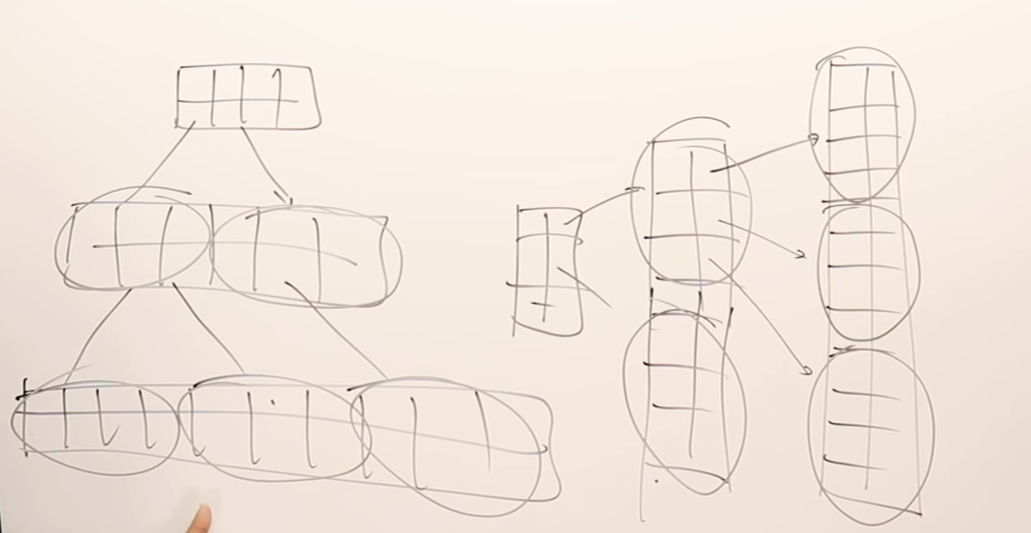
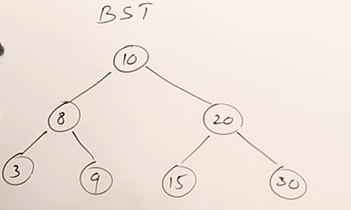
**B tree**

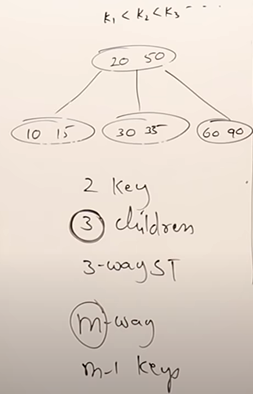
Employee table  


Total 🡪 128 bytes

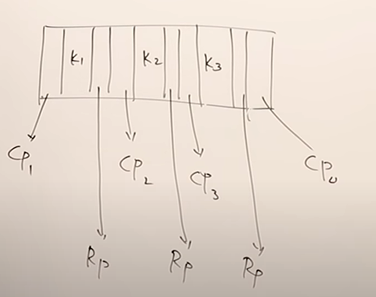
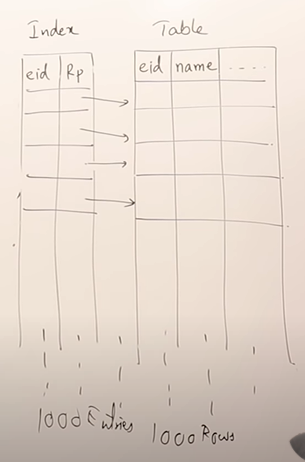
Size of 1 block 🡪 512 bytes

  
As the record size increases, multi-level indexing should increase and   
as the record size decreases, multi-level indexing should not occur.  
So self-manage multi level indexing is required.

