EECS 341 Spring 209
Introduction to Databases

Instructor: Éamon Johnson, ebj8 @ case, Olin 504

Classroom: Rockefeller 301 T-TH 2:30 - 3:45 pm

Web page: on Canvas (https://canvas.case.edu/)

Office hours: TBD

TAs: TBD

Course Style

- **Basic Material Course**

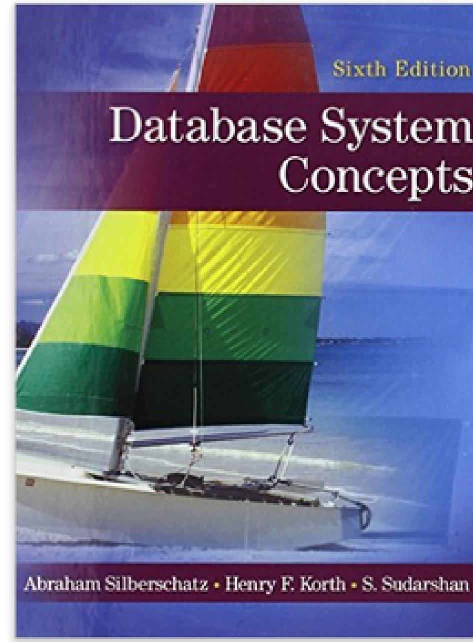
- **Project-oriented course**
  You can learn a lot of project-related stuff in this course—or, a lot less depending on how much time you put into it.
  There have been excellent projects completed in this course in the past!

- Start installing your DBMS (MySQL) right away.

- Start installing Virtual Box with Ubuntu right away.

- Start learning about web-based application development.

Textbook, DBMS, Exams



Text: "Database Systems Concepts", A. Silberschatz, H. Korth, and S. Sudarshan, 6th edition 2011.

Project DBMS: MySQL

Midterm Exam: Tuesday, March 5, 2019, in class

Final Exam: Thursday, April 25, 2019, in class

Quizzes: Dates to be announced.

Project Presentations: May 7, 2019

Grading

Assignments (~6)	20%
Project	30%
Quizzes (2)	10%
Midterm exam	20%
Final exam	20%

- **Late Assignment and Project Policy:** All homeworks are due at noon on the due date. Late submissions are accepted until the midnight of the due date subject to 50% penalty.

- **Makeup Exams**: A priori notice for any exams missed is needed. Makeup exams will NOT be given except with a serious, documented medical or legal excuse. There will not be any makeup quizzes.

Course Objectives

- In-depth understanding of basic concepts in database systems including database modeling, query languages, storage structures and query optimization issues.

- In-depth knowledge of the Entity Relationship Model and the Relational Model, as well as steps in designing, populating and maintaining a database.
- In-depth knowledge of querying relational databases, and query languages, including relational algebra, relational calculus, and SQL (with aggregate functions).
- General knowledge of query processing and transaction management in databases.
- In-depth knowledge of logical database design, schema enhancement and normalization using data dependencies.

Course Objectives (2)

- Hands-on experience on building a database-enabled web-based application that involves:
  - Designing and querying a relational database,
  - Using a (commercial MySQL) database management system effectively,
  - Project: Building a web application with a database backend.
  - Building a basic user interface for the course project.

- Experience in
  - Report writing (project proposal, progress report and final project report are required),
  - Project design, demonstration and presentation,
  - Experience in working in a project team

Course Prerequisites

- Substantial experience with at least one programming language.
  EECS 233: Intro to Data Structures is a prerequisite to this course.

- Knowledge of basic data structures including trees, linked lists, arrays, priority queues, heaps.
  The course covers (somewhat) B+-trees.

- Basic knowledge of Hashing, Searching and Sorting.
  The course will use and refer to these topics without any overview--in the query processing part of the course.

- Knowledge of first-order logic (covered in EECS 302: Discrete math).
  This knowledge is needed for relational calculus coverage.

- Mathematical maturity with set theory, functions, notations, symbolisms, and algorithmic thinking.

Course Topics—First Half

- Overview of Database Systems
- Database Design-- Entity-Relationship Data Model
- Relational Data Model
- Logical Database Design, ER to Relational and Views
- Relational Algebra
- Relational Calculus
- SQL: The Database Language

Course Topics—Second Half

- Web-Enabled Database Application Development
- Relational Database Query Processing Techniques
- Dependency Theory: Relational Database Design, Data Dependencies, Normal Forms, Schema Refinement
- Transaction Management Basics
- Data Warehouses/ OLAP

What Do We *Really* Learn in This Course?

- **Database models, query languages, query processing, transaction management**

  Very important practical material!

