Database Management System 24

Data Storage & Indexing

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Files

- Databases are stored on magnetic disks as files of records
- Unit of information read from or written to disk is a *page*. The size of a page is 4KB, 8KB etc
- Disks can retrieve random page at fixed cost. But, reading several consecutive pages is much cheaper than reading them in random order
- Tapes can only read pages in sequence. they are mostly used to archive data that is not needed on a regular basis
- File organization is the method of arranging a file of records on external storage
- Buffer manager stages pages from external storage to main memory buffer pool. File and index layers make calls to the buffer manager

File Organization

File Organization

- Database is stored as a collection of files
  - Each file is a sequence of records
  - Each record is a sequence of fields
- Basic approach
  - Assume record size is fixed
  - Each file has records of one particular type only
  - Different files are used for different relations

Logical & Physical Records

- **Logical Record**: Ä record"
- **Physical Record**: The unit of transfer between disk and primary storage. Ä page", Ä block"
- *Generally, a physical record consists of more than one logical record*

File Organization...

File Organization & Access Method

- File organization means the physical arrangement of data in a file into records and pages on secondary storage
  Ex: Ordered files

- Access method means the steps involved in storing and retrieving records from a file
  Ex: Using an indexed access method to retrieve a record from an indexed sequential file

Sequential File Organization

Easiest method for file organization. Files are stored sequentially

Pile File Method

- Records are stored in a sequence. Record will be inserted in the order in which they are inserted into tables
- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting
- New records will be added at the end of the file

Sorted File Method

- The new record is always inserted at the end of the file, then it will sort the sequence
- In case of modification of any record, it will update the record and then sort the file

Sequential File Organization...

- Advantages:
  - It contains a fast and efficient method for the huge amount of data
  - In this method, files can be easily stored in cheaper storage mechanism
  - It is simple in design
  - This method is used when most of the records have to be accessed

- Disadvantages:
  - It will waste time as we cannot jump on a particular record that is required
  - Sorted file method takes more time and space for sorting the records

Heap Files

- It is the simplest and most basic type of organization. It works with data blocks. Records are inserted at the end of the file. When the records are inserted, it doesn't require the sorting and ordering of records. When the data block is full, the new record is stored in some other block
- In the file, every record has a unique id, and every page in a file is of the same size
- If we want to search, update or delete the data in heap file organization, then we need to traverse the data from starting of the file till we get the requested record
- If the database is very large then searching, updating or deleting of record will be time-consuming. Here, we need to check all the data until we get the requested record
- Advantages: very good method of file organization for bulk insertion
- Disadvantages: inefficient for the large database because it takes time to search or modify the record

Hash File Organization

- It uses the computation of hash function on some fields of the records. The hash function's output determines the location of disk block where the records are to be placed
- When a new record has to be inserted, then the address is generated using the hash key and record is directly inserted
- When a record has to be retrieved using the hash key columns, then the address is generated, and the whole record is retrieved using that address
- Each record will be stored randomly in the memory

Indexed Sequential Access Method (ISAM)

Indexed Sequential Access Method (ISAM)

- It is an advanced sequential file organization
- Records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file
- If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory
- Advantages:
  - Each record has the address of its data block, searching a record in a huge database is quick and easy
  - Since the index is based on the primary key values, we can retrieve the data for the given range of value
- Disadvantages:
  - It requires extra space in the disk to store the index value
  - When the new records are inserted, then these files have to be reconstructed to maintain the sequence
  - When the record is deleted, then the space used by it needs to be released

B+ File Organization

- It is the advanced method of an indexed sequential access method. It uses a tree-like structure to store records in File
- It uses the same concept of key-index where the primary key is used to sort the records. For each primary key, the value of the index is generated and mapped with the record
- Searching for any record is easier as all the leaf nodes are balanced
- Advantages:
  - Here, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list
  - Traversing through the tree structure is easier and faster
  - The size of the B+ tree has no restrictions
- Disadvantages:
  - This method is inefficient for the static method

Cluster File Organization

Cluster File Organization

- When the two or more records are stored in the same file, it is known as clusters
- These files will have two or more tables in the same data block, and key attributes which are used to map these tables together are stored only once
- This method reduces the cost of searching for various records in different files
- The cluster file organization is used when there is a frequent need for joining the tables with the same condition. These joins will give only a few records from both tables
- In this method, we can directly insert, update or delete any record. Data is sorted based on the key with which searching is done
- Cluster key is a type of key with which joining of the table is performed

Cluster File Organization...

