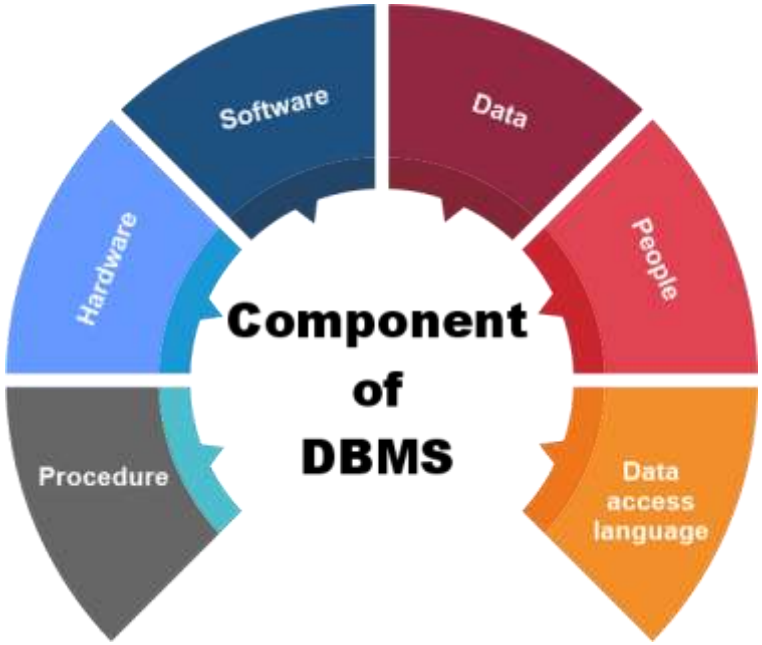
Database Management System (DBMS)

A **Database Management System (DBMS)** is a software system designed to manage databases. It provides a systematic way to create, retrieve, update, and manage data.

Components of a DBMS



1. Hardware

- Here the hardware means the physical part of the DBMS. Here the hardware includes output devices like a printer, monitor, etc., and storage devices like a hard disk.
- In DBMS, information hardware is the most important visible part. The equipment which is used for the visibility of the data is the printer, computer, scanner, etc. This equipment is used to capture the data and present the output to the user.
- With the help of hardware, the DBMS can access and update the database.

- The server can store a large amount of data, which can be shared with the help of the user's own system.
- The database can be run in any system that ranges from microcomputers to mainframe computers. And this database also provides an interface between the real worlds to the database.
- When we try to run any database software like MySQL, we can type any commands with the help of our keyboards, and RAM, ROM, and processor are part of our computer system.

2. Software

- Software is the main component of the DBMS.
- Software is defined as the collection of programs that are used to instruct the computer about its work. The software consists of a set of procedures, programs, and routines associated with the computer system's operation and performance. Also, we can say that computer software is a set of instructions that is used to instruct the computer hardware for the operation of the computers.
- The software includes so many software like network software and operating software. The database software is used to access the database, and the database application performs the task.
- This software has the ability to understand the database accessing language and then convert these languages to real database commands and then execute the database.
- This is the main component as the total database operation works on a software or application. We can also be called as database software the wrapper of the whole physical database, which provides an easy interface for the user to store, update and delete the data from the database.

- Some examples of DBMS software include MySQL, Oracle, SQL Server, dBase, FileMaker, Clipper, Foxpro, Microsoft Access, etc.

3. Data

- The term data means the collection of any raw fact stored in the database. Here the data are any type of raw material from which meaningful information is generated.
- The database can store any form of data, such as structural data, non-structural data, and logical data.
- The structured data are highly specific in the database and have a structured format. But in the case of non-structural data, it is a collection of different types of data, and these data are stored in their native format.
- We also call the database the structure of the DBMS. With the help of the database, we can create and construct the DBMS. After the creation of the database, we can create, access, and update that database.
- The main reason behind discovering the database is to create and manage the data within the database.
- Data is the most important part of the DBMS. Here the database contains the actual data and metadata. Here metadata means data about data.
- For example, when the user stores the data in a database, some data, such as the size of the data, the name of the data, and some data related to the user, are stored within the database. These data are called metadata.

