18 Feb 2025 - DBMS-II - Week 07

→ TAs will be assigned for groups.
  ↳ will make a report of our project

→ Deadline - ~5 days before 3-4 segment end.

Recap: (here) block = page

1. reliability — multiple copies, ECC
2. performance — bit-level-stripping

} strike a balance using RAID configurations
↓ every server needs to be configured

RAID 2, 3, 4 } → not in use

Choice of RAID level

→ Assumption: separate file for each table

→ each row is a record

Fixed length records

→ Assume that record size is fixed (no arrays, lists, etc.)

→ Deletion of record

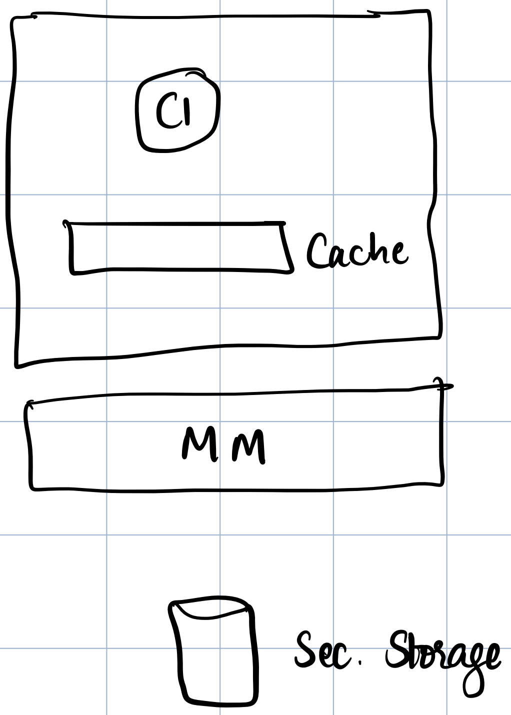
→ shift records, or

→ empty the record

→ scan every time new record needs to be inserted, or

→ maintain a header of free list.

→ More complex implementation for variable length records.



→ von neumann architecture: for any operation all data needs to be moved to core.

→ select * from instructor
where name like 'Kim'

All pages need to be moved to main memory

→ select * from instructor
where ID == 32343

2 pages need to be moved (paste book image

→ other architectures : computation in storage devices, AI accelerators

→ Data communication between disk and main memory.

→ How to minimize no. of accesses?

→ Organization of records

* Heap file organisation : no order

Advantage : insertion is easy

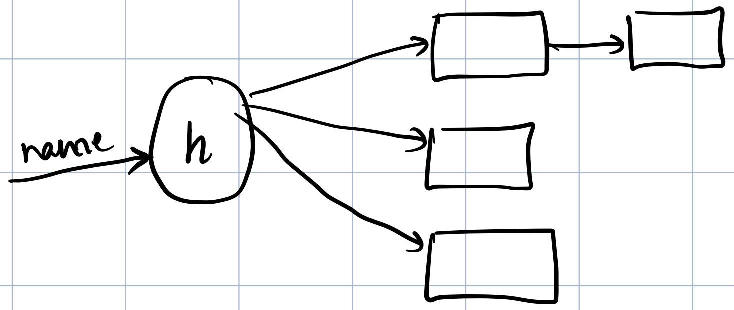
Disadvantage : searching

* Sequential file organization

Advantage : searching   Disadvantage : insertion & deletion require reorganisation

* Hashing file organisation

→ multiple buckets



→ Advantage: searching ⇝ can tell you exactly which page you should go
almost

Sequential File organization

→ disadvantage in insertion: to shift n pages:
load n pages in memory, compute and shift
store n pages back   O(n)

