DBMS QB – UNIT – 2

[4-marks]

1. What is an Entity in the ER Model? Give an example. An **entity** is a real-world object that has an **independent existence**. It is something about which data is collected and stored in a database.

1. An entity represents an **entity type**, which means a group of similar objects. Each entity type has multiple entity instances.
2. An **entity instance** is a single, specific example of an entity. It represents one actual object in the real world.
3. In an ER diagram, an entity is represented using a **rectangle symbol**. This helps in visually identifying entities in database design.

Example:
Employee is an entity, and a particular employee like "Employee with ID 101" is an entity instance.

2. Define Attribute and Relationship in an ER Model with examples.

Attribute

1. An **attribute** is a property or characteristic that describes an entity. It provides detailed information about the entity.
2. Attributes help in storing meaningful data related to an entity. Without attributes, entities have no useful information.
3. Each attribute has a **domain**, which is the set of possible values it can take. This ensures data consistency.
4. In ER diagrams, attributes are represented using an **oval shape**.

Example:
For the entity Student, attributes can be Student_ID, Name, and Age.

Relationship

1. A **relationship** represents an association between two or more entities. It shows how entities are connected.
2. Relationships help in understanding interactions between entities in the real world.
3. A relationship can exist between different entity types. It is an important part of database design.
4. In ER diagrams, relationships are shown using a **diamond symbol**.

Example:
Student enrolls in Course is a relationship between Student and Course entities.

3. List different notations used in an ER Diagram (entity, attribute, relationship).

1. **Entity notation** uses a **rectangle** to represent real-world objects. It clearly identifies what data objects exist in the system.
2. **Attribute notation** uses an **oval** to represent properties of an entity. Attributes describe the details of entities.
3. **Relationship notation** uses a **diamond shape** to represent associations between entities. It shows how entities are linked.
4. **Key attributes** are shown by **underlining the attribute name**. They uniquely identify each entity instance.

These notations make ER diagrams easy to understand and visually clear.

4. What is Specialization in ER Modeling?

1. **Specialization** is a **top-down approach** in ER modeling. A higher-level entity is divided into lower-level sub-entities.
2. It is used when entities have **distinct characteristics** that need to be represented separately. This improves data clarity.
3. Sub-entities inherit all attributes of the main entity. They may also have their own specific attributes.
4. Specialization helps in modeling **detailed and specific information** in a database design.

Example:
An Employee entity can be specialized into Permanent Employee and Contract Employee.

5. What is Generalization? Write one example.

1. **Generalization** is a **bottom-up approach** in ER modeling. Multiple lower-level entities are combined into one higher-level entity.
2. It is done when entities share **common attributes**. These common attributes are moved to the generalized entity.
3. Generalization helps in **reducing data redundancy**. It avoids repeating the same attributes in multiple entities.
4. It improves database design by making it more **organized and reusable**.

Example:
Student and Teacher entities can be generalized into a Person entity because both share common attributes like Name.

6. Define Aggregation in ER Model

1. **Aggregation** is an abstraction technique used in the ER model to represent a **relationship between a whole and its parts**. It is applied when a relationship itself needs to be treated as a higher-level entity.
2. Aggregation is used when **simple relationships are not sufficient** to represent real-world situations clearly. It helps in modeling complex relationships in a structured way.
3. In aggregation, **entities and their relationship are grouped together** and considered as a single abstract entity. This grouped entity can then participate in another relationship.
4. Aggregation is also called a **higher-order relationship**. It improves clarity and avoids confusion in complex ER diagrams.

7. What is a Primary Key? Give an example.

1. A **primary key** is an attribute or a set of attributes that **uniquely identifies each entity instance** in a table. No two records can have the same primary key value.
2. A primary key **cannot contain NULL values**, because every record must be uniquely identified. This ensures data accuracy and reliability.
3. Primary keys help in **maintaining entity integrity** in the database. They prevent duplicate records from being stored.
4. Primary keys are often used to **connect tables** through relationships. They play an important role in relational databases.

Example:
Student_ID in the Student table is a primary key.

8. Define Foreign Key and explain its purpose.

UNIT2_INTRODUCTION TO ENTITY-RE...

1. A **foreign key** is an attribute in one table that **refers to the primary key of another table**. It creates a link between two related tables.
2. The main purpose of a foreign key is to **maintain relationships between tables**. It ensures that data in one table matches data in another table.
3. Foreign keys help in **maintaining data consistency** across tables. They prevent invalid data from being inserted.
4. Foreign keys support **referential integrity** in a database. They ensure that referenced data actually exists.

Example:
Course_ID in Student table referring to Course_ID in Course table.

9. What is Referential Integrity?

1. **Referential integrity** is a constraint that ensures **consistency between related tables** in a relational database. It maintains valid relationships between tables.
2. It ensures that a **foreign key value must either match a primary key value** in another table or be NULL. This avoids invalid references.
3. Referential integrity prevents **orphan records**, which are records that reference non-existing data. This keeps the database reliable.
4. It helps in maintaining **accuracy and correctness of data**. Without referential integrity, database relationships can become inconsistent.

10. List Codd's 12 Rules (Names Only)

1. Rule 0 – Foundation Rule
2. Rule 1 – Information Rule
3. Rule 2 – Guaranteed Access Rule
4. Rule 3 – Systematic Treatment of NULL Values
5. Rule 4 – Dynamic Online Catalog
6. Rule 5 – Comprehensive Data Sublanguage Rule
7. Rule 6 – View Updating Rule
8. Rule 7 – High-Level Insert, Update, Delete
9. Rule 8 – Physical Data Independence
10. Rule 9 – Logical Data Independence
11. Rule 10 – Integrity Independence
12. Rule 11 – Distribution Independence
13. Rule 12 – Non-Subversion Rule

11. Define Selection and Projection in Relational Algebra

Selection (σ)

1. **Selection** is a relational algebra operation used to **select rows (tuples)** from a relation based on a given condition. It filters records that satisfy the condition.
2. Selection does not change the structure of the table. It only reduces the number of rows.
3. It is represented by the symbol **σ (sigma)**.
4. Selection is used when we want specific records, such as students with marks greater than 60.

