

Basic DBMS Concepts

DBMS:

Database is a collection of related data and data is a collection of facts and figures that can be processed to produce information.

Mostly data represents recordable facts. Data aids in producing information, which is based on facts. For example, if we have data about marks obtained by all students, we can then conclude about toppers and average marks.

A **database management system** stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

A **DBMS** is a software that allows creation, definition and manipulation of database. Dbms is actually a tool used to perform any kind of operation on data in database. Dbms also provides protection and security to database. It maintains data consistency in case of multiple users. Here are some examples of popular dbms, MySQL, Oracle, Sybase, Microsoft Access and IBM DB2 etc.

Why use DBMS?

- To develop software applications in less time
- Data independence and efficient use of data
- For uniform data administration
- For data integrity and security
- For concurrent access of data and data recovery from crashes
- To use user friendly declarative query language

Where DBMS is being used?/Applications of DBMS

- Airlines
- Telecom
- Universities/Education
- Sales
- Banking
- Industry
- Online Shopping

Characteristics of DBMS (Advantages)

Traditionally, data was organized in file formats. DBMS was a new concept then, and all the research was done to make it overcome the deficiencies in traditional style of data management. A modern DBMS has the following characteristics –

- **Real-world entity** – A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use students as an entity and their age as an attribute.
- **Relation-based tables** – DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the table names.
- **Isolation of data and application** – A database system is entirely different than its data. A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about data, to ease its own process.
- **Less redundancy** – DBMS follows the rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Normalization is a mathematically rich and scientific process that reduces data redundancy.
- **Consistency** – Consistency is a state where every relation in a database remains consistent. There exist methods and techniques, which can detect attempt of leaving database in inconsistent state. A DBMS can provide greater consistency as compared to earlier forms of data storing applications like file-processing systems.
- **Query Language** – DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and as different filtering options as required to retrieve a set of data. Traditionally it was not possible where file-processing system was used.
- **ACID Properties** – DBMS follows the concepts of Atomicity, Consistency, Isolation, and Durability (normally shortened as ACID). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.
- **Multiuser and Concurrent Access** – DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.
- **Multiple views** – DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.
- **Security** – Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features. For example, a user in the Sales department cannot see the data that belongs to the Purchase department. Additionally, it can also be managed how much data of the Sales department should be displayed to the user. Since

a DBMS is not saved on the disk as traditional file systems, it is very hard for miscreants to break the code.

Functions of DBMS

- Provides data Independence
- Concurrency Control
- Provides Recovery services
- Provides Utility services
- Provides a clear and logical view of the process that manipulates data.

Components of DBMS

The database system can be divided into four components.

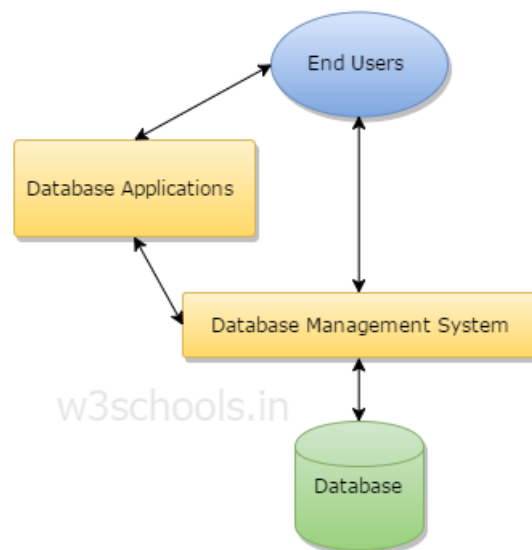
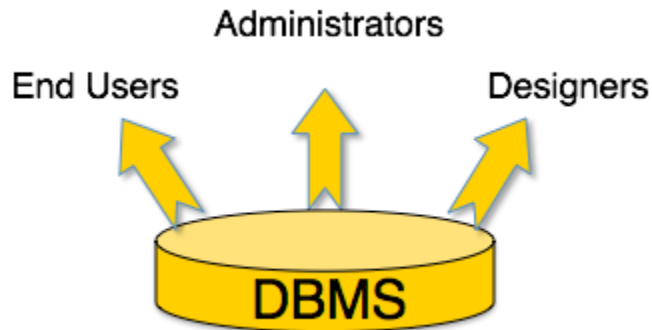


