Indexing

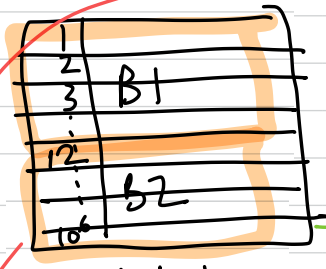
Agenda:

- Intro to Indexing
- How indexes work?
- Disadvantages
- Indexing on multiple cols
- Indexing strings
- CODE

Every seconds → 1000's GB was uploaded to Google Servers.

Assumption: one machine contains all the data.

Database



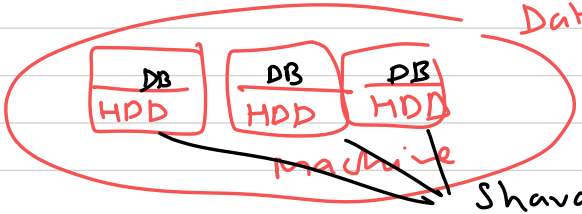
students Table (1MB ---------- 1000's TB)

HDD/SSD (500 GB)

Query: find marks of student with id 12.

google / Facebook

Data centers

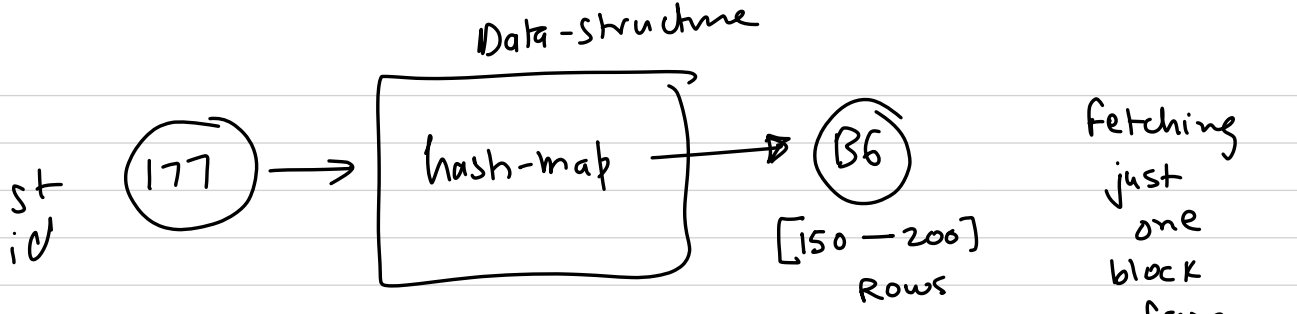


(HLD) Sharding

Block by Block.

↓

Minimize the total reads from disk.

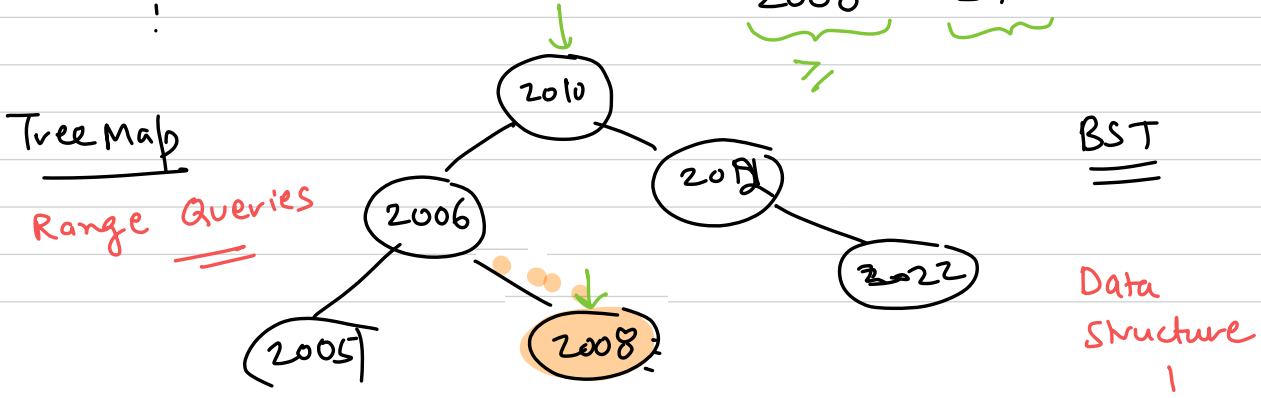


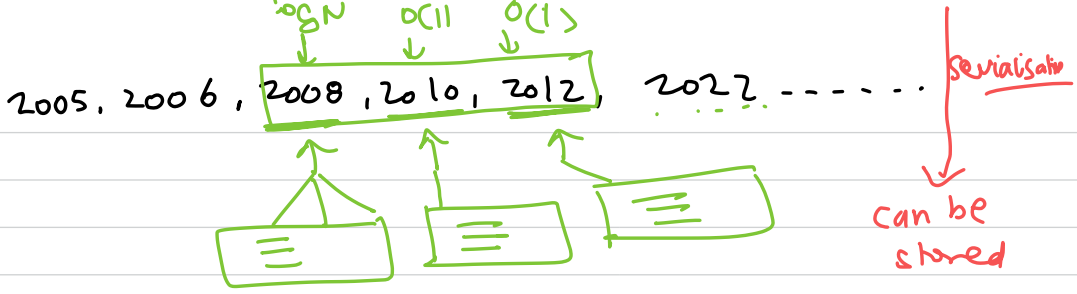
1 → B1
2 → B1
3 → B1
⋮
51 → B2
52 → B2
⋮

