Program : B. Tech (4th Sem)

Database Management System

Relational Database Design and E-R Model , E-R Model , Normalization - UNIT- 2

Topics :

- ➔ Structure of Relational databases
- ➔ Domains
- ➔ Relations
- ➔ Relational algebra – fundamental operators and syntax
- ➔ Relational algebra Queries
- ➔ Tuple relational calculus Basic Concepts
- ➔ Design Process
- ➔ Constraints
- ➔ Keys
- ➔ Design issues
- ➔ E-R Diagrams
- ➔ Weak entity sets
- ➔ Extended E-R features : Generalization
- ➔ Specialization
- ➔ Aggregation
- ➔ Reduction to E-R database schema
- ➔ Data redundancy
- ➔ Normal forms :1NF, 2NF, 3NF, BCNF and 4NF

Relation:

A relation is usually represented as a table, organized into rows and columns. A relationship consists of multiple records. **For example:** student relation which contains tuples and attributes.

Tuple:

The rows of a relation that contain the values corresponding to the attributes are called tuples. **For example:** In the Student relation there are 5 tuples.
The value of tuples contains (10112, Rama, 9874567891,islam ganj, F) etc.

Data Item:

The smallest unit of data in the relation is the individual data item. It is stored at the intersection of rows and columns are also known as cells. **For Example:** 10112, "Rama" etc are data items in Student relation.

Domain:

A **domain** refers to the set of all possible values that an attribute (or column) in a database table can have. It essentially defines the permissible values for a particular attribute to maintain data integrity and consistency in the database. **For Example:** Age may have a domain of integers between 0 and 120.

Attribute:

The smallest unit of data in relational model is an attribute. It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai). **For example:** Stu_No, S_Name, PHONE_NO, ADDRESS, Gender are the attributes of a student relation. In relational databases a column entry in any row is a single value that contains exactly one item only.

Cardinality:

The total number of rows at a time in a relation is called the cardinality of that relation. **For example:** In a student relation, the total number of tuples in this relation is 3 so the cardinality of a relation is 3. The cardinality of a relation changes with time as more and more tuples get added or deleted.

Degree:

The degree of association is called the total number of attributes in a relationship. The relation with one attribute is called unary relation, with two attributes is known a binary relation and with three attributes is known as ternary relation. **For example:** in the Student relation, the total number of attributes is 5, so the degree of the relations is 5. The degree of a relation does not change with time as tuples get added or deleted.

Relational instance:

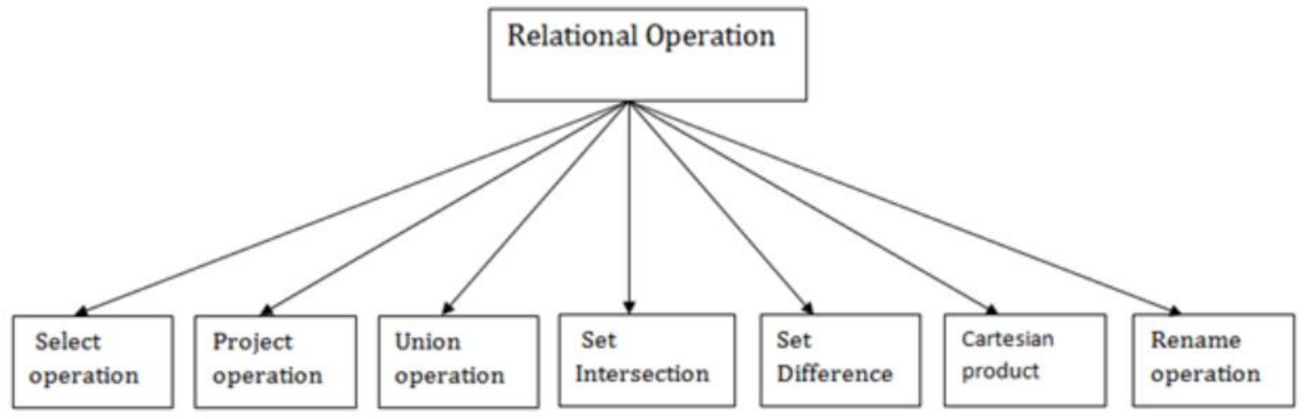
In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

Relational schema:
A relational schema contains the name of the relation and name of all columns or attributes.

Relational key:
In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

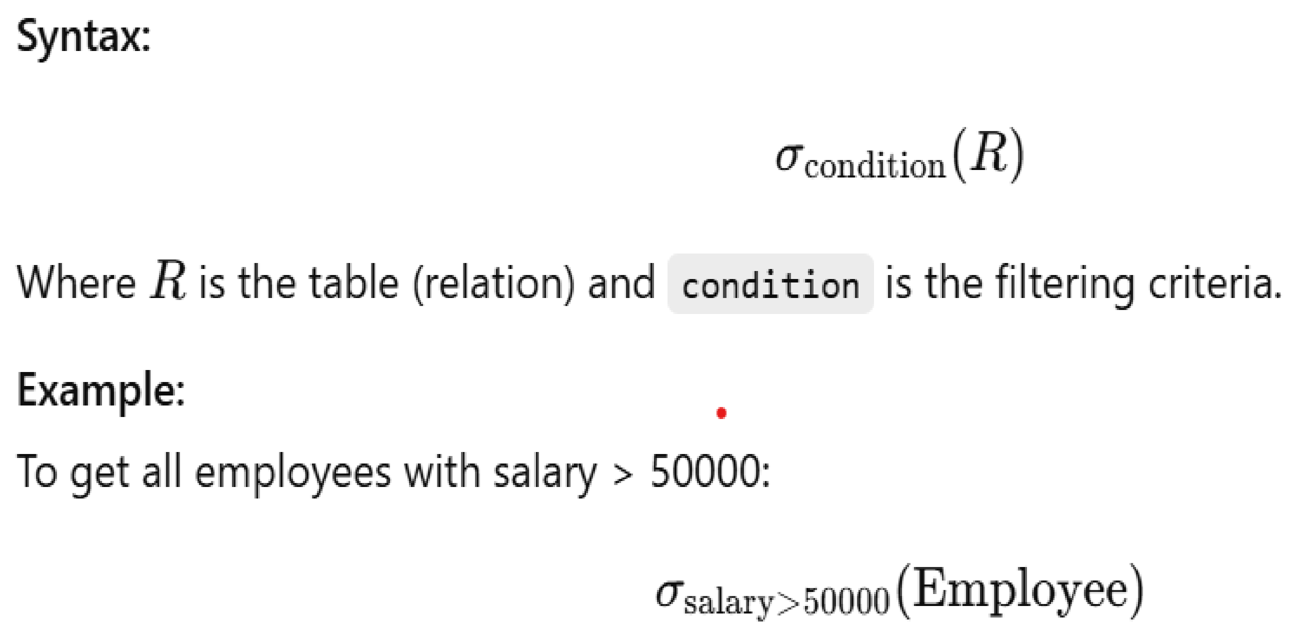
Relational Algebra - Fundamental Operators and Syntax:
Relational algebra is the mathematical foundation of relational databases. It provides a set of operations to manipulate and query data stored in relational database tables. These operations are performed on relations (tables) to produce new relations as results.

Types of Relational operation



Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.
Notation − $\sigma_p(r)$
Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.



Syntax:

$$\sigma_{\text{condition}}(R)$$

Where R is the table (relation) and `condition` is the filtering criteria.

Example:

To get all employees with salary > 50000:

$$\sigma_{\text{salary}>50000}(\text{Employee})$$

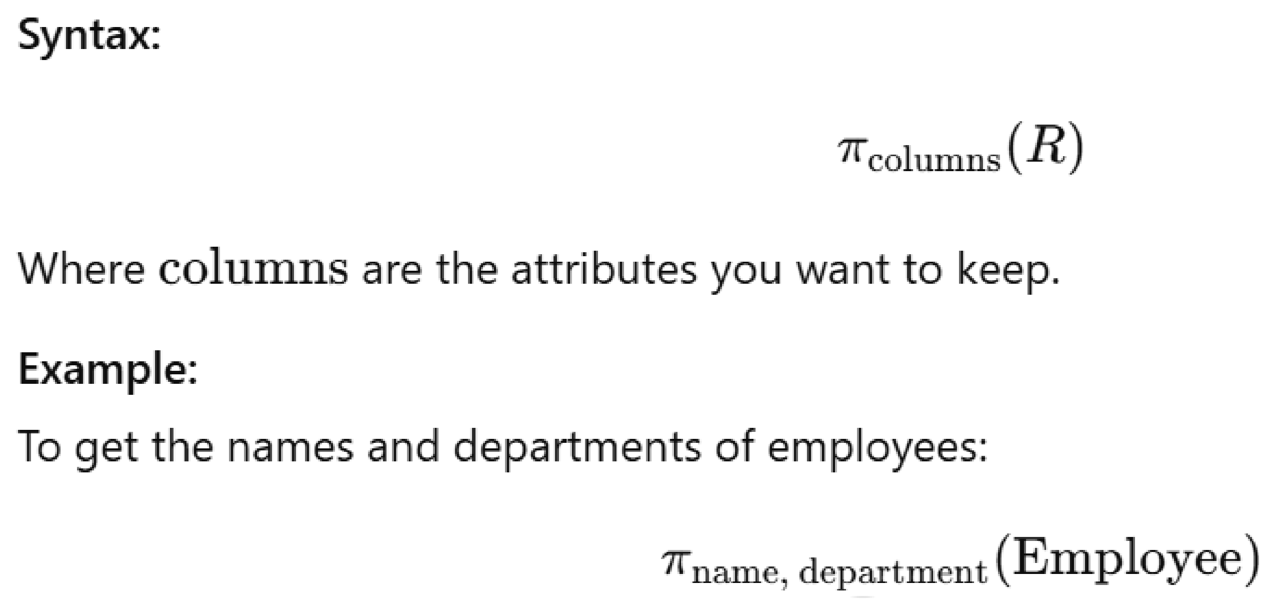
2. Projection (π)

Selects specific columns (attributes) from a table.

Notation − ∏A1, A2, An (r)

Where A1, A2 , An are attribute names of relation r.

Duplicate rows are automatically eliminated, as relation is a set.



Syntax:

$$\pi_{\text{columns}}(R)$$

Where columns are the attributes you want to keep.

