1. DBMS Architecture

There are three main types of DBMS architectures:

Single-tier architecture: In a single-tier architecture, all the components of the DBMS reside on a single server. This is the simplest type of DBMS architecture and is easy to set up and manage. However, it is not scalable and can only handle a limited number of users.

Two-tier architecture: In a two-tier architecture, the DBMS is divided into two layers: the presentation layer and the data layer. The presentation layer resides on the client computer and is responsible for interacting with the user. The data layer resides on the server computer and is responsible for storing and managing the data. This architecture is more scalable than the single-tier architecture and can handle a larger number of users.

Three-tier architecture: In a three-tier architecture, the DBMS is divided into three layers: the presentation layer, the application layer, and the data layer. The presentation layer resides on the client computer and is responsible for interacting with the user. The application layer resides on a separate server computer and is responsible for processing the user requests and communicating with the data layer. The data layer resides on a separate server computer and is responsible for storing and managing the data. This architecture is the most scalable and can handle a very large number of users.

2. Advantages and Disadvantages of DBMS

A Database Management System (DBMS) is software that allows users to create, retrieve, update, and manage data in a structured and organized way. There are several advantages and disadvantages to using a DBMS:

Advantages of DBMS:

Data Integrity and Accuracy: DBMS systems enforce data integrity constraints, ensuring that the data stored in the database is accurate and consistent. This reduces the risk of data anomalies and errors.

Data Security: DBMS provides security features like user authentication and authorization, ensuring that only authorized users can access and modify the data. It also offers encryption options to protect sensitive data.

Data Centralization: All data is stored in a centralized location, making it easier to manage and maintain. This reduces data duplication and ensures data consistency.

Concurrency Control: DBMS systems can handle multiple users accessing and modifying data simultaneously while maintaining data consistency through mechanisms like locking.

Data Redundancy Reduction: By centralizing data storage, DBMS reduces data redundancy, which helps in saving storage space and ensures data consistency.

Data Retrieval: DBMS provides query languages (e.g., SQL) that make it easier to retrieve specific data from the database, even when dealing with large datasets.

Data Backup and Recovery: Most DBMS systems offer backup and recovery features, ensuring that data can be restored in case of system failures or data corruption.

Scalability: DBMS systems can be scaled vertically (adding more resources to a single server) or horizontally (adding more servers to a cluster) to accommodate growing data needs.

Data Independence: DBMS separates the logical structure of the database from its physical storage, providing data independence. This means that changes to the database schema do not affect application programs.

Disadvantages of DBMS:

Cost: Implementing and maintaining a DBMS can be expensive. This includes the cost of software licenses, hardware, training, and ongoing maintenance.

Complexity: DBMS systems can be complex to set up and manage, especially for large-scale databases. Skilled database administrators are often required.

Performance Overhead: DBMS systems introduce some performance overhead due to tasks like query optimization, transaction management, and data indexing.

Vendor Lock-In: Choosing a specific DBMS vendor can lead to vendor lock-in, making it difficult and costly to switch to another system.

Learning Curve: Developers and administrators need to learn the specific features and query languages of the chosen DBMS, which can have a steep learning curve.

Scalability Challenges: While DBMS systems are scalable, achieving high levels of scalability may require significant resources and expertise.

Single Point of Failure: If not properly designed for fault tolerance, a DBMS can become a single point of failure, leading to potential data loss in case of system crashes.

Resource Intensive: DBMS systems can consume a significant amount of CPU, memory, and storage resources, which can be a concern for resource-constrained environments.

Limited Support for Complex Data: Some DBMS systems may not handle complex data types, such as unstructured or semi-structured data, as effectively as other technologies like NoSQL databases.

3. ER DIAGRAMS (STUDENT, HOSPITAL, BANK, EMPLOYEES)

4. **Codd’s Rules**

Codd's Rules, also known as Codd's Twelve Commandments, were formulated by Dr. Edgar F. Codd, a computer scientist and researcher, in the early 1970s. These rules served as a set of principles and criteria to define what constitutes a relational database management system (RDBMS). RDBMS systems, which adhere to Codd's Rules, are designed to ensure data integrity, consistency, and ease of use. Here are Codd's 12 Rules for RDBMS:

Information Rule: All information in the database should be stored in tables, which are the logical data structures representing entities.

Guaranteed Access Rule: Each piece of data (atomic value) in the database should be accessible by specifying a table name, primary key value, and a column name. This ensures precise and direct access to data.

