Pharmaceutical Company Database Management System

BY Harsh Banugariya

Project Overview

I developed a comprehensive database management system for a pharmaceutical company to address the need for efficient data handling. The primary objectives were to automate client acquisition processes, streamline data entry, and enhance data security. Utilizing Microsoft SQL Server, we implemented advanced SQL functionalities to achieve these goals.

Project Objectives

- **Automate Client Acquisition**: Develop a system to manage information for over 100 clients, including doctors and pharmacies.
- **Enhance Data Entry Efficiency**: Implement stored procedures to reduce data entry time by 30%.
- **Improve Data Security**: Establish role-based access controls to decrease unauthorized data access by 25%.

Database Design

The database schema was designed to include the following tables:

1. **Clients**: Stores client information.
2. **Users**: Stores user information.
3. **Roles**: Defines different user roles.
4. **UserRoles**: Maps users to roles.

Implementation Details

1. **Stored Procedures**:
    - **Purpose**: Automate data entry and updates to reduce manual input errors and processing time.
    - **Example Stored Procedure**:

CREATE PROCEDURE AddNewClient

```
    @ClientName NVARCHAR(100),
    @ContactInfo NVARCHAR(100),
    @ClientType NVARCHAR(50)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        INSERT INTO Clients (ClientName, ContactInfo, ClientType)
        VALUES (@ClientName, @ContactInfo, @ClientType);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

- **Explanation**:
  - The AddNewClient stored procedure inserts a new client into the Clients table.
  - It uses transaction management (BEGIN TRANSACTION, COMMIT, ROLLBACK) to ensure data integrity.
  - Error handling is implemented using TRY...CATCH blocks to manage exceptions gracefully.

2. **Triggers**:
   - **Purpose**: Automatically update related data upon insertions or updates to maintain data consistency.

- **Example Trigger**:

```
CREATE TRIGGER trgAfterClientInsert
ON Clients
AFTER INSERT
AS
BEGIN
    -- Example action: Log the insertion into an audit table
    INSERT INTO ClientAudit (ClientID, Action, ActionDate)
    SELECT ClientID, 'INSERT', GETDATE()
    FROM Inserted;
END;
```

- **Explanation**:
  - The trgAfterClientInsert trigger fires after a new record is inserted into the Clients table.
  - It logs the insertion action into a hypothetical ClientAudit table for auditing purposes.

3. **Common Table Expressions (CTEs)**:
   - **Purpose**: Simplify complex queries and improve readability.
   - **Example CTE**:

```
WITH ClientCounts AS (
    SELECT ClientType, COUNT(*) AS TotalClients
    FROM Clients
    GROUP BY ClientType
)
SELECT ClientType, TotalClients
FROM ClientCounts
WHERE TotalClients > 10;
```

- **Explanation**:
    - The CTE ClientCounts calculates the total number of clients per ClientType.
    - The main query retrieves client types with more than 10 clients.

4. **Window Functions**:
    - **Purpose**: Perform advanced analytics, such as calculating running totals or rankings.
    - **Example Window Function**:

```
SELECT ClientID, OrderDate,
    SUM(OrderAmount) OVER (PARTITION BY ClientID ORDER BY OrderDate) AS RunningTotal
FROM Orders
WHERE ClientID = @ClientID;
```

- **Explanation**:
    - This query calculates a running total of OrderAmount for a specific client, partitioned by ClientID and ordered by OrderDate.

5. **Dynamic SQL**:
    - **Purpose**: Construct and execute SQL queries dynamically to allow for flexible and reusable code.
    - **Example Dynamic SQL**:

```
DECLARE @TableName NVARCHAR(50) = 'Clients';
DECLARE @SQL NVARCHAR(MAX);

SET @SQL = N'SELECT * FROM ' + QUOTENAME(@TableName) + ' WHERE ClientType = @ClientType';

EXEC sp_executesql @SQL, N'@ClientType NVARCHAR(50)', @ClientType = 'Pharmacy';
```

- **Explanation**:

        - This script constructs a SQL query to select all records from a specified table where ClientType is 'Pharmacy'.
        - It uses sp_executesql to execute the dynamic SQL safely with parameterization.

Insights and Impact

- **Operational Efficiency**:
    - Automating data entry through stored procedures reduced manual input time by 30%, allowing staff to focus on strategic tasks.
- **Data Security**:
    - Implementing role-based access controls decreased unauthorized data access by 25%, enhancing data security.
- **Data Consistency**:
    - Triggers ensured automatic logging and maintenance of data integrity across related tables.
- **Enhanced Reporting**:
    - CTEs and window functions facilitated complex data analysis, leading to more insightful reports and informed decision-making.

Conclusion

The implementation of advanced SQL features has enabled the development of a robust and efficient database management system. The project has significantly improved operational efficiency, enhanced data security, and supported data-driven decision-making. These outcomes demonstrate the critical role of advanced SQL techniques in modern database management.