Indexed Clusters

- In indexed cluster, records are grouped based on the cluster key and stored together

Hash Clusters

- In hash cluster, instead of storing the records based on the cluster key, we generate the value of the hash key for the cluster key and store the records with the same hash key value

- Advantages:
  - The cluster file organization is used when there is a frequent request for joining the tables with same joining condition
  - It provides the efficient result when there is a 1:M mapping between the tables
- Disadvantages:
  - This method has the low performance for the very large database
  - If there is any change in joining condition, then this method cannot use

Index

- An index is a data structure that organizes data records on disk to optimize the retrieval operations. An index is used to efficiently retrieve all records that satisfy search conditions on the **search key** fields of the index
- The index structure typically provides secondary access path, which provides alternative way of retrieving the records without affecting the physical storage of records on the disk
- On a same file, multiple indexes on different fields can be constructed
- Entire index file is loaded into main memory data
- Index files are much smaller than the original file

Single-Level Indexes

Index is usually defined on a single attribute or field of a file, called *indexing field* or *indexing attribute*

Generally, the index stores each value of the index field along with a list of pointers to all disk blocks that contain records with that field value

The values in the index are ordered so that binary search can be performed on the index. The index file is much smaller than the data file, so searching the index using a binary search is highly efficient

Student

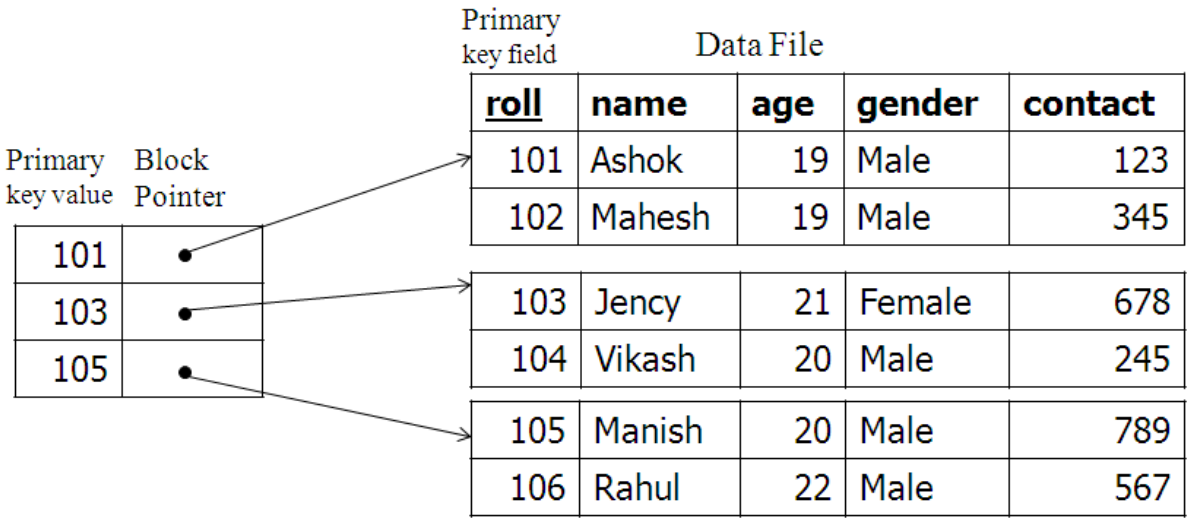
roll	name	age	gender	contact
101	Ashok	19	Male	123
102	Mahesh	19	Male	345
103	Jency	21	Female	678
104	Vikash	20	Male	245
105	Manish	20	Male	789
106	Rahul	22	Male	567

Primary Indexes

Primary Indexes

A primary index is an ordered file whose records are of fixed length with two fields. The first field is of the same data type as the ordering key field of the data file, called the primary key and the second field is a pointer to a disk block

There is one index entry in the index file for each block in the data file. Each index entry has the value of the primary key field for the first record in a block and a pointer to that block as its two field values



Primary Indexes...