Example:

To get the names and departments of employees:

$$\pi_{\text{name, department}}(\text{Employee})$$

3. Union (∪)

Combines rows from two tables with the same structure, removing duplicates.

r ∪ s = { t | t ∈ r or t ∈ s}

Notation − r U s

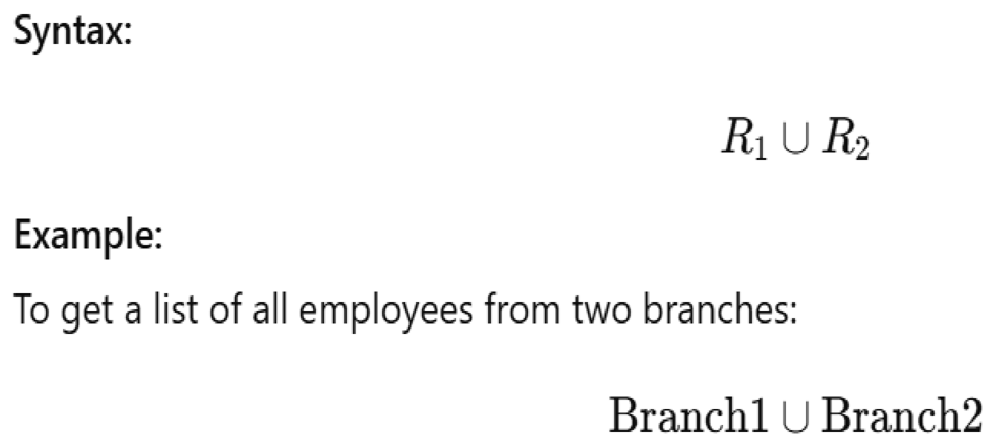
Where r and s are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

r, and s must have the same number of attributes.

Attribute domains must be compatible.

Duplicate tuples are automatically eliminated.



Syntax:

$$R_1 \cup R_2$$

Example:

To get a list of all employees from two branches:

$$\text{Branch1} \cup \text{Branch2}$$

4. Set Difference (−)

Returns rows from the first table that are not in the second table.

Notation − r − s

Finds all the tuples that are present in **r** but not in **s.**

Syntax:

$$R_1 - R_2$$

Example:

To find employees who are in Branch1 but not in Branch2:

$$\text{Branch1} - \text{Branch2}$$

5. Cartesian Product (×)

Combines all rows from two tables.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

Syntax:

$$R_1 \times R_2$$

Example:

To pair each employee with each department:

$$\text{Employee} \times \text{Department}$$

6. Rename (ρ)

Renames a table or its attributes.

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

Notation − $\rho_x(E)$

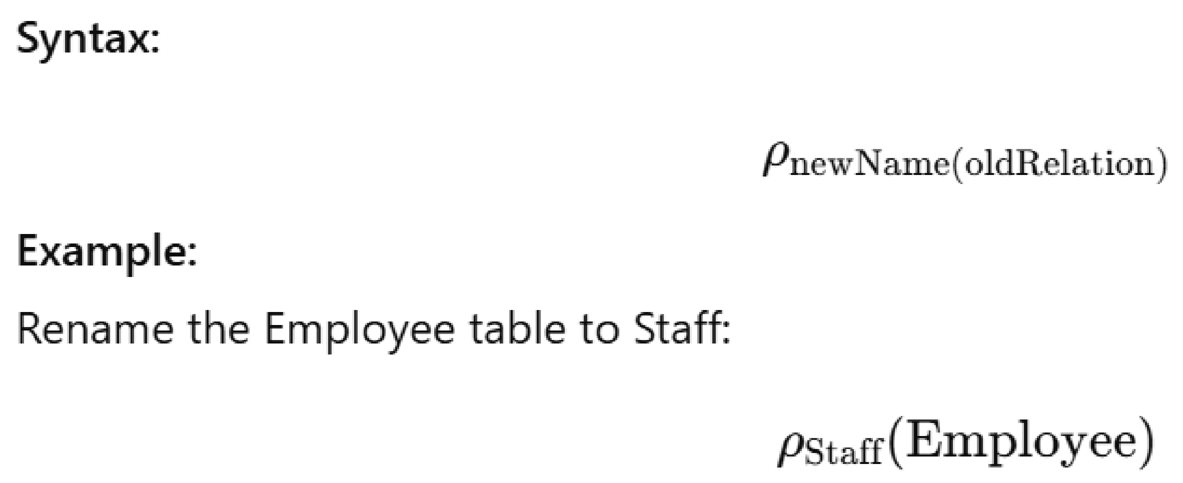
Where the result of expression **E** is saved with name of **x**.

Additional operations are −

Set intersection

Assignment

Natural join



Syntax:

$$\rho_{\text{newName(oldRelation)}}$$

Example:

Rename the Employee table to Staff:

$$\rho_{\text{Staff}}(\text{Employee})$$

7. Intersection (∩)

Returns rows common to two tables.

Syntax:

$$R_1 \cap R_2$$

Example:

To find employees working in both Branch1 and Branch2:

$$\text{Branch1} \cap \text{Branch2}$$

8. Natural Join (⋈)

Combines rows from two tables based on common attributes.

Syntax:

$$R_1 \bowtie R_2$$

Example:

To join the Employee and Department tables based on `department_id`:

$$\text{Employee} \bowtie \text{Department}$$

Consider the following tables:
Employee Table

Emp ID	Name	DeptID	Salary
1	Alice	D01	50000
2	Bob	D02	60000
3	Charlie	D01	55000
4	David	D03	45000

Department Table

DeptID	DeptName	Location
D01	HR	Building A
D02	IT	Building B
D03	Finance	Building A

Questions-

1. Select employees with a salary greater than 50,000.

2. Project only the names of employees and their salaries.

3. Perform a Union of employees from "HR" and "IT" departments.

4. Use Set Difference to find employees who are not in "Building A".

5. Perform a Cartesian Product of the Employee and Department tables.

6. Rename the DeptName column in the Department table to Department.

1. $\pi_{\text{EmpID, Name, DeptID, Salary}}\left(\sigma_{\text{Salary}>50000}(\text{Employee})\right)$

2. $\pi_{\text{Name, Salary}}(\text{Employee})$

3. HR_Employees∪IT_Employees

$\pi_{\text{EmpID, Name, DeptID, Salary}}\left(\sigma_{\text{DeptName}='HR'}(\text{Employee} \bowtie \text{Department})\right) \cup \pi_{\text{EmpID, Name, DeptID, Salary}}\left(\sigma_{\text{DeptName}='IT'}(\text{Employee} \bowtie \text{Department})\right)$

4. $\pi_{\text{EmpID, Name, DeptID, Salary}}(\text{Employee}) - \pi_{\text{EmpID, Name, DeptID, Salary}}\left(\sigma_{\text{Location}='BuildingA'}(\text{Employee} \bowtie \text{Department})\right)$

5. Employee×Department

6 $\rho_{(\text{DeptID, Department, Location})}(\text{Department})$

Tuple Relational Calculus -

Tuple Relational Calculus (TRC) is a non-procedural query language that specifies what to retrieve from a database without describing how to retrieve it. Unlike Relational Algebra (which gives step-by-step operations), TRC focuses on conditions that tuples (rows of data) must satisfy to appear in the output.

It defines queries in terms of properties or conditions.
You describe the result you want, not the process to get it.

Key Terms:

1. **Tuple**: A row of a table.

2. **Variable**: Represents a tuple.

3. **Condition**: A logical statement that filters tuples.

4. **Result**: A set of tuples that satisfy the condition.

Syntax:
In TRC, a query looks like: **{T | P(T)}**
T: A tuple variable.
P(T): A condition (predicate) that describes what the tuple must satisfy.
The query means: "Find all tuples **T** such that **P(T)** is true.“

Consider a table **Student**:

RollNo	Name	Age	City
1	Alice	20	New York
2	Bob	22	Los Angeles
3	Charlie	19	New York

1. Find all students who live in New York.

{T | T ∈ Student AND T.City = "New York"}

Explanation:

T: A tuple from the Student table.
T.City = "New York": The condition that the city of the student must be "New York".

2. Find students who are **older than 20** and live in **Los Angeles**.

{T | T ∊ Student AND T.Age > 20 AND T.City = "Los Angeles"}

Explanation:

T.Age > 20: The student must be older than 20.
T.City = "Los Angeles": The city must be "Los Angeles".

Employee

ID	Name	Dept_Name	Salary
1	Alisha	IT	60000
2	Barbie	HR	45000
3	Chandni	IT	75000
4	David	Sales	50000
5	Enigma	IT	55000
6	Farah	HR	40000
7	Gracia	Sales	70000

Department

Dept_Name	Manager
IT	John
HR	Emma
Sales	Sarah

Questions-

1. Write a TRC query to find the names of employees who earn more than 50,000.

2. Write a TRC query to retrieve all employees who work in the "IT" department.

3. Write a TRC query to find the names of employees whose department manager is "John".

4. Write a TRC query to find the employees who do not work in the "HR" department.

5. Write a TRC query to find employees who earn the highest salary in their department.

Answers –

1. {t[Name] | t∈Employee ∧ t[Salary]>50000}

2. {t | t∈Employee ∧ t[Dept_Name]= ′IT ′}

3. {t[Name] | ∃d (t∈Employee ∧ d∈Department ∧ t[Dept_Name]=d[Dept_Name] ∧ d[Manager]= ′John ′)}

4. $\{t \mid t \in Employee \ \wedge \ t[Dept\_Name] \neq' HR'\}$

5. {t | t∈Employee ∧¬(∃e (e∈Employee ∧ e[Dept_Name]=t[Dept_Name] ∧ e[Salary]>t[Salary]))}

Design Process -

The design process in Database Management System (DBMS) involves systematically creating a database structure that supports the needs of an organization or application while ensuring efficiency, accuracy, and scalability.

1. Requirement Analysis

Objective: Understand the needs of the users and the application.

Tasks:

Identify the purpose of the database.

Gather requirements through discussions with stakeholders.

Define what data needs to be stored and the operations that will be performed on it.

2. Conceptual Design

Objective: Create a high-level representation of the data.

Output: Entity-Relationship Diagram (ERD).

Tasks:

Identify entities (objects or things).

Define attributes for each entity.

Determine relationships between entities (one-to-one, one-to-many, many-to-many).

Define constraints like primary keys, unique attributes, etc.

3. Logical Design
Objective: Transform the conceptual design into a logical model.
Output: Schema in terms of tables, attributes, and keys.
Tasks:
Map entities and relationships into tables.
Normalize the data to eliminate redundancy and maintain integrity (1NF, 2NF, 3NF, etc.).
Define primary keys, foreign keys, and other constraints.

4. Physical Design
Objective: Optimize the logical design for the physical storage system.
Output: Physical storage plan.
Tasks:
Decide on indexes for faster data retrieval.
Partition tables if needed for performance.
Define storage parameters like file organization (e.g., heap, B+ tree).
Choose appropriate database management systems (MySQL, PostgreSQL, etc.).

