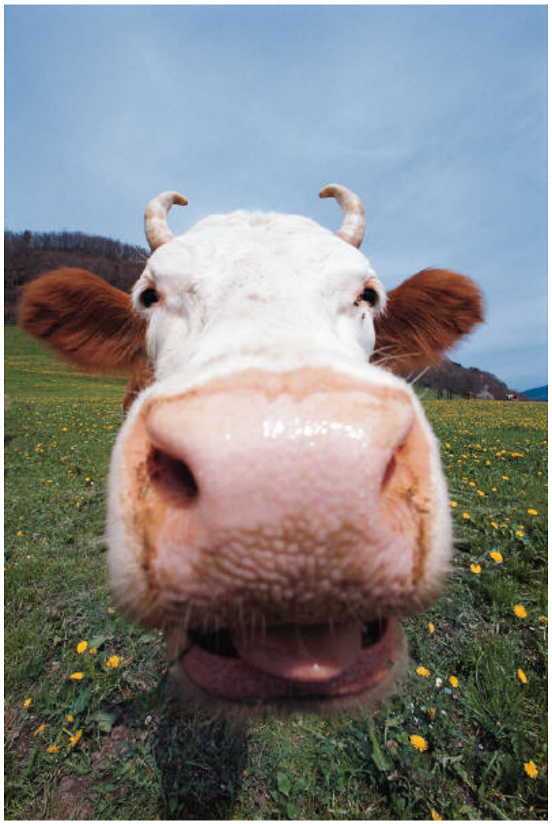


Introduction and Overview

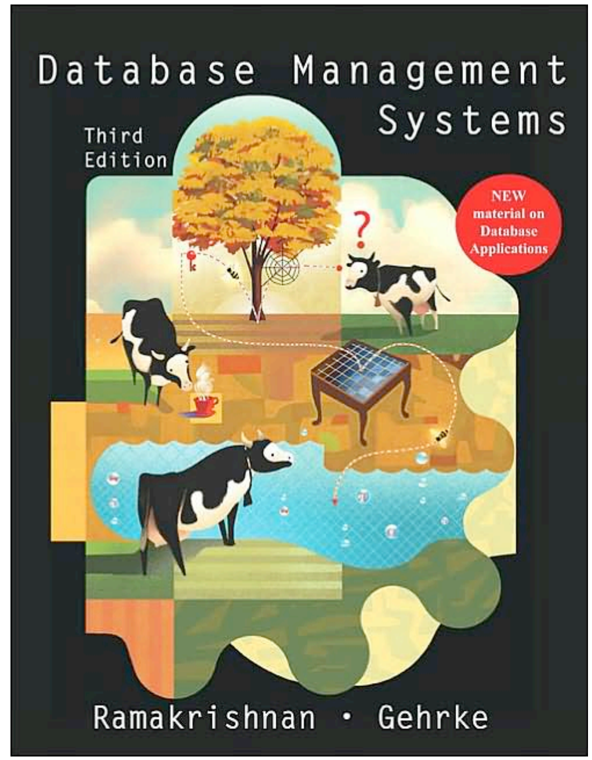
Instructors:

Leonard McMillan

Erik Scott

Course Administrivia

- Optional Book
  - Cow book
  - Somewhat Dense
  - Cover about 80%



- Instructor
  - Leonard McMillan
  - Erik Scott
- Teaching Assistant
  - To be named
- When will we meet?
  - Tuesdays and Thursdays (sans university holidays)

Course Logistics

- Website (not up yet):

  http://www.cs.unc.edu/Courses/comp521-f14

  look here first for
  - News, problem-set hints, lecture notes, and other helpful resources
  - Revisions, solutions, and corrections to problem sets
- Office Hours: TBA
- Grading
  - 5 – Problem sets (worth 6% each)
  - 2 – Midterms (worth 20% each)
  - Final Exam (worth 30%)
- Problem Sets
  - Roughly one every 2-3 weeks, except weeks with quizzes

