Comp 521: Introduction and Overview

Instructor:

Leonard McMillan

COVID-19 ground rules

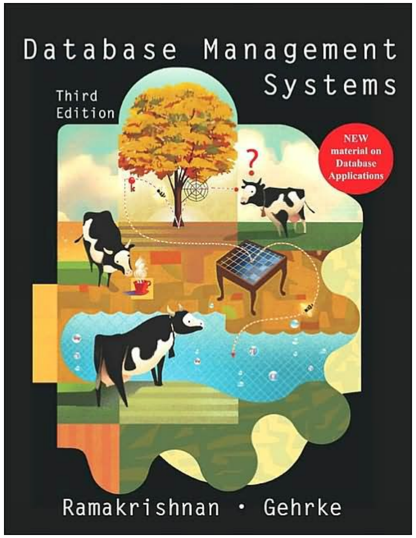
- Course meetings will be simulcast via Zoom.
  - Use it if you are exhibiting any COVID-19 symptom
- If the instructor exhibits symptoms, that day's lecture will be online and announced on the website with at least 1 hour of notice
- ALL students must adhere to UNC's health safety standards
  - Wear a mask
  - Social distancing of at least 6' while in the classroom.

COVID-19 ground rules

- NOT ALLOWED
  - Sitting in the front row
  - Entering through the door at the front of the classroom.
- Exams, problem sets, and live exercises will be online.
- Students will be responsible for cleaning and removing all materials from their desk areas at the end of each class.
- The class may at any time revert to being entirely on-line.

Course Administrivia

- Instructor
  - Leonard McMillan
- Teaching Assistant
  - Boo Fullwood
- Setup
  - Lectures simulcast in Zoom
  - Examinations outside of class
  - Unannounced in-class exercises
  - Bring your laptops/tablets to class!
- All books optional
  - Cow book (Somewhat Dated)
  - I will provide supplements for NoSQL section



Course Logistics

- Website:

  http://csbio.unc.edu/mcmillan/?run=Courses.Comp521F20

  look here first for

  - News, problem-set hints, lecture notes, and other helpful resources
  - Revisions, solutions, and corrections to problem sets
- Office Hours: Wednesdays 10am-noon
- Grading

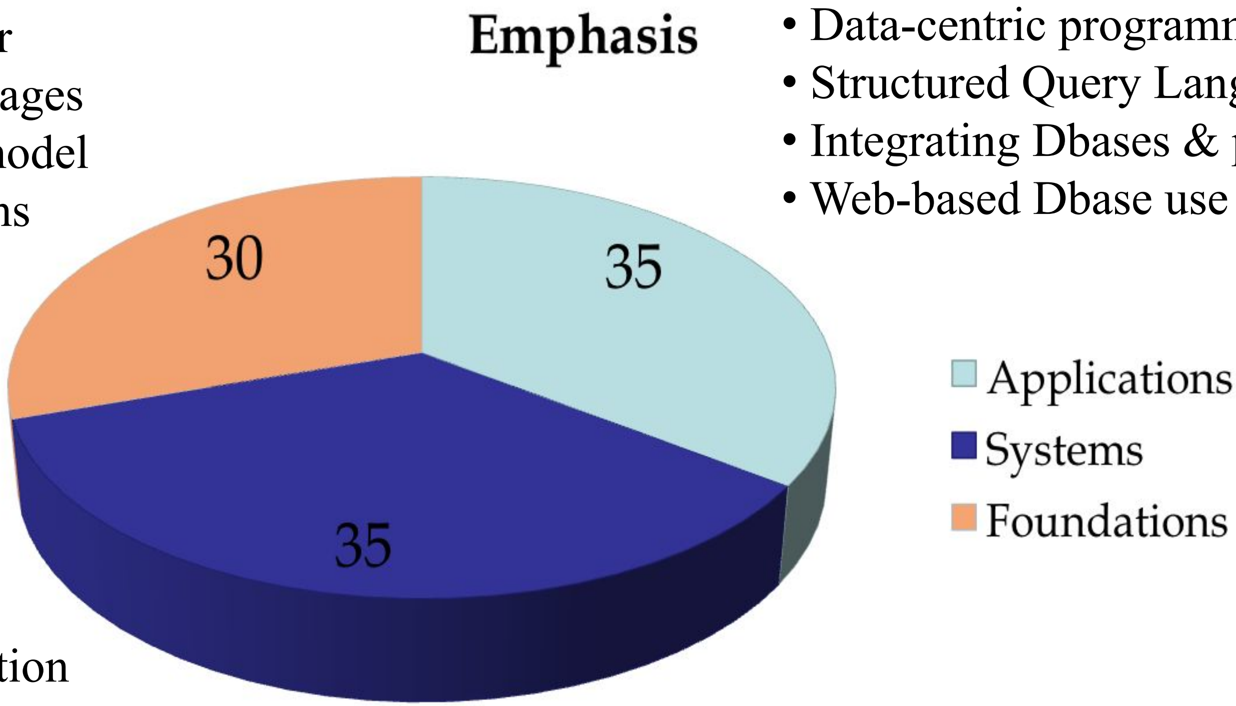
5 – Problem sets (lowest dropped)	30%
N - In-class exercises	10%
Midterm	30%
Final Exam	30%

