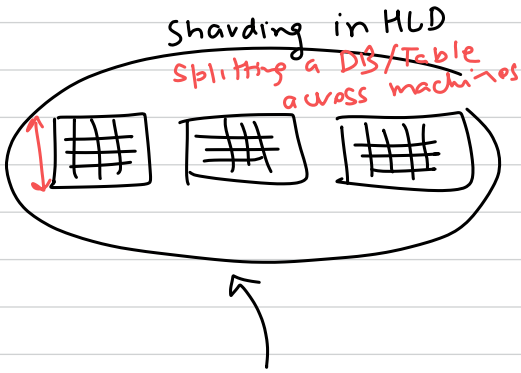
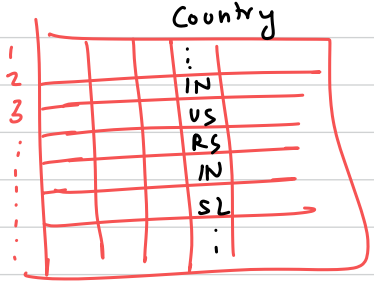
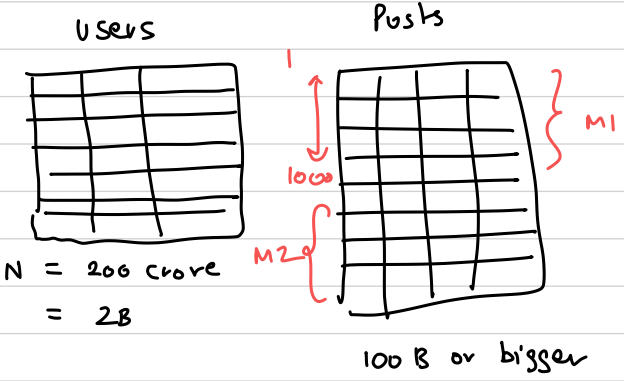
Indexing

- Introduction
- How they work
- Disadv.
  - Multiple cols.
  - Indexing on strings
  - Code





⇒ 10^{10} Records (Rows)



→ Find all users that belong to "IN".

→ O(N) work

→ Linear Search.

