## Chapter 3.2 B - Tree

Although One Level or Two Level Index can help speed up Query, normally in the Business System uses one more normal structure, which is called B - Tree, but the most normally used is called B + Tree.

* *B - Tree can keep adaptive index level with Database File automatically.*
* *Manage all used Storage Block and keep each Block between Half - Full and Full.*

This chapter would focus on B + Tree but not Tree.

### Chapter 3.2.1 Structure of B - Tree

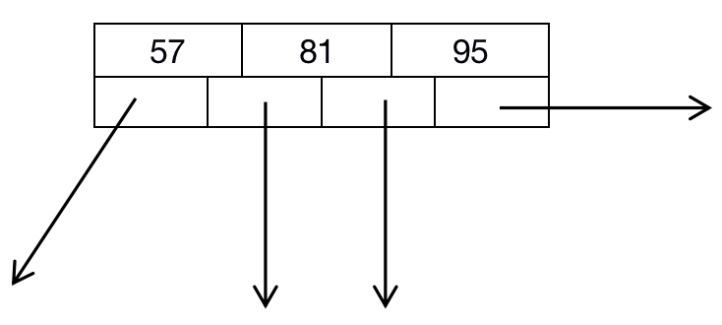
*Organization:*

B - Tree organizes all Storage Blocks into One Tree. This Tree is balanced, which means that all paths from Tree Root to Tree Leaf are the same. Normally, B - Tree has three levels: Root Level, Internal Level and Leaf Level, but also it can be random levels.

*Example:*

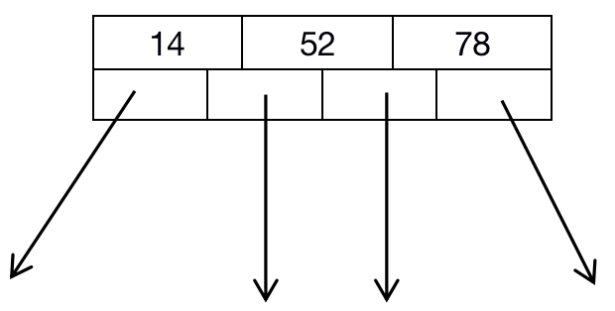
The typical *B - Tree Node:*

*Among B - Tree Nodes, there have three keys, which are 57, 81 and 95. The first three pointers point to the Tree Node which has the exact key value and the last pointer points to Next Node which has a bigger value. This is exact leaf situation, if this leaf node is the last one in the sequence, then the pointer equals to Null.*



The typical *B - Tree Internal Node:*

*Among B - Tree Internal Nodes, there have three keys, 14, 52, and 78. This Node have four pointers, through the first node with Key 14, we can reach all keys which are less than Key 14. Through the second node with Key 52, we can reach those nodes whose nodes are bigger than Key 14 and less than Key 52. Through the third node with Key 78, we can reach those nodes whose nodes are bigger than Key 52 and less than Key 78.*