The index file for a primary index needs substantially fewer blocks than does the data file, for two reasons. First, there are fewer index entries than there are records in the data file. Second,each index entry is typically smaller in size than a data record because it has only two fields. A binary search on the index file hence requires fewer block accesses than a binary search on the data file

When a user wants to insert a record to its correct position in the data file, the existing records should be moved to make space for the new record as well as index entries will be changed

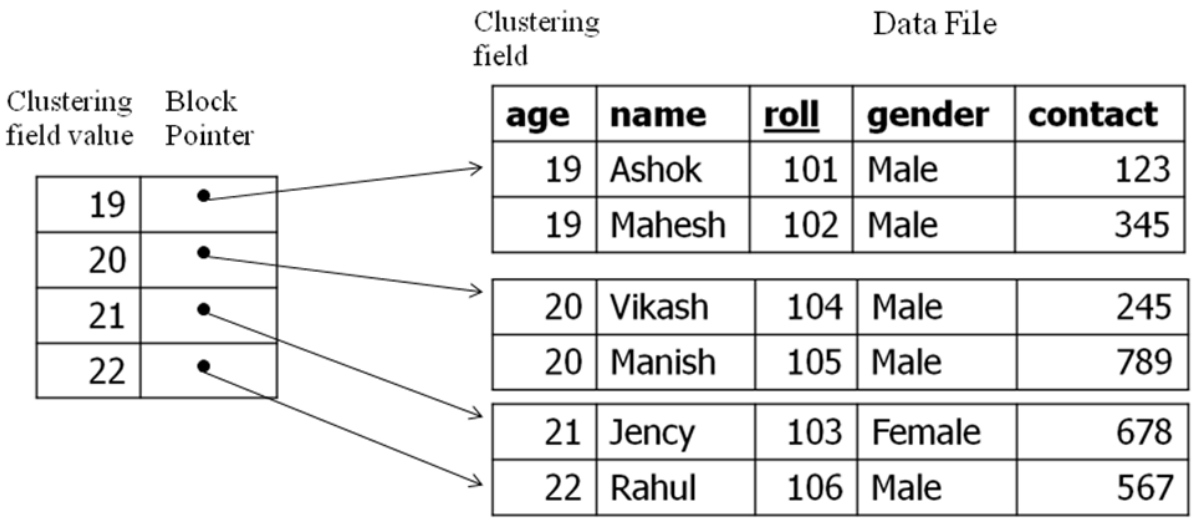
Similarly, deletion process is also difficult due to the index entries updation. But, record deletion is commonly handled by using deletion markers

Clustering Indexes

Clustering Indexes

If records of a file are physically ordered on a non-key field, that field is called the clustering field. Thus, clustering index is the index created on the clustering field to speed up retrieval of records that have the same value for the clustering field

This differs from a primary index, which requires that the ordering field of the data file have a distinct value for each record



Clustering Indexes...

Clustering Indexes...

There is one entry in the clustering index for each distinct value of the clustering field, containing the value and a pointer to the first block in the data file that has a record with that value for its clustering field

