Exercise 1A.

-- 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,

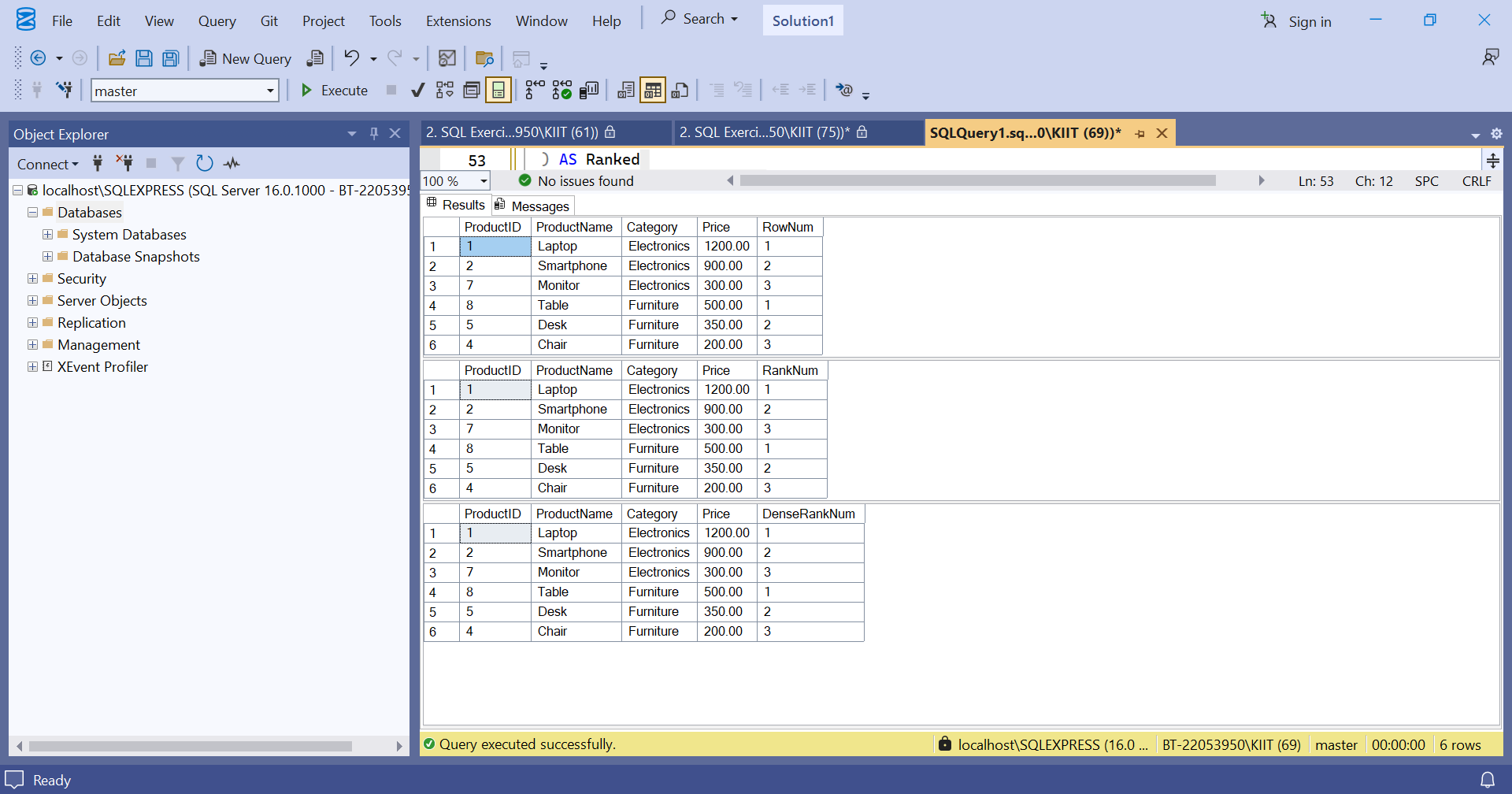
Price,

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

FROM Products

) AS Ranked

WHERE DenseRankNum <= 3;

Exercise 1B.

