SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT-1 DATABASE SYSTEMS CONCEPTS AND ARCHITECTURE- S11BLH41

UNIT 1 DATABASE SYSTEMS CONCEPTS AND ARCHITECTURE 12 Hrs.

History and motivation for database systems - characteristics of database approach - Actors on the scene - Workers behind the scene - Advantages of using DBMS approach - Data Models, Schema, and Instances - Three-Schema Architecture and Data Independence - The Database System Environment - Centralized and Client/Server Architectures for DBMS - Classification of DBMS. Practical: Create a database table, add constraints (primary key, unique, check, not null), insert rows, update and delete rows using SQL DDL and DML commands.

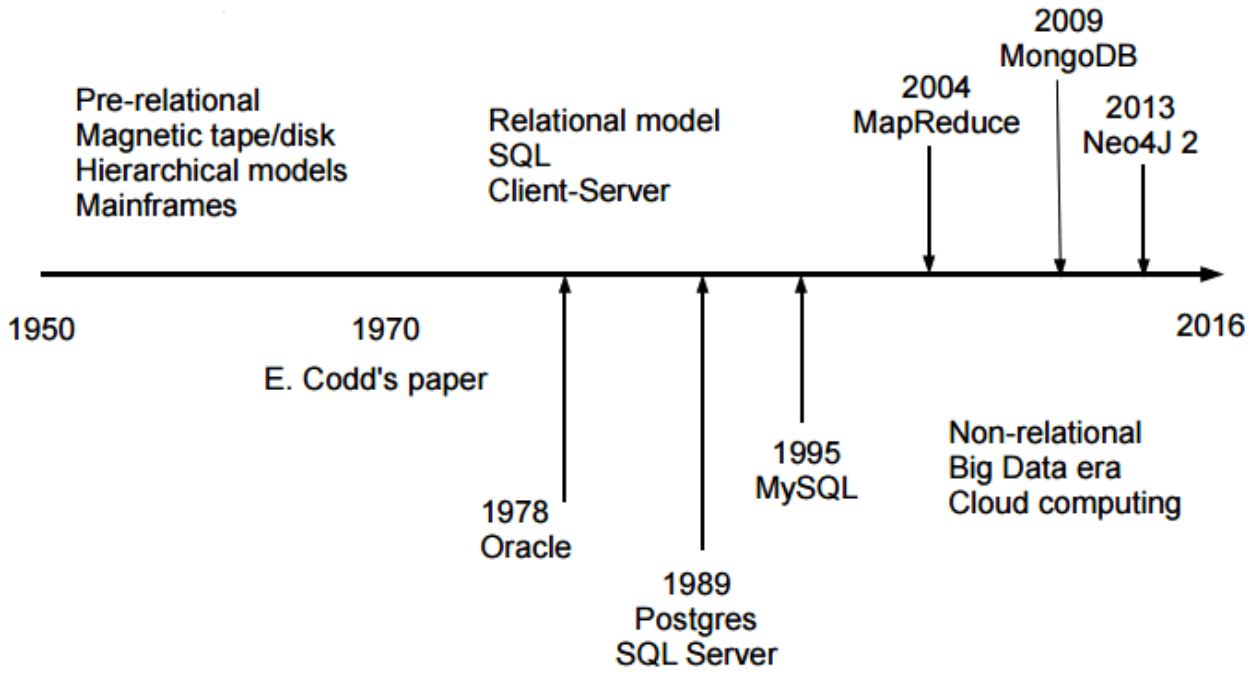
History and Motivation for Database Systems

Origins of Database Systems

Before database management systems (DBMS), organizations relied on **file-based systems** to manage data. These systems were tailored to specific applications, resulting in several inefficiencies, as highlighted below:

- **Data Redundancy and Inconsistency:** Multiple copies of the same data led to inconsistency when updates were not synchronized.
- **Limited Data Sharing:** Programs were tightly linked to their data, restricting reuse across systems.
- **Lack of Scalability:** Growing organizational data overwhelmed file systems.
- **Complex Data Retrieval:** Accessing specific information required writing complex application programs.

Evolution of Database Systems



History of DBMS

1. **1960s: The File-Based Era**
    - Early efforts focused on using magnetic tapes and sequential file systems to store and retrieve data.
    - Limitations included a lack of standard access methods and tedious maintenance.
2. **Late 1960s - Hierarchical and Network Models**
    - The **hierarchical model** (e.g., IBM's IMS) organized data in tree-like structures, improving speed for specific queries but was inflexible.
    - The **network model** (e.g., CODASYL DBTG) allowed more complex many-to-many relationships, enabling flexibility but requiring procedural query languages.
3. **1970s: The Relational Model Revolution**
    - Introduced by **E.F. Codd** in 1970, the **relational model** stored data in tables (relations), making it easier to query using Structured Query Language (SQL).
    - Major contributions include abstraction, independence, and declarative querying.
4. **1980s–2000s: Object-Oriented and Distributed Databases**

- As applications required more diverse data types, object-oriented databases and later hybrid models combining relational principles emerged.
- Distributed databases arose to handle geographically dispersed data, improving fault tolerance and scaling capabilities.

5. **2000s–Present: Big Data and NoSQL Era**
   - The surge in unstructured and semi-structured data from web applications drove the need for **NoSQL databases** (e.g., MongoDB, Cassandra), which prioritize scalability and performance.
   - Cloud-based and real-time systems emerged to cater to modern business needs.

Motivation for Database Systems

The motivation behind database systems, as discussed in the references you provided, revolves around several key points:

1. **Database Management Systems (DBMS) Objectives**: According to *Silberschatz et al.* in *Database System Concepts*, DBMSs were developed to address the need for a more structured and efficient way to manage, store, retrieve, and update vast amounts of data. The main motivation includes providing data independence, reducing redundancy, ensuring data integrity, improving accessibility, and allowing for secure, concurrent access by multiple users. Additionally, the growth of applications, big data, and the need for efficient querying further pushed the need for these systems.
2. **Data Modeling**: *Elmasri and Navathe* in *Fundamentals of Database Systems* highlight that the ability to represent the real-world entities and relationships between them in a structured format was a key motivation behind database systems. This included the development of the entity-relationship (ER) model for conceptual data modeling.
3. **Normalization and Data Integrity**: According to *Majumdar and Bhattacharyya*, the ability to model data in ways that maintain consistency, minimize redundancy, and ensure integrity was another motivator. Normalization of data allows for eliminating unnecessary duplication, making the storage more efficient and ensuring that data anomalies are avoided.

4. **Scalability and Performance**: The work by *Pramod J. Sadalage and Marin Fowler* in *NoSQL Distilled* points to the emergence of NoSQL databases as a response to the scaling challenges encountered by traditional RDBMS. With large-scale distributed systems, NoSQL databases were motivated by the need to handle big data and high velocity data generation, supporting flexible schema, and allowing for horizontal scaling.
5. **Complex Query Handling and Data Security**: The key motivations discussed in the reference materials also include the ability of modern database systems to manage complex queries (such as relational, aggregate, and transactional queries) while maintaining data security across multiple users.
6. **Big Data and Flexibility**: In *Shashank Tiwari's Professional NoSQL*, there's discussion of the challenges presented by the increasing amounts of semi-structured or unstructured data, which led to the creation of databases that provide flexibility and allow efficient storage and retrieval of such data formats. The demand for speed, lower latency, and agility in data management propelled NoSQL systems' rise.