- Problem Sets
  - Roughly one every 2 weeks

The course syllabus is available at the website

Course Breakdown

- Structured data
- Query power
- Query languages
- Relational model
- Normal forms



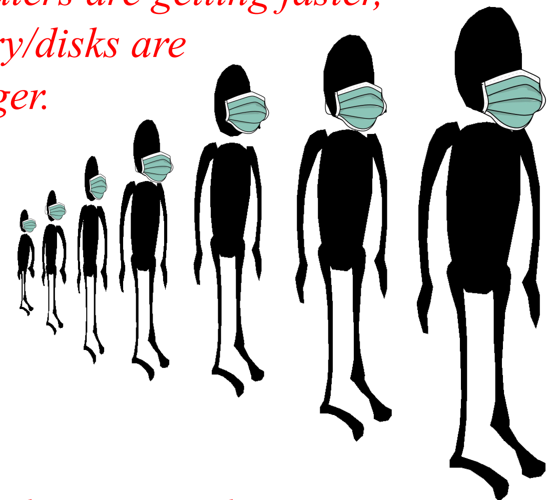
- Data-centric programming
- Structured Query Language
- Integrating Dbases & programs
- Web-based Dbase use

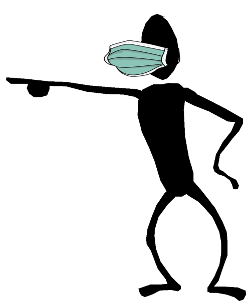
- Physical organization
- Database indexing
- Query evaluations
- Query optimization
- Transactions and concurrency

Where Databases fit into CS

- Writing Programs
  - Syntax
  - Semantics
  - Abstraction
- Designing Algorithms
  - Correctness
  - Efficiency
- Designing Data
  - Generalization
  - Portability
  - Independence
  - Robustness

Data sets are growing far faster than computers are getting faster, and memory/disks are getting larger.





Most applications today are "data-intensive", as opposed to "compute intensive". Raw CPU power is rarely a limiting factor.

What is a Database?

- A very large, integrated collection of "related and queryable" bits.
- Models real-world *enterprise.*
  - Entities (e.g., students, courses)
  - Relationships (e.g., Brittany is taking Comp 521)
- A *Database Management System (DBMS)* is a software package designed to store, access, and manage databases.

