What is indexing in SQL ?

- Indexing creates a look up table to speed up data retrival by creating pointers to where data is stored in a table.
- # Indexing use B-Trees data structure for quicus search
- # Indexing requires disk space for storage
- # Indexing used in read ~~extensive~~ intensive database not in write intensive database.

Concept: # Indexing requires disk space ??

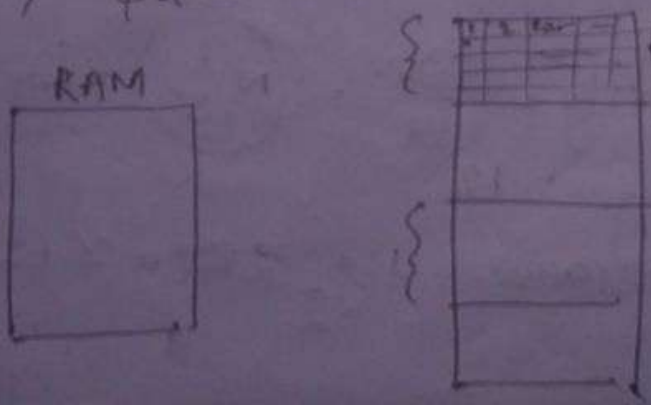
- Our data is always stored in Hard disk as logical Blocks or logical pages. So the number of rows that can fit in a data page / Block depends on the size of each Row.

~~Let say if the size of Block is~~

Let say one block can store 100 Records and in our data table we have 1000 records then for storing that data we need $\frac{1000}{100} = 10$ blocks/pages in our Hard disk

So How searching work without any indexing ↓

Let say we write -> Select * from student where RollNo = 14;



- when we hit Run that code one by one block come from Hard disk to RAM and searching for that particular data
- This searching / computation process is called I/O cost.

  # of I/O cost ↑ time ↑.
- with out indexing if we search it's go for each block and cost high I/O cost.

  Means Time complexity = O(n) (without any indexing) (also called Linear Time)

Q// Consider a Hard disk in which BLOCK size = 1000 Bytes, each record is of size = 250 Bytes. If total no of records are 10000 and the data entered in Hard disk without any order what is avg time complexity to search a record from HD?

Ans:- each record = 250 byte
total record = 10000
total data size = 1000 × 250 = 250000 byte
each block = 1000 byte
total block required = $\frac{250000}{1000}$ = 2500 BLOCK required for store this data

avg time complexity = $\frac{2500}{2}$ = (N/2) time complexity.
= 1250

# of data is ordered $\log_2 2500 \approx 12$

But if data is ordered and without any indexing

time complexity = $\log_2 N$

unordered = time complexity = N

Q// what is avg time complexity to search a record from Index table if Index table entry = 20 Byte (key + Pointer) 10 B 10 B ?

Ans# # size of one block in index table = size of 1 block in HD table.

So 1 Block in index table = 1000 byte

1 record in index table = 20 byte

How many record we can store in a single block of Index table

$= \frac{1000}{20} = 50$ Record we can store in a single index

Now How many records we have to store in index table?

Here Dense and Sparse concept used.

Always remember =>

if data is ordered we use sparse

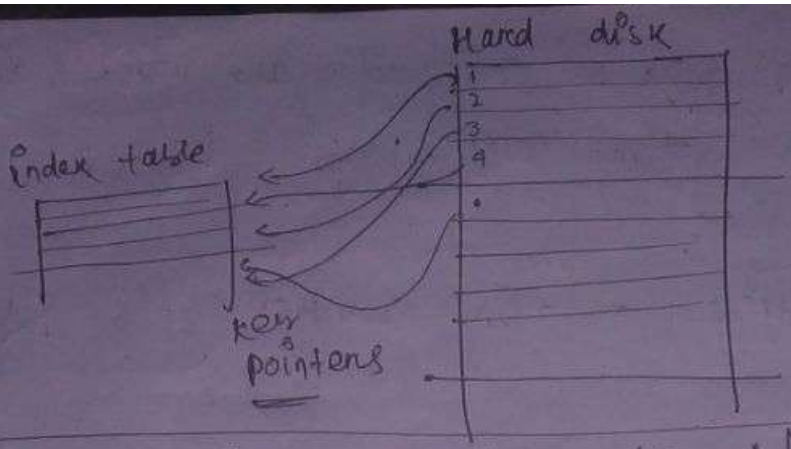
if " " unordered we use Dense

in case of Dense we have to store complete 10000 records pointers in index table.

