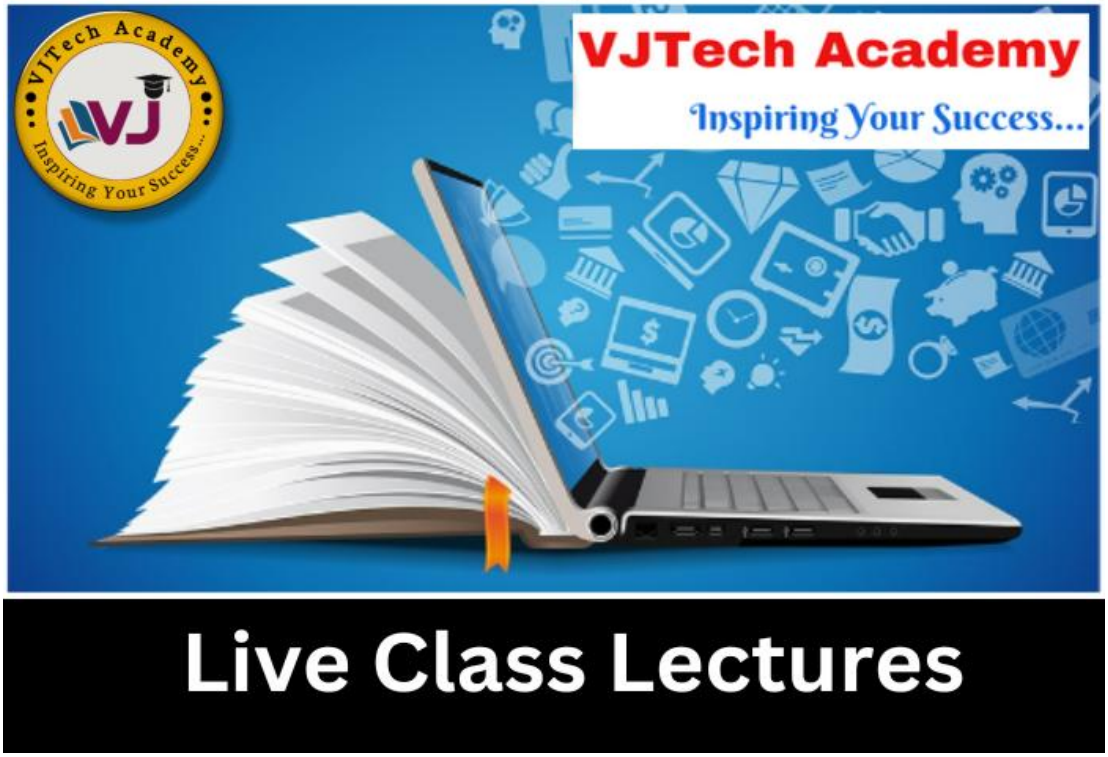
विशाल जाधव सरांचे

VJTech Academy

Inspiring Your Success...

UNIT - 2

RELATIONAL DATA MODEL



Syllabus:

Sr. No.	Main Topic	Subtopics
2.1	**Relational Structure**	– Tables (Relations) – Rows (Tuples) – Domains – Attributes – Entities
2.2	**Keys**	– Super Key – Candidate Key – Primary Key – Foreign Key – Examples and Differences
2.3	**Data Constraints**	– Domain Constraints – Referential Integrity Constraints
2.4	**Entity Relationship Model (ER Model)**	– Strong Entity Set – Weak Entity Set – Types of Attributes – Symbols for ER Diagram – Drawing ER Diagrams
2.5	**Normalization**	– Functional Dependencies – First Normal Form (1NF) – Second Normal Form (2NF) – Third Normal Form (3NF)

Introduction:

The **Relational Data Model (RDM)** is the **most widely used data model in DBMS** for organizing and managing large amounts of structured data efficiently.

In the relational model:

- **Data is stored in tables (relations)** consisting of **rows (tuples) and columns (attributes)**.
- Relationships between data are maintained using **keys and constraints**.
- The **relational model uses mathematical concepts of set theory and relational algebra** for systematic data operations.
- It provides **data independence, simplicity, and flexibility**, making it the backbone of modern database systems.

The relational model was introduced by **Dr. E.F. Codd in 1970**, setting **twelve rules (Codd's Rules) for an ideal relational database**, emphasizing **data consistency, integrity, and ease of use.**

2.1 Relational Structure

The relational structure is the foundation of the relational data model, where data is organized into relations (tables) to represent real-world entities and their relationships systematically.

- Each relation (table) consists of:
  - **Rows (tuples):** Representing individual records.
  - **Columns (attributes):** Representing the properties of the entity.
- Data is stored in a structured, tabular format, ensuring simplicity and flexibility.

➢ **Key Characteristics:**

- **Tabular Representation:** All data is stored in **tables with rows and columns**.
- **Uniqueness:** Each table can have a **primary key** to uniquely identify each row.
- **Data Independence:** Allows changes in structure without affecting data access.
- **Ease of Access:** Data retrieval is easy using **SQL queries**.
- **Relationships:** Tables can be linked using **keys (primary and foreign keys)**.

2.1.1 Tables (Relations)

A **table (also called a relation)** in the **relational data model** is a **two-dimensional structure** used to **store data about a particular entity systematically** in **rows and columns**.

- **Relation = Table**
- Represents **one type of entity** (e.g., Student, Employee).
- Each **row (tuple)** represents a **single record**.
- Each **column (attribute)** represents a **property of the entity**.

➢ **Structure of a Table:**

- **Columns (Attributes):**
  - Each column has a **unique name** within the table.
  - Defines the **type of data** that can be stored (e.g., integer, varchar).
- **Rows (Tuples):**
  - Each row represents **a unique record** in the table.
  - Contains data values corresponding to each attribute.
- **Domain:**
  - The **set of valid values for each attribute** (e.g., Age: 0–100).

➢ **Example: Student Table**

Roll_No	Name	Course	Year
101	Ravi	DBMS	TY
102	Meena	Java	SY
103	Anjali	DBMS	TY

- **Relation:** Student
- **Attributes:** Roll_No, Name, Course, Year
- **Tuples:** Each row containing details of a student.

- **Properties of a Relation (Table):**
  1. Unique Name: Each table has a distinct name in the database.
  2. Atomic Values: Each cell contains a single value (no multivalued attributes).
  3. Uniqueness of Tuples: No two rows are exactly identical.
  4. Order Irrelevance: The order of rows and columns does not affect the relation.
  5. Values in a Column: All values in a column come from the same domain.

2.1.2 Rows (Tuple)

In the relational data model, a row is known as a tuple.

A tuple represents a single, complete record in a table, containing specific data values for each attribute (column) of that table.

- **Key Points:**

1. Each **tuple corresponds to one instance of the entity** the table represents.
2. All values in a tuple are **atomic** (indivisible).
3. Each tuple **follows the structure of the relation** by maintaining the same set of attributes.
4. No two tuples in a relation are **identical**, ensuring **uniqueness** within the table.

- **Example:** Consider the **Student** Table

Roll_No	Name	Course	Year
101	Ravi	DBMS	TY

*The **row: (101, Ravi, DBMS, TY)** is a **tuple** representing **one student's data** in the STUDENT table.*

2.1.3 Domains

A **domain** in DBMS refers to the **set of all possible values that an attribute can take** in a table. It defines the **data type, format, and range of values** that can be stored in a column, ensuring **data validity and consistency**.

