Database Systems
cs5530/6530
Spring 2011

Claudio T. Silva
(heavily based on the course by Juliana Freire)

Today

- Course administrativia
- Why study databases?
- Why use databases?
- Intro to Databases
- Overview of the syllabus

Course Administrativia

Our Objectives in this Course

- First course on database systems: cover the *fundamental database concepts*
- Learn the practical benefits that stem from using a Database Management System (DBMS)
- Use a state-of-the-art DBMS, from designing the schema to loading data and implementing queries
- Cover some advanced topics, including: XML data management and information integration

This course is

- NOT A tutorial on how to use any specific DBMS
  - You will learn the *foundations*, and if you know the foundations you should be able to use *any* DBMS
- NOT A tutorial on SQL
  - You will learn some SQL and *relational algebra,* the algebraic query language that forms the basis for SQL
- NOT A course on database implementation: You will not implement a database system from scratch
  - You will learn how to *use* databases

Practice vs. Theory

- Aim for a balance
  - GOAL assignments: theory (and some practice)
  - Projects: practice and hands-on experience
- I hope this course was effective and that it helped you learn *fundamental database concepts as well as practical benefits of databases*

Mechanics

- Your feedback is welcome:
  - Course organization: Was wiki informative?
  - Textbook: Was it easy to follow?
  - Assignments and projects: Did they help you learn the material and prepare for the tests?
  - Exams: Did they reflect the material we covered?

Textbook

- Required: A First Course in Database Systems – 3rd Edition + GOAL access card, by Ullman and Widom
  - Resource page:
    http://infolab.stanford.edu/~ullman/fcdb.html
  - You will need GOAL access for your homework assignments.
- Other (optional) textbooks
  - Database Management Systems by Ramakrishnan and Gehrke
  - Database System Concepts by Silberschatz, Korth and Sudarshan

3rd vs. Older Editions

- Chapters have been added in 3rd edition
- Chapters have also been removed
- I recommend you get the 3rd edition

Course Format

- For all students
  - 2 lectures/week
  - Homework assignments using GOAL system
  - Programming/querying assignments--graded by TA's
  - 2 midterm exams
- Graduate students (registered for cs6530)
  - Assignments and midterms will have extra questions

Lectures

- Lecture slides in PDF format will be posted shortly before or after the lecture
  - These *complement* the lectures and the material in the textbook
- Many issues discussed in the lectures will be covered in the exams and assignments
  - Try to attend lectures regularly!
- You have to read the textbook!
  - Lectures will **not** cover all the material in the book, but all the required readings are fair materials for exams

Assignments

- Web-based + *some* programming
- GOAL assignments are interactive, you get immediate feedback!
- Programming/query assignments must be submitted electronically *(see class page for details)*
  - Students **must use** the supported systems that are available in the CADE machines, i.e., Oracle and SAXON
  - No partial grade will be given for queries that do not run in the supported systems
  - **Late submission will not be accepted**

Accounts

- You must have an account on the CADE machines
- An account will be created for you on Oracle (the latest version!)
  - More details on this later
- Make sure you receive the messages posted on the class mailing list: cs5530@list.eng.utah.edu
  - The email you currently have on have on file with ACS will be *automatically* added to the list

Project -- CS5530

- Get hands-on experience using a database system
- Apply data management techniques
- Select an application that needs a database and *build application from start to finish*
- Significant amount of programming
- Will be done in stages
  - you will submit some work at the end of each stage
- Will show a demo in the end of the semester
- More information available at
  http://www.cs.utah.edu/classes/cs5530/Project/

Tentative Grade Breakdown

- CS5530:
  - Exams 60%
  - Homework assignments 15%
  - Programming + query assignments 25%
- CS6530:
  - Exams 60%
  - Homework assignments 15%
  - programming+query assignments 25%

(Tentative) Staff and Office Hours

- Instructor: Claudio Silva
  - WEB 4893; Tue 1:40-3:40pm
- TA: TBD
  - Office hours: TBD

