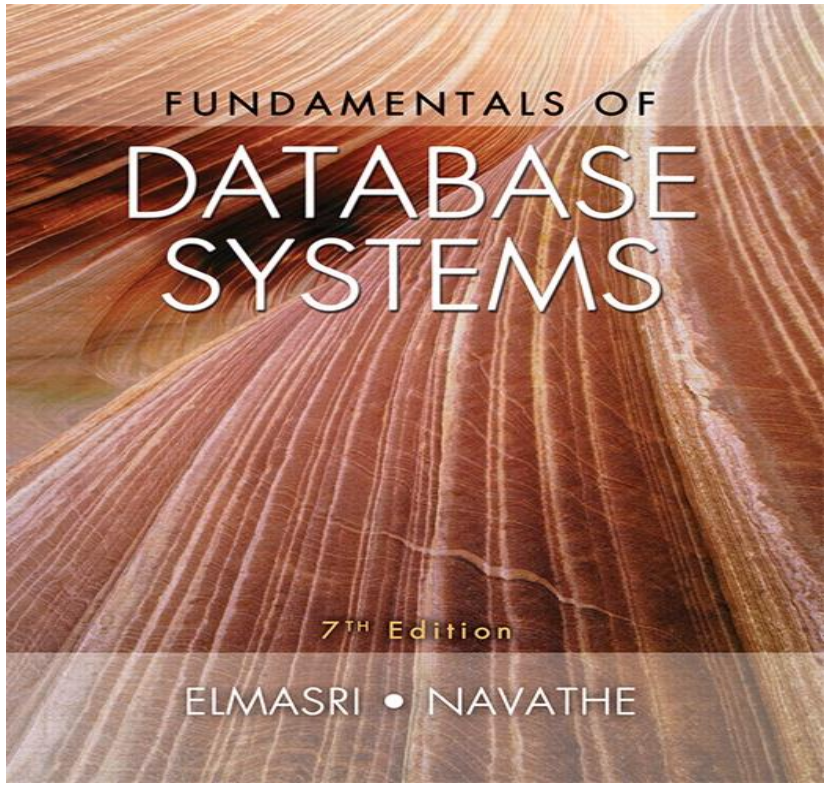
Comp-3150: Database Management Systems

- Ramez Elmasri , Shamkant B. Navathe(2016) Fundamentals of Database Systems (7th Edition), Pearson, isbn 10: 0-13-397077-9; isbn-13:978-0-13-397077-7.

Chapter 17:
Indexing Structures for Files and Physical Database Design



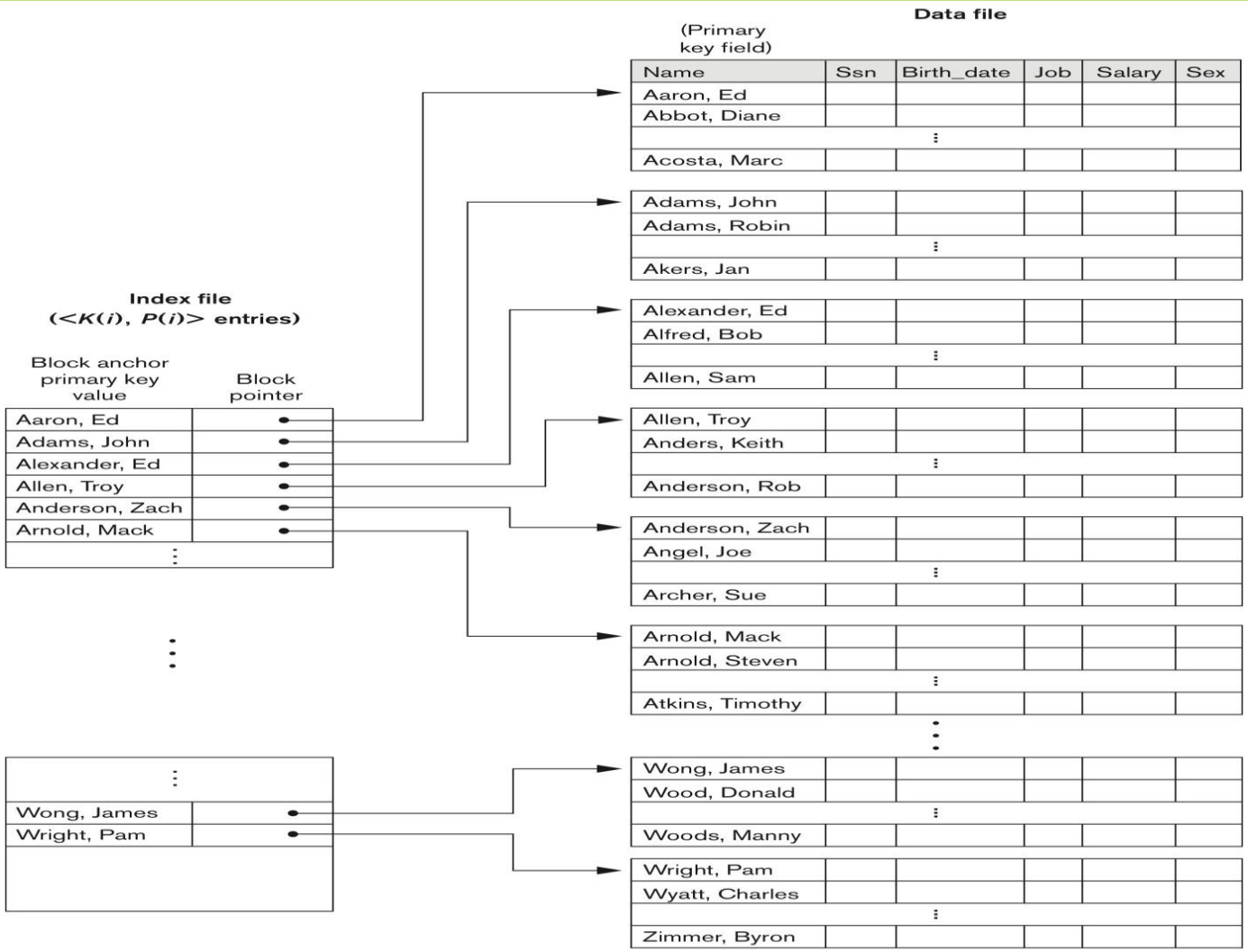
Chapter 17: Indexing Structures for Files and Physical Database Design: Outline