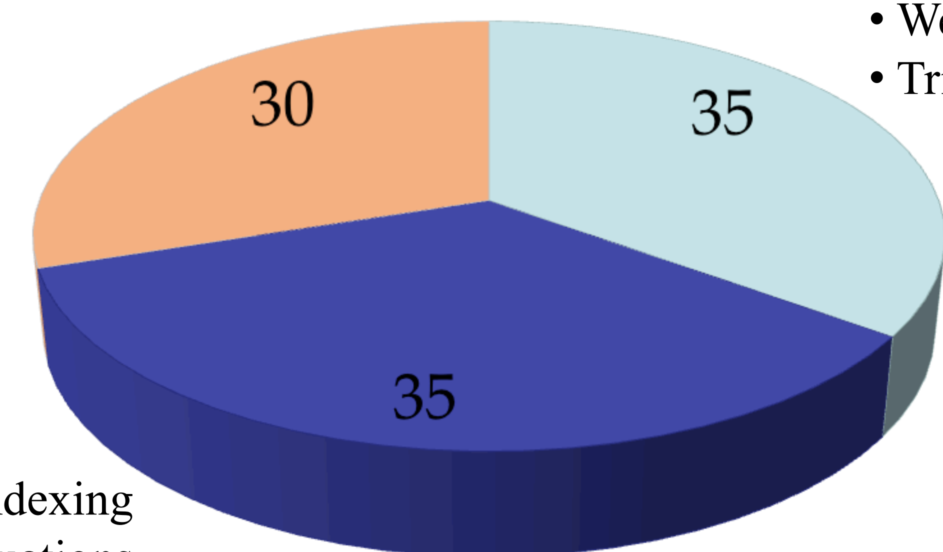
The course syllabus is available at the website

Course Breakdown

- Relational Model
- Relational Algebra
- Relational Calculus
- Normal Forms

Emphasis

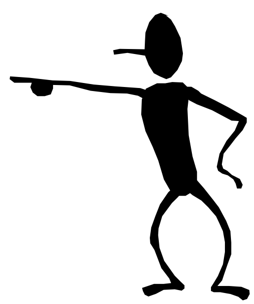
- Structured Query Language
- Integrating Dbases & programs
- Web-based Dbase use
- Triggers and Active databases



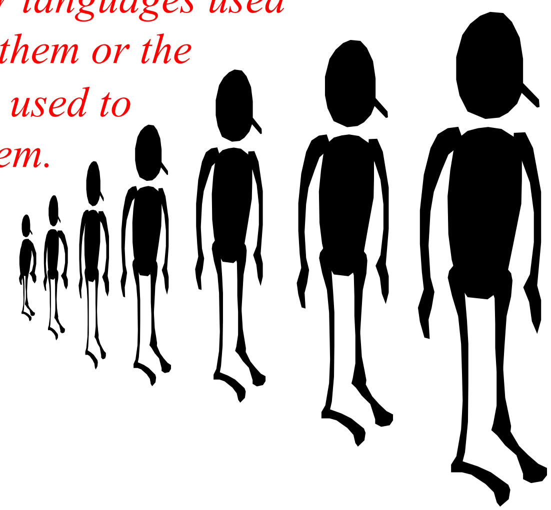
- Database Indexing
- Query Evaluations
- Query Optimization
- Transactions and Concurrency

Where Databases fit into CS

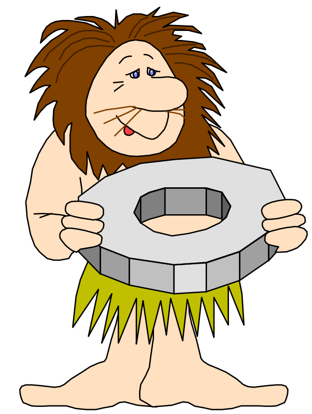
- Designing Programs
  - Syntax
  - Semantics
  - Abstraction
- Designing Algorithms
  - Correctness
  - Efficiency
- Designing Data
  - Generalization
  - Portability
  - Independence
  - Robustness



Data sets are growing far faster than either languages used to process them or the algorithms used to manage them.