Files vs. Databases

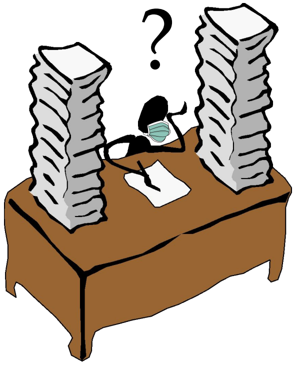
- Applications with LARGE datasets
  that won't fit in main memory and must be managed on secondary storage
- Special code for different types of queries
- Must protect data from inconsistencies caused by multiple concurrent users
- Crash recovery
  If things go wrong what is lost?
- Security and access control
  Does everyone, programmers as well as users, need to see everything?

Why use a Database?

- Data Independence.
- Efficient access.
- Reduced application development time.
- Data integrity and security.
- Uniform data administration.
- Concurrent access, recovery from crashes.

Why Study Databases??

- Shift from *compute centered* to *data centered*
  - at the “low end”: dynamic web spaces
  - at the “high end”: scientific applications
- Datasets increasing in diversity and volume.
  - Digital libraries, interactive video, Human Genome project, Earth-Observing Satellite (EOS) project
  - ... need for DBMS exploding



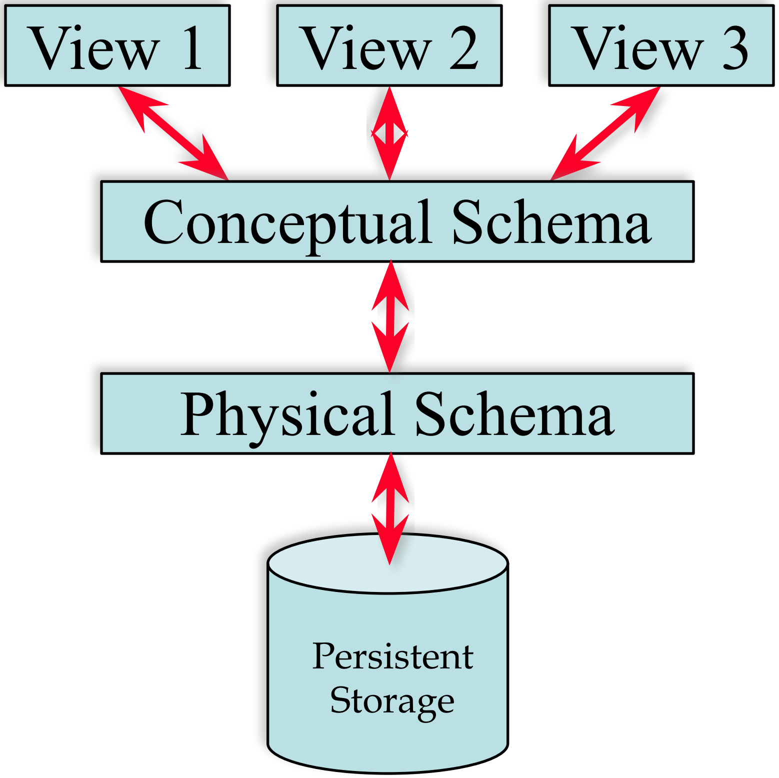
- DBMS encompasses most of CS
  - OS, languages, theory, AI, multimedia, logic

Data Models

- A *data model* is a collection of concepts relating data.
- A *schema* is a particular data organization implementing a data model.
- The *relational model of data* is the most widely used model today.
  - Main concept: *relation*, basically a table with rows and columns.
  - Every relation has a *schema*, which describes the allowed contents of columns, or fields.

Levels of Data Abstraction

- Many *views*, single *conceptual (logical) schema* and *physical schema*.
  - Views describe how users see the data.
  - Conceptual schema defines logical structure
  - Physical schema describes the files and indexes used.



