**Introduction to DBMS**

 A **Database Management System (DBMS)** is software that enables efficient creation, management, and retrieval of databases, acting as a bridge between users and organized data storage.

**Need for DBMS in Modern Applications:**

* **Data Integration:** Combines data from multiple sources for unified views and informed decisions.
* **Data Security:** Protects sensitive data with access controls and authentication.
* **Data Consistency:** Ensures accuracy during concurrent access.
* **Efficient Management:** Handles large datasets, complex queries, and transactions.
* **Backup & Recovery:** Safeguards data and ensures continuity during failures.

**DBMS vs File System**

1. **Data Organization**:
   * File System: Stores data as flat files without inherent relationships.
   * DBMS: Organizes data into structured tables with defined relationships.
2. **Data Redundancy**:
   * File System: Prone to data duplication across multiple files.
   * DBMS: Minimizes redundancy through normalization and centralized storage.
3. **Data Integrity**:
   * File System: Lacks built-in mechanisms to enforce data integrity.
   * DBMS: Enforces integrity constraints to maintain data accuracy.
4. **Data Security**:
   * File System: Offers basic security features, often limited to file-level permissions.
   * DBMS: Provides advanced security controls, including user authentication and authorization.
5. **Data Access**:
   * File System: Access is typically sequential, making complex queries inefficient.
   * DBMS: Supports efficient querying through indexing and optimized access paths.

**Limitations of File-Based Systems:**

* **Data Redundancy**: Duplicated data across files, wasting storage and causing inconsistencies.
* **Data Inconsistency**: Updates in one file may not reflect in others, leading to errors.
* **Limited Data Integrity**: No mechanisms to ensure data accuracy or validation.
* **Security Issues**: Weak security and user access controls.
* **Lack of Advanced Querying**: No support for complex queries, requiring manual search.
* **Data Maintenance Challenges**: Difficult to manage large datasets without specialized tools.
* **Poor Multi-User Support**: Struggles with concurrent access, risking conflicts and data corruption.

**Advantages of DBMS over File Systems:**

1. **Reduced Data Redundancy**:
   * Centralized data storage minimizes duplication.
2. **Improved Data Integrity**:
   * Enforces rules to maintain data accuracy and consistency.
3. **Enhanced Security**:
   * Advanced security features protect data from unauthorized access.
4. **Data Independence**:
   * Changes in data structure do not affect application programs.
5. **Efficient Data Access**:
   * Optimized query processing allows for faster data retrieval.
6. **Concurrent Access**:
   * Supports multiple users accessing data simultaneously without conflicts.
7. **Backup and Recovery**:
   * Provides mechanisms to back up data and recover from failures.

**Components of DBMS**

 A Database Management System (DBMS) is essential for storing, managing, and manipulating data in relational databases.

* **Purpose:** Ensures efficient data organization and accessibility.
* **Components:** Comprises various elements working together to manage data effectively.
* **Functions:** Maintains data integrity, enforces security measures, and optimizes performance.

**The core components of a DBMS are:**

1. Database Engine
2. Database Schema
3. Query Processor
4. Transaction Manager

**1. Database Engine** : The Database Engine is the central part of a DBMS, managing data storage, retrieval, and modification.

* **Interface Role:** It acts as a bridge between the database and applications, handling operations like querying, updating, and inserting data.

**Functions of the Database Engine:**

* **Data Storage Management:** Organizes data on disk or in memory.
* **Data Retrieval:** Optimizes access for efficient query results.
* **Concurrency Control:** Supports simultaneous access for multiple users.
* **Data Integrity:** Maintains data consistency through rules and constraints.
* **Transaction Management:** Ensures reliable processing of transactions.

**Example:** In systems like **MySQL** or **PostgreSQL**, the database engine executes SQL operations (e.g., SELECT, INSERT, UPDATE, DELETE) and ensures proper functionality and performance.

**2. Database Schema:** The Database Schema is the logical structure of a database, outlining tables, fields, data types, and relationships as a blueprint for organizing and managing data.

**Functions of the Database Schema:**

* Defines Tables and Columns: Specifies what tables exist and the columns within those tables.
* Specifies Data Types: Defines the types of data (e.g., integer, text, date) for each field.
* Establishes Relationships: Describes how different tables are related (e.g., one-to-many, many-to-many).
* Implements Constraints: Includes primary keys, foreign keys, unique constraints, etc., to maintain data integrity.

