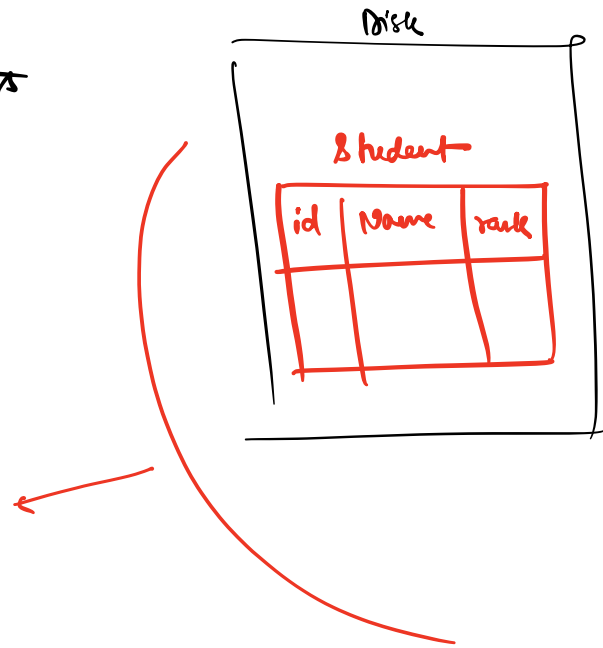
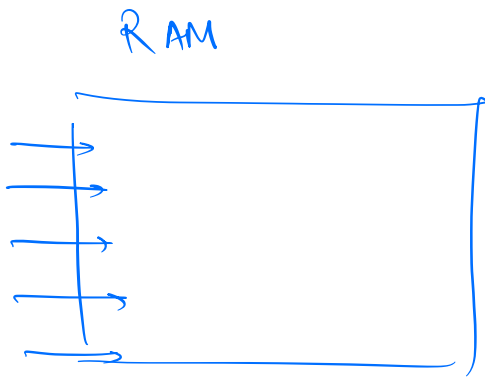


Database \rightarrow data is stored on a disk.

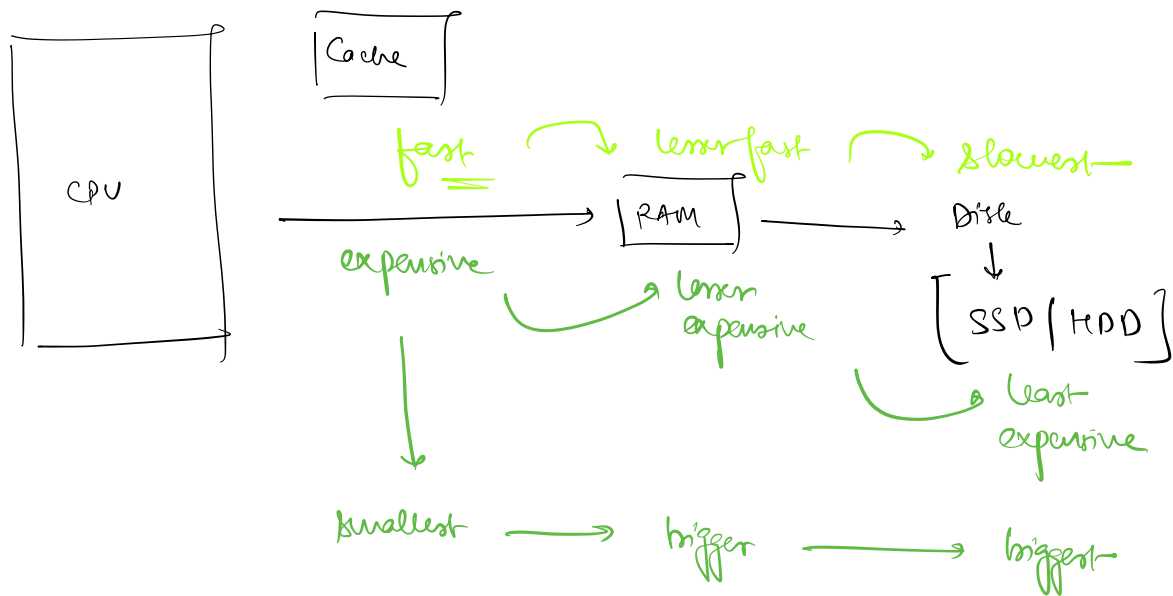
\Rightarrow Select * from students
where rank = 2;



- * OS cannot directly perform operations on disk
- * content from disk is copied into memory and then app^s start working on this data.
- * If a CPU directly works on a disc, it will be wasting a lot of its time!

