## **Database Management Systems (DBMS)**

## A DBMS is a software application that allows users to create, manage, and access databases efficiently. It acts as an intermediary between users and the database, providing tools for data storage, retrieval, manipulation, and security.

### Components of a DBMS:

## **Hardware**: The physical components like storage devices (disks) and processors that run the DBMS software.

## **Software**: The DBMS software itself, which includes the database engine, query language processor, data dictionary, and utilities.

## **Data**: The actual information stored in the database, structured according to a specific model.

## **Procedures**: Guidelines and instructions for managing the database, including security measures, backup procedures, and access control rules.

## **Database Access Language (DAL)**: A specialized language like SQL (Structured Query Language) used to interact with the database and perform operations like data retrieval, insertion, and modification.

## **Users:** People who interact with the database, including database administrators (DBAs), application developers, and end-users.

### **Advantages of DBMS for Users:**

## **Data Sharing and Accessibility**: Multiple users can access and share data efficiently, eliminating data redundancy and inconsistency.

## **Data Security and Integrity:** DBMS provides features like access control, user authentication, and data encryption to ensure data security and maintain data integrity.

## **Data Recovery and Backup:** DBMS offers mechanisms for data backup and recovery in case of system failures or accidental data deletion.

## **Data Consistency:** DBMS enforces data integrity rules to ensure data accuracy and consistency across the database.

## **Improved Performance:** DBMS optimizes data storage and retrieval for efficient data access and faster query execution.

## **Scalability:** DBMS can handle growing data volumes and accommodate changes in user requirements.

## **Database Models**

## There are various database models used to structure data:

## **Flat-File Model**: Data is stored in simple text files with limited organization. It's easy to set up but lacks features for complex data relationships and can be inefficient for large datasets.

## **Hierarchical Model:** Data is organized in a tree-like structure with parent-child relationships. It allows for some data hierarchy but becomes cumbersome for complex relationships.

## **XML Model:** Data is stored in a structured format using Extensible Markup Language (XML). It's flexible and allows for complex data relationships but can be less efficient for data retrieval compared to relational models.

## **Relational Model** **Database Models: Examples, Advantages, and Disadvantages**

## Here's a breakdown of the database models you mentioned, along with examples, advantages, and disadvantages:

## **1. Flat-File Model**

## **Example:** A simple text file storing customer names and contact information.

## **Advantages:**

## Easy to set up and understand.

## No need for complex software.

## Portable across different systems.

## **Disadvantages:**

## Limited data organization and complexity.

## Data redundancy can lead to inconsistencies.

## Difficult to manage large datasets.

## Inefficient for complex data retrieval and manipulation.

## **2. Hierarchical Model**

## **Example:** A file system on your computer, where folders act as parents and files within them are children.

## **Advantages:**

## Provides a clear parent-child hierarchy for data organization.

## Good for representing one-to-many relationships.

## Easier to understand than more complex models.

## **Disadvantages:**

## Limited flexibility for complex data relationships (many-to-many).

## Difficult to add new categories or subcategories without restructuring the hierarchy.

## Data retrieval can be slow and cumbersome if traversing through multiple levels.

## **3. XML Model**

## **Example:** Storing product information on an e-commerce website using XML files.

## **Advantages:**

## Flexible and self-describing format.

## Can represent complex data structures and relationships.

## Platform-independent and easily exchanged between systems.

## **Disadvantages:**

## Can be more complex to set up and manage compared to flat files.

## Data retrieval might be less efficient than relational models for certain queries.

## Requires parsing and processing the XML structure for data access.

## **Additional Notes:**

## **Relational Model:** This is the most widely used model for complex databases. It stores data in tables with rows and columns, allowing for flexible relationships between entities and efficient data manipulation through queries (like SQL).

## **Choosing the Right Model:** The best model for your database depends on the type and complexity of your data, the relationships between data elements, and the desired functionalities. **Consider factors like scalability, performance, and ease of use when making the choice.**

## **DBMS Architecture**

## A DBMS typically has a layered architecture:

## **Internal Level**: Deals with the physical storage of data and how the DBMS manages it. Users don't interact with this level directly.

