

# DBMS

## 1. What is a Database Management System (DBMS)?

---

A **Database Management System (DBMS)** is software that is used to manage and organize databases. It provides an interface to interact with the data stored in a database. The DBMS is responsible for tasks such as storing, retrieving, and updating data, ensuring data integrity, security, and managing concurrency. Examples include MySQL, PostgreSQL, Oracle, and SQL Server.

## 2. What are the primary components of a DBMS?

---

The main components of a DBMS can be broken down into a few key parts:

- **Database Engine:** This is the core service that stores, retrieves, and manages data.
- **Database Schema:** It defines the logical structure of the data, like tables, relationships, and constraints.
- **Query Processor:** This interprets and executes SQL queries, making sure the requests are optimized and efficient.
- **Transaction Manager:** It handles transactions and ensures the **ACID properties** — atomicity, consistency, isolation, and durability.
- **Storage Manager:** This manages how data is stored on disk and retrieved efficiently.

Together, these components ensure that data is well-structured, queries run smoothly, and transactions remain reliable

## 3. What are the advantages of using a DBMS?

---

The advantages of using a DBMS are:

- **Data Integrity:** Ensures that the data is accurate and consistent.
- **Data Security:** Provides controlled access to sensitive data by setting permissions for different users.

- **Efficient Data Retrieval:** Optimizes queries and indexing, allowing faster data retrieval.
- **Reduced Redundancy:** Avoids duplicate data by enforcing normalization.
- **Backup and Recovery:** Offers automatic backup and recovery mechanisms.
- **Concurrent Access:** Allows multiple users to access the database at the same time without conflicts.

#### 4. What is the difference between DBMS and RDBMS?

---

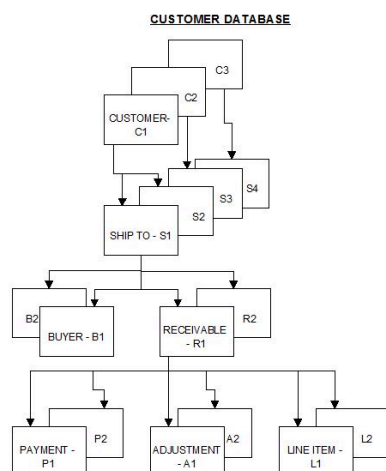
- **DBMS (Database Management System):** A system that allows users to create, store, modify, and delete data. It does not require a relational structure for data organization. Examples: Microsoft Access, XML databases.
- **RDBMS (Relational Database Management System):** A subset of DBMS that stores data in a structured format, using tables (relations), and supports relational operations like joins. It enforces data integrity through keys and supports SQL for querying. Examples: MySQL, Oracle, SQL Server.

#### 5. What are the different types of DBMS?

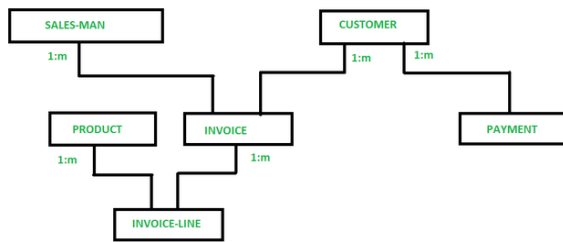
---

The four types of DBMS are:

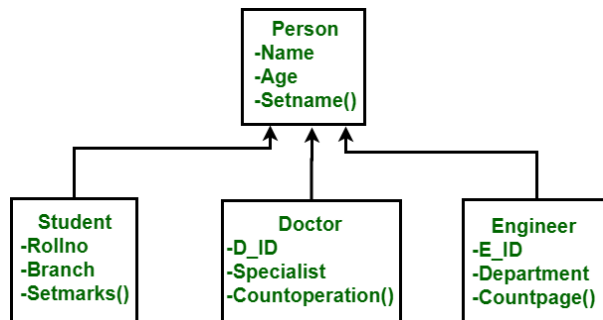
- **Hierarchical DBMS:** Data is organized in a tree-like structure with parent-child relationships. Example: IBM's IMS.



- **Network DBMS:** Data is represented as a graph with many-to-many relationships. Example: Integrated Data Store (IDS).



- **Relational DBMS (RDBMS):** Data is organized in tables (relations) and managed through SQL. Example: MySQL, PostgreSQL.
- **Object-Oriented DBMS:** Data is stored as objects, like in object-oriented programming. Example: ObjectDB.



## 6. What is a relation in DBMS?

---

A **relation** in DBMS is a table that consists of rows and columns. Each row represents a record, and each column represents an attribute or property of the entity being described. Tables are defined by a schema, which specifies the attributes (columns) of the table.

## 7. What is a table in DBMS?

---

A **table** in DBMS is a collection of data organized in rows and columns. It is the primary structure for storing data in a relational database. Each row represents an entity (record), and each column represents an attribute of that entity.

## 8. What are rows and columns in a DBMS?

---

- **Rows (Tuples):** A row represents a single record or entity. Each row contains values for each attribute (column).

- **Columns (Attributes):** A column represents a property or characteristic of the entity. Each column has a data type, such as integer, string, etc.

## 9. What is a primary key? Explain with an example.

---

A **Primary Key** is a unique identifier for each record in a table. It ensures that no two records have the same value for the primary key field. It cannot contain **NULL** values. **Example:** In a `STUDENT` table, `ROLL_NO` could be the primary key because each student has a unique roll number.

ROLL_NO	NAME	ADDRESS
1	Ram	Delhi
2	Suresh	Delhi

## 10. What is a foreign key? Explain with an example.

---

A **Foreign Key** is an attribute in a table that links to the primary key in another table. It creates a relationship between two tables, ensuring referential integrity. **Example:** In a `STUDENT` table, the `BRANCH_CODE` could be a foreign key referencing the primary key `BRANCH_CODE` in the `BRANCH` table.

### Student Table

ROLL_NO	NAME	BRANCH_CODE
1	Ram	CS
2	Suresh	IT

### BRANCH Table

BRANCH_CODE	BRANCH_NAME
CS	Computer Science
IT	Information Technology

Here, `BRANCH_CODE` in `STUDENT` is a foreign key referencing `BRANCH_CODE` in `BRANCH`.

## 11. What is normalization? Why is it important in DBMS?

---

**Normalization** is the process of organizing data in a way that reduces redundancy and dependency. It involves dividing large tables into smaller ones and defining relationships between them to ensure data integrity.

### Importance:

- It eliminates redundant data.
- It prevents anomalies during data operations (insertion, update, deletion).
- It improves data integrity and consistency.

## 12. What is denormalization? How does it differ from normalization?

---

**Denormalization** is the process of combining tables to improve query performance, often by introducing redundancy. While normalization minimizes redundancy, denormalization sacrifices some of it to improve speed for read-heavy operations.

## 13. What is a candidate key in DBMS?

---

A **Candidate Key** is a set of one or more attributes that can uniquely identify a tuple (row) in a relation (table). A relation can have multiple candidate keys, and one of them is chosen as the primary key.