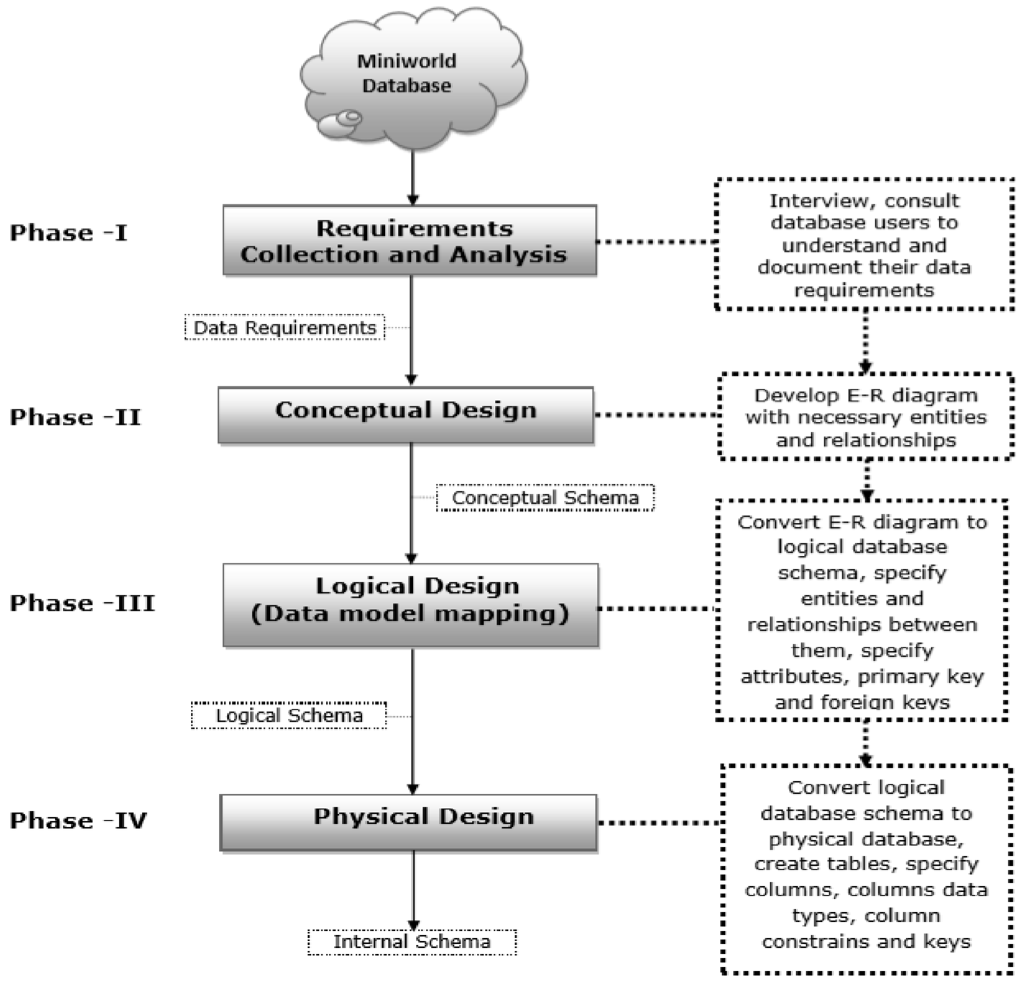
5. Database Implementation
Objective: Implement the physical design in the chosen DBMS.
Tasks:
Create tables, views, indexes, and other database objects.

Populate the database with initial data if necessary.
Implement security measures like user roles and permissions.

6. Testing and Validation
Objective: Ensure the database meets all requirements.
Tasks:
Perform functional testing to validate queries and operations.
Test data integrity, relationships, and constraints.
Check performance under load.

7. Database Maintenance
Objective: Maintain the database for optimal performance and reliability.
Tasks:
Regularly monitor and optimize performance.
Backup and restore as needed.
Apply updates to schema or data as requirements evolve.
Handle issues like deadlocks, crashes, or security breaches.

1. DATABASE DESIGN PROCESS FLOWCHART


Miniworld Database
Phase -I
Requirements Collection and Analysis
Interview, consult database users to understand and document their data requirements
Data Requirements
Phase -II
Conceptual Design
Develop E-R diagram with necessary entities and relationships
Conceptual Schema
Phase -III
Logical Design (Data model mapping)
Convert E-R diagram to logical database schema, specify entities and relationships between them, specify attributes, primary key and foreign keys
Logical Schema
Phase -IV
Physical Design
Convert logical database schema to physical database, create tables, specify columns, columns data types, column constrains and keys
Internal Schema

Keys in DBMS

1. Primary Key
Definition: A column (or set of columns) that uniquely identifies each row in a table.
Key Rules:
Values must be unique.
Cannot have NULL values.

Here, **EmpID** is the Primary Key because:
Every value in this column is unique.
t identifies each employee in the table.

EmpID	Name	DeptID	Salary
1	Alice	D01	50000
2	Bob	D02	60000
3	Charlie	D01	55000

Foreign Key

A column in one table that links to the Primary Key in another table, establishing a **relationship**.

OR

A Foreign Key is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table. The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Department

DeptID	DeptName
D01	HR
D02	IT

Employee

EmpID	Name	DeptID	Salary
1	Alice	D01	50000
2	Bob	D02	60000

In the Employee table, DeptID is a Foreign Key that references the Primary Key DeptID in the Department table.

This ensures that each employee belongs to a valid department.

Candidate Key

A column (or set of columns) that could be a Primary Key because it uniquely identifies rows in the table.
Difference from Primary Key: While there can be multiple Candidate Keys, only one is chosen as the Primary Key.

StudentID	Email	Phone
101	alice@example.com	1234567890
102	bob@example.com	9876543210

Here, both StudentID and Email are Candidate Keys because they can uniquely identify rows. You would choose one (e.g., StudentID) as the Primary Key.

Composite Key

A combination of two or more columns that together uniquely identify a row.

When to Use: When no single column can uniquely identify a row.

CourseEnrollment Table

StudentID	CourseID	EnrollmentDate
101	C01	2023-01-10
101	C02	2023-02-15
102	C01	2023-01-12

Here, neither StudentID nor CourseID alone is unique.
The combination of StudentID and CourseID forms a Composite Key that uniquely identifies each row.

Alternate Key

A Candidate Key that is not chosen as the Primary Key.
Acts as a backup unique identifier.

Example:

Using the Student Table example above:
StudentID is the Primary Key.
Email is an Alternate Key.

Super Key

Definition: A set of one or more attributes that can uniquely identify a row in a table.
Relation to Candidate Key: A Candidate Key is the minimal Super Key.
Example:

Employee Table

EmpID	Name	DeptID	Salary
1	Alice	D01	50000

Any of the following can be a Super Key:
{EmpID}
{EmpID, Name}
{EmpID, Name, DeptID}
The smallest Super Key is the Candidate Key.

Constraints in DBMS –

In relational database design, constraints restrict the values that can be inserted, modified, or deleted in a relation, ensuring data integrity. Unlike Entity-Relationship models, relational databases support these restrictions. Constraints are categorized as:

1. **Implicit Constraints**: Built-in rules inherent to the relational data model (e.g., no duplicate tuples in a relation).
2. **Schema-Based (Explicit) Constraints**: Defined directly in the schema using DDL (e.g., primary keys, foreign keys).
3. **Application-Based (Semantic) Constraints**: Enforced at the application level, as they cannot be directly implemented in the schema.

Relational Constraints

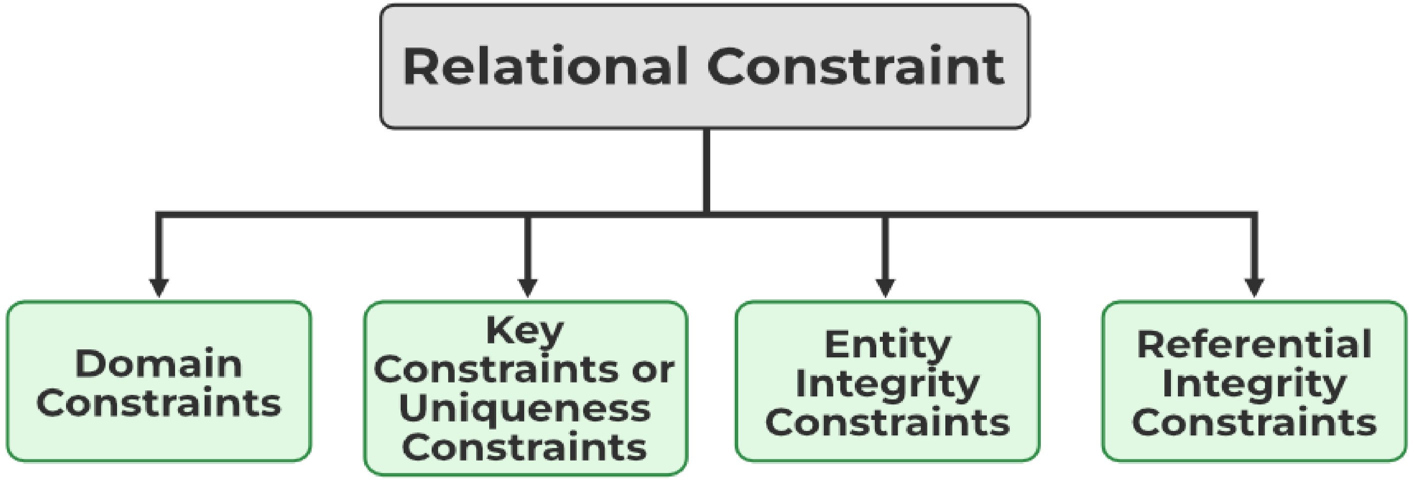
These are the restrictions or sets of rules imposed on the database contents. It validates the quality of the database. It validates the various operations like data insertion, updation, and other processes that have to be performed without affecting the integrity of the data. It protects us against threats/damages to the database. Mainly Constraints on the relational database are of 4 types

Domain constraints

Key constraints or Uniqueness Constraints

Entity Integrity constraints

Referential integrity constraints


Relational Constraint
Domain Constraints
Key Constraints or Uniqueness Constraints
Entity Integrity Constraints
Referential Integrity Constraints

1. Domain Constraints

Every domain must contain atomic values(smallest indivisible units) which means composite and multi-valued attributes are not allowed.

We perform a datatype check here, which means when we assign a data type to a column we limit the values that it can contain. Eg. If we assign the datatype of attribute age as int, we can’t give it values other than int datatype.

Every attribute (column) in a relation (table) is associated with a data type and permissible set of values.

EID	Name	Phone
01	Priya	123456789 234565656

Explanation: In the above relation, Name is a composite attribute and Phone is a multi-values attribute, so it is violating domain constraint.

2. Key Constraints or Uniqueness Constraints

These are called uniqueness constraints since it ensures that every tuple in the relation should be unique.

A relation can have multiple keys or candidate keys(minimal superkey), out of which we choose one of the keys as the primary key, we don't have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes.

Null values are not allowed in the primary key, hence Not Null constraint is also part of the key constraint.

EID	Name	Phone
01	Priya	2345678901
02	Sumit	1234567890
01	Akansha	4567890321

Explanation: In the above table, EID is the primary key, and the first and the last tuple have the same value in EID ie 01, so it is violating the key constraint.

3. Entity Integrity Constraints

Entity Integrity constraints say that no primary key can take a NULL value, since using the primary key we identify each tuple uniquely in a relation.

