Michael Yu

5/14/2021

IT FDN 130: Foundations of Databases & SQL

Assignment 06

<https://github.com/myu404/DBFoundations>

SQL Views, Functions, and Stored Procedures

# Introduction

This assignment goes over SQL views, functions, and stored procedures. These features are callable objects that are reusable code provided to the client to perform SQL queries and execute SQL code. The SQL code complexity is abstracted inside these objects and are represented by a name that the client calls to perform the code.

When created, these features are saved directly to the database instead of a script file. Saving to database promotes reusability of the code for many users and eliminate the need to write repetitive code in script files.

# SQL View

SQL view is a reusable saved query (SELECT statements) in the database. The client user may call the view to perform the defined SQL query and return the data pulled from the query. Creating a view provides a layer of abstraction between the client and the code. The client retrieves data by calling the view saved in database, whereas in the background, the view executes the SQL query code to retrieve the data.

SQL view would be used when a SQL query is commonly performed on the database, such as pulling data on a daily basis. Repeatedly typing the same code to pull the data is not efficient. It is better to save the code to a SQL view and directly call the view to improve efficiency. Additionally, views are useful to save complex SQL queries that are used on a regular basis. Once the complex SQL query is saved as a view, future calls to the view would perform the complex SQL code without the need to writing the code again.

Views are commonly categorized as reporting views and base/basic views.

Reporting views refers to any views that capture SQL queries that return selected data from a table or tables (joins). These views do not necessarily capture every column of data from a table, but it captures the data as defined by the user as being the most useful for the intended purpose.

Base or basic views refers to views that return data from a table. These views are identical copies of the table itself. The intention of providing a copy of the table is to provide a layer of abstraction between the actual table and the view that interacts with the client.

The layer of abstraction allows for change to occur easier without affecting the normal operations from the client. Changes could be made on the table itself without affecting the functionality of the basic view for the client. If the client directly interacts with the table, then changes to the table may affect the client user-experience or might create an error in the program.

Basic views are also commonly used as safeguard for schema changes to a table. When keyword SCHEMABINDING is provided in the view creation, then the view binds to the table schema. Any changes to the table schema/design that will affect the view will be denied. Adding to a table design typically will be allowed because adding to table design will not affect the view.

In general, access permissions could be applied to views and tables. Permissions could be applied to views because views are essentially data just like tables. Defining access permissions to roles or groups provides control on who could read and/or write data to the database. It is common to keep the table private and make the views public to the users.

# Views, Functions, and Stored Procedures

The three common reusable and callable objects saved to databases in SQL are views, functions, and stored procedures. These objects provide names for a set of SQL statements that are commonly called to return data from SQL queries. Providing names to SQL statements improves efficiency and user-experience for the client because the client only needs to write the name of the view, function, or stored procedure to retrieve data in lieu of typing the full SQL query statement to perform the same task.

Each object has similarities and differences from one another.

## Views

As noted in SQL View, a SQL view is a reusable saved query that can be called to return data based on the query. Views are created to only perform SQL SELECT queries from tables in a database.

Views are designed to either report specific data from a table or provide a public interface to a table from a database. Public interface is created to allow the end-user/client to view the table without the possibility of corrupting the table or table data. It provides a layer of abstraction between the client and the code and data. The abstraction layer allows a database admin to make changes to the table without affecting the user-experience from the client side.

Permissions are defined for user roles or groups to control access to tables and views. It is common to make tables private from the user and make views public to the user, such that the user cannot corrupt table data by making direct changes to it.

## Functions

Functions are similar to views, where SQL code is saved to the database as a callable object. There are built-in functions that are native to SQL language, but users may also define their own functions (commonly called “user-defined functions). Functions may have parameters that allows SQL code to be parameterized and return data based on the code defined in the function. Functions could be defined as scalar functions or table functions.

Scalar functions refer to functions that execute SQL code in the function body and return a single value. Depending on the return value, the function could be used as an expression within a SELECT or WHERE clause.

When scalar functions are executed within SELECT clause, the return value is output in every row of data in the table specified by the SELECT statements. It is common to include a column that captures data returned from a function, where the function performs analytical operations on other columns in a table.

Table functions refer to functions that execute SQL code in the function body and return multiple values or a table of values. Depending on the return value, the function could be used as an expression within a FROM clause.