Communications

- *Web page: http://www.eng.utah.edu/~cs5530/*
  (not updated yet)
- ***Wiki: http://fleixeiras.cs.utah.edu/courses/index.php/Tentative_Schedule_--_Spring_2011***

Communications

- If you have questions/problems:
  - *Send email* to **teach-cs5530 [at] list [dot] eng [dot] utah [dot] edu** -- your question will be answered directly by the instructor or a TA, with a copy going to cs5530 [at] list …, and into the email archive
  - *Send email* to **cs5530 [at] list [dot] eng [dot] utah [dot] edu --** this is broadcasted to all students and course staff
- *Note that email directly to Claudio or TAs is not recommended, except for private matters*
  - Make sure to clearly indicate cs5530/6530 in the subject line
- To subscribe to the cs5530 list
  - Put your email address in the sign-up list
  - Use the Sympa web interface – link on course Web page
  https://sympa.eng.utah.edu/sympa/info/cs5530

Now, what about databases?

Why study databases?

- Databases used to be *specialized applications,* now they are a *central component* in computing environments
- Knowledge of database concepts is *essential* for computer scientists

Why study databases?

- Databases are everywhere, even when you don't see them: most activities involve data
  - Banking + credit cards: all transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Telecommunications/networks
  - Sales: customers, products, purchases
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
  - **Web sites**: front end to information stored in databases; e.g., Google, YouTube, Flickr, AddAll…
  - Scientific research, e.g., studying the environment
- Data needs to be *managed*

Why study databases?

- Data is valuable:
  - E.g., bank account records, tax records, student records, your videos and photos…
  - It must be protected - no matter what happens whether we have machine crashes, disk crashes, hurricanes/floods;
  - It also needs to be protected from **people**

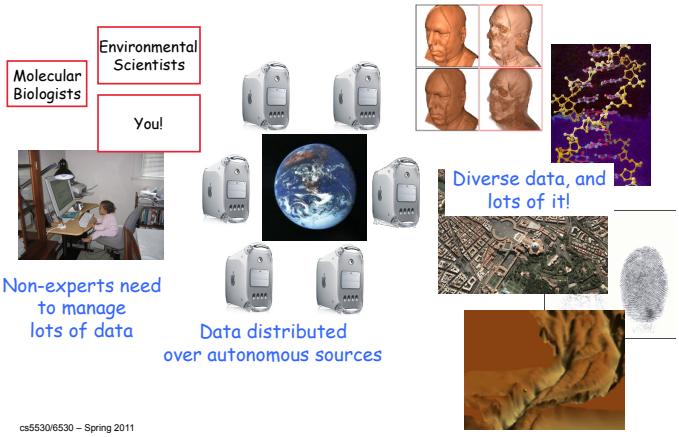
Why study databases?

- Data is often structured:
  - Bank account records all follow the same structure
  - We can exploit this regular structure
    - To retrieve data in useful ways (that is, we can use a *query* language)
    - To store data efficiently

Why study Databases?

- Because the database field has made a number of contributions to basic computer science
  - Databases are behind many of important contributions and impact that CS has had
  - Find, gather, analyze and understand data, e.g., Banks, human genome, ecommerce, Web:
- *Understand concepts and apply to different problems and different areas*
- Because DBMS software is highly successful as a commercial technology (Oracle, DB2, MS SQL Server…)
- Because DB research is highly active and **very** interesting!
  - Lots of opportunities to have practical impact

Database Technology: New Trends



Why use databases?

Slides adapted from G. Lindstrom

A Simple Data Management Problem: Address Book

- Solution 1
  - A blank notebook
  - Entries recorded in pen, in time order
- Advantages
  - Cheap, simple, private, reliable, space efficient
- Disadvantages
  - Hard to search, update (modify entries), share
  - Hard to add information, e.g., email addresses
  - Hard to extract information, e.g., Xmas card labels
  - Multiple entries are repeated (e.g., family addresses)
  - Don't lose it!

