Fundamental DBMS Concepts

CMSC 206 – Database Management Systems

Revision History
27 Sep 2022 : Template update – Ria Borromeo
07 Sep 2019 : Main content – Thomas Laurent

Outline

Introduction

In this lecture, we will ***define*** what a Database Management System (DBMS) is. We will discuss ***what*** they are and ***why*** we use them. We will aslo see the ***abstraction levels*** offered by a DBMS and then finally see ***examples*** of modern DBMSes.

Points to Remember

- Definition of DB and DBMS
- DBMS levels of abstraction
- ACID properties
- Types of DBMS

Database (DB)

Definition : A DB is a collection of ***related*** data, facts about a particular world or process, with ***meaning***, a ***purpose*** and ***target users***.

- As you can see, this is a very broad definition…
- Data takes many forms: numbers, text, images, sounds, video, etc.
- Note that any random collection of data is not a DB.

Figure 1: Classic representation of a database: three stacked cylinders

Database (DB) Examples:

- Personal address book containing one’s contacts’ information
  - *data: name, address, phone number*
- Student records
  - *data: name, address, phone number, birthdate, grades*
- Bank records
  - *data: name, address, phone number, transactions, account balance*
- Online retailer’s catalog
  - *data: product name, description, price, photo*

Database Management System (DBMS)

Definition: A DBMS is a software system that enables users to create, maintain, and exploit a database. It allows users to:

1. **Define the DB:** specify the structure of the database, the data that will be stored, its type and constraints. This constitutes the **metadata** of our data.
2. **Construct the DB:** insert the data in the database and store it durably.
3. **Manipulate the DB:** retrieve and update data from the database.
4. **Share the DB:** allow multiple users to use the database concurrently

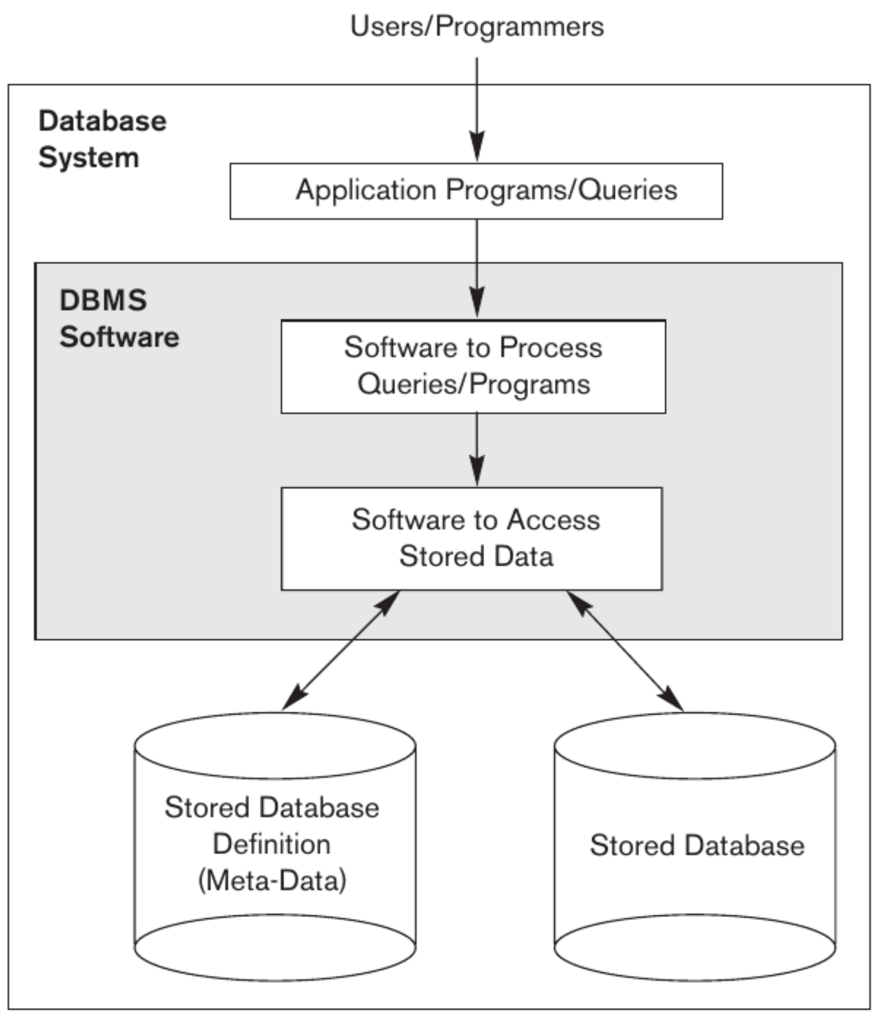
DBMS Examples:

- Microsoft Access, SQL Server, MySQL, PostgreSQL, SQLite
- Redis, Mazon DynamoDB
- MongoDB, Couchbase
- Neo4J, Datastax Enterprise Graph

Spreadsheet software (e.g. Excel, Google Sheets, LibreOffice Calc) are not DBMS!

Database System

Database System =
Database + Database
Management System



Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

Figure 2: A simplified database system[1]

[1] Figure 1.1 Fundamentals of Database Systems, Elmasri and Navathe

Storing a data with a filesystem - Example

- If a DB is just a coherent collection of data, why do we need a DBMS? Why not just keep our data in files that we format in an ad-hoc way and organise them through the directory tree and filenames?
- If we take the previous example of a university, why not have:
    - The student records system work with */school/program/year/student_name_grades.csv* files where each line would record a class taken and the grade obtained
    - The financial office's system work with */school/program/year/student_name_fees.csv* files where each line would record a transaction on the account of the student.
    - The PR system work with */school/program/year.csv* files where each line would record a student name and their contact info.

Storing a data with a filesystem - Example

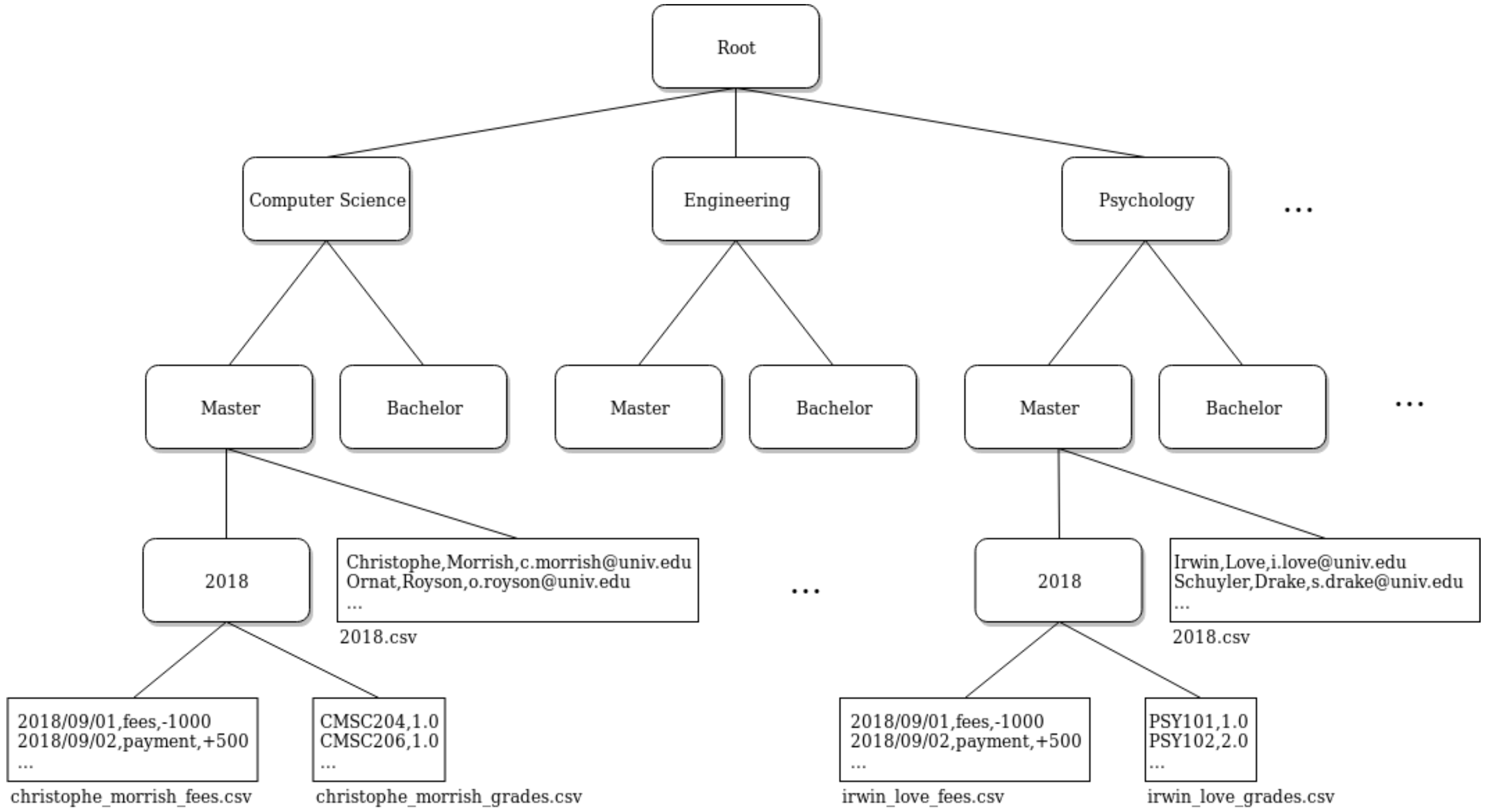


Figure 3: Example of a school managing everything through the file system

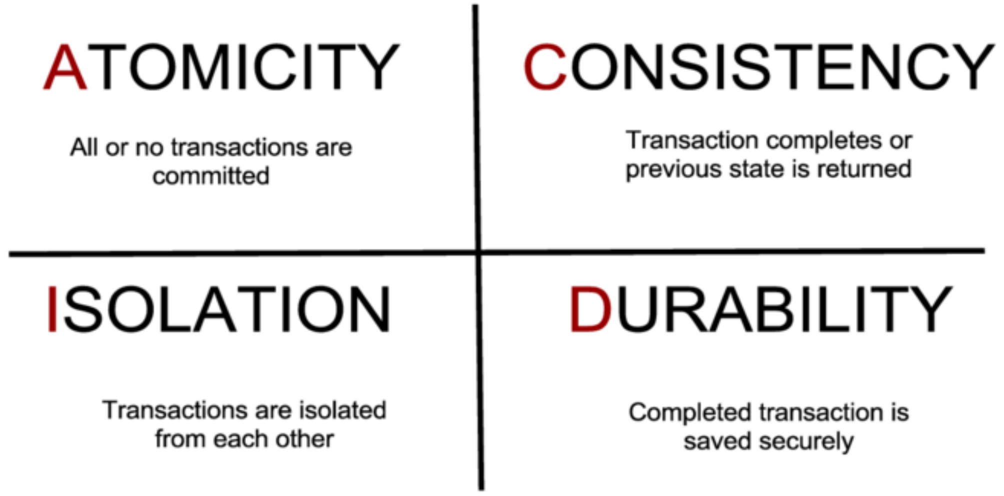
Storing a data with a filesystem - Drawbacks

This example shows several drawbacks of the filesystem approach:

- **Data redundancy**: *student_name* appeared in three (3) different places. We are wasting storage.
- **Risk of inconsistency**: what happens if one person is recorded as Thomas in the student records and Tomas in the financial system? What happens if my PC crashes while I am modifying/saving a file?
- **No concurrency control:** what happens if two people modify fees records at the same time? Will one's modification overrride someone else's?
- **Low flexibility**: the format of the data cannot be easily changed. Similarly, we can only use ad-hoc programs to process the data as there are no standard formats.

We want ACID!