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

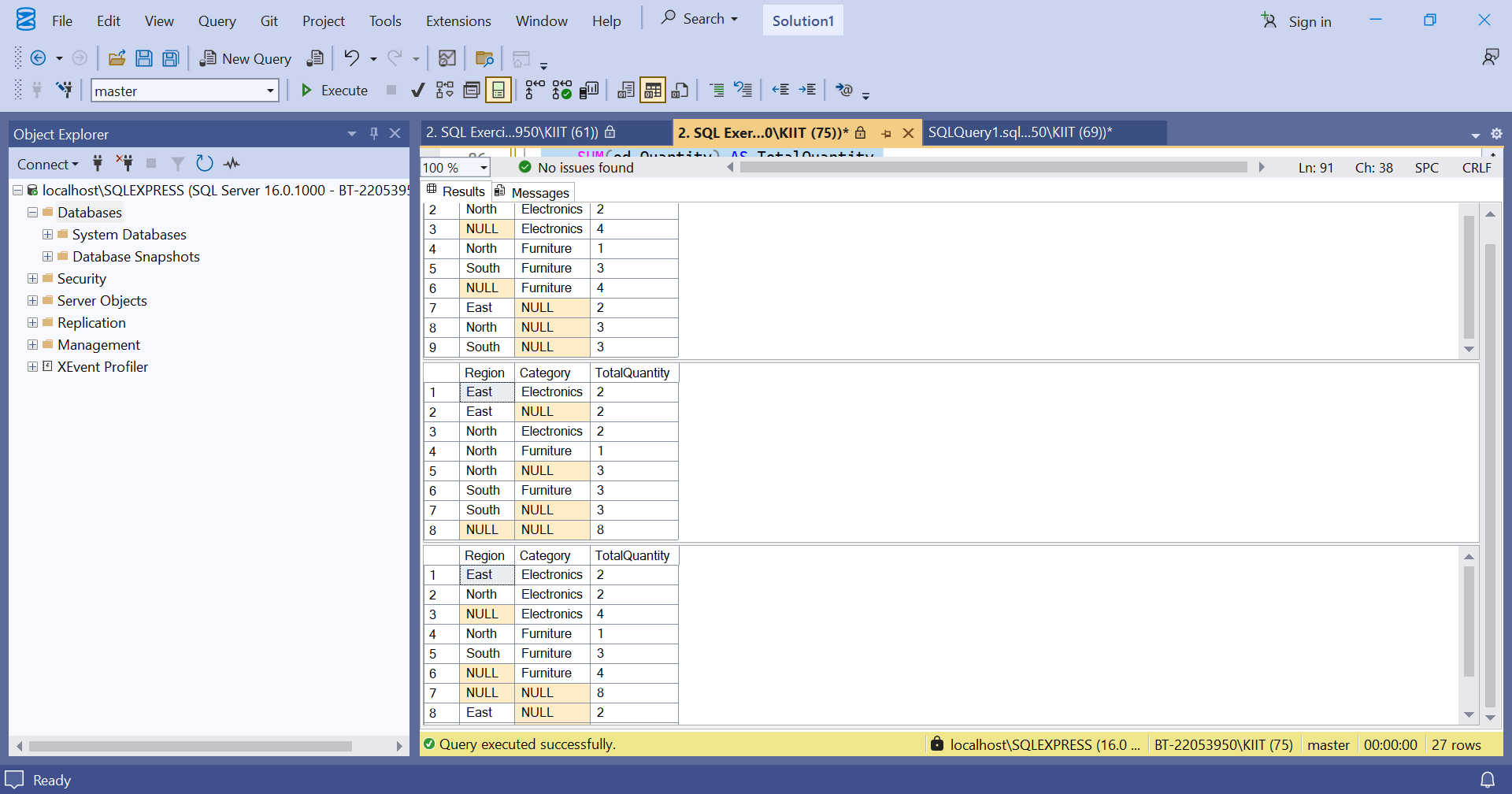
FROM Orders o

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 1C.

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

-- Step 1: Recursive CTE to generate calendar dates from 2025-01-01 to 2025-01-31

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

WITH CalendarCTE AS (

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

UNION ALL

SELECT DATEADD(DAY, 1, CalendarDate)

FROM CalendarCTE

WHERE CalendarDate < '2025-01-31'

)

SELECT \* FROM CalendarCTE

OPTION (MAXRECURSION 0);

GO

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

-- Step 2: Create StagingProducts table

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

IF OBJECT\_ID('StagingProducts', 'U') IS NOT NULL

DROP TABLE StagingProducts;

GO

CREATE TABLE StagingProducts (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2)

);

GO

-- Insert sample data into StagingProducts (some existing, some new)

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

(1, 'Laptop', 'Electronics', 1100.00), -- Existing product with updated price

(2, 'Smartphone', 'Electronics', 750.00), -- Existing product with updated price

(5, 'Monitor', 'Electronics', 300.00), -- New product

(6, 'Keyboard', 'Accessories', 45.00); -- New product

GO

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

-- Step 3: MERGE statement to update/insert products from staging

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

MERGE INTO Products AS Target

USING StagingProducts AS Source

ON Target.ProductID = Source.ProductID

WHEN MATCHED THEN

UPDATE SET

Target.ProductName = Source.ProductName,

Target.Category = Source.Category,

Target.Price = Source.Price

WHEN NOT MATCHED BY TARGET THEN

INSERT (ProductID, ProductName, Category, Price)

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

-- Optional: Output action results

-- OUTPUT $action AS MergeAction, inserted.\*;

