EECS 339
Introduction to Database Systems

Peter Scheuermann

Outline

Databases are Everywhere!

- Database Application Examples:
  - Banking systems:
    - accounts & loans, customers, all transactions (banks, ATMs, internet)
  - Airlines reservation systems:
    - reservations by customers, flight schedules, ticket prizes, freq.flyer info
  - Corporate records
    - Sales: customers, products, purchases - and reports on this
    - Human resources: employee records, salaries, tax deductions
    - Manufacturing: production, inventory, orders, supply chain
    - Universities: student enrollments, course offerings, timetabling, grades
  - Telecommunication: calls, bills, calling/SIM cards
  - Health care: patients, prescriptions, drugs, …
  - …

Databases touch all aspects of our lives!

Databases on the Internet…

- Ebay (in 2005)
  - More than 100 back-end databases
  - ca. 5 billion SQL/day
- Wikipedia: (as of Oct. 2006 - http://stats.wikimedia.org/EN/ChartsWikipediaEN.htm)
  - ca. 350 servers
  - 249 languages, millions of articles
    (engl.: 1.5M with 5GB data, 4.1 million updates/month)
  - Behind each language at least one database cluster (2+ dbms)

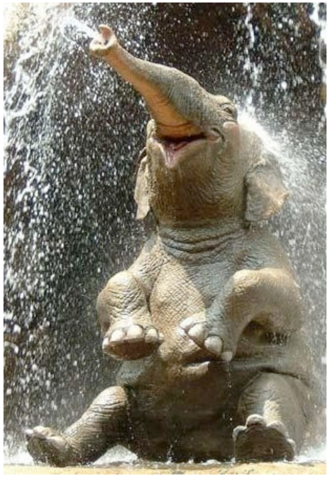
    cf. *Brion Vibber "Scaling and Managing LAMP at Wikimedia", Santa Clara, 2008*.
- May 2008: Yahoo! claims record with 1 Petabyte Database

What is a Database?

- Collection of data central to some enterprise / organisation
- Essential to operation of enterprise
  - Contains the only record of enterprise activity
- An asset in its own right
  - Historical data can guide enterprise strategy
  - Of interest to other enterprises
- State of database mirrors state of enterprise
  - Database is persistent
  - Shared:
    all qualified users have access to <u>the same data</u> for use in a variety of activities.

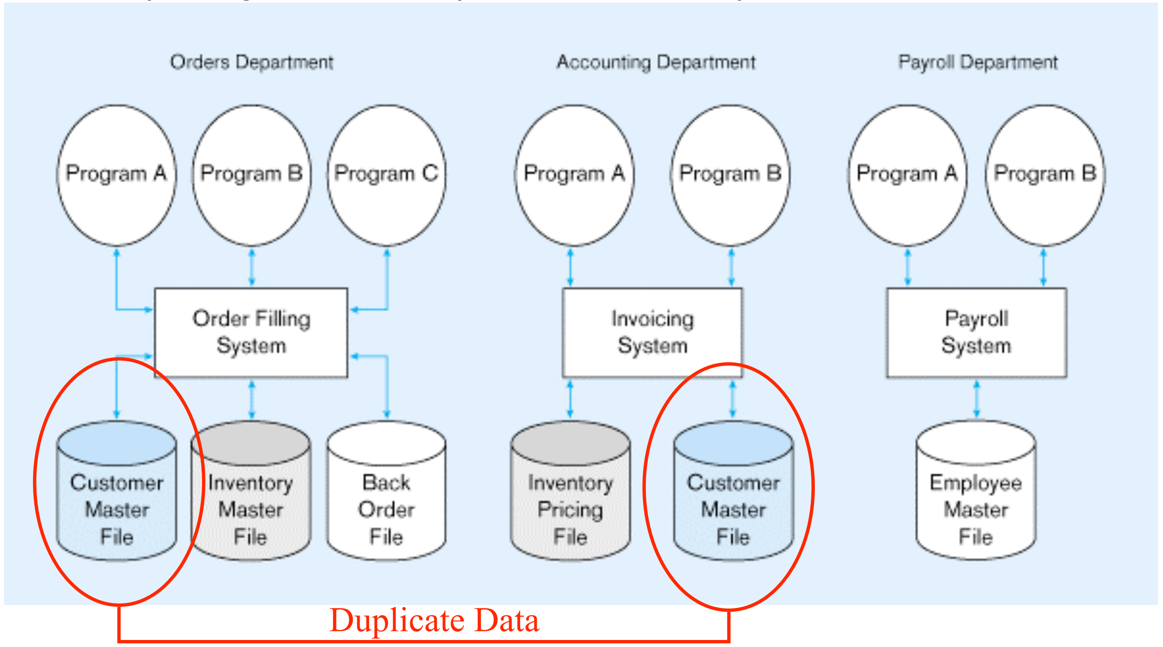
What is a DBMS?