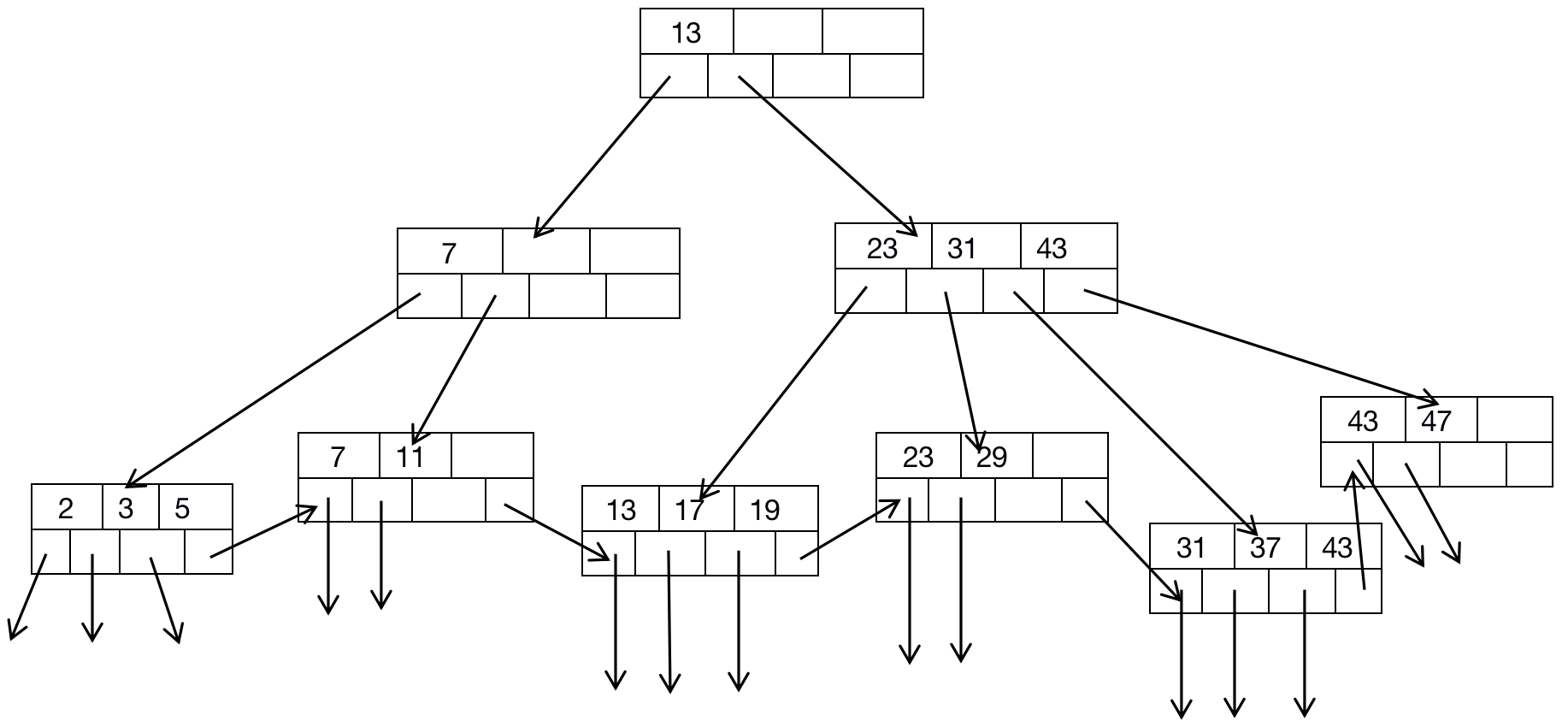
*Attention that, there has no need to fill all nodes with Keys and Pointers. When value of n equals to 3, then in the internal Node, there should have at least one key and two pointers.*

The typical *B - Tree:*

*Among Three Level B - Tree, n = 3. We assume that all keys among B - Tree belong to 2 to 47. Attention that, those values would exist once in Leaf for once. Each Tree Leaf Node has 2 - 3 Key - Value pair, and there also has the pointer which points to the next node. When we look from left to right, then they all have been sorted ascending.*

*There has only two nodes in the Tree Root, which is just the allowed pointer number. Although it is allowed at most 4 pointers, and key value in Root Node divides the tree into two parts with the key value which can be visited from first Tree Node and second Tree Node, which is to say, left child tree with Key Values which are less than key value 13 and the right child tree with Key Values which equals to or bigger than key value 13.*

*Attention that, there has Root Node with the four pointer values, which range from 23, 31, and 43. So the first part of the Tree would be Key Values which are less than 23, the second part of Tree would be Key Values which equal to or bigger than 23, less than 31, and the third part of Tree would be Key Values which equal to Key Value 43 or bigger than Key Value 43.*



*Principle:*

For each B - Tree Structure, there must have a parameter which is n, it decides the structure of all Storage Block. Each Storage Block would store n keys and n + 1 pointers. From some kind of meaning, the Storage Block of B - Tree also has one extra pointer which is used to point to the Next B - Tree Node. We need to make the value of n as big as possible.

