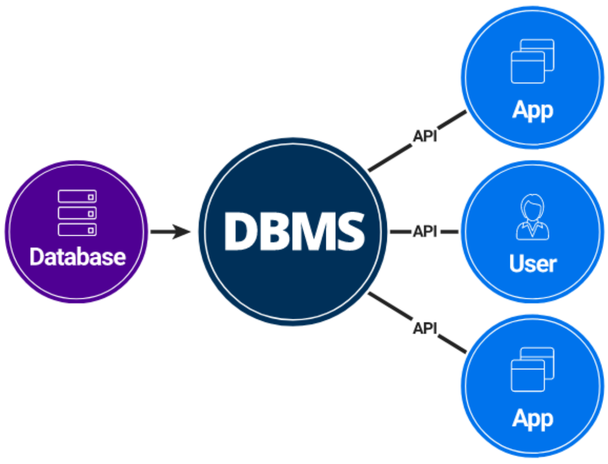
LEC-1: Introduction to DBMS

1. **What is Data?**
   a. Data is a collection of raw, unorganized facts and details like text, observations, figures, symbols, and descriptions of things etc.
   In other words, **data does not carry any specific purpose and has no significance by itself.**
   Moreover, data is measured in terms of bits and bytes – which are basic units of information in the context of computer storage and processing.
   b. Data can be recorded and doesn't have any meaning unless processed.
2. **Types of Data**
   a. Quantitative
      i. Numerical form
      ii. Weight, volume, cost of an item.
   b. Qualitative
      i. Descriptive, but not numerical.
      ii. Name, gender, hair color of a person.
3. What is **Information**?
   a. Info. Is **processed, organized, and structured data.**
   b. It provides **context of the data and enables decision making.**
   c. Processed data that make **sense** to us.
   d. Information is extracted from the data, by **analyzing and interpreting** pieces of data.
   e. E.g., you have data of all the people living in your locality, its Data, when you analyze and interpret the data and come to some conclusion that:
      i. There are 100 senior citizens.
      ii. The sex ratio is 1.1.
      iii. Newborn babies are 100.
      These are information.
4. **Data vs Information**
   a. Data is a collection of facts, while information puts those facts into context.
   b. While data is raw and unorganized, information is organized.
   c. Data points are individual and sometimes unrelated. Information maps out that data to provide a big-picture view of how it all fits together.
   d. Data, on its own, is meaningless. When it's analyzed and interpreted, it becomes meaningful information.
   e. Data does not depend on information; however, information depends on data.
   f. Data typically comes in the form of graphs, numbers, figures, or statistics. Information is typically presented through words, language, thoughts, and ideas.
   g. Data isn't sufficient for **decision-making**, but you can make decisions based on information.
5. What is **Database**?
   a. Database is an electronic place/system where data is stored in a way that it can be **easily accessed, managed, and updated.**
   b. To make real use Data, we need **Database management systems. (DBMS)**
6. What is **DBMS**?
   a. A database-management system (DBMS) is a collection of **interrelated data** and **a set of programs to access those data**. The collection of data, usually referred to as the **database**, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to **store and retrieve database information** that is both convenient and efficient.
   b. A DBMS is the database itself, along with all the software and functionality. It is used to perform different operations, like **addition**, **access**, **updating**, and **deletion** of the data.

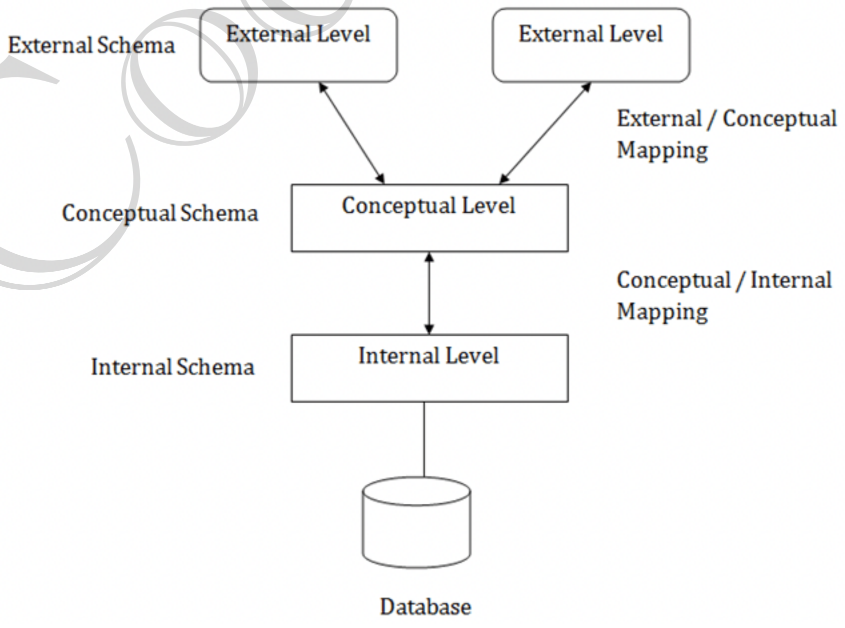


7.