Systematic Treatment of Null Values: The DBMS must allow each field to remain null (or empty), which represents missing or unknown information. It should also distinguish between null values and empty strings or zeros.

Dynamic Online Catalog Based on The Relational Model: The structure (schema) of the database, including table definitions, should be stored in the database itself. Users should be able to query and modify the schema using the same query language as for data retrieval and manipulation.

Comprehensive Data Sublanguage Rule: The DBMS should provide a comprehensive and consistent query language for defining, manipulating, and querying the data. SQL (Structured Query Language) is the most commonly used language for this purpose.

View Updating Rule: All views that are theoretically updatable should be updatable by the system. Views are virtual tables that allow users to work with a subset of the data.

High-Level Insert, Update, and Delete: The DBMS should support high-level (declarative) operations for inserting, updating, and deleting data, allowing users to specify what data they want to manipulate without needing to specify the exact data manipulation procedures.

Physical Data Independence: The physical storage structure of the database (e.g., disk storage or file organization) should be separate from the logical structure (tables and relationships). Changes to the physical storage should not affect the application programs.

Logical Data Independence: Changes to the logical structure of the database (schema) should not affect the existing application programs. This supports the idea that applications are independent of the database structure.

Integrity Independence: The DBMS should provide mechanisms to enforce data integrity constraints, such as primary key, foreign key, and check constraints, independently of the application programs.

Distribution Independence: The DBMS should be able to distribute data across multiple physical locations, whether on the same computer or across a network of computers, without changing the application programs.

Non-Subversion Rule: If the DBMS has a low-level language (such as assembly language) for writing extensions or custom functions, users should not be able to use it to subvert or bypass the integrity rules and security constraints of the system.

5. Relational model

The relational model is a fundamental concept in database management that was introduced by Dr. Edgar F. Codd in a seminal paper published in 1970. It serves as the foundation for organizing and managing data in relational database management systems (RDBMS). The relational model defines data as a set of tables (relations) with a well-defined structure and enforces principles of data integrity, consistency, and simplicity. Here are the key components and concepts of the relational model:

Tables (Relations): The central concept of the relational model is the table, which is also referred to as a relation. Tables are used to represent entities (e.g., customers, products, orders) and their attributes (e.g., names, IDs, dates) in a structured manner. Each table consists of rows and columns, with each row representing a specific record or tuple, and each column representing a specific attribute or field.

Attributes (Fields): Columns in a table represent attributes or fields that describe the properties or characteristics of the entities being modeled. Each attribute has a data type that defines the kind of data it can store (e.g., integer, string, date).

Tuples (Records or Rows): Rows in a table are called tuples or records. Each tuple represents a specific instance or record of the entity being modeled. Tuples contain values for each of the attributes defined by the table's schema.

Keys: Keys are attributes (or combinations of attributes) that uniquely identify each tuple within a table. The primary key is the main key used for uniqueness, and it ensures that no two tuples in the table have the same key values. Foreign keys establish relationships between tables by referencing the primary key of another table.

Domain: A domain defines the set of valid values for an attribute. It specifies the data type and any constraints (e.g., minimum and maximum values) that apply to the attribute.

Relationships: Relationships in the relational model represent connections between tables. These relationships are typically established using foreign keys. For example, in a relational database for a library, a "books" table might have a foreign key that references the "authors" table to establish a relationship between books and authors.

Normalization: Normalization is a process in the relational model that reduces data redundancy and improves data integrity. It involves breaking down large tables into smaller, related tables to eliminate data anomalies like insertion, update, and deletion anomalies.

ACID Properties: The relational model enforces the ACID (Atomicity, Consistency, Isolation, Durability) properties to ensure data integrity and consistency, especially in transactional systems. These properties guarantee that database transactions are reliable and maintain the integrity of the data.

Structured Query Language (SQL): SQL is the standard language for querying and manipulating data in relational databases. It provides commands for creating, retrieving, updating, and deleting data in tables. SQL allows users to express complex queries and transactions in a declarative, SQL-based language.

Data Integrity: The relational model enforces data integrity constraints such as primary key constraints (uniqueness), foreign key constraints (referential integrity), check constraints (validating data values), and domain constraints (valid data types).

6. **Normalization (all forms)**

Normalization is a database design technique used to organize data in a relational database management system (RDBMS) efficiently and reduce data redundancy while ensuring data integrity and consistency. The normalization process involves breaking down large tables into smaller, related tables and defining relationships between them. The primary goal of normalization is to eliminate data anomalies that can occur during data insertion, update, and deletion. It is typically carried out in a series of steps, each represented by a normal form.

