Agenda

- Introduction to Indexing → What?
- How Indexes Work → How?
- Indexes and Range Queries
  - → Data structures used for Indexing (Extra material to read)
- Cons of Indexing
- Indexes on multiple columns
- Indexing on strings
- How to create an index

→ Views not done yet

Quick Introduction

- → Graduated in 2019 from Thapar University
- → Samsung Research in Noida
- → Razorpay (Backend Engineer) (Bangalore)
- → Booking.com (Amsterdam) (SDE 2)

Introduction to Indexing

$10^9 * 10^9 = 10^{18}$

```
for i .. 100:
    for j .. 100:
        doSomething();
```

} Joins $O(N^2)$

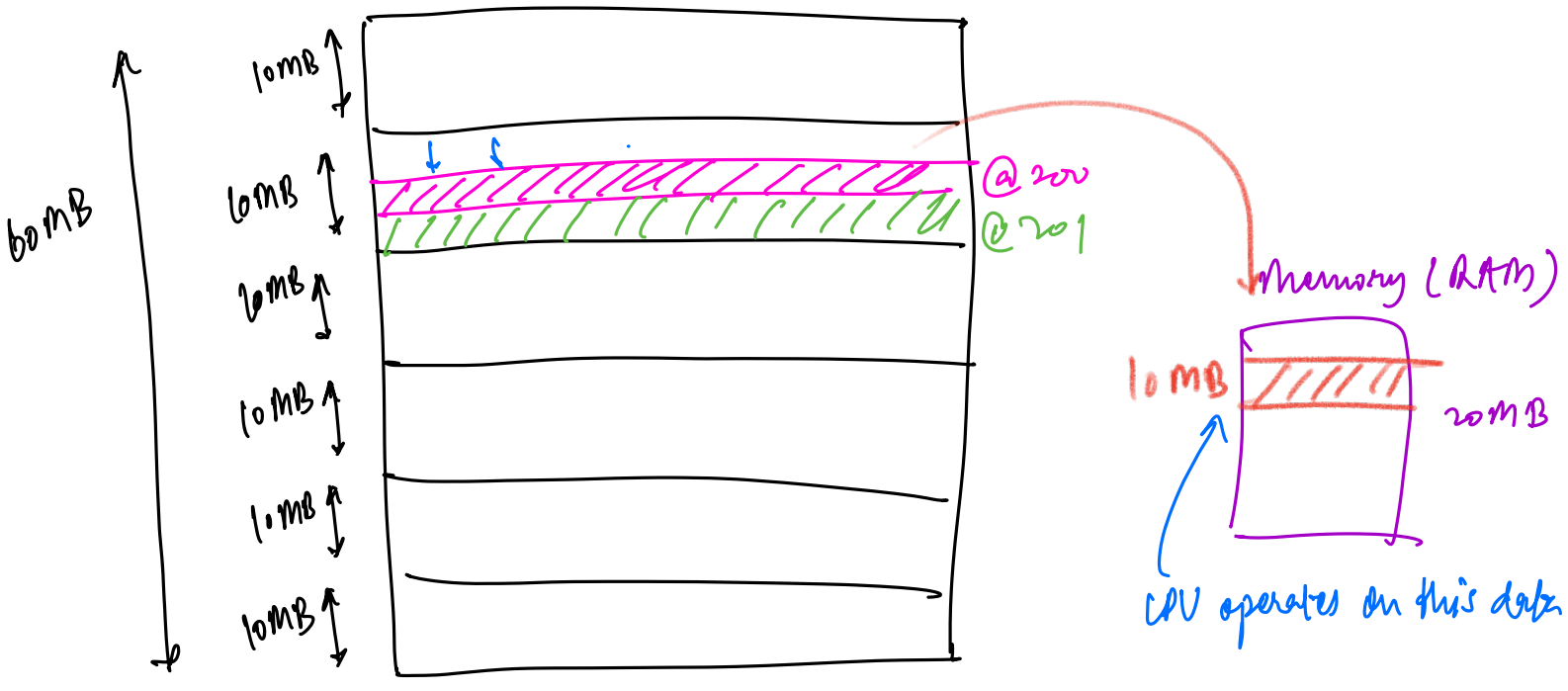
→ Performance will be very very slow

① We need to do some sort of optimization to make our queries fast.