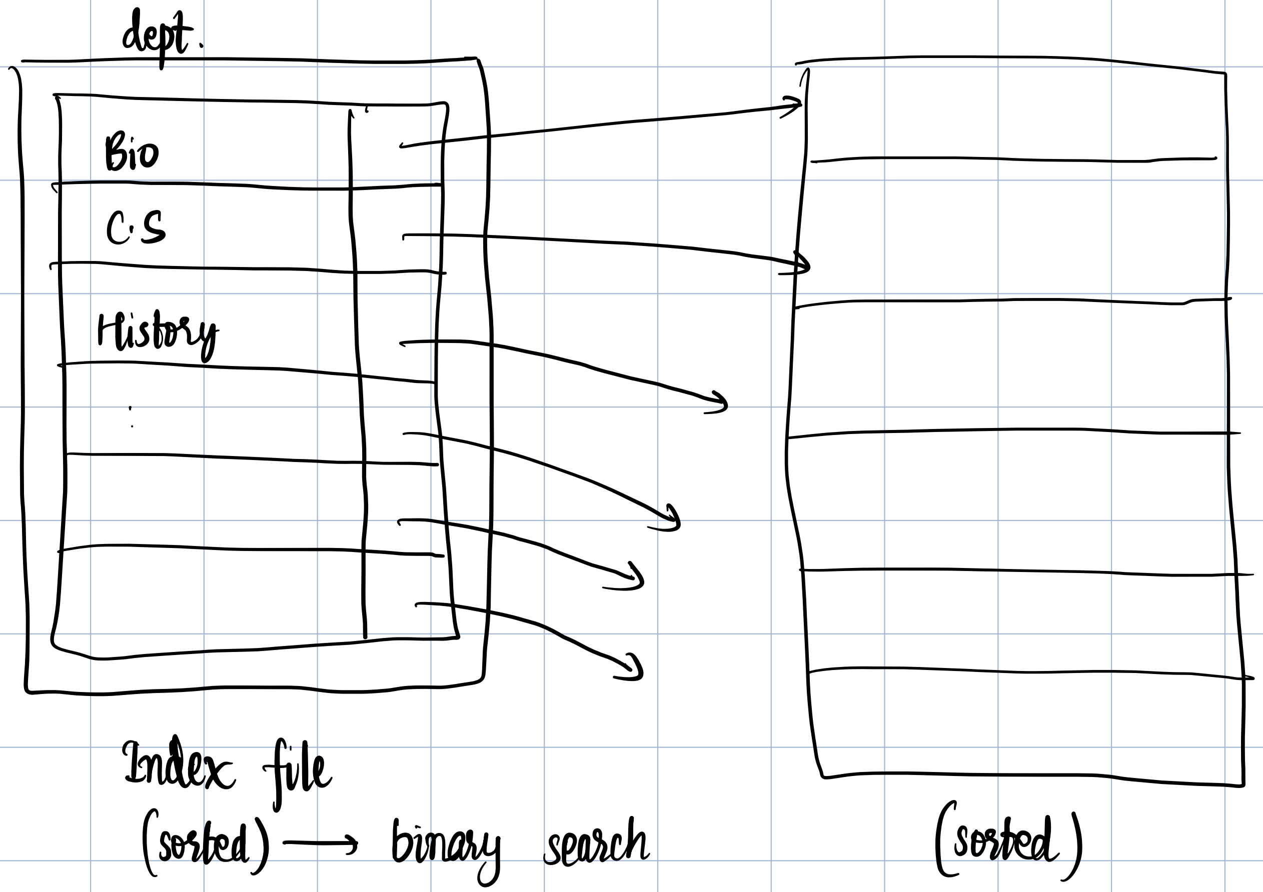
→ alternative: periodic reorganisation when server becomes free.

→ select * from instructor where ID == 83821 } → Binary search $\underline{O(\log n)}$ ↓ still too much

→ select * from instructor where dept like 'History' }

Indexing

→ Maintain indexing in your project.



- → You will need to access lesser no. of pages.
- → Different indexes based on columns : for other queries.

