Vikash Polytechnic, Bargarh

Vikash Polytechnic
Campus: Vikash Knowledge Hub, Barahaguda Canal Chowk, NH6
PO/DIST: Bargarh-768028, Odisha

Lecture Note on

Database Management System

Diploma 4th Semester

Submitted By:- *Pratyush Sarangi*

CHAPTER-1

Basic Concepts of DBMS (Database Management System)

A **Database Management System (DBMS)** is software that manages databases by providing tools to store, retrieve, update, and delete data efficiently. Below are the fundamental concepts of DBMS:

1. Database & DBMS

- **Database**: A collection of related data organized in a structured way.
- **DBMS**: A software system that enables users to define, create, maintain, and control access to the database. Examples include MySQL, PostgreSQL, Oracle, and SQL Server.

2. Data Models

A data model defines how data is structured, stored, and manipulated. Major types:

- **Hierarchical Model**: Data is organized in a tree-like structure (parent-child relationship).
- **Network Model**: Data is stored in a graph structure with multiple parent-child relationships.
- **Relational Model**: Data is stored in tables (relations) using rows and columns.
- **Object-Oriented Model**: Combines database concepts with object-oriented programming.

3. Database Schema & Instance

- **Schema**: The logical structure of a database (tables, fields, constraints).
- **Instance**: The actual data stored in the database at a specific time.

4. Keys in DBMS

- **Primary Key**: Uniquely identifies a record in a table.
- **Foreign Key**: A reference to a primary key in another table, establishing relationships.
- **Candidate Key**: A set of attributes that uniquely identify a record (one is chosen as the primary key).
- **Super Key**: A set of attributes that uniquely identify a record, but may have unnecessary attributes.
- **Composite Key**: A primary key consisting of multiple columns.

5. Normalization

A technique to organize a database efficiently by reducing redundancy and improving data integrity. Normal forms include:

- **1NF (First Normal Form)**: Removes duplicate columns, ensures atomicity.
- **2NF (Second Normal Form)**: Removes partial dependencies.
- **3NF (Third Normal Form)**: Removes transitive dependencies.

6. ACID Properties (Transaction Management)

Transactions in DBMS must follow:

- **Atomicity**: A transaction is either fully completed or not executed at all.
- **Consistency**: Ensures the database remains in a valid state.
- **Isolation**: Transactions do not interfere with each other.
- **Durability**: Changes are permanently saved even in case of system failure.

7. Indexing

A technique to speed up data retrieval using special data structures (e.g., B-trees, Hash Indexes).

8. SQL (Structured Query Language)

SQL is used to interact with databases. Categories:

- **DDL (Data Definition Language)**: CREATE, ALTER, DROP
- **DML (Data Manipulation Language)**: INSERT, UPDATE, DELETE
- **DCL (Data Control Language)**: GRANT, REVOKE
- **TCL (Transaction Control Language)**: COMMIT, ROLLBACK, SAVEPOINT

9. Joins in DBMS

Used to combine data from multiple tables:

- **INNER JOIN**: Returns matching records from both tables.
- **LEFT JOIN**: Returns all records from the left table and matching records from the right.
- **RIGHT JOIN**: Returns all records from the right table and matching records from the left.
- **FULL JOIN**: Returns all records when there is a match in either table.

10. Database Users

- **Database Administrator (DBA)**: Manages and secures the database.
- **Application Programmers**: Develop applications that interact with the database.
- **End Users**: Access data through applications.

Purpose of Database Systems

A **Database Management System (DBMS)** is designed to handle large amounts of data efficiently and securely. The key purposes of database systems are:

1. Data Storage, Retrieval, and Management

- A database system provides a structured way to store large amounts of data.
- It allows easy retrieval and modification of data using **SQL queries**.

2. Data Integrity and Consistency

- Ensures **accuracy and reliability** of data through constraints like **primary keys, foreign keys, and unique constraints**.

- Enforces **ACID (Atomicity, Consistency, Isolation, Durability)** properties to maintain consistency.

3. Data Security and Access Control

- Controls user access with authentication and authorization mechanisms.
- Prevents **unauthorized access** using encryption, role-based permissions, and security policies.

4. Avoids Data Redundancy and Inconsistency

- Eliminates **duplicate data** using techniques like **normalization**.
- Ensures **data consistency** across multiple tables and relationships.

5. Multi-User Access and Concurrency Control

- Supports **multiple users accessing data simultaneously** without conflicts.
- Uses **locking mechanisms and transaction management** to prevent data corruption.

6. Data Backup and Recovery

- Provides automatic **backup and restore** mechanisms to prevent data loss.
- Ensures data can be **recovered after system failures** or accidental deletions.

7. Efficient Query Processing and Optimization

- Uses **indexes, caching, and query optimization** techniques to improve search performance.
- Reduces **query execution time** and enhances efficiency.

8. Relationship Management and Data Linking

- Allows defining relationships between different **entities (tables)** using **foreign keys**.
- Supports **relational databases** (RDBMS) for structured relationships.

9. Scalability and Performance Enhancement

- Handles **large-scale data** with distributed databases and cloud-based DBMS solutions.
- Optimized for **high-performance transactions** and analytics.

10. Decision Making and Reporting

- Used in **business intelligence (BI)** to generate reports and analyze trends.
- Supports **data warehousing and analytics** for better decision-making.

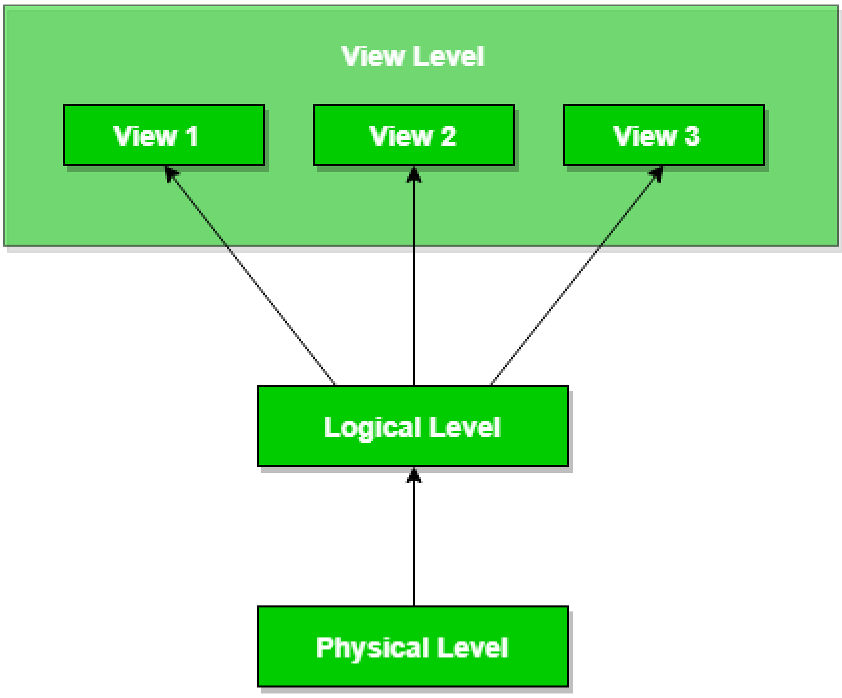
Data Abstraction in DBMS

Data abstraction in a **Database Management System (DBMS)** refers to hiding the complex details of database storage and presenting only the relevant information to users. It helps in **simplifying interactions with the database** by providing different **levels of abstraction**.

