Indexing Structures for Files and Physical Database Design

Introduction

- Indexes used to speed up record retrieval in response to certain search conditions
- Index structures provide secondary access paths
- Any field can be used to create an index
  - Multiple indexes can be constructed
- Most indexes based on ordered files
  - Tree data structures organize the index

17.1 Types of Single-Level Ordered Indexes

- Ordered index similar to index in a textbook
- Indexing field (attribute)
  - Index stores each value of the index field with list of pointers to all disk blocks that contain records with that field value
- Values in index are ordered
- Primary index
  - Specified on the ordering key field of ordered file of records

Types of Single-Level Ordered Indexes (cont'd.)

- Clustering index
  - Used if numerous records can have the same value for the ordering field
- Secondary index
  - Can be specified on any nonordering field
  - Data file can have several secondary indexes

Primary Indexes

- Ordered file with two fields
  - Primary key, *K(i)*
  - Pointer to a disk block, *P(i)*
- One index entry in the index file for each block in the data file
- Indexes may be dense or sparse
  - Dense index has an index entry for every search key value in the data file
  - Sparse index has entries for only some search values

Primary Indexes (cont'd.)

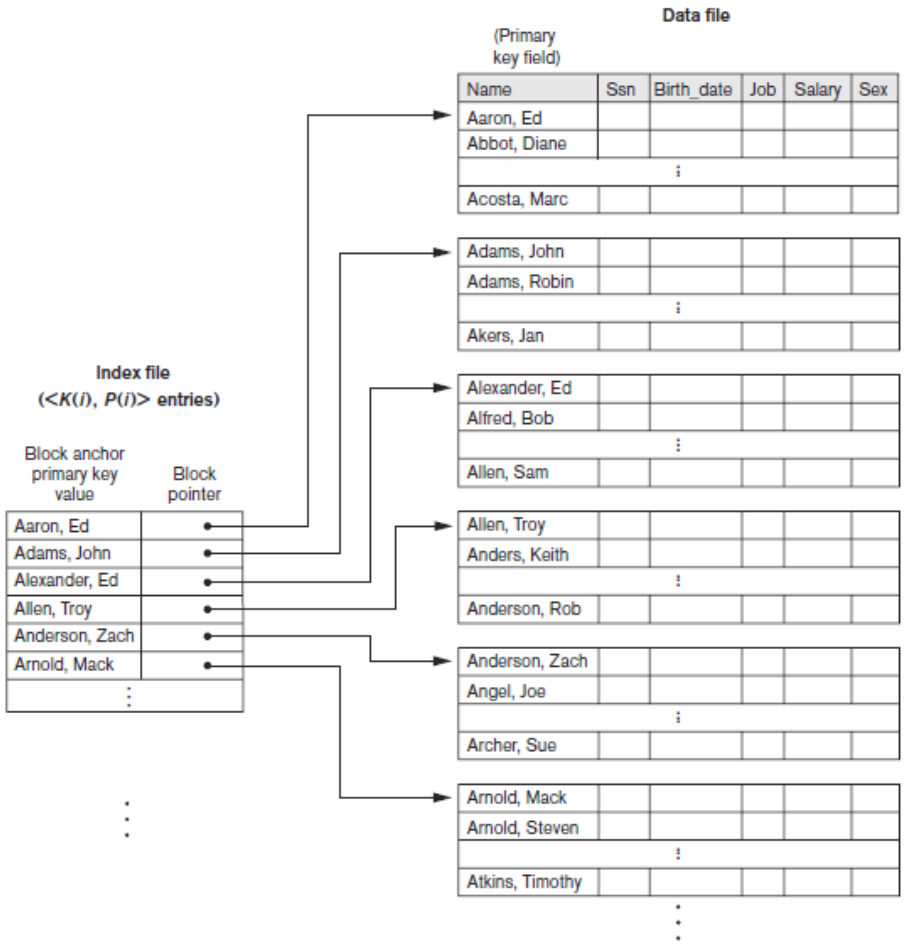


Figure 17.1 Primary index on the ordering key field of the file shown in Figure 16.7

Primary Indexes (cont'd.)