→ Two kind of indices

1. Ordered indices

2. Hash indices

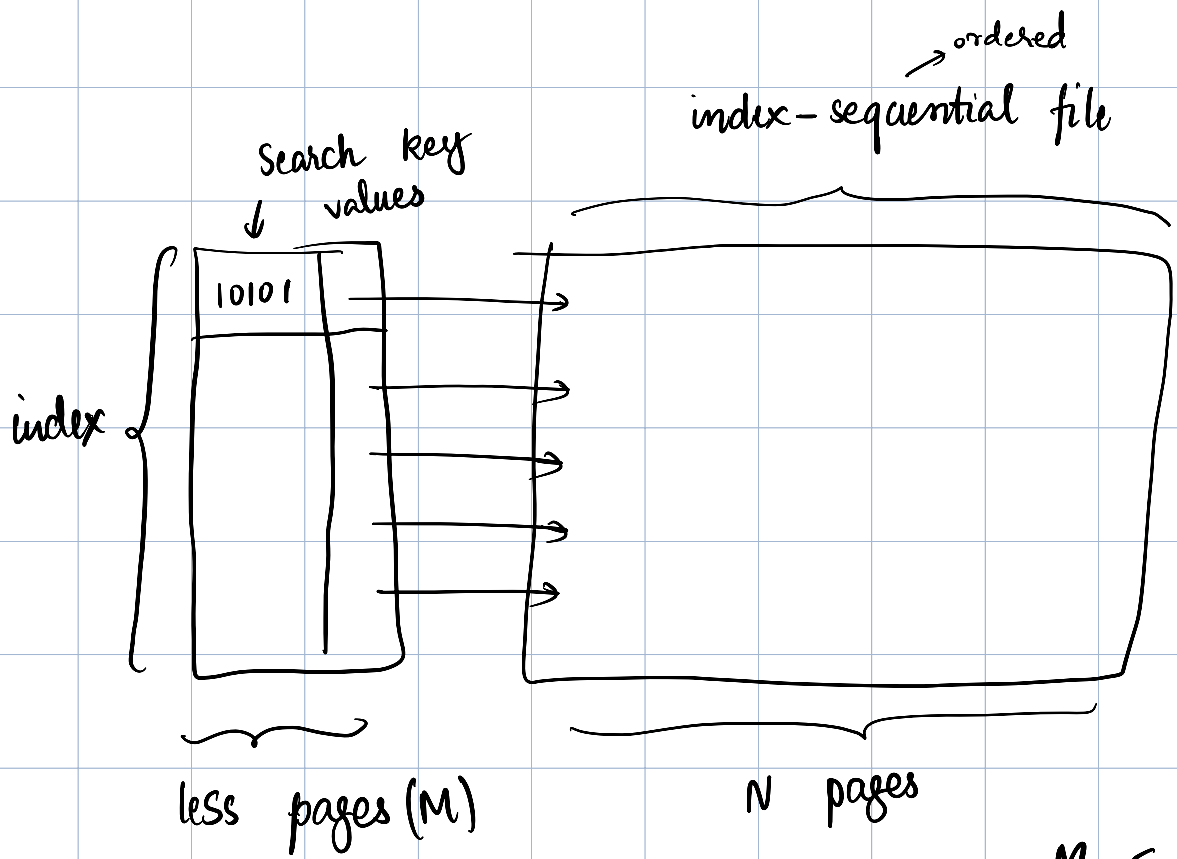
→ Ordered indices

- built on a non- condidate key.
- ordered sequential indices

→ Ordered indices : type

1. Clustering indices (clustered indices / primary indices) → e.g file sorted on dept. and index on dept.
2. Non-clustering indices (non clusteried) → index on salary

19 Feb 2025



$M < N$

→ Apply a binary search on index.

→ cluster ⟶ can be on ID department, etc

If file is sorted on clustered indices ⟶ ?? primary index (not related to primary key)

All instructors where salary $\boxed{= x}$

→ You can create an index, it is called <u>secondary</u>, because you cannot form a cluster (<u>non-clustered index</u>) ($\because$ the file is not sorted on salary.)

each index, array of pointers

→ Primary <u>index</u> is better.
↓
not created automatically (over head)

also insertion or delete
↓
index needs to be updated

the M pages are extra storage.

→ You need to find out which type of query is being commonly matching.

→ if frequently accessed } → create an index (primary/sec)

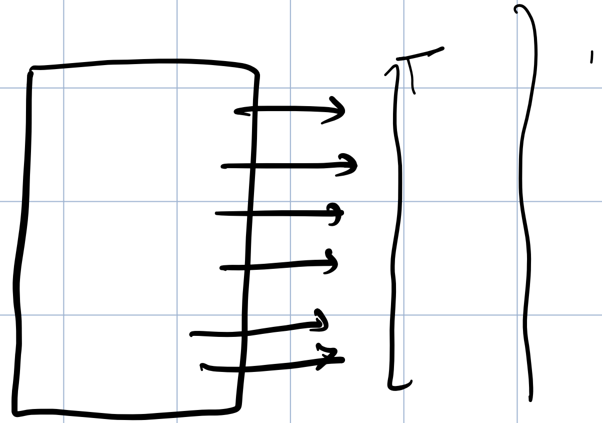
Select instruction where $\boxed{\text{salary} > x}$

↓

range - based query

→ Need different indexing.

Dense index:



o diagram add from slides

→ Performance imp ⇒ dense (?)

if not sparse

Multilevel Indexing

→ index can be very large (1,000,000 tuples example)