EID	Name	Phone
01	Priya	1234678900
02	Parul	6026789010
NULL	Pankaj	1234567890

Explanation: In the above relation, EID is made the primary key, and the primary key can't take NULL values but in the third tuple, the primary key is null, so it is violating Entity Integrity constraints.

4. Referential Integrity Constraints

The Referential integrity constraint is specified between two relations or tables and used to maintain the consistency among the tuples in two relations.

This constraint is enforced through a foreign key, when an attribute in the foreign key of relation R1 has the same domain(s) as the primary key of relation R2, then the foreign key of R1 is said to reference or refer to the primary key of relation R2.

Employee

EID	Name	DNO
01	Deeksha	12
02	Diya	22
04	Vinay	14

Department

DNO	Place
12	Jaipur
13	Mumbai
14	Delhi

Explanation: In the above tables, the DNO of Table 1 is the foreign key, and DNO in Table 2 is the primary key. DNO = 22 in the foreign key of Table 1 is not allowed because DNO = 22 is not defined in the primary key of table 2. Therefore, Referential integrity constraints are violated here.

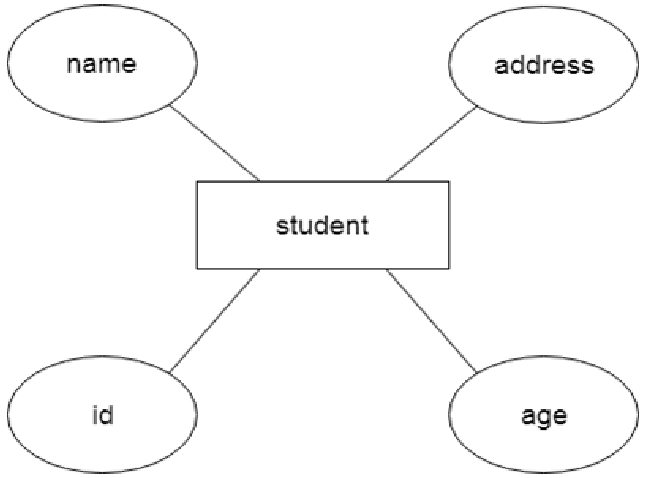
ER (Entity Relationship) Diagram in DBMS

ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.

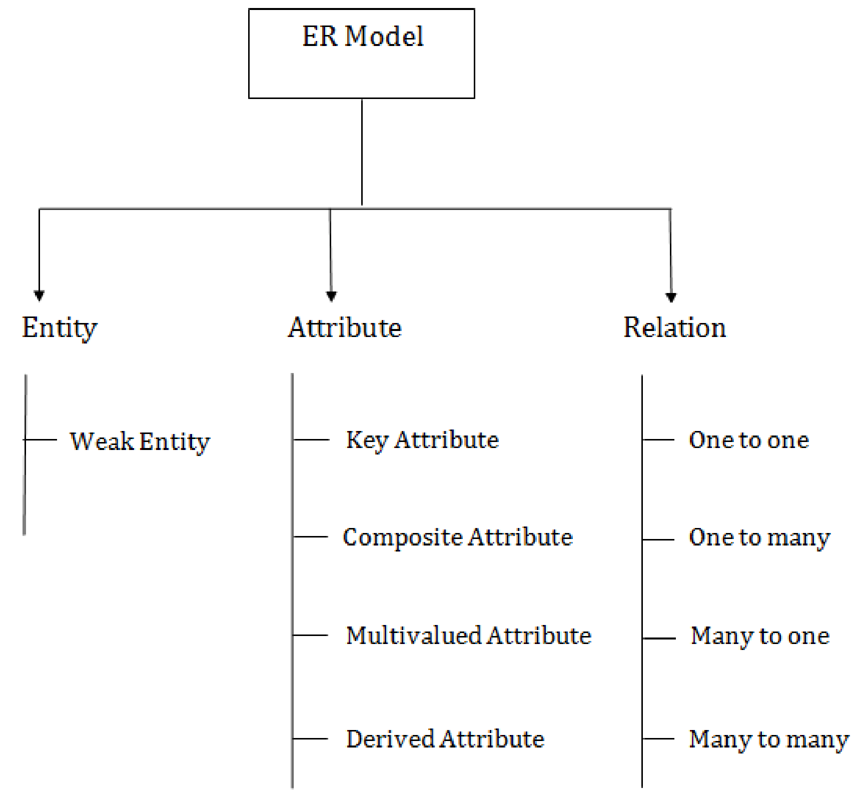
It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.

In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

For example, Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.



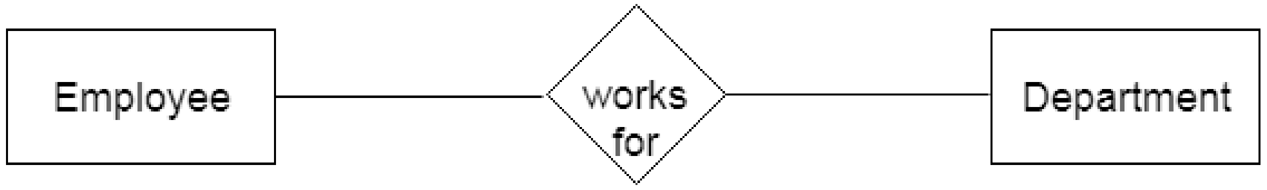
Component of ER Diagram



1. Entity:

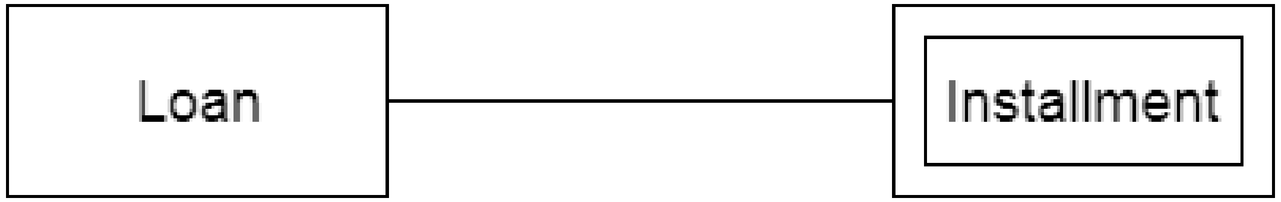
An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



a. Weak Entity

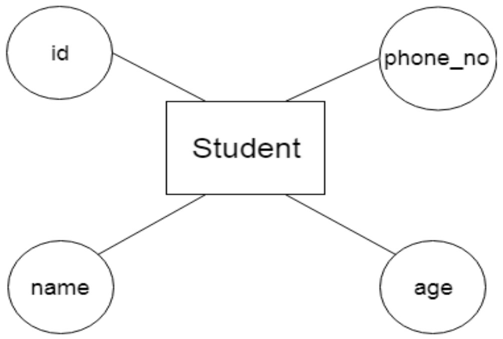
An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



2. Attribute

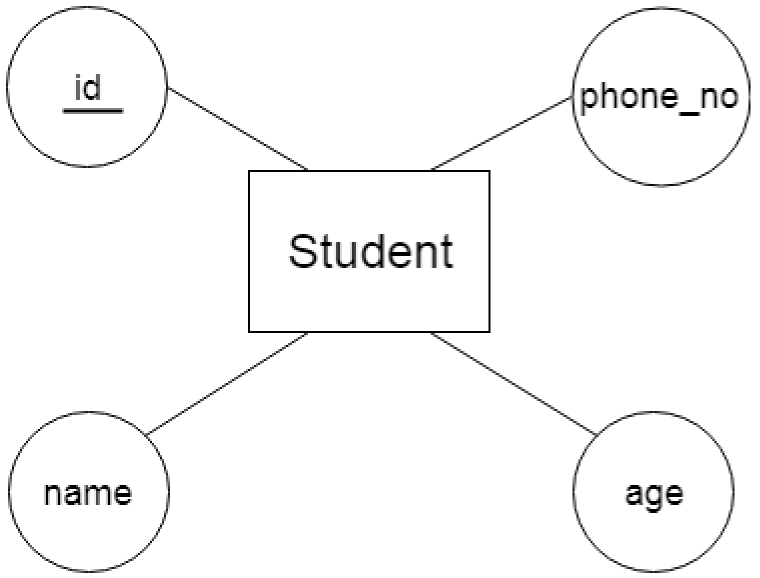
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

For example, id, age, contact number, name, etc. can be attributes of a student.



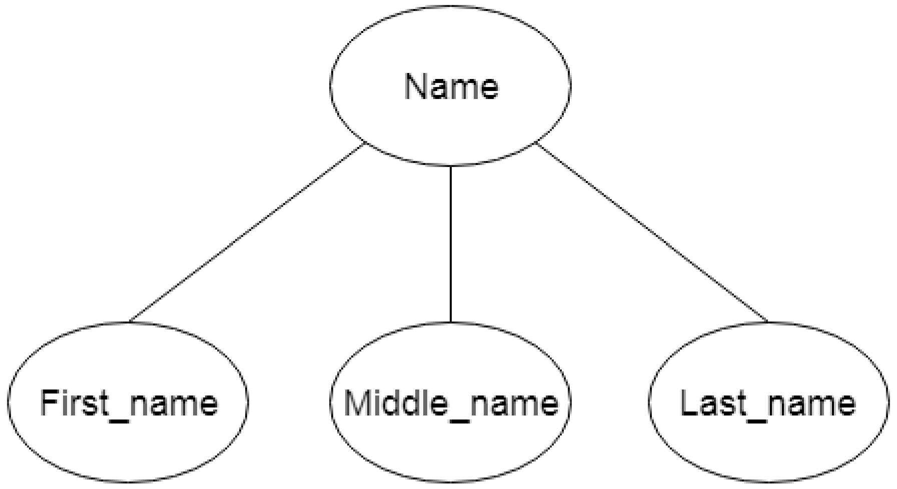
a. Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute

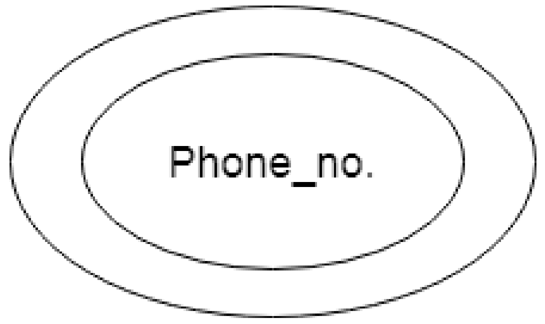
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