- Major problem: insertion and deletion of records
  - Move records around and change index values
  - Solutions
    - Use unordered overflow file
    - Use linked list of overflow records

Clustering Indexes

- Clustering field
  - File records are physically ordered on a nonkey field without a distinct value for each record
- Ordered file with two fields
  - Same type as clustering field
  - Disk block pointer

Clustering Indexes (cont'd.)

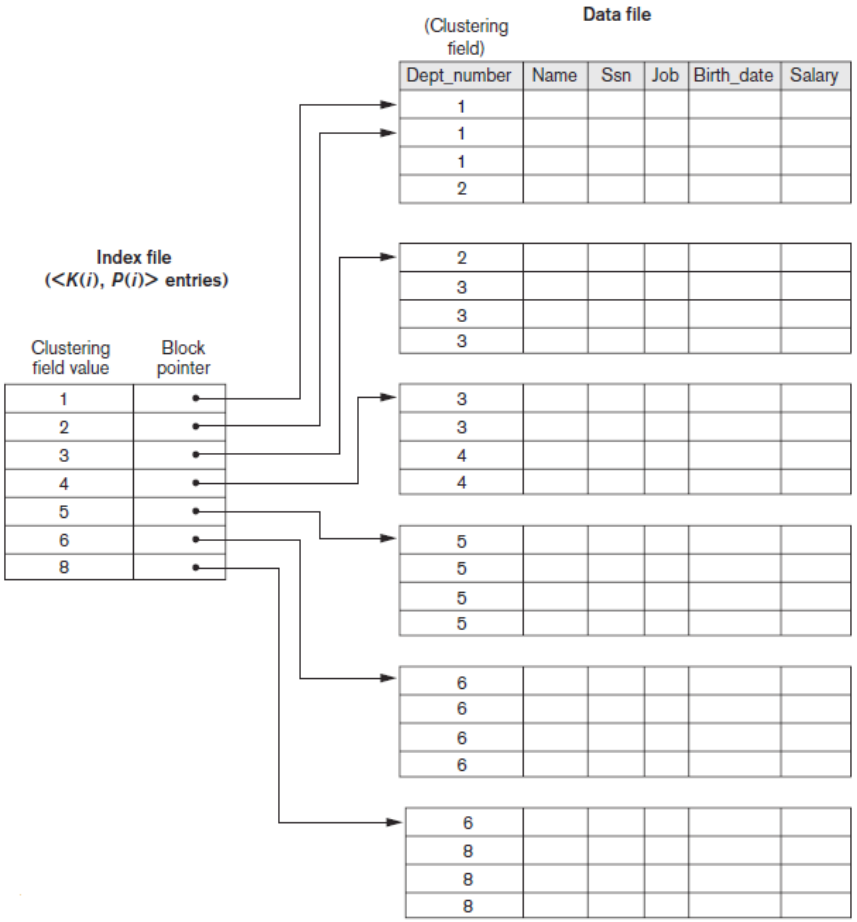


Figure 17.2 A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file

Secondary Indexes

- Provide secondary means of accessing a data file
  - Some primary access exists
- Ordered file with two fields
  - Indexing field, *K(i)*
  - Block pointer or record pointer, *P(i)*
- Usually need more storage space and longer search time than primary index
  - Improved search time for arbitrary record

Secondary Indexes (cont'd.)