- **Example:**
  - The domain of **Age** can be integers from 0 to 120.
  - The domain of **Email** will be a string matching email format.

2.1.4 Attributes

An **attribute** is a **column in a table** representing a **property or characteristic of an entity**. Each attribute has a **name, associated domain, and holds atomic values**. Attributes define what type of data will be stored in the table.

- **Example:**

In a STUDENT table:

- Attributes: Roll_No, Name, Course, Year.

2.1.5 Entities

An **entity** is **anything in the real world that is distinguishable and about which data is to be stored in the database**.

Entities are **represented as rows in tables**, and the collection of similar entities forms an **entity set**.

- **Example:**
  - A **Student** with Roll_No 101, Name Ravi is an entity.
  - All students in a college form an entity set called **STUDENT**.

2.2 Keys In DBMS

In the relational data model, keys are essential to uniquely identify tuples (rows) in a relation (table) and to establish relationships between different tables. They help in maintaining data integrity, consistency, and enable efficient data retrieval in DBMS. Various types of keys are used in databases, each serving a specific purpose in identifying and linking data within and across relations.

- List Of Keys:
  - 2.2.1 Super Key
  - 2.2.2 Candidate Key
  - 2.2.3 Primary Key
  - 2.2.4 Alternate Key
  - 2.2.5 Foreign Key
  - 2.2.6 Composite Key

2.2.1 Super Key

A Super Key is a set of one or more attributes that can uniquely identify each tuple (row) in a relation (table) within a database.

- It ensures that no two rows in the table have the same values for the attributes included in the key.
- It may consist of a single attribute or a combination of attributes to maintain uniqueness.
- Super Keys are used to define Candidate Keys and Primary Keys in a table.

- **Example:** Consider a Student Table

Roll_No	Name	Course	Mobile_No
101	Ravi	DBMS	9876543210
102	Meena	Java	9988776655

Examples of Super Keys:

- {Roll_No}, {Roll_No, Name}
- {Mobile_No}, {Roll_No, Course, Mobile_No}

*Here, **{Roll_No}** and **{Mobile_No}** are sufficient alone to identify each student uniquely, while **{Roll_No, Name}** is also a Super Key but includes redundant attributes.*

2.2.2 Candidate Key

A Candidate Key is a minimal Super Key — a set of one or more attributes that can uniquely identify each tuple in a relation without containing any unnecessary attributes.

- A Candidate Key is the smallest possible combination of attributes that can uniquely identify each row.
- It does not include any extra attributes that do not contribute to uniqueness.
- A relation can have one or more Candidate Keys.
- Out of all Candidate Keys, one is chosen as the Primary Key, while the others are called Alternate Keys.

➢ **Example:**

Roll_No	**Name**	**Course**	**Mobile_No**
101	Ravi	DBMS	9876543210
102	Meena	Java	9988776655

Possible Candidate Keys:

- **{Roll_No}** — uniquely identifies each student.
- **{Mobile_No}** — also unique for each student.

Since both can independently identify tuples and are minimal, both are ***Candidate Keys****.*

2.2.3 Primary Key

A Primary Key is a selected Candidate Key that uniquely identifies each tuple (row) in a relation (table) and does not allow NULL values.

- It can be a single attribute or a combination of attributes (composite key).
- A table can have only one Primary Key, selected from the available Candidate Keys.
- The primary key of one table is used as a foreign key in another table to maintain referential integrity.

➢ **Example:**

Roll_No(PK)	**Name**	**Course**	**Mobile_No** (PK)
101	Ravi	DBMS	9876543210
102	Meena	Java	9988776655

Possible Candidate Keys: {Roll_No}, {Mobile_No}

If we choose **{Roll_No}** as the **Primary Key**, it will uniquely identify each student, and no Roll_No can be NULL or duplicate in the table.

2.2.4 Foreign Key

A Foreign Key is an attribute or a set of attributes in one table that refers to the Primary Key of another table, establishing a relationship between the two tables.

- Foreign keys are used to maintain referential integrity between related tables in a database.
- The table containing the primary key is called the parent (referenced) table, while the table containing the foreign key is called the child (referencing) table.
- The values in the foreign key column must either match values in the parent table's primary key or be NULL.
- Foreign keys help in performing JOIN operations between related tables for data retrieval.

➢ **Example:**

1. **Parent Table (DEPARTMENT):**

Dept_ID(PK)	**Dept_Name**
10	CS
20	IT

2. **Child Table (STUDENT):**

Roll_No(PK)	**Name**	**Dept_ID(FK)**
101	Ravi	10
102	Meena	20

- Here, *Dept_ID* in *STUDENT* is a *Foreign Key* referencing *Dept_ID* in *DEPARTMENT*.
- It ensures only valid *Dept_ID* values present in *DEPARTMENT* can be used in *STUDENT*.

2.2.5 Composite Key

A Composite Key is a combination of two or more attributes that together uniquely identify each tuple (row) in a relation when a single attribute is insufficient for uniqueness.

- Used when a single attribute cannot uniquely identify records in a table.
- All attributes in the composite key are necessary to ensure uniqueness.
- Can also act as a Primary Key if it uniquely identifies each record without allowing NULL values.
- Widely used in associative (junction) tables in many-to-many relationships.

➢ **Example:** Consider the **ENROLLMENT table** (for student-course mapping)

Roll_No	Course_Code	Grade
101	DB101	A
101	CS102	B
102	DB101	B+

- Here, **Roll_No alone is not unique, and Course_Code alone is not unique.**
- The combination **{Roll_No, Course_Code} forms a Composite Key** as it uniquely identifies each enrollment record.

➢ **Summary of Keys in DBMS**

- Keys are attributes or sets of attributes used to uniquely identify records in a table and to establish relationships between tables.
- **Super Key:** Any combination of attributes that can uniquely identify rows (may have extra attributes).
- **Candidate Key:** Minimal Super Key without redundant attributes.
- **Primary Key:** Selected Candidate Key used to uniquely identify records; cannot have NULL values.
- **Alternate Key:** Candidate Keys not chosen as the Primary Key.
- **Foreign Key:** Attribute in one table referencing the Primary Key of another table to maintain relationships.
- **Composite Key:** Combination of two or more attributes together to uniquely identify records.

2.3 Data Constraints

Data constraints in DBMS are **rules applied on data in tables to ensure accuracy, validity, and integrity of the stored data**. They prevent invalid data entry and maintain consistency within the database, ensuring that the data adheres to predefined business rules and structure during insertion, updating, and deletion operations.

Constraints are crucial in maintaining **data reliability** within relational databases, as they control the type of data that can be stored in each column and how data across different tables relate to each other.

- **Types Of Data Constraints:**
  1. Domain Constraints
  2. Referential Integrity Constraints
  3. Key Constraints (optional but important)

---

1. Domain Constraints:

Domain constraints in DBMS specify **the permissible set of values that an attribute (column) can hold in a table**. They define the **data type, format, and range of values** that can be stored, ensuring **data validity and consistency**.

