

1.What is subquery explain with example?

A subquery, also known as an inner query or nested query, is a query embedded within another query. It allows you to retrieve data based on the results of another query. The result of the subquery is used as a condition or filter in the outer query to obtain the final result set.

Subqueries are commonly used when you need to perform a complex query involving multiple tables or when you want to filter data dynamically based on some conditions.

Let's look at an example to illustrate the concept of a subquery:

Consider two tables: "**Customers**" and "**Orders**."

Table: **Customers**

Customer ID	Name	Age
1	John Smith	30
2	Jane Doe	25
3	Bob Johnson	40

Table: **Orders**

OrderID	CustomerID	Product	Quantity
101	1	Laptop	1
102	2	Smartphone	2
103	3	Monitor	3
104	4	Headset	1

Example: Retrieve the names of customers who have placed more than one order.

To achieve this, we can use a subquery to find the customers who have placed more than one order, and then use that result in the main query to fetch their names.

```
```sql
```

```
SELECT Name
FROM Customers
WHERE CustomerID IN (SELECT CustomerID
 FROM Orders
 GROUP BY CustomerID
 HAVING COUNT(OrderID) > 1);
```

Name
<b>John Smith</b>
<b>Jane Doe</b>

In this example, the subquery helps us filter the customer names based on the number of orders they have placed, making the query more efficient and flexible. Subqueries can be used in various scenarios to solve complex querying problems in SQL.

=====

## 2. what is trigger ? with example.

A trigger, in the context of databases, is a special type of stored procedure that automatically executes in response to specific events or actions occurring on a table. These events can be data manipulation operations like **INSERT**, **UPDATE**, **DELETE**, or **TRUNCATE**. Triggers are associated with a particular table and are defined to run either before or after the specified data manipulation operation occurs. They are used to enforce data integrity rules, perform auditing tasks, maintain data consistency, or automate complex actions based on changes in the database.

Let's go through another example to better understand how triggers work:

Consider a table called "**Employees**" with the following structure:

Table: **Employees**

EmployeeID	Name	Department	Salary
1	John	HR	50000
2	Jane	Finance	60000

3	Bob	HR	55000
---	-----	----	-------

Now, let's create a trigger that automatically updates the "**LastModified**" column of the "Employees" table whenever a row is updated.

```
```sql
CREATE TRIGGER update_last_modified
BEFORE UPDATE ON Employees
FOR EACH ROW
BEGIN
  SET NEW.LastModified = CURRENT_TIMESTAMP;
END;
```
```

In this example:

- We created a "**BEFORE**" trigger named "**update\_last\_modified**" that activates before an `
- The "FOR EACH ROW" clause specifies that the trigger should execute for each affected row.
- Inside the trigger body, we use the "SET" statement to update the "**LastModified**" column of the row being updated using the "NEW" keyword.
- We set "**LastModified**" to the current timestamp using "CURRENT\_TIMESTAMP".

**Now, let's see how this trigger works in action:**

```
```sql
-- Update the salary for EmployeeID 1
UPDATE Employees
SET Salary = 52000
WHERE EmployeeID = 1;
```
```

After this update operation, the trigger will be automatically activated, and the "**LastModified**" column for **EmployeeID 1** will be updated with the current timestamp.

The resulting table will be:

Table: **Employees**

| EmployeeID | Name | Department | Salary | Last modified                  |
|------------|------|------------|--------|--------------------------------|
| 1          | John | HR         | 52000  | <b>2023-07-22<br/>12:34:56</b> |
| 2          | Jane | Finance    | 60000  | <b>(unchanged)</b>             |
| 3          | Bob  | HR         | 55000  | <b>(unchanged)</b>             |

In this example, the trigger automatically keeps track of the last modification timestamp, allowing us to know when each employee's record was last updated.

Triggers can be quite powerful, but they should be used thoughtfully and sparingly. Poorly designed triggers or excessive usage can lead to performance issues and maintainability challenges. Therefore, it's essential to understand the impact of triggers on the database and use them judiciously to achieve the desired functionality while maintaining database performance and integrity.

=====

### **3. Integrity Constraints with example**

Integrity constraints in databases are rules that help maintain the accuracy, consistency, and validity of data in tables. They define certain conditions that data must satisfy to ensure the data remains reliable and meaningful. There are several types of integrity constraints, including:

#### **1. Primary Key Constraint:-**

Ensures that a column or a combination of columns uniquely identifies each row in the table. It prevents duplicate and NULL values in the specified column(s).

#### **2. Foreign Key Constraint:-**

Enforces referential integrity between two tables. It ensures that values in a column of one table match the values in another table's primary key column.

#### **3.Unique Constraint:-**

Ensures that the values in a specified column (or combination of columns) are unique and not duplicated in the table.

#### **4. Check Constraint:-**

Allows you to define custom conditions to restrict the values that can be inserted or updated in a column.

Let's see examples of each integrity constraint:

Consider two tables: "**Students**" and "**Courses.**"

Table: **Students**

| StudentID | Name | Age | Major        |
|-----------|------|-----|--------------|
| 1         | John | 20  | Comp.science |
| 2         | Jane | 22  | Chemistry    |
| 3         | Bob  | 19  | Physics      |

Table: **Courses**

| CourseID | CourseName   | Credits |
|----------|--------------|---------|
| 101      | Math         | 3       |
| 102      | Comp.science | 4       |
| 103      | physics      | 3       |

### 1. Primary Key Constraint:-

```
```sql
```

```
CREATE TABLE Students (  
  StudentID INT PRIMARY KEY,  
  Name VARCHAR(50),  
  Age INT,  
  Major VARCHAR(50)  
);  
...
```

In this example, the "**StudentID**" column is defined as the primary key, ensuring that each student has a unique identifier.

2. Foreign Key Constraint:-

```
```sql
```

```
CREATE TABLE Courses (
 CourseID INT PRIMARY KEY,
 CourseName VARCHAR(50),
 Credits INT
);
```

```
CREATE TABLE Students (
 StudentID INT PRIMARY KEY,
 Name VARCHAR(50),
 Age INT,
 Major VARCHAR(50),
 CourseEnrolled INT,
 FOREIGN KEY (CourseEnrolled) REFERENCES Courses(CourseID)
);
```
```

In this example, the "CourseEnrolled" column in the "Students" table is a foreign key referencing the "CourseID" column in the "Courses" table, ensuring that the courses enrolled by students exist in the "Courses" table.

3. Unique Constraint:-

```
```sql
```

```
CREATE TABLE Students (
 StudentID INT PRIMARY KEY,
 Name VARCHAR(50),
 Age INT,
 Major VARCHAR(50),
 Email VARCHAR(100) UNIQUE
```

);

...

In this example, the "Email" column is defined as UNIQUE, ensuring that each email address can only be associated with one student.

#### 4. Check Constraint:-

```
```sql
```

```
CREATE TABLE Students (
```

```
  StudentID INT PRIMARY KEY,
```

```
  Name VARCHAR(50),
```

```
  Age INT,
```

```
  Major VARCHAR(50),
```

```
  GPA FLOAT CHECK (GPA >= 0 AND GPA <= 4.0)
```

```
);
```

```
...
```

In this example, the CHECK constraint ensures that the "GPA" column only allows values between 0 and 4.0, preventing invalid GPAs from being inserted.

Integrity constraints help maintain the quality and reliability of data in databases, and they play a crucial role in defining the structure and relationships between tables in a database schema.

=====

4.What is Normalisation..???

Normalization is a database design technique used to organize data in a relational database efficiently, reduce redundancy, and ensure data integrity. The primary goal of normalization is to eliminate data anomalies, such as update, insertion, and deletion anomalies, by breaking down large, complex tables into smaller, well-structured ones. It involves applying a series of rules, called normal forms, to ensure that the database schema is logically structured and free from data duplication and inconsistencies.

There are different normal forms, from the first normal form (1NF) to the higher normal forms (2NF, 3NF, BCNF, etc.), each building upon the previous ones. The higher the normal form achieved, the better the database design in terms of data organization and maintenance.

Here's an overview of the first three normal forms:

1. First Normal Form (1NF):-

The first normal form requires that the table has a primary key, and each column contains only atomic (indivisible) values. It eliminates repeating groups and ensures that each cell in the table holds only a single value.

Example:

Consider the following table, which is not in 1NF:

Table: **Students**

Student ID	Name	Courses
1	John	Math, Physics, Chemistry
2	Jane	Computer Science

To convert it to 1NF, we split the repeating group into separate rows:

Table: **Students**

StudentID	Name
1	John
2	Jane

Table: **Courses**

CourseID	CourseName
101	Math
102	Physics
103	Chemistry
104	Comp.Science

2. Second Normal Form (2NF):-

The second normal form requires that the table is in 1NF and there are no partial dependencies. A partial dependency exists when a non-key attribute depends on only a part of the composite primary key.

Example:

Consider the following table:

Table: **OrderDetails**

OrderID	ProductID	ProductName	Quantity
101	1	Laptop	2
101	2	Smartphone	1
102	1	Laptop	3

To convert it to 2NF, we split it into two tables, one for orders and another for products:

Table: **Orders**

OrderID
101
102

Table: **Products**

Product ID	ProductName
1	Laptop
2	SmartPhone

Table: **OrderDetails**

OrderID	ProductID	Quantity
101	1	2
101	2	1

102	1	3
-----	---	---

3. Third Normal Form (3NF):-

The third normal form requires that the table is in 2NF, and there are no transitive dependencies. A transitive dependency exists when a non-key attribute depends on another non-key attribute.

Example:

Consider the following table:

Table: **Employees**

EmployeeID	Department	DepartmentManager
1	HR	John
2	Finance	Jane
3	HR	Bob

To convert it to 3NF, we split it into two tables, one for employees and another for departments:

Table: **Employees**

EmployeeID	DepartmentID
1	1
2	2
3	1

Table: **Departments**

DepartmentID	DepartmentName	DepartmentManager
1	HR	John
2	Finance	Jane

Normalization helps create a well-organized database schema that minimizes data duplication and ensures data integrity. However, achieving higher normal forms can sometimes result in increased complexity and join operations, so it's essential to strike a balance between normalization and performance for specific use cases.