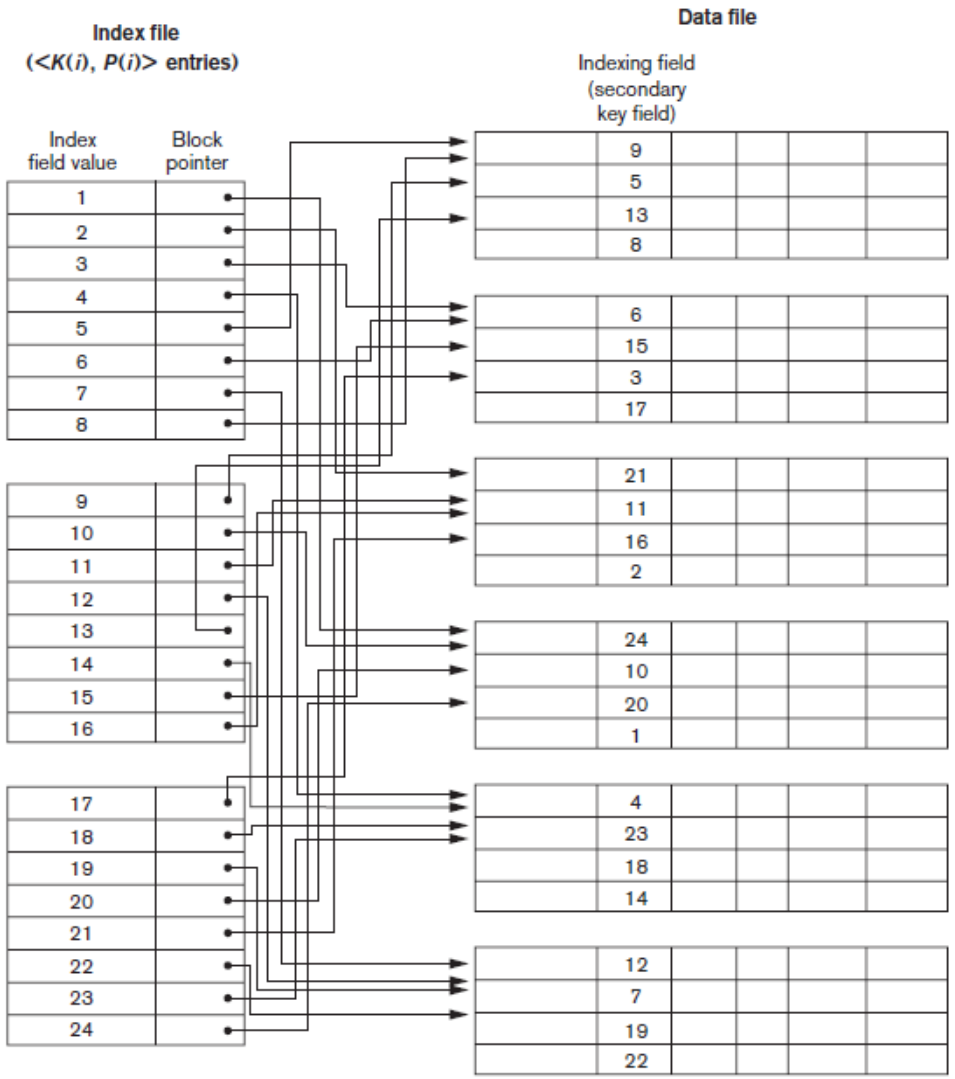


Figure 17.4 Dense secondary index (with block pointers) on a nonordering key field of a file.

Types of Single-Level Ordered Indexes (cont'd.)

	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

Table 17.1 Types of indexes based on the properties of the indexing field

Type of Index	Number of (First-Level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.
[b]For option 1.
[c]For options 2 and 3.

Table 17.2 Properties of index types

17.2 Multilevel Indexes

- Designed to greatly reduce remaining search space as search is conducted
- Index file
  - Considered first (or base level) of a multilevel index
- Second level
  - Primary index to the first level
- Third level
  - Primary index to the second level