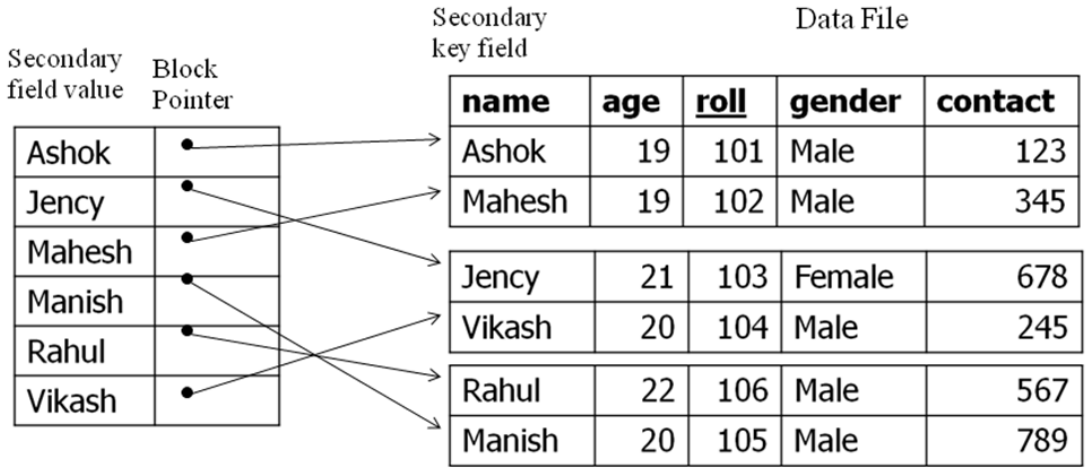
To solve the problem of insertion, it is common to reserve a whole block (or a cluster of contiguous blocks) for each value of the clustering field; all records with that value are placed in the block (or block cluster). This makes insertion and deletion relatively straightforward

Secondary Indexes

Secondary Indexes

A secondary index is also an ordered file with two fields. The first field is of the same data type as some nonordering field of the data file that is an indexing field. The second field is either a block pointer or a record pointer. There can be many secondary indexes for the same file

In this case there is one index entry for each record in the data file, which contains the value of the secondary key for the record and a pointer either to the block in which the record is stored or to the record itself



Secondary Indexes...

A secondary index usually needs more storage space and longer search time than does a primary index, because of its larger number of entries

Secondary index provides a logical ordering on the records by the indexing field

A data file can have either a primary index or a cluster index depending on its ordering field. It can have several secondary indexes, however

Dense/Sparse Indexes

Indexes can also be characterized as dense or sparse:

- A dense index has an index entry for every search key value (and hence every record) in the data file
- A sparse or nondense index has index entries for only some of the search values

Primary index $\rightarrow$ Nondense

Clustering index $\rightarrow$ Nondense

Secondary index $\rightarrow$ Dense

Multilevel Indexing

A multilevel indexing can contain any number of levels, each of which acts as a non-dense index to the level below. The top level contains a single entry

A multilevel index can be created for any type of first-level index (whether it is primary, clustering or secondary) as long as the first-level index consists of more than one disk block

The advantage of multilevel index is that it reduces the number of blocks accessed when searching a record, given its indexing field value

The problems associated with index insertions and deletions still exist because all index levels are physically ordered files

Multilevel Indexing...

Because of the insertion and deletion problem, most multilevel indexes use B-tree or B^+-tree data structures, which leave space in each tree node (disk block) to allow for new index entries

In B-Tree and B^+-Tree data structures, each node corresponds to a disk block. Here, each node is kept between half-full and completely full

An insertion into a node that is not full is quite efficient. On the other hand, if a node is full the insertion causes a split into two nodes

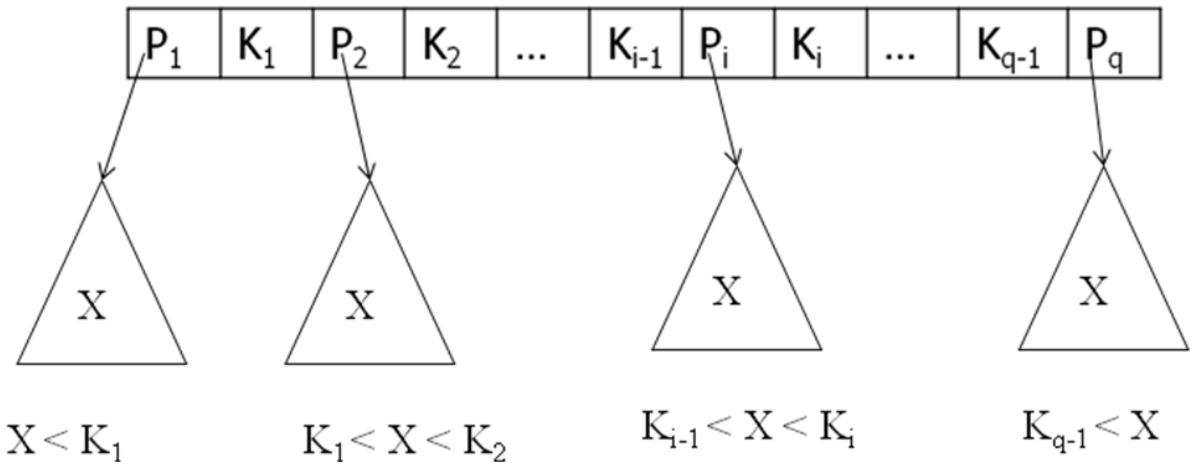
Similarly, a deletion is quite efficient if a node does not become less than half full. If a deletion causes a node to become less than half-full, it must be merged with the neighboring nodes