=====

5. Explain about 1NF 2NF 3NF..?

1. First Normal Form (1NF):-

The first normal form requires that a table has a primary key and that all attributes (columns) contain only **atomic** (indivisible) values. It eliminates repeating groups and ensures that each cell in the table holds only a single value.

For a table to be in 1NF, it must meet the following criteria:

- Each column contains only atomic values (no multivalued attributes or arrays).

- Each row in the table has a unique identifier, known as the primary key.

Example:

Consider the following table, which is not in 1NF:

Table: Students

StudentID	Name	Courses
1	John	Math, Physics, Chemistry
2	Jane	Computer Science

To convert it to 1NF, we split the repeating group into separate rows:

Table: Students

StudentID	Name
1	John
2	Jane

Table: Courses

CourseID	CourseName
101	Math
102	Physics
103	Chemistry
104	Comp Science

2. Second Normal Form (2NF):-

The second normal form requires that a table is in 1NF and that there are no partial dependencies. A partial dependency exists when a non-key attribute depends on only a part of the composite primary key.

For a table to be in 2NF, it must meet the following criteria:

- It is in 1NF.
- All non-key attributes depend on the entire primary key.

Example:

Consider the following table:

Table: **OrderDetails**

OrderID	ProductID	ProductName	Quantity
101	1	Laptop	2
101	2	Smartphone	1
102	1	laptop	3

To convert it to 2NF, we split it into two tables, one for orders and another for products:

Table: **Orders**

OrderID
101
102

Table: **Products**

ProductID	ProductName
1	Laptop
2	Smartphone

Table: **OrderDetails**

OrderID	ProductID	Quantity
101	1	2
101	2	1
102	1	3

3. Third Normal Form (3NF):-

The third normal form requires that a table is in 2NF and that there are no transitive dependencies. A transitive dependency exists when a non-key attribute depends on another non-key attribute.

For a table to be in 3NF, it must meet the following criteria:

- It is in 2NF.
- All non-key attributes depend only on the primary key and not on other non-key attributes.

Example:

Consider the following table:

Table: Employees

EmployeeID	Department	DepartmentManager
1	HR	John
2	Finance	Jane
3	HR	Bob

To convert it to 3NF, we split it into two tables, one for employees and another for departments:

Table: **Employees**

EmployeeID	DepartmentID
1	1
2	2
3	1

Table: **Departments**

DepartmentID	DepartmentName	DepartmentManager
1	HR	John
2	Finance	Jane

Normalization helps create a well-organized database schema that minimizes data duplication and ensures data integrity. By applying these normal forms, we can reduce data anomalies and improve the overall quality of the database design.

=====

6.Explain Boyce-Codd form or BCNF(3.5NF)..???

Boyce-Codd Normal Form (BCNF) is a higher level of database normalization that ensures that a table is free from certain types of anomalies and redundancies. It is an extension of the Third Normal Form (3NF) and is named after Raymond F. Boyce and Edgar F. Codd, who introduced the concept in the 1970s.

BCNF addresses the issue of functional dependencies in a table and eliminates the possibility of having non-trivial functional dependencies on non-key attributes. In simpler terms, it ensures that every determinant (attribute that uniquely determines other attributes) in a table is a candidate key, and no non-key attribute is dependent on any proper subset of a candidate key.

To achieve BCNF, a table must satisfy the following conditions:

1. It must be in 3NF.
2. For every non-trivial functional dependency $X \rightarrow Y$ (where X and Y are sets of attributes) in the table, X must be a superkey.

A non-trivial functional dependency is one where Y is not a subset of X, meaning Y is not already uniquely determined by X. A superkey is a set of attributes that can uniquely identify each row in the table.

Let's consider an example to understand BCNF:

Consider the following table, which is in 3NF:

Table: **EmployeeProjects**

EmployeeID	ProjectID	EmployeeName	ProjectName	ProjectManager
1	101	John	Project-A	Mary
2	102	Jane	Project-B	Bob
3	101	Alice	Project-A	Mary

In this table, there is a functional dependency $ProjectID \rightarrow ProjectName$, which means that the ProjectID uniquely determines the ProjectName. However, the ProjectID is not a candidate key; instead, the combination of EmployeeID and ProjectID forms a candidate key. This violates BCNF.

To bring the table to BCNF, we decompose it into two separate tables:

Table: **Employees**

EmployeeID	EmployeeName
1	John
2	Jane

3	Alice
---	-------

Table: Projects

ProjectID	ProjectName	ProjectManager
101	Project-A	Mary
102	ProjectB	Bob

Now, each table satisfies BCNF, and the redundancy and anomalies have been eliminated.

BCNF is considered an essential level of normalization in database design, but it may not always be practical or necessary for all scenarios. Achieving BCNF could lead to increased data fragmentation, which might affect query performance. Hence, database designers should strike a balance between normalization levels and the specific needs and performance requirements of the application or system.

=====

7.Functional dependency..??

Functional dependency is a fundamental concept in the theory of database normalization. It describes the relationship between attributes (columns) in a database table, where the value of one attribute uniquely determines the value of another attribute. In other words, if a functional dependency exists between attributes A and B, knowing the value of A allows you to determine the value of B.

Functional dependencies are denoted as $X \rightarrow Y$, where X and Y are sets of attributes. This notation indicates that the values of attributes in X uniquely determine the values of attributes in Y.

Let's explore some examples of functional dependencies:

1. Example 1:-

Consider a table "**Employees**" with the following attributes:

EmployeeID	Name	Department	Salary
1	John	HR	50000
2	Jane	Finance	60000
3	Bob	HR	55000

In this table, we can say that "**EmployeeID → Name**" because knowing the value of "**EmployeeID**" uniquely determines the "**Name**" of the employee. Similarly, "**EmployeeID → Department**" and "**EmployeeID → Salary**" are also functional dependencies.

2. Example 2:-

Consider a table "Customers" with the following attributes:

CustomerID	Name	Email	Phone
1	John	john@example.com	555-123-4567
2	Jane	jane@example.com	555-987-6543
3	Bob	bob@example.com	555-111-2222

In this table, "**Email → Name**" is a functional dependency because knowing the "**Email**" uniquely determines the "**Name**" of the customer. Similarly, "**Phone → Name**" and "**Email → Phone**" are also functional dependencies.

Functional dependencies are crucial in the process of database normalization, as they help identify the key attributes and determine the appropriate normalization levels. In normalization, we aim to minimize data redundancy and eliminate update anomalies by decomposing tables based on functional dependencies and achieving higher normal forms like First Normal Form (**1NF**), Second Normal Form (**2NF**), Third Normal Form (**3NF**), and so on

=====

Unit -5

1.What Is Transaction management..???

UNIT - IV

Transaction Management:-

A transaction is an execution of a user program and is seen by the DBMS as a series or list, of action, i.e. the actions can be executed by a transaction includes the reading and writing of database.

* ACID properties:-

These are four important properties of transaction that a DBMS must ensure to maintain data in correct access of database and recovery from system failure in a DBMS.

These are four properties of transaction

⇒ Atomicity ⇒ consistency

⇒ Isolation ⇒ durability

① Atomicity:-

A transaction is atomic. Either all operations in the transaction have to be performed or none should be performed.

② Consistency:-

Transaction preserve database consistency. i.e. A transaction transforms a consistent state of the database into another without necessarily preserving consistency at all intermediate point.

③ Isolation:-

Transactions are isolated from one another. i.e. A transaction's updates are concealed from all others until it commits (or roll back).

④ Durability:-

once a transaction commits, its updates survive in the database even if there is subsequent system crash.

Transactions and Schedules:

A transaction is seen by the DBMS as a series or list of actions. we therefore establish a simple transaction model named as transaction states.

Transaction state:-

A transaction must be in one of the following state.

Active state:-

This is initial state of a transaction, the transaction stays in this state while it is starting execution.

partially committed state:-

This transaction state occurs after the final statement of the transaction has been executed.

Failed state:-

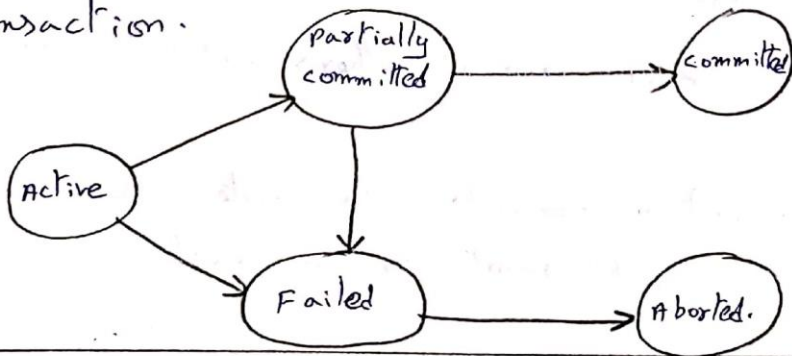
This transaction state occurs after the discovery that normal execution can no longer proceed.

Aborted state:-

This transaction state occurs after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

committed state:-

This transaction state occurs after the successful completion of the transaction.



2. Define Acid Properties..??

ACID properties are a set of four essential characteristics that ensure the reliability and consistency of transactions in a database management system. These properties are essential for maintaining data integrity and ensuring that database transactions are executed correctly, even in the presence of failures or concurrent access by multiple users. The acronym ACID stands for:

1. Atomicity:-

Atomicity ensures that a transaction is treated as a single, indivisible unit of work. It means that either all the operations within the transaction are successfully executed, or none of them are. If any part of the transaction fails, all changes made by the transaction are rolled back, and the database returns to its previous state. Atomicity guarantees that the database remains consistent even in the event of system failures or errors.

2. Consistency:

Consistency ensures that a transaction takes the database from one consistent state to another consistent state. It means that the database must satisfy a set of predefined rules and constraints before and after the transaction execution. In other words, a transaction must not violate any data integrity rules or constraints, ensuring that the database remains in a valid and meaningful state throughout the transaction process.

