DBMS
Department: B.Tech. CSE
II Year – II Semester
UNIT –I

Syllabus: Database System Applications: A Historical Perspective, File Systems versus a DBMS, the Data Model, Levels of Abstraction in a DBMS, Data Independence, Structure of a DBMS Introduction to Database Design: Database Design and ER Diagrams, Entities, Attributes, and Entity Sets, Relationships and Relationship Sets, Additional Features of the ER Model, Conceptual Design With the ER Mode

A Historical Perspective:

A **Database Management System (DBMS)** is a software system used for managing and organizing data in a structured manner, ensuring that the data is easily accessible, manageable, and updatable. The history of DBMS development can be traced through several key phases, each marking a significant evolution in how data was stored, accessed, and manipulated. Here's a historical perspective on DBMS:

1. Pre-DBMS Era (Before 1960s)

Before the advent of modern database systems, data was managed using file-based systems. In this era, data was often stored in flat files, which were essentially simple, unstructured collections of records. These files were usually managed manually, and it was difficult to ensure consistency, integrity, and efficient access to data.

- **Problems with File-Based Systems:**
  - Data redundancy and inconsistency.
  - Difficulty in handling large volumes of data.
  - Lack of data integrity constraints.
  - Complex data retrieval operations.

2. Early DBMS Development (1960s)

In the 1960s, the need for more structured and efficient ways to manage data grew, especially as the number of applications needing access to large datasets increased. During this time, the first generation of DBMS was developed.

- **Charles Bachman and the Network Model (1960s):**
  - Charles Bachman developed the **Integrated Data Store (IDS)**, which was one of the first practical DBMS implementations. IDS used a **network model**, where data was stored in a series of records linked together in a graph structure. This model allowed more complex relationships between data but was still difficult to scale.
- **Hierarchical Model:**
  - Another early model, the **hierarchical model**, represented data in a tree-like structure, where each record had a parent-child relationship. IBM's **IMS (Information Management System)**, developed in the 1960s, is one of the most notable hierarchical DBMS of this time.

3. Relational DBMS (1970s)

The **relational model**, proposed by **E.F. Codd** in 1970, revolutionized DBMS development. Codd's work laid the foundation for modern database systems, providing a mathematical framework for managing data.

- **Relational Model:**
  - The relational model organized data into tables (relations), where each row represented a record and each column represented a field. This model provided a way to represent data independently of the physical storage, making it easier to manage and query large datasets.
- **SQL (Structured Query Language):**
  - SQL was developed as a standard language for querying and manipulating data in a relational database. Initially, SQL was based on the relational model proposed by Codd, and it quickly became the standard query language for relational databases.
- **Notable Systems:**
  - Early relational DBMS like **IBM's DB2** and **Oracle Database** were developed during this time and became widely adopted for enterprise applications.

4. The Growth of Relational DBMS and Commercialization (1980s-1990s)

During the 1980s and 1990s, relational DBMS became the dominant model for commercial database systems, as they offered significant advantages over earlier models.

- **Normalization and Data Integrity:**
  - Techniques such as **normalization** were developed to reduce data redundancy and ensure data integrity. By organizing data into normalized forms, relational databases could store information more efficiently and reduce the potential for data anomalies.
- **Commercial Success:**
  - Major database vendors, including **Oracle**, **IBM**, and **Microsoft**, began to dominate the market. Products like **Oracle Database**, **IBM DB2**, and **Microsoft SQL Server** became industry standards, widely used in various sectors, including banking, retail, and telecommunications.
- **Distributed Databases:**
  - As data volumes increased, the need for distributed databases arose. Distributed DBMS allowed data to be stored across multiple locations, providing improved performance, fault tolerance, and scalability.

5. Object-Oriented and Object-Relational DBMS (Late 1990s-2000s)

In the late 1990s, as object-oriented programming languages became popular, the need for databases that could handle complex data types (e.g., images, audio, video) arose. This led to the development of **Object-Oriented DBMS (OODBMS)** and **Object-Relational DBMS (ORDBMS)**.

- **Object-Oriented DBMS:**
  - Object-oriented DBMS allowed the storage of complex objects in a database. This model integrated the principles of object-oriented programming, such as encapsulation and inheritance, into database management.
- **Object-Relational DBMS:**
  - The **Object-Relational DBMS** combined features of relational and object-oriented databases. It supported both traditional relational tables and complex data types, providing more flexibility in handling diverse data.

6. NoSQL and New Database Paradigms (2000s-Present)

As the internet and cloud computing grew rapidly in the 21st century, new types of data and new usage patterns (such as unstructured data and real-time data processing) led to the development of **NoSQL** databases and other new database paradigms.

