**Types Of Databases**

**1. Flat Files**

* **Definition**: A simple database that stores data in plain text or binary files without any structured relationships.
* **Structure**: Data is stored in rows, with each row representing a record and columns separated by delimiters (e.g., commas in CSV files).
* **Key Features**:
  + No complex relationships.
  + Easy to use and lightweight.
  + Suitable for small datasets.
* **Use Cases**:
  + Configuration files.
  + Log files.
  + Simple data exchange between systems.

**2. Relational Databases (RDBMS)**

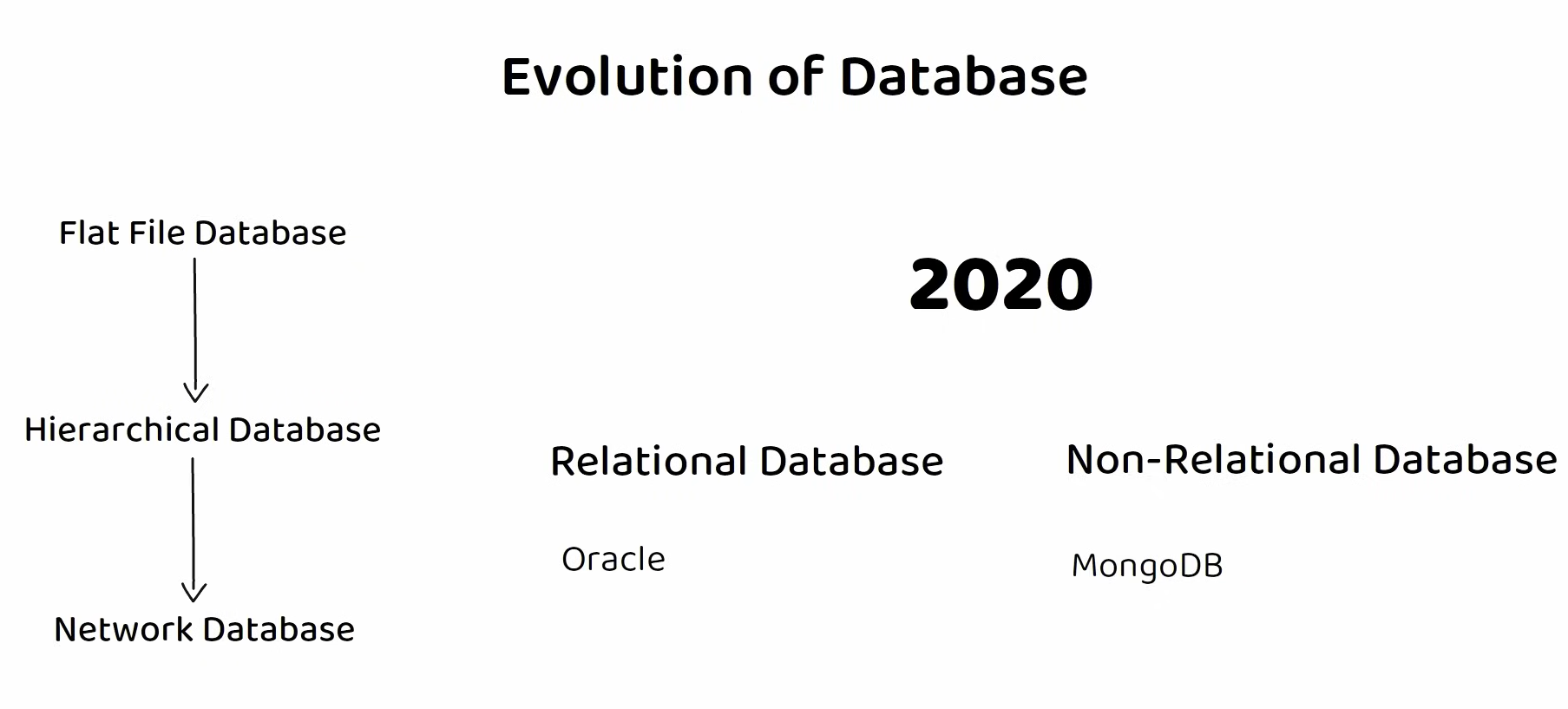
* **Definition**: A structured database where data is stored in tables (rows and columns) with predefined relationships between them.
* **Structure**: Tables use **keys** (primary and foreign) to establish relationships.
* **Key Features**:
  + Follows **ACID properties** for data integrity.
  + Supports SQL (Structured Query Language) for queries.
  + Data normalization to reduce redundancy.
* **Popular Examples**:
  + MySQL
  + PostgreSQL
  + Microsoft SQL Server
* **Use Cases**:
  + Banking and financial systems.
  + Enterprise applications.
  + E-commerce platforms.

