CMSC 724: Introduction

Amol Deshpande

University of Maryland, College Park

January 26, 2012

Outline

Overview of the Class

- Course Website
- We will use a Piazza forum for submitting critiques, questions about the assignments etc.
- No laptops
- Typically won't use detailed slides, but will try to post the class audio/video recordings

Overview of the Class

- We will cover a blend of classic papers + ongoing research
- Textbook:
  - Readings in Database Systems, 4th edition. Mike Stonebraker and Joe Hellerstein.
  - Almost all papers are available online
- Book contains some very nice overview chapters though
  - Modeling: Next class
  - Anatomy/Architecture of a database system: Two classes after that
- This is not CMSC624.
- Prerequisite: CMSC 424
  - Class notes off of my webpage

Grading

- Two exams + 3-4 homework/assignments (50%)
  - Typically small assignments, focus on basics
  - SQL Assignment posted in a couple of days
  - Rest of the *tentative* schedule on the webpage
- Paper critiques + class participation (15%)
  - Critiques mandatory before the class (by 1pm)
  - Some suggestions on how to write on the website
  - Reduced from prior class: average < 1 per class
- 15-min presentation (5%)
  - Two-person teams
  - On a recent paper (VLDB 2011 or SIGMOD 2011)
- A class project (30%)
  - Attempt at objective grading: 4 (proposal) + 10 (lit survey) + 4 (intermediate) + 10 (final)
  - Suggestions soon

Spring 2011 Survey

- Class went:
  - just right (17) vs too fast (5)
- Had background to read the papers:
  - yes (19) vs no (4)
- Overall the work (compared to other 7xx classes)
  - A lot more (9) vs about the same (11)
- SQL/Assignments:
  - 13 (very useful) vs 9 (useful) vs 0 (don't include them)
- CIDR Presentations well liked

Outline

Databases: A Brief History

- 1960's: Computers finally become attractive, and enterprises start using it. Most applications initially used their own data stores.
- Data base: coined in military information systems to denote "shared data banks" by multiple applications
- Why ?
  - Each application had its own format
  - Although the data was there, basically unavailable to other programs
    - Often original object code was lost
- Instead, define a data format, store it as a "data dictionary", and allow general-purpose "data-base management" software to access it
- Issues:
  - How to write data dictionaries? How to access data?
  - Disadvantages of integration: integrity, security, privacy concerns
  - Who controls the data?

Databases: A Brief History

- Objectives of Data-base Management
  - As listed by: Fry and Sibley, in a 1976 ACM Surveys article
- Data Availability
  - Main issue: Data dictionaries
- Data Quality
  - Dealing with integrity violations, data corruption etc.
  - "40% of COBOL programs consist of error-checking stmts"
  - Use "audit trails" (database logs) and "integrity constraints"
- Privacy and Security
  - Need fine-grained control over who sees what in an integrated db
- Management Control
  - DBA functionality

Databases: A Brief History

- Objectives of Data-base Management
  - As listed by: Fry and Sibley, in a 1976 ACM Surveys article
- Data Independence
  - Physical: if the program or ad hoc requests are relatively independent of the physical structure
    - Physical structure includes: pointers, character representations, record-blocking, indexes, sort-order etc...
  - Logical: Ability to make logical changes (say to the schema) without significantly affecting programs
    - Main construct is *views*

Databases: A Brief History

- 1960's
  - Birth of "hierarchical model" and "network model"
    - Both allowed "connecting" records of different types (e.g., connect "accounts" with "customers")
    - Network model attempted to be very general and flexible —- Charlie Bachman received Turing Award for this
  - IBM designed its IMS hierarchical database in 1966 for the Apollo space program; still around today
    - Predates "hard disks"
  - However, both models exposed too much of the internal data structures/pointers etc to the users

Databases: A Brief History

- 1970's: Relational Model
  - Origins in Set Theory
  - Some early work by D.L.Childs (somewhat forgotten)
  - Edgar F. "Ted" Codd: Developed the relational model
    - Elegant, formal model that provided almost complete "data independence"
    - Users didn't need to worry about how the data was stored, processed etc.
    - High level query language (relational algebra)
    - Notion of *normal forms* —- Allowed one to reason about and remove redundancies

Databases: A Brief History

- 1970's: Relational Model
  - Origins in Set Theory
  - Some early work by D.L.Childs (somewhat forgotten)
  - Edgar F. "Ted" Codd: Developed the relational model
    - Elegant, formal model that provided almost complete "data independence"
    - Users didn't need to worry about how the data was stored, processed etc.
    - High level query language (relational algebra)
    - Notion of *normal forms* —- Allowed one to reason about and remove redundancies
  - Famous "debates" between "network" and "relational" folks
  - We will talk more in the next class

Databases: A Brief History

- 1970's: Relational Model
  - Led to two influential projects: INGRES (UC Berkeley), System R (IBM)
  - Also paved the way for a 1977 startup called "Software Development Laboratories"
    - Didn't care about IMS/IDMS compatibility (as IBM had to)

Databases: A Brief History

- 1970's: Relational Model
  - Led to two influential projects: INGRES (UC Berkeley), System R (IBM)
  - Also paved the way for a 1977 startup called "Software Development Laboratories"
    - Didn't care about IMS/IDMS compatibility (as IBM had to)
- 1976: Peter Chen proposed "Entity-Relationship Model"
  - Allowed higher-level, conceptual modeling; easier for humans to think about

