Module 16

What to learn

Normalization

Practice Exercise

Practice 1

Implementing a Stored Procedure to Return JSON Output

Consider a database table named **Products** with the following schema:

ProductID: INT (Primary Key, Auto-increment)
ProductName: VARCHAR(255)
Category: VARCHAR(100)
Price: DECIMAL(10, 2)
StockQuantity: INT

Create a stored procedure named **GetProductsAsJSON**. This procedure should retrieve all rows from the **Products** table and return the result in JSON format.

Make use of **FOR JSON AUTO** or **FOR JSON PATH** clauses in SQL Server to achieve this. Validate the procedure by executing it and checking the JSON output.

Practice 2

Using the SET NOCOUNT ON Statement

Write a stored procedure, **UpdateStock**, to update the stock of a product in the **Products** table. The procedure should take **ProductID** and **QuantityToAdd** as input parameters and update the **StockQuantity** field by adding the specified quantity.

Ensure you use the **SET NOCOUNT ON** within the stored procedure to suppress the message indicating the number of rows affected. Validate the behavior by observing that no messages are displayed when the procedure executes.

Practice 3

Implementing WITH ENCRYPTION for a Stored Procedure

Create a stored procedure called **GetHighValueProducts** that retrieves all products with a price greater than 1000 from the **Products** table. Use the **WITH ENCRYPTION** option to encrypt the definition of the stored procedure. Attempt to retrieve the definition of the procedure using **sp_helptext** to ensure the encryption works as intended.

Practice 4

Exception Handling Using TRY...CATCH

Develop a stored procedure called **AddProduct** to insert a new product into the **Products** table. It should take **ProductName**, **Category**, **Price**, and **StockQuantity** as input parameters. Implement **TRY...CATCH** blocks to handle any errors that occur during the insertion. In case of an error, insert the error details into a separate table named **ErrorLog** with columns: **ErrorMessage** (VARCHAR), **ErrorProcedure** (VARCHAR), and **ErrorTime** (DATETIME).

Practice 5

Combining SET NOCOUNT ON and TRY...CATCH

Create a stored procedure, **UpdateProductPrice**, to update the price of a product in the **Products** table. The procedure should take **ProductID** and **NewPrice** as input parameters. Use **SET NOCOUNT ON** to suppress row count messages and implement **TRY...CATCH** blocks to handle any errors. In case of errors, the details should be logged into the **ErrorLog** table.

Assignment Exercise

Assignment 1

Create ER Diagram and Database Design for the task assigned at the time of mini Project.

Assignment 2

Comprehensive Product Inventory Management System

You are tasked with building a small inventory system to manage products for a retail business. The system involves managing the **Products** table and implementing key functionalities as stored procedures. Follow these steps:

Functional Requirements

- **Create Tables:** Create a table **Products** with the following schema:
  - **ProductID**: INT, Primary Key, Auto-increment
  - **ProductName**: VARCHAR(255)
  - **Category**: VARCHAR(100)
  - **Price**: DECIMAL(10, 2)
  - **StockQuantity**: INT

  Create another table **ErrorLog** with columns: **ErrorMessage** (VARCHAR), **ErrorProcedure** (VARCHAR), and **ErrorTime** (DATETIME).
- **Implement Stored Procedures:**
  - **AddNewProduct:** A procedure to add a new product. Use **TRY...CATCH** for error handling.

- **GetProductsAsJSON:** A procedure to retrieve all products in JSON format using **FOR JSON**.
- **UpdateProductStock:** A procedure to update stock quantity, using **SET NOCOUNT ON** to suppress row count messages.
- **GetHighValueProducts:** An encrypted procedure to fetch products with a price greater than a specified value.

Trigger Errors: Design scenarios to trigger errors, such as by violating unique constraints or expected types. Verify error handling by ensuring details are logged in **ErrorLog**.

Testing: Execute all stored procedures multiple times with different inputs to ensure functionality and error handling.

Business Logic

Ensure that:

- All prices must be non-negative.
- Attempts to insert duplicate products (same name and category) result in errors that are logged in the **ErrorLog**.
- The **Products** table cannot contain products with a stock quantity less than zero.

Expected Output

- All functionalities are implemented and tested.
- Error scenarios are logged in **ErrorLog** with accurate details (procedure name, error message, and timestamp).
- Successful JSON output from the relevant procedure.

Online Reference

No online Reference