3. Isolation:

Isolation ensures that each transaction is executed in isolation from other transactions. Even when multiple transactions are running concurrently, their intermediate states should not be visible to other transactions until they are committed. Isolation prevents interference or conflicts between concurrent transactions, helping to maintain data consistency and prevent data corruption.

4. Durability:

Durability ensures that once a transaction is committed, its changes become permanent and survive any subsequent failures, such as system crashes. The changes are stored safely in the database, and they can be retrieved even if the system encounters a crash or restart. Durability guarantees that the effects of a committed transaction persist in the database and are not lost due to hardware or software failures.

The ACID properties are crucial in ensuring that database transactions are reliable and robust, especially in environments with concurrent access by multiple users. By adhering to these

properties, a database management system can ensure data integrity, prevent data corruption, and provide a high level of confidence in the correctness of data operations. However, it's worth noting that maintaining strict ACID properties might sometimes impact performance, so database systems may provide different transaction isolation levels to strike a balance between consistency and performance based on specific application requirements.

3.What is mean By Dead lock..???

Deadlock is a state in which two or more processes (or transactions) are unable to proceed further because each process is waiting for a resource that is held by another process. This results in a circular waiting pattern, and none of the processes can complete their tasks. Deadlocks can occur in various computer systems, including operating systems, databases, and distributed systems.

Let's understand deadlock with an example involving two processes and two resources:

1. Processes: P1 and P2

2. Resources: R1 and R2

Scenario:

1. Process P1 acquires Resource R1.
2. Process P2 acquires Resource R2.
3. Process P1 requests Resource R2, but it is currently held by Process P2, so P1 must wait.
4. Similarly, Process P2 requests Resource R1, but it is currently held by Process P1, so P2 must wait.

Now, both processes are waiting for resources that are held by each other, creating a circular waiting pattern:

Process P1 -> Waits for Resource R2 -> Held by P2

Process P2 -> Waits for Resource R1 -> Held by P1

As a result, neither Process P1 nor Process P2 can proceed, and they are stuck in a deadlock.

Deadlocks can have severe implications on system performance and resource utilization. If not properly managed, deadlocks can lead to system crashes, data corruption, and a complete halt of all operations. Therefore, it is crucial to detect and resolve deadlocks to ensure the smooth functioning of computer systems.

Deadlock prevention and deadlock avoidance are two common strategies to handle deadlocks:

1. Deadlock Prevention:

This approach focuses on structuring the system in a way that deadlocks cannot occur. It involves ensuring that at least one of the four necessary conditions for deadlock (mutual exclusion, hold and wait, no preemption, circular wait) is not satisfied. However, deadlock prevention may lead to resource underutilization or decreased system performance.

2. Deadlock Avoidance:

This strategy involves dynamically analyzing the resource allocation to determine if granting a resource request will lead to a deadlock. If granting the request would result in a deadlock, the system will delay or deny the request until it is safe to proceed. Deadlock avoidance is more flexible than prevention but requires sophisticated algorithms for resource allocation.

In addition to prevention and avoidance, deadlock detection and deadlock recovery strategies are employed in some systems to identify and resolve deadlocks after they occur.

Handling deadlocks effectively is crucial in multi-user and concurrent systems to ensure the stability and reliability of computer systems and databases.

-----**All The Best**-----

Made by-Raj

Source -Vani madam & web

1. What do you mean by DBMS? Explain characteristics, advantages and disadvantages of DBMS?

Database management system simply refers to software which organize and manages a set of data in a database. It is a system which served as an interface in between the end users of data and database. Database management system enable users to perform a variety of functions like creation of database, storing of data, data updation, table creation in database and lots more via the interface provided on software application. Various popular DBMS software are Oracle, MySQL, FoxPro, SQLite and Microsoft Access. Database is an organized collection of inter-related data, stored and accessed electronically from computer devices. It is designed and build for including data related to certain task. Data is organized in the form of tables, report, views and schema in database.

What is DBMS?

DBMS is a crucial technology which allow users for storing as well as retrieving data with utmost efficiency and appropriate security measures. Users can create the database as per their own requirements. This system also defines the rules for validation and manipulation of data for security of database. DBMS accepts the request for data from user through the application and directs the operating system to provide the specific data.

Advantages of Database Management System

Avoids Database Redundancy

DBMS is quite effective in controlling redundancy of data in record system. Data redundancy involves maintaining several copies of same data. It records all data in one single file of database which saves our storage space and enhance update as well as retrieval speed. Also, this system uses ACID (Atomicity, consistency, isolation and durability) and normalization properties.

Data Abstraction

Data abstraction is one of the key characteristics of DBMS. It involves hiding the complexities of data from basic users. DBMS hides unnecessary data from end users and just show them the relevant set of data required for completion of their task. This way, it makes easy for user to work by reducing the overhead of hidden complexities.

Improved Data Sharing

DBMS has improved the sharing of data by end-users via developing a friendly environment. Data can be easily shared among multiple users within the organization by authorized users. It provides access to more and better-managed data which enables users in responding quickly to environmental changes.

Enhanced Security Of Data

There is a greater risk of data security breaches in case when they are multiple users to access the data. Every organization incurs a large amount of money, time and efforts for ensuring that their data is handled properly. Database management system provides such a framework where there is a better enforcement of data privacy and security policies.

Minimize Data Inconsistency

DBMS avoids the inconsistency of data by controlling the data redundancy. There are more chances of data inconsistency when distinct version of same data appears at different places. When data is displayed only once in system records, it leads to better consistency of data due to which updated values are available immediately to all users.

Better Data Integration

This system ensures better integration of data by providing a wider access to it. Users get a clearer view of big picture as well as an integrated view of organizational operations. This way it can be easily seen how the actions in one business segment influences the actions of another segment.

Improved Data Access

DBMS provides an easy and very efficient data access to users by storing data in well sorted manner. Users can directly retrieve the data as per their requirements without a need to worry about excess or redundant records. All this saves a lot of time.

Enhanced Decision Making

It has enhanced the decision making of organization by offering a wider access to better managed set of data. User are able to produce a good-quality of information when the underlying data is managed efficiently. In order to promote the accuracy, timeliness and validity of data, quality of data is served as a comprehensive approach. Whereas data quality if not guaranteed by DBMS but it provides such framework that facilitates the data quality initiatives.

Improved End-User Productivity

DBMS enables users to make quick and informed decisions in presence of accurate data. When data is combined with tools, it leads to transformation of raw data into a usable set of information. The difference in between the success and failure in global economy is dependent upon the quality of decisions.

Disadvantages of Database management system

Cost Of Hardware And Software

DBMS require high speed processors with large memory size in order to run in an efficient manner. Such hardware components require huge cost as they are quite expensive. In addition to this, software's which are run on such hardware is expensive too.

Data Conversion Cost

Whenever a company adopts a database system from old computer file-based system, it is required to convert the data files into database files. All such processes of converting data files into database are time consuming and costly.

Maintaince Cost

Database management system calls for regular maintaince of its system in order to work efficiently without any interruption. There are large expenses associated with maintaince of DBMS once it is made.

Management Complexity

DMBS is not a child game to run for organization as it involves a lot of complexities. At first, company must have good staff team with better management capabilities. Also, many times it is a difficult task for business to decide from where to pick data and where to save it.

Cost Of Staff Training

DBMS are complex system which require trained staff personnel's for running smoothly. Organizations need to incur high cost for running training programmes in order to make workers capable of running database management system.

Hight Impact Of Failure

The impact of failure is high in database management as compared to file base system. It is because in DBMS whole data related to organization is stored in a single database and in case if database is damaged due to electric failure or data corruption, then whole data may be lost.

Characteristics of DBMS

Some well-known characteristics are present in the DBMS (Database Management System). These are explained below.

1. Real World Entity

The reality of DBMS (Database Management System) is one of the most important and easily understandable characteristics. The DBMS (Database Management System) is developed in such a way that it can manage huge business organizations and store their business data with security.

The Database can store information such as the cost of vegetables, milk, bread, etc. In DBMS (Database Management System), the entities look like real-world entities.

For example, if we want to create a student database, we need some entity. Any student stores their data.

In the Database, then, it should be the real-world entity. The most commonly used properties in the student database are name, age, gender, roll number, etc.

2. Self-explaining nature

In DBMS (Database Management System), the Database contains another database, and another database also contains metadata.

Here the term metadata means data about data.

For example, in a school database, the total number of rows and the table's name are examples of metadata.

So the self-explaining nature means the Database explains all the information automatically itself. This is because, in the Database, all the data are stored in a structured format.

3. Atomicity of Operations (Transactions)

Here, atomicity means either the operation should be performed or not performed. i.e., it should complete the operation on 0% or 100%.

Here DBMS (Database Management System) provides atomicity as a characteristic. This is the most important and useful characteristic of the DBMS (Database Management System). You can completely understand the atomicity with the help of the below example.

For example, every bank has its own Database, and the Database contains all the information about its customers. Let transaction is the most common atomic operation of the bank. If Sona wants to transfer 1000 rupees to the Archita account, it is possible with the help of the atomicity feature of the Database. If there is a problem in the Archita account, if there is a problem in the atomicity of the Database, then the money will be deducted from the Sona account but not credited to the Archita account.

The Database has the feature of atomicity then; such transactions have not occurred at all, and if the transaction fails, then the money will automatically return to the sender account.

Basically, for a successful transaction, the total operation depends on the Database. If the Database works perfectly, the transaction will be successful, and if the Database fails, the whole banking server will be down.

4. Concurrent Access without Anomalies

Here the term anomalies mean multiuser can access the Database and fetch the information without any problem.

For a better understanding, let's take the example of a bank again. Let Sonu give his ATM card to his sister Archita and tell her to withdraw 5000 from the ATM. At the same time, Sonu transferred 2000 rupees to his brother Monu. At the same time, both operations perform successfully. Initially, Sonu had 10000 rupees in his bank account. After both transactions, i.e., transfer and withdraw, when Sonu checks his bank balance, it shows 3000

rupees. This error-free updation of bank balance is possible with the help of the concurrent feature of the Database.

Thus here we see that concurrent is a great feature of the Database.

5. Stores Any Kind of Structured Data

The Database has the ability to store the data in a structured format.

