CHAPTER 16

Disk Storage, Basic File Structures, Hashing, and Modern Storage Architectures

1

16.1 Introduction

- Databases typically stored on magnetic disks
  - Accessed using physical database file structures
- Storage hierarchy
  - Primary storage
    - CPU main memory, cache memory
  - Secondary storage
    - Magnetic disks, flash memory, solid-state drives
  - Tertiary storage
    - Removable media

2

Memory Hierarchies and Storage Devices

- Cache memory
  - Static RAM
  - DRAM
- Mass storage
  - Magnetic disks
    - CD-ROM, DVD, tape drives
- Flash memory
  - Nonvolatile

3

Storage Types and Characteristics

Type	Capacity*	Access Time	Max Bandwidth	Commodity Prices (2014)**
Main Memory- RAM	4GB–1TB	30ns	35GB/sec	$100–$20K
Flash Memory- SSD	64 GB–1TB	50µs	750MB/sec	$50–$600
Flash Memory- USB stick	4GB–512GB	100µs	50MB/sec	$2–$200
Magnetic Disk	400 GB–8TB	10ms	200MB/sec	$70–$500
Optical Storage	50GB–100GB	180ms	72MB/sec	$100
Magnetic Tape	2.5TB–8.5TB	10s–80s	40–250MB/sec	$2.5K–$30K
Tape jukebox	25TB–2,100,000TB	10s–80s	250MB/sec–1.2PB/sec	$3K–$1M+

*Capacities are based on commercially available popular units in 2014.

**Costs are based on commodity online marketplaces.

Table 16.1 Types of Storage with Capacity, Access Time, Max Bandwidth (Transfer Speed), and Commodity Cost

4

Storage Organization of Databases

Slide 16- 5

- Persistent data
  - Most databases
- Transient data
  - Exists only during program execution
- File organization
  - Determines how records are physically placed on the disk
  - Determines how records are accessed

5

16.2 Secondary Storage Devices

Slide 16- 6

- Hard disk drive
- Bits (ones and zeros)
  - Grouped into bytes or characters
- Disk capacity measures storage size
- Disks may be single or double-sided
- Concentric circles called tracks
  - Tracks divided into blocks or sectors
- Disk packs
  - Cylinder

6

Single-Sided Disk and Disk Pack

Slide 16-7

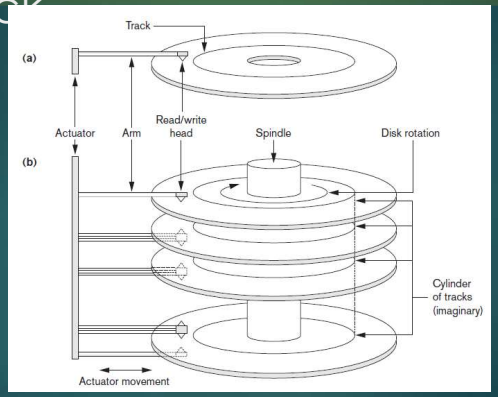

Figure 16.1 (a) A single-sided disk with read/write hardware (b) A disk pack with read/write hardware

7

Sectors on a Disk

Slide 16-8

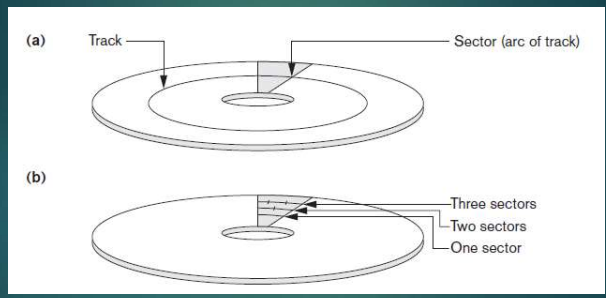

Figure 16.2 Different sector organizations on disk (a) Sectors subtending a fixed angle (b) Sectors maintaining a uniform recording density

8

Secondary Storage Devices (cont'd.)

Slide 16-9

- Formatting
  - Divides tracks into equal-sized disk blocks
  - Blocks separated by interblock gaps
- Data transfer in units of disk blocks
  - Hardware address supplied to disk I/O hardware
- Buffer
  - Used in read and write operations
- Read/write head
  - Hardware mechanism for read and write operations

9

Secondary Storage Devices (cont'd.)