Levels of Data Abstraction

DBMS uses three levels of abstraction:

1. **Physical Level (Lowest Level)**
    - Describes how data is **physically stored** on storage devices (e.g., hard drives, SSDs).
    - Includes details like **file structures, indexing, and data access methods**.
    - Example: Data is stored as **binary files, B-trees, or hash indexes**.
2. **Logical Level (Middle Level)**
    - Defines what data is stored and the **relationships between the data**.
    - Uses **tables, attributes, constraints, and relationships** to organize data.
    - Example: A **student table** with attributes (Student_ID, Name, Age, Course).
3. **View Level (Highest Level)**
    - Provides a **customized view of the database** for users.
    - Hides **complexities** and shows only relevant data to different user groups.
    - Example: A **student portal** shows only student-related details, while an **admin panel** shows all database information.



Types of Database Users in DBMS

In a **Database Management System (DBMS)**, different types of users interact with the database for various purposes. These users can be categorized based on their roles and responsibilities.

1. Database Administrator (DBA)

- **Role**: Manages the overall database system.

- **Responsibilities**:
  - Database **installation, configuration, and maintenance**.
  - Ensures **security and access control**.
  - Performs **backups and recovery**.
  - Monitors **performance tuning** and optimizes database queries.

2. End Users

- **Role**: Use the database for their specific needs.
- **Types of End Users**:
  - **Casual Users**: Occasionally interact with the database using **queries or reports**. Example: A manager generating sales reports.
  - **Naïve Users**: Use pre-built applications without direct interaction with the database. Example: Online banking users.
  - **Sophisticated Users**: Perform complex queries and analysis. Example: Data scientists using SQL for analytics.

3. Application Programmers

- **Role**: Develop applications that interact with the database.
- **Responsibilities**:
  - Write **code using programming languages** (Java, Python, PHP, etc.) that interacts with the database.
  - Use **APIs like JDBC, ODBC** for database connectivity.
  - Ensure **efficient data retrieval and updates** through optimized queries.

4. System Analysts

- **Role**: Design and analyze database requirements for applications.
- **Responsibilities**:
  - Understand **business needs** and translate them into database models.
  - Work with developers and DBAs to ensure **database efficiency**.

5. Database Designers

- **Role**: Create and design the **database schema**.
- **Responsibilities**:
  - Define **tables, relationships, and constraints**.
  - Ensure **data normalization** to avoid redundancy.

6. Data Scientists & Analysts

- **Role**: Use the database for insights and decision-making.
- **Responsibilities**:
  - Perform **data mining, reporting, and analytics**.
  - Use tools like **SQL, Python, R** to extract and analyze data.

Data Definition Language (DDL) in DBMS

Data Definition Language (DDL) is a subset of **SQL (Structured Query Language)** used to define and manage the structure of a database. DDL statements **create, modify, and delete** database objects such as tables, schemas, indexes, and constraints.

Key DDL Commands

1. **CREATE** – Used to create a database object (e.g., table, view, index).

```
CREATE TABLE Students (
Student_ID INT PRIMARY KEY,
Name VARCHAR(50),
Age INT,
Course VARCHAR(50)
);
```

   - Creates a Students table with four columns.

2. **ALTER** – Modifies an existing database object.
   - **Add a new column**:

```
ALTER TABLE Students ADD Email VARCHAR(100);
```

   - **Modify an existing column**:

```
ALTER TABLE Students MODIFY Age SMALLINT;
```

   - **Drop a column**:

```
ALTER TABLE Students DROP COLUMN Course;
```

3. **DROP** – Deletes a database object permanently.
   - **Delete a table**:

```
DROP TABLE Students;
```

   - **Delete a database**:

```
DROP DATABASE College;
```

4. **TRUNCATE** – Removes all records from a table without deleting the structure.

```
TRUNCATE TABLE Students;
```

- Deletes all data but keeps the table intact.

5. **RENAME** – Changes the name of a database object.

RENAME TABLE Students TO Learners;

- Renames the Students table to Learners.

Data Dictionary in DBMS

A **Data Dictionary** is a centralized repository that stores **metadata** — information about the structure, organization, and characteristics of the database. It acts as a **catalog** for all the database objects and their properties.

Key Components of a Data Dictionary

A typical data dictionary contains the following details:

1. **Table Information**
   - Table name
   - Number of columns
   - Primary and foreign keys
2. **Column Information**
   - Column names
   - Data types (e.g., INT, VARCHAR, DATE)
   - Size/length
   - Constraints (e.g., NOT NULL, UNIQUE)
3. **Index Information**
   - Index names
   - Associated columns
   - Index types (e.g., B-tree, Hash)
4. **Relationship Details**
   - Foreign key references
   - Relationships between tables
5. **Views and Stored Procedures**
   - View definitions
   - Stored procedure names and logic
6. **User Access and Security**
   - User roles

   - Access privileges (e.g., READ, WRITE)

Example of Data Dictionary Entry

Attribute	**Description**
Table Name	Students
Column Name	Student_ID
Data Type	INT
Size	4 bytes
Constraints	PRIMARY KEY
Description	Unique identifier for each student

Data Models in DBMS

A **Data Model** defines how data is structured, stored, and manipulated in a **Database Management System (DBMS)**. It provides a **logical framework** for organizing data and relationships.

Types of Data Models

1. Hierarchical Data Model

- **Structure**: Data is organized in a **tree-like hierarchy** (parent-child relationship).
- **Example**: An organization's structure where a **manager (parent) has multiple employees (children)**.
- **Usage**: IBM Information Management System (IMS).
- **Pros**: Fast retrieval, good for hierarchical relationships.
- **Cons**: Rigid structure, difficult to modify.