## 14. What is composite key in DBMS?

---

A **composite key** is a **primary key** that consists of **two or more attributes (columns)**, which together uniquely identify a record in a table. Individually, those columns may not be unique, but when combined, they guarantee uniqueness. It is a type of superkey.

## 15. What is the use of the SQL SELECT statement?

---



The **SELECT** statement is used to query data from one or more tables. It allows you to retrieve specific columns or all columns, optionally applying filters (WHERE), sorting (ORDER BY), and joining multiple tables.

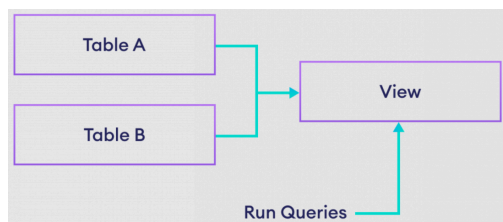
**Example:** SELECT NAME, AGE FROM STUDENT WHERE AGE > 18;

## 16. What is a view in DBMS? How does it differ from a table?

---

A **View** is a virtual table created by querying one or more base tables. It doesn't store data itself; instead, every time you query a view, the DBMS runs the underlying query on the base tables. Unlike a table, a view does not store its own data but presents data from other tables.

-  Advantage → Always shows the most **up-to-date data**.
-  Disadvantage → If the query is **complex (joins, aggregations, etc.)**, it can be **slow**, because the database must recompute results every time.




Here is how we can create a view:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name(s)
WHERE condition;
```

## 17. What is a materialized view in DBMS?

---

A **materialized view** is a database object that contains the results of a query. Unlike a regular view, which is a virtual table (it doesn't store data), a materialized view stores data physically, improving query performance by precomputing and storing results.

-  Advantage → Super **fast to query**, since results are precomputed and saved.

- **✗ Disadvantage** → The data can become **stale** (outdated) if the base tables change, unless the materialized view is refreshed.  
**Use Case:** Materialized views are commonly used for performance optimization in data warehousing and reporting systems, where the same data is frequently queried.  
**Example:**