Slide 16-10

- Disk controller
  - Interfaces disk drive to computer system
  - Standard interfaces
    - SCSI
    - SATA
    - SAS

10

Secondary Storage Devices (cont'd.)

Slide 16-11

- Techniques for efficient data access
  - Data buffering
  - Proper organization of data on disk
  - Reading data ahead of request
  - Proper scheduling of I/O requests
  - Use of log disks to temporarily hold writes
  - Use of SSDs or flash memory for recovery purposes

11

Solid State Device Storage

Slide 16-12

- Sometimes called flash storage
- Main component: controller
- Set of interconnected flash memory cards
- No moving parts
- Data less likely to be fragmented
- More costly than HDDs
- DRAM-based SSDs available
  - Faster access times compared with flash

12

Magnetic Tape Storage Devices

Slide 16-13

- Sequential access
  - Must scan preceding blocks
- Tape is mounted and scanned until required block is under read/write head
- Important functions
  - Backup
  - Archive

13

16.3 Buffering of Blocks

Slide 16-14

- Buffering most useful when processes can run concurrently in parallel

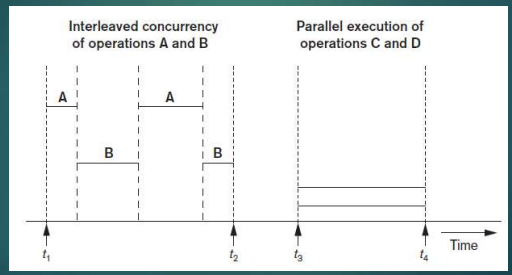

Figure 16.3 Interleaved concurrency versus parallel execution

14

Buffering of Blocks (cont'd.)

Slide 16-15

- Double buffering can be used to read continuous stream of blocks

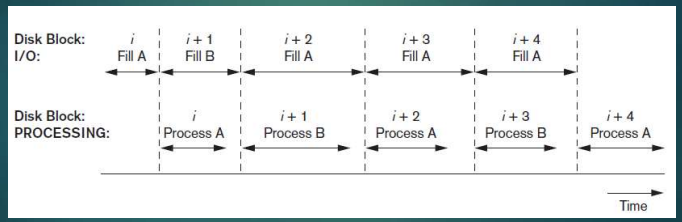

Figure 16.4 Use of two buffers, A and B, for reading from disk

15

Buffer Management and Replacement Strategies

Slide 16-16

- Buffer management information
  - Pin count
  - Dirty bit
- Buffer replacement strategies
  - Least recently used (LRU)
  - Clock policy
  - First-in-first-out (FIFO)

16

16.4 Placing File Records on Disk

Slide 16-17

- Record: collection of related data values or items
  - Values correspond to record field
- Data types
  - Numeric
  - String
  - Boolean
  - Date/time
- Binary large objects (BLOBs)
  - Unstructured objects

17

Placing File Records on Disk (cont'd.)

Slide 16-18

- Reasons for variable-length records
  - One or more fields have variable length
  - One or more fields are repeating
  - One or more fields are optional
  - File contains records of different types

18

Record Blocking and Spanned Versus Unspanned Records

Slide 16-19

- File records allocated to disk blocks
- Spanned records
  - Larger than a single block
  - Pointer at end of first block points to block containing remainder of record
- Unspanned
  - Records not allowed to cross block boundaries

19

Record Blocking and Spanned Versus Unspanned Records (cont'd.)

Slide 16-20

- Blocking factor
  - Average number of records per block for the file

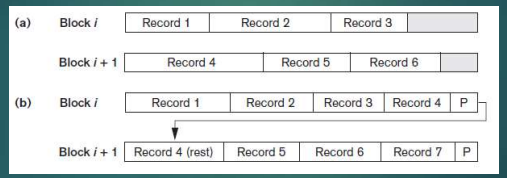

Figure 16.6 Types of record organization (a) Unspanned (b) Spanned

20

Record Blocking and Spanned Versus Unspanned Records (cont'd.)