Solution 2: A Loose-leaf Notebook With n Entries Per Page

- Better:
  - Can now keep entries sorted by principal key, e.g., name
  - Insertions, deletions and updates can be done
- But:
  - All other disadvantages of Solution 1 still apply
  - In particular, no easier to search by other keys, e.g., phone number

Solution 3: A Text File, Managed By A Text Editor

- Advantages
  - Free format
  - Unlimited size
  - Easily copied (e.g., for backup, or paper dump)
  - Shareable (as a unit)
  - Substring searchable
  - Cleanly updatable
  - Powerful, programmable tools
- Solves all our problems, right?
  - Well, … what if our requirements grow?
  - These present some complications

Complication 1: File Gets **Very Large**

- Problem:
  - Searching gets slow and imprecise
  - Search for "Elm Street" yields "Wilhelm Streeter"
- Solution
  - Add indexes over fields commonly searched upon
  - Structure data into fields
    - Search for street="Elm Street"

Database Concepts:
- ***Schema***
- ***Record organization***
- ***Indexes***

Complication 2: Data Redundancy

- Why?
  - Large families, frequent moves
  - Might forget to update addresses of some family members
  - Want space economy, single point of update
  - Importance of residence as separate entity: 1 Xmas card each
- Solution:
  - Separate residences from names: 2 files, one for persons, one for residence
  - But how do we associate a residence with a person?

Database Concepts:
- ***Consistency***
- ***Normalization***
- ***Foreign keys***

Complication 3: Multiple Associations Of Persons and Residences

- Meaning:
  - People can own, rent, manage, visit residences
  - May want constraints on numbers of residences per person
- Examples:
  - many-one (single family), many-many (rich folks), one-many (builder)

Database Concepts:
- ***Relationships***
- ***Cardinality constraints***

Complication 4: Need To Add Information For New Purposes

- Examples:
  - Xmas cards sent and received
  - Post office gives big discount for using Zip+4 addressing
- Requirements:
  - Adding fields and/or new tables

Database Concept:
- ***Schema evolution***

Complication 5: Doing Ad Hoc Analysis and Retrieval

- Example:
  - "Who have we sent cards to each of the past 5 years, but received 2 or fewer cards in return?"
- Requires:
  - Language for expressing analysis and retrieval
  - Implementation that performs analysis and retrieval correctly and efficiently

Database Concepts:
- ***Query languages***
- ***Query optimization and execution***

Complication 6: Want To Organize The Data Differently For Some Users

- Examples:
  - Other family members want to see names and residences together
  - You don't want your kids to see your business entries
- Solution:
  - Use stored queries as "windows" onto the database
  - Data not selected by query is "not there"

Database Concepts:
- ***Joins***
- ***Views***
- ***Security***

Complication 7: Required Existence Of Associated Data

Database Concepts:
- ***Referential integrity***
- ***Consistency***

- Examples:
  - Can't send a Xmas card to someone without an address
  - Names are not unique unless qualified by residence: the John Jones living at 123 Elm Street
- Solutions:
  - Refuse to insert a name unless it is associated with an address
  - Refuse to delete an address if it is associated with a name
  - Or, tolerate multiple non-unique names

Complication 8: Want Programmed Access To Data

- Meaning:
  - Want to write a Java program to search, display, update entries
- Solution:
  - Use data organization to define corresponding datatypes
  - Use access library to open, retrieve, update data

Database Concepts:
- ***Database schemas***
- ***API***
- ***Embedded querying***

Complication 9: Multiple Updates On All Or None Basis

- Examples:
  - Two households merge
  - Requires changing residences of several persons
  - What if your computer crashes during updates?
- Solution:
  - Present illusion that all updates are done simultaneously
  - Implemented by commit or rollback of entire piece of work

Database Concept:
- ***Transactions***
- ***Atomicity***

Complication 10: Your Computer Crashes (Again)

- Will your data still be present
  - Uncorrupted?
  - In what state, given that a transaction was in progress?
- Solution:
  - Make sure old data are safely accessible until latest *commit*