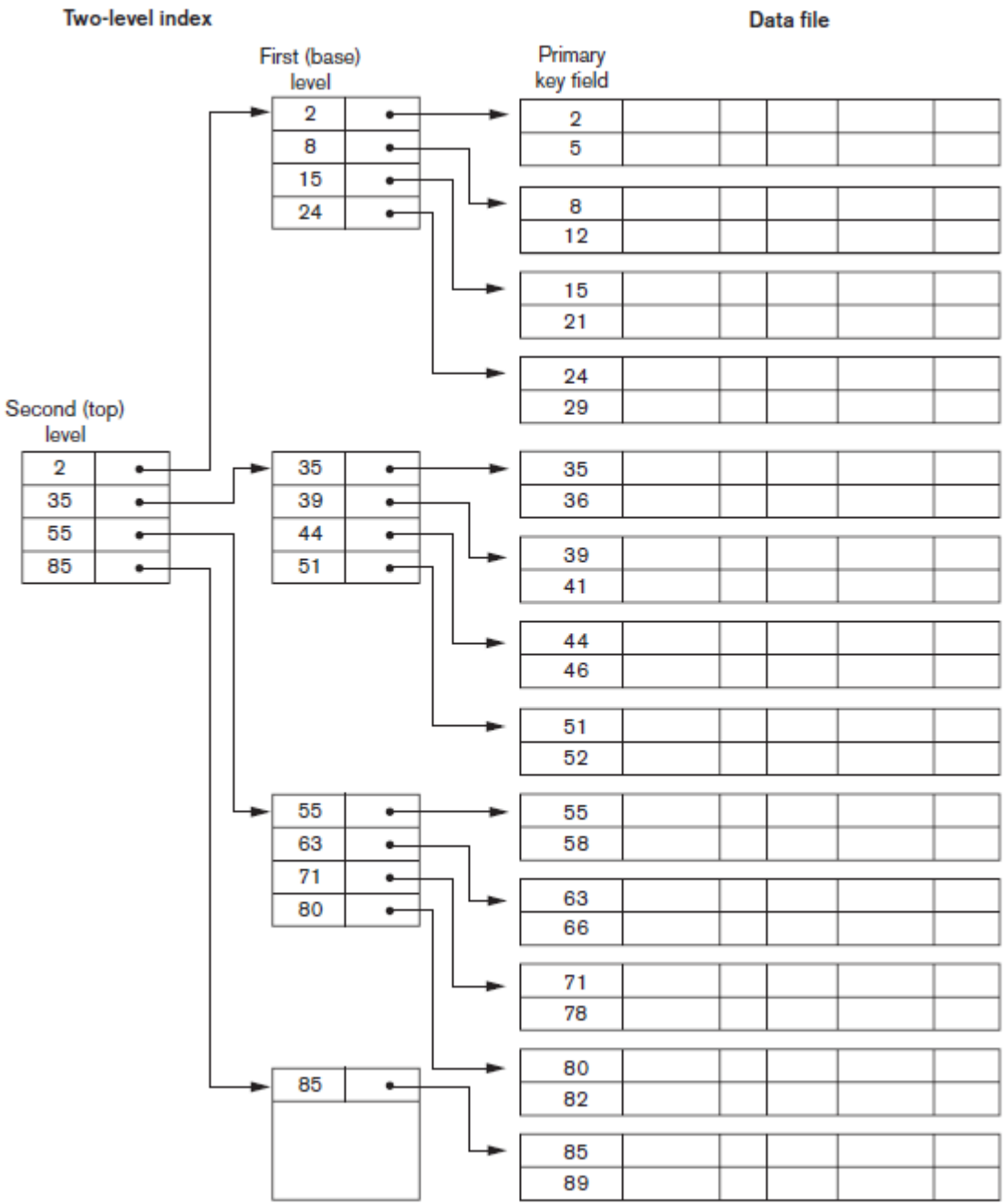
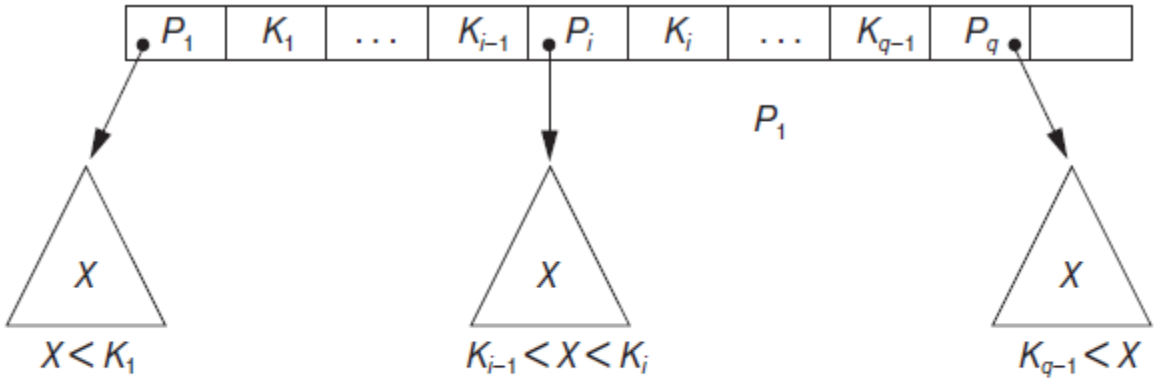


