Today

- DB Internals: Files

Assignments

- HW 10 out, PROJ 3 out
High-level database architecture

- Web Forms
- Application Front Ends
- SQL Interface
  - SQL Commands
  - DBMS
    - Plan Executor
    - Query Evaluation Engine
    - Parser
    - Operator Evaluator
    - Optimizer
    - Transaction Manager
    - Lock Manager
    - Concurrent Control
  - File and Access Methods
  - Buffer Manager
  - Disk Space Manager
  - Recovery Manager
- Storage (on disk)
  - System Catalog
  - Index Files
  - Data Files
Onto Database Internals

Up to now:

- Everything above the DBMS box ...
- The rest of the course ... everything else (as much as we can get to)

The plan

- Start on data storage on disk ("Classical" DBMS approach)
- Warning: broad overview and generalities
- These topics covered more in depth in an OS class

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?
Don’t do this at home!
Basic components of a disk

- Platters
- Disk head
- Arm assembly
- Arm movement
- Spindle
- Sector
- Tracks
- Platters
Types of disks

- HDD = Hard Drive Disk
- SSD = Solid State Disk ... totally different but uses similar “specs”

Platters are always spinning

- e.g., 7200 rpm for HDDs
- no platters in SSDs

A head (typically) can only read or write at any one time

- Most platters are double-sided (so, e.g., 3 platters = 6 heads)
- Only one head reads/writes at a time
- various “scheduling strategies” for storing data across platters

To read a record

- position arm (seek)
- engage head and wait for data to spin by
- read (transfer data)

Each track is made up of fixed-size sectors

- Page size is a multiple of sector size
- page size is the “unit of transfer” to/from disk
- the size depends on system and configuration ... typically 512 bytes or 4KB
- 4kb on ada
Cost of accessing data on disk

Time to access (read/write) data

- **seek time** = moving arms to position disk head on track
- **rotational delay** = waiting for sector to rotate under head
- **transfer time** = actually moving data to/from disk surface

Q: is random I/O more expensive than sequential I/O?

- yes because of seek and rotational delays

**Key to lowering I/O cost:** reduce seek & rotational delays!

- you have to wait for transfer times no matter what
- ... data compression can help though
- where you place pages on disk can make a big difference
- ... e.g., sequentially on same track
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?**

- lets say disk access time (all 3 costs) 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/from 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 (older systems didn’t do this)
- 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)

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:

Managing pages in a file (fixed-length records)

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

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”
Heap File Storage and Slotted Pages

Issues with the previous approach

- variable length records
- dealing with deletions

Slotted Pages

- Most commonly used layout scheme
- Each page starts with a header
- Records are stored from back to front of page
- The page header:
  - keeps track of the number of entries ("slots") ... each slot is a record
  - an offset (pointer) to the start of free space
  - and the offset of the start of each record
- records can be moved to keep them contiguous (header updated as needed)
- each tuple is stored as a sequence of bytes
  - attributes typically stored in same order as create table statement