👉 **Example:**

```
Company
|— HR (Parent)
|   |— Employee1 (Child)
|   |— Employee2 (Child)
|— IT (Parent)
  |— Employee3 (Child)
  |— Employee4 (Child)
```

2. Network Data Model

- **Structure**: Data is stored in a **graph structure** where a child can have multiple parents.

- **Example**: A **university database** where a **student** can be linked to multiple **courses**, and a **course** can have multiple **students**.
- **Usage**: Used in early **CODASYL databases**.
- **Pros**: More flexible than hierarchical models.
- **Cons**: Complex to manage relationships.

👉 **Example:**

Student <---> Course

| |

Teacher <-----> Department

3. Relational Data Model (RDBMS)

- **Structure**: Data is stored in **tables (relations)** with rows (records) and columns (attributes).
- **Example**: MySQL, PostgreSQL, Oracle, SQL Server.
- **Usage**: Most commonly used model today.
- **Pros**: Flexible, easy to use, supports **SQL queries**.
- **Cons**: Performance can be slower for complex queries.

👉 **Example:**
Students Table

Student_ID	Name	Age	Course
101	Alice	22	CS
102	Bob	21	IT

4. Entity-Relationship (E-R) Model

- **Structure**: Uses **entities (objects)** and **relationships** to model data.
- **Example**: A **bank database** where **customers (entities)** have **accounts (relationships)**.
- **Usage**: Useful for database **design and planning**.
- **Pros**: Visual representation, easy to understand.
- **Cons**: Needs conversion to a relational model for implementation.

👉 **Example:**

Customer ----> (Has) ----> Account

5. Object-Oriented Data Model

- **Structure**: Data is stored as **objects**, similar to object-oriented programming (OOP).
- **Example**: Multimedia databases, CAD applications.

- **Usage**: Used in **Object-Oriented DBMS (OODBMS)** like MongoDB.
- **Pros**: Good for complex data types (images, videos, etc.).
- **Cons**: Not widely used compared to relational models.

👉 **Example:**

```
{
 "Student": {
  "Name": "Alice",
  "Age": 22,
  "Courses": ["CS", "Math"]
 }
}
```

Comparison of Data Models

Model	Structure	Example	Use Case
Hierarchical	Tree structure	File systems, IMS	Organization charts
Network	Graph structure	CODASYL databases	Complex relationships
Relational	Tables (rows/cols)	MySQL, Oracle	General databases
E-R Model	Entities/relations	Database design tools	Conceptual modeling
Object-Oriented	Objects (OOP)	MongoDB, ObjectDB	Multimedia, Big Data

Data Independence in DBMS

Data Independence is the ability to modify the **schema** at one level of a database system without affecting the schema at the next higher level. It helps in **reducing dependency between data and applications**, ensuring flexibility and scalability.

Types of Data Independence

1. Logical Data Independence

- Refers to the ability to **change the logical schema** (database structure) without modifying the applications using the data.
- Example: Adding a new column to a table should not require changing existing application code.

✔ **Possible Changes:**
✅ Adding or removing attributes (columns).

☑ Modifying relationships between tables.
☑ Changing integrity constraints.

❌ **Challenges:**
🔷 Achieving full logical independence is difficult because queries may need modifications if table structures change significantly.

2. Physical Data Independence

- Refers to the ability to **change the physical storage** of data without affecting the logical schema.
- Example: Changing the **indexing method** or **storage format** does not require modification of the database schema.

✓ **Possible Changes:**
☑ Changing file organization (e.g., B-trees, hashing).
☑ Modifying storage devices (e.g., moving from SSD to cloud storage).
☑ Using indexing or partitioning to improve performance.

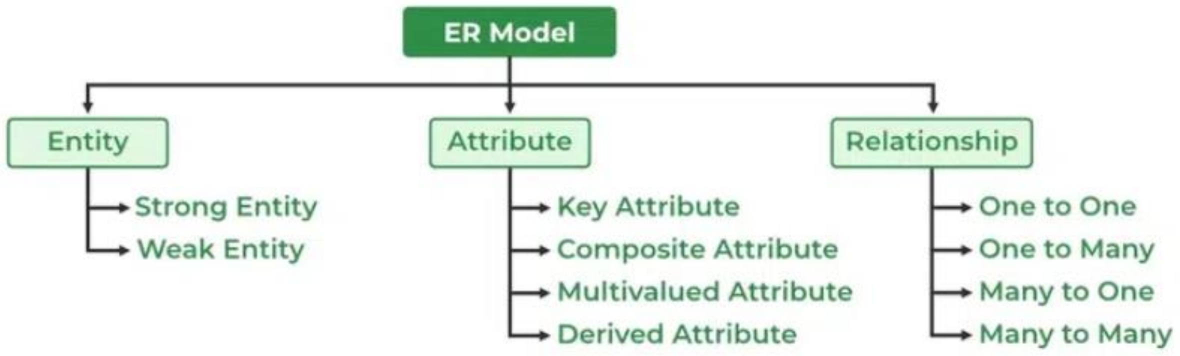
❌ **Challenges:**
🔷 Some performance optimizations may still require minor adjustments in queries.

ER Model

The Entity Relationship Model is a model for identifying entities (like student, car or company) to be represented in the database and representation of how those entities are related. The ER data model specifies enterprise schema that represents the overall logical structure of a database graphically.

Components of ER Diagram

ER Model consists of Entities, Attributes, and Relationships among Entities in a Database System.

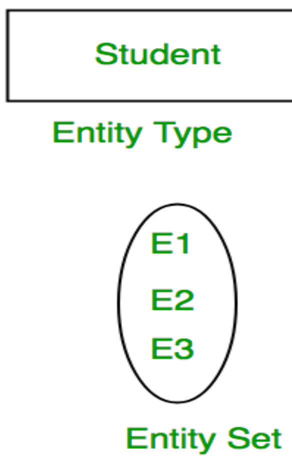


What is Entity?

An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

What is Entity Set?

An Entity is an object of Entity Type and a set of all entities is called an entity set. For Example, E1 is an entity having Entity Type Student and the set of all students is called Entity Set. In ER diagram, Entity Type is represented as:



Entity Set

We can represent the entity set in ER Diagram but can't represent entity in ER Diagram because entity is row and column in the relation and ER Diagram is graphical representation of data.