```
CREATE MATERIALIZED VIEW SalesSummary AS  
SELECT Product, SUM(Quantity) FROM Sales GROUP BY Product;
```

## 18. What are the different types of relationships in DBMS?

---

The three main types of relationships in DBMS are:

- **One-to-One (1:1):** A record in one table is associated with a single record in another table.
- **One-to-Many (1:M):** A record in one table is associated with multiple records in another table.
- **Many-to-Many (M:M):** Multiple records in one table are associated with multiple records in another table.

## 19. Explain the concept of a schema in DBMS.

---

A **schema** in DBMS is the structure that defines the organization of data in a database. It includes tables, views, relationships, and other elements. A schema defines the tables and their columns, along with the constraints, keys, and relationships.

## 20. What are constraints in DBMS, and what are the different types of constraints? Give examples.

---

**Constraints** in DBMS are rules that limit the type of data that can be inserted into a table to ensure data integrity and consistency. Common types of constraints include: **NOT NULL, PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, DEFAULT.**

- **NOT NULL:** Ensures that a column cannot have NULL values. **Example:** `Name VARCHAR(50) NOT NULL;`
- **PRIMARY KEY:** Uniquely identifies each record in a table. It ensures that no duplicate rows exist and that no NULL values are allowed. **Example:** `ID INT PRIMARY KEY;`
- **FOREIGN KEY:** Ensures referential integrity between two tables by linking a column in one table to the primary key in another table. **Example:** `BranchCode INT FOREIGN KEY REFERENCES Branch(BranchCode);`
  - Referential integrity means maintaining **consistency in relationships between tables** using foreign keys. It ensures that a foreign key value in one table always refers to a valid primary key in another table.
  - For example, if we have a `Students` table with a `DepartmentID` as a foreign key referencing the `Departments` table, referential integrity ensures that every `DepartmentID` in `Students` must exist in `Departments`.
  - If someone tries to insert a student with a non-existent department, or delete a department that still has students linked to it, the DBMS prevents it
- **UNIQUE:** Ensures that all values in a column are distinct. Unlike the primary key, it allows multiple NULL values to be shown. **Example:** `Email VARCHAR(100) UNIQUE;`
- **CHECK:** Ensures that values in a column satisfy a specific condition. **Example:** `Age INT CHECK (Age >= 18);`
- **DEFAULT:** Assigns a default value to a column if no value is provided during insertion. **Example:** `Status VARCHAR(10) DEFAULT 'Active';`

## 21. What is the difference between DELETE, TRUNCATE and DROP in SQL?

---

- **DELETE:** Removes specific rows from a table based on a condition ( `WHERE` clause). Each row deletion is logged, so the operation is slower but allows transactions to be rolled back if needed. The table structure remains intact.
- **TRUNCATE:** Removes **all rows** from a table without logging individual row deletions. It is faster than `DELETE` because it deallocates the data pages instead of deleting row by row. In most databases, it cannot be rolled back once committed. The table structure remains.
- **DROP:** Completely removes the **table structure** along with its data, indexes, and constraints. After a `DROP`, the table no longer exists in the database



unless recreated.

## 22. What is an index in DBMS and how is it used?

---

An **index** is a data structure that improves the speed of data retrieval operations on a database table. It works like a table of contents in a book, allowing the database to quickly find the location of a record based on a column value.

## 23. Explain the concept of indexing in DBMS.

---

**Indexing** is a technique used to speed up the retrieval of records from a table by creating a data structure that allows for faster searching. An index provides a quick lookup of data based on the values of one or more columns.

- **Types of Indexes:**
  - **Single-column index:** An index created on a single column of a table to improve lookup speed based on that column.
  - **Composite index:** An index created on two or more columns of a table, useful when queries filter or sort by multiple columns.
  - **Unique index:** An index that ensures all values in the indexed column(s) are unique, preventing duplicate entries.

## 24. What is the role of the Database Administrator (DBA)?

---

A **Database Administrator (DBA)** is responsible for managing and overseeing the entire database environment. Their key responsibilities include:

- **Database Design:** Structuring the database for optimal storage and performance.
- **Backup and Recovery:** Ensuring regular backups and providing recovery solutions in case of failures.
- **Performance Tuning:** Monitoring and optimizing the database's performance.
- **Security Management:** Managing user access, privileges, and enforcing security policies.
- **Data Integrity:** Ensuring data consistency and integrity through constraints

and checks.

- **Upgrades and Patches:** Keeping the database software up-to-date with patches and upgrades.
- **Troubleshooting:** Identifying and resolving database-related issues.

## 25. What is the role of the DBMS in handling data integrity and security?

---

The **DBMS** plays a critical role in ensuring:

- **Data Integrity:** Through constraints like **Primary Keys**, **Foreign Keys**, and **Check Constraints**, the DBMS ensures data consistency and accuracy.
- **Data Security:** DBMS systems provide user authentication, access control, and encryption mechanisms to protect data from unauthorized access and breaches. It also supports role-based access control (RBAC), ensuring that only authorized users can perform certain actions on the data.

## 26. What is an entity-relationship diagram (ERD)?

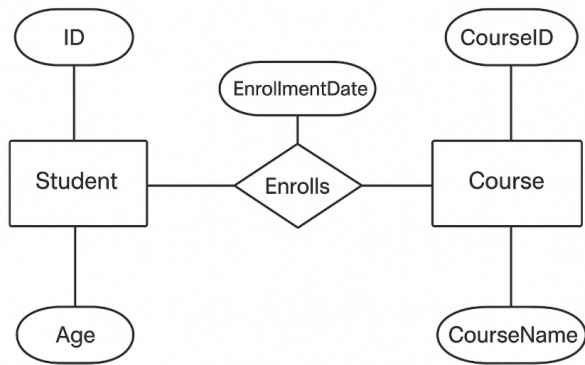
---

An **Entity-Relationship Diagram (ERD)** is a visual representation of the entities within a system and the relationships between those entities. It is used in database design to define the structure of data and how different pieces of data relate to each other.

- **Entities:** Objects or things within the system (e.g., `Student` , `Course` ).
- **Attributes:** Properties or details about an entity (e.g., `Student Name` , `Course Duration` ).
- **Relationships:** How entities interact with each other (e.g., `Student enrolls in Course` ).

### **Example of ERD:**

- A `Student` entity might have attributes like `ID` , and `Age` .
- A `Course` entity might have attributes like `CourseID` , `CourseName` .
- A relationship `Enrolls` connects `Student` and `Course` with attributes like `EnrollmentDate` .

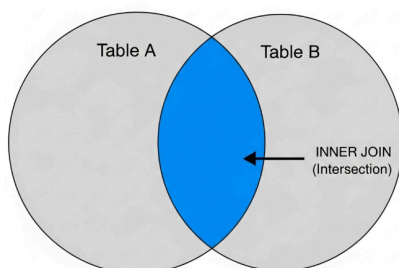


## 27. What is a join in SQL? Name and explain different types of joins.

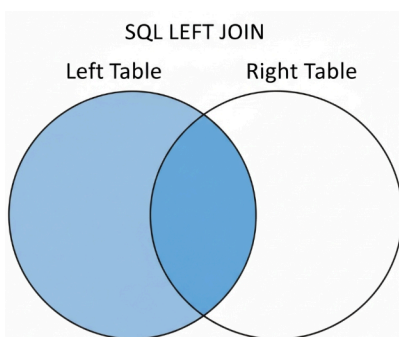
A **JOIN** in SQL is an operation that combines columns from two or more tables based on a related column between them. Joins are used to query data from multiple tables in a relational database.

Here are the different types of **joins** in SQL:

- **INNER JOIN:** Returns only the rows where there is a match in both tables.
  - **Example:** `SELECT * FROM Student INNER JOIN Course ON Student.ID = Course.StudentID;`



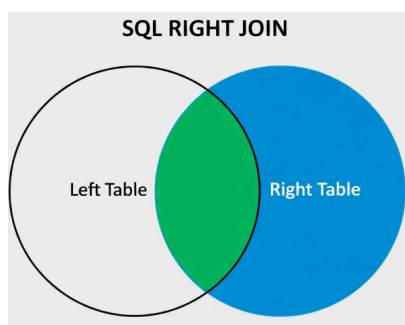
- **LEFT JOIN (or LEFT OUTER JOIN):** Returns all rows from the left table and the matching rows from the right table. If there is no match, NULL values will be returned for columns from the right table.
  - **Example:** `SELECT * FROM Student LEFT JOIN Course ON Student.ID = Course.StudentID;`



- **RIGHT JOIN (or RIGHT OUTER JOIN):** Returns all rows from the right table and the matching rows from the left table. If there is no match, NULL values

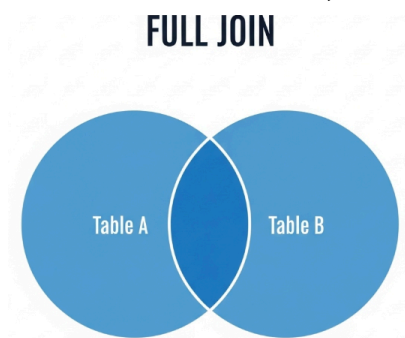
will be returned for columns from the left table.

- **Example:** `SELECT * FROM Student RIGHT JOIN Course ON Student.ID = Course.StudentID;`



- **FULL JOIN (or FULL OUTER JOIN):** Returns all rows when there is a match in either the left or the right table. If there is no match, NULL values will be returned for the columns of the table without a match.

- **Example:** `SELECT * FROM Student FULL JOIN Course ON Student.ID = Course.StudentID;`



- **CROSS JOIN:** Returns the Cartesian product of two tables, i.e., every combination of rows from both tables.
  - **Example:** `SELECT * FROM Student CROSS JOIN Course;`
- **SELF JOIN:** A **SELF JOIN** is a join where a table is joined with itself. It is used when we need to compare rows within the same table. To differentiate the two instances of the same table, aliases are used.
  - **Example:** `SELECT E1.Employee_ID, E1.Employee_Name, E2.Employee_Name AS Manager_Name FROM Employee E1 LEFT JOIN Employee E2 ON E1.Manager_ID = E2.Employee_ID;`

## 28. What is a subquery in SQL? Provide an example.

---

A **subquery** in SQL is a query embedded within another query. It is used to retrieve data that will be used in the outer query. Subqueries can be used in SELECT, INSERT, UPDATE, or DELETE statements.

There are two types of subqueries:

- **Single-row subquery:** Returns a single value.
- **Multiple-row subquery\*** Returns multiple values.

**Example of a subquery\*:** To find the names of students who have a higher age than the average age:

```
SELECT Name FROM Student WHERE Age > (SELECT AVG(Age) FROM Student);
```

## 29. What are aggregate functions in SQL? Name a few examples.

---

**Aggregate functions** in SQL are functions that operate on a group of rows and return a single result. They are often used in conjunction with the **GROUP BY** clause. Here are a few commonly used **aggregate functions** in SQL:

- **COUNT():** Returns the number of rows or non-NULL values in a column
  - Example: `SELECT COUNT(*) FROM Student;`
- **SUM():** Returns the sum of values in a numeric column.
  - Example: `SELECT SUM(Amount) FROM Orders;`
- **AVG():** Returns the average value of a numeric column.
  - Example: `SELECT AVG(Salary) FROM Employees;`
- **MAX():** Returns the maximum value in a column
  - Example: `SELECT MAX(Age) FROM Student;`
- **MIN():** Returns the minimum value in a column.
  - Example: `SELECT MIN(Salary) FROM Employees;`

## 30. What is a transaction in DBMS? What are the properties of a transaction?

---

A **transaction** in DBMS is a sequence of one or more SQL operations executed as a single unit of work. A transaction ensures data integrity, consistency, and isolation, and it guarantees that the database reaches a valid state, regardless of errors or system failures.

**Properties of a transaction (ACID Properties):**

**ACID** stands for **Atomicity**, **Consistency**, **Isolation**, and **Durability**, which are the key properties that guarantee reliable transaction processing:

- **Atomicity:** All operations in a transaction are treated as a single unit. If one operation fails, the entire transaction fails and the database state remains unchanged.
  - *For example:* Suppose you transfer ₹500 from **Account A** to **Account B**. The transaction involves two steps:
    1. Deduct ₹500 from Account A.
    2. Add ₹500 to Account B.
    - If step 1 succeeds but step 2 fails (e.g., system crash), the entire transaction is rolled back so the ₹500 is not deducted.
- **Consistency:** Ensures the database moves from one valid state to another valid state, always following rules and constraints.
  - *For example:* If the bank rule says an account balance cannot go negative, a transaction trying to withdraw ₹2000 from an account with only ₹1500 will fail. The database remains consistent with the rule.
- **Isolation:** Transactions are executed independently, and the intermediate states of a transaction are invisible to other transactions.
  - *For Example:* Two people booking the **last movie ticket** at the same time:
    - Transaction 1: User A selects the ticket.
    - Transaction 2: User B also selects the ticket simultaneously.
    - Isolation ensures only one succeeds, preventing both from booking the same ticket.
- **Durability:** Once a transaction is committed, its changes are permanent, even if a system failure occurs.
  - *For Example:* If you book a flight ticket and get a confirmation message, the booking stays in the database even if the airline's server crashes immediately after.

### 31. Explain the concept of a stored procedure in DBMS.

---

A **stored procedure** is a precompiled collection of one or more SQL statements stored in the database. Stored procedures allow users to execute a series of operations as a single unit, improving performance and reusability. They can accept input parameters, perform operations, and return results.

**Example:**