Figure 17.6 A two-level primary index resembling ISAM (indexed sequential access method) organization

Search Trees and B-Trees

- Search tree used to guide search for a record
  - Given value of one of record's fields



[8]This restriction can be relaxed. If the index is on a nonkey field, duplicate search values may exist and the node structure and the navigation rules for the tree may be modified.

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

Search Trees and B-Trees (cont'd.)

- Algorithms necessary for inserting and deleting search values into and from the tree

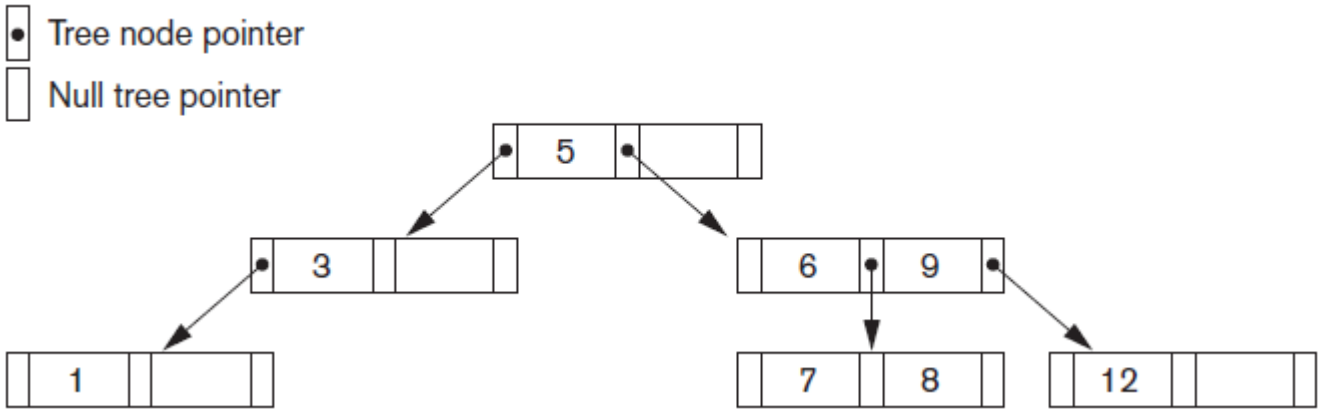


Figure 17.9 A search tree of order p = 3

B-Trees

- Provide multi-level access structure
- Tree is always balanced
- Space wasted by deletion never becomes excessive
  - Each node is at least half-full
- Each node in a B-tree of order *p* can have at most *p-1* search values