**3. Object-Relational Databases (ORDBMS)**

* **Definition**: A hybrid database system that combines features of relational databases and object-oriented programming.
* **Structure**: Allows storage of complex data types such as objects, classes, and inheritance, in addition to tables and rows.
* **Key Features**:
  + Extended SQL capabilities to support objects.
  + Supports user-defined data types, methods, and inheritance.
  + Provides better performance for handling multimedia and complex datasets.
* **Popular Examples**:
  + Oracle Database (with object extensions).
  + PostgreSQL (with object features).
* **Use Cases**:
  + GIS (Geographic Information Systems).
  + Multimedia databases.
  + Applications needing both structured and object-oriented data handling.

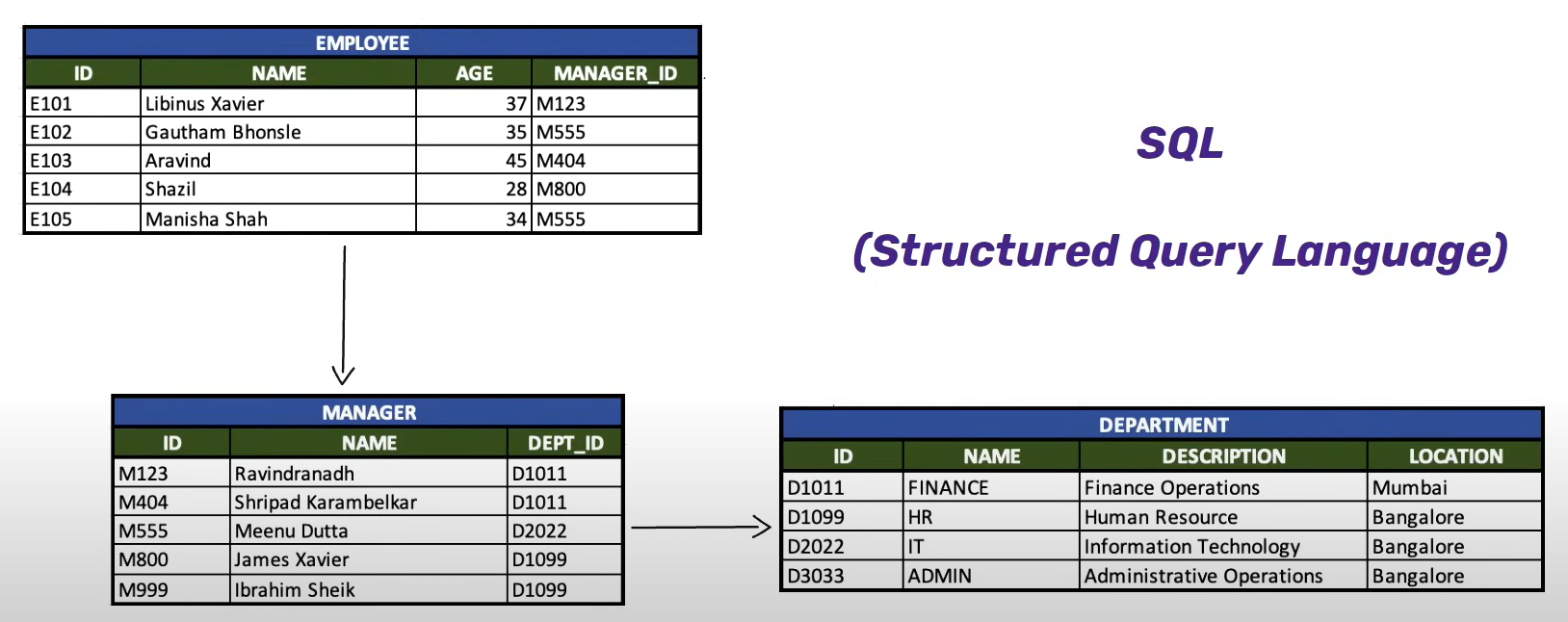
**4. Web-Enabled Databases (Online Databases)**