Figure: Component relationship of DBMS

- **Users:** Users may be of various types such as DB administrator, System developer and End users.
- **Database application :** Database application may be Personal, Departmental, Enterprise and Internal
- **DBMS:** Software that allow users to define, create and manages database access, Ex: MySql, Oracle etc.
- **Database:** Collection of logical data.

DBMS Users/Stakeholders:

A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows –



- **Administrators** – Administrators maintain the DBMS and are responsible for administering the database. They are responsible to look after its usage and by whom it should be used. They create access profiles for users and apply limitations to maintain isolation and force security. Administrators also look after DBMS resources like system license, required tools, and other software and hardware related maintenance.
- **Designers** – Designers are the group of people who actually work on the designing part of the database. They keep a close watch on what data should be kept and in what format. They identify and design the whole set of entities, relations, constraints, and views.
- **End Users** – End users are those who actually reap the benefits of having a DBMS. End users can range from simple viewers who pay attention to the logs or market rates to sophisticated users such as business analysts.

Architecture of DBMS:

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent **n** modules, which can be independently modified, altered, changed, or replaced.

In **1-tier architecture**, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.

If the **architecture of DBMS is 2-tier**, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.

3-tier Architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.

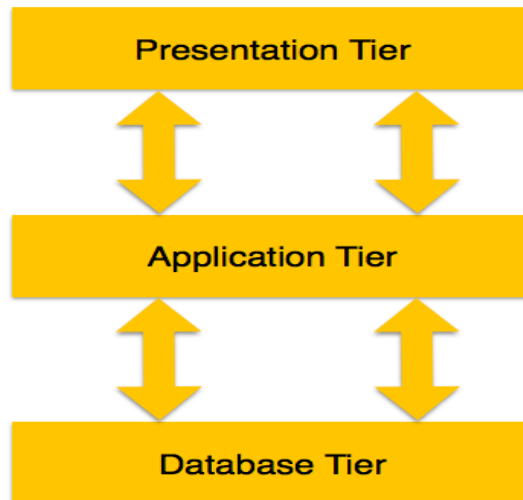
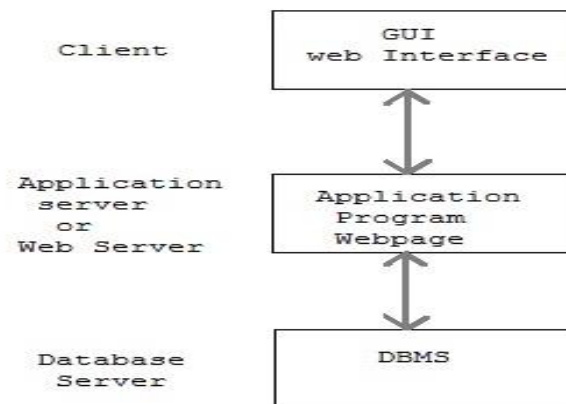


Figure: 3-tier architecture



- **Database (Data) Tier** – At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
- **Application (Middle) Tier** – At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.

- **User (Presentation) Tier** – End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

Data Models

Data models define how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in a DBMS. Data models define how data is connected to each other and how they are processed and stored inside the system.

The very first data model could be flat data-models, where all the data used are to be kept in the same plane. Earlier data models were not so scientific, hence they were prone to introduce lots of duplication and update anomalies.

Entity-Relationship Model

Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints. ER Model is best used for the conceptual design of a database.