4. Procedures

- The procedure is a type of general instruction or guidelines for the use of DBMS. This instruction includes how to set up the database, how to install the database, how to log in and log out of the database, how to manage the database, how to take a backup of the database, and how to generate the report of the database.
- In DBMS, with the help of procedure, we can validate the data, control the access and reduce the traffic between the server and

the clients. The DBMS can offer better performance to extensive or complex business logic when the user follows all the procedures correctly.

- The main purpose of the procedure is to guide the user during the management and operation of the database.
- The procedure of the databases is so similar to the function of the database. The major difference between the database procedure and database function is that the database function acts the same as the SQL statement. In contrast, the database procedure is invoked using the CALL statement of the DBMS.
- Database procedures can be created in two ways in enterprise architecture. These two ways are as below.
- The individual object or the default object.
- The operations in a container.

Advantages for Users

1. **Data Abstraction and Independence**

    - **What it means**: Users don’t need to know the technical details of how data is stored.
    - **Example**: Imagine using an online store. You just see the product listings and your cart, not the complex database behind it.

2. **Efficient Data Access**

    - **What it means**: The system is designed to quickly find and retrieve data.
    - **Example**: When you search for a song on a music app, it appears almost instantly because the database is optimized to find it fast.

3. **Data Integrity and Security**

    - **What it means**: Ensures that the data is accurate and secure.
    - **Example**: In a bank's database, your account balance is always accurate, and only authorized personnel can access your account information.

4. **Data Administration**

    - **What it means**: Centralized control over data management.
    - **Example**: A school's database system allows the admin office to manage all student records in one place rather than having different files scattered everywhere.

5. **Concurrent Access and Crash Recovery**

    - **What it means**: Many users can use the database at the same time, and the system can recover if something goes wrong.
    - **Example**: Multiple people can update their profiles on a social media platform simultaneously, and if the site crashes, your data won't be lost.

6. **Reduced Application Development Time**

    - **What it means**: Easier and faster to develop applications because the DBMS handles many complex tasks.
    - **Example**: When developers build a new feature for an app, they can focus on the feature itself without worrying about how the data is managed behind the scenes.

These advantages make a DBMS a powerful tool for efficiently and securely managing data, which is crucial for any modern application or system.

Database Models

Flat-File Database

A **Flat-File Database** is the simplest kind of database, where all the data is stored in a single table, often in a plain text file.

Features

1. **Single Table or Plain Text File**

- **What it means**: All the data is kept in one place, like a single table in a spreadsheet or a single file like a CSV (Comma-Separated Values) file.
- **Example**: Think of an Excel spreadsheet where you have a list of all your contacts. Each row (or line) represents one contact, and each column holds information like name, phone number, and email.

2. **One Record per Line**
    - **What it means**: Each line or row in the file represents a complete record.
    - **Example**: In your contacts spreadsheet, each row contains all the information about one person, like "Vijay, 9890897697, vijay@gmail.com".

Characteristics

1. **Simple Structure**
    - **What it means**: Easy to understand and work with because everything is in one place.
    - **Example**: Just like your spreadsheet, you can quickly see all your contacts in one view.
2. **Easy to Create and Use**
    - **What it means**: No special software or skills needed to create or manage it.
    - **Example**: You can create and edit your contacts list using common tools like Excel or Notepad.
3. **Not Suitable for Complex Data Relationships**
    - **What it means**: Hard to manage relationships between different types of data.
    - **Example**: If you want to keep track of contacts' addresses separately, it becomes tricky because you can't easily link the contacts to their addresses in a flat-file format.
4. **Lacks Data Integrity Mechanisms**
    - **What it means**: There are no built-in checks to ensure the data is correct and consistent.
    - **Example**: In your contacts spreadsheet, nothing stops you from entering a phone number in the email column by

mistake. There's no automatic way to ensure data is entered correctly.

Why Use a Flat-File Database?

A flat-file database is useful for simple tasks where you don't need complex data relationships or high data integrity. It's great for personal projects or small applications where simplicity and ease of use are more important than the ability to handle complex data and relationships.