GO

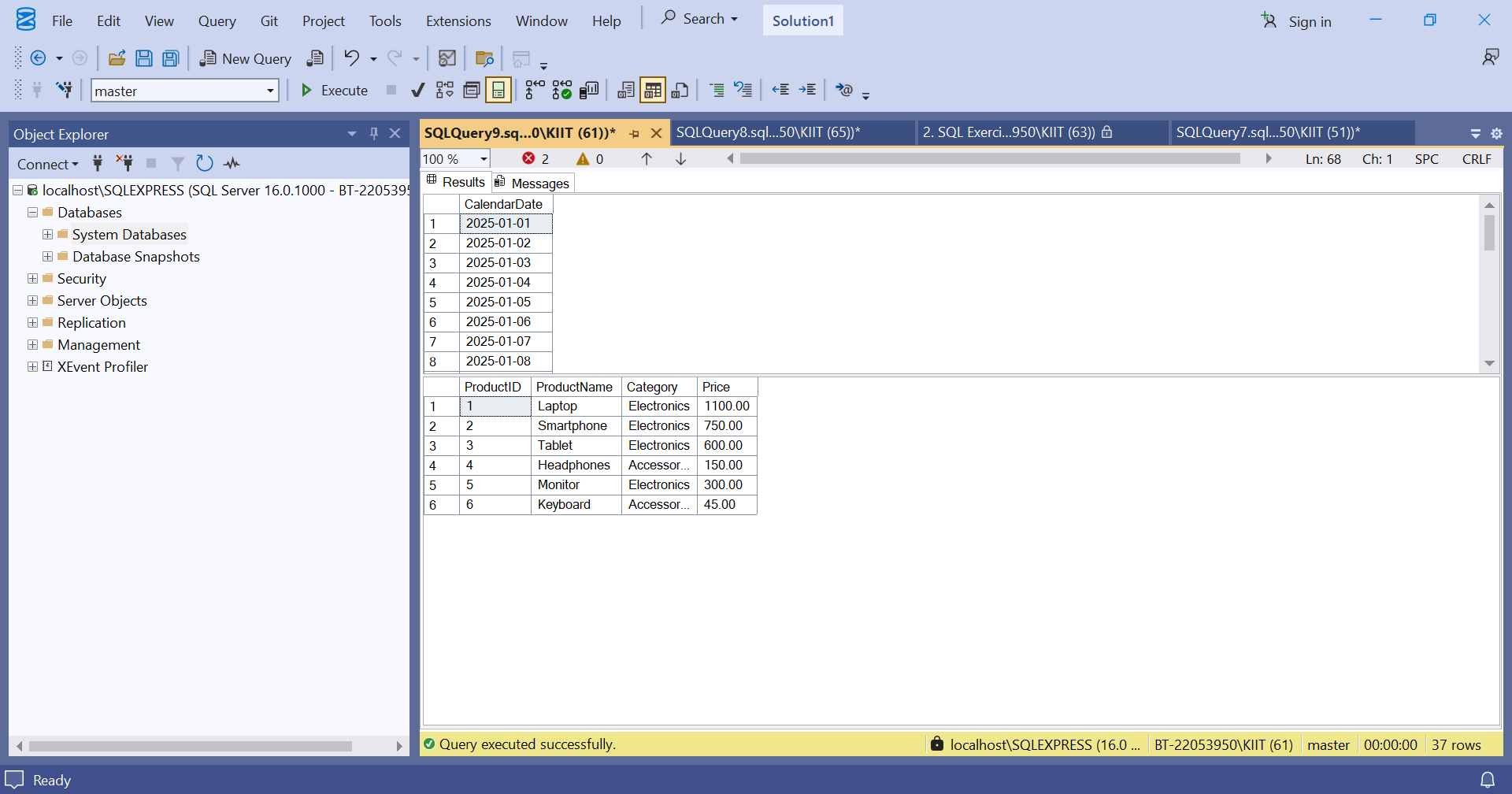
-- ========================================

-- Verify the updated Products table

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

SELECT \* FROM Products ORDER BY ProductID;

GO



Exercise 1D.