8. **DBMS vs File Systems**
    a. **File-processing systems** has major **disadvantages**.
        i. Data Redundancy and inconsistency
        ii. Difficulty in accessing data
        iii. Data isolation
        iv. Integrity problems
        v. Atomicity problems
        vi. Concurrent-access anomalies
        vii. Security problems
    b. Above 7 are also the **Advantages of DBMS** (answer to "**Why to use DBMS?**")

LEC-2: DBMS Architecture

1. **View of Data (Three Schema Architecture)**
    a. The major purpose of DBMS is to provide users with an **abstract view** of the data. That is, the system hides certain details of how the data is stored and maintained.
    b. To simplify user interaction with the system, abstraction is applied through several levels of abstraction.
    c. The **main objective** of three level architecture is to enable multiple users to access the same data with a personalized view while storing the underlying data only once
    d. **Physical level / Internal level**
        i. The lowest level of abstraction describes how the data are stored.
        ii. Low-level data structures used.
        iii. It has **Physical schema** which describes physical storage structure of DB.
        iv. Talks about: Storage allocation (N-ary tree etc), Data compression & encryption etc.
        v. **Goal**: We must define algorithms that allow efficient access to data.
    e. **Logical level / Conceptual level:**
        i. The **conceptual schema** describes the design of a database at the conceptual level, describes ***what*** data are stored in DB, and what ***relationships*** exist among those data.
        ii. User at logical level does not need to be aware about physical-level structures.
        iii. **DBA**, who must decide what information to keep in the DB use the logical level of abstraction.
        iv. **Goal**: ease to use.
    f. **View level / External level:**
        i. Highest level of abstraction aims to simplify users' interaction with the system by providing different view to different **end**-user.
        ii. Each **view schema** describes the database part that a particular user group is interested and hides the remaining database from that user group.
        iii. At the external level, a database contains several schemas that sometimes called as **subschema**. The subschema is used to describe the different view of the database.
        iv. At views also provide a **security** mechanism to prevent users from accessing certain parts of DB.



    g.

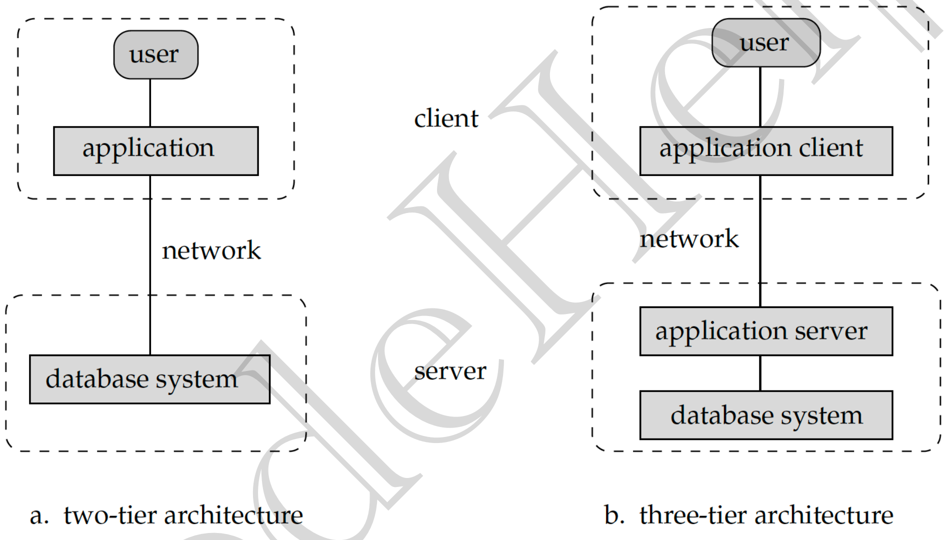
2. **Instances and Schemas**
    a. The collection of information stored in the DB at a particular moment is called an **instance** of DB.

b. The overall design of the DB is called the DB **schema**.
c. Schema is **structural** description of data. Schema **doesn't change frequently**. Data may change frequently.
d. **DB schema** corresponds to the variable declarations (along with type) in a program.
e. We have 3 types of **Schemas**: **Physical, Logical**, several **view schemas** called subschemas.
f. Logical schema is most **important** in terms of its effect on application programs, as programmers construct apps by using logical schema.
g. **Physical data independence**, physical schema change should not affect logical schema/application programs.