- **NoSQL Databases:**
  - NoSQL (Not Only SQL) databases were designed to handle large volumes of unstructured or semi-structured data, offering flexibility and scalability. These databases often used key-value, document, column-family, or graph models, catering to modern applications in big data, real-time analytics, and web services. Examples include **MongoDB**, **Cassandra**, and **Couchbase**.
- **New SQL:**
  - Some databases, often referred to as **NewSQL**, emerged to address the scalability and flexibility needs of modern applications while maintaining the ACID properties of traditional relational databases. Examples include **Google Spanner** and **CockroachDB**.
- **Cloud-Based Databases:**
  - The rise of cloud computing also influenced the DBMS landscape, with cloud-based database systems becoming increasingly popular. Major cloud providers such as **Amazon (AWS RDS)**, **Microsoft Azure**, and **Google Cloud** offer managed database services for various types of DBMS.

7. Current Trends and Future Directions

- **AI and Machine Learning Integration:**
  - The integration of **AI and machine learning** with databases is becoming a significant trend. Modern DBMS systems are increasingly incorporating AI tools to automate data management, optimize queries, and enhance data analytics capabilities.
- **Serverless and Autonomous Databases:**
  - Serverless and **autonomous databases** are evolving, where the database management and maintenance tasks such as scaling, patching, and tuning are handled automatically by cloud services, reducing the overhead for developers and administrators.
- **Blockchain and Distributed Ledger Databases:**
  - **Blockchain** technology, with its focus on immutability and distributed consensus, is influencing the development of new types of databases, especially in industries such as finance and supply chain management.

Conclusion

The history of DBMS reflects an on-going evolution from simple file-based systems to highly sophisticated, cloud-based, and AI-powered systems capable of managing massive amounts of structured, semi-structured, and unstructured data. The development of DBMS continues to advance, driven by the increasing demands of data volume, complexity, and speed. The future of DBMS will likely focus on further automation, increased integration with AI/ML technologies, and the continued evolution of cloud and distributed database systems.

File Systems vs. Database Management Systems (DBMS)

A **File System** and a **Database Management System (DBMS)** are both used for storing and managing data, but they differ significantly in terms of structure, functionality, and capabilities. Below is a comparison between the two:

1. Structure of Data

- **File System:**
  - A **file system** stores data in files, which are organized into directories. Each file can contain any kind of data, including text, images, or binary data, but there is no inherent structure within the file beyond its contents.
  - Files are typically flat, and the organization of the data inside them is up to the application that reads/writes the data.
- **DBMS:**
  - A **DBMS** organizes data into structured formats such as tables (in relational databases), collections (in NoSQL databases), or graphs (in graph databases).
  - The data is stored with a clear, predefined structure, and relationships between different data points are defined by the schema (e.g., tables with rows and columns in relational databases).

2. Data Integrity and Redundancy

- **File System:**
  - In a file system, **data integrity** must be managed manually by the application. There are no built-in mechanisms to ensure that the data is consistent or to prevent data redundancy.
  - **Data redundancy** can easily occur because different applications might store the same data in separate files, leading to inconsistencies.
- **DBMS:**
  - A DBMS enforces **data integrity** using various constraints (e.g., primary keys, foreign keys, unique constraints). It ensures that the data is consistent and valid according to the rules set by the schema.
  - **Normalization** techniques in relational DBMS help minimize **data redundancy**, ensuring data is stored in an efficient, non-repetitive manner.

3. Data Manipulation and Access

- **File System:**
  - Access to data in a file system is relatively low-level and typically requires writing custom code to manage and retrieve data. Data must be processed sequentially or in a custom format that the application understands.
  - **Search operations** are generally slow and inefficient, especially when data volumes increase, as the system lacks indexing and optimization features.
- **DBMS:**
  - A DBMS provides a high-level, abstracted interface for querying and manipulating data. For example, relational DBMS uses **SQL (Structured Query Language)**, which is a powerful language designed for data querying and manipulation.
  - The DBMS uses **indexes** to optimize search queries, significantly improving the performance of data retrieval operations.

4. Concurrency and Multi-user Access

- File System:
  - In a file system, managing **concurrent access** to data is challenging. If multiple users or applications try to read or write to the same file simultaneously, it can lead to conflicts or corruption unless the file system provides some form of locking.
- DBMS:
  - A DBMS is designed to handle **concurrent access** by multiple users. It uses **transaction management** techniques such as **locking**, **isolation levels**, and **ACID (Atomicity, Consistency, Isolation, Durability)** properties to ensure that data remains consistent even when multiple users access it simultaneously.

5. Data Security

- File System:
  - File systems often provide basic file-level security, such as setting permissions to restrict access to specific files or directories. However, this security is limited and doesn't extend to data-level security within the files.
- DBMS:
  - A DBMS provides more advanced security features, including **user roles**, **permissions**, **data encryption**, and **audit logging**, allowing for fine-grained control over who can access, modify, and view specific data.