Slide 16-21

- Allocating file blocks on disk
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation
- File header (file descriptor)
  - Contains file information needed by system programs
    - Disk addresses
    - Format descriptions

21

16.5 Operations on Files

Slide 16-22

- Retrieval operations
  - No change to file data
- Update operations
  - File change by insertion, deletion, or modification
- Records selected based on selection condition

22

Operations on Files (cont'd.)

Slide 16-23

- Examples of operations for accessing file records
  - Open
  - Find
  - Read
  - FindNext
  - Delete
  - Insert
  - Close
  - Scan

23

16.6 Files of Unordered Records (Heap Files)

Slide 16-24

- Heap (or pile) file
  - Records placed in file in order of insertion
- Inserting a new record is very efficient
- Searching for a record requires linear search
- Deletion techniques
  - Rewrite the block
  - Use deletion marker

24

16.7 Files of Ordered Records (Sorted Files)

Slide 16-25

- Ordered (sequential) file
  - Records sorted by ordering field
    - Called ordering key if ordering field is a key field
- Advantages
  - Reading records in order of ordering key value is extremely efficient
  - Finding next record
  - Binary search technique on block

25

	Name	Ssn	Birth_date	Job	Salary	Sex
Block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
Block 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
Block 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					

ordered (sequential) file of EMPLOYEE records with Name as the ordering key field

26

Access Times for Various File Organizations

Slide 16-27

Type of Organization	Access/Search Method	Average Blocks to Access a Specific Record
Heap (unordered)	Sequential scan (linear search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary search	$\log_2 b$

Table 16.3 Average access times for a file of b blocks under basic file organizations

27

16.8 Hashing Techniques

Slide 16-28

- Hash function (randomizing function)
  - Applied to hash field value of a record
  - Yields address of the disk block of stored record
- Organization called hash file
  - Search condition is equality condition on the hash field
  - Hash field typically key field
- Hashing also internal search structure
  - Used when group of records accessed exclusively by one field value

28

Hashing Techniques (cont'd.)

Slide 16-29

- Internal hashing
  - Hash table

Common hash function
h(K) = K mod M

	Name	Ssn	Job	Salary
0				
1				
2				
3				
	⋮			
$M-2$				
$M-1$				

29

Hashing Techniques (cont'd.)

Slide 16-30

- Collision
  - Hash field value for inserted record hashes to address already containing a different record
- Collision resolution
  - Open addressing
  - Chaining
  - Multiple hashing

30

Hashing Techniques (cont'd.)

Slide 16-31

- External hashing for disk files
  - Target address space made of buckets
  - Bucket: one disk block or contiguous blocks
- Hashing function maps a key into relative bucket
  - Table in file header converts bucket number to disk block address
- Collision problem less severe with buckets
- Static hashing
  - Fixed number of buckets allocated

31

Hashing Techniques (cont'd.)

Slide 16-32

- Hashing techniques that allow dynamic file expansion
  - Extendible hashing
    - File performance does not degrade as file grows
  - Dynamic hashing
    - Maintains tree-structured directory
  - Linear hashing
    - Allows hash file to expand and shrink buckets without needing a directory

32

16.9 Other Primary File Organizations

Slide 16-33

- Files of mixed records
  - Relationships implemented by logical field references
  - Physical clustering
- B-tree data structure
- Column-based data storage

33

16.10 Parallelizing Disk Access Using RAID Technology

Slide 16-34

- Redundant arrays of independent disks (RAID)
  - Goal: improve disk speed and access time
- Set of RAID architectures (0 through 6)
- Data striping
  - Bit-level striping
  - Block-level striping
- Improving Performance with RAID
  - Data striping achieves higher transfer rates

34

Parallelizing Disk Access Using RAID Technology (cont'd.)

Slide 16-35

- Improving reliability with RAID
  - Redundancy techniques: mirroring and shadowing
- RAID organizations and levels
  - Level 0
    - Data striping, no redundant data
    - Spits data evenly across two or more disks
  - Level 1
    - Uses mirrored disks

35

Parallelizing Disk Access Using RAID Technology (cont'd.)