The motivation for developing and advancing database systems lies in the need for handling large amounts of structured and unstructured data efficiently, ensuring data integrity, providing concurrency and security, and enabling flexible, scalable systems that can address modern application requirements. The evolution of data models, query languages, and storage systems in response to new technologies and emerging challenges has fueled the development of more sophisticated database management systems.

Characteristics of the Database Approach

1. Self-Describing Nature of a Database System

A database system includes both the data and its metadata (data about data). Metadata resides in a catalog called a **data dictionary**, describing the schema structure, constraints, and other details. This feature enables software applications to interact with data without requiring hardcoding.

2. Data Abstraction and Program Independence

- **Logical Data Independence:** Applications are unaffected by changes to the conceptual schema.
- **Physical Data Independence:** Applications are insulated from changes to the physical data storage methods.
  Data abstraction through the three-schema architecture (external, conceptual, and internal) promotes independence, simplifying system evolution and user interface design.

3. Support for Multiple Views of Data

Different users or applications often require customized perspectives of the same database, referred to as **views**. A database system can present tailored subsets of the data or derived data for these varying purposes without duplicating or altering the base data.

4. Sharing of Data and Multi-User Transaction Processing

Database systems support **multiuser environments**, allowing simultaneous data access while ensuring consistency and isolation through transaction management. They incorporate mechanisms to handle:

- **Concurrency control** to prevent conflicts during simultaneous transactions.
- **Isolation** ensuring one user\u2019s transaction does not interfere with another\u2019s.
- **Durability** ensuring changes persist even after system failures.

5. Minimal Redundancy and Data Consistency

The database approach centralizes data, reducing redundancy by allowing shared data across multiple applications. This decreases storage costs and ensures consistency since data is updated at a single location and reflected universally.

6. Data Integrity and Security

- **Integrity Constraints:** Ensure valid data entry (e.g., primary keys, foreign keys, and domain constraints).

- **Security Measures:** Define and enforce access controls through authentication, user roles, and permissions to safeguard data from unauthorized access or manipulation.

7. Concurrent Access by Multiple Users

Database Management Systems (DBMS) handle concurrent access to ensure:

- **Data Accuracy:** Enforcing isolation levels for transactions.
- **Deadlock Prevention:** Preventing processes from indefinitely waiting for resources.

8. Backup and Recovery Mechanisms

To protect data against failures (e.g., power loss or hardware malfunctions), DBMS provide tools for periodic backups and automatic recovery procedures that restore the database to a consistent state.

9. Support for Data Manipulation via Query Languages

Declarative query languages like **SQL** (Structured Query Language) enable users to interact with data without understanding storage details, supporting commands for:

- Data retrieval (`SELECT` statements).
- Modification (`INSERT`, `UPDATE`, `DELETE` commands).

10. Support for ACID Properties in Transactions

The database system enforces the following properties for reliable transaction processing:

- **Atomicity:** Ensures transactions are all-or-nothing.
- **Consistency:** Transforms the database from one valid state to another.
- **Isolation:** Prevents conflicts in concurrent transactions.
- **Durability:** Guarantees the permanence of committed transactions.

11. Separation of Application Logic and Data Management

Unlike file systems, where data handling logic is embedded in the application, DBMS manages data storage, retrieval, and maintenance independently. This modularity promotes scalability and ease of development.

Actors on the scene

1. **Database Users**:
    - **Application Programmers**: Write programs that access and manipulate data in the database, typically via SQL or other DBMS query languages.
    - **End Users**: Use the applications built by the programmers or direct query tools to interact with the database. They may be casual users or experts who understand the complexities of the data model.
    - **Database Administrators (DBAs)**: Manage and maintain the database systems, ensuring proper functionality, security, backup, recovery, and optimization. DBAs also make decisions about the database structure and user access privileges.
2. **Database System**:
    - The **DBMS** itself plays a critical role as the intermediary between users and the actual data storage system.
    - **DBMS Software**: This refers to the actual software components of a DBMS that receive queries and interact with storage mechanisms.
3. **Data**:
    - The actual **Database** or collection of data that users and the DBMS manage. It includes the actual tables, schemas, views, stored procedures, and indexes.
4. **Query Processor**:
    - The component of the DBMS responsible for interpreting and optimizing SQL queries before passing them to the storage engine. It converts high-level queries into execution plans.
5. **Storage Manager**:
    - This component is responsible for managing physical storage. It ensures that the data is stored efficiently and that users can access it when needed.
    - It interacts with the disk and file system, managing data, indexes, and buffers.
6. **Transaction Manager**:

- This ensures the consistency, isolation, and atomicity of the transactions. It handles operations related to transactions like commit, rollback, and ensuring that transactions do not interfere with each other.

7. **Concurrency Control Manager**:
    - Prevents issues related to multiple transactions accessing the same data simultaneously (e.g., preventing lost updates, inconsistent reads).
8. **Recovery Manager**:
    - Ensures that the database is recoverable in case of system failure. It uses techniques like logging to restore the database to a consistent state after failures.
9. **Security Manager**:
    - Enforces access control and protects sensitive data by ensuring that only authorized users can perform operations. It includes authentication and authorization.

These actors and components all play distinct but interrelated roles in ensuring that the DBMS operates effectively and efficiently, meeting the needs of users while ensuring data integrity, security, and consistency.

Workers behind the Scene

The Database Management System (DBMS) that manage the actual operation, processing, and maintenance of the database. These are the system-level components that make sure everything functions smoothly for the end-users. Here's a breakdown of those key workers:

1. **Query Processor**:
    - The query processor is responsible for parsing and executing SQL queries. When a user submits a query, the query processor translates that query into a form that can be executed by the DBMS.
    - This includes tasks such as syntax checking, optimization of query execution, and converting the query into a series of low-level instructions that can interact with the storage system.
2. **Transaction Manager**:

- The transaction manager ensures that transactions are processed reliably and conform to the ACID properties (Atomicity, Consistency, Isolation, and Durability). It manages transaction execution and guarantees that all transactions meet these properties.
- It is also responsible for handling transaction failures (via rollback) and commit operations to ensure that the database remains in a consistent state.

3. **Storage Manager**:
    - This is the component that manages how data is stored physically on disks. The storage manager handles the organization of data into files and pages, ensuring that records are stored and retrieved efficiently.
    - It is responsible for managing buffer pools, data storage structures (like tables, indexes), and data retrieval mechanisms like the access methods that determine how data is fetched based on the query.
4. **Buffer Manager**:
    - The buffer manager works in conjunction with the storage manager and is responsible for keeping frequently accessed data pages in memory (buffer pool). It optimizes access by ensuring that the DBMS doesn't repeatedly access the slower disk storage.
    - It deals with page replacement algorithms, deciding which pages to bring into memory and which to evict.
5. **Lock Manager**:
    - The lock manager ensures concurrency control by managing locks on the data. When multiple users or transactions are accessing the same data, the lock manager ensures that transactions do not conflict, and the integrity of the database is maintained.
    - It determines which locks are needed and applies them to ensure that data remains consistent when multiple users attempt to update or read the same data at once.
6. **Recovery Manager**:
    - The recovery manager is crucial for ensuring the durability of data, especially after failures like crashes or power outages. This manager is responsible for

maintaining logs that help restore the database to its last consistent state after a failure.
    - The recovery manager works to replay or undo operations from transaction logs to ensure that the database returns to a valid state without data loss.
7. **Authorization and Security Manager**:
    - The security manager controls access to the database by enforcing user authentication and authorization policies. It verifies users' identities and grants appropriate access based on roles and permissions.
    - It ensures that only authorized users can perform certain operations on the database, such as reading or writing specific data.
8. **File Manager**:
    - The file manager interfaces with the operating system's file management system and handles the low-level operations of data storage on disk. It manages files containing data and ensures proper file allocation for records, including file creation, deletion, and organization.

