Today's Agenda :-

```
for each row in t1
    for each row in t2
        // Stuck.
```

1M → 10^9 → N^2

→ $(10^9)^2$

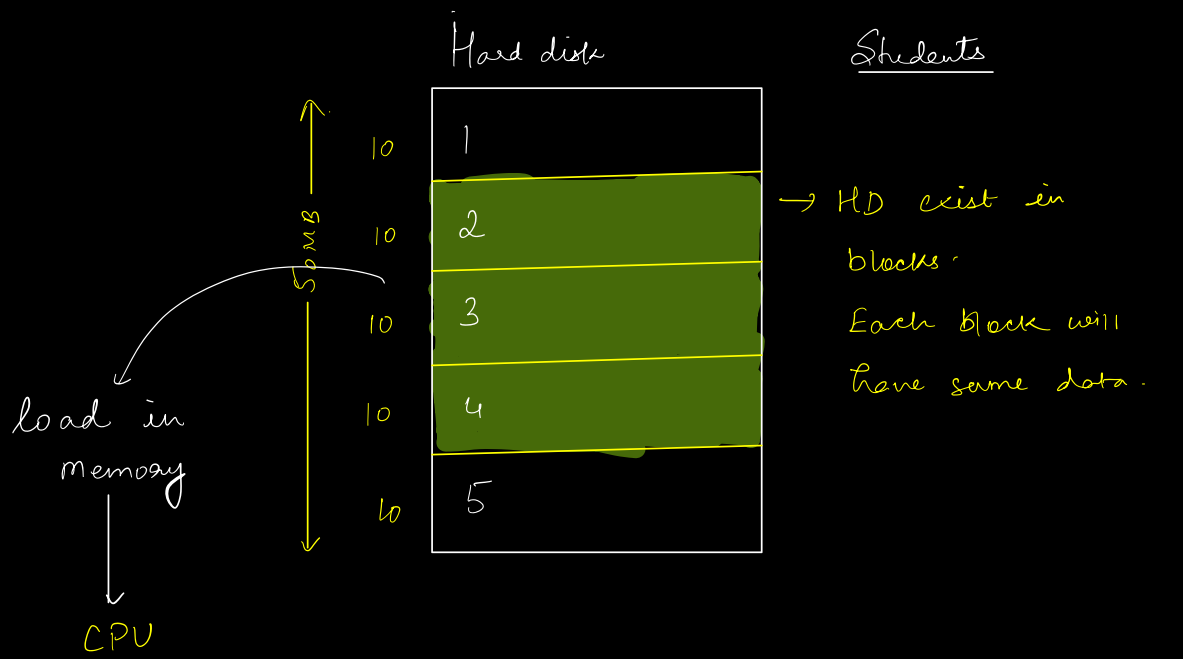
→ 10^{18} iterations

↳ Perf will very slow

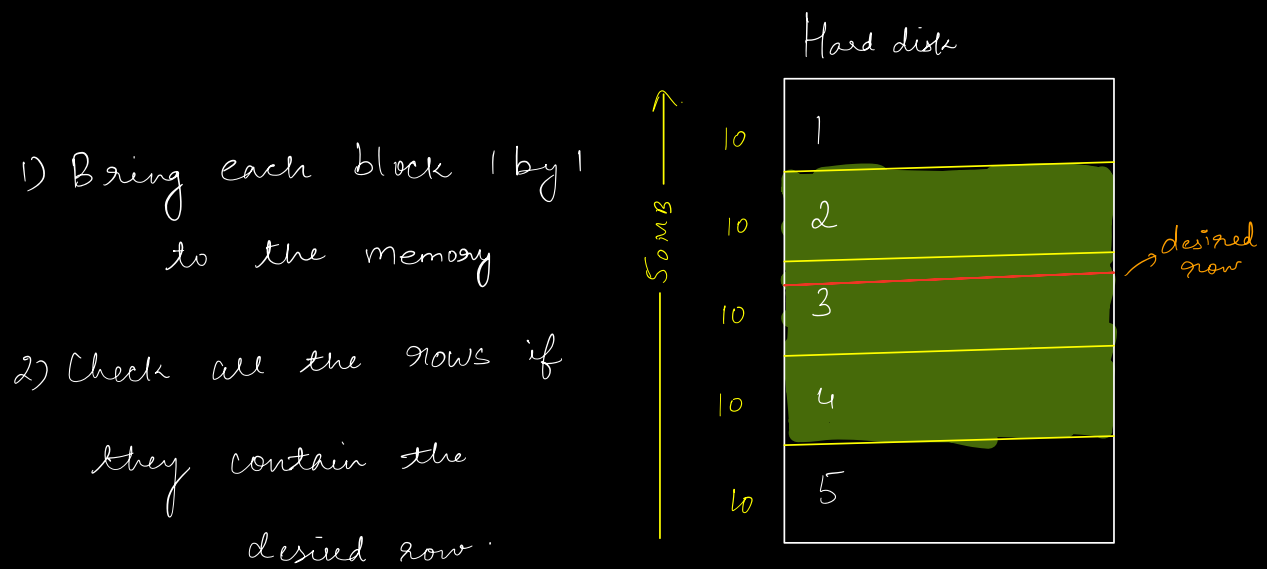
1) DB have to do lot of optimizations to make these queries faster