```
CREATE PROCEDURE GetStudentDetails(IN student_id INT)
BEGIN
    SELECT * FROM Student WHERE ID = student_id;
END;
```

### 32. What are stored functions in DBMS?

---

A **stored function** is a set of SQL statements that can be executed in the database. It accepts input parameters, performs some logic, and returns a value. Stored functions are similar to stored procedures but differ in that they must return a value.

#### Example:

```
CREATE FUNCTION GetEmployeeSalary(EmployeeID INT)
RETURNS DECIMAL(10,2)
BEGIN
    DECLARE salary DECIMAL(10,2);
    SELECT Salary INTO salary FROM Employee WHERE ID = EmployeeID;
    RETURN salary;
END;
```

### 33. What are triggers in DBMS? Provide an example.

---

A **trigger** is a special kind of stored procedure that automatically executes (or "fires") in response to certain events on a table, such as insertions, updates, or deletions. Triggers are used to enforce business rules, maintain consistency, or log changes.

#### Example:

```
CREATE TRIGGER after_student_insert
AFTER INSERT ON Student
FOR EACH ROW
BEGIN
    INSERT INTO AuditLog (Action, StudentID, ActionTime)
    VALUES ('INSERT', NEW.ID, NOW());
END;
```

### 34. What is the difference between a trigger and a stored procedure?

---

#### Trigger:

- A **trigger** is an automatic action executed by the DBMS when a specific event occurs on a table, such as an `INSERT` , `UPDATE` , or `DELETE` .
- It cannot be invoked manually and is tied to a specific event.

#### Stored Procedure:

- A **stored procedure** is a precompiled set of SQL statements that can be executed explicitly by a user or application.
- It is invoked manually, and it can accept input parameters and return output.

### 35. What is the difference between UNION and UNION ALL in SQL?

---

- **UNION**: Combines the result of two queries and removes duplicate rows.
- **UNION ALL** : Combines the result of two queries but does **not** remove duplicates, thus it is faster than `UNION` .

#### Example:

```
SELECT Name FROM Students
UNION
SELECT Name FROM Teachers;  # Removes duplicates

SELECT Name FROM Students
UNION ALL
SELECT Name FROM Teachers;  # Does not remove duplicates
```

### 36. What is a normalization form? Explain different normalization forms.

---

Normalization is the process of structuring a database to reduce redundancy and ensure data integrity. It works by splitting large tables into smaller, related tables while maintaining relationships.. There are several **normal forms (NF)**:

- **1NF (First Normal Form):**



- Each column should hold atomic (indivisible) values, and each row must be unique.
- Example: Instead of storing multiple phone numbers in a single column as 123, 456, we store them in separate rows.
- **2NF (Second Normal Form):**
  - A table is in 1NF and all non-key attributes must be fully dependent on the **entire** primary key (not part of it).
  - Example: If we have a table with (StudentID, CourseID) as the key and an attribute StudentName, then StudentName depends only on StudentID, not on the whole key. This violates 2NF. Splitting into Student(StudentID, Name) and Enrollment(StudentID, CourseID) fixes it.
- **3NF (Third Normal Form):**
  - A table is in 2NF and there should be no **transitive dependency** i.e. non-key attribute depending on another non-key attribute.
  - Example:
    - Let's take the table (Professor, Subject, Department).
    - Now, based on business rules:
      - A professor teaches exactly one subject. So we have Professor → Subject.
      - Each subject belongs to a department. So we also have Subject → Department.
    - From this, we can see that Professor is the candidate key.
    - But here's the issue: Department doesn't depend directly on Professor. Instead, it depends on Subject, and Subject depends on Professor. This creates a **transitive dependency**: Professor → Subject → Department.
    - Since 3NF requires that every non-key attribute must depend directly on the primary key and Department is depending indirectly, this violates 3NF.
    - To fix this, we split the table into two:
      - A Professor table with (Professor, Subject), which captures the dependency Professor → Subject.
      - A Subject table with (Subject, Department), which captures Subject → Department.

- Now every non-key attribute depends directly on the key in its own table, so the design is in 3NF.
- **BCNF (Boyce-Codd Normal Form):**
  - It's a stricter version of 3NF. It ensures that for every functional dependency  $X \rightarrow Y$ ,  $X$  is a candidate key.
  - Sometimes 3NF and BCNF **produce the same decomposition**, but **not always**.

### 37. What is the difference between INNER JOIN and OUTER JOIN?

Aspect	INNER JOIN	OUTER JOIN
Result Set	Returns only matching rows from both tables.	Returns all rows from one or both tables, with NULL where no match is found.
Types	Single type.	Three types: LEFT, RIGHT, FULL.
Use Case	When you need only the intersecting data.	When you need to preserve all data, even with mismatches.
Performance	Generally faster as it deals with fewer rows.	Can be slower due to handling more rows and NULL values.
Reliability	Reliable for matching data across tables.	Reliable when you need to retain all data, but can introduce NULL-related issues.

### 38. What is data redundancy in a database? How can it be reduced?

**Data redundancy** refers to the unnecessary repetition of data in a database. It can lead to inconsistencies, increased storage requirements, and maintenance challenges.

#### Reduction methods:

- **Normalization:** Splitting large tables into smaller ones to eliminate redundancy.
- **Eliminating duplicate data:** Using constraints like UNIQUE and PRIMARY KEY to enforce data consistency.

### 39. What is a deadlock in DBMS? How can it be prevented?

---

A **deadlock** occurs when two or more transactions are blocked because each transaction is waiting for the other to release resources. This results in a situation where none of the transactions can proceed.

#### Prevention techniques:

- **Lock ordering:** Enforces a consistent order in which transactions acquire locks to avoid circular wait conditions.
- **Timeouts:** Roll back a transaction automatically if it waits beyond a specified time for a resource.
- **Deadlock detection:** Regularly check for deadlock cycles and terminate one of the involved transactions to release resources.

### 40. What is a database cursor? How is it used?

---

A cursor in DBMS is essentially a pointer that allows us to fetch and process rows from a query result one by one. Normally, SQL works on sets of data at once, but sometimes we need row-by-row operations and that is where cursors are useful. There are two main types:

- **Implicit cursors:** These are created automatically by the database when we run simple queries like `SELECT` , `INSERT` , `UPDATE` , or `DELETE` . For example, in SQL, whenever we run a `SELECT INTO` statement, Oracle internally uses an implicit cursor.
- **Explicit cursors:** These are declared and controlled by the programmer when we need more control over how results are fetched. For example, if we want to loop through employees and give each one a bonus row by row, we can declare and use an explicit cursor.
  - **Example:** `DECLARE cursor_example CURSOR FOR SELECT * FROM Employee;`
  - This statement declares an **explicit cursor** called `cursor_example` . It's basically telling the database:
    - *I want to create a pointer that can iterate over the result set produced by `SELECT * FROM Employee`*

- At this point, no data is fetched yet and we've only defined the cursor. To actually use it, we need to:
  1. **OPEN** the cursor → this runs the query and makes the result set available.
  2. **FETCH** rows → one by one (or in batches) from the cursor into variables.
  3. **CLOSE** the cursor → release the memory once processing is done.
- **Example:**