These **behind-the-scenes workers** play essential roles in ensuring that data is efficiently managed, queries are processed correctly, and transactions are handled in a way that guarantees both integrity and consistency. The collective interaction of these components ensures smooth operation of a DBMS while maintaining security, concurrency, and failure resilience.

Advantages of using DBMS approach

Database Management System (DBMS) approach over traditional file-based systems. These advantages include:

1. Data Redundancy and Inconsistency Control

- In a DBMS, data is typically stored in a centralized manner, which reduces redundancy (repetition of data) compared to file-based systems, where multiple copies of the same data may exist.
- A DBMS allows for a consistent view of data by ensuring that updates to data are centralized and propagated to all relevant parts of the system, reducing inconsistency.

2. Data Independence

- DBMS provides **logical data independence**, meaning that changes to the logical schema (e.g., changes to table structure) do not affect the external schema (how the users access the data).
- **Physical data independence** is also achieved, where changes to the physical storage of data do not require changes in the DBMS or applications.

3. Improved Data Security

- A DBMS can enforce strict access controls and permissions, ensuring that only authorized users have access to certain types of data.
- Encryption mechanisms and other security measures can be built in to secure data against unauthorized access or modification.

4. Data Integrity

- DBMS enforces data integrity constraints (e.g., primary keys, foreign keys, check constraints) to ensure that the data entered into the system is valid, correct, and reliable.
- This helps in maintaining consistency and accuracy of data across different parts of the system, reducing the risk of erroneous data.

5. Efficient Data Access

- The DBMS provides powerful query processing tools, allowing users to retrieve, insert, update, or delete data quickly using structured query languages (like SQL).
- It supports indexing and optimization techniques that improve data retrieval speed, making it much more efficient than a traditional file-based system.

6. Data Concurrency Control

- A DBMS allows multiple users to access and manipulate the data concurrently while ensuring consistency and avoiding conflicts.

- The transaction management system of the DBMS handles issues like **locking**, **deadlock resolution**, and the **isolation** of transactions, ensuring that concurrent operations do not corrupt the data.

7. Backup and Recovery

- A DBMS has built-in mechanisms for data backup and recovery. Regular backups of data can be automatically performed, and the DBMS ensures that, in case of a system crash or failure, the data can be restored to a consistent state with minimal loss.

8. Centralized Control

- A DBMS allows for centralized control over the data. This ensures consistency in how data is handled, accessed, and modified.
- Centralization simplifies administration tasks like auditing, enforcing policies, monitoring usage, and backup management.

9. Reduced Application Development Time

- With the database system providing powerful query processing, data integrity, and access controls, application developers can focus on higher-level logic rather than dealing with low-level data storage and retrieval operations.
- The DBMS offers a unified interface for multiple applications, enabling quicker development and easier maintenance.

10. Support for Transaction Management

- Transactions are managed by the DBMS to ensure the ACID properties (Atomicity, Consistency, Isolation, Durability). This guarantees that data remains consistent even in cases of system failure, crashes, or conflicts during data manipulation.

11. Flexibility and Scalability

- A DBMS offers high flexibility for managing different types of data and can scale to accommodate growing amounts of data and user requests.
- Modern DBMS technologies allow for scalability, enabling the system to handle increasing workloads without significant performance degradation.

12. Multi-User Support

- Multiple users can interact with the database simultaneously through various interfaces or applications, each having its view and access rights.
- The DBMS efficiently manages user requests, ensuring safe concurrent access to data.

13. Easier Maintenance and Administration

- Since data is stored and maintained in one centralized system, database administration tasks like data consistency checks, updates, performance monitoring, and tuning become easier.
- The DBMS simplifies the complexity of managing multiple file systems.

By providing these benefits, a **DBMS approach** significantly enhances data management, security, performance, and scalability, making it a preferred choice over traditional file systems for handling large and complex data storage and retrieval needs.

Data Models, Schema, and Instances

Data models, **schema**, and **instances** are important concepts fundamental to understanding how data is represented, structured, and manipulated within a DBMS. Here's a breakdown of each concept:

1. Data Models

A **data model** defines how the data is logically structured, stored, and manipulated in the DBMS. It provides a framework for designing and describing the relationships between data elements. Different types of data models are used in DBMSs, and the text typically focuses on the following key models:

- **Relational Data Model**:
    - The relational data model is based on the concept of **tables** (relations), where data is stored in rows and columns. Each row represents a record, and each column represents an attribute of the record. The relational model is widely used in modern DBMSs.
    - Key concepts: tables, keys (primary key, foreign key), relationships (one-to-many, many-to-many), and normalization.
- **Entity-Relationship Model (ER Model)**:
    - The ER model is used to design and conceptualize data by representing entities (objects), their attributes, and relationships between them. It is used at the early stages of database design.
    - Key components include entities, attributes, relationships, and keys.
- **Hierarchical Model**:
    - In this model, data is represented in a tree-like structure where each record has a parent-child relationship. It was one of the earliest models for DBMS but is less common in modern systems.
- **Network Model**:
    - The network model represents data using a graph, where nodes are records, and edges represent relationships between them. Each record can have multiple parent and child records. It is more flexible than the hierarchical model but less used today compared to the relational model.

Data models enable DBMS designers and users to represent data in an understandable and usable way, facilitating the querying, manipulation, and storage of information.

2. Schema

A **schema** is the logical description of the entire database. It defines the structure of the database, including its tables, relationships, constraints, views, and other elements. A schema specifies how data is organized and how the relationships among data entities are constructed. The schema is defined during the database design phase and does not change frequently unless the structure of the database is modified.

Schemas can be categorized into different levels:

- **Physical Schema**: Describes how the data is stored on the hardware (e.g., files, indices). This level is concerned with the performance aspects of storage and retrieval.
- **Logical Schema**: Represents the logical view of the database, including tables, fields, and the relationships among them. It is independent of physical storage.
- **External Schema** (or View Schema): Refers to the way different users (or applications) see the data in the database. Different users may have different views based on their needs (e.g., a customer may only see specific customer-related data, while an administrator may have access to all data).

The concept of **three levels of schema architecture** (external, logical, and physical) enables **data independence** by separating user views from internal database implementation.

3. Instances

An **instance** of a database is the **current state** of the database at a particular point in time. It includes all the actual data in the database, i.e., the content of the tables, records, and attributes. The instance is a dynamic entity that can change over time as data is added, modified, or deleted.

Key points about instances:

- An instance corresponds to the **data stored in the database** and is distinct from the schema (which is the structural definition of the database).
- **When data is inserted, deleted, or updated**, the database instance changes accordingly.
- The **schema** remains unchanged, but the **instance** can evolve as users perform various operations like insertions, updates, or deletions of data.