So we need total = $\frac{10000}{50} = 200$ blocks in index table

So time complexity = $\log_2 200 + 1 = 8 + 1 = 9$

For Searching in index table — then search in Hard disk Block



in Dence time complexity = $\log_2 N + 1$

- in case of sparese data is ordered that's why we don't store all data in index table we just store the enchore data for just the first row of each block in HD.

So how many block's needed in index table

$= \frac{2500}{50} = 50$ blocks

Time complexity = $\log_2 50 + 1 = 6 + 1 = 7$

in Sparese time complexity = $\log_2 N + 1$

Types of indexing :-

① Primary indexing
② Secondary indexing
③ clustered indexing

of data is ↓	ordered	unordered
of data is unique	primary index	Secondary index
of data is not unique	clustered index	Secondary index

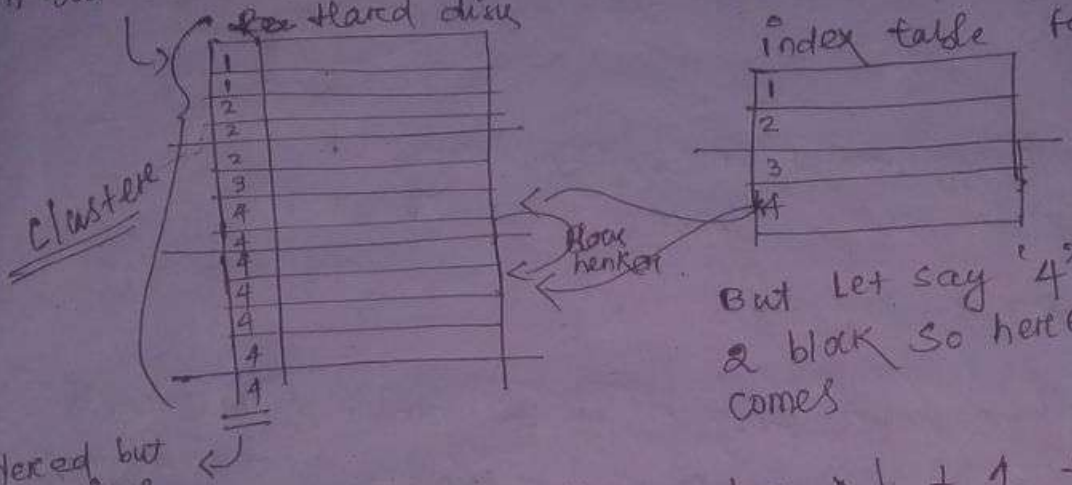
Primary index is usally made as Sparse

- we make it Dence but we have to manage many pointers and need many blocks.
- Time Complexity => $\log_2 N + 1$

Complex index is also sparse.

But here block henker concept come

in complex index we store singe data fore an single



ordered but not unique.

But Let say '4' Present in 2 block so here block henker comes

Time complexity = $\log_2 N + \underbrace{1}_{\text{Block in hard disk}} + \underbrace{1}_{\text{Block henker}} + \underbrace{\cdots}_{\text{If many block henker comes}}$

$\log_2 N$: Index table

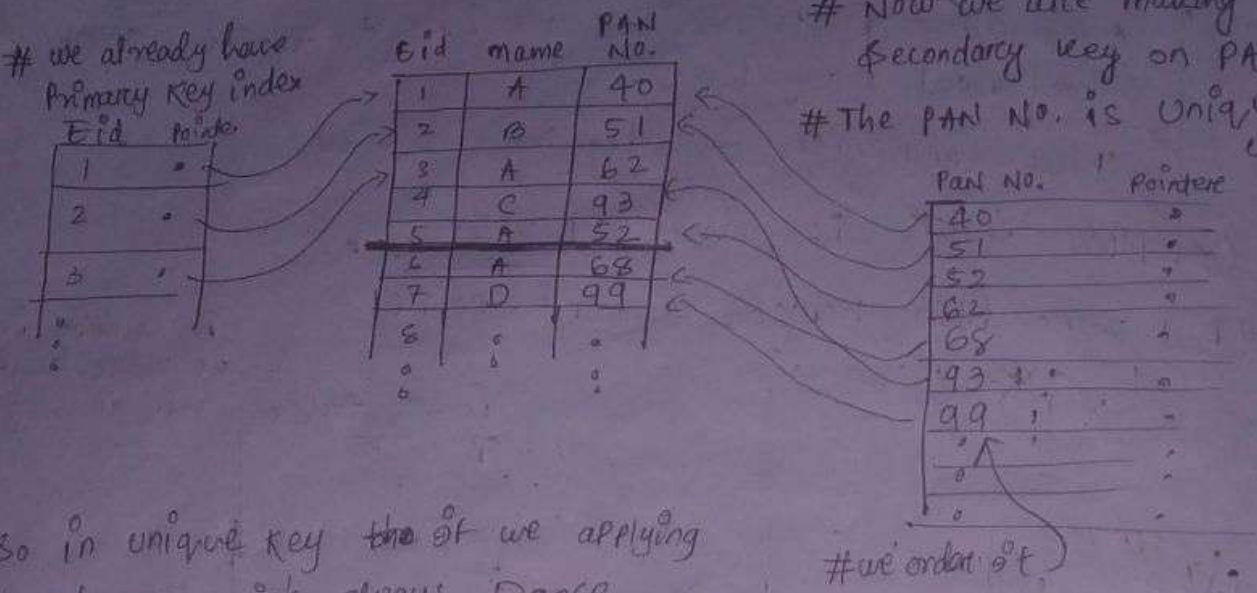
SEcondary Index :