Database Concept:
- ***Data durability***
- ***Recovery***

Complication 11: Two Computers In Your Household

- How can data be shared?
  - USB key? Ugh, multiple version headaches
  - Let's assume the database is shared somehow
  - What if one user is merging households, another is splitting one up?
  - What are meaningful results?
- A common policy:
  - Transactions are atomic
  - They appear to run one after the other, in some order

Database Concepts:
- ***Transaction isolation***
- ***Concurrency control***
- ***Transaction serializability***

Complication 12: A Home Computer And A Business Computer

- Is there one database or two?
  - Want speed, reliability of local data at each site
  - But logically, one database for maintenance and querying
  - Data communication between them (most of the time …)
  - Want some capability for independent operation (robustness)
- Solutions:
  - Personal data on the home computer
  - Business data on the business computer
  - Common logical view

Database Concepts:
- ***Distributed databases***
- ***Data partitioning***
- ***Data replication***

Complication 13: Want To Add Family Photos, Sound & Video Clips

- Those shoe boxes in the attic must go!
  - Besides, you want to be able to bore your friends at a distance

- Requirements
  - Ability to capture, store and play new media
  - Logical integration into existing data
  - Querying: all photos of Maria Beatriz smiling

Database Concepts:
- ***Multi-media data***
- ***Query by content***

Complication 13: Your Uncle Louie Gets The Genealogy Bug

- His grand vision:
  - All family members pool their databases over the Internet
  - Together, all genealogy relationships can be recorded
- But:
  - Aunt Sarah is paranoid: will not reveal birthdates
  - You are too: you don't want your business associates in the genealogy database
  - Everyone wants complete control over safety of their own data
  - People use different formats for records, and different name abbreviations for entries

Database Concepts:
- ***Federated databases***
- ***Data integration***

Complication 14: You Become President

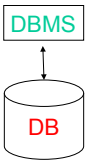
- Of USA, of University, of a large organization
  - Your address list grows to hundreds of thousands or more
  - You realize it contains useful information *in the large*
- Examples
  - Which are top 10 zip codes on the list?
  - Which zip codes have addresses that are most likely to send cards to you when you send cards to them?
  - Which of those zip codes are in states that had less than 5% difference in Republican / Democratic presidential votes in 2004?

Database Concepts:
- ***Data mining***
- ***Online analytical processing***

Database Systems: The Basics

Databases and Database Management Systems

- Database (DB) is an integrated collection of data
  - Models real-world objects
    - Entities (e.g., people, residence, Christmas cards)
    - Relationships (e.g., John Doe lives on 123 Elm St)
  - Captures *structure* – allows data to be queried
- A Database Management System (DBMS) is a software suite designed to store and manage databases
  - Provides environment that is both *convenient* and *efficient* to use.
  - Address all *complications* discussed



Storing Data: Database vs File System

- Once upon a time database applications were built on top of file systems…
- But this has many drawbacks:
  - Data redundancy, inconsistency and isolation
    - Multiple file formats, duplication of information in different files
  - Difficulty in accessing data
    - Need to write a new program to carry out each new task, e.g., search people by zip code or last name; update telephone number
  - Integrity problems
    - Integrity constraints (e.g., num_residence = 1) become part of program code -- hard to add new constraints or change existing ones
- Atomicity of updates
  - Failures may leave database in an inconsistent state with partial updates carried out, e.g., John and Mary get married, add new residence, update John's entry, and database crashes while Mary's entry is being updated…

Storing Data: Database vs File System (cont.)

- Concurrent access by multiple users
  - Needed for performance: can you imagine if only 1 person at a time could buy a ticket from Delta?
  - Uncontrolled concurrent access can lead to inconsistencies, e.g., the same seat could be sold multiple times…
    - There are 3 seats left; I buy 2 seats; John buys 3 seats at the same time
    - If I hit enter 1st there will be 1 seat left; if John is faster there will be 0; but 5 seats have been sold and we will fight at the airport!

Database systems offer solutions to all the above problems

Why use Database Systems?