- Indexing Structures for Files and Physical Database design
- 1. Types of Single Level Ordered Indexes
- 2. Multilevel Indexes
- 3. Dynamic Multilevel Indexes Using B-Trees and B+-Trees

Indexing Structures for Files and Physical Database Design

- This chapter assumes that a file already exists with some primary organization such as unordered, ordered or indexed.
- We focus on additional auxillary access structures called:
  - indexes used to speed up retrieval of records when answering queries. (See Fig. 17.1)
- Index structures are:
  - additional files on disk which provide secondary alternative access paths to primary data file on disk.
  - They allow efficient access to records based on the indexing fields used to construct the index.
  - Any field or set of fields of the file can be used to construct an index.
  - A file can have multiple indexes on different fields.

Figure 17.1 Primary index on the ordering key field of the file shown in Figure 16.7.



Indexing Structures for Files and Physical Database Design

- These indexes use different data structures to speed up the search.
- To find a record that meets a condition on an index field,
  - the index is searched.
  - This leads to pointers to one or more disk blocks in the data file where the required records are located.
- The most common types of indexes are:
  - based on ordered files, and
  - use of tree data structures to organize the index.
  - Indexes can also be based on hashing or other search data structures.

Indexing Structures for Files and Physical Database Design

- Multi level tree structured indexes include:
  - indexed sequential access method (ISAM) which is a static structure,
  - B-trees and B+-trees which are data structures commonly used in DBMS to implement dynamically changing multi level indexes.
- B+ tree is the default structure for generating indexes on demand in most relational DBMSs.