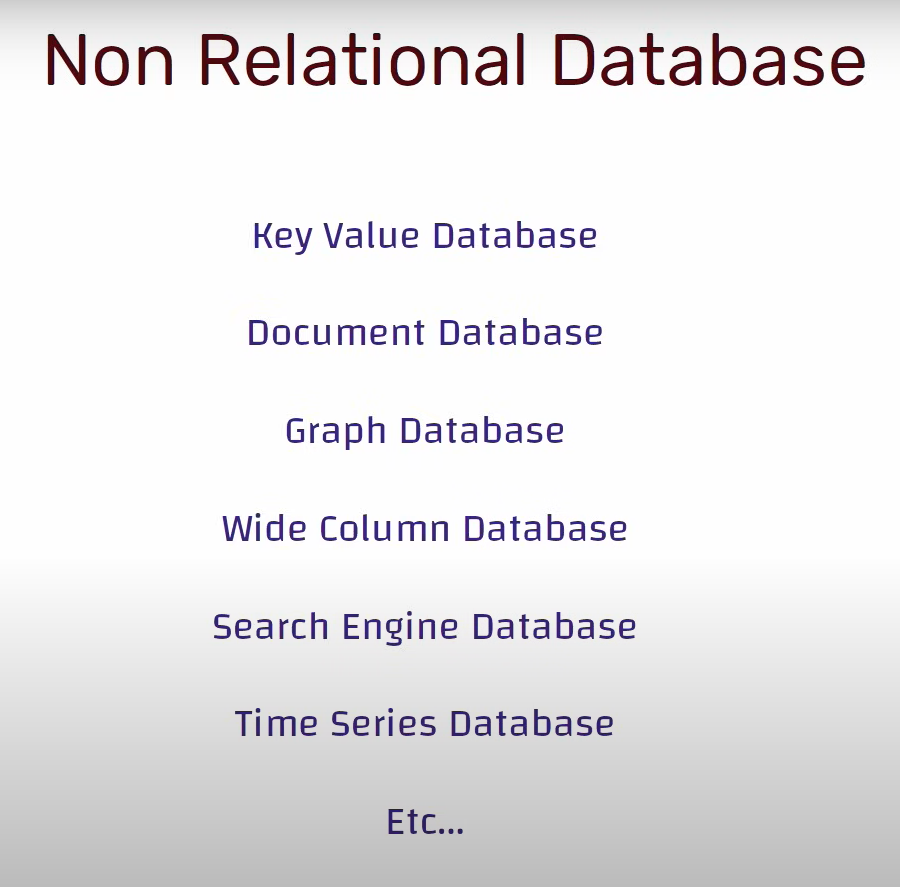
* **Definition**: Databases accessible over the internet, often used in web applications to dynamically store and retrieve data.
* **Structure**: Typically relational or NoSQL databases hosted on servers and accessed via APIs or web interfaces.
* **Key Features**:
  + Scalable and highly available.
  + Real-time access to data over the web.
  + Can integrate with web applications for dynamic content.
* **Popular Examples**:
  + Firebase Realtime Database.
  + Amazon DynamoDB.
  + MongoDB Atlas.
* **Use Cases**:
  + Social media platforms.
  + Online shopping sites.
  + Cloud-based applications.



A table with numbers and letters

Description automatically generated with medium confidence





A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

**Benefits of a Database Management System (DBMS)**

1. **Minimal Data Redundancy**:
   * Reduces duplication of data across the system.
   * Data is stored in a structured and centralized way, eliminating unnecessary copies.
   * Ensures efficient use of storage space.
2. **Consistency of Data**:
   * Ensures that any changes in data are reflected throughout the database.
   * Maintains integrity across multiple accesses or modifications.
   * Helps avoid data anomalies (insertion, update, and deletion anomalies).
3. **Integration of Data**:
   * Combines data from different sources into a unified database structure.
   * Facilitates relationships between tables for better organization and processing.
   * Enables comprehensive analysis and reporting by linking diverse data.
4. **Sharing of Data**:
   * Provides controlled access to multiple users simultaneously.
   * Facilitates collaboration by allowing users to access shared data based on roles or permissions.
   * Enhances productivity in multi-user environments.
5. **Ease of Application Development**:
   * Offers standardized query languages (like SQL) for easy data manipulation.
   * Provides tools for creating and managing applications, reducing development time.
   * Allows developers to focus on business logic rather than managing data storage.
6. **Uniform Security, Privacy, and Integrity Controls**:
   * Implements centralized security policies to safeguard data.
   * Protects sensitive information through user authentication, encryption, and access controls.
   * Enforces rules to maintain data accuracy and integrity (e.g., constraints, triggers).
7. **Data Accessibility and Responsiveness**:
   * Allows quick access to data through indexing and query optimization.
   * Ensures users can retrieve and manipulate data efficiently.
   * Supports high-performance requirements for modern applications.
8. **Reduced Program Maintenance**:
   * Centralized data management minimizes the need for updates across multiple applications.
   * Database schema changes have minimal impact on application code.
   * Reduces costs and time spent on maintaining programs.
9. **Data Independence**:
   * Separates data structure (schema) from application logic.
   * Applications remain unaffected by changes in the physical data storage or structure.
   * Ensures flexibility and adaptability in evolving systems.

