Database Management Systems

A **database** is a collection of data, typically describing the activities of one or more related organizations. For example, a university database might contain information about the following:

- *Entities* such as students, faculty, courses, and classrooms.
- *Relationships* between entities, such as students' enrollment in courses, faculty teaching courses, and the use of rooms for courses.

A **database management system**, or **DBMS**, is software designed to assist in maintaining and utilizing large collections of data, and the need for such systems, as well as their use, is growing rapidly. The alternative to using a DBMS is to use ad hoc approaches that do not carry over from one application to another; for example, to store the data in files and write application-specific code to manage it. The use of a DBMS has several important advantages

A **database management system**, or **DBMS**, is software designed to assist in maintaining and utilizing large collections of data, and the need for such systems, as well as their use, is growing rapidly. The alternative to using a DBMS is to use ad hoc approaches that do not carry over from one application to another; for example, to store the data in files and write application-specific code to manage it. The use of a DBMS has several important advantages

Database Applications:

. Banking: transactions
. Airlines: reservations, schedules
. Universities: registration, grades
. Online retailers: order tracking, customized recommendations
. Manufacturing: production, inventory, orders, supply chain
. Human resources: employee records, salaries, tax deductions

Why file systems over DBMS?

To understand the need for a DBMS, let us consider a motivating scenario:

- A company has a large collection (say, 500 GB) of data on employees, departments, products, sales, and so on.
- This data is accessed concurrently by several employees.
- Questions about the data must be answered quickly, changes made to the data by different users must be applied consistently.
- Access to certain parts of the data (e.g., salaries) must be restricted.

We can try to deal with this data management problem by storing the data in a collection of operating system files. This approach has many drawbacks, including the following:

1. We probably do not have 500 GB of main memory to hold all the data. We must therefore store data in a storage device such as a disk and bring relevant parts into main memory for processing as needed.

2. Even if we have 500 GB of main memory, on computer systems with 32-bit addressing, we cannot refer directly to more than about 4 GB of data! We have to program some method of identifying all data items.

3. We have to write special programs to answer each question that users may want to ask about the data. These programs are likely to be complex because of the large volume of data to be searched.

3. We must protect the data from inconsistent changes made by different users accessing the data concurrently. If programs that access the data are written with such concurrent access in mind, this adds greatly to their complexity.

4. We must ensure that data is restored to a consistent state if the system crashes while changes are being made.

5. Operating systems provide only a password mechanism for security. This is not sufficiently flexible to enforce security policies in which different users have permission to access different subsets of the data.

3. We must protect the data from inconsistent changes made by different users accessing the data concurrently. If programs that access the data are written with such concurrent access in mind, this adds greatly to their complexity.

4. We must ensure that data is restored to a consistent state if the system crashes while changes are being made.

5. Operating systems provide only a password mechanism for security. This is not sufficiently flexible to enforce security policies in which different users have permission to access different subsets of the data.

A DBMS is a piece of software that is designed to make the preceding tasks easier.

By storing data in a DBMS, rather than as a collection of operating system files, we can use the DBMS's features to manage the data in a robust and efficient manner.

As the volume of data and the number of users grow hundreds of gigabytes of data and thousands of users are common in current corporate databases DBMS support becomes indispensable.

ADVANTAGES OF A DBMS

Using a DBMS to manage data has many advantages:

Data independence: Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

ADVANTAGES OF A DBMS

Using a DBMS to manage data has many advantages:

Data independence: Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

Efficient data access: A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.

ADVANTAGES OF A DBMS

Using a DBMS to manage data has many advantages:

Data independence: Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

Efficient data access: A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.

Data integrity and security: If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce *access controls* that govern what data is visible to different classes of users.

Data administration: When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and for fine-tuning the storage of the data to make retrieval efficient.

Data administration: When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and for fine-tuning the storage of the data to make retrieval efficient.

Concurrent access and crash recovery: A DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Further, the DBMS protects users from the effects of system failures.

Data administration: When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and for fine-tuning the storage of the data to make retrieval efficient.

Concurrent access and crash recovery: A DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Further, the DBMS protects users from the effects of system failures.

Reduced application development time: Clearly, the DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick development of applications.

More about DBMS

One fundamental characteristic of the database approach is : provides some level of data abstraction.

Data abstraction refers to the process of hiding the complex implementation details of a database from the users and exposing only the essential features. This helps in simplifying user interaction with the database.

A **data model** is a conceptual tool that helps in structuring and organizing the data in a database. It defines how data is connected and how it can be processed and stored.

Relational Data Model:

• **Structure:** Organizes data into tables (relations) with rows and columns. Each table represents an entity, and rows represent records.

• **Example:** Customer and order tables in an e-commerce database.

• **Key Feature:** Uses Structured Query Language (SQL) for data manipulation.

