Database Management Systems

UNIT - I :INTRODUCTION & DATABASE DESIGN AND E-R MODEL

Database System Applications - Purpose of Database Systems - View of Data - Database Languages -Database Design - Database Architecture - Database Users and Administrators. The Entity-Relationship Model – Entity Sets –Relationship sets – Attributes –Entity-Relationship Diagrams –Weak Entity Sets –Extended E-R Features

Definition: Database Management System

A Database Management System (DBMS) is a software system that allows users to define, store, manage, and retrieve data in a structured and organized way. It acts as an interface between the database and the end-user or applications; ensuring data is kept accurate, secure, and accessible, and handles tasks like creating, updating, and querying data, as well as managing user access and security.

Key functions of a DBMS:

- **Data Definition:** Users can define the structure of the database, including the data types, fields, and relationships between tables using a data access language like SQL.
- **Data Storage:** The DBMS manages the physical storage of data, typically on a computer's hardware like hard disks.
- **Data Manipulation:** It enables users to create, modify, and delete data from the database.
- **Data Retrieval:** Users can query the database to retrieve specific information efficiently, for example, by asking "Show me all the documents from last year".
- **Data Integrity and Security:** It enforces rules to maintain data accuracy and consistency, and it provides security by controlling access to the data and protecting it from unauthorized users.
- **Data Backup and Recovery:** It includes functions to back up the database and recover it in case of failure.
- **Concurrency Control:** It allows multiple users to access the database at the same time without creating conflicts or data inconsistencies.

Problems with Traditional File-Based Systems

Before the introduction of modern DBMS, data was managed using basic file systems on hard drives. While this approach allowed users to store , retrieve and update files as needed, it came with numerous challenges.

- **Data Redundancy:** Duplicate entries across files
- **Inconsistency:** Conflicting or outdated information
- **Difficult Access:** Manual file search required
- **Poor Security:** No control over data access
- **Single-User Access:** No support for collaboration
- **No Backup/Recovery:** Data loss was often permanent

Types of DBMS

There are several types of Database Management Systems (DBMS), each tailored to different data structures, scalability requirements and application needs. The most common types are as follows:

1. Relational Database Management System (RDBMS)

- It organizes data into tables (relations) composed of rows and columns.
- Uses primary keys to uniquely identify rows and foreign keys to establish relationships between tables.
- Queries are written in **SQL (Structured Query Language)**, which allows for efficient data manipulation and retrieval.

Examples: MySQL oracle, Microsoft SQL Server and Postgre SQL.

2. NoSQL DBMS

- They are designed to handle large-scale data and provide high performance for scenarios where relational models might be restrictive.
- They store data in various non-relational formats, such as key-value pairs, documents, graphs or columns.

- These flexible data models enable rapid scaling and are well-suited for unstructured or semi-structured data.

Examples: MongoDB, DynamoDB and Redis.

3. Object-Oriented DBMS (OODBMS)

- It integrates object-oriented programming concepts into the database environment, allowing data to be stored as objects.
- Supports complex data types and relationships, making it ideal for applications requiring advanced data modeling and real-world simulations.

Examples: ObjectDB, db4o.

4. Hierarchical Database

- Organizes data in a tree-like structure, where each record (node) has a single parent and has multiple children.
- This model is similar to a file system with folders and subfolders.
- It is efficient for storing data with a clear hierarchy, such as organizational charts or file directories.
- Navigation is fast and predictable due to the fixed structure.
- It lacks flexibility and difficult to restructure or handle complex many-to-many relationships.

Example: IBM Information Management System (IMS).

5. Network Database

- It uses a graph-like model to allow more complex relationships between entities.
- Unlike the hierarchical model, it permits each child to have multiple parents, enabling many-to-many relationships.
- Data is represented using records and sets, where sets define the relationships.
- It is more flexible than the hierarchical model and better suited for applications with complex data linkages.

Example: Integrated Data Store (IDS), TurboIMAGE.

6. Cloud-Based Database

- They are hosted on cloud computing platforms like AWS, Azure or Google Cloud.

- They offer on-demand scalability, high availability, automatic backups and remote accessibility.

- These databases can be relational (SQL) or non-relational (NoSQL) and are maintained by cloud service providers, reducing administrative overhead.
- They support modern application requirements, including distributed access and real-time analytics.

Example: Amazon RDS (for SQL), MongoDB Atlas (for NoSQL), Google BigQuery.

Database Sytem Applications

Databases are fundamental to modern information systems and have widespread applications across various domains. Their primary purpose is to efficiently store, manage, and retrieve large volumes of structured data.

Key Applications of Databases:

- **Business and E-commerce:**

  Databases are crucial for managing inventory, sales data, customer information (CRM), supply chain logistics, online orders, product catalogs, and financial transactions.

- **Banking and Finance:**

  They are used to manage customer accounts, loan information, credit card transactions, investment portfolios, real-time market data, and financial reporting.

- **Education:**

  Educational institutions utilize databases for student records, academic performance tracking, course management, library systems, and administrative processes like enrollment and attendance.

- **Healthcare:**

  Databases are integral to Electronic Health Records (EHR) systems, storing patient information, medical history, diagnoses, treatment plans, and appointment scheduling.

- **Telecommunications:**

  Databases manage customer records, call details, billing information, network usage data, and service provisioning.

- **Social Media:**

  Social media platforms rely heavily on databases to store user profiles, posts, messages, media content, and manage user interactions.