6. Backup and Recovery

- File System:
  - In a file system, backup and recovery mechanisms must be implemented at the file level. If a file is lost or corrupted, the entire file needs to be restored from backups, and the application must handle recovery.
- DBMS:
  - A DBMS offers built-in **backup and recovery** mechanisms. It can perform **incremental backups**, restore individual records or tables, and even support automatic recovery from crashes while maintaining data integrity.

7. Data Relationships

- File System:
  - In a file system, there are no built-in mechanisms to handle **relationships** between different pieces of data. If an application needs to create relationships between files, it has to implement this logic itself.
- DBMS:
  - A DBMS is designed to handle **data relationships** explicitly. In a relational DBMS, relationships are established using foreign keys, while other DBMS types (e.g., graph DBMS) have mechanisms to define complex relationships between data entities.

8. Scalability

- File System:
  - As data grows in a file system, managing large files and efficiently retrieving them becomes more difficult. File systems generally don't provide advanced scalability features for large datasets or complex queries.
- DBMS:
  - A DBMS is designed for **scalability**, capable of handling large amounts of data, and can be optimized for performance through indexing, partitioning, and clustering. Distributed databases can also scale horizontally by spreading data across multiple servers.

9. Use Cases

- File System:
  - File systems are suitable for situations where data is simple, unstructured, or does not require complex querying. Examples include storing documents, images, system files, and logs.
- DBMS:
  - A DBMS is ideal for applications where data is structured, requires complex querying, and needs to be accessed by multiple users simultaneously. Examples include online transaction systems, customer relationship management (CRM) systems, and enterprise resource planning (ERP) systems.

The Data Model of DBMS:

The **data model** in a **Database Management System (DBMS)** refers to the conceptual framework that defines the structure, organization, and constraints of data within the database. It defines how data is stored, represented, and manipulated. There are several types of data models used in DBMSs, and each data model has its own way of organizing and interacting with data. Below is an overview of the key types of data models used in DBMSs:

1. Hierarchical Data Model

- **Overview:** The hierarchical data model organizes data in a tree-like structure where each record has a single parent (except the root) and can have multiple children. It resembles a hierarchy, where the data is represented as nodes connected by links.
- **Structure:**
  - **Parent-Child Relationship:** Each data element (or record) has one or more child records, and each child record has exactly one parent.
  - **Tree Structure:** The data model is represented as a tree with branches and nodes. The top node is the root, and all other nodes are descendants of the root.
- **Example:** A company database might have a hierarchy like:

  rust
  Copy
  Company -> Department -> Employee

- **Advantages:**
  - Simple to understand and use for certain hierarchical relationships (e.g., organization structures).
  - Efficient retrieval for hierarchical data.
- **Disadvantages:**
  - Inflexible when handling many-to-many relationships.
  - Data redundancy is possible if a child record is shared across multiple branches.

2. Network Data Model

- **Overview:** The network data model is an extension of the hierarchical model but allows more complex relationships, including many-to-many relationships. It uses a graph structure with nodes and connections (called **links**), where nodes represent records and links represent relationships between records.
- **Structure:**
  - **Graph Structure:** The data is represented as a graph where nodes (records) are connected by links (relationships).
  - **Many-to-Many Relationships:** Unlike the hierarchical model, the network model supports many-to-many relationships, where a record can have multiple parent and child records.
- **Example:** An employee could be part of multiple projects, and a project could involve multiple employees. A project-employee relationship can be represented using links in the network model.
- **Advantages:**
  - Supports more complex relationships (e.g., many-to-many).
  - More flexible and versatile than the hierarchical model.
- **Disadvantages:**
  - More complex to design and manage compared to the hierarchical model.
  - Requires more effort to maintain and traverse relationships.

3. Relational Data Model

- **Overview:** The relational data model, proposed by **E.F. Codd** in 1970, is the most widely used data model in modern DBMSs. It organizes data into tables (relations), where each table consists of rows (records) and columns (attributes). The relational model is based on set theory and first-order predicate logic.
- **Structure:**
  - **Tables (Relations):** Data is stored in tables, where each table represents an entity (e.g., customers, orders).
  - **Rows (Tuples):** Each row represents a record (e.g., an individual customer, order).
  - **Columns (Attributes):** Each column represents a data attribute or property (e.g., customer name, order date).
  - **Primary Key:** A unique identifier for each row in a table, ensuring no duplicate records.
  - **Foreign Key:** A reference to the primary key of another table, used to establish relationships between tables.
- **Example:** A customer-order database might consist of two tables:
  - **Customer Table**: CustomerID, CustomerName, Email
  - **Order Table**: OrderID, CustomerID, OrderDate The relationship between customers and orders is established via the CustomerID (a foreign key in the **Order Table**).
- **Advantages:**
  - Simple to understand and use.
  - Supports powerful querying via **SQL (Structured Query Language)**.
  - High flexibility and scalability.
  - Data integrity and consistency are easily maintained through constraints (e.g., primary and foreign keys).
- **Disadvantages:**
  - May not perform well with very large datasets in some use cases (e.g., complex joins on huge datasets).
  - More complex queries may require optimization.

