Agenda

- → Views (last class)
- → Intro to Indexing
- → How Indexes work?
- → Indexes and ranged queries
- → cons of indexing
- → Indexing on multiple col^m
- → Indexing on strings
- → LiL BiT Practical.

start @ 9:10 PM.

Intro to Indexing

"select * from table where id = 'someth"

for row ....

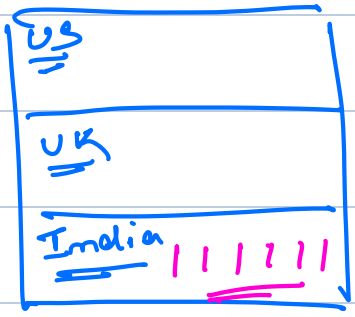
O(n) ?

SLOW!!.

(2M) we don't want to iterate over every row all the time.

Library to get a book. history of India

librarian → pointer (history section).



× 30 books

× 30 books

{India}, 20 book.

(2000) ⟹ (20)

→ what is at the starting page!!.
(Table of contents).

⟹ end
(index)

word : page 34, 39.
sorted

→ Tables are actually stored on the disk

CPU — L1, L2 caches — fastest. (small).

Memory (RAM) — (fast).

disk — (slow).

operating system / kernel.

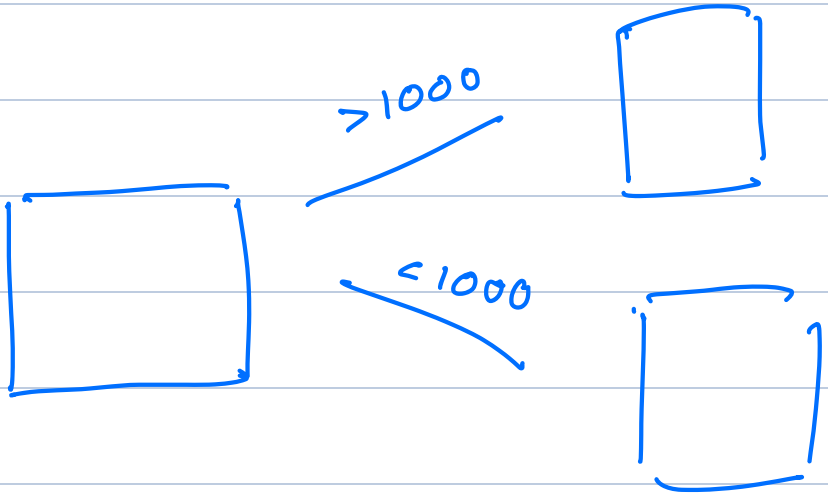
can only read from the memory.

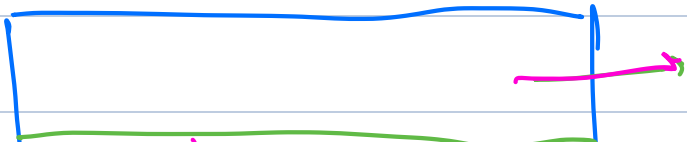
disk reads are slow!!.

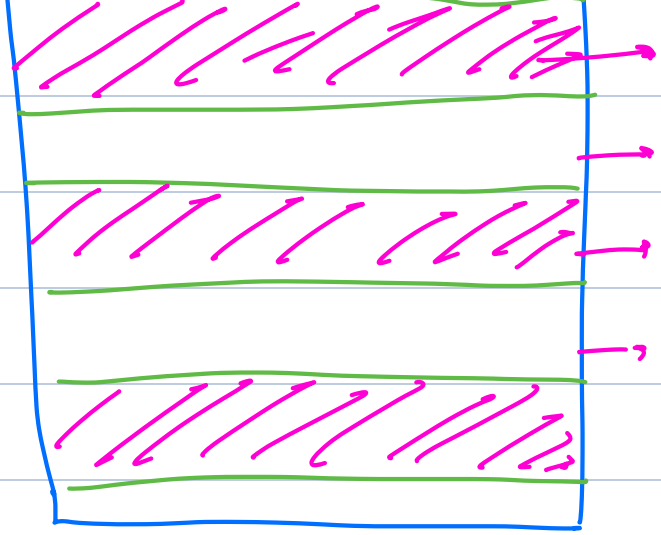
→ avoid reading from disk.

