Module 9

What to learn

- Batch
- Declare Variable
  - Setting values of the variable
- If-else Statement
- Case Statement
- while loop
- Processing JSON
  - ISJSON
  - JSON_VALUE
  - JSON_QUERY
  - JSON_MODIFY
- Convert JSON collections to a rowset
  - OPENJSON
- Convert SQL Server data to JSON or export JSON
  - FOR JSON PATH
  - FOR JSON AUTO
  - WITHOUT_ARRAY_WRAPPER

Practice Exercise

Practice 1

Implement the task explained on msdn site.

Practice 2

Understanding Stored Procedures

Create a stored procedure named **GetCustomerById** to retrieve a customer by their unique ID. Use a parameter to accept the customer's ID. Work on optimizing the query structure for better performance, and handle scenarios where the ID is not found by returning an appropriate message.

Practice 3

Using Built-in Stored Procedures

Write a query using the SQL Server built-in stored procedure **sp_help** to display the schema of an existing table name provided as input. Use this to understand the table structure, including column names, types, and constraints.

Practice 4

Creating a Stored Procedure with Input and Output Parameters

Create a stored procedure named **CalculateDiscount** that takes an order total (input parameter) and returns the discount amount (output parameter). Apply the following discount rules: 10% discount for orders above $1000 and 5% for orders between $500 and $1000.

Practice 5

Executing Stored Procedures

Execute the **CalculateDiscount** stored procedure you created earlier to test it with different order totals, such as $450, $750, and $1250. Check the output parameter values and ensure the logic applied is correct according to the given discount rules.

Practice 6

Returning Multiple Result Sets

Create a stored procedure named **GetCustomerAndOrders** that will return two result sets. The first result set is the list of all customers, and the second result set is the list of all orders. Ensure data from both queries is appropriately retrieved in a single execution.

Practice 7

Returning JSON Output from Stored Procedures

Create a stored procedure named **GetOrderDetailsAsJSON** that retrieves order details (Order ID, Order Total, and Customer Name) in JSON format using **FOR JSON PATH**. Test its output to ensure it works as expected.

Practice 8

Using SET NOCOUNT ON

Create a stored procedure named **UpdateCustomerAddress** that updates a customer address based on their ID. Use **SET NOCOUNT ON** to suppress the message of the row count affected by the update. Observe the effect of using **SET NOCOUNT ON**.

Practice 9

Using WITH ENCRYPTION

Create a stored procedure named **EncryptProcedureDemo** that retrieves customer details. Use the **WITH ENCRYPTION** clause to encrypt the stored procedure and then attempt to view its definition. Ensure encryption is implemented successfully.

Practice 10

Implementing Exception Handling with TRY...CATCH

Create a stored procedure named **SafeInsertCustomer** that inserts a new customer into the **Customers** table. Use a **TRY...CATCH** block to handle any potential errors (e.g., duplicate ID or NULL values in non-nullable columns). Log the error message in an **ErrorLog** table.

Practice 11

Combining Topics

Create a stored procedure named **ProcessOrder** that accepts Customer ID and Order details (e.g., Product ID, Quantity) as inputs. Perform the following actions within the procedure: Insert the order data into the **Orders** table, calculate the total price, and return the generated **Order ID** and Total Price as output parameters. Use **TRY...CATCH** for exception handling if any issues occur during insertion, and ensure the procedure uses **SET NOCOUNT ON** for performance optimization.

Assignment Exercise

Assignment 1

Create a batch Select Banking as 'Bank Dept', Insurance as 'Insurance Dept' and Services as 'Services Dept' from employee table 5 Students Name, Address, City, DOB, Standard need to be inserted in the student table, need to fetch these result from json variable. and select output in the json format

Assignment 2

Real-World Scenario: Inventory Management System

Objective: Build a set of stored procedures to manage an inventory system for a small retail company. The system must allow adding inventory items, updating stock levels, retrieving inventory details, and generating sales reports.

Functional Flow

Create a table **Products** with the following columns:

ProductID (Primary Key, INT, Auto Increment)
ProductName (VARCHAR)
StockQuantity (INT)
Price (DECIMAL)

Create a table **Sales** with the following columns:

SaleID (Primary Key, INT, Auto Increment)
ProductID (Foreign Key from Products table)
QuantitySold (INT)
SaleDate (DATETIME)

TotalSaleAmount (DECIMAL)

Implement the following stored procedures:

a. AddProduct

A stored procedure to add a new product to the **Products** table.
Accept **ProductName, StockQuantity,** and **Price** as input parameters, and validate that the stock and price are non-negative.

b. UpdateStock

A stored procedure to update the stock level of a product.
Accept **ProductID** and **StockQuantity** as input parameters. Use **TRY...CATCH** for exception handling to ensure that the product ID exists in the Products table.

c. GetProductDetails

A stored procedure that retrieves product details (ProductID, ProductName, StockQuantity, and Price) for a given **ProductID**. Return the data in JSON format using **FOR JSON PATH**.

d. RecordSale

A stored procedure to record a sale in the **Sales** table.
Accept **ProductID**, **QuantitySold**, and **SaleDate** as input parameters, compute the **TotalSaleAmount** based on the product's price, and update the product's stock in the **Products** table. Use a **TRY...CATCH** block to manage error handling if the stock is insufficient or the ProductID is invalid.

e. GenerateSalesReport

A stored procedure that generates a report of sales, grouped by **SaleDate**. Include the total sales amount and number of products sold for each date. Use **SET NOCOUNT ON** for performance optimization and return report data in JSON format.

Business Logic

Ensure appropriate validations for all stored procedures (e.g., non-negative stock and sale quantities).

Use the **WITH ENCRYPTION** option for the **GenerateSalesReport** stored procedure to hide its definition.

Handle errors gracefully with **TRY...CATCH**, and log errors into an **ErrorLog** table for debugging and audit purposes.

Test the system with example data for products and sales, and demonstrate the output for each stored procedure.

Online Reference

https://docs.microsoft.com/en-us/sql/relational-databases/json/json-data-sql-...

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