4. Object-Oriented Data Model

- **Overview:** The object-oriented data model integrates **object-oriented programming (OOP)** concepts into the database system. It allows the database to store objects rather than just data values. These objects contain both **data (attributes)** and **methods (functions)** to operate on the data.
- **Structure:**
  - **Objects:** Data is stored in objects that contain both **attributes** (data) and **methods** (functions to operate on the data).
  - **Classes and Inheritance:** Objects are instances of **classes**, and they can inherit attributes and methods from other classes.
  - **Encapsulation:** Data and its related methods are bundled together within objects, following OOP principles.
- **Example:** A class Person might have attributes like Name, Age, and Address, and methods like DisplayInfo() or UpdateAge().
- **Advantages:**
  - Provides a natural representation of real-world entities.
  - Supports inheritance, polymorphism, and encapsulation.
  - Can model complex data more easily (e.g., multimedia, geographic data).
- **Disadvantages:**
  - More complex than relational models and requires specialized DBMSs.
  - May not be as mature or widely adopted as relational databases for certain use cases.

5. Entity-Relationship (ER) Model

- **Overview:** The **Entity-Relationship (ER)** model is used for high-level conceptual design of databases. It focuses on the entities (objects) in the system and the relationships between those entities. ER models are often used as blueprints to design relational databases.
- **Structure:**
  - **Entities:** Represent real-world objects or concepts (e.g., `Customer`, `Order`).
  - **Attributes:** Represent the properties of entities (e.g., `CustomerName`, `OrderDate`).
  - **Relationships:** Define how entities are related to each other (e.g., a `Customer` places an `Order`).
  - **Primary Keys:** Uniquely identify entities.
- **Example:** In an ER diagram, entities like `Customer` and `Order` are connected by a relationship like `places`, with attributes associated with each entity and relationship.
- **Advantages:**
  - Helps in designing the database at a high level, before implementation.
  - Provides an easy-to-understand graphical representation of the system.
- **Disadvantages:**
  - Not a data model for implementation but a conceptual tool; it needs to be translated into a relational model or another data model for actual DBMS implementation.

6. NoSQL Data Model

- **Overview:** The **NoSQL (Not Only SQL)** data model is used in databases designed to handle large-scale, unstructured, or semi-structured data. It provides flexibility for data that doesn't fit neatly into the rigid structure of relational databases.
- **Types of NoSQL Data Models:**
  - **Document Model:** Stores data as documents (e.g., JSON, BSON), where each document can have a flexible schema. Example: **MongoDB**.
  - **Key-Value Model:** Stores data as key-value pairs, where each key maps to a value (could be a string, list, or complex structure). Example: **Redis**.
  - **Column-Family Model:** Stores data in columns rather than rows, optimizing for write-heavy workloads. Example: **Cassandra**.
  - **Graph Model:** Stores data as nodes (entities) and edges (relationships), ideal for graph-based data like social networks. Example: **Neo4j**.
- **Advantages:**
  - High scalability and flexibility.
  - Suitable for unstructured or semi-structured data.
  - Often optimized for distributed, high-performance, and big data use cases.
- **Disadvantages:**
  - Lacks the formal structure and integrity mechanisms found in relational models.
  - May require specialized knowledge to design and maintain.

Database Management System (DBMS), levels of abstraction:

In a **Database Management System (DBMS)**, **levels of abstraction** refer to the different ways in which data is represented and organized, enabling users and applications to interact with the database without worrying about its internal complexities. These abstraction levels allow for data independence, meaning that the way data is stored can change without affecting how users interact with it.

There are typically **three levels of abstraction** in a DBMS:

1. Physical Level (Internal Level)

- **Overview:** The physical level is the lowest level of abstraction in a DBMS. It deals with the **physical storage** of data on the storage medium (e.g., hard drive, SSD). It specifies **how** the data is stored, organized, and managed at the hardware level, focusing on the efficient storage of data structures like files, blocks, indexes, and the use of storage devices.
- **Key Aspects:**
  - Defines how data is actually stored on storage devices (e.g., disk blocks, data structures like B-trees, or hashing).
  - Involves **data compression**, **indexing**, **file organization**, and **storage allocation**.
  - Focuses on performance optimization, such as disk I/O efficiency and retrieval times.
- **Example:** The physical storage might use data files stored in binary format on disk or disk blocks that represent tables or indexes. It also considers how to handle concurrency (locking, transactions) and data recovery (backup strategies).
- **Data Independence:** Changes at this level (e.g., changing storage devices or optimizing storage structures) do not affect higher levels of the DBMS.

