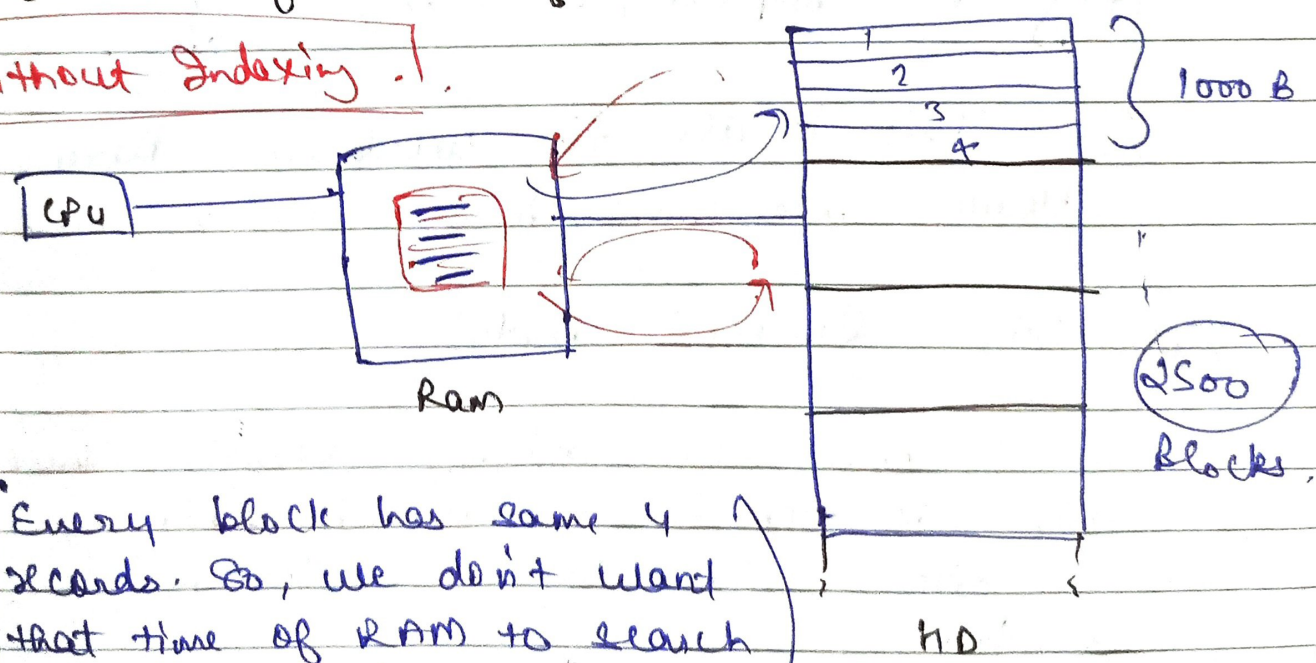
→ (Let, Index is of 2-3 page, then just see that topic on Index & then directly go to that page).

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, i.e, always 4.)

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

&

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

⇒ **I/O Cost :-**

Best Case = 1

Worst Case = 2500 = N

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

Avg Time Complexity : O(N)

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

Ordered Data :-

then,
we use Binary Search here, but it searches on sorted data (either ascending or descending).

& then,
Time Complexity : $O(\log_2 N)$

Here, ⇒ $\log_2 2500$.

⇒ 12 approx

Hence,
we need approx 12 blocks in ordered data.

So, this both case are without Indexing

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

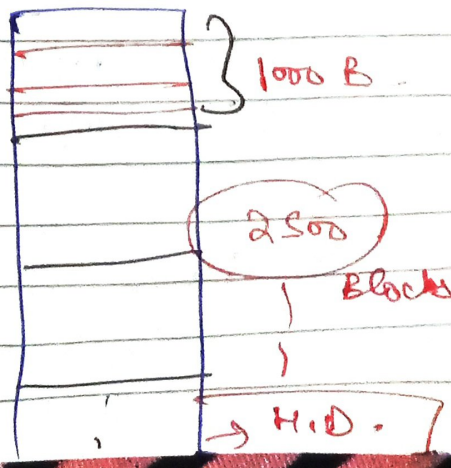
[I/O cost is low.]

---

93. Numerical on I/O Cost in Indexing :-

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 = 20B (key 10B + pointer 10B).

⇒ [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 — 1000 B

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

* Index Table Entry = 20 B (Key 10B + pointer 10B).

Key value → वो value जो हम search करते (Topic name), Roll no. etc.

pointer → page. No.

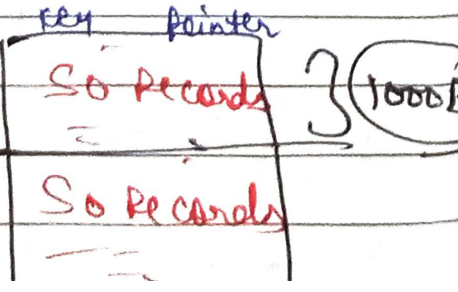
Each block in Index Table contains Record = $\frac{\text{Block Size}}{\text{Index Entry size}}$

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

$= 50$.

