Introduction_to_SQL

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) is used to define or change the structure of database objects like tables.

Example:

CREATE TABLE Students (ID INT, Name VARCHAR(50));

* DML (Data Manipulation Language) is used to insert, update, or delete data from tables.

Example:

INSERT INTO Students VALUES (1, 'John');

* DQL (Data Query Language) is used to fetch data from the database.

Example:

SELECT * FROM Students;

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 valid and consistent data is stored in the database.

* PRIMARY KEY: Uniquely identifies each row and does not allow NULL values.

Example: StudentID in a Students table.

* UNIQUE: Ensures all values in a column are different.

Example: Email column in a Users table must be unique.

* FOREIGN KEY: Creates a relationship between two tables.

Example: CustomerID in Orders table refers to CustomerID in Customers table.

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?

* The LIMIT clause is used to restrict the number of rows returned in a result, while OFFSET skips a specified number of rows before starting to return rows.
* To get the 3rd page of results with 10 records per page, you skip the first 20 records and show the next 10:

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

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 CTE (Common Table Expression) is a temporary result set that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement. It makes complex queries easier to read and manage.

Example:

```
WITH HighSalary AS (
  SELECT name, salary FROM Employees WHERE salary > 50000
)
SELECT * FROM HighSalary;
```

* This creates a temporary result named HighSalary and then selects from it.

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

* Normalization in SQL is the process of organizing data to reduce redundancy and improve data integrity.
* 1NF (First Normal Form): Ensures each column has atomic (single) values and no repeating groups.

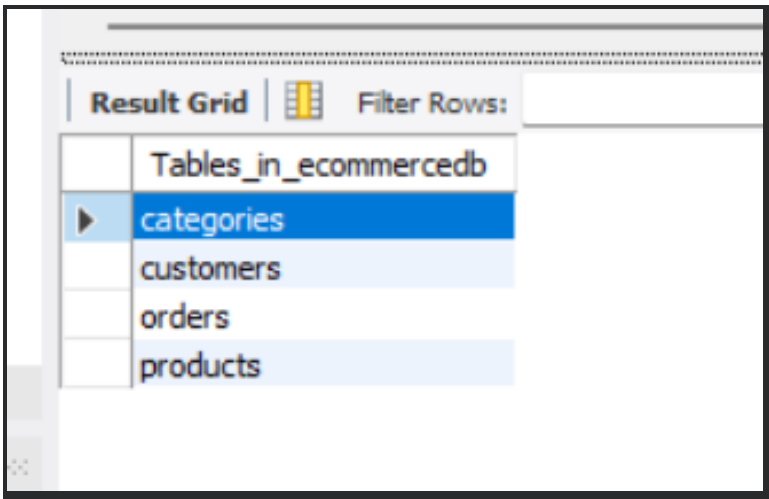
* 2NF (Second Normal Form): Must be in 1NF and all non-key columns must depend on the whole primary key (applies to composite keys).
* 3NF (Third Normal Form): Must be in 2NF and no non-key column should depend on another non-key column.

