Introduction to Database Systems

UVic C SC 370

Dr. Daniel M. German
Department of Computer Science
University of Victoria

May 5, 2004 Version: 1.1.1

Overview

- What is a DBMS? what is a relational DBMS?
- Why do we need them?
- How do we represent and store data in a DBMS?
- How does it support concurrent access and system failures?

Motivation: how do we store lots of data?

- Assume you work for Walmart (and database management systems have not been invented) and you are asked to write a collection of programs that can store and retrieve every single sell in every store of the chain (could be a Tbyte of info)

Motivation...

- How do you do it?
  - How do you find and retrieve data?
  - How many files do you need?
  - How many disks do you need?
- And if we add complexity:
  - How do you operate on the data?
  - How do you allow concurrent access and modifications to the data?
- **The problems are not trivial**

Database

- A **database** is a collection of data, typically describing the activities of one or more related organizations. A database is composed of:
  - **Entities**
  - **Relations**
- A **Database Management System** or **DBMS** is software designed to assist in maintaining and utilizing large collections of data.

What are we going to cover?

- **Database design and application development**: how do we represent the world with a database?
- **Data analysis**: how can we answer questions about the enterprise using this data?
- **Concurrency and robustness**: How does a DBMS allow many users to access data concurrently, and how does it protect against failures?
- **Efficiency and Scalability**: How does the database cope with large amounts of data?

A bit of history

- Early 1960s: **Charles Bachman** at GE creates the first general purpose DBMS *Integrated Data Store*. It creates the basis for the *network model* (standardized by CODASYL)
- Late 1960s: **IBM** develops the *Information Management System* (IMS). It uses an alternate model, called the *hierarchical data model*. SABRE is created around IMS.
- 1970: **Edgar Codd**, from *IBM* creates the *relational data model*. In 1981 Codd receives the Turing Award for his contributions to database theory. *Codd passed away 2 weeks ago (April 2003)*

A bit of history...

- 1980s **SQL**, developed by *IBM*, becomes the standard query language for databases. SQL is standardized by ISO.
- 1980s and 1990s, IBM, Oracle, Informix and others develop powerful DBMS.
- In the Internet Age, DBMS are showing how useful they can be.

Why do we use a DBMS?

- **Data independence**
- **Efficient data access**
- **Data integrity and security**
- **Data administration**
- **Concurrent access and crash recovery**
- **Reduced application development time**

See textbook, section 1.4

The Relational Data Model: introduction

- A **data model** is a collection of high level description constructs that hide many low-level storage details
- Most current DBMS use the **relational data model**
- The central data description in this model is the **relation** (a set of tuples –same as in set theory mathematics)
- For convenience, we refer to each tuple as a row
- A **schema** is a description of the data in terms of the data model. In the relational model the schema looks like:

$$RelationName(field_1 : type_1, ..., field_n : type_n)$$

Relational Data Model: example

- A relation of students:

$$Students(sid : string, name : string, age : integer, gpa : real)$$

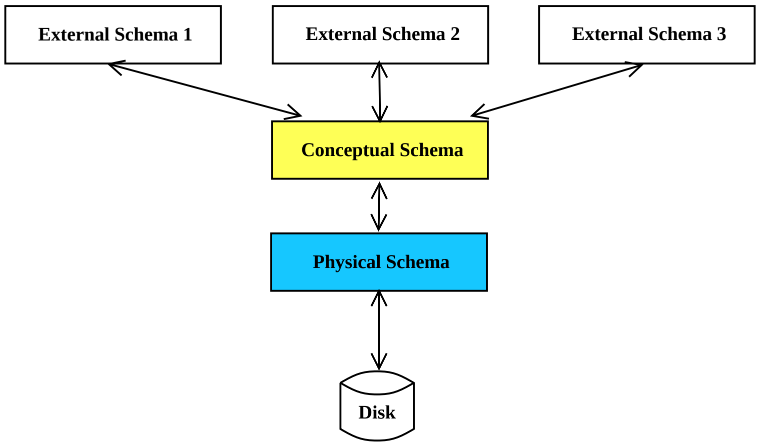
- An instance of the students relation can be represented as:

sid	*name*	*login*	*age*	*gpa*
53666	Jones	Jonescs	18	7.4
53668	Smith	smithee	18	7.8
53650	Smith	smithmath	19	6.4
53831	Madayan	madayan@music	11	8.0
53832	Guldy	guldu@music	12	2.0

Relational Data Model: example

- Each row is a *tuple* in the relation (a *record* in the DBMS)
- We can add **integrity constraints** (assertions on the data) such as every student has a different *sid*

Levels of Abstraction in a DBMS



Conceptual Schema

- The **the conceptual schema** describes the data stored in the database
  - In a relational database it describes all the relations stored in the database
- Creating a good conceptual schema is not a simple task, and it is called **conceptual database design**. It involves:
  - Determining the different relations needed
  - The number of fields per relation
  - The type of each field
  - etc.