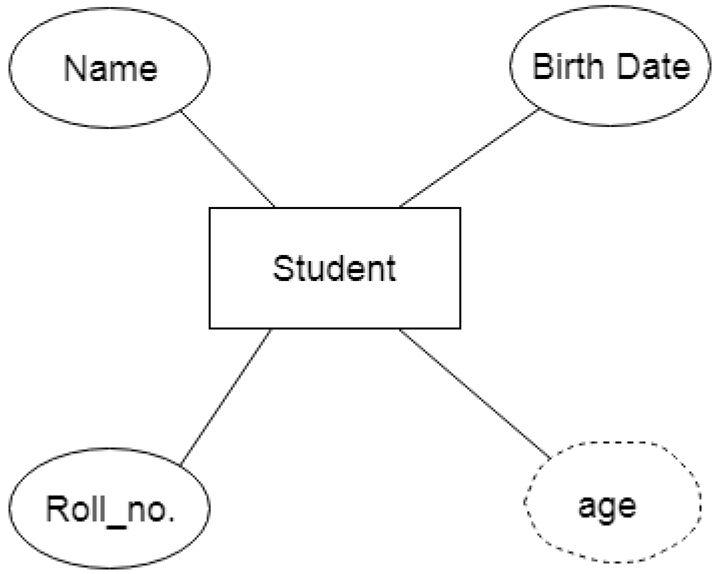
For example, a student can have more than one phone number.



d. Derived Attribute

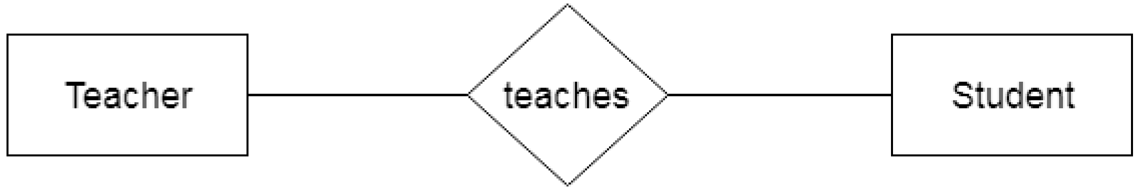
An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

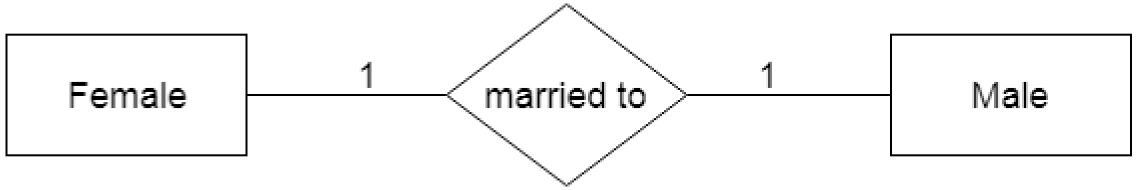
A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.



Types of relationship are as follows:

a. One-to-One Relationship

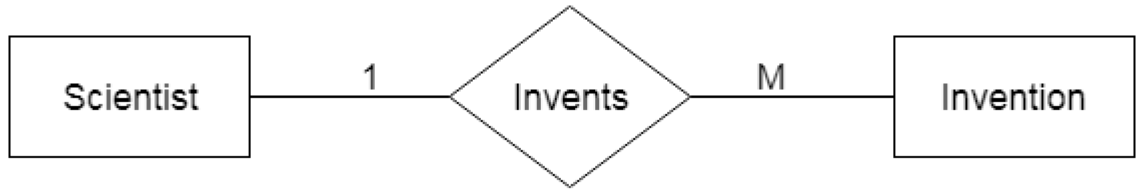
When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.
For example, A female can marry to one male, and a male can marry to one female.



b. One-to-many relationship

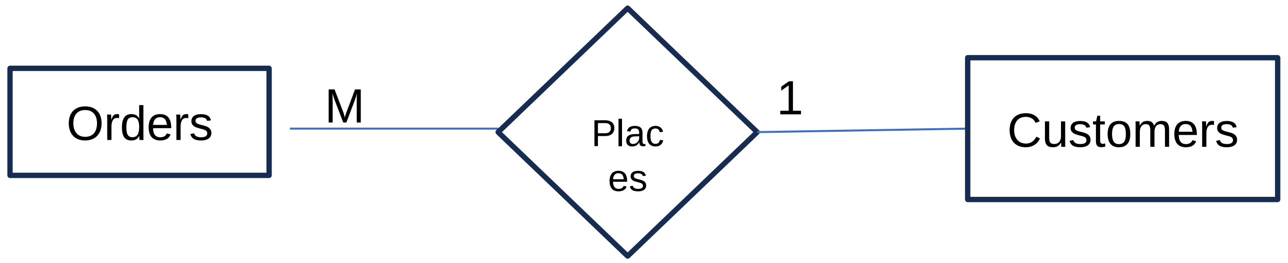
When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c. Many-to-one relationship

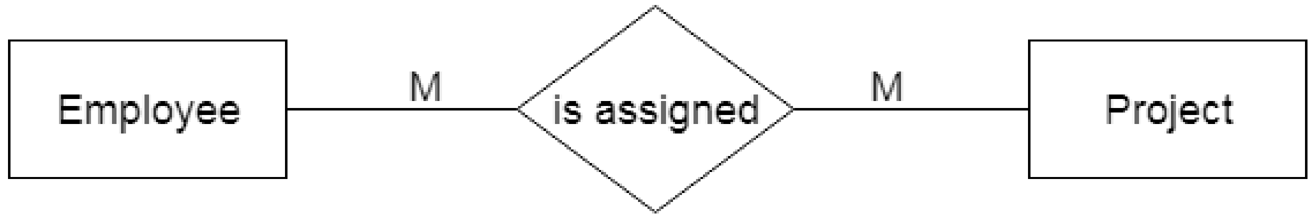
When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.



d. Many-to-many relationship

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.



ER Design Issues

When designing a database, several challenges or issues can arise that impact its efficiency, scalability, and reliability. Here are the basic design issues of an ER database schema -

1) Use of Entity Set vs Attributes

The use of an entity set or attribute depends on the structure of the real-world enterprise that is being modelled and the semantics associated with its attributes. It leads to a mistake when the user use the primary key of an entity set as an attribute of another entity set. Instead, he should use the relationship to do so. Also, the primary key attributes are implicit in the relationship set, but we designate it in the relationship sets.

2) Use of Entity Set vs. Relationship Sets

It is difficult to examine if an object can be best expressed by an entity set or relationship set. To understand and determine the right use, the user need to designate a relationship set for describing an action that occurs in-between the entities. If there is a requirement of representing the object as a relationship set, then its better not to mix it with the entity set.

3) Use of Binary vs n-array Relationship Sets

Generally, the relationships described in the databases are binary relationships. However, non-binary relationships can be represented by several binary relationships. For example, we can create and represent a ternary relationship 'parent' that may relate to a child, his father, as well as his mother. Such relationship can also be represented by two binary relationships i.e, mother and father, that may relate to their child. Thus, it is possible to represent a non-binary relationship by a set of distinct binary relationships.

4) Placing Relationship Attributes

The cardinality ratios can become an affective measure in the placement of the relationship attributes. So, it is better to associate the attributes of one-to-one or one-to-many relationship sets with any participating entity sets, instead of any relationship set. The decision of placing the specified attribute as a relationship or entity attribute should possess the characteristics of the real world enterprise that is being modelled.

For example, if there is an entity which can be determined by the combination of participating entity sets, instead of determining it as a separate entity. Such type of attribute must be associated with the many-to-many relationship sets.

Thus, it requires the overall knowledge of each part that is involved in designing and modelling an ER diagram. The basic requirement is to analyse the real-world enterprise and the connectivity of one entity or attribute with other.

Weak entity sets

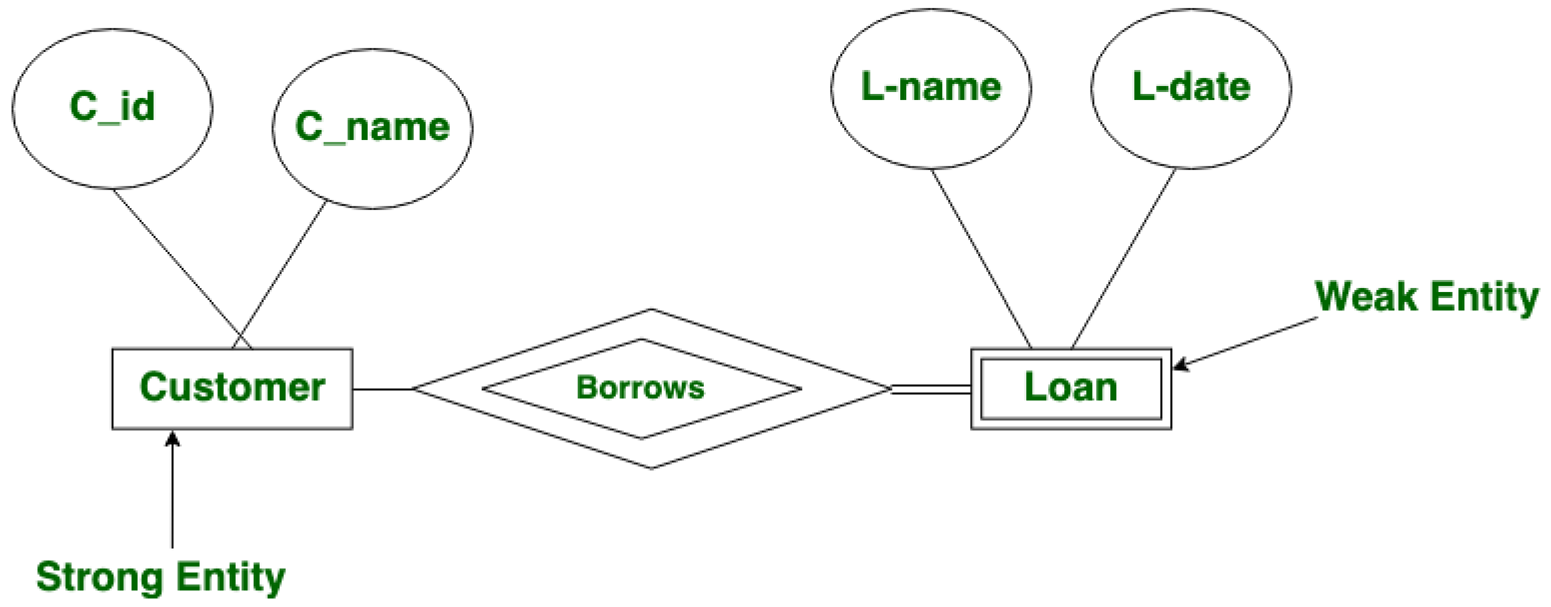
An entity type should have a key attribute which uniquely identifies each entity in the entity set, but there exists some entity type for which key attribute can’t be defined. These are called Weak Entity type.

The entity sets which do not have sufficient attributes to form a primary key are known as **weak entity sets** and the entity sets which have a primary key are known as strong entity sets.

