Exercise 1.

-- Using ROW\_NUMBER

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10,2)

);

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES

(1, 'Laptop', 'Electronics', 1200.00),

(2, 'Smartphone', 'Electronics', 900.00),

(3, 'Headphones', 'Electronics', 150.00),

(4, 'Chair', 'Furniture', 200.00),

(5, 'Desk', 'Furniture', 350.00),

(6, 'Bookshelf', 'Furniture', 150.00),

(7, 'Monitor', 'Electronics', 300.00),

(8, 'Table', 'Furniture', 500.00);

SELECT \*

FROM (

SELECT

ProductID,

ProductName,

Category,

Price,

ROW\_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) AS RowNum

FROM Products

) AS Ranked

WHERE RowNum <= 3;

-- Using RANK

SELECT \*

FROM (

SELECT

ProductID,

ProductName,

Category,

Price,

RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS RankNum

FROM Products

) AS Ranked

WHERE RankNum <= 3;

-- Using DENSE\_RANK

SELECT \*

FROM (

SELECT

ProductID,

ProductName,

Category,

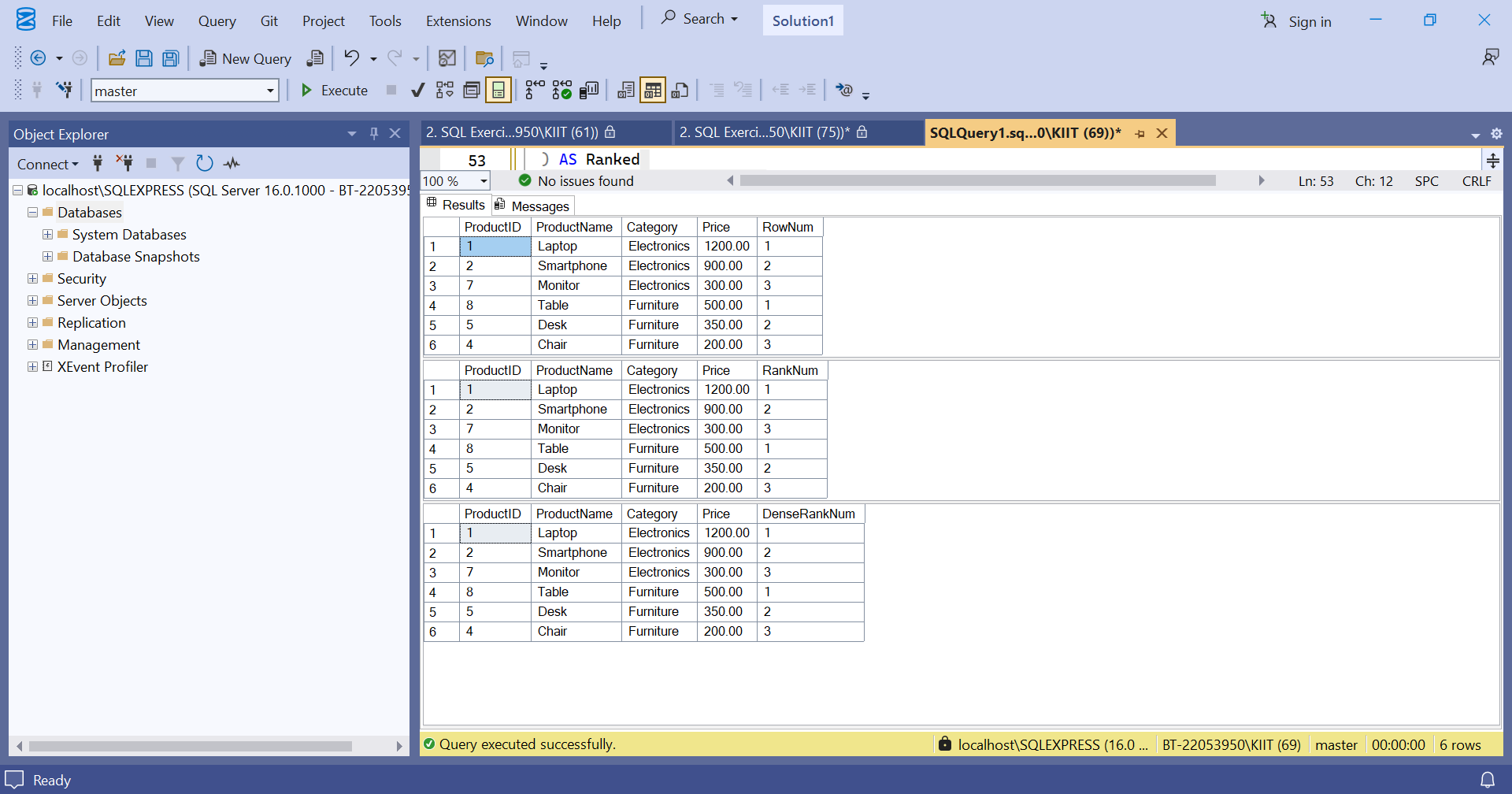
Price,

DENSE\_RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS DenseRankNum

FROM Products

) AS Ranked

WHERE DenseRankNum <= 3;

Exercise 2.

DROP TABLE IF EXISTS OrderDetails, Orders, Products, Customers;

-- Create Customers

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100),

Region VARCHAR(50)

);

INSERT INTO Customers (CustomerID, CustomerName, Region) VALUES