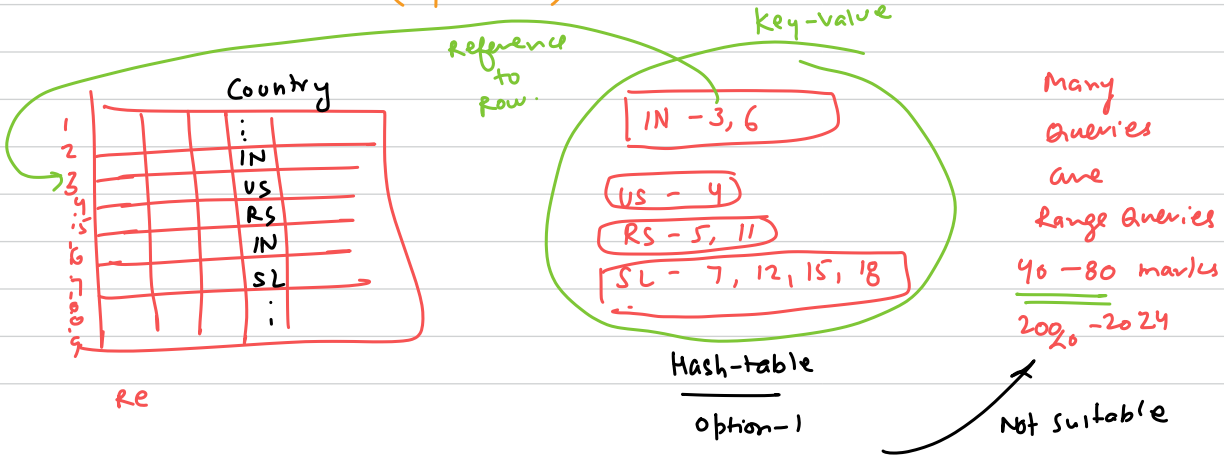
→ Time : 10^8 inst ⇒ 1s

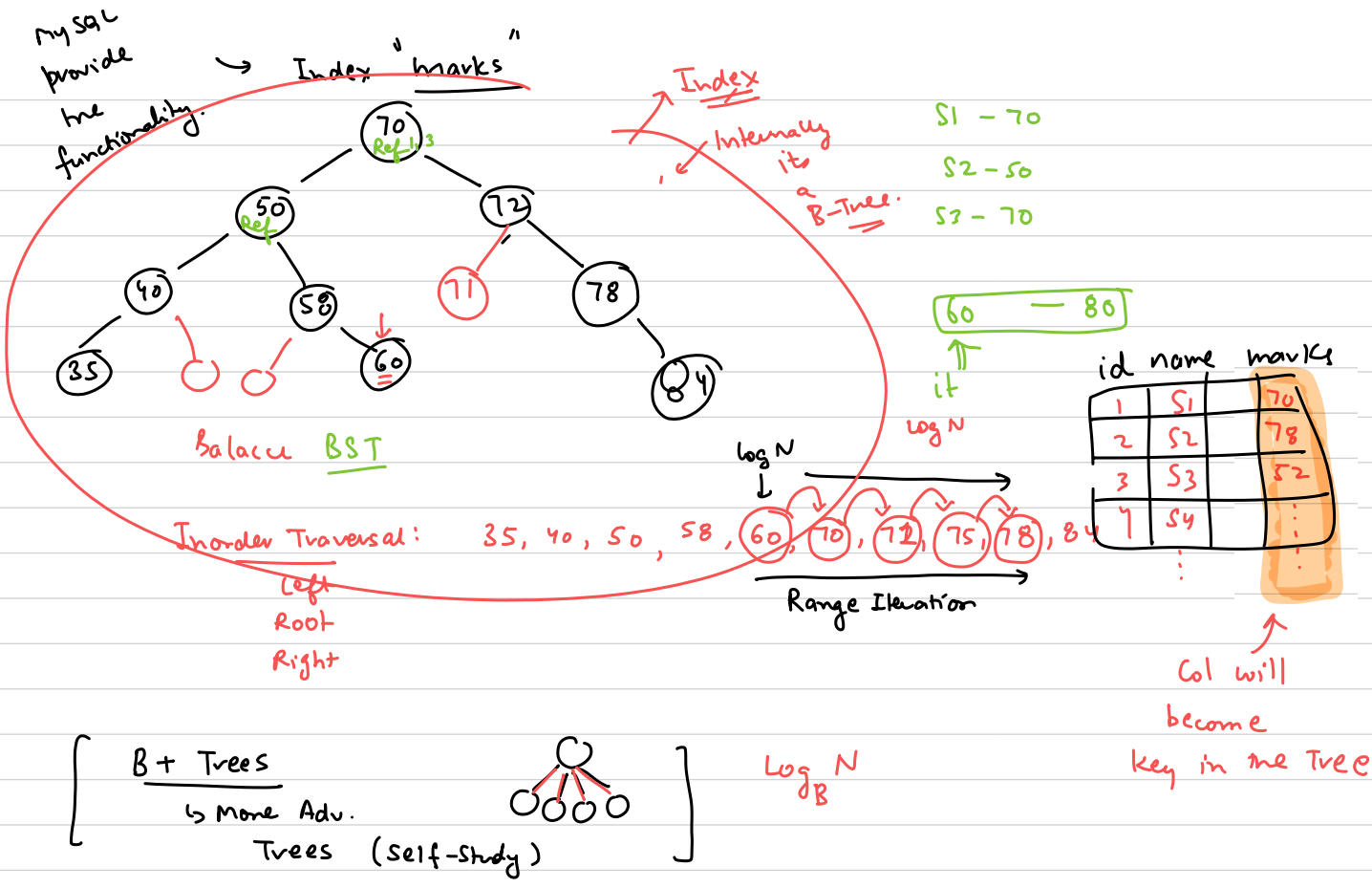
10^{10} inst ⇒ $\frac{1}{10^8} \times 10^{10}$ = 100s (Quite slow & bad :( )