**Example:** A database schema for an e-commerce application might define tables like Customers, Orders, and Products, where each table is linked via foreign keys to ensure referential integrity.

**3. Query Processor:** The Query Processor interprets SQL queries, serving as a bridge between the user and the database engine.

* Query Optimization: It converts high-level queries into efficient low-level operations for the DBMS to execute.

**Functions of the Query Processor:**

* Parsing: It first checks the syntax and semantics of the query to ensure it is valid.
* Optimization: The query processor analyzes the query to determine the most efficient way to execute it.
* Execution: After optimization, the query processor sends the query plan to the database engine for execution.

**Example:** When you execute a query like SELECT \* FROM Employees WHERE Age > 30, the query processor breaks down the SQL command and formulates an execution plan that determines how best to retrieve the data from the database, such as whether to use an index or full table scan.

**4. Transaction Manager:** The Transaction Manager handles database transactions, ensuring they are treated as a single unit of work.

* **Purpose:** It maintains the **ACID properties** (Atomicity, Consistency, Isolation, Durability) to ensure reliable and consistent transaction processing.

**Functions of the Transaction Manager:**

* **Atomicity**: Ensures that a transaction is completed entirely or not at all (i.e., it is atomic).
* **Consistency**: Guarantees that the database remains in a valid state before and after the transaction.
* **Isolation**: Ensures that transactions are executed in isolation, preventing interference from other transactions.
* **Durability**: Ensures that once a transaction is committed, the changes are permanent, even if the system crashes.

**Example:**

* **Transaction Integrity:** In online banking, the transaction manager ensures both debit and credit operations are completed successfully as a single unit.
* **Error Handling:** If an issue occurs (e.g., network failure), the transaction is rolled back to maintain consistency and prevent data loss.

**Advantages and Disadvantages of DBMS**:

**Advantages of DBMS:**

* **Data Integrity and Security:** Ensures consistent data and restricts unauthorized access.
* **Data Abstraction and Independence:** Separates data from application logic, allowing changes without affecting applications.
* **Efficient Data Access:** Optimizes query processing for faster data retrieval.
* **Concurrent Access and Crash Recovery:** Manages simultaneous data access and restores data after failures.

**Disadvantages of DBMS:**

* **Complexity:** Requires specialized knowledge for setup and maintenance.
* **Cost:** Involves expenses for software, hardware, and training.
* **Performance Overhead:** May introduce latency due to abstraction layers.

**DBMS Architecture**

DBMS architecture defines the structure and interaction between the database, users, and applications, impacting the system's scalability, efficiency, and flexibility.

* DBMS architectures are classified by layers separating the user interface, application logic, and data storage. The three types are:
  + 1-Tier Architecture
  + 2-Tier Architecture
  + 3-Tier Architecture

**1-Tier Architecture**

In 1-Tier Architecture, the database, application logic, and user interface are all hosted on a single machine.

* **Use Case:** Commonly used in standalone applications where users interact directly with the database without a network or middleware.

**Key Features:**

* **Single Layer**: The database, application, and user interface are all integrated into one system.
* **Direct Interaction**: Users interact directly with the database via a front-end application installed on the same machine.
* **Limited Scalability**: It is best suited for small applications with fewer users.
* **Ideal Use Case**: Personal or small-scale applications, where complex multi-user support is not necessary.

**Example Scenario:**

A local inventory management system where the database, application, and user interface are all on one desktop.

**2-Tier Architecture**

The 2-Tier Architecture divides the DBMS into two layers: the client layer for user interaction and the server layer where the database resides.

* The client communicates directly with the server to send queries and receive

**Key Features:**

* **Client-Server Model**: The client handles the user interface, and the server handles data management.
* **Database Server**: A dedicated database server manages data and processes queries.
* **Direct Communication**: Clients communicate directly with the database server via a network.

**Example Scenario:**

A business system where employees use a desktop application to access a central database located on a server.