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

1. Create the following tables with appropriate data types and constraints:

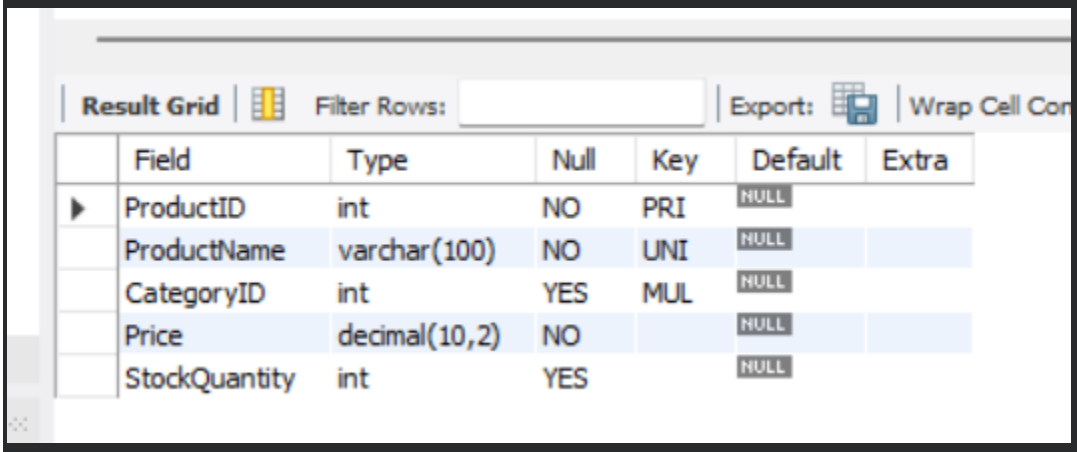
```
-- Create Database (skip this part if using an online editor)
CREATE DATABASE ECommerceDB;
USE ECommerceDB;

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

-- Create 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)
);
```

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

-- Create 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)
);

SHOW TABLES;