Slide 16-36

- RAID organizations and levels (cont'd.)
  - Level 2
    - Hamming codes for memory-style redundancy
    - Error detection and correction
  - Level 3
    - Single parity disk relying on disk controller
  - Levels 4 and 5
    - Block-level data striping
    - Data distribution across all disks (level 5)

36

Parallelizing Disk Access Using RAID Technology (cont'd.)

Slide 16-37

- RAID organizations and levels (cont'd.)
  - Level 6
    - Applies P+Q redundancy scheme
    - Protects against up to two disk failures by using just two redundant disks
- Rebuilding easiest for RAID level 1
  - Other levels require reconstruction by reading multiple disks
- RAID levels 3 and 5 preferred for large volume storage

37

RAID Levels

Slide 16-38

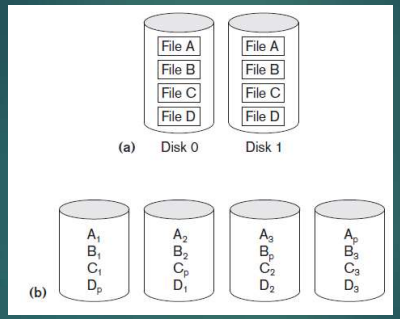

Figure 16.14 Some popular levels of RAID (a) RAID level 1: Mirroring of data on two disks (b) RAID level 5: Striping of data with distributed parity across four disks

38

16.11 Modern Storage Architectures

Slide 16-39

- Storage area networks
  - Online storage peripherals configured as nodes on high-speed network
- Network-attached storage
  - Servers used for file sharing
  - High degree of scalability, reliability, flexibility, performance
- iSCSI
  - Clients send SCSI commands to SCSI storage devices on remote channels

39

Modern Storage Architectures (cont'd.)

Slide 16-40

- Fibre Channel over IP (FCIP)
  - Fibre Channel control codes and data translated into IP packets
  - Transmitted between geographically distant Fibre Channel SANs
- Fibre Channel over Ethernet (FCoE)
  - Similar to iSCSI without the IP

40

Modern Storage Architectures (cont'd.)

Slide 16-41

- Automated storage tiering
  - Automatically moves data between different storage types depending on need
    - Frequently-used data moved to solid-state drives
- Object-based storage
  - Data managed in form of objects rather than files made of blocks
  - Objects carry metadata and global identifier
  - Ideally suited for scalable storage of unstructured data

41

16.12 Summary

Slide 16-42

- Magnetic disks
  - Accessing a disk block is expensive
- Commands for accessing file records
- File organizations: unordered, ordered, hashed
- RAID
- Modern storage trends

42

CHAPTER 17

Indexing Structures for Files and Physical Database Design

43

Introduction

Slide 17-44

- 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

44

17.1 Types of Single-Level Ordered Indexes

Slide 17-45

- 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

45

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

Slide 17-46

- 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

46

Primary Indexes

Slide 17-47

- 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

47

Primary Indexes (cont'd.)

Slide 17-48

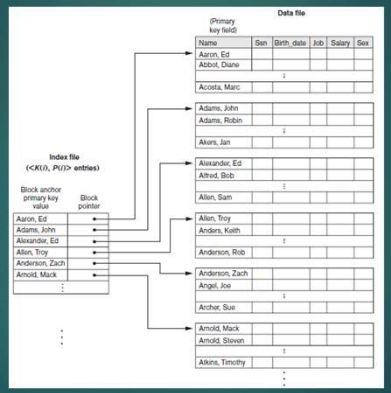

Figure 17.1 Primary index on the ordering key field of the file shown in Figure 16.7

48

Primary Indexes (cont'd.)

Slide 17-49

- 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

49

Clustering Indexes

Slide 17-50

- 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

50

Clustering Indexes (cont'd.)

Slide 17-51

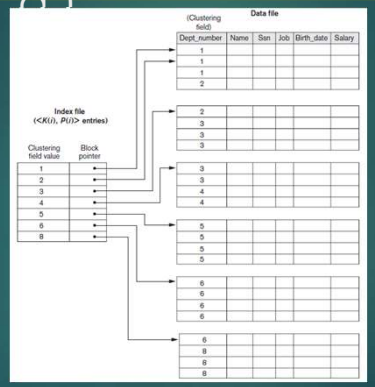

Figure 17.2 A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file

51

Secondary Indexes

Slide 17-52

- 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

52

Secondary Indexes (cont'd.)

