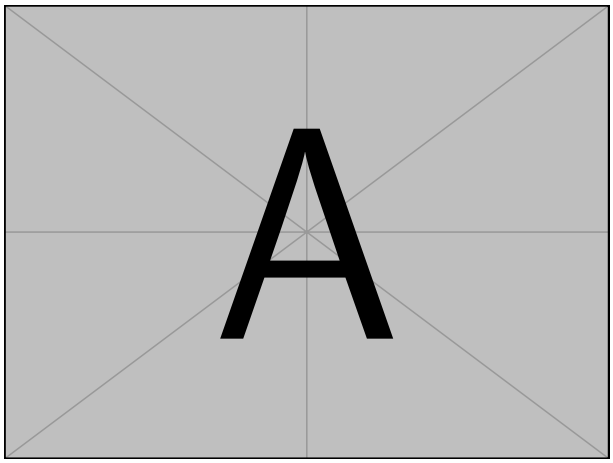
DBMS Notes: Unit 1 Introduction to Database Systems



Course Code: CS502
Generated: October 8, 2025

Contents

1 Core Concepts: Data, Database, and DBMS

Data: Raw, unorganized facts and figures. For example, "Alex", "25", "London".

Database (DB): A structured, organized collection of logically related data. It models a real-world system. For example, a table of students with columns for name, age, and city.

Database Management System (DBMS): Software that acts as an intermediary between the user and the database. It allows users to create, read, update, and delete data (**CRUD** operations) in a controlled and efficient manner. Examples: MySQL, PostgreSQL, Oracle, SQL Server.

1.1 File System vs. DBMS

Before DBMS, data was often stored in flat files (like CSV or plain text). This approach had major problems that DBMS aims to solve.

Feature	Traditional File System	DBMS
Data Redundancy	High (data is duplicated across files)	Low (controlled redundancy)
Data Inconsistency	High (updates may not propagate everywhere)	Low (updates are atomic)
Data Access	Difficult and requires custom programs	Easy using a standard query language (SQL)
Security	Poor (relies on OS permissions)	Fine-grained (access control per user/table)
Concurrent Access	Prone to errors (e.g., the "lost update" problem)	Managed via locking and transactions
Data Integrity	Must be enforced by application code	Enforced by the DBMS via constraints

2 Levels of Data Abstraction

A major advantage of a **DBMS**is **data independence**—the ability to change the schema at one level without affecting higher levels. This is achieved through three levels of abstraction.

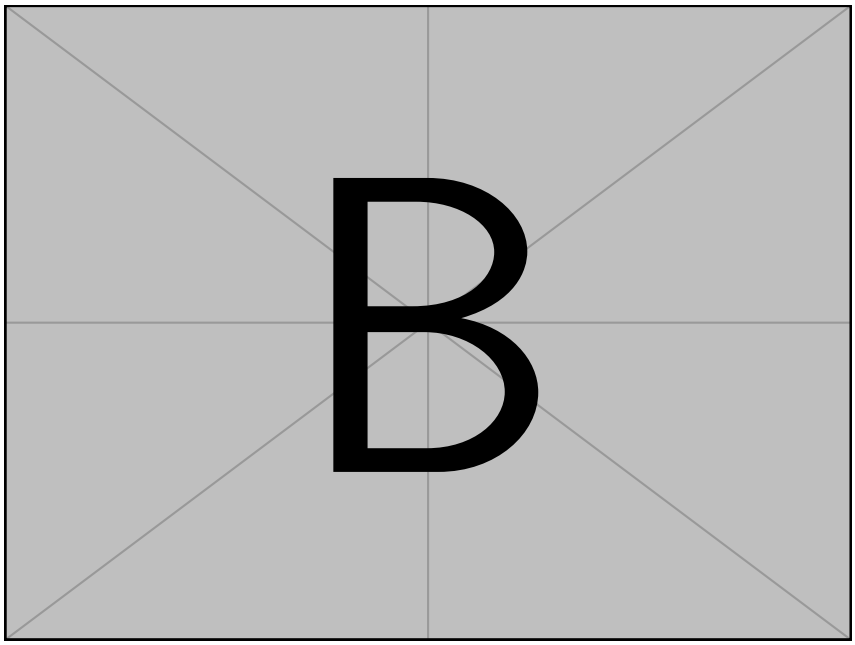


Diagram showing View Level on top, Logical Level in middle, Physical Level at bottom.

1. **Physical Level (Internal):** The lowest level. It describes **how** the data is physically stored on disk (e.g., file organization, indexing, data structures). This is the most complex level, handled by the **DBMS**.

2. **Logical Level (Conceptual):** The middle level. It describes **what** data is stored in the database and the relationships between that data. This is the level where database administrators (DBAs) work, defining tables and constraints.

3. **View Level (External):** The highest level. It describes only a part of the entire database, customized for a particular group of users. It hides unnecessary details and provides a level of security. For example, a student might only see their own grades, not the entire 'Grades' table.

3 Database Users and Architecture

3.1 Users of a DBMS

- **Database Administrator (DBA):** The "super-user" responsible for managing the entire **DBMS**, including security, backups, performance tuning, and granting access.

- **Application Programmers:** Developers who write applications that interact with the database. They access the database through APIs (e.g., JDBC, ODBC).

- **Naive Users:** End-users who interact with the database through well-defined user interfaces (e.g., a bank teller using a banking application). They are unaware of the underlying **DBMS**.

- **Sophisticated Users:** Users like data analysts who interact directly with the database using a query language like SQL.

3.2 DBMS Architecture

This describes how the components of a database system are organized.

- **2-Tier Architecture:** A client-server model where the application logic is either on the client side or the server side. A client directly connects to the database server. Simple to set up but less scalable.
- **3-Tier Architecture:** The most common architecture for web applications. It separates the system into three logical tiers:
  1. **Presentation Tier:** The user interface (e.g., a web browser).
  2. **Application Tier:** The business logic (e.g., a web server running Python/Java). This tier communicates with the database.
  3. **Data Tier:** The database server itself.

4 Smart Quirks & Key Insights

4.1 Data Independence is the True Superpower

The three levels of abstraction aren't just for organization; they provide **data independence**.

- **Physical Data Independence:** You can change the physical storage (e.g., switch to faster SSDs, change the file indexing method) without rewriting the application code. This is a huge operational advantage.
- **Logical Data Independence:** You can change the logical schema (e.g., add a new column to a table) without changing the external views or most application programs.

This separation of concerns is arguably the single most important benefit of a **DBMS**over a file system.

4.2 The Weirdness of NULL

In SQL, 'NULL' doesn't mean zero, an empty string, or false. It means **"a value is unknown or does not exist"**. It behaves strangely in comparisons:

- '5 = NULL' is not false, it is *unknown*.
- 'NULL = NULL' is also *unknown*.

This three-valued logic (TRUE, FALSE, UNKNOWN) is a common source of bugs for new developers. You must always check for nulls explicitly with 'IS NULL' or 'IS NOT NULL'.

4.3 Declarative vs. Procedural

Interacting with a file system is **procedural**: you write code that specifies step-by-step *how* to find and process the data. SQL, the language of DBMS, is **declarative**: you specify *what* data you want, and the **DBMS**'s query optimizer figures out the most efficient way to get it. This is a massive leap in productivity and abstraction.