Types of Entity

There are two types of entity:

1. Strong Entity

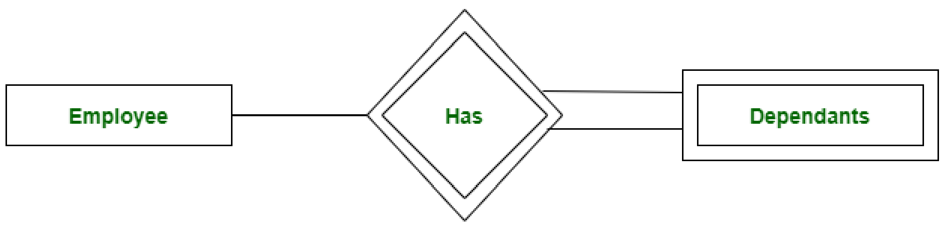
A Strong Entity is a type of entity that has a key Attribute. Strong Entity does not depend on other Entity in the Schema. It has a primary key, that helps in identifying it uniquely, and it is represented by a rectangle. These are called Strong Entity Types.

2. Weak Entity

An Entity type has a key attribute that uniquely identifies each entity in the entity set. But some entity type exists for which key attributes can't be defined. These are called Weak Entity types .

For 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** .

A weak entity type is 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.



What is Attributes?

Attributes are the properties that define the entity type. For example, 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.

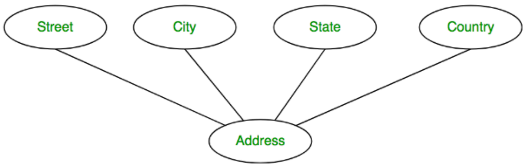
Types of Attributes

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 underlying lines.

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.



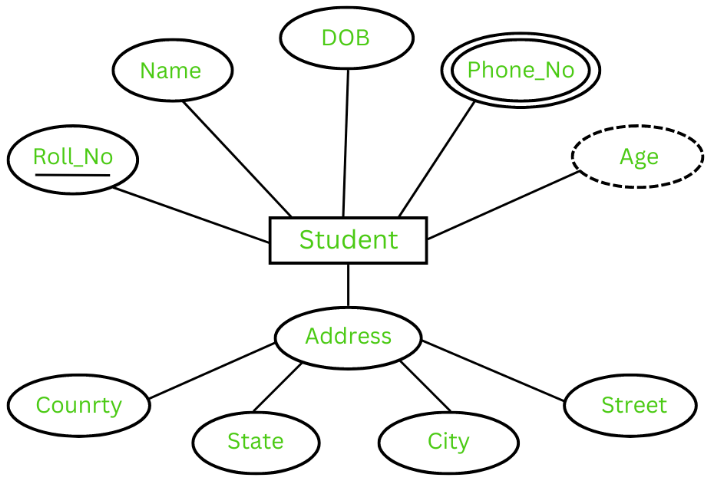
3. Multivalued Attribute

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.

4. Derived Attribute

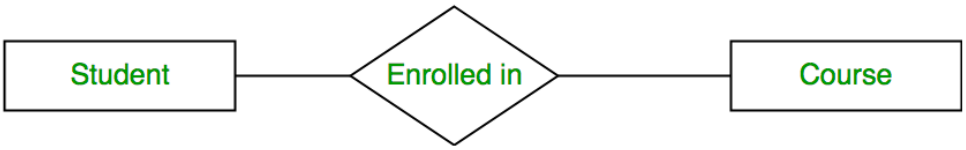
An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, the derived attribute is represented by a dashed oval.

The Complete Entity Type Student with its Attributes can be represented as:

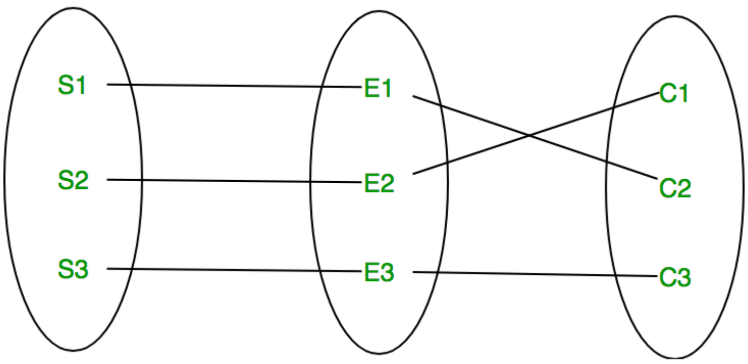


Relationship Type and Relationship Set

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.



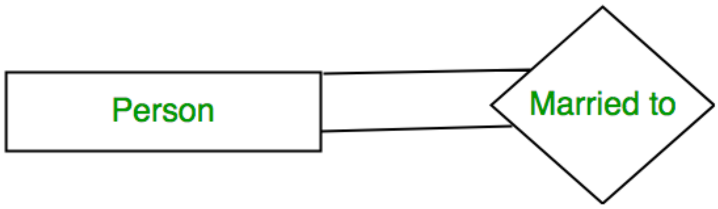
A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 as enrolled in C2, S2 as enrolled in C1, and S3 as registered in C3.



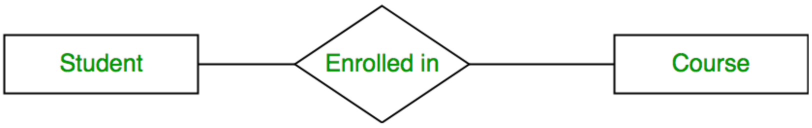
Degree of a Relationship Set

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

1. Unary 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.



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.



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.

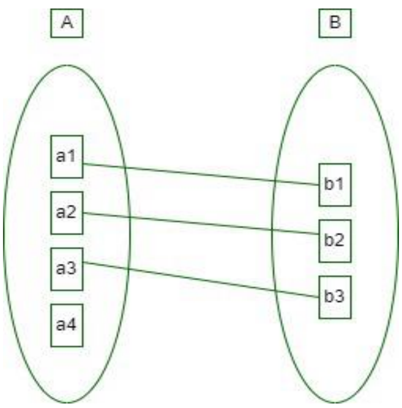
What is Cardinality?

The 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.

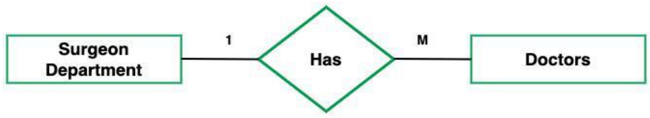
the total number of tables that can be used in this is 2.

