Today

- DB Internals: Files

Assignments

- HW10 due
- HW11 out (due next Thurs)
- Extra Credit out
- Proj 3 out
High-level database architecture

[Diagram showing the components of a database system, including Web Forms, Application Front Ends, SQL Interface, SQL Commands, DBMS with Plan Executor, Parser, Operator Evaluator, Optimizer, File and Access Methods, Buffer Manager, Disk Space Manager, Transaction Manager, Lock Manager, Recovery Manager, Index Files, Data Files, System Catalog, and Storage (on disk)].
Up to now: Everything above the DBMS box ...

Rest of course: Quick dive into the box ...

- Start with data storage on disk ("Classical" DBMS approach)
- Warning: broad overview and generalities

10,000 foot view of query execution

- Given an SQL query
- Translate it to an internal representation (relational algebra)
- Find a set of possible (equivalent) query plans
  - different orderings of SQL operators (select, from, where, ...)
  - different algorithms ("implementations") for executing each operator
- Pick a good plan (based on estimated cost)
- Execute the plan ...

Questions we’ll look at

- How is data stored on disk?
- And then how are operators implemented? (if time allows)
Query cost usually measured in **number of page I/Os**

- often simplified to assume each page I/O costs the same

**Why fuss over disk access time?**

- worst case, not all data can fit into main memory
  - and/or minimize memory “budget”
  - e.g., allow “small” amount for DBMS (which means more for apps)

- lets say disk access time is about **5 milliseconds** (ms)
  - range for HDDs is 2.5ms to 10ms
  - SSDs often under 0.1ms

- whereas memory access time is about **50 nanoseconds** (ns)
  - range from 50ns to 150ns
  - $5 \text{ ms} = 5,000,000 \text{ ns}$
  - thus disk access is (about) **100,000 times slower** than memory access

- Contrast this with:
  - 1 sec. to pick up a piece of paper
  - 100,000 seconds to drive (fast) to San Francisco and back ($\approx 28$ hours)
Various notions of a “Page” in a DBMS

“Hardware” Page = chunk of data read/written at one time to/form disk

“OS” Page (aka “Block”) = chunk of data used for virtual mem/paging
• usually same size or multiple of a page

“Database” Page = chunk of data managed by DBMS
• also one or more hardware pages
• e.g., IBM DB2/Oracle 4kb, SQLServier/Postgres 8kb, MySQL 16kb
DBMS File Storage Management

Each database stored in one or more files

• The OS treats these as standard files
• But, the DBMS handles all runtime access and management
• Files organized as a collection of database pages
• Each page has a unique page id
• Each tuple assigned a unique record id (e.g., page-id + offset)

DBMSs try to minimize number of page (block) transfers

• e.g., by keeping as many pages as possible in main memory
• the buffer is the portion of main memory for storing disk pages
• the buffer manager is responsible for allocation/managing buffer space

Buffer manager

• when a block is requested ...
  – provides address of block in main memory (if in the buffer)
• if not in main memory, buffer manager adds it ...
  – replaces (throws out) other blocks to make space
  – thrown out block written back to disk if modified (since last write)
  – once space is allocated, reads in the block from disk to the buffer
  – returns the address
Database File Organization

Approaches we'll discuss

- Files organized as Heaps (collection of records organized as pages)
- Indexes

Basic idea of heap files

- If we assume a fixed record size (more later)

<table>
<thead>
<tr>
<th>File Header</th>
<th>Record 1</th>
<th>Record 2</th>
<th>...</th>
<th>Record n</th>
<th>Record 1</th>
<th>Record 2</th>
<th>...</th>
<th>Record n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Each page (block) has exactly $n$ records

• Header keeps track of various information (checksums, page size, etc.)
We’ll often draw pages (blocks) like this:

```
Block 1
<v1, v2, ..., vn>
<v1, v2, ..., vn>
<v1, v2, ..., vn>
<v1, v2, ..., vn>
<v1, v2, ..., vn>

Block 2
<v1, v2, ..., vn>
<v1, v2, ..., vn>
<v1, v2, ..., vn>
<v1, v2, ..., vn>
<v1, v2, ..., vn>
```

Managing pages in a file (fixed-length records)

- Store record $i$ at byte $s \times (i - 1)$ \hspace{1cm} ($s$ is record size)
- Record access is simple ... (unless record larger than a block$^1$)

Q: How should we delete record $i$?

- Option 1: Move (shift) records $i + 1$, ..., $n$ to $i$, ..., $n - 1$
- Option 2: Move record $n$ to $i$
- Option 3: Just delete $i$ and keep a free list of available slots

$^1$Most DBMSs support this through “overflow pages”