What is a Database?



- A very large, integrated collection of “related” 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 and manage databases.

Files vs. Databases

- Application must stage large datasets between main memory and secondary storage (e.g., buffering, page-oriented access, 32-bit addressing, etc.)
- Special code for different queries
- Must protect data from inconsistencies caused by multiple concurrent users
- Crash recovery
- Security and access control

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 *computation* to *information*
  - 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

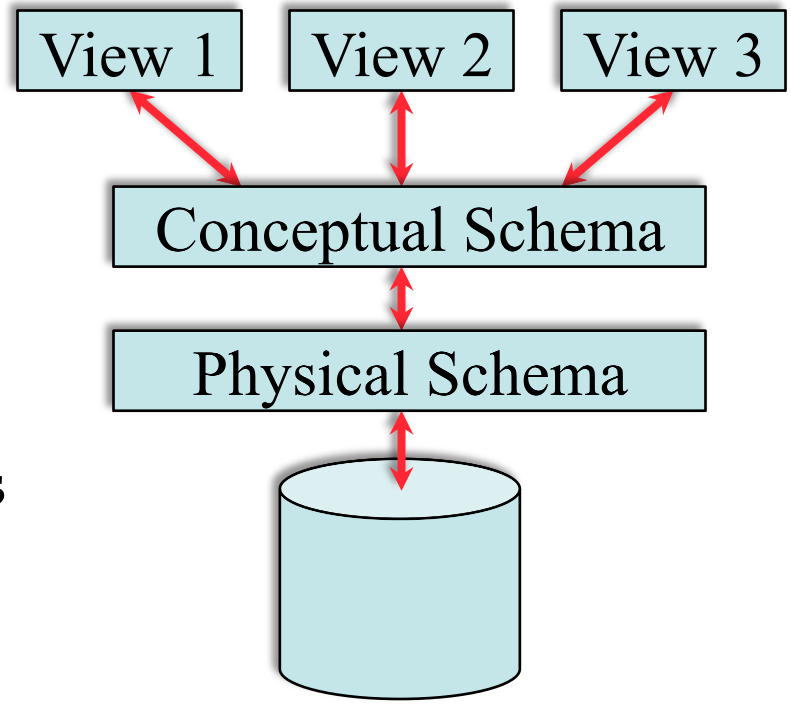
- DBMS encompasses most of CS
  - OS, languages, theory, AI, multimedia, logic

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 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(sid: string, name: string, login: string, dob: date, gpa: real)*
  - *Courses(cid: string, cname: string, credits: integer)*
  - *Enrolled(sid: string, cid: string, grade: string)*
- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.
- External Schema (View):
  - *Course_info(cid: string, enrollment: integer)*

*Data Independence**

- Applications insulated from how data is actually structured and stored.
- *Logical data independence*: Protection from changes in *logical* structure of data.
- *Physical data independence*: Protection from changes in *physical* structure of data.

☛ *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, 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!

Scheduling Concurrent Transactions

- DBMS ensures that execution of {T1, ... , Tn} is equivalent to some *serial* execution T1′ ... Tn′.
  - Before reading/writing an object, a transaction requests a lock on the object, and waits till the DBMS gives it the lock. All locks are released at the end of the transaction. (Strict Two-Phase Locking (2PL) protocol.)
  - Idea: If an action of Ti (say, writing X) affects Tj (which perhaps reads X), one of them, say Ti, will obtain the lock on X first and Tj is forced to wait until Ti completes; this effectively orders the transactions.
  - What if Tj already has a lock on Y and Ti later requests a lock on Y? (Deadlock!) Ti or Tj is aborted and restarted!

