Lecture 2: Storage Management
Recap
Why you should take this course?

- You want to learn how to make database systems **scalable**, for example, to support web or mobile applications with millions of users.
- You want to make applications that are highly **available** (*i.e.*, minimizing downtime) and operationally robust.
- You have a natural curiosity for the way things work and want to know what goes on inside major websites and online services.
- You are looking for ways of making systems easier to maintain in the long run, even as they grow and as requirements and technologies change.
- If you are good enough to write code for a database system, then you can write code on almost anything else.
History of Database Systems

- **1960s**: Hierarchical data model (Tree)
- **1970s**: Network data model (Graph)
- **1980s**: Relational data model (Relation)
- **1990s**: Object-Oriented Data Model
- **2000s**: Data Warehouses – OLAP workload
- **2000s**: NoSQL systems
- **2010s**: NewSQL systems
- **2010s**: Hybrid systems – OLTP + OLAP workload
- **2010s**: Cloud systems
- **2020s**: Specialized systems (e.g., Time Series DBMSs, GPU-based DBMSs)
Today’s Agenda

• Programming Assignments
• Anatomy of a DBMS
• Hardware Properties
• Storage Management
Programming Assignments
Machine Setup

- Operating System (OS): Ubuntu 18.04
- **Build System**: cmake
- Testing Library: Google Testing Library (gtest)
- **Continuous Integration (CI) System**: Gradescope
- Memory Error Detector: valgrind memcheck
C++ Topics

- STL map
- File I/O
- Threading (later assignments)
- Smart Pointers (later assignments)
Assignment 1

- **Goal:** Help brush up your C++ programming skills
- Design a program to locate the occurrences of a word in a *text file*.
- Knowledge of basic data structures and algorithm design
Anatomy of a DBMS
Anatomy of a Database System [Monologue]
Anatomy of a Database System [Monologue]

- **Process Manager**
  - Manages client connections
- **Query Processor**
  - Parse, plan and execute queries on top of storage manager
- **Transactional Storage Manager**
  - Knits together buffer management, concurrency control, logging and recovery
- **Shared Utilities**
  - Manage hardware resources across threads
Anatomy of a Database System [Monologue]

- Process Manager
  - Connection Manager + Admission Control
- Query Processor
  - Query Parser
  - Query Optimizer (a.k.a., Query Planner)
  - Query Executor
- Transactional Storage Manager
  - Lock Manager
  - Access Methods (a.k.a., Indexes)
  - Buffer Pool Manager
  - Log Manager
- Shared Utilities
  - Memory, Disk, and Networking Manager
The Problem
Requirements

There are different classes of requirements:

- **Data Independence**
  - application logic must be shielded from physical storage implementation details
  - physical storage can be reorganized
  - hardware can be changed

- **Scalability**
  - must scale to (nearly) arbitrary data size
  - efficiently access to individual tuples
  - efficiently update an arbitrary subset of tuples

- **Reliability**
  - data must never be lost
  - must cope with hardware and software failures

- ...
Layered Architecture

- implementing all these requirements on “bare metal” is hard
- and not desirable
- a DBMS must be maintainable and extensible

Instead: use a **layered architecture**
- the DBMS logic is split into levels of functionality
- each level is implemented by a specific layer
- each layer interacts only with the next lower layer
- simplifies and modularizes the code
A Simple Layered Architecture

Purpose

- query translation and optimization
- managing records and access paths
- DB buffer and hardware interface

Access Granularity

- declarative queries
- sets of records
- records
- page
A Simple Layered Architecture (2)

- layers can be characterized by the data items they manipulate
- lower layer offers functionality for the next higher level
- keeps the complexity of individual layers reasonable
- rough structure: physical $\rightarrow$ low level $\rightarrow$ high level

This is a reasonable architecture, but simplified.
A more detailed architecture is needed for a complete DBMS.
A More Detailed Architecture

Query Interface
- SQL,...
  - granularity: relation, view, ...
  - data structures: logical schema, integrity constraints
  - granularity: logical record, key, ...

Record Interface
- FIND NEXT record, STORE record
  - granularity: logical record, key, ...
  - data structures: access path, physical schema ...
  - granularity: physical record, ...

Record Access
- write record, insert in B-tree,...
  - granularity: physical record, ...
  - data structures: free space inventory, page indexes ...
  - granularity: page, segment

DB Buffer
- access page j, release page j
  - granularity: page, segment
  - data structures: page table, block map ...
  - granularity: block, file

File Interface
- read block k, write block k
  - granularity: block, file
  - data structures: free space inventory, extent table ...
  - granularity: track, cylinder, ...

Device Interface
- DB
  - granularity: relation, view, ...
  - data structures: logical schema, integrity constraints
  - granularity: logical record, key, ...
  - physical record, ...