Summary

Imagine you have a list of your favorite movies. You can keep this list in a notebook (flat-file database) where each page has one movie's details (one record per line). It's straightforward and easy to use, but if you start adding more complex information, like actors and their other movies, it gets messy because the notebook can't handle these relationships well. Also, there's no way to automatically check if you accidentally write the wrong release year or misspell a movie title, just like there's no automatic data checking in a flat-file database.

Hierarchical Database

A **Hierarchical Database** is a type of database that organizes data in a tree-like structure, where each record (or node) has a single parent but can have multiple children.

Features

1. **Tree-like Structure**

    - **What it means**: Data is arranged in a hierarchy that looks like a tree. Each piece of data (node) is linked to a single higher-level node (parent) and can have multiple lower-level nodes (children).
    - **Example**: Imagine a family tree. Your grandparents are at the top, your parents are below them, and you and your siblings are below your parents.

2. **Single Parent per Record**

    - **What it means**: Each record has one parent, similar to how in a family tree each person has one set of parents.
    - **Example**: In an organization chart, an employee reports to one manager, so the employee record has one parent (the manager).

Characteristics

1. **Fast Access and Update Operations**

    - **What it means**: Retrieving and updating data is quick because the structure is straightforward and navigable.
    - **Example**: In an organization chart, finding all employees under a specific manager is fast because you can easily follow the tree structure from the manager down.

2. **Complex Parent-Child Relationships**

    - **What it means**: The tree structure can handle intricate and detailed relationships between data points.
    - **Example**: In a university database, a department can have multiple courses, each course can have multiple students, and so on. This hierarchical structure can effectively manage these layers of relationships.

3. **Difficulty in Managing Many-to-Many Relationships**

    - **What it means**: The hierarchical model struggles with relationships where a record needs to have multiple parents or be associated with multiple records at the same level.
    - **Example**: If a student can enroll in multiple courses and each course can have multiple students, representing this in a hierarchical database becomes challenging. You might have to duplicate data or use complex workarounds.

Why Use a Hierarchical Database?

Hierarchical databases are particularly useful when dealing with data that has a clear, hierarchical relationship. They are efficient for scenarios where the data access patterns follow the hierarchy, such as organizational charts, file systems, and parts catalogues.

Summary

Imagine a hierarchical database as an organization chart at a company. The CEO is at the top (root), below the CEO are managers (parents), and below each manager are their team members (children). This structure makes it easy to see who reports to whom and find information quickly. However, if you try to show relationships that don't fit this tree-like structure, like an employee who works on multiple projects for different managers, it gets complicated. The hierarchical database is great for clear and straightforward parent-child relationships but struggles with more complex, many-to-many connections.

XML Database

An **XML Database** is a type of database that uses XML (eXtensible Markup Language) to store data. XML is a flexible, text-based format that allows for the representation of complex and nested data structures.

Features

1. **Uses XML Format to Store Data**
    - **What it means**: Data is stored in XML format, which uses tags to define elements and structure.
    - **Example**: Imagine writing a list of books where each book has a title, author, and publication year. In XML, it might look like this:

```
<books>
  <book>
    <title>Harry Potter</title>
    <author>J.K. Rowling</author>
    <year>1997</year>
```

```
  </book>
  <book>
    <title>The Hobbit</title>
    <author>J.R.R. Tolkien</author>
    <year>1937</year>
  </book>
</books>
```

2. **Allows Storage of Complex and Nested Data Structures**
    - **What it means**: You can represent data with multiple levels and complex relationships easily.
    - **Example**: In the above XML, each <book> element can contain other elements like <title>, <author>, and <year>, which shows how data can be nested.

Characteristics

1. **Flexible Schema**
    - **What it means**: The structure of the data is not fixed and can change easily.
    - **Example**: You can add new elements to your XML data without needing to redesign the entire database. If you want to add a <publisher> element to the book data, you can just include it:

```
<book>
  <title>Harry Potter</title>
  <author>J.K. Rowling</author>
  <year>1997</year>
  <publisher>Bloomsbury</publisher>
</book>
```

2. **Self-Descriptive Data**
    - **What it means**: Each piece of data includes tags that describe what the data is.
    - **Example**: In the XML snippet, <title>Harry Potter</title> clearly indicates that "Harry Potter" is a title because it's enclosed in <title> tags.

