Module 17

What to learn

Normalization

Practice Exercise

Practice 1

Refer Videos and ppt. Practice the example provided in video

Practice 2

Practice Exercise: Create a Basic Scalar Function

Write a scalar UDF to calculate the square of a given integer. Define a function, `dbo.SquareNumber`, that accepts an integer input and returns its square. Test this function by using it in a SELECT statement with a test value of your choice.

Hint: Use the `RETURNS` keyword in your function's definition.

Objective: Understand how to write a simple scalar User-Defined Function (UDF) in SQL.

Practice 3

Practice Exercise: Create a Multi-Statement Table-Valued Function

Create a table-valued function named `dbo.GetEvenNumbers` that accepts a list of integers as input (comma-separated string) and returns a table containing only the even numbers from the list. Use the `TABLE` return type and write logic to filter out only even numbers in a multi-statement format.

Hint: Split the string into rows and filter for even values using the MOD operator.

Objective: Practice implementing a table-valued function.

Practice 4

Practice Exercise: Drop and Recreate a Function

Assume you have the following scalar function `dbo.GetFullName` that concatenates a first name and last name into a full name. Drop the existing function and recreate it with an additional middle name parameter. Test the newly altered function with a sample query.

Objective: Learn to modify functions by dropping and recreating them.

Practice 5

Practice Exercise: Using Built-In System Functions

Write a SQL query that uses a built-in system function to find the current database name, server name, and current user executing the query. Use the functions `DB_NAME`, `HOST_NAME`, and `SUSER_NAME` in your query.

Hint: Built-in system functions are pre-defined in SQL Server and can be directly invoked in queries.

Objective: Understand how and when to use built-in system functions.

Practice 6

Practice Exercise: Best Practices for Writing UDFs

Refactor the following poorly written scalar UDF to improve performance and best practices:

```
CREATE FUNCTION dbo.PoorlyWrittenFunction(@ID INT) RETURNS NVARCHAR(50) BEGIN DECLARE
```

Ensure the function only retrieves the necessary column and includes error handling for cases where no record exists. Test the function on a sample `Employees` table.

Objective: Apply best practices while creating functions.

Assignment Exercise

Assignment 1

Assignment: Sales Data Management System

Scenario: A company wants to use SQL UDFs to manage its sales data effectively. You are tasked with implementing a set of custom SQL functions for their database. The company maintains the following tables:

Products: Contains details about products (ProductID, ProductName, UnitPrice).
Sales: Tracks individual sales transactions (SaleID, ProductID, Quantity, SaleDate, TotalAmount).
Employees: Records employee details (EmployeeID, FirstName, LastName, Department).

Requirements:

Create Scalar Functions: Write a scalar function `dbo.GetFullEmployeeName` that accepts an employee's ID and returns their full name in 'FirstName LastName' format. Additionally, write another

scalar function `dbo.CalculateDiscount` that calculates a 10% discount on a given sale amount and returns the discounted price.
Create Table-Valued Functions: Create a table-valued function `dbo.GetSalesByProduct` that takes a ProductID as input and returns all sales transactions related to that product, including SaleDate and Quantity details.
Drop and Alter Functions: Assume the company wants to enhance the `dbo.CalculateDiscount` function to accept a custom discount percentage instead of a fixed 10%. Drop the current function and recreate it with this change. Test your new function on a sample sales table.
Using Built-In Functions: Use built-in system functions to write a query that lists all employees who executed SQL queries in the current session, along with the database name and server they accessed.
Implementation of Best Practices: Ensure all UDFs follow SQL best practices, such as minimizing multi-statement logic for performance and using proper error handling. Refactor the existing `dbo.GetFullEmployeeName` function to include proper handling for cases where no record exists for a given ID.

Expected Output: Implement and test all UDFs with sample data for tables `Products`, `Sales`, and `Employees`. Provide sample queries showing the practical usage of these functions.

Online Reference

No online Reference

Introduction to Relational Databases

Introduction to Select Statement

Filtering Results with WHERE Statements

Utilizing Joins

Executing Sub queries and Unions

Aggregating Data

Advanced Data Aggregations

Built in Functions

Query Optimization

Modifying Data

Advanced Data Modification

- Stored Procedure
- Transaction
- Error handling
- Designing Tables
- triggers