

CS3743 Introduction to Database Systems

Data Storage

Dr. Weining Zhang

Department of Computer Science
University of Texas at San Antonio

October 29, 2009

Outline

- 1 Disk Storage
 - Disk Devices
 - Disk and Buffer Management
- 2 File Structures
 - File Organization
 - Operations on Files
- 3 External Hashing
 - Static Hashing

Database Storage Hierarchy

- Main Memory: volatile, random access, fast
 - Necessary for data operation
 - Too expensive for large data storage
- Disk: non-volatile, pseudo-random access, large space, speed not too bad
 - Good balance of speed, size, and cost
- Tape: non-volatile, sequential access, large space, slow access, can add new cartridges
 - Good for back-up, too slow for database operations

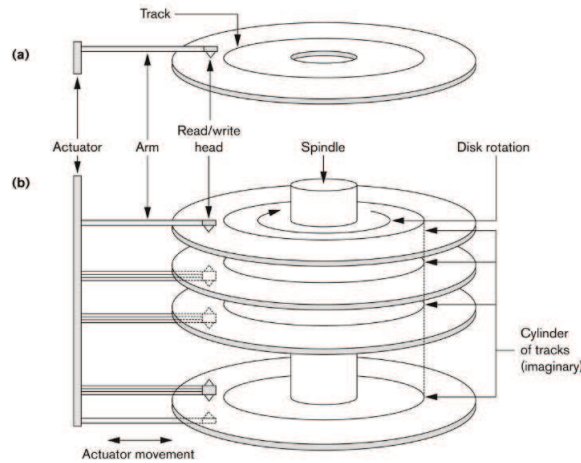
Disk Device

- Disks store data in tracks on surfaces.
- Storage space is divided into drives, cylinders, tracks and blocks (or pages)
 - A **drive** consists of multiple cylinders
 - A **cylinder** consists of all tracks that have the same radius (one track per surface)
 - A **track** consists of multiple blocks
 - Track capacities vary typically from 4 to 50 KB or more
 - A **block** contains B bytes, where B is typically between 512 to 4096
- Blocks are address units. Block address has the form (*drive#*, *cylinder#*, *surface#*, *block#*)

Disk Device (cont.)

Figure 13.1

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



Disk Access Time (cont.)

- Typical access time
 - Average seek time ranges from 8 to 40 millisecond
 - seek times depend on distance to travel
 - Average rotation delay is time for half revolution
 - E.g., 8.33ms (at 3600 RPM)
 - Transfer time = block size / transfer rate
 - transfer rate (TR) ranges from 1 to 100 MB per second
- Read/write n blocks
 - Random access: $AS = n \cdot (ST + RD + TT)$
 - Sequential access: $AS = ST + RD + n \cdot TT$

Bottom line: random I/O is expensive

Disk Access Time

- To read/write a block, the read/write head must move to the destination track, wait for the destination block to arrive, and to read/write every byte of the block
 - Blocks must be copied between disk and main memory before data is processed
- **Access time:** Time between a request and the completion of data transfer

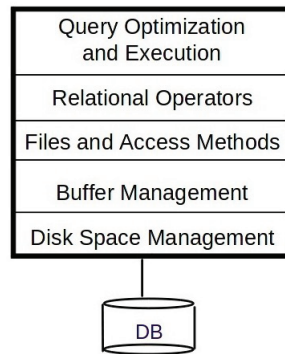
$$AccessTime = SeekTime + RotationDelay + TransferTime$$

- Seek Time (ST): Time to position read-write head over target track/cylinder
- Rotation Delay (RD): Time to rotate requested block so that its beginning is under read-write head
- Transfer Time (TT): Time to read/write a block (or for the whole block to pass through under read-write head)

Sample Disk Drive: IBM Ultrastar 36LP

- Formatted capacity: 36.9 GB
- Sector size: 512 to 528 variable (2-byte inc)
- Platters: 10
- Max. recording density: 350000 BPI
- Track density: 18400 TPI (per surface)
- Rotation speed: 7200 RPM
- Average rotation delay: 4.17 ms
- Sustained data rate: 19.5-31.9 MB
- Seek time: average 6.8ms, next track 0.6ms

A Layered Architecture of DBMS



Disk Manager

- Disk manager provides a program interface for managing disk storage space
 - Allocate one or more pages.
 - Return page id (page address)
 - De-allocate disk pages
 - Free page address
 - Read and write disk blocks. Copy pages between disk and main memory
- Maintains usage information
 - A map of free disk blocks
 - A map of files with their first blocks

Buffer Manager

- A **buffer** consists of a pool of buffer frames in main memory
 - Each frame holds a copy of one disk block
 - Each frame has a count of access requests and a flag of content change
- **Buffer manager** provides a program interface for pinning/unpinning a block and flushing a frame
 - It maintains an allocation table to map block# with frame#
 - It uses a pincount and a dirty bit to keep track of usage of a frame
- BM uses a **replacement policy** to determine which frame to reuse when buffer is full.
 - Least-recently-used (LRU). Reuse the frame that has not been used for the longest time.