2. Logical Level (Conceptual Level)

- **Overview:** The logical level is an abstraction that describes **what data is stored** and the **relationships among the data** without considering how the data is stored physically. This level focuses on the **logical structure** of the data, such as tables, views, entities, relationships, and constraints. It represents the **conceptual view** of the database schema.
- **Key Aspects:**
  - Describes the **entities**, **attributes**, and **relationships** within the database.
  - Specifies the **logical structure** of the database, including tables, views, indexes, and constraints (e.g., primary keys, foreign keys).
  - **Data integrity constraints**, like **unique constraints**, **foreign keys**, and **check constraints**, are defined at this level.
  - Focuses on **data models** (e.g., relational model, object-oriented model) and their logical design.
- **Example:** A **Customer** table might have the attributes CustomerID, CustomerName, Email, and PhoneNumber, with relationships between the **Customer** table and other tables like **Orders**.
- **Data Independence:** The logical structure is independent of the physical storage, meaning that changes to how data is stored (e.g., changing indexes or file formats) don't affect the logical model.

3. View Level (External Level)

- **Overview:** The view level is the highest level of abstraction. It focuses on **how users and applications interact** with the data, defining **user views** or **subschemas** of the database. This level does not show the complete database schema but rather presents customized views of the data tailored to the needs of specific users or applications.
- **Key Aspects:**
  - Defines various **views** of the database, which are specific to different user groups or applications. For example, a **Manager** might have a view of sales data, while a **Customer Support Representative** might have a view of customer information.
  - **Access control** and **permissions** are defined at this level to restrict access to sensitive data.
  - **Virtual tables** or **views** in SQL can be created to present data from one or more underlying tables, without altering the actual data storage.
- **Example:**
  - **User View:** A user might see a simplified view of a **Customer** table, showing only `CustomerName` and `Email`, while a manager might see the full table with `CustomerID`, `CustomerName`, `Email`, and `PhoneNumber`.
  - A **view** in SQL might combine data from multiple tables (e.g., joining **Customers** and **Orders**) into a single virtual table that represents all orders placed by customers.
- **Data Independence:** Changes in the logical schema (such as adding a new attribute to a table) should not require any changes to the views or user interfaces.

Diagram of the Three Levels of Abstraction

```
+---------------------+
|   View Level        |    <-- User views (virtual tables, customized data)
+---------------------+
|   Logical Level     |    <-- Database schema (tables, relationships, constraints)
+---------------------+
|   Physical Level    |    <-- Physical storage (disk, files, data structures)
+---------------------+
```

Benefits of Levels of Abstraction

1. **Data Independence:**
   - The primary benefit of these abstraction levels is **data independence**, which allows changes at one level (e.g., physical or logical) to occur without impacting other levels.
   - **Logical data independence** means changes in the logical schema (e.g., adding a column or changing relationships) don't require modifications to user views.
   - **Physical data independence** means changes to the physical storage (e.g., changing the storage medium or reorganizing data) don't affect the logical or external views.
2. **Simplified Database Design:**
   - Each level provides a **simplified view** of the database, making it easier for database designers, administrators, and users to work with the system at their respective levels of concern without being overwhelmed by complexities at other levels.
3. **Improved Security and Access Control:**
   - Different user views at the **view level** allow for **fine-grained access control**. Users can only access data that is relevant to them, without needing to understand the full underlying schema.
4. **Enhanced Flexibility:**
   - Because users and applications interact with the **view level** and **logical level**, the physical details of storage and implementation can be modified or optimized over time without impacting end users or applications.

Example: Applying the Levels of Abstraction in a University Database

1. **Physical Level:**
   - Data might be stored on a disk using an optimized indexing system. Records for students, courses, and enrollments might be stored in specific file formats or blocks for efficient querying.
2. **Logical Level:**
   - The logical schema might define tables like Student, Course, and Enrollment with fields such as StudentID, CourseID, and EnrollmentDate. It will also define relationships (e.g., students can enroll in multiple courses, and each course can have many students) and constraints (e.g., a student can't enroll in the same course twice).
3. **View Level:**
   - A view might be created for the **Registrar**, showing student names and their course enrollments. Another view might be created for a **Student**, showing only their courses and grades, while hiding sensitive information such as student IDs or financial data.