B-Tree Structures

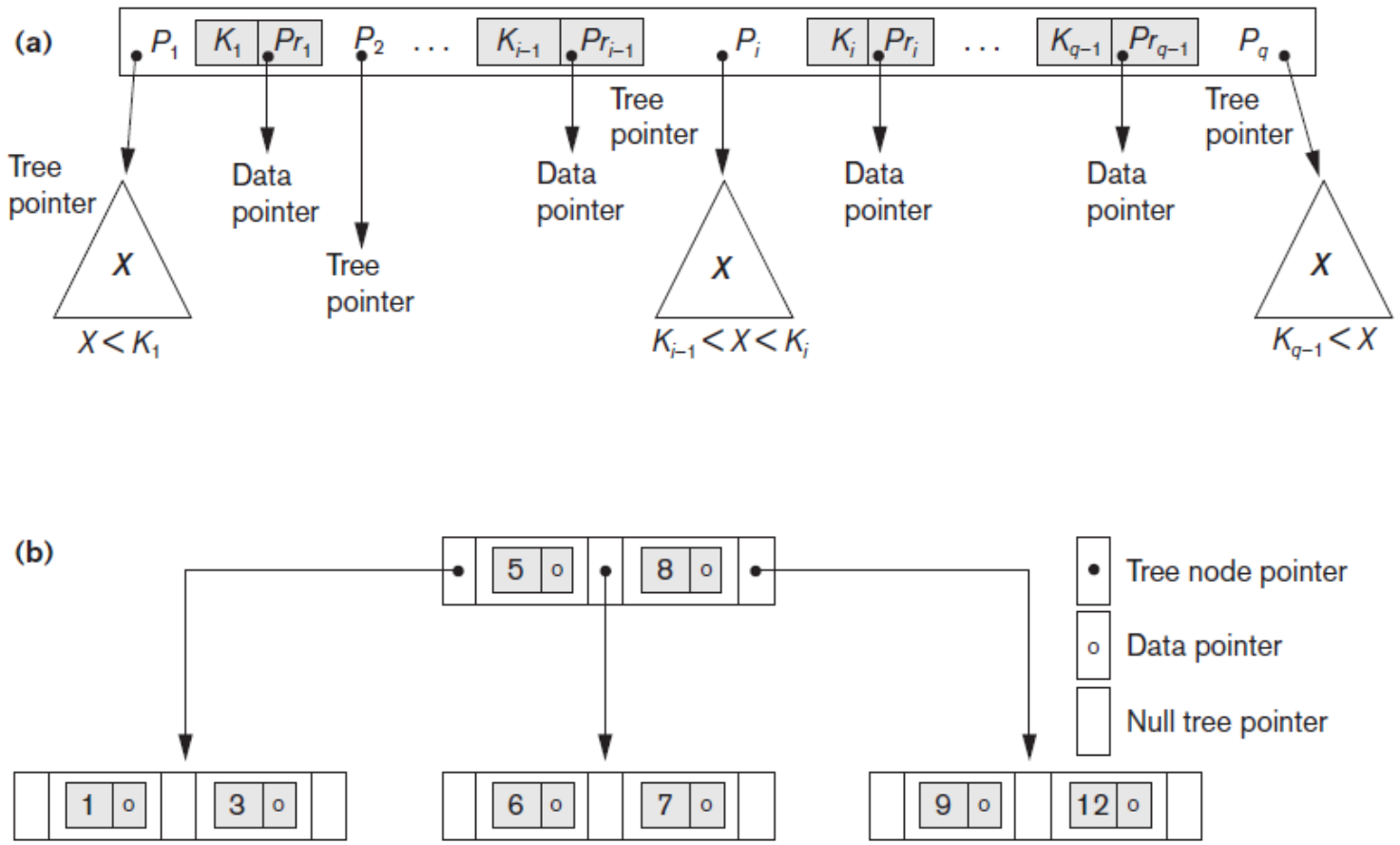


Figure 17.10 B-tree structures (a) A node in a B-tree with *q−1* search values (b) A B-tree of order *p=3.* The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6

B+ -Trees

- Data pointers stored only at the leaf nodes
  - Leaf nodes have an entry for every value of the search field, and a data pointer to the record if search field is a key field
  - For a nonkey search field, the pointer points to a block containing pointers to the data file records
- Internal nodes
  - Some search field values from the leaf nodes repeated to guide search

B+ -Trees (cont'd.)

