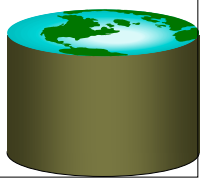
CS186: Introduction to Database Systems

Joe Hellerstein
and Christopher Olston

Fall 2005



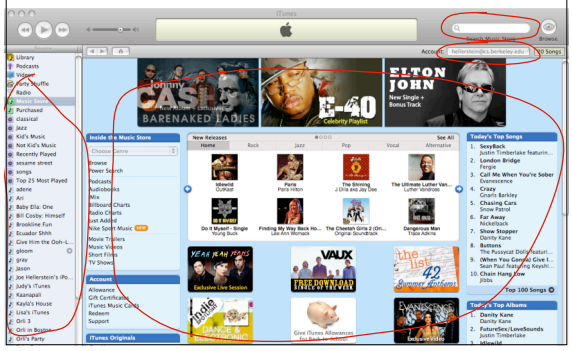
Queries for Today

- What?
- Why?
- Who?
- How?
- For instance?

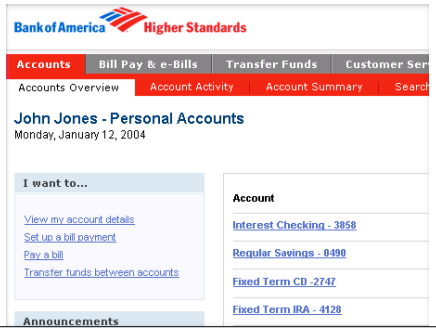
What: Database Systems Then

What: Database Systems Today

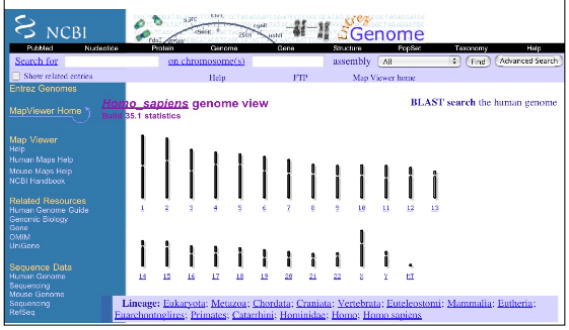
What: Database Systems Today



What: Database Systems Today



What: Database Systems Today



So… What is a Database?

- We will be broad in our interpretation
- A Database:
  - A very large, integrated collection of data.
- Typically models a real-world "enterprise"
  - Entities (e.g., teams, games)
  - Relationships (e.g. The A's are playing in the World Series)
- Might surprise you how flexible this is
  - Web search:
    - Entities: words, documents
    - Relationships: word in document, document links to document.
  - P2P filesharing:
    - Entities: words, filenames, hosts
    - Relationships: word in filename, file available at host

What is a Database Management System?

- A Database Management System (DBMS) is:
  - A software system designed to store, manage, and facilitate access to databases.
- Typically this term used narrowly
  - Relational databases with transactions
    - E.g. Oracle, DB2, SQL Server
  - Mostly because they predate other large repositories
    - Also because of technical richness
  - When we say DBMS in this class we will usually follow this convention
    - But keep an open mind about applying the ideas!

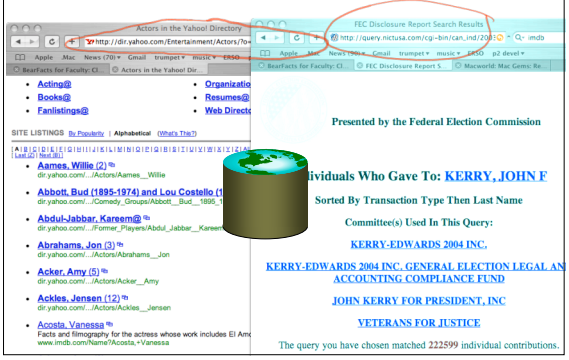
What: Is the WWW a DBMS?

- Fairly sophisticated search available
  - Crawler indexes pages on the web
  - Keyword-based search for pages
- But, currently
  - data is mostly unstructured and untyped
  - search only:
    - can't modify the data
    - can't get summaries, complex combinations of data
  - few guarantees provided for freshness of data, consistency across data items, fault tolerance, …
  - Web sites typically have a (relational) DBMS in the background to provide these functions.
- The picture is changing quickly
  - Information Extraction to get structure from unstructured
  - New standards e.g., XML, Semantic Web can help data modeling

What: "Search" vs. Query

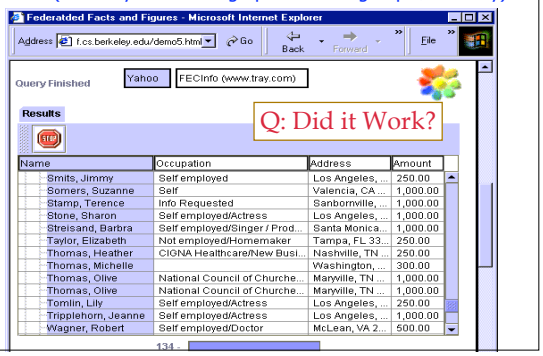
- What if you wanted to find out which actors donated to John Kerry's presidential campaign?
- Try "actors donated to john kerry" in your favorite search engine.
- If it isn't "published", it can't be searched!