In most of the websites, we see that only student database examples are given for a better understanding, but the important fact is that the Database has the ability to store an unlimited amount of data.

DBMS has the ability to store any type of data that exists in the real world, and these data are structured way. It is another type of very important characteristic of DBMS.

6. Integrity

Here the term integrity means the data should be correct and consistent in nature. Let's understand this by taking an example.

Let's say there is a bank named ABC bank, and ABC bank has its own Database for the storage of its customer data. If we try to enter the account details of ABC bank and the account details are not available in the bank, then the Database gives the incorrect output. However, if a customer changes their address but the new address is not updated in the Database, it is called data inconsistency.

So the data available in the Database should be correct as well as consistent.

If someone's account has zero balance and later the customer deposits 6000 rupees in his account, if the new account balance is not updated in the Database, it creates a problem for the customer.

7. Ease of Access (The DBMS Queries)

The file and folder system was used to store the data before the DBMS came to the market. Searching for the student's name was a very difficult task at that time. This is because every search operation is done manually in the file and folder system. But when DBMS comes into the market, it is very easy to access the Database.

In DBMS, we can search any kind of stored data by applying a simple search operation query. It is so much faster than manual searching.

In DBMS, there is a CRUD operation (here CRUD means Create, Read, Update & Delete) by which we can implement all the types of query in the Database.

8. SQL and No-SQL Databases

There are two types of databases (not DBMS): SQL and No-SQL.

The SQL databases store the data in the form of Tables, i.e., rows and columns. The No-SQL databases can store data in any form other than a table. For instance: the very popular MongoDB stores the data in the form of JSON (JavaScript Object Notation).

The availability of SQL and No-SQL databases allows us to choose the method of storing the data as well.

There should not be any debate between SQL and No-SQL databases. The one that we require for a particular project is better for that project, while the other might be better for some other use.

This is a characteristic of DBMS because DBMS allows us to perform operations on both kinds of databases. So, we can run queries and operations on SQL as well as No-SQL databases.

9. ACID Properties

The DBMS follows certain properties to maintain consistency in the Database. These properties are usually termed ACID Properties.

However, we have already talked about some of these properties, but it is very important to mention the ACID properties as a whole.

ACID stands for Atomicity, Consistency, Isolation, and Durability.

We have already talked about atomicity and consistency. Atomicity means the transaction should either be 0% or 100% completed, and consistency means that the change in data should be reflected everywhere in a database.

Isolation means that multiple transactions can occur independently without the interference of some other transactions.

Durability means that the chances of a successful atomic transaction, i.e., a transaction that has been 100% completed, should reflect in the Database.

10. Security

The Database should be accessible to the users in a limited way.

The access to make changes to a database by the user should be limited, and the users must not be given complete access to the entire Database.

Unauthorized users should not be allowed to access the Database.

Authentication: The DBMS has authentication for various users that directly refers to the limit to which the user can access the Database. Authentication means the process of logging in of the user only with the rights that he/she has been authorized to. For instance, in any organization, the admin has access to make changes to the Database of the organization as some new employee might have joined the organization or someone might have left it. However, the employees have access only to their personal profiles and can make changes to them only. They cannot access the Database of any other employee or the organization as a whole.

2.Explain three-schema architecture with a neat diagram?

It shows the DBMS architecture.

Mapping is used to transform the request and response between various database levels of architecture.

Mapping is not good for small DBMS because it takes more time.

In External / Conceptual mapping, it is necessary to transform the request from external level to conceptual schema.

In Conceptual / Internal mapping, DBMS transform the request from the conceptual to internal level.

Objectives of Three schema Architecture

The main objective of three level architecture is to enable multiple users to access the same data with a personalized view while storing the underlying data only once. Thus it separates the user's view from the physical structure of the database. This separation is desirable for the following reasons:

Different users need different views of the same data.

The approach in which a particular user needs to see the data may change over time.

The users of the database should not worry about the physical implementation and internal workings of the database such as data compression and encryption techniques, hashing, optimization of the internal structures etc.

All users should be able to access the same data according to their requirements.

DBA should be able to change the conceptual structure of the database without affecting the user's

Internal structure of the database should be unaffected by changes to physical aspects of the storage.

1. Internal Level

DBMS Three schema Architecture

The internal level has an internal schema which describes the physical storage structure of the database.

The internal schema is also known as a physical schema.

It uses the physical data model. It is used to define that how the data will be stored in a block.

The physical level is used to describe complex low-level data structures in detail.

The internal level is generally is concerned with the following activities:

Storage space allocations.

For Example: B-Trees, Hashing etc.

Access paths.

For Example: Specification of primary and secondary keys, indexes, pointers and sequencing.

Data compression and encryption techniques.

Optimization of internal structures.

Representation of stored fields.

2. Conceptual Level

DBMS Three schema Architecture

The conceptual schema describes the design of a database at the conceptual level. Conceptual level is also known as logical level.

The conceptual schema describes the structure of the whole database.

The conceptual level describes what data are to be stored in the database and also describes what relationship exists among those data.

In the conceptual level, internal details such as an implementation of the data structure are hidden.

Programmers and database administrators work at this level.

3. External Level

DBMS Three schema Architecture

At the external level, a database contains several schemas that sometimes called as subschema. The subschema is used to describe the different view of the database.

An external schema is also known as view schema.

Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.

The view schema describes the end user interaction with database systems.

Mapping between Views

The three levels of DBMS architecture don't exist independently of each other. There must be correspondence between the three levels i.e. how they actually correspond with each other. DBMS is responsible for correspondence between the three types of schema. This correspondence is called Mapping.

There are basically two types of mapping in the database architecture:

Conceptual/ Internal Mapping

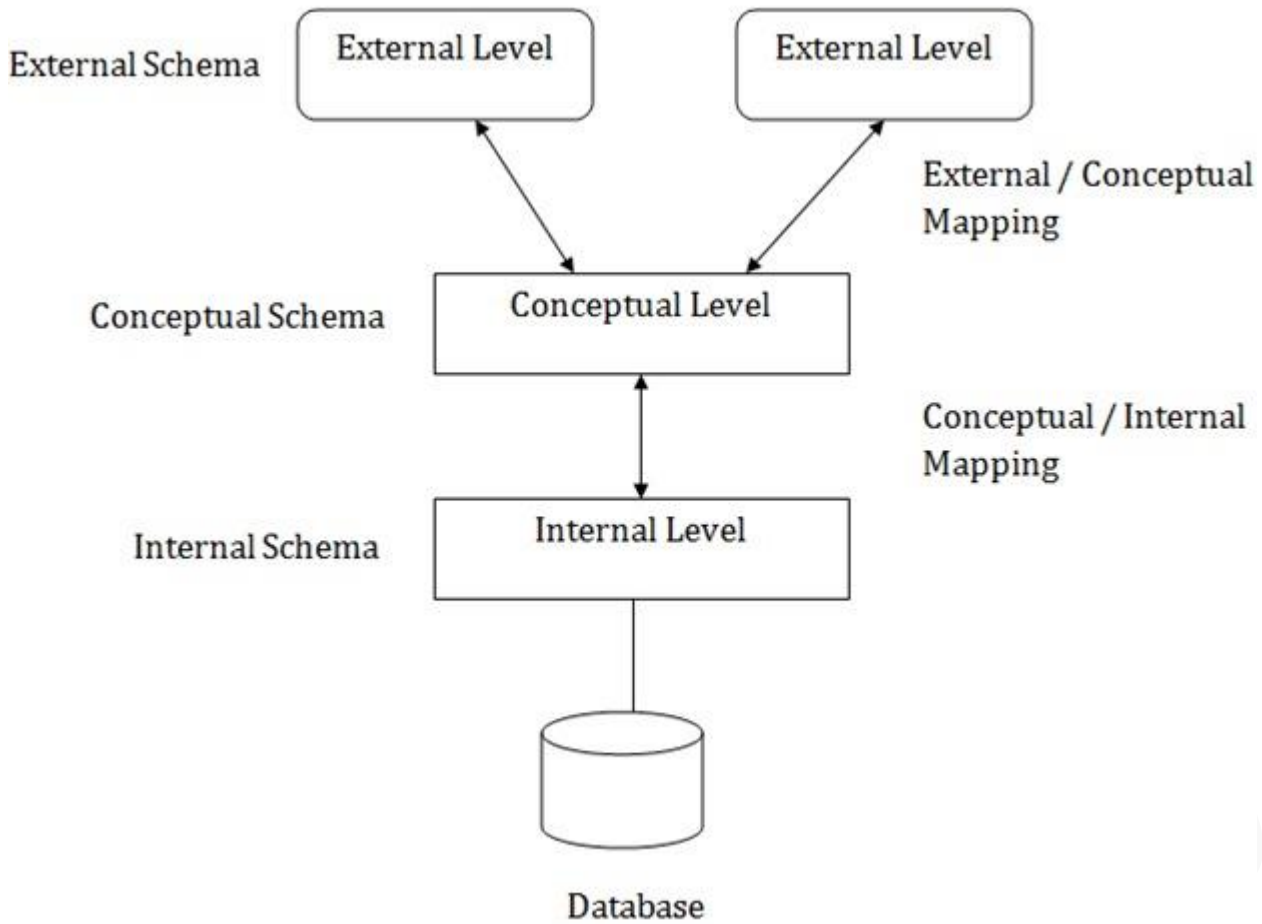
External / Conceptual Mapping

Conceptual/ Internal Mapping

The Conceptual/ Internal Mapping lies between the conceptual level and the internal level. Its role is to define the correspondence between the records and fields of the conceptual level and files and data structures of the internal level.

