03 Database Storage
Part 1
Project 0 due Sunday. Quick adaptation to C++ is a prerequisite for this course!

Homework 1 available, due Friday Feb 3\textsuperscript{rd}.

Project 1 will be released next Monday, January 30\textsuperscript{th}.
LAST CLASS

We now understand what a database looks like at a logical level and how to write queries to read/write data (e.g., using SQL).

Unfinished business: Window functions.
WINDOW FUNCTIONS

Performs a "sliding" calculation across a set of tuples that are related.
Like an aggregation but tuples are not grouped into a single output tuples.

SELECT ... FUNC-NAME(...) OVER (...) 
FROM tableName

How to “slice” up data
Can also sort

Aggregation Functions
Special Functions
Aggregation functions:
→ Anything that we discussed earlier

Special window functions:
→ `ROW_NUMBER()` → # of the current row
→ `RANK()` → Order position of the current row.

```sql
SELECT *, ROW_NUMBER() OVER () AS row_num
FROM enrolled
```
WINDOW FUNCTIONS

The **OVER** keyword specifies how to group together tuples when computing the window function. There are many ways to define a window, e.g., **PARTITION BY** to specify a group.

```
SELECT cid, sid, ROW_NUMBER() OVER (PARTITION BY cid)
FROM enrolled
ORDER BY cid
```
WINDOW FUNCTIONS

You can also include an ORDER BY in the window grouping to sort entries in each group.

```
SELECT *,
    ROW_NUMBER() OVER (ORDER BY cid)
FROM enrolled
ORDER BY cid
```
Find the student(s) with the second highest grade for each course.

```
SELECT * FROM (  
    SELECT *, RANK() OVER (PARTITION BY cid  
        ORDER BY grade ASC) AS rank  
    FROM enrolled) AS ranking  
WHERE ranking.rank = 2
```

Group tuples by cid
Then sort by grade
LAST CLASS

We now understand what a database looks like at a logical level and how to write queries to read/write data (e.g., using SQL).

Unfinished business: Window functions

We will next learn how to build software that manages a database (i.e., a DBMS).
COURSE OUTLINE

Relational Databases
Storage
Execution
Concurrency Control
Recovery
Distributed Databases
Potpourri

Query Planning
Operator Execution
Access Methods
Buffer Pool Manager
Disk Manager
DISK-BASED ARCHITECTURE

The DBMS assumes that the primary storage location of the database is on non-volatile disk.

The DBMS's components manage the movement of data between non-volatile and volatile storage.
STORAGE HIERARCHY

- Memory
- CPU
- CPU Registers
- CPU Caches
- DRAM
- SSD
- HDD
- Network Storage

- Faster
- Smaller
- Expensive
- Slower
- Larger
- Cheaper

Volatile
Random Access
Byte Addressable

Non-Volatile
Sequential Access
Block Addressable

Persistence Memory
Fast Network Storage
ACCESS TIMES

Latency Numbers Every Programmer Should Know

1 ns L1 Cache Ref
4 ns L2 Cache Ref
100 ns DRAM
16,000 ns SSD
2,000,000 ns HDD
~50,000,000 ns Network Storage
1,000,000,000 ns Tape Archives

Source: Colin Scott
SEQUENTIAL VS. RANDOM ACCESS

Random access on non-volatile storage is almost always much slower than sequential access.

DBMS will want to maximize sequential access.
→ Algorithms try to reduce number of writes to random pages so that data is stored in contiguous blocks.
→ Allocating multiple pages at the same time is called an extent.
SYSTEM DESIGN GOALS

Allow the DBMS to manage databases that exceed the amount of memory available.

Reading/writing to disk is expensive, so it must be managed carefully to avoid large stalls and performance degradation.

Random access on disk is usually much slower than sequential access, so the DBMS will want to maximize sequential access.
DISK-ORIENTED DBMS

Get page #2

Pointer to page #2

Interpret the layout of page #2...

Lectures #3-5

Lectures #12-13

Execution Engine

Lecture #6
WHY NOT USE THE OS?

The DBMS can use memory mapping (mmap) to store the contents of a file into the address space of a program.

The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn’t need to worry about it.
MEMORY MAPPED I/O PROBLEMS

Problem #1: Transaction Safety
→ OS can flush dirty pages at any time.

Problem #2: I/O Stalls
→ DBMS doesn't know which pages are in memory. The OS will stall a thread on page fault.

Problem #3: Error Handling
→ Difficult to validate pages. Any access can cause a SIGBUS that the DBMS must handle.

Problem #4: Performance Issues
→ OS data structure contention. TLB shootdowns.
WHY NOT USE THE OS?

There are some solutions to some of these problems:

→ **madvise**: Tell the OS how you expect to read certain pages.
→ **mlock**: Tell the OS that memory ranges cannot be paged out.
→ **msync**: Tell the OS to flush memory ranges out to disk.
WHY NOT USE THE OS?

