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

Answer:

Fundamental Differences between DDL, DML, and DQL Commands in SQL

1. DDL (Data Definition Language)

DDL commands are used to define and modify the structure of database objects such as tables and databases.

Key Points:

- Used for creating, altering, and deleting database structures
- Affects the database schema

Example:

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

---

2. DML (Data Manipulation Language)

DML commands are used to insert, update, and delete data stored in database tables.

Key Points:

- Used to manage data inside tables
- Changes the data but not the structure

Example:

```
INSERT INTO employees VALUES (1, 'Amit');
```

---

3. DQL (Data Query Language)

DQL commands are used to retrieve data from database tables.

Key Points:

- Used for querying and fetching data
- Does not modify data

Example:

```
SELECT * FROM employees;
```

Summary Table

Command Type	Purpose	Example
DDL	Defines database structure	CREATE
DML	Manipulates data	INSERT
DQL	Retrieves data	SELECT

Answer

DDL commands define and modify database structures, DML commands manipulate the data stored in tables, and DQL commands retrieve data from the database.

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 control the type of data that can be stored in a table.
They are used to maintain data accuracy, consistency, and integrity in a database.

Common Types of SQL Constraints

1. PRIMARY KEY

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

Scenario:
In an employees table, `employee_id` is used as a primary key to ensure each employee has a unique ID.

---

2. NOT NULL

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

Scenario:
In a students table, the `name` column should not be NULL because every student must have a name.

---

3. FOREIGN KEY

Description:
A FOREIGN KEY is used to link two tables and maintain referential integrity.

Scenario:
In an orders table, `customer_id` is a foreign key that refers to the `customer_id` in the customers table to ensure each order belongs to a valid customer.

Answer

SQL constraints are used to enforce rules on data to maintain accuracy and consistency. Common constraints include PRIMARY KEY for uniqueness, NOT NULL to prevent missing values, and FOREIGN KEY to maintain relationships between tables.

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

LIMIT Clause

The LIMIT clause is used to restrict the number of records returned by a query.
It controls how many rows are displayed in the result set.

Example:

```
SELECT * FROM employees

LIMIT 10;
```

(This returns only 10 records.)

OFFSET Clause

The OFFSET clause is used to skip a specific number of records before starting to return rows.
It controls from which record the result set should begin.

Example:

```
SELECT * FROM employees

OFFSET 10;
```

(This skips the first 10 records.)

Using LIMIT and OFFSET Together (Pagination)

If each page contains 10 records:

- Page 1 → OFFSET 0
- Page 2 → OFFSET 10
- Page 3 → OFFSET 20

To retrieve the third page of results, we skip the first 20 records and then display the next 10 records.

```
SELECT * FROM employees

LIMIT 10 OFFSET 20;
```

Answer

LIMIT controls the number of rows returned, while OFFSET specifies how many rows to skip. To get the third page with 10 records per page, use `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.

Answer :

Common Table Expression (CTE) in SQL

What is a CTE?

A Common Table Expression (CTE) is a temporary result set that can be used within a SQL query.
It is defined using the WITH keyword and helps make complex queries easier to read and manage.

A CTE exists only for the duration of the query and is not stored permanently in the database.

Main Benefits of Using a CTE

- Improves readability of complex queries
- Makes queries easier to understand and maintain
- Allows reuse of the same result set within a query
- Useful for breaking complex logic into smaller parts

Simple Example of a CTE

```
WITH high_salary_employees AS (

    SELECT employee_id, name, salary

    FROM employees

    WHERE salary > 50000

)
```

```
SELECT *

FROM high_salary_employees;
```

Explanation:
The CTE `high_salary_employees` stores employees with salary greater than 50,000 and is then used in the main SELECT query.

Answer

A Common Table Expression (CTE) is a temporary result set defined using the WITH clause. It helps simplify complex queries, improves readability, and makes SQL code easier to manage and reuse.

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

Concept of SQL Normalization

Normalization is the process of organizing data in a database to reduce data redundancy and improve data integrity.
It involves dividing large tables into smaller, related tables and defining relationships between them.

Primary Goals of Normalization

The main goals of normalization are:

- To eliminate duplicate data
- To reduce data redundancy
- To maintain data consistency and integrity
- To avoid update, insert, and delete anomalies

First Three Normal Forms

First Normal Form (1NF)

A table is in First Normal Form (1NF) if:

- Each column contains atomic (indivisible) values
- There are no repeating groups or multivalued attributes

Example:
A table should not store multiple phone numbers in one column.