\rightarrow CPU will make I/O calls to bring data from disc to memory

\rightarrow while data is brought, it will do something else



: A table is sorted in a disk sorted by primary keys.

(PK)

id	name	batch-id	psp	Address
1	Kartick	3	90	A
2	Sumit	4	95	B
3	Abhishek	7	88	A
4	Kiran	8	92	D

Select * from Students where

batch-id = 7;

→ 20 student out of 1 million goes to memory and fetches the info to RAM to process

use less → IM-26

(PK)

id	name	batch-id	psp
✓ 1	Kartick	3	90
✓ 2	Sumit	4	95
✓ 3	Ashwini	7	88
(4)	Kiran	8	92

Select * from table where id = 4;

[4]

(memory) → present in RAM

batch-id	addresses
1	A, Z, B
2	B, C, D
3	C, D, A, G
4	D, E, K, J
5	E, P, Q, R
6	F, S
...	!

↓ Million records in table

⇒ Student with batchId → 15

Case I • No sorting | No table → IM

Case II • Sorting | no table → 15
(sorted by pk)

Case III • Sorting | table → 1

<u>pk</u>	id	name	<u>address</u>
	1	A	H 111
	2	B	H 222
	3	C	H 333
	4	D	
	5	E	
	6	F	
	7	G	
	

id = 10

⇒ no sorting | no extra table → IM

⇒ sorting | no extra table → 10

⇒ sorting | extra table
→ ①


PAM
memory

id	add
1	#111
2	222
3	333
4	444
5	.
6	.
7	.

Students

id	name	batch-id	psp	rank
		1		
		3		→ A
		3		
		1		→ B
		1		
				→ C

select * from Students where batch-id = 3



batch	memory
1	A
1	B
1	C
3	A
3	B

⇒ no useless disk reads will happen b/c we can check for id before fetching from memory.

∴ This table that has been created for improved reads is called 'Index'

