Balanced Tree

What it is and how it works.

Course: Database Management

Submitted By: Ruchir Babbar

Date: December 12, 2006

Table of contents

Introduction 3
  B-tree 3
  Index 3
Creation of B-tree index 4
  Creation of root and leaf node 5
  Creation of branch node 6
How B-tree works 7
  When doing a search 7
  When inserting new element 8
  When deleting an element 9
    Deletion from leaf node 9
    Deletion from an internal node 9
Rebalancing B-tree after deletion 10
An Example 11
    Search a record 11
    Insert a record 12
    Delete a record 13
References: 15

Introduction

B-tree

B-trees are tree like data structures most commonly used in databases and file systems. B-trees keep the data sorted and allow amortized logarithmic time insertions and deletions. It is a technique for organizing indexes to keep the access time to the minimum. It stores the data key in a balanced hierarchy that continually realigns itself as items are inserted and deleted.

B-tree was created by Rudolf Bayer and Ed McCreight. The word *B* in B-tree was never explained by its developers however it is widely believed that *B* stands for *Balanced*.

Index

An index is a feature in a database that allows quick access to the rows in a table. An index is created using one or more columns of the table. Indexes are often optimized for quick searches, usually via balanced tree. An index in a relational database is a copy of a part of a table structured in a specific format.

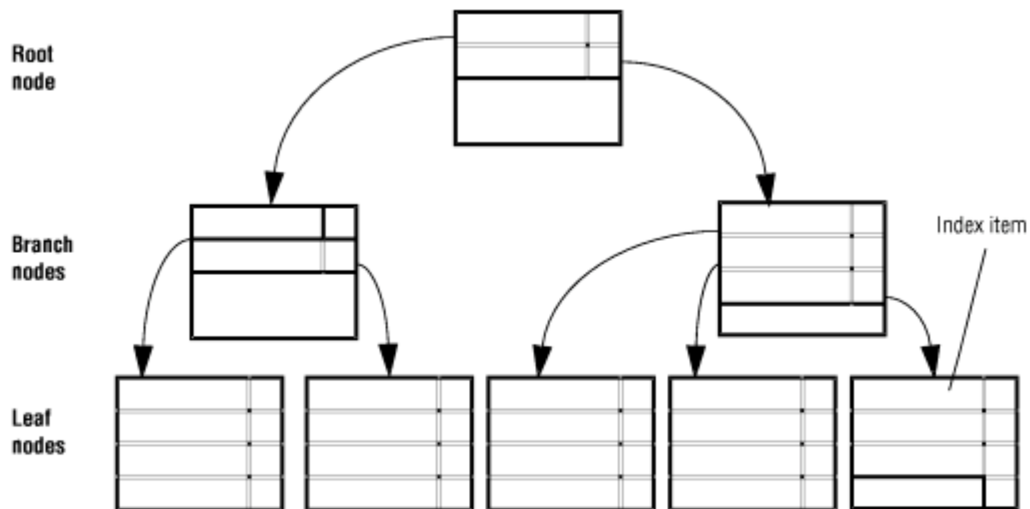
Note: This paper does not deal with the technical side of the B-tree.

Creation of B-tree index

A B-tree index is usually created on columns which contain mostly unique values. A B-tree index is most effective when a query retrieves less than twenty percent of rows in a table

B-tree contains following different types of nodes:

- One root node: A node that contains node pointed to branch nodes
- Two or more branch nodes: Branch node contain pointer to leaf nodes or other branch nodes
- Many leaf nodes: A leaf node contains index items and horizontal pointers to other leaf nodes.

