Indexes

Agenda

1. What are indexes
2. How indexes work
   - ↳ B & B+ tree
3. Cons of indexes
4. Indexes on multiple columns
5. Indexes on strings
6. Indexes PRACTICAL
   - ↳ EXPLAIN.

Eg for join

```
for i ---- 100;        // 10^9
    for j ---- 100;
        doSomething();
```

} joins

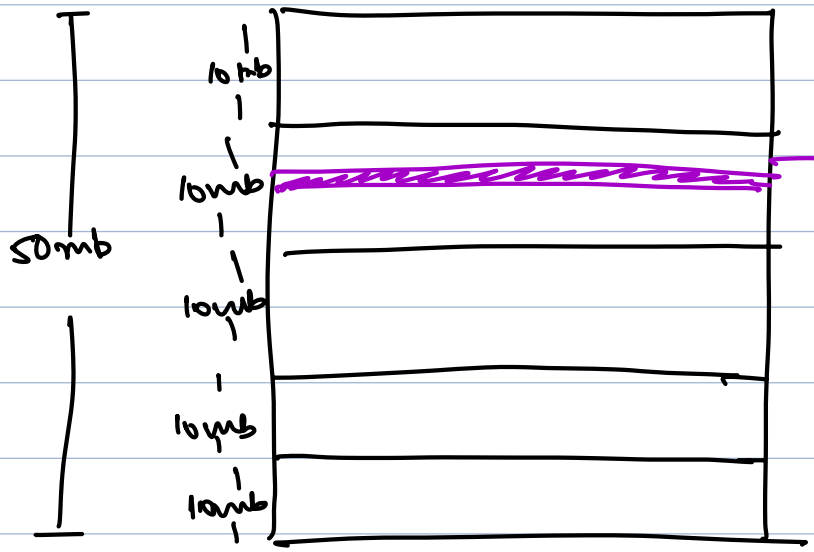
$10^9 \times 10^9 = 10^{18}$

if the dB exactly was doing this

↳ Performance will be very slow.

1. dB have to do a lot of optimizations to make queries fast.
2. dB stores data on disk

Hard Disk

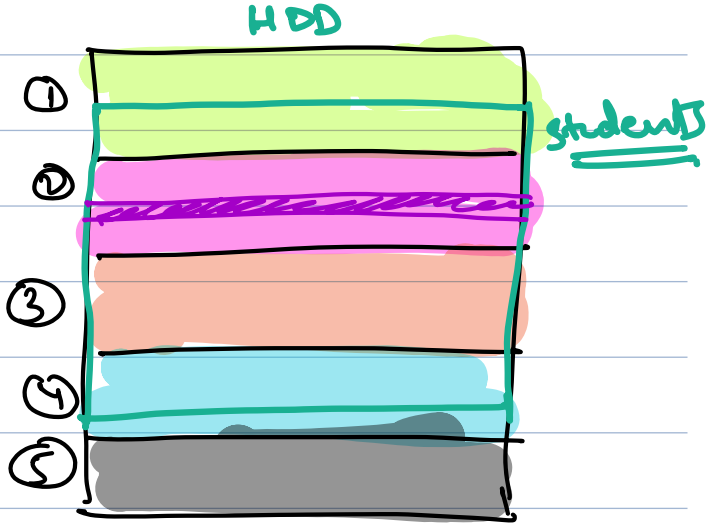


HDD

Each block has data.

Our data is saved on some block.

OS gives the memory complete block that contains the data.



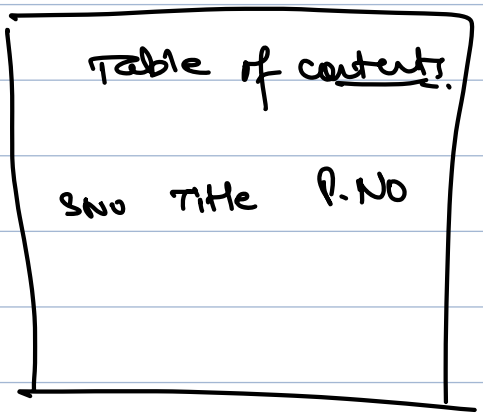

```
select *
from students
where 'id = 100';
```

① Bring block 1 to memory
② Check all rows if they contain id = 100

2, 3, ...

NOT OPTIMAL.

⇒ unnecessary blocks that are being fetched are a waste of performance.