Slide 17-53

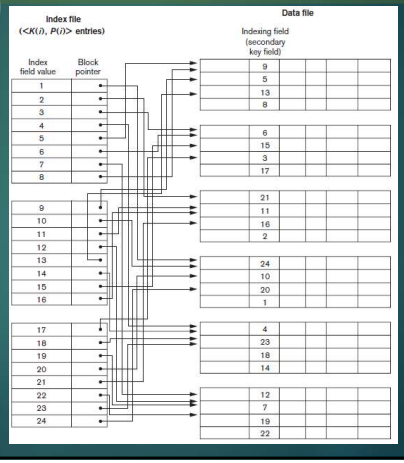

Figure 17.4 Dense secondary index (with block pointers) on a nonordering key field of a file.

53

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

Slide 17-54

	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

54

17.2 Multilevel Indexes

Slide 17-55

- 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

55

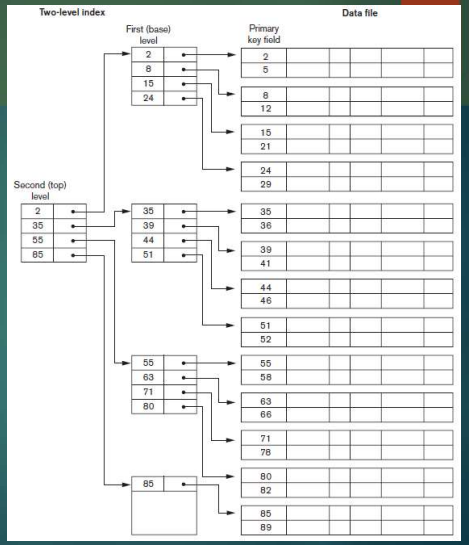

Figure 17.6 A two-level primary index resembling ISAM (indexed sequential access method) organization

56

Slide 17-57

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

- Tree data structure terminology
  - Tree is formed of nodes
  - Each node (except root) has one parent and zero or more child nodes
  - Leaf node has no child nodes
    - Unbalanced if leaf nodes occur at different levels
  - Nonleaf node called internal node
  - Subtree of node consists of node and all descendant nodes