(1, 'Alice', 'North'),

(2, 'Bob', 'South'),

(3, 'Charlie', 'East');

-- Create Products

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50)

);

INSERT INTO Products (ProductID, ProductName, Category) VALUES

(1, 'Laptop', 'Electronics'),

(2, 'Desk', 'Furniture'),

(3, 'Monitor', 'Electronics');

-- Create Orders

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES

(1, 1, '2025-06-01'),

(2, 2, '2025-06-02'),

(3, 3, '2025-06-03');

-- Create OrderDetails

CREATE TABLE OrderDetails (

OrderDetailID INT PRIMARY KEY,

OrderID INT,

ProductID INT,

Quantity INT,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

INSERT INTO OrderDetails (OrderDetailID, OrderID, ProductID, Quantity) VALUES

(1, 1, 1, 2),

(2, 1, 2, 1),

(3, 2, 2, 3),

(4, 3, 3, 2);

SELECT

c.Region,

p.Category,

SUM(od.Quantity) AS TotalQuantity

FROM Orders o

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Customers c ON o.CustomerID = c.CustomerID

JOIN Products p ON od.ProductID = p.ProductID

GROUP BY GROUPING SETS (

(c.Region),

(p.Category),

(c.Region, p.Category)

);

-- Using ROLLUP

SELECT

c.Region,

p.Category,

SUM(od.Quantity) AS TotalQuantity

FROM Orders o

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Customers c ON o.CustomerID = c.CustomerID

JOIN Products p ON od.ProductID = p.ProductID

GROUP BY ROLLUP (c.Region, p.Category);