DESCRIBE Categories;
DESCRIBE Products;
DESCRIBE Customers;
DESCRIBE Orders;
```



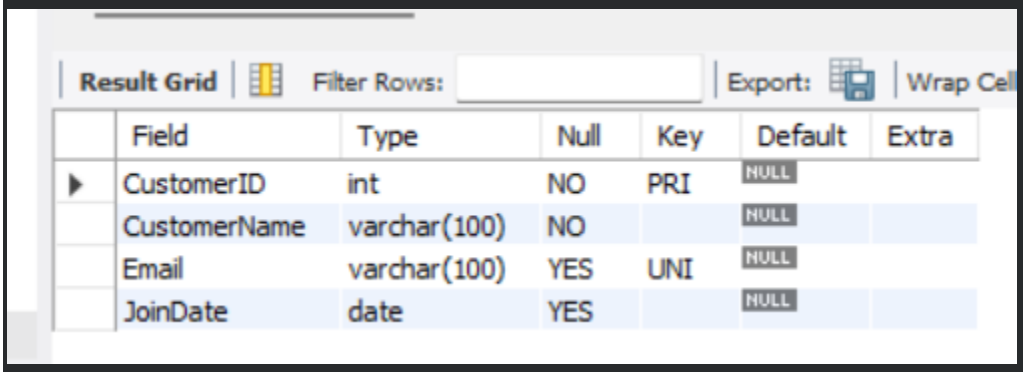
Result Grid | Filter Rows:

Tables_in_ecommercedb
categories
customers
orders
products



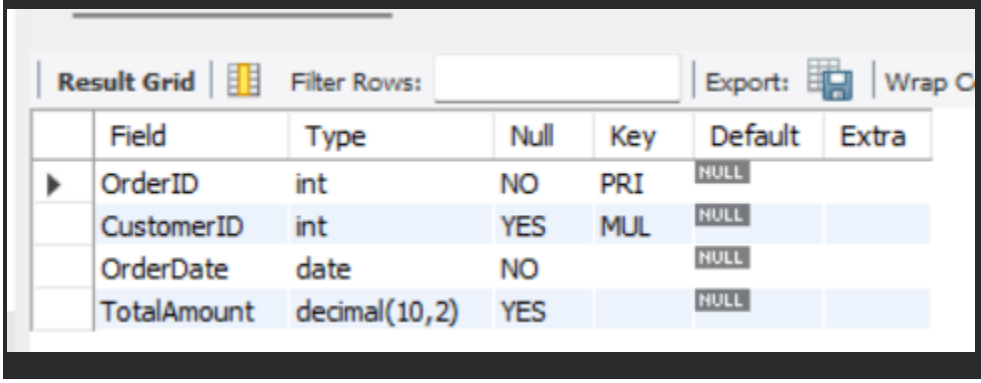
Result Grid | Filter Rows: | Export: | Wrap Cell Con

Field	Type	Null	Key	Default	Extra
ProductID	int	NO	PRI	NULL	
ProductName	varchar(100)	NO	UNI	NULL	
CategoryID	int	YES	MUL	NULL	
Price	decimal(10,2)	NO		NULL	
StockQuantity	int	YES		NULL	



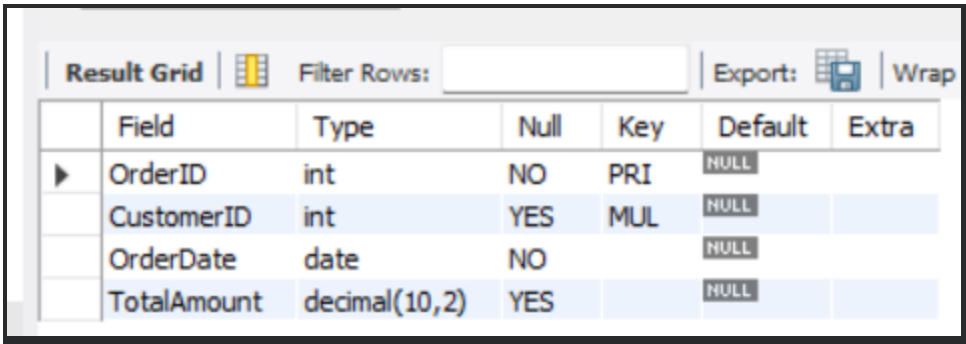
Result Grid | Filter Rows: | Export: | Wrap Cell

Field	Type	Null	Key	Default	Extra
CustomerID	int	NO	PRI	NULL	
CustomerName	varchar(100)	NO		NULL	
Email	varchar(100)	YES	UNI	NULL	
JoinDate	date	YES		NULL	



Result Grid | Filter Rows: | Export: | Wrap C

Field	Type	Null	Key	Default	Extra
OrderID	int	NO	PRI	NULL	
CustomerID	int	YES	MUL	NULL	
OrderDate	date	NO		NULL	
TotalAmount	decimal(10,2)	YES		NULL	



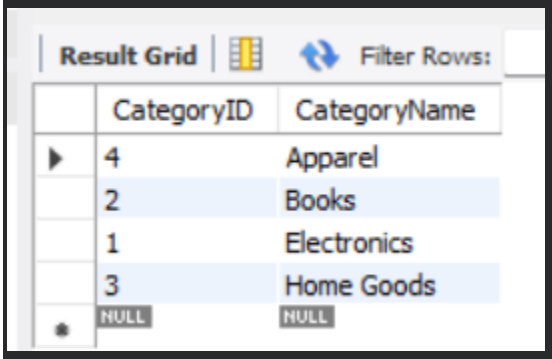
Result Grid | Filter Rows: | Export: | Wrap

Field	Type	Null	Key	Default	Extra
OrderID	int	NO	PRI	NULL	
CustomerID	int	YES	MUL	NULL	
OrderDate	date	NO		NULL	
TotalAmount	decimal(10,2)	YES		NULL	

2. Insert the following records into each table

```
INSERT INTO Categories (CategoryID, CategoryName) VALUES
(1, 'Electronics'),
(2, 'Books'),
(3, 'Home Goods'),
(4, 'Apparel');

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 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 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);