## **Conceptual Level:** Provides a logical view of the data structure, independent of physical storage details. Users see the data through a schema that defines entities, attributes, and relationships.

## **External Level:** Represents the user's view of the data. Users interact with the database at this level using a query language like SQL.

Imagine a library management system built with a DBMS. Here's how the layered architecture would work:

* **Internal Level (Physical Level):**
  + This is the "behind the scenes" level. Users don't interact with it directly.
  + Here, the DBMS physically stores data on storage devices like hard drives.
  + It manages data structures like indexes and access methods to optimize data retrieval.
  + Details include storage allocation, data formatting, and physical access paths.
* **Conceptual Level (Data Model Level):**
  + This level provides a logical view of the data, independent of physical storage details.
  + It defines the overall structure of the data using an entity-relationship model (ERM).
  + The ERM represents entities (e.g., Books, Authors, Members), their attributes (e.g., book\_title, author\_name, member\_id), and the relationships between them (e.g., borrowing relationship between Members and Books).
  + This schema acts as a blueprint for the database, defining the data organization without revealing physical storage specifics.
* **External Level (View Level):**
  + This is the user's view of the data. Users interact with the database at this level.
  + The DBMS provides a way for users to see and manipulate data without needing to know the physical storage details.
  + Users can interact through:
    - **Query Languages:** A language like SQL allows users to write queries to retrieve, insert, update, and delete data.
    - **Application Programs:** Software applications can interact with the database using APIs or embedded SQL to access and manage data.

**Example:**

* A librarian might use a library management software that interacts with the DBMS at the external level.
* The software would use SQL queries to retrieve information about books, authors, and members.
* Behind the scenes, the DBMS would translate these queries into instructions for the internal level to access and retrieve the data efficiently from physical storage.

This layered architecture provides a separation of concerns:

* The internal level handles the physical storage and retrieval of data efficiently.
* The conceptual level defines the logical structure of the data, independent of storage details.
* The external level allows users to interact with the data using a user-friendly interface or query languages.

## **Data Constraints**

## Constraints are rules enforced by the DBMS to maintain data integrity and consistency. Here are some common types:

## **Primary Key Constraint:** Uniquely identifies each row in a table. A table can only have one primary key.

## **Foreign Key Constraint:** References a primary key in another table, ensuring data consistency across related tables.

## **Data Type Constraint:** Specifies the type of data allowed in a column (e.g., integer, string, date).

## **NOT NULL Constraint:** Ensures that a column cannot have null values.

## **CHECK Constraint:** Defines a specific condition that a column value must satisfy.

## **Database Normalization**

## Normalization is a process of organizing data in a database to minimize redundancy and improve data integrity. Here are some normalization levels:

## **First Normal Form (1NF):** Eliminates repeating groups within a table by creating separate tables for them.

## **Second Normal Form (2NF):** Ensures that all non-key attributes are fully dependent on the primary key, eliminating partial dependencies.

## **Third Normal Form (3NF):** Eliminates transitive dependencies, where a non-key attribute depends on another non-key attribute that ultimately depends on the primary key.

## **Boyce-Codd Normal Form (BCNF):** A stricter form of 3NF, ensuring there are no determinant dependencies among non-key attributes.

## Normalization in Practice

## 1NF: Consider a table storing customer information along with order details. If multiple orders are listed for each customer in the same table, this violates 1NF. You can create a separate "Orders" table with a foreign key referencing the "Customers" table's primary key.

## 2NF: Imagine an "Orders" table with columns for customer ID, order ID, product details (product name, price), and quantity. The "product details" might be partially dependent on the customer (not the

## 

## **Entity-Relationship Modeling for RDBMS**

Entity-relationship modeling (ERM) is a conceptual approach to database design. It helps visualize the data elements (entities) and their relationships within a system. This visualization is represented by an Entity-Relationship Diagram (ERD).