② DB stores data on disk. (slow) V/S Program ⇒ RAM (very fast)

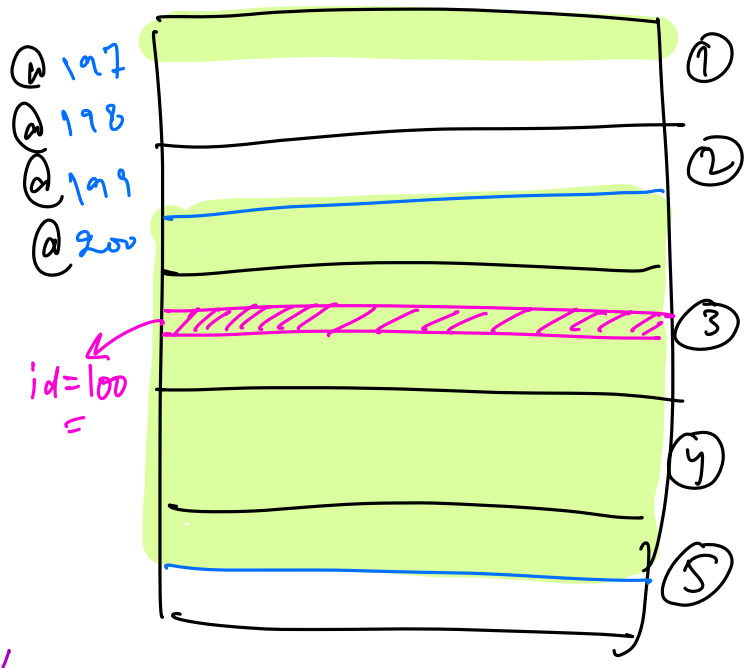
Cache (4MB)

HDD is 80x slower than RAM (memory)



Imagine a table with 100 M rows.

select * from students where id = 100;

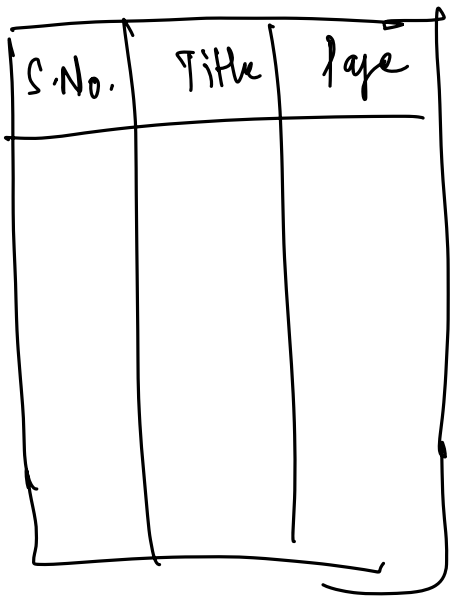


① Bring blocks ~~2, 3, 4, 5~~ 3 to memory

② check all the rows for student_id = 100.

→ Unnecessary blocks that are fetched, are a waste of performance.

Index



index in book ⇒ find a topic/page faster

index in DB ⇒ find blocks of disk faster
↳ blocks that contain the rows that I want to fetch.

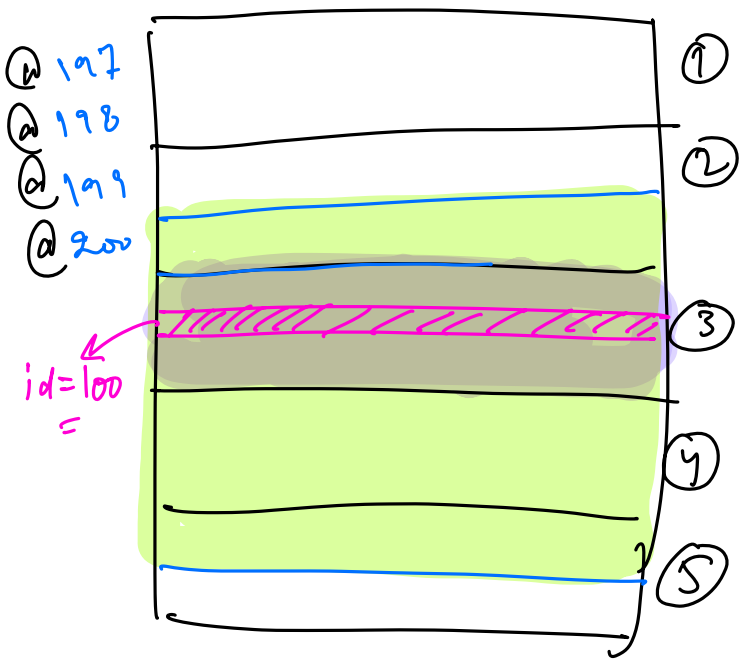
Purpose of indexes ⇒ Reduce # of disk block accesses to fetch data.

Huge table with 100 M rows.