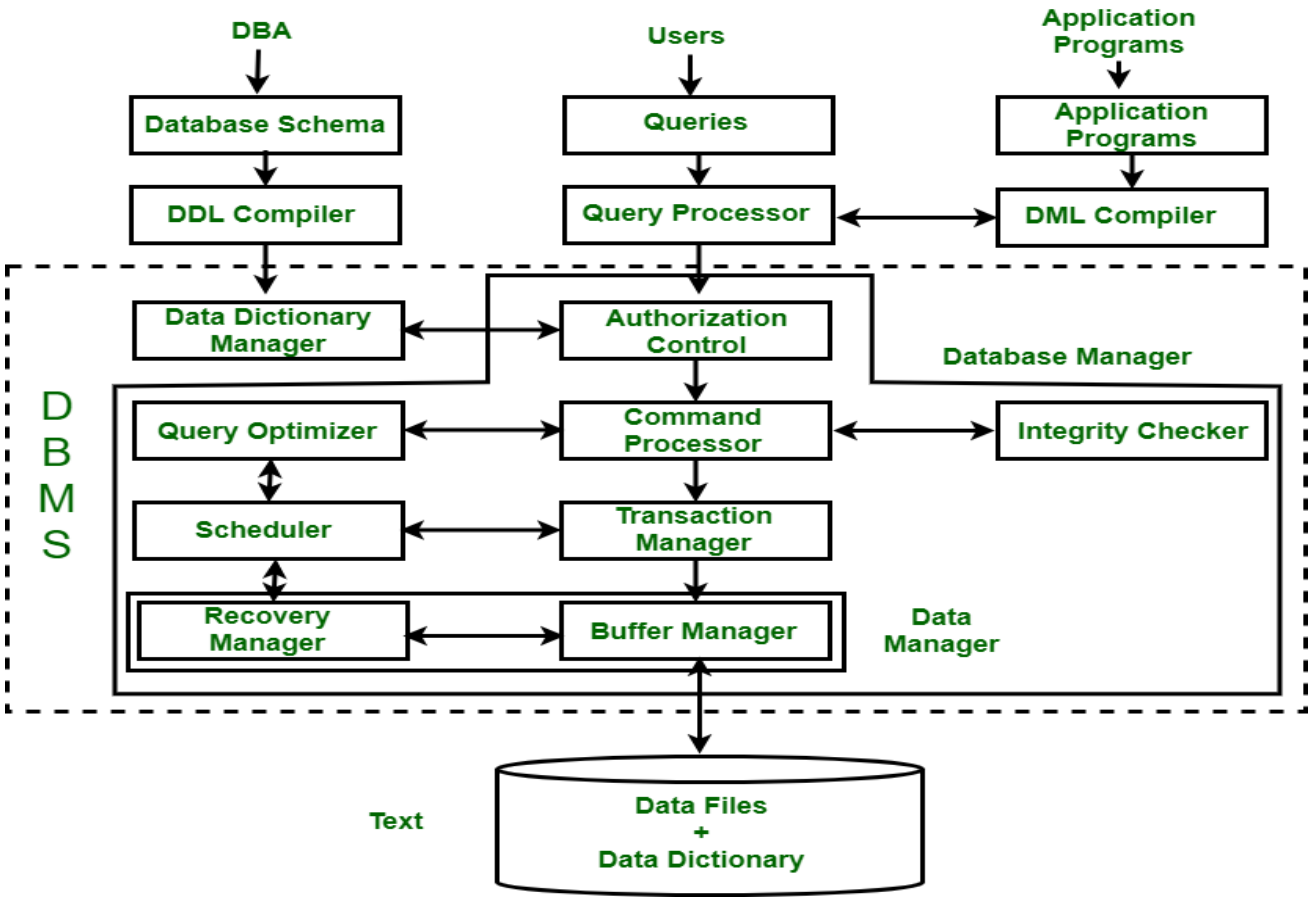
The **levels of abstraction** in a DBMS (Physical, Logical, and View) help separate concerns and provide a structured approach to database management. This multi-layered approach ensures **data independence**, simplifies database management, and improves security and flexibility in how users and applications interact with data. Each level plays an essential role in optimizing database performance, maintaining data integrity, and providing appropriate access to various stakeholders.

Structure of a DBMS

A Database Management System (DBMS) is software that allows users to define, store, maintain, and manage data in a structured and efficient manner. It acts as an intermediary between data and users, allowing disparate data from different applications to be managed. A DBMS simplifies the complexity of data processing by providing tools to organize data, ensure its integrity, and prevent unauthorized access or loss of data. In today's data-driven world, DBMS are essential for applications such as banking systems, e-commerce platforms, education, and medical systems. They not only store and manage large amounts of data, but also provide functionality that provides performance, security, and scalability for multiple users with multiple access levels.

Components of a Database System

Query Processor, Storage Manager, and Disk Storage. These are explained as following below.



1. Query Processor:

It interprets the requests (queries) received from end user via an application program into instructions. It also executes the user request which is received from the DML compiler.

Query Processor contains the following components –

- **DML Compiler:** It processes the DML statements into low level instruction (machine language), so that they can be executed.
- **DDL Interpreter:** It processes the DDL statements into a set of table containing meta data (data about data).
- **Embedded DML Pre-compiler:** It processes DML statements embedded in an application program into procedural calls.
- **Query Optimizer:** It executes the instruction generated by DML Compiler.