There are several normal forms, each building on the previous one. The most commonly discussed normal forms are the First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boyce-

Codd Normal Form (BCNF), and Fourth Normal Form (4NF). Let's discuss these normal forms in detail:

First Normal Form (1NF):

In 1NF, each table must have a primary key, and all columns in the table must contain atomic (indivisible) values. This means that each cell in the table should contain a single, indivisible piece of data.
1NF ensures that there is no repetition of groups of related data within a row.
Second Normal Form (2NF):

A table is in 2NF if it is in 1NF and if all non-key attributes are fully functionally dependent on the entire primary key (no partial dependencies).
This means that a table should not have attributes that depend on only a portion of the primary key; they should depend on the entire key.
Third Normal Form (3NF):

A table is in 3NF if it is in 2NF, and all non-key attributes are functionally dependent on the primary key (no transitive dependencies).
Transitive dependencies occur when an attribute depends on another attribute that, in turn, depends on the primary key.
Boyce-Codd Normal Form (BCNF):

BCNF is an extension of 3NF. A table is in BCNF if, for every non-trivial functional dependency $X \rightarrow Y$, X is a superkey.
A superkey is a set of attributes that can uniquely identify a tuple in the table.
Fourth Normal Form (4NF):

4NF addresses multi-valued dependencies. A table is in 4NF if, for every non-trivial multi-valued dependency $X \twoheadrightarrow Y$, X is a superkey.
Multi-valued dependencies occur when an attribute can have multiple values for a single set of values in another attribute.
Normalization involves dividing tables and creating relationships between them to minimize redundancy. By adhering to normalization rules, you can achieve several benefits:

Data Integrity: Normalization helps maintain data integrity by reducing the risk of anomalies such as update anomalies, insertion anomalies, and deletion anomalies.

Efficient Storage: Smaller, well-organized tables require less storage space, leading to more efficient database performance.

Simplified Updates: Normalized tables make it easier to update data consistently and accurately because data is stored in only one place.

7. **Transactions and Properties of Transaction (ACID)**

A database transaction is a unit of work that is executed within a database management system (DBMS). Transactions are essential in ensuring the consistency, integrity, and reliability of data within a database, especially in multi-user environments. Transactions are often associated with a set of properties collectively known as ACID properties:

Atomicity (A): Atomicity is the "all or nothing" property of a transaction. It ensures that a transaction is treated as a single, indivisible unit of work. If any part of a transaction fails, the entire transaction is rolled back, and the database remains unchanged. In other words, either all changes made by the transaction are applied, or none of them are. This property ensures that the database remains in a consistent state.

Consistency (C): Consistency guarantees that a transaction brings the database from one consistent state to another. The database must satisfy a set of integrity constraints before and after the transaction. If a transaction violates any of these constraints, it is rolled back, and the database remains unchanged. Consistency ensures that the data remains accurate and in a valid state throughout the transaction.

Isolation (I): Isolation ensures that multiple transactions can execute concurrently without interfering with each other. Each transaction should be unaware of other concurrent transactions and should perceive the database as if it were the only transaction running. Isolation prevents issues like dirty reads, non-repeatable reads, and phantom reads by controlling the visibility of uncommitted data.

Durability (D): Durability guarantees that once a transaction is committed, its changes are permanent and will survive any subsequent failures, such as system crashes or power outages. The DBMS ensures that all changes made by a committed transaction are stored safely and can be recovered in the event of a failure. Durability is typically achieved through techniques like write-ahead logging and data persistence.

These ACID properties are crucial for maintaining data integrity and ensuring the reliability of database transactions, especially in scenarios where multiple users or processes access and modify the database simultaneously. Transactions allow applications to handle complex operations involving multiple steps while guaranteeing that the database remains in a consistent and reliable state. ACID compliance is a fundamental requirement for many mission-critical applications, such as financial systems, e-commerce platforms, and healthcare databases, where data accuracy and reliability are of utmost importance.