Operations for Reading/Writing a Block

- To **read a block**
 - If the block is not in buffer, pin the block
 - If the block is not in buffer, find a frame for the block based on the replacement policy
 - If this frame is dirty (updated), write it back to disk
 - Read the new block into the frame
 - Update usage info
 - Return the frame address
- To **write a block**
 - If the block is not in buffer, pin the block in buffer
 - Write to the block in its buffer frame

File Organization
File of Records

- Logically, a file is a sequence of records
- A data **record** consists of a collection of fields (or values, items).
 - Each **field** is formed of a number of bytes and is interpreted according to the type of the information stored
 - A **record format** is a collection of field names and their corresponding types
- Records of a file can have a **fixed-format** or a **variable-format** with a **fixed-length** or a **variable-length**.
 - Fixed-format records may have fixed number of variable-length fields
 - Variable-format records can have a variable number of fixed-length fields

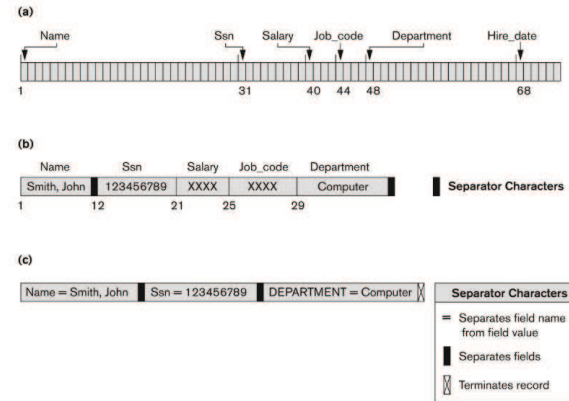
File Organization
Record Blocking

- Unspanned blocking.** Each record is store in whole in a block
 - Blocking factor** (bfr): the max number of records a block can hold

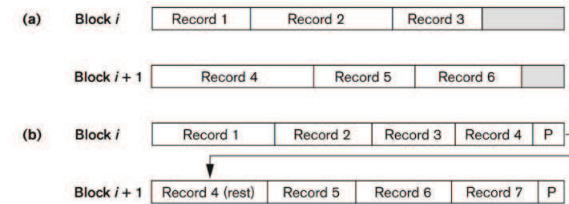
$$bfr = \lfloor \frac{blockSize}{recordSize} \rfloor$$
 - There may be empty space in a block if an integral number of records do not fit in one block
 - Best for fixed-length records
- Spanned blocking.** A record may be stored in part in different blocks
 - Each block can be fully packed
 - Need to keep pointers to various parts of the record

File Organization
Locate Records and Fields

- To store and retrieve records, the system needs to identify a record and a field in a string of bytes.



File Organization
Record Blocking (cont.)

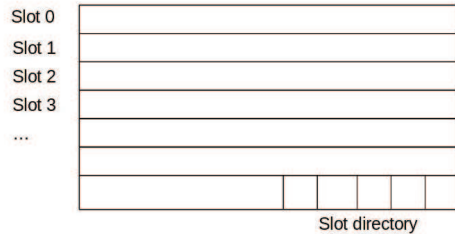


- A file has one or more header pages storing information such as record format, length, field names and their data types, and the addresses of the file blocks on disk.
- The physical disk blocks allocated to a file can be contiguous, linked, or indexed.

File Organization

Page/Block Format

- Physical layout of a block/page that enables efficient storage and retrieval of records
 - There are many options
- Slotted Page Format (for fixed-length records)



- Slot directory keeps the count of records, flags occupied slots
- Record ID (rid) is <Page#, Slot#>

File Organization

File Size In Blocks

Example

A file contains 200,000 fixed-length records, with record length 600 bytes. The disk has a Block size 2048 bytes (or 2 KB). Calculate the number of blocks of the file.

- Unspanned packing.
 - $bfr = \lfloor \frac{2048}{600} \rfloor = 3 \text{ rec/block}$ (with 248 bytes/block unused)
 - $FileSize = \lceil \frac{200000}{3} \rceil = 66667 \text{ blocks}$
- Spanned packing.
 - $FileSize = \lceil \frac{200000 \times 600}{2048} \rceil = 58594 \text{ blocks}$

Operations on Files

Typical File Operations

- OPEN: Readies the file for access, and associates a pointer that will refer to a current file record at each point in time.
- FIND: Searches for the first file record that satisfies a certain condition, and makes it the current file record.
- FINDNEXT: Searches for the next file record (from the current record) that satisfies a certain condition, and makes it the current file record.
- READ: Reads the current file record into a program variable.
- INSERT: Inserts a new record into the file & makes it the current file record.
- DELETE: Removes the current file record from the file, usually by marking the record to indicate that it is no longer valid.
- MODIFY: Changes the values of some fields of the current file record.
- CLOSE: Terminates access to the file.
- REORGANIZE: Reorganizes the file records. For example, the records marked deleted are physically removed from the file or a new organization of the file records is created.
- READ_ORDERED: Read the file blocks in order of a specific field of the file.

