CMSC 724: Introduction

Amol Deshpande

University of Maryland, College Park

January 29, 2009
Today

Overview
- Grading etc. . .
- A crash course in why databases are needed
- What this class is about, What we will cover

Course Website

Course Forum
- “Register” → Go to “User CP” → join group “CMSC 724 Spring 2009”
- We will use this for news, discussions, and for posting paper critiques

Couple of things
- No laptops
- Typically won’t use detailed slides
We will cover:
- A blend of classic papers + ongoing research

Textbook:

Not CMSC624!!

Prerequisite: CMSC 424
- Class notes off of my webpage

Grading:
- A class project (35%)
- Two exams + two homeworks (50%)
  - One (covering background) very soon
- Paper critiques + class participation (15%)
  - Critiques mandatory before the class
What is a DBMS?

- Manage data
  - Store, update, answer queries over etc..

- What kind of data?
  - Everywhere you see…
  - Personal (emails, data on your computer)
  - Enterprise
  - Banks, supermarkets, universities, airlines etc etc
  - Scientific (biological, astronomical)
  - Etc…
Simple Banking Application
Need to store information about:
  - Accounts
  - Customers
Need to support:
  - ATM transactions
  - Queries over the data
Instructive to see how a naive solution will work
Data stored in files in ASCII format

- #-separated files in /usr/db directory
  - /usr/db/accounts
    - AccountNumber # Balance
      - 101 # 900
      - 102 # 700
      - ...
  - /usr/db/customers
    - CustomerName # CustomerAddress # AccountNumber
      - Johnson # 101 University Blvd # 101
      - Smith # 1300 K St # 102
      - Johnson # 101 University Blvd # 103
      - ...

A file system-based solution
A file system-based solution

- Write application programs to support the operations
  - In your favorite programming language
  - To support withdrawals by a customer for amount $X from account Y
    - Scan /usr/db/accounts, and look for Y in the 1st field
    - Subtract $X from the 2nd field, and rewrite the file
  - To support finding names of all customers on street Z
    - Scan /usr/db/customers, and look for (partial) matches for Z in the address field

  . . .
1. Data redundancy and inconsistency
   - No control of redundancy
     - CustomerName # CustomerAddress # AccountNumber
     - Johnson # 101 University Blvd # 101
     - Smith # 1300 K St # 102
     - Johnson # 101 University Blvd # 103
   - Inconsistencies
     - Data in different files may not agree
     - Very critical issue
   - Especially true when programs/data organization evolve over time
2. Evolution of the database is hard
   - Delete an account
     - Will have to rewrite the entire file
   - Add a new field to the accounts file, or split the customers file in two parts:
     - Rewriting the entire file least of the worries
     - Will probably have to rewrite all the application programs
What’s wrong with this solution?

3. Difficulties in Data Retrieval
   - No sophisticated tools for selective data access
     - Access only the data for customer X
     - Inefficient to scan the entire file
   - Limited reuse
     - Find customers who live in area code 301
     - Unfortunately, no application program already written
     - Write a new program every time?
4. Semantic constraints

- Semantic integrity constraints become part of program code
  - *Balance should not fall below 0*
  - Every program that modifies the balance will have to enforce this constraint
- Hard to add new constraints or change existing ones
  - *Balance should not fall below 0 unless overdraft-protection enabled*
- Now what?
- Rewrite every program that modifies the balance?
What’s wrong with this solution?

5. Atomicity problems because of failures
   - Query: Jim transfers $100 from Acct #55 to Acct #376

   Program:
   1. Get balance for acct #55
   2. If balance55 > $100 then
   3. balance55 := balance55 - 100
   4. update balance55 on disk
   5. get balance from database for acct #376
   6. balance376 := balance376 + 100
   7. update balance376 on disk

   Must be **atomic**
   - Do all the operations or none of the operations
What’s wrong with this solution?

6. Durability problems because of failures
   Query: Jim transfers $100 from Acct #55 to Acct #376

   Program:
   1. Get balance for acct #55
   2. If balance55 > $100 then
   3. balance55 := balance55 - 100
   4. update balance55 on disk
   5. get balance from database for acct #376
   6. balance376 := balance376 + 100
   7. update balance376 on disk
   8. print receipt

   After reporting success to the user, the changes better be there when he checks tomorrow
What’s wrong with this solution?