**3-Tier Architecture**

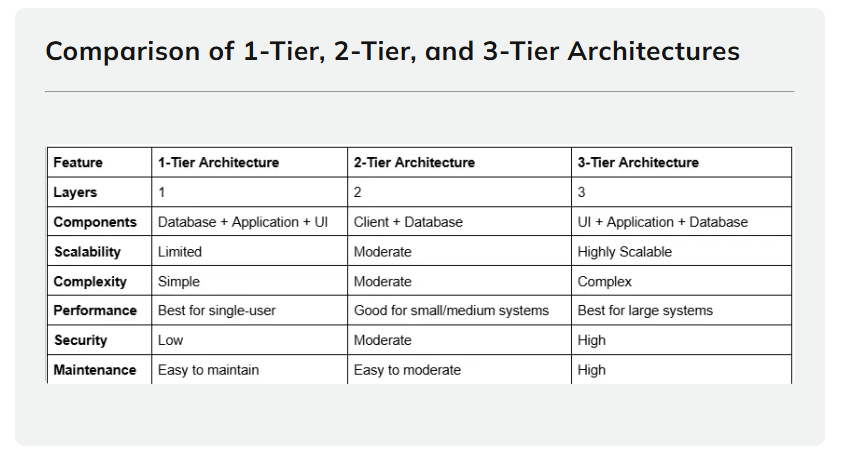
**In 3-Tier Architecture,**a system divided into three layers:

1. **Presentation Layer:** User interface (e.g., web browser, app).
2. **Application Layer:** Handles business logic and communication between UI and database.
3. **Data Layer:** Stores and manages data.

**Key Features:**

* **Separation of Concerns:** Independent roles for UI, business logic, and data management.
* **Scalability:** Each tier can scale or modify independently.
* **Centralized Access:** Application layer ensures secure and efficient data handling.

**Example:** Online banking systems where the UI processes user transactions, the application layer handles logic, and the database stores account data.



**Pros and Cons of 3-Tier Architecture**

**Pros:**

* **Scalability**: Ideal for large-scale systems with many users.
* **Flexibility**: Changes to one tier can be made without impacting the others.
* **Enhanced Security**: The business logic layer can filter requests, ensuring better security for database interactions.
* **Easier Maintenance**: Each layer can be updated or replaced independently.

**Cons:**

* **Complexity**: More complex to set up and maintain compared to 1-Tier and 2-Tier architectures.
* **Latency**: Communication between the layers can introduce some delay or overhead.
* **Higher Cost**: Requires more resources for infrastructure and maintenance.

**Database Users**

1. **Database Administrators (DBAs):**
   * **Role:** Oversee database management, including design, implementation, maintenance, and security.
   * **Responsibilities:** Backup and recovery, performance tuning, user access control.
2. **Application Programmers:**
   * **Role:** Develop applications that interact with the database.
   * **Responsibilities:** Write and optimize SQL queries, ensure application efficiency and security.
3. **End Users:**
   * **Role:** Individuals who interact with the database through applications to perform tasks.
   * **Responsibilities:** Input, update, and retrieve data as needed for their specific functions.

**Database Models**

A database model is a framework for defining and structuring data.

* **Purpose:** Determines how data is stored, retrieved, and manipulated.
* **Importance:** Helps select the right model for specific applications.

**Key Types of Database Models:**

* **Hierarchical Model**
* **Network Model**
* **Relational Model** (Primary Focus)
* **Object-Oriented Model**

**Hierarchical Model :** The Hierarchical Model organizes data in a tree-like structure, where each record has one parent and multiple children, resembling an organizational chart.

**Key Features:**

* **Parent-Child Relationship**: Each node (record) has a parent node, except for the root node, and can have multiple child nodes.
* **Fixed Structure**: Data must follow a predefined path, and navigation is done in a top-down manner.
* **Data Retrieval**: Traversing the hierarchy is required to retrieve related data.

**Example:**

* A **company's organizational chart**:
  + Root node: **CEO**
  + Child nodes: **Managers**
  + Further children: **Employees** working under each manager.

**Pros:**

* Simple and easy to understand.
* Excellent for data with a clear hierarchical relationship (e.g., file systems).

**Cons:**

* Inflexible structure.
* Difficult to handle many-to-many relationships.

**Network Model:** The **Network Model** organizes data in a web-like structure, allowing each child to have multiple parents. It uses sets, with owner-records linked to member-records.

**Key Features:**

* **Many-to-Many Relationships**: A record can have multiple parent and child records, unlike the Hierarchical Model.
* **Pointers**: Records are connected using pointers, which represent relationships between data.
* **Flexibility**: More flexible than the Hierarchical Model, enabling complex relationships.