A semantic data model is a more abstract, high-level data model. eg. **entity-relationship (ER) model .**

A widely used semantic data model called the **entity-relationship (ER) model** allows us to pictorially denote entities and the relationships among them.

Entity-Relationship Model (ER Model):

• **Structure:** Uses entities, attributes, and relationships to represent data and its relationships visually.

• **Example:** ER diagram for a university database showing students, courses, and enrollments.

• **Key Feature:** Often used in the design phase of database development

The Relational Model

The relational model in a Database Management System (DBMS) is a framework for organizing and managing data using a structure called tables (or relations).

The central data description construct in this model is a **relation**, which can be thought of as a set of **records**.

A description of data in terms of a data model is called a **schema**. It acts as a blueprint for how the data is organized and how the relationships among data elements are managed.

The Relational Model

In the relational model, the schema for a relation specifies its name, the name of each fi**eld** (or **attribute** or **column**), and the type of each field. (Gives the structure of the table) As an example, student information in a university database may be stored in a relation with the following schema:

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

sid	*name*	*login*	*age*	*gpa*
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure 1.1 An Instance of the Students Relation

Each row in the Students relation is a **record** that describes a student.

Every row follows the schema of the Students relation.

The schema can therefore be regarded as a template for describing a student.

Levels of Abstraction in a DBMS

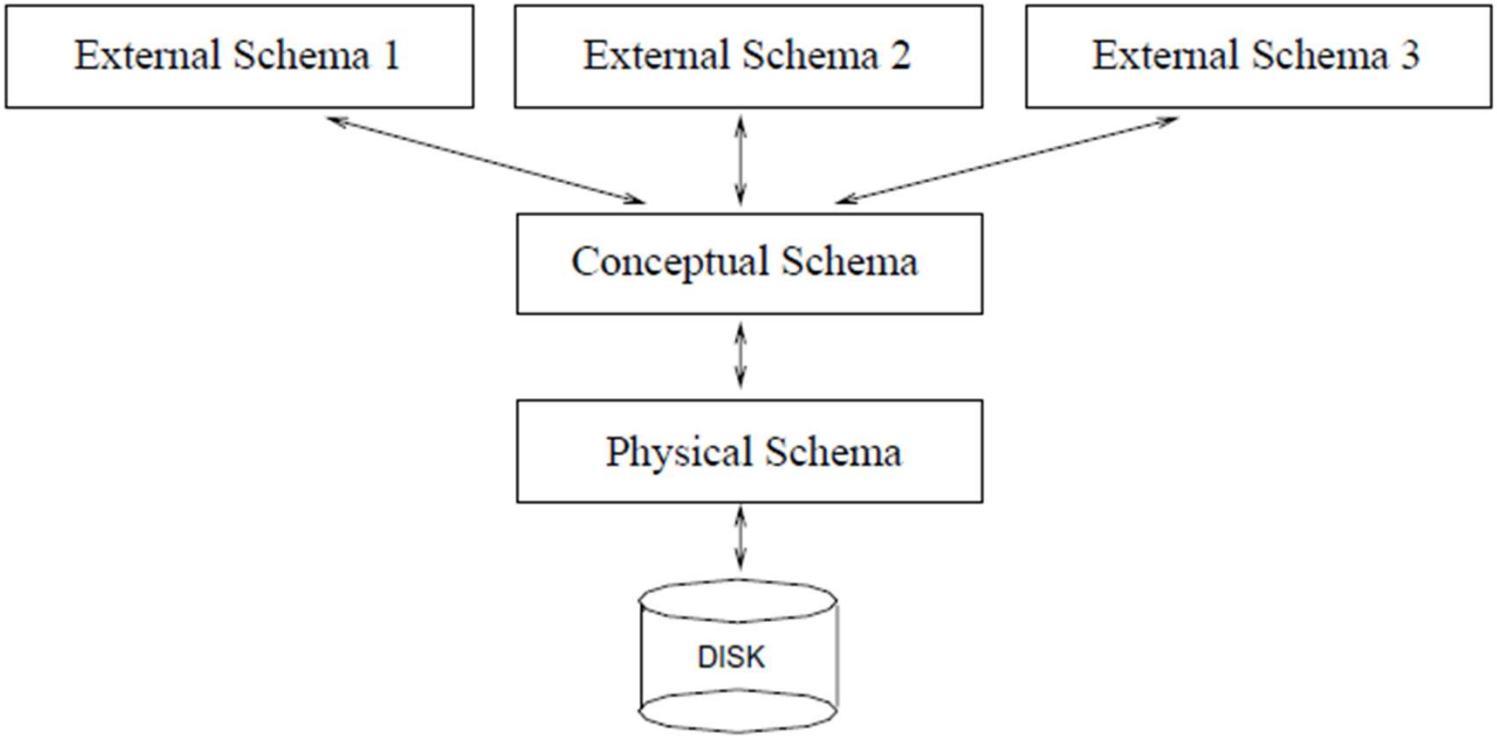
levels of abstraction help manage the complexity of data by breaking down the database architecture into manageable layers.