External/ Conceptual Mapping

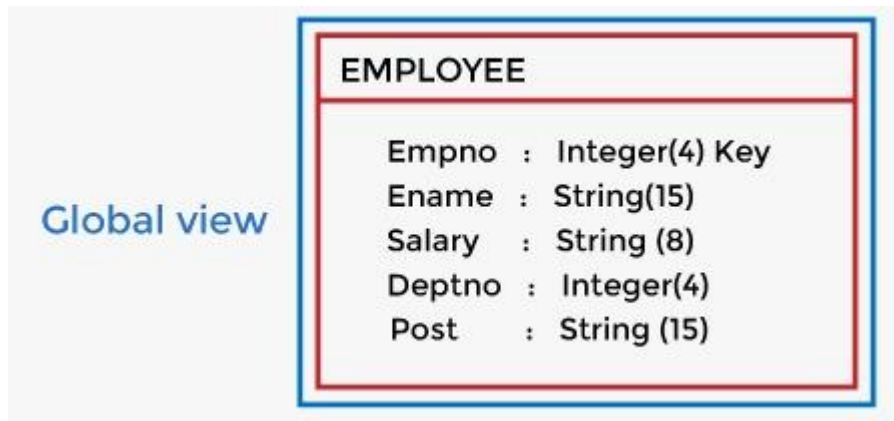
The external/Conceptual Mapping lies between the external level and the Conceptual level. Its role is to define the correspondence between a particular external and the conceptual view.



Internal view

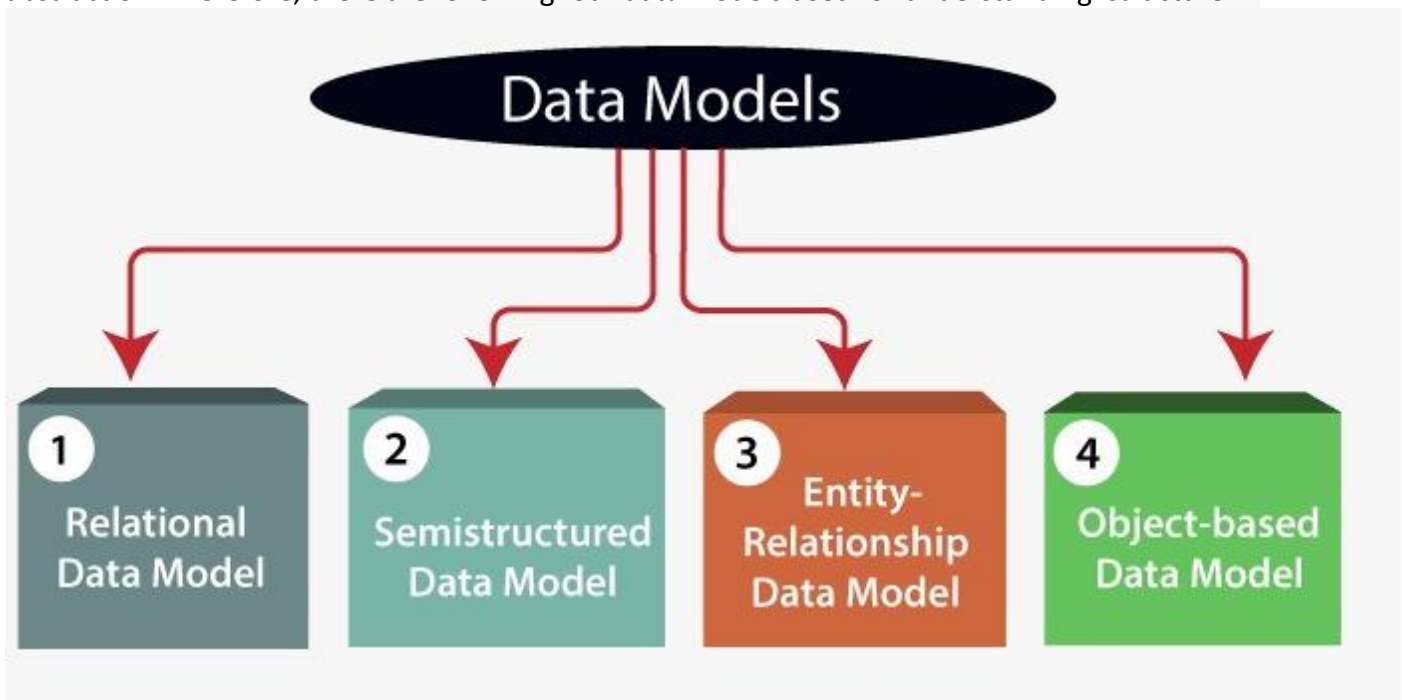
STORED_EMPLOYEE record length 60

Empno : 4 decimal offset 0 unique
Ename : String length 15 offset 4
Salary : 8,2 decimal offset 19
Deptno : 4 decimal offset 27
Post : string length 15 offset 31



3). What is datamodel? Explain different datamodel with example?

Data Model is the modeling of the data description, data semantics, and consistency constraints of the data. It provides the conceptual tools for describing the design of a database at each level of data abstraction. Therefore, there are following four data models used for understanding structure...



A data model in DBMS (Database Management System) is a conceptual representation of data, structures, relationships, and constraints used to organize and manipulate data in a database. It provides a way to describe how data is stored, accessed, and managed within the database system.

There are several different data models, each with its own approach to representing data. Some common data models include:

Relational Data Model: The relational data model organizes data into tables, where each table represents an entity, and the columns represent attributes or properties of that entity. The relationships between tables are defined through keys, such as primary keys and foreign keys. Examples of relational databases include MySQL, Oracle, and Microsoft SQL Server.

Example:

Consider a database for a university where you have tables for students, courses, and enrollment. The students table may have columns like student ID, name, and major. The courses table may have columns like course ID, title, and instructor. The enrollment table would establish a relationship between students and courses, containing columns like student ID and course ID.

Hierarchical Data Model: The hierarchical data model organizes data in a tree-like structure, where each record or entity has a parent-child relationship. Each parent can have multiple children, but each child has only one parent. This model is often used in mainframe database systems.

Example:

In a hierarchical database for an organization, the top-level entity could be the organization itself, with departments as its children, and employees as children of departments.

Network Data Model: The network data model is similar to the hierarchical model but allows for more complex relationships. It represents data as a collection of records connected by pointers. Each record can have multiple relationships with other records. This model is also commonly used in mainframe systems.

Example:

In a network database for a library, the record for a book can be connected to multiple records for authors, publishers, and categories.

Object-Oriented Data Model: The object-oriented data model represents data as objects, similar to object-oriented programming. It stores data and the methods or operations that can be performed on that data. This model is useful for representing complex data structures and is often used in object-oriented databases.

Example:

In an object-oriented database for a social media platform, you can have objects for users, posts, comments, and relationships between them.

These are just a few examples of data models in DBMS. Each data model has its own strengths and weaknesses, and the choice of data model depends on the specific requirements of the application and the data being stored.

4). What do you mean by E-R model?

Create an E-R diagram of Library management System with relationship weak and strong.

The ER (Entity-Relationship) model is a conceptual data model used to describe the entities or objects in a system, their attributes, and the relationships between them. It provides a graphical representation of the database schema and is widely used in database design.

In the ER model, entities represent real-world objects, concepts, or things with distinct characteristics. Attributes describe the properties or characteristics of an entity. Relationships define how entities are associated or connected to each other.

The key components of the ER model are:

Entity: An entity is a real-world object or concept with its own set of attributes. It can be a person, place, thing, event, or concept.

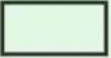




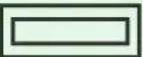
Attribute: An attribute is a property or characteristic of an entity. It describes the data that can be stored for an entity. For example, a person entity may have attributes such as name, age, and address.

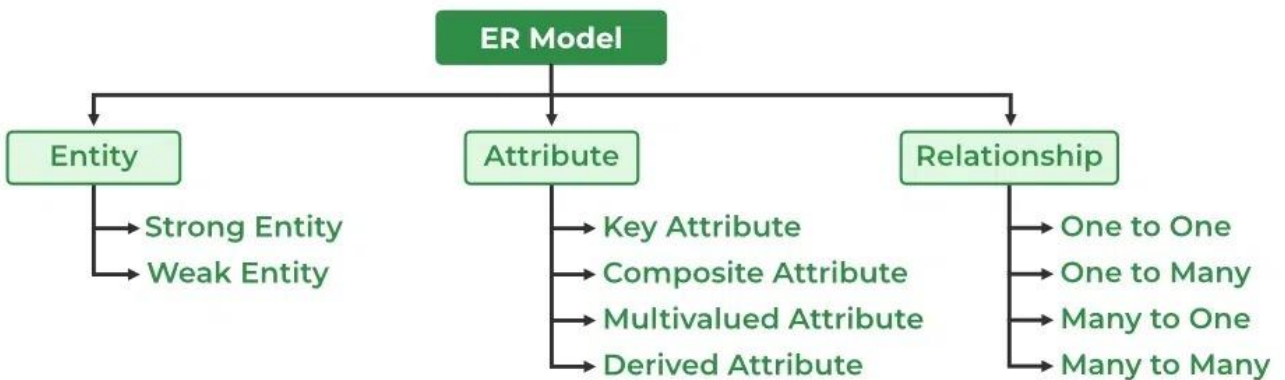
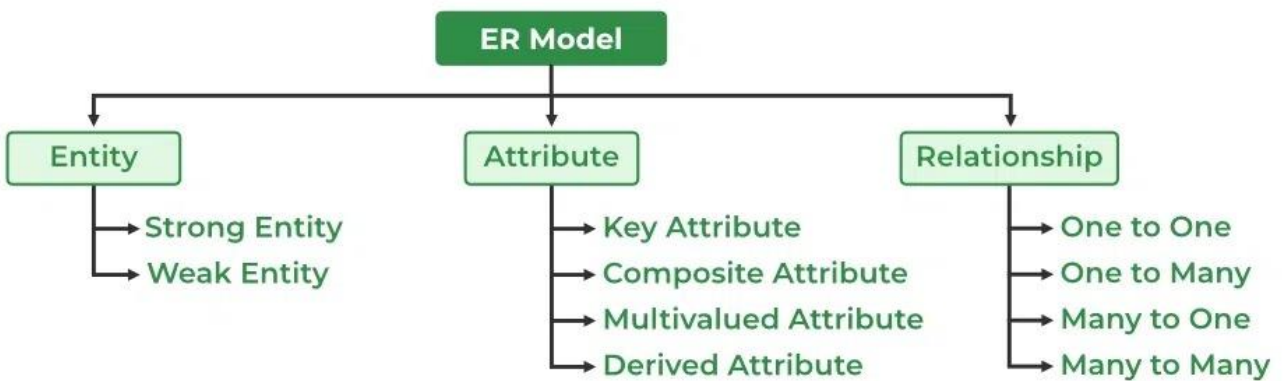
Relationship: A relationship represents an association or connection between two or more entities. It describes how entities are related to each other. Relationships can have cardinality constraints, such as one-to-one, one-to-many, or many-to-many.

