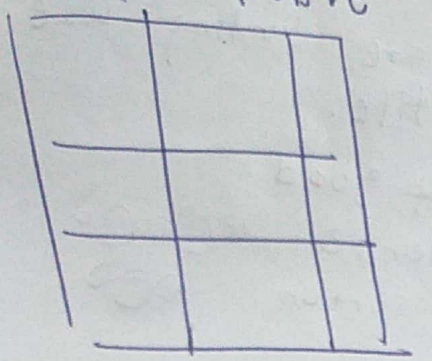
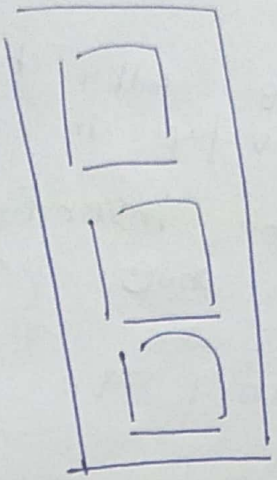
atabase table => DB file.

db Table



DB File



What operations do we have to do with this DB file (read all the table)

-> Search
-> Modify - Insert
update
Delete

① Scanning the table

② seek -> range the table, Eqn.

① Scan -> Assume your table has Columns Sno, Sname, age. for Scanning you want to

```
Select *
from student
```

} for that you have to read all the records of student table.

• Read all - Scan the file Top to bottom

② If you want

```
select *
from student
where sno = 1
```

} you don't need everything. You want to seek some record.

Seek -> range

-> Equality

To perform these operations fast how we should organize our data within this file.

① Heap file organization

There is no specific order of files => Heap

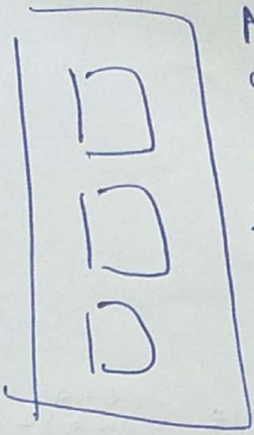
Insert ✓

Search ✗

modify ✗ delete ✗

If records in random order in heap files
What we can perform fast => . Insert

② **Sequential**



Assume all records are sorting use age value in this file.

✗ **for insertion** → not good.
Insert කරල ආයෙ order එ පිළිවෙලට හදන් ඕන. ඒ නිසා insertion වලට හොඳ නෑ.

✗ **update** → update කරන value එ ඕනෑකෙනෙක්ට ලේසි. ඒත් value එ update උනාම ආයෙ ඇවිත් record එ දාලා sorted order එ හදන් ඕන.

✗ **delete** → delete කරල ආයෙ පිළිවෙලට හදන් ඕන.

Searching is fast for updating & deleting but after that you want to move records here & there.

✓ **Search**
→ Scan ✓
→ Seak — range — sorted files are good for range selection.
When you want to find age between 20 - 40
— equality

Assume you want to search age 20 record
equality වලට හොයන්ට හොඳ hash.

③ **Hashed**

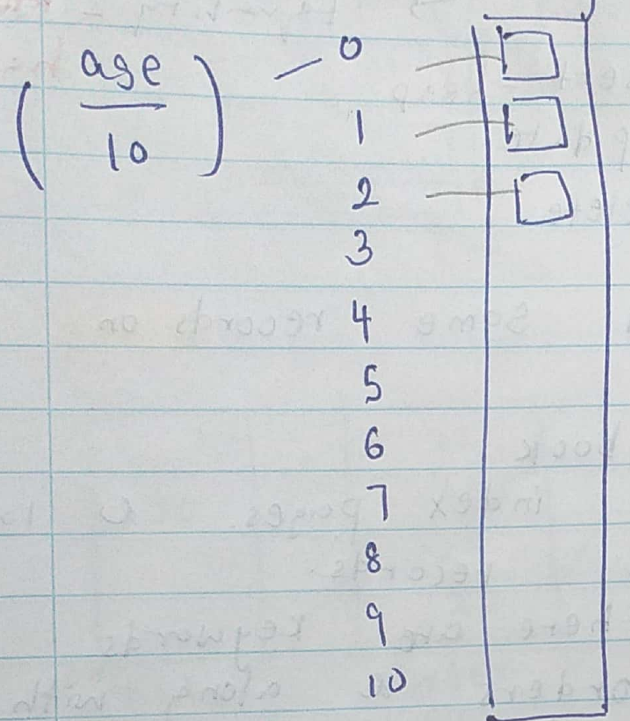
Here it is using some (hash) function

(fh) ← this is organizing records within the file.

