Introduction

CPS 116
Introduction to Database Systems

A few words about myself (and databases)

- Have been doing (and enjoying) research in databases ever since grad school (1995)
- But didn’t take any database course as an undergrad
  - Just didn’t appreciate it
  - Now, why would you want to take 116?

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
Misc. course information

- Course website: http://www.cs.duke.edu/courses/fa11/08/cps116/
  - Course information; tentative syllabus and reference sections in the book; lecture slides, assignments, programming notes
- Gradiance ("Online Accelerated Learning"): see course website for purchase information
- Blackboard: for grades only
- Mailing list: cps116@cs.duke.edu
  - Messages of general interest only
- No "official" recitation sessions; help sessions for assignments, project, and exams to be scheduled

Grading

\[
\begin{array}{ccc}
[90\%, 100\%] & A- / A / A+ \\
[80\%, 90\%) & B- / B / B+ \\
[70\%, 80\%) & C- / C / C+ \\
[60\%, 70\%) & D \\
[0\%, 60\%) & F \\
\end{array}
\]

- 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

Course load

- Four homework assignments (35%)
  - Including Gradiance as well as additional 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
Example projects

- SensorDB: a system for managing, cleansing, and visualizing sensor data collected from the Duke Forest
- SuperDatabase: GUI for creating schema with rich datatypes, as well as editing and querying such data
  - Andy Ewing, MacRae Linton, Congyi Wu, and David Zhang, 2007
- yourTunes: social music networking
  - Nick Patrick, 2006
- Facebook
  - Tyler Brock and Beth Trushkowsky, 2005
- Web-based K-ville tenting 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 queries (questions) 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 account with lower than $500 balance with a $5 fee
Sounds simple!

ASCII file
Accounts/branches separated by newlines
Fields separated by #'s

Query

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
Observations

- Tons of tricks (not only in storage and query processing, but also in concurrency control, recovery, etc.)
- Different tricks may work better in different usage scenarios (example?)
- Same tricks get used over and over again in different applications

The birth of DBMS – 1

(Figure from Hans-J. Schek’s VLDB 2000 slides)

The birth of DBMS – 2

(Figure from Hans-J. Schek’s VLDB 2000 slides)
The birth of DBMS – 3

(Figure 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 \( \rightarrow \) branches
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, application programmers need to know how data is organized physically (e.g., which indexes exist)
  - To write efficient code, application 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 data model: data is stored in relations (tables)
- A declarative query language: SQL
  
  ```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
Modern DBMS features

- Persistent storage of data
- Logical data model; declarative queries and updates
  → physical data independence
  - Relational model is the dominating technology today
  - XML is a hot wanna-be
- 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
```
Final balance = $350

Homer withdraws $100:  \[ \text{read balance; } $400 \]
\[ \text{if balance > amount then} \]
\[ \text{balance = balance - amount; } $300 \]
\[ \text{write balance; } $300 \]

Marge withdraws $50:  \[ \text{read balance; } $400 \]
\[ \text{if balance > amount then} \]
\[ \text{balance = balance - amount; } $350 \]
\[ \text{write balance; } $350 \]

Concurrency control in DBMS

- Appears similar to concurrent programming problems?
  - But data not main-memory variables
- Appears similar to file system concurrent access?
  - Approach taken by MySQL in the old days
  - Still used by SQLite (as of Version 3)

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?
Summary of modern DBMS features

- 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 per minute)
  - High availability (≥ 99.999% uptime)

Major DBMS today

- Oracle
- IBM DB2 (from System R, System R*, Starburst)
- Microsoft SQL Server
- Teradata
- Sybase
- Informix (acquired by IBM)
- PostgreSQL (from UC Berkeley's Ingres, Postgres)
- Tandem NonStop (acquired by Compaq, now HP)
- MySQL (acquired by Sun)
  - Microsoft Access
  - SQLite
  - BerkeleyDB (acquired by Oracle)

Modern DBMS architecture

- OS layer is bypassed for performance and safety
- We will be filling in many details for the DBMS box
People working with databases

- End users: query/update databases through application user interfaces (e.g., Amazon.com, 1-800-DISCOVER, etc.)
- Database designers: design database “schema” to model aspects of the real world
- Database application developers: build applications that interface with databases
- Database administrators (a.k.a. DBA’s): load, back up, and restore data, fine-tune databases for performance
- DBMS implementors: develop the DBMS or specialized data management software, implement new techniques for query processing and optimization

AYBABTU? 

(“us” = relational databases)

- Most of the data is not in relational databases!
  - Personal data
  - Web
  - Scientific data
  - System data
- Data management is expanding to these areas
  - This course will look beyond relational databases too

Course roadmap

- 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
- Topics beyond databases
  - Data warehousing and data mining
  - Web and keyword searches