- Data independence and efficient access
  - Easy + efficient access through declarative query languages and optimization
- Data integrity and security
  - Safeguarding data from failures and malicious access
- Concurrent access
- Reduced application development time
- Uniform data administration
- *When should you not use a database?*

What's in a DBMS?

	Data model	*Query language*	*Transactions and crash recovery*
"above the water"	Logical DB design Relational Model XML data model	SQL, QBE, views XPath, XQuery	Transactions
"below the water"	Map data to files Clustering Indexes	Query optimization Query evaluation	Locking Concurrency control Recovery Logs

Designing a database: The Conceptual Model

- What are the *entities* and *relationships* among these entities in the application?
- What information about these entities and relationships should we store in the database?
- What are the *integrity constraints* or *business rules* that hold?
- Different applications have different needs, and different perspectives – even to model the *same* object

billing department: patient(id, name, insurance, address)
visit(patientId, procedure, date, charge)
inpatient: patient(id,name,age,address)
alergies(id,alergies)
prescription(patientId,date,medicine)

Designing a database: The Conceptual Design

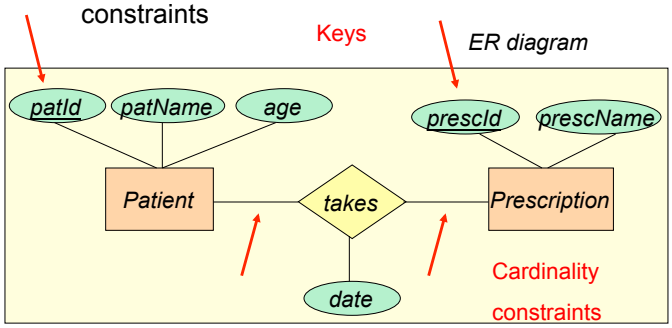
- What are the *entities* and *relationships* among these entities in the enterprise?
- What information about these entities and relationships should we store in the database?
- What are the *integrity constraints* or *business rules* that hol
- Dif ifferent

Requires a good understanding of the *semantics* of the application

billing department: patient(id, name, insurance, address)
visit(patientId, procedure, date, charge)
inpatient: patient(id,name,age,address)
alergies(id,alergies)
prescription(patientId,date,medicine)

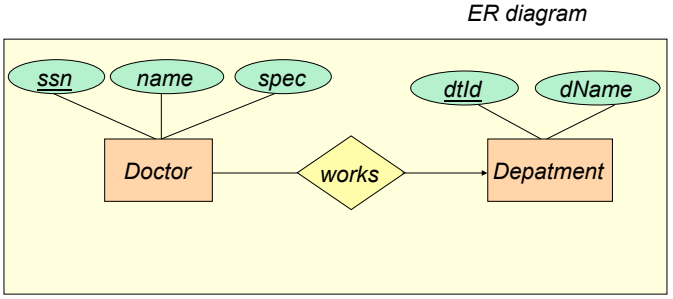
The Entity Relationship (ER) Data Model

- A *data model* is a collection of concepts for describing data, relationships, semantics and constraints



ER: Another Example

- A department has many doctors, but a doctor can only work in one department

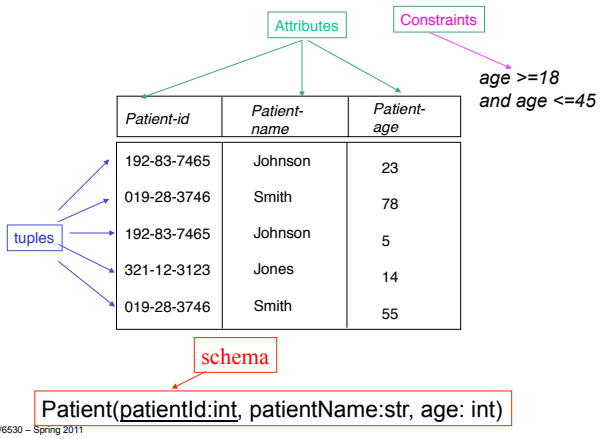


Relational Data Model

- ER used for conceptual design is then mapped into the relational 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 columns, or fields
- A *schema* is a description of a particular collection of data, using a given data model