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! →

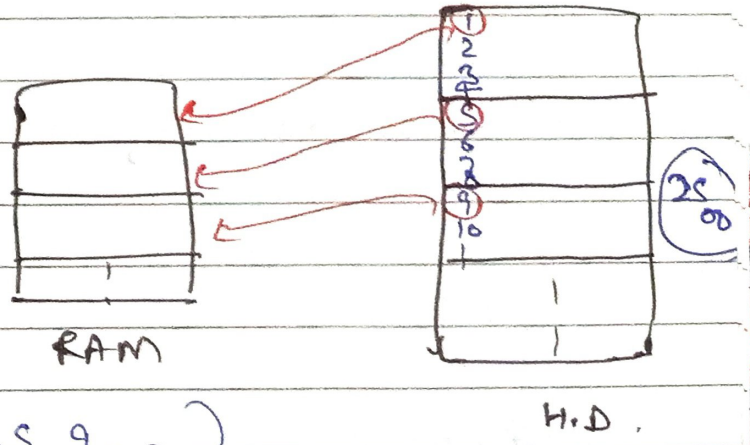
1.) **Dense!** → used when unordered data.

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!,
If our data is ordered → then,
i.e, [Sparse] →

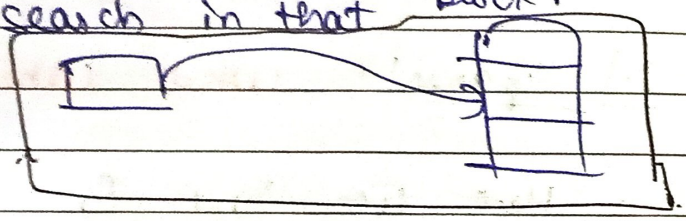
No. of ~~records~~ Blocks in Index = $\frac{2500}{50}$ (50) → 50

Then, search time → $\log_2 50$

$\boxed{\Rightarrow 6 \text{ approx}}$

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 = $\frac{\text{All records}}{\text{Record in one block of Index}}$

$= \frac{\cancel{10,000}\ 200}{\cancel{50}}$

Now, $\boxed{= 200}$.

Searching → $\log_2 200 + 1$
$+ 8 + 1$
$= 9$ approx

$\log_2 2^8 + 1$
$8 + 1 = 9$
$2^8 = 256$

→ Here, we use $(\log_2 200)$ also, बस Index में तो unordered data को ordered/sorted होकर ही जाएगा ना।

we use $\log_2(n)$ here.

54. Types of Indexed :→

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

key → uniqueness,
non key → not unique,

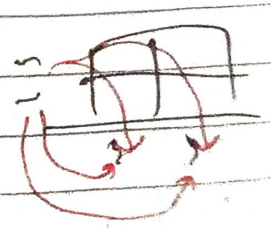
Imp.

(#) Table :-

ordered file	primary Index	clustered Index
unordered file	Secondary Index	Secondary Index
	key	Non key

At most 1.

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 :→

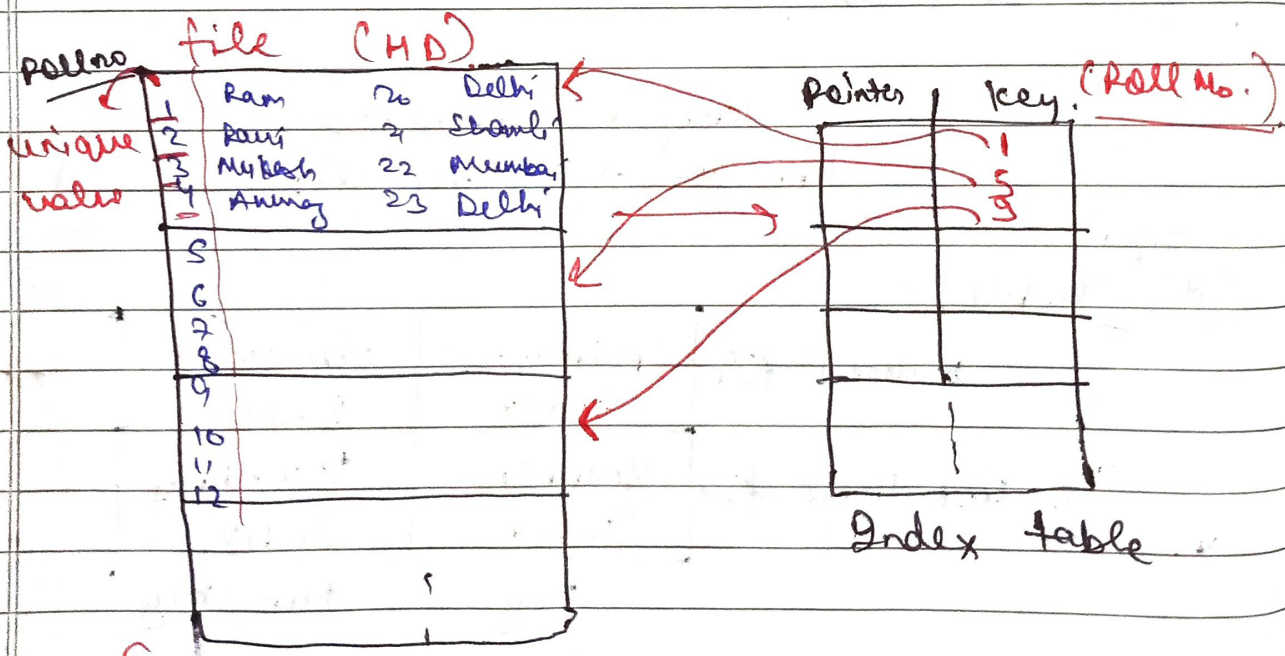
→ In a table, if we made any of att. primary key, then By default → primary Index is operated on them automatically.

