Question 1 : Explain the fundamental differences between DDL, DML, and DQL commands in SQL. Provide one example for each type of command.
Ans.Understanding the distinction between these SQL command types is like understanding the difference between **building a house**, **moving furniture in**, and **looking through the windows**.
Each category serves a specific purpose in the lifecycle of a database.

1. DDL (Data Definition Language)

The Architect. DDL commands are used to define or modify the **structure** (schema) of the database. You aren't touching the actual data entries; you are defining the containers (tables, indexes, etc.) that will hold them.

+1

- **Impact:** Changes are usually permanent and auto-committed.
- **Key Keywords:** `CREATE`, `ALTER`, `DROP`, `TRUNCATE`.
- **Example:** Creating a new table to store user profiles.CREATE TABLE Users (
- UserID int,
- Username varchar(255),
- Email varchar(255)
-);

2. DML (Data Manipulation Language)

The Resident. DML is used to **manage the data within** those structures. Once the table exists, you use DML to add, change, or remove the actual records.

- **Impact:** Affects the rows of a table. These changes can often be rolled back (undone) before they are committed.
- **Key Keywords:** `INSERT`, `UPDATE`, `DELETE`.
- **Example:** Adding a specific person to your `Users` table.

INSERT INTO Users (UserID, Username, Email)
VALUES (1, 'JaneDoe', 'jane@example.com');

3. DQL (Data Query Language)

The Observer. DQL is used solely for **fetching or retrieving** data. It doesn't change the structure or the data itself; it just asks the database to show you specific information based on your criteria.

- **Impact:** Read-only. It has zero effect on the database state.
- **Key Keywords:** `SELECT`.
- **Example:** Finding the email of a user with a specific ID.

```
SELECT Email FROM Users
WHERE UserID = 1;
```

Question 2 : What is the purpose of SQL constraints? Name and describe three common types of constraints, providing a simple scenario where each would be useful.
Ans. **1. DDL (Data Definition Language)**

The Architect. DDL commands are used to define or modify the **structure** (schema) of the database. You aren't touching the actual data entries; you are defining the containers (tables, indexes, etc.) that will hold them.

- **Impact:** Changes are usually permanent and auto-committed.
- **Key Keywords:** `CREATE`, `ALTER`, `DROP`, `TRUNCATE`.
- **Example:** Creating a new table to store user profiles.

2. DML (Data Manipulation Language)

The Resident. DML is used to **manage the data within** those structures. Once the table exists, you use DML to add, change, or remove the actual records.

- **Impact:** Affects the rows of a table. These changes can often be rolled back (undone) before they are committed.
- **Key Keywords:** `INSERT`, `UPDATE`, `DELETE`.
- **Example:** Adding a specific person to your `Users` table.

3. DQL (Data Query Language)

The Observer. DQL is used solely for **fetching or retrieving** data. It doesn't change the structure or the data itself; it just asks the database to show you specific information based on your criteria.

- **Impact:** Read-only. It has zero effect on the database state.
- **Key Keywords:** `SELECT`.
- **Example:** Finding the email of a user with a specific ID.

The Fundamental Difference

Think of a spreadsheet: **LIMIT** tells the database how many rows to "pick up," while **OFFSET** tells the database how many rows to "jump over" before it starts picking them up.

- **LIMIT:** Defines the **maximum number of rows** to return in the result set. It controls the size of your "page."
- **OFFSET:** Defines the **starting point**. It specifies the number of rows to skip from the very beginning of the result set.

Using them for Pagination

To retrieve a specific page of results, you have to do a little bit of "index math." If you want the **3rd page** with **10 records per page**, you need to skip the first two pages.

- **Page 1:** Rows 1–10 (Skip 0)
- **Page 2:** Rows 11–20 (Skip 10)
- **Page 3:** Rows 21–30 (Skip 20)

The Formula:

To calculate the offset for any page, use:

```
OFFSET = (PageNumber - 1) * RecordsPerPage
```

The SQL Query:

To get the 3rd page (skipping 20 records and taking the next 10):

SQL

SELECT * FROM Products

ORDER BY ProductID -- Always use ORDER BY with pagination to ensure consistency

LIMIT 10 OFFSET 20;

Main Benefits of Using CTEs

- **Readability:** It follows a "top-down" logic. You define your data logic first and then use it, making the code much easier for humans to read compared to nested subqueries.
- **Reusability:** You can reference the same CTE multiple times within the same query, preventing you from writing the same logic twice.
- **Maintenance:** If you need to change the logic of your temporary data, you only have to change it in one place (the CTE definition) rather than hunting through multiple subqueries.
- **Recursion:** CTEs have a unique superpower: **Recursive CTEs**. They can reference themselves, which is the standard way to query hierarchical data like organizational charts or family trees.

Simple Example: Calculating Department Bonuses

Imagine you want to find all employees who earn more than the average salary in their company. Without a CTE, you'd use a messy subquery. With a CTE, it's clean:

```
WITH AverageSalaryCTE AS (

   -- This "temporary table" calculates the average once

   SELECT AVG(Salary) as AvgSal

   FROM Employees

)

SELECT EmployeeName, Salary

FROM Employees, AverageSalaryCTE

WHERE Salary > AverageSalaryCTE.AvgSal;
```

Question 5 : Describe the concept of SQL Normalization and its primary goals. Briefly explain the first three normal forms (1NF, 2NF, 3NF).
Ans. **The Primary Goals**

- **Eliminate Redundant Data:** Prevent the same piece of info (like a customer's phone number) from being stored in multiple rows.
- **Ensure Data Dependencies:** Make sure data is stored logically (e.g., a product's price should be in a `Products` table, not an `Orders` table).
- **Prevent Anomalies:** Avoid issues where deleting a record accidentally deletes unrelated information, or updating one record leaves "ghost" versions of the old data elsewhere.