SELECT * FROM Categories;
SELECT * FROM Products;
SELECT * FROM Customers;
SELECT * FROM Orders;
```



Result Grid | Filter Rows: | Edit: | Export/

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
NULL	NULL	NULL	NULL	NULL

Result Grid | Filter Rows: | Edit:

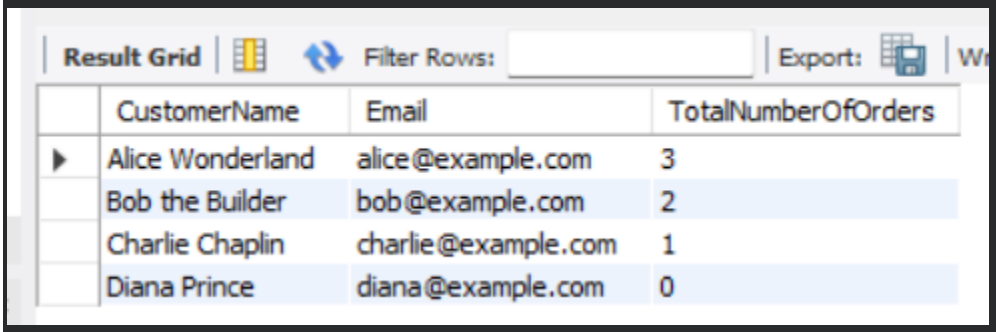
CustomerID	CustomerName	Email	JoinDate
1	Alice Wonderland	alice@example.com	2023-01-10
2	Bob the Builder	bob@example.com	2022-11-25
3	Charlie Chaplin	charlie@example.com	2023-03-01
4	Diana Prince	diana@example.com	2021-04-26
NULL	NULL	NULL	NULL

Result Grid | Filter Rows: | Edit:

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
NULL	NULL	NULL	NULL

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.

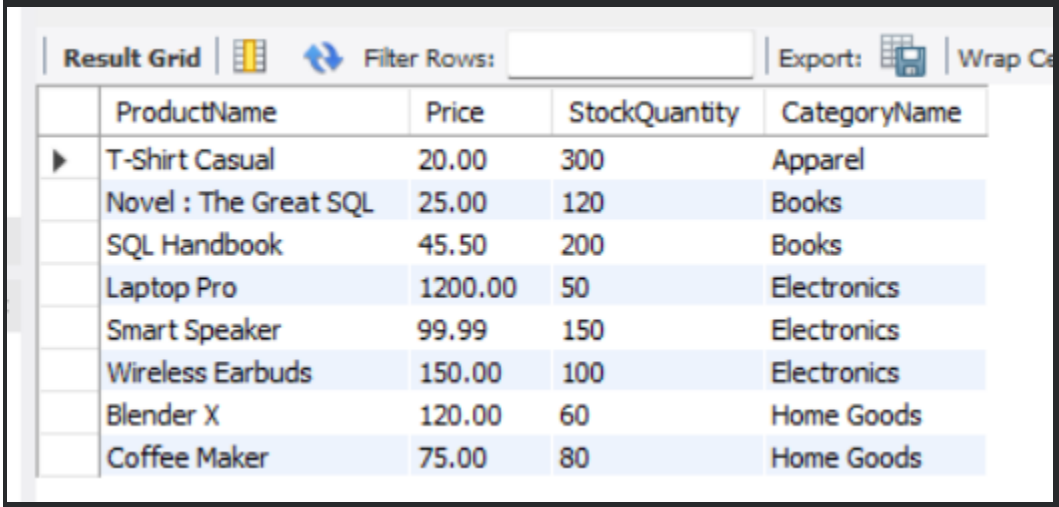
```
SELECT
    c.CustomerName,
    c.Email,
    COUNT(o.OrderID) AS TotalNumberOfOrders
FROM
    Customers c
LEFT JOIN
    Orders o
    ON c.CustomerID = o.CustomerID
GROUP BY
    c.CustomerID, c.CustomerName, c.Email
ORDER BY
    c.CustomerName ASC;
```



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. 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
INNER JOIN
    Categories c
    ON p.CategoryID = c.CategoryID
ORDER BY
    c.CategoryName ASC,
    p.ProductName ASC;
```