Using Sets, it can be represented as:

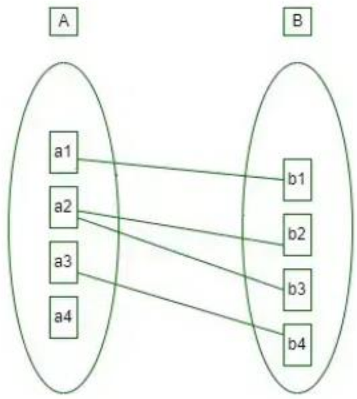


2. One-to-Many: In one-to-many mapping as well where each entity can be related to more than one entity and the total number of tables that can be used in this is 2. Let us assume that one surgeon department can accommodate many doctors. So the Cardinality will be 1 to M. It means one department has many Doctors.

total number of tables that can used is 3.



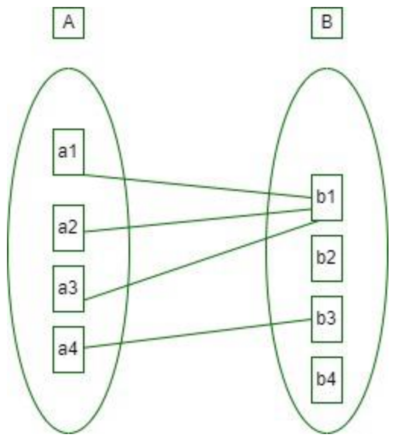
Using sets, one-to-many cardinality can be represented as:



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. 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.

The total number of tables that can be used in this is 3.

Using Sets, it can be represented as:



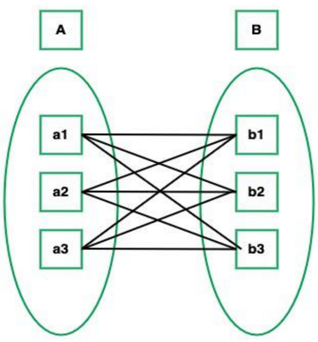
In this case, each student is taking only 1 course but 1 course has been taken by many students.

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.

the total number of tables that can be used in this is 3.

many to many cardinality

Using Sets, it can be represented as:



In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3, and S4. So it is many-to-many relationships.

Participation Constraint

[Participation Constraint](#)is applied to the entity participating in the relationship set.

1. Total Participation – Each entity in the entity set must participate in the relationship. If each student must enroll in a course, the participation of students will be total. Total participation is shown by a double line in the ER diagram.

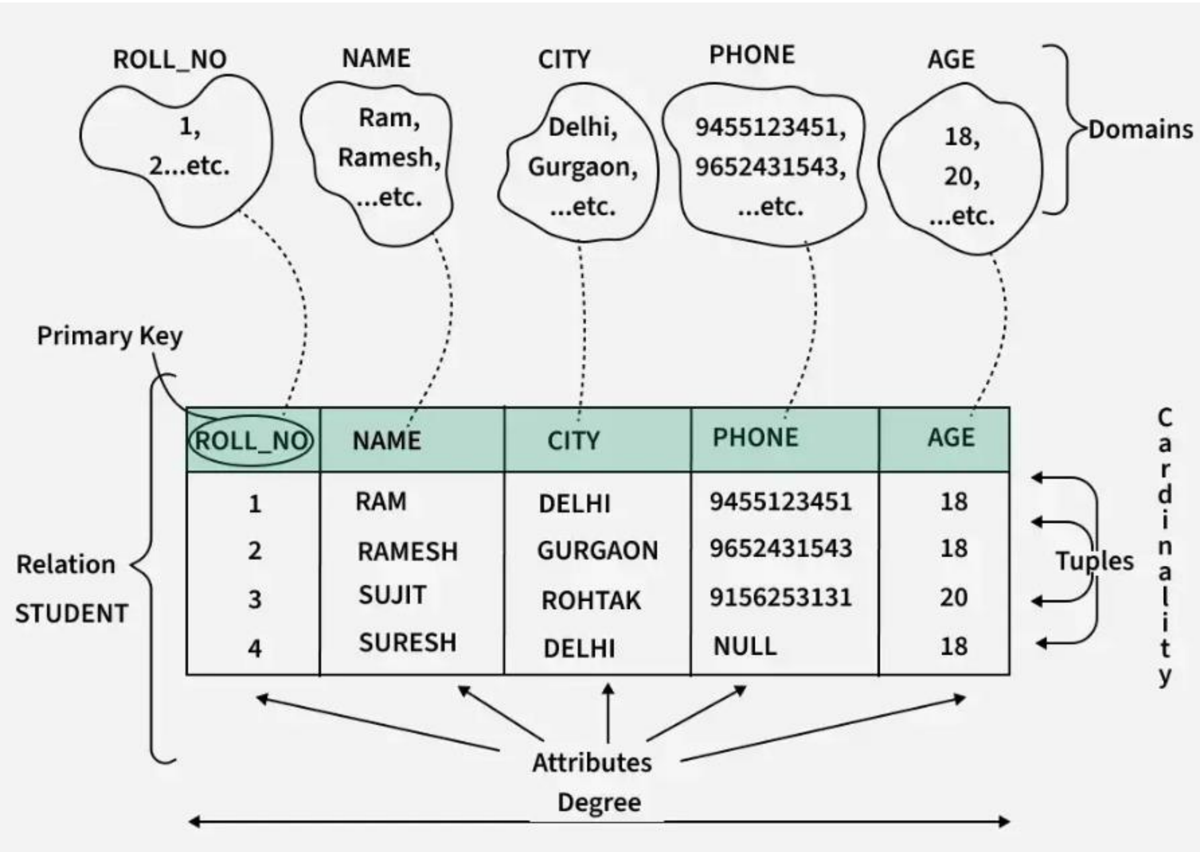
2. Partial Participation – The entity in the entity set may or may NOT participate in the relationship. If some courses are not enrolled by any of the students, the participation in the course will be partial.

Relational Model

The **Relational Model** represents data and their relationships through a collection of tables. Each table also known as a relation consists of rows and columns. Every column has a unique name and corresponds to a specific attribute, while each row contains a set of related data values representing a real-world entity or relationship. This model is part of the record-based models which structure data in fixed-format records each belonging to a particular type with a defined set of attributes.

The relational model represents how data is stored in **Relational Databases**. A relational database consists of a collection of tables each of which is assigned a unique name. Consider a relation STUDENT with attributes **ROLL_NO**, **NAME**, **ADDRESS**, **PHONE**, and **AGE** shown in the table.