## Stored Procedures

Stored procedures are similar to views, where it provides names to sets of SQL queries, creates a layer of abstraction, saved to database, and are reusable callable objects. SQL code and queries are defined within a stored procedure and are called through the stored procedure’s name.

Unlike views and functions, stored procedures do not return data that could be used in an expression. A stored procedure can only be executed through the “EXECUTE” keyword and process the data per the stored procedure definition. Additionally, stored procedures are not

## Commonality and Differences

Views, functions, and stored procedures all provide names to a set of SQL statements. These names could be called by the client to perform SQL queries without the need to write the entire SQL statement. All three of the SQL features could be created to return data based on the SQL queries.

Views are saved SQL SELECT statements that are called through a view name. The data returned from views range from specific columns of data from a table (reporting views) to an entire table (base/basic views).

Functions are similar to views, where functions also save SQL SELECT statements, but also have the capability to accept parameters and input arguments to provide a more dynamic view. Functions accept parameters that are referenced within the function body and SQL statement definition. Unlike view, functions could be designed as scalar functions where only a single value of data is returned or designed as table functions where table of values are returned.

Both views and functions are limited to saving SELECT statement in their definition.

Stored procedures are like views and functions, where it can save SELECT statements, but could also be written with other SQL code beyond SELECT statements. Stored procedures also do not return values that could be used in an expression or within another SELECT statement. Stored procedures are standalone executable code that are executed through the EXECUTE keyword.

# Module 06 Assignment

Module 6’s assignment provides the opportunity to create SQL views and call the views through SELECT statements.

See Appendix for SQL code.

# Summary

This assignment covers SQL views, functions, and stored procedures. Each of these features provide a name for a set of SQL queries, saved to the database, and are callable objects. Implementing a database with these features improves efficiency for the client when performing repetitious tasks such as querying from the same table with complex SELECT statements.

# Appendix

## Assignment06DB\_YuMichael.sql

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--

-- Title: Assignment06

-- Author: YourNameHere

-- Desc: This file demonstrates how to use Views

-- Change Log: When,Who,What

-- 2021-05-14,MYu,Created File

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--

Begin Try

    Use Master;

    If Exists(Select Name From SysDatabases Where Name = 'Assignment06DB\_YuMichael')

     Begin

      Alter Database [Assignment06DB\_YuMichael] set Single\_user With Rollback Immediate;

      Drop Database Assignment06DB\_YuMichael;

     End

    Create Database Assignment06DB\_YuMichael;

End Try

Begin Catch

    Print Error\_Number();

End Catch

go

Use Assignment06DB\_YuMichael;

-- Create Tables (Module 01)--

Create Table Categories

([CategoryID] [int] IDENTITY(1,1) NOT NULL

,[CategoryName] [nvarchar](100) NOT NULL

);

go

Create Table Products

([ProductID] [int] IDENTITY(1,1) NOT NULL

,[ProductName] [nvarchar](100) NOT NULL

,[CategoryID] [int] NULL

,[UnitPrice] [mOney] NOT NULL

);

go

Create Table Employees -- New Table

([EmployeeID] [int] IDENTITY(1,1) NOT NULL

,[EmployeeFirstName] [nvarchar](100) NOT NULL

,[EmployeeLastName] [nvarchar](100) NOT NULL

,[ManagerID] [int] NULL

);

go

Create Table Inventories

([InventoryID] [int] IDENTITY(1,1) NOT NULL

,[InventoryDate] [Date] NOT NULL

,[EmployeeID] [int] NOT NULL -- New Column

,[ProductID] [int] NOT NULL

,[Count] [int] NOT NULL

);

go

-- Add Constraints (Module 02) --

Begin  -- Categories

    Alter Table Categories

     Add Constraint pkCategories

      Primary Key (CategoryId);

    Alter Table Categories

     Add Constraint ukCategories

      Unique (CategoryName);

End

go

Begin -- Products

    Alter Table Products

     Add Constraint pkProducts

      Primary Key (ProductId);

    Alter Table Products

     Add Constraint ukProducts

      Unique (ProductName);

    Alter Table Products

     Add Constraint fkProductsToCategories

      Foreign Key (CategoryId) References Categories(CategoryId);