-- Using CUBE

SELECT

c.Region,

p.Category,

SUM(od.Quantity) AS TotalQuantity

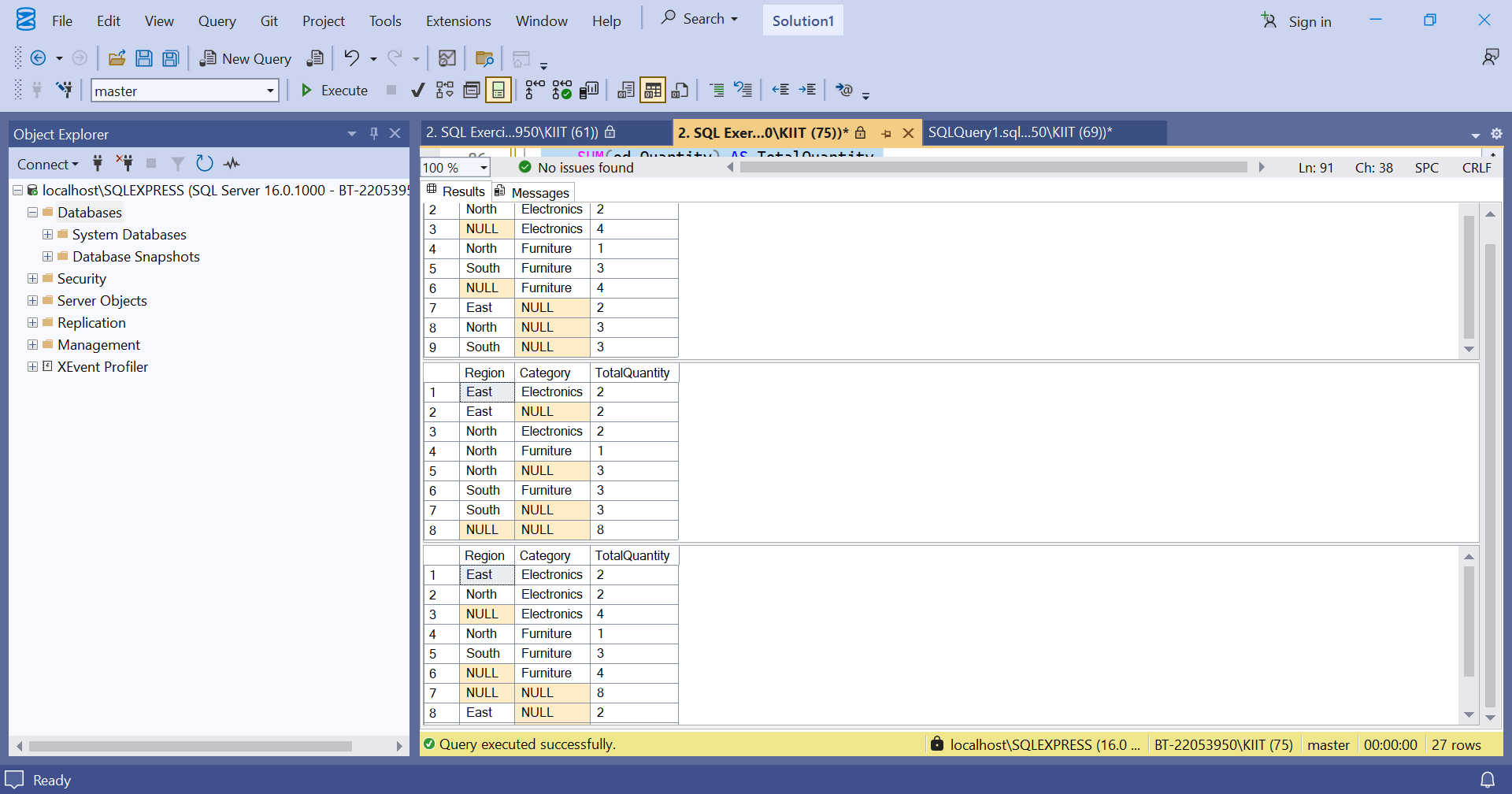
FROM Orders o

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Customers c ON o.CustomerID = c.CustomerID

JOIN Products p ON od.ProductID = p.ProductID

GROUP BY CUBE (c.Region, p.Category);

Exercise 3.

-- 1. Clean up existing tables (optional)

DROP TABLE IF EXISTS StagingProducts;

DROP TABLE IF EXISTS Products;

-- 2. Create Products table with Price column

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2) -- Must be here!

);

-- 3. Insert sample products

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES

(1, 'Laptop', 'Electronics', 1000.00),

(2, 'Desk', 'Furniture', 300.00),

(3, 'Monitor', 'Electronics', 250.00);

-- 4. Create StagingProducts table with Price column

CREATE TABLE StagingProducts (

ProductID INT,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2) -- Must be here too!

);

-- 5. Insert updated/new products in staging

INSERT INTO StagingProducts (ProductID, ProductName, Category, Price) VALUES

(2, 'Desk', 'Furniture', 320.00), -- Update price

(3, 'Monitor', 'Electronics', 250.00), -- No change

(4, 'Chair', 'Furniture', 150.00); -- New product

-- 6. Recursive CTE to generate calendar from 2025-01-01 to 2025-01-31

WITH Calendar AS (

SELECT CAST('2025-01-01' AS DATE) AS DateValue

UNION ALL

SELECT DATEADD(DAY, 1, DateValue)

FROM Calendar

WHERE DateValue < '2025-01-31'

)

SELECT \* FROM Calendar

OPTION (MAXRECURSION 0);

-- 7. MERGE statement: update existing or insert new products

MERGE INTO Products AS Target

USING StagingProducts AS Source

ON Target.ProductID = Source.ProductID

WHEN MATCHED THEN

UPDATE SET

ProductName = Source.ProductName,

Category = Source.Category,

Price = Source.Price

WHEN NOT MATCHED BY TARGET THEN

INSERT (ProductID, ProductName, Category, Price)

VALUES (Source.ProductID, Source.ProductName, Source.Category, Source.Price);

-- 8. Verify final Products table

SELECT \* FROM Products ORDER BY ProductID;

Exercise 4.