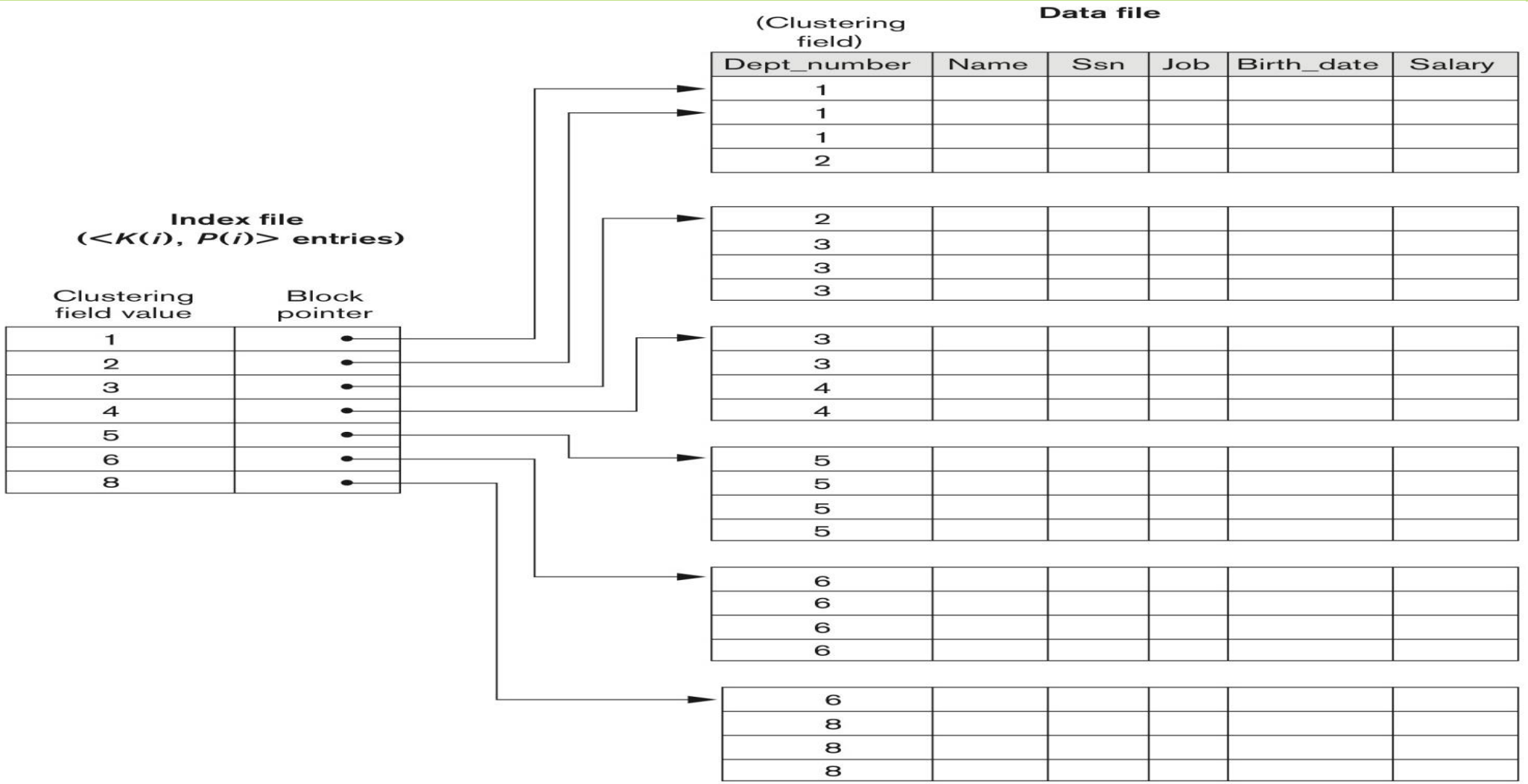
1. Types of single level ordered indexes

- For a file with a given record structure consisting of several fields,
  - an index access structure is usually defined on a single field called indexing field.
- The index normally 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 we can do a binary search on the index.
- If both the data file and the index file are ordered,
  - since the index file is usually much smaller than the data file searching the index using a binary search provides much faster data retrieval results.

1. Types of single level ordered indexes

- Thus, tree structured multilevel indexes:
  - extend the binary search idea and reduce the search space from two way partitioning at each search step to a n_ary partitioning.
- A primary index is specified on the ordering key field of the ordered file.
- A clustering index (e.g., Fig. 17.2) can be specified on a non-key clustering field if this field points to numerous records in the file that have the same value.
  - Example, with gpa field as clustered index, you can have an index value that points to physical record leading to all students with gpa >= 80
- An index file search uses the values of the search field to find a pointer to the data block containing desired record. However, an extendible hashing directory structure uses a hash value that is computed with a hash function that is based on the index key field (eg. Key mod 1000) to locate the block address.

Figure 17.2 A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file.