Search Trees

A search tree is a special type of tree that is used to search a record, given the value of one of the record's fields

A search tree of order 'p' is a tree such that each node contains at most 'p-1' search values and 'p' pointers in the order $< P_1, K_1, P_2, K_2, \ldots P_{q-1}, K_{q-1}, P_q >$, where $q \leq p$; each P_i is a pointer to a child node and each K_i is a search value from some ordered set of values

All of these search values are unique

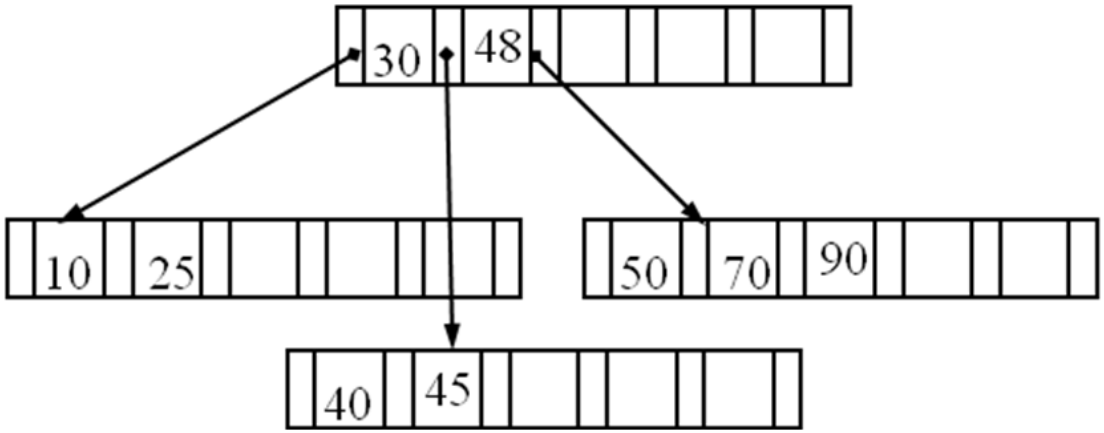


Search Trees...

Search Trees...

Two constraints must hold at all times on the search tree:

- Within each node, $K_1 < K_2 < ... < K_{q-1}$
- For all values X in the subtree pointed at by P_i, X < K_1; K_{i-1} < X < K_i for 1 < i < q; and K_{i-1} < X for i = q



When a user wants search a value X, he has to follow the appropriate pointer P_i according to the formula in condition 2

A search tree is balanced when all of its leaf nodes are at the same level

B-Trees

B-tree is a specialized multi-way tree designed especially for use on disk. A B-tree of order 'p' can be defined as follows:

- Each internal node is of the form <P_1, <K_1>, P_2, <K_2> ... <K_{q-1}>, P_q> , where q $\leq$ p. Each P_i is a tree pointer, which is a pointer to another node in the B-tree
- Within each node, $K_1 < K_2 < ... < K_{q-1}$
- Each node has at most 'p' tree pointers
- For all search key field values X in the subtree pointed by P_i, the rule is X < K_1; K_{i-1} < X < K_i for 1 < i < q; and K_{i-1} < X for i = q
- All non-leaf nodes except the root have at least $\lceil$ p / 2 $\rceil$ children. The root is either a leaf node, or it has from two to p children
- A node with q tree pointers, q $\leq$ p, has q-1 search key field values
- All leaf nodes are at the same level. Leaf nodes have the same structure as internal nodes except that all of their tree pointers P_i are null