3. **Data Models:**
    a. Provides a way to describe the **design** of a DB at **logical level.**
    b. Underlying the structure of the DB is the Data Model; a collection of conceptual tools for describing **data, data relationships, data semantics & consistency constraints**.
    c. E.g., **ER** model, **Relational** Model, **object-oriented** model, **object-relational** data model etc.
4. **Database Languages**:
    a. **Data definition language (DDL)** to specify the database schema.
    b. **Data manipulation language (DML)** to express database queries and updates.
    c. **Practically**, both language features are present in a single DB language, e.g., SQL language.
    d. DDL
        i. We specify consistency constraints, which must be checked, every time DB is updated.
    e. DML
        i. Data manipulation involves
            1. **Retrieval** of information stored in DB.
            2. **Insertion** of new information into DB.
            3. **Deletion** of information from the DB.
            4. **Updating** existing information stored in DB.
        ii. **Query language**, a part of DML to specify statement requesting the retrieval of information.
5. **How is Database accessed from Application programs?**
    a. Apps (written in host languages, C/C++, Java) interacts with DB.
    b. E.g., Banking system's module generating payrolls access DB by executing DML statements from the host language.
    c. API is provided to send DML/DDL statements to DB and retrieve the results.
        i. Open Database Connectivity (**ODBC**), Microsoft "C".
        ii. Java Database Connectivity (**JDBC**), Java.
6. **Database Administrator (DBA)**
    a. A person who has **central control** of both the data and the programs that access those data.
    b. **Functions** of DBA
        i. Schema Definition
        ii. Storage structure and access methods.
        iii. Schema and physical organization modifications.
        iv. Authorization control.
        v. Routine maintenance
            1. Periodic backups.
            2. Security patches.
            3. Any upgrades.
7. **DBMS Application Architectures**: Client machines, on which remote DB users work, and server machines on which DB system runs.
    a. **T1** Architecture
        i. The client, server & DB all present on the same machine.

b. **T2** Architecture
   i. App is partitioned into 2-components.
   ii. Client machine, which invokes DB system functionality at server end through query language statements.
   iii. API standards like **ODBC** & **JDBC** are used to interact between client and server.

c. **T3** Architecture
   i. App is partitioned into 3 logical components.
   ii. Client machine is just a frontend and doesn't contain any direct DB calls.
   iii. Client machine communicates with App server, and App server communicated with DB system to access data.
   iv. **Business** logic, what action to take at that condition is in App server itself.
   v. T3 architecture are best for **WWW** Applications.
   vi. **Advantages**:
      1. **Scalability** due to distributed application servers.
      2. **Data integrity**, App server acts as a middle layer between client and DB, which minimize the chances of data corruption.
      3. **Security**, client can't directly access DB, hence it is more secure.



