Disks & Files

Yanlei Diao
UMass Amherst

Slides Courtesy of R. Ramakrishnan and J. Gehrke
DBMS Architecture

- Query Parser
- Query Rewriter
- Query Optimizer
- Query Executor
- Lock Manager for Concurrency
- Access Methods (Buffer Manager)
- Log Manager for Recovery
- Disk Space Manager
- DB
Outline

- Disks and Disk Space Manager
- Disk-Resident Data Structures
  - Files of records
  - Indexes (next lecture)
Memory Hierarchy

- **Main Memory** (RAM)
  - Random access, fast, usually volatile
  - Main memory for currently used data

- **Magnetic Disk**
  - Random access, relatively slow, nonvolatile
  - Persistent storage for all data in the database

- **Tape**
  - Sequential scan (read the entire tape to access the last byte), nonvolatile
  - For archiving older versions of the data
Disks and DBMS Design

- A database is stored on disks. This has major implications on DBMS design!
  - **READ**: transfer data from disk to RAM for data processing.
  - **WRITE**: transfer data (new/modified) from RAM to disk for persistent storage.
  - Both are high-cost operations relative to in-memory operations, so must be planned carefully!
Basics of Disks

- Unit of storage and retrieval: *disk block* or *page*.
  - A contiguous sequence of bytes.
  - Size is a DBMS parameter, 4KB, 8KB, or larger.

- Unlike RAM, *time to retrieve a page varies*!
  - It depends upon the location on disk.
  - Relative placement of pages on disk has major impact on DBMS performance!
Components of a Disk

- Spindle and Platters
  E.g. spin at 7200 or 15,000 rpm (revolutions per minute)

- Arm assembly and Disk heads
  - Arm assembly moves in or out, e.g., 2-10ms
  - Only one head reads/writes at any one time.
Data on Disk

- A platter consists of **tracks**.
  - single-sided platters
  - double-sided platters
- Tracks under heads make a **cylinder** (imaginary!)
- Each track is divided into **sectors** (whose size is fixed).
- **Block (page) size** is a multiple of **sector size** (DBMS parameter).
Accessing a Disk Page

- Time to access (read/write) a disk block:
  1. *seek time* (moving arms to position a disk head on a track)
  2. *rotational delay* (waiting for a block to rotate under the head)
  3. *transfer time* (actually moving data to/from disk surface)

- Seek time and rotational delay dominate.
  - *seek time*: 2 to 10 msec
  - *rotational delay*: 0 to 10 msec
  - *transfer rate*: <1msec/page, up to 100’s MB/sec

- Key to lower I/O cost: reduce seek/rotation delays!

Hardware vs. software solutions?
Arranging Pages on Disk

- Software solution using the ‘next’ block concept:
  - blocks on the same track, followed by
  - blocks on the same cylinder, followed by
  - blocks on an adjacent cylinder.

- Pages in a file should be arranged sequentially on disk (by `next`), to minimize seek and rotational delay.
  - What is the cost of scanning the file, called a sequential scan?
  - Contrast this to reading pages in a file in a random order.
Disk Space Manager

- Lowest layer of DBMS managing space on disk. Higher levels call it to:
  - allocate/de-allocate a page
  - allocate/de-allocate a sequence of pages
  - read/write a page

- Requests for a sequence of pages are satisfied by allocating the pages sequentially on disk!
  - Higher levels don’t need to know any details.
Outline

- Disks and Disk Space Manager
- Disk-Resident Data Structures
  - Files of records
  - Indexes (next lecture)
Access Methods

- **Access methods** are routines that manage disk-based data structures.

- **File of records:**
  - Abstraction of disk storage for query processing
  - *Record id (rid)* is sufficient to physically locate a record

- **Indexes:**
  - Auxiliary data structures
  - Given a value in the *index search key* field(s), find the record ids of records with this value
File of Records

- Abstraction of disk-resident data for query processing: *a file of records residing on multiple pages*
  - A number of *fields* are organized in a *record*
  - A collection of records are organized in a *page*
  - A collection of pages are organized in a *file*
Creating Tables

CREATE TABLE Sailors
(   sid INTEGER,
    sname VARCHAR(50) NOT NULL,
    rating INTEGER,
    age REAL,
    PRIMARY KEY (sid));

CREATE TABLE Boats
(   bid INTEGER,
    bname CHAR (20),
    color CHAR(20),
    PRIMARY KEY (bid)
    UNIQUE (bname));

CREATE TABLE Reserves
(   sid INTEGER,
    bid INTEGER,
    day DATE,
    PRIMARY KEY (sid,bid,day),
    FOREIGN KEY (sid) REFERENCES Sailors (sid)
    ON DELETE NO ACTION ON UPDATE CASCADE
    FOREIGN KEY (bid) REFERENCES Boats (bid)
    ON DELETE SET DEFAULT ON UPDATE CASCADE);
Record Format: *Fixed Length*

- **Fixed length record**: (1) the number of fields is fixed, (2) each field has a fixed length.

- Store fields consecutively in a record. How do we find *i’th* field of the record?

> **Base address (B)**  

**Address** = **B**+**L1**+**L2**

- Record type: the *number of fields* and *type of each field* (defined in the schema), stored in *system catalog*.
Record Format: *Variable Length*

- **Variable length record:** (1) number of fields is fixed, (2) some fields are of variable length

![Diagram of variable length record](image)

2nd choice offers direct access to i’th field; but small directory overhead.
Page Format

- How do we store a collection of records in a page?

- View a page as a collection of slots, one for each record.
- A record is identified by \( rid = \langle \text{page id, slot #} \rangle \)
  - Record ids (rids) are used in indexes. More on this later…
Page Format: **Fixed Length Records**

If we move records for free space management, we may change rids! Later erroneous results or poor performance.
Page Format: **Variable Length Records**

- Rid = (i,N)
- … Rid = (i,2)
- Rid = (i,1)

**Page i**

- Search?
- Insert?
- Delete?

- The ability to move records on a page without changing their rids is very important!
Compaction: get all slots whose offset is not -1, sort by start address, move their records up in sorted order. No change of rids!
File of Records

- **File**: a collection of pages, each containing a collection of records. Typically, one file for each relation.
  - **Updates**: insert/delete/modify records
  - **Sequential scan**: scan all records (possibly with some conditions on the records to be retrieved)
  - **Index scan**: read a record given a record id (rid)

- Files in DBMS versus Files in OS?
Heap (Unordered) Files

- **Heap file**: contains records in no particular order.
- As a file grows and shrinks, disk pages are allocated and de-allocated.
- To support record-level operations, we must:
  - keep track of the *pages* in a file
  - keep track of the *records* on a page
  - keep track of *free space* on pages
Heap File Using a Page Directory

- A directory entry per page: a pointer to the page, # free bytes on the page.
- The directory is a collection of pages; a linked list is one implementation.
  - Much smaller than the linked list of all data pages.
- Search for space for insertion: fewer I/Os.