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

-> **DDL (Data Definition Language)**
Purpose:
DDL commands are used to define, modify, or delete the structure of database objects such as tables, views, and indexes.

Key Characteristics:

Changes the database schema

Usually auto-committed (cannot be rolled back easily)

Does not manipulate actual data, only structure

Common DDL Commands:

CREATE, ALTER, DROP, TRUNCATE

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

DML (Data Manipulation Language)
Purpose:

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

Key Characteristics:

Works on data inside tables

Changes can be rolled back (with transaction control)

Requires COMMIT to save changes permanently

Common DML Commands: INSERT, UPDATE, DELETE

```
INSERT INTO Employees (EmpID, EmpName, Salary)
VALUES (101, 'Rami', 25000);
```

DQL (Data Query Language)
Purpose:

DQL commands are used to retrieve or query data from database tables.

Key Characteristics:

Read-only operations

Does not modify data

Used mainly for reporting and analysis

Common DQL Command:

SELECT

```
SELECT EmpName, Salary
FROM Employees
WHERE Salary > 40000;
```

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.

-> **Purpose of SQL Constraints**

SQL constraints are rules applied to table columns to ensure data accuracy, consistency, and integrity in a database.
They prevent invalid data from being inserted, updated, or deleted, helping maintain reliable and meaningful data.

Three Common Types of SQL Constraints

1. PRIMARY KEY Constraint

Purpose:

Uniquely identifies each record in a table

Does not allow NULL values

Ensures no duplicate rows

Scenario:
In an Employee table, each employee must have a unique ID.

```
CREATE TABLE Employee (
   EmpID INT PRIMARY KEY,
   EmpName VARCHAR(50),
   Dept VARCHAR(30)
);   # Every employee has a unique EmpID.
```

2. NOT NULL Constraint

Purpose:

Ensures a column cannot have NULL values

Forces mandatory data entry

Scenario:
In a Student table, the student name must always be provided.

```
CREATE TABLE Student (
   StudentID INT,
   StudentName VARCHAR(50) NOT NULL,
   Age INT
);     #Prevents inserting a student record without a name.
```

3. FOREIGN KEY Constraint

Purpose:

Maintains referential integrity between two tables

Ensures values in one table exist in another table

Scenario:
Each order must belong to a valid customer.

```
CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   CustomerID INT,
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);  #Prevents creating an order for a non-existent customer.
```

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?

-> LIMIT and OFFSET are SQL clauses used to control how many rows are returned and from where the result set starts. They are commonly used for pagination.

LIMIT

Purpose: Specifies the maximum number of rows to return.
Controls how many records appear in the result.

```
SELECT * FROM Employees
LIMIT 5;
```

#Returns only 5 rows.

OFFSET

Purpose: Specifies the number of rows to skip before returning results.

Controls which row to start from

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

#Skip the first10 rows.

-> To retrieve the third page of results when each page contains 10 records, you use LIMIT and OFFSET together as follows:

```
SELECT *
FROM Employees
LIMIT 10 OFFSET 20;
```

OFFSET 20 → skips the first 20 records

LIMIT 10 → returns the next 10 records

Result → third page of data

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.

-> A Common Table Expression (CTE) is a temporary result set that you can reference within another SQL statement, such as a `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement. It provides a way to define a temporary result set that exists for the duration of the query.

Main Benefits of CTE

1. Improves Readability
    - Breaks complex queries into logical, readable parts
    - Easier to understand than nested subqueries
2. Simplifies Complex Queries
    - Allows you to reference the result set multiple times in the same query
3. Supports Recursive Queries
    - Useful for hierarchical data (e.g., employee–manager relationships)
4. Better Maintainability
    - Makes queries easier to debug and modify

Simple SQL Example Using a CTE :

Find employees whose salary is greater than the average salary.

```
WITH AvgSalary AS (
   SELECT AVG(Salary) AS avg_sal
   FROM Employees
)
SELECT EmpName, Salary
FROM Employees, AvgSalary
WHERE Employees.Salary > AvgSalary.avg_sal;
```

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 tables efficiently, minimize data redundancy, and ensure data integrity [1]. The primary goals are to eliminate redundant data (e.g., storing the same information in multiple places) and to prevent data anomalies (insertion, update, and deletion anomalies) that can occur when data is not organized logically [1, 2].

