Introduction

CSE462 Database Concepts

Demian Lessa

Department of Computer Science and Engineering
State University of New York, Buffalo

January 19, 2011

What is CSE462 about?

The Database Concepts course studies the representation and management of data.

Terminology

- **Database** is a collection of logically related data elements (facts).
- A **Database Management System (DBMS)** is a software system providing an interface to the database. It allows users to create, maintain, and query databases.
- A **database system** is a term typically used to represent the DBMS and its associated database, both conforming to the principles of a particular data model.
- A **data model** describes how data elements are structured.
  - Hierarchical: trees or hierarchies.
  - Network: nodes and their relationships.
  - Relational: sets satisfying logical predicates.
  - Object-Oriented: objects as used in OOP.
  - Semi-Structured: the structure of the data elements is usually described as part of the data itself (i.e., self-describing).

Terminology (cont.)

- **Schema** is a description of how data is structured.
  - Usually defined at set-up time and rarely changes.
  - Part of the database's metadata.
  - Design can be complex, may use methodologies and/or tools.
- **Data** is the actual "instance" of the database.
  - Must adhere to the database's schema (if any).
  - May change rapidly.
  - Schema :: instance in DBs **as** types :: variables in PLs.
- **Data Definition Language** (**DDL**)
  - Commands for defining and modifying database schemas.
- **Data Manipulation Language** (**DML**)
  - Commands to manipulate data in the database.
  - **CRUD** operations: **C**reate, **R**ead, **U**pdate, and **D**elete data.
  - Also called a query language.

Roles

- **Database Designer**
  - Establishes the schema.
- **Database Administrator (DBA)**
  - Defines and modifies schemas.
  - Defines and modifies physical storage and access methods.
  - Grants and revokes users authority to access the database.
  - Monitors and tunes database performance.
  - Creates and supervises backup and restore procedures.
  - In short, the one who keeps the system running smoothly.
- **Database Programmer (DBP)**
  - Writes database programs using the query language.
  - E.g., writing views and stored procedures in Oracle PL/SQL.
- **Database Application Developer**
  - Writes applications that query and modify the database (**CRUD**).
  - E.g., writing Java applications that connect to Oracle via JDBC.

Examples of DBMSs

Well-known:

- IBM DB2 and Oracle (commercial)
  - relational, object-oriented, XML
  - cross-platform, express edition
- PostgreSQL (open source)
  - relational, object-oriented, XML
  - cross-platform
- MS SQL Server (commercial)
  - relational, XML
  - **Windows only**, express edition
- MySQL (open source and commercial)
  - relational
  - cross-platform

Examples of DBMSs (cont.)

Not so well-known:

- Derby, H2, HSQLDB (open source)
  - relational
  - cross-platform, server and embedded (**Java**) modes
- SQLite (public domain)
  - relational
  - cross-platform, **embedded only** (**C**/**C++**)
- db40 (open source and commercial)
  - object-oriented
  - cross-platform, server and embedded modes (**Java**, **.NET**)
- Neo5j (open source and commercial)
  - graph (network)
  - cross-platform, **embedded only** (**Java**)
- InfoGrid (open source)
  - graph (network)
  - cross-platform, server (**Java**)

DBMS Interfaces

- Interactive (DBA, DBP)
  - console: SQL*Plus, psql, mysql
  - GUI: TOra, pgadmin3, MySQL Workbench
- Database Abstraction APIs (Applications Programming)
  - Abstracts away particularities of the DBMS.
  - Applications become (in theory) portable across DBMSs.
  - Advanced features of particular DBMSs may not be available.
  - Requires a driver/provider specification (e.g. connection string).
  - E.g., ODBC, OLEDB, and ADO.Net (Windows).
  - E.g., JDBC (Java), DBI (Perl), DB-API (Python), MDB2 (PHP).
- DBMS APIs (Applications Programming)
  - Many DBMSs expose their features through native APIs.
  - Applications are locked-in to a particular DBMS (API).
  - Advanced features may be fully accessible through the APIs.
  - E.g., libpq (PostgreSQL), libmysql (MySQL), SQL Native Client (MS SQL Server).

Chronology

- 1950s and early 1960s
  - Mainframe computing.
  - Data processing uses magnetic tapes for storage.
  - Data on tapes is accessed sequentially.
- Late 1960s and 1970s
  - Hard disks provide random access to data.
  - Network and hierarchical models in widespread use.
  - Seminal work on the relational data model by Ted Codd.
  - High-performance transaction processing.
- 1980s
  - Relational databases evolve into commercial systems.
  - SQL becomes an industrial standard (ANSI SQL-86).
  - Parallel and distributed databases.
  - Object oriented databases.

Chronology (cont.)

- 1990s
  - Large decision support and data-mining applications.
  - Large multi-terabyte data warehouses.
  - Internet and web commerce.
- 2000s
  - XML and XQuery standards.
  - Automated database administration.
  - High-speed networks.
- 2010s
  - NoSQL databases (déjà vu?).
  - Mobile computing.
  - Cloud computing.

Example #1: To-Do List

Register for Spring semester	due 12/15/10
Interview with Google	on 12/18/10
Go to the gym	every Mon-Wed-Fri
Attend orientation week	from 01/09/11 to 01/14/11
Buy a new computer	

To-Do List: activities/tasks and their associated dates.