2) DB stores data on disk.

Disk RAM



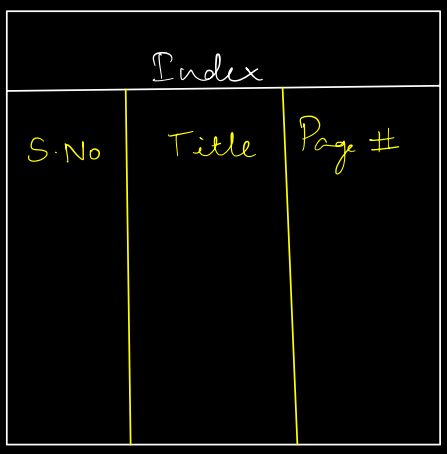

```
Select *
from Students
where id = 100
```



Con :- Unnecessary block are fetched that affect performance.

Merge Sort

CLRS 800 pages



Index in Book - Find a desired page faster

Index in DB - Find blocks of disk which contains desired row faster.

Purpose of Index :- Reduce # of disk block access to fetch data.

How Index work ?

Huge table has millions of rows

```
Select *
from Students
where id = 100
```

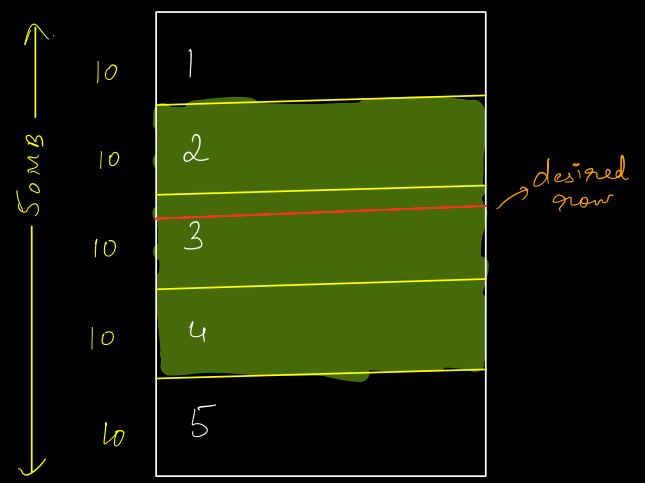
→ Consider that We have an index on id column

Hashmap / Map

Index

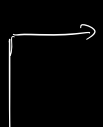
id	Block #
1	1
2	1
3	1
4	2
5	2
⋮	⋮
100	3

Hard disk



w/o index ⇒ 3

w index ⇒ 1



```
Select * from Students
where name = "Umang"
```

→ 7 blocks

id	Name	PSP
1	Umang	
2		
3	Umang	
4		

Index

Name	List<Block>
Umang	[1, 3]
Amit	[1]
Achyut	[6, 7, 8]
⋮	⋮

w/o index → 7 blocks

w index → 2 blocks

↳ where x = y

(2) Find all students with psp b/w 70 & 90

Index (HM)

psp	Block #
40	[1]
60	[2, 3]
70	[5]
80	[6]
⋮	⋮

51.5

```
for (i = 70; i <= 90; i++)
    if i is in map
        map.get(i)
```

Checking a value → O(1)