-- 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

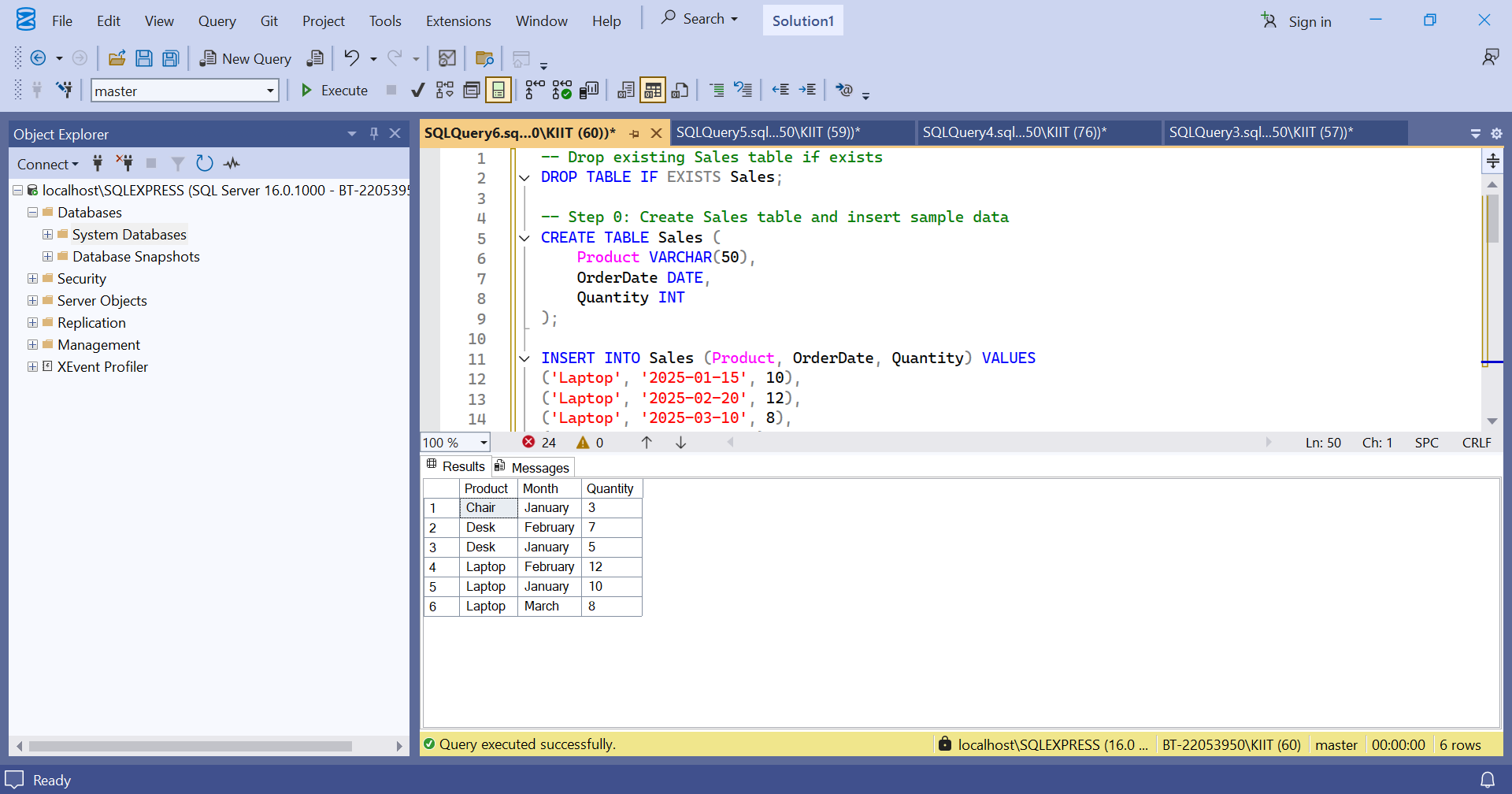
UNPIVOT (

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

) AS UnpivotTable

WHERE Quantity IS NOT NULL

ORDER BY Product, Month;

Exercise 1E.

-- Step 1: Create CTE to count orders per customer

WITH OrderCounts AS (

SELECT

CustomerID,

COUNT(OrderID) AS TotalOrders

FROM Orders

GROUP BY CustomerID

)