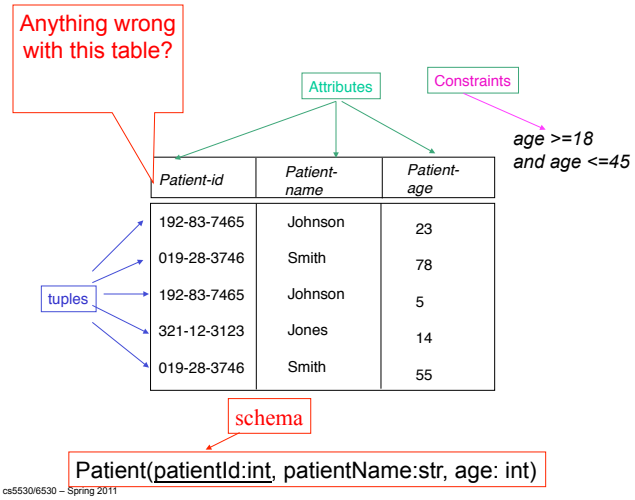
Patient(patientId:int, patientName:str, age: int)
Takes(patientId:int,prescId,prescDate:date)
Prescription(prescId:int, presName:str)

Relational Model: Terminology



Pitfalls in Relational Database Design

- Find a "good" collection of relation schemas
- Bad design may lead to
  - Repetition of information → inconsistencies!
    - E.g., keeping people and addresses in a single file
  - Inability to represent certain information
    - E.g., a doctor that is both a cardiologist and a pediatrician
- Design Goals:
  - Avoid redundant data
  - Ensure that relationships among attributes are represented
  - Ensure constraints are properly modeled: updates check for violation of database integrity constraints



Query Languages

- *Query languages:* Allow *manipulation* and *retrieval* of data from a database
- Queries are posed wrt data model
  - Operations over objects defined in data model
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic
  - Allows for optimization
- Query Languages != programming languages
  - QLs support easy, efficient access to large data sets
  - QLs not expected to be "Turing complete"
  - QLs not intended to be used for complex calculations

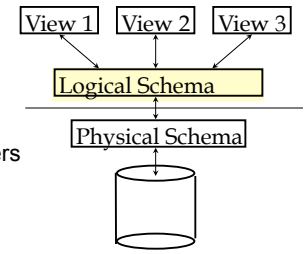
Query Languages

- *Query languages:* Allow *manipulation* and *retrieval* of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on l
  - Allows for much optimization.
- Query Languages != programming
  - QLs not expected to be "Turing complete".
  - QLs support easy, efficient access to large data sets.
  - QLs not intended to be used for complex calculations.

a language that can compute anything that can be computed

Levels of Abstraction

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



Key to good performance

Levels of Abstraction

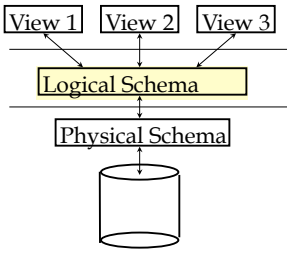
Programming Languages

Databases

Methods

Record/type definitions

Block of consecutive bytes



Data Independence

- Applications insulated from how data is structured and stored
- *Logical data independence*: Protection from changes in *logical* structure of data
  - Changes in the logical schema do not affect users as long as their views are still available
- *Physical data independence:* Protection from changes in *physical* structure of data
  - Changes in the physical layout of the data or in the indexes used do not affect the logical relations

One of the most important benefits of using a DBMS!