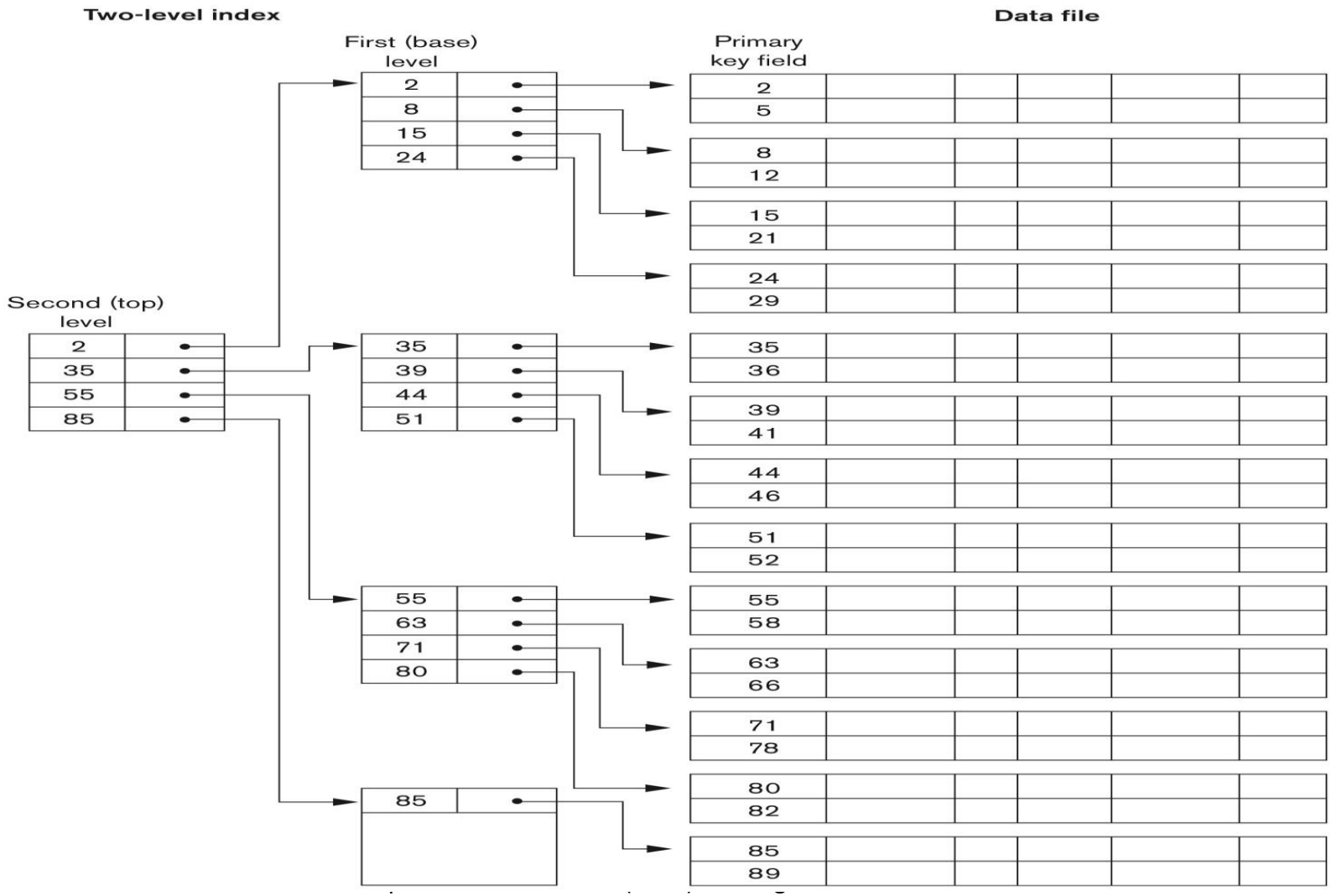
Index Entry

- 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 in the index's two field values.
- The two field values of index entry 'i' are:
  - <K(i),P(i)> where K(i) is the key value for entry i and P(i) is the pointer to disk block for record i.

2. Multilevel index

- For ordered indexes a binary search is applied to the index :
  - to locate pointers to a disk block in the file.
- A binary search requires about $\log_2 bi$ visits for an index file with bi blocks.
- With multilevel index (i) with blocking factor for the index (bfr_i),
  - that is larger than 2, the part of the index file searched at each step will be further reduced by bfr_i called fan out of the multi level index.
- The record search space is divided into two halves at each step during a binary search but divided n-ways where n= the fan out (fo) at each step with multi level index.
- Searching a multi level index requires about $\log_{fo} bi$ block access which is smaller than for binary search.

- **Figure 17.6** A two-level primary index resembling ISAM (indexed sequential access method) organization.