- **Transportation and Logistics:**

  Databases are used for managing reservations (e.g., airline, railway), tracking shipments, optimizing routes, and managing fleet information.

- **Government:**

  Government agencies employ databases for managing citizen records, tax information, social security data, and various public services.

- **Manufacturing:**

  Databases support manufacturing processes by managing production details, inventory levels, order tracking, and supply chain management.

- **Data Science and Analytics:**
  Databases serve as the foundation for storing and organizing data used in data analysis, business intelligence, and machine learning applications to extract insights and make predictions.

Purpose of Database Systems

The fundamental purpose of a database is to efficiently and securely store, manage, and retrieve large volumes of structured data. This encompasses several key objectives:

- **Data Storage and Organization:**

  Databases provide a structured framework for organizing data, typically using tables, schemas, and indexes, to ensure logical coherence and ease of access. This allows for systematic management of information, whether it's customer details, product inventories, or financial transactions.

- **Efficient Data Retrieval and Manipulation:**

  The structured nature of databases enables rapid retrieval of specific information and efficient execution of operations like insertion, deletion, and updating of records. This is crucial for applications requiring quick access to data, such as e-commerce platforms or real-time analytics.

- **Data Integrity and Consistency:**

  Databases enforce rules and constraints to maintain data accuracy and consistency. This includes mechanisms to prevent data corruption, ensure data validity, and manage concurrent access from multiple users without compromising data integrity.

- **Data Security:**

  Databases incorporate security features like user authentication, access controls, and data encryption to protect sensitive information from unauthorized access or manipulation.

- **Data Sharing and Concurrency:**

  Database management systems (DBMS) facilitate the sharing of data among multiple users and applications while managing concurrent access to prevent conflicts and ensure data consistency.

- **Data Abstraction:**

  Databases provide users with an abstract view of the data, hiding the complex low-level details of how data is physically stored and maintained, thereby simplifying data interaction for end-users and developers.

In essence, databases transform raw data into organized information, which can then be used to derive knowledge and support informed decision-making across various domains and applications.

View of data

Abstraction is one of the main features of database systems. Hiding irrelevant details from user and providing abstract view of data to users, helps in easy and efficient **user-database** interaction. The top level of that architecture is "view level". The view level provides the "**view of data**" to the users and hides the irrelevant details such as **data relationship, database schema, constraints, security** etc from the user.

1. Data abstraction: Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction.
2. Instance and schema: Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema. The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular

database; the value of these variables at a moment of time is called the instance of that database.

The three main levels of data abstraction, or views of data, are:

Physical Level (Internal View)

This is the lowest level of abstraction, describing how the data is physically stored on the storage devices. It deals with details like file organization, indexing, data compression, and physical storage structures (e.g., blocks, cylinders). This view is primarily for database administrators and system programmers, who need to optimize performance and manage the physical storage of data.

Conceptual Level (Logical View)

This level describes what data is stored in the database and the relationships between data entities. It represents the overall logical structure of the database, including entities, attributes, and relationships, without detailing the physical storage. Database designers and programmers typically work at this level, defining the schema and ensuring data integrity.

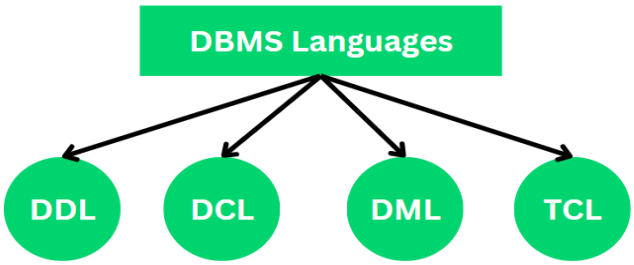
External Level (View Level)

This is the highest level of abstraction, providing customized views of the data for individual users or applications. Users at this level interact with a subset of the database tailored to their specific needs and permissions, without needing to know the entire database structure or physical storage details. Views, in SQL, are virtual tables created based on queries, offering this personalized perspective and enhancing data security by restricting access to sensitive information.

These levels of abstraction collectively enable data independence, meaning changes at one level do not necessarily require changes at higher levels, thus simplifying database management and application development.

Database Languages

Database languages are specialized languages used to interact with a database. They allow users to perform different tasks such as defining, controlling and manipulating the data. There are several types of database languages in DBMS, categorized into the following four main types:



1. DDL (Data Definition Language)
2. DCL (Data Control Language)
3. DML (Data Manipulation Language)
4. TCL (Transaction Control Language)

1. DDL (Data Definition Language)

The DDL stands for **Data Definition Language,** Which is used to define the database's internal structure and Pattern of the Database. It is used to define and modify the structure of the database itself, including the tables, views, indexes and other schema-related objects. It deals with the creation and modification of database schema, but it doesn't deal with the data itself.

Following are the five **DDL commands in SQL:**

- **CREATE:** Used to create database objects like tables, indexes or views.
- **ALTER:** Used to modify the structure of an existing database object, such as adding a new column to a table.
- **DROP:** Used to delete database objects.

- **TRUNCATE:** Used to remove all rows from a table, without affecting the structure.
- **RENAME:** Used to change the name of a database object.

CREATE Command

The CREATE is a DDL command used to create databases, tables, triggers and other database objects.

Syntax:

CREATE TABLE Students (column1 INT,column2 VARCHAR(50),column3 INT);

Alter Command

ALTER is a DDL command which changes or modifies the existing structure of the database and it also changes the schema of database objects. We can also add and drop constraints of the table using the ALTER command.