```
OPEN cursor_example;
FETCH NEXT FROM cursor_example;
-- process row here
CLOSE cursor_example;
```

#### 41. What is a lock. Explain the major difference between a shared lock and an exclusive lock during a transaction in a database?

---

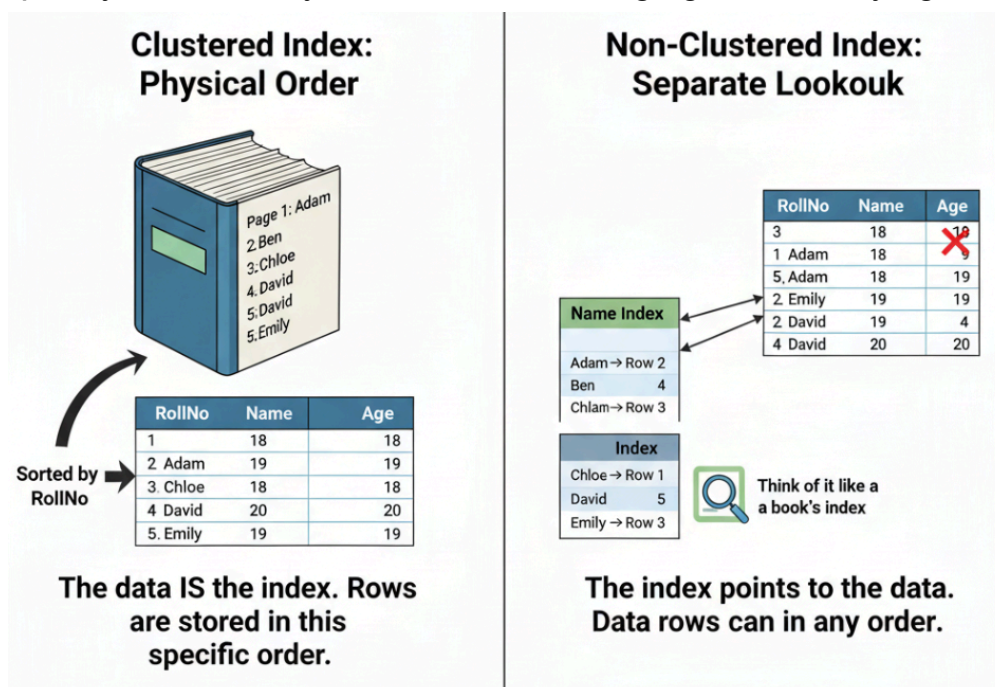
A lock in a database is basically a mechanism used to ensure data consistency and integrity when multiple transactions are running concurrently. It prevents two or more transactions from updating the same data at the same time, which could otherwise lead to anomalies. Now, there are two main types of locks:

- **Shared Lock (S-Lock):** This is used when a transaction only wants to *read* a data item. Multiple transactions can hold a shared lock on the same piece of data at the same time, since reading doesn't change the value. For example, if two users are reading the balance of an account, they can both hold shared locks simultaneously.
- **Exclusive Lock (X-Lock):** This is required when a transaction wants to *write* or update a data item. Only one transaction can hold an exclusive lock at a time, and while it's held, no other transaction can either read or write that data. This ensures we don't get inconsistent results. For example, if I'm updating an account balance, no one else can even read that record until my transaction is completed and the lock is released.

So, the major difference is: **shared locks allow concurrent reads but no writes, while exclusive locks allow only one transaction to both read and write the data.**

#### 42. What is the difference between a clustered and non-clustered index?

- A **clustered index** determines the **physical order** of data in a table. In other words, the rows are actually stored on disk in the same order as the clustered index. That's why a table can have only **one clustered index**, because data rows can be physically ordered in only one way.
- A non-clustered index, on the other hand, is a **separate structure** that stores the index key values along with pointers (row locators) to the actual data rows in the table. Since it doesn't affect the physical order of the data, a table can have **multiple non-clustered indexes** to support different query patterns.
- For example: In a `Students` table, if we make a clustered index on `RollNo`, the table is physically sorted by roll number. If we also create a non-clustered index on `Name`, the database builds a separate lookup structure that helps quickly find rows by name without changing the underlying order of storage.



#### 43. What is the importance of the COMMIT and ROLLBACK operations?

- **COMMIT:** Saves all changes made during the current transaction to the database permanently.
- **ROLLBACK:** Reverses all changes made during the current transaction, restoring the database to its previous state.
- Both operations ensure the **ACID** properties of transactions: **Atomicity** and **Durability**.

#### 44. What is the difference between a superkey and a candidate key?

- 
- **Superkey:** A set of one or more attributes that can uniquely identify a row in a table. It may contain unnecessary attributes.
  - **Candidate Key:** A minimal superkey that uniquely identifies a row, with no redundant attributes. A table can have multiple candidate keys, and one is chosen as the **Primary Key**.

#### 45. Explain the difference between a primary key and a unique key.

---

##### **Primary Key:**

- Uniquely identifies each record in a table.
- Cannot contain `NULL` values.
- A table can have only **one** primary key.

##### **Unique Key:**

- Ensures that all values in a column (or a set of columns) are unique across all rows.
- Can contain `NULL` values (unlike a primary key).
- A table can have **multiple** unique keys.

#### 46. What is referential integrity in DBMS?

---

**Referential Integrity** ensures that relationships between tables are maintained correctly. It requires that the foreign key in one table must match a primary key or a unique key in another table (or be `NULL`). This ensures that data consistency is maintained, and there are no orphan records in the database.

**Example:** In the `Orders` table, if the `CustomerID` is a foreign key, it should match a valid `CustomerID` in the `Customers` table or be `NULL`.

#### 47. How does DBMS handle concurrency control?

---

**Concurrency control** ensures that multiple transactions running at the same time do not cause conflicts or lead to inconsistent data. A DBMS uses several techniques to manage concurrency:

- **Locking:** Transactions acquire locks on data items to control access.
  - **Shared Lock (S-lock):** Allows multiple transactions to read the same data but prevents writes.
  - **Exclusive Lock (X-lock):** Allows one transaction to both read and write the data, blocking others from accessing it.
- **Timestamp Ordering:** Each transaction is assigned a unique timestamp. The DBMS uses these timestamps to determine the order of execution, ensuring consistency.
- **Optimistic Concurrency Control:** Transactions execute without acquiring locks. Before commit, the DBMS checks for conflicts; if any are found, the transaction is rolled back.
- **Two-Phase Locking (2PL):** Transactions go through two phases:
  - **Growing phase:** Locks are acquired but not released.
  - **Shrinking phase:** Locks are released but no new ones are acquired.
    - This guarantees **serializability** and helps avoid conflicts.

#### 48. What are the advantages of using foreign keys in DBMS?

---

- **Enforcing Data Integrity:** Ensures that the value of a foreign key matches a valid primary key or unique key, maintaining consistency.
- **Referential Integrity:** Prevents orphaned records in the DB by enforcing valid relationships between tables.
- **Easy Data Maintenance:** Helps with cascading updates and deletions, meaning changes in the referenced table can automatically propagate to the referencing table (if configured with `ON UPDATE CASCADE` or `ON DELETE CASCADE` ).
- **Improved Query Efficiency:** With foreign keys, database queries that join related tables are more efficient and meaningful.

#### 49. What is a transaction log in DBMS?

---

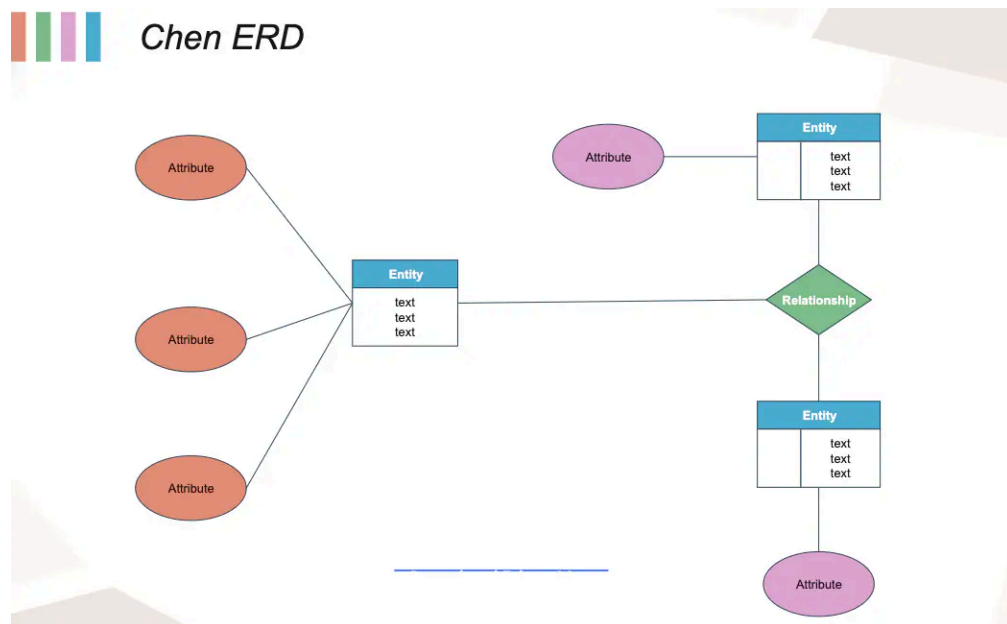
A **transaction log** is a record that keeps track of all transactions executed on a database. It ensures that changes made by transactions are saved, and in case of a system failure, the log can be used to recover the database to its last consistent state. The transaction log contains:

- The details of each transaction (e.g., start, commit, rollback).
- Information about data modifications (insertions, updates, deletions).
- Details about the data before and after the change.

50. **What are the differences between an ER diagram and a relational schema?**

### Entity-Relationship Diagram (ERD):

- A **conceptual blueprint** that models entities, relationships, and attributes of the database. It visually represents the structure of the database.
- Used in the database design phase to understand how data entities relate to each other.



- Example:

### Relational Schema:

- A **logical schema** that defines the structure of a relational database, including tables, columns, relationships, and constraints.
- Represents how data is physically organized in tables with constraints such as primary keys, foreign keys, and data types.
- Example:

```
STUDENTS (StudentID, FirstName, LastName, DateOfBirth, MajorID)
COURSES (CourseID, CourseName, Credits, Department)
ENROLLMENTS (EnrollmentID, StudentID, CourseID, Grade)
MAJORS (MajorID, MajorName, DepartmentHead)
```

51. **What is the purpose of the GROUP BY clause in SQL?**



---

The **GROUP BY** clause is used in SQL to group rows that have the same values in specified columns into summary rows, often with aggregate functions like `COUNT` , `SUM` , `AVG` , `MIN` , or `MAX` . It is typically used to organize data for reporting or analysis.

**Example:** This groups the employees by department and counts the number of employees in each department.

```
SELECT Department, COUNT( * ) FROM Employees GROUP BY Department;
```

## 52. What are the different phases of the DBMS query processing cycle?

---

The DBMS query processing cycle consists of several phases that transform the high-level query (SQL) into executable actions:

- **Parsing:** The SQL query is parsed to check its syntax and semantics. The DBMS ensures that the query is valid according to the SQL syntax and the database schema.
- **Translation:** The query is translated into an internal form, such as a relational algebra expression or an execution plan.
- **Optimization:** The DBMS optimizes the query to determine the most efficient execution plan, considering factors like indexes, joins, and available resources.
- **Execution:** The optimized query plan is executed by the query processor, which accesses the data from the database and returns the results.

## 53. What are the different types of backups in DBMS?

---

There are several types of backups in DBMS:

- **Full Backup** – A complete copy of the entire database (data + schema).
  - Easiest to restore, since it's self-contained.
  - Slow and storage-heavy.
- **Incremental Backup** – Saves **only the changes made since the last backup** (whether that last backup was full or incremental).
  - Very fast and storage-efficient.

- Restore is slower, since you must apply the **full backup + every incremental backup in sequence**.
- **Differential Backup** – Saves **all the changes made since the last full backup** (ignores incremental ones).
  - Easier to restore than incremental → you just need the **last full backup + the latest differential backup**.
  - Larger in size than incremental, because each differential backup keeps growing until the next full backup.
- **Transaction Log Backup** – Backs up the **transaction log** (records of all operations).
  - Allows **point-in-time recovery** (restore the DB to an exact moment before failure).
  - Requires regular log backups to avoid log file bloating.

#### 54. What is the use of the "WITH CHECK OPTION" in SQL views?

---

The **"WITH CHECK OPTION"** is used when creating a view in SQL to ensure that any insert or update operation on the view must adhere to the conditions defined in the view's WHERE clause. If the inserted or updated data violates these conditions, the operation will be rejected.

**Example:** Here, if a user tries to insert or update a `Student` record with a status other than 'Active', the operation will fail.