All values in a range → O(range)

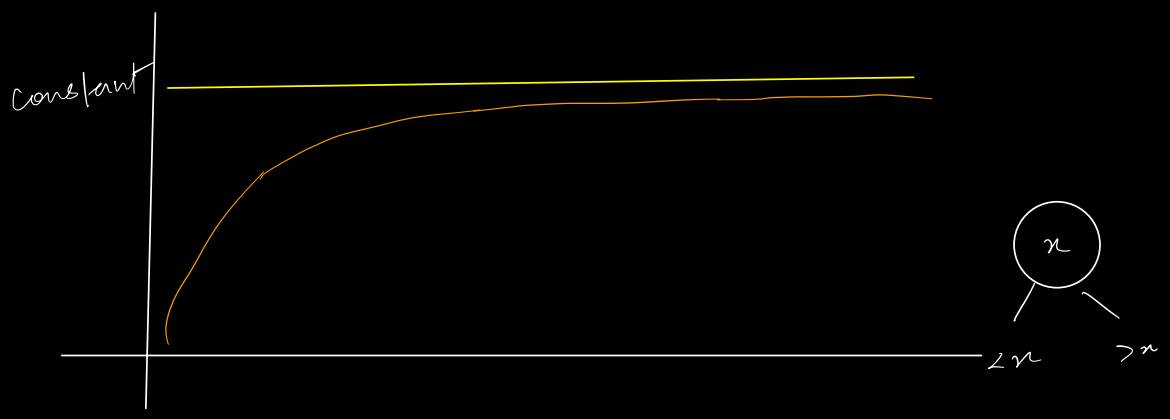
Sorted Keys

TreeMap	→ Java
ordered_map	→ C++

↳ Balanced Binary Search Tree (BBST)

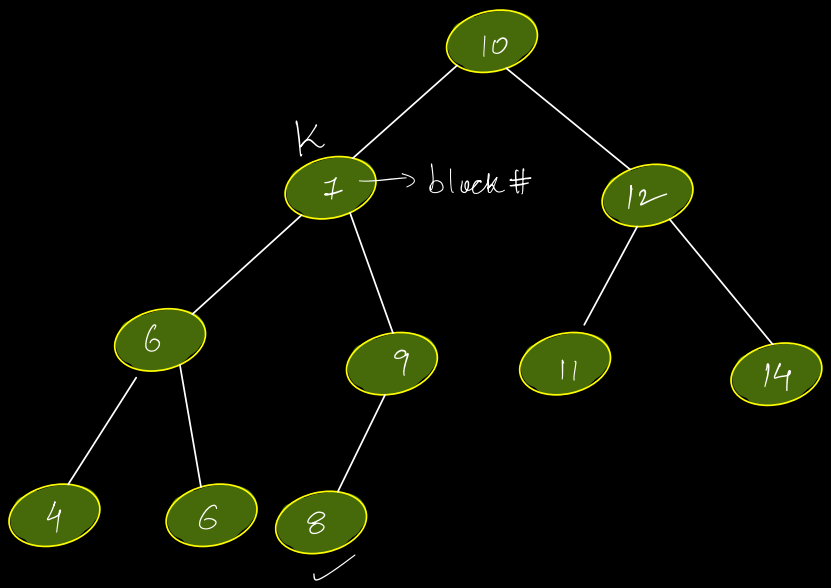
↳ Height of the tree : $O(\log_2 N)$ → No of nodes

Check a value in BBST → $\log_2 N \simeq O(1)$



Range

Get all values in a range from BBST ?



Given a node, how do you next bigger node

$\sim \log_2 N$

(7) & 28

1) Go to 7

2) Keep going to next node till you reach 28.

Indexes in DB work very similar to Treemaps.