Range queries

↳ Films released in year

2008 — 2012

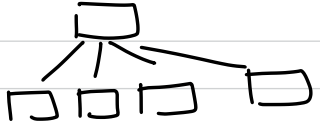




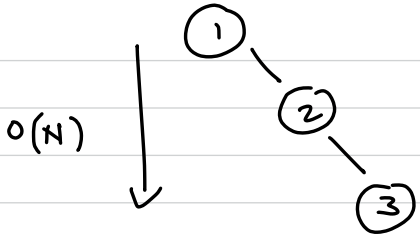
→ Convert DS into a "Index File" and store it on a disk.

→ when ever the index is needed, the file is loaded into memory and look up can be done using the DS.

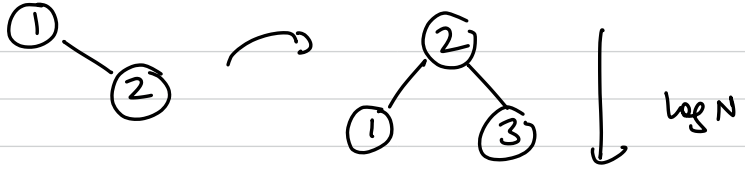
⇒ B+ Tree
- n-ary tree
- Height balanced.
- self-balancing tree



[Optional Read]

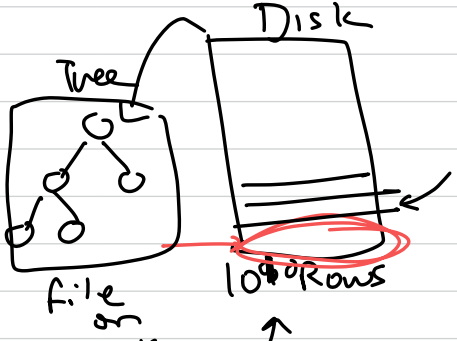


Skewed



Self-balancing Tree.

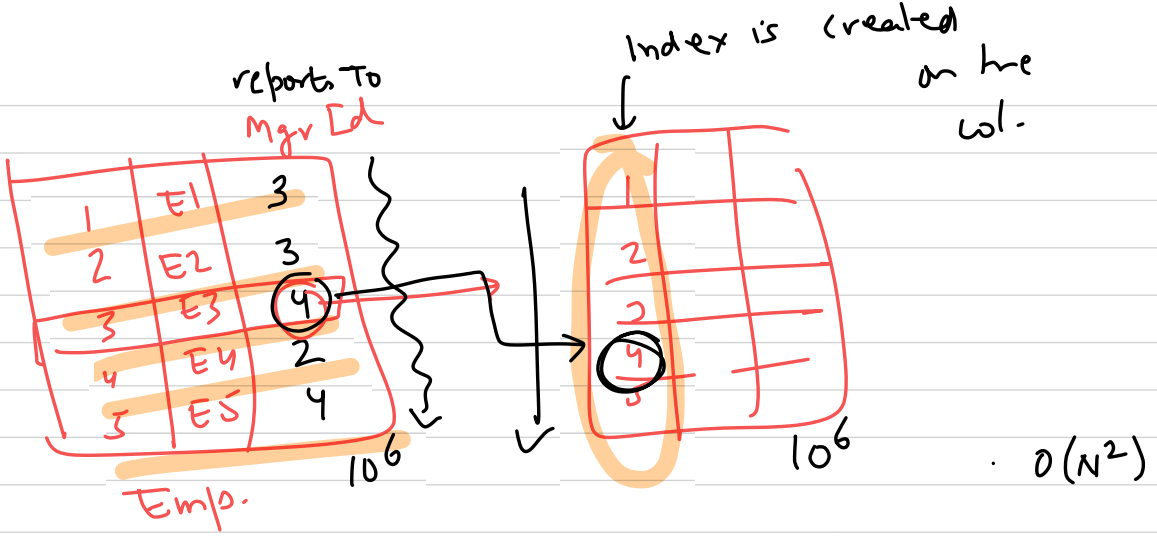
- Advantage:
  - ↳ To make queries faster



- Disadvantge:
  - ↳ Writes will be slower
  - ↳ Extra Storage.

10^8 ins → 1s

10^{10} ins → 100 s



$O(N^2)$

EXPLAIN ANALYSE

↳ JOIN

↳ Table Scan

↳ Index lookup.

$O(N)$

```
explain analyze
select * from table where col=name;

DROP INDEX name ON table;

CREATE INDEX name on table(colname);
```

Indexes on multiple cols.

Index → (name, batch_id)

↳ Tree will be sorted acc to name first, then acc to batch_id.

help in

⇒ name
⇒ name, marks
⇒ name, batch_id

Index → (batch_id, name);

marks of Naman from B2.

B+ Tree

Indexing on Strings. (Boost performance)

Email id's

create an index on prefix (5 letters)

10^6