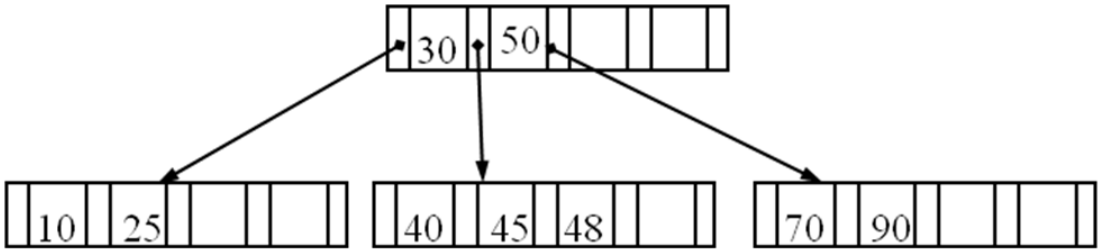
B-Trees...

Insertion

The insertion to a B-tree is an easier process. A B-tree starts with a single root node at level 0. The rules for the insertion to B-tree are:

- It attempts to insert the new key into a leaf. If this would result in that leaf becoming too big, split the leaf into two, promoting the middle key to the leaf's parent
- If the insertion would result in the parent becoming too big, split the parent into two, promoting the middle key. This strategy might have to be repeated all the way to the top
- If necessary, the root is split in two and the middle key is promoted to a new root, making the tree one level higher

Ex: Draw the B-tree of the order 5 for the data items 10, 50, 30, 70, 90, 25, 40, 45, 48



B-Trees...

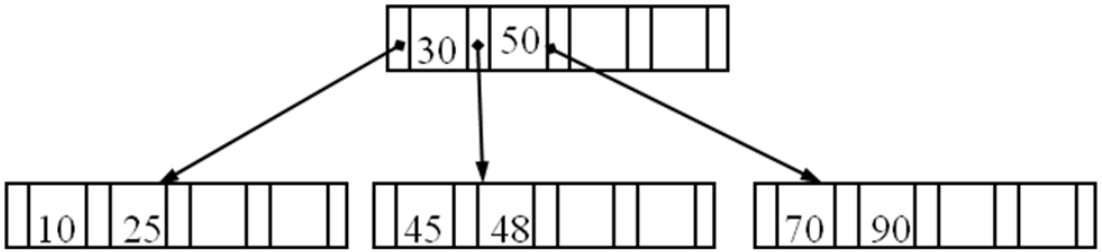
Deletion

At deletion, removal should be done from a leaf:

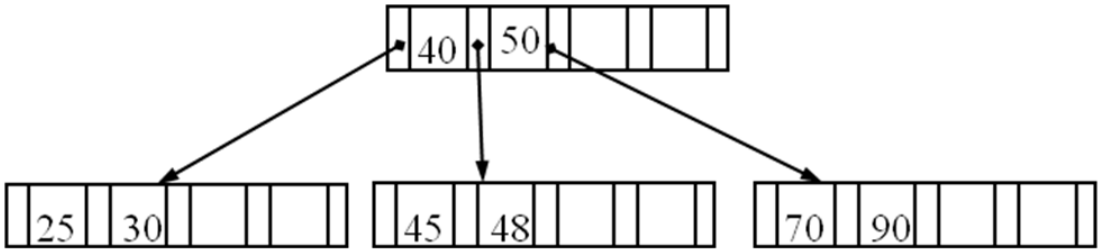
- If the key is already in a leaf node, and removing it doesn't cause that leaf node to have too few keys, then simply remove the key to be deleted
- If the key is not in a leaf, then it is guaranteed that its predecessor or successor will be in a leaf. In this case, delete the key and promote the predecessor or successor key to the non-leaf deleted key's position
- If first or second condition lead to a leaf node containing less than the minimum number of keys, then look at the siblings immediately adjacent to the leaf:
  - If one of them has more than the minimum number of keys, then promote one of its keys to the parent and take the parent key into the lacking leaf
  - If neither of them has more than the minimum number of keys, then the lacking leaf and one of its neighbours can be combined with their shared parent; & the new leaf will have the correct number of keys. If this step leaves the parent with too few keys, repeat the process up to the root itself

B-Trees...