Syntax: ALTER TABLE Students ADD column_name datatype;

Drop Command

DROP is a DDL command used to delete/remove the database objects from the SQL database. We can easily remove the entire table, view or index from the database using this DDL command.

Syntax: DROP Table Table_name;

Truncate Command

The TRUNCATE command is used to delete all the records from a table without removing the structure. Unlike the DELETE command, which can remove specific rows and can be rolled back, TRUNCATE is a more efficient way to delete all rows in a table without logging individual row deletions.

Syntax: TRUNCATE TABLE table_name;

Rename Command

The RENAME command in a Database Management System (DBMS) is used to change the name of a database object, such as a table, column or index. This command is helpful when we want to give a more meaningful or appropriate name to an object without needing to recreate it.

Syntax: ***ALTER TABLE Old_Table_Name RENAME TO New_Table_Name;***

DCL (Data Control Language)

DCL stands for **Data Control Language**. It is used to control the access permissions of users to the database. DCL commands help grant or revoke privileges to users, determining who can perform actions like reading or modifying data. DCL commands are transactional, meaning they can be rolled back if necessary. The two main DCL commands are:

- **Grant:** Gives user access to the database
- **Revoke:** Removes access or permissions from the user

Grant Command

The GRANT command in a Database Management System (DBMS) is used to provide specific permissions or privileges to users or roles. This command allows administrators to control access to database objects, ensuring that only authorized users can perform certain actions, such as selecting, inserting, updating or deleting data.

Syntax:

GRANT privileges ON objectTO user_or_role [WITH GRANT OPTION];

Example: ***GRANT SELECT, INSERT ON students TO user;***

Revoke Command

The REVOKE command in a Database Management System (DBMS) is used to remove previously granted permissions or privileges from users or roles. This command is essential for managing access control, ensuring that users do not have more privileges than necessary to perform their tasks.

Syntax: ***REVOKE privileges ON object FROM user_or_role;***

Example: ***REVOKE ALL PRIVILEGES ON students FROM user;***

DML (Data Manipulation Language)

The DML (Data Manipulation Language) is used to manage and manipulate data within a database. With DML, you can perform various operations such as inserting,

updating, selecting and deleting data. These operations allow you to work with the actual content in your database tables. Here are the key DML commands:

- **SELECT:** Retrieves data from the table based on specific criteria.
- **INSERT:** Adds new rows of data into an existing table.
- **UPDATE:** Modifies existing data in a table.
- **DELETE:** Removes data from a table.

SELECT Command

The SELECT command in SQL (Structured Query Language) is used to retrieve data from one or more tables in a database. It is the most commonly used commands in SQL, allowing users to specify which columns and rows of data they want to retrieve and how they want that data organized. The SELECT statement can be used in various ways, such as selecting all data from a table, filtering records based on conditions or sorting the results.

Syntax: ***SELECT * FROM Table_Name***

Insert Command

The INSERT command in SQL (Structured Query Language) is used to add new records or rows to a table in a database. It is a key operation in database management, essential for populating tables with new data. This command can insert data into all columns or specific columns of a table.

Syntax:

INSERT INTO Table_Name (Column 1, Column 2, Column 3, Column 4) VALUES (Value 1, Value 2,Value 3, Value 4);

Update Command:The UPDATE command in SQL (Structured Query Language) is used to modify existing records in a table. This command enables users to change the values of one or more columns for specific rows based on a condition (criteria). It is important for maintaining and adjusting data in a database when necessary.

Syntax: UPDATE Table_Name SET Name = 'New_Value' WHERE Name = 'Old_Value';

Delete Command

The DELETE command in SQL (Structured Query Language) is used to remove one or more existing records from a table in a database. It is an essential operation for managing data, enabling users to delete specific rows that meet certain conditions or criteria.

Syntax: DELETE FROM Table Name WHERE Column = Value;

4. TCL (Transaction Control Language)

The TCL full form is Transaction Control Language commands are used to manage and control transactions in a database, grouping them into logical units. These commands help ensure the integrity of data and consistency during complex operations. Here are the two main commands in this category:

- **Commit:** Saves all the changes made during the current transaction to the database. These are very useful in the banking sector.
- **Rollback:** used to restore the database to its original state from the last commit. This command also plays an important role in Banking Sectors.

Commit Command

The COMMIT command is used **to save all changes made during a transaction in the database.** This command ensures that modifications made by DML statements (such as INSERT, UPDATE or DELETE) become permanent in the database.
Syntax: Commit;

ROLLBACK Command

The rollback command is used to restore the database to its state at the last COMMIT, effectively undoing any changes made since that point. It helps ensure data consistency by allowing the reversal of partial or erroneous operations.

Syntax: ROLLBACK;

Database design

Database design is the process of creating a detailed, logical model of a database and then implementing that model. Its purpose is to ensure that data is organized efficiently, stored correctly, and can be easily retrieved and updated. A good database design reduces redundancy, improves data integrity, and enhances application performance.

Key Steps in Database Design:

Requirements Collection and Analysis:

- Gather and analyze user requirements to understand what data needs to be stored, how it will be used, and what operations will be performed.
- This involves identifying entities, attributes, and relationships relevant to the system.

Conceptual Database Design (ER Modeling):

- Develop a high-level description of the data using an Entity-Relationship (ER) model.
- Entities represent real-world objects (e.g., Student, Course), attributes describe their properties (e.g., Student Name, Course ID), and relationships define how entities are connected (e.g., Student enrolls in Course).
- This step focuses on representing data conceptually, independent of a specific DBMS.