Projection (π)

1. **Projection** is used to **select columns (attributes)** from a relation. It removes unwanted attributes from the result.
2. Projection reduces the number of columns but keeps all rows.
3. It is represented by the symbol **π (pi)**.
4. Projection is useful when only selected information like name and ID is required.

12. What is a Cartesian Product? Give one simple example.

1. **Cartesian Product** is a relational algebra operation that combines **every tuple of one relation with every tuple of another relation**.
2. If one table has *m* rows and another has *n* rows, the result will have $m \times n$ rows.
3. It produces a large number of records and is generally used with conditions.
4. Cartesian product is the base operation for forming joins.

Example: If Student table has 2 rows and Course table has 3 rows, the Cartesian Product will have **6 rows**.

13. What is a Join? List different types of Joins.

1. A **Join** is an operation used to **combine related data from two or more tables** based on a common attribute.
2. Joins help in retrieving meaningful information by connecting tables logically.
3. Without joins, data stored in different tables cannot be combined easily.
4. Joins are very important in relational databases.

Types of Joins

- Inner Join , Left Join , Right Join , Full Join

14. Define Super Key and Candidate Key

Super Key

1. A **Super Key** is a set of one or more attributes that can **uniquely identify a tuple** in a relation.
2. It may contain extra attributes that are not necessary for unique identification.
3. Every primary key is a super key, but not every super key is a primary key.
4. Super keys help in identifying records uniquely.

Candidate Key

1. A **Candidate Key** is a **minimal super key**. It uniquely identifies a tuple with no extra attributes.
2. A relation can have more than one candidate key.
3. One candidate key is chosen as the primary key.
4. Candidate keys are important for designing tables correctly.

15. What is a Tuple and what is a Relation?

Tuple

1. A **tuple** represents a **single row** in a table. It contains values for all attributes of the relation.
2. Each tuple represents one real-world record.
3. Tuples are also called records.
4. For example, details of one student form a tuple.

Relation

1. A **relation** is a **table** consisting of rows and columns.
2. Rows are called tuples and columns are called attributes.
3. A relation stores data about a particular entity.
4. For example, a Student table is a relation containing student records.

[5 -marks]

1] Types of Attributes in ER Model

Attributes are properties that describe an entity in an ER model.
First, a **simple attribute** cannot be divided into smaller parts and stores atomic data. For example, an employee ID is a simple attribute because it uniquely identifies an employee and cannot be broken further.

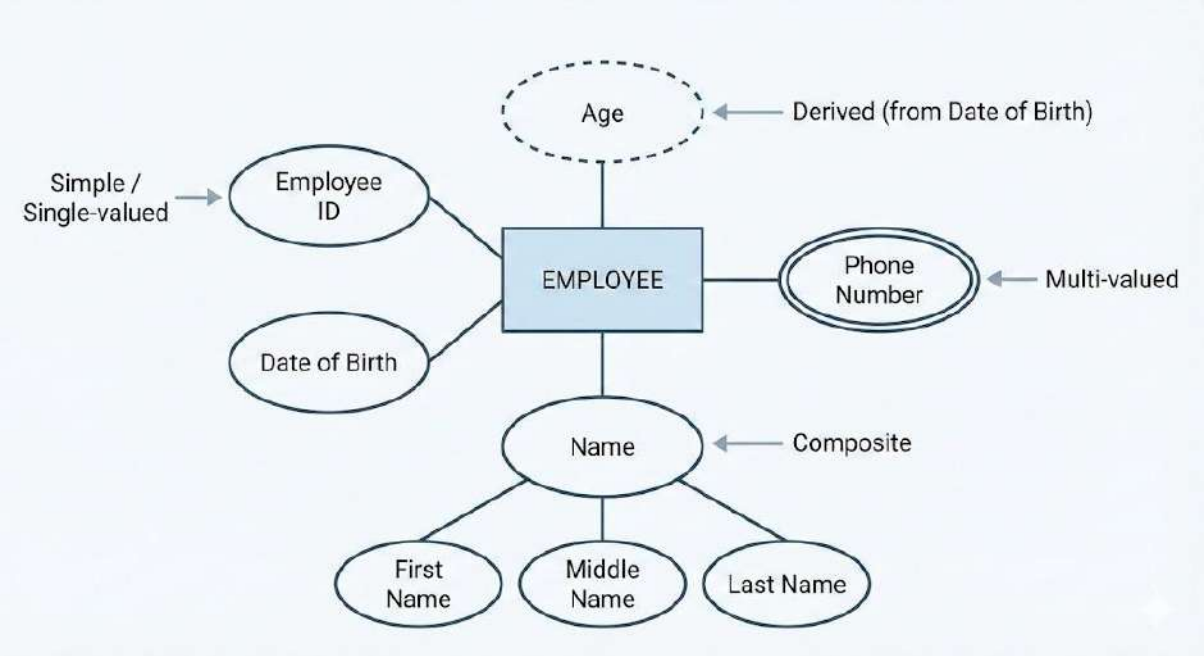
Second, a **composite attribute** can be divided into sub-attributes. For example, a name can be split into first name, middle name, and last name, which helps in storing structured information.

Third, a **single-valued attribute** can have only one value for each entity instance. Age of a student is single-valued because at a particular time, a student has only one age.

Fourth, a **multi-valued attribute** can have more than one value for a single entity. For example, an employee can have multiple phone numbers, making phone number a multi-valued attribute.

Finally, a **derived attribute** is calculated from other attributes and does not need to be stored permanently. For example, age can be derived from date of birth.