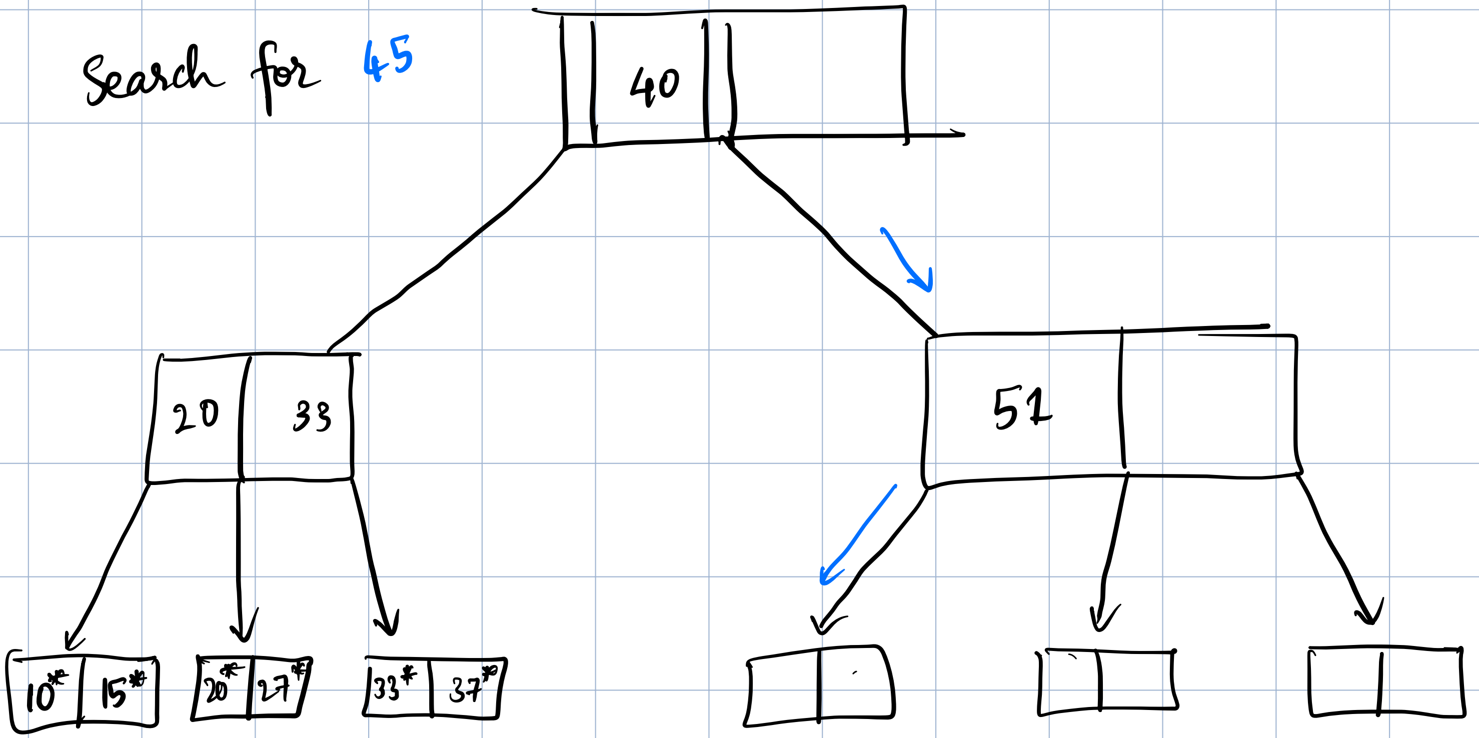
→ primary and secondary index diagram

→ Secondary index can only be dense.

Index update

Tree structured indexing

Search for 45



Search ✓

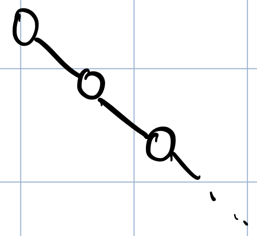
Insertion ⟶ tree does not remain balanced → B^+ trees

BSTs

⇝ nodes in left subtree < node
right >

→ average → log n

→ worst case : n (linked list)



AVL trees

→ Rotation

→ double rotation

Data structures ⇝ remember properties
(not for this course, but in general)

Quiz : upto AVL trees.

