Question 1: Explain the fundamental differences between DDL, DML, and DQL commands in SQL.

Provide one example for each type of command.
Answer:

The fundamental difference between DDL, DML, and DQL commands lies in their **purpose and scope** within the database system.

Command Type	Full Form	Purpose	Example
DDL	**Data Definition Language**	Used to **define** or modify the structure (schema) of database objects (tables, indexes, views, etc.). These commands affect the database structure itself.	CREATE TABLE table_name (column_name data_type);
DML	**Data Manipulation Language**	Used to **manage** the data within the schema objects, allowing users to insert, update, or delete records. These commands affect the data stored in the tables.	INSERT INTO table_name (column1) VALUES (value1);
DQL	**Data Query Language**	Used solely to **retrieve** data from the database. It consists only of the SELECT command.	SELECT column1, column2 FROM table_name WHERE condition;

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.
Answer:

The purpose of **SQL Constraints** is to enforce rules on the data columns of a table to limit the type of data that can go into a table. They ensure the **accuracy** and **reliability** of the data in the database.

Three Common Types of Constraints:

1. **PRIMARY KEY**
    - **Description:** A field or set of fields that uniquely identifies each record in a table. It must contain unique values and cannot contain NULL values.
    - **Scenario:** In a Students table, the **StudentID** column should be the PRIMARY KEY to ensure that every student has a unique identifier and no record is duplicated.
2. **FOREIGN KEY**
    - **Description:** A field in one table that uniquely identifies a row of another table, establishing a link between the two. It enforces **referential integrity**.
    - **Scenario:** In an Orders table, the **CustomerID** column should be a FOREIGN KEY referencing the Customers table. This ensures that an order can only be placed by an existing customer.
3. **NOT NULL**
    - **Description:** Ensures that a column cannot have a **NULL** value, guaranteeing that data is always present in that column for every record.
    - **Scenario:** In a Employees table, the **LastName** column should be NOT NULL because it is essential information for identifying an employee.

Question 3: Explain the difference between LIMIT and OFFSET clauses in SQL.

How would you use them together to retrieve the third page of results, assuming each page has 10 records?

Answer:

- **LIMIT Clause:** Used to **restrict** the number of rows returned by a query, specifying the maximum number of rows to retrieve.
- **OFFSET Clause:** Used to **skip** a specified number of rows from the beginning of the result set before starting to return the required result rows.

Retrieving the Third Page of Results

To retrieve the third page of results with 10 records per page, we need to skip the first two pages ($2 \times 10 = 20$ records).

- **LIMIT** (Page Size) = 10
- **OFFSET** (Records to Skip) = 20

The query is:

SQL
SELECT columns
FROM table_name
ORDER BY some_column -- ORDER BY is crucial for consistent pagination

```
LIMIT 10          -- Retrieve 10 records (the third page)
OFFSET 20;        -- Skip the first 20 records (pages 1 and 2)
```

Question 4: What is a Common Table Expression (CTE) in SQL, and what are its main benefits?

Provide a simple SQL example demonstrating its usage.
Answer:

A **Common Table Expression (CTE)** is a temporary, named result set that you can reference within a single SQL statement (e.g., SELECT, INSERT, UPDATE, or DELETE). It is defined using the **WITH** clause.

Main Benefits:

1. **Readability:** CTEs break down complex queries into simpler, logical, and more readable blocks.
2. **Reusability:** The CTE can be referenced multiple times within the same query.
3. **Recursion:** CTEs are necessary for performing recursive queries (queries that reference themselves).

Simple SQL Example:

This example calculates the average price of all products (in the CTE) and then selects products that are above that average.

```
SQL
WITH AveragePrice AS (
   -- Define the CTE: Calculate the average price
   SELECT AVG(Price) AS AvgProductPrice
   FROM Products
)
-- Use the CTE: Select products above the calculated average
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AvgProductPrice FROM AveragePrice);
```

Question 5: Describe the concept of SQL Normalization and its primary goals.

Briefly explain the first three normal forms (1NF, 2NF, 3NF).
Answer:

SQL Normalization is the systematic process of organizing the columns and tables of a relational database to minimize **data redundancy** and improve **data integrity**.

Primary Goals:

1. **Eliminate Redundant Data:** Avoid storing the same data in multiple places to save storage and prevent inconsistencies.
2. **Ensure Data Dependencies Make Sense:** Ensure that the data is logically stored, meaning that only related data is grouped together in a table.

The First Three Normal Forms:

1. **First Normal Form (1NF)**:
    - The table must contain only **atomic** (indivisible) values in each column.
    - There must be no repeating groups of data.
2. **Second Normal Form (2NF)**:
    - The table must be in **1NF**.
    - There must be **no partial dependencies**. This means no non-key attribute can depend on only a *part* of a composite primary key.
3. **Third Normal Form (3NF)**:
    - The table must be in **2NF**.
    - There must be **no transitive dependencies**. This means non-key attributes must not depend on other non-key attributes; all non-key attributes must depend directly on the primary key.