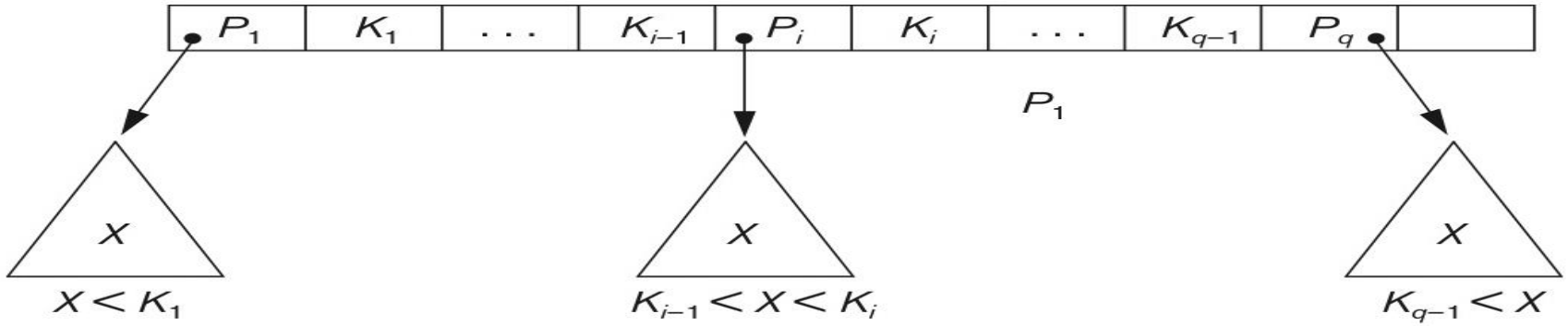


3. Dynamic Multilevel indexes using B-trees and B^+-trees

- 3.1 Search Trees and B-trees
- A search tree is a tree used to guide the search for a record given the value of one of the record's fields.
- The multilevel index is a variation of a search tree:
  - where each node in the multilevel index has as many as fo(fan out) pointers and fo key values (eg, Fig. 17.6). In Fig 17.6, the fo is 4 for 4 pointers.
  - The index field values in each node guide us to the next node until we reach the data file block that contains the required records. Eg. Get the record with id 29.
  - By following a pointer we restrict our search at each level to a subtree of the search tree. For example to answer the query of record with id 29, we need to bring in only 3 blocks rather than 13 blocks in order to find it.
- A search tree and B-tree (e.g., Fig 17.9 and 17.10) is slightly different from a multilevel index in that 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 $<P_1,K_1, P_2,K_2, \ldots, P_{q-1},K_{q-1}\ Pq>$ where $q<=p$.

3. Dynamic Multilevel indexes using B-trees and B^+-trees

- 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 search values are assumed unique and the following constraints should hold at all times in a search tree.
  - (1) Within each node $K_1 < K_2 < \ldots. K_{q-1}$
  - (2) For all values X in subtree pointed at by P_i, we have $K_{i-1} < X < K_i$ for $1<i<q$. (see Fig. 17.8)



- **Figure 17.8** A node in a search tree with pointers to subtrees below it.

3. Dynamic Multilevel indexes using B-trees and B^+-trees

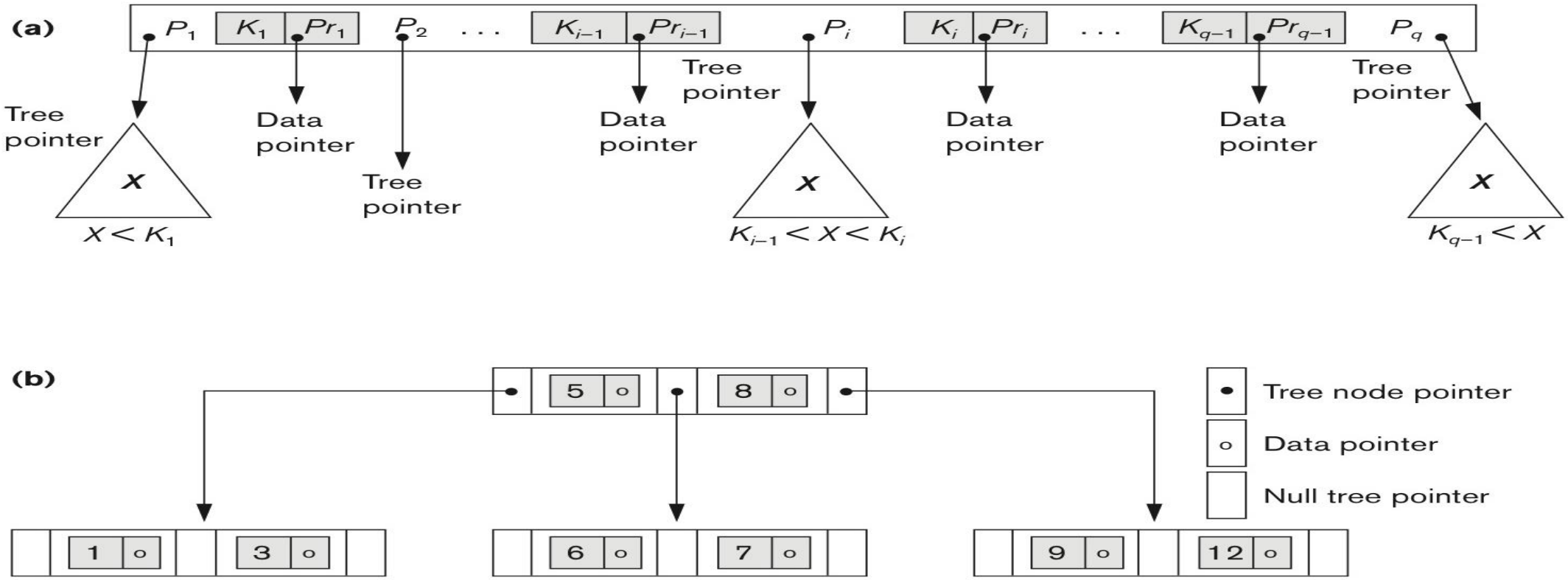
- To search for a value X, we follow the appropriate pointer P_i according to formula condition 2.
- Each key value in the tree is associated with a pointer to the record in the data file having that value.
- When a new record is inserted in the file,
  - we must update the search tree by inserting an entry in the tree containing the search field value of the new record and a pointer to the new record.
- A balanced tree (B-tree) has its leaf nodes at the same depth level and guarantees more evenly distributed nodes and search speed.
- B-tree has additional constraint to ensure that the tree is always balanced.