Primary Goals of Normalization

Eliminate data redundancy (avoid repeated data)

Ensure data consistency and accuracy

Reduce data anomalies (insert, update, delete anomalies)

Improve database maintainability

Optimize storage usage

The first three normal forms are:

- **First Normal Form (1NF):** A table is in 1NF if all column values are atomic (indivisible) and there are no repeating groups of data within a row [1, 3]. Each cell must contain a single value, and each column must have a unique name.
- **Second Normal Form (2NF):** A table is in 2NF if it is in 1NF and all non-key attributes are fully dependent on the primary key [1, 3]. This means that for tables with a composite key, no non-key attribute can depend solely on only a *part* of the primary key.
- **Third Normal Form (3NF):** A table is in 3NF if it is in 2NF and there are no transitive dependencies [1, 3]. A transitive dependency occurs when a non-key attribute is dependent on another non-key attribute rather than directly on the primary key. This form ensures that all attributes are directly related to the key, not to other non-key data points [1].

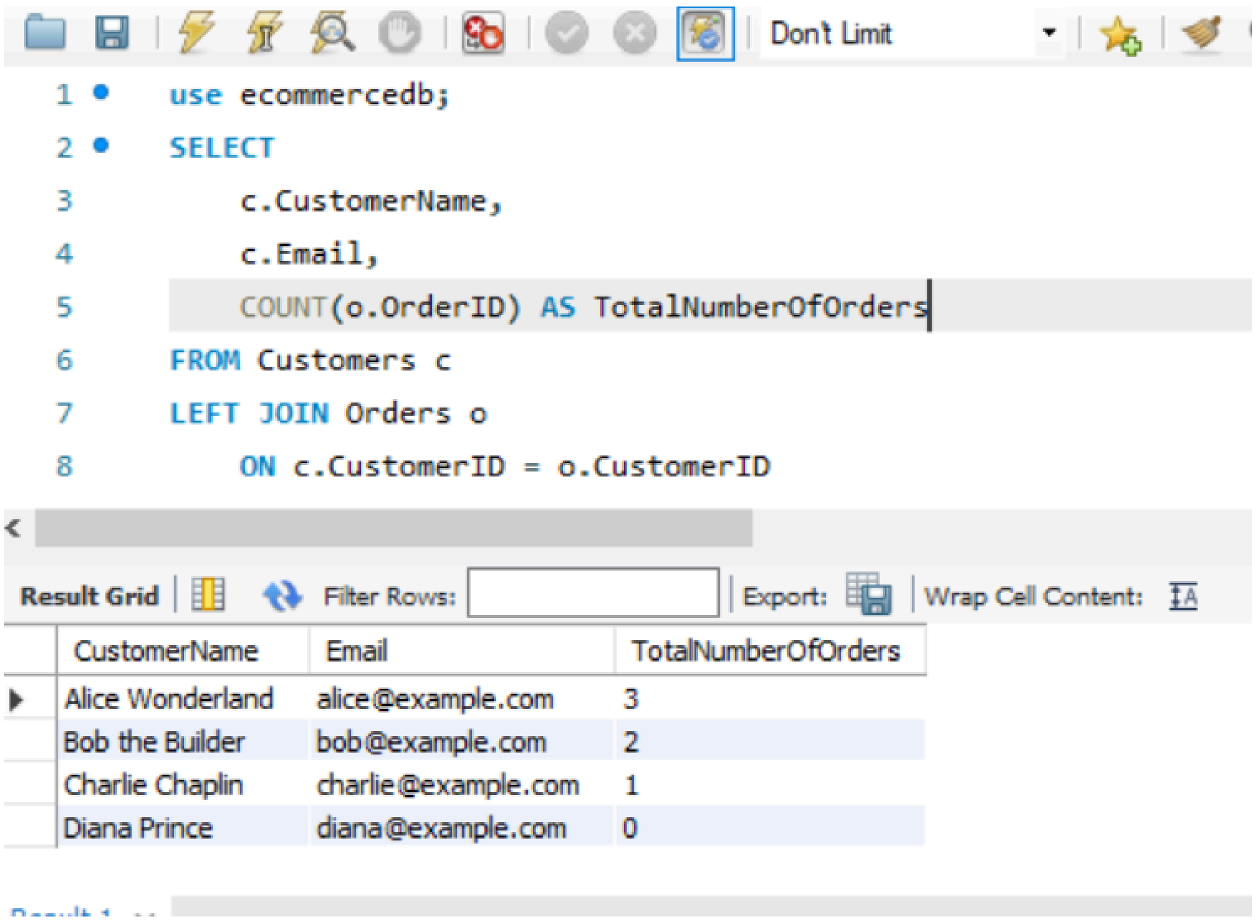
Question 6 :