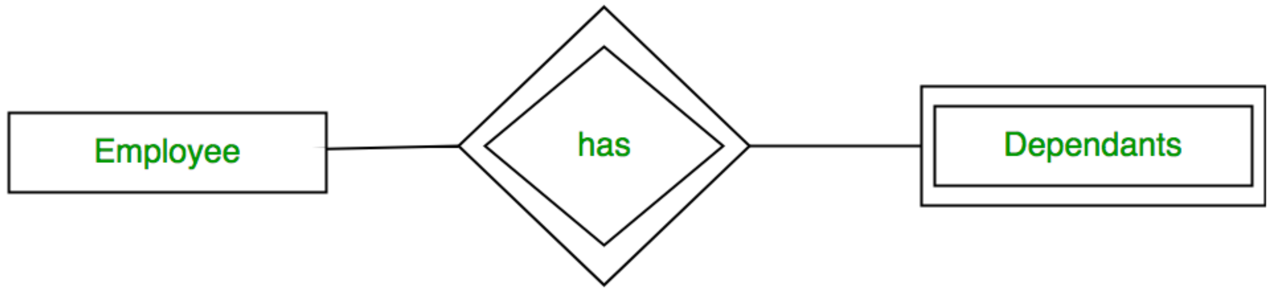
As the weak entities do not have any primary key, they cannot be identified on their own, so they depend on some other entity (known as owner entity). The weak entities have total participation constraint(existence dependency) in its identifying relationship with owner identity. Weak entity types have partial keys. Partial Keys are set of attributes with the help of which the tuples of the weak entities can be distinguished and identified.

Note – Weak entity always has total participation but Strong entity may not have total participation.

Weak entity is **depend on strong entity** to ensure the existence of weak entity. Like strong entity, weak entity does not have any primary key, It has partial discriminator key. Weak entity is represented by double rectangle. The relation between one strong and one weak entity is represented by double diamond.



Weak entities are represented with **double rectangular** box in the ER Diagram and the identifying relationships are represented with double diamond. Partial Key attributes are represented with dotted lines.



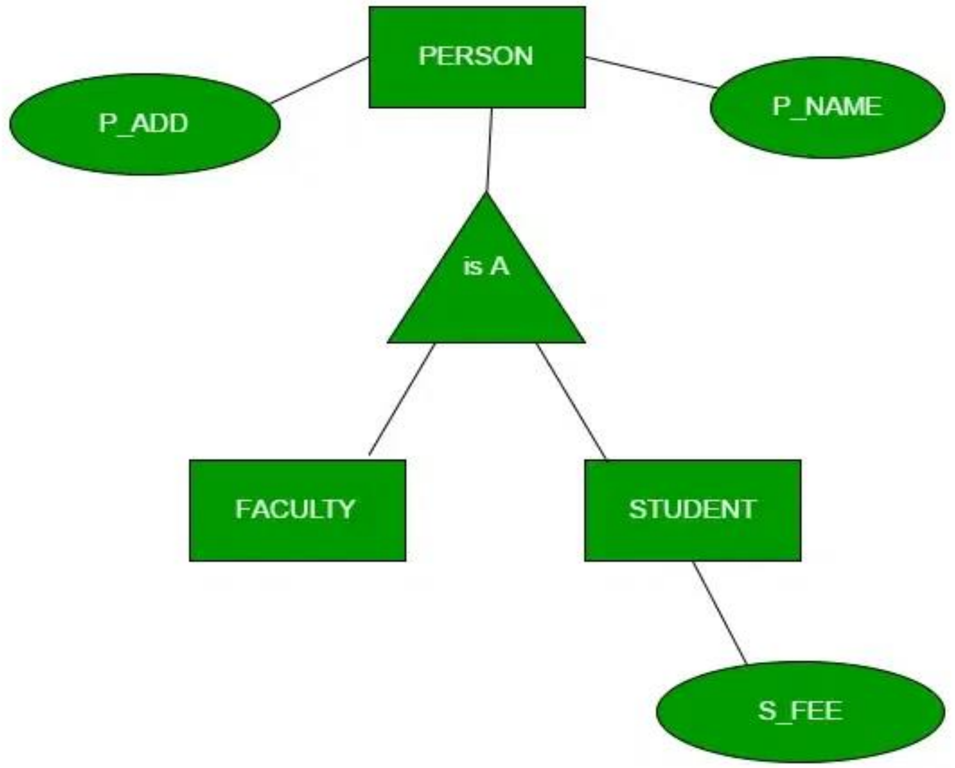
Extended E-R features : Generalization, Specialization and Aggregation

Using the ER model for bigger data creates a lot of complexity while designing a database model, So in order to minimize the complexity Generalization, Specialization, and Aggregation were introduced in the ER model. These were used for data abstraction. In which an abstraction mechanism is used to hide details of a set of objects.

Generalization

Generalization is the process of extracting common properties from a set of entities and creating a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher-level entity if they have some attributes in common. For Example, STUDENT and FACULTY can be generalized to a higher-level entity called PERSON. In this case, common attributes like P_NAME, and P_ADD become part of a higher entity (PERSON), and specialized attributes like S_FEE become part of a specialized entity (STUDENT).

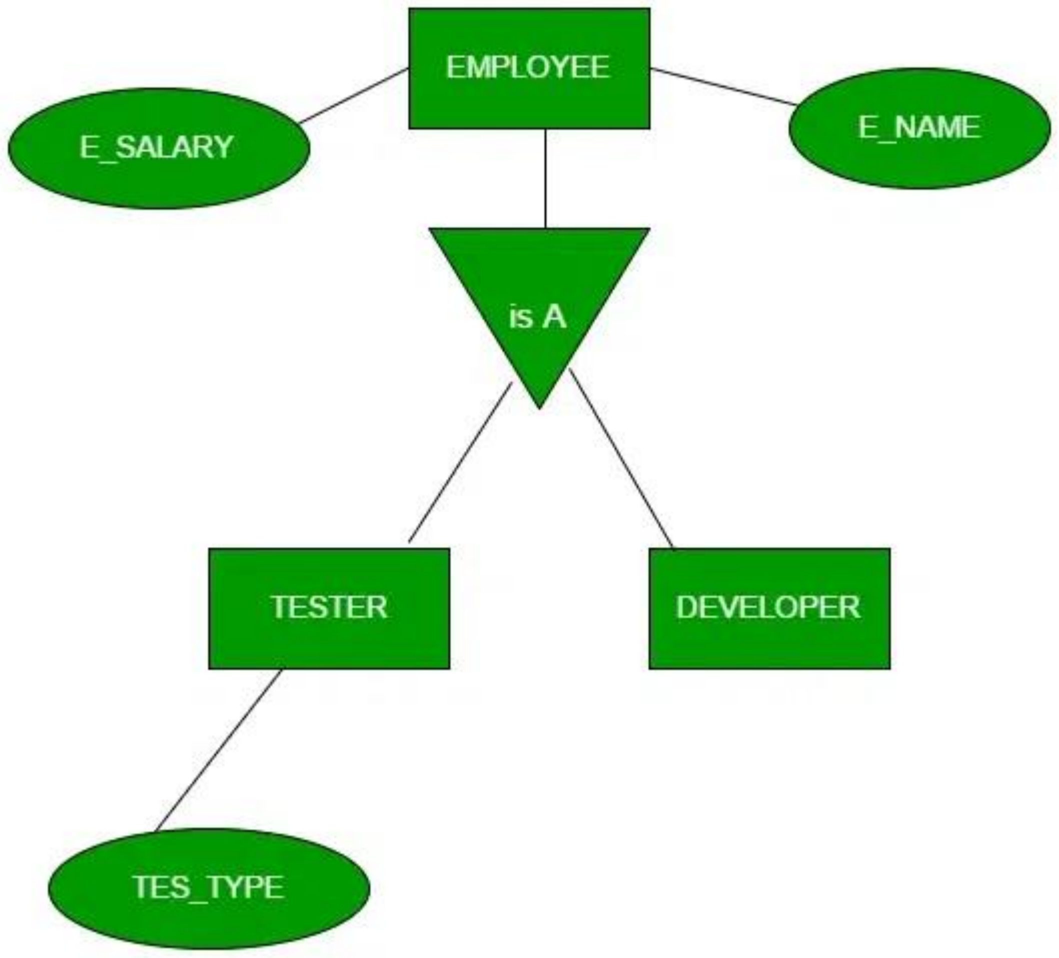
Generalization is also called as ‘ Bottom-up approach”.



Specialization

In specialization, an entity is divided into sub-entities based on its characteristics. It is a top-down approach where the higher-level entity is specialized into two or more lower-level entities. For Example, an EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER, etc. as shown in Figure 2. In this case, common attributes like E_NAME, E_SAL, etc. become part of a higher entity (EMPLOYEE), and specialized attributes like TES_TYPE become part of a specialized entity (TESTER).

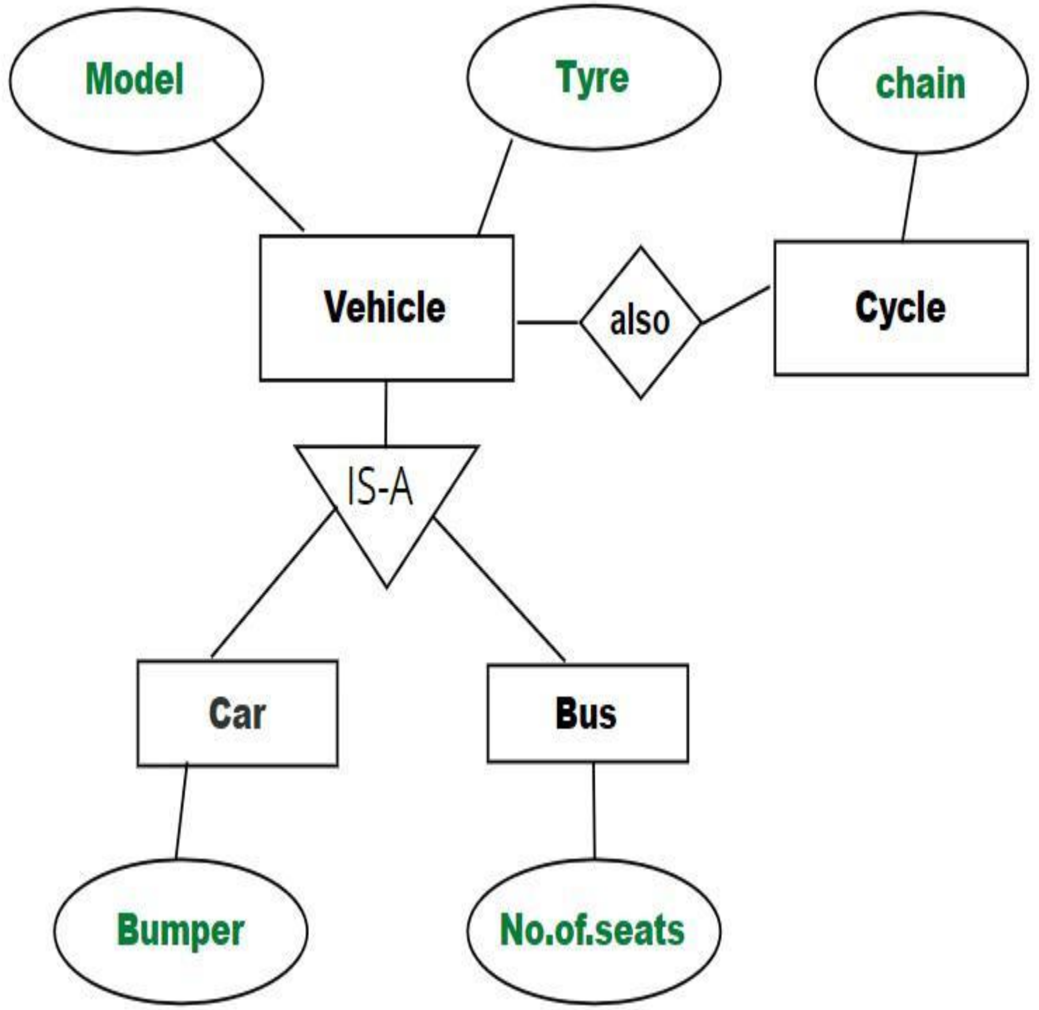
Specialization is also called as " Top-Down approch".