Logical Database Design (Relational Schema Conversion):

- Convert the conceptual ER model into a logical schema, typically a relational schema for relational databases.
- This involves mapping entities to tables, attributes to columns, and relationships to foreign keys.
- Normalization techniques are often applied in this stage to reduce data redundancy and improve data integrity.

Schema Refinement:

Analyze the relational schema to identify and resolve potential problems like redundancy and anomalies.

- Further normalization (e.g., to 3NF, BCNF) and restructuring of relations are performed to achieve an optimal design.

Physical Database Design:

- Refine the database design for a specific DBMS and physical storage considerations.
- This includes defining data types, indexing strategies, clustering tables, and optimizing storage structures to enhance performance.

Application and Security Design:

- Integrate the database design with application development, considering how applications will interact with the database.
- Implement security measures, including user roles, permissions, and integrity constraints, to protect data.

Importance of Good Database Design:

- **Data Consistency and Integrity:** Ensures data accuracy and reliability.
- **Reduced Redundancy:** Minimizes duplicated data, saving storage space and preventing inconsistencies.
- **Improved Performance:** Facilitates faster data retrieval and manipulation through optimized structures and indexing.
- **Enhanced Security:** Allows for effective implementation of access controls and data protection.
- **Easier Maintenance:** A well-designed database is easier to manage, update, and evolve over time.

Database Architecture

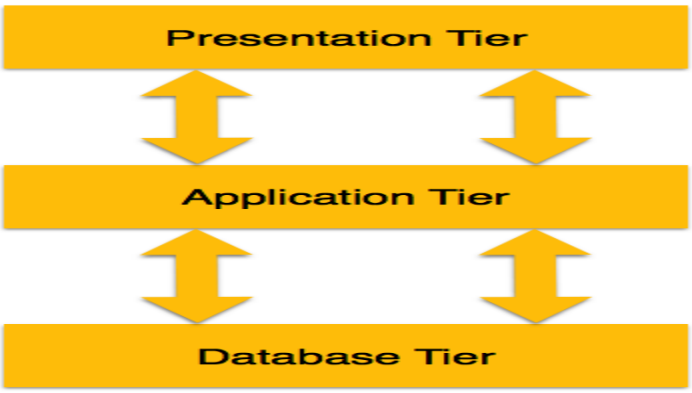
The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but

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.
- **Example:** A standalone desktop application like Microsoft Access or a personal Excel spreadsheet where the data and application are on the same machine.
- 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.
- **Example:** A banking application where a teller's workstation (client) directly connects to the bank's database server to process transactions.

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.



- **Example:** A web application where the user interacts with a web browser (client), which communicates with an application server, and the application server then interacts with the database server for data retrieval and storage.

- **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

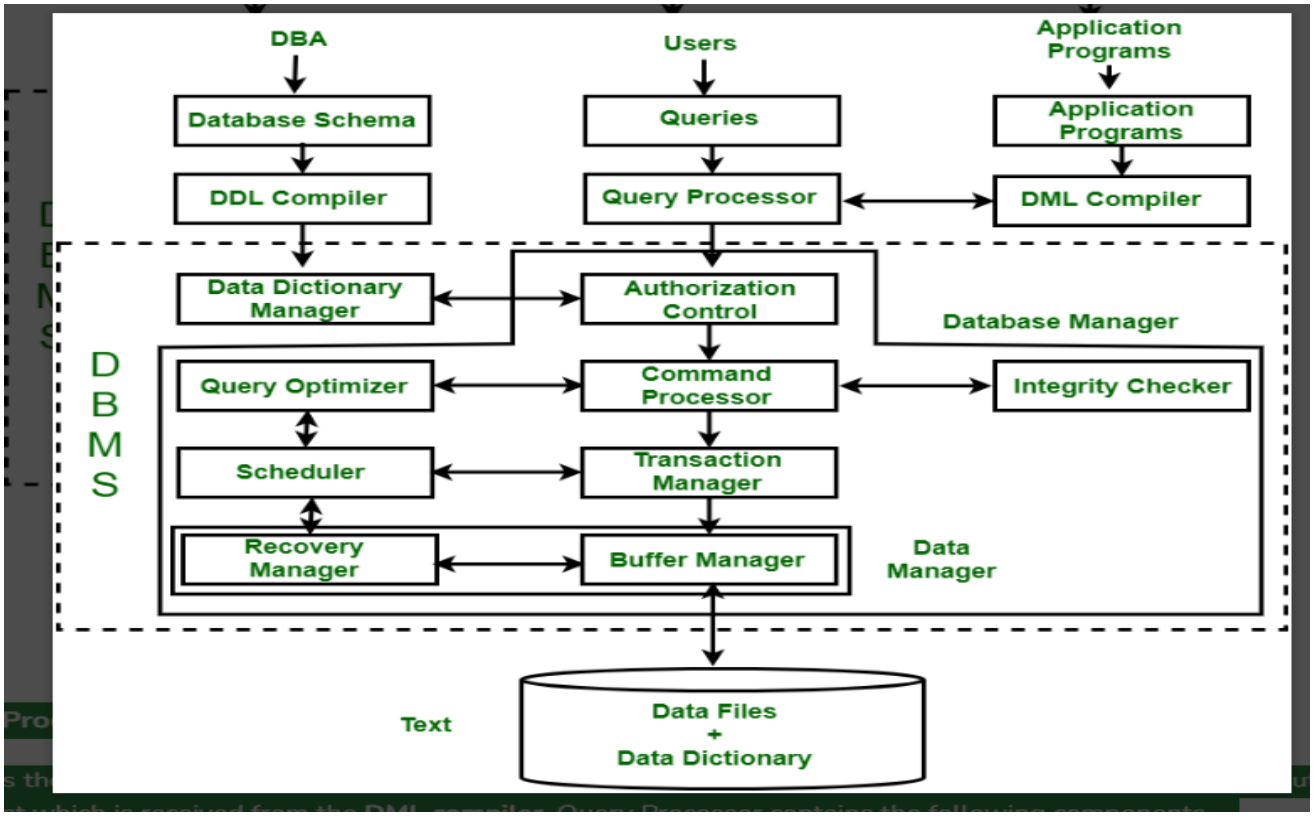
  abstracted view of the database. End-users are unaware of any existence of the database beyond the application.