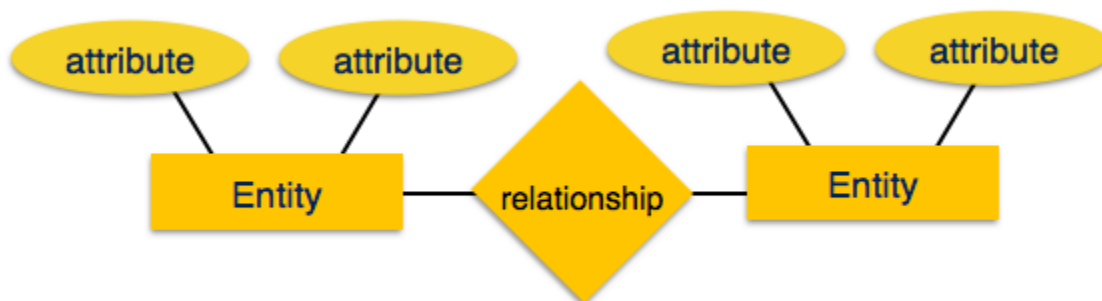


Figure: Entity relationship

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes

Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

- **Simple attribute** – Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute** – Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- **Derived attribute** – Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.
- **Single-value attribute** – Single-value attributes contain single value. For example – Social_Security_Number.
- **Multi-value attribute** – Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

These attribute types can come together in a way like –

- simple single-valued attributes (Id)
- simple multi-valued attributes(emailid)
- composite single-valued attributes (Name)
- composite multi-valued attributes (address, phone no(STD,TIN))

Entity-Set and Keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, the roll_number of a student makes him/her identifiable among students.

- **Super Key** – A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key** – A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Primary Key** – A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.\
- **Foreign keys** are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables. For example- In the below example the Stu_Id column in Course_enrollment table is a foreign key as it points to the primary key of the Student table.

Relationship

The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.

Relationship Set

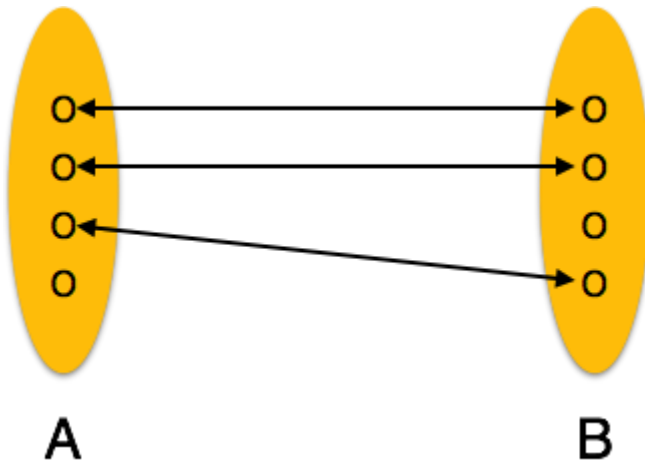
A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

Degree of Relationship:

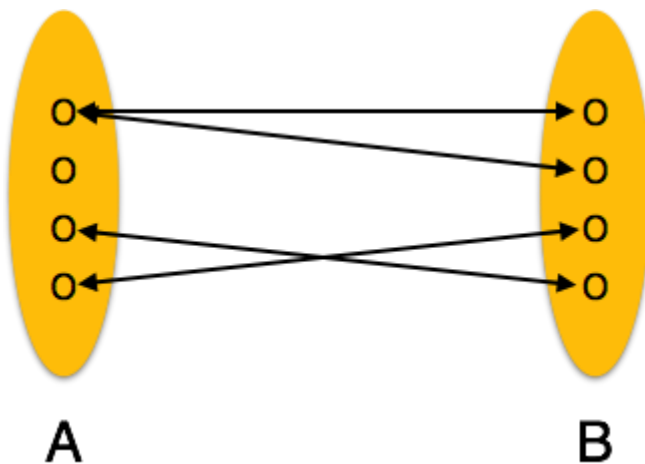
1. Unary (1 degree): Man (marry)
2. Binary (two degree): Teacher and student (teach)
3. Ternary (three degree): Seller, warehouse, product (Supply)