-- Step 2: Select customers with more than 3 orders

SELECT

c.CustomerID,

c.Name,

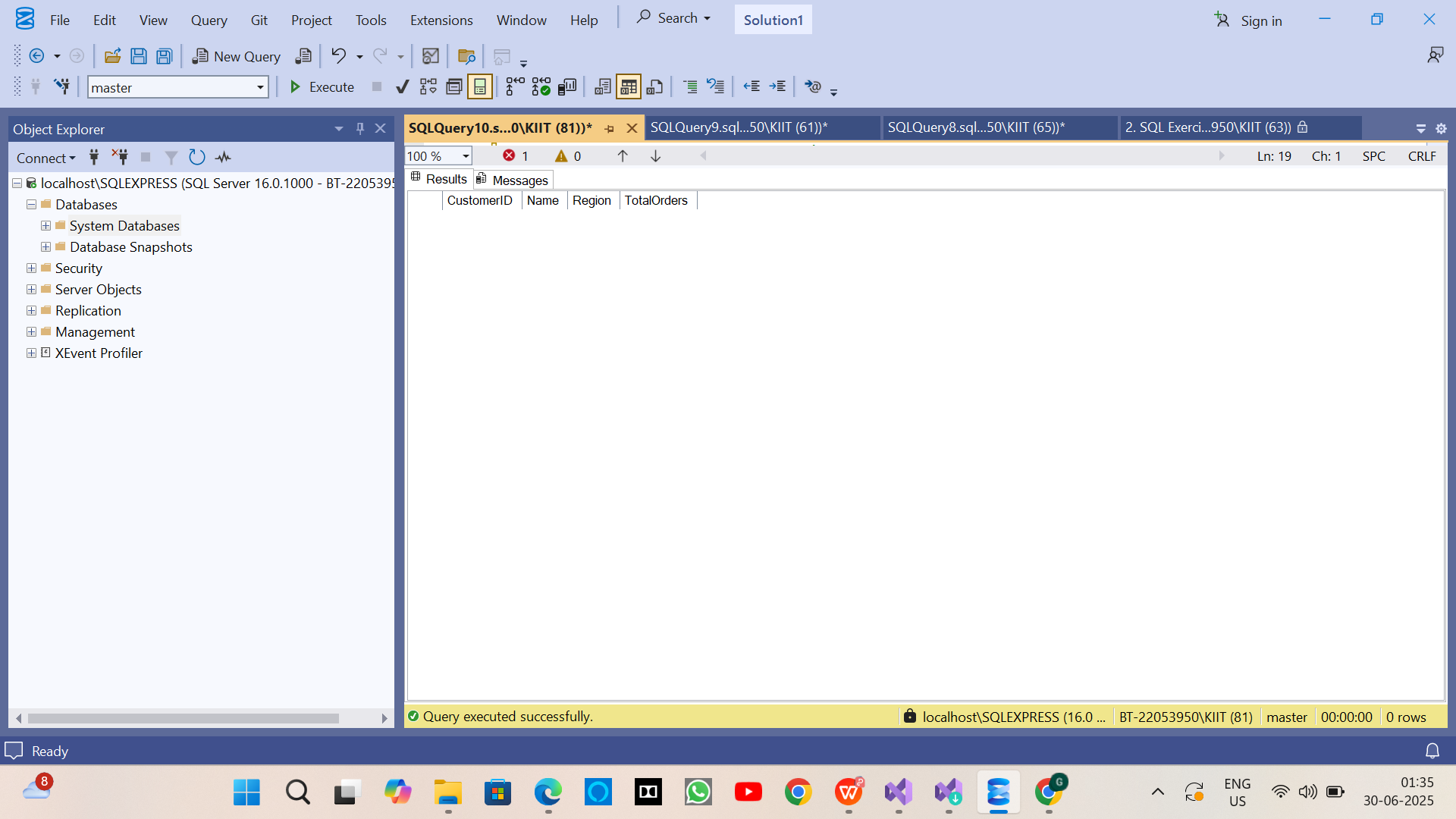
c.Region,

oc.TotalOrders

FROM Customers c

JOIN OrderCounts oc ON c.CustomerID = oc.CustomerID

WHERE oc.TotalOrders > 3;



Exercise 2.

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

-- [Optional] Drop tables if they already exist

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

IF OBJECT\_ID('OrderDetails', 'U') IS NOT NULL DROP TABLE OrderDetails;

IF OBJECT\_ID('Orders', 'U') IS NOT NULL DROP TABLE Orders;

IF OBJECT\_ID('Products', 'U') IS NOT NULL DROP TABLE Products;

IF OBJECT\_ID('Customers', 'U') IS NOT NULL DROP TABLE Customers;

GO

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

-- Create Tables

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

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

Region VARCHAR(50)

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

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)

);

GO

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

-- Insert Sample Data

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

INSERT INTO Customers (CustomerID, Name, Region) VALUES

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

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

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

(4, 'David', 'West');

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

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

(2, 'Smartphone', 'Electronics', 800.00),

(3, 'Tablet', 'Electronics', 600.00),

(4, 'Headphones', 'Accessories', 150.00);

INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES

(1, 1, '2023-01-15'),

(2, 2, '2023-02-20'),

(3, 3, '2023-03-25'),

(4, 4, '2023-04-30');

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

(1, 1, 1, 1),

(2, 2, 2, 2),

(3, 3, 3, 1),

(4, 4, 4, 3);

GO

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

-- EXERCISE 1: Non-Clustered Index on ProductName

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

-- Step 1: Query before index creation

SELECT \* FROM Products WHERE ProductName = 'Laptop';

-- Step 2: Create a non-clustered index on ProductName

CREATE NONCLUSTERED INDEX IX\_Products\_ProductName ON Products(ProductName);

GO

-- Step 3: Query after index creation

SELECT \* FROM Products WHERE ProductName = 'Laptop';

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

-- EXERCISE 2: Clustered Index on OrderDate

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

-- Note: Clustered index can only be created if table has no existing clustered index (usually the PK).

-- Since OrderID is the PK and already clustered by default, we must drop the PK and recreate both.

-- Step 1: Query before index

SELECT \* FROM Orders WHERE OrderDate = '2023-01-15';

-- Step 2: Drop PK and recreate clustered index on OrderDate

ALTER TABLE Orders DROP CONSTRAINT PK\_\_Orders\_\_C3905BAF4A2B13BC;

GO

CREATE CLUSTERED INDEX IX\_Orders\_OrderDate ON Orders(OrderDate);

GO

-- Step 3: Query after index

SELECT \* FROM Orders WHERE OrderDate = '2023-01-15';

-- (Recreate PK if needed, as non-clustered)

ALTER TABLE Orders ADD CONSTRAINT PK\_Orders\_OrderID PRIMARY KEY NONCLUSTERED (OrderID);

GO

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

-- EXERCISE 3: Composite Index on CustomerID + OrderDate

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

-- Step 1: Query before composite index

SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';

