Storing Data: Disks and Files

Yanlei Diao
UMass Amherst
Feb 22, 2006
Memory Hierarchy

- **Primary storage:** Main Memory (and cache)
  - Random access, fast; usually volatile
  - Main memory for currently used data.
- **Secondary storage:** Magnetic Disk
  - Random access, relatively slow; nonvolatile
  - Disk for the main database.
- **Tertiary storage:** Tape
  - Sequential scan (read the entire tape for the last type); nonvolatile
  - Tapes for archiving older versions of the data.
Disks and DBMS Design

- DBMS stores information on disks.
- This has major implications for DBMS design!
  - **READ**: transfer data from disk to main memory (RAM) for data processing.
  - **WRITE**: transfer data from RAM to disk for persistent storage.
  - Both are high-cost operations, relative to in-memory operations, so must be planned carefully!
Why Not Store Everything in Main Memory?

- **Main memory is volatile.** We want data to be saved between runs. (Obviously!)
- **Costs too much.** $100 will buy you either 1GB of RAM or 160GB of disk today.
- **32-bit addressing limitation.**
  - $2^{32}$ bytes can be directly addressed in memory.
  - Number of objects cannot exceed this number.
Basics of Disks

- Unit of storage and retrieval: disk block or page.
  - A disk block/page is a contiguous sequence of bytes.
  - Size of a DBMS parameter, 4KB or 8KB.
- Disks support direct access to a page.
- Unlike RAM, time to retrieve a page varies!
  - It depends upon the location on disk.
  - Therefore, relative placement of pages on disk has major impact on DBMS performance!
Components of a Disk

- **Platters** spin (say, 7200rpm).
- **Arm assembly** is moved in or out to position a head on a desired **track**.
- Only one head reads/writes at any one time.
- Tracks under heads make a **cylinder** (imaginary!).
- Each **track** is divided into **sectors** (whose size is fixed).
- **Block size** is a multiple of **sector size**.
Accessing a Disk Page

- Time to access (read/write) a disk block:
  - seek time (moving arms to position disk head on track)
  - rotational delay (waiting for block to rotate under head)
  - transfer time (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 20msec
  - Rotational delay varies from 0 to 10msec
  - Transfer rate is about 1msec per 4KB page
- Key to lower I/O cost: reduce seek/rotation delays! Hardware vs. software solutions?
Arranging Pages on Disk

- `Next’ block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder

- Blocks in a file should be arranged sequentially on disk (by `next’), to minimize seek and rotational delay.

- For a sequential scan, pre-fetching several pages at a time is a big win!
Lowest layer of DBMS software manages space on disk.

Higher levels call upon this layer to:
- allocate/de-allocate a page
- read/write a page

Request for a sequence of pages must be satisfied by allocating the pages sequentially on disk!
- Higher levels don’t need to know how this is done, or how free space is managed.
Files

- Access method layer offers an abstraction of data on disk: 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
Record Format: Fixed Length

- Information of a *record type* e.g., the *number of fields* and *field types* is stored in the *system catalog*.
- *Fixed length record*: (1) the number of fields is fixed, (2) each field has a *fixed* length.
- Store fields consecutively in a record.
- Finding *i’th* field does not require scan of record.
Record Format: Variable Length

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

  - Second offers direct access to i’th field, efficient storage of *nulls* (special *don’t know* value); small directory overhead.
Page Format

- How to store a collection of records on a page?
- Consider a page as a collection of slots, one for each record.
- A record is identified by \( \text{rid} = \langle \text{page id, slot #} \rangle \)
- Record ids (rids) are used in indexes (Alternatives 2 and 3).
Page Format: Fixed Length Records

Moving records for free space management changes rid! May not be acceptable.
Page Format: Variable Length Records

- Can move records on page without changing rid; so, attractive for fixed-length records too. (*level of indirection*)
Page or block is OK when doing I/O, but higher levels of DBMS operate on records and files of records.

FILE: A collection of pages, each containing a collection of records. Must support:
- insert/delete/modify record
- read a particular record (specified using record id)
- scan all records (possibly with some conditions on the records to be retrieved)
Unordered (Heap) Files

- Simplest file structure contains records in no particular order.
- As 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 *free space* on pages
  - keep track of the *records* on a page
- There are many alternatives for keeping track of this.
Heap File Implemented as a List

- (heap file name, header page id) stored in a known place.
- Two doubly linked lists, for full pages & pages with space.
  - Each page contains 2 `pointers’ plus data.
- Upon insertion, scan the list of pages with space, or ask disk space manager to allocate a new page
A directory entry per page; it can include the number of free bytes on the page.

The directory is a collection of pages; linked list implementation is just one alternative.
- **Much smaller than linked list of all HF pages!**
System Catalogs

- For each index:
  - structure (e.g., B+ tree) and search key fields
- For each relation:
  - name, file name, file structure (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- For each view:
  - view name and definition
- Plus statistics, authorization, buffer pool size, etc.

*Catalogs are themselves stored as relations!*
### AttrCat\((\text{attr\_name}, \text{rel\_name}, \text{type}, \text{position})\)

<table>
<thead>
<tr>
<th>attr_name</th>
<th>rel_name</th>
<th>type</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr_name</td>
<td>Attribute_Cat</td>
<td>string</td>
<td>1</td>
</tr>
<tr>
<td>rel_name</td>
<td>Attribute_Cat</td>
<td>string</td>
<td>2</td>
</tr>
<tr>
<td>type</td>
<td>Attribute_Cat</td>
<td>string</td>
<td>3</td>
</tr>
<tr>
<td>position</td>
<td>Attribute_Cat</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td>sid</td>
<td>Students</td>
<td>string</td>
<td>1</td>
</tr>
<tr>
<td>name</td>
<td>Students</td>
<td>string</td>
<td>2</td>
</tr>
<tr>
<td>login</td>
<td>Students</td>
<td>string</td>
<td>3</td>
</tr>
<tr>
<td>age</td>
<td>Students</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td>gpa</td>
<td>Students</td>
<td>real</td>
<td>5</td>
</tr>
<tr>
<td>fid</td>
<td>Faculty</td>
<td>string</td>
<td>1</td>
</tr>
<tr>
<td>fname</td>
<td>Faculty</td>
<td>string</td>
<td>2</td>
</tr>
<tr>
<td>sal</td>
<td>Faculty</td>
<td>real</td>
<td>3</td>
</tr>
</tbody>
</table>
Summary

- Disks provide cheap, non-volatile storage.
  - Random access
  - But cost depends on location of page on disk
  - Important to arrange data sequentially to minimize seek and rotation delays.
Summary (Contd.)

- **Variable length record format** with field offset directory offers support for direct access to i’th field and null values.

- **Slotted page format** supports variable length records and allows records to move on page.
Summary (Contd.)

- **File organization** keeps track of pages in a file, supports abstraction of a collection of records.
  - Pages (full or with free space) identified using linked list or directory structure.

- **Indexes** support efficient retrieval of records based on the values in some fields.

- **Catalog relations** store information about relations, indexes and views. *(Information that is common to all records in a given collection.)*