B tree Applications

09 October 2020 18:49

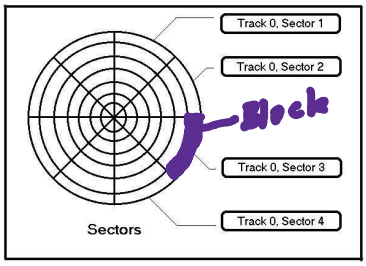
To understand B tree application in Database indexing, let us discuss the following file organization methods.

Three types of file organization

1) Sequential
   - Records are stored in sequential order.
   - Accessing Records starts from beginning and proceeds one by one till end of the record
   - After updation/deletion entire file is rewritten.
2) Direct access (Random access)
   - Records can be stored in any order. Need not be sequential.
   - Accessing a Record is done by knowing Starting position of file, Size of record, Relative Record number.
   - Hence no need to access from the start of file. Directly a record is accessed.
3) Indexed Sequential Access method. (ISAM).

- Both sequential & direct access.
- Data is stored in secondary device like Hard disk in terms of Blocks.

Consider the following diagram



- Block is the smallest unit of memory in which data is stored.
- It is the intersection of track and sector.
- Data stored in block is identified by block address. (ie) Track no, sector no, offset
- Typical block size is 512 bytes but it can vary based on overall size of hard disk.

Example:

Suppose we want to store employee records.

Each employee record is as follows:

Emp id - 10 bytes

Emp name - 50 "

Dept. - 10 "
Section - 8 "
Address - 50 "
128 "

Each Record takes 128 bytes
Suppose 100 Records are there.
"How many blocks needed?"

Block size = 512

∴ In one block = $\frac{512}{128} = 4$

4 Records/block can be stored.

Hence to store 100 Records,

Number of blocks = $\frac{100}{4} = 25$ blocks.

Consider sequential file organization.
To retrieve a record, maximum 25 block access (disk access) is needed.

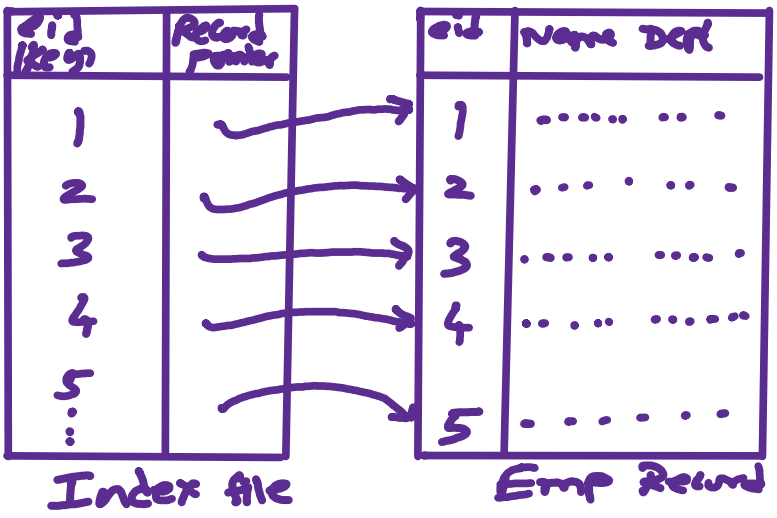
- A Program running in main memory give instruction to access a record stored in hard disk in which data is stored in terms of blocks.
- Hard disk is accessed

at the maximum 25 times
- Accessing hard disk is costlier in terms of CPU time.

- "How to reduce this?"

- Index file is used.

- Index file is created based on a key as follows.



- Each key in the index file has a record pointer, which points to a record in Emp. Record.

- The index file is also stored in hard disk in a block.

- "What is the Size of index file?"

- Index file consists of
  Emp id - 10 bytes
  Record Pointer - 6 bytes

16 bytes

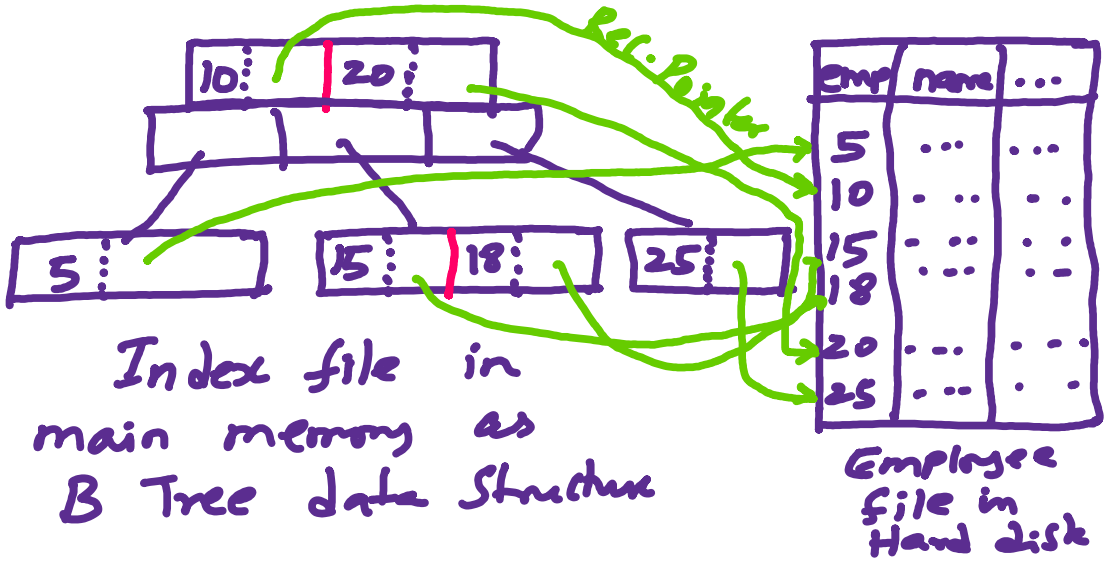
- Each record in index file takes 16 bytes.
- Number of index file records in a block $= \frac{512}{16} = 32$
- 32 records can be stored in a block
- To store 100 records of index file,
  $= \frac{100}{32} \simeq 4$ blocks.
- Hence 4 blocks are needed.
- Now, the program in main memory give instruction to read index file instead of employee file.
- To access a record,
  Maximum 4 blocks of index file has to read.
  From the index file, we know the block address of particular record.
  Hence one more block is read to retrieve data.
- Hence total no. of disk