- **Key Points**
  - A **domain** defines the valid set of values for an attribute.
  - Ensures that **only valid data is inserted** into the table, preventing invalid entries.
  - Enforced using **data types (INTEGER, CHAR, DATE), value ranges, formats, and allowable values.**
  - Helps in maintaining **data integrity and accuracy** in a database.
- **Example:**

Roll_No	Name	Age	Email
101	Ravi	20	ravi@gmail.com
102	Meena	19	meena@gmail.com

- **Domain Constraints applied:**
  - **Roll_No:** INTEGER, > 0
  - **Name:** VARCHAR(50), only alphabets allowed
  - **Age:** INTEGER, between 17 and 30
  - **Email:** Valid email format (@ and domain)
- **Illustration: Invalid Entry Example:** If a user tries to insert:

Roll_No	Name	Age	Email
-5	1234	50	ravi#gmailcom

- **Violations:**
  - Roll_No = -5 violates domain (should be > 0).
  - Name = 1234 violates domain (should be alphabets).
  - Age = 50 violates domain (should be between 17 and 30).
  - Email is in an invalid format.

Such entries will be **rejected by the DBMS due to domain constraints.**

---

2. Referential Integrity Constraints

Referential Integrity Constraints in DBMS ensure that relationships between tables remain consistent, maintaining valid references between primary keys in the parent table and foreign keys in the child table.

They prevent actions that would leave foreign key references dangling (pointing to non-existent records), thus maintaining data integrity across related tables.

- **Key Points**
  - Enforced using **foreign keys referencing primary keys** in another table.
  - Ensures **every foreign key value in the child table matches a primary key value in the parent table or is NULL**.
  - Prevents **deletion of referenced data in the parent table** if dependent data exists in the child table.
  - Prevents **insertion of invalid foreign key values** in the child table.
  - Helps maintain **logical consistency across tables in relational databases**.

- **Example: Parent Table: DEPARTMENT**

Dept_ID	Dept_Name
10	CS
20	IT

- **Child Table: STUDENT**

Roll_No	Name	Dept_ID
101	Ravi	10
102	Meena	20
103	Anjali	30

- **Explanation:**
  - *Dept_ID* in *STUDENT* is a foreign key referencing *Dept_ID* in *DEPARTMENT*.
  - The value 30 in STUDENT is invalid as it ***does not exist in DEPARTMENT***, violating referential integrity.

---

3. Key Constraints

Key constraints in DBMS ensure that attributes designated as keys (primary key, candidate key) maintain uniqueness and non-nullability within a table**,** allowing each tuple (row) to be uniquely identified**.**

They are critical for maintaining data integrity and preventing duplicate or incomplete records in the database**.**

- **Key Points**
  - Applied to **Primary Keys and Candidate Keys** in a table.
  - **Uniqueness:** No two rows can have the same value for key attributes.
  - **Not Null:** Key attributes **cannot have NULL values** to ensure valid identification of each record.
  - Helps in **indexing and fast data retrieval**.
  - Ensures **reliable relationships** between tables when used as foreign keys in other tables.

- **Example:** Consider the **EMPLOYEE table:**

Emp_ID	Name	Department
501	Anil	HR
502	Sunita	IT
502	Rajesh	Finance

- **Explanation:**
  - **Emp_ID is defined as the Primary Key.**
  - The repeated value 502 violates the **key constraint** of uniqueness.

Similarly, if:

Emp_ID	Name	Department
NULL	Anil	HR

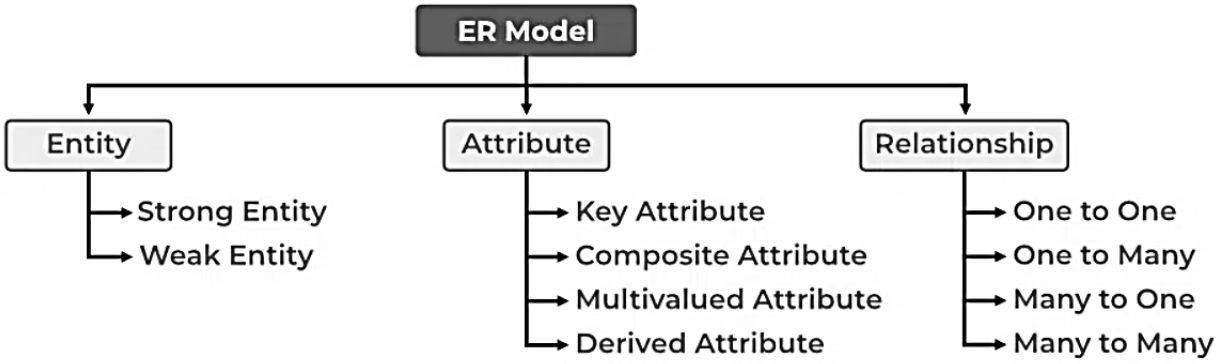
It violates the **key constraint** as primary keys cannot have NULL values.

2.4 Entity Relationship Model (ER Model)

- The Entity Relationship (ER) Model is a conceptual data model used to design and represent the structure of a database systematically. It visually describes how data is connected and how it will be stored in the database using an ER Diagram.
- The ER Model was introduced by Peter Chen in 1976 to help designers communicate with stakeholders clearly during the database design phase. It focuses on representing real-world objects (entities), their properties (attributes), and the associations (relationships) between them.
- ER Diagrams act as blueprints for database design and help in transforming user requirements into a relational schema easily.

➢ **Main Components of ER Diagram (Model):**

1. **Entities** – Real-world objects or concepts that have an independent existence in the system (e.g., Student, Course, Employee).
2. **Attributes** – Properties or characteristics that describe entities (e.g., Name, Age, CourseName).
3. **Relationships** – Associations that define how entities are related to each other (e.g., A Student *enrolls* in a Course, An Employee *works in* a department).



2.4.1 Entity

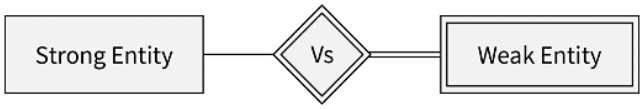
- An **Entity** is a **real-world object, person, place, concept, or event about which data is stored in a database**. It must have a **distinct existence** in the real world and can be either physical (e.g., Student, Book) or abstract (e.g., Department, Course).
- In an **ER Diagram**, an entity is **represented by a rectangle** and is labelled with the entity's name. Each entity is described using **attributes**, which define its properties.
- Entities are the **core building blocks** in the ER Model as they hold meaningful data and connect through relationships to represent how data elements interact in the database.

➢ **Types of Entities:** Entities are mainly classified into two types:

1. **Strong Entity:**
   - A **Strong Entity** is an entity that **has a primary key of its own**.
   - It can be **uniquely identified by its own attributes**, without relying on any other entity.
   - Represented in an ER diagram with a **single-lined rectangle**.
   - **Example:**
     - **STUDENT** entity with Roll_No as Primary Key.
     - **DEPARTMENT** entity with Dept_ID as Primary Key.

---

2. **Weak Entity:**
   - A **Weak Entity cannot be uniquely identified by its own attributes alone**.
   - It **depends on a Strong Entity** for its identification and is always related to it through a **Identifying Relationship**.
   - Weak Entities have a **partial key** and rely on the Primary Key of the Strong Entity.
   - Represented in an ER diagram with a **double-lined rectangle**, and its identifying relationship with a **double-lined diamond**.
   - **Example: DEPENDENT** entity in an Employee-Dependent scenario.
     - *Dependent* depends on *Employee* for identification.
     - The combination of Employee_ID (from EMPLOYEE) and Dependent_Name can uniquely identify each Dependent.



2.4.2 Attributes

- **Attributes** are **properties that describe the characteristics of an entity** in a database. They store values that provide detailed information about each entity instance.
- In an **ER Diagram**, attributes are shown as **ellipses connected to the entity rectangle**. They help in **clearly structuring the data of the entity for effective database design**.
- For example, for the **STUDENT** entity, attributes could include *Roll_No, Name, Age, and Course*.