2. Storage Manager:

Storage Manager is a program that provides an interface between the data stored in the database and the queries received. It is also known as Database Control System. It maintains the consistency and integrity of the database by applying the constraints and executing the DCL statements. It is responsible for updating, storing, deleting, and retrieving data in the database.

It contains the following components –

- **Authorization Manager:** It ensures role-based access control, i.e,. checks whether the particular person is privileged to perform the requested operation or not.

- **Integrity Manager:** It checks the integrity constraints when the database is modified.
- **Transaction Manager:** It controls concurrent access by performing the operations in a scheduled way that it receives the transaction. Thus, it ensures that the database remains in the consistent state before and after the execution of a transaction.
- **File Manager:** It manages the file space and the data structure used to represent information in the database.
- **Buffer Manager:** It is responsible for cache memory and the transfer of data between the secondary storage and main memory.

3. Disk Storage:

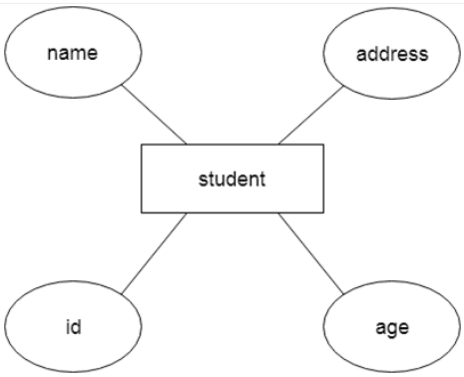
It contains the following components:

- **Data Files:** It stores the data.
- **Data Dictionary:** It contains the information about the structure of any database object. It is the repository of information that governs the metadata.
- **Indices:** It provides faster retrieval of data item.

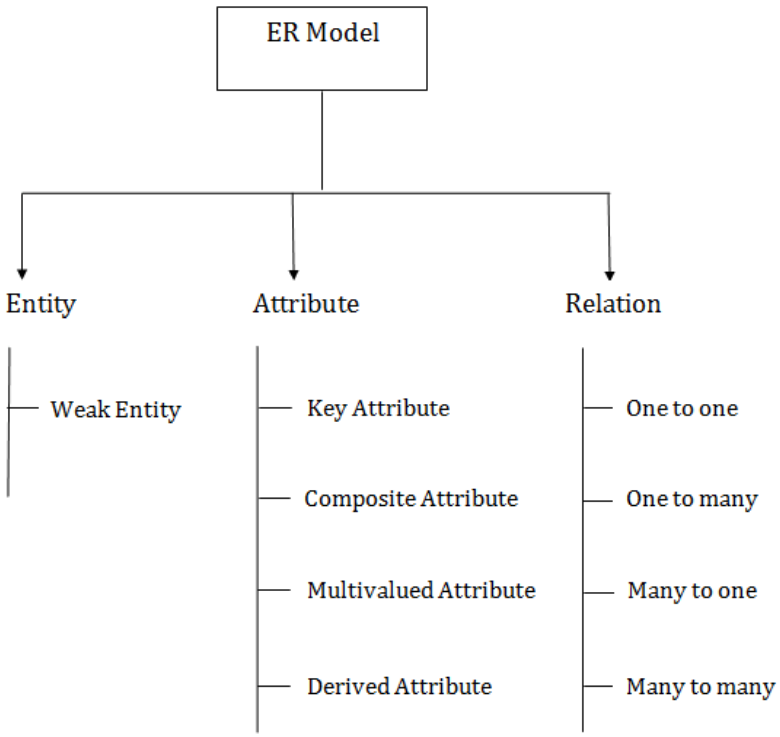
ER (Entity Relationship) Diagram in DBMS

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

For example, Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.



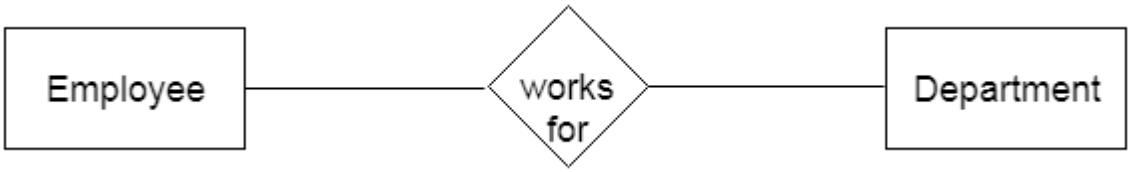
Component of ER Diagram



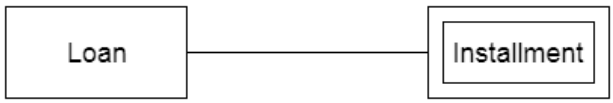
1. Entity:

An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.