- 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.

Components of a Database System

Query Processor, Storage Manager, and Disk Storage.



Query Processor

It interprets the requests (queries) received from end user via an application program into instructions. It also executes the user request which is received from the **DML compiler**.

Query Processor contains the following components -

- **DML Compiler:** It processes the DML statements into low level instruction (machine language), so that they can be executed.
- **DDL Interpreter:** It processes the DDL statements into a set of table containing meta data (data about data).
- **Embedded DML Pre-compiler:** It processes DML statements embedded in an application program into procedural calls.
- **Query Optimizer:** The Query Optimizer executes instructions generated by the DML Compiler and improves query execution efficiency by choosing the best query plan, considering factors such as indexing, join order, and available

system resources. For instance, if a query involves joining two large tables, the optimizer will select the **best join order to minimize query execution time**.

2. Storage Manager

Storage Manager is an interface between the data stored in the database and the queries received. It is also known as Database Control System. It maintains the consistency and integrity of the database by applying the constraints and executing the DCL statements. It is responsible for updating, storing, deleting, and retrieving data in the database. It contains the following components:

- **Authorization Manager:** It ensures role-based access control, i.e,. checks whether the particular person is privileged to perform the requested operation or not.
- **Integrity Manager:** It checks the **integrity constraints** when the database is modified.
- **Transaction Manager:** It controls concurrent access by performing the operations in a scheduled way that it receives the transaction. Thus, it ensures that the database remains in the consistent state before and after the execution of a transaction.
- **File Manager:** It manages the file space and the data structure used to represent information in the database.
- **Buffer Manager:** It is responsible for **cache memory** and the transfer of data between the secondary storage and main memory.

3. Disk Storage

It contains the following essential components:

- **Data Files:** It stores the actual data in the database.
- **Data Dictionary:** It contains the information about the **structure of database.**
- **objects** such as tables, constraints, and relationships. It is the repository of information that governs the metadata.

- **Indices:** Provides **faster data retrieval** by allowing the DBMS to find records quickly, improving query performance.

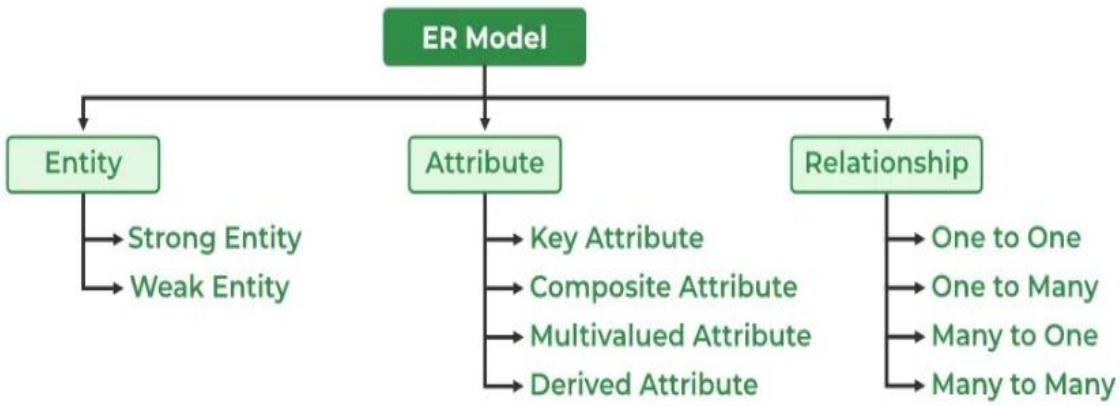
Entity-Relationship Model (ER Model)

The Entity-Relationship Model (ER Model) is a conceptual model for designing a databases. This model represents the logical structure of a database, including entities, their attributes and relationships between them.

The ER model is a high-level, conceptual data model that uses three basic concepts:

- Entities: Real-world objects, concepts, or things, such as a Student, Product, or Order. In a relational database, an entity becomes a table.
- Attributes: The properties or characteristics of an entity. For example, a Student entity might have attributes like StudentID, Name, and Age. Attributes become columns in a table.

- Relationships: Associations between entities, such as a Student enrolling in a Course. Relationships are crucial for defining how data is linked across different tables.



Entity

An Entity represents a real-world object, concept or thing about which data is stored in a database. It act as a building block of a database. Tables in relational database represent these entities.

Example of entities:

- **Real-World Objects:** Person, Car, Employee etc.
- **Concepts:** Course, Event, Reservation etc.
- **Things:** Product, Document, Device etc.

The entity type defines the structure of an entity, while individual instances of that type represent specific entities.