DS for indexes

- 1) Sort by key
- 2) allows fast access

BST → search = $O(h)$

$$\log N \leq h \leq N$$

Balanced BST → AVL / Red black trees

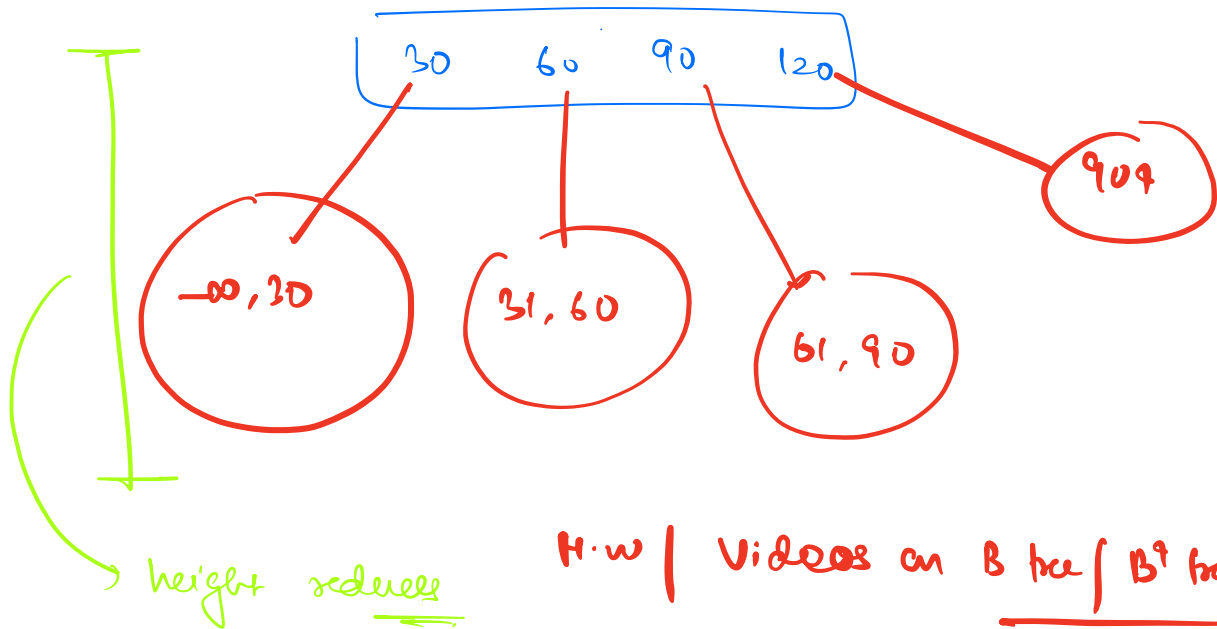
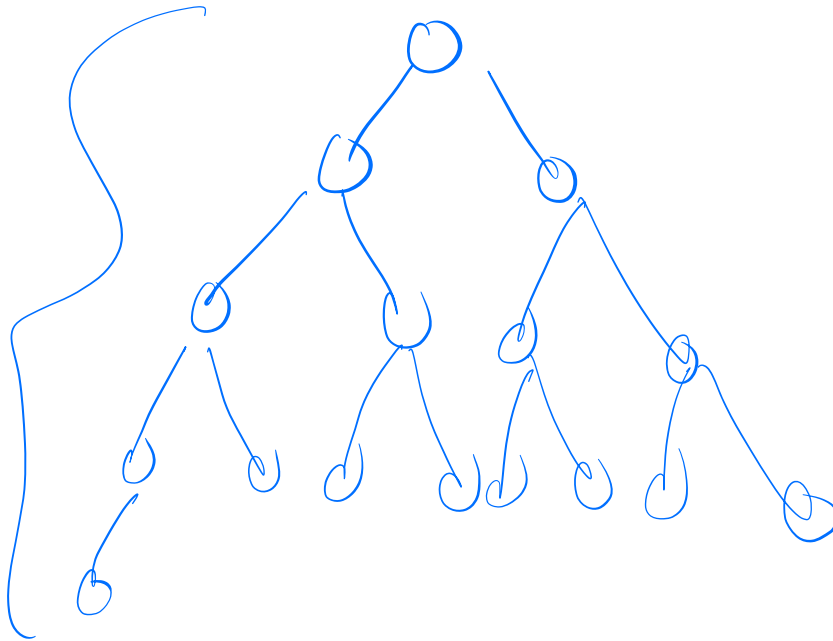
TreeMap
OrderedMap

B^+_{free} / B^+_{free}

16

h : 4

OLW



H.w | Videos on B tree / B⁺ tree

⇒ Indexes :

- Prevents unnecessary disk fetches
- leads to faster queries

⇒ Indexes are updated for all C/U/D queries.

⇒ Indexes are updated during writes, that makes write slower.

→ When to create indexes:-

i) Don't create indexes when you create a table.

ii) Create indexes if you want to speed up a specific query

- ✗ Don't create indexes on prediction.
- ✗ Create indexes on access patterns.

↓
data access

iii) Do lots of performance testing.

Read Heavy

Social
media

Write Heavy

Select * from table where name = "Kiran";

Name	Date	Addr
Kiran		
Kiran		
Kiran		

$$k \rightarrow [1, 3, 6, \dots]$$
$$z \rightarrow [6, 4, 4, 8, 9]$$
$$S \rightarrow [\quad]$$