Select * from students where id = 100;

Index

id (key)	Block Address (value)
1	1
2	5
3	3
4	4
5	2
.	.
100	3
.	.
100 M	1



HashMap map.get(100) → O(1)

w/o hashmap → 4 blocks (100M students)
with hashmap → 1 block (10000 students)

Select * from students where name = "Alok";

key: name	value: Block address List
Alok	[1,4]
Abhi	[1]
Pallavi	[2,4]
Rohit	[3]
Arpit	[3]

Students

id	name	psp	
1	Alok	80	→ Block1
2	Abhi	90	
8	Pallavi	100	Block2
7	Ritvtosh	99	
5	Rohit	89	Block3.
6	Arpit	90	
7	Vitul	99	
8	Alok	98	Block4
9	Pallavi	89	

Without index ⇒ 4 blocks
With index ⇒ 2 blocks

HashMap<String, List<Integer>>
map = new HashMap<>();

GOAL

1. You want to clear interviews.
2. You want to become a better engineer (Do better at your current job).

---

find all students with psp between 60 and 70.

index

psp	blocks
40	1
31	2
49	1
67	4
82	1
82	4
40	2
39	3
30.2	4

Students

id	name	psp	
1	Alok	80	→ Block1
2	Akhi	90	
8	Pallavi	70	Block2
7	Ritutosh	99	
5	Rohit	89	
6	Arpit	90	Block3.
7	Vitul	99	
8	Alok	98	Block4
9	Pallavi	89	

≥ 20 and ≤ 40

for i = 20 ⇒ 40:
  if in my map:
    map.get(i)

20.1
20.39
30.41

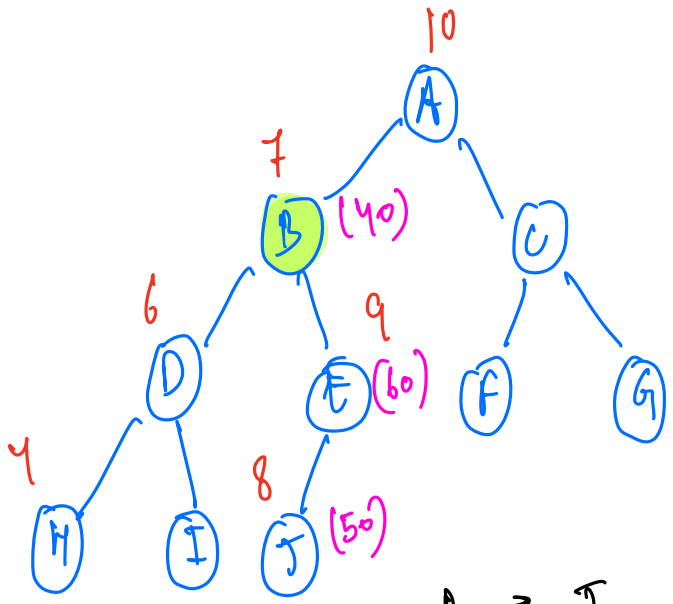
HM :- check a value ⇒ O(1)
get all values in a range [L, R] ⇒ O(N)

① Sorted
↗
Key ⇒ values.
(Sorted)

TreeMap ⇒ (BBST)
order-map
(Balanced Binary Search Tree)

T.C to check a value ⇒ O(log(N))

Ques



B → J

Given a node, get the next bigger node.

O(log N)

40 to 60

① Go to 40
② Keep going to next node until you go > 60.

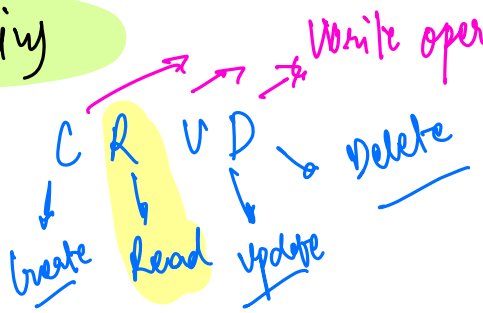
ptr	block
10	1
20	2
30	10
→ 40	9 →
→ 50	3 —
→ 60	6
70	

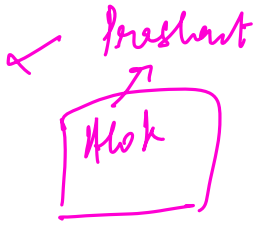
Internally indexes use → B/B+ trees. (optional)

① How indexes work
② What is the benefit of indexes
③ Why Hashmap is not good.

Break → 10 min

Cons of indexing





(1) Writes will be slower.