(no need to draw for 4 marks jus for ref)

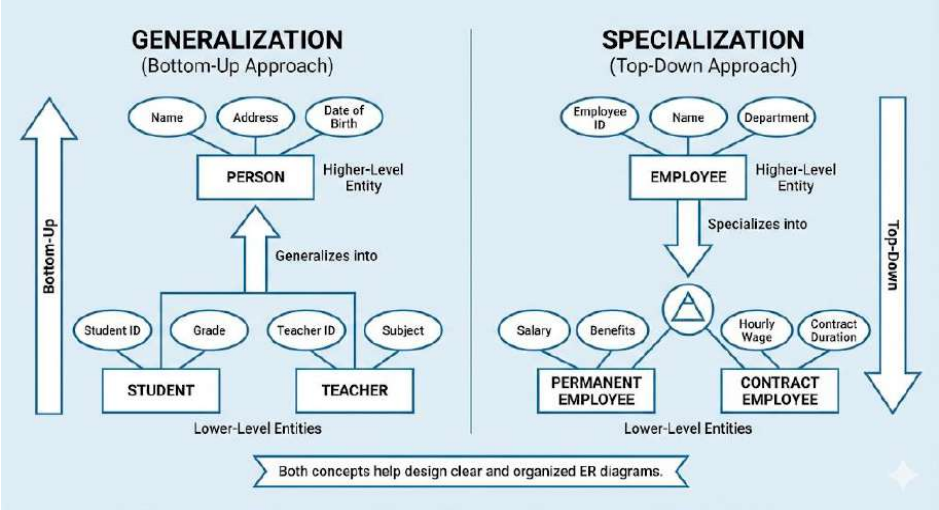


2. Comparison between Generalization and Specialization

Generalization is a bottom-up approach in ER modeling. It combines multiple lower-level entities into a single higher-level entity based on common attributes. This reduces data redundancy and improves reusability. For example, Student and Teacher entities can be generalized into a Person entity.

Specialization is a top-down approach. In this, a higher-level entity is divided into multiple lower-level entities based on specific characteristics. It allows more detailed data representation. For example, an Employee entity can be specialized into Permanent Employee and Contract Employee.

Generalization focuses on similarities, while specialization focuses on differences. Both help in designing clear and organized ER diagrams.



3. Aggregation in ER Model

Aggregation is an abstraction technique used in ER modeling to represent a relationship between a whole and its parts. It is used when a relationship itself needs to participate in another relationship. Aggregation is also called a higher-order relationship.

In complex systems, some relationships cannot be directly connected to entities. Aggregation allows grouping of entities and their relationship into a single abstract entity. This improves clarity and avoids confusion in ER diagrams.

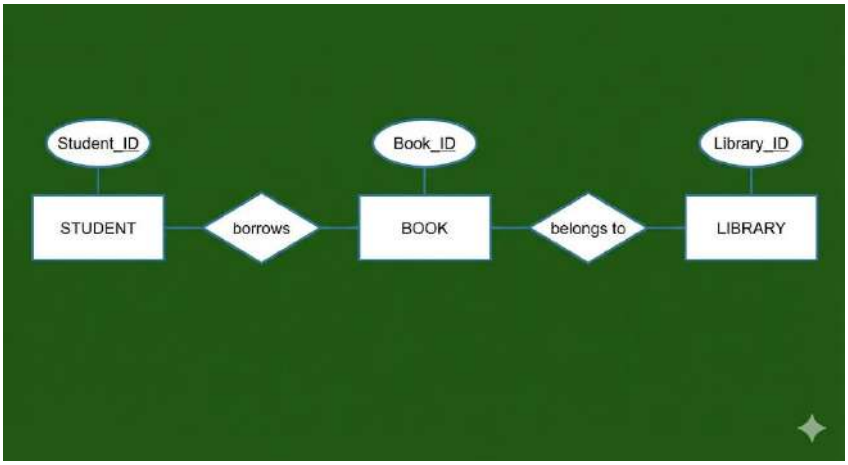
For example, an Employee works on a Project, and this work may require Machinery. The WORKS_FOR relationship is aggregated and linked to the Machinery entity.

4. ER Diagram for Library Management System

A Library Management System can be represented using an ER diagram with basic entities and relationships. The main entities are **Student**, **Book**, and **Library**.

The Student entity represents users of the library and stores student details. The Book entity represents books available in the library. The Library entity manages the collection of books.

A Student borrows a Book, which is shown using the “borrows” relationship. A Book belongs to a Library, represented by the “belongs to” relationship. This simple ER diagram clearly shows how students interact with books in a library system.



5. Structure of the Relational Model

The relational model represents data in the form of **tables**, also called relations. Each table consists of rows and columns. Rows are called tuples and represent individual records, while columns are called attributes and represent data fields.

Each table must have a **primary key** that uniquely identifies each row. The relational model maintains relationships between tables using keys. It provides a simple and logical way to store and access data.

For example, a Student table may have attributes such as Student_ID, Name, and Age. Each row represents one student, and Student_ID acts as the primary key.

6. Various Types of Keys in a Relational Model (with examples)

Keys are used in the relational model to **uniquely identify records** in a table and to maintain relationships between tables. They help in avoiding duplication and ensuring data accuracy.

1. **Primary Key**
   A primary key uniquely identifies each row in a table. It cannot contain duplicate values and cannot be NULL.
   Example: Student_ID in a Student table uniquely identifies each student.

2. **Candidate Key**
   Candidate keys are attributes that can uniquely identify a record. One of the candidate keys is selected as the primary key.
   Example: Student_ID and Email_ID can both uniquely identify a student.

3. **Super Key**
   A super key is a set of one or more attributes that uniquely identify a record. It may contain extra attributes.
   Example: {Student_ID, Name} is a super key.

4. **Foreign Key**
   A foreign key is an attribute in one table that refers to the primary key of another table. It helps in maintaining relationships.
   Example: Course_ID in Student table refers to Course_ID in Course table.

5. **Composite Key**
   A composite key is formed by combining two or more attributes to uniquely identify a record.
   Example: {Student_ID, Course_ID} in an enrollment table.

7. Referential Integrity Constraint (with example)

Referential integrity is a rule that **maintains consistency between related tables** in a relational database. It ensures that relationships between tables remain valid.

1. Referential integrity ensures that a **foreign key value must match a primary key value** in the referenced table. This prevents invalid data entries.

2. It avoids **orphan records**, which are records that reference non-existing data. This keeps the database consistent.

3. If a primary key value is updated or deleted, referential integrity rules decide what happens to related records. Common actions include restrict, cascade, or set NULL.

4. **Example:**
   If Course_ID is a primary key in Course table, then Course_ID in Student table must already exist in the Course table. A student cannot be enrolled in a non-existing course.