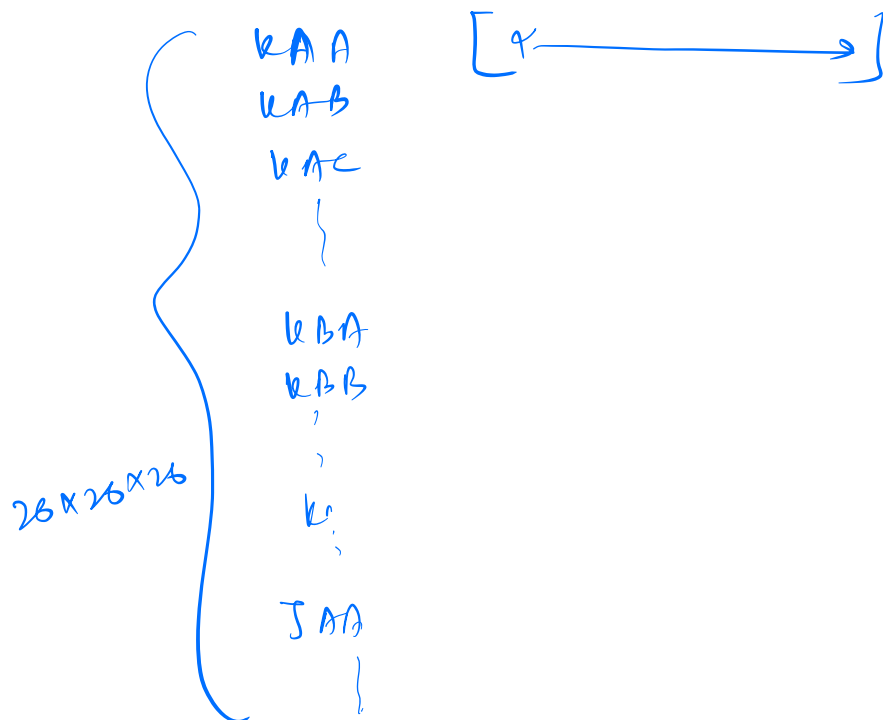
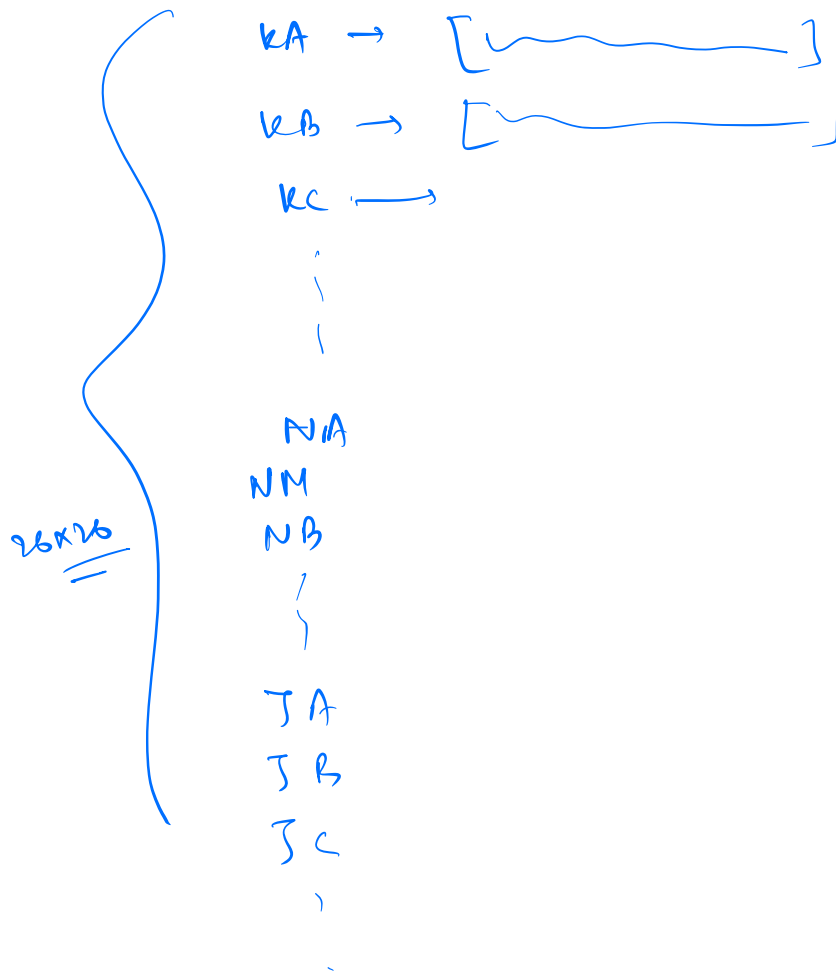
A