Types of Relationship/Mapping Cardinalities:

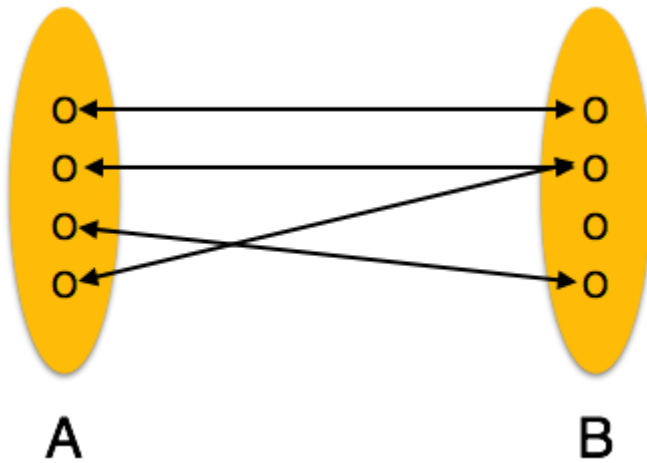
- **One-to-one** – One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



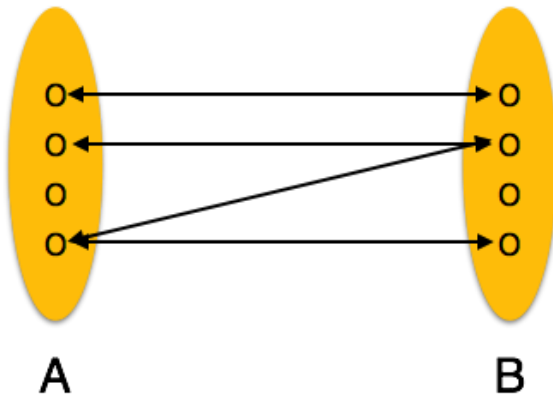
- **One-to-many** – One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



- **Many-to-one** – More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.

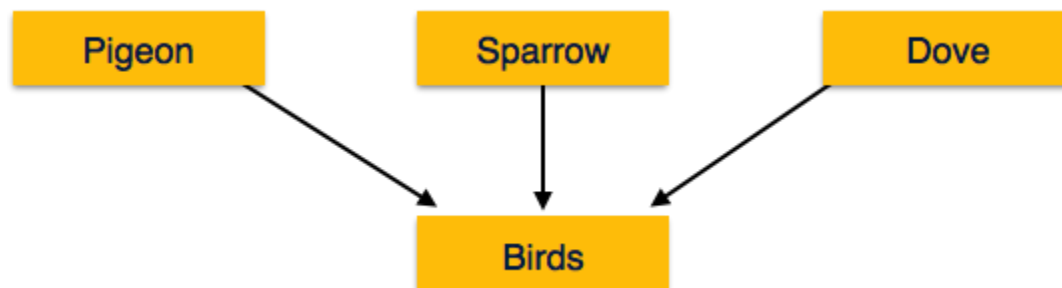


- **Many-to-many** – One entity from A can be associated with more than one entity from B and vice versa.



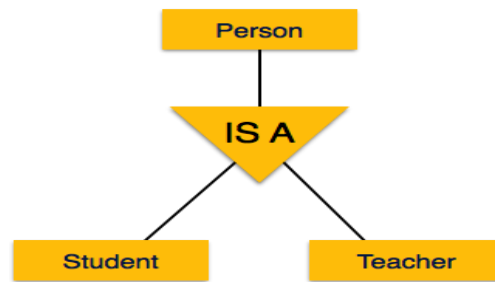
Generalization

As mentioned above, the process of generalizing entities, where the generalized entities contain the properties of all the generalized entities, is called generalization. In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For example, pigeon, house sparrow, crow and dove can all be generalized as Birds.