Second Normal Form (2NF)

A table is in Second Normal Form (2NF) if:

- It is already in 1NF
- All non-key attributes are fully dependent on the primary key

This removes partial dependency.

Third Normal Form (3NF)

A table is in Third Normal Form (3NF) if:

- It is already in 2NF
- There is no transitive dependency

Non-key attributes should depend only on the primary key.

Answer

SQL normalization is the process of organizing data to reduce redundancy and improve integrity. The first three normal forms—1NF, 2NF, and 3NF—ensure atomic values, remove partial dependency, and eliminate transitive dependency, respectively.

QUESTION: 6.

Question 6 : Create a database named `ECommerceDB` and perform the following tasks:

1. Create the following tables with appropriate data types and constraints:
   - Categories
     - CategoryID (INT, PRIMARY KEY)
     - CategoryName (VARCHAR(50), NOT NULL, UNIQUE)
   - Products
     - ProductID (INT, PRIMARY KEY)
     - ProductName (VARCHAR(100), NOT NULL, UNIQUE)
     - CategoryID (INT, FOREIGN KEY → Categories)
     - Price (DECIMAL(10,2), NOT NULL)
     - StockQuantity (INT)
   - Customers
     - CustomerID (INT, PRIMARY KEY)
     - CustomerName (VARCHAR(100), NOT NULL)
     - Email (VARCHAR(100), UNIQUE)
     - JoinDate (DATE)
   - Orders
     - OrderID (INT, PRIMARY KEY)
     - CustomerID (INT, FOREIGN KEY → Customers)
     - OrderDate (DATE, NOT NULL)
     - TotalAmount (DECIMAL(10,2))
2. **Insert the following records into each table**
   - **Categories**

CategoryID	Category Name
1	Electronics
2	Books
3	Home Goods
4	Apparel

ProductID	ProductName	CategoryID	Price	StockQuantity
101	Laptop Pro	1	1200.00	50
102	SQL Handbook	2	45.50	200
103	Smart Speaker	1	99.99	150
104	Coffee Maker	3	75.00	80
105	Novel : The Great SQL	2	25.00	120
106	Wireless Earbuds	1	150.00	100
107	Blender X	3	120.00	60
108	T-Shirt Casual	4	20.00	300

- Customers

CustomerID	CustomerName	Email	Joining Date
1	Alice Wonderland	alice@example.com	2023-01-10
2	Bob the Builder	bob@example.com	2022-11-25
3	Charlie Chaplin	charlie@example.com	2023-03-01
4	Diana Prince	diana@example.com	2021-04-26

- **Orders**

OrderID	CustomerID	OrderDate	TotalAmount
1001	1	2023-04-26	1245.50
1002	2	2023-10-12	99.99
1003	1	2023-07-01	145.00
1004	3	2023-01-14	150.00
1005	2	2023-09-24	120.00
1006	1	2023-06-19	20.00

Answer:

Create Database ECommerceDB, Create Tables, and Insert Records

---

Step 1: Create Database

```
CREATE DATABASE ECommerceDB;
```

Step 2: Use the Database

```
USE ECommerceDB;
```

Step 3: Create Tables

Categories Table

```
CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY,
    CategoryName VARCHAR(50) NOT NULL UNIQUE
);
```

Products Table

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100) NOT NULL UNIQUE,
    CategoryID INT,
    Price DECIMAL(10,2) NOT NULL,
    StockQuantity INT,
```

```
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);
```

Customers Table

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    JoinDate DATE
);
```

Orders Table

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE NOT NULL,
    TotalAmount DECIMAL(10,2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Step 4: Insert Data

Insert into Categories

```
INSERT INTO Categories VALUES

(1, 'Electronics'),

(2, 'Books'),

(3, 'Home Goods'),

(4, 'Apparel');
```

Insert into Products

```
INSERT INTO Products VALUES

(101, 'Laptop Pro', 1, 1200.00, 50),

(102, 'SQL Handbook', 2, 45.50, 200),

(103, 'Smart Speaker', 1, 99.99, 150),

(104, 'Coffee Maker', 3, 75.00, 80),

(105, 'Novel : The Great SQL', 2, 25.00, 120),

(106, 'Wireless Earbuds', 1, 150.00, 100),

(107, 'Blender X', 3, 120.00, 60),

(108, 'T-Shirt Casual', 4, 20.00, 300);
```