What we are missing, and that DBMS provide to different degrees is the ACID properties:


ATOMICITY
All or no transactions are committed
CONSISTENCY
Transaction completes or previous state is returned
ISOLATION
Transactions are isolated from each other
DURABILITY
Completed transaction is saved securely

Atomicity: Every operation or group of operations, called transaction, should be completely executed or not be executed at all. It follows the all-or-nothing principle. For example, we cannot take money from a bank and not credit it somewhere else.

Consistency: Every transaction in the DB will bring it from a valid state to another valid state with regards to the constraints put on the data.

Isolation: If transactions are executed concurrently on the DB, they will have the same effect as if they were run in sequence and not interfere with one another.

Durability: Once a transaction is committed, it will stay committed, even after a system failure.

Abstraction through the DBMS

- To allow us to achieve ACID properties, the DBMS provides different abstractions. There are three levels of abstraction offered by the DBMS, which should work independently, which means we could change the way we implement one level without impacting the others.

Levels of Abstraction

- **Physical level:** how the data is stored on disk. Here, we consider file formats, encoding, etc. This is usually handled 100% by the DBMS.
- **Logical/Conceptual level:** We ask, *what data is being stored? What is the type of the data? How are different data items related?*
  - (e.g. the ”lecturer” item in the record for a class must be a ”person” record)? Who can access what part of the data?
- **View level:** The details of the data types are hidden. Different views are created depending on your accessible data. Your boss may view your salary but you may not view his for example.

Levels of Abstraction

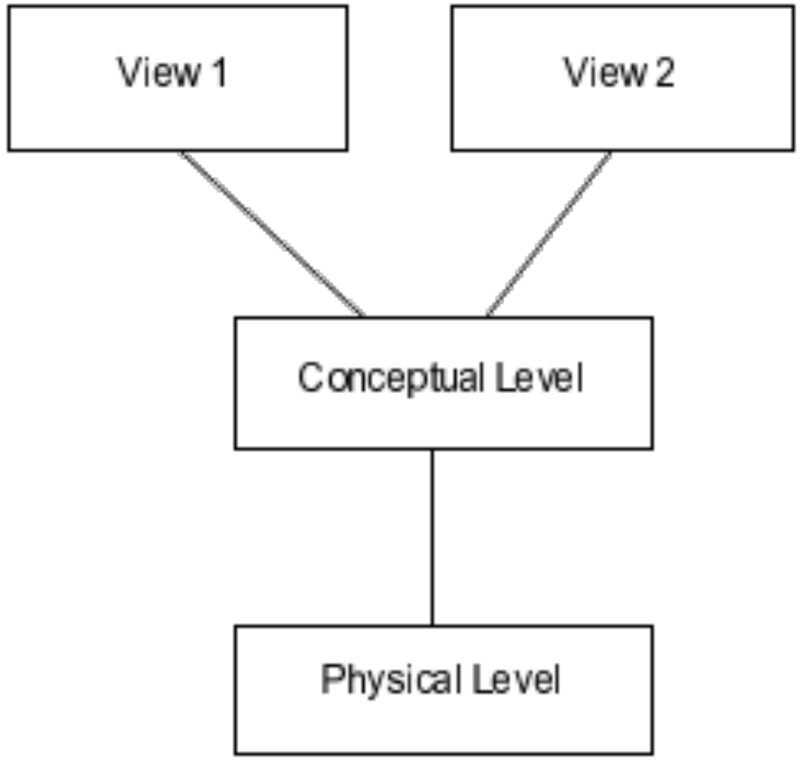


Figure 4: The three levels of abstraction

- Each level is a higher level abstraction of the data. Each level is of concern to different people. The DBMS administrator is interested in how the files are saved to disk. A programmer building something on top of the DB is interested in the data types but not in how the data is actually stored. The end user is just interested in browsing the data.

Types of DBMS

- **Relational (RDBMS):** the focus of this course. Data is very structured into tables and columns with strong constraints on them. ACID properties are strongly enforced.
- **Object oriented (OODB):** simplifies the interaction between Object-Oriented programming languages and relational databases. Objects can be saved into and retrieved directly from the DB.
- **NoSQL:** Emerged with Big Data. It is much faster than RDBMS but ACID is sometimes relaxed. e.g. Key/Value stores, column stores, document stores, graph databases, etc.