**Storage and File Structure**

Jongpil Yoon  
Spring 2001  
Southern University at Baton Rouge

---

**Outline**

- Overview of Physical Storage Media
- Magnetic Disks
- RAID
- Tertiary Storage
- Storage Access
- File Organization
- Organization of Records in Files
- Data-Dictionary Storage
- Storage Structures for Object-Oriented Databases

---

**Classification of Physical Storage Media**

- Speed with which data can be accessed
- Cost per unit of data
- Reliability
  - data loss on power failure or system crash
  - physical failure of the storage device

Can differentiate storage into:

- **volatile storage**: loses contents when power is switched off
- **non-volatile storage**: contents persist even when power is switched off. Includes secondary and tertiary storage, as well as battery-backed up main-memory.

---

**Storage Hierarchy**
Physical Storage Media

- **Cache**: fast and most costly form of storage; volatile; managed by the hardware/operating system.
- **Main memory**:
  - general-purpose machine instructions operate on data resident in main memory
  - fast access, but generally too small to store the entire database
  - sometimes referred to as core memory
  - volatile: contents of main memory are usually lost if a power failure or system crash occurs

Physical Storage Media (Cont.)

- **Flash memory**: reads are roughly as fast as main memory; data survives power failure; but can support a only limited number of write/erase cycles
- **Magnetic-disk storage**: primary medium for the long-term storage of data; typically stores entire database.
  - data must be moved from disk to main memory for access, and written back for storage
  - **direct-access**: possible to read data on disk in any order
  - usually survives power failures and system crashes; disk failure can destroy data, but is much less frequent than system crashes

Physical Storage Media (Cont.)

- **Optical storage**: non-volatile. ROM most popular form. Write-once, read-many (WORM) optical disks used for archival storage.
- **Tape storage**: non-volatile, used primarily for backup (to recover from disk failure), and for archival data
  - **sequential-access**: much slower than disk
  - very high capacity (5GB tapes are common)
  - tape can be removed from drive ⇒ storage costs much cheaper than disk.

Storage Hierarchy (Cont.)

- **primary storage**: Fastest media but volatile (cache, main memory)
- **secondary storage**: next level in hierarchy, non-volatile, moderately fast access time; also called **on-line storage** (flash memory, magnetic disks)
- **tertiary storage**: lowest level in hierarchy, non-volatile, slow access time; also called **off-line storage** (magnetic tape, optical storage)
Magnetic Disks Mechanism

- Read-write head: device positioned close to the platter surface; reads or writes magnetically encoded information.
- Surface of platter divided into circular tracks, and each track is divided into sectors. A sector is the smallest unit of data that can be read or written.
- To read/write a sector
  - disk arm swings to position head on right track
  - platter spins continually; data is read/written when sector comes under head
- Head-disk assemblies: multiple disk platters on a single spindle, with multiple heads (one per platter) mounted on a common arm.
- Cylinder $i$ consists of $i^{th}$ track of all the platters

Magnetic Disks

- Surface of platter divided into circular tracks, and each track is divided into sectors. A sector is the smallest unit of data that can be read or written.
- To read/write a sector
  - disk arm swings to position head on right track
  - platter spins continually; data is read/written when sector comes under head
- Head-disk assemblies: multiple disk platters on a single spindle, with multiple heads (one per platter) mounted on a common arm.
- Cylinder $i$ consists of $i^{th}$ track of all the platters

Disk Subsystem

- Disk controller: interfaces between the computer system and the disk drive hardware.
  - accepts high-level commands to read or write a sector
  - initiates actions such as moving the disk arm to the right track and actually reading or writing the data.
**Performance Measures of Disks**

- **Access time**: the time it takes from when a read or write request is issued to when data transfer begins. Consists of:
  - **Seek time**: time it takes to reposition the arm over the correct track. Average seek time is 1/3rd the worst case seek time.
  - **Rotational latency**: time it takes for the sector to be accessed to appear under the head. Average latency is 1/2 of the worst case latency.

- **Data-transfer rate**: the rate at which data can be retrieved from or stored to the disk.

- **Mean time to failure (MTTF)**: the average time the disk is expected to run continuously without any failure.

**Optimization of Block Access**

- **Block**: a contiguous sequence of sectors from a single track
  - data is transferred between disk and main memory in blocks
  - sizes range from 512 bytes to several kilobytes

- File organization: optimize block access time by organizing the blocks to correspond to how data will be accessed. Store related information on the same or nearby cylinders.