1
2
3
4
5
6
7
8
9

Assume my hash function is $\left(\frac{age}{10}\right)$ remainder value

So this hash function tells if your remainder 0, you stay here. (store)



Age = 20 නම් hash function 1 අනුව $\left(\frac{20}{10}\right)$ remainder ලැබෙය 0 යි.

ඒ අනුව ඔයා 0ට අයිති ස්ථානයේ store කරනවා.

මෙ particular location එක ⟹ bucket

- File is a collection of buckets.

h (r) → bucket in which record r belongs.
hash (r) function එක් නම්, r කියන record එක අදාළ වන bucket (location) එක කියනවා.

Any age record will be directed to any bucket out of those 10 buckets. So If we have organized all data within this file, equality selection run faster.

Assume we have age = 35 record, apply to hash function, remainder 5 values in one bucket. So we have to read only 1 particular bucket from those 10 buckets. (Not reading entire file)

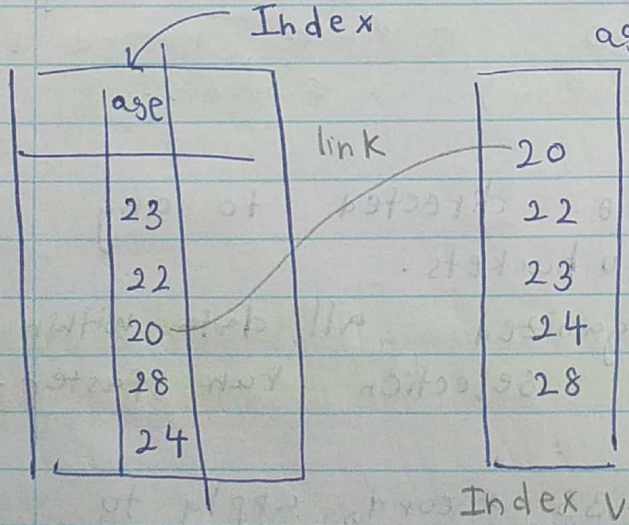
No: Date: __/__/____

Search → Scan → range - sequential
→ seek → equality - hashed

Modify → Insert - heap
update
delete

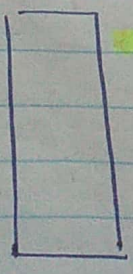
- Assume you have to find some records on a book.
① you can read whole book
② you can go to the index pages & locate to the relavant topic records.
- (In index pages there are keywords with alphabatical orders & along with that some page numbers. so you can straight locate to page number)

- Indexes in the database do the same -



③ different alternatives to available to link those index & table record

Alt 1 → Data record with key value (K)



you have only single file. No db file & index file (no 2 files). Both in same file.

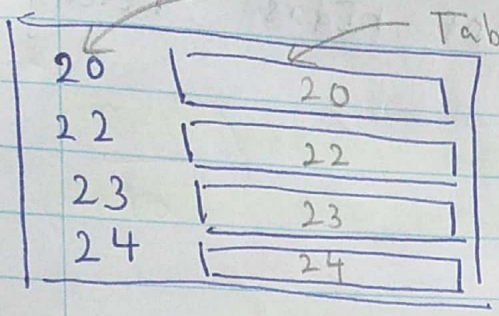
So then this single file you have all age values

Atlas

Date: __/__/____

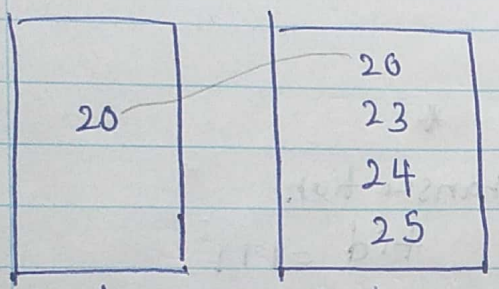
Alt 1

Index entry



Next to age Values record is there.

Alt 2



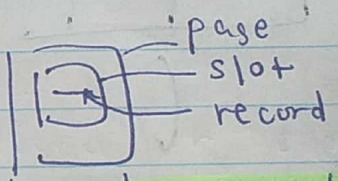
Data file

Index file

There are two files.
- data
- Index

data is in 2 different files. how to connect those index entries & table records.

In alt 2, they maintain Search key along with rId.

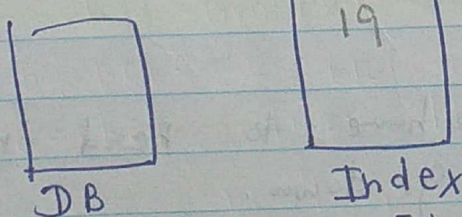


record Id = page # + slot #

Index file ல், Index = 20 அதை தரும் rId = 10. அப்போ, rId ல் இருந்து நமக்கு index ல் page number ம் slot number ம் தெரிந்தால் நாம் நினைத்த record ஐ எளிதாக பெறலாம்.