Ensuring Atomicity

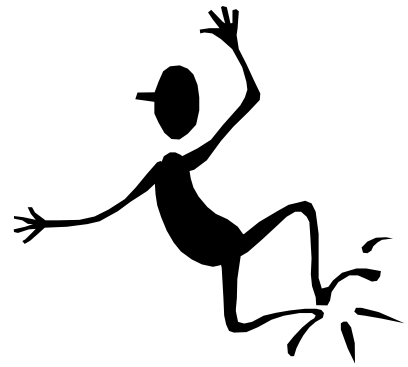
- DBMS ensure *atomicity* (all-or-nothing property) even if system crashes in the middle of a Xact.
- Idea: Keep a *log* (history) of all actions carried out by the DBMS while executing a set of Xacts:
  - Before a change is made to the database, the corresponding log entry is forced to a safe location. (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:
  - *Ti writes an object*: The old value and the new value.
    - Log record must go to disk <u>*before*</u> the changed page!
  - *Ti 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

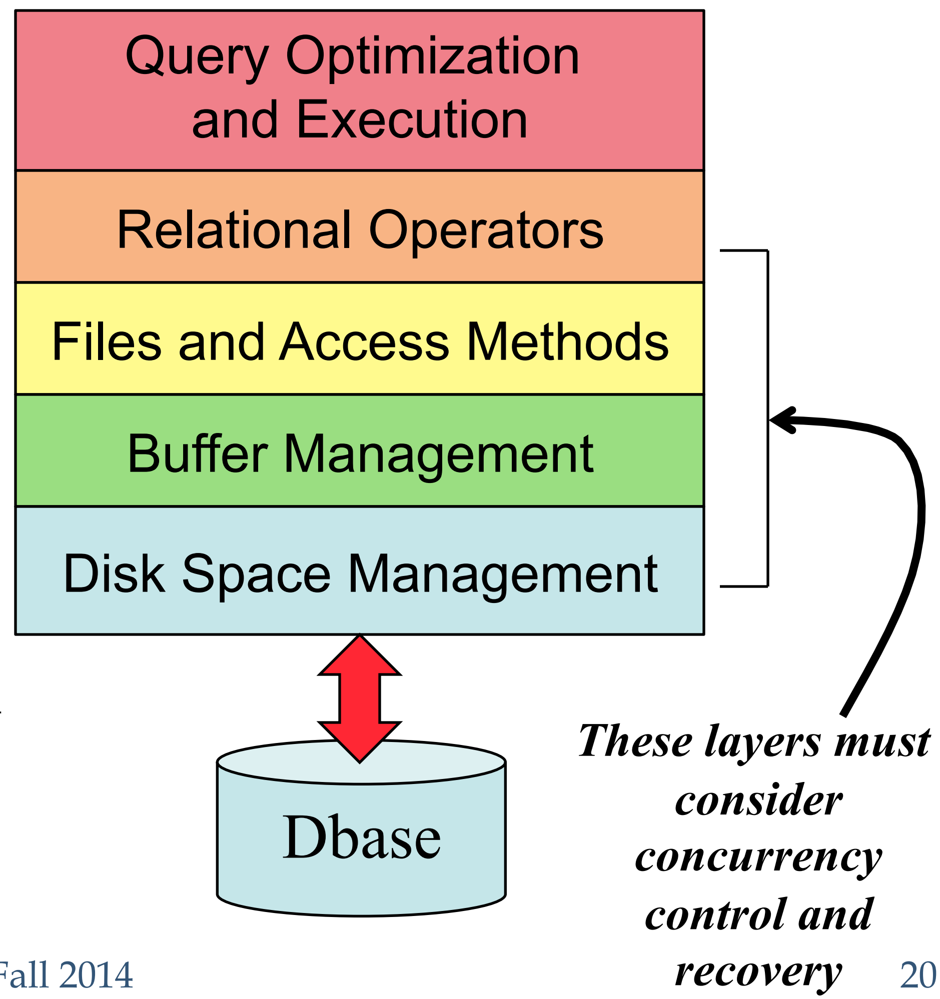


- *Database administrator (DBA)*
  - Designs logical/physical schemas
  - Handles security and authorization
  - Data availability, crash recovery
  - Database tuning as needs evolve

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

- Modeling Data
- The Entity-Relationship (ER) model