What: A "Database Query" Approach



"Yahoo Actors" JOIN "FECInfo"

(Courtesy of the Telegraph research group @Berkeley)



What: Is a File System a DBMS?

- Thought Experiment 1:
  - You and your project partner are editing the same file.
  - You both save it at the same time.
  - Whose changes survive?

A) Yours B) Partner's C) Both D) Neither E) ???

- Thought Experiment 2:
  - –You're updating a file.
  - –The power goes out.
  - –Which changes survive?

A) All B) None C) All Since Last Save D) ???

What: Is a File System a DBMS?

- Thought Experiment 1:

Q: How do you write programs over a subsystem when it promises you only "???" ?

A: Very, very carefully!!

A) Y E) ???

–Which changes survive?

A) All B) None C) All Since Last Save D) ???

OS Support for Data Management

- Data can be stored in RAM
  - this is what every programming language offers!
  - RAM is fast, and random access
  - Isn't this heaven?
- Every OS includes a File System
  - manages *files* on a magnetic disk
  - allows *open, read, seek, close* on a file
  - allows protections to be set on a file
  - drawbacks relative to RAM?

Database Management Systems

- What more could we want than a file system?
  - Simple, efficient *ad hoc*[1] queries
  - concurrency control
  - recovery
  - benefits of good data modeling
- S.M.O.P.[2]? Not really…
  - as we'll see this semester
  - in fact, the OS often gets in the way!

[1] ad hoc: formed or used for specific or immediate problems or needs
[2] SMOP: Small Matter Of Programming

Current Commercial Outlook

- A major part of the software industry:
  - Oracle, IBM, Microsoft
  - also Sybase, Informix (now IBM), Teradata
  - smaller players: java-based dbms, devices, OO, …
- Well-known benchmarks (esp. TPC)
- Lots of related industries
  - data warehouse, document management, storage, backup, reporting, business intelligence, ERP, CRM, app integration
- Traditional Relational DBMS products dominant and evolving
  - adapted for extensibility (user-defined types), native XML support.
  - Microsoft merger of file system/DB…?
- Open Source coming on strong
  - MySQL, PostgreSQL, Apache Derby, BerkeleyDB, Ingres, EigenBase
- And of course, the other "database" technologies
  - Search engines, P2P, etc.

What database systems will we cover?

- We will be try to be broad and touch upon
  - Relational DBMS (e.g. Oracle, SQL Server, DB2, Postgres)
  - Document search engines (e.g. Google, Yahoo! Search, Verity, Spotlight)
  - "Semi-structured" DB systems (e.g. XML repositories like Xindice)
- Starting point
  - We assume you have used web search engines
  - We assume you don't know relational databases
    - Yet they pioneered many of the key ideas
  - So focus will be on relational DBMSs
    - With frequent side-notes on search engines, XML issues

Why take this class?

A. Database systems are at the core of CS
B. They are incredibly important to society
C. The topic is intellectually rich
D. A capstone course for undergrad
~~E. It isn't that much work~~
F. Looks good on your resume

Let's spend a little time on each of these

Why take this class?

A. Database systems are the core of CS

- Shift from computation to information
  - True in corporate computing for years
  - Web, p2p made this clear for personal computing
  - Increasingly true of scientific computing
- Need for DB technology has exploded in the last years
  - Corporate: retail swipe/clickstreams, "customer relationship mgmt", "supply chain mgmt", "data warehouses", etc.
  - Web:not just "documents". Search engines, e-commerce, blogs, wikis, other "web services".
  - Scientific: digital libraries, genomics, satellite imagery, physical sensors, simulation data
  - Personal: Music, photo, & video libraries. Email archives. File contents ("desktop search").

Why take this class?

B. DBs are incredibly important to society

- "Knowledge is power." -- Sir Francis Bacon
- "With great power comes great responsibility." -- SpiderMan's Uncle Ben

Policy-makers should understand technological possibilities.
Informed Technologists needed in public discourse on usage.

Why take this class?

