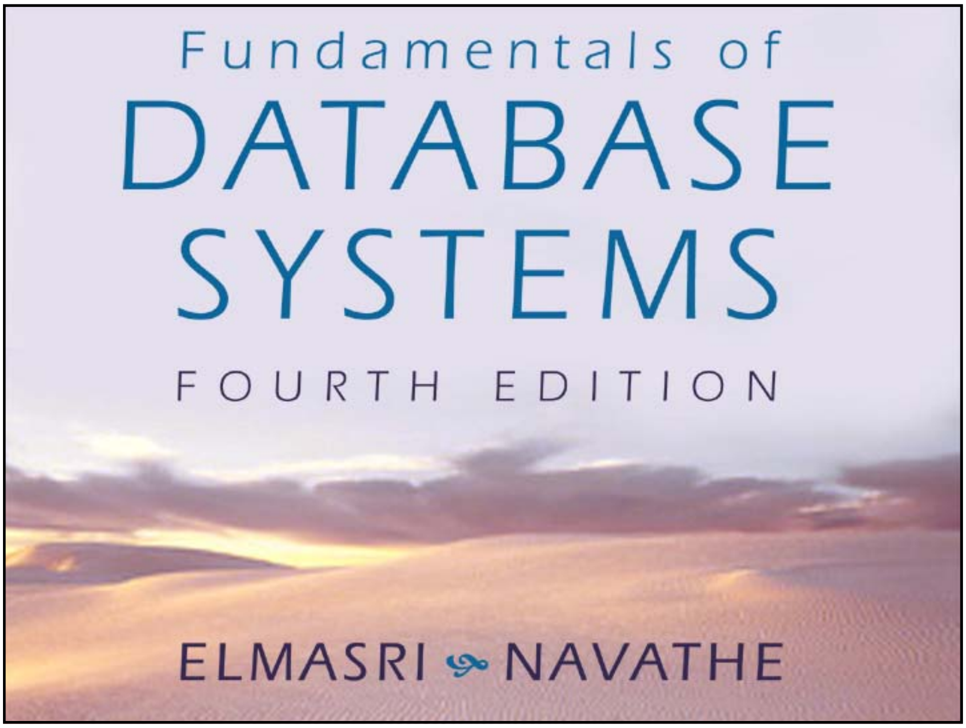

Chapter 14

Indexing Structures for Files

Chapter Outline

- Types of Single-level Ordered Indexes
  - Primary Indexes
  - Clustering Indexes
  - Secondary Indexes
- Multilevel Indexes
- Dynamic Multilevel Indexes Using B-Trees and B+-Trees
- Indexes on Multiple Keys

Indexes as Access Paths

- A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- The index is usually specified on one field of the file (although it could be specified on several fields)
- One form of an index is a file of entries **<field value, pointer to record>,** which is ordered by field value
- The index is called an *access path* on the field.

Indexes as Access Paths (contd.)

- The index file usually occupies considerably less disk blocks than the data file because its entries are much smaller
- A binary search on the index yields a pointer to the file record
- 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**, on the other hand, has index entries for only some of the search values

Indexes as Access Paths (contd.)

Example: Given the following data file:
EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ...)
Suppose that:
record size R=150 bytes
block size B=512 bytes
r=30000 records

Then, we get:
blocking factor Bfr= B div R= 512 div 150= 3 records/block
number of file blocks b= (r/Bfr)= (30000/3)= 10000 blocks

For an index on the SSN field, assume the field size V_{SSN}=9 bytes,
assume the record pointer size P_R=7 bytes. Then:
index entry size $R_I=(V_{SSN}+ P_R)=(9+7)=16$ bytes
index blocking factor Bfr_I= B div R_I= 512 div 16= 32 entries/block
number of index blocks b= (r/ Bfr_I)= (30000/32)= 938 blocks
binary search needs log2bI= log2938= 10 block accesses

This is compared to an average linear search cost of:
(b/2)= 30000/2= 15000 block accesses
If the file records are ordered, the binary search cost would be:
$\log_2 b = \log_2 30000 = 15$ block accesses

Types of Single-Level Indexes

- **Primary Index**
  - Defined on an ordered data file
  - The data file is ordered on a *key field*
  - Includes one index entry *for each block* in the data file; the index entry has the key field value for the *first record* in the block, which is called the *block anchor*
  - A similar scheme can use the *last record* in a block.
  - A primary index is a nondense (sparse) index, since it includes an entry for each disk block of the data file and the keys of its anchor record rather than for every search value.

- INSERT FIGURE 14.1

Types of Single-Level Indexes

- **Clustering Index**
  - Defined on an ordered data file
  - The data file is ordered on a *non-key field* unlike primary index, which requires that the ordering field of the data file have a distinct value for each record.
  - Includes one index entry *for each distinct value* of the field; the index entry points to the first data block that contains records with that field value.
  - It is another example of *nondense* index where Insertion and Deletion is relatively straightforward with a clustering index.

- INSERT FIGURE 14.2

- INSERT FIGURE 14.3

Types of Single-Level Indexes

- **Secondary Index**
  - A secondary index provides a secondary means of accessing a file for which some primary access already exists.
  - The secondary index may be on a field which is a candidate key and has a unique value in every record, or a nonkey with duplicate values.
  - The index is 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 (and hence, indexing fields) for the same file.
  - Includes one entry *for each record* in the data file; hence, it is a *dense index*

- INSERT FIGURE 14.4

- INSERT FIGURE 14.5

- INSERT TABLE 14.2

Multi-Level Indexes

- Because a single-level index is an ordered file, we can create a primary index *to the index itself* ; in this case, the original index file is called the *first-level index* and the index to the index is called the *second-level index.*
- We can repeat the process, creating a third, fourth, ..., top level until all entries of the *top level* fit in one disk block
- A multi-level index can be created for any type of first-level index (primary, secondary, clustering) as long as the first-level index consists of *more than one* disk block

- INSERT FIGURE 14.6

Multi-Level Indexes

- Such a multi-level index is a form of *search tree* ; however, insertion and deletion of new index entries is a severe problem because every level of the index is an *ordered file.*

- INSERT FIGURE 14.8

- INSERT FIGURE 14.9

Dynamic Multilevel Indexes Using B-Trees and B+-Trees

- Because of the insertion and deletion problem, most multi-level indexes use B-tree or B+-tree data structures, which leave space in each tree node (disk block) to allow for new index entries
- These data structures are variations of search trees that allow efficient insertion and deletion of new search values.
- In B-Tree and B+-Tree data structures, each node corresponds to a disk block
- Each node is kept between half-full and completely full

Dynamic Multilevel Indexes Using B-Trees and B+-Trees (contd.)

- An insertion into a node that is not full is quite efficient; if a node is full the insertion causes a split into two nodes
- Splitting may propagate to other tree levels
- 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 neighboring nodes

Difference between B-tree and B+-tree

- In a B-tree, pointers to data records exist at all levels of the tree
- In a B+-tree, all pointers to data records exists at the leaf-level nodes
- A B+-tree can have less levels (or higher capacity of search values) than the corresponding B-tree

- INSERT FIGURE 14.10

- INSERT FIGURE 14.11

- INSERT FIGURE 14.12

- INSERT FIGURE 14.13