57

Slide 17-58

Tree Data Structure

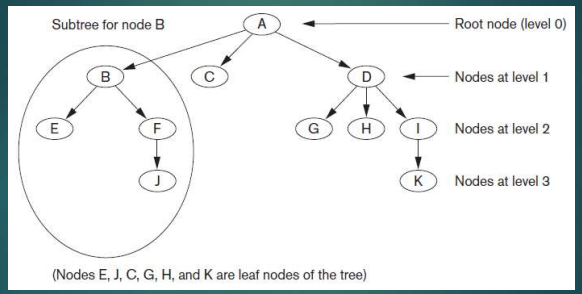

Figure 17.7 A tree data structure that shows an unbalanced tree

58

Slide 17-59

Search Trees and B-Trees

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

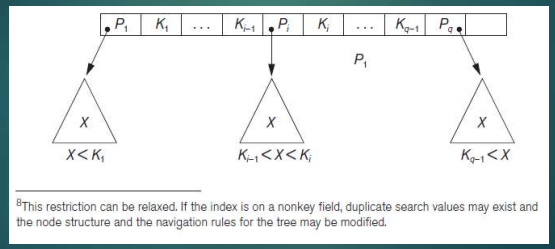

Figure 17.8 A node in a search tree with pointers to subtrees below it

59

Slide 17-60

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

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

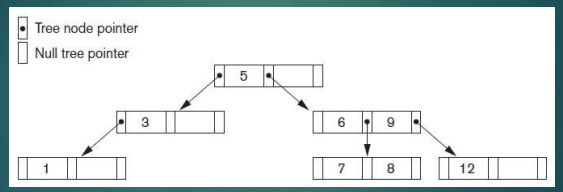

Figure 17.9 A search tree of order p = 3

60

B-Trees

Slide 17-61

- 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

61

B-Tree Structures

Slide 17-62

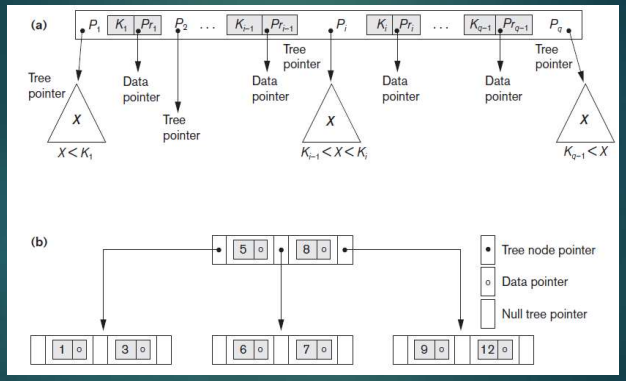

Figure 17.10 B-tree structures (a) A node in a B-tree with *q−1* search values (b) A B-tree of order *p=3*. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6

62

B+ -Trees

Slide 17-63

- 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

63

B+ -Trees (cont'd.)

Slide 17-64

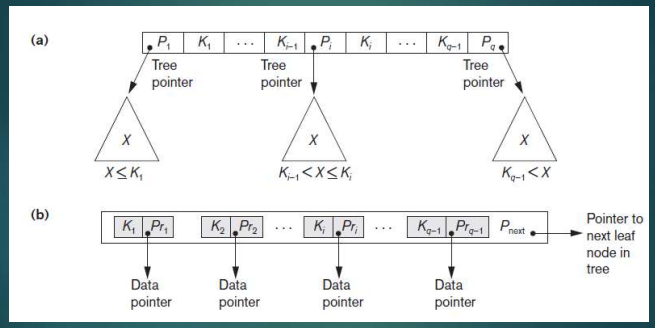

Figure 17.11 The nodes of a B+-tree (a) Internal node of a B+-tree with *q−1* search values (b) Leaf node of a B+-tree with *q−1* search values and *q−1* data pointers

64

Searching for a Record With Search Key Field Value K, Using

Slide 17-65

```
n ← block containing root node of B+-tree;
read block n;
while (n is not a leaf node of the B+-tree) do
    begin
    q ← number of tree pointers in node n;
    if K ≤ n.K1 (*n.Ki refers to the ith search field value in node n*)
        then n ← n.P1 (*n.Pi refers to the ith tree pointer in node n*)
        else if K > n.Kq–1
            then n ← n.Pq
            else begin
                search node n for an entry i such that n.Ki–1 < K ≤ n.Ki;
                n ← n.Pi
                end;
    read block n
    end;
search block n for entry (Ki, Pri) with K = Ki; (* search leaf node *)
if found
    then read data file block with address Pri and retrieve record
    else the record with search field value K is not in the data file;
```

Algorithm 17.2 Searching for a record with search key field value K, using a B+ -Tree

65

17.4 Indexes on Multiple Keys

Slide 17-66

- 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

66

Indexes on Multiple Keys (cont'd.)

Slide 17-67

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

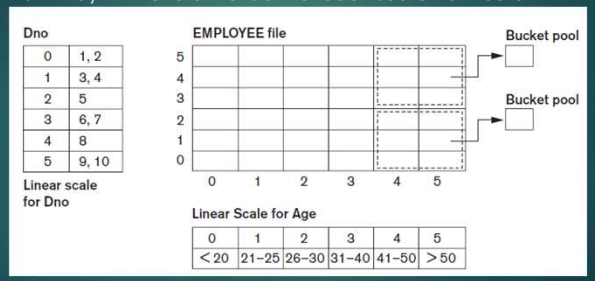

Figure 17.14 Example of a grid array on Dno and Age attributes

67

17.5 Other Types of Indexes

Slide 17-68

- 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

68

Hash Indexes (cont'd.)