**Example:**

* A **project management database**:
  + One employee can work on multiple projects.
  + Multiple employees can work on the same project.

**Pros:**

* Flexible for complex relationships.
* Better at managing many-to-many relationships.

**Cons:**

* More complex structure than the Hierarchical Model.
* Harder to navigate and manage.

**Relational Model :** The **Relational Model** organizes data into tables with rows and columns. Primary keys uniquely identify records, and foreign keys establish relationships between tables.

**Advantages:**

* **Simplicity**: Easy to understand and implement.
* **Data Integrity**: Supports constraints like primary keys and foreign keys to ensure data consistency.
* **Flexibility**: Allows complex queries using SQL to retrieve and manipulate data.

**Relational Model Operations:**

* **SELECT**: Retrieve data from one or more tables.
* **INSERT**: Add new records to a table.
* **UPDATE**: Modify existing records.
* **DELETE**: Remove records from a table.

**Key Features of Relational Model :**

* **Tables (Relations)**: Data is stored in tables, with rows representing records and columns representing attributes.
* **Primary Key**: A unique identifier for each record in a table.
* **Foreign Key**: A key used to link tables together, pointing to the primary key in another table.
* **Normalization**: The process of organizing data to reduce redundancy and dependency.

**Example:**

* A **customer-order relationship**:
  + **Customers Table**: Contains customer information.
  + **Orders Table**: Contains orders made by customers, linked to the CustomerID using a foreign key.

**Object-Oriented Model :** The Object-Oriented Model uses OOP concepts like classes, objects, inheritance, and encapsulation to store and manage data.

**Key Features:**

* **Objects**: Data is represented as objects, similar to objects in object-oriented programming languages.
* **Classes**: Data types are defined in classes, and objects are instances of these classes.
* **Inheritance**: Allows objects to inherit properties and behaviors from parent objects, supporting data reuse.
* **Encapsulation**: Combines data and methods in a single object, promoting data security and integrity.

**Example:**

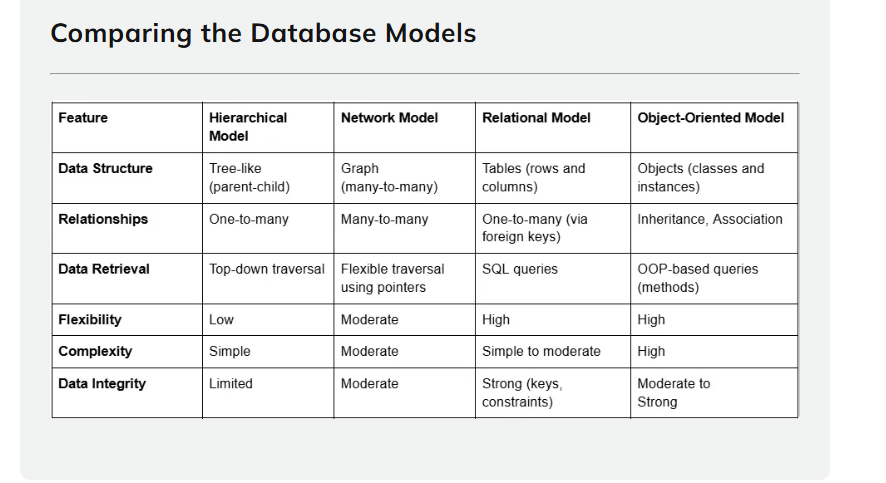
* A **product catalog** :
  + A class Product might contain attributes like ProductID, Name, Price, and Category.
  + A class Electronics could inherit from Product and add specific attributes like Warranty Period.