➢ **Types of Attributes:**Attributes are classified into the following types with clear distinctions:

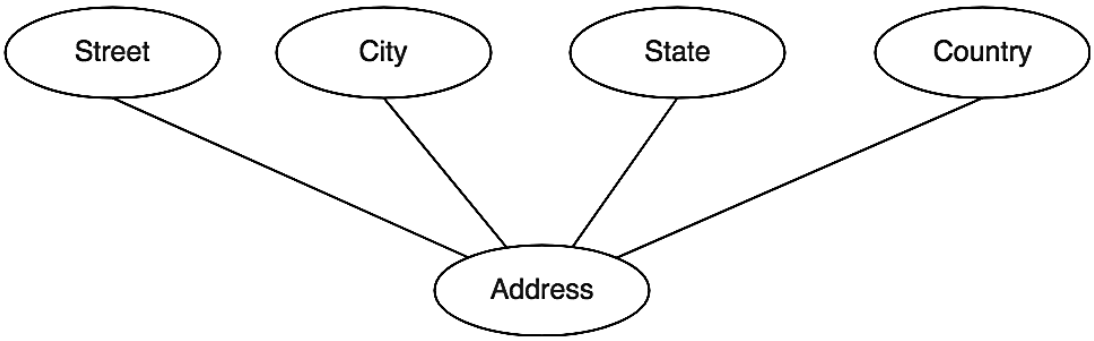
---

1. Simple Attribute

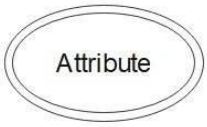
- These attributes **cannot be divided further into smaller parts**.
- They represent **atomic values**, ensuring data simplicity.
- Example: *Roll_No, Age* are simple attributes, as they store values that cannot be logically split further.

2. Composite Attribute

- These attributes can be **divided into smaller sub-parts**, where each sub-part also has meaning.
- Useful when the data needs to be stored in structured subfields for flexibility.
- Example: *Address* can be divided into *Street, City, State, and Pincode*. Each part can be stored and queried separately if needed.

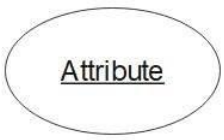


3. **Single-Valued Attribute**
   - An attribute that **can hold only a single value for a particular entity at a time**.
   - It ensures clear, non-redundant storage of attribute values.
   - Example: *Date_of_Birth* of a student will have only one valid date.
4. **Multi-Valued Attribute**
   - An attribute that can **hold multiple values for a single entity**.
   - Represented in ER diagrams using **double ellipses**.
   - Requires special handling in relational models using separate tables to store multiple values.
   - Example: *Phone_Numbers* for an employee, as an employee can have multiple contact numbers.

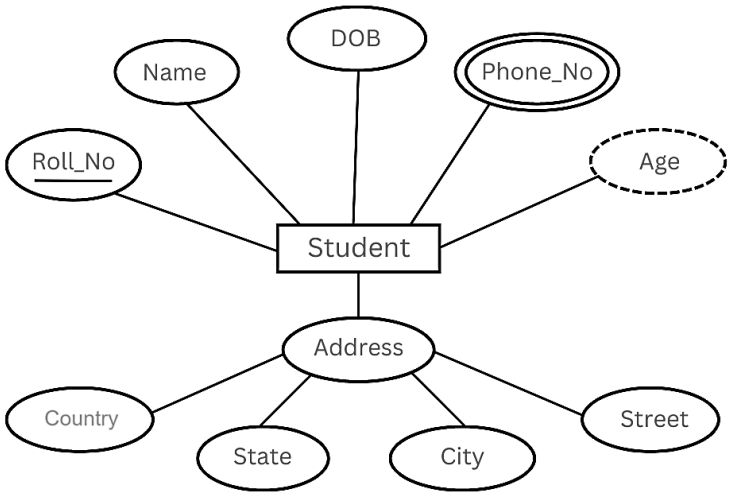


5. **Derived Attribute**
   - An attribute whose value can be **derived or calculated from other stored attributes**.
   - Represented by a **dashed ellipse in ER diagrams**.
   - Derived attributes reduce redundancy by not storing values that can be computed when needed.
   - Example: *Age* can be derived using *Current Date - Date_of_Birth*.

6. **Key Attribute**
   - An attribute that **uniquely identifies an entity within an entity set**.
   - Helps maintain uniqueness and is essential for primary key identification during relational mapping.
   - Represented by **underlining the attribute in ER diagrams**.
   - Example: *Roll_No* in STUDENT, as it uniquely identifies each student.



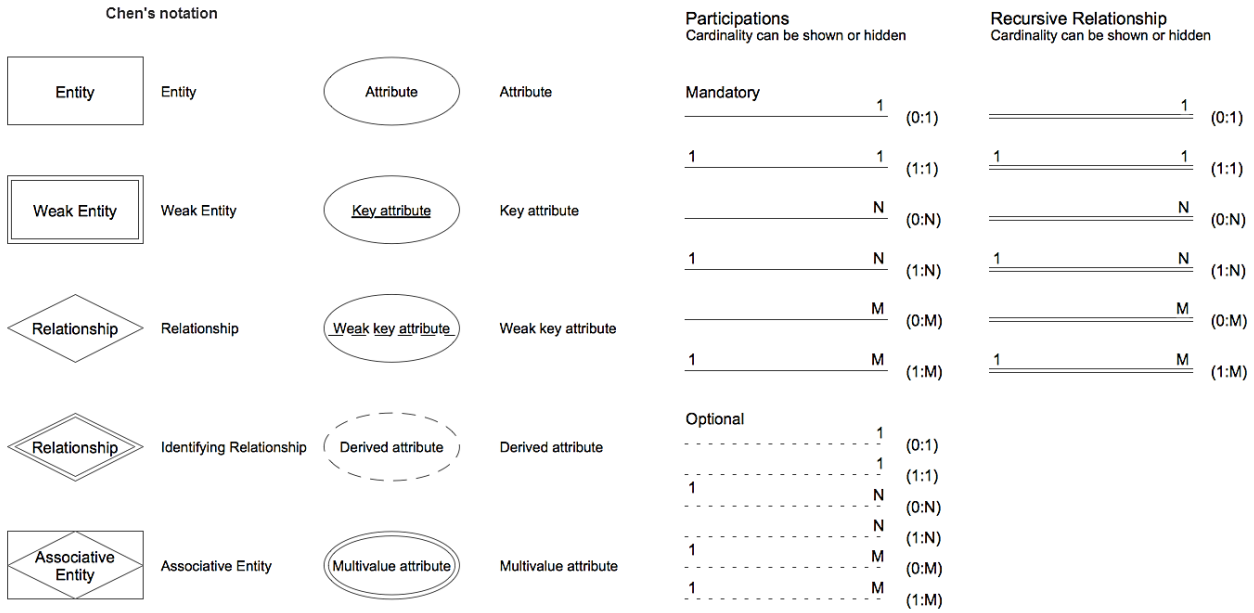
- *The Complete Entity Type Student with its Attributes can be represented as:*