Example:

- **Schema**: Describes the structure of the database, such as a table Student with attributes StudentID, Name, Age, and GPA.
- **Instance**: Represents actual records in the Student table at a given point, such as:
- StudentID | Name | Age | GPA

- ---
- 1 | Alice | 20 | 3.8
- 2 | Bob | 22 | 3.5
- 3 | Carol | 21 | 3.9

In summary:

- **Data Models** provide the framework for defining and organizing data.
- **Schema** defines the logical structure and relationships of the data in the database.
- **Instances** represent the current, actual data stored in the database at a specific time.

These concepts together help define how a database is organized, manipulated, and represented. They form the foundation of how information is stored and queried in a DBMS.

Three-Schema Architecture and Data Independence

The **Three-Schema Architecture** is discussed as an important concept for achieving **data independence**. Data independence means that changes made at one level of a database do not affect other levels. The three-schema architecture helps achieve this separation by dividing the database into three levels:

Three-Schema Architecture

The three-schema architecture consists of three different levels of abstraction, each describing the database from a different perspective. These levels are:

1. **Internal Schema** (Physical Level):
    - This level describes **how data is stored** on the physical storage medium (disk, cloud storage, etc.).
    - It defines the **physical structures** used to store data, such as files, records, indices, and other storage techniques.
    - It focuses on the **efficient storage** of the data and defines the physical access methods used to retrieve it.
2. **Conceptual Schema** (Logical Level):

- The conceptual schema represents the **logical structure** of the entire database. It defines **what data is stored** in the database and the relationships among the data.
- This level describes data without considering how it is physically stored.
- The schema includes entities, attributes, tables, and relationships but does not specify the details of how they are stored.

3. **External Schema** (View Level):
   - This level describes how users or **applications view the data**. Different users or groups of users might have different views of the database depending on their needs (e.g., a manager may see only summaries of the data, while a database administrator may have full access).
   - The external schema defines **user-specific views** of the data, such as restricting access to certain records, hiding sensitive information, or presenting data in a particular format.

Data Independence

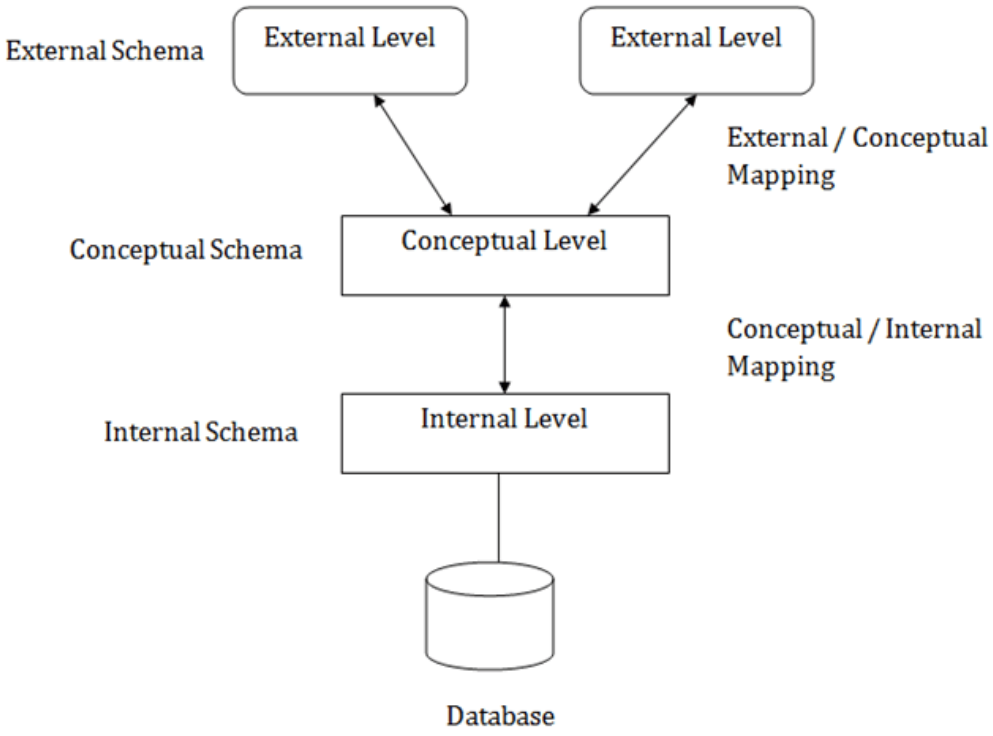
Data independence is the ability to change the schema at one level of the architecture without affecting the schema at the other levels. There are two types of data independence:

- **Logical Data Independence**:
  - Refers to the ability to change the conceptual schema (logical structure) without affecting the external schemas or application programs.
  - Example: Changing the structure of the database tables (such as adding a new attribute) without requiring users or applications to modify their views.
  - This is considered a **high level of data independence** because changes at the logical level often impact the user and applications less than changes at the physical level.
- **Physical Data Independence**:
  - Refers to the ability to change the internal schema (physical storage) without affecting the conceptual schema or the external schemas.

- Example: Changing the way data is physically stored (e.g., changing the indexing structure or switching from one storage medium to another) without requiring any changes to how users see or interact with the data.
- This is considered a **high level of data independence** because changes at the physical level usually require minimal changes to the logical structure of the database.

Diagram of Three-Schema Architecture

Here's a simple diagram representing the Three-Schema Architecture:



External Schema: Represents the views of users or applications (user-specific).

- **Conceptual Schema**: Describes the logical structure of the database (without considering physical storage).
- **Internal Schema**: Describes how data is actually stored on the physical media (files, indexes, etc.).

Benefits of the Three-Schema Architecture and Data Independence

1. **Separation of Concerns**: Each schema abstracts away certain details. The physical storage and structure are hidden from the user (and application programmers), which leads to cleaner separation between the data model and the applications using it.
2. **Flexibility in Storage Management**: The DBMS can modify the internal schema (e.g., changing storage structures or indexing methods) without impacting how users access the data at the external level.
3. **Adaptability for Changes**: Changes in the logical schema (e.g., adding or removing attributes) are isolated from the external schema. Thus, it avoids the need for altering existing applications and user queries.
4. **Improved Security**: The external schema can be customized to expose only necessary data to different users, which can be critical for maintaining security.

In summary, the **Three-Schema Architecture** provides an organized approach to manage data abstraction levels in a database system, while **data independence** ensures flexibility in database design and maintenance, thus reducing the potential impact of structural changes on user interactions.

The Database System Environment

Database System Environment is an essential concept that describes the context in which the database system operates. This environment encompasses the entire system, including its components, users, and interactions. It is often presented as the broader context of the DBMS, illustrating how the DBMS interacts with other software and hardware components, as well as with the users.