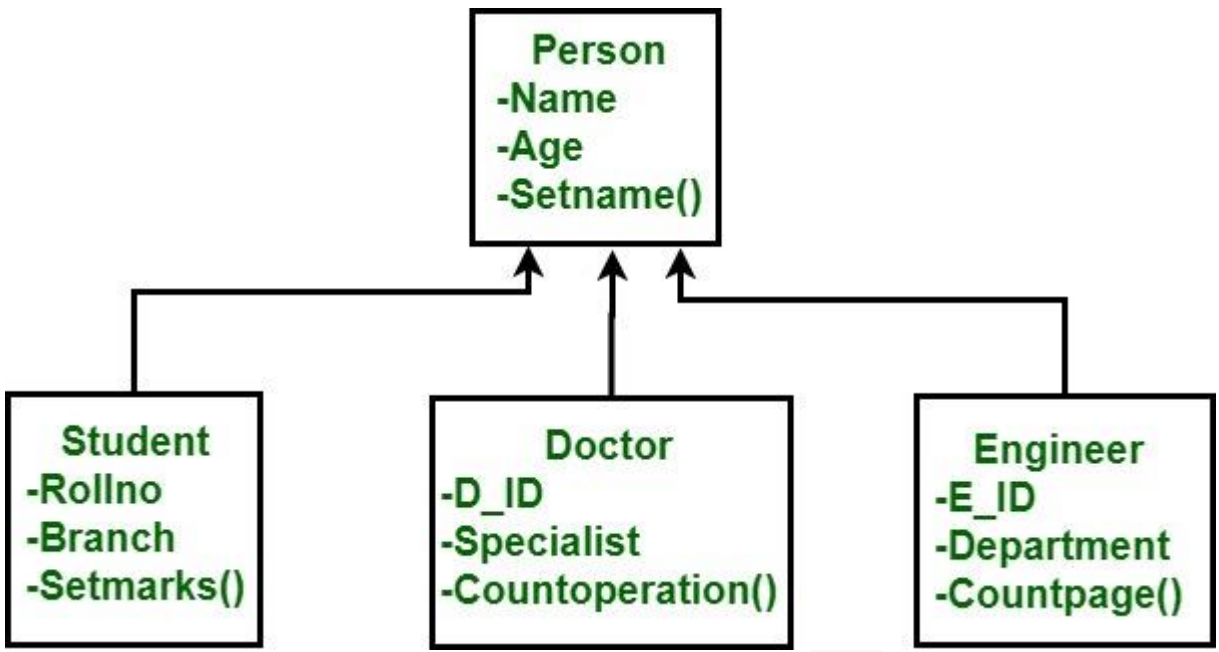
Cardinality: Cardinality defines the number of instances of one entity that can be associated with the instances of another entity in a relationship. It specifies the participation constraints between entities in a relationship.

The ER model is typically represented using ER diagrams, which use graphical symbols to depict entities, attributes, and relationships. The symbols include rectangles for entities, ovals for attributes, and diamonds for relationships.

ER modeling helps in understanding the structure of the database, identifying entities and their relationships, and capturing the business rules and constraints of the system. It serves as a foundation for database design and is often used as a starting point for transforming the ER model into a relational database schema.

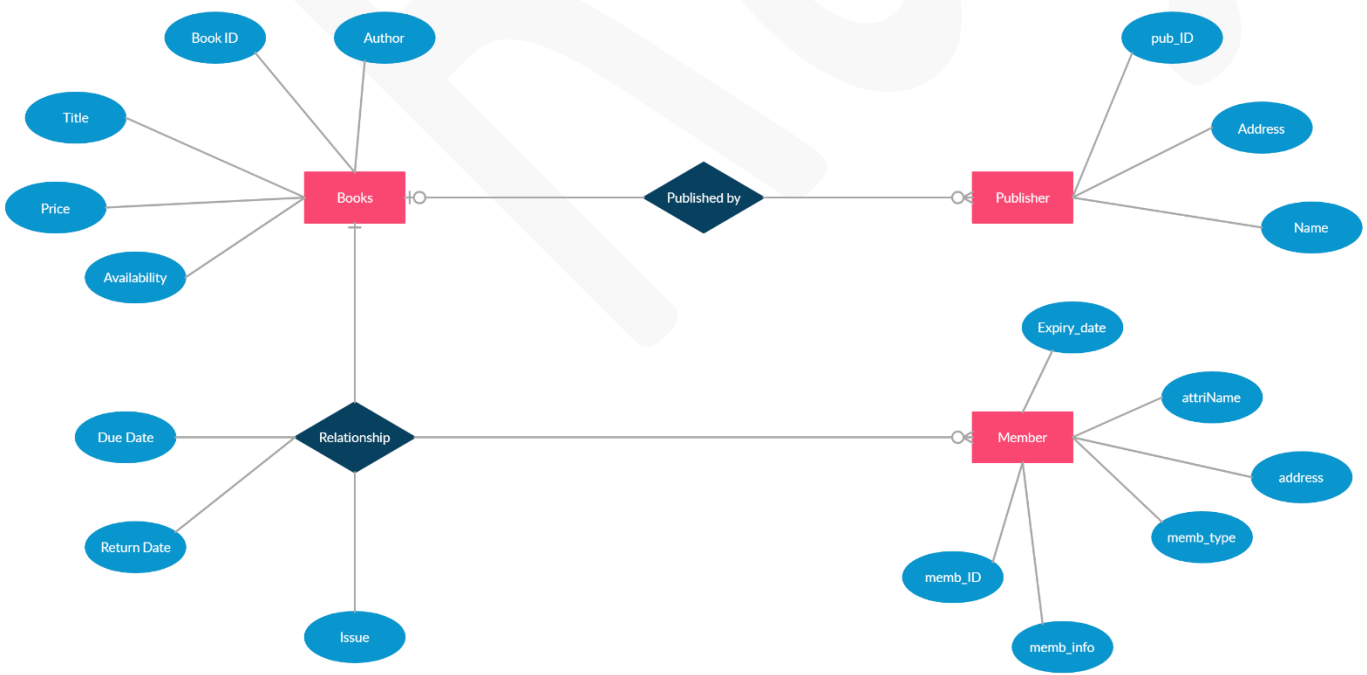
Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity





Create an E-R diagram of Library management System with relationship weak and strong.

E-R Diagram of Library Management System



The relationships in this ER diagram are as follows:

There is a one-to-many relationship between the "Member" entity and the "Borrowing" entity. A member can have multiple borrowing transactions, but each borrowing transaction is associated with only one member.

There is a one-to-many relationship between the "Book" entity and the "Borrowing" entity. A book can be borrowed multiple times, but each borrowing transaction is associated with only one book.

The primary keys are underlined, and foreign keys are indicated with "(FK)".

Please note that this is a simplified example, and in a real-world library management system, there could be additional entities, attributes, and relationships depending on the specific requirements and functionality of the system.

In this ER diagram:

The "Member" entity represents library members and contains attributes like **member_id**, name, email, and phone_number.

The "Book" entity represents books in the library and includes attributes such as **book_id**, title, author, and publication_year.

The "Borrowing" entity represents the borrowing transactions between members and books. It has attributes like **borrowing_id**, **member_id** (foreign key referencing member_id in the Member entity), **book_id** (foreign key referencing book_id in the Book entity), date_borrowed, due_date, and return_date.

The relationships in this ER diagram are as follows:

There is a one-to-many relationship between the "**Member**" entity and the "**Borrowing**" entity. A member can have multiple borrowing transactions, but each borrowing transaction is associated with only one member.

There is a one-to-many relationship between the "Book" entity and the "Borrowing" entity. A book can be borrowed multiple times, but each borrowing transaction is associated with only one book.

The primary keys are underlined, and foreign keys are indicated with "(FK)".

Please note that this is a simplified example, and in a real-world library management system, there could be additional entities, attributes, and relationships depending on the specific requirements and functionality of the system.

5). What is Integrity Constraints and explain its types?

In the context of database management systems (DBMS), integrity constraints are rules or conditions that are enforced on the data within a database to ensure its accuracy, consistency, and reliability. These constraints define the valid states and relationships that the data must adhere to. They help maintain data integrity and prevent the database from containing invalid or inconsistent data.

There are several types of integrity constraints commonly used in DBMS:

Entity Integrity Constraint (Primary Key Constraint):

This constraint ensures that each row in a table has a unique identifier, known as the primary key. It guarantees the uniqueness and non-nullity of the primary key column, preventing duplicate or missing values.

Referential Integrity Constraint (Foreign Key Constraint):

This constraint establishes a relationship between two tables based on a common column, known as the foreign key. It ensures that the values in the foreign key column of the referencing table match the values in the primary key column of the referenced table. It helps maintain consistency and enforce data relationships.

Domain Integrity Constraint:

This constraint defines the permissible values and data types for a column in a table. It ensures that the data entered into the column adheres to the defined domain or data type. For example, a column with a data type of integer should only contain numeric values.

Key Constraint (Unique Constraint):

This constraint ensures that the values in a column or a combination of columns are unique within a table. It prevents duplicate values from being inserted into the specified column(s).

Check Constraint:

This constraint defines a condition that must be true for the data within a column. It allows you to enforce custom business rules or specific conditions on the data. For example, a check constraint can be used to ensure that a column only contains positive values.

These integrity constraints collectively enforce data consistency, accuracy, and reliability in a database. By applying these rules, DBMS ensures that the data remains valid and maintains the integrity of the database structure.

6). What is data Independence and its types?

