Introduction

CPS 196.3
Introduction to Database Systems

Course goals

- Random things you might do (for fun or profit) after taking this course
  - Develop your own database-driven Web sites (like Amazon, eBay, etc.)
  - Be a “power user” of commercial database systems
  - Upgrade your Web sites with XML
  - Explain to friends why MySQL is not a “real” database system without InnoDB or Berkeley DB support
  - … …

Course roadmap

- Relational databases
  - Relational algebra, database design, SQL, application programming
- XML
  - Data model and query languages, application programming, interplay between XML and relational databases
- Database internals
  - Storage, indexing, query processing and optimization, concurrency control and recovery
- Hot research topics
  - Stream data processing, keyword queries on relational databases, semantic Web
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 inside DBMS

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
  - Each account belongs to a branch, has a number, an owner, a balance, …
  - Each branch has a location, a manager, …
  - Persistence: Homer will be pretty upset if his balance disappears after a power outage
  - Query: What's the balance in Homer Simpson's account?
  - Modification: Homer withdraws $100
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

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

- Tons of tricks (not only in query processing, but also in storage, concurrency control, recovery, etc.)
- Different tricks may work better in different usage scenarios
- Same tricks get used over and over again in different applications
- We need a library, or better yet, a server (to support sharing, backup, etc.)

The birth of DBMS – 1

(Pretty drawing stolen from Hans-J. Schek’s VLDB 2000 slides)

The birth of DBMS – 2

(Pretty drawing stolen 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
  - How about branches \(\rightarrow\) accounts?
What’s wrong?

- When data/workload characteristics change
  - The best navigation strategy changes
  - The best way of organizing the data changes
- 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 change!

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
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;
  homework at ATM1 withdraws $100
  Marge at ATM2 withdraws $50
  Initial balance = $400, final balance = ?
  ```

Final balance =

- Homer withdraws $100:
  ```
  read balance;
  if balance > amount then
    balance = balance - amount;
  write balance;
  ```

- Marge withdraws $50:
  ```
  read balance;
  if balance > amount then
    balance = balance - amount;
    write balance;
  ```
Final balance = 

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

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 
  - 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 
- Log updates; undo/redo during recovery
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
- NCR Teradata
- Sybase
- Informix (acquired by IBM)
- PostgreSQL (from UC Berkeley’s Ingres, Postgres)
- Tandem NonStop (acquired by Compaq, now HP)
- MySQL and Microsoft Access

Modern DBMS architecture

- Many details will be filled in the DBMS box
Course information

- **Book**
- **Web site**
  - [http://www.cs.duke.edu/courses/fall03/cps196.3/](http://www.cs.duke.edu/courses/fall03/cps196.3/)
  - Course information
  - Tentative syllabus and reference sections in GMUW
  - Lecture slides, assignments, programming notes
- **Blackboard**: for grades only
- **Newsgroup**: duke.cs.cps196-3

Course load

- **4 homework assignments** (28%)
- **Course project** (25%)
  - Details to be given in the third week of class
- **Presentation** (7%)
  - Sign-up sheet to be given in the third week of class
- **Midterm** (20%)
- **Final** (20%)
  - Comprehensive, but with emphasis on the second half of the course