**DBMS Lec#01**

**🡪 What is a Database?**

A database is an organized collection of data that can be accessed, managed, and updated.

* Example: A database can store student records, employee data, product inventory, etc.

**🡪 What is a DBMS?**

A **Database Management System (DBMS)** is software that allows users to create, manage, and interact with databases efficiently.

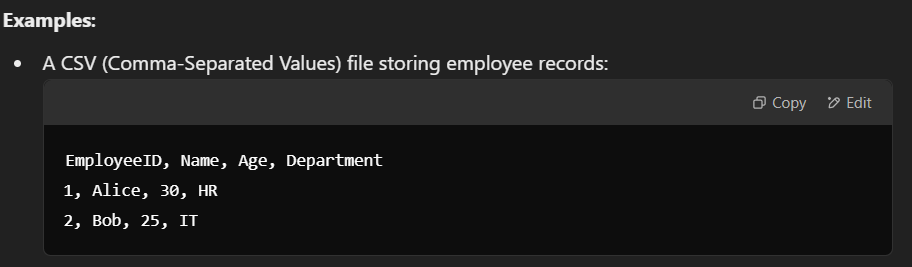
**🡪 Types of DATABASES:**

**1. Flat Files**

A flat file database is the simplest form of a database, often stored as a plain text file or a spreadsheet.

**Key Characteristics:**

* Data is stored in a **single table**.
* There are no relationships between different data entities.
* Suitable for small datasets or simple tasks.



**2. Relational Databases**

A **relational database** organizes data into **tables** (relations) with rows (records) and columns (fields). Data in different tables can be related using **keys** (e.g., primary key, foreign key).

**Examples**: MySQL, Oracle, SQL Server, PostgreSQL.

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

* Definition: An object-relational database combines features of both relational databases and object-oriented databases.
* Structure: Extends relational database capabilities with support for object-oriented features like inheritance, polymorphism, and complex data types.
* Examples: PostgreSQL, Oracle with object extensions.

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

* Definition: Web-enabled databases are designed to be accessed over the internet through web applications.
* Structure: Can be relational or non-relational and are often hosted in cloud environments.
* Examples: Firebase, MongoDB Atlas, Amazon RDS.

**🡪 Limitations of File-based Approach**

1. **Separation and Isolation of Data**

* Each program maintains its own set of data. Users of one program may be unaware of potentially useful data held by other programs.
* Imagine each department in a company keeps its data in separate locked drawers, and they don’t share with each other.
* For example, the Sales team has customer contact info, but the Support team doesn’t know about it, even though it could help them solve issues.
* Result: Useful data is hidden, and people have no idea it exists.

**2. Duplication of Data**

* Same data is held by different programs. Wasted space and potentially different values and/or different formats for the same item.
* Think of multiple people in the same office keeping their own lists of the same customers. One list says "John" lives in City A, while another says "John" lives in City B.
* This wastes storage space and causes confusion because the data is inconsistent.

**3. Atomicity of Updates**

* Failure may lead database to an inconsistent state with partial updates carried out
* Picture transferring money from one wallet to another. If something goes wrong halfway (like the app crashing), some money may leave the first wallet but never reach the second. The wallets are in an "in-between" state.
* It’s critical that the transfer either fully happens or doesn’t happen at all.

**4. Data Dependence**

* File structure is defined in the program code.
* Imagine your documents can only be opened with a specific software version. If you change the software, the document can no longer be read because it depends on how it was written initially.
* Similarly, in file systems, if you change the structure of how data is stored, the old programs can’t read it.

**5. Incompatible File Formats**

* Programs are written in different languages, and so cannot easily access each other’s files.
* If one person writes a letter in English and another writes in Spanish, they won’t understand each other.
* In file systems, different programs may store data in formats that other programs can’t easily read or use.

**6. Integrity Problems**

* Let’s say a rule states that an account balance can’t go below zero, but this rule isn’t clearly enforced. Someone might try to withdraw more money than is in the account, causing errors.
* In file systems, these rules aren’t built-in, making it harder to enforce them consistently.

**7. Fixed Queries / Need for Many Programs**

* Programs are written to satisfy particular functions. Any new requirement needs a new program
* Imagine you have a calculator that can only add numbers. If you want it to multiply or divide, you’d need to create a new tool.
* Similarly, in file systems, every time you need to perform a new task (like a new type of report), you have to write a whole new program.

**8. Data Redundancy and Inconsistency**

* Think of having the same contact saved multiple times on your phone with different spellings. It takes up extra space and creates confusion when you try to contact them.
* File systems often store the same data in multiple files, leading to inconsistency and wasted storage.

**9. Difficulty in Accessing Data**

* Imagine you have to dig through multiple folders to find the one piece of information you need. It’s frustrating and time-consuming.
* In file systems, if you want new data, you often need to write a new program, which is slow and inefficient.

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

**1. Minimal Data Redundancy**

* What it means: Data is stored only once and shared across the system, avoiding duplication.
* Example: Instead of saving customer details in separate files for Sales and Support teams, all departments use a single database.
* Why it’s good: Saves storage space and prevents confusion caused by having multiple versions of the same data.