- You probably have one in your cell phone/pda!
- How is this different from plain text?
- Whay can you possibly do with it but not so easily with plain text?

Example #2: Calendar and Contacts

Event	When	Who	Where
Lunch	01/20 12:00	Joe	Commons
Dinner	01/22 19:00	Mary	Panera
Doctor	02/03 14:00	Dr. House	Rosewell
Movie	01/23 19:00	Mary	Galleria Mall

Calendar: Events, dates, times, people, and locations.

Who	Phone	E-Mail	Office
Joe	1-2343	joed	222 Bell Hall
Mary	7-3345	mryjn	765 Swift St.
Dr. House	4-9887	housemd	12 Rosewell Park

Contacts: People and ways to contact them.

- How do we integrate contact information into our calendar?

Example #2: Calendar and Contacts (cont.)

Should we expand the calendar table to include contact data?

Event	When	Who-name	Who-phone	Who-email	Who-office	Where
Lunch	01/20 12:00	Joe	1-2343	joed	222 Bell Hall	Commons
Dinner	01/22 19:00	Mary	7-3345	mryjn	765 Swift St.	Panera
Movie	01/23 19:00	Mary	7-3345	mryjn	765 Swift St.	Galleria Mall
...						

Calendar: As before, but expanded with contact information.

Do you see any problems with the above?

What if we linked calendar and contact data?

- Could we use contact names?
- Are there potential problems?
- How do we follow links?
- How would you do it programmatically?

Example #3: University Information System

- Person attributes:
  - SSN
  - UBID
  - Given name
  - Family name
  - ...
- Student **is a** Person who:
  - registers for courses
  - attends lectures and recitations
  - receives grades
  - ...
- This is an example of yet another type of information.
  - Where else have you seen such relationships before?

Database Approach

DBMS Features

- Database definition by specifying schemas using a DDL.
  - Schema definitions may be kept in the DBMS itself (**catalog**).
- Database manipulation through **CRUD** operations using a DML.
- Efficient, concurrent access to very large volumes of data.
- Scale to large datasets while retaining efficiency.
  - Personal address book, ~10^2–10^3KB.
  - Departamental calendar, 10^1+ users, ~10^2–10^3MB.
  - ERP/CRM systems, 10^3+ users, ~10^1–10^3GB.
  - Amazon, ~60 million users, ~40+TB.
  - IRS data warehouse, ~150TB in 2007.
  - AT&T call records, ~320TB.
  - Yahoo! browsing habits, ~2PB in 2008, ~10^1PB by 2009 (?).
  - NSA call database, world climate, biological (genome), gis, etc.

Database Approach (cont.)

DBMS Features (cont.)

- **A**tomicity.
  - Data modifications are grouped into **transactions**.
  - A transaction is an atomic unit of work– completely succeeds or fails.
- **C**onsistency.
  - Transactions take the database from one consistent state to another.
- **I**solation.
  - Operations cannot access data modified by incomplete transactions.
- **D**urability.
  - Recover committed transaction updates against any kind of crash.
- Protection data from unauthorized access.
- Active response to database modifications.
- Distribution across multiple sites.

Database Approach (cont.)

Abstraction: Logical vs Physical

A **logical data model** hides representation details and provides users with a **conceptual view** of the database. Each user may see a different view of the database, consisting only of data of interest to that user.

Storage and access methods (e.g., file organization on disk, indexes) may be modified without affecting the conceptual view. E.g.,

```
SELECT When, Where
FROM Calendar
WHERE Who = 'Mary';
```

Users need not know whether data is stored as, say, fixed or variable length strings, or whether an index is used to answer the query. This is **physical data independence**.

Database Approach (cont.)

Abstraction: Logical vs Physical (cont.)

Logical data independence is slightly more subtle and provides insulation from changes in the logical structure of the database.

Conceptual views provide an interface that must remain unchanged after changes to the logical database structure, and an implementation that must be adapted to account for such changes. As far as users (and programs) are concerned, the views are virtually unchanged.

Database Approach (cont.)

Abstraction: Declarativity

A **declarative query language** allows users to describe the data of interest rather than encode computational steps to retrieve such data.

Based on the relative costs of available access methods, the DBMS determines the best strategy (plan) to retrieve query answers. E.g.,

```
SELECT When, Where
FROM Calendar
WHERE Who = 'Mary';
```

Users are oblivious as to whether the DBMS performs, e.g., a table scan or an index scan to answer the query.

Database Approach (cont.)

Benefits

- Sharing data across multiple users.
- Representing complex relationships among data.
- Enforcing integrity constraints on the database.
- Controlling redundancy in data storage and in development and maintenance efforts.
- Restricting unauthorized users from accessing the data.
- Providing multiple views for different classes of users.
- Providing backup and recovery services.
- Providing efficient query processing.
- Providing persistent storage for users and programs.
- Drawing rule-based inferences and actions.

Database Approach (cont.)

Benefits

- Reduced application development time.
- Flexibility: database structure may evolve as new requirements are defined.
- Availability of up-to-date information– critical for OLTP systems such as airline, hotel, car reservations.
- Economy of scale: by consolidating data and applications across departments, wasteful overlap of resources and personnel can be avoided.

Questions? Comments?

We will revisit all these topics in more depth during the semester.