→ **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)

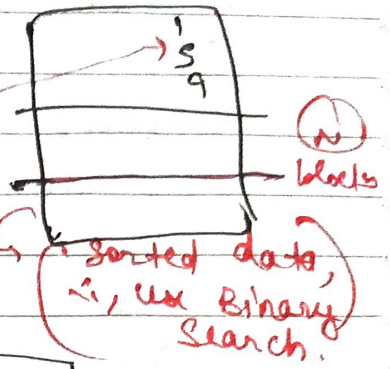
So, we use Sparse here.

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

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

Primary Index is sparse.

Ex:1) we have to find 3,
then,
we know it is found in block of '1'.



Hence,

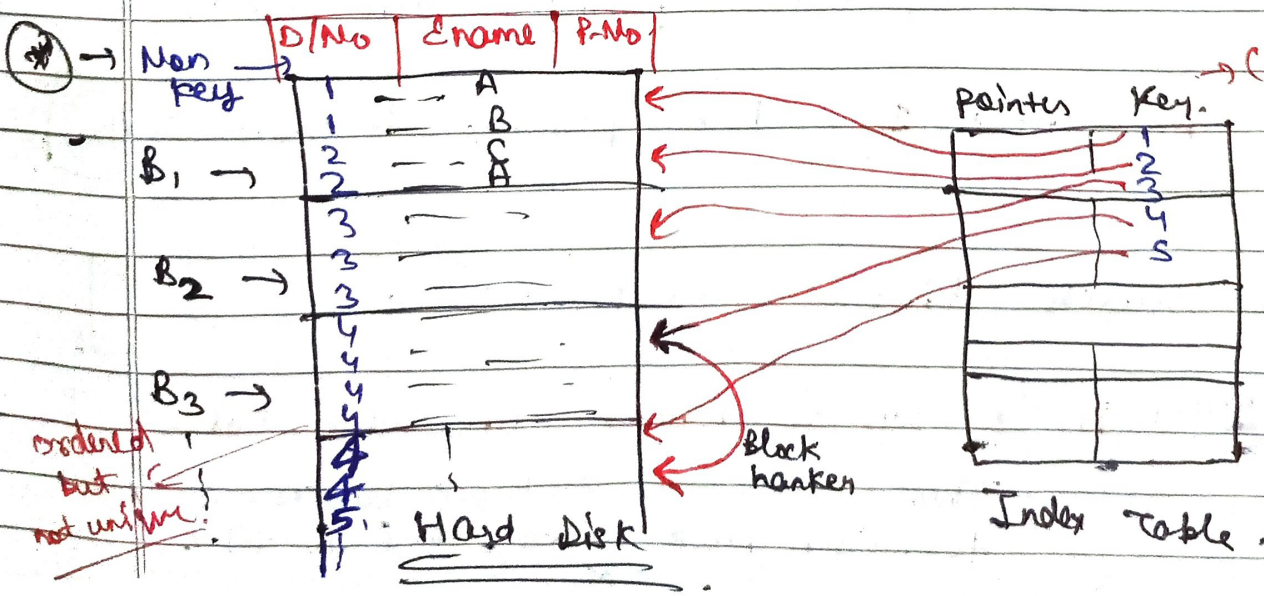
$$\boxed{\text{Total Search Time} = (\log_2 N + 1)}$$

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

---

36 Clustered Index :→

→ data must be ordered & with Non key. (ie, not unique).
↳ ie,
(value may be repeated but must be sorted.)



→ 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 pointer] 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.

→ इसे एक से ज्यादा primary Index & Cluster Index नहीं बना सकते, bcz no need. & 1 Table has only 1 primary key ही।
So,
→ At most 1, (both primary & Cluster Index)

→ Total Search Time = $(\log_2 N + 1) + 1$

Where, N is No. of blocks in Index Table.
(These extras are ∀ Block pointers. for, Each block pointer, there is +1).