8. **Joins**

JOINs are used to combine data from two or more tables based on a related column between them. JOINs are fundamental for querying and retrieving data that spans multiple tables in a database. MySQL supports various types of JOINs, including INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN. Here's an overview of these JOIN types:

INNER JOIN:

An INNER JOIN returns only the rows that have matching values in both tables being joined.
Syntax:

```
SELECT columns
FROM table1
INNER JOIN table2 ON table1.column = table2.column;
```

LEFT JOIN (or LEFT OUTER JOIN):

A LEFT JOIN returns all the rows from the left table and the matching rows from the right table. If there are no matching rows in the right table, NULL values are returned for the right table's columns.
Syntax:

```
SELECT columns
FROM table1
LEFT JOIN table2 ON table1.column = table2.column;
```

RIGHT JOIN (or RIGHT OUTER JOIN):

A RIGHT JOIN is the reverse of a LEFT JOIN. It returns all the rows from the right table and the matching rows from the left table. If there are no matching rows in the left table, NULL values are returned for the left table's columns.
Syntax:

```
SELECT columns
FROM table1
RIGHT JOIN table2 ON table1.column = table2.column;
```

9. SET Operators

Set operators in SQL allow you to combine the results of two or more SELECT statements, returning a single result set. The most commonly used set operators are UNION, INTERSECT, and EXCEPT (or MINUS, depending on the database system). Here's an explanation of each set operator with examples and syntax:

UNION Operator:

The UNION operator combines the result sets of two or more SELECT statements into a single result set, removing duplicate rows.
Syntax:

```
SELECT columns FROM table1
UNION
SELECT columns FROM table2;
```

Example:

```
SELECT employee_name FROM employees
UNION
SELECT customer_name FROM customers;
```

INTERSECT Operator (Not supported in MySQL):

The INTERSECT operator returns rows that exist in both result sets of two SELECT statements.
Syntax:

```
SELECT columns FROM table1
INTERSECT
SELECT columns FROM table2;
```

Example

```
SELECT product_name FROM products
INTERSECT
SELECT product_name FROM orders;
```

EXCEPT or MINUS Operator (Not supported in MySQL):

The EXCEPT (or MINUS) operator returns rows that exist in the first result set but not in the second result set.
Syntax:

```
SELECT columns FROM table1
EXCEPT
SELECT columns FROM table2;
```

Example:

```
SELECT department_name FROM departments
EXCEPT
SELECT department_name FROM employees;
```

10.DML Statements

DML (Data Manipulation Language) statements in SQL are used to manipulate and interact with data stored in a database. These statements allow you to insert, retrieve, update, and delete data in database tables. The primary DML statements in SQL are:

SELECT: The SELECT statement is used to retrieve data from one or more tables. It allows you to specify the columns you want to retrieve and apply various filtering and sorting criteria.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

INSERT: The INSERT statement is used to add new rows of data into a table.

Syntax:

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

UPDATE: The UPDATE statement is used to modify existing data in a table.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

DELETE: The DELETE statement is used to remove rows of data from a table.

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

11. TCL (COMMIT, ROLLBACK)

TCL (Transaction Control Language) is a set of SQL commands that allows you to manage transactions in a database. TCL commands are used to control the start, execution, and termination of transactions. The primary TCL commands in SQL are COMMIT, ROLLBACK, and SAVEPOINT. Let's focus on COMMIT and ROLLBACK, which are the most commonly used TCL commands:

COMMIT:

The COMMIT command is used to make permanent all the changes made during the current transaction. Once a transaction is committed, its changes become permanent and cannot be undone.

Syntax:

```
COMMIT;
```

Eg:

```
BEGIN; -- Start a transaction
UPDATE customers SET balance = balance - 100 WHERE customer_id = 101;
INSERT INTO transactions (customer_id, amount) VALUES (101, -100);
COMMIT; -- Commit the transaction
```

ROLLBACK:

The ROLLBACK command is used to undo or cancel all the changes made during the current transaction. It effectively "rolls back" the database to the state it was in before the transaction started.

Syntax:

```
ROLLBACK;
```

Eg:

```
BEGIN; -- Start a transaction
UPDATE customers SET balance = balance - 100 WHERE customer_id = 101;
INSERT INTO transactions (customer_id, amount) VALUES (101, -100);
ROLLBACK; -- Cancel the transaction
```

Here's how transactions work with COMMIT and ROLLBACK:

A transaction begins with the BEGIN or START TRANSACTION command.

Within the transaction, you can execute one or more SQL statements that modify data.
If all the operations in the transaction are successful and you want to make the changes permanent, you use the COMMIT command.
If any issues or errors occur during the transaction, you use the ROLLBACK command to undo the changes and restore the database to its previous state.