  Competency is tested in assignments and exams.

- **Project Building; Team Work!**

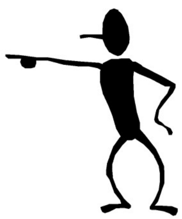
- **Report Writing**

- **A lot of informally** (yet reasonably precisely) **expressed concepts/terminology**.

How to do well

- Be resourceful!
- Install a database immediately (MySQL)
- Read the book before class, then come to class to discuss the material
- Go to office hours
- Talk to your peers
- Be honest: all words and code must be your own
  - No notes from peer discussions
  - No copying from the internet

Why do we need a database?

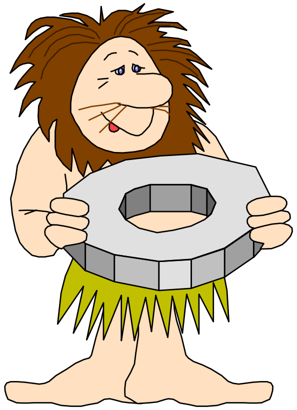


- Find me a book
  - Give me some time …
- We have what you need
  - Wait, it was gone….
- I can’t find the book
  - Maybe someone took it without telling me…

Databases: Convenient and Efficient

- Find me a book
  - Here you go!
- We have what you need
  - Happy to serve!
- I can’t find the book
  - Wait for the new order

What is a Database?



- A very large, integrated collection of bits.
- Models real-world *enterprise.*
  - Entities (e.g., students, courses)
  - Relationships (e.g., student X is taking EECS341)
- 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 inconsistency due to multiple concurrent users
- Crash recovery
- Security and access control

Major Functionality of Database

- Data Independence
  - No need for knowing physical schema
- Efficient access
  - Query language and execution optimization
- Reduced application development time
  - Unified data format, APIs, ...
- Data integrity and security
  - Role-based access control
- 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
  - the need for DBMS is exploding

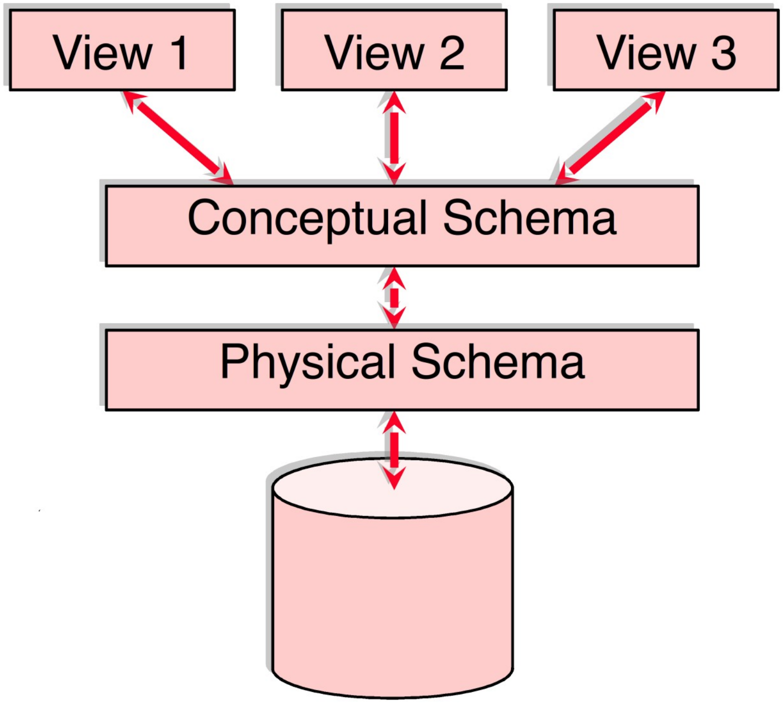
- DBMS encompasses most of CS
  - OS, languages, theory, AI, multimedia, logic

Data Models

- A *data model* is a collection of concepts for describing data.
- A *schema* is a description of a particular collection of data, using the a given 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 *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!

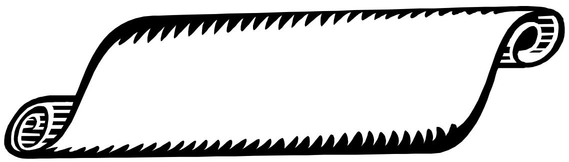
Scheduling Concurrent Transactions

- 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!)