-- Drop existing Sales table if exists

DROP TABLE IF EXISTS Sales;

-- Step 0: Create Sales table and insert sample data

CREATE TABLE Sales (

Product VARCHAR(50),

OrderDate DATE,

Quantity INT

);

INSERT INTO Sales (Product, OrderDate, Quantity) VALUES

('Laptop', '2025-01-15', 10),

('Laptop', '2025-02-20', 12),

('Laptop', '2025-03-10', 8),

('Desk', '2025-01-05', 5),

('Desk', '2025-02-17', 7),

('Chair', '2025-01-22', 3);

-- Step 1: Aggregate sales by Product and Month

WITH MonthlySales AS (

SELECT

Product,

DATENAME(MONTH, OrderDate) AS MonthName,

SUM(Quantity) AS TotalQuantity

FROM Sales

GROUP BY Product, DATENAME(MONTH, OrderDate)

),

-- Step 2: Pivot monthly sales (months as columns)

PivotedSales AS (

SELECT Product, [January], [February], [March]

FROM (

SELECT Product, MonthName, TotalQuantity

FROM MonthlySales

) AS SourceTable

PIVOT (

SUM(TotalQuantity)

FOR MonthName IN ([January], [February], [March])

) AS PivotTable

)

-- Step 3: Unpivot the pivoted data back into rows

SELECT Product, Month, Quantity

FROM PivotedSales

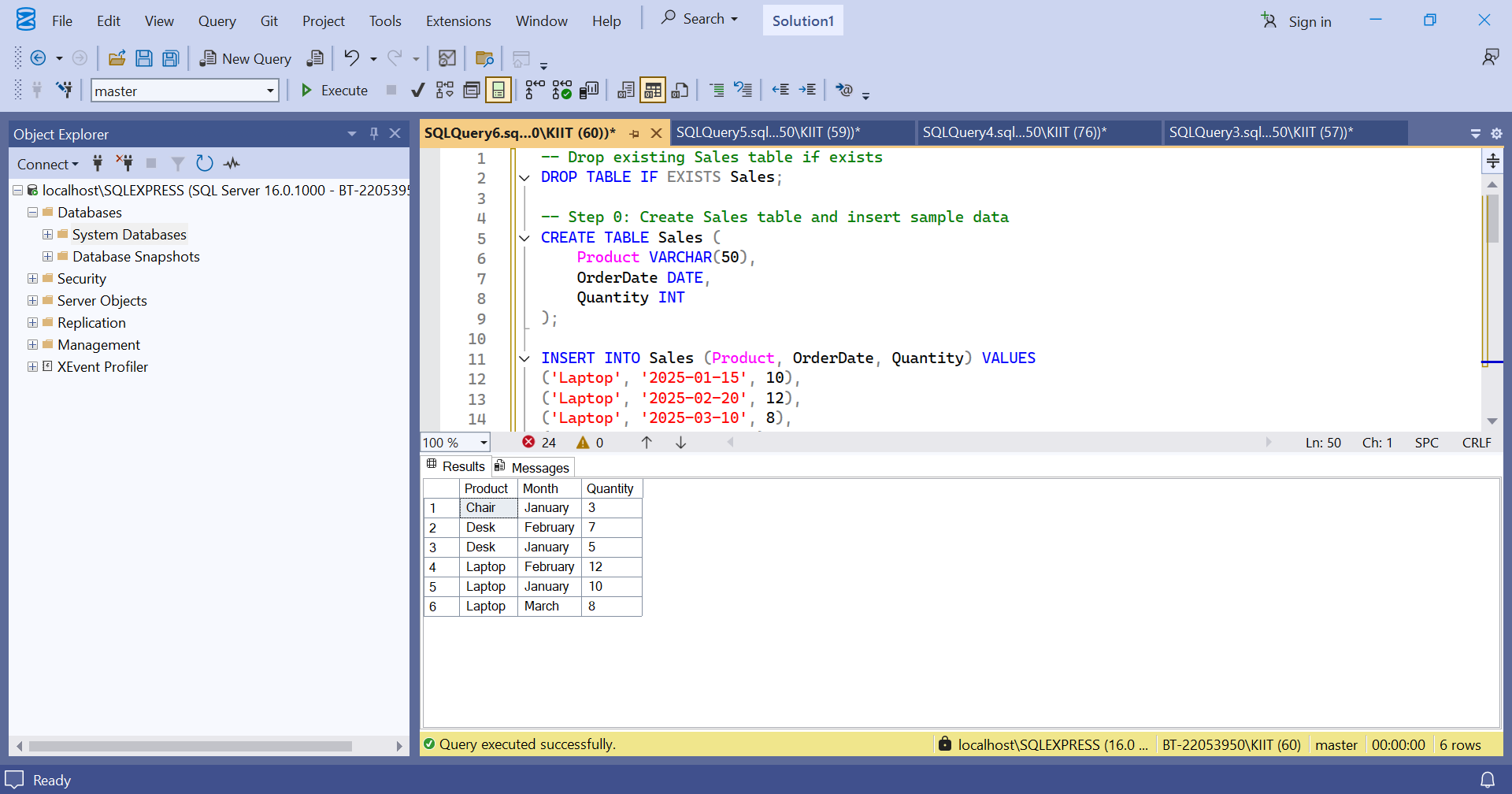
UNPIVOT (

Quantity FOR Month IN ([January], [February], [March])

) AS UnpivotTable

WHERE Quantity IS NOT NULL

ORDER BY Product, Month;

Exercise 5.

WITH CustomerOrderCounts AS (

SELECT

o.CustomerID,

COUNT(o.OrderID) AS OrderCount

FROM Orders o

GROUP BY o.CustomerID

)

SELECT

c.CustomerID,

c.CustomerName, -- Change this to your actual column name

coc.OrderCount

FROM CustomerOrderCounts coc

JOIN Customers c ON c.CustomerID = coc.CustomerID

WHERE coc.OrderCount > 3;

Exercise 2.

-- Then run the Exercise 5 query

WITH CustomerOrderCounts AS (

SELECT

o.CustomerID,

COUNT(o.OrderID) AS OrderCount

FROM Orders o

GROUP BY o.CustomerID

)

SELECT

c.CustomerID,