3. B-trees

- A B-tree of order p can be defined as having the following properties:
  - (1) Each internal node in B-tree (fig 17.10 a) is of the form $<P_1,<K_1,Pr_1>, P_2, <K_2,Pr_2>,\ldots\ldots, P_{q-1},<K_{q-1},Pr_{q-1}>,P_q>$ where $q<=p$.
    - Each P_i is a tree pointer (a pointer to another node in the B-tree), each Pr_i is a data record pointer (a pointer to the record whose search key field value is equal to K_i).
  - (2) Within each node $K_1<K_2<\ldots K_{q-1}$
  - (3) For all search key field value X in the subtree pointed at by P_i, we have $K_{i-1}<X<\ldots K_i$ for $1<i<q$
  - (4) Each node has at most p tree pointers
  - (5) Each node except the root and the leaf nodes has at least ceiling(p/2) tree pointers. The root has at least two tree pointers unless it is the only node in the tree.
  - (6) A node with q tree pointers $q<=p$, has q-1 search key field values (and hence has q-1 data pointers).
  - (7) 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.

3. B-trees

Figure 17.10 B-tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.



3. B-trees – Insertion and Deletion of values

- Fig 17.10(b) shows a B-tree of order p=3 (maximum number of pointers) and 2 (or P-1) key values at each node.
- Algorithms exist for loading , B-tree records, inserting and deleting records and retrieving records through B-tree. They all focus on meeting the requirements of the B-tree structure as defined in conditions 1 to 7 above.
- For example, insert the records with key values 8,5,1,7,3,12,9,6 in a B-tree of order 3
  - 1. Start building the tree from the root node at level 0.
  - 2. Once the root node is full with p-1 search key values, the root node splits into two nodes at level 1.
  - 3. Only the middle value is kept in the root node and the rest of the values are split evenly between the other two nodes.
  - 4. When a non root node is full and a new entry is inserted into it, that node is split into two nodes at the same level and the middle entry is moved to the parent node along with two pointers to the new split nodes.
  - 5. If the parent node is full, it is also split. Splitting can propagate all the way to the root node creating a new level if the root is split.

3. B-trees – Insertion and Deletion of values

- If deletion of a value causes a node to be less than half full,
  - it is combined with its neighboring nodes, and this can propagate all the way to the root.
- Each B-Tree node can have at most p tree pointers, p-1 data pointers and p-1 search key values (see Fig 17.10), Fig 17.12 and Fig. 17.13.

Figure 17.12 An example of insertion in a B+-tree with $p = 3$ and $p_{leaf} = 2$.