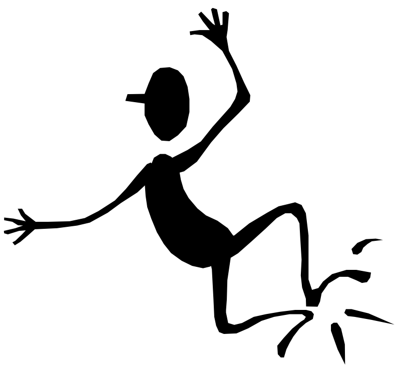
The Log



- 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 *before* 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 *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 are valuable to...

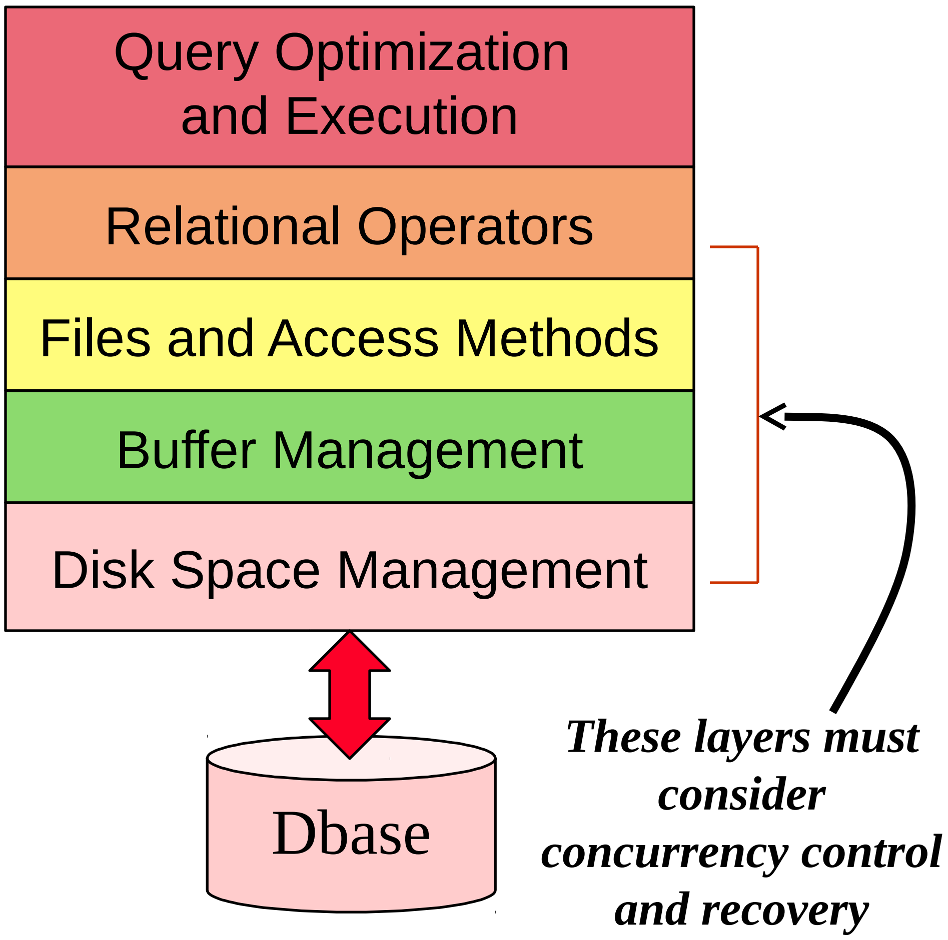
- End users (Banks, Retailers, Scientists)
- DBMS vendors (Oracle, IBM, Microsoft)
- DB application programmers
  - Makes life easier since Dbase provides guarantees
- *Database administrator (DBA)*
  - Designs logical/physical schemas
  - Handles security and authorization
  - Data availability, crash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



Last three must understand how a DBMS works!

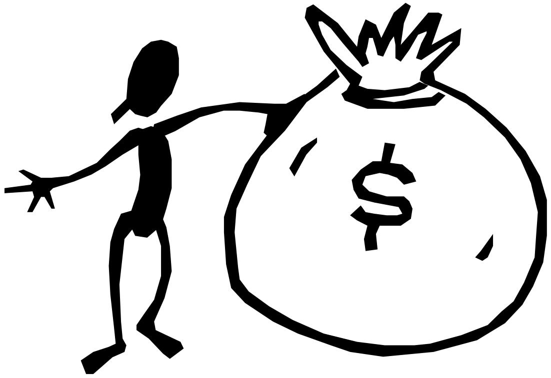
Structure of a DBMS

- 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 give data independence.
- A DBMS typically has a layered architecture.
- DBMS R&D is one of the broadest, most exciting growth areas in CS.



Next

- Introduction to Relational Model
- Modeling Data
- The Entity- Relationship (ER) model