- Index in a book helps find page faster
- Index in a DB → helps find blocks of disk faster

↓

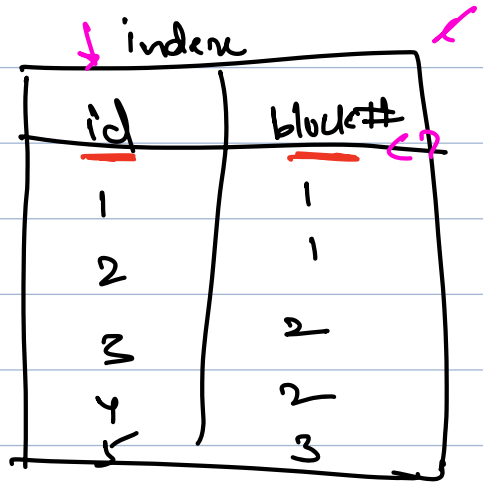
block / blocks that contain the rows that we desire.

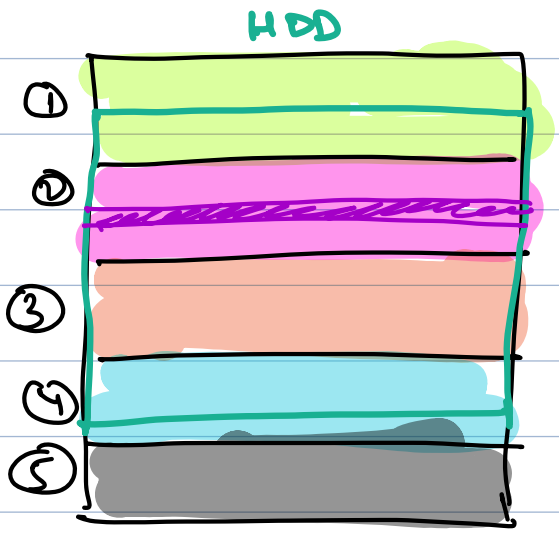
Purpose of indexes: Reduce no. of disk block accesses to fetch data.

How do indexes work..

Huge table 100M records/rows

select * from students where id = 100;





Map | Hashmap
(Key / Value)
(fast)

without hashmap → 4 blocks
with hashmap → 1 block

Q Do indexes work on non-unique columns?

```
select * from students
where name = 'Ujjwal';
```

students

id	name	psp	
1	Ujjwal	100	→ Block1
2	Meera	80	
3	Ujjwal	78	→ Block 3

index

name	# Block
Ujjwal	[1, 3]
Meera	[2]
Ajaay	[4,5,6]

HM < String, List<block no> >

select * from students
where name = 'Ujjwal';

without index → 6
with index → 2

1 ✓
2 ✗
3 ✓
4 ✗
5 ✗
6 ✗

where x = y;

⇒ there are also queries about ranges.

Eg. find all the students with name >= 'M' and name <= 'O';

↓

start with M, N, O

Q find all students with rollno b/w 50 & 70.

index

rollno	block
42	5
66	2
100	8
22	1

treemap < int, string >
get all keys b/w 20 & 40

for i = 20 → 40:
if i in map:
map.get(i);

with a hashmap it is very tough to get all values with a range.

+

not optimal

Hashmap - check a value → O(1)

get all values in a range → O(N)

range query like [L - - - - R]