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

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)
);

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

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

INSERT INTO Categories (CategoryID, CategoryName)
VALUES
(1, 'Electronics'),
(2, 'Books');

INSERT INTO Products (ProductID, ProductName, CategoryID, Price, StockQuantity)
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);
SELECT * FROM Categories;
INSERT INTO Categories (CategoryID, CategoryName)
VALUES
(3, 'Home Goods'),
(4, 'Apparel');

INSERT INTO Products (ProductID, ProductName, CategoryID, Price, StockQuantity)
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 (CustomerID, CustomerName, Email, JoinDate)
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 (OrderID, CustomerID, OrderDate, TotalAmount)
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);
```

7. 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.

```
->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;
```

OUTPUT



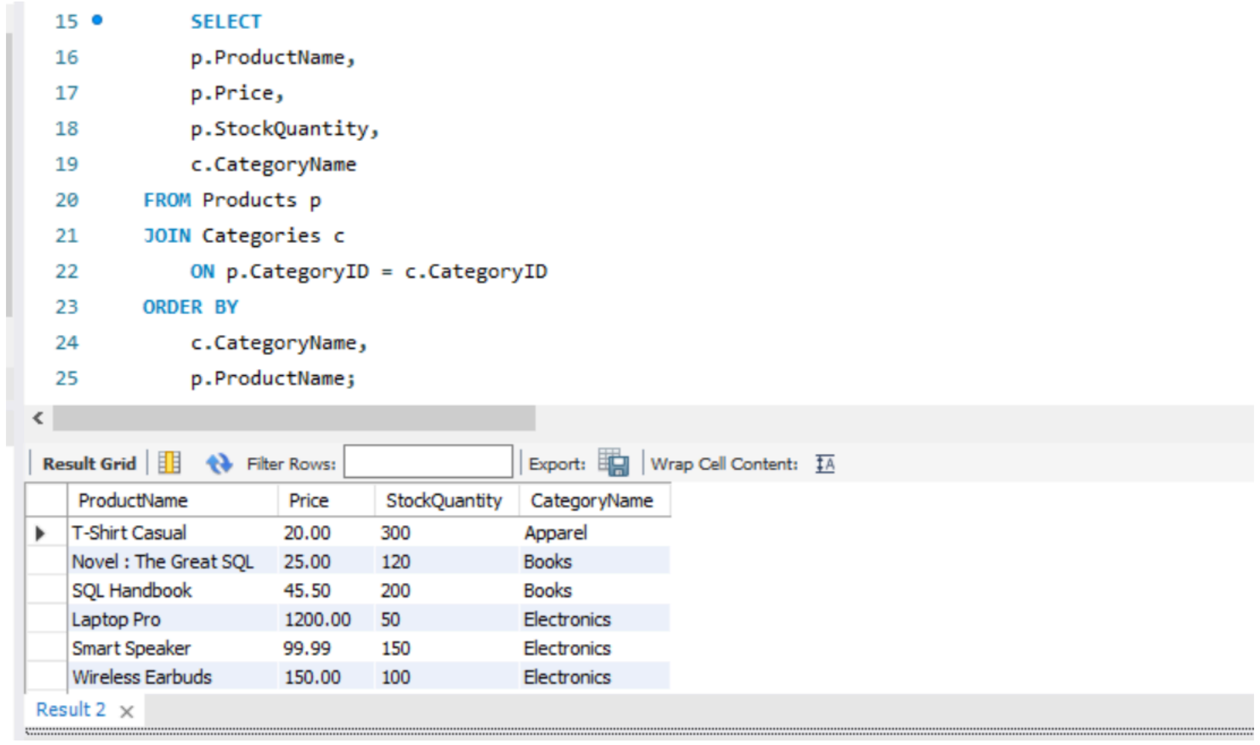
```
use ecommercedb;
SELECT
    c.CustomerName,
    c.Email,
    COUNT(o.OrderID) AS TotalNumberOfOrders
FROM Customers c
LEFT JOIN Orders o
    ON c.CustomerID = o.CustomerID
```

CustomerName	Email	TotalNumberOfOrders
Alice Wonderland	alice@example.com	3
Bob the Builder	bob@example.com	2
Charlie Chaplin	charlie@example.com	1
Diana Prince	diana@example.com	0

8.Question: 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

