Storing Data: Disks and Files

Instructor: Chen Li
Today’s Topics

- Course overview
- Project 1 overview
- Files of records
Course Overview
Project 1 Overview
Disks and Files

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

- **Costs too much.** Dell wants (in early 2014) $65 for 500GB of disk, $600 for 256GB of SSD, and $57 for 4GB of RAM ($0.13, $2.34, $14.25 per GB)

- **Main memory is volatile.** We want data to be saved between runs. (Obviously!)

- Your typical (basic) 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)

- And we also have L1 & L2 caches, SSD, …
Storage Hierarchy & Latency (Jim Gray): How Far Away is the Data?

10^9 Tape / Optical Robot 2,000 years

10^6 Disk 2 years

100 Memory 1.5 hr

10 On Board Cache 10 min

2 On Chip Cache

1 Registers 1 min
Disks

- Secondary storage device of choice.
- Main advantage over tapes: *random access* vs. *sequential*.
- Data is stored and retrieved in units called *disk blocks* or *pages*.
- Unlike RAM, time to retrieve a disk page varies depending upon location on disk.
  - Therefore, relative placement of pages on disk has a *major* impact on DBMS performance!
  - (SSDs simplify things a bit in this respect)
Components of a Disk

- The platters spin (5400 rpm)
- The arm assembly is moved in or out to position a head on a desired track
- Tracks under heads form a cylinder (imaginary!)
- Only one head reads/writes at any one time.

- Block size is a multiple of sector size (which is fixed)
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 1 msec per 4KB page *(old)*

- 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`) in order to minimize seek and rotational delay

- For a sequential scan, *prefetching* several pages at a time is a big win!
RAID (Redundant Array of Inexpensive Disks)

- Disk Array: Arrangement of several disks that gives abstraction of a single, large disk.
- Goals: Increase performance and reliability.
- Two main techniques:
  - Data striping: Data is partitioned; size of a partition is called the striping unit. Partitions are distributed over several disks.
  - Redundancy: More disks => more failures. Redundant information allows reconstruction of data if a disk fails.
RAID 0: No redundancy (just striping)
RAID 1: Mirrored (two identical copies)
Disk Space Management

- Lowest layer of DBMS software manages the space on disk.
- Higher levels call upon this layer to:
  - allocate/de-allocate a page
  - read/write a page
- A 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.
Next topic: Files of Records

- Page or block is OK when doing I/O, but higher levels of DBMS operate on records, and thus want files of records.

- **FILE**: A collection of pages, each containing a collection of records. Must support:
  - Insert (append)/delete/modify record
  - Read a particular record (specified using record id)
  - Scan all records (possibly with some conditions on the records to be retrieved)
Example

CREATE TABLE Emp(id INT, gender CHAR(1), name VARCHAR(30), Salary float );
Record Formats: Fixed Length

- Information about field types is the same for all records in file; it is stored in the system catalogs. (Note: Record field info in Project 1 passed in “from above”…!)
- Finding the $i^{th}$ field of a record does not require scanning the record.
Record Formats: Variable Length

- Several alternative formats (# fields is fixed):

  Fields Delimited by Special Symbols
  
  Fields Preceded by Field Lengths

Some thought questions for you:
(1) What’s true of the second format but not the first?
(2) What annoying disadvantage do both formats share?
(3) And, how do we know the field count in each case?
Record Formats: Variable Length (continued)

- Variable-length fields with a directory:
  
  This format:
  
  (1) Offers direct access to the $i$'th field.
  (2) Helps support efficient storage of null values. (Q: How?)
  (3) Just requires a small directory overhead.
  (4) Can even help with ALTER TABLE ADD COLUMN! (Q: How?)

Array of field offsets (a.k.a. directory)
Record Formats: Variable Length
(further continued 😊)

- More variations on a theme...

Addition of null flags:

<table>
<thead>
<tr>
<th>4</th>
<th>0000</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>v1</td>
<td>v2</td>
<td>v3</td>
<td>v4</td>
</tr>
</tbody>
</table>

Inlining of fixed-size fields:

<table>
<thead>
<tr>
<th>4</th>
<th>0000</th>
<th>(F1)</th>
<th>F2</th>
<th>(F3)</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>l1</td>
<td>v2</td>
<td>l3</td>
<td>v4</td>
</tr>
</tbody>
</table>

|     |      |      | v1 |      | v3 |