The First Three Normal Forms

1. First Normal Form (1NF): Atomic Values

To satisfy 1NF, your table must have no "multi-valued" attributes. Every cell must contain a single, **atomic** value, and each record must be unique.

- **The Rule:** No lists or arrays inside a single cell.
- *Example:* If a user has two phone numbers, you don't put them in one cell separated by a comma; you create two separate rows or a separate table.

2. Second Normal Form (2NF): No Partial Dependencies

To reach 2NF, you must already be in 1NF. Additionally, all non-key columns must depend on the **entire** primary key.

- **The Rule:** This applies mainly to tables with "Composite Keys" (keys made of two or more columns). You shouldn't have a column that only relates to *part* of that key.
- *Example:* In a `Student_Course` table, the "Course Name" shouldn't be there because it depends only on the `CourseID`, not the `StudentID`. You'd move "Course Name" to its own `Courses` table.

3. Third Normal Form (3NF): No Transitive Dependencies

To reach 3NF, you must be in 2NF. Additionally, non-key columns should not depend on other non-key columns.

- **The Rule:** "The key, the whole key, and nothing but the key."
- *Example:* In an `Employees` table, you might have `ZipCode` and `City`. Since `City` is determined by the `ZipCode` (a non-key column), you have a transitive dependency. You should move `ZipCode` and `City` to a separate location table.

Question 6 : Create a database named ECommerceDB and perform the following tasks:

Ans. **1. Database and Table Creation**

We use **DDL** commands here. Note the order: `Categories` and `Customers` must exist before `Products` and `Orders` can reference them.

```
-- Create the Database

CREATE DATABASE ECommerceDB;

USE ECommerceDB;



-- 1. Categories Table

CREATE TABLE Categories (

    CategoryID INT PRIMARY KEY,

    CategoryName VARCHAR(50) NOT NULL UNIQUE

);



-- 2. Customers Table

CREATE TABLE Customers (

    CustomerID INT PRIMARY KEY,

    CustomerName VARCHAR(100) NOT NULL,

    Email VARCHAR(100) UNIQUE,

    JoinDate DATE

);
```

```
-- 3. Products Table (References Categories)
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100) NOT NULL UNIQUE,
    CategoryID INT,
    Price DECIMAL(10,2) NOT NULL,
    StockQuantity INT,
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);

-- 4. Orders Table (References Customers)
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE NOT NULL,
    TotalAmount DECIMAL(10,2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

2. Data Insertion

Now we use **DML** commands to populate the tables with the records you provided.

Populating Categories and Products

```
INSERT INTO Categories (CategoryID, CategoryName) VALUES

(1, 'Electronics'),

(2, 'Books'),

(3, 'Home Goods'),

(4, 'Apparel');



INSERT INTO Products (ProductID, ProductName, CategoryID, Price, StockQuantity)
VALUES

(101, 'Laptop Pro', 1, 1200.00, 50),

(102, 'SQL Handbook', 2, 45.50, 200),

(103, 'Smart Speaker', 1, 99.99, 150),

(104, 'Coffee Maker', 3, 75.00, 80),

(105, 'Novel: The Great SQL', 2, 25.00, 120),

(106, 'Wireless Earbuds', 1, 150.00, 100),

(107, 'Blender X', 3, 120.00, 60),

(108, 'T-Shirt Casual', 4, 20.00, 300);

INSERT INTO Customers (CustomerID, CustomerName, Email, JoinDate) VALUES
```

(1, 'Alice Wonderland', 'alice@example.com', '2023-01-10'),

(2, 'Bob the Builder', 'bob@example.com', '2022-11-25'),

(3, 'Charlie Chaplin', 'charlie@example.com', '2023-03-01'),

(4, 'Diana Prince', 'diana@example.com', '2021-04-26');

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount) VALUES

(1001, 1, '2023-04-26', 1245.50),

(1002, 2, '2023-10-12', 99.99),

(1003, 1, '2023-07-01', 145.00),

(1004, 3, '2023-01-14', 150.00),

(1005, 2, '2023-09-24', 120.00),

(1006, 1, '2023-06-19', 20.00);

Question 7 : Generate a report showing CustomerName, Email, and the TotalNumberofOrders for each customer. Include customers who have not placed any orders, in which case their TotalNumberofOrders should be 0. Order the results by CustomerName.
Ans. The key here is using a **LEFT JOIN**. If we used a standard `INNER JOIN`, Diana Prince (who hasn't placed an order) would be excluded from the results. By using `LEFT JOIN`, we keep all customers and fill in the blanks with `NULL` where orders don't exist. We then use `COUNT()` and `COALESCE()` (or `IFNULL`) to turn those empty spaces into a clean
SELECT

C.CustomerName,

C.Email,

```
    COUNT(O.OrderID) AS TotalNumberOfOrders

FROM Customers C

LEFT JOIN Orders O ON C.CustomerID = O.CustomerID

GROUP BY C.CustomerID, C.
```

Key Components Explained

- **LEFT JOIN:** Ensures that every customer from the "left" table (Customers) appears in the list, even if they don't have a matching record in the "right" table (Orders).
- **COUNT(O.OrderID):** We count a specific column from the Orders table. If a customer has no orders, this count naturally results in 0 because COUNT ignores NULL values.
- **GROUP BY:** Since we are using an aggregate function (COUNT), we must group by the non-aggregated columns.
- **ORDER BY:** Sorts the final report alphabetically by the customer's name.

Expected Output

CustomerName	Email	TotalNumberOfOrders
Alice Wonderland	alice@example.com	3
Bob the Builder	bob@example.com	2
Charlie Chaplin	charlie@example.com	1
Diana Prince	diana@example.com	0