Introduction

Introduction to Databases
CompSci 316 Fall 2014

About us

• Instructor: Jun Yang
  • Been doing (and enjoying) research in databases ever since grad school (1995)
    • Didn't take any database as an undergrad
    • Now working on data-intensive systems and computational journalism
• TA: Brett Walenz
  • PhD student in Computer Science
  • Working on computational journalism

What comes to your mind…

…when you think about “databases”?

http://www.quackit.com/pix/database/tutorial/dbms_sql_server.gif
But these use databases too...

Facebook uses MySQL to store posts, for example...

And this...

Trend: Moore’s Law reversed

- **Moore’s Law:**
  Processing power doubles every 18 months
- Amount of data doubles every 9 months
  - Disk sales (# of bits) doubles every 9 months
  - Parkinson’s Law:
    Data expands to fill the space available for storage
    - As of 2009, Facebook ingested 15 terabytes of data per day;
      as of 2010, Walmart handled 1 million transactions per hour;
      both maintained 2.5-petabyte databases
    - CERN’s Large Hadron Collider will produce 15 petabytes/year

Moore’s Law reversed:
Time to process all data doubles every 18 months!
Democratizing data (and analysis)

- And it’s not just about money and science
- Democratization of data: more data—relevant to you and the society—are becoming available
  - “Government in the sunshine”: spending reports, school performance, crime reports, corporate filings, campaign contributions, …
  - “Smart planet”: sensors for phones and cars, roads and bridges, buildings and forests, …
- But few people know how to analyze them
- You will learn how to help bridge this divide

Misc. course info

- Website: http://sites.duke.edu/compsci316_01_f2014/
  - Course info; tentative schedule and reference sections in the book; lecture slides, assignments, help docs, …
- Programming: VM required; $100 worth of credits for VMs in the cloud, courtesy of Amazon
- Q&A on Piazza; grades, sample solutions on Sakai
- Watch your email for announcements
- Office hours to be posted

Grading

- [90%, 100%] A- / A / A+
- [80%, 90%) B- / B / B+
- [70%, 80%) C- / C / C+
- [60%, 70%) D
- [0%, 60%) F

- No “curves”
- Scale may be adjusted downwards (i.e., grades upwards) if, for example, an exam is too difficult
- Scale will not go upwards—mistake would be mine alone if I made an exam too easy
Duke Community Standard

• See course website for link
• Group discussion for assignments is okay (and encouraged), but
  • Acknowledge any help you receive from others
  • Make sure you "own" your solution
• All suspected cases of violation will be aggressively pursued

Course load

• Four homework assignments (35%)
  • Gradiance: immediately and automatically graded
  • Plus written and programming problems
• Course project (25%)
  • Details to be given in the third week of class
• Midterm and final (20% each)
  • Open book, open notes
  • Final is comprehensive, but emphasizes the second half of the course

Projects from previous years

• Expose.js: natural language querying
  • E.g.: “find beers served by bar with name Satisfaction”
  • Ben Schwab, James Hong, Jesse Hu, 2013
• Zoom: object-relational mapping lib in GO for Redis
  • Work with objects in GO automatically persisted to Redis
  • Alex Browne, 2013
• Pickup Coordinator: an iPhone app that lets you coordinate carpool/pickups with others
  • Adam Cue, Kevin Esoda, Kate Yang, 2012
• Mobile Pay: quick way to make a transaction between two people on their phones
  • Michael Deng, Kevin Gao, Derek Zhou, 2012
More past examples

• Chumchi: a social website with relevant feeds
  • Kirill Kliment, 2011
• FriendsTracker app: where are my friends?
  • Anthony Lin, Jimmy Mu, Austin Benesh, Nic Dinkins, 2011
• ePrint iPhone app
  • Ben Getson and Lucas Best, 2009
• Making iTunes social
  • Nick Patrick, 2006; Peter Williams and Nikhil Arun, 2009
• Duke Schedulator: ditch ACES—plan schedules visually!
  • Alex Beutel, 2008
• SensorDB: manage/clean/visualize sensor data from Duke Forest
  • Ashley DeMass, Jonathan Jou, Jonathan Odom, 2007
• Facebook
  • Tyler Brock and Beth Trushkowsky, 2005
• Web-based K-villetenting management
  • Zach Marshall, 2005

So, what is a database system?

From Oxford Dictionary:
• Database: an organized body of related information
• Database system, DataBase Management System (DBMS): a software system that facilitates the creation and maintenance and use of an electronic database
What do you want from a DBMS?

- Keep data around (persistent)
- Answer questions (queries) about data
- Update data

- Example: a traditional banking application
  - Data: Each account belongs to a branch, has a number, an owner, a balance, ...; each branch has a location, a manager, ...
  - Persistency: Balance can’t disappear after a power outage
  - Query: What’s the balance in Homer Simpson’s account?
    - What’s the difference in average balance between Springfield and Capitol City accounts?
  - Modification: Homer withdraws $100; charge accounts with lower than $500 balance a $5 fee

Sounds simple!

- Text files
  - Accounts/branches separated by newlines
  - Fields separated by #’s

Query by programming

- What’s the balance in Homer Simpson’s account?
  - A simple script
    - Scan through the accounts file
    - Look for the line containing “Homer Simpson”
    - Print out the balance
Query processing tricks

- Tens of thousands of accounts are not Homer’s

What happens when the query changes to: What’s the balance in account 00142-00857?