A DBMS is a complex software system discuss the types of software components that constitute a DBMS and the types of computer system software with which the DBMS interacts It consists of two parts: The top part: various users and their interfaces, The lower part: storage of data and processing of transactions Access to the disk is controlled primarily by the operating system (OS) buffer management module to schedule disk read/write, because this has a considerable effect on performance

A higher-level stored data manager module: controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog

DDL Statement: used by the DBA for defining the database and tuning it

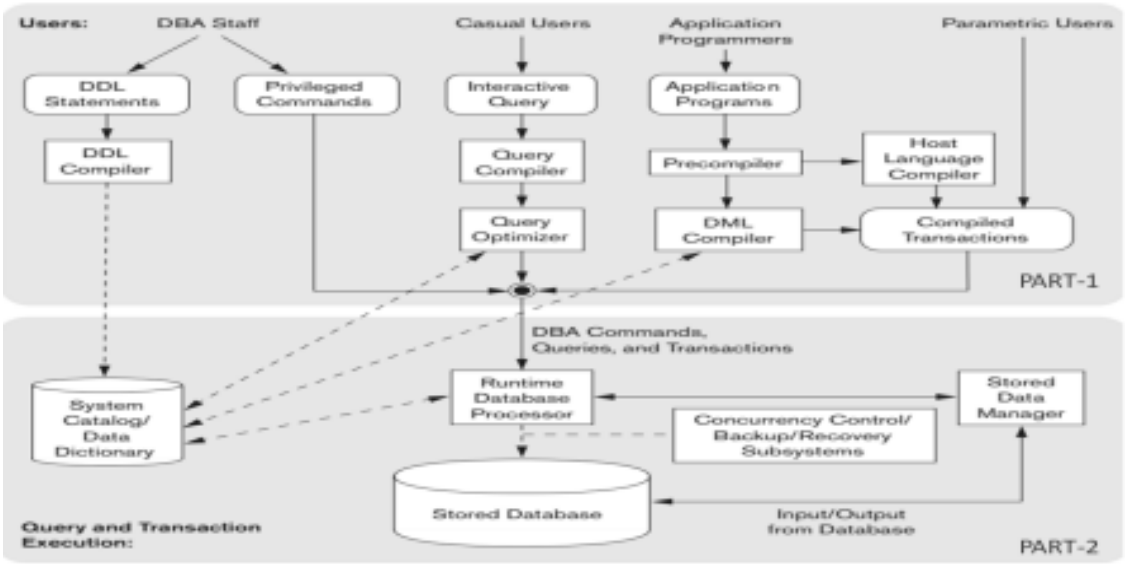
DDL Compiler: processes schema definitions and stores descriptions of the schemas in the DBMS catalog

Privileged commands: The commands which are exclusively used by the DBA

Interactive query: The interface by which the casual users and persons with occasional need for information from the database interact

Query compiler: Queries are parsed and validated for correctness of the query syntax

Query optimizer: Responsible for optimizing the query and its execution



Pre-compiler: Extracts DML commands from an application program written in a host programming language

DML compiler: Compilation of pre-compiled DMLC commands into object code for database access

Host language compiler: The rest of the program is sent to the host language compiler.

Compiled transactions: Canned transactions are executed repeatedly by parametric users, who simply supply the parameters to the transactions

Database System Environment: Key Components

1. **Hardware**:
    - The physical devices where the DBMS runs and where the data is stored. This includes the **CPU**, **disk storage**, **memory**, and **network infrastructure**. It also extends to **secondary storage devices** (such as hard disks and solid-state drives) used for persistent storage of data, as well as **network hardware** that allows multiple users to access the database.
2. **Software**:
    - This refers to the **DBMS software** itself, which manages the database and its operations. The DBMS provides a set of tools and functionalities for creating, manipulating, managing, and securing the database.
    - **Operating System (OS)**: The underlying operating system that manages the hardware and provides services to the DBMS (e.g., memory management, file management, process scheduling).
    - Other **supporting software** could include **network protocols**, **user interface software**, and various **middleware** that facilitate communication between the DBMS and applications or users.
3. **Data**:
    - This refers to the **actual database**—the structured collection of information stored and managed by the DBMS. It is organized in the form of tables, relationships, schemas, and constraints, allowing the database to efficiently handle data storage, access, and modification.
    - Data may include **metadata** (information about the data), which provides context about how the actual data is organized and related.
4. **Users**:

- Users can be categorized into different types based on their roles and level of interaction with the DBMS.
    - **Database Administrators (DBAs)**: These are the experts responsible for the management, maintenance, and security of the database system.
    - **End Users**: These users interact with the database through applications or query languages. They can be classified into:
        - **Casual Users**: Use high-level query interfaces and may not be familiar with the database's internal details.
        - **Naive Users**: Have little knowledge of the system and use simple applications to perform specific tasks like querying and data entry.
        - **Application Programmers**: Write programs that interact with the DBMS, such as data-driven applications, by using query languages (e.g., SQL) to manipulate the database.

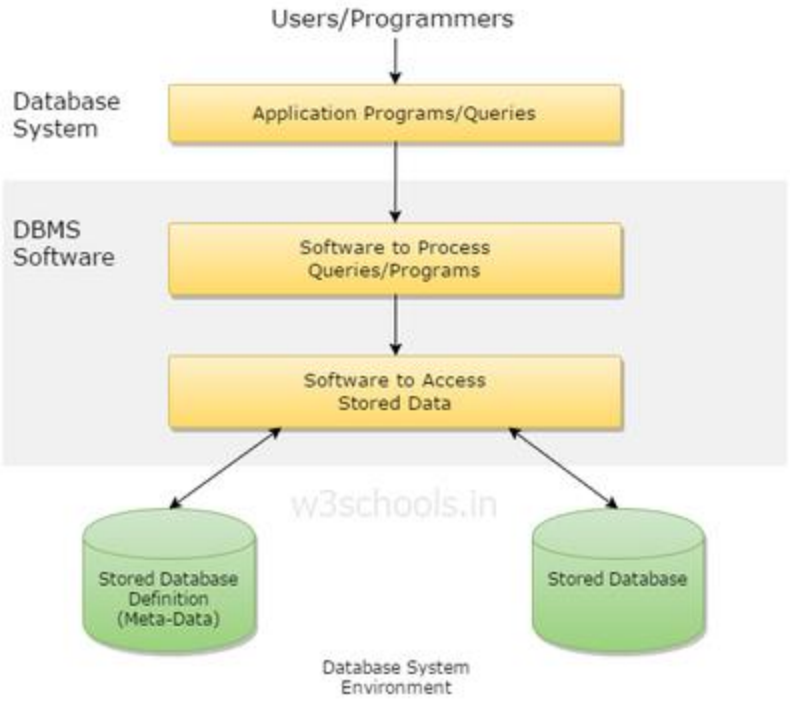
5. **Data Models**:
    - The **data model** defines how data is structured, organized, and related within the database. Different data models can be used to design and describe the structure of the database:
        - **Relational Model** (e.g., tables with rows and columns)
        - **Hierarchical Model** (e.g., tree-like structure)
        - **Network Model** (e.g., graph-based structure)
    - The choice of data model influences how data is accessed, queried, and manipulated.
6. **Database Languages**:
    - A DBMS interacts with users and applications through database languages that allow for the manipulation, querying, and definition of data. There are different types of database languages:
        - **Data Definition Language (DDL)**: Used to define the database structure (e.g., creating tables, defining relationships, and constraints).
        - **Data Manipulation Language (DML)**: Used to insert, update, delete, and retrieve data from the database.

        - **Data Control Language (DCL)**: Used to define user access and control privileges (e.g., grant or revoke permissions).
        - **Query Language (e.g., SQL)**: A specialized language that is used by end users, application programmers, and DBAs to communicate with the database for querying and updating data.

7. **Transactions**:
    - In a database environment, a **transaction** is a unit of work that is performed on the database. Transactions are important because they ensure that the database maintains its **ACID** properties:
        - **Atomicity**: Transactions are all-or-nothing operations.
        - **Consistency**: Transactions preserve the integrity of the database.
        - **Isolation**: Transactions are isolated from one another, preventing conflicts.
        - **Durability**: Once a transaction is committed, it is permanently stored.
    - The transaction system in the DBMS helps to manage **concurrency control** (handling simultaneous transactions) and ensures **data integrity**.