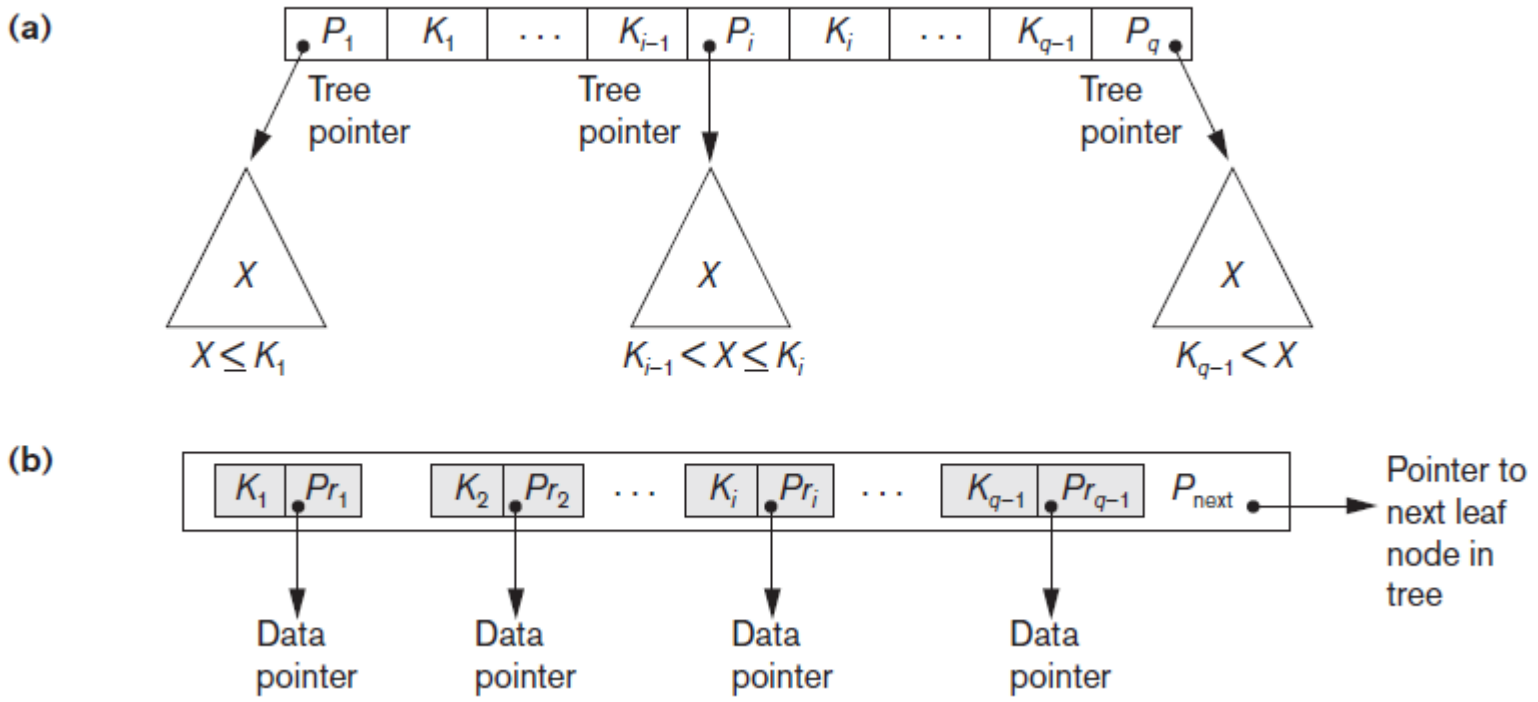


Figure 17.11 The nodes of a B+-tree (a) Internal node of a B+-tree with *q−1* search values (b) Leaf node of a B+-tree with *q−1* search values and *q−1* data pointers

17.4 Indexes on Multiple Keys

- Multiple attributes involved in many retrieval and update requests
- Composite keys
  - Access structure using key value that combines attributes
- Partitioned hashing
  - Suitable for equality comparisons

Indexes on Multiple Keys (cont'd.)