-- Step 2: Create composite non-clustered index

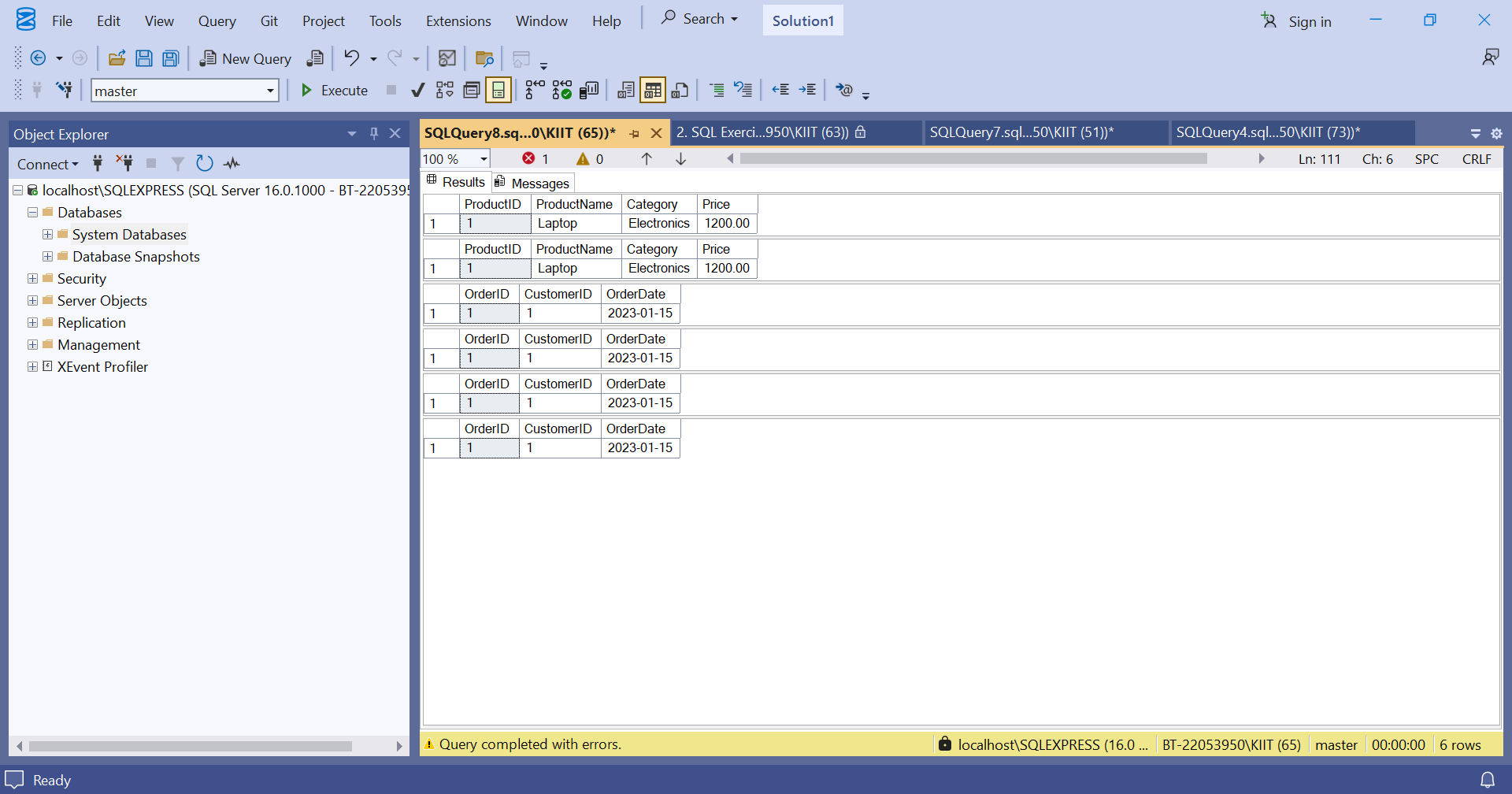
CREATE NONCLUSTERED INDEX IX\_Orders\_CustomerID\_OrderDate

ON Orders(CustomerID, OrderDate);

GO

-- Step 3: Query after index

SELECT \* FROM Orders WHERE CustomerID = 1 AND OrderDate = '2023-01-15';



Exercise 4.

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

-- SCHEMA CREATION

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

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

);

GO

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

-- INSERT SAMPLE DATA

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

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Marketing');

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, 'Michael', 'Johnson', 3, 7000.00, '2018-07-30'),

(4, 'Emily', 'Davis', 4, 5500.00, '2021-11-05');

GO

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

-- EXERCISE 1: Create Stored Procedure

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

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

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

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

GO

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

-- EXERCISE 2: Modify Stored Procedure

-- (Already included Salary above)

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

-- No separate code needed as Salary is already included.