- like the name suggest Secondary here we already have a index like Primary index, then we make an second index fore efficiently access the records.
- Two cases will forem, where we applying Secondary index one is when the attribute is unique and one when not unique.

Always Remember when our data is unordered we use Secondary index

- Then two case forem -> Unique
- or not unique.

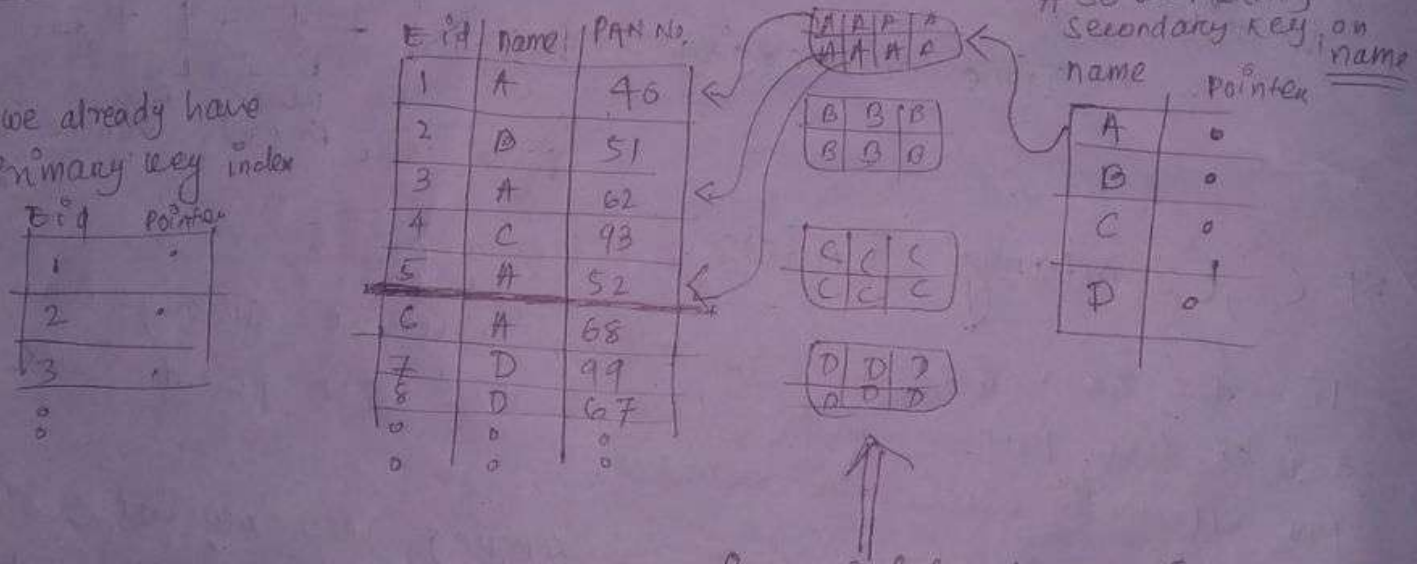
Example-1 Case-1 when we have Unordered & unique data



#So in unique key ~~the~~ of we applying Secondary key it's always Dense.

Time Complexity $\log_2 N + 1$

Case-2 when we have Unordered & not unique data.



Here an interemidiate layer will form (Block of Record Pointers.

Here also it's always Dense.

Time complexity => $\log_2 N + 1 + 1$

1 fore interemidiate layer.

1 fore finding the Perticular block.