① Sorted

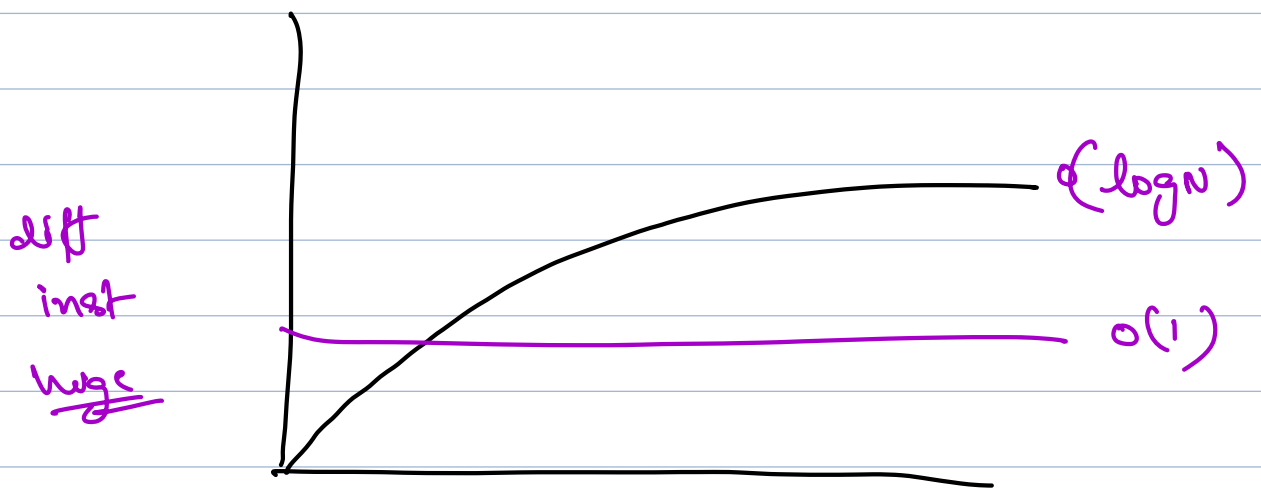
② Key : Value → TreeMap.

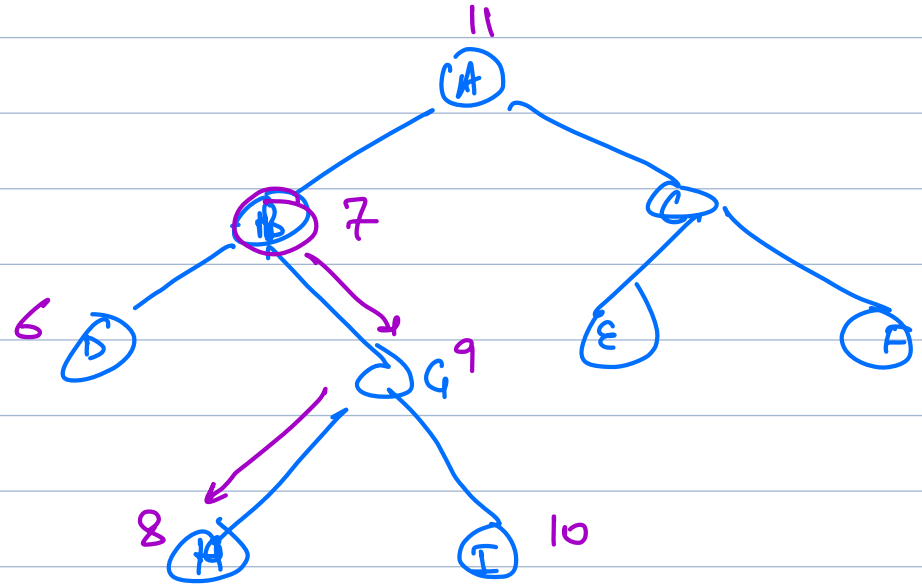
Key → sorted

sorted map

TreeMap → works on Balanced BST

check a value → O(logN)





BST

Q given a node
get the next bigger node?

B → J O(logN)