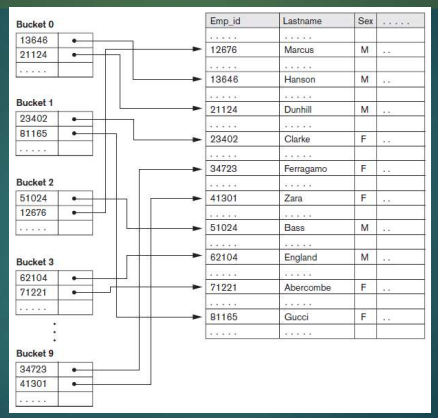

Figure 17.15 Hash-based indexing

Bitmap Indexes

- Used with a large number of rows
- Creates an index for one or more columns
  - Each value or value range in the column is indexed
- Built on one particular value of a particular field
  - Array of bits
- Existence bitmap
- Bitmaps for B+ -tree leaf nodes

Function-Based Indexing

- Value resulting from applying some function on a field (or fields) becomes the index key
- Introduced in Oracle relational DBMS
- Example
  - Function UPPER(Lname) returns uppercase representation

```
CREATE INDEX upper_ix ON Employee (UPPER(Lname));
```

  - Query

```
SELECT First_name, Lname
FROM Employee
WHERE UPPER(Lname)= "SMITH".
```


17.6 Some General Issues Concerning Indexing

- Physical index
  - Pointer specifies physical record address
  - Disadvantage: pointer must be changed if record is moved
- Logical index
  - Used when physical record addresses expected to change frequently
  - Entries of the form (K, K_P)

Index Creation

Slide 17-73

- General form of the command to create an index

```
CREATE [ UNIQUE ] INDEX <index name>
ON <table name> ( <column name> [ <order> ] { , <column name> [ <order> ] } )
[ CLUSTER ] ;
```

  - Unique and cluster keywords optional
  - Order can be ASC or DESC
- Secondary indexes can be created for any primary record organization
  - Complements other primary access methods

73

Indexing of Strings

Slide 17-74

- Strings can be variable length
- Strings may be too long, limiting the fan-out
- Prefix compression
  - Stores only the prefix of the search key adequate to distinguish the keys that are being separated and directed to the subtree

74

Tuning Indexes

Slide 17-75

- Tuning goals
  - Dynamically evaluate requirements
  - Reorganize indexes to yield best performance
- Reasons for revising initial index choice
  - Certain queries may take too long to run due to lack of an index
  - Certain indexes may not get utilized
  - Certain indexes may undergo too much updating if based on an attribute that undergoes frequent changes

75

Additional Issues Related to Storage of Relations and Indexes

Slide 17-76

- Enforcing a key constraint on an attribute
  - Reject insertion if new record has same key attribute as existing record
- Duplicates occur if index is created on a nonkey field
- Fully inverted file
  - Has secondary index on every field
- Indexing hints in queries
  - Suggestions used to expedite query execution

76

Additional Issues Related to Storage of Relations and Indexes (cont'd.)

Slide 17-77

- Column-based storage of relations
  - Alternative to traditional way of storing relations by row
  - Offers advantages for read-only queries
  - Offers additional freedom in index creation

77

17.7 Physical Database Design in Relational Databases

Slide 17-78

- Physical design goals
  - Create appropriate structure for data in storage
  - Guarantee good performance
- Must know job mix for particular set of database system applications
- Analyzing the database queries and transactions
  - Information about each retrieval query
  - Information about each update transaction

78

Physical Database Design in Relational Databases (cont'd.)

Slide 17-79

- Analyzing the expected frequency of invocation of queries and transactions
  - Expected frequency of using each attribute as a selection or join attribute
  - 80-20 rule: 80 percent of processing accounted for by only 20 percent of queries and transactions
- Analyzing the time constraints of queries and transactions
  - Selection attributes associated with time constraints are candidates for primary access structures

79

Physical Database Design in Relational Databases (cont'd.)

Slide 17-80

- Analyzing the expected frequency of update operations
  - Minimize number of access paths for a frequently-updated file
    - Updating the access paths themselves slows down update operations
- Analyzing the uniqueness constraints on attributes
  - Access paths should be specified on all *candidate key* attributes that are either the primary key of a file or unique attributes

80

Physical Database Design Decisions

Slide 17-81

- 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

81

17.8 Summary

Slide 17-82

- 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

82