c.Name,

coc.OrderCount

FROM CustomerOrderCounts coc

JOIN Customers c ON c.CustomerID = coc.CustomerID

WHERE coc.OrderCount > 3;

WITH CustomerOrderCounts AS (

SELECT

o.CustomerID,

COUNT(o.OrderID) AS OrderCount

FROM Orders o

GROUP BY o.CustomerID

)

SELECT

c.CustomerID,

c.Name,

coc.OrderCount

FROM CustomerOrderCounts coc

JOIN Customers c ON c.CustomerID = coc.CustomerID

WHERE coc.OrderCount > 3;

INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES

(5, 1, '2023-05-01'),

(6, 1, '2023-06-01'),

(7, 1, '2023-07-01');

Exercise 4.

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate)

VALUES (@FirstName, @LastName, @DepartmentID, @Salary, @JoinDate);

END;

-- Modify existing stored procedure to include Salary

ALTER PROCEDURE sp\_GetEmployeesByDept

@DepartmentID INT

AS

BEGIN

SELECT EmployeeID, FirstName, LastName, Salary

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

DROP PROCEDURE IF EXISTS sp\_InsertEmployee;

-- Assumes sp\_GetEmployeesByDept exists

EXEC sp\_GetEmployeesByDept @DepartmentID = 2;

CREATE PROCEDURE sp\_CountEmployeesByDept

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

CREATE PROCEDURE sp\_CountEmployeesByDept

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

CREATE PROCEDURE sp\_TotalSalaryByDept

@DepartmentID INT,

@TotalSalary DECIMAL(18,2) OUTPUT

AS

BEGIN

SELECT @TotalSalary = SUM(Salary)

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

-- Example usage

DECLARE @SalaryTotal DECIMAL(18,2);

EXEC sp\_TotalSalaryByDept @DepartmentID = 1, @TotalSalary = @SalaryTotal OUTPUT;

SELECT @SalaryTotal AS TotalSalary;

CREATE PROCEDURE sp\_UpdateEmployeeSalary

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

END;

-- Example usage

EXEC sp\_UpdateEmployeeSalary 1, 5500.00;