- *Schemas are defined using a Data-Description Languages (DDLs); data is modified/queried using Data-Management Languages (DMLs).*

Example: University Database

- Conceptual *schema*:
  - Students *have a* ***name, pid, onyen, birthdate***
  - Courses *have a* ***course number, semester, year, credit hours, instructor***
  - Enrolled *in course connects* ***pid, course number, semester, year, grade***
- Physical *schema*:
  - How is the data stored? Students 200 disk blocks, Courses 20 blocks, Enrolled 30 blocks
  - How does one search through it or process it?
for every student scan enrolled records?
for every enrolled record scan students?
- External *schema* (View, derived values and/or customized presentations):
  - CourseSize *course number, semester, year, enrollment*
  - StudentInfo *name, semesters enrolled, gpa*

*Data Independence**

- Applications insulated from the details of how data is actually structured and stored.
- *Logical data independence*: Protection from *changes* in *logical* structure of data. For example, adding a *home address* to a student
- *Physical data independence*: Protection from changes in *physical* structure of data. Store as comma separated file or a serialized object.

* *One of the most important benefits of using a DBMS!*

Concurrency Control

- Concurrent execution of multiple user queries is essential for good DBMS performance.
  - Because disk accesses are frequent, and relatively slow, it is important to keep the cpu humming by working on several user programs concurrently.
- Interleaving actions of different user programs can lead to inconsistency: e.g., check is cleared while account balance is being computed.
- DBMS ensures such problems don't arise: users can pretend they are using a single-user system.

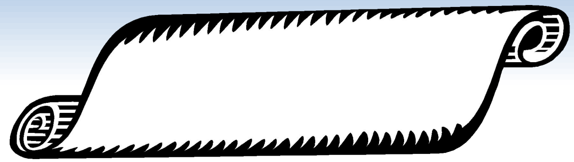
Database Transactions

- Key concept is of a *transaction (Xact)*, which is an *atomic* sequence of database actions.
- Each transaction, when executed completely, must leave the DB in a *consistent state* if DB is consistent when the transaction begins.
  - Users can specify some simple *integrity constraints* on the data, and the DBMS will enforce these constraints.
  - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
  - Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the user's responsibility!

Ensuring Atomicity

- DBMS ensure *atomicity* (all-or-nothing property) even if system crashes in the middle of a Xact.
- Idea: Keep a *log* (history) of all actions carried out by the DBMS while executing a set of Xacts:
  - Before a change is made to the database, the corresponding log entry is forced to a safe location. (Write-Ahead Log (*WAL) protocol*)
  - After a crash, the effects of partially executed transactions are *undone* using the log. (Thanks to WAL, if log entry wasn't saved before the crash, corresponding change was not applied to database!)

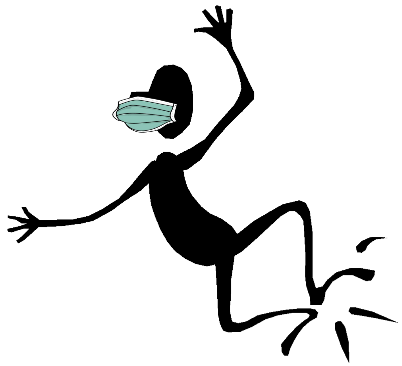
The Log



- The following actions are recorded in the log:
  - *T_i writes an object*: The old value and the new value.
    - Log record must go to disk *before* the changed page!
  - *T_i commits/aborts*: A log record indicating this action.
- Log records chained together by Xact id, so it's easy to undo a specific Xact (e.g., to resolve a deadlock).
- Log is often *duplexed* and *archived* on "stable" storage.
- All log related activities (and in fact, all CC related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

Databases make these folks happy ...