Table STUDENT



Key Terms

- **Attribute:** Attributes are the properties that define an entity. e.g. ROLL_NO, NAME, ADDRESS.
- **Relation Schema:** A relation schema defines the structure of the relation and represents the name of the relation with its attributes. e.g. STUDENT (ROLL_NO, NAME, ADDRESS, PHONE, and AGE) is the relation schema for STUDENT. If a schema has more than 1 relation it is called Relational Schema.
- **Tuple:** Each row in the relation is known as a tuple. The above relation contains 4 tuples one of which is shown as:

1	RAM	DELHI	9455123451	18

- **Relation Instance:** The set of tuples of a relation at a particular instance of time is called a **relation instance**. It can change whenever there is an insertion, deletion or update in the database.
- **Degree:** The number of attributes in the relation is known as the degree of the relation. The STUDENT relation defined above has degree 5.
- **Cardinality:** The number of tuples in a relation is known as cardinality. The STUDENT relation defined above has cardinality 4.
- **Column:** The column represents the set of values for a particular attribute. The column ROLL_NO is extracted from the relation STUDENT.
- **NULL Values:** The value which is not known or unavailable is called a NULL value. It is represented by NULL. e.g. PHONE of STUDENT having ROLL_NO 4 is NULL.
- **Relation Key:** These are basically the keys that are used to identify the rows uniquely or also help in identifying tables. These are of the following types:
    - Primary Key
    - Candidate Key
    - Super Key
    - Foreign Key
    - Alternate Key
    - Composite Key

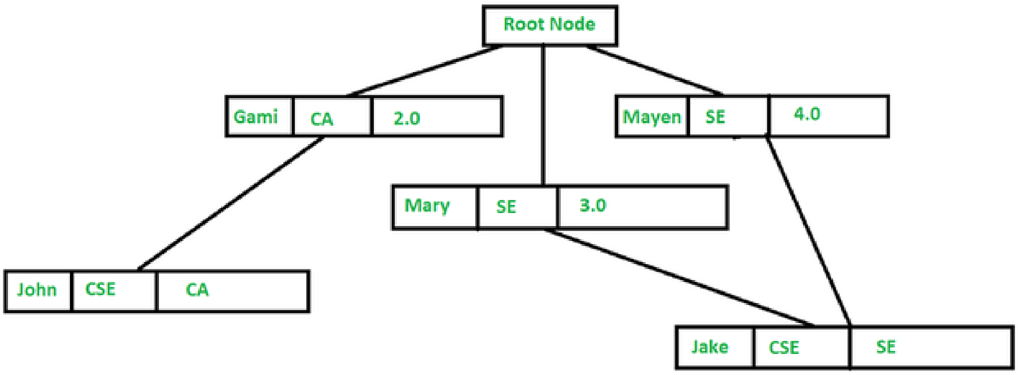
Hierarchical Model in DBMS

- The **hierarchical model** is a **tree-like** database model where data is organized in a hierarchy using **parent-child relationships**. It was one of the earliest models used in **DBMS**, mainly implemented in systems like **IBM's Information Management System (IMS)**.

Characteristics of the Hierarchical Model

1. **Tree Structure**: Data is represented in a **hierarchical (tree) structure** with a **single root node**.
2. **Parent-Child Relationship**:
   - Each **parent** can have multiple **children**.
   - Each **child** has only **one parent**.
3. **1:N Relationship**: A parent node can have many child nodes, but a child node has only one parent.
4. **Fast Data Retrieval**: Since records are stored in a predefined structure, queries are fast.
5. **Rigid Structure**: Modifying relationships can be complex due to the **fixed hierarchy**.

Example 1: Consider the below Student database system hierarchical model.



In the above-given figure, we have few students and few course-enroll and a course can be assigned to a single student only, but a student can enroll in any number of courses and with this the relationship becomes one-to-many. We can represent the given hierarchical model like the below relational tables:

FACULTY Table

Name	Dep	Course-taught
John	CSE	CA
Jake	CSE	SE
Royal	CSE	DBMS

STUDENT Table

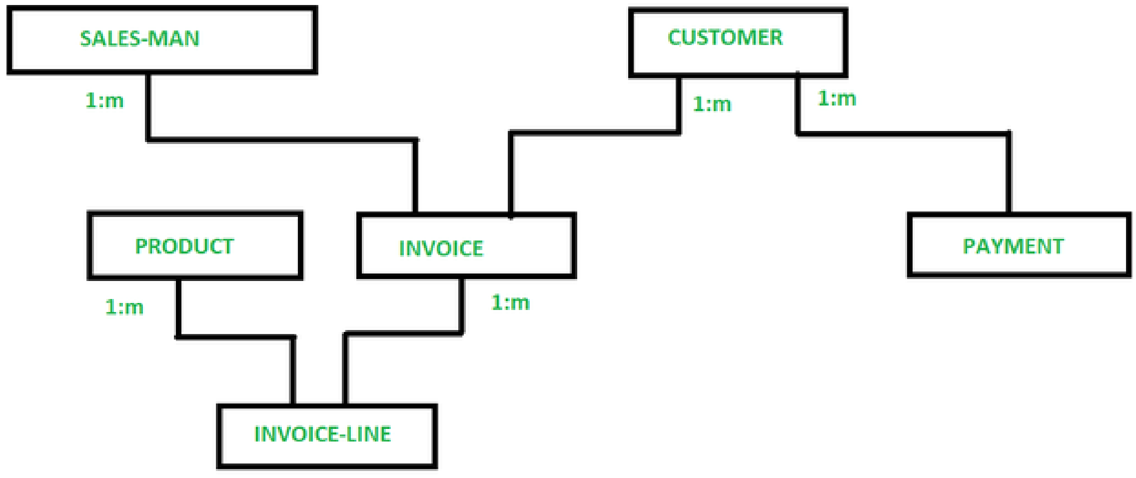
Name	Course-enroll	Grade
Gami	CA	2.0
Mary	SE	3.0
Mayen	SE	4.0

Network model

The Network Model in a Database Management System (DBMS) is a data model that allows the representation of many-to-many relationships in a more flexible and complex structure compared to the Hierarchical Model. It uses a graph structure consisting of nodes (entities) and edges (relationships) to organize data, enabling more efficient and direct access paths.

Example : Network model for a Finance Department.

Below we have designed the network model for a Finance Department:



So, in a network model, a one-to-many (1: N) relationship has a link between two record types. Now, in the above figure, SALES-MAN, CUSTOMER, PRODUCT, INVOICE, PAYMENT, INVOICE-LINE are the types of records for the sales of a company. Now, as you can see in the given figure, INVOICE-LINE is owned by PRODUCT & INVOICE. INVOICE has also two owners SALES-MAN & CUSTOMER.