**2. Consistency of Data**

* What it means: Since there’s only one version of the data, everyone sees the same, accurate information.
* Example: If a customer’s phone number is updated, it’s reflected everywhere instantly.
* Why it’s good: Eliminates errors caused by outdated or conflicting data.

**3. Integration of Data**

* What it means: All data is stored in one place and connected, making it easier to find relationships between different types of information.
* Example: Linking sales data with customer data to generate reports on customer purchase history.
* Why it’s good: Makes it easy to combine and analyze data from different sources.

**4. Sharing of Data**

* What it means: Multiple users and departments can access the database simultaneously.
* Example: A marketing team uses the database to track customer preferences while the sales team uses it to check recent purchases.
* Why it’s good: Promotes collaboration and better decision-making.

**5. Ease of Application Development**

* What it means: Since the database handles data storage, retrieval, and security, developers can focus on building useful applications without worrying about these details.
* Example: A developer can quickly create an app to track employee attendance by using the database for storing data.
* Why it’s good: Saves time and effort in creating and maintaining applications.

**6. Uniform Security, Privacy, and Integrity Controls**

* What it means: DBMS has built-in tools to enforce security rules, protect sensitive data, and ensure the data follows certain rules.
* Example: A rule in the database can ensure only managers can view employee salaries.
* Why it’s good: Prevents unauthorized access and ensures sensitive data is handled correctly.

**7. Data Accessibility and Responsiveness**

* What it means: Data can be quickly accessed and retrieved using queries (like SQL).
* Example: You can instantly find all customers who made purchases in the last month.
* Why it’s good: Saves time and provides fast answers for decision-making.

**8. Reduced Program Maintenance**

* What it means: Since the database manages data centrally, there’s less need to rewrite or update programs when changes are made.
* Example: If you add a new field to the customer table (like “Birthday”), you don’t need to change every program accessing it.
* Why it’s good: Makes updates easier and faster, saving effort in the long run.

**9. Data Independence**

* What it means: Changes to how data is stored (internally) don’t affect how programs access the data.
* Example: If you switch from storing data as text files to binary format, the programs using the database don’t need to be rewritten.
* Why it’s good: Provides flexibility and reduces dependency on specific storage methods.

**🡪 Tiers in a System:**

When we talk about **tiers** in a system, it’s about how work is divided into **layers** to keep things organized and efficient. Think of it like building a house where each floor has a specific purpose. Here’s how it applies in software systems:

**1. Single-Tier (1-Tier) Architecture**

* **What it is:** Everything happens in one place (like a small shop where one person does everything—takes orders, makes food, and serves it).
* **Example:** A simple desktop application where the program and data are on the same computer.
* **Why it’s used:** Easy to set up but not good for collaboration or large-scale tasks.

**2. Two-Tier (2-Tier) Architecture**

* **What it is:** Work is split into **two parts**:
  1. A user interface (front end) where you interact with the system.
  2. A database (back end) where the data is stored.
* **Example:** A banking application where you use a desktop app to access your account, and the bank’s database stores your transactions.
* **Why it’s used:** Good for small-scale systems, but it struggles with lots of users at once.

**3. Three-Tier (3-Tier) Architecture**

* **What it is:** Work is divided into **three parts**:
  1. **Presentation Layer (Front End):** What users see and interact with (e.g., a website or app).
  2. **Application Layer (Logic):** Where the business rules and processing happen (e.g., calculating prices or verifying passwords).
  3. **Database Layer (Back End):** Where all data is stored.
* **Example:** Online shopping platforms like Amazon, where the app shows products, processes your orders, and stores the data in a database.
* **Why it’s used:** More scalable and flexible, ideal for large systems with many users.

**🡪 3 Schema Architecture**

**1. External Level (View Level)**

* **Purpose**: This level is closest to the users and describes how different users or applications view the data.
* **Features**:
  + Multiple external schemas (views) can exist for different user groups.
  + Each view shows only the relevant data to the user, hiding the rest of the database.
  + Provides **logical data independence** (changes in the logical structure of the database do not affect user views).
* **Example**: A student database might have one view for students (showing grades) and another for faculty (showing attendance and performance).

**2. Conceptual Level (Logical Level)**

* **Purpose**: This level describes the overall structure of the entire database for a community of users.
* **Features**:
  + Represents the global view of the database.
  + Defines entities, relationships, constraints, and data types.
  + Acts as a bridge between the external and internal levels.
  + Provides **physical data independence** (changes in the physical storage do not affect the conceptual schema).
* **Example**: A conceptual schema might define tables like Students, Courses, and Enrollments, along with their relationships.

**3. Internal Level (Physical Level)**

* **Purpose**: This level describes how the data is physically stored in the database.
* **Features**:
  + Deals with storage structures, file organization, indexing, and access methods.
  + Focuses on performance optimization and efficient data retrieval.
  + Describes data storage in terms of bytes, blocks, and pointers.
* **Example**: The internal schema might specify how a Students table is stored on disk, including indexes and compression technique

**Key Benefits of the Three-Schema Architecture:**

1. **Data Independence**:
   * **Logical data independence:** Changes in the conceptual schema do not affect external views.
   * **Physical data independence**: Changes in the internal schema do not affect the conceptual schema.