Luis Valderrama

May 26, 2021

IT FDN 130 A

Module 08

SQL Stored Procedures

# Intro

During the course of the eighth module, I learned the basic concepts and practical uses of Stored Procedures, “Sproc” available in the MS SQL RDMS. This document addresses the uses, similarities and differences between return codes, input parameters and output parameters. This document also presents examples used in the database created for this assignment, [*Assignment08DB\_LuisValderrama*](file:///C:\Users\Luis\Documents\Foundation%20of%20Databases%20&%20SQL%20Programing\8-Stored%20Procedures\Assignment08\Assignment08DB_LuisValderrama.sql)*, (2021) (external).*

# Stored Procedures

*In SQL Server is a group of one or more Transact-SQL statements or a reference to a Microsoft .NET Framework common runtime language (CLR) method.* [*https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver15*](https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver15)*, (2017) (external).* **Data Analysts use User-Defined Functions (UDF) to**:

* **Reduce network traffic.** The commands in a procedure are executed as a single batch of codes, reducing the network traffic.
* Apply **stronger security.** 
  + Multiple users can perform operations on underlying database objects through a procedure, even if the users and programs do not have direct permissions on those underlying objects. The procedure controls what processes and activities are performed and protects the underlying database objects. This eliminates the requirement to grant permissions at the individual object level and simplifies the security layers.
  + Only the call executes the procedure is visible. Therefore, malicious users cannot see tables and database object names.
  + Procedure parameters help guard against SQL injection attacks.
  + Procedures can be encrypted, helping to obfuscate the source code.
* **Reuse codes.** The code for any repetitious database operation is the perfect candidate for encapsulation in procedures. This eliminate needless rewrites of the same code which decreases code inconsistencies and allows the code to be accessed and executed by any user.
* **Improve performance.** Procedures compile the first time it is executed and creates an execution plan that is reused for subsequent executions. Since the query processor does not have to create a new plan, it typically takes less time to process the procedure.
* Implement **easier maintenance.** When users call procedures and keep database operations in the data tier, only the procedures must be updated for any changes in the underlying database. The application tier remains separate and does not have to know how about any changes to database layouts, relationships, or processes.

The example presented below on (figure 1.1) used in the [*Assignment08DB\_LuisValderrama*](Assignment08DB_LuisValderrama.sql)*, (2021) (external)* displays the structure of a SProc created to execute a process that inserts data into the Products table. SProc are used in more complex situations while similar to Views and Functions, SProcs are established using CREATE PROCEDURE or (PROC) followed by the table’s attributes, in this case ProductName, CategoryID, and UnitPrice, all with ‘@’ symbol in front. SProcs are also contained within BEGIN and END transaction statements, defining the start and conclusion of the procedure, as well as TRY-CATCH block for error handling during the TRANSACTION block of the batch which in this example is to insert values into the table. In addition to the error messaging, the procedure also includes a RETURN CODE to indicate the status of the SProc used to troubleshoot and track the branches of logic being processed.

Create Procedure pInsProducts

(@ProductName nvarchar (100)

,@CategoryID int

,@UnitPrice money)

-- Author: Luis Valderrama

-- Desc: Insert Sproc

-- Change Log: When,Who,What

-- 2021-05-23, Luis Valderrama, Created Insert Sproc.

AS

Begin

Declare @RC int = 0;

Begin Try

Begin Transaction

Insert Into Products(ProductName, CategoryID, UnitPrice)

Values (@ProductName, @CategoryID, @UnitPrice);

Commit Transaction

Set @RC = +1

End Try

Begin Catch

Rollback Transaction

Print Error\_Message()

Set @RC = -1

End Catch

Return @RC;

End

go

***FIGURE 1.1: Example of a Stored Procedure to insert data into a table.***

# Return Codes, Input Parameters, and Output Parameters

Parameters are used to exchange data between SProcs and functions and the application or tool that called the SProc or function.

* **Return code** indicate the execution status of a procedure. The value returned is an integer.
* **Input parameters** allow the caller to pass a data value to the SProc or function.
* **Output parameters** allow the SProc to pass a data value or a cursor variable back to the caller. User-Defined functions cannot specify output parameters

The **primary differences** between Return Codes and Output parameters, is that

* With a return code the user specifies the return code for a procedure using the RETURN statement, generally to convey “failure” or “success”.
* Return codes are commonly used in control-of-flow blocks within procedures to set the return code value for each possible error situation.
* Output parameters, the user must save the return code in a variable when the procedure is executed in order to use the return code in the calling program, and generally can return one or more values.

The example presented below on (figure 1.2) used in the [*Assignment08DB\_LuisValderrama*](Assignment08DB_LuisValderrama.sql)*, (2021) (external)* displays the syntax to EXECUTE a SProc to insert basic data into the Categories table. Additionally, the syntax includes a RETURN CODE to create a custom message to indicate whether the procedure was successful or not. Additionally, since I entered data into a table with an identity option on the primary key, in this case CategoryID, it was necessary to automatically generate a new ID.

Declare @Status int;

Declare @LastID int;

Exec @Status = pInsCategories

@CategoryName = 'A';

Print @Status;

Select Case @Status

When +1 Then 'Categories Insert was successful!'

When -1 Then 'Categories Insert failed! Common Issue: Duplicate Data'

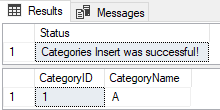
End as [Status];

Select \* From vCategories Where CategoryID = @@Identity;

go

***FIGURE 1.2: Example of a Stored Procedure test to insert data into a table.***

To further illustrate the example, (figure 1.3) below displays the results from the previously created Sproc.



***FIGURE 1.3: Example of a Stored Procedure test to insert data into a table.***

# Summary

To Recap, the eighth module taught me the basic concepts and practical uses of Stored Procedures available in the MS SQL RDMS as well as similarities and differences between Return Codes, Input Parameters and Out Parameters. The database created for this assignment, [*Assignment08DB\_LuisValderrama*](file:///C:\Users\Luis\Documents\Foundation%20of%20Databases%20&%20SQL%20Programing\8-Stored%20Procedures\Assignment08\Assignment08DB_LuisValderrama.sql)*, (2021) (external)*, applies functionalities learned throughout prior modules and the practical application of SQL Views.