Deletion of 40



Deletion of 10



The maximum number of items in a B-tree of order p and height h = p^{h+1} - 1. For example, when the order p=5 and height h=1, the maximum number of nodes= 5^2-1=24

B^+-Trees

The leaf nodes of the B^+-tree are usually linked together to provide ordered access on the search field to the records. These leaf nodes are similar to the first level of an index

Internal nodes of the B^+-tree correspond to the other levels of a multilevel index. The structure of the **internal nodes** of a B^+-tree of order p is as follows:

- Each internal node is of the form <P_1, <K_1>, P_2, <K_2> ... <K_{q-1}>, P_q> , where q $\leq$ p and each P_i is a tree pointer
- Within each internal node, $K_1 < K_2 < ... < K_{q-1}$
- Each internal node has at most 'p' tree pointers
- For all search field values X in the subtree pointed by P_i, the rule is X $\leq K_1$; $K_{i-1} <$ X $\leq K_i$ for 1 < i < q; and $K_{i-1} <$ X for i = q
- Each internal node has at least $\lceil$ p / 2 $\rceil$ children. The root has at least two children if it is an internal node
- An internal node with q tree pointers, q $\leq$ p, has q-1 search key field values

B^+-Trees...

The structure of the **leaf nodes** of a B^+-tree of order p is as follows:

- Each leaf node is of the form «K_1, Pr_1>, <K_2, Pr_2> ... <K_{q-1}, Pr_{q-1}>, P_{next}>, where q $\leq$ p, each Pr_i is a data pointer and P_{next} is the pointer to the next leaf node
- Each leaf node has at least $\lceil$ p / 2 $\rceil$ values
- All leaf nodes are at the same level
- Within each leaf node, $K_1 \leq K_2 \leq ... \leq K_{q-1}$

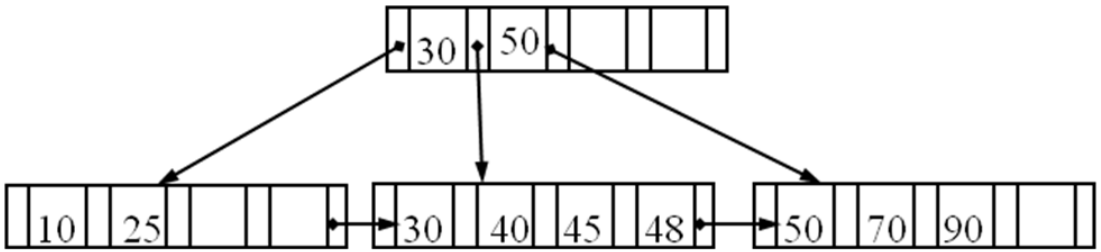
The pointers in internal nodes are the tree pointers, which point to blocks that are tree nodes, whereas the pointers in leaf nodes are the data pointers to the data file records

B^+-Trees...

Insertion

The rules for the insertion of a data item to the B^+-tree are:

- Find correct leaf L
- Put data entry onto L. There are two options for this entry:
  - If L has enough space, done!
  - Else, split L into L and a new node L2. Redistribute entries evenly and copy up the middle key. Also, insert index entry pointing to L2 into parent of L
- For each insertion, this process will be repeated



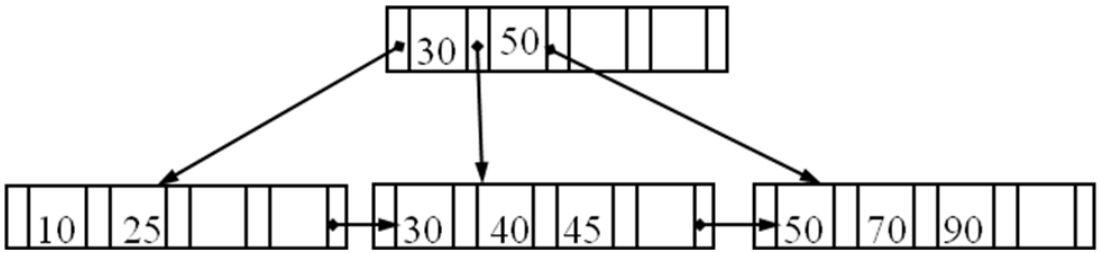
B^+-Trees...