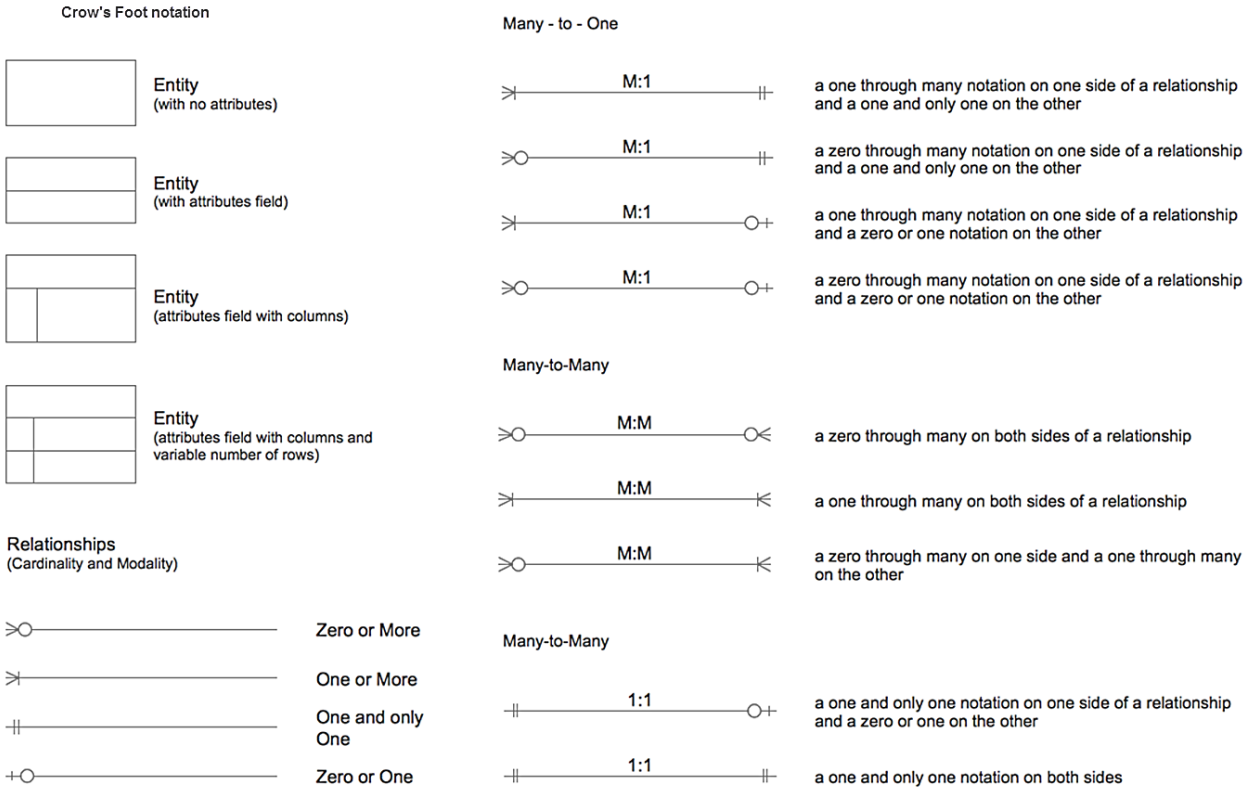


- **Key Elements in the Diagram (One-Liners)**

- **Roll_No:** Key attribute (Primary Key) to uniquely identify a student.
- **Name, DOB:** Simple, single-valued attributes of the student.
- **Phone_No:** Multi-valued attribute (double ellipse).
- **Age:** Derived attribute (dashed ellipse) from *DOB*.
- **Address:** Composite attribute split into *Country, State, City, Street*.

➢ Symbols For ER Diagram





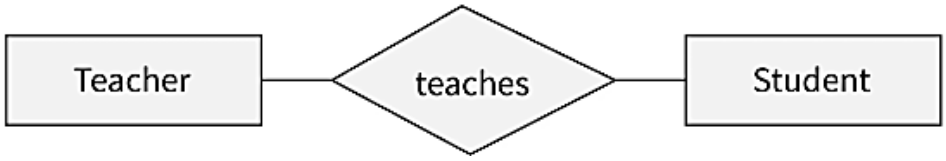
2.4.3 Relationships

A **relationship in DBMS represents the logical association between two or more entity sets** to capture meaningful connections in real-world scenarios. It helps in organizing data to avoid redundancy and ensures that the data is interconnected in a structured manner.

In **Entity-Relationship (ER) Diagrams**, relationships are denoted using a **diamond (rhombus) symbol** and labelled clearly to indicate the nature of the association between entities.

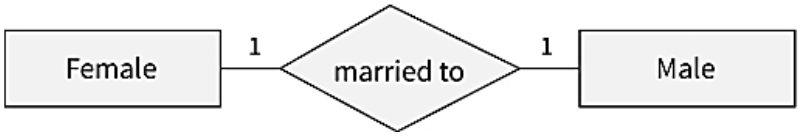
- ✓ Relationships help in **enforcing data integrity and simplifying complex queries** by defining how data in one entity relates to data in another.
- ✓ They also **enable consistent data retrieval** by maintaining clear connections across different tables in a relational database.

Example: In a college database, the *TEACHER* entity is related to the *STUDENT* entity through the *TEACHES* relationship.



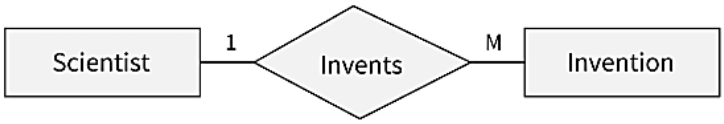
- ➢ **Types of Relationships in DBMS:** There are **4 primary types of relationships** in database design:

1. **One-to-One (1:1) Relationship**
   - **Definition:** A single instance of one entity is associated with a single instance of another entity.
   - **Purpose:** Used when one entity is uniquely associated with another.
   - **Example:** A *HUSBAND* is married to one *WIFE*, and a *WIFE* is married to one *HUSBAND*.
   - **Practical Use:** Storing additional details in a separate table for security or clarity.



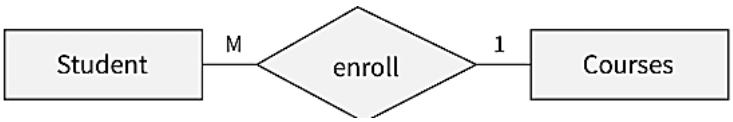
2. One-to-Many (1:M) Relationship

- **Definition:** A single instance of an entity is associated with multiple instances of another entity.
- **Purpose:** The most common relationship type used to represent hierarchy and dependencies.
- **Example:** A *SCIENTIST* can invent multiple *INVENTIONS*, but each *INVENTION* is invented by one *SCIENTIST*.
- **Practical Use:** Managing lists, orders, or hierarchical data in applications.



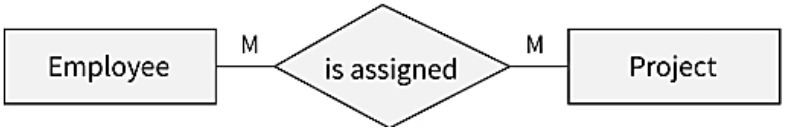
3. Many-to-One (M:1) Relationship

- **Definition:** Multiple instances of an entity are associated with a single instance of another entity.
- **Purpose:** Inverse of One-to-Many, showing multiple entities depending on a single entity.
- **Example:** Many *STUDENTS* are enrolled in one *COURSE*, but each *COURSE* can have many *STUDENTS*.
- **Practical Use:** Managing class enrollments or department assignments.