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

-- EXERCISE 3: Delete Stored Procedure

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

DROP PROCEDURE IF EXISTS sp\_GetEmployeesByDepartment;

GO

-- Recreate it for remaining exercises

CREATE PROCEDURE sp\_GetEmployeesByDepartment

@DepartmentID INT

AS

BEGIN

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

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

GO

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

-- EXERCISE 4: Execute Stored Procedure

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

EXEC sp\_GetEmployeesByDepartment @DepartmentID = 1;

GO

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

-- EXERCISE 5: Return Data from a Stored Procedure

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

CREATE PROCEDURE sp\_GetEmployeeCountByDepartment

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS EmployeeCount

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

GO

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

-- EXERCISE 6: Use Output Parameters

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

CREATE PROCEDURE sp\_GetTotalSalaryByDepartment

@DepartmentID INT,

@TotalSalary DECIMAL(10,2) OUTPUT

AS

BEGIN

SELECT @TotalSalary = SUM(Salary)

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

GO

-- Execute with output

DECLARE @SalaryTotal DECIMAL(10,2);

EXEC sp\_GetTotalSalaryByDepartment @DepartmentID = 2, @TotalSalary = @SalaryTotal OUTPUT;

SELECT @SalaryTotal AS TotalSalary;

GO

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

-- EXERCISE 7: Multiple Parameters (Update Salary)

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

CREATE PROCEDURE sp\_UpdateEmployeeSalary

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

END;

GO

EXEC sp\_UpdateEmployeeSalary 1, 5600.00;

GO

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

-- EXERCISE 8: Conditional Logic (Bonus)

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

CREATE PROCEDURE sp\_GiveBonus

@DepartmentID INT,

@BonusAmount DECIMAL(10,2)

AS

BEGIN

UPDATE Employees

SET Salary = Salary + @BonusAmount

WHERE DepartmentID = @DepartmentID;

END;

GO

EXEC sp\_GiveBonus 1, 500.00;

GO

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

-- EXERCISE 9: Transactions

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

CREATE PROCEDURE sp\_UpdateSalaryWithTransaction

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

BEGIN TRANSACTION;

BEGIN TRY

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

THROW;

END CATCH

END;

GO

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

-- EXERCISE 10: Dynamic SQL

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

CREATE PROCEDURE sp\_GetEmployeesByFilter

@FilterColumn NVARCHAR(50),

@FilterValue NVARCHAR(100)

AS

BEGIN

DECLARE @SQL NVARCHAR(MAX);

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

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

END;

GO

EXEC sp\_GetEmployeesByFilter 'FirstName', 'Jane';

GO

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

-- EXERCISE 11: Error Handling

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

CREATE PROCEDURE sp\_UpdateSalaryWithErrorHandling

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

BEGIN TRY

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

PRINT 'Salary updated successfully.';

END TRY

BEGIN CATCH

PRINT 'Error updating salary: ' + ERROR\_MESSAGE();