40 → 60

① go to 40

② keep going to next node till you reach 60

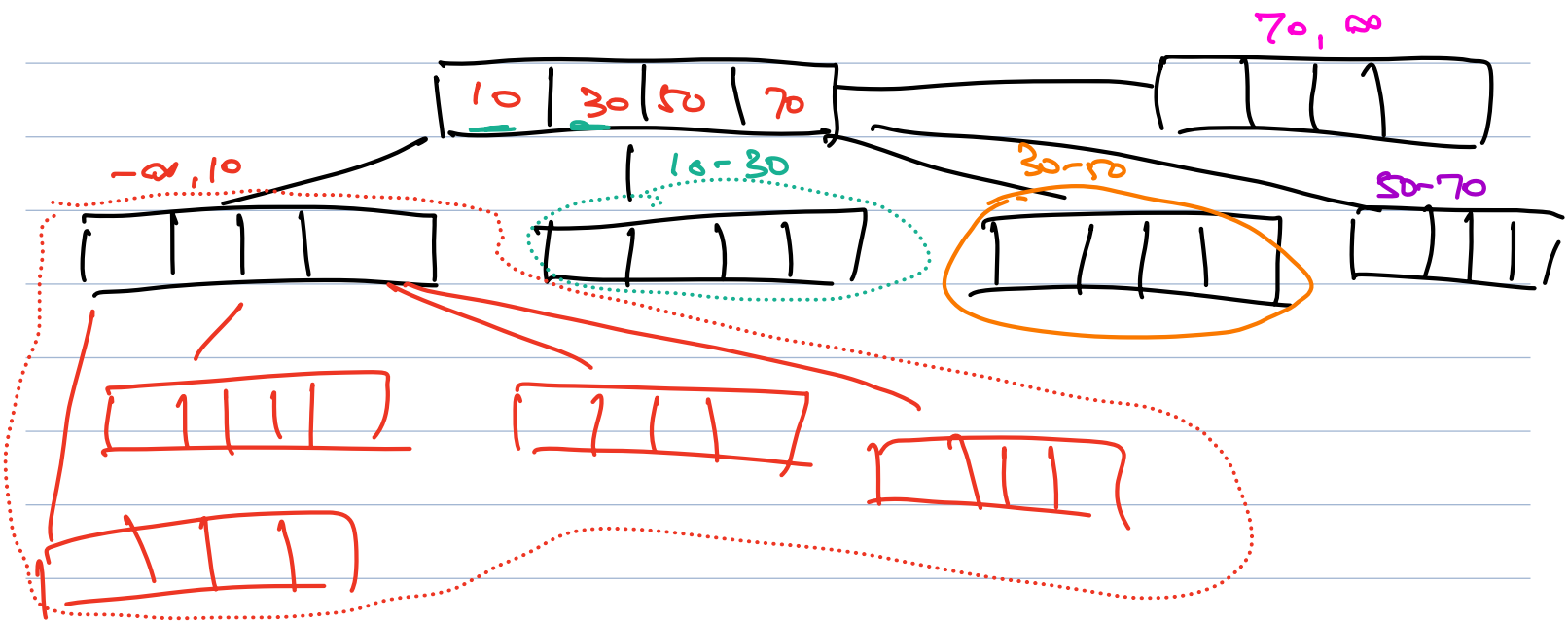
Indexes work very similar to Treemap

↓

Its used is used to used no. of disk access to increase performance of SQL query.

Indexes use data structures called B/B+ trees,

→ instead of every node have ≤ 2 children.
every node can have ≤ m children.



Because a B tree has many children
⇒ The height of B tree will be lower.

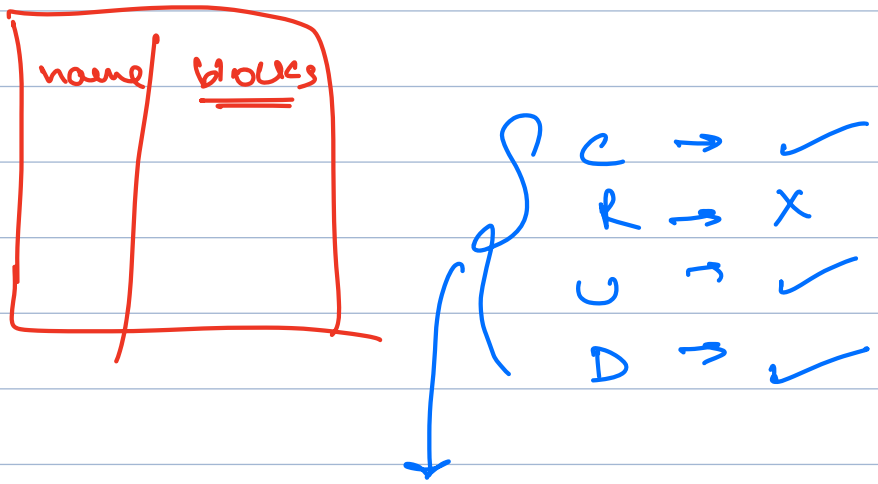
O(H)

Checking a value → O(H)
getting a range → O(H) } faster

Are you clear with:

1. How indexes work
2. What is benefit of index
3. Why hashmap is not good

CONS of indexes.