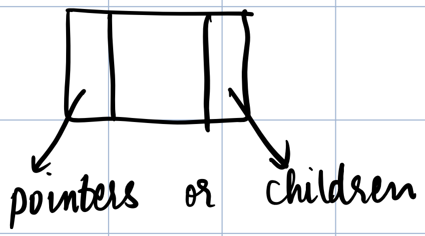
21 Feb 2025

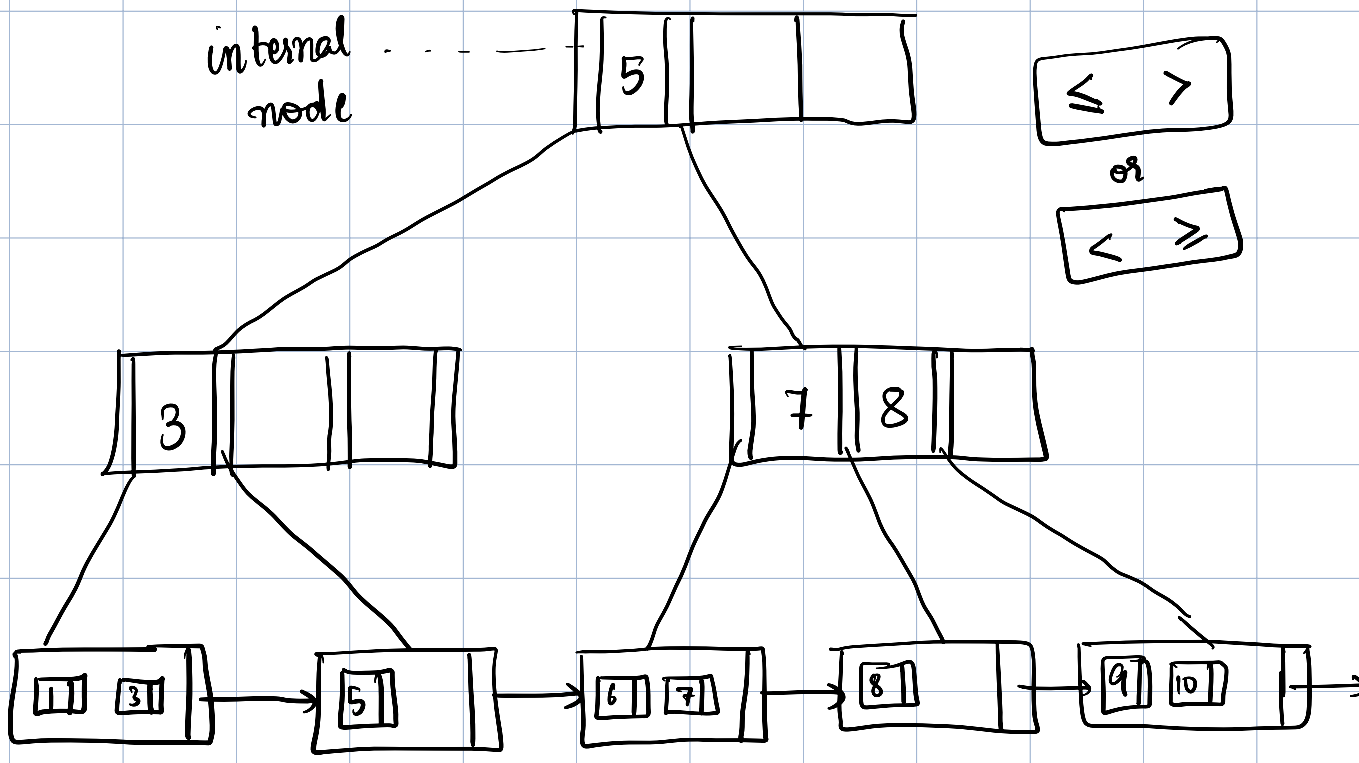
→ B^+ trees are good for range selection

→ Previous methods of indexing are not good, but can be used.

→ B^+ trees

- → leaf nodes – original key values
- → other nodes – support keys
  - – help you efficiently find keys





$N = 4$

maximum no. of pointers

$N - 1 = 3 \} \rightarrow$ values per node

$\underbrace{\text{Min} = \lfloor N/2 \rfloor}_{\text{other conditions, continue reading}}$

$\boxed{1 \,|}$ → points to the corresponding page numbers

Data entries (leaf)

→ B^+ file $\boxed{1 \,|}$ → record (k^*) → ⟨k, rid-list⟩

→ ⟨k, rid⟩ → best (record id)

Note : table ≡ corresponding file for the table (in the secondary storage)

All entries in a leaf node must be ordered.

→ B-trees : duplicates not allowed ??

→ Overhead for B^+ trees is less compared to other indexing methods (??)

→ Clustered and non-clustered indexing ⇝ similar (slides diagrams)

A B^+ tree is a rooted tree satisfying the following conditions:
(slides)

→ Non-leaf nodes in a B^+ tree

$P_1, P_2, \ldots, P_n$ → sorted

P_1	K_1	P_2	K_2	P_3	. . .	K_{n-1}	P_n

→ K_i's are also sorted.

→ Leaf nodes

→ Queries on B^+ - tree (slides)

→ Why are leaf nodes linked?

- ↪ Suppose you want to query all instructors from names 'G ... ' to 'J ...'

  Follow pointers after one search.

→ Only insert in the leaf node → then adjust.

→ How to build B^+ trees → not req. in this course.

Quiz 3