access (no. of blocks) is 4+1 = 5

- In sequential it is 25, which is reduced to 5 using index file.

- Now in main memory, the index file searching is done using "B Tree" data structure.

- This is where "B Tree" finds application in database index. Consider the following B tree.



Index file in main memory as B Tree data structure

Employee file in Hard disk

- Index file stored in hard disk is now loaded in main memory as B-Tree data structure.

- Required Record is identified by searching employee key. From the employee key

the corresponding Record Pointer identifies the record in employee file.

- To search Index file B-tree data structure is used.
- The index file used here is one-one mapping. For each key there is a corresponding record. This is called <u>dense index</u>.
- When the index file also grows, we have to go for multi level indexing.

Multi level Indexing

Suppose, in the above example Consider index file grows upto 1000 records.

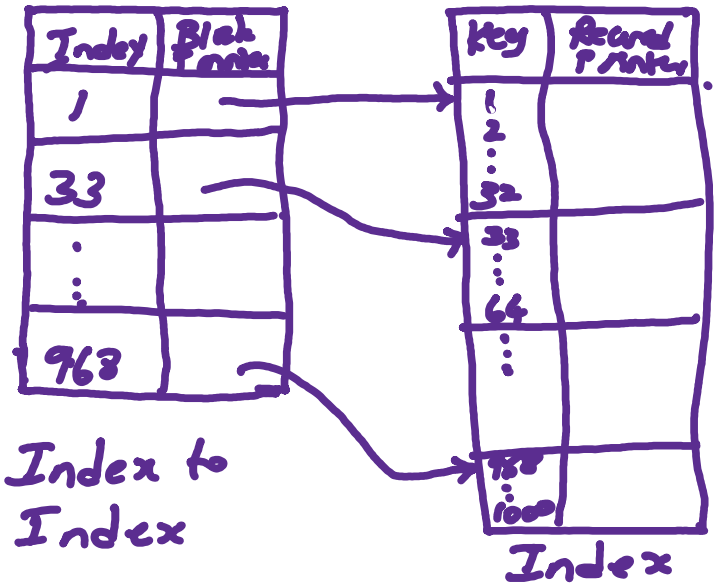
Now Number of records/block $\Big\} = \frac{512}{16} = 32$

For 1000 records Number of blocks $\Big\} = \frac{1000}{32}$

$\simeq 32$ blocks

Since the number of blocks increases, we can have

index to index file as follow.



Now the index to index contains starting address of each block of index.

Here index 1 points to block address of 1, from that we can search key sequentially within that block from 1 to 32.

- Now to store this index to index,

  In one block $\} = \frac{512}{16} = 32$ records

  $\therefore$ 32 Records of index to index can be stored in $\} = \frac{32}{32} = 1$ block

Hence Index to index - 1 block

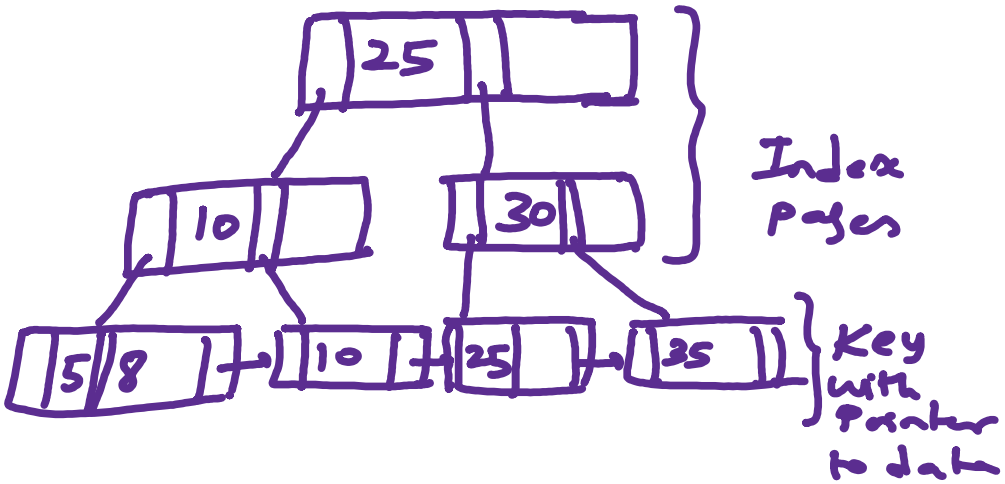
To retrieve a record,

1 disk access of index to index + 1 disk access of index + 1 disk access of Employee record.

= 3

∴ If we have index file only we need 32 disk access, but with the index to index it is reduced to 3 disk access.

- This multi level indexing is represented in main memory using B+ tree data structure
- The higher levels (non leaf) represents index pages and leaf level represents key values.
- Consider following B+ tree



- B+ data structure is used in multi level