**Binary search tree**  
  
1 key and 2 children 🡪 2 way search tree

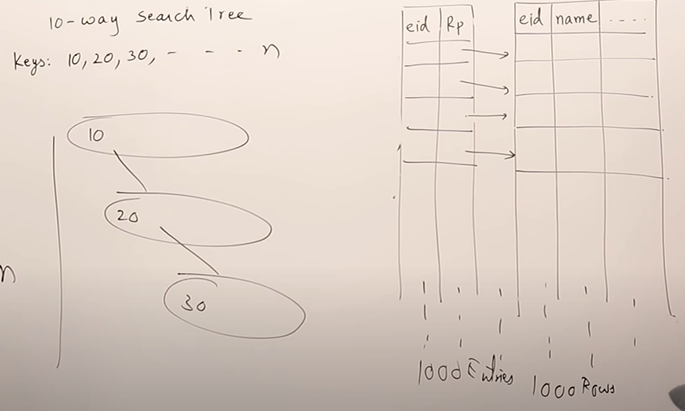
**M-way search tree**  
2 keys and 3 children/node(degree of the node) 🡪 3 way search tree  
m-1 keys and m children/node (degree of the node) 🡪 m way search tree

Lets take and example 4-way search trees (4 child and 3 keys)

   
Node structure  
Rp 🡪 Record Pointer (used to link with the data-base)

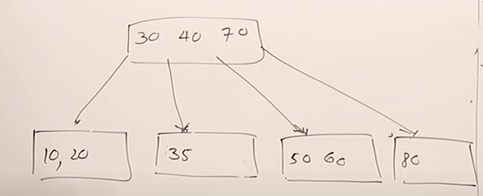


Index Table

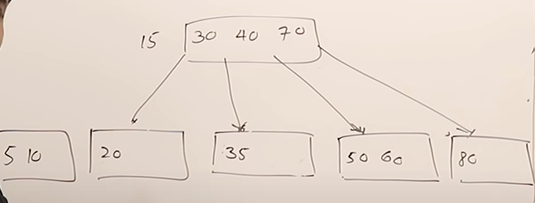
  
There is no-order of creating and inserting the data in the m-way search tree. We can use any-order. For n keys we can create n nodes and that is time consuming. So as a whole there is no guidelines of creating node in m-way search tree.

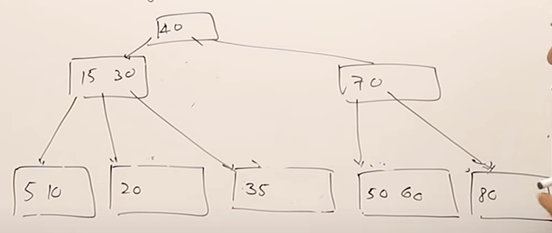
Now, B-trees are m-way search tree with rules to create node.  
**Rules**

1. The current node must be filled with at-least half of the children (i.e ceil value of m/2 )
2. Root can have minimum 2 children.
3. All leaf nodes must be at the same level.
4. Creation process is **Bottom up.**

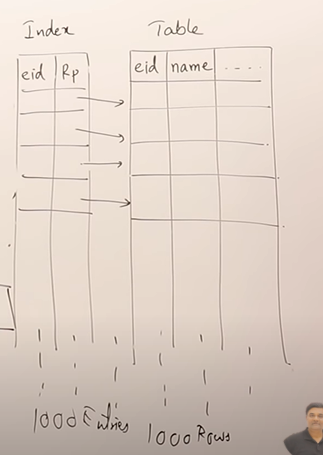
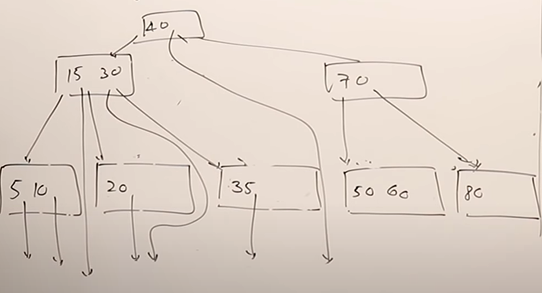


Now want to insert **15**

  
Problem with 15,30,40,70



Every key will have record pointer which is pointing towards the date-base



**B+ trees**

1)All the keys are present in the leaf nodes.  
2)In non-leaf nodes the duplicates are present.  
3)The record pointer will be only from leaf nodes.  
4)All the leaf nodes which are at the same level will form a linked list.

Rather than B tree, B+ trees is more like multi-level indexing