B+ Tree

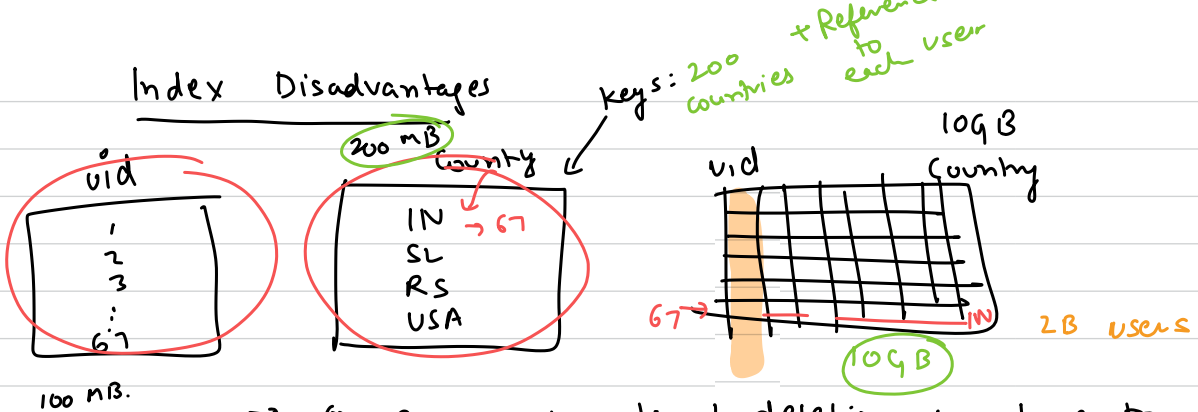
Index : Data structure that the Database uses to quickly fetch data.

With every table, you can create one or more indexes. (optional)




MySQL provide the functionality.
Index "marks"
70
50
72
40
58
71
78
35
60
84
Balacu BST
Index
Internally it a B-Tree.
S1 - 70
S2 - 50
S3 - 70
60 — 80
it
log N
id name marks
1 S1 70
2 S2 78
3 S3 52
4 S4
Inorder Traversal: 35, 40, 50, 58, 60, 70, 72, 75, 78, 84
Left
Root
Right
log N
Range Iteration
Col will become key in the Tree
B+ Trees
is More Adv. Trees (self-study)
Log_B N

Index Disadvantages



→ On every insertion / deletion you have to update all indexes for that ~~#~~ table

→ More Memory : Hard-disk for permanent storage
Load them into "RAM" while executing queries Searching in RAM is several (20x) faster than disk. (Heap Memory)

Lesser memory compared to Table.

When should we create?

→ ~~TABLE CREATE~~

→ only done for "Performance Critical Queries"

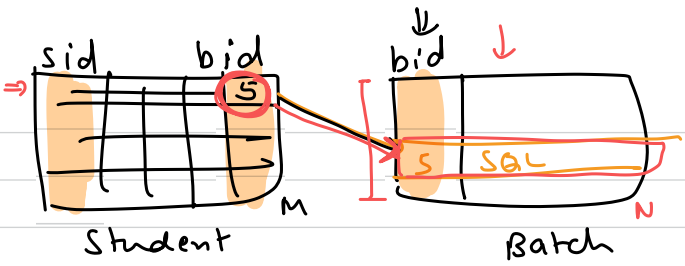
Query
↓
Columns
↓
"marks"

you decide on which col or cols I should create index or multiple indexes for the given table.

Default Beh:

- Index is created for PK
- FK for all tables

Additional indexes can be created based upon requirement.



Index
↳ s_id

Index
↳ b_id

used in JOIN by default

JOINS become to slow if indexing is not utilised