Analytics Avenue for Research and Development

-In the journey of empowering the digital minds

Worksheet-4

Windows Function

1. Finding Top 3 Salaries with Ties Within Each Department

Table Schema: Employees: - EmployeeID (INT): Unique identifier for each employee. - Name (VARCHAR): Employee's name. - Salary (DECIMAL): Employee's salary. - Department (VARCHAR): The department in which the employee works.

Question: Find the top 3 highest-paid employees in each department, including employees tied at the same salary level. Use the RANK() function to calculate the rank based on salary within each department. Return the employee's EmployeeID, Name, Salary, Department, and their rank.

ANSWER:

```
WITH RankedEmployees AS (
   SELECT
       EmployeeID,
       Name,
       Salary,
       Department,
       RANK() OVER (PARTITION BY Department ORDER BY Salary DESC) AS Rank
   FROM Employees
)
SELECT
   EmployeeID,
   Name,
   Salary,
   Department,
   Rank
FROM RankedEmployees
WHERE Rank <= 3
```

```
ORDER BY Department, Rank;
```

2. Finding the Employee with the Second-Highest Salary Without Using RANK

Table Schema: Employees: - EmployeeID (INT): Unique identifier for each employee. - Name (VARCHAR): Employee's name. - Salary (DECIMAL): Employee's salary.

Question:

Write a query to find the employee(s) with the second-highest salary, but without using RANK() or DENSE_RANK(). Return the EmployeeID, Name, and Salary.

ANSWER:

```
SELECT EmployeeID, Name, Salary
FROM Employees
WHERE Salary = (
  SELECT MAX(Salary)
  FROM Employees
  WHERE Salary < (SELECT MAX(Salary) FROM Employees)
);
```

3. Rolling Sum Over the Last 3 Transactions for Each Customer

Table Schema: Transactions:

- TransactionID (INT): Unique identifier for each transaction. - CustomerID (INT): Identifier for the customer making the transaction. - Amount (DECIMAL): The amount of the transaction. - TransactionDate (DATE): Date of the transaction. **Question:** For each customer, calculate a rolling sum of their transaction amounts for the last 3 transactions. Use the SUM() function with a window frame to calculate this rolling sum. Return the TransactionID, CustomerID, Amount, and RollingSum for each transaction.

ANSWER:

```
SELECT
  TransactionID,
  CustomerID,
  Amount,
  SUM(Amount) OVER (
        PARTITION BY CustomerID
        ORDER BY TransactionDate
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ) AS RollingSum
FROM Transactions;
```

4. Finding the Highest and Lowest First Transaction for Each Customer

Table Schema: Sales: - CustomerID (INT): Unique identifier for the customer. - SaleDate (DATE): Date of the sale. - Amount (DECIMAL): The sale amount.

Question:

Find the customers who have the highest and lowest first transaction amounts. Use the FIRST_VALUE() function to find each customer's first transaction based on SaleDate, then find the maximum and minimum of these first transactions. Return the CustomerID and their first transaction amount.

ANSWER:

```
WITH FirstTransactions AS (
    SELECT
        CustomerID,
        FIRST_VALUE(Amount) OVER (
            PARTITION BY CustomerID
            ORDER BY SaleDate
        ) AS FirstTransactionAmount
    FROM Sales
)
SELECT
    CustomerID,
    FirstTransactionAmount
FROM FirstTransactions
WHERE FirstTransactionAmount = (
    SELECT MAX(FirstTransactionAmount) FROM FirstTransactions
)
```

```
OR FirstTransactionAmount = (
   SELECT MIN(FirstTransactionAmount) FROM FirstTransactions
);
```

5. Alternating RANK Between Ascending and Descending Order Within Each Category

Table Schema: Products: - ProductID (INT): Unique identifier for each product. - Category (VARCHAR): The category to which the product belongs. - Price (DECIMAL): Price of the product.

Question: Write a query to assign alternating ranks to products within each category. The highest price should be ranked first, followed by the lowest price, and this pattern should continue alternately. Use ROW_NUMBER() to create alternating ranks. Return the ProductID, Category, Price, and AlternatingRank.

ANSWER:

```
WITH RankedProducts AS (
   SELECT
      ProductID,
      Category,
      Price,
      ROW_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) AS DescendingRank,
      ROW_NUMBER() OVER (PARTITION BY Category ORDER BY Price ASC) AS AscendingRank
   FROM Products
)
SELECT
   ProductID,
   Category,
   Price,
   CASE
      WHEN DescendingRank % 2 = 1 THEN DescendingRank
      ELSE AscendingRank
   END AS AlternatingRank
FROM RankedProducts
ORDER BY Category, AlternatingRank;
```

6. Employees Whose Salary is Higher than the Next Employee

Table Schema:

Employees: - EmployeeID (INT): Unique identifier for each employee. - Name (VARCHAR): Employee's name. - Salary (DECIMAL): Employee's salary.