Index :- Data Structure that is used to reduce # of disk access to improve performance of SQL Queries.

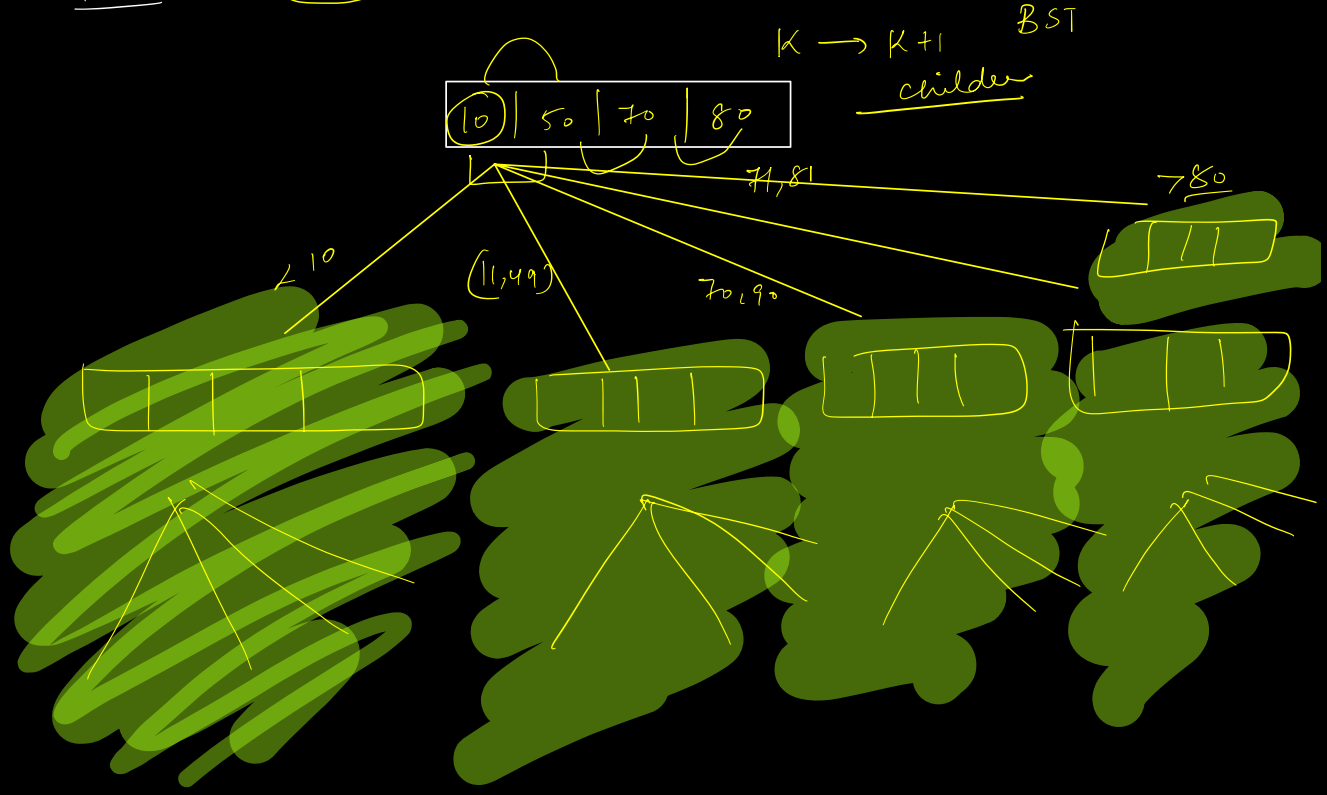
↓

Indexes uses a data structure called B Tree / B+ Trees.

Instead of every node having $x = 2$ children, every node in B/B+ can have $x = K$ children.