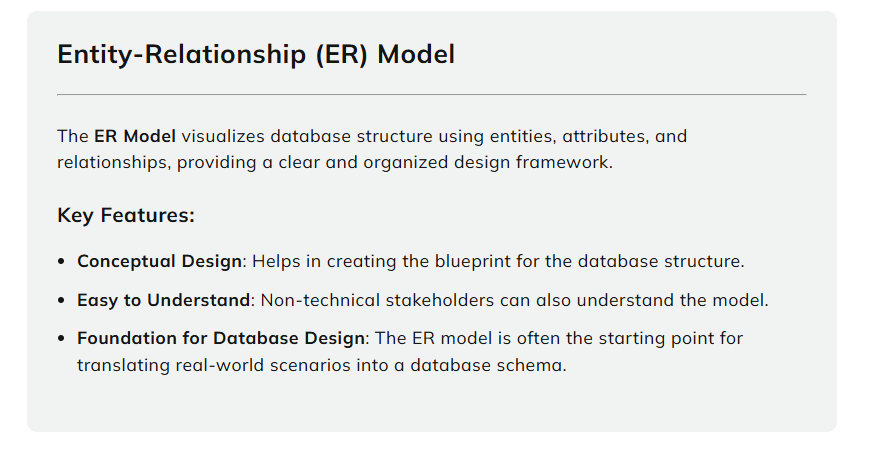
**Advantages:**

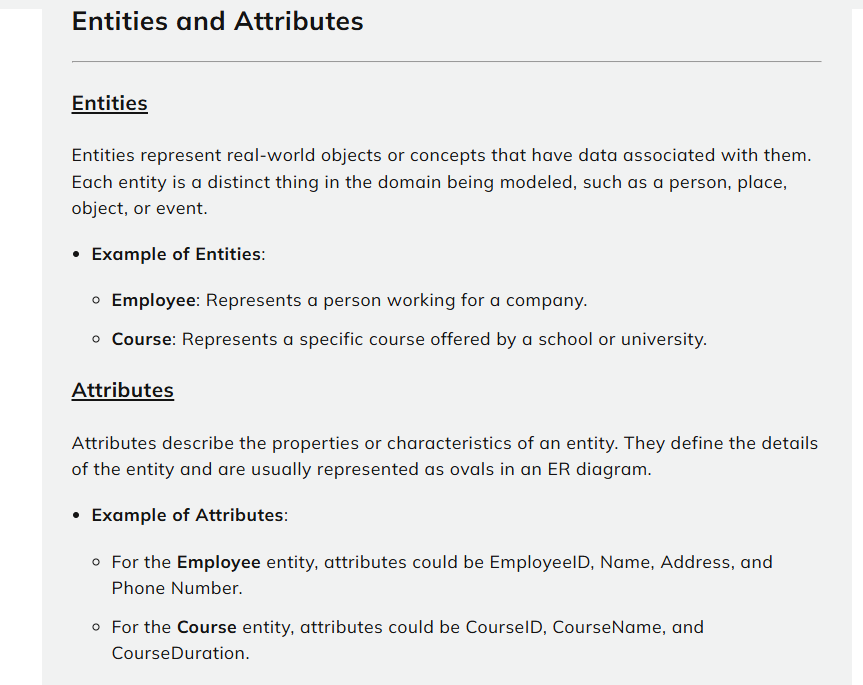
* **Natural Mapping:** Easily represents real-world objects in the database.
* **Inheritance:** Simplifies handling complex data and reduces redundancy.
* **Encapsulation:** Combines data and operations for better security and maintenance.