Device Interface
- DB
  - granularity: relation, view, ...
  - data structures: logical schema, integrity constraints
  - granularity: logical record, key, ...
  - physical record, ...

Storage Management

Anatomy of a DBMS
Hardware Properties
Impact of Hardware

Must take hardware properties into account when designing a storage system.

For a long time dominated by Moore’s Law:

*The number of transistors on a chip doubles every 18 month.*

Indirectly drove a number of other parameters:

- main memory size
- CPU speed
  - no longer true!
- HDD capacity
  - start getting problematic, too. density is very high
  - only capacity, not access time
Memory Hierarchy

- **Capacity**
  - bytes
  - K-M bytes
  - G bytes
  - T bytes
  - T bytes
  - T-P bytes

- **Latency**
  - 1ns
  - <10ns
  - <100ns
  - ms
  - sec
  - sec-min

AWS Glacier
Memory Hierarchy (2)

There are huge gaps between hierarchy levels

- traditionally, main memory vs. disk is most important
- but memory vs. cache etc. also relevant

The DBMS must aim to maximize locality.
Non-Uniform Memory Access
Hard Disk Access

Hard Disks are still the dominant external storage:

- rotating platters, mechanical effects
- transfer rate: ca. 150MB/s
- seek time ca. 3ms
- huge imbalance in random vs. sequential I/O!
Hard Disk Access (2)

The DBMS must take these effects into account

- sequential access is much more efficient
- traditional DBMSs are designed to maximize sequential access
- gap is growing instead of shrinking
- even SSDs are slightly asymmetric (and have other problems)
- DBMSs try to reduce number of writes to random pages by organizing data in contiguous blocks.
- Allocating multiple pages at the same time is called a segment
Hard Disk Access (3)

Techniques to speed up disk access:

- do not move the head for every single tuple
- instead, load larger chunks. typical granularity: one page
- page size varies. traditionally 4KB, nowadays often 16K and more (trade-off)
Hard Disk Access (4)

The page structure is very prominent within the DBMS

• granularity of I/O
• granularity of buffering/memory management
• granularity of recovery

Page is still too small to hide random I/O though

• sequential page access is important
• DBMSs use read-ahead techniques
• asynchronous write-back
Database System Architectures

Storage Management

Disk-Centric Database System
- The DBMS assumes that the primary storage location of the database is HDD.

Memory-Centric Database System (MMDB)
- The DBMS assumes that the primary storage location of the database is DRAM.

Buffer Management
The DBMS’s components manage the movement of data between non-volatile and volatile storage.
## Access Times

<table>
<thead>
<tr>
<th>Access Time</th>
<th>Hardware</th>
<th>Scaled Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5 ns</td>
<td>L1 Cache</td>
<td>0.5 sec</td>
</tr>
<tr>
<td>7 ns</td>
<td>L2 Cache</td>
<td>7 sec</td>
</tr>
<tr>
<td>100 ns</td>
<td>DRAM</td>
<td>100 sec</td>
</tr>
<tr>
<td>350 ns</td>
<td>NVM</td>
<td>6 min</td>
</tr>
<tr>
<td>150 us</td>
<td>SSD</td>
<td>1.7 days</td>
</tr>
<tr>
<td>10 ms</td>
<td>HDD</td>
<td>16.5 weeks</td>
</tr>
<tr>
<td>30 ms</td>
<td>Network Storage</td>
<td>11.4 months</td>
</tr>
<tr>
<td>1 s</td>
<td>Tape Archives</td>
<td>31.7 years</td>
</tr>
</tbody>
</table>

Source: Latency numbers every programmer should know
Storage Management
Storage Manager

- The **storage manager** is responsible for maintaining a database’s files.
  - Some do their own **scheduling** of I/O operations to improve spatial and temporal locality of pages.
- It organizes the files as a collection of pages.
  - Tracks data being read from and written to pages.
  - Tracks the available free space.
Database Pages

- A **page** is a fixed-size block of data.
  - It can contain tuples, meta-data, indexes, log records. . .
  - Most systems do not mix page types.
  - Some systems require a page to be self-contained. Why?
- Each page is given a unique identifier.
  - The DBMS uses an **indirection layer** to map page ids to physical locations.
  - This is implemented as a page directory table.
Database Pages

- There are three different notions of "pages" in a DBMS:
  - Hardware Page (usually 4 KB)
  - OS Page (usually 4 KB)
  - Database Page (512 B – 16 KB)
- By hardware page, we mean at what level the device can guarantee a "failsafe write".
Disk Block Mapping

The units of database space allocation are disk blocks, extents, and segments.