Example: University Database

- Logical schema:
  - *Students(sid: string, name: string, login: string, age: integer, gpa:real)*
  - *Courses(cid: string, cname:string, credits:integer)*
  - *Enrolled(sid:string, cid:string, grade:string)*
- Physical schema:
  - Students stored in id order
  - Index on last name
- External Schema (View):
  - *Course_info(cid:string,enrollment:integer)*

"above the water"

"below the water"

Data model	*Query language*	*Transactions and crash recovery*
Logical DB design Relational Model XML data model	SQL, QBE, views XPath, XQuery	Transactions
Map data to files Clustering Indexes	Query optimization Query evaluation	Locking Concurrency control Recovery Logs

Let's dive now…

Storage and Indexing

- The *DB administrator* designs the physical structures
- Nowadays, database systems can do (some of) this automatically: autoadmin, index advisors
- File structures: sequential, hashing, clustering, single or multiple disks, etc.
- Indexes:
  - Select attributes to index
  - Select the type of index
- Storage manager is a module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system:
  - interaction with the file manager
  - efficient storing, retrieving and updating of data

Query Optimization and Evaluation

- DBMS must provide efficient access to data
  - In an emergency, can't wait 10 minutes to find patient allergies
- Declarative queries are translated into imperative query plans
  - Declarative queries → logical data model
  - Imperative plans → physical structure
- Relational optimizers aim to find the best imperative plans (i.e., shortest execution time)
  - In practice they avoid the worst plans…

Transaction: An Execution of a DB Program

- Key concept is *transaction*, which is an *atomic* sequence of database actions (reads/writes)
- Each transaction, executed completely, must leave the DB in a *consistent state* if DB is consistent when the transaction begins
  - Ensuring that a transaction (run alone) preserves consistency is ultimately the programmer's responsibility!
- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures
  - DBMS ensures *atomicity* (all-or-nothing property)

Concurrency Control

- Concurrent execution of user programs 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., nurse and doctor can simultaneously edit a patient record
- DBMS ensures such problems don't arise: users can pretend they are using a single-user system

Ensuring Atomicity

- DBMS ensures *atomicity* (all-or-nothing property) even if system crashes in the middle of a transaction
  - If there is power outage, will the patient database become inconsistent?
- Idea: Keep a *log* (history) of all actions carried out by the DBMS while executing a set of transactions
  - Before a change is made to the database, the corresponding log entry is forced to a safe location.
  - After a crash, the effects of partially executed transactions are *undone* using the log; and if log entry wasn't saved before the crash, corresponding change was not applied to database!

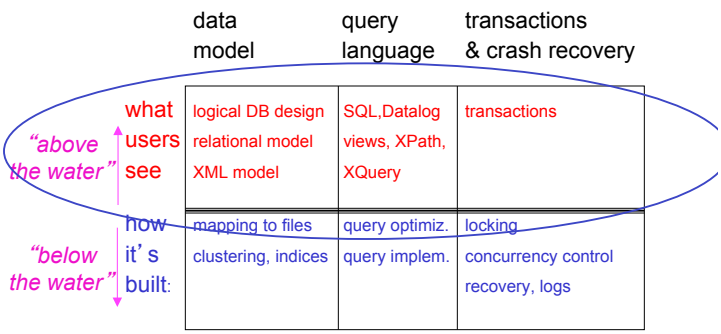
Databases make these folks happy ...

- End users
- DBMS vendors: $20B industry
- DB application programmers
  - E.g., smart webmasters
- Database administrator (DBA)
  - Designs logical /physical schemas
  - Handles security and authorization
  - Data availability, crash recovery
  - Database tuning as needs evolve

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
- *DBMS R&D is one of the broadest, most exciting areas in CS!*

What's in this course?



Overview of the Syllabus

- Introduction to databases
- Modeling data – capture the semantics of applications
  - entity relationship, relational model
- Querying data -- relational query languages
  - SQL, Datalog, Embedded SQL
- Constraints – maintaining data integrity and security
- Relational database design
- XML databases
  - XML data model, storing and publishing XML, XML query languages
- Advanced Topics:
  - Integrating data from multiple data sources
  - Information retrieval on the Web: How do search engines work?
  - Data Mining and OLAP
  - Scientific Data Management: Workflows and Provenance

"The real value of Computer Science education is less in acquiring a skill with technology than in teaching students to manage complexity; to navigate and assess information; to master modeling and abstraction; and to think analytically in terms of algorithms, or step-by-step procedures."

Ed Lazowska