**Limitations of a File-Based Approach**

**1. Separation and Isolation of Data**

* **What it means**: Each program maintains its own set of data in separate files. These programs don't "talk" to each other, meaning data in one program can't easily be accessed by another.
* **Example**: Imagine a retail store where the inventory is stored in one file (used by the warehouse team) and sales transactions are stored in another file (used by the cashier system). If a product is sold, the warehouse file may not automatically update to reflect the new inventory level. This separation makes data management inefficient.
* **Problem**: Lack of integration leads to duplicate efforts and missed opportunities to use all available data.

**2. Duplication of Data**

* **What it means**: The same data is stored in multiple places, leading to redundancy.
* **Example**: In a university, student information (like name and ID) might be stored separately by the admissions office, the library system, and the sports club. If a student changes their phone number, it has to be updated in all these places. Failing to update all files creates inconsistencies.
* **Problem**: Wasted storage space and the risk of conflicting data (e.g., different phone numbers for the same student in different systems).

**3. Atomicity of Updates**

* **What it means**: Transactions or updates in a file-based system are not guaranteed to complete entirely. If something goes wrong midway, the system is left in an incomplete or inconsistent state.
* **Example**: Consider transferring money between two bank accounts. The program deducts $500 from Account A and adds it to Account B. If the system crashes after deducting but before adding the amount, Account B won’t reflect the deposit. The bank records are now inconsistent.
* **Problem**: This lack of transaction control can lead to financial errors or operational inefficiencies.

**4. Data Dependence**

* **What it means**: The structure of the data is tightly coupled with the program code. If the data structure changes, the programs must be rewritten to accommodate the changes.
* **Example**: Suppose a hospital has patient records stored in a file format with fields like Name, Age, and Disease. If the hospital wants to add a new field like "Blood Type," all existing programs accessing these files must be updated.
* **Problem**: Making such changes is time-consuming, costly, and error-prone.

**5. Incompatible File Formats**

* **What it means**: Files created by one program may not be accessible to another program if they use different formats or programming languages.
* **Example**: A company’s HR system might use Excel spreadsheets, while the payroll system uses a custom text-based format. Integrating the two systems to calculate employee salaries becomes difficult.
* **Problem**: Sharing data between programs becomes inefficient and may require additional manual steps or conversion tools.

**6. Integrity Problems**

* **What it means**: Rules that ensure data accuracy (like “account balance must always be greater than zero”) are embedded in the program code instead of being enforced by the data storage system itself.
* **Example**: A supermarket tracks product quantities. If someone accidentally enters a negative number in the stock field, the program doesn’t automatically flag this error because the integrity check isn’t built into the file.
* **Problem**: Adding or updating these rules requires modifying all related programs, which is tedious and prone to errors.

**7. Fixed Queries and Proliferation of Application Programs**

* **What it means**: Each file-based program is created to handle specific tasks. Any new requirement (like a new type of report or query) means writing a new program from scratch.
* **Example**: A logistics company wants to generate a report showing the number of deliveries made by each driver in the past month. If the current system doesn’t already have this functionality, a new program must be written to fetch and format the data.
* **Problem**: This leads to a growing number of programs, making the system harder to maintain and manage.

**8. Data Redundancy and Inconsistency**

* **What it means**: Storing duplicate data in different files can lead to conflicting values.
* **Example**: A car dealership stores customer information separately for sales and service departments. If a customer updates their address with the sales team, but the service team doesn’t receive the update, two different addresses may exist for the same customer.
* **Problem**: Redundancy wastes storage and creates confusion when conflicting data exists.