```
->SELECT
   p.ProductName,
   p.Price,
   p.StockQuantity,
   c.CategoryName
FROM Products p
JOIN Categories c
   ON p.CategoryID = c.CategoryID
ORDER BY
   c.CategoryName,
   p.ProductName;
```

OUTPUT


```
        SELECT
        p.ProductName,
        p.Price,
        p.StockQuantity,
        c.CategoryName
    FROM Products p
    JOIN Categories c
        ON p.CategoryID = c.CategoryID
    ORDER BY
        c.CategoryName,
        p.ProductName;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ProductName	Price	StockQuantity	CategoryName
T-Shirt Casual	20.00	300	Apparel
Novel : The Great SQL	25.00	120	Books
SQL Handbook	45.50	200	Books
Laptop Pro	1200.00	50	Electronics
Smart Speaker	99.99	150	Electronics
Wireless Earbuds	150.00	100	Electronics

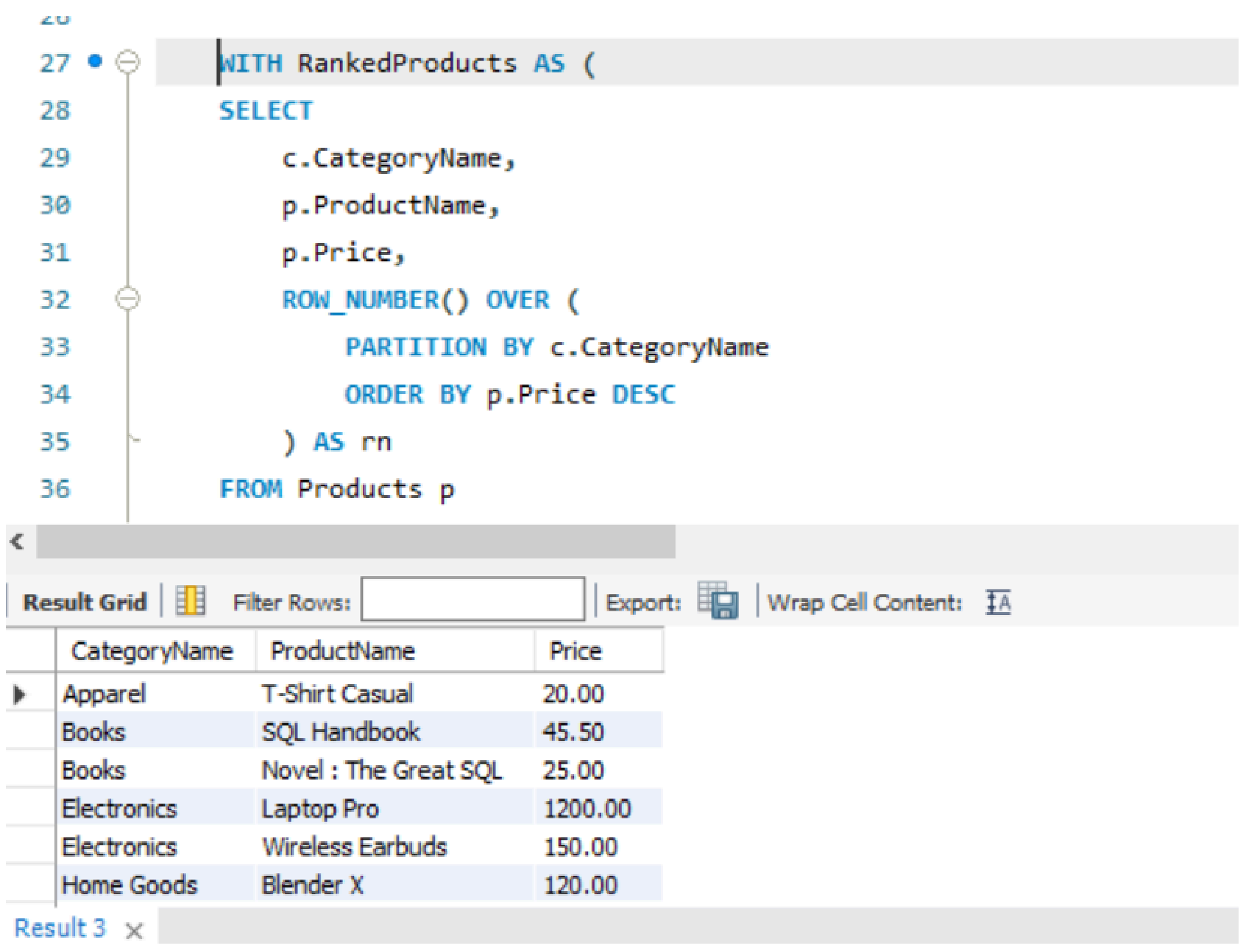
Result 2

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.

```
-> 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;

OUTPUT



```
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
```

CategoryName	ProductName	Price
Apparel	T-Shirt Casual	20.00
Books	SQL Handbook	45.50
Books	Novel : The Great SQL	25.00
Electronics	Laptop Pro	1200.00
Electronics	Wireless Earbuds	150.00
Home Goods	Blender X	120.00

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.