LEC-3: Entity-Relationship Model

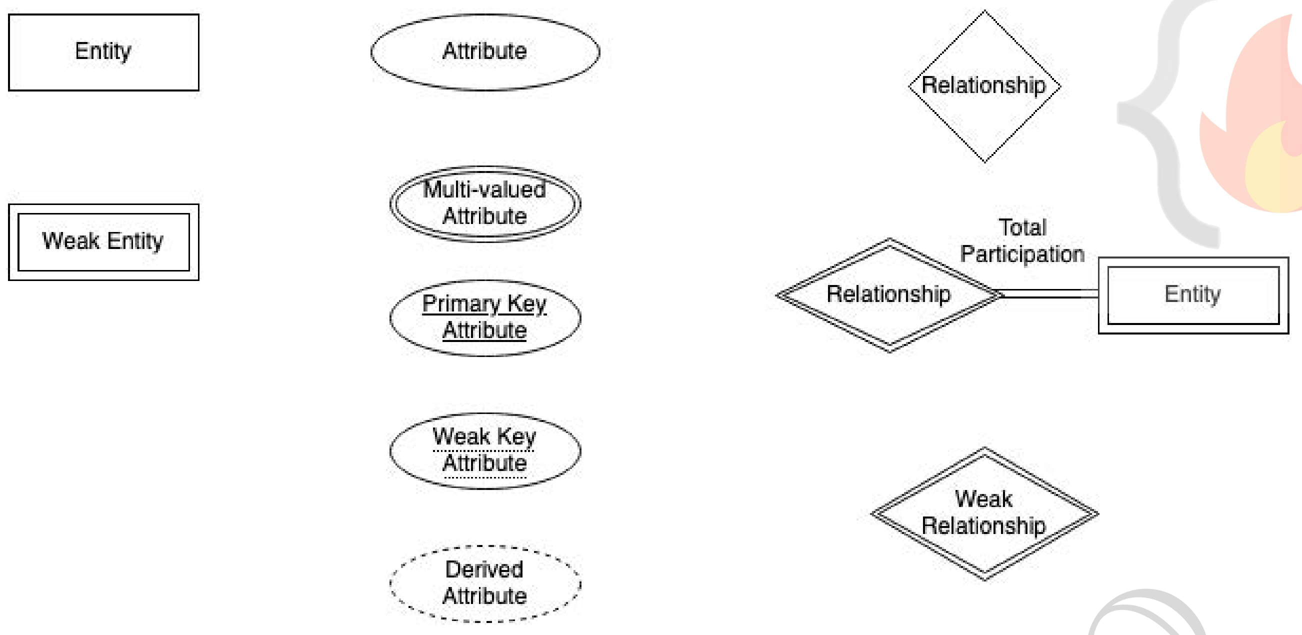
1. **Data Model**: Collection of conceptual tools for **describing data, data relationships, data semantics, and consistency constraints.**
2. **ER Model**
   1. It is a high level data model based on a perception of a **real** world that consists of a collection of basic objects, called **entities** and of **relationships** among these objects.
   2. Graphical representation of ER Model is **ER diagram,** which acts as a **blueprint** of DB.
3. **Entity**: An Entity is a "**thing**" or "**object**" in the real world that is **distinguishable** from all other objects.
   1. It has **physical existence**.
   2. Each student in a college is an entity.
   3. Entity can be **uniquely** identified. (By a primary attribute, aka Primary Key)
   4. **Strong Entity**: Can be uniquely identified.
   5. **Weak Entity**: Can't be uniquely identified., depends on some other strong entity.
      1. It doesn't have sufficient attributes, to select a uniquely identifiable attribute.
      2. Loan -> Strong Entity, Payment -> Weak, as instalments are sequential number counter can be generated separate for each loan.
      3. **Weak entity depends on strong entity for existence.**
4. **Entity set**
   1. It is a set of entities of the **same** type that share the **same** properties, or attributes.
   2. E.g., Student is an entity set.
   3. E.g., Customer of a bank
5. **Attributes**
   1. An entity is represented by a set of attributes.
   2. Each entity has a value for each of its attributes.
   3. For each attribute, there is a set of **permitted values**, called the **domain**, or **value** set, of that attribute.
   4. E.g., Student Entity has following attributes
      - A. Student_ID
      - B. Name
      - C. Standard
      - D. Course
      - E. Batch
      - F. Contact number
      - G. Address
   5. **Types of Attributes**
      1. **Simple**
         1. Attributes which can't be divided further.
         2. E.g., Customer's account number in a bank, Student's Roll number etc.
      2. **Composite**
         1. Can be divided into subparts (that is, other attributes).
         2. E.g., Name of a person, can be divided into first-name, middle-name, last-name.
         3. If user wants to refer to an entire attribute or to only a component of the attribute.
         4. Address can also be divided, street, city, state, PIN code.
      3. **Single-valued**
         1. Only one value attribute.
         2. e.g., Student ID, loan-number for a loan.
      4. **Multi-valued**
         1. Attribute having more than one value.
         2. e.g., phone-number, nominee-name on some insurance, dependent-name etc.
         3. Limit constraint may be applied, upper or lower limits.
      5. **Derived**
         1. Value of this type of attribute can be derived from the value of other related attributes.

        2. e.g., Age, loan-age, membership-period etc.

    6. **NULL Value**
        1. An attribute takes a null value when an entity does not have a value for it.
        2. It may indicate "not applicable", value doesn't exist. e.g., person having no middle-name
        3. It may indicate "unknown".
            1. Unknown can indicate missing entry, e.g., name value of a customer is NULL, means it is missing as name must have some value.
            2. Not known, salary attribute value of an employee is null, means it is not known yet.