3. **Platform-Independent**
    - **What it means**: XML can be used across different systems and platforms without compatibility issues.
    - **Example**: You can create an XML file on a Windows computer, and it can be read and used on a Mac or Linux system without any problem.
4. **Suitable for Web Applications and Data Interchange**
    - **What it means**: XML is widely used in web applications and for exchanging data between different systems.
    - **Example**: Many web services and APIs use XML to send and receive data. When you fetch weather data from a web service, it might come in XML format so that any system can understand and use it.

Why Use an XML Database?

XML databases are ideal when you need to store and manage data that is complex, hierarchical, and frequently exchanged between different systems. They are especially useful for web applications and services that require flexible data representation.

Summary

Think of an XML database like a well-organized notebook where each page has labeled sections. If you're keeping track of your favorite recipes, each recipe can have sections for ingredients, instructions, and notes. XML helps you store this information in a way that's easy to read and understand, with labels for each part. Plus, you can add new sections anytime without messing up the whole notebook. This makes XML databases great for storing detailed, structured information and sharing it across different devices and systems, like when you share recipes with friends who use different types of devices

Levels of a DBMS Architecture

1. **External Level**

    - **What it means**: This is the view of the database that individual users or applications see. It's tailored to their specific needs, showing only the relevant data.
    - **Example**: Think of a university database. A student might see their own grades and courses, while a professor might see the courses they teach and the grades of their students.

2. **Conceptual Level**

    - **What it means**: This is an abstract view of the entire database, combining all the data but without focusing on how it's stored. It's like a blueprint of the database structure.
    - **Example**: For the university database, this level includes all entities (students, courses, professors) and relationships (students enrolled in courses, professors teaching courses) without worrying about where or how this data is physically stored.

3. **Internal Level**

    - **What it means**: This is the physical view of the database, detailing how data is actually stored on the hardware. It deals with data structures and file systems.
    - **Example**: For the university database, this level would describe how student records are stored on disk, how indexes are used to speed up queries, and how data is backed up.

Types of Constraints

1. **Domain Constraints**

    - **What it means**: These restrict the kind of data that can be entered into a column. It ensures that the data is valid and within acceptable limits.

- **Example**: If a column is meant to store ages, a domain constraint might ensure that only numbers between 0 and 150 are allowed.

2. **Key Constraints**

   - **What it means**: These ensure that each record in a table can be uniquely identified. The primary key constraint is a common key constraint.
   - **Example**: In a table of students, the student ID might be a primary key, ensuring that each student has a unique ID.

3. **Entity Integrity Constraints**

   - **What it means**: These ensure that the primary key of a table is never null, making sure every record can be uniquely identified.
   - **Example**: In a student table, every student must have a non-null student ID.

4. **Referential Integrity Constraints**

   - **What it means**: These ensure that a foreign key value always refers to an existing primary key in another table. It maintains the logical relationships between tables.
   - **Example**: In a university database, if a student's record refers to a course ID, referential integrity ensures that the course ID actually exists in the courses table.

5. **Check Constraints**

   - **What it means**: These enforce a specific condition on the data values in a column.
   - **Example**: In a table of employees, a check constraint might ensure that the salary is always greater than 0.

Summary

Levels of a DBMS Architecture

- **External Level**: Like different views in a library catalog for students, professors, and librarians, showing only what's relevant to each.
- **Conceptual Level**: Like a master map of the entire library showing all sections and how they connect, without worrying about how books are stored on shelves.
- **Internal Level**: Like the detailed layout of how books are physically arranged on the shelves and how the catalog system is implemented.

Types of Constraints

- **Domain Constraints**: Ensure data fits expected types (e.g., age must be a number between 0 and 150).
- **Key Constraints**: Ensure every record is unique (e.g., every student has a unique ID).
- **Entity Integrity Constraints**: Ensure primary key is never missing (e.g., every student must have an ID).
- **Referential Integrity Constraints**: Ensure relationships between tables are valid (e.g., student course IDs must exist in the courses table).
- **Check Constraints**: Ensure data meets specific conditions (e.g., salary must be greater than 0).