- **Nonvolatile write buffers**: speed up disk writes by writing blocks to a non-volatile RAM buffer immediately; controller then writes to disk whenever the disk has no other requests.

- **Log disk**: a disk devoted to writing a sequential log of block updates; this eliminates seek time. Used like nonvolatile RAM.

**RAID**

- **Redundant Arrays of Inexpensive Disks**: disk organization techniques that take advantage of utilizing large numbers of inexpensive, mass-market disks.

- Originally a cost-effective alternative to large, expensive disks

- Today RAIDs are used for their higher reliability and bandwidth, rather than for economic reasons. Hence the “I” is interpreted as independent, instead of inexpensive.

**Improvement of Reliability via Redundancy**

- The chance that some disk out of a set of \( n \) disks will fail is much higher than the chance that a specific single disk will fail. E.g., a system with 100 disks, each with MTTF of 100,000 hours (approx. 11 years), will have a system MTTF of 1000 hours (approx. 41 days).

- **Redundancy**: store extra information that can be used to rebuild information lost in a disk failure

- E.g. **Mirroring** (or **shadowing**)
  - duplicate every disk. Logical disk consists of two physical disks.
  - every write is carried out on both disks
  - if one disk in a pair fails, data still available in the other
Improvement in Performance via Parallelism

- Two main goals of parallelism in a disk system:
  1. Load balance multiple small accesses to increase throughput
  2. Parallelize large accesses to reduce response time
- Improve transfer rate by striping data across multiple disks.

- **Bit-level striping:** split the bits of each byte across multiple disks
  - In an array of eight disks, write bit $i$ of each byte to disk $i$.
  - Each access can read data at eight times the rate of a single disk.
  - But seek/access time worse than for a single disk.

- **Block-level striping:** with $n$ disks, block $i$ of a file goes to disk $(i \mod n)+1$.

RAID Levels

- Schemes to provide redundancy at lower cost by using disk striping combined with parity bits
- Different RAID organizations, or RAID levels, have differing cost, performance and reliability characteristics
- **Level 0:** Striping at the level of blocks; non-redundant. Used in high-performance applications where data loss is not critical.
- **Level 1:** Mirrored disks; offers best write performance. Popular for applications such as storing log files in a database system.

Storage Access

- A database file is partitioned into fixed-length storage units called **blocks**. Blocks are units of both storage allocation and data transfer.
- Database system seeks to minimize the number of block transfers between the disk and memory. We can reduce the number of disk accesses by keeping as many blocks as possible in main memory.
- **Buffer:** portion of main memory available to store copies of disk blocks.
- **Buffer manager:** subsystem responsible for allocating buffer space in main memory.
**Buffer Manager**

- Programs call on the buffer manager when they need a block from disk
  - The requesting program is given the address of the block in main memory, if it is already present in the buffer.
  - If the block is not in the buffer, the buffer manager allocates space in the buffer for the block, replacing (throwing out) some other block, if required, to make space for the new block.
  - The block that is thrown out is written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk.
  - Once space is allocated in the buffer, the buffer manager reads in the block from the disk to the buffer, and passes the address of the block in main memory to the requester.

**Buffer-Replacement Policies**

- Most operating systems replace the block least recently used (LRU)
  - LRU: use past pattern of block references as a predictor of future references
- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user’s query to predict future references. LRU can be a bad strategy for certain access patterns involving repeated scans of data
- Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable

**Buffer-Replacement Policies (2)**

- **Pinned block**: memory block that is not allowed to be written back to disk.
- **Toss-immediate** strategy: frees the space occupied by a block as soon as the final tuple of that block has been processed
- Most recently used (MRU) strategy: system must pin the block currently being processed. After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block.
- Buffer manager can use statistical information regarding the probability that a request will reference a particular relation
  - E.g., the data dictionary is frequently accessed. Heuristic: keep data-dictionary blocks in main memory buffer

**File Organization**

- The database is stored as a collection of files. Each file is a sequence of records. A record is a sequence of fields.
- One approach:
  - assume record size is fixed
  - each file has records of one particular type only
  - different files are used for different relations
  This case is easiest to implement; will consider variable length records later.
**Fixed-Length Records**

- Simple approach:
  - Store record \(i\) starting from byte \(n^*(i-1)\), where \(n\) is the size of each record.
  - Record access is simple but records may cross blocks.
- Deletion of record \(i\) - alternatives:
  - move records \(i+1, \ldots, n\) to \(i, \ldots, n-1\)
  - move record \(n\) to \(i\)
  - Link all free records on a free list

**Free Lists**

- Store the address of the first record whose contents are deleted in the file header.
- Use this first record to store the address of the second available record, and so on
- Can think of these stored addresses as *pointers* since they “Point” to the location of a record.