Difference Between the Network Model and the Hierarchical Model

Feature	Hierarchical Model	Network Model
Structure	Tree-like structure	Graph structure
Relationships	One-to-many (single parent, multiple children)	Many-to-many (multiple parents and children)
Flexibility	Less flexible	More flexible
Data Access	Single access path	Multiple access paths
Redundancy	Higher redundancy due to rigid hierarchy	Lower redundancy due to shared relationships
Complexity	Simpler to design and implement	More complex to design and manage
Usage Scenario	Suitable for simple, hierarchical data structures	Suitable for complex, interconnected data structures
Efficiency	Efficient for hierarchical traversal	Efficient for complex queries and data retrieval
Example	Organizational chart	Telecommunications network

RDBMS?

RDBMS stands for Relational Database Management Systems. It is a program that allows us to create, delete, and update a relational database. A Relational Database is a database system that stores and retrieves data in a tabular format organized in the form of rows and columns. It is a smaller subset of DBMS which was designed by E.F Codd in the 1970s. The major DBMSs like SQL, My-SQL, and ORACLE are all based on the principles of relational DBMS.

Query in RDBMS

Creating a Table

Syntax:

CREATE TABLE table_name (

column1_name datatype constraint,

column2_name datatype constraint,

);

Example:

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

BirthDate DATE,

Salary DECIMAL(10, 2)

);

2. Inserting Data into a Table

Syntax:

INSERT INTO table_name (column1_name, column2_name, …)

VALUES (value1, value2, …);

Example:

INSERT INTO Employees (EmployeeID, FirstName, LastName, BirthDate, Salary)

VALUES (1, ‘John’, ‘Doe’, ‘1985-06-15’, 55000.00);

3. Querying Data (SELECT)

Syntax:

SELECT column1_name, column2_name, …

FROM table_name

WHERE condition;

Example:

```
SELECT FirstName, LastName, Salary

FROM Employees

WHERE Salary > 50000;
```

4. Deleting Data from a Table

Syntax:

```
DELETE FROM table_name

WHERE condition;
```

Example:

```
DELETE FROM Employees

WHERE EmployeeID = 1;
```

5. . Dropping a Table

Syntax:

```
DROP TABLE table_name;
```

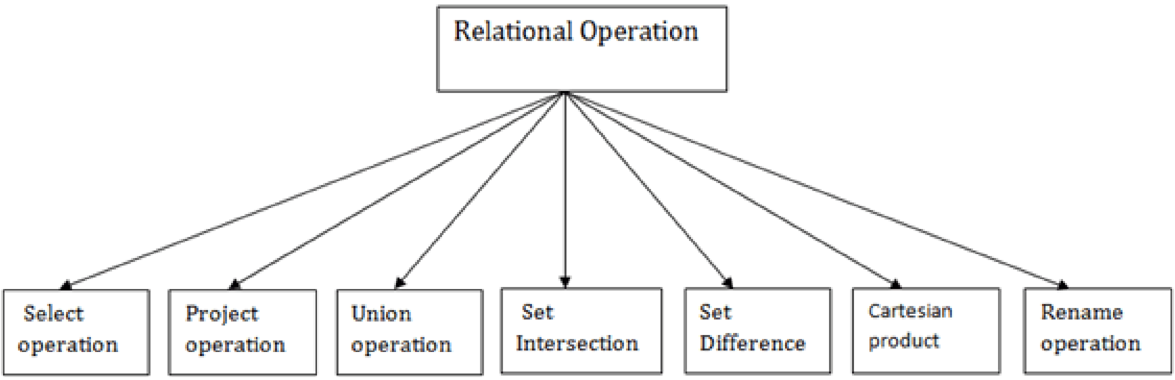
Example:

```
DROP TABLE Employees;
```

Relational Algebra

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

Types of Relational operation



1. Select Operation:

- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).

Notation: σ p(r)

Where:

σ is used for selection prediction
r is used for relation
p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500

Roundhill	L-11	900
Perryride	L-16	1300

Input:

1. σ BRANCH_NAME="perryride" (LOAN)

Output:

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

2. Project Operation:

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by ∏.

1. Notation: ∏ A1, A2, An (r)

Where

A1, **A2**, **A3** is used as an attribute name of relation **r**.

Example: CUSTOMER RELATION

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye

Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input:

1. ∏ NAME, CITY (CUSTOMER)

Output:

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

3. Union Operation:

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by ∪.

1. Notation: R ∪ S

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

Input:

1. ∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

4. Set Intersection:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection ∩.

1. Notation: R ∩ S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Smith
Jones

5. Set Difference:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).

1. Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Jackson
Hayes
Willians
Curry

6. Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.

1. Notation: E X D

Example:

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Input:

1. EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal

2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

7. Rename Operation:

The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

1. ρ(STUDENT1, STUDENT)

Functional Dependencies

A **functional dependency (FD)** is a constraint between two sets of attributes in a **relational database**. It **defines a relationship** where one attribute uniquely determines another attribute.

Mathematical Notation:

If **X → Y**, then:

- **X (determinant)** uniquely determines **Y (dependent attribute)**.
- If two rows have the same value of **X**, they must have the same value of **Y**.

Example:

Consider a **Student Table**:

Student_ID	Name	Age	Course
101	John	22	CS
102	Alice	21	IT

Here, **Student_ID → Name, Age, Course**, meaning:

- If two students have the same **Student_ID**, they must have the same **Name, Age, and Course**.

Types of Functional Dependencies

1. Trivial Functional Dependency

📌 **Rule**: If **Y ⊆ X**, then **X → Y** is trivial.

Example:

- {Student_ID, Name} → Name
- {Student_ID, Name} → {Student_ID, Name} (Always holds).

2. Non-Trivial Functional Dependency

📌 **Rule**: If **Y is not a subset of X**, then **X → Y** is non-trivial.

Example:

- Student_ID → Name (Name is not part of Student_ID).

3. Partial Functional Dependency

📌 **Rule**: If a **part of a composite key** determines a **non-key attribute**, then it's a **partial dependency**.

Example (Before 2NF - Partial Dependency exists):

Order_ID	Product	Customer	Order_Date
1	Laptop	John	2024-03-20
1	Mouse	John	2024-03-20

- **Composite Key**: {Order_ID, Product}
- **Partial Dependency**: Order_ID → Customer, Order_Date (Only Order_ID determines Customer and Order_Date, not Product).

Solution: Remove the dependency to achieve **2NF**.

4. Transitive Functional Dependency

📌 **Rule**: If **X** → **Y** and **Y** → **Z**, then **X** → **Z** (Indirect dependency).