- A **Database Management System (DBMS)** is a program that manages a database:
  - Stores the database on some mass storage providing fail safety (backup / recovery)
  - Supports a high-level access language (e.g. SQL)
    - Application describes database accesses using that language.
    - DBMS interprets statements of language to perform requested database access.
  - Provides transaction management to guarantee correct concurrent access to shared data


[Source: Disney, FL]

Example: File Processing System

- In the early days, information systems were built on top of file systems.
  - Why change? Aren't file systems fast and easy to use?



File System versus DBMS

- Let’s have an example:
  Comparison of an information system implemented on top of the file system (with C/Java/…) versus a realisation based on a DBMS
  - Scenario: Sales system

    Data: customers, orders, products, …
    Applications: orderings, payments & delivery, marketing,…

File vs. DBMS: Data Definition

File System:

```
TYPE customer  = RECORD
  cid  : CARDINAL;
  name : ARRAY [1..20] OF CHAR;
  city : ARRAY [1..20] OF CHAR;
  balance: REAL;
  rabat: REAL;
END;
TYPE product  = RECORD
  pnr   : CARDIONAL;
  title : ARRAY [1..30] of CHAR;
  weight: REAL;
  price : REAL;
  stock : CARDINAL;
END;
TYPE order = RECORD …
```

Data definitions repeated in each program…

Database System:

```
CREATE TABLE customers (
  cid      INTEGER CHECK (cid>0),
  name     VARCHAR(20),
  city     VARCHAR(20),
  balance FLOAT,
  rabat    FLOAT )
CREATE TABLE products (
  pnr    INTEGER CHECK (pnr>0),
  title  VARCHAR(30),
  weight FLOAT,
  price  FLOAT,
  stock  INTEGER CHECK (stock>=0)
)
CREATE TABLE orders ( …  )
```

Data definitions once in the central data dictionary of a DBMS.
=> Same for all applications

File vs. DBMS: Data Access

File System:

```
VAR o : order;
p : product;
fh1, fh2: file_handle;
fh1 = Open(order_file);
WHILE NOT EOF(fh1) DO
o := getnext(fh1);
IF o.month = 10 AND o.day >= 19 THEN
   fh2 = Open(product_file);
   WHILE NOT eof(fh2) DO
     p := getnext(fh2);
     IF p.pnr=o.pnr AND p.stock<100 THEN
       WriteCard(o.pnr, o.quantity);
     END;
    END;
    close(fh2):
END;
END; close(fh1);
```

You have to program it by hand, over and over and over …

Database System:

```
SELECT pnr, quantity
FROM   orders o, products p
WHERE  o.month = 10 AND o.day >= 18
  AND  o.pnr = p.pnr
  AND  p.stock<100
```

Declarative queries
- easy to read and maintain
- evtl. usable from different applications
- efficient evaluation due to automatic optimization

Example: Scientific Data Management

A relatively simple Perl script from a Bioinformatics project that processes a DNA data file (binning of unique 'short-reads')

```
#!perl
use strict;
my $in  = shift or die; # illumina sequence file
my $out = shift or die; # output file for binned sequences
open (FH, "$in") or die;
my @lines = <FH>;
close FH;
my %reads;
for my $line (@lines) {
    chomp $line;
    my $read;
    if ($line =~ /^([A|T|C|G]+)/) {    # check: valid DNS sequence without 'N'?
        $read = $1;
        if ($reads{$read}->{count}) {
            $reads{$read}->{count}++;
        } else
       {   $reads{$read}->{count} = 1;    }
 }}
@lines = "";
open (OUT, ">$out") or warn;
my $m =1; # for rank
for my $read (sort {$reads{$b}->{count} <=> $reads{$a}->{count}} keys %reads){
    print OUT ">$reads{$read}->{count}\-$m\n$read\n";  # count, rank
    $m++;
}
close OUT;
```