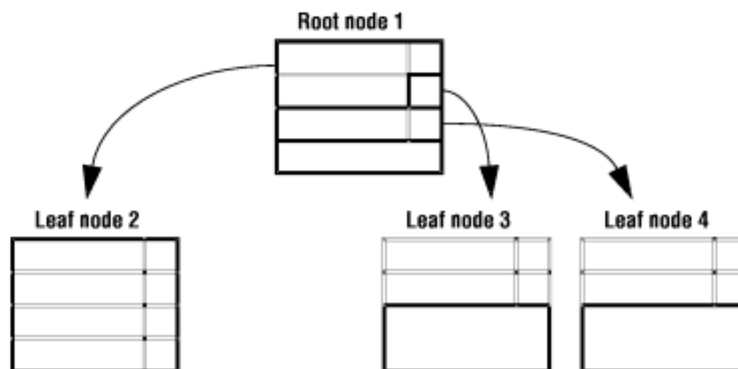


Creation of root and leaf node

When an index is created in an empty table, the database server allocates a single index page. This page represents the root node and remains empty until data is inserted in the table.

At the start the root node works the same way as the leaf node. Each time a new row is inserted in the table, the database server creates and inserts an in the root node.

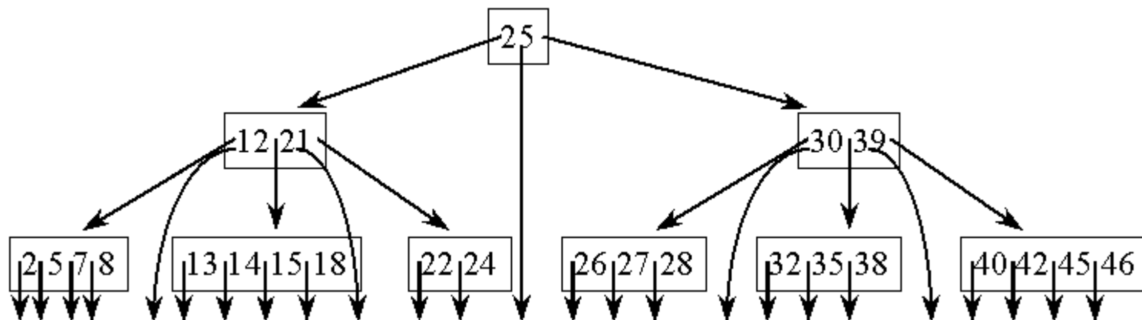
When the root node is full of the index items then database server splits the root node by first creating two leaf nodes, then moving approximately half of the root-node entries to each of the newly created leaf nodes. In last step it puts the pointers to leaf nodes in the root node.



Creation of branch node

As new rows are added to the table, the database server fills the root node with the node pointers to the entire existing leaf node. When the database server splits another leaf node and the root node has no room for an additional node pointer then database server splits the root node and divides its contents among two newly created branch nodes. As index items are added, more and more leaf nodes are split and more branch nodes are added. Eventually, the root node fills with pointers to these branch nodes.

When the similar situation occurs again the root node is split again and creates another branch level between the root node and lower branch level. The B-tree structure can continue to grow in this way to a maximum of 20 levels. Branch nodes can either point to other branch node below them or to leaf nodes. A B-tree is kept balanced by requiring that all leaf nodes are at the same depth.