7. Concurrent access anomalies

Joe@ATM1: Withdraws $100 from Acct #55
1. Get balance for acct #55
2. If balance55 > $100 then
   a. balance55 := balance55 - 100
   b. dispense cash

Jane@ATM1: Withdraws $100 from Acct #55
1. Get balance for acct #55
2. If balance55 > $100 then
   a. balance55 := balance55 - 100
   b. dispense cash
   c. update balance55

Balance would only reflect one of the two operations
   • Bank loses money
8. Security Issues

Need fine grained control on who sees what

- Only the manager should have access to accounts with balance more than $100,000
- How to enforce that if there is only one accounts file?

- Database management systems provide an end-to-end solution to all of these problems
The key insight is what's called *data abstraction*. Probably *the* most important purpose of a DBMS is hiding *low-level details* from the users of the system. The goal is achieved through the use of logical abstractions.
Data Abstraction

What data users and application programmers see?

What data is stored?
describe data properties, data semantics, data relationships etc..

How data is actually stored?
e.g. are we using disks? which file system?
Logical level:
- Provide an abstraction of tables
- Two tables can be accessed:
  - *accounts*: columns – account number, balance
  - *customers*: columns – name, address, account number

View level:
- A teller (non-manager) can only see a part of the accounts table
  - Not containing high balance accounts
Physical Level:
- Each table is stored in a separate ASCII file
- # separated fields

Identical to what we had before?
- BUT the users are not aware of this
  - They only see the tables
  - The application programs are written over the tables abstraction

- Can change the physical level without affecting users
- In fact, can even change the logical level without affecting the teller
DBMS at a glance

- **Data Models**
  - Conceptual representation of the data

- **Data Retrieval**
  - How to ask questions of the database
  - How to answer those questions

- **Data Storage**
  - How/where to store data, how to access it

- **Data Integrity**
  - Manage crashes, concurrency
  - Manage semantic inconsistencies
DBMS Solutions?

- Data redundancy and inconsistency
  - Normal Forms
- Evolution of the database is hard
  - Data abstraction, declarative interfaces
- Difficulties in data retrieval
  - Declarative query languages
  - Indexes, query optimizer, buffer managers etc.
- Semantic Constraints
  - Normal forms, declarative integrity constraints
- Atomicity, Durability, Concurrency (ACID)
  - Locking/logging, concurrency control, recovery
- Security Issues
  - Views, authorization mechanisms (GRANT, REVOKE)
What’s left??

- Enterprise data
  - Wal-mart: 583 terabytes of sales and inventory data
    - Adds a billion rows every day
  - Neilsen Media Research: 20GB a day; total 80-100TB
  - Real-time data processing. Data mining.

- Web
  - Data integration. Querying distributed sources

- Scientific Databases (biological, astronomical)
  - Imagine real-time genome sequencing!
  - Except for the metadata (who, where etc), no idea how to deal with this data
  - Even metadata management is problematic – errors, inconsistencies
New applications

- Digital libraries
- Increasing amounts of multi-media data
  - Camera, audio sensors etc.
  - Memex !!
    - Record everything you see/hear (the MyLifeBits project)
- Semi-structured and unstructured data
  - XML, Text
  - Information retrieval, extraction (Avatar@IBM)
- “Data streams”
  - Continuous high-rate data (e.g. stock data, network monitoring, sensors)
  - Much recent work, but still fluid (e.g. no language)
New applications

- The world-wide “sensor web” (SensorMap@MS)
  - Wireless sensor networks are becoming ubiquitous.
  - RFID: Possible to track every single piece of product throughout its life
    - “Britain to log vehicle movements through cameras. 35 million reads per day”
  - Bio-sensors to monitor patients round the clock.
  - Camera/audio sensor networks (e.g. traffic cameras)
  - “Anthrax” sensors

- Many challenges
  - Data interoperability, dealing with errors/uncertainty in the data, distributed processing, need for statistical modeling, visualization etc..
Other pressing issues

- Handling spatio-temporal data
  - SQL is not natural to deal with temporal data
- How do we guarantee the data will be there 10 years from now?
  - Data preservation/archival
- Privacy and security !!!
  - Every other day we see some database leaked on the web
- Interaction/visualization..
My research interests

- Managing imprecise, probabilistic, incomplete information in databases
- Probabilistic/statistical modeling of data
- Adaptive query processing
- Data streams
- Data management in Sensor Networks
Next class...

- History of databases + Data modeling
  - Reading: The first chapter in the book
- 2/5/08: Anatomy of a database system (second chapter in the book)