Data independence in DBMS refers to the ability to modify the database schema (the structure or organization of the database) without affecting the applications or programs that access the data. It provides a layer of abstraction between the physical implementation of the database and the way data is presented to users or applications.

There are two types of data independence:

Physical Data Independence:

Physical data independence allows you to modify the physical storage structures or techniques of the database without affecting the logical schema or the way data is accessed. It enables changes in the storage

hardware, file organization, indexing methods, or any other physical-level modifications, without requiring alterations to the application programs or queries that use the data.

For example, if you decide to change the underlying storage from a traditional file system to a distributed database or switch from magnetic disks to solid-state drives (SSDs), physical data independence ensures that the applications accessing the database will continue to work without any modifications.

Logical Data Independence:

Logical data independence enables you to modify the logical schema of the database without impacting the external schema or the way data is presented to users or applications. It allows you to add, modify, or remove tables, relationships, attributes, or constraints in the logical schema without affecting the existing applications that rely on the data.

For instance, if you need to add a new attribute to a table or create a new table with a relationship to existing tables, logical data independence ensures that the existing applications or queries can still retrieve and manipulate the data without any changes.

By providing data independence, DBMS allows for flexibility and modularity in database design and maintenance. It separates the logical view of data from its physical implementation, making it easier to adapt the database to evolving requirements, optimize performance, and manage changes without disrupting the functionality of applications or programs that rely on the data.

7).What is Relational Algebra?

Relational algebra is a theoretical framework and a fundamental concept in database management systems (DBMS) that provides a set of operations for manipulating relational data. It serves as a foundation for query languages like SQL and helps in defining and expressing various database operations.

Relational algebra operates on relations, which are typically represented as tables in a database. Each table consists of rows (tuples) and columns (attributes). The operations in relational algebra allow users to retrieve, combine, and transform data stored in these tables.

Here are some key operations in relational algebra:

Selection (σ):

This operation selects rows from a table that satisfy a given condition or predicate. It narrows down the data based on specified criteria. For example, selecting all employees with a salary greater than \$50,000

Projection (π):

Projection operation extracts specific columns (attributes) from a table, creating a new relation with only the selected attributes. It reduces the width of the table. For instance, projecting the employee table to only retrieve the employee names and their corresponding job titles.

Union (U):

Union combines rows from two compatible relations (tables) to create a new relation. The resulting relation contains all distinct rows from both relations. For example, combining the results of two queries to obtain a single result set.

Intersection (\cap):

Intersection returns only the common rows between two relations. It retrieves rows that exist in both relations. For instance, finding employees who are present in both the "Sales" and "Marketing" departments.

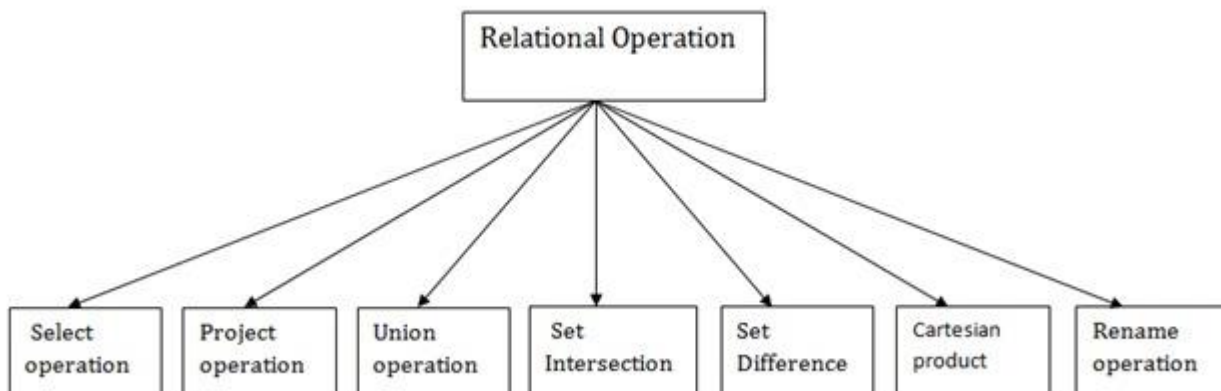
Set Difference (-):

Set difference operation retrieves rows from one relation that do not appear in another relation. It subtracts rows of one relation from another. For example, obtaining a list of products that are not currently in stock.

Cartesian Product (\times):

Cartesian product combines each row from one relation with every row from another relation. It results in a new relation with all possible combinations of rows. For example, finding all possible pairs of customers and products.

These are some of the basic operations in relational algebra. Additional operations like join, division, and renaming can be defined using these fundamental operations. Relational algebra provides a formal framework for expressing queries and operations in a precise and mathematical manner, facilitating the design and implementation of efficient database systems.



1. Select Operation:

The select operation selects tuples that satisfy a given predicate.

It is denoted by sigma (σ).

Notation: $\sigma p(r)$

Where:

σ is used for selection prediction

r is used for relation

p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like $=, \neq, \geq, <, >, \leq$.

For example: LOAN Relation

BRANCH_NAME	LOAN NUMBER	AMOUNT
Downtown	L-17	2000
Redwood	L-23	1500
Perryride	L-15	1500
Downtown	L-14	500
Mianus	L-13	900
Perryride	L-16	1300

Input:

σ BRANCH_NAME="perryride" (LOAN)

Output:

<u>BRANCH_NAME</u>	<u>LOAN NO</u>	<u>AMOUNT</u>
<u>Perryride</u>	<u>L-15</u>	<u>1500</u>
<u>Perryride</u>	<u>L-16</u>	<u>1300</u>

2. Project Operation:

This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.

It is denoted by π .

Notation: π A1, A2, An (r)

Where

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

Example: CUSTOMER RELATION

<u>NAME</u>	<u>STREET</u>	<u>CITY</u>
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	BRANCH_NAME	Rye
Johnson	Alma	Brooklyn
Brook	Senator	Brooklyn

Input:

π NAME, CITY (CUSTOMER)

Output:

<u>NAME</u>	<u>CITY</u>
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brook	Brooklyn

3. Union Operation:

Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.

It eliminates the duplicate tuples. It is denoted by \cup .

Notation: R \cup S

Example:

DEPOSITOR RELATION

<u>CUSTOMER-NAME</u>	<u>ACCOUNT_NO</u>
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

<u>CUSTOMER NAME</u>	<u>LOAN-NO</u>
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

Input:

$\uparrow \uparrow$ CUSTOMER_NAME (BORROW) \cup $\uparrow \uparrow$ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER NAME
Johnson

Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
mayer

4. Set Intersection:

Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.

It is denoted by intersection \cap .

Notation: $R \cap S$

Example: Using the above DEPOSITOR table and BORROW table

8. Define

a) What is Domain in DBMS?

In the context of database management systems (DBMS), a domain refers to the set of allowable values that a specific attribute or column in a database table can take. It defines the data type and constraints associated with the attribute.

A domain specifies the range of values that a particular attribute can have, ensuring data integrity and consistency within the database. It helps in enforcing data validation rules and constraints, preventing the entry of invalid or inappropriate values into the database.

For example, consider a database table called "Employees" with an attribute "Age." The domain for the "Age" attribute could be defined as positive integers between 18 and 65. Any value entered into the "Age" column must adhere to this domain definition; otherwise, it would be considered invalid.

By defining domains, DBMSs ensure data accuracy, integrity, and consistency by restricting the values that can be stored in the database. Domains help enforce business rules and provide a level of control over the data entered into the database.

b) What is Tuple in DBMS?

In the context of database management systems (DBMS), a tuple refers to a row or record in a relational database table. It represents a single instance or occurrence of an entity being described by the table.

A tuple consists of a collection of attribute values that correspond to the columns of the table. Each attribute value within a tuple represents a specific piece of data associated with the entity being represented by that tuple.

For example, consider a database table called "Students" with columns such as "StudentID," "Name," and "Age." Each row in the table represents a tuple, which contains attribute values for each column. A tuple in this table could be something like:

(StudentID: 101, Name: John Smith, Age: 20)

This tuple represents a specific student in the "**Students**" table, where the **StudentID** is **101**, the Name is **John Smith**, and the **Age** is **20**.

Tuples in a relational database are considered atomic, meaning they cannot be divided further. They represent a single unit of data that is indivisible within the context of the database. Tuples are used to store and retrieve data, perform operations like searching and filtering, and establish relationships between tables through keys.

Define a. Generalization

In database management systems (DBMS), generalization refers to the process of deriving a more general or abstract entity from a set of more specific entities. It is a concept used in database design and modeling to create hierarchical relationships between entities, allowing for data organization and abstraction...

Generalization is closely related to the concept of inheritance in object-oriented programming. It involves identifying common attributes and relationships among a group of entities and combining them to form a more generalized entity. The resulting entity represents the shared characteristics and relationships of the specific entities.