DBMS (almost) always wants to control things itself and can do a better job than the OS.
→ Flushing dirty pages to disk in the correct order.
→ Specialized prefetching.
→ Buffer replacement policy.
→ Thread/process scheduling.

The OS is not your friend.
DATABASE STORAGE

**Problem #1:** How the DBMS represents the database in files on disk.

**Problem #2:** How the DBMS manages its memory and moves data back-and-forth from disk.
TODAY'S AGENDA

File Storage
Page Layout
Tuple Layout
The DBMS stores a database as one or more files on disk typically in a proprietary format. → The OS doesn't know anything about the contents of these files.

Early systems in the 1980s used custom filesystems on raw storage. → Some DBMSs still support this. → Most newer DBMSs do not do this.
The *storage manager* is responsible for maintaining a database's files.

- Some do their own scheduling for reads and writes to improve spatial and temporal locality of pages.

It organizes the files as a collection of *pages*.

- Tracks data read/written to pages.
- Tracks the available 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.

Each page is given a unique identifier.
→ The DBMS uses an indirection layer to map page IDs to physical locations.
There are three different notions of "pages" in a DBMS:
→ Hardware Page (usually 4KB)
→ OS Page (usually 4KB)
→ Database Page (512B-16KB)

A hardware page is the largest block of data that the storage device can guarantee failsafe writes.
Different DBMSs manage pages in files on disk in different ways.

→ Heap File Organization
→ Tree File Organization
→ Sequential / Sorted File Organization (ISAM)
→ Hashing File Organization

At this point in the hierarchy we don't need to know anything about what is inside of the pages.
A heap file is an unordered collection of pages with tuples that are stored in random order.
→ Create / Get / Write / Delete Page
→ Must also support iterating over all pages.

It is easy to find pages if there is only a single file.
Need meta-data to keep track of what pages exist in multiple files.
The DBMS maintains special pages that tracks the location of data pages in the database files.
→ Must make sure that the directory pages are in sync with the data pages.

The directory also records meta-data about available space:
→ The number of free slots per page.
→ List of free / empty pages.
TODAY'S AGENDA

File Storage
Page Layout
Tuple Layout
Every page contains a header of metadata about the page's contents.

→ Page Size
→ Checksum
→ DBMS Version
→ Transaction Visibility
→ Compression Information

Some systems require pages to be self-contained (e.g., Oracle).
PAGE LAYOUT

For any page storage architecture, we now need to decide how to organize the data inside of the page. → We are still assuming that we are only storing tuples.

Two approaches:

→ Tuple-oriented
→ Log-structured ← Next Class
TUPLE STORAGE

How to store tuples in a page?

Strawman Idea: Keep track of the number of tuples in a page and then just append a new tuple to the end.

→ What happens if we delete a tuple?
→ What happens if we have a variable-length attribute?
SLOTTED PAGES

The most common layout scheme is called slotted pages.

The slot array maps "slots" to the tuples' starting position offsets.

The header keeps track of:
→ The # of used slots
→ The offset of the starting location of the last slot used.
RECORD IDS

The DBMS needs a way to keep track of individual tuples.
Each tuple is assigned a unique record identifier.
→ Most common: page_id + offset/slot
→ Can also contain file location info.

An application cannot rely on these IDs to mean anything.

PostgreSQL
CTID (6-bytes)

SQLite
ROWID (8-bytes)

Oracle
ROWID (10-bytes)
TODAY'S AGENDA

File Storage
Page Layout
Tuple Layout
A tuple is essentially a sequence of bytes.

It's the job of the DBMS to interpret those bytes into attribute types and values.
Each tuple is prefixed with a header that contains meta-data about it. → Visibility info (concurrency control) → Bit Map for NULL values.

We do not need to store meta-data about the schema.
TUPLE DATA

Attributes are typically stored in the order that you specify them when you create the table.

This is done for software engineering reasons (i.e., simplicity).

However, it might be more efficient to lay them out differently.

```
CREATE TABLE foo (  
a INT PRIMARY KEY,  
b INT NOT NULL,  
c INT,  
d DOUBLE,  
e FLOAT  
);
```
**DENORMALIZED TUPLE DATA**

DBMS can physically *denormalize* (e.g., "pre join") related tuples and store them together in the same page.

→ Potentially reduces the amount of I/O for common workload patterns.
→ Can make updates more expensive.

Not a new idea.

→ IBM System R did this in the 1970s.
→ Several NoSQL DBMSs do this without calling it physical denormalization.
CONCLUSION

Database is organized in pages.
Different ways to track pages.
Different ways to store pages.
Different ways to store tuples.
NEXT CLASS

Log-Structured Storage
Value Representation
Catalogs