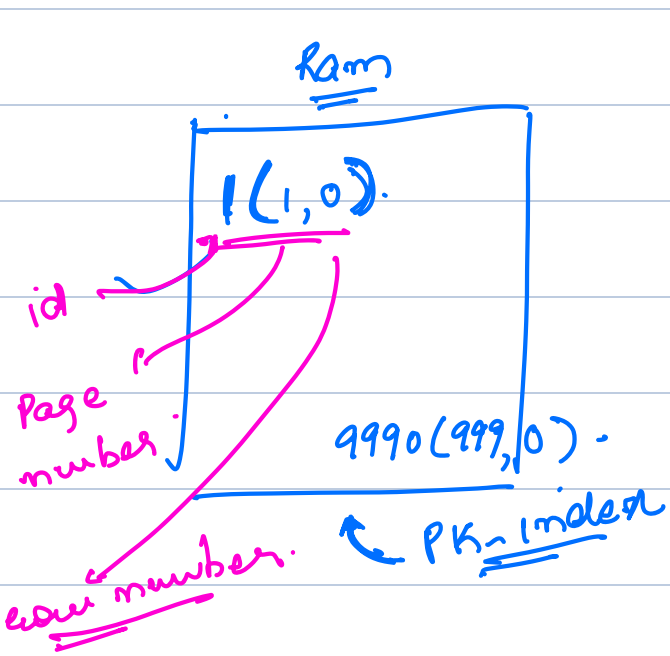
→ slow.

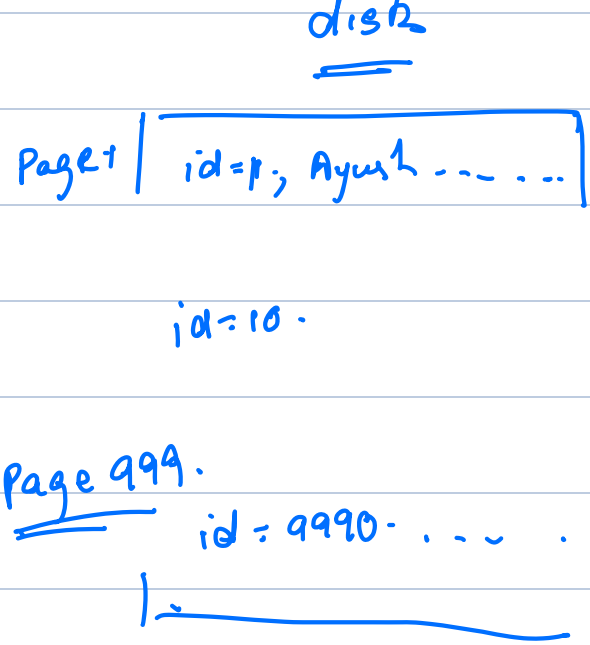
→ expensive











id (1) is in Page 1 on 0th row.

id = 789 10^3

999 pages. { full table scan }.

"select * from students where id = 789"

used PK-index

To become

faster.

Indexes essentially help you reduce the look up on the disk for every query.

How do Indexes work??

→ "select * from table where id = 100"

find page where id (100) is stored.

map.

id	page.
100	1
102	2
104	4
⋮	

O(n) ✗

↓

O(1). read.

without map (O(n)).

with map (O(1)).

→ "select * from users where name = 'Ayush';"

name.	page.
ayush.	[1, 3, 5, 7]

page. 1

Ayush.

Siv	[3, 4, 5)
Shreyas	(6, 10, 11)

page 63
Ayush.

without map (1000 words) n pages.

page 257
Ayush.

with a map (3, 4 pages).

Ranged queries

" select * from students
where
PSP between 40.1 and 90.1; "

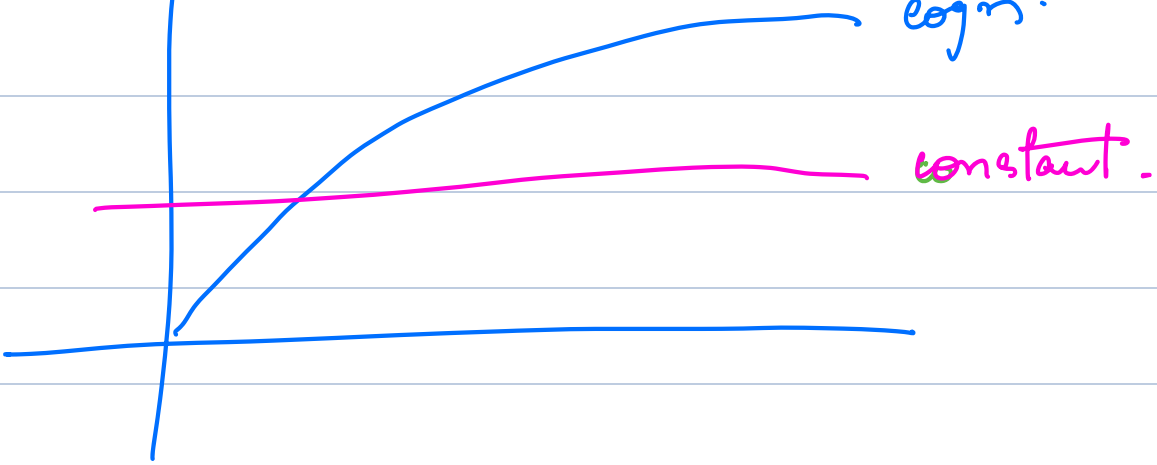
worst case map ⇒ range O(n).