8. Any Five Rules of Codd's 12 Rules

Codd's rules define what a true relational database system should follow. These rules ensure data consistency, integrity, and independence.

1. **Rule 1 – Information Rule**
   All data must be stored in the form of tables. Each table consists of rows and columns only.
2. **Rule 2 – Guaranteed Access Rule**
   Each data item must be accessible by specifying table name, primary key, and column name. This avoids hidden data.
3. **Rule 3 – Systematic Treatment of NULL Values**
   NULL values must be handled uniformly. NULL represents missing or unknown information, not zero or blank.
4. **Rule 8 – Physical Data Independence**
   Changes in physical storage should not affect application programs. This allows easy database maintenance.
5. **Rule 10 – Integrity Independence**
   Integrity constraints should be stored in the database, not in application programs. This improves reliability.

9. Selection, Projection, and Rename Operators in Relational Algebra

Relational algebra operators are used to **retrieve and manipulate data** from relations.

1. **Selection (σ)**
   Selection retrieves rows from a table that satisfy a given condition. It filters records based on criteria.
   Example: Selecting students with age greater than 20.
2. **Projection (π)**
   Projection retrieves specific columns from a table. It removes unwanted attributes from the result.
   Example: Displaying only Student_ID and Name from Student table.
3. **Rename (ρ)**
   Rename changes the name of a relation or its attributes. It is useful when using the same table multiple times.
   Example: Renaming Student table as S.
4. These operators help in writing complex queries in a structured and mathematical way. They form the foundation of SQL.

10. SQL Equivalent of Union, Intersection, and Set Difference

SQL provides set operations to combine or compare results of two queries. These operations work on **compatible tables**.

1. **UNION**
   UNION combines the results of two queries and removes duplicate records. Both queries must have same number of columns.
   Example: Combining student lists from two departments.
2. **UNION ALL**
   UNION ALL also combines results but does not remove duplicates. It is faster than UNION.
3. **INTERSECTION (INTERSECT)**
   INTERSECT returns only the common records present in both queries. It finds matching data.
   Example: Students enrolled in both Course A and Course B.
4. **SET DIFFERENCE (EXCEPT / MINUS)**
   This operation returns records present in the first query but not in the second.
   Example: Students enrolled in Course A but not in Course B.

11. Need of Relational Algebra in DBMS

Relational Algebra is a **formal query language** used in DBMS to retrieve and manipulate data from relational databases. It forms the theoretical foundation of SQL.

1. Relational Algebra provides a **mathematical way to describe database queries**. Because it is based on set theory, queries are precise and unambiguous. This helps in understanding how data is actually processed in DBMS.
2. It helps in **query optimization**. DBMS internally converts SQL queries into relational algebra expressions and then optimizes them for efficient execution.
3. Relational Algebra is **procedural**, meaning it explains *how* to get the result step by step. This makes it useful for learning and analyzing query execution.
4. It is important for **database design and learning concepts**. Even though users do not write relational algebra directly, it helps developers and students understand DBMS internals clearly.

12. Types of Joins (Inner, Left, Right, Full)

Joins are used to **combine data from two or more tables** based on a related column. Different joins return different results.

1. **Inner Join**
   An inner join returns only those records that have matching values in both tables. Rows without matching values are excluded. It is the most commonly used join.
2. **Left Join (Left Outer Join)**
   A left join returns all records from the left table and matching records from the right table. If there is no match, NULL values are shown for the right table.
3. **Right Join (Right Outer Join)**
   A right join returns all records from the right table and matching records from the left table. Unmatched rows from the left table show NULL values.
4. **Full Join (Full Outer Join)**
   A full join returns all records from both tables. If there is no match, NULL values are filled where data is missing.

13. Converting Simple ER Model Components into Relational Tables

Converting an ER model into relational tables is called **ER-to-Relational mapping**. It helps in implementing a database design.

1. **Entity to Table Conversion**
   Each entity in the ER model is converted into a table. The attributes of the entity become columns of the table.
2. **Primary Key Selection**
   The key attribute of the entity becomes the primary key of the table. This uniquely identifies each record.
3. **Relationship Mapping**
   For one-to-many relationships, the primary key of the "one" side is added as a foreign key in the "many" side table.
4. **Many-to-Many Relationships**
   Many-to-many relationships are converted into a separate table. This table contains primary keys of both entities as foreign keys.

Example:
Student and Course with "enrolls" relationship → create an Enrollment table.

14. Cross Product and its SQL Representation

Cross Product is a relational algebra operation that **combines every row of one table with every row of another table**.

1. Cross product is also called **Cartesian Product**. If table A has m rows and table B has n rows, the result will have m × n rows.
2. It produces a very large result set. Because of this, it is usually used with selection conditions.
3. Cross product forms the **base for joins**. Joins are implemented by applying conditions on cross product results.
4. In SQL, cross product is represented using **CROSS JOIN** or by listing tables without a join condition.

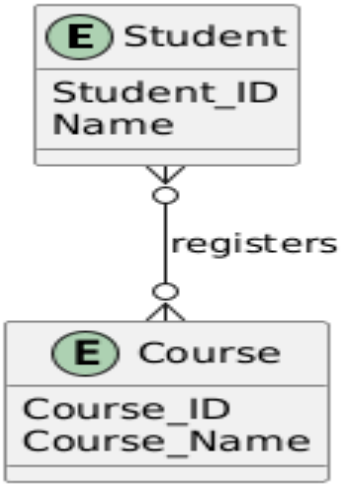
SQL Example:

```
SELECT * FROM Student CROSS JOIN Course;
```

15. ER Diagram for Student–Course Registration System

A Student–Course registration system allows students to register for courses. The main entities are **Student** and **Course**.

1. **Student Entity** stores student details like Student_ID and Name.
2. **Course Entity** stores course details like Course_ID and Course_Name.
3. The relationship **Registers** shows that students can register for courses.
4. This is a **many-to-many relationship**, because one student can register for many courses and one course can have many students.



[10 - marks]

1. Explain the different types of attributes in ER Model

Attributes are **properties or characteristics** that describe an entity in the ER model. Different types of attributes are used to represent data clearly.

