Introduction

CPS 216

Advanced Database Systems

Course goals

- Become a "power user" of commercial database systems
- Learn to apply database ideas/techniques to new applications and other areas of computer science
- Get a solid background for doing database research

CPS 216 vs. CPS 196.3

- Undergraduate database courses (e.g., CPS 196.3 last semester) tend to emphasize more on database design and application programming
- CPS 216 emphasizes more on the implementation techniques of database systems
  - More advanced indexing, query processing, and optimization techniques (e.g., R-trees, linear hashing, histograms, adaptive query processing, distributed databases, XML indexing, etc.)

☞ Those of you who took CPS 196.3 (or an equivalent undergrad database course) before may get different homework problems from the rest of the class

Course roadmap

4

- The basics
  - Relational algebra, database design, SQL, application programming
  - ☞Materials overlap with CPS 196.3 and are covered at a faster pace
- The internals
  - Storage, indexing, query processing and optimization, concurrency control and recovery
  - ☞Some fundamentals overlap with CPS 196.3, but the rest is new
- The extras
  - XML and XML indexing
  - ☞Introduction overlaps with CPS 196.3, but indexing is new

What is a database system?

5

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?

6

- Answer queries (questions) about data
- Update data
- And keep data around (persistent)

- 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, …
  - Query: What's the balance in Homer Simpson's account?
  - Modification: Homer withdraws $100
  - Persistency: Homer will be pretty upset if his balance disappears after a power outage

7

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

8

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

9

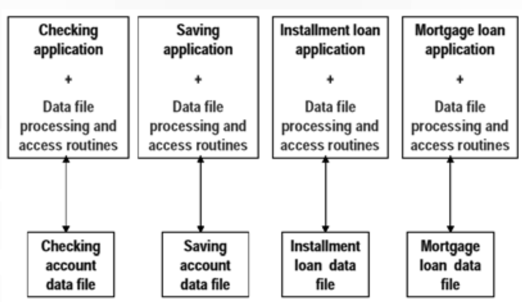
Query processing tricks

- ❖ Tens of thousands of accounts are not Homer's
  - ☞Cluster accounts: Those owned by "A..." go into file A; those owned by "B..." go into file B; etc.
    - Change the script to decide which file to search
- ❖ What happens when the query changes to: What's the balance in accounts 00142-00857?

Observations

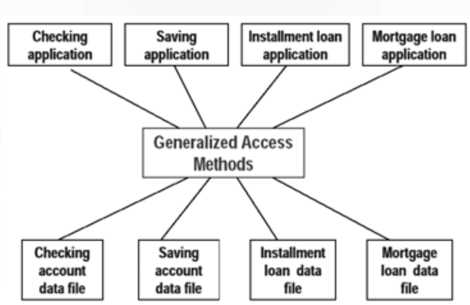
- ❖ Tons of tricks (not only in query processing, but also in storage, concurrency control, recovery, etc.)
- ❖ Different tricks may work better in different usage scenarios
- ❖ Same tricks get used over and over again in different applications
- ☞ We need a library, or better yet, a server (to support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

The birth of DBMS – 1



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

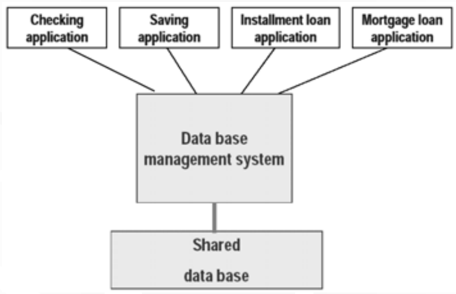
The birth of DBMS – 2



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

The birth of DBMS – 3

13



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

Early efforts

14

- "Factoring out" data management functionalities and from applications 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

15

- 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?

- 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

```
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

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?

relational inside

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 = ?

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:

```
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 initially (fun reading: http://openacs.org/philosophy/why-not-mysql.html)
  - But want to control at much finer granularity
    - Or else one withdrawal would lock up all accounts!

Recovery in DBMS

- Example: balance transfer

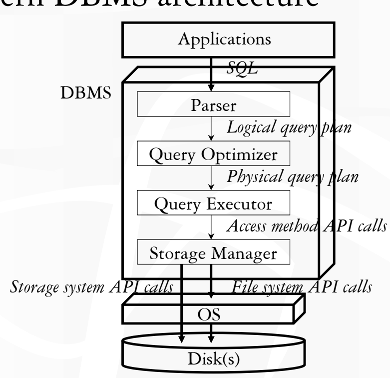
```
decrement the balance of account X by $100;
increment the balance of account Y by $100;
```

- 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 $\rightarrow$ physical data independence
- Multi-user concurrent access
- Safety from system failures
- Performance, performance, performance
  - Massive amounts of data (terabytes $\sim$ petabytes)
  - High throughput (thousands $\sim$ millions transactions per minute)
  - High availability ($\geq$ 99.999% uptime)

Modern DBMS architecture



28

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

29

Course information

- Books
  - Required: *Readings in Database Systems* (a.k.a. the red book), 3rd Ed., edited by M. Stonebraker and J. M. Hellerstein.
  - Recommended: *Database Systems: The Complete Book*, by H. Garcia-Molina, J. D. Ullman, and J. Widom.
- Web site: `http://www.cs.duke.edu/courses/spr02/cps216/`
  - Course info, office hours, syllabus, reference sections in GMUW
  - Lecture slides, assignments, programming notes
- Blackboard: for posting grades only
- Newsgroup: `duke.cs.cps216`

30

Course load

- 4 homework assignments (30%)
  - Programming included
- Open-ended course project (30%)
  - Details to be given in the third week of class
- Midterm (20%)
- Final (20%)
  - Comprehensive, but with emphasis on the second half of the course

Reading assignment for next week

31

- In the red book:
  A Relational Model of Data for Large Shared Data Banks, by E. F. Codd
  - Note: If you are new to relational model and algebra, do not read this paper until we cover these topics in lecture on Monday