Introduction to SQL and Advanced Functions | Assignment:

Assignment Code: DA-AG-014:

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

1 DDL – Data Definition Language

Purpose: Defines or changes the structure of the database.

Think of DDL like **building or modifying a house** (creating walls, rooms, etc.).

What it works on:

- Tables
- Databases
- Schema structure

Common DDL Commands:

- `CREATE`
- `ALTER`
- `DROP`
- `TRUNCATE`

Example:

```
CREATE TABLE students (
   id INT,
   name VARCHAR(50),
   age INT
);
```

It creates a new table named 'students'.

DML – Data Manipulation Language

Purpose: Works with the data inside the table.

Think of DML like adding, updating, or removing furniture inside the house.

What it works on:

- Rows (records) inside tables

Common DML Commands:

- `INSERT`
- `UPDATE`
- `DELETE`

Example:

```
INSERT INTO students (id, name, age)
VALUES (1, 'Rahul', 22);
```

It adds a new row (record) into the students table.

DQL – Data Query Language

Purpose: Retrieves (reads) data from the database.

Think of DQL like looking inside the house to see what is there.

Main DQL Command:

- SELECT

Example:

SELECT * FROM students;

It shows all records from the students table.

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.

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

Simple meaning:
Constraints help prevent wrong or duplicate data from entering your database.

For example, in a banking system, you cannot allow:

- Two customers with the same account number
- A negative age
- A NULL value in required fields

1 PRIMARY KEY Constraint

Purpose:

- Uniquely identifies each row in a table
- Cannot contain NULL

- Cannot have duplicate values

Each table can have only one Primary Key

Example Scenario:

In a students table, each student must have a unique Student_ID.

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT
);
```

NOT NULL Constraint

Purpose:

Ensures a column cannot have NULL (empty) values

Example Scenario:

In an employees table, the name should never be empty.

```
CREATE TABLE employees (
    emp_id INT,
    name VARCHAR(50) NOT NULL,
    salary INT
);
```

UNIQUE Constraint

Purpose:

- Ensures all values in a column are different
- Unlike Primary Key, it can allow one NULL (depending on database system)

Example Scenario:

In a users table, each email must be unique.

```
CREATE TABLE users (
    user_id INT,
    email VARCHAR(100) UNIQUE
);
```

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?

Difference Between LIMIT and OFFSET in SQL

(Both clauses are mainly used for pagination (showing results page by page instead of all at once).

1 LIMIT

- Controls how many rows to return.
- It restricts the number of records in the result.

Example:

SELECT * FROM employees

LIMIT 10;

Returns only 10 rows.

2 OFFSET

- Skips a specified number of rows before starting to return results.
- It does not return data by itself – it works with LIMIT.

Example:

SELECT * FROM employees

LIMIT 10 OFFSET 5;

Skips the first 5 rows, then returns the next 10 rows.

Retrieve the Third Page (10 Records per Page)

Step 1: Use Formula

OFFSET=(PageNumber−1)×RecordsPerPage\text{OFFSET} = (\text{PageNumber} - 1) \times \text{RecordsPerPage}OFFSET=(PageNumber−1)×RecordsPerPage

For Page 3:

(3−1)×10=20(3 - 1) \times 10 = 20(3−1)×10=20

So:

- Skip first 20 rows
- Return next 10 rows

Final Query:

SELECT *

FROM employees

```
ORDER BY employee_id

LIMIT 10 OFFSET 20;
```

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.

What is a Common Table Expression (CTE) in SQL?

A Common Table Expression (CTE) is a temporary result set that you define using the WITH keyword, and then use inside a SELECT, INSERT, UPDATE, or DELETE statement.

Simple meaning:
A CTE is like creating a temporary named result that makes your query cleaner and easier to understand.

It exists only during the execution of that query.

Why Do We Use CTE? (Main Benefits)

1 Improves Readability

Breaks complex queries into smaller, logical parts.

2 Makes Complex Queries Easier

Especially useful with:

- Aggregations
- Multiple joins
- Subqueries

3 Can Be Reused in the Same Query

You can reference the CTE multiple times in the main query.

4 Supports Recursive Queries

Used for hierarchical data like:

- Employee–Manager structure
- Category trees

Basic Syntax:

```
WITH cte_name AS (

    SELECT column1, column2

    FROM table_name

    WHERE condition

)

SELECT *

FROM cte_name;
```

Simple Example

Scenario:

We want to find employees whose salary is above the average salary.

Without CTE → usually done with subquery
With CTE → cleaner and more readable

Example Using CTE:

```
WITH avg_salary_cte AS (

    SELECT AVG(salary) AS avg_salary

    FROM employees

)
```

```
SELECT e.name, e.salary

FROM employees e

JOIN avg_salary_cte a

ON e.salary > a.avg_salary;
```

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

SQL Normalization is a database design technique used to organize data in tables properly to:

- Reduce data redundancy (duplicate data)
- Improve data integrity
- Avoid update, insert, and delete anomalies
- Make the database more structured and efficient

Primary Goals of Normalization

1. Eliminate duplicate data
2. Ensure data consistency
3. Improve data integrity
4. Reduce storage wastage
5. Make maintenance easier

First Three Normal Forms:

1NF – First Normal Form

Rule:

- Each column must contain atomic (single) values
- No multiple values in one cell
- Each record must be unique (Primary Key required)

2NF – Second Normal Form

Rule:

- Must already be in 1NF
- No partial dependency
- All non-key columns must depend on the entire primary key

3NF – Third Normal Form

Rule:

- Must be in 2NF
- No transitive dependency
- Non-key columns should not depend on another non-key column