(2) Storage requirements would increase

→ Don't create indexes prematurely

→

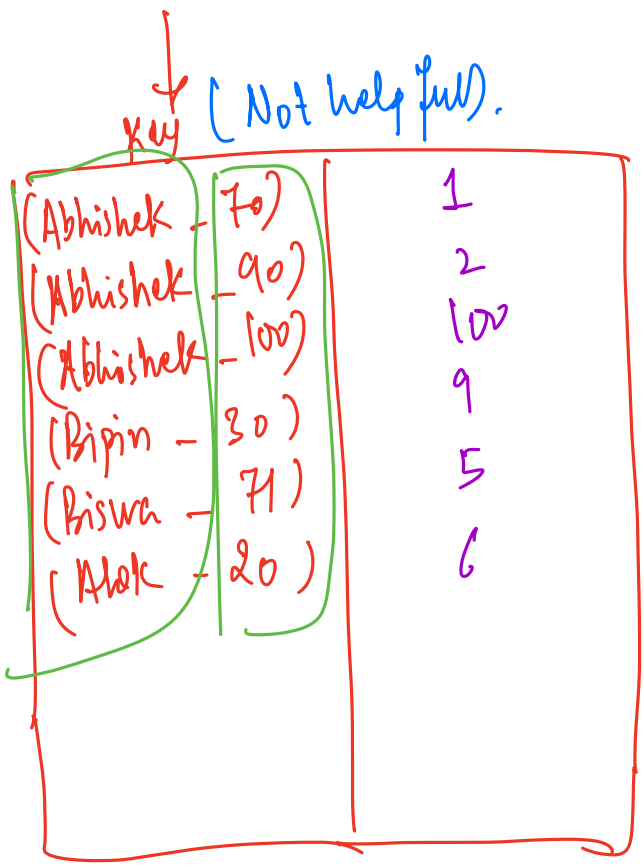
Indexing on Multiple Columns.

(id)
(name)
(name, psp)
(name, psp, rank) } →

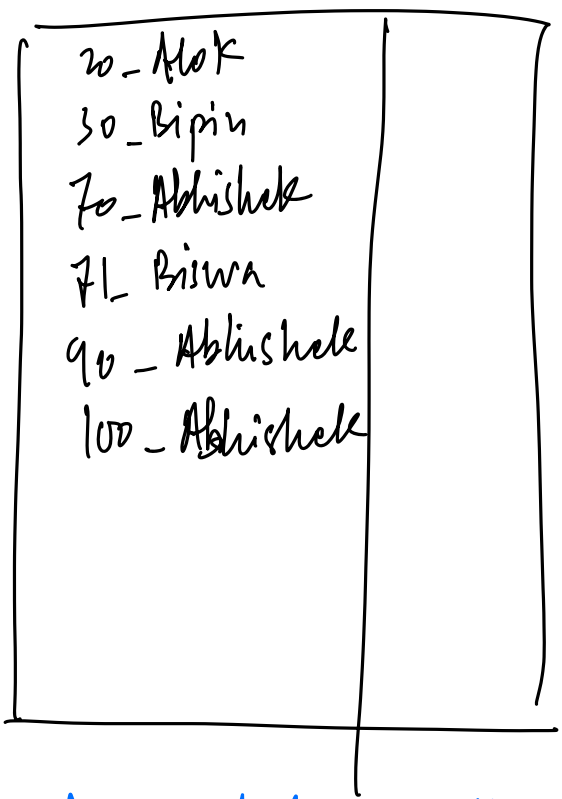
(name, psp)

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

creating an index (name, psp) ≠ create an index on name and create an index on psp



psp-name (✓)



If you have an index on (name, psp), it won't help with a query on just psp.

Scenarios

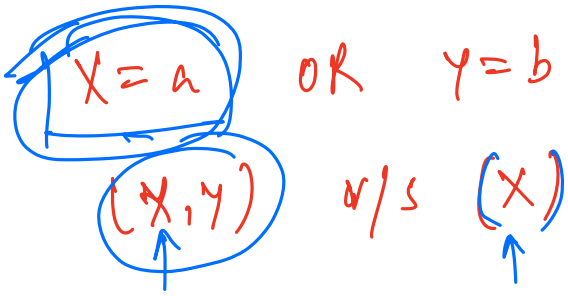
query on	index on	help	not help
① name	psp		✗
② name	(name) (psp)	✓	
③ name	(psp, name)		✗
④ name	(name, psp)	✓	
⑤ name = X and psp = Y	(psp, name)	✓	
psp = Y and name = X			
⑥ name = X or psp = Y	(psp, name)	✓	

psp = Y or name = X

* If a query is on column – X
  an index with X as prefix will help.
  (X, a, b, c)
  (X, a, b)
  (X)
  (a, X) → X

* x = a and y = b
  y = b and x = a

(X) (Y) →
(X, Y) → Better
(Y, X) → same.


X = a
OR
Y = b
(X, Y)
v/s
(X)