Entity Set

An entity refers to an individual object of an entity type, and the collection of all entities of a particular type is called an entity set. For example, E1 is an entity that belongs to the entity type "Student," and the group of all students forms the entity set.

Types of Entity

There are two main types of entities:

1. Strong Entity

A Strong Entity is a type of entity that has a key Attribute that can uniquely identify each instance of the entity. A Strong Entity does not depend on any other Entity in the Schema for its identification. It has a primary key that ensures its uniqueness and is represented by a rectangle in an ER diagram.

2. Weak Entity

A Weak Entity cannot be uniquely identified by its own attributes alone. It depends on a strong entity to be identified. A weak entity is associated with an identifying entity (strong entity), which helps in its identification.

A weak entity are represented by a double rectangle. The participation of weak entity types is always total. The relationship between the weak entity type and its identifying strong entity type is called identifying relationship and it is represented by a double diamond.

Example:

A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents can't exist without the employee. So dependent will be a Weak Entity Type and Employee will be identifying entity type for dependent, which means it is Strong Entity Type.

Strong Entity and Weak Entity

Attributes in ER Model

Attributes are the properties that define the entity type. For example, for a Student entity Roll_No, Name, DOB, Age, Address, and Mobile_No are the attributes that define entity type Student. In ER diagram, the attribute is represented by an oval.

Attribute

Types of Attributes

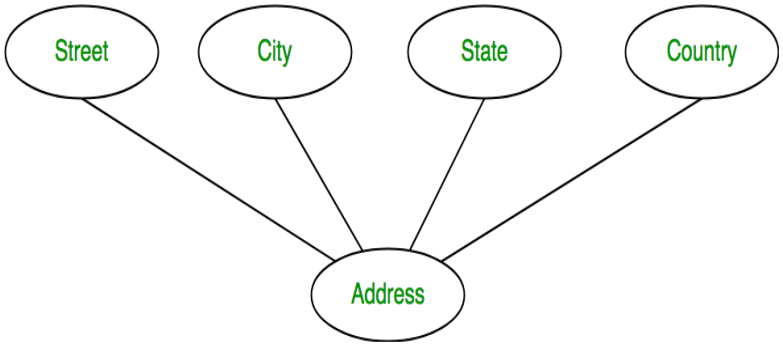
1. Key Attribute

The attribute which uniquely identifies each entity in the entity set is called the key attribute. For example, Roll_No will be unique for each student. In ER diagram, the key attribute is represented by an oval with an underline.

Key Attribute

2. Composite Attribute

An attribute composed of many other attributes is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country. In ER diagram, the composite attribute is represented by an oval comprising of ovals.



Composite Attribute

3. Multivalued Attribute

An attribute consisting of more than one value for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval.

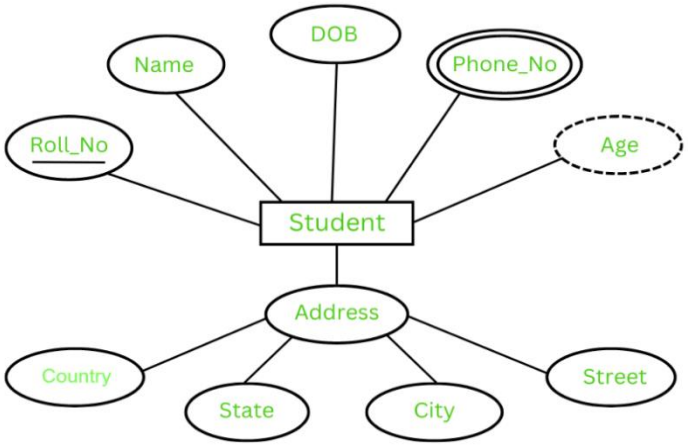
Multivalued Attribute

4. Derived Attribute

An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB).

In ER diagram, the derived attribute is represented by a dashed oval.

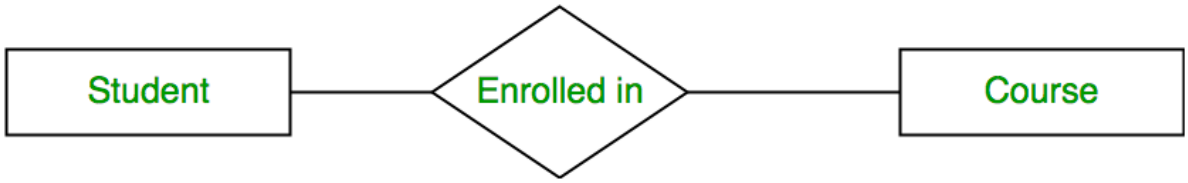
Derived Attribute



E-R model Diagram

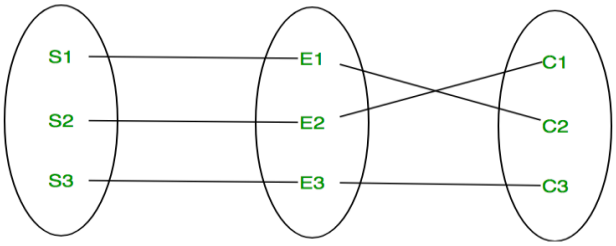
Relationship Type and Relationship Set

A Relationship Type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, the relationship type is represented by a diamond and connecting the entities with lines.



Entity-Relationship Set

A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 as enrolled in C2, S2 as enrolled in C1, and S3 as registered in C3.