Operations on Files

Unordered File

- Also called a **heap file** or a pile file.
- New records are appended to the end of the file.
- A sequential search through the file is necessary for retrieving a record.
 - This requires to read on average a half of the blocks of the file, and is hence quite expensive.
- Record insertion is quite efficient.
- Reading the records in order of a particular field requires a sorting of the records.

Ordered File

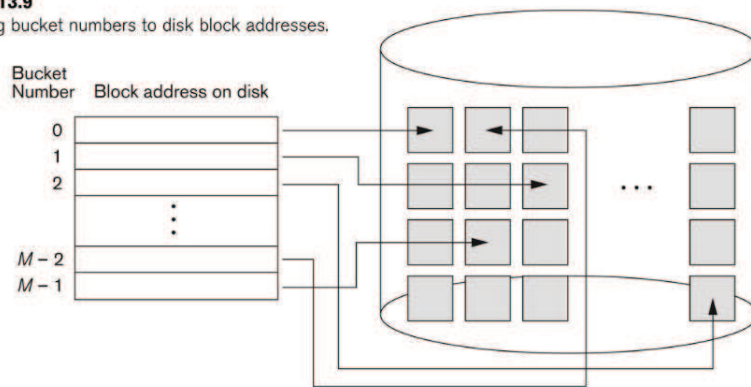
- Also called a **sequential file**.
- Records in the file are in a sorted order on a given field.
- Insertion is expensive. Records must be inserted to a correct location.
 - To improve efficiency, it is common to keep a separate, unordered overflow (or transaction) file for new records. This file is periodically merged with the main ordered file.
- A **binary search** can be used to retrieve a record of a given ordering field value.
 - This requires to read on average $O(\log_2 B)$ blocks, where B is file size in blocks, an improvement over sequential search.
- Reading the records in order of the ordering field is quite efficient

Hashed Files

- Hashing for disk files is called External Hashing
- The file blocks are divided into M equal-sized buckets, numbered $bucket_0, bucket_1, \dots, bucket_{M-1}$.
 - Typically, a bucket corresponds to one (or a fixed number of) disk block.
- A record fields is designated to be the hash key of the file.
- The record with hash key value K is stored in $bucket_i$, where $i = h(K)$, and h is the hashing function.
- Search is very efficient on the hash key.
- Collision occurs when a new record hashes to a bucket that is already full.

Hashed Files (cont.)

Figure 13.9
Matching bucket numbers to disk block addresses.



Collision Resolution Methods

- Open Addressing
 - Proceeding from the collision address, checks subsequent positions in order until an open (unused) position is found.
- Chaining.
 - Extending the array with a number of overflow positions and adding a pointer field to each record location.
 - A collision is resolved by placing the new record in an unused overflow location and setting the pointer in the collision location to point to that overflow location.
- Rehashing.
 - Use a second hash function if the first results in a collision. If another collision results, use open addressing or a third hash function.

Chaining

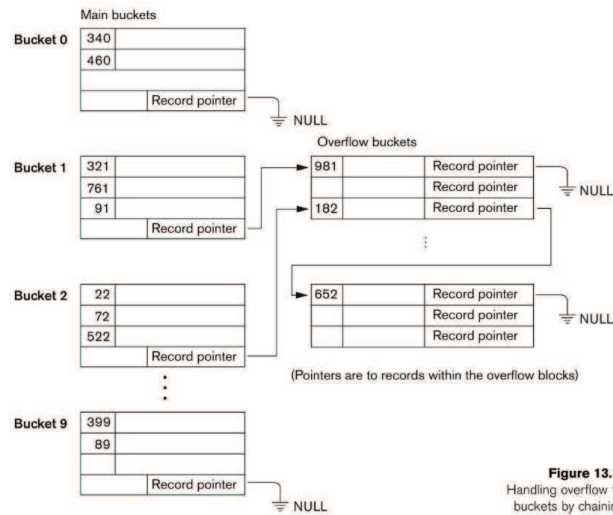


Figure 13.10
Handling overflow for
buckets by chaining.

Collision Prevention

- To reduce overflow records, a hash file is typically kept 70-80% full.
- The hash function h should distribute the records uniformly among the buckets
 - Otherwise, search time will be increased because many overflow records will exist.
- Main disadvantages of static external hashing:
 - Fixed number of buckets M is a problem if the number of records in the file grows or shrinks.
 - Ordered access on the hash key is quite inefficient (requires sorting the records).