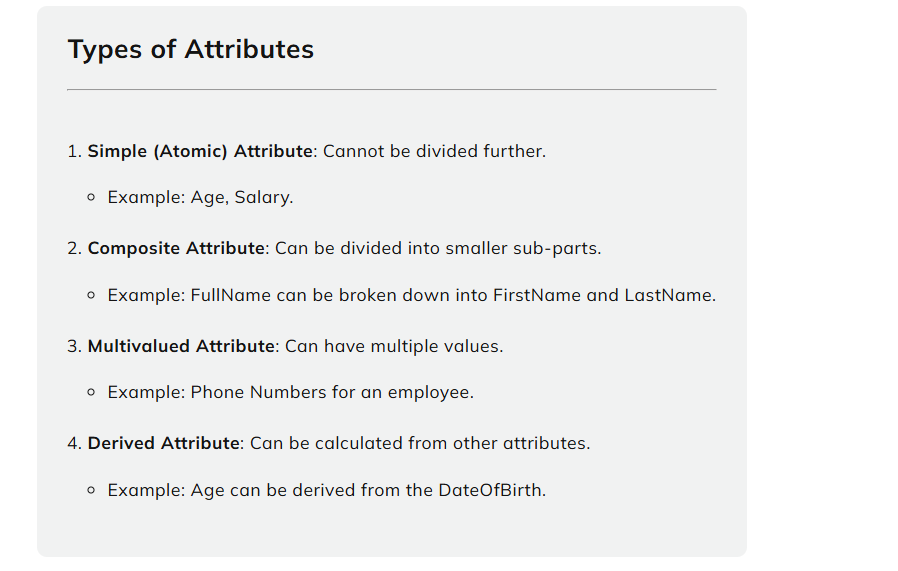
**Challenges:**

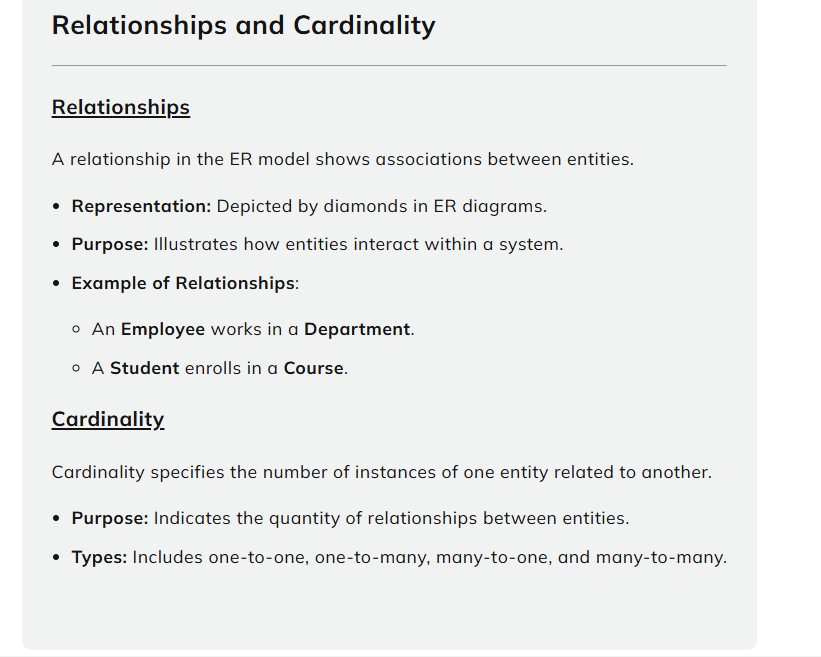
* **Complexity**: More difficult to implement and manage compared to the Relational Model.
* **Limited Support**: Not all DBMS support object-oriented features natively.











**Types of cardinality**

1. **One-to-One (1:1)**: One instance of an entity is associated with only one instance of another entity.
   * Example: A **Person** has only one **Passport**.
   * Notation: A line connecting two entities with a "1" at both ends.
2. **One-to-Many (1:M)**: One instance of an entity is associated with multiple instances of another entity.
   * Example: A **Department** has many **Employees**, but each **Employee** works in one **Department**.
   * Notation: A line connecting two entities with a "1" at one end and an "M" at the other end.
3. **Many-to-One (M:1)**: Many instances of an entity are associated with a single instance of another entity.
   * Example: Many **Orders** can belong to one **Customer**.
   * Notation: A line connecting two entities with an "M" at one end and a "1" at the other end.
4. **Many-to-Many (M:N)**: Multiple instances of one entity are associated with multiple instances of another entity.
   * Example: A **Student** can enroll in many **Courses**, and a **Course** can have many **Students**.
   * Notation: A line connecting two entities with "M" at both ends.

**Entity-Relationship Diagram (ERD)**

ER Diagrams are graphical representations of the ER Model. They use specific symbols to depict entities, attributes, relationships, and cardinality.

**key symbols and their meanings:**

1. **Rectangle (Entity)**: Represents an entity.
   * Example: A rectangle labeled **Employee** represents an **Employee** entity.
2. **Ellipse (Attribute)**: Represents an attribute of an entity.
   * Example: An ellipse labeled **Name** connected to the **Employee** entity represents the **Name** attribute.
3. **Diamond (Relationship)**: Represents a relationship between entities.
   * Example: A diamond labeled **WorksIn** connecting **Employee** and **Department** represents the relationship that an employee works in a department.
4. **Line**: Connects entities, attributes, and relationships.
   * A solid line connects an entity to its attribute.
   * A solid or dashed line connects an entity to a relationship.

