Assignment Code: DA-AG-014
Introduction to SQL and Advanced
Functions | Assignment

Question 1: Explain the fundamental differences between DDL, DML, and DQL commands in SQL. Provide one example for each type of command.

Answer: SQL (Structured Query Language) is used to manage and manipulate data stored in relational databases. SQL commands are categorized based on their purpose. Three of the most important categories are **DDL**, **DML**, and **DQL**. Each of these types serves a distinct function within database operations.

1. DDL (Data Definition Language)

DDL commands are used to define, create, modify, or delete the structure of database objects such as tables, schemas, and indexes. These commands affect the database schema rather than the data itself.

Example:

```
CREATE TABLE employees (
   id INT PRIMARY KEY,
   name VARCHAR(50),
   salary INT );
```

2. DML (Data Manipulation Language)

DML commands handle the manipulation of data stored in tables. This includes inserting new records, updating existing ones, and deleting unnecessary data.

Example:

```
INSERT INTO employees (id, name, salary)
VALUES (1, 'Aman', 50000);
```

3. DQL (Data Query Language)

DQL commands are used to retrieve data from the database. They do not modify data; they only read and display it.

Example:

```
SELECT name, salary FROM employees;
```

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: SQL constraints are rules applied to database tables to maintain the accuracy, reliability, and consistency of data. They prevent invalid or inconsistent data from being inserted, ensuring the database remains trustworthy and logically correct. Constraints play a key role in enforcing data integrity, controlling relationships between tables, and restricting the type of data that can be stored.

Purpose of SQL Constraints: The main purpose of SQL constraints is to enforce rules at the database level so that only valid data is stored. By restricting the type, range, uniqueness, or relationship of data, constraints protect the database from errors such as duplicate entries, missing values that are required, or references to non-existing records. In short, constraints help maintain data integrity and prevent data corruption.

Common Types of SQL Constraints:

1. PRIMARY KEY Constraint: The PRIMARY KEY uniquely identifies each record within a table.

It ensures that:

- No two rows have the same primary key value.
- The key column cannot contain NULL values.

Scenario: In an Employees table, each employee must have a unique ID. The PRIMARY KEY ensures that two employees cannot share the same identifier.

```
CREATE TABLE Employees (
  EmpID INT PRIMARY KEY,
  Name VARCHAR(50) );
```

2. UNIQUE Constraint: The UNIQUE constraint ensures that all values in a specific column are different, preventing duplicate entries. Unlike the primary key, a UNIQUE column can contain a NULL value (unless combined with NOT NULL).

Scenario: In a user's table, each user must have a unique email address. The UNIQUE constraint prevents duplicate email registrations.

```
CREATE TABLE Users (
  UserID INT PRIMARY KEY,
  Email VARCHAR(100) UNIQUE );
```

3. FOREIGN KEY Constraint: The FOREIGN KEY establishes a relationship between two tables by ensuring that the value in one table corresponds to an existing value in another. It maintains referential integrity across related tables.

Scenario: In an Orders table, each order must be associated with a valid user. The FOREIGN KEY ensures that an order cannot reference a non-existent user.

```
CREATE TABLE Orders (
  OrderID INT PRIMARY KEY,
  UserID INT,
  FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Conclusion: SQL constraints are essential for maintaining high-quality, consistent, and reliable data within a database. By enforcing rules such as uniqueness, mandatory values, and valid relationships, constraints help prevent errors and ensure the database remains structurally sound. Understanding and applying constraints correctly is a foundational skill in database management.

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:

Introduction:

When dealing with large datasets, it is often necessary to display results in smaller portions or "pages." SQL provides the **LIMIT** and **OFFSET** clauses to control how many rows are returned and from which point the retrieval should begin. These clauses are commonly used in pagination.

- **LIMIT Clause**

Purpose: The LIMIT clause specifies the maximum number of records to return from a query. It restricts the result set to a certain number of rows.

Example:

```
SELECT * FROM products

LIMIT 10;
```

"This returns the first 10 records from the *products* table."

- **OFFSET Clause**

Purpose: The **OFFSET** clause tells the database how many rows to skip before starting to return results.

Example:

```
SELECT * FROM products

OFFSET 5;
```

"This skips the first 5 records and starts returning results from the 6th row onward."

Using LIMIT and OFFSET Together for Pagination

To retrieve a specific "page" of records, OFFSET and LIMIT are used together.

Goal: Retrieve the **third page**, where each page contains **10 records**.

Logic:

- Page 1 → skip 0 records → OFFSET = 0
- Page 2 → skip 10 records → OFFSET = 10
- **Page 3 → skip 20 records → OFFSET = 20**

SQL Query:

```
SELECT * FROM products
```

```
LIMIT 10
OFFSET 20;
```

Explanation:

- OFFSET 20 skips the first 20 records.
- LIMIT 10 returns the next 10 records, which correspond to page 3.

Conclusion: The LIMIT clause controls *how many* records are returned, while the OFFSET clause determines *where* the retrieval starts. Together, they make it possible to implement efficient pagination, such as retrieving the third set of 10 records from a larger dataset.

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:

Introduction: A Common Table Expression (CTE) is a temporary, named result set created within a SQL query using the WITH keyword. It exists only for the duration of the query and helps structure complex logic in a cleaner and more readable way.

CTE: A CTE allows you to define a query result that can be referenced later in the main query. It behaves like a temporary table or a short-lived view, but without requiring you to create or store anything in the database.

Main Benefits of Using a CTE

1. Improved Readability - Complex queries become easier to understand because logic is broken into smaller, named parts.

2. Reusability Within a Query - You can reference the same CTE multiple times in the same query instead of duplicating code.

3. Easier to Write Recursive Queries - CTEs support recursion, which is useful for hierarchical data like organizational charts or category trees.

Example of a Simple CTE Usage:

The following example selects students older than 20 using a CTE:

```
WITH OlderStudents AS (
   SELECT StudentID, Name, Age
   FROM Students
   WHERE Age > 20
)
SELECT *
FROM OlderStudents;
```

Explanation:

The WITH block creates a temporary result set named OlderStudents. The main SELECT retrieves data from this CTE as if it were a table.

Conclusion: A Common Table Expression simplifies complex queries by making them more readable, reusable, and easier to maintain. Its ability to structure logic and support recursion makes it an essential tool for efficient SQL query writing.

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

Answer:

Introduction: SQL normalization is the process of organizing data in a relational database to reduce redundancy and improve data integrity. It involves structuring tables and defining relationships in a way that ensures the database remains consistent, efficient, and free from unnecessary duplication.

Primary Goals of Normalization

1. Eliminate Data Redundancy: Prevent storing the same data in multiple places.
2. Ensure Data Integrity: Reduce the chances of anomalies during insert, update, or delete operations.
3. Improve Database Efficiency: Make queries faster and storage more optimized by structuring data logically.

Normal Forms

1. First Normal Form (1NF)

Definition: A table is in 1NF if:

- All values are atomic (no multiple values in one cell).
- No repeating groups or arrays.
- Each record is unique.

Example Issue: A student having multiple phone numbers stored in one column.

Fix: Split multiple values into separate rows or a separate table.

2. Second Normal Form (2NF)

Definition: A table is in 2NF if:

- It is already in 1NF.
- Every non-key attribute depends on the entire primary key, not just part of it.

Applies To: Tables with composite primary keys.

Example Issue: In a table with primary key (StudentID, CourseID), storing StudentName creates a partial dependency, since StudentName depends only on StudentID.

Fix: Move attributes that depend on only part of the key to another table.

3. Third Normal Form (3NF)

Definition: A table is in 3NF if:

- It is already in 2NF.
- There are no transitive dependencies, i.e., non-key attributes do not depend on other non-key attributes.

Example Issue: A table stores (StudentID, City, CityPincode). CityPincode depends on City, not on StudentID → transitive dependency.

Fix: Create a separate table for city-related information.

Conclusion: Normalization organizes data efficiently by removing redundancy, ensuring clean relationships, and preventing anomalies. Understanding 1NF, 2NF, and 3NF is essential for designing reliable and scalable databases.