- Write operation (C, U, D) → changing data on disk

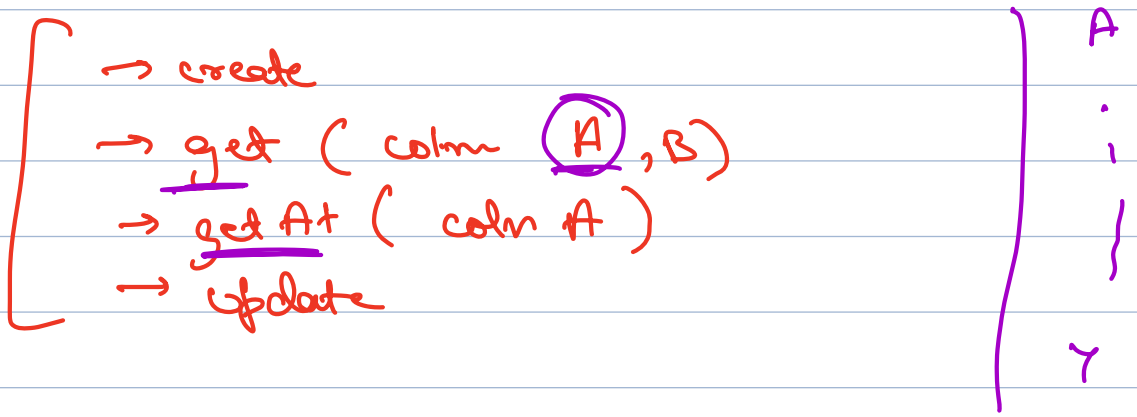
1. Writes are slower.

→ Typically a copy of index is also stored on disk.

2. Storage requirement increases.

* Don't create index prematurely.
→ Create an index only when you see the need of it.

[Break] 08:30 AM.



Indexes on multiple columns.

(id)
(name)

(name, psp)
(name, psp, rank)

→ we can have index on multiple columns..

Index → (name, psp)

query

```
select *
from students
where psp = 80;
```

will this index help ? → NO

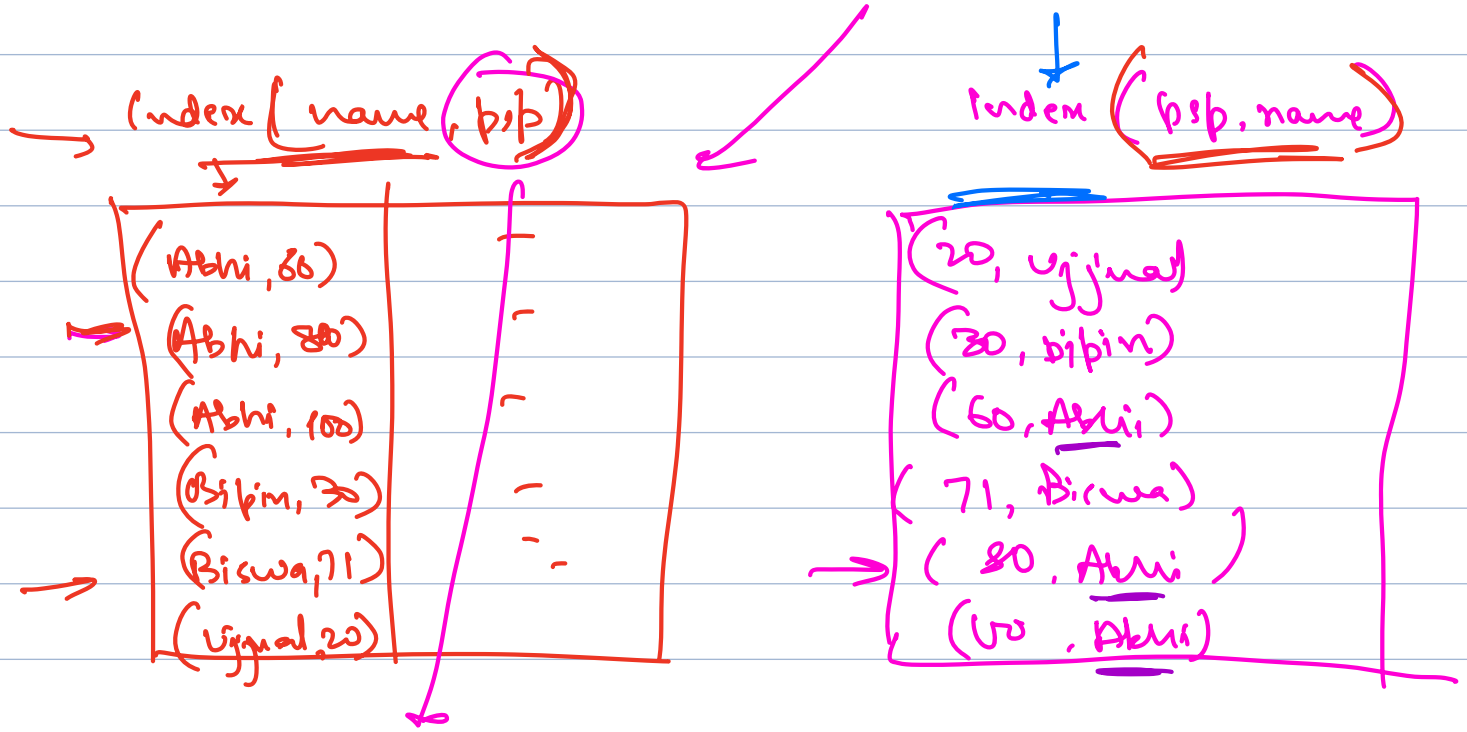
if we create an index on (name, psp)