↓ Customisable

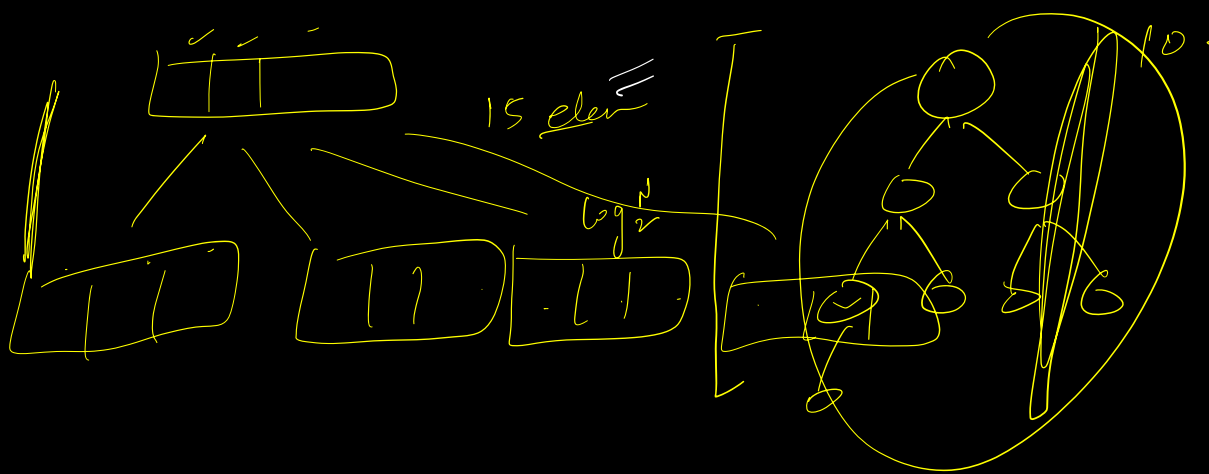
$K = 4$ $x = 4$

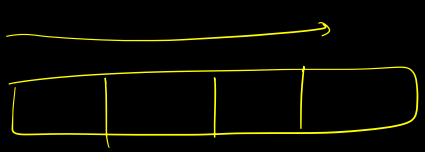


Because BTrees stores multiple elements in node,

height of BTrees $h = \log_2 N$

All operations on BTrees = $O(\log_2 H)$

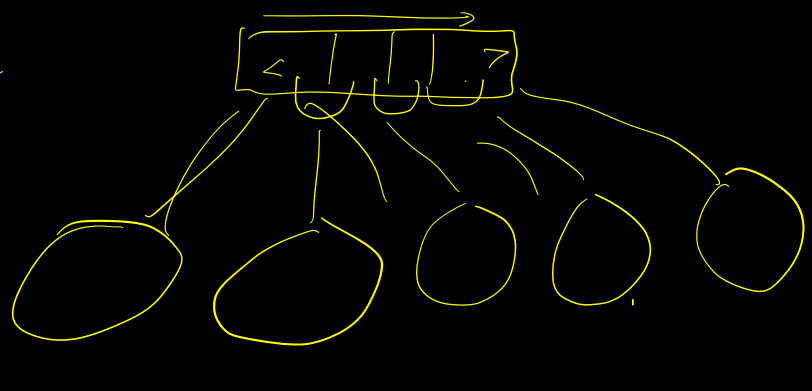




BTrees

$H = \log_2 N$

$O(H)$



① How indexes work ?

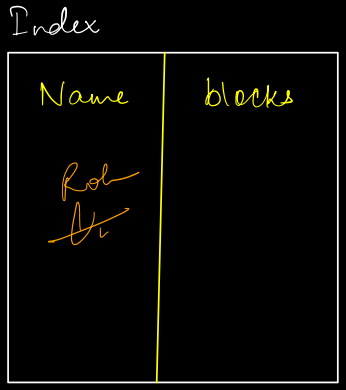
② What is the benefit of index

③ Why hashmap is not good for indexing ?

Break of 7 Min

- How to create index
- Indexing on multi columns
- Cons of Index
- Index on Strings

Cons of Indexing :-



CRUD

(C U D) → Change the data on disk

⇓

Update your index as well.