Here's a breakdown of the key concepts:

**Entities:** Real-world objects or concepts that you want to store information about in the database. Examples include customers, products, orders, etc. In an ERD, entities are represented by rectangles.

**Attributes:** Characteristics that describe an entity. Each entity has specific attributes that provide details about it. For instance, a "Customer" entity might have attributes like customer\_id, name, address, etc. Attributes are shown as ovals connected to their respective entities.

**Relationships:** Associations between two entities. These connections show how entities interact with each other. Relationships are depicted as diamonds connecting the entities involved.

**Degree of Relationships:** The number of entities participating in a relationship. A binary relationship involves two entities (e.g., Order placed by Customer). Ternary relationships involve three (e.g., Employee works on Project), and so on.

**Cardinality of Relationships:** Specifies the number of occurrences of one entity associated with a single occurrence of another entity in a relationship. Here are the common cardinalities:

* One-to-One (1:1): One instance of entity A relates to one instance of entity B (e.g., a customer can have one phone number).
* One-to-Many (1:M): One instance of entity A relates to many instances of entity B (e.g., a customer can have many orders).
* Many-to-One (M:1): Many instances of entity A relate to one instance of entity B (e.g., many orders can be placed by one customer).
* Many-to-Many (M:N): Many instances of entity A relate to many instances of entity B (e.g., students can enroll in many courses, and a course can have many students enrolled).

**Relational Database Model:** A structured approach to organizing data in a relational database. It uses tables (relations) with rows and columns to store information. Each table represents an entity, and columns represent the entity's attributes. Relationships between entities are established through foreign keys, which reference primary keys in related tables.

**Creating an ERD based on a Scenario:**

Let's consider a scenario for a library management system. Here's a possible ERD:

* Entities: Book, Author, Member, Borrowing
* Attributes:
  + Book: book\_id, title, ISBN, publication\_year
  + Author: author\_id, name, nationality
  + Member: member\_id, name, contact\_info
  + Borrowing: borrowing\_id, book\_id (foreign key referencing Book), member\_id (foreign key referencing Member), borrow\_date, return\_date (optional)
* Relationships:
  + Book is written by Author (Many-to-Many): A book can have many authors, and an author can write many books. This is implemented using an associative entity "Book\_Author" with foreign keys to Book and Author tables.
  + Member borrows Book (One-to-Many): A member can borrow many books, but a book can only be borrowed by one member at a time (assuming single borrowing at a time).

This is a simplified example, but it demonstrates how ER modeling helps visualize and structure a database for a relational system.

**Example -**  
Imagine a library database. We want to keep track of the books and the members who borrow them.

**Entities:**

1. **Book:** This represents a book in the library.
2. **Member:** This represents a member who borrows books.

**Attributes:**

* **Book:**
  + book\_id (unique identifier for the book)
  + title (name of the book)
  + author (name of the author)
  + genre (category of the book)
* **Member:**
  + member\_id (unique identifier for the member)
  + name (member's full name)
  + contact\_info (phone number or email address)

**Relationships:**

* **Borrows:** This relationship connects books and members. A member can borrow one or more books, and a book can be borrowed by one member at a time (assuming single borrowing at a time).

**ERD Representation:**

Here's how this ERD would look:

1. Rectangles represent entities: **Book** and **Member**.
2. Ovals represent attributes within each rectangle.
3. A diamond represents the relationship: **Borrows**. Lines connect the entities to the relationship.

(borrows) ---------> Member

Book (book\_id, title, author, genre)

---------> (member\_id, name, contact\_info)

**Cardinality:**

In this example, the cardinality between Book and Borrows is **One-to-Many**. A book can be borrowed by one member at a time (one book), but a member can borrow many books (many borrows).

This is a basic example, but it demonstrates the core concepts of ER diagrams. As the complexity of your data increases, you can introduce additional entities, attributes, and relationships to model a more comprehensive database structure.