Databases: A Brief History

- 1970's: Relational Model
  - Led to two influential projects: INGRES (UC Berkeley), System R (IBM)
  - Also paved the way for a 1977 startup called "Software Development Laboratories"
    - Didn't care about IMS/IDMS compatibility (as IBM had to)
- 1976: Peter Chen proposed "Entity-Relationship Model"
  - Allowed higher-level, conceptual modeling; easier for humans to think about
- 1980: Commercialization/wide-spread acceptance
  - SQL emerged as a standard, in large part because of IBM's backing
  - People still sometimes complain about its limitations
- Late 80's: Object-oriented, object-relational databases
  - Enriching the expressive power of relational model
  - Other proposals for semantic data models

Databases: A Brief History

- Late 80's, early 90's
  - Many database companies, but starting to consolidate
  - Parallel databases beginning to emerge
  - Data mining/OLAP (online analytical processing)
  - Focus on client tools for application development: PowerBuilder (Sybase), Oracle Developer etc.
  - Client-server model becomes popular
- Mid/late 90's
  - Web arrives: Growth of *middleware* that connects web apps to databases
    - Active server pages, Enterprise Java Beans, ColdFusion
  - OLAP matures and becomes mainstream

Databases: A Brief History

- Early 00's to mid 00's
  - A sudden boom in data warehousing/analytics
    - Companies like: Aster Data, Greenplum, Vertica, Kickfire, and probably 10 others
    - Some consolidation recently
  - Column-stores, analytics, streaming data, become important
- Late 00's
  - *map-reduce*: framework for large-scale data analysis
  - *key-value stores:* framework for "scale-first" data management
  - Databases late to react, but have adopted
    - Exploring different design points in integration of databases and map-reduce
    - Transactions and consistency in distributed key-value stores
- Next?

Databases: A Brief History

- A taxonomy of "structured storage" market: Hamilton
- Feature-first: RDBMSs
  - Needed by: enterprise financial systems, human resources, CRM systems
- Scale-first
  - Ability to scale to very large data paramount
  - Parallel DB work sometimes, but most prefer NoSQL solutions
    - Problem is usually transaction processing overhead
- Simple Structured Storage
  - Applications that just need a basic key-value store (e.g., to store images)
  - Examples include: BerkeleyDB, Amazon SimpleDB, many of the key-value stores
- Purpose-optimized Stores
  - Stream processing engines
  - Special-purpose data warehousing products

Databases: Major Conferences

- ACM SIGMOD (Originally SIGFIDET)
- VLDB (very large databases)
- IEEE ICDE (intl. conf. data engineering)
- EDBT (european database technology)
- PODS, ICDT
  - Theory focused
- CIDR
  - A new systems focused conference, perhaps the best one right now to attend

Outline

CMSC 424 in 15 Slides

- Why do we need a database
- What are the key technologies inside a traditional relational database

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...

Example

- 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

A *file system*-based solution

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

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 all customers on street Z
    - Scan /usr/db/customers, and look for (partial) matches for Z in the address field
  - ...

What's wrong with this solution?

- 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
    - Lead to "update", "insertion", and "deletion" anamolies
  - Especially true when programs/data organization evolve over time

What's wrong with this solution?

- 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
- 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?

What's wrong with this solution?

- 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

    CRASH

    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

CRASH -

  - 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

   c. update balance55

- Balance would only reflect one of the two operations
  - Bank loses money

What's wrong with this solution?

- 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?

What's wrong with this solution?

- 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

Outline

Data Abstraction

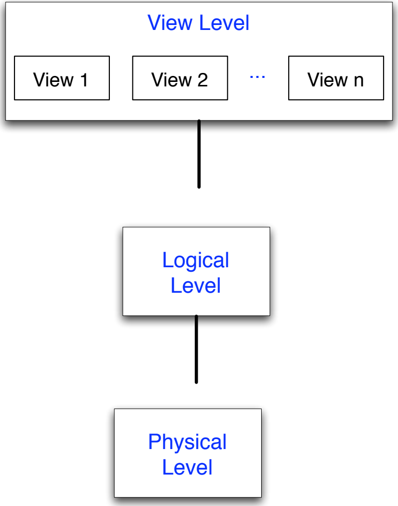
- The key insight, that Ted Codd had, is whats called *data abstraction* (or *data independence*)
- Probably *the* most important purpose of a DBMS
- Goal: Hiding *low-level details* from the users of the system
- Through use of logical abstractions

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 ?



Data Abstraction: Banking Example

- Physical Level:
  - Each table is stored in a separate ASCII file
  - # separated fields
- 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

Data Abstraction: Banking Example

- 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

Outline

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

- Uncertain data management; scalable statistical modeling of data
- Large graph databases: managing and querying
- Energy efficient computing in data centers
- Older work
  - Adaptive query processing
  - Data streams
  - Data management in Sensor Networks

Next class. . .

- History of databases + Data modeling
  - Reading: The first chapter in the book
  - Reading: Ted Codd's paper (summary required)
- After that: Architecture of a database system