    Alter Table Products

     Add Constraint ckProductUnitPriceZeroOrHigher

      Check (UnitPrice >= 0);

End

go

Begin -- Employees

    Alter Table Employees

     Add Constraint pkEmployees

      Primary Key (EmployeeId);

    Alter Table Employees

     Add Constraint fkEmployeesToEmployeesManager

      Foreign Key (ManagerId) References Employees(EmployeeId);

End

go

Begin -- Inventories

    Alter Table Inventories

     Add Constraint pkInventories

      Primary Key (InventoryId);

    Alter Table Inventories

     Add Constraint dfInventoryDate

      Default GetDate() For InventoryDate;

    Alter Table Inventories

     Add Constraint fkInventoriesToProducts

      Foreign Key (ProductId) References Products(ProductId);

    Alter Table Inventories

     Add Constraint ckInventoryCountZeroOrHigher

      Check ([Count] >= 0);

    Alter Table Inventories

     Add Constraint fkInventoriesToEmployees

      Foreign Key (EmployeeId) References Employees(EmployeeId);

End

go

-- Adding Data (Module 04) --

Insert Into Categories

(CategoryName)

Select CategoryName

 From Northwind.dbo.Categories

 Order By CategoryID;

go

Insert Into Products

(ProductName, CategoryID, UnitPrice)

Select ProductName,CategoryID, UnitPrice

 From Northwind.dbo.Products

  Order By ProductID;

go

Insert Into Employees

(EmployeeFirstName, EmployeeLastName, ManagerID)

Select E.FirstName, E.LastName, IsNull(E.ReportsTo, E.EmployeeID)

 From Northwind.dbo.Employees as E

  Order By E.EmployeeID;

go

Insert Into Inventories

(InventoryDate, EmployeeID, ProductID, [Count])

Select '20170101' as InventoryDate, 5 as EmployeeID, ProductID, ABS(CHECKSUM(NewId())) % 100 as RandomValue

From Northwind.dbo.Products

Union

Select '20170201' as InventoryDate, 7 as EmployeeID, ProductID, ABS(CHECKSUM(NewId())) % 100 as RandomValue

From Northwind.dbo.Products

Union

Select '20170301' as InventoryDate, 9 as EmployeeID, ProductID, ABS(CHECKSUM(NewId())) % 100 as RandomValue

From Northwind.dbo.Products

Order By 1, 2

go

-- Show the Current data in the Categories, Products, and Inventories Tables

Select \* From Categories;

go

Select \* From Products;

go

Select \* From Employees;

go

Select \* From Inventories;

go

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Questions and Answers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

'NOTES------------------------------------------------------------------------------------

 1) You can use any name you like for you views, but be descriptive and consistent

 2) You can use your working code from assignment 5 for much of this assignment

 3) You must use the BASIC views for each table after they are created in Question 1

------------------------------------------------------------------------------------------'

\*/

-- Question 1 (5% pts): How can you create BACIC views to show data from each table in the database.

-- NOTES: 1) Do not use a \*, list out each column!

--        2) Create one view per table!

--        3) Use SchemaBinding to protect the views from being orphaned!

go

Create View vCategories

With SchemaBinding

As

    Select

        [CategoryID]

        , [CategoryName]

    From dbo.Categories

go

go

Create View vProducts

With SchemaBinding

As

    Select

        [ProductID]

        , [ProductName]

        , [CategoryID]

        , [UnitPrice]

    From dbo.Products

go

go

Create View vEmployees

With SchemaBinding

As

    Select

        [EmployeeID]

        , [EmployeeFirstName]

        , [EmployeeLastName]

        , [ManagerID]

    From dbo.Employees

go

go

Create View vInventories

With SchemaBinding

As

    Select

        [InventoryID]

        , [InventoryDate]

        , [EmployeeID]

        , [ProductID]

        , [Count]

    From dbo.Inventories

go

-- Question 2 (5% pts): How can you set permissions, so that the public group CANNOT select data

-- from each table, but can select data from each view?

Deny Select On Categories To Public;

Deny Select On Products To Public;

Deny Select On Employees To Public;

Deny Select On Inventories To Public;

Grant Select On vCategories To Public;

Grant Select On vProducts To Public;

Grant Select On vEmployees To Public;

Grant Select On vInventories To Public;

-- Question 3 (10% pts): How can you create a view to show a list of Category and Product names,

-- and the price of each product?

