CS 405G: Introduction to Database Systems

Storage
Outline

- It’s all about disks!
  - That’s why we always draw databases as
  - And why the single most important metric in database processing is the number of disk I/O’s performed

- Storing data on a disk
  - Record layout
  - Block layout
The Storage Hierarchy

- Main memory (RAM) for currently used data
- Disk for the main database (secondary storage).
- Tapes for archiving older versions of the data (tertiary storage).

Jim Gray’s Storage Latency Analogy: How Far Away is the Data?

- **Registers**: 1 min
- **On Chip Cache**: 10 min
- **On Board Cache**: 1.5 hr
- **Memory**: 2 Years
- **Disk**: 2,000 Years
- **Tape/Optical Robot**: 2,000 Years
- **Lexington**: 1.5 hr
- **This Lecture Hall**: 10 min
- **This Room**: 10 min
- **My Head**: 1 min
- **Andromeda**: 10^9 years
A typical disk

- Spindle rotation
- Arm movement
- Disk arm
- Disk head
- Tracks
- Platter
- Cylinders

"Moving parts" are slow
Higher-density sectors on inner tracks and/or more sectors on outer tracks

A block is a logical unit of transfer consisting of one or more sectors
Disk access time

Sum of:

- **Seek time**: time for disk heads to move to the correct cylinder
- **Rotational delay**: time for the desired block to rotate under the disk head
- **Transfer time**: time to read/write data in the block (= time for disk to rotate over the block)
Random disk access

Seek time + rotational delay + transfer time

- **Average seek time**
  - Time to skip one half of the cylinders?
  - Not quite; should be time to skip a third of them (why?)
  - “Typical” value: 5 ms

- **Average rotational delay**
  - Time for a half rotation (a function of RPM)
  - “Typical” value: 4.2 ms (7200 RPM)

- **Typical transfer time**
  - .08msec per 8K block
Sequential Disk Access Improves Performance

Seek time + rotational delay + transfer time

- Seek time
  - 0 (assuming data is on the same track)
- Rotational delay
  - 0 (assuming data is in the next block on the track)
- Easily an order of magnitude faster than random disk access!
Performance tricks

- Disk layout strategy
  - Keep related things (what are they?) close together: same sector/block! same track! same cylinder! adjacent cylinder

- Double buffering
  - While processing the current block in memory, **prefetch** the next block from disk (overlap I/O with processing)

- Disk scheduling algorithm

- Track buffer
  - Read/write one entire track at a time

- Parallel I/O
  - More disk heads working at the same time
Files

- Blocks are the interface for 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
- fetch 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.
  - We’ll consider 2
The header page id and Heap file name must be stored someplace.

- Database "catalog"

Each page contains 2 `pointers’ plus data.
The entry for a page 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!*
Record layout

Record = row in a table

- **Variable-format records**
  - Rare in DBMS—table schema dictates the format
  - Relevant for semi-structured data such as XML

- **Focus on fixed-format records**
  - With fixed-length fields only, or
  - With possible variable-length fields
Record Formats: Fixed Length

- All field lengths and offsets are constant
  - Computed from schema, stored in the system catalog
- Finding $i^{th}$ field done via arithmetic.
Fixed-length fields

- **Example:**
  ```sql
  CREATE TABLE Student(SID INT, name CHAR(20), age INT, GPA FLOAT);
  ```

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>24</th>
<th>28</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart (padded with space)</td>
<td>10</td>
<td>2.3</td>
<td></td>
</tr>
</tbody>
</table>

  - Watch out for alignment
    - May need to pad; reorder columns if that helps
  - What about **NULL**?
    - Add a bitmap at the beginning of the record
Record Formats: Variable Length

- Two alternative formats (# fields is fixed):

  Fields Delimited by Special Symbols

  Array of Field Offsets

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

- **Example:**
  ```sql
  CREATE TABLE Student(SID INT, name CHAR(20), age INT, GPA FLOAT, picture BLOB(32000));
  ```

- **Student records get “de-clustered”**
  - Bad because most queries do not involve `picture`

- **Decomposition (automatically done by DBMS and transparent to the user)**
  - `Student(SID, name, age, GPA)`
  - `StudentPicture(SID, picture)`
Block layout

How do you organize records in a block?

- Fixed length records
- Variable length records
  - **NSM** (N-ary Storage Model) is used in most commercial DBMS
**Record id** = <page id, slot #>. *In first alternative, moving records for free space management changes rid; may not be acceptable.*
NSM

- Store records from the beginning of each block
- Use a directory at the end of each block
  - To locate records and manage free space
  - Necessary for variable-length records

Why store data and directory at two different ends?
Both can grow easily
**Options**

- Reorganize after every update/delete to avoid fragmentation (gaps between records)
  - Need to rewrite half of the block on average
- What if records are fixed-length?
  - Reorganize after delete
    - Only need to move one record
    - Need a pointer to the beginning of free space
  - Do not reorganize after update
    - Need a bitmap indicating which slots are in use
System Catalogs

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

Catalogs are themselves stored as relations!
### Attr_Cat(attr_name, rel_name, type, 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>
Indexes (a sneak preview)

- A Heap file allows us to retrieve records:
  - by specifying the rid, or
  - by scanning all records sequentially

- Sometimes, we want to retrieve records by specifying the values in one or more fields, e.g.,
  - Find all students in the “CS” department
  - Find all students with a gpa > 3

- Indexes are file structures that enable us to answer such value-based queries efficiently.
Summary

- Disks provide cheap, non-volatile storage.
  - Random access, but cost depends on the location of page on disk; important to arrange data sequentially to minimize *seek* and *rotation* delays.
DBMS vs. OS File Support

- DBMS needs features not found in many OS’s, e.g., forcing a page to disk, controlling the order of page writes to disk, files spanning disks, ability to control pre-fetching and page replacement policy based on predictable access patterns, etc.
- Variable length record format with field offset directory offers support for direct access to i’th field and null values.