Name: Hinda Nguyen

Date: August 20th, 2025

Course: IT FDN 130

GitHub: https://github.com/hindanguyenle/DBFoundations

Assignment 06 - SQL Views, Functions, and Stored Procedures

Contents

Introduction

In relational database management, the ability to efficiently retrieve, organize, and present data is essential for both developers and analysts. SQL provides various tools to achieve this, including Views, Functions, and Stored Procedures. Each plays a distinct role in improving query readability, reusability, and performance, while also contributing to database security and maintainability. This paper explains when to use a SQL View and explores the similarities and differences between a View, a Function, and a Stored Procedure.

When to Use a SQL View

A SQL View is best used when you need to simplify complex queries, present data in a specific format, or restrict user access to sensitive columns. Views act as saved queries that can join multiple tables, apply filters, and format results without requiring the user to understand the underlying SQL logic. For example, in this week's assignment, the `CREATE VIEW dbo.vInventoriesByProductsByDates` view joins the **Products** and **Inventories** tables to present inventory counts alongside product details in a single, easy-to-query dataset. Instead of rewriting a multi-table join every time, a developer can query the view as if it were a regular table, ensuring consistency and improving readability. Views can also enhance security by exposing only the necessary fields to certain users, preventing direct access to the base tables.

```sql
CREATE VIEW dbo.vInventoriesByProductsByDates
WITH SCHEMABINDING
AS
SELECT TOP 999999999
    p.ProductName,
    i.InventoryDate,
    i.[Count]
FROM dbo.Products AS p
JOIN dbo.Inventories AS i
    ON p.ProductID = i.ProductID
ORDER BY
    p.ProductName,
    i.InventoryDate,
    i.[Count];
GO
```

Figure 1. ***Example usage of*** **CREATE VIEW** ***for this assignment.***

Ordering Results in Views

In SQL Server, ORDER BY is not allowed directly inside a view definition unless it is paired with a TOP clause. This is because views are intended to represent a set of data, not a fixed sequence. If a specific ordering is desired in a view, one approach is to use TOP with a very large number (for example, TOP 999999999) alongside the ORDER BY clause.

However, best practice is to avoid enforcing ordering within a view. Instead, return the unordered data from the view and apply ORDER BY in the query that retrieves data from the view. This keeps the view flexible and reduces overhead, while still allowing ordering when needed by the consumer query.

```
CREATE VIEW dbo.vInventoriesByProductsByDates
WITH SCHEMABINDING
AS
SELECT
    p.ProductName,
    i.InventoryDate,
    i.Count
FROM dbo.Products AS p
JOIN dbo.Inventories AS i
    ON p.ProductID = i.ProductID;
GO

SELECT *
FROM dbo.vInventoriesByProductsByDates
ORDER BY ProductName, InventoryDate, [Count];
```

Figure 1. ***Example usage of*** **CREATE VIEW** ***without using*** **ORDER BY** ***in the view.***

Example:

- **With ORDER BY in the view (using TOP)**:
  Ensures the data is returned in a specific order whenever the view is queried, but adds unnecessary processing if ordering is not always required. Refer to Figure 1.

- **Without ORDER BY in the view**:
  The ordering is applied only in the outer query, which is more efficient and adaptable to different use cases. Refer to Figure 2.

Differences and Similarities Between a View, Function, and Stored Procedure

Views

A View is essentially a virtual table representing the result of a predefined query. It does not store physical data and cannot accept parameters. Views are primarily used for reading data, simplifying complex queries, presenting results in a specific format, or restricting access to certain columns. For instance, the `CREATE VIEW dbo.vInventoriesByProductsByCategoriesByEmployees` example from the assignment combined multiple joins—including categories, products, inventories, and employees—into one reusable, read-only dataset.

```sql
CREATE VIEW dbo.vInventoriesByProductsByCategoriesByEmployees
WITH SCHEMABINDING
AS
SELECT
    c.CategoryID,
    c.CategoryName,
    p.ProductID,
    p.ProductName,
    i.InventoryID,
    i.InventoryDate,
    i.[Count],
    e.EmployeeID,
    e.EmployeeFirstName,
    e.EmployeeLastName,
    -- Employee full name
    EmployeeName = e.EmployeeFirstName + ' ' + e.EmployeeLastName,
    -- Manager full name
    ManagerName = m.EmployeeFirstName + ' ' + m.EmployeeLastName
FROM dbo.vCategories c
JOIN dbo.vProducts p ON c.CategoryID = p.CategoryID
JOIN dbo.vInventories i ON p.ProductID = i.ProductID
JOIN dbo.vEmployees e ON i.EmployeeID = e.EmployeeID
LEFT JOIN dbo.vEmployees m ON e.ManagerID = m.EmployeeID;
GO

SELECT *
FROM dbo.vInventoriesByProductsByCategoriesByEmployees
ORDER BY CategoryName, ProductName, InventoryID, EmployeeName;
```

Figure 3. ***A view can join multiple base views or tables to centralize query logic.***

Functions

A Function can accept parameters, perform calculations or transformations, and return either a single value (scalar function) or a set of rows (table-valued function). Functions are often used for reusable computations, such as formatting values, calculating totals, or returning filtered datasets. Hypothetically, based on this week's database, a table-valued function could accept a `CategoryID` and return only the inventory records for products in that category, including employee and manager details. Unlike views, functions can contain procedural logic, but they are more restricted than stored procedures when it comes to modifying data.

Stored Procedures

A Stored Procedure is the most versatile of the three. It can accept parameters, include conditional logic and loops, and execute multiple SQL statements, including `INSERT`, `UPDATE`, and `DELETE`. Stored procedures can also return multiple result sets and output parameters, making them ideal for complex business logic. In the context of this week's database, a stored procedure could accept a product name, calculate new stock levels, and update the related records in multiple tables within one transaction.

Similarities

Views, functions, and stored procedures all store SQL code within the database for reusability, maintainability, and performance benefits. They help centralize logic, reduce repetitive query writing, and can improve security by controlling how users interact with data.

Conclusion

Understanding when and how to use Views, Functions, and Stored Procedures is essential for building efficient, maintainable, and secure database systems. Views provide a convenient way to simplify data access and enforce security, Functions offer parameterized logic for calculations and data transformation, and Stored Procedures handle complex operations and workflows. Mastering these tools ensures more robust database design and supports long-term scalability in any SQL-based application.