Specialization

Inheritance: It is an important feature of generalization and specialization

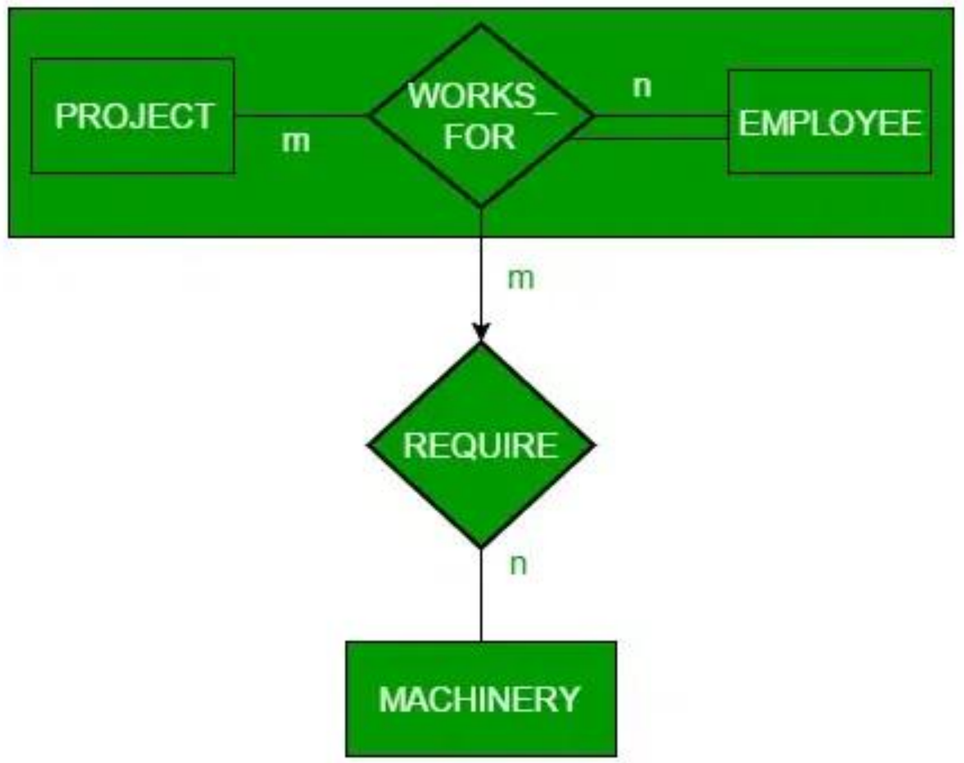
Attribute inheritance : It allows lower level entities to inherit the attributes of higher level entities and vice versa. In diagram **Car** entity is an inheritance of **Vehicle** entity ,So Car can acquire attributes of **Vehicle.** Example:car can acquire **Model** attribute of **Vehicle.**



Aggregation

An ER diagram is not capable of representing the relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher-level entity. Aggregation is an abstraction through which we can represent relationships as higher-level entity sets.

For Example, an Employee working on a project may require some machinery. So, REQUIRE relationship is needed between the relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into a single entity and relationship REQUIRE is created between the aggregated entity and MACHINERY.

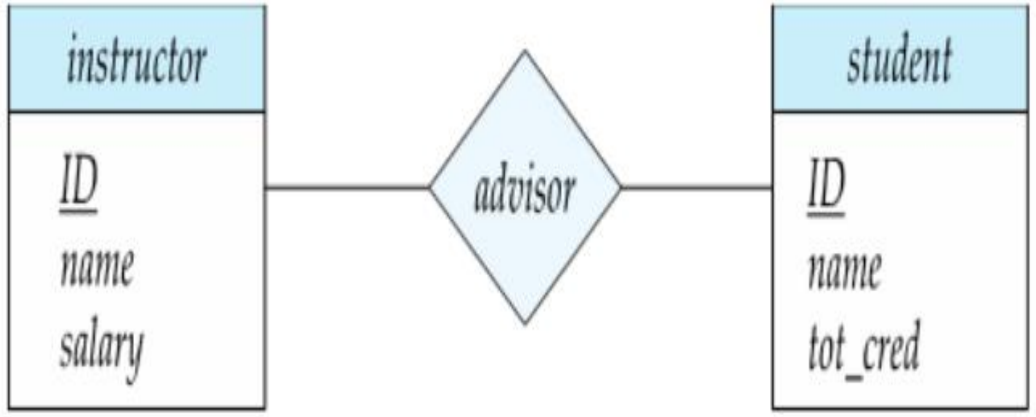


Aggregation

Reduction of E-R diagrams

Let there be ‘m’ number of entities and ‘n’ number of relationships in a given E-R diagram.The number of relations, after converting the E-R diagram to relations, is **m+n**.

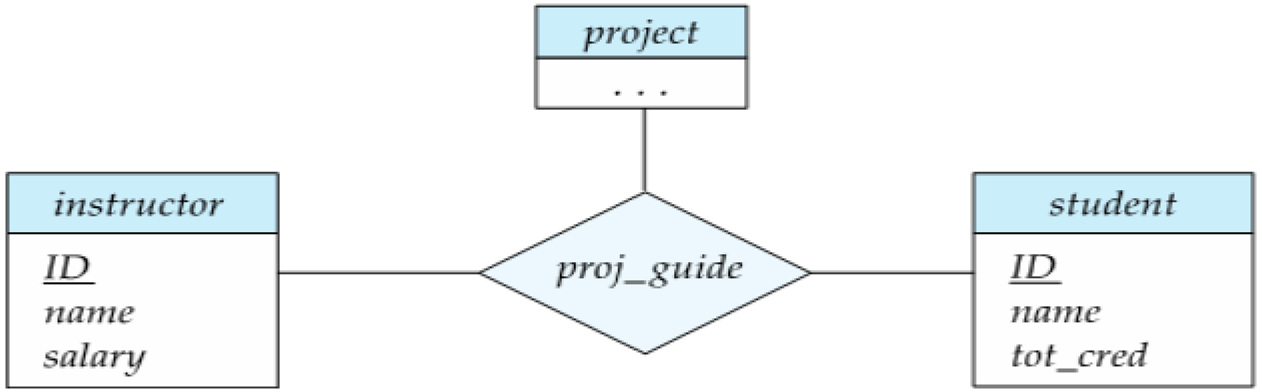
Example -



In the above example, m = 2 and n = 1. (Two entities and one relationship)

The attributes in the relationship table are the combination of primary keys of the entities that are connected through this relationship.

Table-1 : INSTRUCTOR	:	ID, I-name, salary
Table-2 : STUDENT	:	ID, S-name, tot_credits
Table-3 : ADVISOR	:	I-ID, S-ID



In the above example, the tables are :

Table-1 : INSTRUCTOR	:	I-ID, I-name, salary
Table-2 : STUDENT	:	S-ID, I-name, tot_cred
Table-3 : PROJECT	:	P-ID, P-Name
Table-4 : PROJ-GUIDE	:	I-ID, S-ID, P-ID

Data redundancy

In DBMS, when the same data is stored in different tables, it causes data redundancy. Sometimes, it is done on purpose for recovery or backup of data, faster access of data, or updating data easily. Redundant data costs extra money, demands higher storage capacity, and requires extra effort to keep all the files up to date.
Sometimes, unintentional duplicity of data causes a problem for the database to work properly, or it may become harder for the end user to access data. Redundant data unnecessarily occupy space in the database to save identical copies, which leads to space constraints, which is one of the major problems.

Example - there is a "Student" table that contains data such as "Student_id", "Name", "Course", "Session", "Fee", and "Department". As you can see, some data is repeated in the table, which causes redundancy.

Student_id	**Name**	**Course**	**Session**	**Fee**	**Department**
101	Devi	B. Tech	2022	90,000	CS
102	Sona	B. Tech	2022	90,000	CS
103	Varun	B. Tech	2022	90,000	CS
104	Satish	B. Tech	2022	90,000	CS
105	Amisha	B. Tech	2022	90,000	CS

Problems that are caused due to redundancy in the database

Redundancy in DBMS gives rise to anomalies, and we will study it further. In a database management system, the problems that occur while working on data include inserting, deleting, and updating data in the database.

We will understand these anomalies with the help of the following student table

student_id	student_name	student_age	dept_id	dept_name	dept_head
1	Shiva	19	104	Information Technology	Jaspreet Kaur
2	Khushi	18	102	Electronics	Avni Singh
3	Harsh	19	104	Information Technology	Jaspreet Kaur

1. Insertion Anomaly:
Insertion anomaly arises when you are trying to insert some data into the database, but you are not able to insert it.

Example: If you want to add the details of the student in the above table, then you must know the details of the department; otherwise, you will not be able to add the details because student details are dependent on department details.

2. Deletion Anomaly:
Deletion anomaly arises when you delete some data from the database, but some unrelated data is also deleted; that is, there will be a loss of data due to deletion anomaly.

Example: If we want to delete the student detail, which has student_id 2, we will also lose the unrelated data, i.e., department_id 102, from the above table.

3. Updating Anomaly:
An update anomaly arises when you update some data in the database, but the data is partially updated, which causes data inconsistency.

Example: If we want to update the details of dept_head from Jaspreet Kaur to Ankit Goyal for Dept_id 104, then we have to update it everywhere else; otherwise, the data will get partially updated, which causes data inconsistency.

Advantages of data redundancy in DBMS

Provides Data Security: Data redundancy can enhance data security as it is difficult for cyber attackers to attack data that are in different locations.

Provides Data Reliability: Reliable data improves accuracy because organizations can check and confirm whether data is correct.

Create Data Backup: Data redundancy helps in backing up the data.

Disadvantages of data redundancy in DBMS

Data corruption: Redundant data leads to high chances of data corruption.

Wastage of storage: Redundant data requires more space, leading to a need for more storage space.

High cost: Large storage is required to store and maintain redundant data, which is costly.
How to reduce data redundancy in DBMS

.