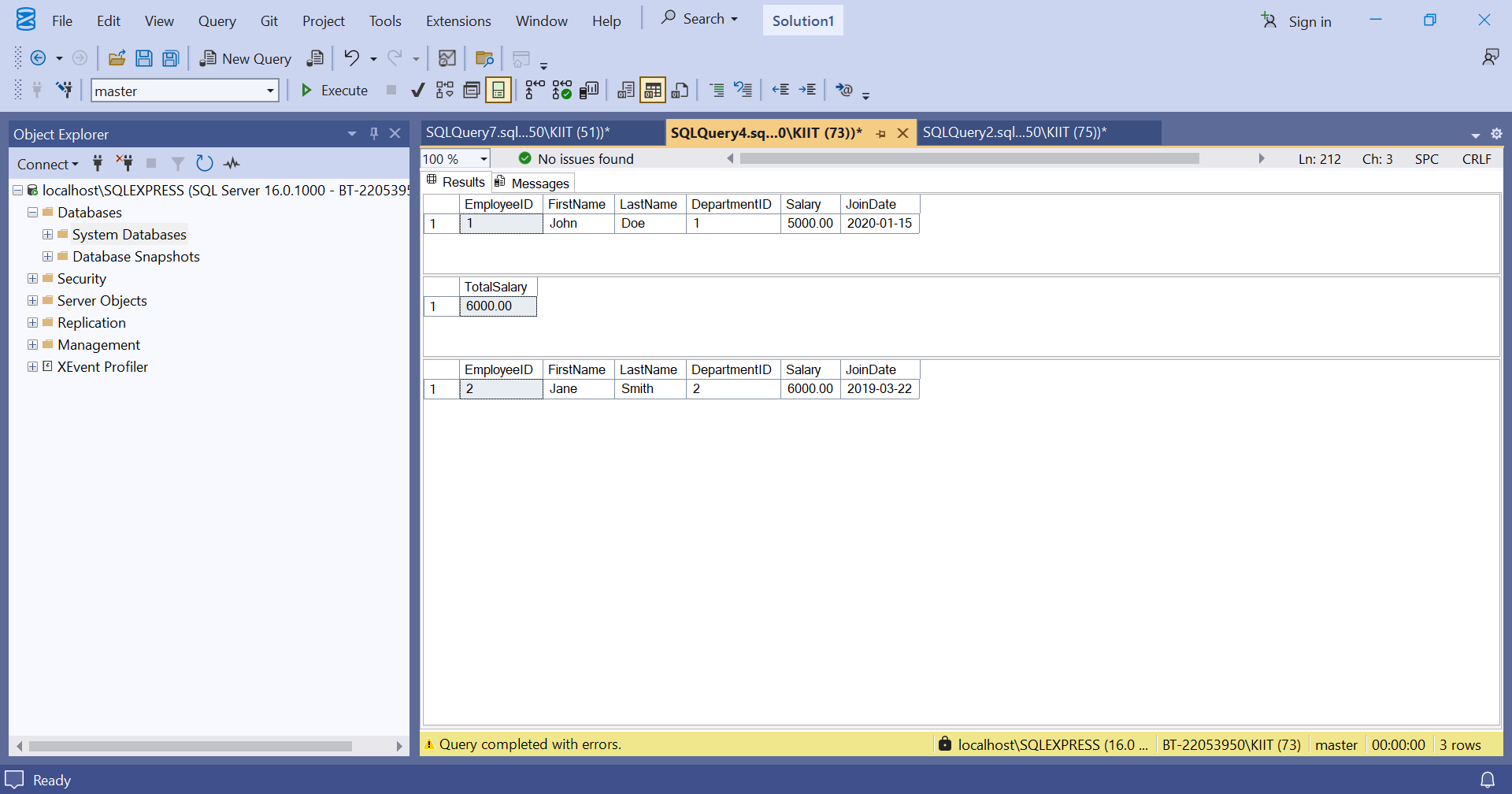
END CATCH

END;

GO

EXEC sp\_UpdateSalaryWithErrorHandling 99, 7500.00;

GO



Exercise 5.

-- ------------------------

-- SCHEMA & SAMPLE DATA SETUP

-- ------------------------

DROP TABLE IF EXISTS Employees;

DROP TABLE IF EXISTS Departments;

GO

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

GO

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

);

GO

INSERT INTO Departments VALUES (1, 'HR'), (2, 'IT'), (3, 'Finance');

GO

INSERT INTO Employees 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');

GO

-- ------------------------

-- EXERCISE 1: SCALAR FUNCTION - fn\_CalculateAnnualSalary

-- ------------------------

IF OBJECT\_ID('dbo.fn\_CalculateAnnualSalary', 'FN') IS NOT NULL

DROP FUNCTION dbo.fn\_CalculateAnnualSalary;

GO

CREATE FUNCTION dbo.fn\_CalculateAnnualSalary (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 12;

END;

GO

-- ------------------------

-- EXERCISE 2: TABLE-VALUED FUNCTION - fn\_GetEmployeesByDepartment

-- ------------------------

IF OBJECT\_ID('dbo.fn\_GetEmployeesByDepartment', 'IF') IS NOT NULL

DROP FUNCTION dbo.fn\_GetEmployeesByDepartment;

GO

CREATE FUNCTION dbo.fn\_GetEmployeesByDepartment (@DeptID INT)

RETURNS TABLE

AS

RETURN

(

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

FROM Employees

WHERE DepartmentID = @DeptID

);

GO

-- ------------------------

-- EXERCISE 3: SCALAR FUNCTION - fn\_CalculateBonus (10%)

-- ------------------------

IF OBJECT\_ID('dbo.fn\_CalculateBonus', 'FN') IS NOT NULL

DROP FUNCTION dbo.fn\_CalculateBonus;

GO

CREATE FUNCTION dbo.fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.10;

END;

GO

-- ------------------------

-- EXERCISE 4: MODIFY BONUS FUNCTION TO 15%

-- ------------------------

ALTER FUNCTION dbo.fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.15;

END;

GO

-- ------------------------

-- EXERCISE 5: DELETE BONUS FUNCTION

-- ------------------------

DROP FUNCTION IF EXISTS dbo.fn\_CalculateBonus;

GO

-- ------------------------

-- RECREATE BONUS FUNCTION (needed for nested function use)

-- ------------------------

CREATE FUNCTION dbo.fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.15;

END;

GO

-- ------------------------

-- EXERCISE 6: EXECUTE fn\_CalculateAnnualSalary

-- ------------------------

SELECT EmployeeID, FirstName, LastName, Salary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees;

GO

-- ------------------------

-- EXERCISE 7: RETURN ANNUAL SALARY FOR EMPLOYEEID = 1

-- ------------------------

SELECT FirstName, Salary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees

WHERE EmployeeID = 1;

GO

-- ------------------------

-- EXERCISE 8: GET EMPLOYEES FROM FINANCE (ID = 3)

-- ------------------------

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

GO

-- ------------------------

-- EXERCISE 9: NESTED FUNCTION - fn\_CalculateTotalCompensation

-- ------------------------

IF OBJECT\_ID('dbo.fn\_CalculateTotalCompensation', 'FN') IS NOT NULL

DROP FUNCTION dbo.fn\_CalculateTotalCompensation;

GO

CREATE FUNCTION dbo.fn\_CalculateTotalCompensation (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

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

END;

GO

-- ------------------------

-- EXERCISE 10: TEST NESTED FUNCTION (Modified Bonus Logic Included)

-- ------------------------

SELECT EmployeeID, FirstName, Salary,

dbo.fn\_CalculateTotalCompensation(Salary) AS TotalCompensation

FROM Employees;

GO