1) Writes are slower ✓

2) Storage req increases

i) Don't create index prematurely

2) Create an index only when you see the need.

|

Perf metric

Index on Multiple Columns :-

(id)

(Name)

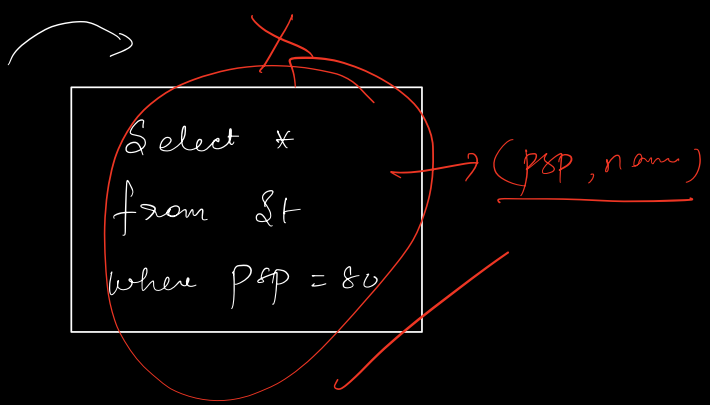
(Name, psp)

(psp, name, rank)

} We can index on mul cols as well.

Assume : (Name, psp)

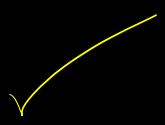
Name	psp
Achyut	76
Abhishek	30
Abhay	50
Abhi	60
Gokul	80
Vinay	51



Creating a index on (Name, psp)

not same as

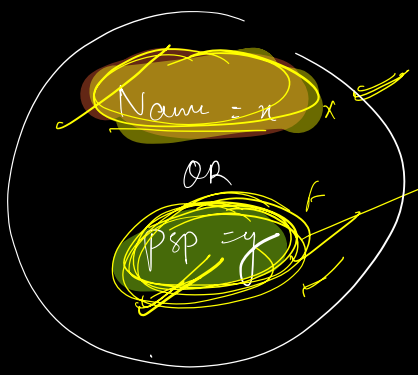
(psp, name)



Scenarios

Query on	Index on	Yes/No
Name	psp	✗
Name	(Name) (psp)	✓
Name	(psp, Name)	✗
Name	(Name, psp)	✓
Name == x && psp == y	(psp, Name)	✓

↳ psp == y & Name == x



(psp, Name)

(psp), (Name)

t
a = b

f
a ≠ b

Indexing on Strings :-

Users

id	Name	email

Create an email ol.

Select *
from Users
where email = '____'

N N
↳ O(N)

N != M

Wrong

Wrong

email	block
umang@scaler.com	2
Smartest@gmail.com	10
Cumang12345 6972@gmail.com	51