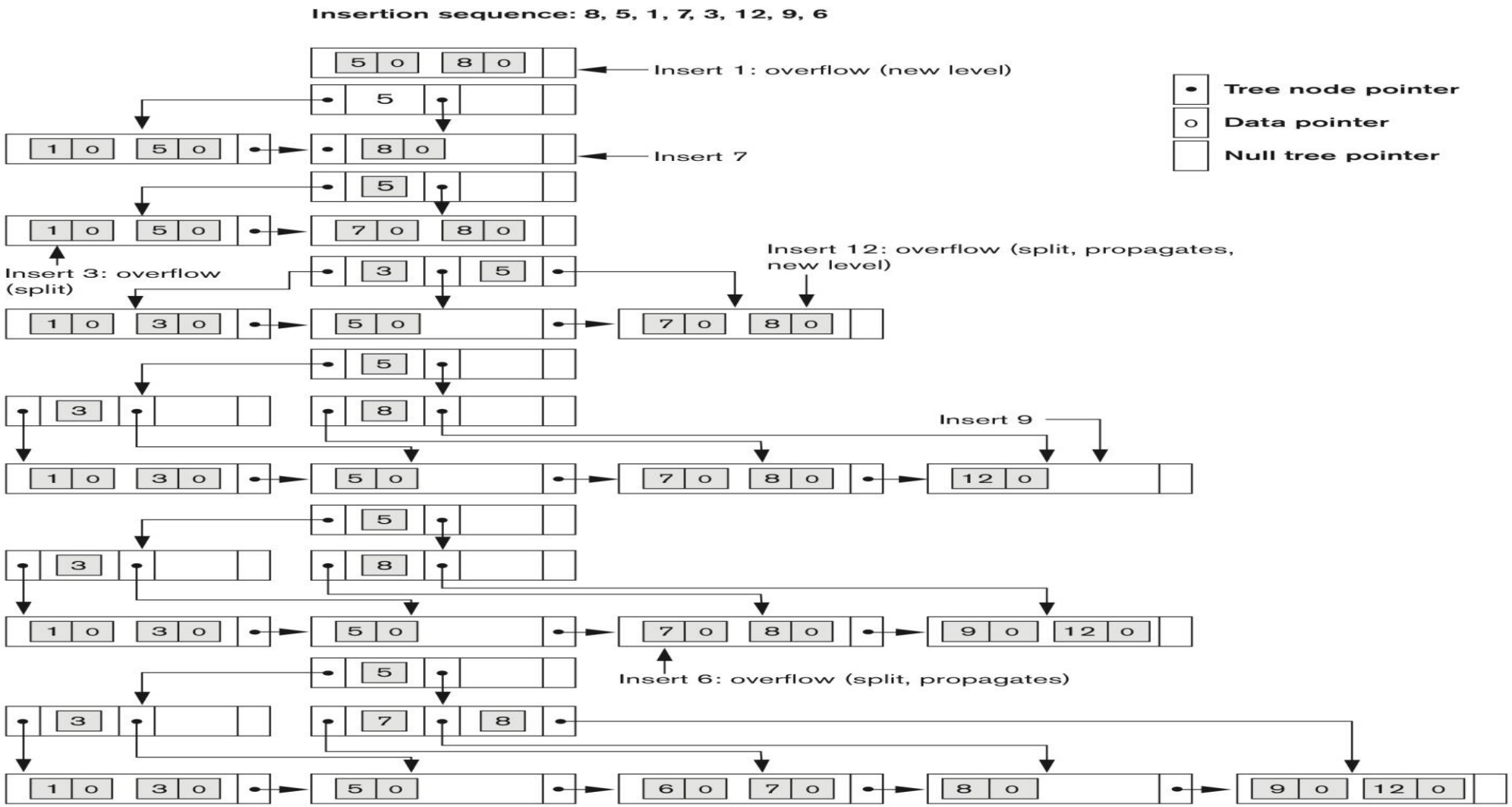
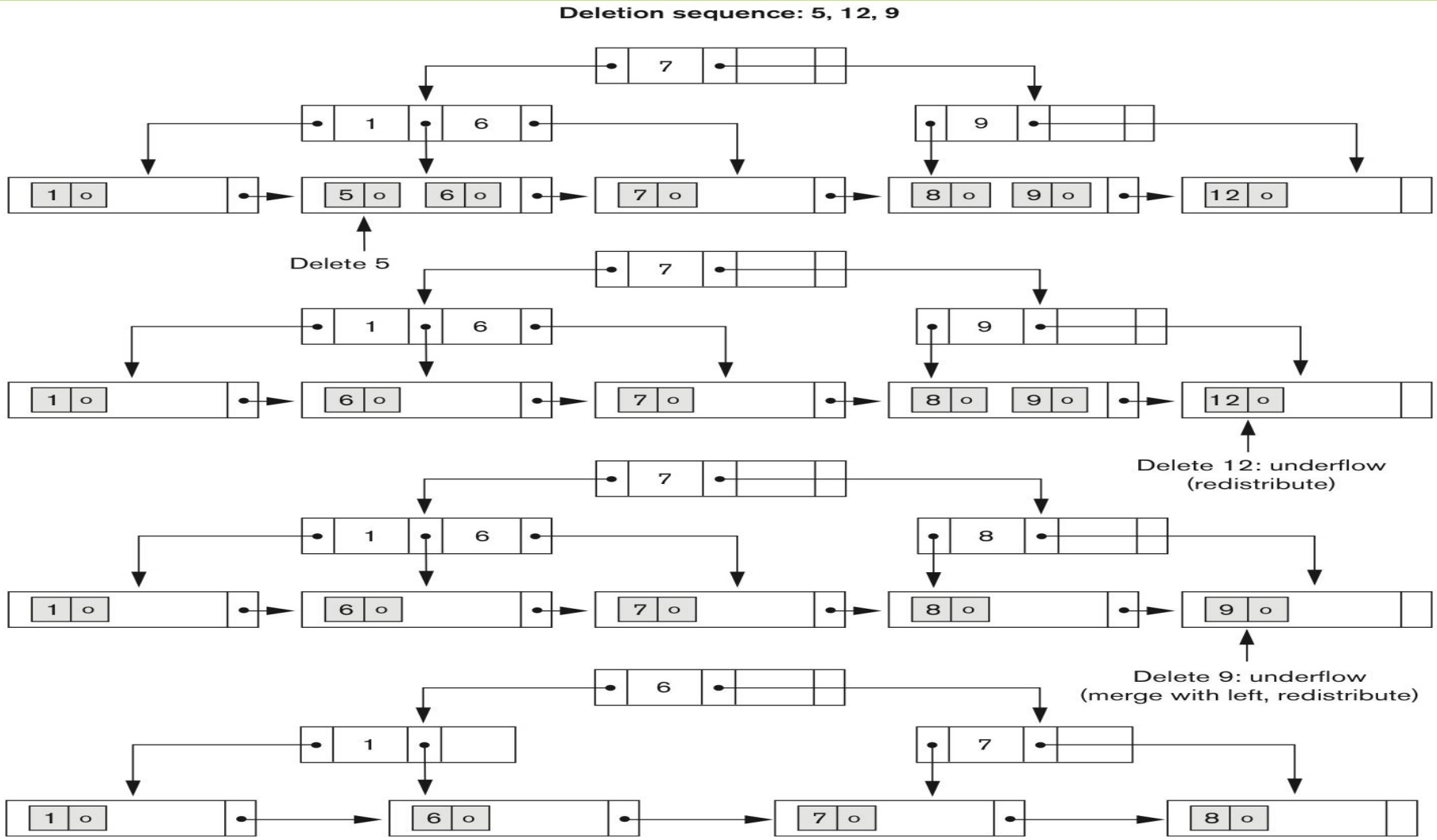