-- Order the result by the Category and Product!

-- Here is an example of some rows selected from the view:

-- CategoryName,ProductName,UnitPrice

-- Beverages,Chai,18.00

-- Beverages,Chang,19.00

-- Beverages,Chartreuse verte,18.00

go

Create View vProductsByCategories

As

    Select Top 1000000000

        [CategoryName]

        , [ProductName]

        , [UnitPrice]

    From vCategories

        Inner Join vProducts

            On vCategories.CategoryID = vProducts.CategoryID

    Order By [CategoryName], [ProductName];

go

-- Question 4 (10% pts): How can you create a view to show a list of Product names

-- and Inventory Counts on each Inventory Date?

-- Order the results by the Product, Date, and Count!

-- Here is an example of some rows selected from the view:

--ProductName,InventoryDate,Count

--Alice Mutton,2017-01-01,15

--Alice Mutton,2017-02-01,78

--Alice Mutton,2017-03-01,83

go

Create View vInventoriesByProductsByDates

As

    Select Top 1000000000

        [ProductName]

        , [InventoryDate]

        , [Count]

    From vProducts

        Inner Join vInventories

            ON vProducts.ProductID = vInventories.ProductID

    Order By [InventoryDate], [ProductName], [Count];

go

-- Question 5 (10% pts): How can you create a view to show a list of Inventory Dates

-- and the Employee that took the count?

-- Order the results by the Date and return only one row per date!

-- Here is an example of some rows selected from the view:

-- InventoryDate,EmployeeName

-- 2017-01-01,Steven Buchanan

-- 2017-02-01,Robert King

-- 2017-03-01,Anne Dodsworth

go

Create View vInventoriesByEmployeesByDates

As

    Select Distinct Top 1000000000

        [InventoryDate]

        , [EmployeeFirstName] + ' ' + [EmployeeLastName] AS [EmployeeName]

    From vInventories

        Inner Join vEmployees

            ON vInventories.EmployeeID = vEmployees.EmployeeID

    Order By [InventoryDate];

go

-- Question 6 (10% pts): How can you create a view show a list of Categories, Products,

-- and the Inventory Date and Count of each product?

-- Order the results by the Category, Product, Date, and Count!

-- Here is an example of some rows selected from the view:

-- CategoryName,ProductName,InventoryDate,Count

-- Beverages,Chai,2017-01-01,72

-- Beverages,Chai,2017-02-01,52

-- Beverages,Chai,2017-03-01,54

go

Create View vInventoriesByProductsByCategories

As

    Select Top 1000000000

        [CategoryName]

        , [ProductName]

        , [InventoryDate]

        , [Count]

    From vCategories

        Inner Join vProducts

            On vCategories.CategoryID = vProducts.CategoryID

        Inner Join vInventories

            On vProducts.ProductID = vInventories.ProductID

    Order By [CategoryName], [ProductName], [InventoryDate], [Count] Desc;

go

-- Question 7 (10% pts): How can you create a view to show a list of Categories, Products,

-- the Inventory Date and Count of each product, and the EMPLOYEE who took the count?

-- Order the results by the Inventory Date, Category, Product and Employee!

-- Here is an example of some rows selected from the view:

-- CategoryName,ProductName,InventoryDate,Count,EmployeeName

-- Beverages,Chai,2017-01-01,72,Steven Buchanan

-- Beverages,Chang,2017-01-01,46,Steven Buchanan

-- Beverages,Chartreuse verte,2017-01-01,61,Steven Buchanan

go

Create View vInventoriesByProductsByEmployees

As

    Select Top 1000000000

        [CategoryName]

        , [ProductName]

        , [InventoryDate]

        , [Count]

        , [EmployeeFirstName] + ' ' + [EmployeeLastName] AS [EmployeeName]

    From vCategories

        Inner Join vProducts

            On vCategories.CategoryID = vProducts.CategoryID

        Inner Join vInventories

            On vProducts.ProductID = vInventories.ProductID

        Inner Join vEmployees

            On vInventories.EmployeeID = vEmployees.EmployeeID

    Order By [InventoryDate], [CategoryName], [ProductName], [EmployeeName];

go

-- Question 8 (10% pts): How can you create a view to show a list of Categories, Products,

-- the Inventory Date and Count of each product, and the Employee who took the count

-- for the Products 'Chai' and 'Chang'?