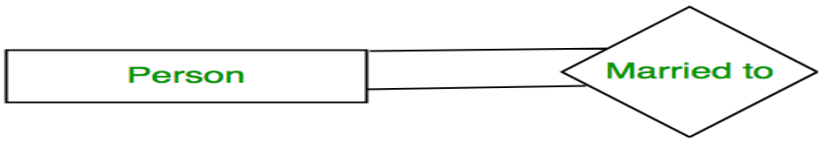


Relationship Set

Degree of a Relationship Set

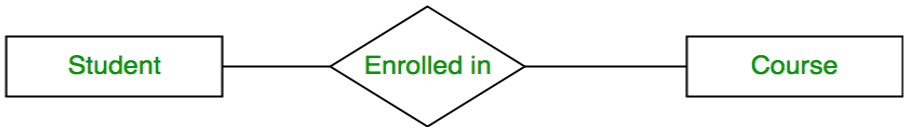
The number of different entity sets participating in a relationship set is called the degree of a relationship set.

1. Unary/Recursive Relationship: *When there is only ONE entity set participating in a relation, the relationship is called a unary relationship. For example, one person is married to only one person.*



Unary Relationship

2. Binary Relationship: *When there are TWO entities set participating in a relationship, the relationship is called a binary relationship. For example, a Student is enrolled in a Course.*



Binary Relationship

3. ***Ternary Relationship:*** *When there are three entity sets participating in a relationship, the relationship is called a ternary relationship.*

4. N-ary Relationship: *When there are n entities set participating in a relationship, the relationship is called an n-ary relationship.*

Cardinality in ER Model

The maximum number of times an entity of an entity set participates in a relationship set is known as cardinality.

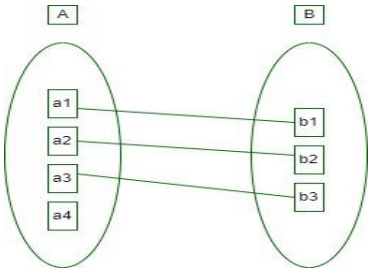
Cardinality can be of different types:

1. One-to-One

When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one. Let us assume that a male can marry one female and a female can marry one male. So the relationship will be one-to-one.

One to One Cardinality

Using Sets, it can be represented as:



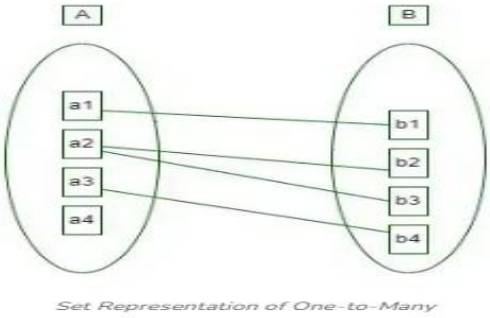
Set Representation of One-to-One

2. One-to-Many

In one-to-many mapping as well where each entity can be related to more than one entity. Let us assume that one surgeon department can accommodate many doctors. So the Cardinality will be 1 to M. It means one department has many Doctors.

one to many cardinality

Using sets, one-to-many cardinality can be represented as:



Set Representation of One-to-Many

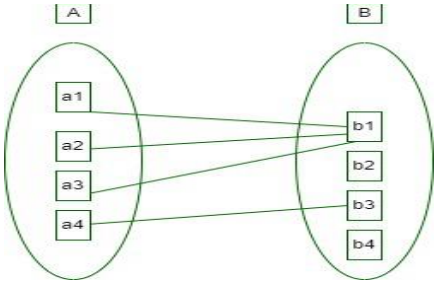
3. Many-to-One

When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one.

many to one cardinality

Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.

Using Sets, it can be represented as:



Set Representation of Many-to-One

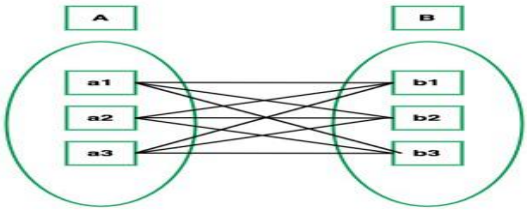
In this case, each student is taking only 1 course but 1 course has been taken by many students.

4. Many-to-Many

When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

many to many cardinality

Using Sets, it can be represented as:



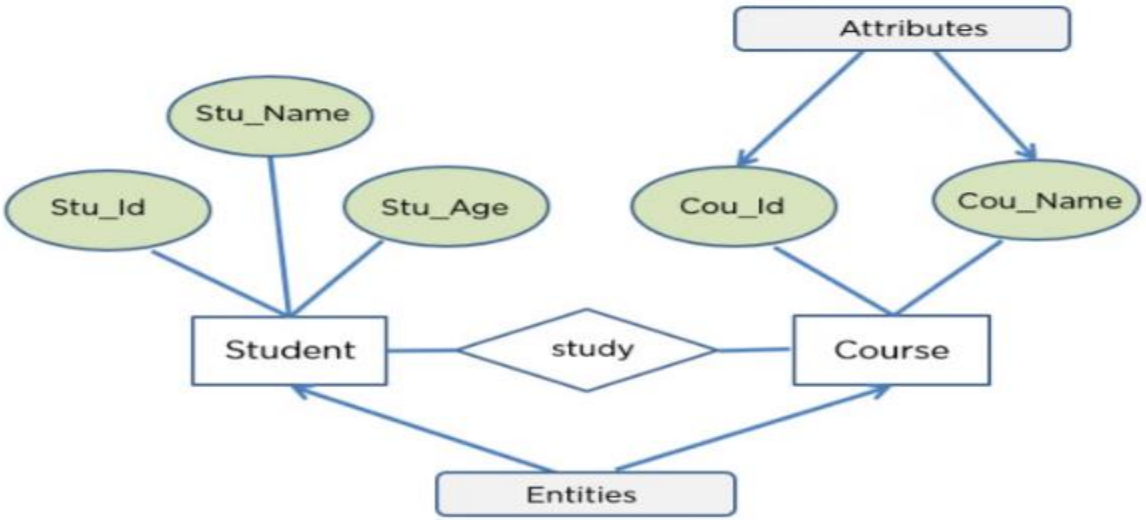
Many-to-Many Set Representation

In this example, student b1 is enrolled in b1 and b3 and Course b3 is enrolled by a1, a2, and a3. So it is many-to-many relationships.