**9. Difficulty in Accessing Data**

* **What it means**: Retrieving specific or new data requires custom programs to be written.
* **Example**: A company stores employee attendance records in a flat file. If the HR team wants to check which employees were absent on specific dates, a new program must be created to process the data and extract the information.
* **Problem**: Writing new programs for every query is inefficient and increases dependency on developers.

**Schema**

Schema refers to the structure and design of the database. It is also known as the metadate. It is similar/analogous to the data types and variables in programming languages.

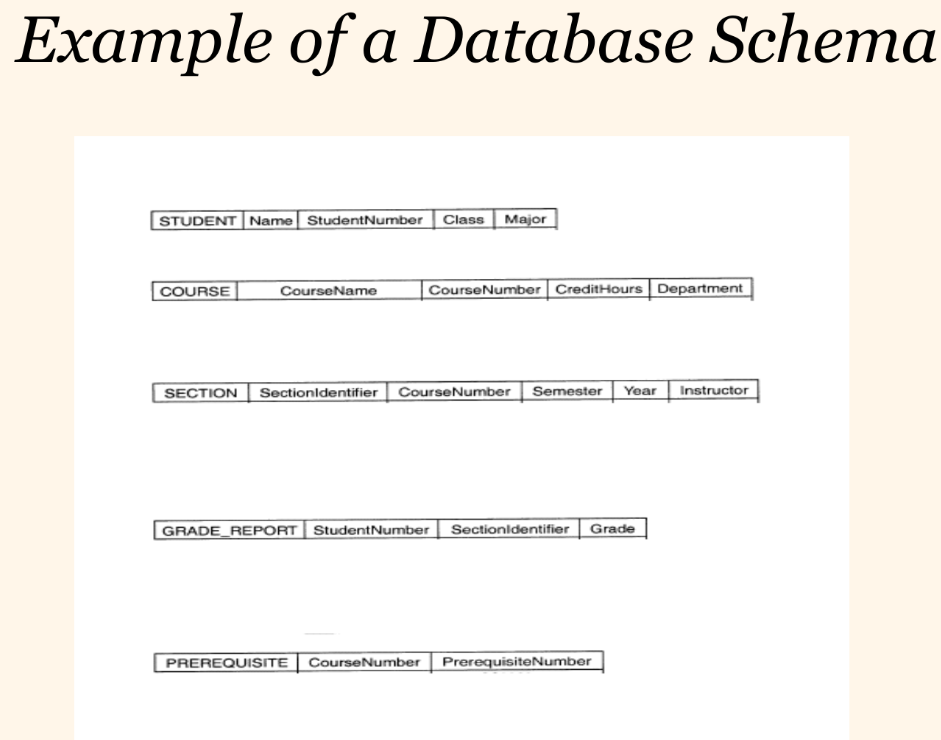
* **Physical schema:** Database design at the physical level.
* **Logical schema:** Database design at the logical level/application programs.
* **Sub-Schema:** Database design at the external level. There may be several of these with each having a different view of database.

**Instance**

Instance refers to the actual content of the database at a particular point in time. It is similar/analogous to the data/content/value stored in the variable.