```
CREATE VIEW ActiveStudents AS
SELECT * FROM Students WHERE Status = 'Active'
WITH CHECK OPTION;
```

Note: **By default** → Views are used mainly for **reading/querying data**, but in many cases, you **can also update, insert, or delete rows through a view**, provided certain conditions are met.

#### 55. Explain the concept of a B-tree and B+ tree in DBMS.

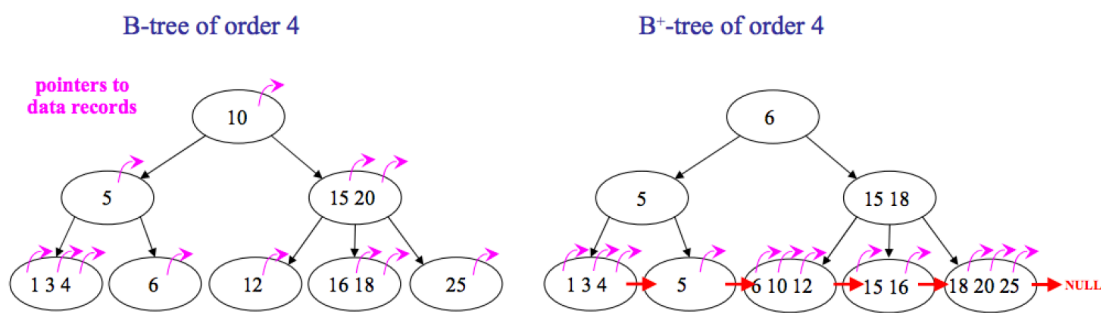
---

**B-tree (Balanced Tree):**

- A **B-tree** is a self-balancing tree data structure that maintains sorted data and allows searches, insertions, deletions in logarithmic time.
- B-trees are used in databases and file systems to store large amounts of data. All nodes in a B-tree can have multiple children, which increases the efficiency of searching.
- Stores data in both internal and leaf nodes.

#### **B+ tree:**

- A **B+ tree** is an extension of the B-tree and is widely used in databases for indexing. It differs in the fact that it has a linked list at the leaf level and stores all records in the leaf nodes.
- The internal nodes of the B+ tree store only keys and pointers to the next level of the tree, while the leaf nodes contain actual data or pointers to the data.
- Stores data only in the leaf nodes and uses the internal nodes for indexing.



### 56. What is a hashing technique in DBMS? How does it work?

A **hashing technique** in DBMS is used to map data (such as a key) to a fixed-size value or address, using a hash function. It is primarily used for quick data retrieval, particularly in hash tables. **Example:** A hash function might map a StudentID of 123 to a bucket index of 3. The student's record would be stored in the corresponding bucket.

#### **How it works:**

- A hash function takes the key (e.g., a record's ID) and calculates a **hash value**.
- This hash value determines the **bucket** or **slot** where the data is stored.
- When searching for a record, the hash function is applied again to the key to find the corresponding bucket.

### 57. Explain the concept of data partitioning in DBMS.

**Data partitioning** is the process of dividing large datasets into smaller, more manageable segments (partitions) to improve performance, scalability, and availability. Each partition can be stored and processed separately.

Types of partitioning:

- **Horizontal Partitioning:** Divides data by rows. For example, splitting data by time range (e.g., 2020 data in one partition, 2021 in another).
- **Vertical Partitioning:** Divides data by columns. For example, putting frequently accessed columns in one partition and less frequently accessed columns in another.
- **Range Partitioning:** Data is divided based on a range of values (e.g., age groups, date ranges).
- **Hash Partitioning:** Data is distributed across partitions based on a hash value derived from a key column.

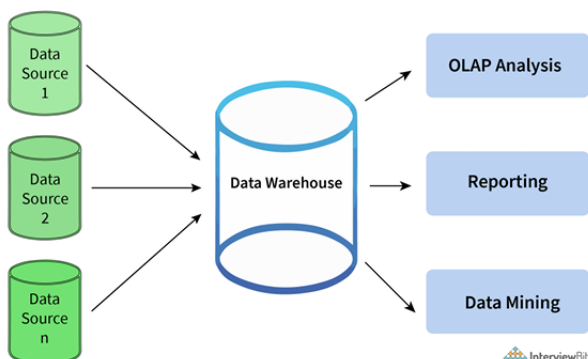
## 58. How is DBMS different from a file-based system?

Aspect	DBMS	File-based System
<b>Data Organization</b>	Data is stored in tables (relations) with structured schemas, supporting complex queries and relationships.	Data is stored in flat files without any relationship between data.
<b>Data Redundancy</b>	Minimizes redundancy through normalization.	Data redundancy is common as data may be duplicated across different files.
<b>Data Integrity</b>	Ensures data integrity through constraints (e.g., primary keys, foreign keys, and check constraints).	Data integrity is hard to enforce, as files don't have built-in integrity checks.
<b>Security</b>	Provides advanced security features like user authentication, access control, and encryption.	Security is managed at the file system level, which is less robust than DBMS security features.
<b>Concurrency Control</b>	Handles concurrent access using locking mechanisms, ensuring data consistency.	No built-in concurrency control; data corruption may occur when multiple users access the same file.

Aspect	DBMS	File-based System
<b>Data Access</b>	Supports complex querying and transaction management (e.g., SQL).	Data access is limited to basic file operations like reading, writing, and appending.
<b>Backup and Recovery</b>	Automatic backup and recovery mechanisms, often integrated into the system.	Backup and recovery mechanisms are manual and may not be as robust.
<b>Scalability</b>	Easily scalable to handle large datasets and multiple users.	Not as scalable; managing large amounts of data and multiple users is difficult.
<b>Maintenance</b>	Centralized maintenance, with tools for optimization, tuning, and troubleshooting.	Maintenance is usually handled at the file system level, which may require manual intervention.
<b>Transaction Management</b>	Supports ACID (Atomicity, Consistency, Isolation, Durability) properties for reliable transaction management.	No built-in support for transactions or ACID properties, making it prone to errors during data modification.

## 59. What is meant by Data Warehousing?

The process of collecting, extracting, transforming, and loading data from multiple sources and storing them into one database is known as data warehousing. A data warehouse can be considered as a central repository where data flows from transactional systems and other relational databases and is used for data analytics. A data warehouse comprises a wide variety of organization's historical data that supports the decision-making process in an organization.



## 60. Explain different levels of data abstraction in a DBMS

---

DBMS provides different levels of abstraction to hide unnecessary details from the user and make interaction with data easier.

- **Physical Level:** This is the lowest level of abstraction. It describes how data is physically stored on disk; things like file structures, indexing, and storage details. These details are managed by the DBMS and hidden from developers and users.
  - **Logical or Conceptual Level:** This is the middle level where developers and database administrators work. It defines *what data* is stored in the database and the relationships among them, without worrying about physical storage. For example, it describes tables, attributes, and constraints.
  - **External or View Level:** This is the highest level, closest to end users. It presents only a part of the database as needed, hiding schema details and storage. For example, a `Customer_View` might show just `Name` and `Email` from the `Customers` table, even though the table has many more columns. A query result is also an example of view-level abstraction.
- So in short: **Physical** → **how data is stored**, **Logical** → **what data and relations exist**, **External** → **how data is presented to users**.

#### 61. What is the difference between Intension and Extension in a database?

---

In a database, we distinguish between **intension** and **extension**:

- **Intension (Schema):** It refers to the **blueprint** of the database. It defines the schema — the tables, attributes, and relationships. Intension is usually specified during database design and remains largely unchanged over time. For example, saying a `Students` table has attributes `(RollNo, Name, Age)` is part of the intension.
  - **Extension (Instance):** It refers to the **actual data** present in the database at a given point in time. Since data changes due to insert, update, or delete operations, the extension keeps changing. For example, the actual rows inside the `Students` table today form its extension — which may be different tomorrow.
- So, simply put: **Intension = schema (fixed design)**, **Extension = data (dynamic content at a snapshot in time).**

#### 62. Explain about types of anomalies ?

---

There are **3 main types of anomalies** that occur when a database is not normalized:

- **Insertion Anomaly**

- Problem: You **cannot insert new data** unless some other unrelated data is present.
- Example: In a table storing (StudentName, CourseName, Instructor), if a new course is introduced but no student has enrolled yet, you cannot insert the course without putting a dummy student.

- **Update Anomaly**

- Problem: If data is **stored redundantly**, updating it in one place but not in another leads to **inconsistencies**.
- Example: If a professor's office number is stored in multiple rows (for each student in the course), and the professor changes office, you must update all rows. Missing one causes inconsistent data.

- **Deletion Anomaly**

- Problem: When deleting some data, you also **lose unintended information**.
- Example: If the last student enrolled in a course drops it and you delete that row, you also lose information about the course itself.

These anomalies are the **reason normalization exists** (1NF → 2NF → 3NF ...) to reduce redundancy and maintain data integrity.

### 63. Explain the difference between a 2-tier and 3-tier architecture in a DBMS

---

- **2-Tier Architecture:**

- This is basically a **client-server setup**. The client directly interacts with the database server. There's no middleware. So, the application logic and database logic are tightly coupled.
- For example, in a simple Railway Reservation System using MS Access, the client directly sends SQL queries to the database and gets results back.