Example (Before 3NF - Transitive Dependency exists):

Order_ID	Customer	Customer_Address
1	John	NY, USA

- **Transitive Dependency**: Order_ID → Customer → Customer_Address
- **Solution:** Remove transitive dependency to achieve **3NF**.

5. Multivalued Functional Dependency (MVD)

📌 **Rule**: If **X** →→ **Y**, then for each value of **X**, there is a **set of multiple values** of **Y**.

Example:

A **student** can have multiple **phone numbers** and **emails**.

Student_ID	Phone	Email
101	987654	john@gmail.com
101	123456	john@xyz.com

- **MVD:** Student_ID →→ Phone and Student_ID →→ Email.
- **Solution:** Create separate tables (**4NF**).

6. Join Dependency (JD)

📌 **Rule**: If a table can be split into smaller tables without losing data, it has a **join dependency**.

Example:

A **Company Table** has **Employees, Departments, Projects**.

- If splitting into Employee-Department and Department-Project does not lose information, it has a **join dependency** (5NF needed).

Closure of Functional Dependencies

📌 **Closure of X (X^+)** is the **set of all attributes** that can be functionally determined from **X**.

Example: Finding Closure

Given:

1. A → B
2. B → C
3. C → D

Find A^+:

1. **A** → **B** → {A, B}
2. **B** → **C** → {A, B, C}
3. **C** → **D** → {A, B, C, D}

So, **A^+ = {A, B, C, D}**.

Key Concept: Minimal Cover (Canonical Cover)

A **minimal cover** (canonical cover) is a simplified set of **functional dependencies**.

📌 **Steps to Find Minimal Cover:**

1. **Remove redundant FDs** (Minimize the set).
2. **Remove extraneous attributes** (Unnecessary parts of a determinant).
3. **Convert to singleton RHS** (Each FD should have only one attribute on the right side).

Functional Dependencies & Normalization

Functional Dependency	Related Normal Form
Partial Dependency	2NF
Transitive Dependency	3NF
Multivalued Dependency	4NF
Join Dependency	5NF

Real-World Example

📌 **Bank Database**

Account_ID	Customer	Balance	Branch
101	Alice	5000	NY

- **Functional Dependencies**:
  - Account_ID → Customer, Balance, Branch (One account has one customer).
  - Branch → City (Each branch is in one city).

Normalization

Normalization is the process of **organizing data** in a relational database to **reduce redundancy and improve data integrity**. It involves **dividing large tables** into smaller, related tables and defining relationships between them.

Purpose of Normalization:

✅ Eliminate **data redundancy** (duplicate data).
✅ Ensure **data consistency** (avoid anomalies).
✅ Improve **data integrity** (accurate and reliable data).
✅ Reduce **insertion, update, and deletion anomalies**.

Types of Anomalies in an Unnormalized Database

1. **Insertion Anomaly** → Difficulty adding new data without including unnecessary data.
2. **Update Anomaly** → Changing data in multiple places leads to inconsistency.
3. **Deletion Anomaly** → Removing data accidentally deletes valuable information.

Normalization Forms (NF)

Normalization is done in **stages**, called **Normal Forms (NF)**. The most common forms are:

1. First Normal Form (1NF) - Eliminating Repeating Groups

📌 **Rule**:

- Each column should have **atomic (indivisible) values**.
- Each column should contain **only one value per row** (no **repeating groups**).

❌ **Unnormalized Table (UNF)**

Order_ID	Customer	Products	Quantity
1	John	Laptop, Mouse	1, 2
2	Alice	Phone, Charger	1, 1

✅ **1NF Table (Atomic Values)**

Order_ID	Customer	Product	Quantity
1	John	Laptop	1
1	John	Mouse	2

Order_ID	Customer	Product	Quantity
2	Alice	Phone	1
2	Alice	Charger	1

2. Second Normal Form (2NF) - Removing Partial Dependencies

📌 **Rule**:

- **Must be in 1NF**.
- Every **non-key column** must be **fully dependent** on the **entire primary key** (no **partial dependency**).

❌ **1NF Table (Partial Dependency)**

Order_ID (PK)	Product (PK)	Customer	Order_Date
1	Laptop	John	2024-03-20
1	Mouse	John	2024-03-20

- **Problem:** Customer and Order_Date depend only on Order_ID, not Product.

✅ **2NF Tables (Breaking into Two Tables)**

Order Table

Order_ID (PK)	Customer	Order_Date
1	John	2024-03-20

Order_Details Table

Order_ID (FK)	Product	Quantity
1	Laptop	1
1	Mouse	2

Now, **each non-key column is fully dependent on the entire primary key**.

3. Third Normal Form (3NF) - Removing Transitive Dependencies

📌 **Rule**:

- **Must be in 2NF**.
- **No transitive dependency** → A **non-key column** should **not depend on another non-key column**.

❌ **2NF Table (Transitive Dependency)**

Order_ID (PK)	Customer	Customer_Address	Order_Date
1	John	NY, USA	2024-03-20

- **Problem:** Customer_Address depends on Customer, not Order_ID.

✅ **3NF Tables (Removing Transitive Dependency)**

Customer Table

Customer_ID (PK)	Customer_Name	Customer_Address
C1	John	NY, USA

Order Table

Order_ID (PK)	Customer_ID (FK)	Order_Date
1	C1	2024-03-20

Now, **each non-key attribute depends only on the primary key**.

Higher Normal Forms

📌 **4NF (Fourth Normal Form)**: Removes **multi-valued dependencies**.
📌 **5NF (Fifth Normal Form)**: Removes **join dependencies**.
📌 **BCNF (Boyce-Codd Normal Form)**: Stronger form of 3NF (removes anomalies in complex relationships).

Advantages of Normalization

✅ **Reduces Data Redundancy** (No duplicate data).
✅ **Improves Data Integrity** (Data is consistent and accurate).
✅ **Eliminates Anomalies** (Insertion, update, and deletion anomalies).
✅ **Efficient Storage** (Minimizes unnecessary data).

Disadvantages of Normalization

❌ **Complex Queries** (More joins due to table division).
❌ **Performance Overhead** (Joins slow down retrieval).
❌ **Increased Design Complexity** (Difficult to manage in large databases).

Comparison of Normal Forms

Normal Form	Main Rule
1NF	No repeating groups, atomic values
2NF	No partial dependency

Normal Form	Main Rule
3NF	No transitive dependency
BCNF	Stronger 3NF (No functional dependencies in candidate keys)
4NF	No multi-valued dependencies
5NF	No join dependencies

Would you like **real-world examples** or **SQL queries** for normalization? 🚀