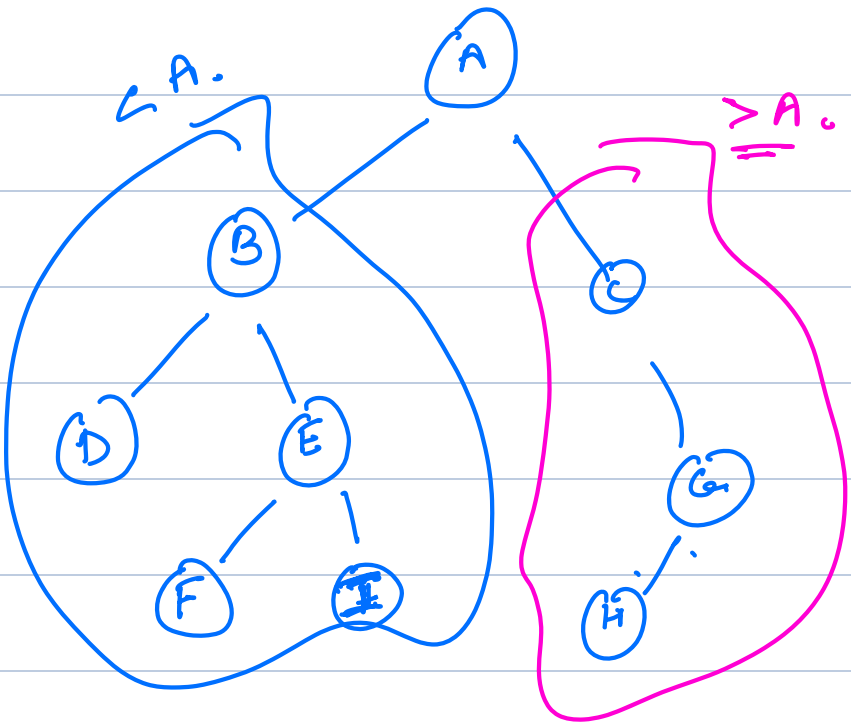
① sorting helps searching
② we want faster look-ups.

Maps are not sorted.

⇓

ordered maps
tree maps
(Balanced BST).





① go to 40.1

② keep going / traversing till next node > 90.1

Data structures : B Tree / B+ Trees.

① How we storing more info in reduced height

② can we fetch more info on a single look up?

③ B trees are specifically designed for storing data on disks where on a single fetch, we can get more info.

Break Till 10:55

Cons of Indexing

C R U D.

Reads are faster!!.

① writes are slower.

→ also update index

index are also stored on disk.

Indexes are stored in memory.

② More storage space on disk.

→ never create indexes prematurely

$2 * 10^9$ bytes

Indexing on multiple col^n.

id name email PSP.

→ select * from stu where PSP = 80.

index (name (PSP)).

① sorted
② lookups

order by (name, PSP)

index (name, PSP).

create Tree → if name match sort on PSP.

Ayush	30.
Siv	90.
Kavya	60
Suvajit	30
Ayush	60.

(PSP, name)

30	Ayush
90	Siv
60	Ayush.

Index (name, PSP)

≠

Index (name)

&

Index (PSP).

Query on	Index	help	not help.
name	PSP.		✗.
name	(name) and (PSP)	✓	
name	(PSP, name)		✗.
name.	(name, PSP)	✓	
name = x and PSP = y	(PSP, name)	✓	

name = x and PSP = y.

name = x

or

PSP = y.

(PSP, name).

select * from

where name = xp
psb = 4.

(A) or (B)

if a query is a col^m x.
any index with x as prefix will help.

(x, a, b, c) → help.

(x) = help.

(a, x) → won't help.

(x, a) → help.

Indexing on strings

select * from students where email = "abc@scaler.com"

slow (no index string matching).

→ should we index the string completely or partly?

@ gmail . com .

space

Ayush . saraswat .

Ayush saraswat .

faster ?? . whole string

less space first 5 letter

email	page
Ayush.saraswat	1 .
Ayush saraswat	2 .

Ayush [1, 2]

users 2 Billions

index on number of characters	distinct keys in index	page access / fetch.
1	26	2B/26
2	26 × 26	2B/(26)²
⇒ char	(26)⁷ .	≅ 1

2B

a ... [2B/26]

b [2B/26]

c

D

7.

increase in number letter

space ↑

fetch ↓.

Select * from Table
where address like '% Ambala %'.

queries like these, don't use index
and do a full table scan.

'ambala%'

could use an index