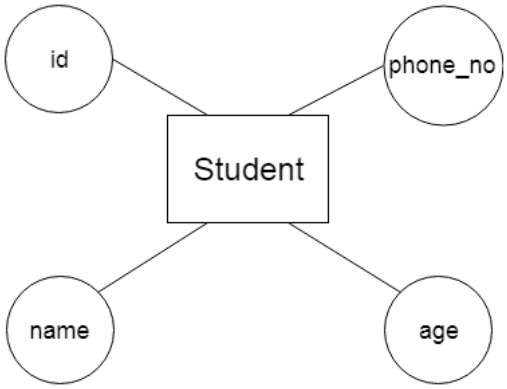


a. Weak Entity: An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



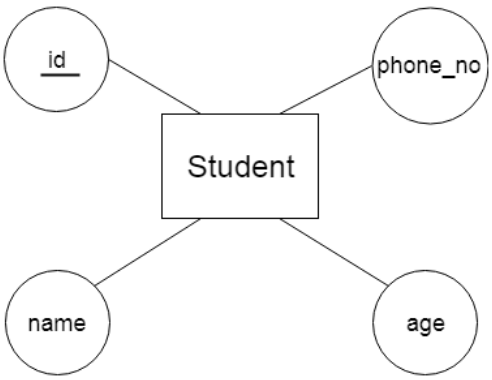
2. Attribute

The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.
For example, id, age, contact number, name, etc. can be attributes of a student.



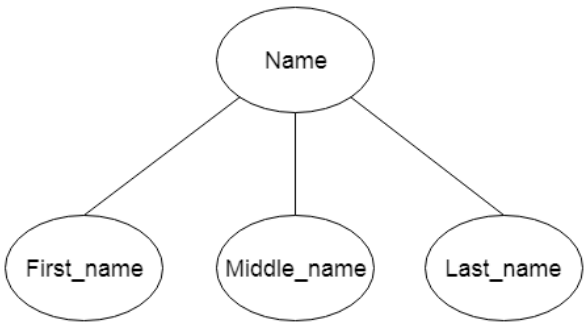
a. Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute

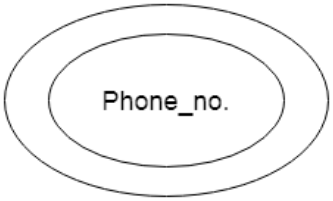
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

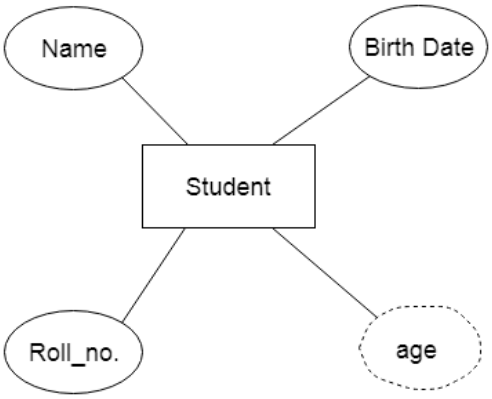
For example, a student can have more than one phone number.



d. Derived Attribute

An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.

Types of relationship are as follows:

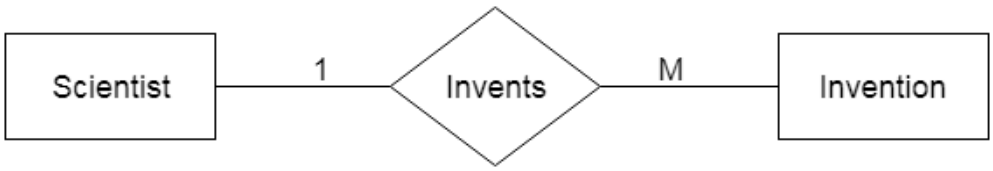
a. One-to-One Relationship

When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.
For example, A female can marry to one male, and a male can marry to one female.

b. One-to-many relationship

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

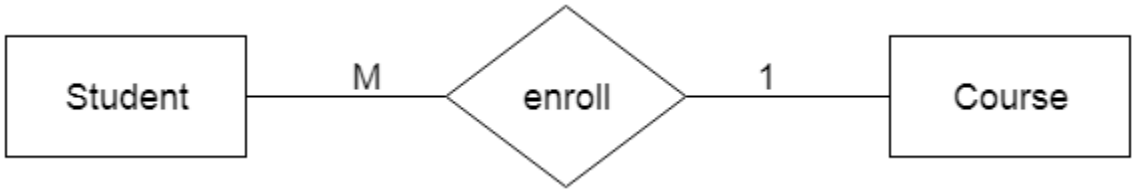
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c. Many-to-one relationship

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

For example, Student enrolls for only one course, but a course can have many students.



d. Many-to-many relationship

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.