≠

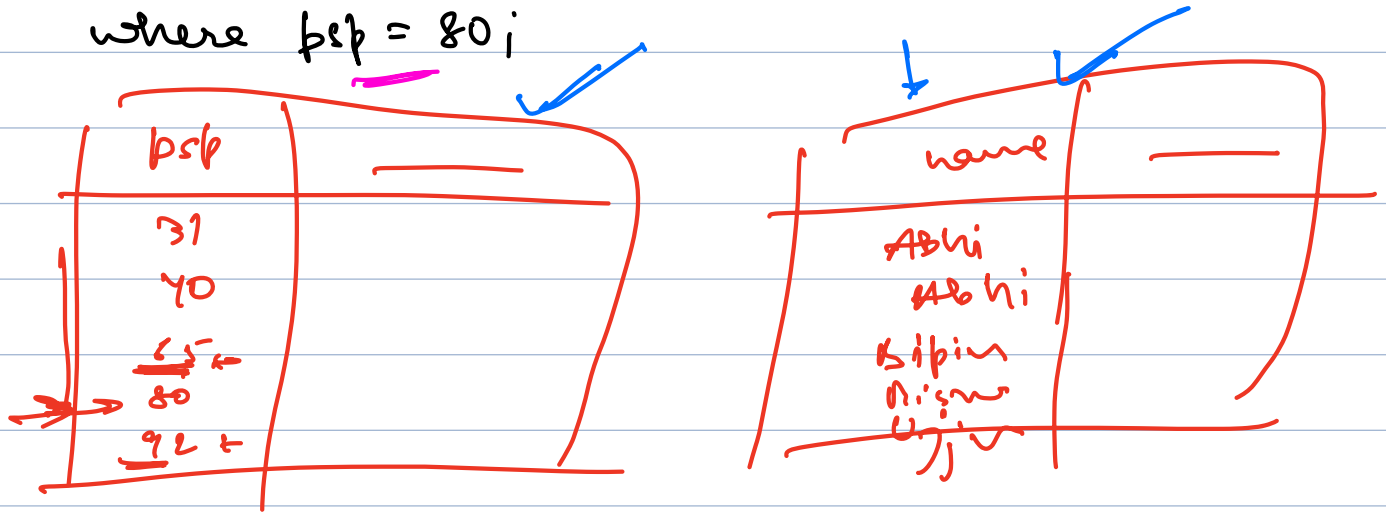
creating index on (name)

+

creating index on (psp)



```
select *
from students
where psp = 80;
```



Scenario:

Query	Index on	Help	Not Help
(1) name	psp		✓
(2) name	(name) (psp)	✓	
(3) name	(psp, name)		✓
(4) name	(name, psp)	✓	
(5) name = x and psp = y	(psp, name)	✓	

* If a query is on col (x)
any index with x in the prefix will help.