Database System Environment: Interactions and Flow

A typical flow in the database system environment might look like this:



The Core Functions of the Database System Environment:

- **Data Definition**: The DBMS provides the tools to define the structure and types of data, including relationships among data elements.
- **Data Storage**: Data is stored on physical devices and manipulated according to the schema and transaction rules.
- **Data Manipulation**: Users can insert, update, delete, or query the data through supported query languages.
- **Concurrency Control**: Ensures multiple users can work with the database simultaneously without conflicts or data corruption.
- **Backup and Recovery**: The DBMS ensures that data is regularly backed up, and it can recover from failure scenarios using logs and transaction management.

Benefits of the Database System Environment

- **Centralized Management**: A single database system provides uniformity, control, and maintenance, allowing organizations to manage and query large amounts of data efficiently.
- **Security**: The environment can be secured using access controls and encryption, ensuring that only authorized users can access sensitive data.
- **Efficiency**: The DBMS handles queries, optimizations, and concurrency issues, making data access and management more efficient for users and applications.

In summary, the **Database System Environment** involves the combination of hardware, software, data models, users, and transactions that work together to ensure efficient database operations. It highlights how the DBMS integrates into a larger computational ecosystem while ensuring data management, security, and performance.

Centralized Architecture and **Client/Server Architecture for DBMS**

Centralized Architecture and **Client/Server Architecture** are two types of architectural models for database systems. Here's a detailed explanation based on the textbook:

1. Centralized Database Architecture

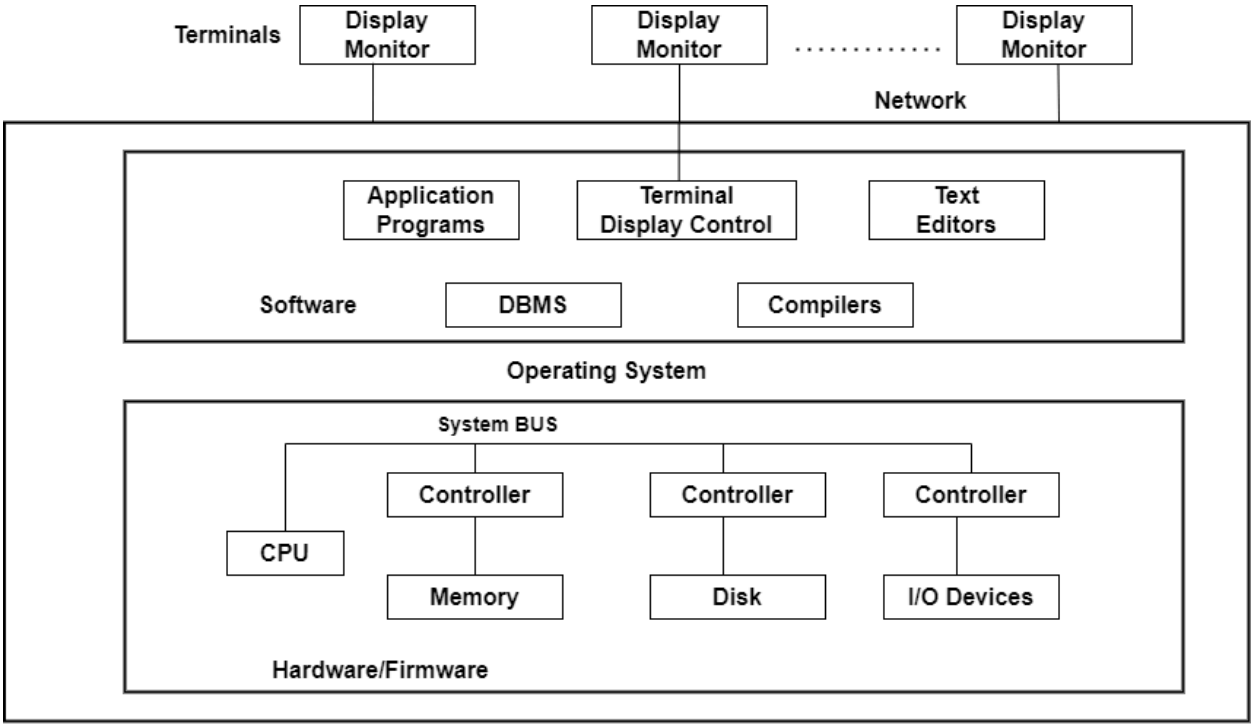
Centralized architecture refers to a system where all the **database management tasks** are handled by a **single central computer** (a server). The system does not have distribution in terms of multiple nodes; the entire database, its management, and access happens in one location.

Characteristics of Centralized Database Architecture:

- **Single Database**: All data is stored in a central location and can be accessed by various clients through network connections.
- **Simplicity**: Since all database processing happens on a central server, the management and maintenance of the system are relatively easier.
- **Centralized Control**: A **database administrator (DBA)** has full control over the database and its operations. The user requests (e.g., queries, data updates) are processed and managed by the central server.

- **Reduced Communication Overhead**: Communication among users typically happens over a single network path, reducing complexities in data synchronization and network architecture.
- **Potential Bottleneck**: A limitation of this architecture is the risk of a performance bottleneck at the central server, especially when there are many users accessing it simultaneously.

Example Architecture:



Centralized architecture of DBMS

Architectures for DBMSs have generally followed trends seen in architectures for larger computer systems. The primary processing for all system functions, including user application programs, user interface programs, and all DBMS capabilities was handled by mainframe computers in earlier systems. The primary cause of this was that the majority of users accessed such systems using computer terminals with limited processing power and merely display capabilities. Only display data and controls were delivered from the computer system to the

display terminals, which were connected to the central node by a variety of communications networks, while all processing was done remotely on the computer system.

The majority of users switched from terminals to PCs and workstations as hardware prices decreased. Initially, Database Systems operated on these computers in a manner akin to how they had operated display terminals. As a result, the DBMS itself continued to operate as a centralized DBMS, where all DBMS functionality, application program execution, and UI processing were done on a single computer. The physical elements of a centralized architecture Client/server DBMS designs emerged as DBMS systems gradually began to take advantage of the user side's computing capability.

Advantages:

- **Simple Architecture**: Easy to design and implement.
- **Centralized Control and Management**: Easier to maintain security, backups, and consistency because everything is in one place.
- **Cost-Effective for Small Scale**: Ideal for smaller organizations or projects with limited database access and less demand for parallel operations.

Disadvantages:

- **Scalability Issues**: The central server might become overloaded if there is heavy client traffic or an increasing number of requests.
- **Single Point of Failure**: If the central server fails, the entire system becomes unavailable.
- **Limited Performance**: Processing large queries or simultaneous requests can be slow because everything is routed through one server.