CREATE PROCEDURE sp\_GiveBonus

@DepartmentID INT,

@BonusAmount DECIMAL(10,2)

AS

BEGIN

UPDATE Employees

SET Salary = Salary + @BonusAmount

WHERE DepartmentID = @DepartmentID;

END;

-- Example usage

EXEC sp\_GiveBonus 1, 500.00;

CREATE PROCEDURE sp\_UpdateSalaryWithTransaction

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION;

THROW;

END CATCH

END;

CREATE PROCEDURE sp\_DynamicEmployeeSearch

@FilterColumn NVARCHAR(50),

@FilterValue NVARCHAR(100)

AS

BEGIN

DECLARE @SQL NVARCHAR(MAX);

SET @SQL = 'SELECT \* FROM Employees WHERE ' + QUOTENAME(@FilterColumn) + ' = @value';

EXEC sp\_executesql @SQL, N'@value NVARCHAR(100)', @value = @FilterValue;

END;

-- Example usage

EXEC sp\_DynamicEmployeeSearch 'FirstName', 'John';

CREATE PROCEDURE sp\_SafeUpdateSalary

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

BEGIN TRY

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

END TRY

BEGIN CATCH

DECLARE @ErrorMessage NVARCHAR(4000);

SET @ErrorMessage = ERROR\_MESSAGE();

PRINT 'Error updating salary: ' + @ErrorMessage;

END CATCH

END;

Exercise 5.

-- =========================

-- Setup: Sample Tables and Data

-- =========================

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID),

Salary DECIMAL(10,2),

JoinDate DATE

);

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'), (2, 'IT'), (3, 'Finance');

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary, JoinDate) VALUES

(1, 'John', 'Doe', 1, 5000.00, '2020-01-15'),

(2, 'Jane', 'Smith', 2, 6000.00, '2019-03-22'),

(3, 'Bob', 'Johnson', 3, 5500.00, '2021-07-01');

-- =========================

-- Exercise 1: Scalar Function - Annual Salary

-- =========================

CREATE FUNCTION fn\_CalculateAnnualSalary (

@MonthlySalary DECIMAL(10,2)

)

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @MonthlySalary \* 12;

END;

-- =========================

-- Exercise 2: Table-Valued Function - Employees by Department

-- =========================

CREATE FUNCTION fn\_GetEmployeesByDepartment (

@DeptID INT

)

RETURNS TABLE

AS

RETURN

(

SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary, JoinDate

FROM Employees

WHERE DepartmentID = @DeptID

);

-- =========================

-- Exercise 3: Scalar Function - Calculate Bonus (10%)

-- =========================

CREATE FUNCTION fn\_CalculateBonus (

@Salary DECIMAL(10,2)

)

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.10;

END;

-- =========================

-- Exercise 4: Modify Bonus Function to 15%

-- =========================

ALTER FUNCTION fn\_CalculateBonus (

@Salary DECIMAL(10,2)

)

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.15;

END;

-- =========================

-- Exercise 5: Delete Bonus Function

-- =========================

DROP FUNCTION IF EXISTS fn\_CalculateBonus;

-- Re-create to allow next steps to work

CREATE FUNCTION fn\_CalculateBonus (

@Salary DECIMAL(10,2)

)

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.15;

END;

-- =========================

-- Exercise 6: Execute fn\_CalculateAnnualSalary for All Employees

-- =========================

SELECT

EmployeeID, FirstName, LastName, Salary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees;

-- =========================

-- Exercise 7: Annual Salary for EmployeeID = 1

-- =========================

SELECT

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees

WHERE EmployeeID = 1;

-- =========================

-- Exercise 8: Employees from Finance Department (ID = 3)

-- =========================

SELECT \* FROM dbo.fn\_GetEmployeesByDepartment(3);

-- =========================

-- Exercise 9: Nested Function - Total Compensation

-- =========================

CREATE FUNCTION fn\_CalculateTotalCompensation (

@Salary DECIMAL(10,2)

)

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN dbo.fn\_CalculateAnnualSalary(@Salary) + dbo.fn\_CalculateBonus(@Salary);

END;

-- =========================

-- Exercise 10: Use Modified Nested Function

-- =========================

SELECT

EmployeeID, FirstName, LastName, Salary,

dbo.fn\_CalculateTotalCompensation(Salary) AS TotalCompensation

FROM Employees;