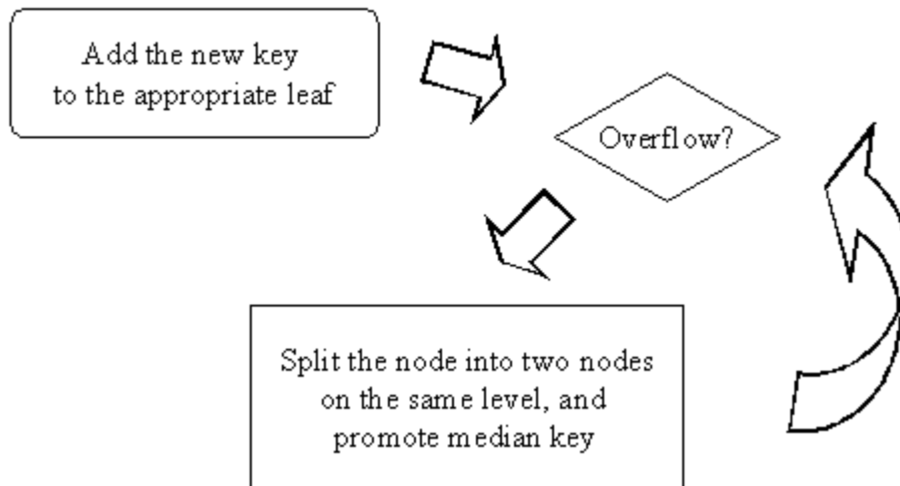
How B-tree works

When doing a search

Search starts at the root node. A binary search (the search similar to what is done when looking up for the information is dictionary or encyclopedia) is done on the keys in the node. If the key for which the search was done is found in the node then search is complete. If the key is not found in node then it looks for the key that is larger than the given key and the hey that is smaller than the given key and then moves to the appropriate in between and continues its search till it finds the required key or till it reaches the end of tree (which ever is first).

When inserting new element

All the insertion happens at the leaf node of the B-tree. It starts with searching the tree for the leaf node where the element should be. If the leaf node contains fewer elements that the maximum legal allowed numbers and there is room for one more element then the new element is inserted in the node. If not, then leaf node is split into two nodes and promotes one extra key to parent node and new key is added to appropriate node.



When deleting an element

Deletion from leaf node

If node has more than the minimum number of entries, then one can be deleted without any further actions

Deletion from an internal node

Each element in internal node acts as a separation values for two sub trees. Upon deletion of that element two cases arise. In first case, both of the two child nodes to the left and right of the deleted element have the minimum number of elements and they can be joined into a single node and the number of those should not exceed the allowed values in a node.

In second case, one of the two child nodes contain more than the minimum number of elements then the new separator for those sub trees must be found.

Rebalancing B-tree after deletion

If deleting and element from a leaf node has brought it under the minimum size, then some element is re-distributed to bring all nodes up to the minimum. In some cases, during the rearrangement, any deficiencies are moved up to parent and redistribution is applied iteratively up the tree, sometimes even to root node.

An Example

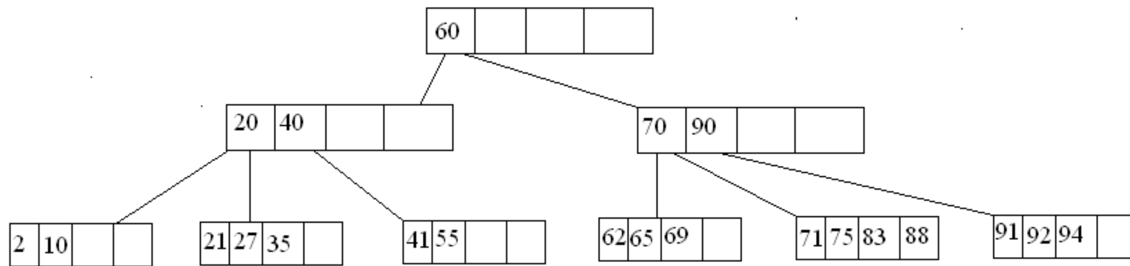


Figure: 1.1

Note: each node can hold up to 4 records

Search a record

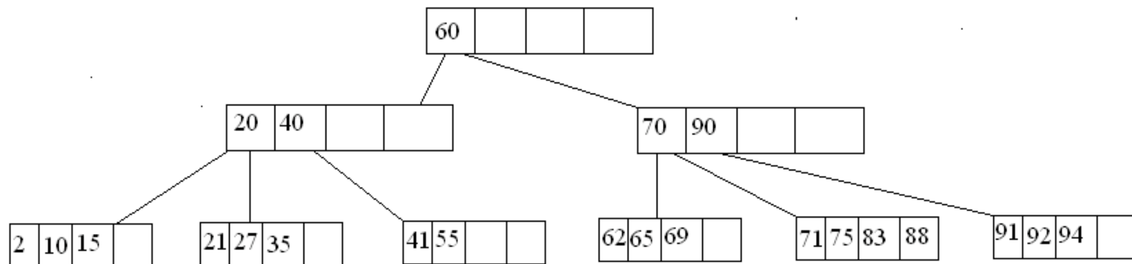
If we are to search for the record with key 35 following steps will be performed