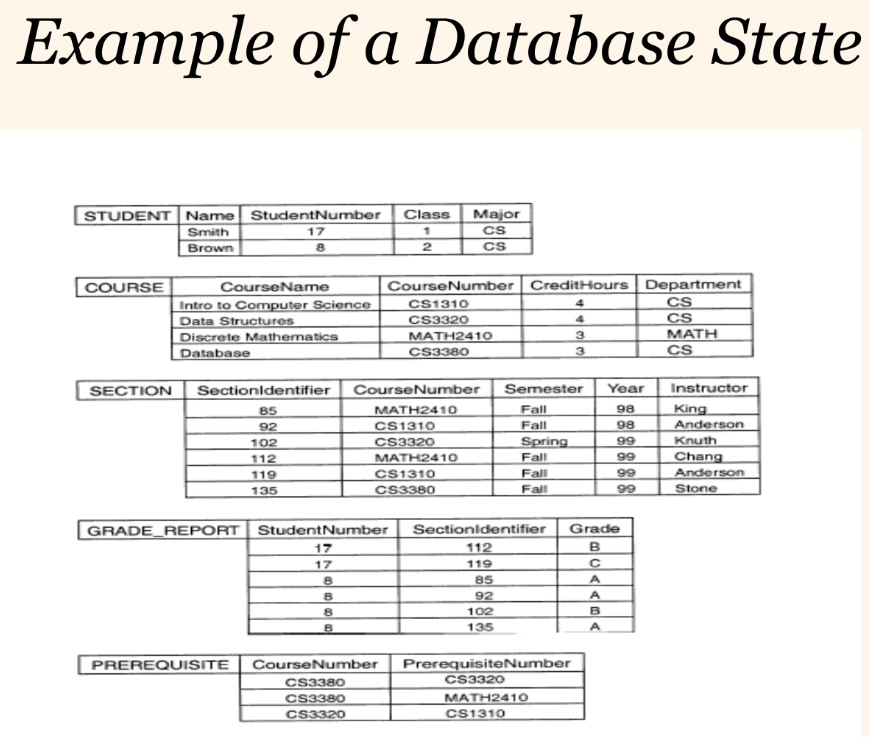
**Database Schema**

* Is the description of a database.
* It is specified during the database design and is not expected to change frequently.
* It is represented as a diagram called schema diagram.
* A schema diagram displays the structure of each record type but not the actual instance of a record.
* Each object in a schema is called a schema construct.



**Database State/Instance**

* A Database state or instance is the data in the database at a particular moment of time.
* Every update operation changes the database from one state to another.
* The Schema is sometimes called the **intension**, and the database state an **extension** of the **schema**.



**Data Independence**

* Physical representation and location of data and the use of that data are separated
* The application doesn’t need to know how or where the database has stored the data, but just how to ask for it
* Moving a database from one DBMS to another should not have a material effect on application program
* Recording, adding fields, etc. in the database should not affect applications

**Data Independence**

Data independence refers to the ability to change the structure of the database or storage system without affecting how users interact with it.

Changes in database doesn’t affect the programs using it. Meaning, we don’t have to modify and update each and every program that is using that database every time there is an update in the database structure i.e. addition of a new field.

**1. Logical Data Independence:** Ability to modify the conceptual schema without affecting the external schema or application programs i.e. adding a new relation or a new field in a relation.

**2. Physical Data Independence:** Ability to modify the internal schema (the physical storage of data on hardware) without affecting the conceptual schema i.e. moving the data to a new DBMS.

Physical data independence deals with the internal schema. It is the ability to modify the physical schema without causing application programs to be rewritten. Examples include changing the compression techniques, hashing algorithms, SSD, location of the database.

**Database Languages**

**1. Data Definition Language (DDL)**

DDL is used to define and manage the structure of database objects like tables, views, indexes, and schemas.

* Integrity Constraints: DDL supports defining constraints such as PRIMARY KEY, FOREIGN KEY, and CHECK to ensure data integrity.
* Security Constraints: Commands like GRANT and REVOKE manage user permissions.

**2. Data Manipulation Language (DML)**

DML enables users to query and manipulate the data stored in the database.

* **Data Retrieval:** Queries using **SELECT** statements to fetch data.
* **Data Insertion:** The **INSERT** command adds new records to the database.
* **Data Update:** The **UPDATE** command modifies existing records.
* **Data Deletion:** The **DELETE** command removes records from the database.

**3. Procedural DML**

Procedural DML requires the user to specify the exact sequence of operations that the database must perform to retrieve or manipulate the desired data.

* Provides complete control over how operations are executed.
* Often embedded in programming languages (like PL/SQL for Oracle).
* Suitable for complex operations involving loops, conditionals, and iterative logic.

**4. Non-Procedural DML**

Non-Procedural DML allows the user to specify *what data* is required without dictating *how* to obtain it.

* Higher-level abstraction compared to procedural DML.
* Focuses on what to do, not how to do it.
* Easier to use for simple queries and operations.
* Typically implemented in Structured Query Language (SQL).

**Three-Level Architecture**

**External Level**

* User’s view of the database
* Describes that part of database that is relevant to a particular user

**Conceptual Level**

* Community view of the database
* Describes what data is stored in database and relationships among the data

**Internal Level**

* Physical representation of the database on the computer
* Describes how the data is stored in the database

**Objectives of Three-Level Architecture**

* **Access Control:** All users should be able to access their own personalized data
* The user’s view is immune to change made in other views
* **Data Abstraction:** User don’t not need to know physical database storage details
* **Data Independence:** Database Administrator (DBA) should be able to change database storage structures without affecting the user’s views
* Internal structure of database should be unaffected by changes to physical aspects of storage (Physical Data Independence)
* DBA should be able to change conceptual structure (Logical Data Independence) of database without affecting all users