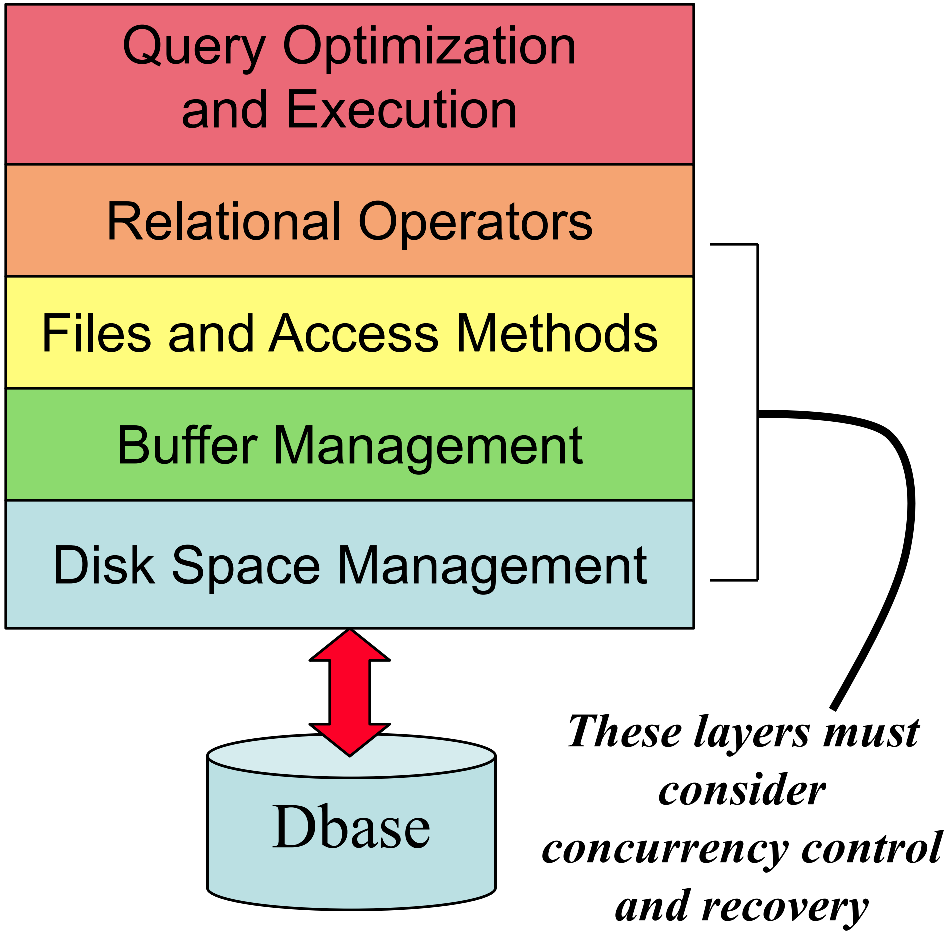
- End users (Banks, Retailers, Scientists)
- DBMS vendors (Oracle, IBM, Microsoft)
- DB application programmers
  - Makes life easier since
    Dbase provides guarantees
- *Database administrator (DBA)*
  - Designs logical/physical schemas
  - Handles security and authorization
  - Data availability, crash recovery
  - Database tuning as needs evolve



Last three must understand how a DBMS works!

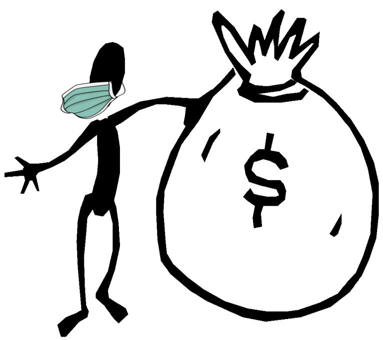
Structure of a DBMS

- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components.
- This is one of several possible architectures; each system has its own variations.



Summary

- DBMS used to maintain, query large datasets.
- Benefits include recovery from system crashes, concurrent access, quick application development, data integrity, and security.
- Levels of abstraction provide data independence.
- A DBMS typically has a layered architecture.
- DBAs hold responsible jobs and are well-paid! ☺
- DBMS R&D is one of the broadest, most exciting growth areas in CS.



Next Time

- Data Modeling
- The E-R approach