Insert into Customers

```
INSERT INTO Customers VALUES

(1, 'Alice Wonderland', 'alice@example.com', '2023-01-10'),

(2, 'Bob the Builder', 'bob@example.com', '2022-11-25'),

(3, 'Charlie Chaplin', 'charlie@example.com', '2023-03-01'),

(4, 'Diana Prince', 'diana@example.com', '2021-04-26');
```

Insert into Orders

```
INSERT INTO Orders VALUES

(1001, 1, '2023-04-26', 1245.50),

(1002, 2, '2023-10-12', 99.99),

(1003, 1, '2023-07-01', 145.00),

(1004, 3, '2023-01-14', 150.00),

(1005, 2, '2023-09-24', 120.00),

(1006, 1, '2023-06-19', 20.00);
```

Final Answer

The database ECommerceDB is created successfully.
All required tables are created with proper data types and constraints, and the given records are inserted correctly.

Question 7 : Generate a report showing CustomerName, Email, and the TotalNumberofOrders for each customer. Include customers who have not placed any orders, in which case their TotalNumberofOrders should be 0. Order the results by CustomerName.

Answer:

SQL Report Using JOIN and Aggregate Function

Concept

This report is used to display customer details along with the total number of orders placed by each customer.
It also includes customers who have not placed any orders.

To achieve this, LEFT JOIN and COUNT() aggregate function are used.

Objectives

The main objectives of this report are:

- To display customer name and email
- To calculate the total number of orders for each customer
- To include customers with no orders and show their order count as 0
- To sort the result by customer name

SQL Concepts Used

LEFT JOIN

LEFT JOIN returns all records from the Customers table, even if there is no matching record in the Orders table.

COUNT() Aggregate Function

COUNT() is used to count the total number of orders for each customer.
If a customer has no orders, the count returned is 0.

SQL Query

```
SELECT

    c.CustomerName,

    c.Email,

    COUNT(o.OrderID) AS TotalNumberofOrders

FROM Customers c

LEFT JOIN Orders o

    ON c.CustomerID = o.CustomerID

GROUP BY

    c.CustomerID,

    c.CustomerName,

    c.Email

ORDER BY

    c.CustomerName;
```

Explanation

- CustomerName and Email are selected from the Customers table
- LEFT JOIN is used to join the Orders table
- COUNT(OrderID) calculates the total number of orders for each customer
- Customers without orders are included with TotalNumberofOrders as 0

- The result is ordered by CustomerName

Answer

This SQL query generates a report showing each customer’s name, email, and total number of orders.
Customers who have not placed any orders are also included, with their total number of orders shown as 0.
The output is sorted in ascending order by customer name.

Question 8 : Retrieve Product Information with Category: Write a SQL query to display the ProductName, Price, StockQuantity, and CategoryName for all products. Order the results by CategoryName and then ProductName alphabetically.

Answer :

Retrieve Product Information with Category

Concept

This query is used to display product details along with their category information.
It helps in understanding which products belong to which category and shows their price and stock quantity.

To get category details, an INNER JOIN is used between the Products and Categories tables.

Objectives

The main objectives of this query are:

- To display product name, price, and stock quantity
- To display the category name for each product
- To combine product and category information
- To sort the result first by category name and then by product name alphabetically

---

SQL Concepts Used

INNER JOIN

INNER JOIN is used to retrieve records that have matching values in both the Products and Categories tables.

ORDER BY Clause

ORDER BY is used to sort the result set first by CategoryName and then by ProductName in alphabetical order.

SQL Query

```
SELECT

    p.ProductName,

    p.Price,

    p.StockQuantity,

    c.CategoryName

FROM Products p

INNER JOIN Categories c

    ON p.CategoryID = c.CategoryID

ORDER BY

    c.CategoryName,

    p.ProductName;
```

Explanation

- ProductName, Price, and StockQuantity are selected from the Products table
- CategoryName is selected from the Categories table
- INNER JOIN connects Products and Categories using CategoryID
- The result is sorted by CategoryName and then ProductName alphabetically

---

Answer

This SQL query retrieves product information along with their category names.
It displays the product name, price, stock quantity, and category name for all products.
The result is ordered alphabetically by category name and then by product name.