Figure 17.13 An example of deletion from a B^+-tree



3. B^+-tree

- B^+-tree is a variation of the B tree
- In a B-tree, every value of the search field appears once at some level in the tree along with a data pointer.
- In a B^+-tree, data pointers are stored only at the leaf nodes of the tree.
- Thus, the structure of the leaf nodes differ from those of the internal nodes.
- The leaf nodes have an entry for every value of the search field along with a data pointer to the record or onto block.
- The leaf nodes of the B^+-tree are usually linked to provide ordered access on the search field to the records.

3. B+ tree

- The structure of the internal node of the B^+-tree is:
  - (1) Each internal node is of the form $<P_1, K_1, P_2, K_2, \ldots, P_{q-1}, K_{q-1}, P_q>$ where $q<=p$ and each P_i is a tree pointer (see Fig 17.11).
  - (2) Within each internal node, $K_1<K_2<\ldots<K_{q-1}$
  - (3) For all search field values X in the subtree pointed at by P_i, we have $K_{i-1}<X<=K_i$ for $1<i<q$
  - (4) Each internal node has at most p tree pointers
  - (5) Each internal node except the root has at least ceiling(p/2) tree pointers. The root node has at least two tree pointers if it is an internal node.
  - (6) An internal node with q pointers $q<=p$ has q-1 search field values.

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

- 1. Each leaf node is of the form $<<K_1,Pr_1>,<K_2,Pr_2>,\ldots\ldots,<K_{q-1},Pr_{q-1}>,P_{next}>$ where $q<=p$, each Pr_i is a data pointer and P_{next} points to the next leaf node of the B^+-tree.
- 2. Within each leaf node, $K_1<= K_2<=K_{q-1}$ for $q<=p$
- 3. Each Pr_i is a data pointer that points to the record whose search field value is K_i to a file block containing the record (or block of records)
- 4. Each leaf node has at least ceiling(p/2) values
- 5. All leaf nodes are at the same level

- The pointers in internal nodes are tree pointers while those in leaf nodes are data pointers to data file records.
- Check Fig 17.12 for examples on insertions and Fig 17.13 on deletions from a B^+-tree.