2. Client/Server Architecture

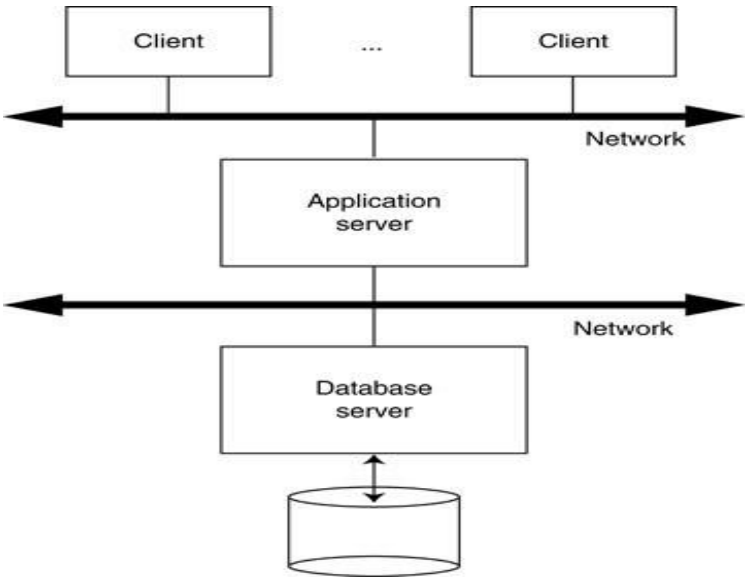
In **Client/Server Architecture**, the database system is divided into two main components: the **client** and the **server**. The **client** is typically a user-facing application that sends requests to the

server, where the database management system (DBMS) is hosted. The server then processes the database requests and sends results back to the clients.

Characteristics of Client/Server Architecture:

- **Distributed Components**: The system separates the database functionality into two parts: the **client application** and the **server** that holds the database.
- **Client Side**: Clients handle the user interface, send requests (like queries, insertions) to the server, and display results.
- **Server Side**: The DBMS resides here, and it executes the queries, manages the database, enforces security, handles access control, and does the actual data processing.
- **Communication Protocols**: Clients and servers communicate using well-defined communication protocols like **TCP/IP** (Transmission Control Protocol/Internet Protocol) or **HTTP**.

Example Architecture:



- **Clients (Application Layer)**: This is where the **user interaction** occurs, and the clients send SQL queries or commands to the server.

- **Server (Database Management Layer)**: The server contains the database, handles all data management tasks, performs computations, and responds to client requests.

Advantages:

- **Scalability**: You can scale up the number of clients, allowing a large number of users to interact with the system concurrently.
- **Flexibility**: Clients can be located on different machines (e.g., desktops, mobile phones, workstations).
- **Separation of Concerns**: The user interface and application logic are separated from the data management and storage, improving maintenance and security.
- **Load Distribution**: Unlike centralized systems, server-based systems allow you to balance the processing load by having the server manage the data and the client only handling the presentation layer.

Disadvantages:

- **Increased Complexity**: The architecture requires a good design for communication between clients and the server.
- **Performance Overhead**: Communication between clients and the server can introduce latency, especially if there are a large number of concurrent clients.
- **Server Dependency**: If the server goes down, the system will be unavailable to all clients.

Comparison: Centralized vs. Client/Server Architecture

Feature	Centralized Architecture	Client/Server Architecture
Data Location	Single central database server	Database is on the server, client handles UI
Scalability	Difficult to scale under heavy traffic	Highly scalable, more clients can be added easily
Server Load	All load is on the central server	Server load can be distributed, allowing for better scalability

Feature	Centralized Architecture	Client/Server Architecture
Fault Tolerance	Single point of failure	Redundant clients or servers can improve fault tolerance
Client	Clients directly connect to the DB	Clients use applications to send queries to the server
Complexity	Simple architecture but with limitations	More complex, but scalable and flexible
Examples	Small businesses, personal systems	Large businesses, cloud-based applications, multi-tier apps

- **Centralized Architecture** works well in small-scale environments where database access is infrequent, and scalability is not a critical issue.
- **Client/Server Architecture** is better for distributed, large-scale applications where many users need access to the database concurrently. It offers greater flexibility and scalability but is more complex to manage and requires effective communication protocols between clients and servers.

Each architecture has its place based on the size, usage, and specific needs of an organization or system. For large-scale applications, **Client/Server** architecture is preferred, while for simpler or smaller setups, **Centralized Architecture** might suffice.

Classification of DBMS

Classification of DBMS refers to different ways database systems can be classified based on factors like the **number of users**, **data models**, and **distribution**.

Here are the main classifications of DBMS:

1. Classification Based on the Number of Users

a. Single-User DBMS:

A **Single-User DBMS** is designed to handle one user at a time. It is suitable for personal or small-scale applications where the database only needs to support a single user.

- **Characteristics:**
  - Only one user accesses the database at any given time.
  - Can be used in systems like desktop applications or embedded devices.
  - Examples: Microsoft Access, local SQLite databases in apps.

b. Multi-User DBMS:

A **Multi-User DBMS** is designed to handle multiple users who can concurrently access and modify the database. It includes features such as **concurrency control** and **transaction management** to maintain the consistency and isolation of each user's operations.

- **Characteristics:**
  - Multiple users access the database at the same time.
  - Includes mechanisms to prevent conflicts, ensure data integrity, and control concurrency.
  - Suitable for business applications like ERP systems, social media platforms, etc.
  - Examples: MySQL, PostgreSQL, Oracle Database.

2. Classification Based on the Number of Processors

a. Centralized DBMS:

In a **Centralized DBMS**, the database management system and the data itself reside on a single server or machine.

- **Characteristics:**
  - A single machine or server is responsible for processing all database requests.
  - Good for systems where resources are centralized and not spread out geographically.
  - Examples: Traditional RDBMS running on standalone servers.

b. Distributed DBMS:

A **Distributed DBMS** uses multiple interconnected databases that are distributed across different machines or locations but are treated as a unified system.

- **Characteristics:**
    - Data is distributed across different nodes, either on separate computers or geographically dispersed locations.
    - The DBMS provides transparency (users do not need to know where data is stored or how it is distributed).
    - It handles issues like **data fragmentation, replication, and consistency**.
    - Examples: Google Bigtable, Apache Cassandra.

c. Parallel DBMS:

A **Parallel DBMS** uses multiple processors and disks to execute queries concurrently, providing better performance, especially for large-scale data processing.

- **Characteristics:**
    - The database is designed to run on a **parallel computing architecture**, using multiple CPUs or cores simultaneously.
    - Improves performance and can process large datasets efficiently by splitting tasks across multiple processors.
    - Example: IBM DB2 with Parallel Sysplex, Oracle Exadata.

3. Classification Based on the Data Model

a. Relational DBMS (RDBMS):

An **RDBMS** is based on the relational model, where data is organized into tables (relations) and is accessed using SQL (Structured Query Language).

- **Characteristics:**
    - Data is stored in tables consisting of rows and columns.

- Strong theoretical foundation based on **set theory** and **first-order predicate logic**.
- Supports relational operations such as **join, select, union, etc.**
- Examples: Oracle Database, MySQL, PostgreSQL.

b. Hierarchical DBMS:

A **Hierarchical DBMS** stores data in a tree-like structure where each record has a single parent, and each child can have multiple parents.

- **Characteristics:**
  - Organizes data in a tree structure (parent-child relationships).
  - Well-suited for applications requiring hierarchical data organization (e.g., organizational structures or product categorizations).
  - Query operations are typically navigational (you need to know the paths from parent to child).
  - Example: IBM Information Management System (IMS).

c. Network DBMS:

A **Network DBMS** is similar to a hierarchical DBMS, but it allows more complex relationships among data. Records can have multiple parents, creating a graph or network of interconnected data.