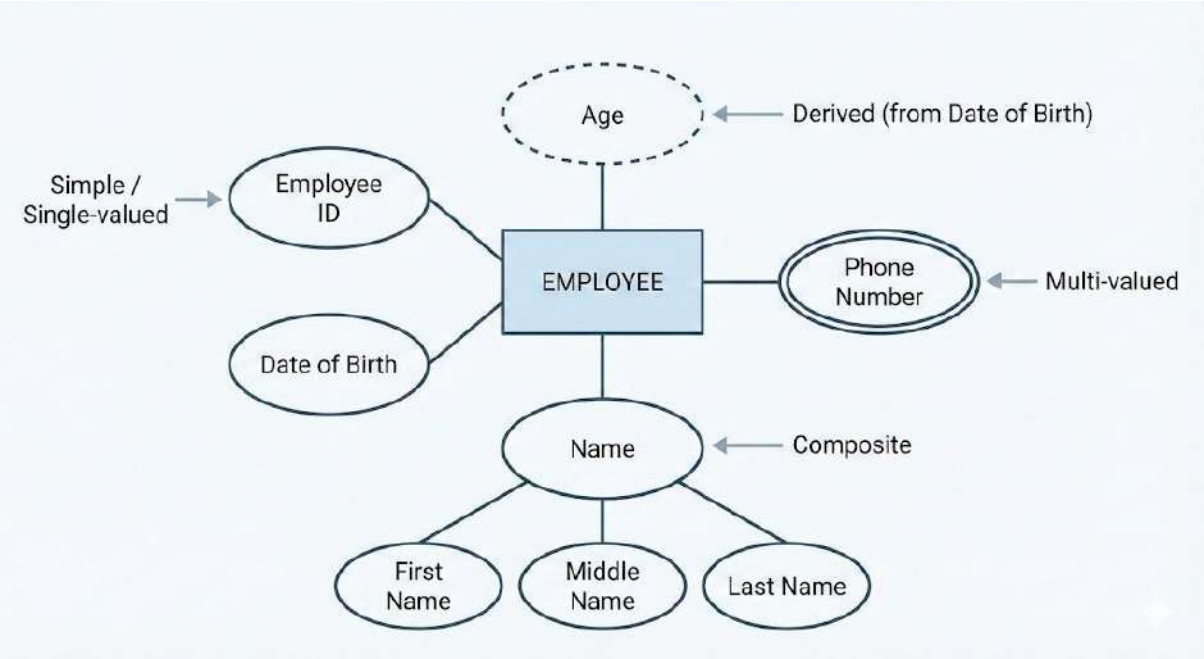
1. **Simple Attribute**
   A simple attribute is an attribute that **cannot be divided into smaller parts**. It stores atomic information and is easy to manage in a database.
   Example: Employee_ID is a simple attribute because it cannot be further split.

2. **Composite Attribute**
   A composite attribute can be **divided into sub-attributes**. It helps in representing structured and meaningful data.
   Example: Name can be divided into First_Name, Middle_Name, and Last_Name.

3. **Single-Valued Attribute**
   A single-valued attribute holds **only one value for each entity instance**. This ensures clarity and avoids confusion in data storage.
   Example: Age of a student is single-valued at a given time.

4. **Multi-Valued Attribute**
   A multi-valued attribute can store **more than one value** for a single entity. It is used when an entity can have multiple values for the same property.
   Example: An employee may have multiple Phone_Numbers.

5. **Derived Attribute**
   A derived attribute is **calculated from other attributes** and does not need to be stored permanently.
   Example: Age can be derived from Date_of_Birth.



2. Compare Generalization and Specialization with examples

Generalization and specialization are **abstraction techniques** used in ER modeling to organize data efficiently.

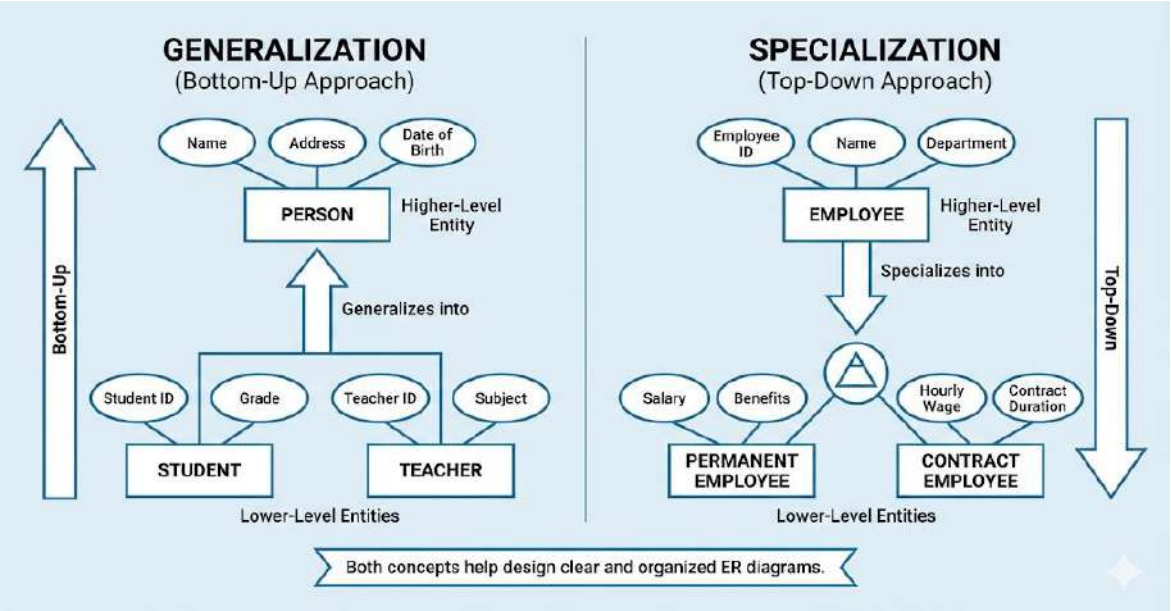
1. **Meaning**
   Generalization is the process of **combining multiple entities into one higher-level entity** based on common attributes.
   Specialization is the process of **dividing one entity into multiple sub-entities** based on distinct characteristics.