Example of a Conceptual Schema

```
Students(sid:string, name: string, login: string,
         age: integer, gpa: integer,gpa: real)
Faculty(fid: string, fname: string, sal: real)
Courses(cid: string, cname: string, credits: integer)
Teaches(fid: string, cid: string)
Enrolled(sid: string, cid: string, grade: string)
...
```


Physical Schema

- The **physical schema** specifies how the relations are actually stored in secondary storage devices
- It also specifies auxiliary data structures (**indexes**) used to speed up the access to the relations
- Decisions about the physical schema depend upon:
  - Understanding how the data is going to be accessed
  - The facilities provided by the DBMS

Example of Physical Schema

- Store all relations in unsorted files of records
- Create indexes in the first column of every relation, and in the *sal* column of *faculty*
- ...

External Schema

- The **external schema** is a refinement of the conceptual schema
- Allows customized and authorized access to individual users or groups of users
- Every database has **one** conceptual and **one** physical schema, but it can have *many* external schemas
- Each external schema: users
  - is tailored to a particular group of users
  - consists of **one or more views** and **relations** of the conceptual schema
- A **view** is conceptually a relation, but its records are not stored in the database; instead, they are computed from other relations.

Example of External Schema

```
CourseInfo(cid: string, fname: string,
           enrollment: integer)
```


Data Independence

- **Data Independence** means that programs are isolated from changes in the way the data is structured and stored.
- As long as we maintain the external schema, we can modify the other 2 schemas of an application
  - **Logical Data Independence**: users are shielded from the logical structure of the data (e.g. a relation is split into 2 or more)
  - **Physical Data Independence**: As long as the conceptual schema remains the same, we can change the storage details of the application without affecting the user.

Queries in a DBMS

- Why do we need a DBMS? **to answer queries**
- A DBMS provides a specialized language, called **query language** to ask questions to the DBMS

Transaction Management

- What happens when a DBMS has more than one concurrent user?
- When several users access (and possibly modify) a database concurrently, the DBMS must order their request carefully to **avoid conflicts**
- DBMS should also protect users from system failures:
  - it should make sure data is not lost
  - it should deal with crashes *in the middle* of a **transaction**
- **Transaction**: a transaction is a conceptually indivisible group of operations that a user wants to perform (for example, getting transferring money from one account to another)

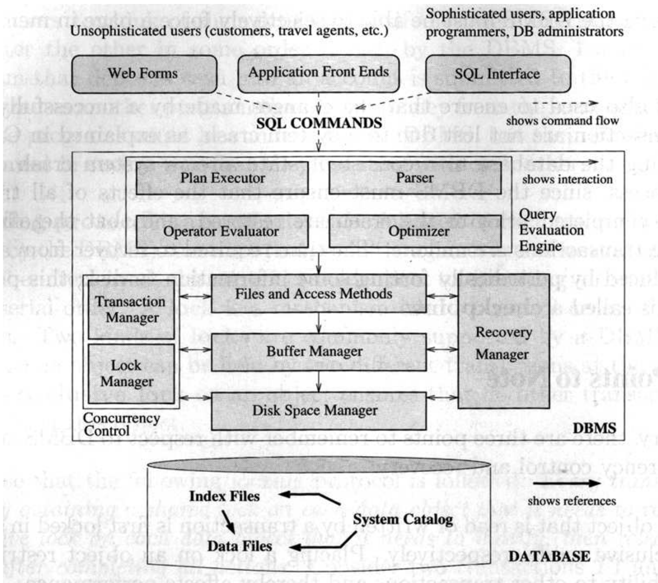
Concurrent Execution of Transaction

- An important task of a DBMS is to schedule concurrent accesses in a way that every user can ignore the fact that others are accessing the data at the same time
- A DBMS allows user to think that their programs are executed in isolation
- Locking has to be implemented to allow transactions to be interleaved
  - **Shared Locks**: allow several transactions to hold (and access) an object at the same time
  - **Exclusive Locks**: only one transaction can hold the object

Incomplete Transactions and System Crashes

- What happens if a DBMS crashes in the middle of a transaction?
- When the DBMS recovers, the incomplete transaction should be undone.
- The DBMS maintains a **log** of everything it writes
- The log is created **before** the operation is done: **write-ahead log**

Structure of a DBMS



People who deal with databases

- End user (maybe through a program):
  - Wide range of skills and needs
- Programmers:
  - Usually combine code with DBMS commands
- Database administrator: responsible for:
  - Design of the conceptual and physical schemas
  - Security and Authorization
  - Data availability and failure recovery
  - Database tuning

What we are going to learn here

- How to be a programmer: effectively write code that is combined with DBMS commands
- How to be a (junior) DBMS administrator: we are going to explore some of the issues related to running a DBMS.
- And we are going to do it using a SQL DBMS (postgresql)