Introduction to RDBMs

What is a Database?

Examples of Databases:
- Oracle
- Informix
- Card Catalog
- File Cabinet
- Rolodex

A database organizes a collection of data, and facilitates the maintenance and accessing of that data.

DBMS – A (very) brief history

1960s
- First Network and Hierarchical Databases appear

1970s
- Birth of the Relational Data Model (Codd, 1970)
- Entity-Relational Modeling introduced (Chen, 1976)
- These discoveries led to commercial database systems

1980s
- Personal Computer era leads to desktop & C/S RDBMs
- Large commercial vendors begin to appear

1990s
- Internet era ushers in return to thin-client era
- Advanced DBs: Distributed, O-R, Knowledge-based

What is a Database?

A database is an implementation of a data model
- The data model is a representation of the “real world”
- The “real world” is mapped to a Universe of Discourse
  - Defines the scope of the data and relationships that can be expressed in the model.

Each of the examples from the previous slide are implementations (in some form or another), of a data model

Typically, we think of a database as a software implementation of a data model.
What is a DBMS?

A *Database Management System* (DBMS) is a software system that includes the database, plus a supporting set of services:
- Transactions
- Concurrency Control
- Fault Tolerance
- Recovery
- Data access
- Security
- Etc…

DBMS Architecture

**Internal/Physical Schema:**
- Describes the physical storage structure of the database via a physical data model.

**Conceptual Schema**
- Describes the structure of the whole database to users
- Hides details about the physical storage implementation.

**External/View level**
- Each external schema describes a part of the database that a particular user group is interested in
- Hides the rest of the database from that user group.
  - A view can be comprised of a subset of database data or from virtual data (which is derived from the database).

DBMS Components

(Elmasri & Navathe p. 41)
DBMS Architectures: 1-tier

Centralized DBMS:
- User connects through a remote terminal – but all processing is done at centralized site.

DBMS Architectures: 2-tier

Clients:
- Use appropriate interfaces through a software module to access and utilize the various server resources.
  - ODBC: Open Database Connectivity standard
  - JDBC: for Java programming access
- Connected to the servers via some form of a network.

DBMS Architectures: 3-tier

Common for Web apps
Application Server Layer:
- Stores the web connectivity software and the business logic part of the application used to access the corresponding data from the database server
- Acts like a conduit for sending partially processed data between the database server and the client.

3-tier Architecture Enhances Data Independence:
- Database server only accessible via middle tier
- Clients cannot directly access database server

DBMS Example

Why is a DBMS needed at all?
- Can’t we just code these things ourselves?

Consider a “real world” application:
“ASU is re-engineering its database to incorporate all information about Arizona State University. This includes information on students, faculty, alumni, staff, community outreach programs, academic units, sponsored units, Colleges, Schools, donors, research projects, …”

We can write this application ourselves, we’re great programmers – right?
Software issues:
- How much memory do we need
  - Each individual type might be mapped to a class in Java
  - Student, Faculty, Staff
  - The size of the object is roughly the size of the data held in that object.
  - We also must store binary objects
    - Pictures of staff, research projects, facilities, etc.
    - There may be other digital media
- Can we fit this all in memory?
  - Your typical PC nowadays ships with 256MB RAM.
  - A low-end server ships with 1GB RAM.
  - Higher-end server do have higher memory capacities, but can your runtime environment address them all?
  - Virtual Memory support in the OS helps
- What is the cost of swapping out to disk?

What about overlapping read/writes to disk? We want many users to use our database at once
- Transactions
  - Preserve the ACID properties of a database-related action
- Concurrency Control
  - Algorithms and strategies for dealing with multiple clients reading and writing to the same physical data store.

What about ensuring our data is not lost?
- Persistence
  - RAM is volatile storage
  - Our users need to know there data is not lost if the program or the computer crashes
- Solution: write data to the file system

Transactions & Concurrency Control Example

```
begin <transaction T1>
  loc = index.findOffset('111-22-3333');
  db.seek(loc);
  studentRec = db.read(sizeof(ASUStudent));
  studentRec.year++;
  db.seek(loc);
  db.write(studentRec);
end <T1>
```

```
begin <transaction T2>
  loc = index.findOffset('111-22-3333');
  db.seek(loc);
  studentRec = db.read(sizeof(ASUStudent));
  studentRec.year++;
  db.seek(loc);
  db.write(studentRec);
end <T2>
```

- If student with SSN '111-22-3333' was originally a freshman (year 1), then:
  - What year is s/he after executing both transactions?
  - What year should s/he be in?
- We can solve this with techniques like monitors and semaphores
DBMS Example – Data Independence

What happens if we want to port our database to new media?

- A new, faster disk array comes out, but the physical structure is different (pages sizes, clusters, etc.)
- If we have tied our components and query language to the underlying physical structure, then we have to rewrite all this stuff?

