

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 : The fundamental differences between DDL, DML, and DQL commands in SQL, along with examples:

1. DDL (Data Definition Language)

Purpose:

DDL commands are used to define and manage the structure of database objects such as tables, schemas, and indexes.

Key Characteristics:

- They deal with database schema (structure).
- Changes made by DDL commands are automatically committed — they cannot be rolled back.
- Affect the overall design, not the data itself.

Common DDL Commands:

CREATE, ALTER, DROP, TRUNCATE, RENAME

Example:

```
CREATE TABLE Employees (  
    EmpID INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Salary DECIMAL(10,2)  
);
```

This command creates a new table named Employees with columns EmpID, Name, and Salary.

2. DML (Data Manipulation Language)

Purpose:

DML commands are used to manipulate and manage data stored inside tables.

Key Characteristics:

- They deal with data inside the schema objects.
- Changes made by DML commands can be rolled back or committed using transactions.
- They are used for inserting, updating, and deleting data.

Common DML Commands:

INSERT, UPDATE, DELETE

Example:

INSERT INTO Employees (EmpID, Name, Salary)

VALUES (101, 'Amit Sharma', 50000);

This command adds a new record into the Employees table.

3. DQL (Data Query Language)

Purpose:

DQL commands are used to query and retrieve data from the database.

Key Characteristics:

- They deal with fetching data — not modifying it.
- The main goal is to get information using certain conditions.
- Results are returned in the form of result sets.

Common DQL Command:

SELECT

Example:

SELECT Name, Salary FROM Employees WHERE Salary > 40000;

This command retrieves the names and salaries of employees earning more than ₹40,000.

Summary Table

Type	Full Form	Purpose	Example Command
DDL	Data Definition Language	Define database structure	CREATE TABLE
DML	Data Manipulation Language	Manipulate table data	INSERT INTO
DQL	Data Query Language	Retrieve data	SELECT

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 : Purpose of SQL Constraints

SQL constraints are rules applied to table columns to ensure the accuracy, consistency, and integrity of the data stored in a database.

They prevent invalid or inconsistent data from being entered and maintain reliable relationships between tables.

Three Common Types of Constraints

1. PRIMARY KEY Constraint

Purpose:

Ensures that each record in a table is uniquely identifiable.
A primary key column cannot have duplicate or NULL values.

Example Scenario:

In an Employees table, every employee should have a unique ID.

Example SQL:

```
CREATE TABLE Employees (  
    EmpID INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Department VARCHAR(30)  
);
```

Here, EmpID ensures each employee record is unique and identifiable.

2. FOREIGN KEY Constraint

Purpose:

Maintains referential integrity between two tables by creating a link between them.
A foreign key in one table refers to the primary key of another table.

Example Scenario:

In a Departments table and an Employees table, each employee must belong to a valid department.

Example SQL:

```
CREATE TABLE Departments (  
    DeptID INT PRIMARY KEY,  
    DeptName VARCHAR(50)  
);
```

```
CREATE TABLE Employees (  
    EmpID INT PRIMARY KEY,  
    Name VARCHAR(50),  
    DeptID INT,  
    FOREIGN KEY (DeptID) REFERENCES Departments(DeptID)  
);
```

This ensures that an employee's DeptID must exist in the Departments table.

3. CHECK Constraint

Purpose:

Ensures that the values in a column meet a specific condition or rule.

Example Scenario:

In a Products table, the price of a product should always be positive.

Example SQL:

```
CREATE TABLE Products (  
    ProductID INT PRIMARY KEY,  
    ProductName VARCHAR(50),  
    Price DECIMAL(10,2) CHECK (Price > 0)  
);
```

This ensures no product can be inserted or updated with a price less than or equal to zero.

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 : Difference Between LIMIT and OFFSET in SQL

1. LIMIT Clause

- Purpose:
The LIMIT clause is used to specify the maximum number of records to return from a query.
- Use Case:
It's commonly used when you need only a subset of data — for example, showing only the first 10 rows on a page.

Example:

```
SELECT * FROM Employees LIMIT 10;
```

→ Returns the first 10 records from the Employees table.

2. OFFSET Clause

- Purpose:
The OFFSET clause tells SQL how many rows to skip before starting to return rows.
- Use Case:
It's typically used with LIMIT for pagination — skipping previous records to show the next set.