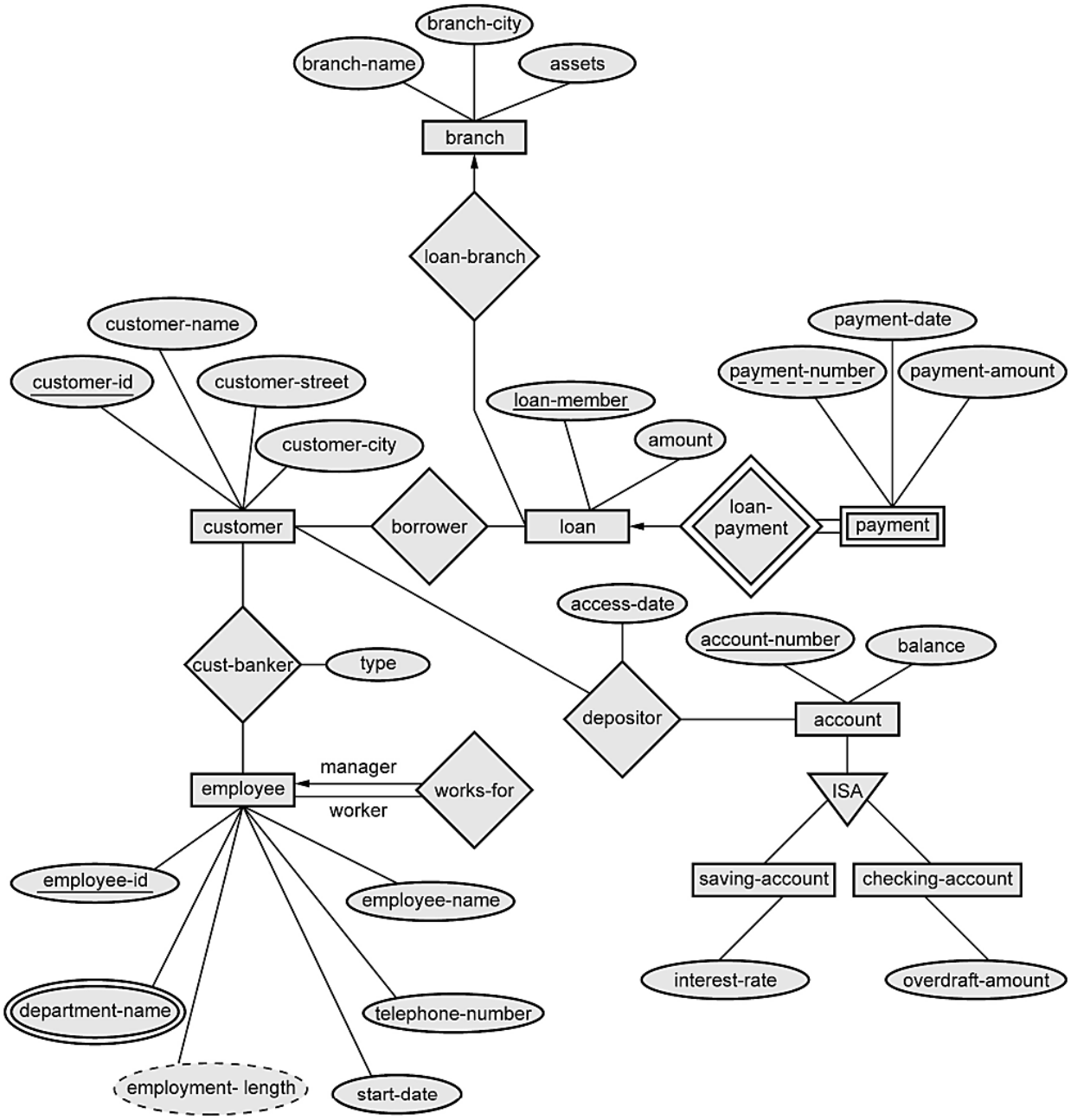
4. Many-to-Many (M:N) Relationship

- **Definition:** Multiple instances of one entity are associated with multiple instances of another entity.
- **Purpose:** Represents complex associations where both entities participate in multiple relationships.
- **Example:** *EMPLOYEES* can work on multiple *PROJECTS*, and each *PROJECT* can have multiple *EMPLOYEES*.
- **Practical Use:** Requires an associative entity (junction table) when implemented in a relational database to break into two One-to-Many relationships.