The Same in SQL (MS SQL Syntax)

```
  SELECT ROW_NUMBER()OVER(ORDER BY COUNT(*)DESC),
         COUNT(*),
         r_sequence
    FROM Read
   WHERE CHARINDEX('N',r_sequence) = 0
   GROUP BY r_sequence
```

- Assumes input DNA data is now stored in a ‘Read’ table
- Basically a group-by/aggregation query
  - ignores all DNA sequences that include a ‘N’ (unknown) symbol
- Tricky part is to determine the ‘rank’ of a DNA sequence
  - Note: above’s syntax is proprietary to Microsoft SQL Server

Disadvantages of File Processing

- Program-Data Dependence
  - All programs contain full descriptions of each data file they use
- Data Redundancy (Duplication of data)
  - Different systems/programs have separate copies of the same data
  - No centralized control of data => Integrity problems!
- Limited Data Sharing
  - Required data in several, (potentially incompatible) files.
- Lengthy Development Times
  - Programmers must design their own file formats
  - For each new data access task, a new program is required.
- Excessive Program Maintenance
  - 80% of information systems budget

Problems with Data Dependency

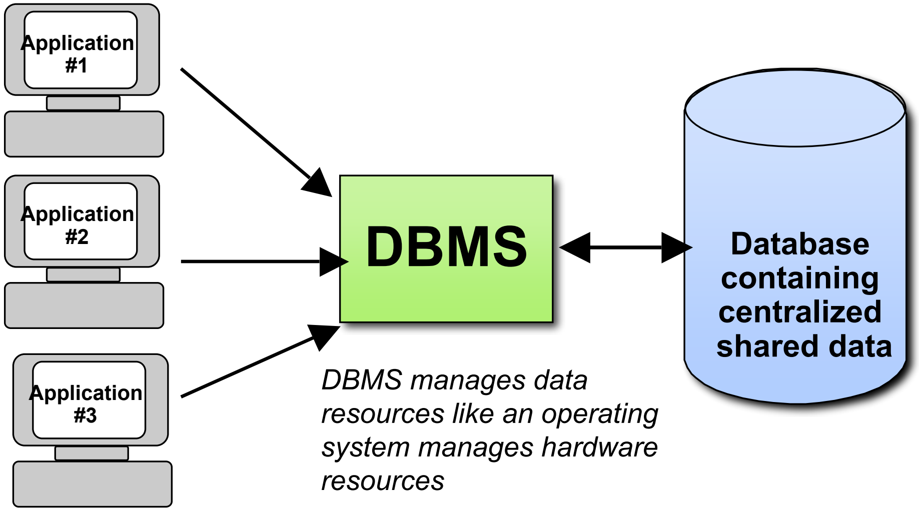
- Each application programmer must maintain their own data
- Each application program needs to include code for the metadata of each file
- Each application program must have its own processing routines for reading, inserting, updating and deleting data
- Lack of coordination and central control
- Non-standard file formats

Problems with Data Redundancy

- Waste of space to have duplicate data
- Causes more maintenance headaches
- The biggest Problem:
  - When data changes in one file, could cause inconsistencies
  - Compromises **data integrity**

Solution: The Database Approach

- Central repository of shared data
- Data is managed by a DBMS
- Stored in a standardized, convenient form



DBMS manages data resources like an operating system manages hardware resources

Advantages of Databases

- **Program-Data Independence**
  - Metadata stored in DBMS, so applications don't need to worry about data formats
  - Data queries/updates managed by DBMS so programs don't need to process data access routines
  - Results in:
    - Reduced application development time
    - Increased maintenance productivity
    - Efficient access

- **Minimal Data Redundancy**
  - Leads to increased data integrity/consistency

Advantages of Databases (cont’d)

