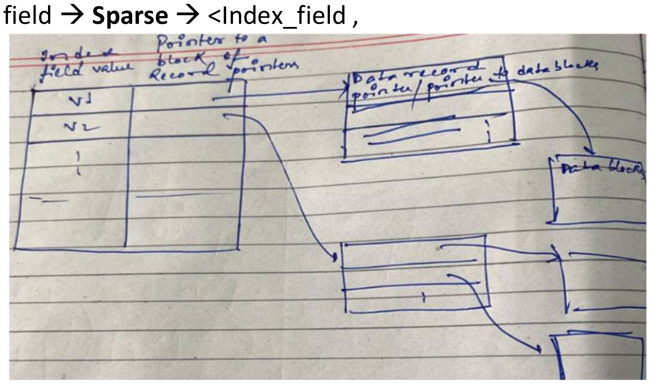
PRIMARY INDEX→

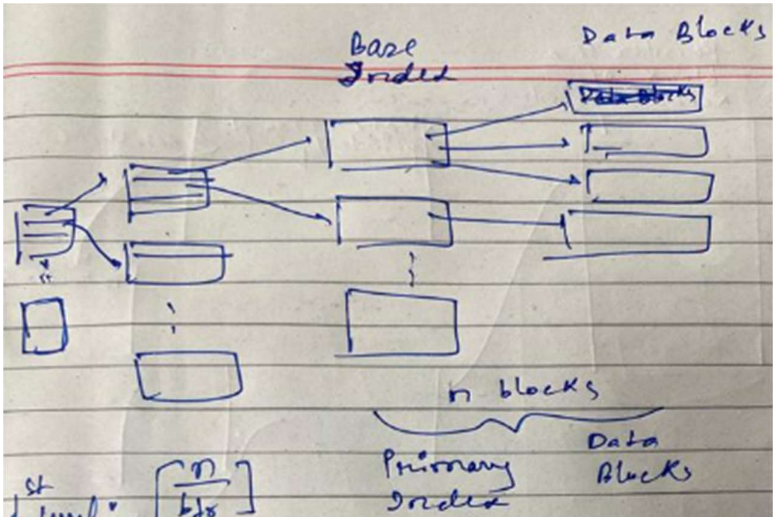
- **Primary Index** →
  - Data File is ordered on key
  - Ordering Key Field is the index field
  - Primary Index File is an ordered file of fixed length record. It is ordered on indexing field (same as ordering key)
  - Records are of the form ***<INDEX_FIELD , RECORD_OR_BLOCK_POINTER>***
  - [1 2 … bfr] [bfr+1 bfr+2 … 2bfr] … […] ***([…] is a block)***
  - **Block Anchor / Anchoring Record:** 1st Record of Data Block
  - In the index file , ***# Records = # Data Blocks (Corresponding to block anchor , there is an entity in index file)***
  - Primary Index is sparse → Size of Primary Index << Size of Data File
    - # Index Records << # Data records
    - Size of Index Record is smaller than that of Data Record (so , binary search on Index File needs much less # block accesses than on Data File)
  - Find the data for ordering or indexing key with value v_k → Suppose , index file is spread across blocks like → (1 , b1) , (101, b2) , (201 , b3) , … → Find such that Key(i) <= v_k < Key(i + 1) → Get corresponding address value → Here , 1 , 101 , … are the anchors (starting record)
  - eg :- **For Data File** → 40, 000 data records , Size of data record = 40B , Size of block = 1024B , bfr = floor(1024 / 40) = 20 → So , ceil(40,000 / 20) = 2500 data block → 12 = ceil($\log_2$ 2500) block accesses needed …
    For Primary Index → #Entries = #Data Blocks = 2500 , Size of index Record = 10B , bfr = floor(1024 / 10) = 100 , #Index Blocks = ceil(2500 / 100) = 25 → ceil($\log_2$ 25) + 1 = 6 block accesses are needed …
  - **Advantages:** Searching faster on less #block accesses , Range Query is easier (only look for min and max value) …
  - **Disadvantages:** When data records are inserted or deleted , need same in Primary Index File , insertion and deletion in sorted contiguous file is O(n) → If insertion / deletion of data records leads to change in block anchor

- **Cluster Index** →
  - Data file is ordered on a non-key field
  - Indexing is to be done on that ordering field (non-key)
  - Ordered File of fixed length record , order based on certain ordering of non-key field which is index field
  - Records are of the form ***<INDEX_FIELD_VALUE , RECORD_OR_BLOCK_POINTER>***
  - For each distinct value of ordering / indexing non-key field , there is an entry ***<DISTINCT_VALUE, BLOCK_IN_WHICH_IT_FIRST_APPEARS>***
  - eg :- [D1 D1 … D1] [D1 D1 … D1] [D1 D1 … D1 D2 … D2] …. → <D1 , b1> , <D2 , b3> , …
  - **Disadvantages:** Same as Primary Index (Less severe than Primary Key because of many occurrence of Di)

- **Secondary Index** →
  - Index Field (Can be Key or Non-Key) → Data File is not ordered on index field → Go for secondary index
  - Data File is not ordered on indexing field and it is a key → **Dense** → For each data record , there is an entry in index file , fixed length: <INDEX_FIELD , BLOCK_OR_RECORD_POINTER>
  - eg :- 50,000 Data Records , 40B per record , floor(1024 / 50) = 20 records per block , floor(50,000 / 20) = 2500 data blocks → Avg 2500 / 2 = 1250 block accesses [Normal in Cluster Index – No Binary Search]
    50,000 Index Records , 10B per Index record , floor(1024 / 10) = 100 records per block , 50,000 / 100 = 500 Index Blocks → ceil($\log_2$ 500) + 1 = 9 + 1 = 10 block access [Secondary Index]
  - Data File is not ordered on indexing field and it is non-key field → **Sparse** → <Index_field , Set_Of_Block_or_Record_Pointers> → For each distinct value of Index Field → **Variable Length Record** → Or, we can go for **Dense** → Fixed Length Record → <Index , Pointer>
  - **Sparse and Fixed Length Record** → Additional Level of indirection is introduced → 1st Level (Index Field Value (Ordered) maps to Pointer to a block of Record Pointer) → 2nd Level (Set of Pointer to Data Block) → 3rd Level (We get data block)



- **Multilevel Index** →
  - **Multi Level Of Indirection** → Suppose , 'n' blocks in Base Index , after 't' levels has n / (bfr ^ t) blocks → So , time needed = 't' Block Accesses after searching in n / (bfr ^ t) blocks
  - **Disadvantage:** We have to change so many ordered files during insertion and deletion



1st level: $\left\lceil \frac{n}{bfr} \right\rceil$

$\downarrow$

no of blocks

2nd level: $\left\lceil \frac{n}{(bfr)^2} \right\rceil$

at t^{th} level: $\left\lceil \frac{n}{(bfr)^t} \right\rceil$

Multilevel Indexing