Normalization in Database Design

Normalization is the process of organizing a database to reduce redundancy and improve data integrity. It involves dividing a database into tables and defining relationships between them according to rules designed to safeguard the data and make the database more flexible by eliminating redundancy and inconsistent dependency.

Steps of Normalization

1. First Normal Form (1NF)

Goal: Ensure that each column contains atomic (indivisible) values and that each column contains values of a single type.

Steps:

1. **Eliminate repeating groups**: Make sure each table column contains unique values.
2. **Ensure atomicity**: Each column should contain atomic values, meaning no columns should contain multiple values.

Example:
Suppose we have the following table with repeating groups:

StudentID	StudentName	Courses
1	Alice	Math, Physics
2	Bob	Chemistry

Convert it to 1NF:

StudentID	StudentName	Course
1	Alice	Math
1	Alice	Physics
2	Bob	Chemistry

2. Second Normal Form (2NF)

Goal: Ensure that the table is in 1NF and that all non-key columns are fully dependent on the primary key.

Steps:

1. **Ensure 1NF**: Start with a table that is already in 1NF.
2. **Eliminate partial dependencies**: Remove any columns that depend on part of a composite primary key to separate tables.

Example:
Starting with the 1NF table:

StudentID	StudentName	Course
1	Alice	Math
1	Alice	Physics
2	Bob	Chemistry

Identify partial dependencies. "StudentName" depends only on "StudentID", not on the combination of "StudentID" and "Course".

Separate into two tables:

- **Students Table**:

StudentID	StudentName
1	Alice
2	Bob

- **Enrollments Table**:

StudentID	Course
1	Math
1	Physics
2	Chemistry

3. Third Normal Form (3NF)

Goal: Ensure that the table is in 2NF and that all non-key columns are non-transitively dependent on the primary key.

Steps:

1. **Ensure 2NF**: Start with a table that is already in 2NF.
2. **Eliminate transitive dependencies**: Remove any columns that depend on non-key columns.

Example:
Suppose the Enrollments table also includes the instructor for each course:

StudentID	Course	Instructor
1	Math	Prof. A
1	Physics	Prof. B
2	Chemistry	Prof. C

"Course" determines "Instructor" (Course → Instructor), creating a transitive dependency.

Separate into two tables:

- **Enrollments Table**:

StudentID	Course
1	Math
1	Physics
2	Chemistry

- **Courses Table**:

Course	Instructor
Math	Prof. A
Physics	Prof. B
Chemistry	Prof. C

4. Boyce-Codd Normal Form (BCNF)

Goal: Ensure that the table is in 3NF and that every determinant is a candidate key.

Steps:

1. **Ensure 3NF**: Start with a table that is already in 3NF.
2. **Eliminate any remaining anomalies**: Make sure that for any functional dependency (A → B), A is a superkey.

Example:
Consider the Courses table in 3NF:

Course	Instructor
Math	Prof. A
Physics	Prof. B
Chemistry	Prof. C

If a course can be taught by multiple instructors, this table wouldn't be in BCNF. Assume that "Course" and "Instructor" together make a candidate key, but "Instructor" also determines "Course" (Instructor → Course).

To conform to BCNF, split the table further:

- **CourseInstructors Table**:

Course	Instructor
Math	Prof. A
Math	Prof. D
Physics	Prof. B
Chemistry	Prof. C

This ensures that all dependencies are on candidate keys and the design eliminates redundancy and maintains data integrity.

Summary

- **Normalization** is about organizing data in a database to reduce redundancy and improve data integrity.
- **1NF**: Make sure each column contains atomic values, with no repeating groups.
- **2NF**: Ensure the table is in 1NF and remove partial dependencies by creating separate tables.
- **3NF**: Ensure the table is in 2NF and remove transitive dependencies by creating additional tables.
- **BCNF**: Ensure the table is in 3NF and that every determinant is a candidate key, further refining the database structure to avoid any anomalies.

This process helps create a well-structured database that's efficient, easy to maintain, and scalable.