**Free Lists (Cont.)**

- More space efficient representation: reuse space for normal attributes of free records to store pointers. (No pointers stored in in-use records.)
- **Dangling pointers** occur if we move or delete a record to which another record contains a pointer; that pointer no longer points to the desired record.
- Avoid moving or deleting records that are pointed to by other records; such records are pinned.

**Variable-Length Records**

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields.
  - Record types that allow repeating fields (used in some older data models).
- **Representation**
  - Byte string representation: Attach an end-of-record (⊥) control character to the end of each record
    - Difficulty with deletion
    - Difficulty with growth
  - Pointer representation
Variable-Length Records: Slotted Page Structure

- Header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record

- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must then be updated.

- Pointers should not point directly to record; instead they should point to the entry for the record in header.

Variable-Length Records (Cont.)

- Fixed-length representation:
  - reserved space
  - pointers

- Reserved space: can use fixed-length records of a known maximum length; unused space in shorter records filled with a null or end-of-record symbol.

<table>
<thead>
<tr>
<th></th>
<th>Percyridge</th>
<th>A-102</th>
<th>400</th>
<th>A-201</th>
<th>900</th>
<th>A-218</th>
<th>700</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Round Hill</td>
<td>A-305</td>
<td>350</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Manus</td>
<td>A-215</td>
<td>700</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Redwood</td>
<td>A-222</td>
<td>700</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Brighten</td>
<td>A-217</td>
<td>750</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pointer Method

- Pointers: the maximum record length is not known; a variable-length record is represented by a list of fixed-length records, chained together via pointers.

Pointer Method (Cont.)

- Disadvantage to pointer structure; space is wasted in all records except the first in a chain.

- Solution is to allow two kinds of block in file:
  - Anchor block: contains the first records of chain
  - Overflow block: contains records other than those that are the first records of chains.
**Organization of Records in Files**

- **Heap**: a record can be placed anywhere in the file where there is space.
- **Sequential**: store records in sequential order, based on the value of the search key of each record.
- **Hashing**: a hash function is computed on some attribute of each record; the result specifies in which block of the file the record should be placed.
- **Clustering**: records of several different relations can be stored in the same file; related records are stored on the same block.

**Sequential File Organization**

- Suitable for applications that require sequential processing of the entire file.
- The records in the file are ordered by a search-key.

<table>
<thead>
<tr>
<th>Location</th>
<th>Code</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>750</td>
</tr>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Downtown</td>
<td>A-110</td>
<td>600</td>
</tr>
<tr>
<td>Miami</td>
<td>A-215</td>
<td>700</td>
</tr>
<tr>
<td>Perryridge</td>
<td>A-102</td>
<td>400</td>
</tr>
<tr>
<td>Perryridge</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Perryridge</td>
<td>A-218</td>
<td>700</td>
</tr>
<tr>
<td>Redwood</td>
<td>A-222</td>
<td>700</td>
</tr>
<tr>
<td>Route Hill</td>
<td>A-305</td>
<td>350</td>
</tr>
</tbody>
</table>

**Sequential File Organization (2)**

- Deletion: pointer chains.
- Insertion: must locate the position in the file where the record is to be inserted.
  - if there is free space insert there.
  - if no free space, insert the record in an overflow block.
  - In either case, pointer chain must be updated.
- Need to reorganize the file from time to time to restore sequential order.

**Clustering File Organization**

- Simple file structure stores each relation in a separate file.
- Can instead store several relations in one file using a clustering file organization. E.g., clustering organization of customer and depositor.

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hayes</td>
<td>A-102</td>
<td>Main</td>
</tr>
<tr>
<td>Hayes</td>
<td>A-220</td>
<td>Main</td>
</tr>
<tr>
<td>Hayes</td>
<td>A-503</td>
<td>Main</td>
</tr>
<tr>
<td>Turner</td>
<td>Putnam</td>
<td>Stamford</td>
</tr>
<tr>
<td>Turner</td>
<td>A-305</td>
<td></td>
</tr>
</tbody>
</table>

- Good for queries involving depositor| customer, and for queries involving one single customer and his accounts.
- Bad for queries involving only customer.
- Results in variable size records.