Question:

Find the employees whose salary is higher than the salary of the next employee (when sorted by EmployeeID). Use the LEAD() function to compare the current employee's salary with the salary of the next employee. Return the EmployeeID, Name, and Salary of the employees who meet this condition.

ANSWER:

```
SELECT
  EmployeeID,
  Name,
  Salary
FROM
  (SELECT
    EmployeeID,
    Name,
    Salary,
    LEAD(Salary) OVER (ORDER BY EmployeeID) AS NextSalary
  FROM Employees) AS EmployeeWithNext
WHERE
  Salary > NextSalary;
```

7. Calculate Year-to-Date Sales for Each Product

Table Schema: Sales: - SaleID (INT): Unique identifier for each sale. - ProductID (INT): Identifier for the product sold. - SaleDate (DATE): Date of the sale. - Amount (DECIMAL): Amount of the sale.

Question: For each product, calculate the year-to-date (YTD) sales by summing up the sales from the beginning of the year until the current SaleDate. Use the SUM() window function with a frame to calculate the cumulative sum. Return the ProductID, SaleDate, Amount, and YearToDateSales.

ANSWER:

```
SELECT
  ProductID,
    SaleDate,
    Amount,
    SUM(Amount) OVER (
        PARTITION BY ProductID
        ORDER BY SaleDate
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS YearToDateSales
FROM Sales
WHERE YEAR(SaleDate) = YEAR(CURDATE()) -- Ensure only the current year's sales are considered
ORDER BY ProductID, SaleDate;
```

8. Ranking Employees by Hire Date and Salary Within a Department

Table Schema: Employees: - EmployeeID (INT): Unique identifier for each employee. - Name (VARCHAR): Employee's name. - Salary (DECIMAL): Employee's salary. - HireDate (DATE): Date when the employee was hired. - Department (VARCHAR): The department in which the employee works.

Question: Write a query to rank employees within each department based on their hire date (oldest first) and, in case of ties, based on their salary (highest first). Use RANK() and PARTITION

BY department to calculate this rank. Return EmployeeID, Name, HireDate, Salary, Department, and Rank.

ANSWER:

```
SELECT
    EmployeeID,
    Name,
    HireDate,
    Salary,
    Department,
    RANK() OVER (
        PARTITION BY Department
        ORDER BY HireDate ASC, Salary DESC
    ) AS Rank
```

```
FROM Employees;
```

9. Identifying Employees Who Earn More Than Their Department's Average Salary

Table Schema: Employees: - EmployeeID (INT): Unique identifier for each employee. - Name (VARCHAR): Employee's name. - Salary (DECIMAL): Employee's salary. - Department (VARCHAR): The department in which the employee works.

Question: For each department, calculate the average salary, then find employees whose salary is greater than their department's average. Use AVG() as a window function to calculate the average salary per department. Return the EmployeeID, Name, Salary, Department, and AverageDeptSalary.

ANSWER:

```
SELECT
   EmployeeID,
   Name,
   Salary,
   Department,
   AVG(Salary) OVER (PARTITION BY Department) AS AverageDeptSalary
FROM Employees
WHERE Salary > AVG(Salary) OVER (PARTITION BY Department)
ORDER BY Department, Salary DESC;
```

10. Finding the Difference Between the Current and Previous Transaction for Each Customer

Table Schema: Transactions: - TransactionID (INT): Unique identifier for each transaction. - CustomerID (INT): Unique identifier for the customer making the transaction. - Amount (DECIMAL): Amount of the transaction. - TransactionDate (DATE): Date of the transaction. **Question:** Write a query to calculate the difference between the current and previous transaction amount for each customer, ordered by TransactionDate. Use the LAG() function to get the previous transaction amount. Return the TransactionID, CustomerID, Amount, and the difference (Amount - PreviousAmount) as AmountDifference.

ANSWER:

```
SELECT
   TransactionID,
   CustomerID,
   Amount,
    Amount - LAG(Amount) OVER (PARTITION BY CustomerID ORDER BY TransactionDate) AS
AmountDifference

FROM Transactions

ORDER BY CustomerID, TransactionDate;
```