(x, a, b, c) ✓

(x, a) ✓

(x, b) ✓

(x) ✓

(a, x) ✗

Scenario

x = a and y = b

Index

"(x) (y)" ✓

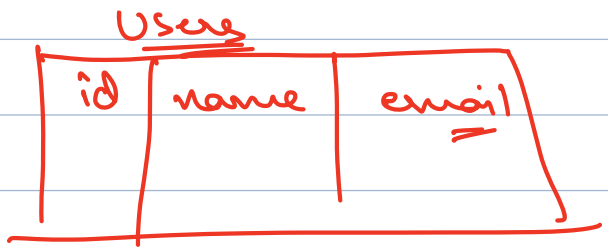
(x, y) —

x = a OR y = b (x)

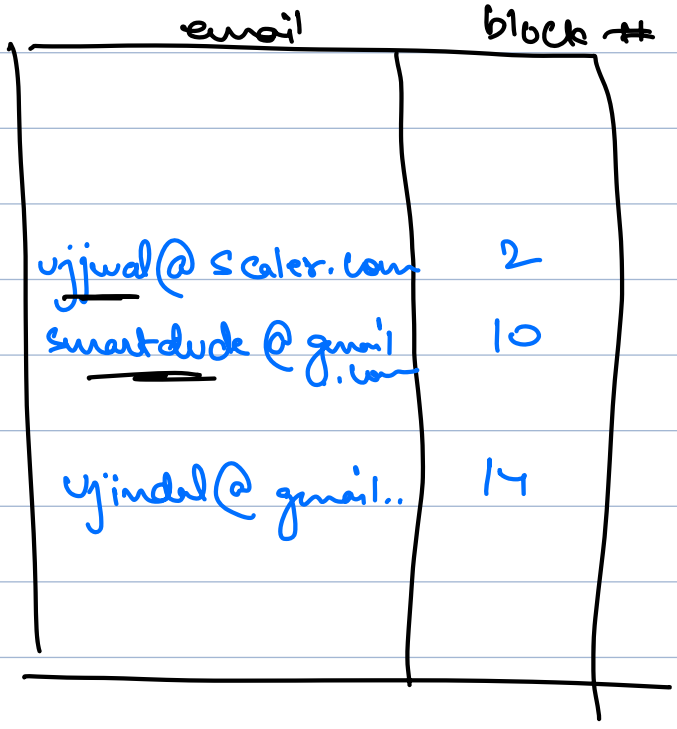
(x, y) ✗

AND query multiple colun index will help
OR query multiple colum index WONT help.

Indexes on Strings.



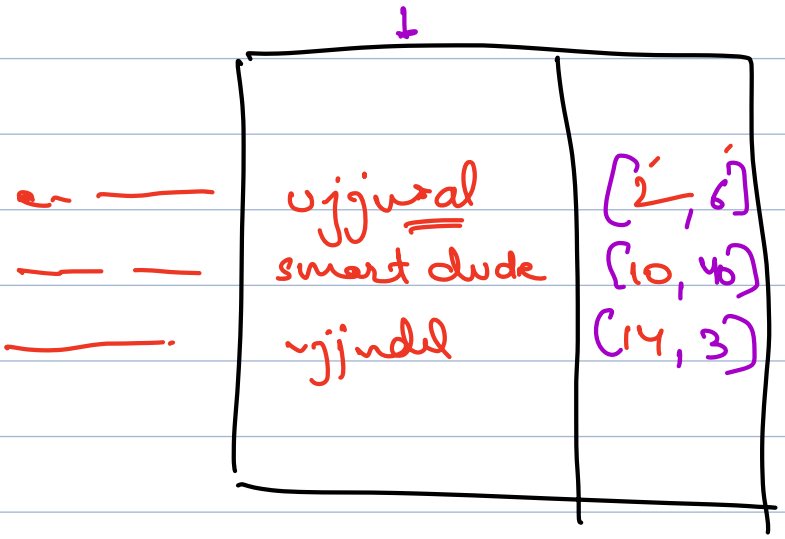

```
select *
from users
where email = ' ';
```



① Size of the index will be huge.
② String matching will be slow.

How do we solve this?

Instead of creating an index on complete email, we create index only on part before '@'.



→ Size of the index is reduced.

① space is saved.

```
select *
  from users
where email = 'ujjwal@scaler.com' ;
```

⇒ will have to fetch 4 blocks instead of 1 block earlier.

[without index → all block
with complete index → 1
with partial index → 4]

Typically when you have string columns, index is created on a prefix of the column instead of entire column.

↓

Enough mostly.

Address → index address[:r];

Google → 2×10^9

index on how many characters	# users	# distinct keys in index	disk access
1	2×10^9	26	$2 \times 10^9 / 26$
2	2×10^9	26×26	$2 \times 10^9 / 26 \times 26$
6	2×10^9	26^6	$2 \times 10^9 / 26^6$

2 select *
from users
where address like '%toronto%';

↓

An index will not help here.

Full Text Index.