*Example:*

Assume that we have the size of 4096 bytes of Storage Block, and the integer value occupies 4 bytes, the pointer occupies 8 bytes. As long as we do not need to consider the occupation size of Storage Block Head, then we hope to find the integer n which has the biggest value. 4 \* n + 8 \* (n + 1) <= 4096, then n takes 340.

*Rules of B - Tree:*

*Key in Leaf Node:*

* The keys in Leaf Node are all key copies of Data File, these keys are sorted, and distributed in all Leaf Nodes.

*Root Node:*

* There would be at least two pointers in the Root Node. All nodes point to the Tree Node in the next Level of B - Tree.

*Leaf Node:*

* In Leaf Node, the last node would points to the next Tree Node Storage Block, whose keys are all equal or bigger than those of Current Node Storage Block. In all other n pointers in Leaf Node, there would at least [ (n + 1) / 2 ] pointers to points to Data Records. Unused pointers would be seen as Null pointer and point to nowhere. If the ith pointer has been used, then it would points to the ith record.

*Internal Node:*

* In Internal Tree Node, all n + 1 pointers can be used to point to Storage Block of Tree Node in the Next Level of B - Tree. As the same, there would be at least [ (n + 1) / 2 ] pointers being used. If there are j pointers used, then also have j - 1 keys in the Storage Block, here, assume that these keys are K1, K2, K3, ..., Kj - 1. The First pointer points to those keys that are less than the Key K1. The Second pointer points to those keys that equals to or bigger than Key K1 but less than Key K2...At last, the Last pointer points to those keys that are bigger than Kj - 1.

*(Attention that, those keys are far less than K1 or much bigger than Kj - 1 can not be accessed by this block, but can be accessed by the same other blocks.)*

*(N + 1)th Leaf Node Pointer:*

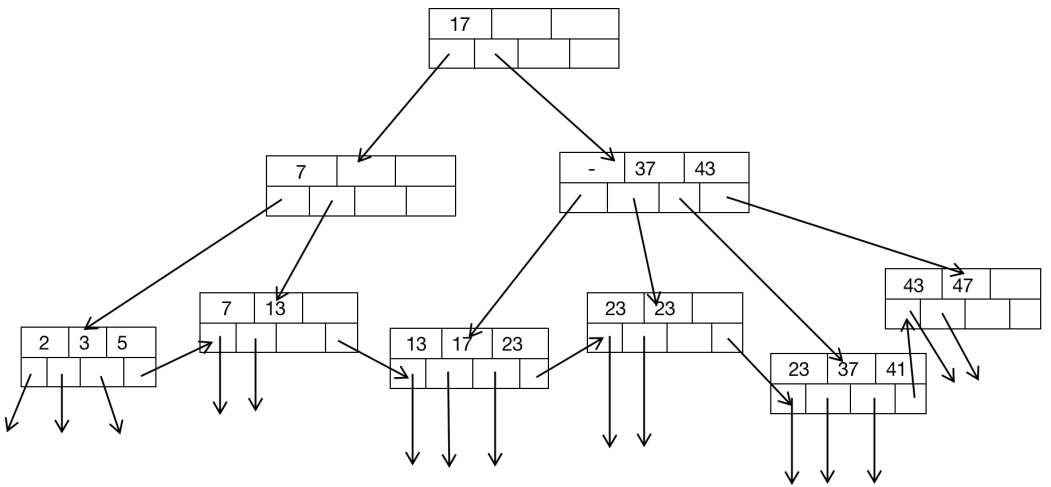
* All used keys and pointers are normally stay in start of Data Block, and the (n + 1)th Leaf Node Pointer is one exception, it points to the next Leaf Node.