2. **Approach Used**
   Generalization follows a **bottom-up approach**, where lower-level entities are merged.
   Specialization follows a **top-down approach**, where a higher-level entity is broken down.

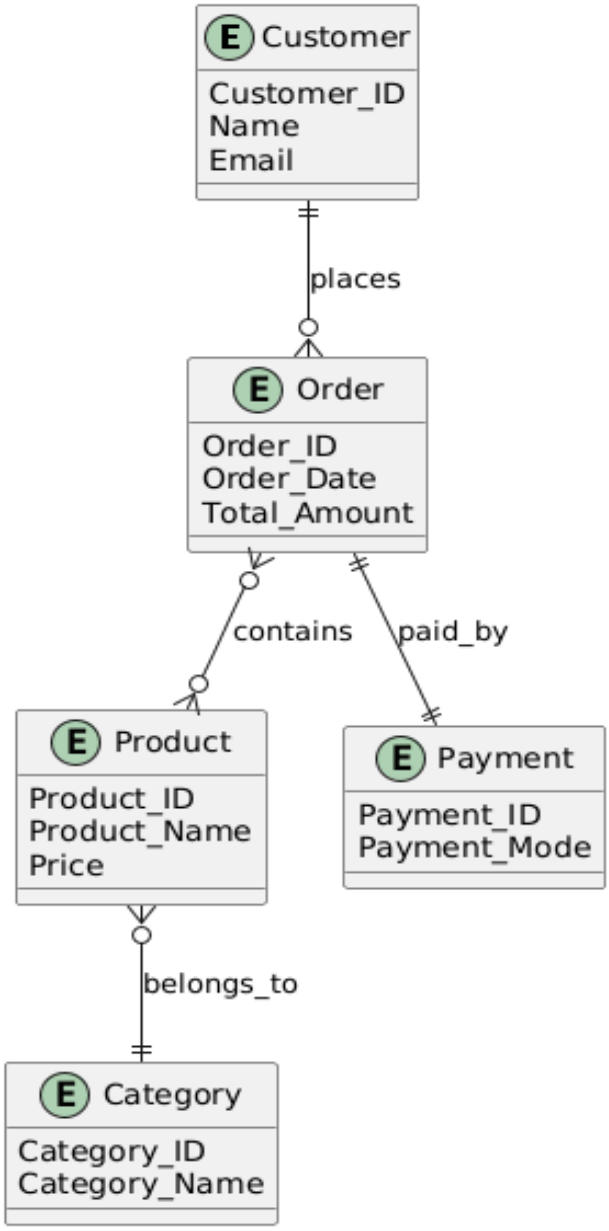
3. **Purpose**
   Generalization helps in **reducing redundancy** by storing common attributes at one place.
   Specialization helps in **representing detailed and specific information** clearly.

4. **Attribute Handling**
   In generalization, common attributes are moved to the generalized entity.
   In specialization, sub-entities inherit attributes from the main entity and may have their own attributes.

5. **Examples**
   Generalization: Student and Teacher can be generalized into a Person entity.
   Specialization: Employee can be specialized into Permanent_Employee and Contract_Employee.



3] Draw a complete ER diagram for an Online Shopping System and explain all entities, attributes, and relationships



Entities and Attributes

1. **Customer**
   Customer is an entity representing users of the online shopping system. Its attributes include Customer_ID, Name, and Email, which uniquely identify and describe each customer.

2. **Order**
   Order represents purchases made by customers. Attributes like Order_ID, Order_Date, and Total_Amount store order-related details.

3. **Product**
   Product represents items available for sale. Attributes such as Product_ID, Product_Name, and Price describe each product.

4. **Category**
   Category groups similar products together. It helps in organizing products for easy browsing.

5. **Payment**
   Payment stores transaction details. Attributes include Payment_ID and Payment_Mode.

Relationships

- A Customer **places** an Order
- An Order **contains** Products
- A Product **belongs to** a Category
- An Order is **paid by** Payment

4. Explain the Relational Data Model in detail

The **Relational Data Model** represents data in the form of **tables (relations)**. Each table consists of rows and columns, making data easy to understand and manage.

1. **Relation**
   A relation is a table that stores data about an entity. Each relation has a unique name and consists of tuples and attributes.

2. **Tuple**
   A tuple is a **single row** in a relation. It represents one record or one real-world entity instance.

3. **Attribute and Domain**
   An attribute is a column in a table. The domain of an attribute is the **set of possible values** that attribute can take.

4. **Degree**
   Degree is the **number of attributes** in a relation. For example, a table with 4 columns has degree 4.

5. **Cardinality**
   Cardinality is the **number of tuples** in a relation. It represents how many records are stored in the table.

6. **Relation Schema**
   Relation schema defines the structure of a relation. It includes the relation name and its attributes, for example:
   STUDENT(Student_ID, Name, Age).

7. **Integrity Rules**
   Integrity rules ensure correctness of data. Entity integrity ensures primary keys are not NULL, and referential integrity ensures valid foreign key references

5. Different Types of Keys in DBMS

Keys are attributes used to **identify records uniquely** and to **maintain relationships** between tables.

1. Super Key

- A **super key** is a set of one or more attributes that can uniquely identify a tuple in a relation.
- It may contain extra attributes that are not necessary for uniqueness.
- Every primary key is a super key, but not every super key is a primary key.
- **Example:** {Student_ID}, {Student_ID, Name} in a Student table.

2. Candidate Key

- A **candidate key** is a **minimal super key**.
- It uniquely identifies records without any redundant attributes.
- A table can have more than one candidate key.
- **Example:** Student_ID and Email_ID.

3. Primary Key

- A **primary key** is one candidate key selected to uniquely identify records.
- It cannot have duplicate or NULL values.
- It ensures **entity integrity**.
- **Example:** Student_ID.

4. Alternate Key

- Alternate keys are candidate keys **not chosen as primary key**.
- They still uniquely identify records.
- **Example:** Email_ID if Student_ID is primary key.

5. Composite Key

- A **composite key** consists of **two or more attributes** together.
- Used when a single attribute is not sufficient.
- **Example:** {Student_ID, Course_ID} in Enrollment table.