Specialization

Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics. Take a group 'Person' for example. A person has name, date of birth, gender, etc. These properties are common in all persons, human beings. But in a company, persons can be identified as employee, employer, customer, or vendor, based on what role they play in the company.

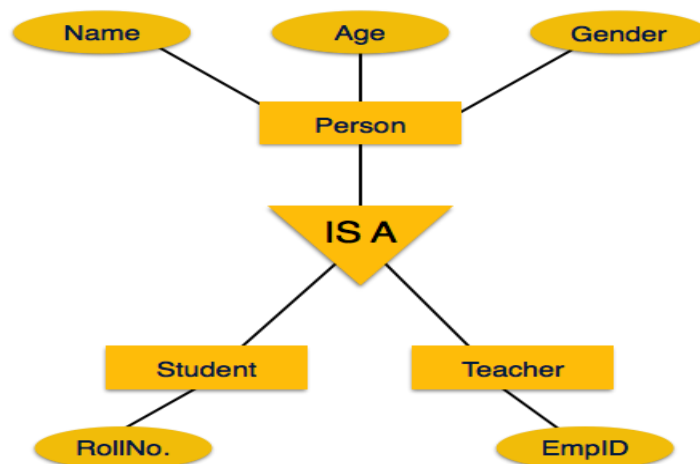


Similarly, in a school database, persons can be specialized as teacher, student, or a staff, based on what role they play in school as entities.

Inheritance

We use all the above features of ER-Model in order to create classes of objects in object-oriented programming. The details of entities are generally hidden from the user; this process known as **abstraction**.

Inheritance is an important feature of Generalization and Specialization. It allows lower-level entities to inherit the attributes of higher-level entities.



For example, the attributes of a Person class such as name, age, and gender can be inherited by lower-level entities such as Student or Teacher.

Symbols:

An **entity–relationship model (ER model)** is a systematic way of describing and defining a business process. An ER model is typically implemented as a database. The main components of E-R model are: entity set and relationship set.

Here are the geometric shapes and their meaning in an E-R Diagram –

Rectangle: Represents Entity sets.

Ellipses: Attributes

Diamonds: Relationship Set

Lines: They link attributes to Entity Sets and Entity sets to Relationship Set

Double Ellipses: Multivalued Attributes

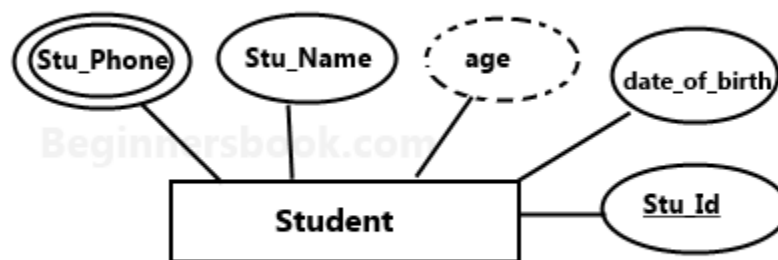
Dashed Ellipses: Derived Attributes

Double Rectangles: Weak Entity Sets

Double Lines: Total participation of an entity in a relationship set

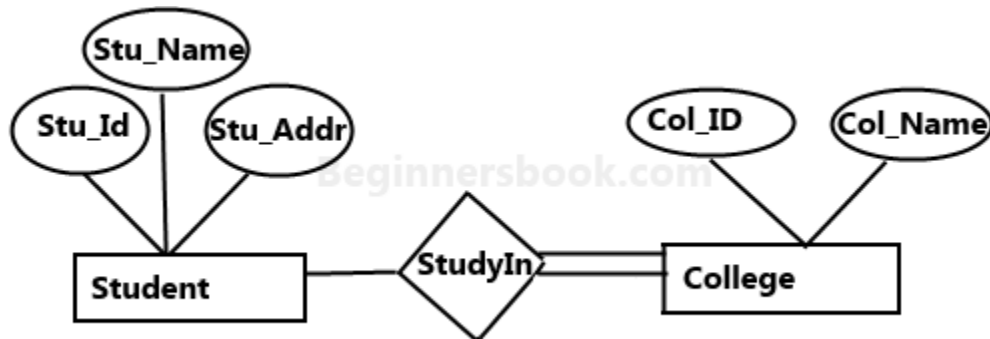
Multivalued Attributes: An attribute that can hold multiple values is known as multivalued attribute. We represent it with double ellipses in an E-R Diagram. E.g. A person can have more than one phone numbers so the phone number attribute is multivalued.