- Improved Data Sharing
  - Different users get different views of the data
  - Efficient concurrent access
- Enforcement of Standards
  - All data access is done in the same way
- Improved Data Quality
  - Integrity constraints, data validation rules
- Better Data Accessibility/ Responsiveness
  - Use of standard data query language (SQL)
- Security, Backup/Recovery, Concurrency
  - Disaster recovery is easier

Data versus Information

- **Data**: stored representations of raw facts or meaningful objects such as images and sounds that relate to people, objects, events, and other entities.
  - Structured: numbers, text, dates
  - Unstructured: images, video, documents

- **Information** refers to data that has been processed in some form (filtering, formatting, summarizing).
  It has been rendered appropriate for decision making or other kinds of use in particular contexts

- **Metadata**: Data that describes data

Metadata

- A key idea in DBMSs is for the database itself to store descriptions of the format of the data
- This is stored in the “System Catalogue” or “Data Dictionary”
  - as part of the SQL standard: INFORMATION_SCHEMA
- Eg, you can find that each employee has
  - Identifier which is integer
  - Name which is a string of up to 30 characters
  - Address which is string of up to 60 characters
  - Salary which is integer
- This sort of information is often called *meta-data*

Metadata as Data

- The schema, and other meta-data, is essential to working with the data
  - Data is useless if one can’t interpret it
  - Eg Oliver Ghica, 162,47,447
    - Is 162 weight in pounds, salary in $/hrs, height in cms, room number, or something else?
  - Over time, the people may leave who know what a value or string means
- Meta-data should be stored with the data it describes
  - And there should be some facilities for asking about and updating the meta-data

=> Another core idea of database systems

Short History of DBMS

- **Early Database Applications:**
  - The Hierarchical and Network Models (CODASYL) were introduced in mid 1960’s and dominated during the seventies. A bulk of the worldwide database processing still occurs using these models (legacy systems).
- **Relational Model based Systems:**
  - The model that was originally introduced in 1970 by **E. Codd** (Turing award!). It was researched and experimented with in IBM and universities. Relational DBMS products emerged in the 1980’s and are now the norm since the 1990s.
- **Object-oriented applications:**
  - OODBMSs were introduced in late 1980’s and early 1990’s to cater to the need of complex data processing in CAD and other applications. They sought to integrate the class definitions in an OO language (eg C++) with the way the data is stored persistently. Their use has not taken off much…
- **Data on the Web and E-commerce Applications:**
  - To express many different sorts of data that need to be exchanged between cooperating businesses, the recent standard is **XML (eXtended Markup Language)**. Main features: data is *semi-structured* and *self-describing.*

Central DBMS Services

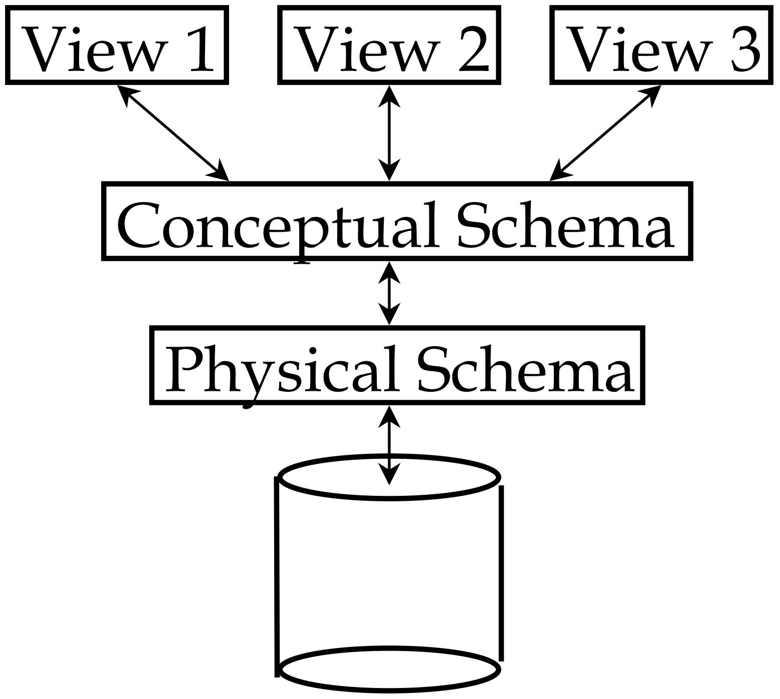
- Data Independence
- Declarative Querying
- Transaction Management
  & Concurrency Control