6. Foreign Key

- A **foreign key** refers to the primary key of another table.
- It maintains **referential integrity**.
- **Example:** Course_ID in Student table referencing Course table.

6. Relational Algebra Operators (with syntax & examples)

Relational algebra is a **procedural query language** used to retrieve data from relations.

1. Selection (σ)

- Selects rows based on a condition.
- Does not change the structure of the table.
- **Syntax:** σ condition (Relation)
- **Example:** σ Age > 20 (Student)

2. Projection (π)

- Selects specific columns from a table.
- Removes duplicate columns automatically.
- **Syntax:** π attribute_list (Relation)
- **Example:** π Name, Age (Student)

3. Rename (ρ)

- Renames a relation or its attributes.
- Useful in complex queries.
- **Syntax:** ρ NewName (Relation)
- **Example:** ρ S (Student)

4. Set Operations

- Include **Union (∪), Intersection (∩), Set Difference (−)**.
- Relations must be union-compatible.
- **Example:** Student1 ∪ Student2

5. Cross Product (×)

- Combines every tuple of one relation with every tuple of another.
- Produces large result sets.
- **Example:** Student × Course

6. Joins

- Combines related tuples using conditions.
- Types: Inner Join, Left Join, Right Join, Full Join.
- **Example:** Student ⋈ Course

7. Explain different types of Joins with examples, diagrams, and SQL

A **Join** is used to **combine rows from two tables** based on a related column. Joins help in retrieving meaningful data spread across multiple tables.

1. Inner Join

- Inner Join returns **only matching records** from both tables.
- Rows without matching values are not included.
- It is the most commonly used join.
- It gives precise and filtered results.

Example: Students enrolled in courses.

SQL:

```
SELECT *
FROM Student
INNER JOIN Course
ON Student.Course_ID = Course.Course_ID;
```

2. Left Join (Left Outer Join)

- Left Join returns **all records from the left table**.
- Matching records from the right table are included.
- If no match exists, NULL values are shown.
- Useful when left table data is mandatory.

SQL:

```
SELECT *
FROM Student
LEFT JOIN Course
ON Student.Course_ID = Course.Course_ID;
```

3. Right Join (Right Outer Join)

- Right Join returns **all records from the right table**.
- Matching records from the left table are included.
- NULL is shown when there is no match.
- Used less frequently than Left Join.

SQL:

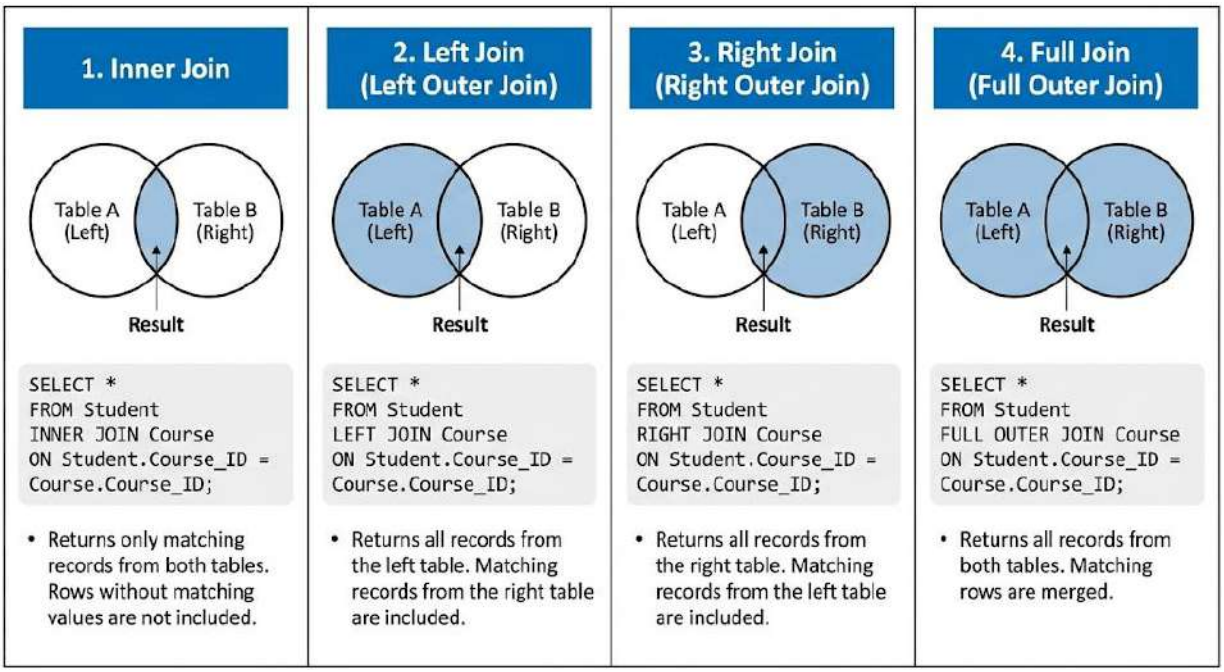
```
SELECT *
FROM Student
RIGHT JOIN Course
ON Student.Course_ID = Course.Course_ID;
```

4. Full Join (Full Outer Join)

- Full Join returns **all records from both tables**.
- Matching rows are merged.
- Non-matching rows contain NULL values.
- Useful for complete data comparison.

SQL:

```
SELECT *
FROM Student
FULL OUTER JOIN Course
ON Student.Course_ID = Course.Course_ID;
```



8. Step-by-step Conversion of ER Model into Relational Schema

Converting an ER model into a relational schema is called **ER-to-Relational Mapping**. It helps in implementing the database.

Step 1: Convert Strong Entities

- Each strong entity becomes a **separate table**.
- All attributes become columns.
- The key attribute becomes the **primary key**.

Example:
Student(Student_ID, Name, Age)

Step 2: Convert Weak Entities

- Weak entities become tables.
- Primary key is a **combination of its partial key and owner's primary key**.
- Owner's key acts as a foreign key.

Step 3: Convert One-to-Many Relationships

- Add the primary key of the "one" side as a **foreign key** in the "many" side table.
- No separate table is needed.