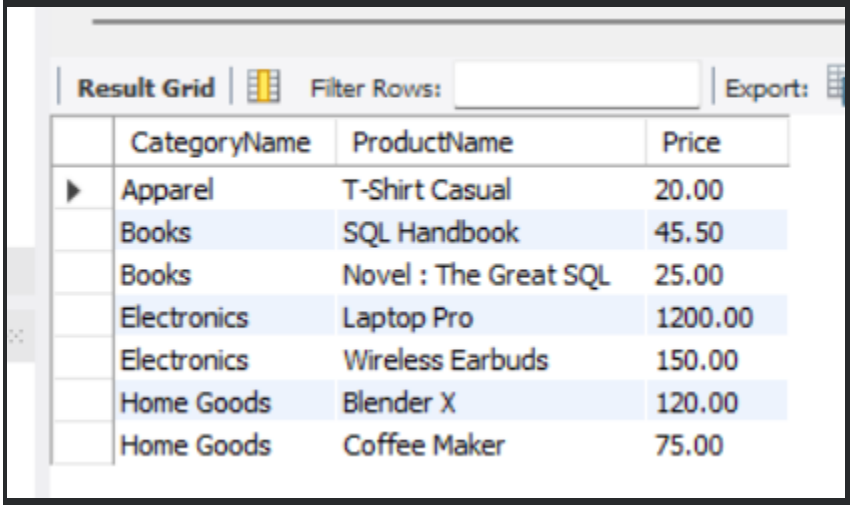


ProductName	Price	StockQuantity	CategoryName
T-Shirt Casual	20.00	300	Apparel
Novel : The Great SQL	25.00	120	Books
SQL Handbook	45.50	200	Books
Laptop Pro	1200.00	50	Electronics
Smart Speaker	99.99	150	Electronics
Wireless Earbuds	150.00	100	Electronics
Blender X	120.00	60	Home Goods
Coffee Maker	75.00	80	Home Goods

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 RankNo
    FROM
        Products p
    INNER JOIN
        Categories c
        ON p.CategoryID = c.CategoryID
)
SELECT
    CategoryName,
    ProductName,
    Price
FROM
    RankedProducts
WHERE
    RankNo <= 2
ORDER BY
```

```
    CategoryName ASC,
    Price DESC;
```



Result Grid | Filter Rows: | Export:

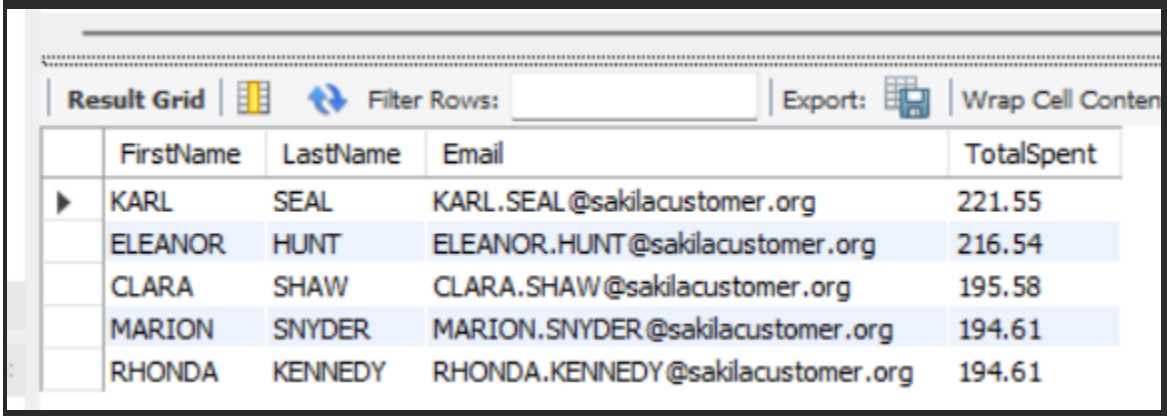
CategoryName	ProductName	Price
Apparel	T-Shirt Casual	20.00
Books	SQL Handbook	45.50
Books	Novel : The Great SQL	25.00
Electronics	Laptop Pro	1200.00
Electronics	Wireless Earbuds	150.00
Home Goods	Blender X	120.00
Home Goods	Coffee Maker	75.00

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.

Identify the top 5 customers based on the total amount they've spent. Include customer name, email, and total amount spent.