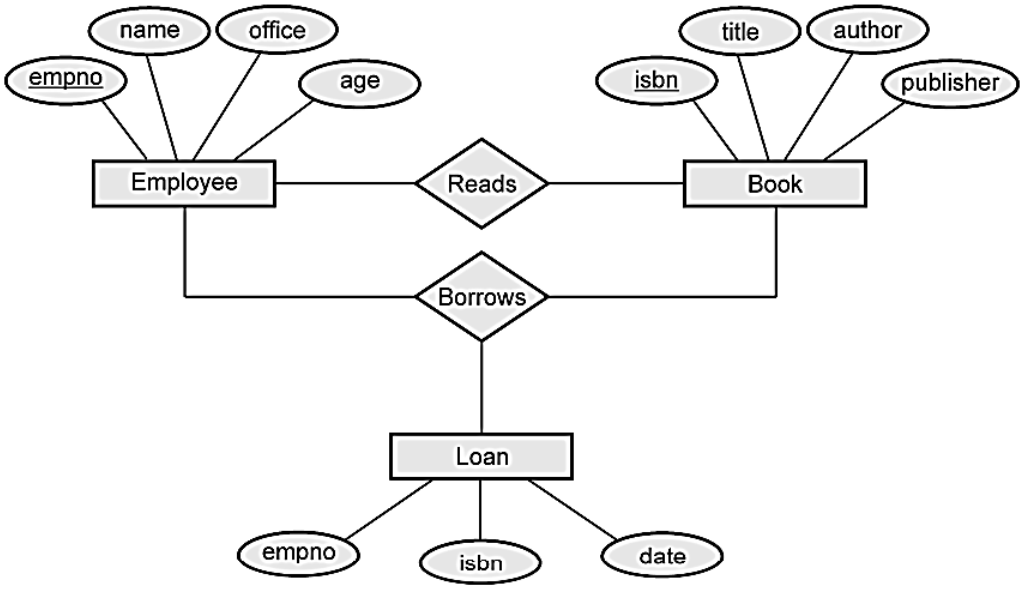
- **Drawing ER Diagrams:**

1. Draw an ER diagram for banking system (Home-Loan Application)

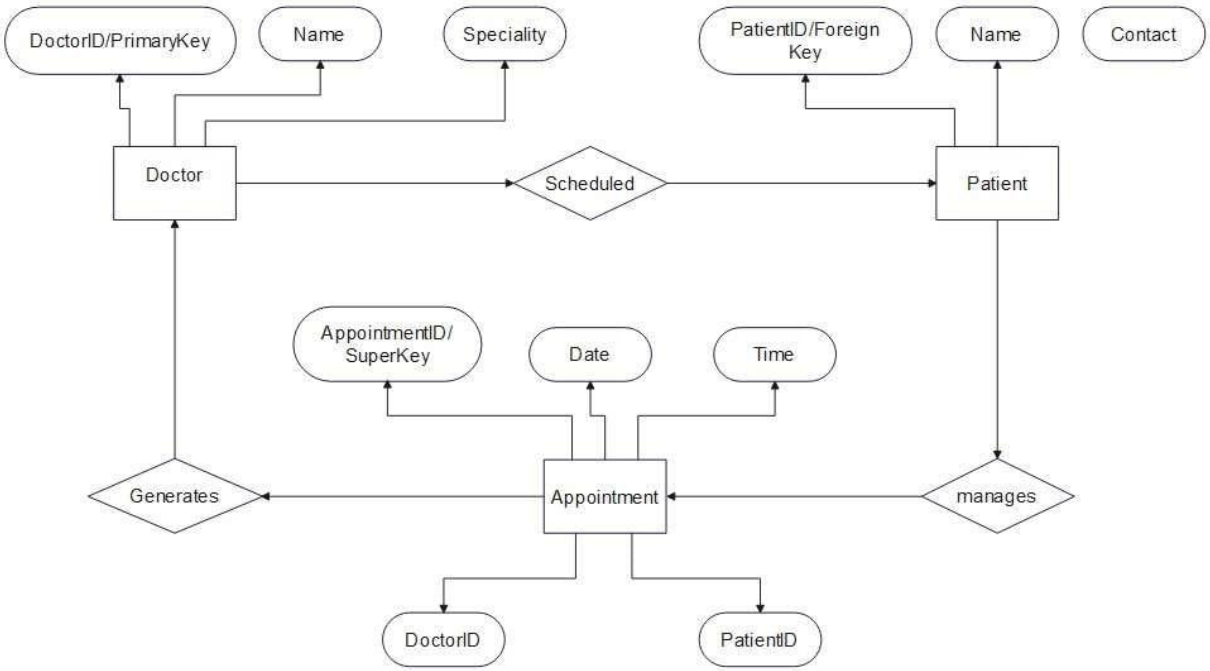


2. Consider the relation schema given in following Figure. Design and draw an ER diagram that capture the information of this schema.

   Employee(empno,name,office,age)
   Books(isbn,title,authors,publisher)
   Loan(empno,isbn,date)



3. Draw ER Diagram for Hospital Management System (PYQ)



2.5 Normalization

Normalization is a **systematic process of organizing data in a database to reduce redundancy and improve data integrity**. It involves dividing large tables into smaller, well-structured tables and defining clear relationships among them.

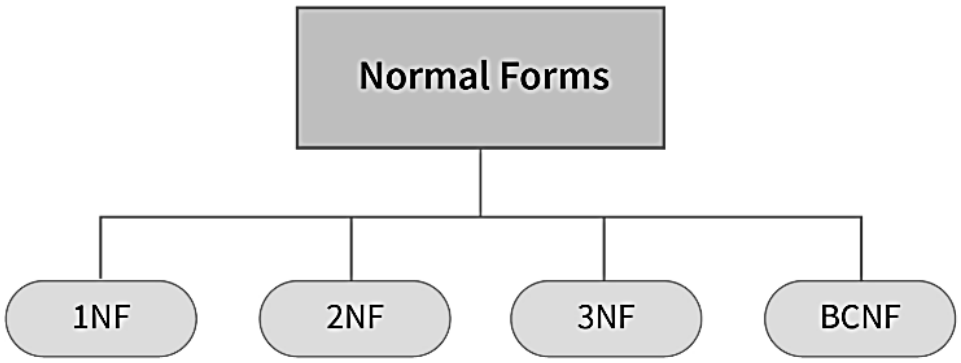
The **main objectives of normalization** are:

- To eliminate data redundancy (repeated data).
- To ensure data consistency and integrity.
- To organize data logically for efficient data retrieval and updates.
- To simplify database maintenance.

Normalization was first proposed by **E.F. Codd** in 1970 and is achieved through a series of stages called **normal forms**.

- **Need for Normalization**
  1. **Avoid Data Anomalies:** Prevents insertion, deletion, and update anomalies in the database.
  2. **Minimize Redundancy:** Reduces duplicate data to save storage and avoid inconsistency.
  3. **Maintain Data Integrity:** Ensures that data dependencies are logical and accurate.
  4. **Easy Maintenance:** Simplifies database design, making it easier to modify and manage.

Normal Forms: Normalization progresses through **normal forms (NF),** where each form addresses specific types of redundancy and dependency issues:



2.5.1 Functional Dependency

- **Functional Dependence (FD)** is a **core concept in relational database design** that describes the relationship between attributes in a relation.
- It states that **the value of one attribute (or a set of attributes) uniquely determines the value of another attribute (or a set of attributes)** in the same relation.
- Functional dependence helps in **understanding data redundancy and plays a critical role in normalization, ensuring data integrity, consistency, and efficient design of relational databases**.

➢ **Definition:** In a relation **R**, if attribute **Y is functionally dependent on attribute X**, it is represented as:

$$\mathbf{X \rightarrow Y}$$

This means:

- For each unique value of **X**, there is **exactly one corresponding value of Y**.
- *X is called the Determinant*, and *Y is the Dependent Attribute*.

➢ **Real-Time Example**

Consider a Student Database with the following table:

Roll_No	Name	Department
101	Ravi	CS
102	Meena	IT
103	Suraj	CS

Here:

- *Roll_No → Name* (Each roll number has exactly one associated name.)
- *Roll_No → Department* (Each roll number has exactly one department.)

In practice:

- Knowing a student's Roll_No allows us to determine their Name and Department accurately, ensuring consistent data retrieval.

- **Types of Functional Dependencies**

1. Trivial Functional Dependency

- A **functional dependency (FD) X → Y is called trivial** if **Y is a subset of X**.
- This means that the dependent attribute(s) already exist on the left-hand side, so this dependency **does not provide new information** and **always holds true** in any relation.

- **Example:**

  If **X = {Roll_No, Name}**, then:

  - {Roll_No, Name} → Roll_No (Trivial FD, as Roll_No is part of X.)
  - {Roll_No, Name} → {Roll_No, Name} (Trivial FD, as Y = X.)

- **Key Points:**
  - Trivial dependencies do not help in identifying redundancy.
  - They are useful while testing for **normal forms**.

---

2. Non-Trivial Functional Dependency

- A **functional dependency X → Y is non-trivial if Y is not a subset of X**.
- Non-trivial FDs **help in identifying redundancy and dependency issues** within a relation, making them important for normalization.

- **Example:** Given a **STUDENT** relation:

Roll_No	Name	Department
101	Ravi	CS

  - Roll_No → Name (Non-trivial FD, as Name is not part of Roll_No.)
  - Roll_No → Department (Non-trivial FD, as Department is not part of Roll_No.)

- **Key Points:**
  - Non-trivial FDs are useful in identifying which attributes are determined by others.
  - They are essential for normalization steps to reduce redundancy.

3. Multivalued Functional Dependency (MVD)

- A **multivalued dependency (MVD) X ↠ Y** exists in a relation **R** if, for a single value of **X**, there exist **multiple independent values of Y**, irrespective of the other attributes.
- MVD indicates that **Y values are independent of other attributes but are associated with X**.

➢ **Example:** Consider a **STUDENT** relation:

Roll_No	Course	Hobby
101	DBMS	Painting
101	DBMS	Singing
101	CN	Painting
101	CN	Singing

Here:

- *Roll_No ↠ Course* (Multiple courses for one student.)
- *Roll_No ↠ Hobby* (Multiple hobbies for one student.)

Key Points:

- MVDs lead to data redundancy.
- They are handled in **Fourth Normal Form (4NF)** during advanced normalization.
- MVD is represented using a **double-headed arrow (↠)**.