Data Models

- A *data model* is a collection of concepts for describing data.
- A *schema* is a description of a particular collection of data, using the a given data model.
- The *relational model of data* is the most widely used model today.
  - Main concept: *relation*, basically a table with rows and columns.
  - Every relation has a *schema*, which describes the columns, or fields.

Levels of Abstraction

- Many *views*, single *conceptual (logical) schema* and *physical schema*.
  - Views describe how users see the data.
  - Conceptual schema defines logical structure
  - Physical schema describes the files and indexes used.



** Schemas are defined using DDL; data is modified/queried using DML.*

Example: University Database

- Conceptual schema:
  - *Students(sid: string, name: string, login: string, age: integer, gpa:real)*
  - *Courses(cid: string, cname:string, credits:integer)*
  - *Enrolled(sid:string, cid:string, grade:string)*
- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.
- External Schema (View):
  - *Course_info(cid:string,enrollment:integer)*

Data Independence *

- Applications insulated from how data is structured and stored.
- *Logical data independence*: Protection from changes in *logical* structure of data.
- *Physical data independence*: Protection from changes in *physical* structure of data.

* *One of the most important benefits of using a DBMS!*

Transaction: An Execution of a DB Program

- Key concept is *transaction*, which is an *atomic* sequence of database actions (reads/writes).
- Each transaction, executed completely, must leave the DB in a *consistent state* if DB is consistent when the transaction begins.
  - Users can specify some simple *integrity constraints* on the data, and the DBMS will enforce these constraints.
  - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
  - Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the user's responsibility!

Queries in a DBMS

- DBMS provides a specialized language for accessing data
  - **Query Language**
  - Can be further distinguished between
    - DML - Data Manipulation Language
    - DDL - Data Definition Language
    - DCL - Data Control Language
- Standard for relational DBMS: **SQL**
  - Based on formal query languages:
    Relational Algebra and Relational Calculus
- Queries are evaluated as efficient as possible
  - Huge influence of physical design

Declarative Queries: “What” not “How”

- It is convenient to indicate declaratively *what* information is needed, and leave it to the system to work out *how* to process through the data to extract what you need
  - Programming is hard, and choosing between different computations is hard
- Users should be offered a way to express their requests declaratively
  - A query language can be based on logic
  - Select…where…

DB System Requirements

- **High Availability**: on-line => must be operational while enterprise is functioning
- **High Reliability**: correctly tracks state, does not loose data, controlled concurrency
- **High Throughput**: many users => many transactions/sec
- **Low Response Time**: on-line => users are waiting
- **Long Lifetime**: complex systems are not easily replaced
  - Must be designed so they can be easily extended as the needs of the enterprise change
- **Security**: sensitive information must be carefully protected since system is accessible to many users
  - Authentication, authorization, encryption

Roles in Design, Implementation and Maintenance of a DB System

- System Analysts
  - specifies system using input from customer; provides complete description of functionality from customer's and user's point of view
  - Conceptual database design
- Database Designer
  - specifies structure of data that will be stored in database (logical & physical databse schemas)
- DB Application Programmer
  - implements application programs (transactions) that access data and support enterprise rules
- Database Administrator (DBA)
  - maintains database once system is operational: space allocation, performance optimization, database security, deals with failures and congestion
- End-Users
  - often unaware that they are dealing with data in a DBMS
- DBMS Vendor's Software Engineers

Summary

- DBMS used to maintain & query large datasets that are shared by many application programs/users
- Some powerful ideas:
  - Program-Data Independence
  - Controlled Data Redundancy
  - Declarative Queries
  - Transactions
- Every ‘knowledge worker’ or scientists needs database know-how, as do all IT experts (application developers, software engineers, system analysts, …) - not just DBAs
- Databases are one of the broadest and most useful areas in CS and IS