What happens if we add a new attribute to a data type?

- We can get disk fragmentation – “holes” in the physical layer

What happens if our stakeholders force a change to our support Operating System?

- The OS provides our I/O APIs

These issues are known as Physical Data Independence and Logical Data Independence (stay tuned)

DBMS Example – Distributed Architecture

To support our multiple users, we need to provide remote access to our database

- Client-server architectures
  - Users or programs needing to read or write data from/to the database need a way to “connect”
  - Can write a low-level socket protocol
    - Not portable
    - Lot of code goes into supporting the protocol
    - Must deal with failure scenarios due to bursty or unstable traffic
      - e.g. What happens if the users network connection goes down in the middle of a transaction?
    - Efficiency – the underlying transport layer protocol can have impacts on network efficiency and data marshalling.

DBMS Example – Distributed Architecture

To support our multiple users, we need to provide remote access to our database

- Client-server architectures
  - Users or programs needing to read or write data from/to the database need a way to “connect”
  - Can write a low-level socket protocol
    - Not portable
    - Lot of code goes into supporting the protocol
    - Must deal with failure scenarios due to bursty or unstable traffic
      - e.g. What happens if the users network connection goes down in the middle of a transaction?
    - Efficiency – the underlying transport layer protocol can have impacts on network efficiency and data marshalling.

DBMS Example – Query Languages

We need to provide a way for clients to search and update the database.

- Query Languages
  - “Programming” each query is labor-intensive and would be tightly coupled to the physical schema
  - Not friendly for end users
  - Solution: Abstract into a query language
    - Now we have to write the interpreter!

- Query Processor
  - An program that interprets queries specified in our abstract language and then makes the proper native calls to fetch the data.

Data Independence

Logical Data Independence:

- The capacity to change the conceptual schema without having to change the external schemas and their associated application programs.

Physical Data Independence:

- The capacity to change the internal schema without having to change the conceptual schema.
- For example, the internal schema may be changed when certain file structures are reorganized or new indexes are created to improve database performance.
**DBMS Example – Fault Tolerance**

What happens if our disk crashes?
- Backups – Hopefully we will have devised an archive/restore facility
- Data integrity / Consistent state
  - What about the transactions we are in the middle of?
    - Did they finish?
    - What do we tell users when the system comes back up?
- Prevention
  - Is the system capable of “hot-swapping” to an alternate physical storage device?
    - Mirroring
    - Journaling

This area of study is known as **Fault Tolerance**

**DBMS Example - Security**

How do we protect the security of data & the system?
- Authentication – OK, there are known solutions
- Authorization
  - There is sensitive data in our database!
    - Social Security Numbers
    - Pay levels
  - What data are you allowed to see?
  - What behaviors are you allowed to invoke?
  - Implementing an access control list (ACL) and mapping it down to the level of fine-grained data and behaviors is a complex and potentially inefficient task.
- Separation of concerns
  - Programmers and Administrators do not have the right to see your sensitive data!

**DBMS Advantages**

A production-ready DBMS provides all of these benefits for you:
- Persistence
- Transaction Support
- Concurrency Control
- Client/Server Architecture Support
- Query Languages and Query Processing
- Physical Data Independence
- Logical Data Independence
- Fault Tolerance
- Security

**DBMS Disadvantages**

Why would you **not** use a DBMS?
- Introduces overhead on simple operations
- Cost and licensing issues
  - TCO - do you need special hardware?
  - Can your customer base afford it?
- Learning Curve / Training Issues
- Need for specialized skills to maintain
- Software Maintenance
  - “Today’s software are tomorrow’s legacy applications”
- Risk of data migration issues
- Simplicity of relational data modeling (for RDBMS)

Do you need a DBMS to store your address book?
DBMS Actors on the Scene

Database Administrators (DBAs)
- System configuration and tuning
- Security, Archival, and Recovery procedures

- Database Implementors
  - Database Designers
    - Requirements Analysis (data perspective)
    - Create and maintain the data model
  - Systems Analysts
    - Determine the hardware and software requirements
    - Cross-level optimizations: physical and logical
  - Application Programmers
    - Implement specific views for groups of users
    - Workflows – transaction sequences – support biz processes

DBMS – Actors on the Scene (cont.)

End Users
- Participate in requirements analysis
- May vary in frequency and types of access
  - Casual Users: occasional usage, use query language
  - Naive Users: constant, continuous usage
    - Interface through applications: workflow, etc.
    - Canned transactions – provide specific business functions
  - Sophisticated Users:
    - The ability to use various DBMS technologies on their own
  - Stand-alone Users: maintain personal databases
  - Mobile Users:
    - Check-out data for a period of time, then check it back in
    - May communicate with a database asynchronously (email)