**Notations for Cardinality**

* **1**: Represents "One" cardinality.
* **M**: Represents "Many" cardinality.
* **Crow's Foot**: A symbol resembling a "crow's foot" represents the "many" side of a relationship.
* **Straight Line**: Represents the "one" side of a relationship.

**Example ER Diagram:**

 [Employee] -- (WorksIn) -- [Department]

| |

(Name) (DeptName)

(EmployeeID) (DeptID)

In this diagram:

* **Employee** and **Department** are entities.
* **WorksIn** is a relationship between them.
* **Employee** has attributes like EmployeeID and Name.
* **Department** has attributes like DeptID and DeptName.

**Relational Database**

A **Relational Database** organizes data into structured tables known as **relations**, making it easy to store and retrieve large amounts of data.

Each table consists of rows and columns, where:

* **Rows** represent individual records.
* **Columns** represent attributes of the records.

Relational databases are widely used due to their flexibility, scalability, and support for powerful querying languages like SQL.

**Relation and Tuple**

**Relation**

 A **Relation** is a table in a relational database. It consists of a set of **tuples** (rows), where each tuple represents a record and each record contains values for specific attributes (columns).

* **Example**:
  + Table name: **Students**
  + Attributes: StudentID, Name, Age, Course

**Tuple**

 A **Tuple** is a single row within a relation (table). It represents a set of related data values for a given record. In SQL, a tuple corresponds to a record or row of data.

* **Example**:
  + (1, 'John Doe', 22, 'Computer Science')

Each value in the tuple corresponds to an attribute .

**Attribute and Domain**

**Attribute**

An attribute is a column in a table that represents a specific type of data for each record.

* **Purpose:** Attributes define the properties or characteristics of the data in the table.
* **Example**: In a **Students** table, the attributes could be StudentID, Name, Age, and Course.

**Domain**

A domain is the set of allowable values for an attribute in a relation.

* **Purpose:** It specifies the type of data an attribute can store, such as integer, string, or date.
* **Example**:
  + **Age** attribute: Domain = Integer (Positive Numbers)
  + **Name** attribute: Domain = String (Alphabetical Characters)

**Schema:**

A schema is the structure that defines how data is organized in a relational database, including tables, columns, and data types.

* **Purpose:** It specifies relationships, constraints, and the logical design, ensuring data organization and integrity.
* **Example**:
  + **Students** schema might consist of the table name, the attribute names (StudentID, Name, Age, Course), and the data types for each attribute (e.g., INT, VARCHAR, DATE).

Schemas define how the data is stored, but not the actual data itself.

**Instance**

An instance is the actual data stored in a database at a specific moment, representing the real-time content of the schema.

* **Dynamic Nature:** It changes over time as data is inserted, updated, or deleted, acting as a snapshot of the database at any given point.
* **Example**:
  + **Students Instance**: A table with specific rows of data:

Each time you query or update the database, you interact with a specific instance of the schema.

**Keys in Relational Databases**

**Keys in relational databases** define relationships between tables, ensure data integrity, and enable efficient data retrieval, playing a crucial role in structuring and managing data.

**1. Primary Key**

 A Primary Key is a column or set of columns that uniquely identifies each record in a table, ensuring data integrity and consistency.

**Key Characteristics:**

* **Uniqueness**: Every value in the primary key column(s) must be unique.
* **Non-null**: A primary key cannot have a NULL value.
* **Single**: Each table can only have one primary key.

**Example:**

 CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(50),

HireDate DATE

);

 In the above example, EmployeeID is the primary key that uniquely identifies each employee in the table.

**2. Foreign Key:**

A foreign key is a column that links two tables by referencing the primary key in another table.

* **Purpose:** It enforces referential integrity by ensuring all foreign key values correspond to valid records in the referenced table.

**Key Characteristics:**

* **Referential Integrity**: Ensures that the foreign key value exists in the referenced table.
* **Nullable**: A foreign key can have NULL values, which indicates no relationship in some cases.

**Example:**

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

EmployeeID INT,

OrderDate DATE,

FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)

);

In this example, EmployeeID in the Orders table is a foreign key that references the EmployeeID in the Employees table, linking each order to a specific employee.

**3. Candidate Key :** A Candidate Key is a set of columns that can uniquely identify rows in a table, qualifying it to serve as a potential primary key.