Shapes in ER Diagram-  
In an Entity-Relationship Diagram (ERD), there are three main shapes used to represent different components of the data model:

1. \*\* Rectangles:\*\* Represent **Entities**. These are the real-world objects or concepts that you want to store information about in your database. Examples include customers, products, orders, employees, departments, etc.
2. \*\* Ovals (or Ellipses):\*\* Represent **Attributes**. These are the characteristics that describe an entity and provide details about it. Each entity has specific attributes. For instance, a "Customer" entity might have attributes like customer\_id, name, address, phone number, etc. Ovals are connected to their respective entities using lines.
3. \*\* Diamonds:\*\* Represent **Relationships**. These shapes depict associations between two entities, showing how they interact with each other. Relationships connect the entities involved.

Links - <https://vertabelo.com/blog/one-to-one-relationship-in-database/#:~:text=A%20one%2Dto%2Done%20relationship%20in%20an%20ER%20diagram%20is,with%20a%20single%20straight%20line>.  
  
Example of 1 to 1 relationships

Let’s see some real-life examples of one-to-one relationships:

* **Country - capital city**: Each country has exactly one capital city. Each capital city is the capital of exactly one country.
* **Person - their fingerprints**. Each person has a unique set of fingerprints. Each set of fingerprints identifies exactly one person.
* **Email - user account**. For many websites, one email address is associated with exactly one user account and each user account is identified by its email address.
* **Spouse - spouse**: In a monogamous marriage, each person has exactly one spouse.
* **User profile - user settings**. One user has one set of user settings. One set of user settings is associated with exactly one user.

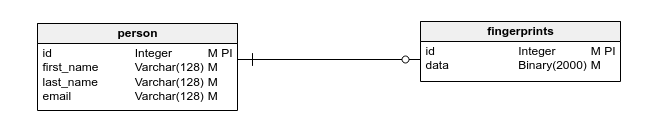
A one-to-one relationship in an ER diagram is denoted, like all relationships, with a line connecting the two entities. The “one” cardinality is denoted with a single straight line. (The “many” cardinality is denoted with a crow’s foot symbol .)

The one-to-one relationship between country and capital can be denoted like this:



The perpendicular straight lines mean “**mandatory**”. This diagram shows that it’s mandatory for a capital to have a country and it’s mandatory for a country to have a capital.

Another possibility is for one or both of the sides of the relationship to be **optional**. An optional side is denoted with an open circle. This diagram says that there is a one-to-one relationship between a person and their fingerprints. A person is mandatory (fingerprints must be assigned to a person), but fingerprints are optional (a person may have no fingerprints assigned in the database).



One to many   
  
**Real-World Scenarios for one-to-many relationships**

Let's explore a few common real-world scenarios that often involve one-to-many relationships:

1. Customer-Orders: A customer can place multiple orders, while each order belongs to a single customer.

2. Parent-Child: In a hierarchical structure, a parent can have multiple children, while each child has only one parent.

3. Project-Task: A project may have multiple tasks assigned to it, but each task is associated with only one project.

Understanding these scenarios can help you apply the concept of one-to-many relationships effectively in your database design.

Link - <https://miro.com/diagramming/er-diagram-one-to-many-relationship/>  
  
Examples of Many to Many Relationships

**E-commerce Platform:**

* Entities:
  + Products (product\_id, name, description, price, etc.)
  + Customers (customer\_id, name, email, address, etc.)
* Relationship:
  + Customers can purchase many products, and a product can be purchased by many customers.

**Social Networking Service:**

* Entities:
  + Users (user\_id, name, email, etc.)
  + Groups (group\_id, name, description, etc.)
* Relationship:
  + Users can join many groups, and a group can have many members (users).

**University Course Registration:**

* Entities:
  + Students (student\_id, name, major, etc.)
  + Courses (course\_id, name, instructor, department, etc.)
* Relationship:
  + Students can enroll in many courses, and a course can have many students enrolled.