- **Characteristics:**
  - Uses a graph or network structure where each record can have multiple relationships with other records.
  - The **navigation** of the network is more flexible than in a hierarchical model.
  - Data is organized into sets (rather than a strict hierarchy).
  - Example: Integrated Data Store (IDS), Raima DBMS.

d. Object-Oriented DBMS (OODBMS):

An **OODBMS** uses object-oriented programming principles to represent and manage data. Data in the system is represented as objects, similar to how data is represented in programming languages like Java and C++.

- **Characteristics:**
  - Data and behavior (methods) are encapsulated into objects.
  - Supports complex data types, inheritance, polymorphism, and more.
  - Example: db4o, ObjectDB.

e. NoSQL DBMS (Not Only SQL):

NoSQL DBMS refers to a wide variety of database models that do not follow the traditional relational model. These include **document-based, key-value, column-family, and graph databases**.

- **Characteristics:**
  - Optimized for high scalability and performance for large datasets.
  - Can store **unstructured** or **semi-structured** data.
  - Does not require a fixed schema and can handle rapidly changing data.
  - Example: MongoDB (Document), Cassandra (Column-Family), Neo4j (Graph), Redis (Key-Value).

4. Classification Based on the DBMS's Mode of Operation

a. Online Transaction Processing (OLTP) Systems:

OLTP Systems are designed to handle high volumes of short online transactions (insert, update, delete, etc.). These systems are typically used in real-time business applications.

- **Characteristics:**
  - Handles many small transactions that require high performance and quick responses.
  - Common in banking, shopping cart systems, and reservation systems.
  - Example: MySQL, Microsoft SQL Server.

b. Online Analytical Processing (OLAP) Systems:

OLAP Systems are designed for querying and analyzing data for decision support, often using large data volumes for reporting and analysis purposes. These systems support complex queries involving aggregations and multi-dimensional data.

- **Characteristics:**
    - Handles complex queries with large datasets for analytical reporting and business intelligence.
    - Used for trend analysis, forecasting, and big data analytics.
    - Example: IBM Cognos, Oracle OLAP.

c. Data Warehousing Systems:

Data Warehousing Systems store and manage large amounts of historical data designed for analytical processing rather than transactional operations.

- **Characteristics:**
    - Primarily used for decision support systems (DSS).
    - Data is often sourced from OLTP systems and other external sources.
    - Involves data integration, cleansing, and processing for analytics.
    - Example: Snowflake, Google BigQuery.

The **classification of DBMS** allows us to understand the **types of systems** and their applicability based on various parameters, such as the **number of users**, **data model**, and **performance requirements**. The appropriate choice of DBMS depends on the use case and requirements of the system, whether it needs high transaction support, complex analytics, large-scale distributed processing, or flexible data handling.

Practical: Create a database table, add constraints (primary key, unique, check, not null), insert rows, update and delete rows using SQL DDL and DML commands.

Practical example in SQL for creating a database, adding constraints, inserting rows, and then updating and deleting data.

1. Creating a Database and Table

First, we'll create a database and a table within it. The table will have various columns and constraints.

```
-- Create the database
CREATE DATABASE School;

-- Switch to the database
USE School;

-- Create the 'Students' table
CREATE TABLE Students (
    student_id INT NOT NULL,              -- Primary Key column
    first_name VARCHAR(50) NOT NULL,      -- Not Null constraint
    last_name VARCHAR(50) NOT NULL,       -- Not Null constraint
    age INT CHECK (age >= 18 AND age <= 100),  -- Check constraint: Age must be between 18 and 100
    email VARCHAR(100) UNIQUE,            -- Unique constraint: email should be unique
    PRIMARY KEY (student_id)              -- Primary Key constraint
);
```

- The student_id column is set to **NOT NULL** and is the **PRIMARY KEY**.
- first_name and last_name are required (i.e., **NOT NULL**).
- The age column uses a **CHECK** constraint to ensure the age is between 18 and 100.
- The email column is marked **UNIQUE**, so all values in this column must be distinct.

2. Inserting Rows into the Table

Now let's insert some rows into the table, respecting the constraints:

```
-- Insert rows into the 'Students' table
INSERT INTO Students (student_id, first_name, last_name, age, email)
```

```
VALUES (1, 'John', 'Doe', 20, 'john.doe@example.com');

INSERT INTO Students (student_id, first_name, last_name, age, email)
VALUES (2, 'Jane', 'Smith', 22, 'jane.smith@example.com');

INSERT INTO Students (student_id, first_name, last_name, age, email)
VALUES (3, 'David', 'Jones', 19, 'david.jones@example.com');
```

In this case:

- We're inserting values for each column (except for student_id, which is being manually assigned since we declared it as a part of the primary key).

3. Updating Rows

Next, we will perform an **UPDATE** operation to modify data in the table. For instance, updating the email address of a student:

```
-- Update a student's email
UPDATE Students
SET email = 'john.doe@university.com'
WHERE student_id = 1;
```

- This will update the email of the student whose student_id is 1.

4. Deleting Rows

Next, we'll demonstrate how to delete a record. For example, we can delete the student record with student_id = 3:

```
-- Delete a student by student_id
DELETE FROM Students
WHERE student_id = 3;
```

This will remove the student with the student_id of 3 from the table.

Complete SQL Script

Below is a complete SQL script that demonstrates creating the table, inserting, updating, and deleting rows:

```
-- Step 1: Create Database and Table with Constraints
CREATE DATABASE School;
USE School;

CREATE TABLE Students (
    student_id INT NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    age INT CHECK (age >= 18 AND age <= 100),
    email VARCHAR(100) UNIQUE,
    PRIMARY KEY (student_id)
);

-- Step 2: Insert Rows
INSERT INTO Students (student_id, first_name, last_name, age, email)
VALUES (1, 'John', 'Doe', 20, 'john.doe@example.com');
INSERT INTO Students (student_id, first_name, last_name, age, email)
VALUES (2, 'Jane', 'Smith', 22, 'jane.smith@example.com');
INSERT INTO Students (student_id, first_name, last_name, age, email)
VALUES (3, 'David', 'Jones', 19, 'david.jones@example.com');

-- Step 3: Update a Row
UPDATE Students
SET email = 'john.doe@university.com'
WHERE student_id = 1;
```

```
-- Step 4: Delete a Row
DELETE FROM Students
WHERE student_id = 3;

-- Optional Step: Select data to verify
SELECT * FROM Students;
```

Explanation:

- **Step 1:** We create the School database and a Students table with various constraints.
- **Step 2:** We insert three rows into the table, each representing a student with specified details.
- **Step 3:** We update the email of the student with student_id = 1.
- **Step 4:** We delete the student with student_id = 3.

Expected Output of SELECT * FROM Students:

Before the deletion:

student_id	first_name	last_name	age	email
1	John	Doe	20	john.doe@example.com
2	Jane	Smith	22	jane.smith@example.com
3	David	Jones	19	david.jones@example.com

After running the deletion query:

student_id	first_name	last_name	age	email
1	John	Doe	20	john.doe@university.com

student_id	first_name	last_name	age	email
2	Jane	Smith	22	jane.smith@example.com

This concludes a practical example demonstrating SQL DDL (Data Definition Language) and DML (Data Manipulation Language) commands. Let me know if you'd like any further clarification or enhancements to this example!