The data in a DBMS is described at three levels of abstraction,

- Physical
- Conceptual
- External

The database description consists of a schema at each of these three levels of abstraction: the *conceptual*, *physical*, and *external* schemas.

These levels help manage the complexity of the data and provide a clear separation between how data is stored, how it is logically structured, and how it is presented to users.


External Schema 1
External Schema 2
External Schema 3
Conceptual Schema
Physical Schema
DISK

Conceptual Schema

The **conceptual schema** (sometimes called the **logical schema**) describes the stored data in terms of the data model.

In a relational DBMS, the conceptual schema describes all relations that are stored in the database.

If we take a sample university database, these relations contain information about *entities*, such as students and faculty, and about *relationships*, such as students' enrollment in courses.

An example in the context of a university database.

```
Students(sid: string,name: string, login: string, age: integer, gpa: real)
Faculty(fid: string, fname: string, sal: real)
Courses(cid: string, cname: string, credits: integer)
Rooms(rno: integer, address: string, capacity: integer)
Enrolled(sid: string, cid: string, grade: string)
Teaches(fid: string, cid: string)
Meets In(cid: string, rno: integer, time: string)
```

Physical Schema

The **Physical schema** specifies additional storage details. Essentially, the physical schema summarizes how the relations described in the conceptual schema are actually stored on secondary storage devices such as disks and tapes.

We must decide what file organizations to use to store the relations, and create auxiliary data structures called **indexes** to speed up data retrieval operations.

External Schema

External schemas, in a Database Management System (DBMS) refers to the view or subset of the database that a particular user or application interacts with.

It provides a **tailored perspective** of the database to **different users** or groups of users.

External schemas abstract away the complexity of the entire database structure and **present a simplified** and **customized view**.

Each external schema consists of a collection of one or more **views** and relations from the conceptual schema. There will be only one physical and conceptual schema but different external schema associated with a database system.

For example, we might want to allow students to find out the names of faculty members teaching courses, as well as course enrollments not their salaries. So both faculty and student have two views.

Data Independence

A very important advantage of using a DBMS is that it offers **data independence**.

The three-schema architecture can be used to further explain the concept of **data independence.**

which can be defined as **the capacity to change the schema at one level of a database system without having to change the schema at the next higher level**.

Data Independence

A very important advantage of using a DBMS is that it offers **data independence**.

The three-schema architecture can be used to further explain the concept of **data independence.**

which can be defined as **the capacity to change the schema at one level of a database system without having to change the schema at the next higher level**.

Logical data independence is the capacity to **change the conceptual schema without having to change external schemas** or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item).

Data Independence

A very important advantage of using a DBMS is that it offers **data independence**.

The three-schema architecture can be used to further explain the concept of **data independence.**

which can be defined as **the capacity to change the schema at one level of a database system without having to change the schema at the next higher level**.

Physical data independence deals with the ability to **change the physical schema** (storage structure) **without affecting the logical schema** or application programs.

Changes to the physical schema include reorganizing storage files, changing indexing structures, or migrating to a different storage device (e.g., from hard disk to SSD).

Efficient data access: Techniques used by DBMS

Indexing:
The DBMS creates indexes on frequently searched columns, such as product IDs and customer names. Indexes allow the DBMS to quickly locate specific records without scanning the entire table.

Example:
When a customer searches for a product by its ID, the DBMS uses the index to directly access the product's data, significantly reducing the search time.

Caching:
The DBMS stores frequently accessed data in memory caches. This minimizes the need to read data from slower external storage.

Example:
Popular products' details are cached in memory. When customers view these products, the DBMS retrieves the data from the cache, providing faster response times.

Efficient data access: Techniques used by DBMS

Partitioning:
The DBMS splits large tables into smaller, more manageable pieces called partitions. This can be based on criteria such as date or region.

Example:
Transaction records are partitioned by month. When generating a sales report for a specific month, the DBMS only scans the relevant partition instead of the entire table.

Data Compression:
The DBMS compresses data to reduce storage space and improve I/O performance.

Example:
Text data, such as product descriptions, is compressed. When accessing this data, the DBMS decompresses it on-the-fly, reducing the amount of data read from the disk.

Efficient data access: Techniques used by DBMS

Query Optimization:
The DBMS optimizes SQL queries to minimize the number of disk accesses. It uses techniques like join optimization, predicate pushdown, and query rewriting.

Example:
A complex query that joins multiple tables is optimized to access only the necessary data.