Question 9 : Write a SQL query that uses a Common Table Expression (CTE) and a Window Function (specifically ROW_NUMBER() or RANK()) to display the CategoryName, ProductName, and Price for the top 2 most expensive products in each CategoryName.

Answer:

SQL Query

```
WITH RankedProducts AS (
    SELECT
        c.CategoryName,
        p.ProductName,
        p.Price,
        ROW_NUMBER() OVER (
            PARTITION BY c.CategoryName
            ORDER BY p.Price DESC
        ) AS rn
    FROM Products p
    JOIN Categories c
        ON p.CategoryID = c.CategoryID
)
SELECT
    CategoryName,
    ProductName,
    Price
FROM RankedProducts
WHERE rn <= 2
```

```
ORDER BY CategoryName, Price DESC;
```

Explanation

- The Common Table Expression (CTE) ranks products within each category based on price.
- `ROW_NUMBER()` assigns a unique rank to each product per category in descending order of price.
- `PARTITION BY CategoryName` ensures ranking is done separately for each category.
- The final query selects only the top 2 most expensive products from each category.

If you want a version using `RANK()` instead of `ROW_NUMBER()`, I can provide that as well.

Question 10 : You are hired as a data analyst by Sakila Video Rentals, a global movie rental company. The management team is looking to improve decision-making by analyzing existing customer, rental, and inventory data. Using the Sakila database, answer the following business questions to support key strategic initiatives. Tasks & Questions: 1. Identify the top 5 customers based on the total amount they've spent. Include customer name, email, and total amount spent. 2. Which 3 movie categories have the highest rental counts? Display the category name and number of times movies from that category were rented. 3. Calculate how many films are available at each store and how many of those have never been rented. 4. Show the total revenue per month for the year 2023 to analyze business seasonality. 5. Identify customers who have rented more than 10 times in the last 6 months.

Answer:

Role: Data Analyst at Sakila Video Rentals

The following SQL queries are written using the Sakila database to answer the given business questions and support management decision-making.

---

1. Top 5 Customers Based on Total Amount Spent

```
SELECT

    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,

    c.email,

    SUM(p.amount) AS total_amount_spent

FROM customer c

JOIN payment p

    ON c.customer_id = p.customer_id

GROUP BY c.customer_id, c.first_name, c.last_name, c.email

ORDER BY total_amount_spent DESC

LIMIT 5;
```

Explanation:
This query calculates the total amount spent by each customer and returns the top five customers based on total spending.

---

2. Top 3 Movie Categories with the Highest Rental Counts

```
SELECT

    cat.name AS category_name,

    COUNT(r.rental_id) AS rental_count

FROM category cat

JOIN film_category fc

    ON cat.category_id = fc.category_id

JOIN inventory i

    ON fc.film_id = i.film_id

JOIN rental r

    ON i.inventory_id = r.inventory_id

GROUP BY cat.name

ORDER BY rental_count DESC

LIMIT 3;
```

Explanation:
This query identifies the three most rented movie categories by counting how many times films from each category were rented.

---

3. Number of Films Available at Each Store and Films Never Rented

```
SELECT

    s.store_id,

    COUNT(i.inventory_id) AS total_films,

    SUM(CASE

            WHEN r.rental_id IS NULL THEN 1

            ELSE 0

        END) AS never_rented_films

FROM store s

JOIN inventory i

    ON s.store_id = i.store_id

LEFT JOIN rental r

    ON i.inventory_id = r.inventory_id

GROUP BY s.store_id;
```

Explanation:
This query shows the total number of films available in each store and identifies how many of those films have never been rented.

---

4. Total Revenue Per Month for the Year 2023

```
SELECT

    MONTH(payment_date) AS month,
```

```
    SUM(amount) AS total_revenue
FROM payment
WHERE YEAR(payment_date) = 2023
GROUP BY MONTH(payment_date)
ORDER BY month;
```

Explanation:
This query calculates monthly revenue for the year 2023 to help analyze business seasonality.

---

5. Customers Who Rented More Than 10 Times in the Last 6 Months

```
SELECT
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    COUNT(r.rental_id) AS total_rentals
FROM customer c
JOIN rental r
    ON c.customer_id = r.customer_id
WHERE r.rental_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY c.customer_id, c.first_name, c.last_name
HAVING COUNT(r.rental_id) > 10;
```

Explanation:
This query identifies customers who rented movies more than ten times in the last six months, helping to recognize loyal and high-engagement customers.