```
Client (UI + Application Logic)
    |
    | SQL Queries
```

v  
Database Server

- **3-Tier Architecture:**

- Here, we add a **middle layer**, usually called the **application server**. This separates the presentation (client) from the database logic. The client interacts with the application, and the application interacts with the database.
- This makes the system more **secure, scalable, and maintainable**, because clients never directly talk to the database.
- A real-world example would be a university registration website – the browser (client) interacts with the application server, which then queries the database.

```
Client (UI Layer)
  |
  v
Application Server (Business Logic Layer)
  |
  v
Database Server (Data Layer)
```

#### 64. What is the difference between SQL and NoSQL databases ?

---

The key difference between **SQL** and **NoSQL** databases lies in their **data model, schema, and scalability approach**.

- **SQL Databases (Relational):**

- Store data in **tables with rows and columns**.
- Follow a **fixed schema** (data structure must be defined in advance).
- Support **ACID properties** (Atomicity, Consistency, Isolation, Durability) → reliable for transactions.
- Use **SQL (Structured Query Language)** for queries.
- Best for structured data and complex relationships.
- **Example:** MySQL, PostgreSQL, Oracle DB.

- **NoSQL Databases (Non-relational):**

- Stores data in flexible formats such as **documents (MongoDB)**, **key-value pairs (Redis)**, **wide-column stores (Cassandra)**, or **graphs**



(Neo4j).

- **Schema-less or dynamic schema** → structure can evolve.
- Designed for **horizontal scalability** (distributed systems, big data).
- Best for unstructured or semi-structured data.
- **Example:** MongoDB, Cassandra, Redis, Neo4j.

## 65. What is the difference between PostgreSQL and SQL ?

---

The key difference between **PostgreSQL** and **MySQL** lies in their **data model, standards compliance, performance focus, and extensibility**.

- **PostgreSQL (Object-Relational Database)**
  - **Open-source ORDBMS** → Unlike MySQL, which is purely relational, PostgreSQL is “object-relational.” This means it not only supports traditional tables and relations, but also advanced data types and custom extensions.
  - **Relational + Object-relational models** → PostgreSQL allows columns to store arrays, JSON, geospatial data (PostGIS), etc. MySQL supports JSON, but PostgreSQL integrates it deeply (e.g., indexing JSON fields with GIN).
  - **Highly standards-compliant** → PostgreSQL sticks very closely to ANSI SQL standards and implements advanced features (CTEs, recursive queries, window functions). MySQL often skips these or implements partial versions for simplicity.
  - **Fully ACID-compliant by default** → PostgreSQL enforces strict transaction guarantees out of the box. MySQL’s default engine (InnoDB) is also ACID-compliant, but older engines like MyISAM are not, which means PostgreSQL is stricter by design.
  - **Advanced indexing support** → PostgreSQL offers B-tree, Hash, GiST, GIN, BRIN, etc. MySQL mostly sticks to B-tree and hash indexes. These advanced indexes allow PostgreSQL to handle text search, geospatial queries, and analytics more efficiently.
  - **Extensibility** → PostgreSQL lets developers create custom data types, operators, and even write stored procedures in multiple languages (PL/pgSQL, Python, C). MySQL is far less flexible in this regard.
  - **Use cases** → PostgreSQL is chosen where data is **complex** and requires **heavy analytics** (finance, GIS, research, enterprise systems).

- **MySQL (Relational Database)**

- **Open-source RDBMS** → Purely relational, simpler than PostgreSQL. Designed for fast reads and simplicity.
- **Primarily relational** → Supports JSON, but without advanced indexing and functions like PostgreSQL. Used mainly for traditional row-column data.
- **Practical approach (performance over standards)** → MySQL sometimes ignores strict SQL standards to improve usability and speed. PostgreSQL is stricter but may be heavier.
- **Storage engines:**
  - **InnoDB (default)** → ACID-compliant, supports transactions, foreign keys, row-level locking → reliable for transactional systems.
  - **MyISAM (older)** → Non-ACID, no transactions, table-level locking → good for read-heavy workloads, but unsafe against crashes. PostgreSQL doesn't use this concept; it has one consistent storage system.
- **Indexing** → Limited mostly to B-tree and hash indexes. This makes it faster for simple queries, but less powerful for specialized queries compared to PostgreSQL.
- **Use cases** → MySQL is widely used in **web apps, CMS (WordPress, Shopify), and e-commerce** where workloads are mostly read-heavy and schema is simple