- A disk block is the smallest unit of data used by a database.
- An extent is a logical unit of database storage space allocation made up of a number of contiguous disk blocks.
- A segment is made up of one or more extents (and is hence not always contiguous on disk).
System Catalog

- A DBMS stores **meta-data** about databases in its internal catalog.
  - List of tables, columns, indexes, views
  - List of users, permissions
  - Internal statistics (e.g., disk reads, storage space allocation)

- Almost every DBMS stores their catalog as a **private database**.
  - Wrap object abstraction around tuples.
  - Specialized code for “bootstrapping” catalog tables. Why?
Data Representation

- **INTEGER/BIGINT/SMALLINT/TINYINT**
  - C/C++ Representation

- **FLOAT/REAL vs. NUMERIC/DECIMAL**
  - IEEE-754 Standard / Fixed-point Decimals

- **VARCHAR/VARBINARY/TEXT/BLOB**
  - Header with length, followed by data bytes.

- **TIME/DATE/TIMESTAMP**
  - 32/64-bit integer of (micro)seconds since Unix epoch
N-ary Storage Model (NSM)

- The DBMS stores all attributes for a single tuple contiguously in a page.
N-ary Storage Model (NSM)

```
SELECT * FROM useracct
WHERE userName = ? AND userPass = ?
```

Use index to access the particular user’s tuple.
**N-ary Storage Model (NSM)**

```
INSERT INTO useracct VALUES (?, ?, ..., ?)
```

Add the user’s tuple using `std::memcpy`.  

Diagram showing a disk page with multiple tuples and a database connection labeled 'App Memory'.

Storage Management
N-ary Storage Model (NSM)

Useless data accessed for this query.

```
SELECT COUNT(U.lastLogin), EXTRACT(month FROM U.lastLogin) AS month
FROM useracct AS U
WHERE U.hostname LIKE '%.gov'
GROUP BY EXTRACT(month FROM U.lastLogin)
```
N-ary Storage Model (NSM)

• Advantages
  ▶ Fast inserts, updates, and deletes.
  ▶ Good for queries that need the entire tuple.

• Disadvantages
  ▶ Not good for scanning large portions of the table and/or a subset of the attributes.
Decomposition Storage Model (DSM)

- The DBMS stores the values of a **single attribute** for all tuples contiguously in a page.
  - Also known as a "column store"
- Ideal for OLAP workloads where read-only queries perform large scans over a subset of the table’s attributes.
Decomposition Storage Model (DSM)

- The DBMS stores the values of a single attribute for all tuples contiguously in a page.
  - Also known as a "column store".

![Diagram of DSM Disk Page]

- Header
- userID
- userName
- userPass
- hostname
- lastLogin
SELECT COUNT(U.lastLogin), EXTRACT(month FROM U.lastLogin) AS month
FROM useracct AS U
WHERE U.hostname LIKE '%.gov'
GROUP BY EXTRACT(month FROM U.lastLogin)
Workload Characterization

- On-Line Transaction Processing (OLTP)
  - Fast operations that only read/update a small amount of data each time.
  - OLTP Data Silos
- On-Line Analytical Processing (OLAP)
  - Complex queries that read a lot of data to compute aggregates.
  - OLAP Data Warehouse
- Hybrid Transaction + Analytical Processing
  - OLTP + OLAP together on the same database instance
# Workload Characterization

<table>
<thead>
<tr>
<th>Workload</th>
<th>Operation Complexity</th>
<th>Workload Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLTP</td>
<td>Simple</td>
<td>Writes</td>
</tr>
<tr>
<td>OLAP</td>
<td>Complex</td>
<td>Reads</td>
</tr>
<tr>
<td>HTAP</td>
<td>Medium</td>
<td>Mixture</td>
</tr>
</tbody>
</table>

Source
Storage Models

- A DBMS encodes and decodes the tuple’s bytes into a set of attributes based on its schema.
- It is important to choose the right storage model for the target workload:
  - OLTP → Row-Store
  - OLAP → Column-Store
Buffer Pool Meta-Data

- The **page table** keeps track of pages that are currently in memory.
- Also maintains additional meta-data per page:
  - Dirty Flag
  - Pin/Reference Counter
Buffer Replacement Policies

- When the DBMS needs to free up a frame to make room for a new page, it must decide which page to evict from the buffer pool.
- Policies:
  - FIFO
  - LFU
  - LRU
  - CLOCK
  - LRU-k
  - 2Q
Conclusion
Parting Thoughts

- Database systems have a layered architecture.
- Design of database system components affected by hardware properties.
- Database is physically organized as a collection of pages on disk.
- The units of database space allocation are disk blocks, extents, and segments.
- The DBMS can manage that sweet, sweet memory better than the OS.
- Leverage the semantics about the query plan to make better decisions.
- It is important to choose the right storage model for the target workload.
Next Class

- Recap of access methods
- Submit exercise sheet #1 via Gradescope.