- Check the key 35 against key 60 (35< 60). Go to next key. In this case there is no other key so move to other node.
- Check 35 against 20 (35>20). Go to next key (35<40). Go to next key. Since there is no next key move to the leaf node which is greater than 20 and less than 40.
- Check 35 against 21 (35>21). Go to next key (35>27). Go to next key (35=35) return successful.

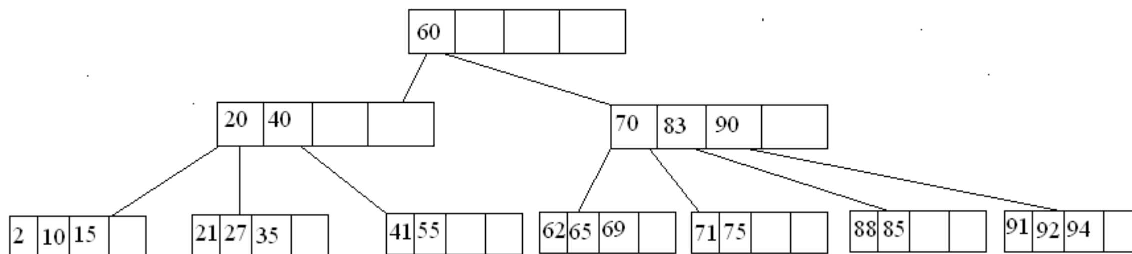
Trace track in this case: 60, 20, 40, 21, 27, and 35 successful.

Insert a record

Suppose inserting key 15. From figure 1.1 is it clear that it will go in left most leaf node.



Suppose inserting the key 85. From figure 1.1 it is clear that it should go in second to last leaf node however that will cause the over flow in leaf node. So, a new empty node is created. The middle key of the node is moved to the parent node and new key is filled in the new node and a new pointer is created from parent node to the leaf node.



Delete a record

Suppose deleting key 35 from B-tree. From the figure 1.1 it is clear that it will be deleted from second left most leaf node. Since after deletion of the key the record will still have minimum legal (required for the existence of node) keys, no further action is required.

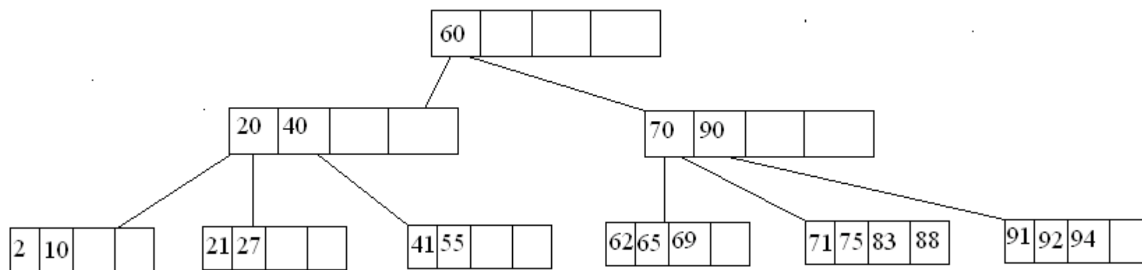


Figure 1.2

Suppose we delete the key 90 from figure 1.2. Since key 90 is in parent node, following steps will be performed to delete the key:

- Locate the next highest key in the leaf node (this search is always performed in leaf node).
- Overwrite the key to be deleted (in this case 90) with the leaf node key (in this case 91) and then delete key 91 from the leaf node.