* **Alternate Keys:** Only one candidate key is selected as the primary key, while the others are called alternate keys.

**Key Characteristics:**

* **Uniqueness**: Every value in the candidate key columns must be unique.
* **Non-null**: A candidate key cannot contain NULL values.

**Example:**

 Consider a table storing information about books:

CREATE TABLE Books (

BookID INT,

ISBN VARCHAR(20),

Title VARCHAR(100),

PRIMARY KEY (BookID)

);

In this case, both BookID and ISBN are candidate keys because they can each uniquely identify a book. However, only BookID is selected as the primary key.

**4. Super Key**

 A **Super Key** is a set of columns that uniquely identifies a record in a table, potentially including extra, unnecessary columns.

* **Relation to Primary Key:** Every primary key is a super key, but not all super keys qualify as primary keys due to the inclusion of redundant attributes.

**Key Characteristics:**

* **Uniqueness**: A super key must uniquely identify each record.
* **Redundant Columns**: A super key may contain extra, unnecessary columns, making it less efficient than the minimal set of columns in a candidate key.

**Example:**

* BookID, ISBN, and the combination of BookID and Title are all super keys, as they uniquely identify each record in the Books table.
* **Unnecessary Column:** The combination of BookID and Title is a super key but includes an extra column (Title) that is not required for unique identification.

**5. Composite Key:** A Composite Key is a primary key made up of two or more columns, ensuring records in a table are uniquely identified.

* **Purpose:** Used when a single column cannot uniquely identify records, with the combined values guaranteeing uniqueness.

**Key Characteristics:**

* **Multiple Columns**: A composite key is made up of two or more columns.
* **Uniqueness**: The combination of values in the composite key columns must be unique across all rows.

**Example:**

 Consider a table that records student enrollments in courses:

 CREATE TABLE Enrollments (

StudentID INT,

CourseID INT,

EnrollmentDate DATE,

PRIMARY KEY (StudentID, CourseID)

);

In this case, the composite key consists of both StudentID and CourseID. Together, these two columns uniquely identify each enrollment record.

**Integrity Constraints:**

* **Ensure Data Accuracy and Consistency:** Integrity constraints enforce relationships and prevent incorrect modifications in database tables.
* **Types of Constraints:**
  + **Entity Integrity**
  + **Referential Integrity**
  + **Domain Constraints**

**1. Entity Integrity:**

* **Ensures Unique Identification:** Guarantees that each row in a table is uniquely identifiable, preventing data ambiguity.
* **Primary Key Constraint:**
  + **Purpose:** Uniquely identifies each record with unique, non-NULL values.
  + **Example:**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(50),

Department VARCHAR(50)

);

* + **Importance:** Prevents duplicate records and ensures each record can be accessed without confusion.

**2. Referential Integrity:**

* **Definition:** Ensures that relationships between tables remain consistent by making sure foreign keys always reference valid primary keys in related tables.
* **Foreign Key Constraint:**
  + **Purpose:** Ensures a foreign key in one table refers to an existing primary key in another table, maintaining the relationship between tables.
  + **Example:**

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

Amount DECIMAL(10, 2),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

* **Actions on Foreign Keys:**
  + **ON DELETE CASCADE:** Automatically deletes related records when a parent record is deleted.
  + **ON UPDATE CASCADE:** Automatically updates related records when the referenced primary key is updated.
  + **Example:**

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

Amount DECIMAL(10, 2),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

ON DELETE CASCADE

)

**3. Domain Constraints**

**Domain Constraints:** Define the allowed values, data types, and formats for a column, ensuring only valid data is entered.

* **Data Type**: Defines the type of data a column can hold (e.g., INTEGER, VARCHAR, DATE).

**Example**:

 CREATE TABLE Products (

ProductID INT PRIMARY KEY,

Name VARCHAR(100),

Price DECIMAL(10, 2) -- Price must be a decimal value

);

* **Check Constraints**: Enforces rules on the values of a column, such as restricting ranges or requiring specific conditions.
  + **Example**:

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Age INT,

CHECK (Age > 18) -- Only allows ages above 18

);

* **Default Values**: Specifies a default value for a column if no value is provided during insertions.
  + **Example**:

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

OrderStatus VARCHAR(20) DEFAULT 'Pending' -- Default status is 'Pending'

);