```
SELECT
    c.first_name AS FirstName,
    c.last_name AS LastName,
    c.email AS Email,
    SUM(p.amount) AS TotalSpent
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
    TotalSpent DESC
LIMIT 5;
```



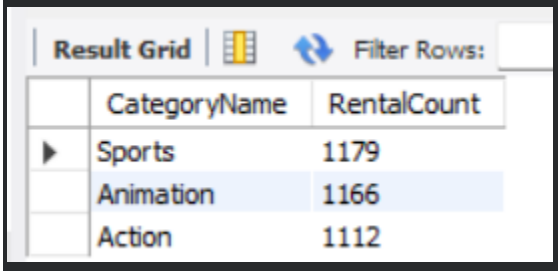
Result Grid | Filter Rows: | Export: | Wrap Cell Conten

FirstName	LastName	Email	TotalSpent
KARL	SEAL	KARL.SEAL@sakilacustomer.org	221.55
ELEANOR	HUNT	ELEANOR.HUNT@sakilacustomer.org	216.54
CLARA	SHAW	CLARA.SHAW@sakilacustomer.org	195.58
MARION	SNYDER	MARION.SNYDER@sakilacustomer.org	194.61
RHONDA	KENNEDY	RHONDA.KENNEDY@sakilacustomer.org	194.61

Find the top 3 movie categories with the highest rental counts. Display the category name and number of rentals.

```
SELECT
    cat.name AS CategoryName,
    COUNT(r.rental_id) AS RentalCount
FROM
    rental r
JOIN
    inventory i ON r.inventory_id = i.inventory_id
JOIN
    film f ON i.film_id = f.film_id
JOIN
    film_category fc ON f.film_id = fc.film_id
JOIN
    category cat ON fc.category_id = cat.category_id
GROUP BY
    cat.name
```

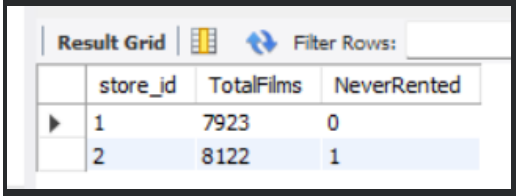
```
ORDER BY
   RentalCount DESC
LIMIT 3;
```



CategoryName	RentalCount
Sports	1179
Animation	1166
Action	1112

Calculate how many films are available at each store and how many of those have never been rented.

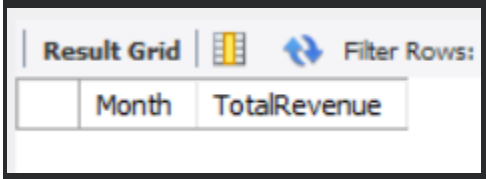
```
SELECT
   s.store_id,
   COUNT(i.inventory_id) AS TotalFilms,
   SUM(CASE WHEN r.rental_id IS NULL THEN 1 ELSE 0 END) AS
NeverRented
FROM
   store s
JOIN
   inventory i ON s.store_id = i.store_id
LEFT JOIN
   rental r ON i.inventory_id = r.inventory_id
GROUP BY
   s.store_id;
```



store_id	TotalFilms	NeverRented
1	7923	0
2	8122	1

Show the total revenue per month for the year 2023 to analyze business seasonality.

```
SELECT
   DATE_FORMAT(p.payment_date, '%Y-%m') AS Month,
   SUM(p.amount) AS TotalRevenue
FROM
   payment p
WHERE
   YEAR(p.payment_date) = 2023
GROUP BY
   DATE_FORMAT(p.payment_date, '%Y-%m')
ORDER BY
   Month;
```



Identify customers who have rented more than 10 times in the last 6 months.

```
SELECT
   c.first_name AS FirstName,
   c.last_name AS LastName,
   c.email AS Email,
   COUNT(r.rental_id) AS RentalsInLast6Months
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
ORDER BY
   RentalsInLast6Months DESC;
```