1) Size of index will be large

2) String matching will also be slow

→ 1M users

@yahoo.com

↓

10 characters

4B

Stored as unicodes

40 bytes

email	block
umang	2
Smartest	
Cumang12345 6972	

↳ 40B * 10^9

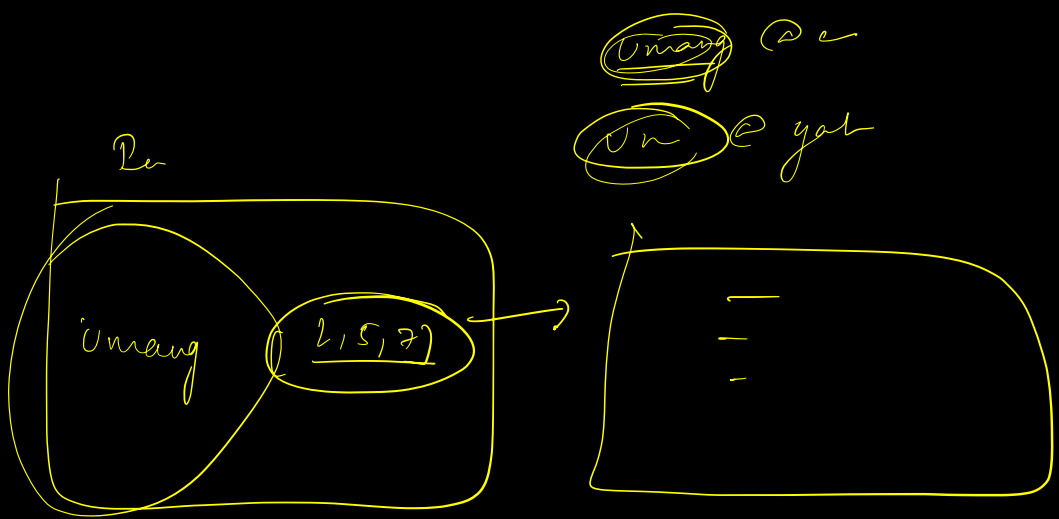
↳ 40 GB

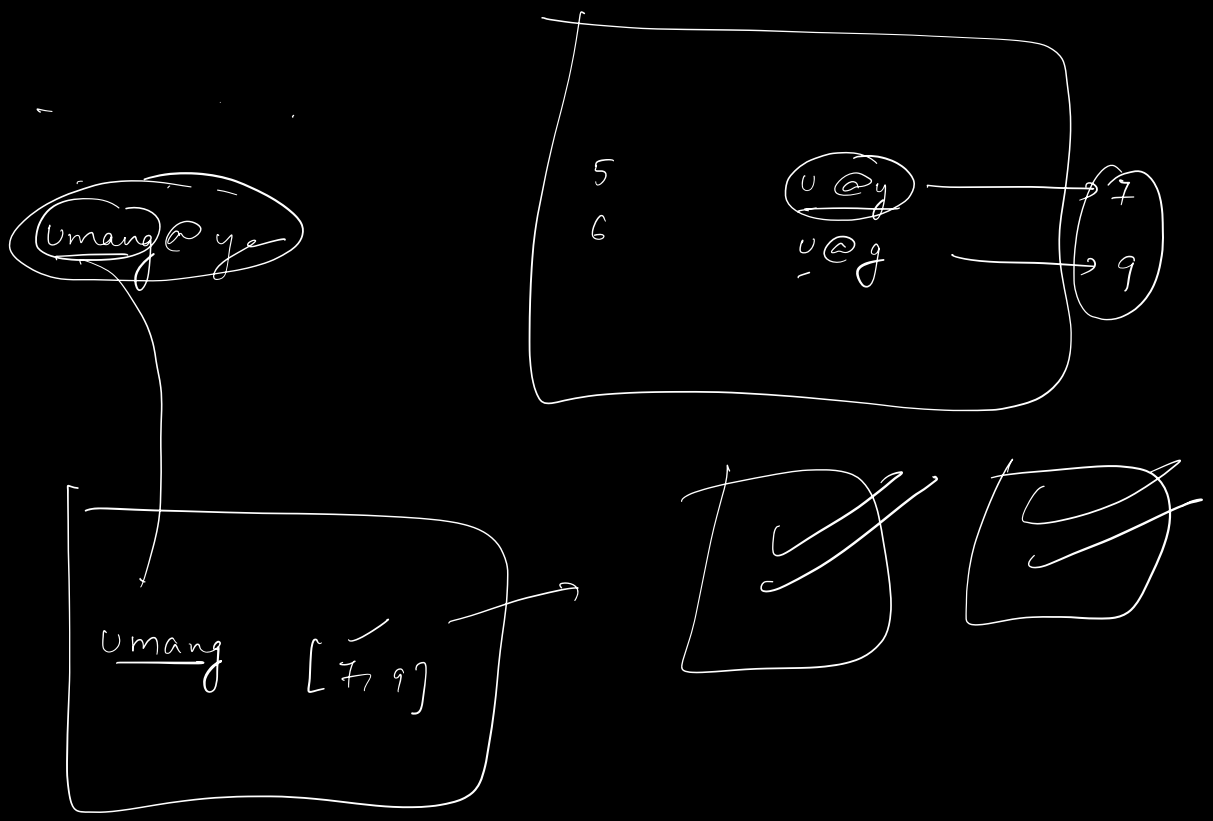
Instead of creating index on complete email, & create & index on only before @.

① Space is saved.

Typically when you have string columns index is created only on prefix of string instead of complete string.

↳ enough performant





How to create index ?

CREATE INDEX ——

On table_name(col_name) ——