12. Primary Key, Foreign Key

Primary Key and Foreign Key are two important concepts in relational databases that establish relationships between tables and enforce data integrity. Here's a detailed explanation of each:

Primary Key:

A Primary Key is a unique identifier for a record (row) in a database table. It ensures that each record in the table is distinct and can be uniquely identified.
Key characteristics of a Primary Key:
Uniqueness: Each value in the Primary Key column(s) must be unique within the table. No two rows can have the same Primary Key value.
Non-null: The Primary Key column(s) cannot contain NULL values. Every row must have a valid Primary Key value.
A Primary Key can consist of one or more columns. When it consists of multiple columns, it is called a composite key.
Typically, the Primary Key is used as a reference for establishing relationships with other tables through Foreign Keys.
Example:

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);
```

Foreign Key:

A Foreign Key is a field (or a set of fields) in one table that is used to establish a link between the data in two tables. It creates a relationship between the tables based on a common column.
Key characteristics of a Foreign Key:
Referential Integrity: The values in the Foreign Key column(s) must match values in the Primary Key column(s) of another table. This ensures

referential integrity, meaning that the data in related tables remains consistent.
Cascade Actions: Foreign Keys can have cascade actions defined, such as ON DELETE CASCADE or ON UPDATE CASCADE, which specify what should happen when a referenced row is deleted or updated.
Foreign Keys are used to enforce data integrity and maintain relationships between tables in a relational database.
Example:

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

13.DDL COMMANDS

DDL (Data Definition Language) commands in SQL are used to define, manage, and control the structure of a database, including tables, schemas, indexes, constraints, and other database objects. DDL commands allow you to create, alter, and drop database objects. Here are some common DDL commands in SQL:

CREATE TABLE:

The CREATE TABLE command is used to create a new table in a database, specifying its columns, data types, constraints, and other attributes.
Syntax:

```
CREATE TABLE table_name (
    column1 data_type constraints,
    column2 data_type constraints,
    ...
);
```

ALTER TABLE:

The ALTER TABLE command is used to modify an existing table, such as adding, modifying, or dropping columns, adding constraints, or changing the table's name.
Syntax (Adding a column):

```
ALTER TABLE table_name
ADD column_name data_type constraints;
```

DROP TABLE:

The DROP TABLE command is used to delete an existing table and all of its data from the database.
Syntax:

```
DROP TABLE table_name;
```

14. Triggers

Triggers in a relational database are special database objects that are automatically executed or fired in response to specific events or actions that occur within the database. Triggers are used to enforce data integrity, implement business rules, log changes, or automate tasks. They are typically associated with tables and are defined to respond to events such as INSERT, UPDATE, DELETE, or even DDL (Data Definition Language) events like CREATE, ALTER, or DROP. Here are the types of triggers in more detail:

AFTER Triggers:

An AFTER trigger is executed after the triggering event has occurred (e.g., after an INSERT, UPDATE, or DELETE operation). It is often used for auditing, logging changes, and enforcing data integrity constraints.
Example uses of AFTER triggers:
Logging changes made to a table.
Automatically updating related data after a change.
BEFORE Triggers:

A BEFORE trigger is executed before the triggering event takes place. It allows you to inspect and potentially modify the data that is about to be inserted, updated, or deleted.
Example uses of BEFORE triggers:
Validating data before it is inserted or updated.

Setting default values for certain columns.

```
CREATE TRIGGER after_insert_example
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO employee_log (employee_id, action, action_date)
    VALUES (NEW.employee_id, 'INSERT', NOW());
END;
```

15. What is Views and Type of Views

A view is a virtual table that is based on the result of a SELECT query. Unlike physical tables, views do not store data themselves; instead, they provide a way to present data from one or more underlying tables in a structured and customizable manner. Views are often used to simplify complex queries, enhance security, and provide a more user-friendly interface to the data.

Here are the types of views in a database:

1. Simple View:

A simple view is based on a single table and presents data directly from that table. It is essentially a virtual table that provides controlled access to the data in the underlying table, allowing you to restrict the columns or rows that users can see.

Syntax for Creating a Simple View:

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

Example of a Simple View:

```
CREATE VIEW employee_names AS
SELECT employee_id, first_name, last_name
FROM employees;

SELECT * FROM employee_names;
```

2. Complex View:

A complex view, on the other hand, is based on multiple tables and may involve joins, subqueries, and other SQL operations to retrieve data from various sources. Complex views are used to combine data from different tables and present it as a unified result set.

Syntax for Creating a Complex View:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table1
JOIN table2 ON join_condition
WHERE condition;
```

Example of a Complex View:

```
CREATE VIEW order_details AS
SELECT o.order_id, c.customer_name, p.product_name, oi.quantity
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id;
```

```
SELECT * FROM order_details;
```