15/03

File → fixed length record
unspanned, bfr
↳ Block 0, 1, 2, ..., bfr − 1

ith record:
Block & offset
⌊i/bfr⌋ ↖ i mod bfr.

$\lfloor 71/50 \rfloor \Rightarrow 1$
$71 \% 50 = 21$

File
↳ unordered/heap
↳ ordered

Storing records doesn't follow any ordering (no ordering on data) (Stored in the order of insertion) (APPEND)

ordered based on certain field (ordering field)
↳ ordered on the roll

Opn
- Retrieval — Does not modify (Set of records satisfying condition) — Simple | complex
- add/del/update — Change the content

Student-result → Static → unlikely to be changed
item-stock → dynamic → can "

ordered → ordering on key
→ ordering on nonkey

→ Binary search on ordered key.
→ Roll >= 50 & Roll <= 100
↳ Search 50 by bin search
↳ then sequential.

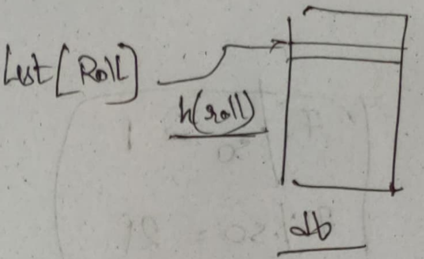
update ordered val
↳ delete ~~followed by ins~~ followed by insert.

Unordered file

ADV: Insertion
Retrieval → sequential
(worst case → read all)
(O(n) → n = no of blocks)
[no of accesses = n]
average = n/2

Deletion → seq search
↳ reuse is problem. } → ∴ logical deletion
⇓
reorganisation required. (difficult)

Internal hashing → in primary memory
⇓
table / array



Hash key
⇓
on which hash function is applied
↪ index to table

Disadv: only go for equality of field not >, < etc.

Collision
↪ linear probing : problem is deletion (what if the record, which collided, is deleted)
↪ chaining is better : easy deletion / insertion / search.

External hashing
Disk
↪ no of blocks

hash(hash key field)
⇓
Bucket no.
(Collection of no of blocks)

→ collision prob is less
⇓
As the bucket can hold number of record
⇓
$\underbrace{\text{No of hash collision} <= \text{Bucket capacity}}$
⇓
No prob.

Indexing : To make the search with the file faster.
↪ overhead : index file
↪ Dense index : For every ~~dat~~ data rec, there is an entry record in index file
↪ Sparse index ~~reco~~
↪ else sparse. (index rec < data rec)

Index
→ ↪ ordered file (ordered on index ~~field)~~ field)
→ ↪ mostly fixed length record.
→ ↪ 2 fields : ⟨index field value, Record/block pointer⟩

insertion in index file is problem.

Data file

ordered ~~unordered~~ unordered

(Datafile not ordered on indx field)

Indexfield / orderingf
- key (primary index)
- nonkey (cluster index)

Sparse keys

secondary index
- → key
- → nonkey

Dense sometimes (mostly)
· Sparse rarely