Example:

```
SELECT * FROM Employees LIMIT 10 OFFSET 10;
```

→ Skips the first 10 rows and returns the next 10.

Using LIMIT and OFFSET Together for Pagination

If each page contains 10 records, and you want to retrieve the third page:

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

✅ SQL Query for Page 3:

SELECT * FROM Employees

LIMIT 10 OFFSET 20;

Explanation:

- LIMIT 10 → fetch 10 rows
- OFFSET 20 → skip the first 20 rows (2 pages × 10 records per page)

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 in SQL that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.

It is defined using the WITH keyword and exists only for the duration of the query.

Syntax:

```
WITH cte_name AS (
    -- SQL query
)
SELECT * FROM cte_name;
```

Main Benefits of Using a CTE

1. ✅ Improves Readability and Organization
 - Makes complex queries easier to read and maintain.
 - You can break down big SQL statements into smaller, logical parts.
2. 💻 Supports Recursion
 - CTEs can call themselves, making it easier to handle hierarchical or tree-like data (e.g., employee-manager relationships).
3. ♻️ Reusability within a Query
 - You can refer to the same CTE multiple times in a query instead of writing the same subquery repeatedly.
4. 🧩 Simplifies Nested Queries
 - Helps replace deeply nested subqueries with cleaner, structured code.

Simple Example:

Suppose you have a table Employees:

EmpID	Name	Department	Salary
1	Amit Sharma	HR	50000
2	Neha Gupta	IT	60000
3	Raj Mehta	IT	70000
4	Sia Verma	HR	55000

Goal: Find employees who earn more than the average salary in their department.

Using a CTE:

```
WITH AvgSalary AS (  
    SELECT Department, AVG(Salary) AS AvgDeptSalary  
    FROM Employees  
    GROUP BY Department  
)  
SELECT e.Name, e.Department, e.Salary  
FROM Employees e  
JOIN AvgSalary a  
ON e.Department = a.Department  
WHERE e.Salary > a.AvgDeptSalary;
```

Explanation:

- The CTE AvgSalary calculates the average salary for each department.
- The main query then uses that result to find employees earning more than their department average.





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

Answer : Concept of SQL Normalization

Normalization is the process of organizing data in a database to reduce data redundancy (duplication) and improve data integrity (accuracy and consistency).

It involves dividing large tables into smaller, related tables and defining relationships between them using primary and foreign keys.

Primary Goals of Normalization

1.  Eliminate data redundancy – Avoid storing the same data in multiple places.
2.  Ensure data integrity – Keep the data consistent and accurate.
3.  Simplify maintenance – Easier to insert, update, or delete records without anomalies.
4.  Improve query efficiency – Organized structure helps faster access and manipulation.

Normal Forms (Up to 3NF)

1. First Normal Form (1NF)

Rule:

- Each column must contain atomic (indivisible) values.
- Each record must be unique (identified by a primary key).
- No repeating groups or arrays allowed.

Example:

StudentID Name Subjects

1 Riya Math, Science

 Not in 1NF — because Subjects contains multiple values.

 In 1NF — split subjects into separate rows:

StudentID Name Subject

1 Riya Math

1 Riya Science


2. Second Normal Form (2NF)

Rule:

- Must be in 1NF.
- No partial dependency — non-key columns must depend on the entire primary key, not just part of it.
(Applies mainly to tables with composite primary keys.)

Example:

StudentID CourseID StudentName CourseName

 Not in 2NF: StudentName depends only on StudentID, not the full key (StudentID, CourseID).

✅ In 2NF:

Split into two tables:

Students: (StudentID, StudentName)

Courses: (CourseID, CourseName)

Enrollments: (StudentID, CourseID)

3. Third Normal Form (3NF)

Rule:

- Must be in 2NF.
- No transitive dependency — non-key columns should depend only on the primary key, not on other non-key columns.

Example:

EmpID EmpName DeptID DeptName

❌ Not in 3NF: DeptName depends on DeptID, not directly on EmpID.

✅ In 3NF:

Split into two tables:

Employees: (EmpID, EmpName, DeptID)

Departments: (DeptID, DeptName)