Introduction to SQl and Advanced Function

Q1. Explain the fundamental difference between DDL, DML, and DQL commands in SQL. Provide one example for each type of command.

Ans). DDL, DML and DQL are categories of SQL commands, each serving a fundamentally different purpose in how a database is defined, modified and queried.

1) **DDL (Data definition Language)**
   - **Fundamental purpose:** DDL commands are used to define, create, modify or delete the structure (schema) of database objects, such as tables, views, indexes and schemas. These commands affect the database design, not the actual data stored inside the tables
   - **Key characteristic:**
     - Works on database structure
     - Changes are usually auto-committed(permanent)
   - **Example:** <<Query in sql

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

2) **DML (Data Manipulation Language)**
   - **Fundamental purpose:** DML commands are used to manipulate the data stored inside database tables. This includes inserting new records, updating existing records, and deleting records.
   - **Key Characteristic:**
     - Works on the data itself
     - Changes can usually be rolled back (transaction - controlled)
   - **Example : <<**

```
INSERT INTO Employees (EmpID, Name, Salary)
VALUES (101, 'Amit', 50000);      >>
```

3) **DQL (Data Query Language)**
   - **Fundamental purpose:** DQL commands are used to retrieve or query data from the database without modifying the database structure or the stored data. The main goal is to fetch information for analysis and reporting.
   - **Key characteristic:**
     - Read- only operations
     - Does not change data or structure
   - **Example:**

```
<<    SELECT Name, Salary FROM Employees
      WHERE Salary > 40000;    >>
```

Q 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. SQL constraints are rules applied to table columns to maintain data integrity, accuracy, and consistency in a database. They prevent invalid data from being inserted, updated, or deleted ensuring that the data stored in tables follows defined business and relational rules.

1. PRIMARY KEY Constraint

Description:
The PRIMARY KEY constraint uniquely identifies each record in a table. It does not allow **duplicate values** or **NULL values**.

Purpose:

- Ensures each row can be uniquely identified
- Maintains entity integrity

Scenario:
In an **Employees** table, each employee must have a unique ID so that no two employees share the same identifier.

2. NOT NULL Constraint

Description:
The NOT NULL constraint ensures that a column cannot have NULL (empty) values.

Purpose:

- Enforces mandatory fields
- Prevents missing critical data

Scenario:
In a **Customers** table, the customer's email address must always be provided for communication, so the email column should not allow NULL values.

3. FOREIGN KEY Constraint

Description:
The FOREIGN KEY constraint establishes a relationship between two tables by enforcing that a value in one table must exist in another referenced table.

Purpose:

- Maintains referential integrity

- Prevents orphan records

Scenario:
In an **Orders** table, each order must be linked to a valid customer. The customer_id in Orders references the customer_id in the Customers table

Q3. 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 ?

Ans. **LIMIT** and **OFFSET** are SQL clauses used to control the number of rows returned by a query, mainly for pagination.

- **LIMIT** specifies how many records should be returned.
- **OFFSET** specifies how many records to skip before starting to return rows.

Key Difference:

- **LIMIT** controls the page size (number of rows per page).
- **OFFSET** controls the starting position of the result set.

They are commonly used together to divide large result sets into smaller manageable pages.

```
<<  SELECT *

    FROM Employees

    ORDER BY EmpID

     LIMIT 10 OFFSET 20; >>
```

Explanation:

- **OFFSET 20** skips the first 20 records
- **Limit 10** returns the next 10 records
- This retrieves records 21 to 30, which represent the third page

Q4. What is a Common Table Expression (CTE) in SQL and what are its main benefits? Provide a simple SQL example demonstrating its usage.

Ans. A Common Table Expression (CTE) is a temporary, named result set defined within a SQL query using the **WITH** keyword. It exists only for the duration of the query and can be referenced like a table or view inside that query.

CTEs are mainly used to simplify complex queries, improve readability and enable logical step-by-step data processing.

Main Benefits of CTEs

1. **Improved Readability**
   CTEs break complex queries into smaller, logical parts, making them easier to understand

and maintain.

2. **Reusability Within a Query**
   A CTE can be referenced multiple times within the same SQL statement, avoiding repeated subqueries.

3. **Simplifies Complex Logic**
   CTEs make queries involving joins, aggregations, or calculations more structured and cleaner than nested subqueries.

4. **Supports Recursive Queries**
   CTEs can be recursive, allowing hierarchical or tree-structured data (e.g., employee-manager relationships) to be queried easily.

Practical Example

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

SQL Using CTE
```
WITH AvgSalary AS (
     SELECT AVG(Salary) AS avg_salary
     FROM Employees
 )
 SELECT Name, Salary
 FROM Employees
 WHERE Salary > (SELECT avg_salary FROM AvgSalary);
```

Explanation of the Example

- The CTE `AvgSalary` calculates the **average salary**.
- The main query uses the CTE result to filter employees.
- This approach is **clearer and more readable** than writing the average salary calculation inside the WHERE clause directly.

Q5. Describe the concept of SQL Normalization and its primary goals. Briefly explain the first three normal forms (1NF,2NF,3NF).

Ans. SQL Normalization is the process of organizing data in a relational database to minimize redundancy and dependency. It involves dividing large tables into smaller, well-structured tables and defining proper relationships between them.

Primary Goal of Normalization

1. **Eliminate data redundancy**
   Avoid storing the same data in multiple places.

2. **Ensure data consistency and integrity**
   Changes made in one place are reflected correctly without anomalies.

3. **Reduce data anomalies**
   Prevent:

   - **Insertion anomalies**

   - **Update anomalies**

   - **Deletion anomalies**

4. **Improve database maintainability**
   Makes the database easier to understand, modify, and scale.

First Three Normal Forms

First Normal Form (1NF)

- **Concept:**
  A table is in **1NF** if:
  - Each column contains **atomic (indivisible) values**
  - There are **no repeating groups or multi-valued attributes**
  - Each record can be uniquely identified

1NF Solution: Split multi-valued data into separate rows.

Second Normal Form (2NF)

- **Concept:**
  A table is in **2NF** if:
  - It is already in **1NF**
  - All **non-key attributes are fully dependent on the entire primary key**
    (No partial dependency)

Applies mainly to tables with composite primary keys.

Third Normal Form (3NF)

- **Concept:**
  A table is in **3NF** if**:**
  - It is already in 2NF
  - There is no transitive dependency
    (Non-key attributes should not depend on other non-key attributes)