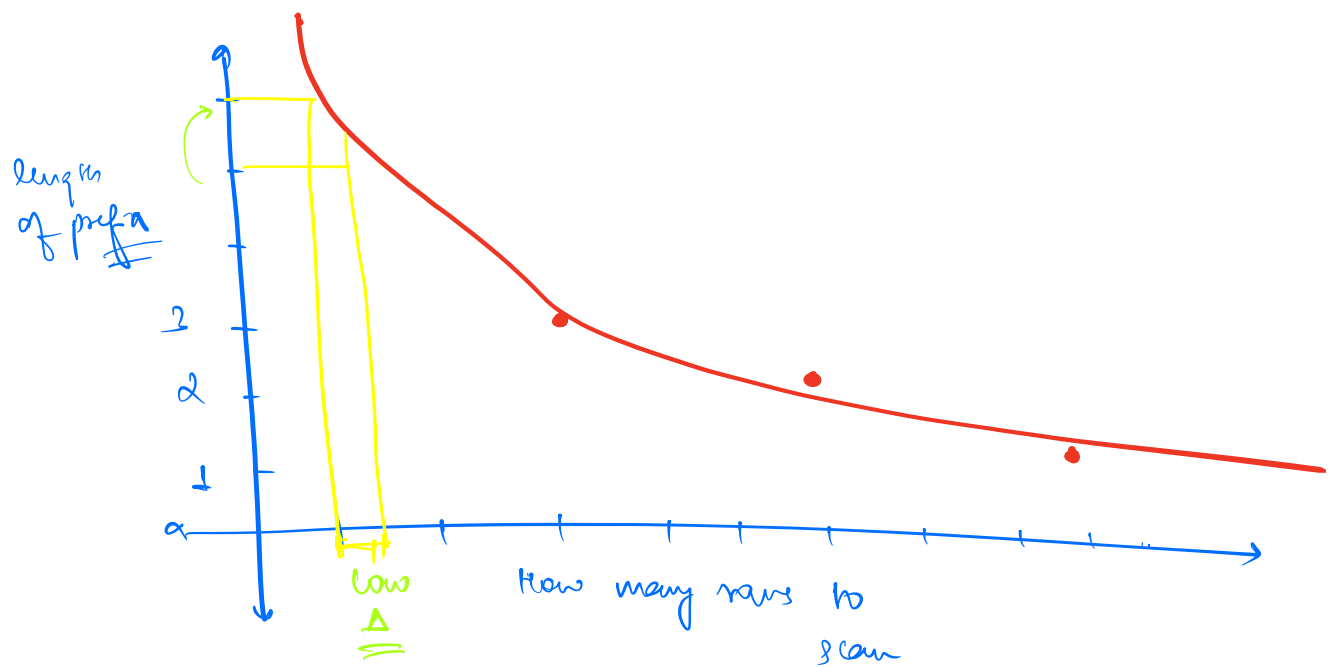
P

Q

—

26





(26)¹
 (26)²
 (26)³
 (26)⁴

power is dependent on
prefix length

two types

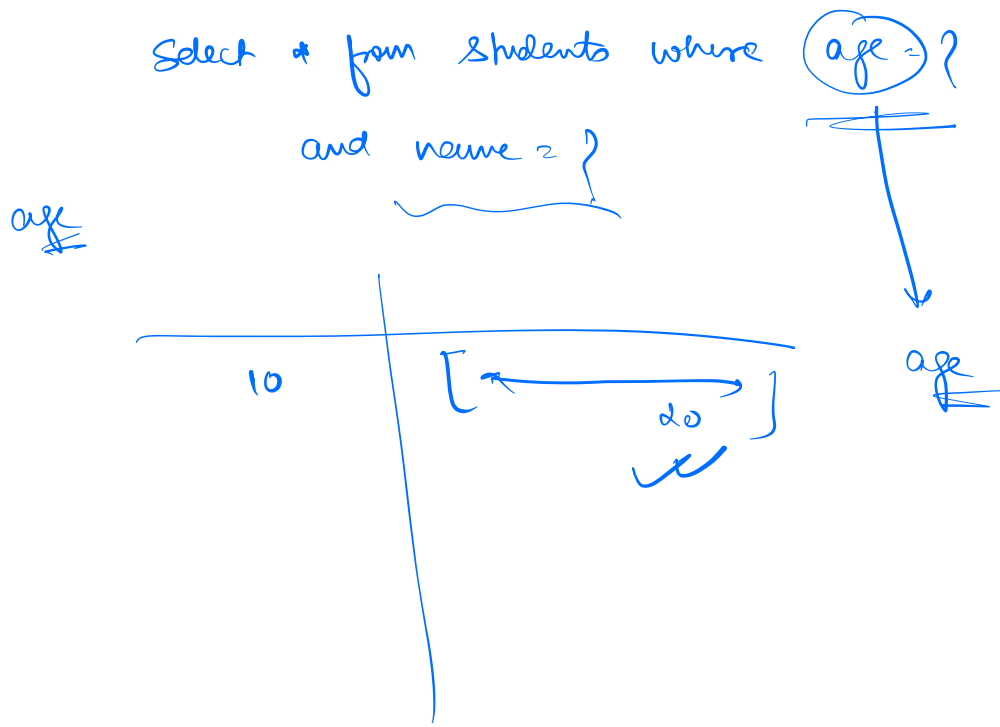
→ 1) Composite

" 2) Full text index

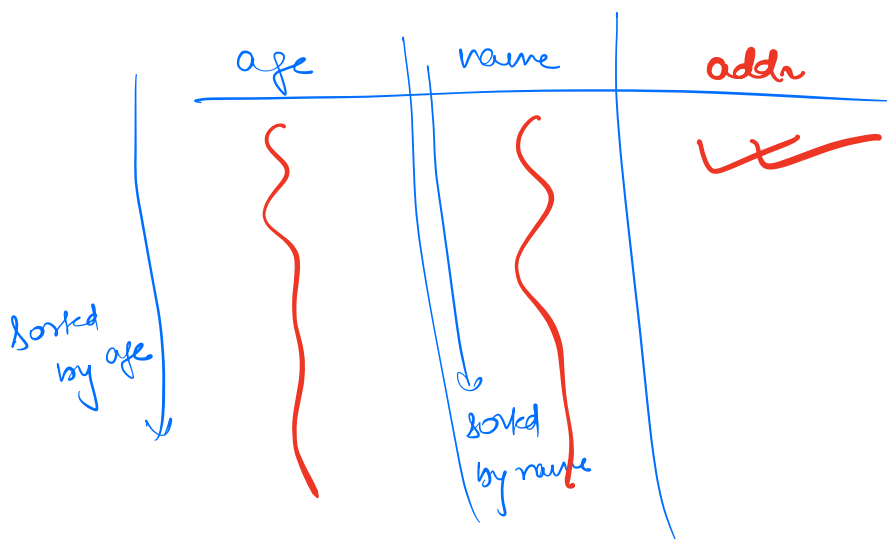


like query

1) Composite Index:



Composite idx (age, name)



age	name
10	A
10	B
10	C
12	X
12	Y
15	P
15	S
15	T

$(A, B) \rightarrow (B, A)$

columns -				
A	B	C	D	E
✓				

idx (A, B, C, D, E)

Query	fast
C	slow
A	fast
AC	fast
ABC	fast

depends (Ac (Abc)

A	C
1	P
1	Q
1	R
2	B
2	B
2	D
2	E

A	B	C
1	X	P
1	Y	Q
1	Z	R
2	Q	B
2	R	B
2	S	D
2	T	E

⇒ composite idk ⇒ choose the columns with higher cardinality before other.

↓
[max distinct
values]