### Chapter 3.2.2 Application of B - Tree

B - Tree is one of power tool which can be used to construct Index. Below are some useful instances:

1. *Query Key in B - Tree are Main Key in Data File, and Index are Dense Index, which is to say, the Leaf Node in the Data File records each Key - Pointer Pair. The Data File can be sorted according to Main Key, but it also can be not sorted according to Main Key.*
2. *Data File is sorted according to Main Key, and B + Tree is Sparse Index. Set up one Key - Data Pair in each block in Leaf Node.*
3. *Data File is sorted according to Non - Main Key, and this attribute is the Query Key of B + Tree. The Leaf Node would set up one Key - Data Pair for each attribute value in Data file.*

Another variant of B - Tree enables *Duplication Keys* exist in some Applications. Below is such a B - Tree:



*Definition:*

If we do allow the existence of Duplication Keys in B - Tree, then we need to modify the definition of Internal Node. Assume that Keys in one Internal Node is K1, K2, ..., Kn, then Ki would be the smallest new value which can be visited from ( i + 1 )th Sub - Tree. *‘New’ means that Key Value has not existed in the left of sub - Tree, but Key Value has existed for once in the current Sub - Tree.*

*(Attention that, in some situation, Ki can be null, but the corresponding pointer is still needed.)*

*Example:*

For B - Tree with no duplicates turns to with duplicates, we need to update Key Value of Root Node. To be more clear, we need to update the value of 13 to 17. Although 13 is the smallest key value in the left Tree,but it is not the newest value in this Sub - Tree, because it has appeared in the Left Tree.

We still need to do some updates of the second child node of the Root Node. The second node value equals to 37, since it is the new key value in the fifth Node from left to right. More interesting is that, the first Key Value equals to Null, because the forth Node has no new Key Value. If we want to find the key value 17, then we need to start from the first node in Second Part of B - Tree but not the second node. Or we can find the needed 23, we also need to start from the first node of Second Part of B - Tree.

*Attention:*

*Query 13:*

Query from the first Leaf Node but not the second Leaf Node when we want to query node with key value 13.

*Query value among 24 - 36:*

Query from the third Leaf Node, but when we can not find the required key value, then we do not need to proceed to the right and continue. For example, if Key Value 24 exists in the Leaf Node, then it would exist in the fifth Tree Node, and substitute the value of 24 as 37.

### Chapter 3.2.3 Query B - Tree

### Assume that there has no Duplication Key in Leaf Node and B - Tree is Dense Index, since each Query Key appears in Data File would appear in Leaf Node.

### Assume we have a B - Tree Index, and we want to find record with Query Key K. We need to find from Root to Leaf recursively, the Query Process is:

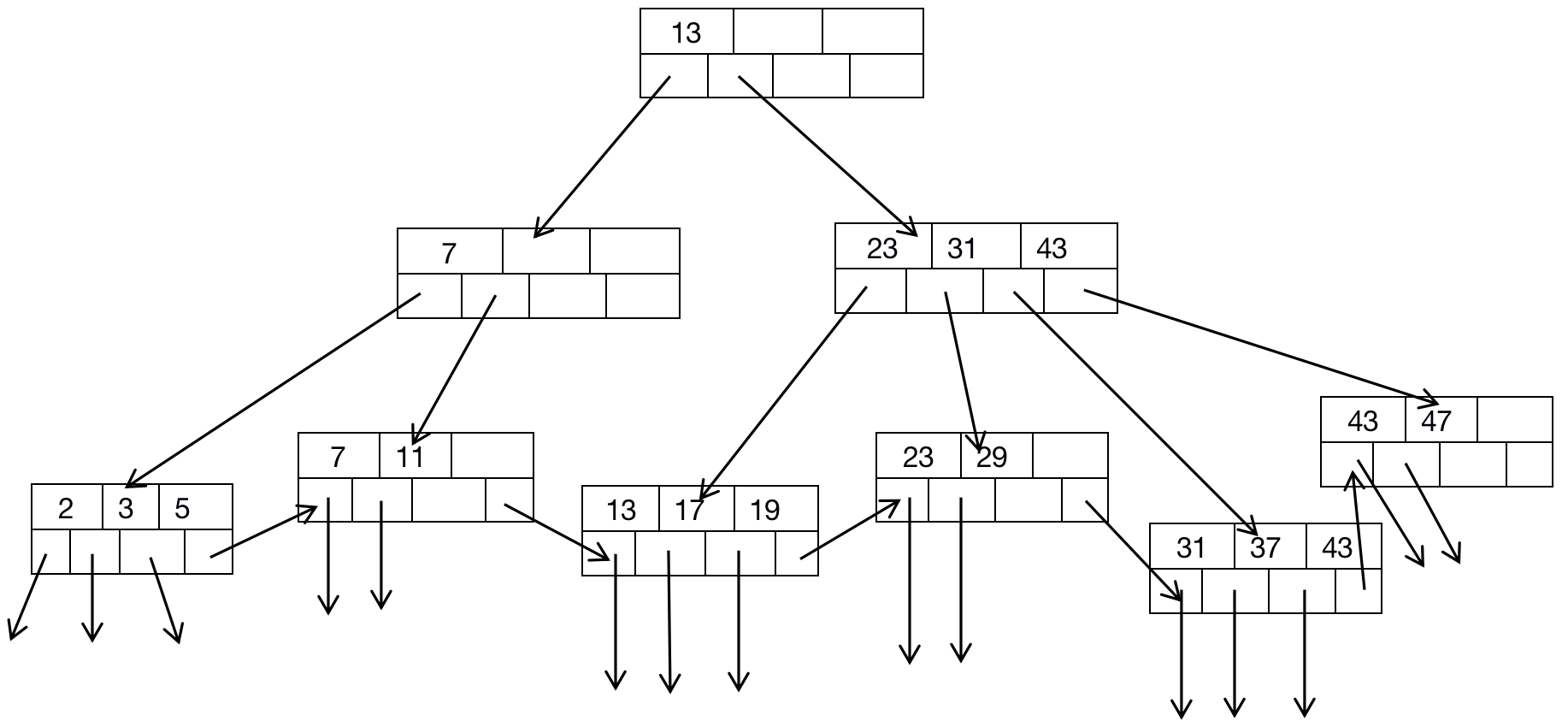
### *Basics:*

*If we stay in the Leaf Node, then we need to query from Key Value field. If the ith key equals to K, then we can get required value from ith pointer.*

### *Conclusion:*

### *If we stay in some Internal Node, and its Key Value are K1, k2, ..., Kn, we need to decide which route to go and query, which is to say, there has only one Leaf Node with Key Value K. If K is less than Ki, then it would be the first Node. If K1 <= K < K2, then it would be the second Node. We need to query the node recursively.*

### *Example:*

Assume we have one B - Tree, and we need to find the record with Key Value 40. 

We start from Root Node, among which there has one Key Value 13. Since 13 is less than 40, then we turn left and find the second B - Tree node with Key Value of 23, 31, and 43.

In this Node, we find that 31 <= 40 < 43, therefore we go to fifth Tree Node with Key Value 31, 37 and 43, among which there has not got the Key Value 40. So we can make sure that there has no Key Value 43 in all Leaf Node.

Attention that, if we want to find the node with Key Value 37, just as the steps above, we go to the fifth Tree Node and find the Key Value 37. So we can go further to query the Record with Key Value 37.

### Chapter 3.2.4 Range Query

### Chapter 3.2.5 Insertion into B - Tree

### Chapter 3.2.6 Deletion from B - Tree

### Chapter 3.2.7 Productivity of B - Tree