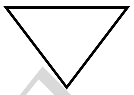
6. **Relationships**
    1. **Association** among two or more entities.
    2. e.g., Person has vehicle, Parent has Child, Customer borrow loan etc.
    3. **Strong Relationship**, between two independent entities.
    4. **Weak Relationship**, between weak entity and its owner/strong entity.
        1. e.g., Loan <instalment-payments> Payment.
    5. **Degree of Relationship**
        1. Number of entities participating in a relationship.
        2. **Unary**, Only one entity participates. e.g., Employee manages employee.
        3. **Binary**, two entities participates. e.g., Student takes Course.
        4. **Ternary** relationship, three entities participates. E.g, Employee works-on branch, employee works-on job.
        5. Binary are **common**.
7. **Relationships Constraints**
    1. **Mapping Cardinality / Cardinality Ratio**
        1. Number of entities to which another entity can be associated via a relationship.
        2. **One to one,** Entity in A associates with at most one entity in B, where A & B are entity sets. And an entity of B is associated with at most one entity of A.
            1. E.g., Citizen has Aadhar Card.
        3. **One to many,** Entity in A associated with N entity in B. While entity in B is associated with at most one entity in A.
            1. e.g., Citizen has Vehicle.
        4. **Many to one,** Entity in A associated with at most one entity in B. While entity in B can be associated with N entity in A.
            1. e.g., Course taken by Professor.
        5. **Many to many,** Entity in A associated with N entity in B. While entity in B also associated with N entity in A.
            1. Customer buys product.
            2. Student attend course.
    2. **Participation Constraints**
        1. Aka, **Minimum cardinality constraint.**
        2. **Types**, Partial & Total Participation.
        3. **Partial Participation,** not all entities are involved in the relationship instance.
        4. **Total Participation**, each entity must be involved in at least one relationship instance.
        5. e.g., Customer borrow loan, loan has total participation as it can't exist without customer entity. And customer has partial participation.
        6. **Weak entity has total participation constraint, but strong may not have total.**
8. **ER Notations**

Symbols used in ER Diagram



LEC-4: Extended ER Features

1. **Basic ER Features** studied in the **LEC-3,** can be used to model most DB features but when complexity increases, it is better to use some Extended ER features to model the DB Schema.
2. **Specialisation**
    1. In ER model, we may require to subgroup an entity set into other entity sets that are distinct in some way with other entity sets.
    2. **Specialisation** is **splitting** up the entity set into further **sub entity sets** on the basis of their **functionalities**, **specialities** and **features**.
    3. It is a **Top-Down** approach.
    4. e.g., **Person** entity set can be divided into **customer**, **student**, **employee**. Person is **superclass** and other specialised entity sets are **subclasses**.
        1. We have **"is-a"** relationship between superclass and subclass.
        2. Depicted by **triangle** component.
    5. **Why** Specialisation?
        1. Certain attributes may only be applicable to a few entities of the parent entity set.
        2. DB designer can show the distinctive features of the sub entities.
        3. To group such entities we apply Specialisation, to overall **refine** the DB blueprint.



3. **Generalisation**
    1. It is just a **reverse** of Specialisation.
    2. DB Designer, may encounter certain properties of two entities are overlapping. Designer may consider to make a new generalised entity set. That generalised entity set will be a super class.
    3. **"is-a"** relationship is present between subclass and super class.
    4. e.g., **Car**, **Jeep** and **Bus** all have some common attributes, to avoid data repetition for the common attributes. DB designer may consider to Generalise to a new entity set "**Vehicle**".
    5. It is a **Bottom-up** approach.
    6. **Why** Generalisation?
        1. Makes DB more **refined** and **simpler**.
        2. Common attributes are not **repeated**.
4. **Attribute Inheritance**
    1. **Both** Specialisation and Generalisation, has attribute inheritance.
    2. The attributes of higher level entity sets are inherited by lower level entity sets.
    3. E.g., **Customer & Employee** inherit the attributes of **Person**.
5. **Participation Inheritance**
    1. If a parent entity set participates in a relationship then its child entity sets will also participate in that relationship.
6. **Aggregation**
    1. **How to show relationships among relationships?** - Aggregation is the technique.
    2. **Abstraction** is applied to treat relationships as higher-level entities. We can call it Abstract entity.
    3. **Avoid redundancy** by aggregating relationship as an entity set itself.