C. The topic is intellectually rich.

- representing information
  - data modeling
- languages and systems for querying data
  - complex queries & query semantics*
  - over massive data sets
- concurrency control for data manipulation
  - controlling concurrent access
  - ensuring transactional semantics
- reliable data storage
  - maintain data semantics even if you pull the plug

* semantics: the meaning or relationship of meanings of a sign or set of signs

Why take this class?

D. The course is a capstone.

- We will see
  - Algorithms and cost analyses
  - System architecture and implementation
  - Resource management and scheduling
  - Computer language design, semantics and optimization
  - Applications of AI topics including logic and planning
  - Statistical modeling of data

Why take this class?

~~E. It isn't that much work.~~

- Bad news: It is a lot of work.
- Good news: the course is front loaded
  - Most of the hard work is in the first half of the semester
  - Load balanced with most other classes

Why take this class?

F. Looks good on my resume.

- Yes, but why? This is not a course for:
  - Oracle administrators
  - IBM DB2 engine developers
    - Though it's useful for both!
- It is a course for well-educated computer scientists
  - Database system concepts and techniques increasingly used "outside the box"
    - Ask your friends at Microsoft, Yahoo!, Google, Apple, etc.
    - Actually, they may or may not realize it!
  - A rich understanding of these issues is a basic and (un?)fortunately unusual skill.

Who?

- Instructors
  - Prof. Joe Hellerstein, UC Berkeley
  - Dr. Christopher Olston, Yahoo! Research
  - cs186profs@db.cs.berkeley.edu
- TAs
  - John Lo
  - Nathan Burkhart
  - Alex Rasmussen

How? Workload

- Projects with a "real world" focus:
  - Modify the internals of a "real" open-source database system: PostgreSQL
    - Serious C system hacking
    - Measure the benefits of our changes
  - Build a web-based application w/PostgreSQL, Apache & PHP): SQL + PHP
- Other homework assignments and/or quizzes
- Exams – 1 Midterm & 1 Final
- Projects to be done in groups of 2
  - Pick your partner ASAP
- The course is "front-loaded"
  - most of the hard work is in the first half

How? Administrivia

- http://inst.eecs.berkeley.edu/~cs186
- Prof. Office Hours:
  - Hellerstein: 685 Soda Hall, TBA (check web page)
  - Olston: 687 Soda Hall, Thursday 2PM
- TAs
  - Office Hours: TBA (check web page)
- Discussion Sections WILL NOT meet this week

How? Administrivia, cont.

- Textbook
  - Ramakrishnan and Gehrke, 3rd Edition
- Grading, hand-in policies, etc. will be on Web Page
- Cheating policy: zero tolerance
  - We have the technology…
- Team Projects
  - Teams of 2
  - Peer evaluations.
    - Be honest! Feedback is important. Trend is more important than individual project.
- Class bulletin board - ucb.class.cs186
  - read it regularly and post questions/comments.
  - mail broadcast to all TAs will not be answered
  - mail to the cs186 course account will not be answered
- Class Blog for announcements

Agenda for the rest of today

- A "free tasting" of central concepts in DB field:
  - queries (vs. search)
  - data independence
  - transactions
- Next Time
  - the Relational data model
- Today's lecture is from Chapter 1 in R&G
- Read Chapter 2 for next class.

Describing Data: Data Models

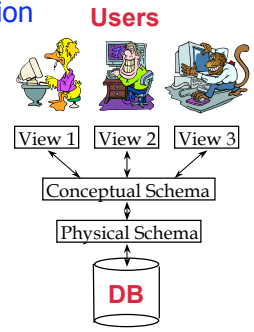
- A *data model* is a collection of concepts for describing data.
- A *schema* is a description of a particular collection of data, using a given data model.
- The *relational model of data* is the most widely used model today.
  - Main concept: *relation*, basically a table with rows and columns.
  - Every relation has a *schema*, which describes the columns, or fields.

Example: University Database

- 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)*

Levels of Abstraction

- Views describe how users see the data.
- Conceptual schema defines logical structure
- Physical schema describes the files and indexes used.



Example: University Database

- Conceptual 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:
  - 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 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.
- Q: Why is this particularly important for DBMS?

Because databases and their associated applications persist.

Agenda …

- A "free tasting" of central concepts in DB field:
  - queries (vs. search)
  - data independence
  - transactions

Concurrent execution of user programs

- Why?
  - Utilize CPU while waiting for disk I/O
    - (database programs make heavy use of disk)
  - Avoid short programs waiting behind long ones
    - e.g. ATM withdrawal while bank manager sums balance across all accounts

Concurrent execution

- Interleaving actions of different programs: trouble!

  Example:
  - Bill transfers $100 from savings to checking
    Savings –= 100; Checking += 100
  - Meanwhile, Bill's wife requests account info.

  Bad interleaving:
  - Savings –= 100
  - Print balances
  - Checking += 100

  – Printout is missing $100 !