-- Here is an example of some rows selected from the view:

-- CategoryName,ProductName,InventoryDate,Count,EmployeeName

-- Beverages,Chai,2017-01-01,72,Steven Buchanan

-- Beverages,Chang,2017-01-01,46,Steven Buchanan

-- Beverages,Chai,2017-02-01,52,Robert King

go

Create View vInventoriesForChaiAndChangByEmployees

As

    Select Top 1000000000

        \*

    From vInventoriesByProductsByEmployees

    WHERE [ProductName] IN ('Chai', 'Chang')

    Order By [InventoryDate], [CategoryName], [ProductName];

go

-- Question 9 (10% pts): How can you create a view to show a list of Employees and the Manager who manages them?

-- Order the results by the Manager's name!

-- Here is an example of some rows selected from the view:

-- Manager,Employee

-- Andrew Fuller,Andrew Fuller

-- Andrew Fuller,Janet Leverling

-- Andrew Fuller,Laura Callahan

go

Create View vEmployeesByManager

As

    Select Top 1000000000

        Manager.EmployeeFirstName + ' ' + Manager.EmployeeLastName AS [Manager]

        , Employee.EmployeeFirstName + ' ' + Employee.EmployeeLastName AS [Employee]

    From vEmployees As Employee

        Inner Join vEmployees As Manager

            On Manager.EmployeeID = Employee.ManagerID

    Order By [Manager], [Employee];

go

-- Question 10 (20% pts): How can you create one view to show all the data from all four

-- BASIC Views?

-- Here is an example of some rows selected from the view:

-- CategoryID,CategoryName,ProductID,ProductName,UnitPrice,InventoryID,InventoryDate,Count,EmployeeID,Employee,Manager

-- 1,Beverages,1,Chai,18.00,1,2017-01-01,72,5,Steven Buchanan,Andrew Fuller

-- 1,Beverages,1,Chai,18.00,78,2017-02-01,52,7,Robert King,Steven Buchanan

-- 1,Beverages,1,Chai,18.00,155,2017-03-01,54,9,Anne Dodsworth,Steven Buchanan

go

Create View vInventoriesByProductsByCategoriesByEmployees

As

    -- List all columns for SELECT query to return all data and avoid column name conflicts. SELECT \* statement returns an error.

    Select Top 1000000000

        [c].[CategoryID]

        , [c].[CategoryName]

        , [p].[ProductID]

        , [p].[ProductName]

        , [p].[UnitPrice]

        , [i].[InventoryID]

        , [i].[InventoryDate]

        , [i].[Count]

        , [e].[EmployeeID]

        , [e].[EmployeeFirstName] + ' ' + [e].[EmployeeLastName] AS [Employee]

        , [m].[EmployeeFirstName] + ' ' + [m].[EmployeeLastName] AS [Manager]

    From dbo.vCategories As c

        Inner Join dbo.vProducts As p

            On c.CategoryID = p.CategoryID

        Inner Join dbo.vInventories As i

            On p.ProductID = i.ProductID

        Inner Join dbo.vEmployees As e

            On e.EmployeeID = i.EmployeeID

        Inner Join dbo.vEmployees As m

            On m.EmployeeID = e.ManagerID

    Order By [CategoryName], [ProductName];

go

-- Test your Views (NOTE: You must change the names to match yours as needed!)

Select \* From [dbo].[vCategories]

Select \* From [dbo].[vProducts]

Select \* From [dbo].[vInventories]

Select \* From [dbo].[vEmployees]

Select \* From [dbo].[vProductsByCategories]

Select \* From [dbo].[vInventoriesByProductsByDates]

Select \* From [dbo].[vInventoriesByEmployeesByDates]

Select \* From [dbo].[vInventoriesByProductsByCategories]

Select \* From [dbo].[vInventoriesByProductsByEmployees]

Select \* From [dbo].[vInventoriesForChaiAndChangByEmployees]

Select \* From [dbo].[vEmployeesByManager]

Select \* From [dbo].[vInventoriesByProductsByCategoriesByEmployees]

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/