Introduction

CPS 116
Introduction to Database Systems

Random things to do after this course

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 traditional databases
  - Web searches
  - Data warehousing and data mining
  - Continuous queries: data streams, publish/subscribe, sensor data

Misc. course information

- Book
  - *Database Systems: The Complete Book*, by H. Garcia-Molina, J. D. Ullman, and J. Widom
- Web site
  - `http://www.cs.duke.edu/courses/fall06/cps116/`
  - Course information; tentative syllabus and reference sections in GMUW; lecture slides, assignments, programming notes
- 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

[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 never go upwards—mistake would be mine alone if I made an exam too easy

Course load

- Four homework assignments (35%)
  - Include 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

- Facebook$^+$
  - Tyler Brock and Beth Trushkowsky
- Web-based K-ville tenting management
  - Zach Marshall
- Working with Duke immunologists on a system for capturing and managing computational biology workflows
- Working with Duke & Princeton biologists on a Baboon (real, not acronym) database

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!

```
1001#Springfield#Mr. Morgan
```

```
... ...
00987-00654#Ned Flanders#2500.00
00123-00456#Homer Simpson#400.00
00142-00857#Montgomery Burns#1000000000.00
... ...
```

❖ ASCII file

❖ Accounts/branches separated by newlines

❖ Fields separated by #'s

Query

```
1001#Springfield#Mr. Morgan
```

```
... ...
00987-00654#Ned Flanders#2500.00
00123-00456#Homer Simpson#400.00
00142-00857#Montgomery Burns#1000000000.00
... ...
```

❖ 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

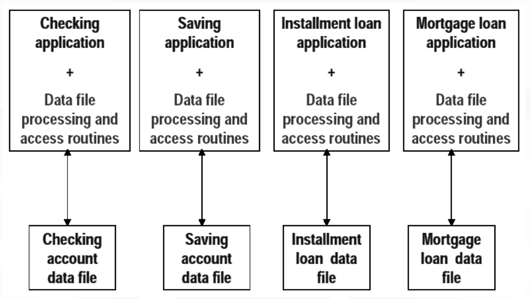
☞And the list goes on…

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

Observations

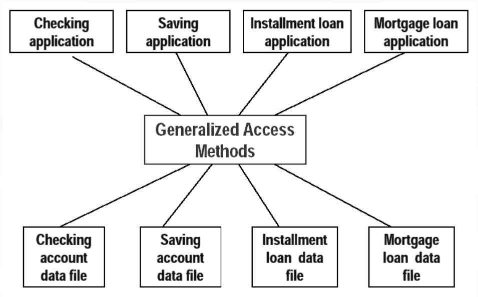
- ❖ Tons of tricks (not only in storage and query processing, but also in concurrency control, recovery, etc.)
- ❖ Different tricks may work better in different usage scenarios (example?)
- ❖ Same tricks get used over and over again in different applications

The birth of DBMS – 1


Checking application + Data file processing and access routines
Saving application + Data file processing and access routines
Installment loan application + Data file processing and access routines
Mortgage loan application + Data file processing and access routines
Checking account data file
Saving account data file
Installment loan data file
Mortgage loan data file

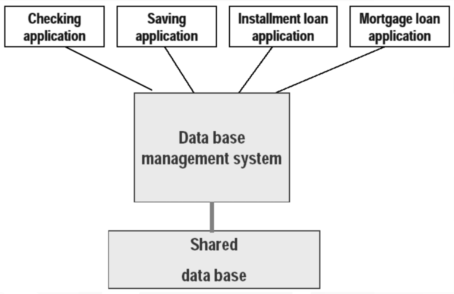
(Pretty drawing stolen from Hans-J. Schek's *VLDB* 2000 slides)

The birth of DBMS – 2


Checking application
Saving application
Installment loan application
Mortgage loan application
Generalized Access Methods
Checking account data file
Saving account data file
Installment loan data file
Mortgage loan data file

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

The birth of DBMS – 3


Checking application
Saving application
Installment loan application
Mortgage loan application
Data base management system
Shared data base

(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 → 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, 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

```
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:

```
read balance; $400
```

Marge withdraws \$50:

```
read balance; $400
if balance > amount then
    balance = balance - amount; $350
    write balance; $350
```

Homer:

```
if balance > amount then
    balance = balance - amount; $300
    write balance; $300
```

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;
```

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 (fun reading: `http://openacs.org/philosophy/why-not-mysql.html`)
  - But

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?

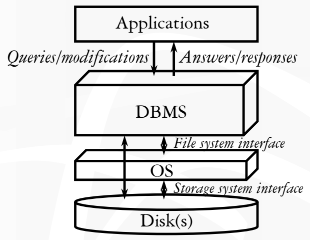
Summary of modern DBMS features

- ❖ Persistent storage of data
- ❖ Logical data model; declarative queries and updates $\rightarrow$ 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 ($\geq$ 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
- ? Microsoft Access

Modern DBMS architecture



- ❖ OS layer is bypassed for performance and safety
- ❖ Many details will be filled in 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