Observations

- There are many techniques—not only in storage and query processing, but also in concurrency control, recovery, etc.
- Different techniques may work better in different usage scenarios
- Same techniques get used over and over again in different applications

The birth of DBMS – 1

From Hans-J. Schek’s VLDB 2000 slides
Early efforts

• “Factoring out” data management functionalities from applications and standardizing these functionalities is an important first step
  - CODASYL standard (circa 1960’s)
    - Bachman got a Turing award for this in 1973

• But getting the abstraction right (the API between applications and the DBMS) is still tricky
CODASYL

• Query: Who have accounts with 0 balance managed by a branch in Springfield?
• Pseudo-code of a CODASYL application:
  Use index on account(balance) to get accounts with 0 balance;
  For each account record:
    Get the branch id of this account;
    Use index on branch(id) to get the branch record;
    If the branch record’s location field reads “Springfield”:
    Output the owner field of the account record.
• Programmer controls “navigation”: accounts → branches
  • How about branches → accounts?

What’s wrong?

• The best navigation strategy & the best way of organizing the data depend on data/workload characteristics
With the CODASYL approach
• To write correct code, programmers need to know how data is organized physically (e.g., which indexes exist)
• To write efficient code, programmers also need to worry about data/workload characteristics
   Can’t cope with changes in data/workload characteristics

The relational revolution (1970’s)

• A simple model: data is stored in relations (tables)
• A declarative query language: SQL
  SELECT Account.owner
  FROM Account, Branch
  WHERE Account.balance = 0
  AND Branch.location = 'Springfield'
  AND Account.branch_id = Branch.branch_id;
• Programmer specifies what answers a query should return, but not how the query is executed
• DBMS picks the best execution strategy based on availability of indexes, data/workload characteristics, etc.
   Provides physical data independence
Physical data independence

- Applications should not need to worry about how data is physically structured and stored
- Applications should work with a logical data model and declarative query language
- Leave the implementation details and optimization to DBMS
- The single most important reason behind the success of DBMS today
  - And a Turing Award for E. F. Codd in 1981

Standard DBMS features

- Persistent storage of data
- Logical data model; declarative queries and updates → physical data independence
  - Relational model is the dominating technology today
  - What else?

DBMS is multi-user

- Example
  
  ```
  get account balance from database
  if balance > amount of withdrawal then
    balance = balance - amount of withdrawal;
    dispense cash;
    store new balance into database;
  Homer at ATM1 withdraws $100
  Marge at ATM2 withdraws $50
  Initial balance = $400, final balance = ?
  Should be $250 no matter who goes first
  ```

Final balance = $300

Homer withdraws $100:  Marge withdraws $50:
read balance; $400
if balance > amount then
   balance = balance - amount; $300
write balance; $300
read balance; $400
if balance > amount then
   balance = balance - amount; $350
write balance; $350

Final balance = $350

Homer withdraws $100:  Marge withdraws $50:
read balance; $400
if balance > amount then
   balance = balance - amount; $300
write balance; $300
read balance; $400
if balance > amount then
   balance = balance - amount; $350
write balance; $350

Concurrency control in DBMS

• Similar to concurrent programming problems?
  • But data not main-memory variables
• Similar to file system concurrent access?
  • Lock the whole table before access
    • Approach taken by MySQL in the old days
    • Still used by SQLite (as of Version 3)
  • But want to control at much finer granularity
    • Or else one withdrawal would lock up all accounts!
Recovery in DBMS

- Example: balance transfer
decrement the balance of account X by $100;
increment the balance of account Y by $100;
- Scenario 1: Power goes out after the first instruction
- Scenario 2: DBMS buffers and updates data in memory (for efficiency); before they are written back to disk, power goes out
- How can DBMS deal with these failures?

Standard DBMS features: summary

- Persistent storage of data
- Logical data model; declarative queries and updates → physical data independence
- Multi-user concurrent access
- Safety from system failures
- Performance, performance, performance
  - Massive amounts of data (terabytes–petabytes)
  - High throughput (thousands–millions transactions/hour)
  - High availability (≥ 99.999% uptime)

DBMS architecture today

- Much of the OS may be bypassed for performance and safety
- We will be filling in many details of the DBMS box throughout the semester
AYBABTU?

“Us” = relational databases
• Most data are not in them!
  • Personal data, web, scientific data, system data, ...
• Text and semi-structured data management
  • XML, JSON, ...
• “NoSQL” movement
  • MongoDB, Cassandra, BigTable, HBase, ...
• This course will look beyond relational databases

Use of AYBABTU inspired by Garcia-Molina
Image: http://upload.wikimedia.org/wikipedia/en/0/03/Aybabtu.png

Course components

• Relational databases
  • Relational algebra, database design, SQL, app programming
• XML
  • Data model and query languages, app programming, interplay between XML and relational databases
• Database internals
  • Storage, indexing, query processing and optimization, concurrency control and recovery
• Advanced topics (TBD)
  • Data warehousing and data mining, Web search and indexing, parallel data processing/MapReduce, etc.

Announcements (Tue. Aug. 25)

• Permission numbers will be emailed this Thursday evening based on the wait list
  • Contact me if you cannot get onto the wait list for some reason (e.g., prerequisites)
• Amazon AWS credit codes will be emailed based on the enrollment list by next Monday
• This Thursday: our first language of the semester—relational algebra!