Derived Attribute: A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by dashed ellipses in an E-R Diagram. E.g. Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).



Total Participation of an Entity set:

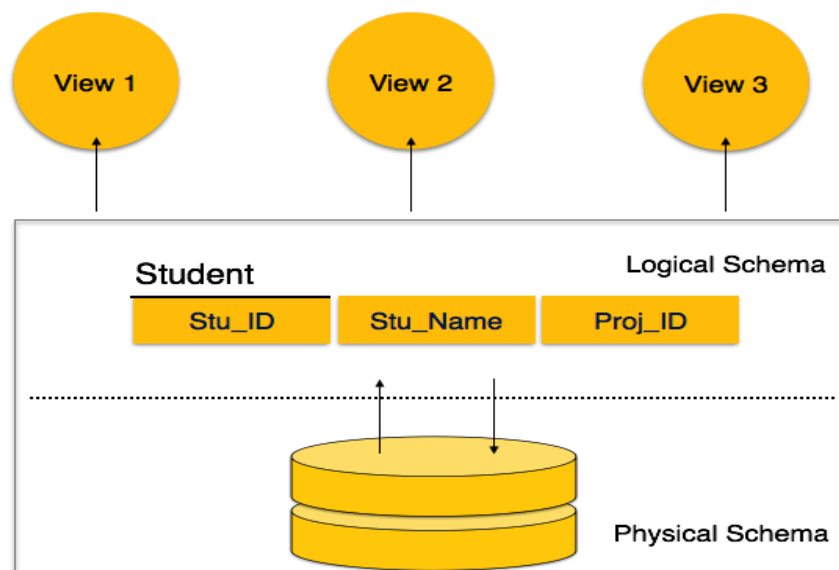
A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set. For example: In the below diagram each college must have at-least one associated Student.

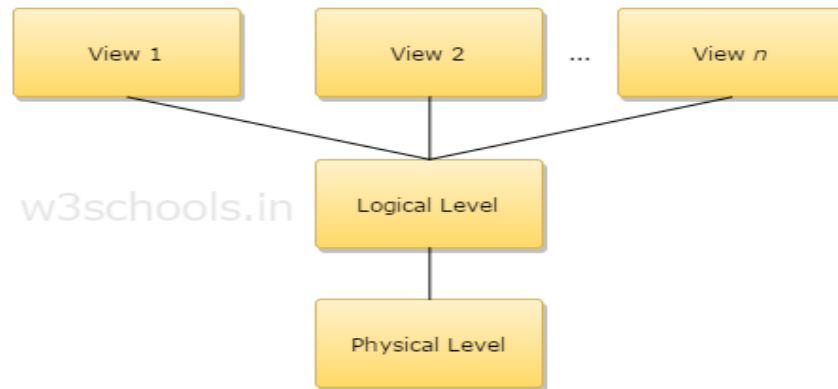


Database Schema

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.





Three Levels of Data Abstraction

A database schema can be divided broadly into two categories –

- **Physical Database Schema** – This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
- **Logical Database Schema** – This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

Database Instance

It is important that we distinguish these two terms individually. Database schema is the skeleton of database. It is designed when the database doesn't exist at all. Once the database is operational, it is very difficult to make any changes to it. A database schema does not contain any data or information.

A database instance is a state of operational database with data at any given time. It contains a snapshot of the database. Database instances tend to change with time. A DBMS ensures that its every instance (state) is in a valid state, by diligently following all the validations, constraints, and conditions that the database designers have imposed.