- If we know rId, we know What is the page & what is the slot.

Alt 3



problem is to handle duplicate data.

Asume this is a age of sliit student so there are plenty of records of age=19 in db record & index file.

(*) Good to use redundant data.

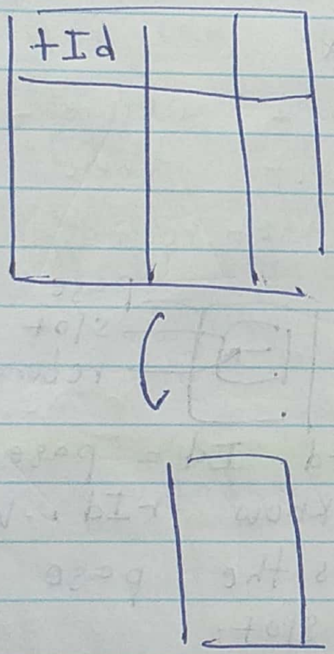
To handle that we (maintain) store 19 once in Index file & maintain list of rId's belong to that age = 19.

Reduce index file size

speed up

(*) Bank System

Transaction Table



Index file

Select *
from transaction
where tid = 111

To speed up this query, we add indexes

for a one minute this system can have million of transactions.

We are creating indexes for speed up searching But here get in information from index file that also a problem now due to huge number of records.

To find out one entry, you have to read millions of entries. Performance go down.

Index 1 ක් [illegible] record 1 [illegible]
record Id 1 [illegible] [illegible] record , [illegible] [illegible]

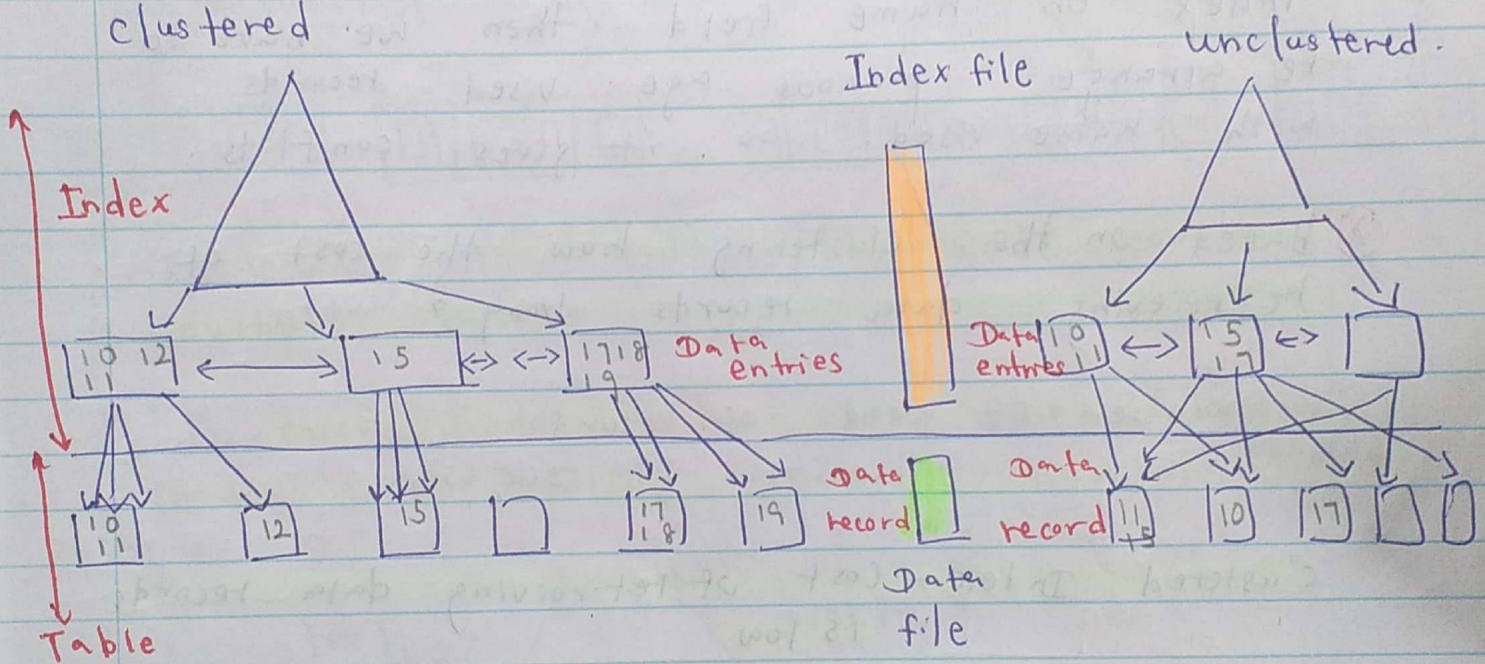
No: Date: / /

without reading millions of entries in index file how to find out particular record.