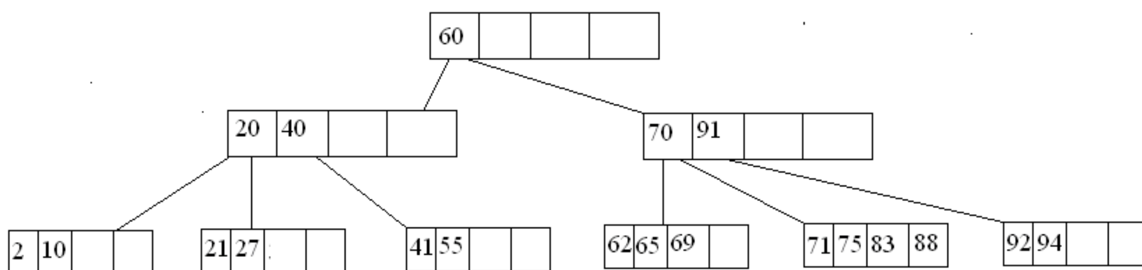
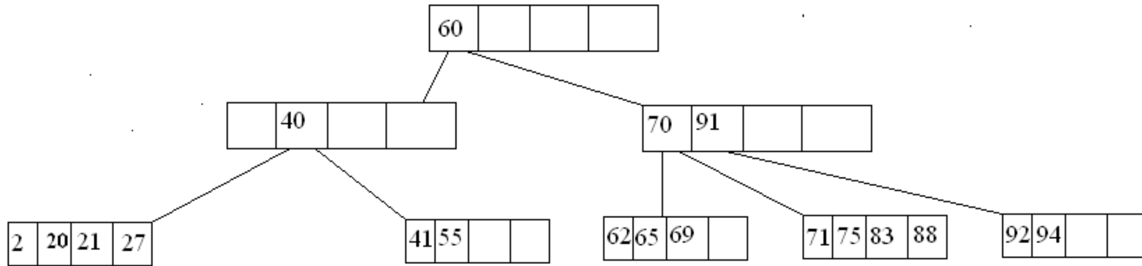


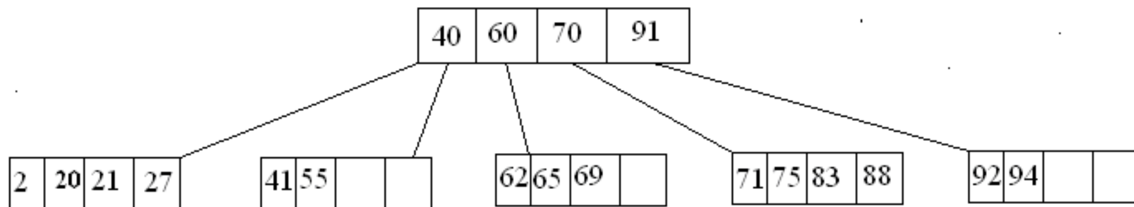
Figure 1.3

Now, suppose we are deleting key 10 from figure 1.3 it is clear that it has to be deleted from the left most leaf node. However the deletion of key 7 will cause imbalance in the tree. To balance the tree following steps will be performed.

- Parent node, delete node and neighbor node will be combined into single node.



This causes imbalance in internal node. Therefore we need to repeat the process. The Internal node and the root node will collapse into one node to restore the balance.



References:

http://en.wikipedia.org/wiki/B-tree

http://en.wikipedia.org/wiki/Index_%28database%29

http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.adref.doc/adref235.htm

http://www.answers.com/topic/b-tree

http://publib.boulder.ibm.com/infocenter/rbhelp/v6r3/index.jsp?topic=/com.ibm.redbrick.doc6.3/perf/perf25.htm

http://sky.fit.qut.edu.au/~maire/baobab/lecture/

http://www.ittc.ku.edu/~sgauch/647/s00/notes/Ch6b/sld002.htm

http://www.stonehill.edu/compsci/CS325/BTree.ppt