Concurrency Control

- DBMS ensures such problems don't arise
- Users can pretend they are using a single-user system. (called "Isolation")
  – Thank goodness!

Key concept: **Transaction**

- an atomic sequence of database actions (reads/writes)
- takes DB from one consistent state to another

Example

- Here, *consistency* is based on our knowledge of banking "semantics"
- In general, up to writer of transaction to ensure transaction preserves consistency
- DBMS provides (limited) automatic enforcement, via integrity constraints
  – e.g., balances must be >= 0

Concurrent transactions

- Goal: **execute xacts {T1, T2, … Tn}, and ensure a consistent outcome**
- *One option:* **"serial" schedule (one after another)**
- *Better:* **allow interleaving of xact actions, as long as outcome is equivalent to some serial schedule**

Possible Enforcement Methods

- Optimistic: **permit arbitrary interleaving, then check equivalence to serial sched.**
- Pessimistic: **xacts set *locks* on data objects, such that illegal interleaving is impossible**

Locking example

- T1 (Bill): *Savings –= 100; Checking += 100*
- T2 (Bill's wife): *Print(Checking); Print(Savings)*

– T1 and T2 both lock Savings and Checking objects
– If T1 locks Savings & Checking first, T2 must wait

A wrinkle …

- T1 (Bill): *Savings –= 100; Checking += 100*
- T2 (Bill's wife): *Print(Checking); Print(Savings)*

Suppose:
1. T1 locks Savings
2. T2 locks Checking
- Now neither transaction can proceed!

– called "deadlock"
– DBMS will abort and restart one of T1 and T2
– Need "undo" mechanism that preserves consistency

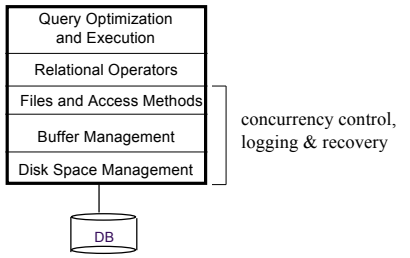
– Undo mechanism also necessary if system crashes between "Savings –= 100" and "Checking += 100" …

Ensuring Transaction Properties

- DBMS ensures:
  - ***atomicity*** **even if xact aborted (due to deadlock, system crash, …)**
  - ***durability*** **of committed xacts, even if system crashes.**
- Idea: Keep a *log* of all actions carried out by the DBMS:
  - Record all DB modifications in log, *before* they are executed
  - To abort a xact, undo logged actions in reverse order
  - If system crashes, must:
    1) **undo** partially executed xacts (ensures **atomicity**)
    2) **redo** committed xacts (ensures **durability**)
  - *trickier than it sounds!*

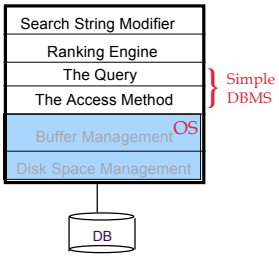
Architecture of a DBMS …

Typical DBMS architecture



FYI: A text search engine

- Less "system" than DBMS
  - Uses OS files for storage
  - Just one access method
  - One hardwired query
    - regardless of search string
- Typically no concurrency or recovery management
  - Read-mostly
  - Batch-loaded, periodically
  - No updates to recover
  - OS a reasonable choice
- Smarts: text tricks
  - Search string modifier (e.g. "stemming" and synonyms)
  - Ranking Engine (sorting the output, e.g. by word or document popularity)
  - no semantics: WYGIWIGY



Advantages of a DBMS

- Data independence
- Efficient data access
- Data integrity & security
- Data administration
- Concurrent access, crash recovery
- Reduced application development time
- So why not use them always?
  - Expensive/complicated to set up & maintain
  - This cost & complexity must be offset by need
  - General-purpose, not suited for special-purpose tasks (e.g. text search!)

Databases make these folks happy ...

- DBMS vendors, programmers
  - Oracle, IBM, MS …
- End users in *many* fields
  - Business, education, science, …
- DB application programmers
  - Build data entry & analysis tools on top of DBMSs
  - Build web services that run off DBMSs
- Database administrators (DBAs)
  - Design logical/physical schemas
  - Handle security and authorization
  - Data availability, crash recovery
  - Database tuning as needs evolve

…must understand how a DBMS works

Summary

- DBMS used to maintain, query large datasets.
  - can manipulate data and exploit *semantics*
- Other benefits include:
  - recovery from system crashes,
  - concurrent access,
  - quick application development,
  - data integrity and security.
- Levels of abstraction provide data independence.
- In this course we will explore:
  1) How to be a sophisticated user of DBMS technology
  2) What goes on inside the DBMS