To speed up, we're developing different data structures.

⇓

Access methods → tree based indexses
hash based indexes.
(how to arrange data within this index file)



data entries => Data entries in index file are
When we create index you copy values from the table column into index file & organize records in the index file.
So all index entries in data entries area.

Data records - full record in the table

(*) If index is clustered, Index entry order & table record order is same sorted order

No: Date: __/__/____

Unclustered - Index entry order & table entry order is not same.

(*) Why we can't have 2 clustered index in table =>

Assume you create index on age column & data records are also sorted as ~~it~~ indexes

If you say you want to add another clustered index on name field, then we have to re arrange previous age vised records with name vised. So it gives conflits.

(*) Based on the clustering how the cost of retrrieving data records vary?

↓

number of reads of records
number of writes of records

Clustered Index - Cost of retreiving data records is low.

No:

Date: __/__/____

How to find out index entries

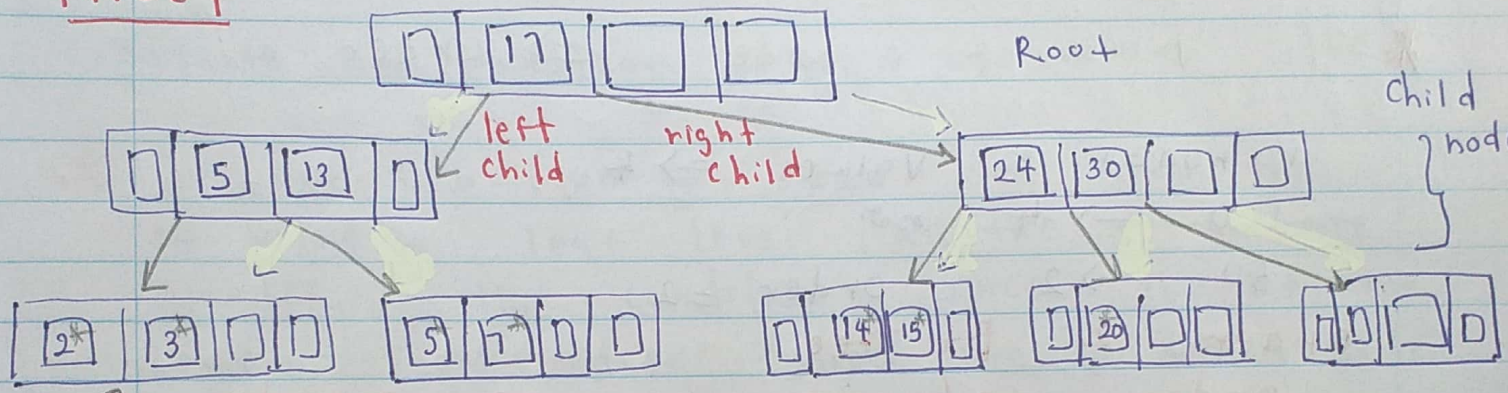
Tree Based Index → B+ Tree

Hash Based Index → E Hash

Tree Based Index → good for range selection

Hash index → Good for Equality selection

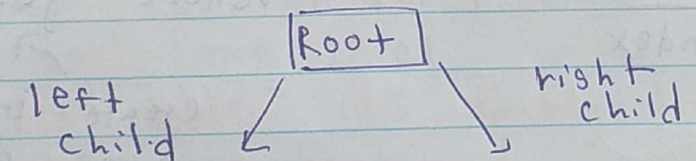
Tree



- Root & all nodes, Each node can have multiple Values

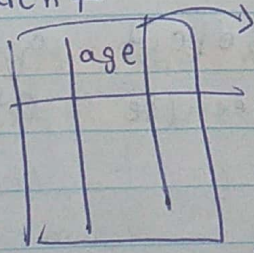
In every node it has left child & right child

In the leaf level all Values are in sorted orden



- less than
- greater than or equal

(*) student



Index is a B+ Tree index. If its a tree structure those age Values are copying to index file.

B+ Tree leaf level have all values took from table column to develop index.

only leaf level has Values came from table column

From leaf level we link Actos table records.