-> **Task 1: Top 5 Customers by Total Amount Spent**

```
 SELECT
    CONCAT(c.first_name, ' ', c.last_name) AS CustomerName,
    c.email,
    SUM(p.amount) AS TotalAmountSpent
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
    TotalAmountSpent DESC
LIMIT 5;
```

Task 2: Top 3 Movie Categories with Highest Rental Counts

```
SELECT
    cat.name AS CategoryName,
    COUNT(r.rental_id) AS RentalCount
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
    RentalCount DESC
LIMIT 3;
```

Task 3: Films Available per Store and Never Rented Films

```
SELECT
    s.store_id,
    COUNT(DISTINCT i.film_id) AS TotalFilmsAvailable,
    COUNT(DISTINCT CASE
        WHEN r.rental_id IS NULL THEN i.film_id
    END) AS NeverRentedFilms
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;
```

Task 4: Total Revenue per Month for the Year 2023

```
SELECT
    MONTH(payment_date) AS Month,
    SUM(amount) AS TotalRevenue
FROM payment
WHERE YEAR(payment_date) = 2023
GROUP BY
    MONTH(payment_date)
ORDER BY
    Month;
```

Task 5: Customers with More Than 10 Rentals in the Last 6 Months

```
SELECT
    CONCAT(c.first_name, ' ', c.last_name) AS CustomerName,
    c.email,
    COUNT(r.rental_id) AS TotalRentals
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, c.email
HAVING
    COUNT(r.rental_id) > 10;
```