- Grid files
  - Array with one dimension for each search attribute

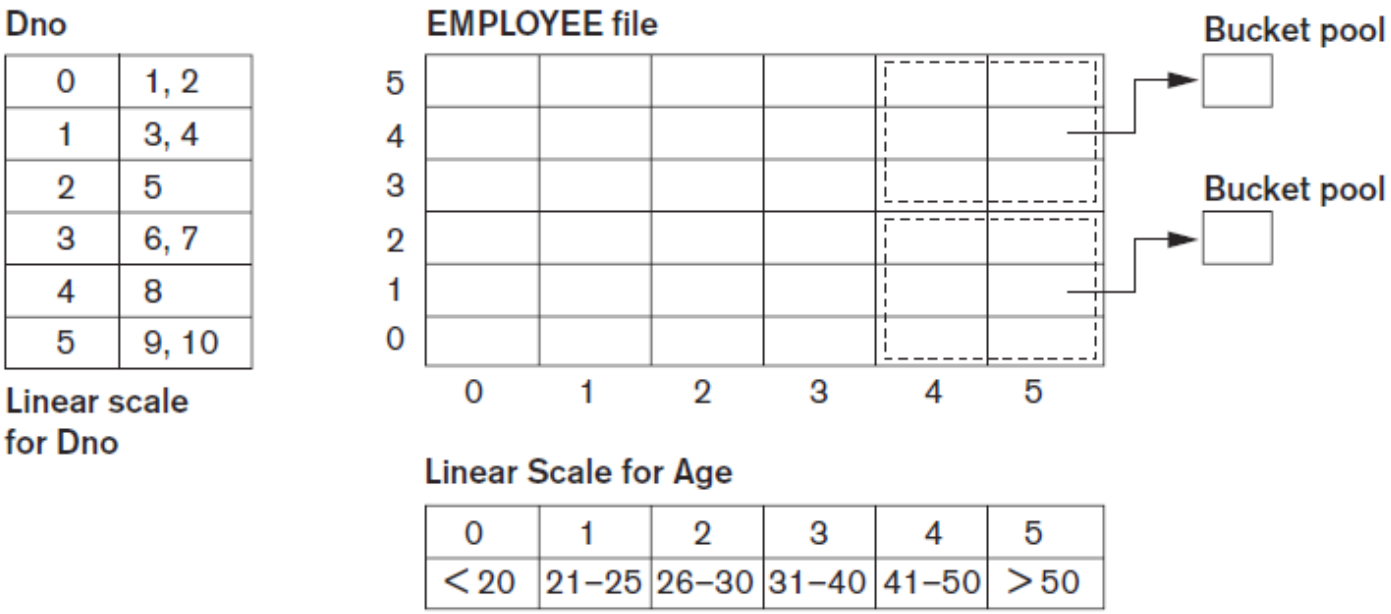


Figure 17.14 Example of a grid array on Dno and Age attributes

17.5 Other Types of Indexes

- Hash indexes
  - Secondary structure for file access
  - Uses hashing on a search key other than the one used for the primary data file organization
  - Index entries of form (K, P_r) or (K, P)
    - P_r: pointer to the record containing the key
    - P: pointer to the block containing the record for that key

Physical Database Design Decisions

- Design decisions about indexing
  - Whether to index an attribute
    - Attribute is a key or used by a query
  - What attribute(s) to index on
    - Single or multiple
  - Whether to set up a clustered index
    - One per table
  - Whether to use a hash index over a tree index
    - Hash indexes do not support range queries
  - Whether to use dynamic hashing
    - Appropriate for very volatile files

Summary

- Indexes are access structures that improve efficiency of record retrieval from a data file
- Ordered single-level index types
  - Primary, clustering, and secondary
- Multilevel indexes can be implemented as B-trees and B+ -trees
  - Dynamic structures
- Multiple key access methods
- Logical and physical indexes