---

4. Transitive Functional Dependency

- A **transitive dependency occurs when X → Z through an intermediate attribute Y, i.e., X → Y and Y → Z, then X → Z**.
- Transitive dependency often leads to redundancy and anomalies in a relation and is removed while achieving **Third Normal Form (3NF)**.

➢ **Example:** Given a relation:

Roll_No	Dept_ID	Dept_Name
101	D1	CS
102	D2	IT

- Roll_No → Dept_ID
- Dept_ID → Dept_Name
- Therefore, **Roll_No → Dept_Name** (Transitive Dependency).

➢ **Real-World Example:**

If:

- Employee_ID → Dept_ID
- Dept_ID → Dept_Location

Then:

- Employee_ID → Dept_Location (Transitive dependency through Dept_ID.)

Key Points:

- Transitive dependencies can cause **update, insertion, and deletion anomalies**.
- They are removed during **3NF** to improve database design.

➢ **Summary:**

Type	Definition	Purpose in Design	Example
Trivial FD	Dependent attribute is part of determinant.	Always true; no effect on redundancy.	{Roll_No, Name} → Roll_No
Non-Trivial FD	Dependent is not part of determinant.	Shows actual dependency used in normalization.	Roll_No → Name
Multivalued FD	One attribute determines multiple independent multivalued facts.	Identified for 4NF to avoid repetition.	Roll_No →→ Phone_No
Transitive FD	Non-key depends on another non-key attribute.	Removed in 3NF to avoid indirect dependency.	Emp_ID → Dept_ID → Dept_Name

2.5.2 First Normal Form (1NF):

First Normal Form (1NF) is the **first step in the normalization process** in relational database design to ensure that the data in a table is **well-structured and easily manageable**.

A relation is in **1NF if it satisfies the following conditions**:

- Each column in the table contains **only atomic (indivisible) values**.
- All entries in a column are of the **same data type**.
- Each record in the table is **unique**, ensuring that there are **no duplicate rows**.
- **There are no repeating groups or arrays** within a column.

The main purpose of 1NF is to **eliminate multi-valued and composite attributes** to maintain **data consistency, simplicity, and clarity** in the database.

➢ **Example:** Consider a table storing student information and courses:

Before Applying 1NF:

Roll_No	Name	Courses
101	Ravi	DBMS, CN
102	Meena	DBMS
103	Suraj	OS, DBMS, CN

In the above table, the **Courses** column has **multiple values in a single cell** (multi-valued attribute), which **violates the rule of atomicity**, and hence, the table is **not in 1NF**.

After Applying 1NF:

To convert the table into 1NF, we split the multi-valued entries into **separate rows** so that each cell has only **one value**.

Roll_No	Name	Course
101	Ravi	DBMS
101	Ravi	CN
102	Meena	DBMS
103	Suraj	OS
103	Suraj	DBMS
103	Suraj	CN

Explanation:

- Now, each column contains **atomic values** with **only one value per cell**.
- The **multi-valued Courses attribute** has been eliminated.
- The table is now in **First Normal Form (1NF)**, making it:
  - Easier to query.
  - Easier to maintain.
  - Ready for further normalization to reduce redundancy and anomalies.

2.5.3 Second Normal Form (2NF)

Second Normal Form (2NF) is the **second step in the normalization process** in relational database design.

A relation is in **2NF if:**

- It is already in **First Normal Form (1NF)**.
- **All non-prime (non-key) attributes are fully functionally dependent on the *entire* primary key**, not on just a part of it.

The main goal of 2NF is to **remove partial dependencies**, which occur when a non-key attribute depends only on a part of a **composite primary key**.

➢ **Why 2NF is Needed?**

- To **eliminate redundancy** caused by partial dependencies.
- To avoid **update, insertion, and deletion anomalies**.
- To ensure **data integrity** and efficient storage.

➢ **Example:** Consider the following relation storing course enrollment data:

Table Before Applying 2NF:

Student_ID	Course_ID	Student_Name	Course_Name
101	DBMS	Ravi	Database
101	CN	Ravi	Networking
102	DBMS	Meena	Database

Key Points:

- Composite primary key: **{Student_ID, Course_ID}**.
- **Student_Name depends only on Student_ID**.
- **Course_Name depends only on Course_ID**.

This means **partial dependencies exist**, so the table is **not in 2NF**.

➢ **Conversion to 2NF:**

To convert the table into 2NF:

- Remove partial dependencies by creating separate tables.
- Ensure that **non-key attributes depend on the entire key**.

After Applying 2NF:

- **Table 1: STUDENT**

Student_ID	Student_Name
101	Ravi
102	Meena

- **Table 2: COURSE**

Course_ID	Course_Name
DBMS	Database
CN	Networking

- **Table 3: ENROLLMENT**

Student_ID	Course_ID
101	DBMS
101	CN
102	DBMS

Explanation:

- **Partial dependencies are removed** by splitting the table.
- **Student_Name depends only on Student_ID** (handled in the STUDENT table).
- **Course_Name depends only on Course_ID** (handled in the COURSE table).
- The **ENROLLMENT table now only stores keys, ensuring relationships.**

2.5.4 Third Normal Form (3NF)

Third Normal Form (3NF) is the **third level of normalization** in relational database design.

A relation is in **3NF** if:

- It is already in **Second Normal Form (2NF)**.
- It **does not contain any transitive dependencies** — meaning **non-key attributes do not depend on other non-key attributes**.

The goal of 3NF is to **remove transitive dependencies** so that **each non-key attribute depends only on the primary key**.

➢ **Why 3NF is Needed?**

- To eliminate **hidden redundancy** due to indirect (transitive) dependencies.
- To avoid **update, insertion, and deletion anomalies**.
- To ensure **each non-key attribute describes only the key**.

➢ **Example-** Consider the following table storing employee data:

- **Table Before Applying 3NF**

Emp_ID	Emp_Name	Dept_ID	Dept_Name
101	Ravi	D1	IT
102	Meena	D2	HR
103	Suraj	D1	IT

➢ **Key Points:**

- Primary Key: **Emp_ID**
- Emp_ID → Dept_ID
- Dept_ID → Dept_Name
- So, **Emp_ID → Dept_Name** (transitive dependency).

This indirect dependency **violates 3NF**.

➢ **Conversion to 3NF**

To convert the relation to 3NF:

- **Remove transitive dependency** by splitting the table into two relations.

➢ **After Applying 3NF**

- **Table 1: EMPLOYEE**

Emp_ID	Emp_Name	Dept_ID
101	Ravi	D1
102	Meena	D2
103	Suraj	D1

- **Table 2: DEPARTMENT**

Dept_ID	Dept_Name
D1	IT
D2	HR

➢ **Explanation**

- Now, **Dept_Name depends only on Dept_ID**, not indirectly on Emp_ID.
- Each non-key attribute depends **only on the primary key** of its table.
- **Transitive dependency has been removed**, ensuring the design is clearer and free of hidden redundancy.

DBMS Unit 2 – Practice Questions

1. Define First Normal Form (1NF). Explain with a suitable example how to convert an unnormalized table to 1NF.
2. Differentiate between Super Key, Candidate Key, Primary Key, and Foreign Key with examples.
3. What is Functional Dependency? Explain its types with suitable examples.
4. What are the different types of attributes in an ER diagram? Explain with examples.
5. Explain the process of converting a relation from 2NF to 3NF with an example.
6. List and explain the different types of relationships in an ER model with real-time examples.

Author : Vishal Jadhav Sir | Contact : +91-9730087674

Edit By : Ghansham Irashetti