We can reduce data redundancy using the following methods:

Database Normalization: We can normalize the data using the normalization method. In this method, the data is broken down into pieces, which means a large table is divided into two or more small tables to remove redundancy. Normalization removes insert anomaly, update anomaly, and delete anomaly.

Deleting Unused Data: It is important to remove redundant data from the database as it generates data redundancy in the DBMS. It is a good practice to remove unwanted data to reduce redundancy.

Normal forms :1NF, 2NF, 3NF, BCNF and 4NF

Normalization is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

Normalization of DBMS

In database management systems (DBMS), normal forms are a series of guidelines that help to ensure that the design of a database is efficient, organized, and free from data anomalies. There are several levels of normalization, each with its own set of guidelines, known as normal forms.

Important Points Regarding Normal Forms in DBMS

First Normal Form (1NF): This is the most basic level of normalization. In 1NF, each table cell should contain only a single value, and each column should have a unique name. The first normal form helps to eliminate duplicate data and simplify queries.

Second Normal Form (2NF): 2NF eliminates redundant data by requiring that each non-key attribute be dependent on the primary key. This means that each column should be directly related to the primary key, and not to other columns.

Third Normal Form (3NF): 3NF builds on 2NF by requiring that all non-key attributes are independent of each other. This means that each column should be directly related to the primary key, and not to any other columns in the same table.

Boyce-Codd Normal Form (BCNF): BCNF is a stricter form of 3NF that ensures that each determinant in a table is a candidate key. In other words, BCNF ensures that each non-key attribute is dependent only on the candidate key.

Fourth Normal Form (4NF): 4NF is a further refinement of BCNF that ensures that a table does not contain any multi-valued dependencies.

Normal forms help to reduce data redundancy, increase data consistency, and improve database performance. However, higher levels of normalization can lead to more complex database designs and queries. It is important to strike a balance between normalization and practicality when designing a database.

Advantages of Normal Form

Reduced data redundancy: Normalization helps to eliminate duplicate data in tables, reducing the amount of storage space needed and improving database efficiency.

Improved data consistency: Normalization ensures that data is stored in a consistent and organized manner, reducing the risk of data inconsistencies and errors.

Simplified database design: Normalization provides guidelines for organizing tables and data relationships, making it easier to design and maintain a database.

Improved query performance: Normalized tables are typically easier to search and retrieve data from, resulting in faster query performance.

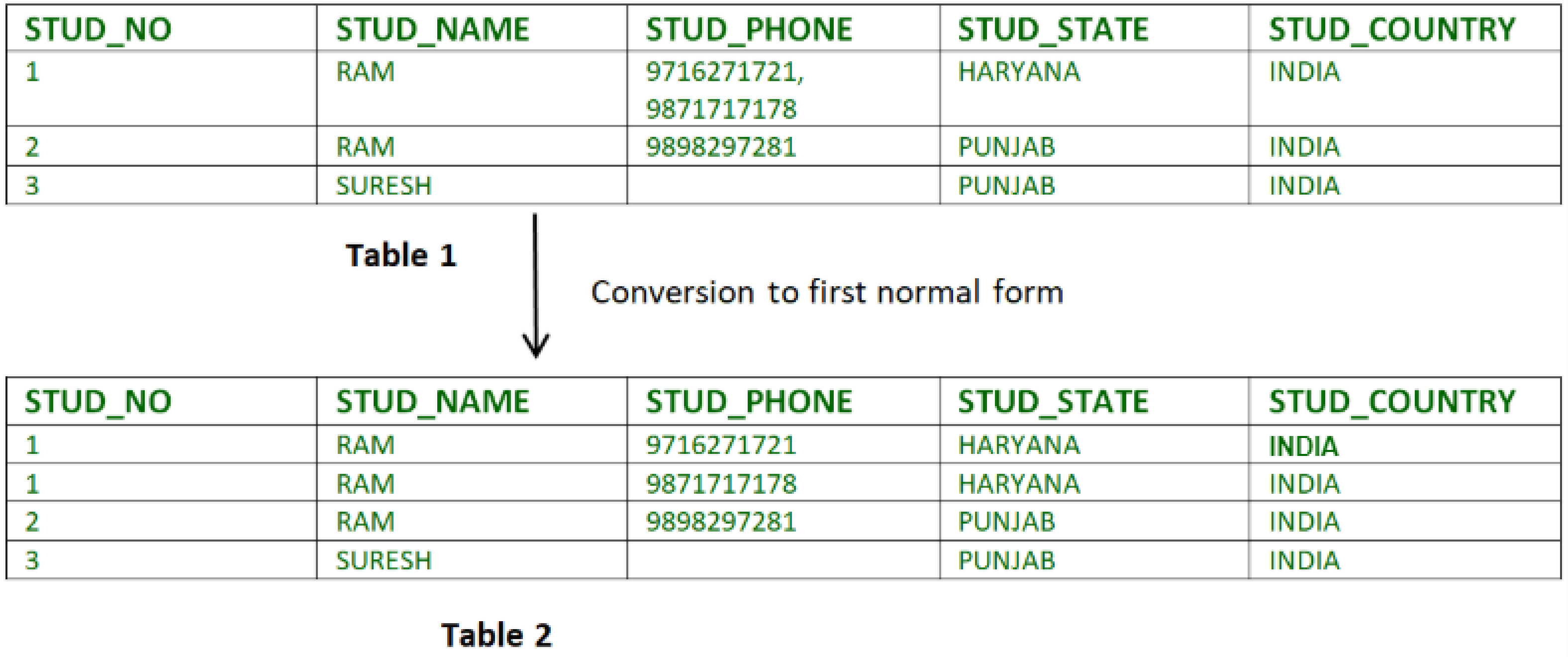
Easier database maintenance: Normalization reduces the complexity of a database by breaking it down into smaller, more manageable tables, making it easier to add, modify, and delete data.

Overall, using normal forms in DBMS helps to improve data quality, increase database efficiency, and simplify database design and maintenance.

First Normal Form

If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

Example 1 – Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.



STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	INDIA
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

Example 2 –

ID	Name	Courses
1	A	c1, c2
2	E	c3
3	M	c2, c3

In the above table Course is a multi-valued attribute so it is not in 1NF. Below Table is in 1NF as there is no multi-valued attribute

ID	Name	Course
1	A	c1
1	A	c2
2	E	c3
3	M	c2
3	M	c3

Second Normal Form

A relation is in 2NF if it is in 1NF and any non-prime attribute (attributes which are not part of any candidate key) is not partially dependent on any proper subset of any candidate key of the table. In other words, we can say that, every non-prime attribute must be fully dependent on each candidate key.

A functional dependency X->Y (where X and Y are set of attributes) is said to be in **partial dependency**, if Y can be determined by any proper subset of X.

However, in 2NF it is possible for a prime attribute to be partially dependent on any candidate key, but every non-prime attribute must be fully dependent(or not partially dependent) on each candidate key of the table.

Example 1 – Consider table-3 as following below.

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

{Note that, there are many courses having the same course fee} Here, COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO; COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO; COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO; Hence, COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ; But, COURSE_NO -> COURSE_FEE, i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF. To convert the above relation to 2NF, we need to split the table into two tables such as :

Table 1: STUD_NO, COURSE_NO

Table 2: COURSE_NO, COURSE_FEE

Table 1

STUD_NO	COURSE_NO
1	C1
2	C2
1	C4
4	C3
4	C1

Table 2

COURSE_NO	COURSE_FEE
C1	1000
C2	1500
C3	1000
C4	2000
C5	2000

NOTE: 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking C1 course, we don’t need to store its Fee as 1000 for all the 100 records, instead, once we can store it in the second table as the course fee for C1 is 1000.

Third Normal Form

A relation is said to be in third normal form, if we did not have any transitive dependency for non-prime attributes. The basic condition with the Third Normal Form is that, the relation must be in Second Normal Form.

Below mentioned is the basic condition that must be hold in the non-trivial functional dependency X -> Y:

X is a Super Key.

Or

Y is a Prime Attribute (this means that element of Y is some part of Candidate Key).

BCNF

BCNF (Boyce-Codd Normal Form) is just a advanced version of Third Normal Form. Here we have some additional rules than Third Normal Form. The basic condition for any relation to be in BCNF is that it must be in Third Normal Form.

We have to focus on some basic rules that are for BCNF:

1. Table must be in Third Normal Form.
2. In relation X->Y, X must be a superkey in a relation.

Fourth Normal Form

Fourth Normal Form contains no non-trivial multivalued dependency except candidate key. The basic condition with Fourth Normal Form is that the relation must be in BCNF.

The basic rules are mentioned below.

1. It must be in BCNF.
2. It does not have any multi-valued dependency.

Example 1: In relation STUDENT given in Table 4, FD set: {STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE}
Candidate Key: {STUD_NO}
For this relation in table 4, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true.
So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form.

To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as: STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE) STATE_COUNTRY (STATE, COUNTRY)
Consider relation R(A, B, C, D, E) A -> BC, CD -> E, B -> D, E -> A All possible candidate keys in above relation are {A, E, CD, BC} All attributes are on right sides of all functional dependencies are prime.

Example 2: Find the highest normal form of a relation R(A,B,C,D,E) with FD set as {BC->D, AC->BE, B->E}

Step 1: As we can see, (AC)+ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.

Step 2: Prime attributes are those attributes that are part of candidate key {A, C} in this example and others will be non-prime {B, D, E} in this example.

Step 3: The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute. The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not a proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).
The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute. So the highest normal form of relation will be 2nd Normal form.
For example consider relation R(A, B, C) A -> BC, B -> A and B both are super keys so above relation is in BCNF.

Applications of Normal Forms in DBMS

Data consistency: Normal forms ensure that data is consistent and does not contain any redundant information. This helps to prevent inconsistencies and errors in the database.

Data redundancy: Normal forms minimize data redundancy by organizing data into tables that contain only unique data. This reduces the amount of storage space required for the database and makes it easier to manage.

Response time: Normal forms can improve query performance by reducing the number of joins required to retrieve data. This helps to speed up query processing and improve overall system performance.

Database maintenance: Normal forms make it easier to maintain the database by reducing the amount of redundant data that needs to be updated, deleted, or modified. This helps to improve database management and reduce the risk of errors or inconsistencies.

Database design: Normal forms provide guidelines for designing databases that are efficient, flexible, and scalable. This helps to ensure that the database can be easily modified, updated, or expanded as needed.

Some Important Points about Normal Forms

BCNF is free from redundancy caused by Functional Dependencies.

If a relation is in BCNF, then 3NF is also satisfied.

If all attributes of relation are prime attribute, then the relation is always in 3NF.

A relation in a Relational Database is always and at least in 1NF form.

Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.

If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).

Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.

There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it’s generally not required to go beyond BCNF.

THANK YOU

CollegeDekho

REVOLUTIONIZING EDUCATION. TRANSFORMING INDIA.

GETMYUNI PrepBytes IELTSMaterial.com ImaginXP GetGIS.ORG UNIPRO EDUCATION ASSURED