Example:
Department → Employee
Employee(Emp_ID, Name, Dept_ID)

Step 4: Convert Many-to-Many Relationships

- Create a **new table** for the relationship.
- Include primary keys of both entities as foreign keys.
- Combined keys act as the primary key.

Example:
Enrollment(Student_ID, Course_ID)

Step 5: Convert Attributes

- Simple attributes → columns.
- Composite attributes → split into sub-attributes.
- Multi-valued attributes → separate table.
- Derived attributes → usually not stored.

Final Example

ER: Student enrolls in Course
Relational Schema:

Student(Student_ID, Name)

Course(Course_ID, Course_Name)

Enrollment(Student_ID, Course_ID)

9] Solve a case study (e.g., Hospital Management / Banking System / College Database): o Draw an ER diagram o Identify entities, attributes, relationships o Convert the ER diagram into relational tables

Case Study – College Database System

Problem Statement : A college database system stores information about Students, Courses, and Departments.

- A student can enroll in multiple courses.
- A course can be taken by many students.
- Each course belongs to one department.

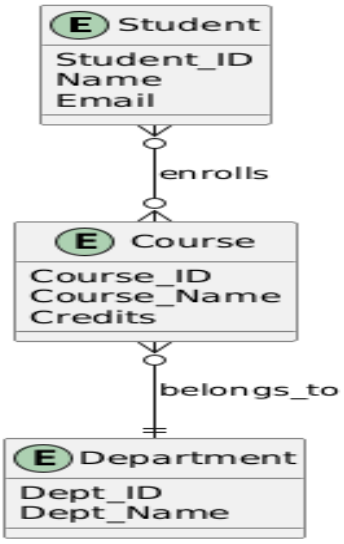
Entities & Attributes

1. Student
    - Student_ID
    - Name
    - Email
2. Course
    - Course_ID
    - Course_Name
    - Credits
3. Department
    - Dept_ID
    - Dept_Name

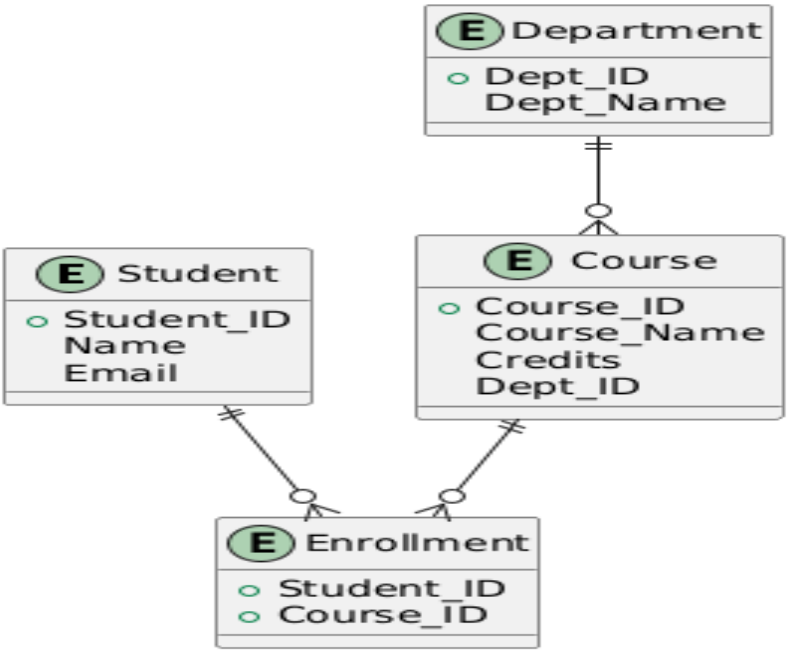
Relationships

- Student enrolls in Course (Many-to-Many)
- Course belongs to Department (Many-to-One)

ERD :-



Relational Schema :-



10. Compare ER Model and Relational Model. Discuss their roles in database design

The ER Model and Relational Model are two important models used at different stages of database design. They serve different purposes but are closely related.

Comparison between ER Model and Relational Model

1. Level of Abstraction
   The ER Model is a high-level conceptual model. It focuses on understanding real-world data requirements.
   The Relational Model is a logical model. It focuses on how data is actually stored in tables.

2. Representation
   The ER Model represents data using entities, attributes, and relationships, usually in graphical form (ER diagrams).
   The Relational Model represents data using tables (relations) consisting of rows and columns.

3. Purpose
   The ER Model is mainly used during the database design phase to communicate with users and understand requirements.
   The Relational Model is used during database implementation in a DBMS.

4. User Understanding
   ER diagrams are easy for non-technical users to understand.
   Relational tables are more suitable for developers and DBMS systems.

Role in Database Design

- ER Model helps in requirement analysis and conceptual design.
- Relational Model helps in logical design and actual database creation.
  Both together ensure a correct and efficient database design.

11. Importance of Relational Algebra with Examples of Operations

Relational Algebra is a formal and procedural query language used in DBMS. It forms the foundation of SQL and helps in understanding how queries are processed internally.

Importance of Relational Algebra

1. Relational Algebra provides a mathematical way to retrieve data from relations. This makes queries precise and unambiguous.
2. It helps in query optimization. DBMS internally converts SQL queries into relational algebra expressions.
3. It explains how data is retrieved step by step, which is useful for learning DBMS concepts.
4. It is essential for understanding database internals and query execution plans.

Relational Algebra Operations

1. Selection (σ)

- Selection retrieves specific rows from a relation based on a condition.
- It does not change the structure of the table, only filters tuples.

Example:
σ Age > 20 (Student)
This selects students whose age is greater than 20.

2. Projection (π)

- Projection retrieves specific columns from a relation.
- It removes unnecessary attributes from the result.

Example:
π Name, Age (Student)
This displays only Name and Age of students.

3. Join (⋈)

- Join combines tuples from two relations based on a common attribute.
- It helps in retrieving related data from multiple tables.

Example:
Student ⋈ Course
This joins Student and Course tables based on a common key.

4. Set Operations

- Set operations include Union (∪), Intersection (∩), and Set Difference (−).
- They work on relations that are union-compatible.

Examples:

- Student_A ∪ Student_B → combines both relations
- Student_A ∩ Student_B → common records
- Student_A − Student_B → records only in Student_A