Entity Relationship Diagrams

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database.

A simple ER Diagram:



In the following diagram we have two entities Student and College and their relationship. The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time. Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.

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

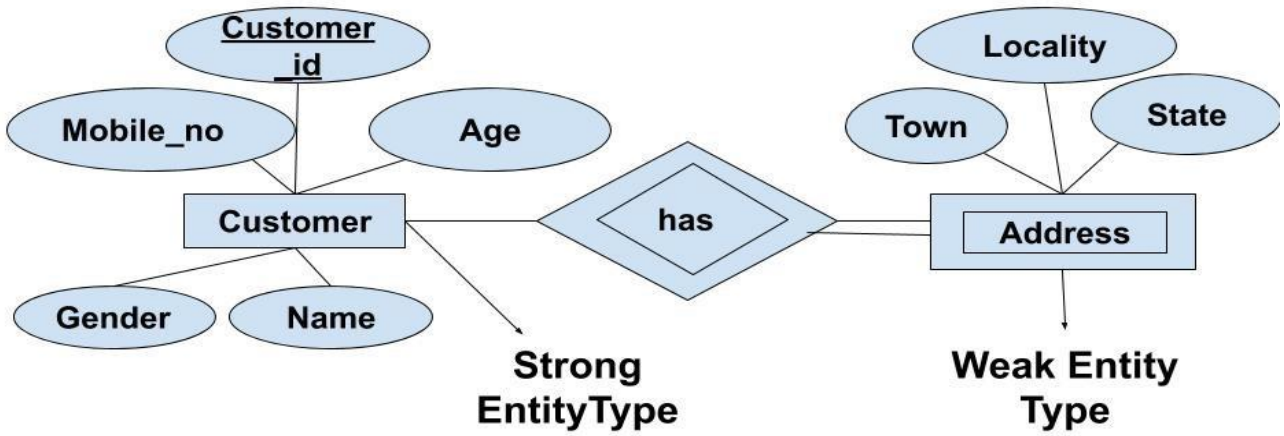
Weak Entity type/ Sets

Weak entity type doesn't have a key attribute. Weak entity type can't be identified on its own. It depends upon some other strong entity for its distinct identity. This can be understood with a real-life example. There can be children only if the parent exits. There can be no independent existence of children. There can be a room only if building exits. There can be no independent existence of a room.

A weak entity is represented by a double outlined rectangle. The relationship between a weak entity type and strong entity type is called an identifying relationship and shown with a double outlined diamond instead of a single outlined diamond. This representation can be seen in the diagram below.

Example : If we have two tables of Customer(Customer_id, Name, Mobile_no, Age, Gender) and Address(Locality, Town, State, Customer_id).

Here we cannot identify the address uniquely as there can be many customers from the same locality. So, for this, we need an attribute of Strong Entity Type i.e 'Customer' here to uniquely identify entities of 'Address' Entity Type.



Entity Set

Extended ER Features

Extended Entity-Relationship (EER) model features in DBMS, such as generalization, specialization, aggregation, and inheritance, enhance the basic ER model to handle complex databases.

These features enable a more accurate, hierarchical representation of real-world data by creating higher-level entities from lower-level ones (generalization), breaking down entities into specific subclasses (specialization), modeling relationships between whole and part entities (aggregation), and allowing subclasses to inherit properties from their superclasses (inheritance).

Generalization:

This is a bottom-up approach where common attributes from multiple lower-level entity sets are combined to form a single, higher-level entity set.
Example: Car, Bus, and Motorcycle entities can be generalized into a single "Vehicle" entity.

Specialization:
The opposite of generalization, specialization is a top-down approach where a higher-level entity set is broken down into two or more lower-level entity sets, based on specific characteristics.
Example: A "Person" entity might be specialized into "Student" and "Employee" entities.

Aggregation:

This feature allows a relationship to be treated as a higher-level entity itself. It models relationships between entire entities, representing situations where one entity is a whole and another is a part of it.

Example: Consider a "Student" entity and a "Course" entity. A relationship like "Student-Takes-Course" can be aggregated to represent "Student-Takes-Course-In-A-Sem".

Inheritance:

An important outcome of specialization and generalization, inheritance allows subclasses to automatically acquire the attributes and relationships of their parent superclasses.

Example: If "Employee" is a superclass and "Manager" is a subclass, the "Manager" will inherit all attributes of an "Employee" (like Name, ID) and can also have its own unique attributes (like Department).

Benefits of Extended ER Features

- **Enhanced Clarity and Structure:**

EER features provide a more detailed and organized database design, making it easier to understand.

- **Improved Maintainability:**

By reducing redundancy and complexity, the EER model simplifies database maintenance and updates.

- **Better Representation:** The features allow for a more accurate and precise modeling of real-world data and complex relationships.