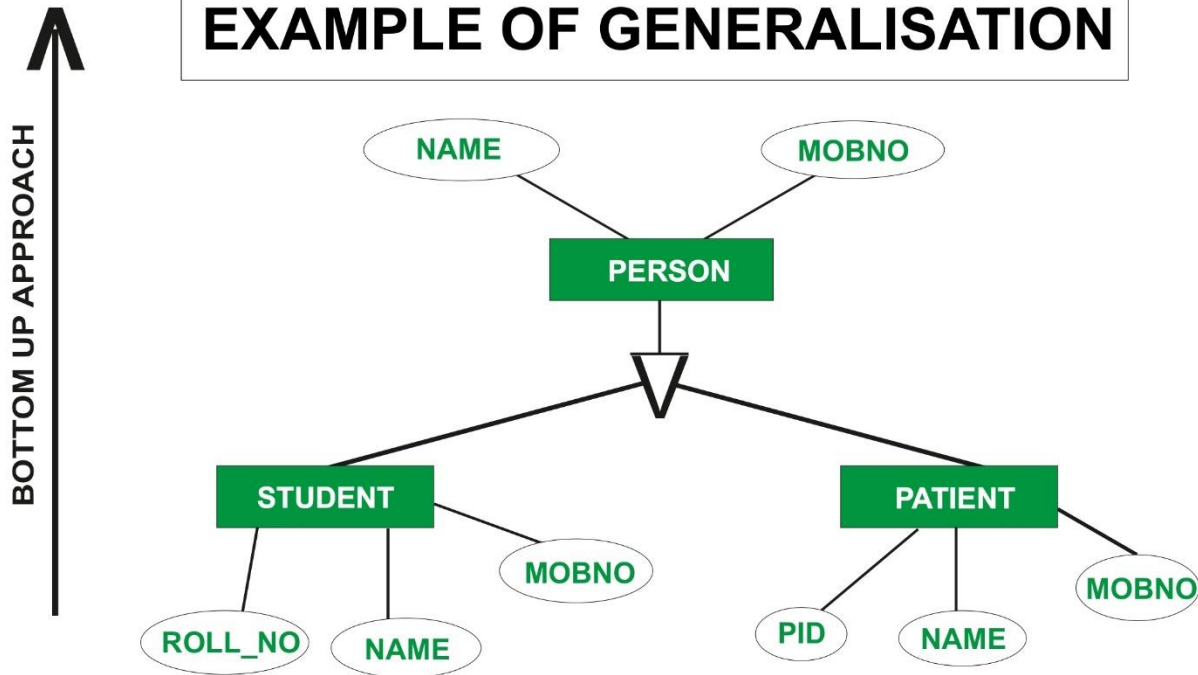
Here are some key points to understand about generalization in DBMS:

Superclass and Subclass: In a generalization hierarchy, the more generalized entity is referred to as the superclass or the parent entity. The more specific entities that inherit attributes and relationships from the superclass are called subclasses or child entities.

Inheritance of Attributes: When a subclass is derived from a superclass, it inherits all the attributes of the superclass. The subclass may also add its own specific attributes to further specialize the entity. This inheritance allows for the sharing of common attributes across entities and reduces redundancy in the database schema.

Inheritance of Relationships: Similarly, relationships defined for the superclass are inherited by the subclasses. This means that the relationships established between the superclass and other entities can also be applied to the subclasses. It helps in maintaining consistency and integrity in the database structure.

EXAMPLE OF GENERALISATION

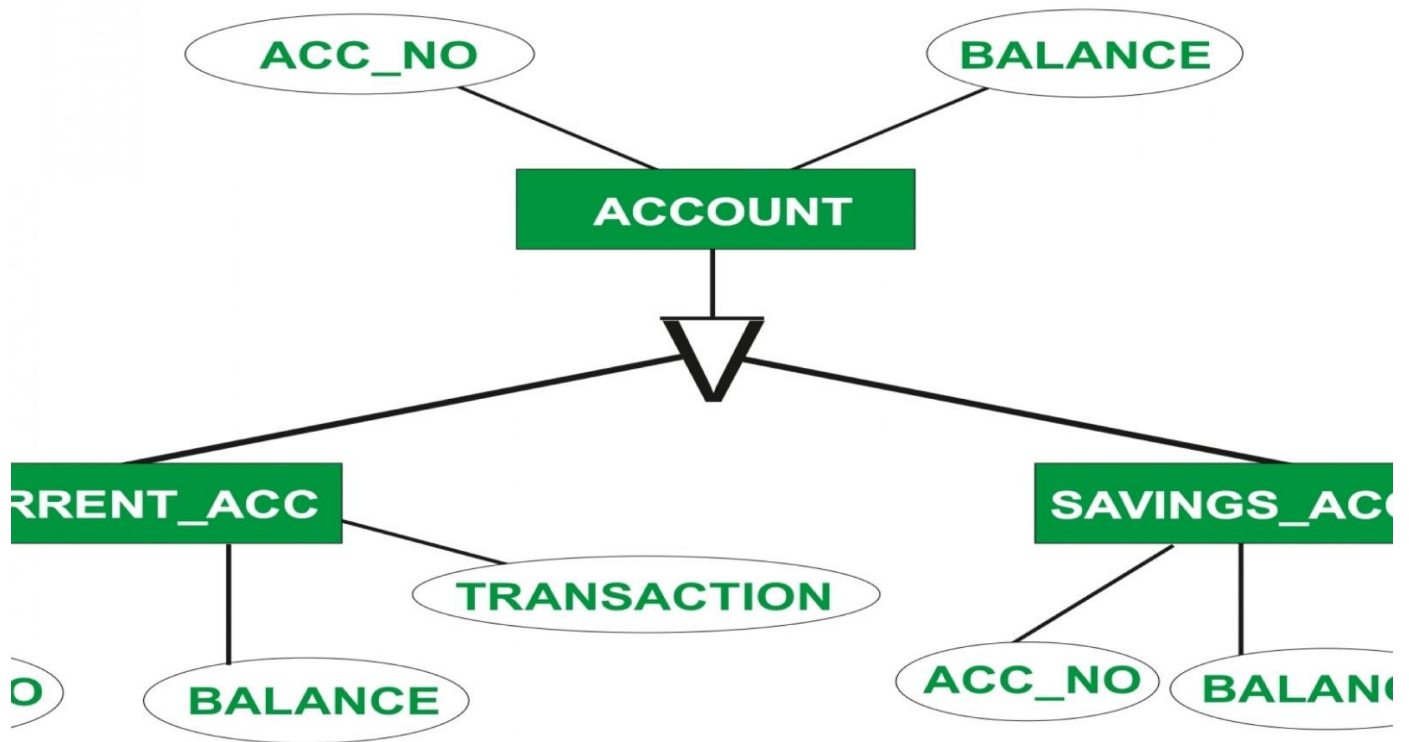


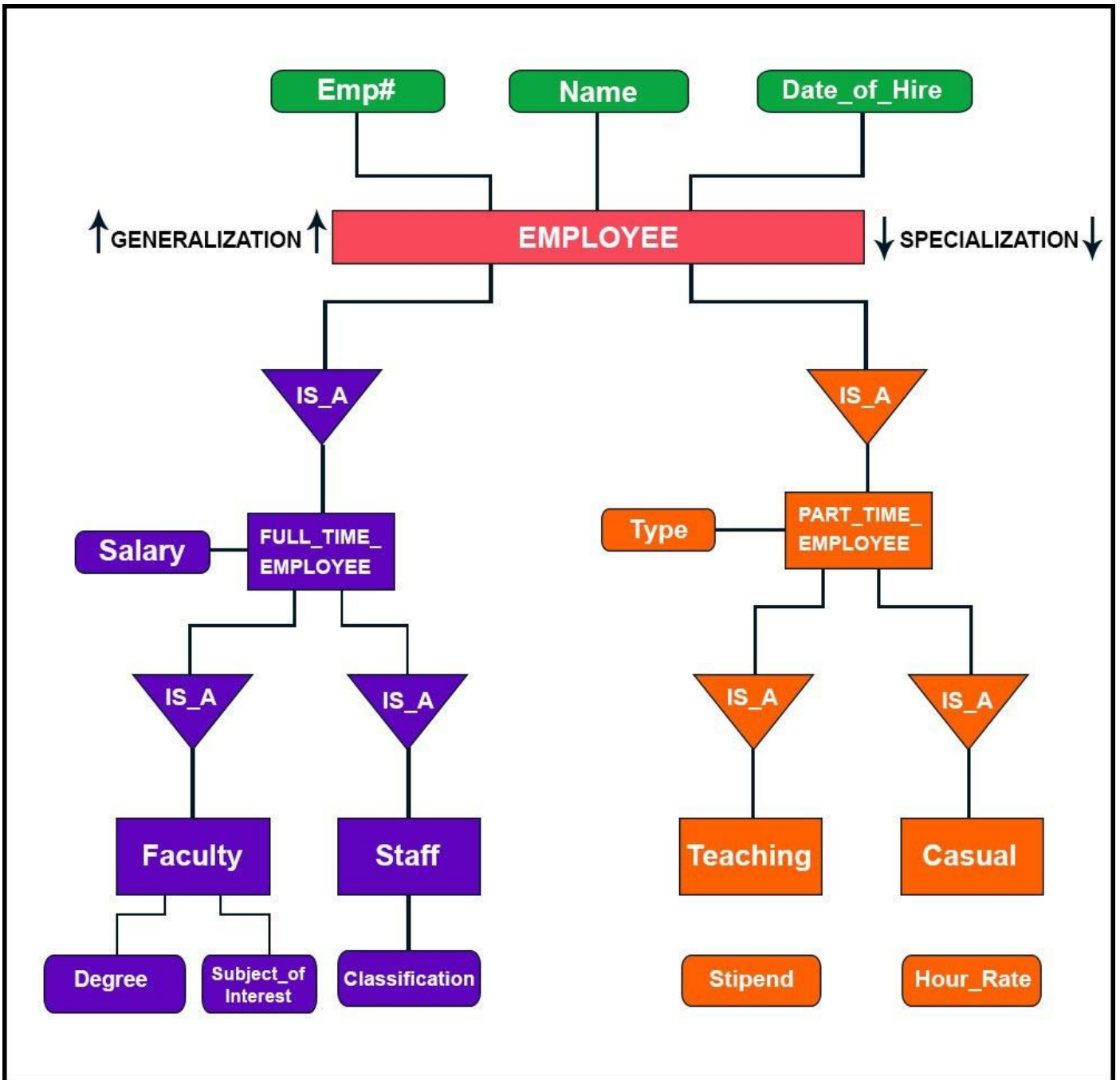
Specialization: Specialization is the reverse process of generalization. It involves creating specific entities from a more general entity. It is achieved by identifying subsets of attributes and relationships from the superclass and defining them in separate subclasses.

Generalization Hierarchies: Generalization hierarchies represent the hierarchical structure formed by generalizing and specializing entities. They illustrate the relationships and inheritance between entities in a graphical or tree-like format. The hierarchy allows for efficient organization and retrieval of data based on the level of abstraction needed.

Generalization is an essential concept in database design as it enables data modeling with increased flexibility, reusability, and maintainability. It supports the concept of data abstraction by allowing users to work with higher-level, more generalized entities while still retaining the ability to access and manipulate the specific details when necessary.

EXAMPLE OF SPECIALISATION





Source-Vani madam

Made by-Rajasekharreddy