Deletion

The rules for deleting a data item from the B^+-tree are:

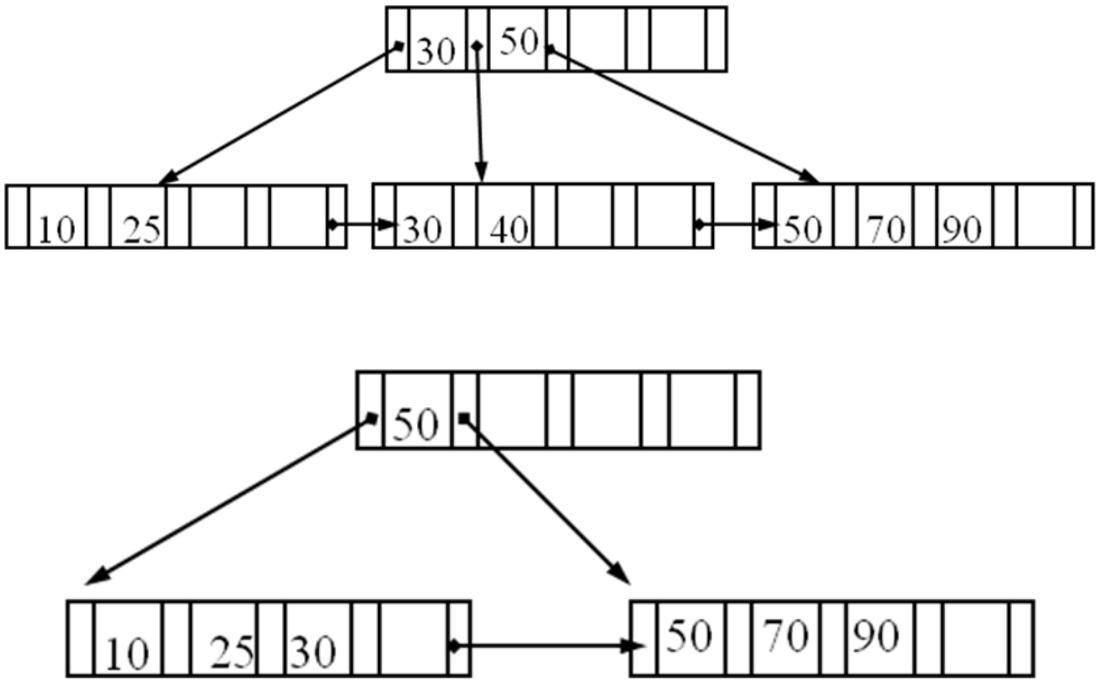
- Find the correct leaf L
- Remove the entry. There may be two possible cases for this deletion:
  - If L is at least half-full, done!
  - Else, try to re-distribute the entries by borrowing from the adjacent node or sibling. If re-distribution fails, merge L and the sibling. If merge occurred, then delete the entry (pointing to L or sibling) from parent of L
- The merging process could propagate to root; thus, decreasing the height

Deletion of 48



B^+-Trees...

Deletion of 45 followed by 40



The maximum number of records stored in a B^+-tree of order p and height h = p^h- p^{h-1}. Similarly, the minimum number of records stored = $2(p/2)^{h-1}$; and the minimum number of keys in a B^+-tree of order p and height h = $2(p/2)^h$ - 1

B-Trees vs. B^+-Trees

- B^+-tree searching is faster than the B-tree searching; because in B^+-tree, all records are stored at the leaf level of the tree; and only keys are stored in interior nodes
- B^+-tree takes more storage space than B-tree, because it uses extra pointers than B-tree
- Insertion and deletion operations in B-tree are more complex than those in B^+-tree