**Pasos para trabajar con Procedimientos y Funciones Almacenadas en el Motor de bases de Datos SQL Server Management Studio**

Primero y princincial:
Para la creación de procedimientos almacenados se requiere el permiso CREATE PROCEDURE y el permiso ALTER en el esquema en el que se va a crear el procedimiento.

Hay dos maneras de crear un Procedimiento Almacenado:

- Una es utilizando la IU del Manejador de BD SQL Server Management Studio (SSMS)
- O la otra es a través del uso del lenguaje extendido Transact-SQL, en una ventana Consulta de SSMS, para ejecutar un procedimiento almacenado.

Con mi grupo de proyecto decidimos trabajar con la segunda opción que es a través del uso del lenguaje extendido Transact-SQL.

**Definición, Creación, y Ejecución de un Procedimiento Almacenado para Inserción de lotes de datos a una tabla CON la sentencia Bulk Insert:**

1) En SSMS, nos conectamos a una **instancia de SQL Server**
2) Seleccionamos **"Nueva consulta"** en la barra de herramientas.
3) Con la instrucción **CREATE PROCEDURE ImportarClientesDesdeCSV** definimos un nuevo procedimiento almacenado llamado ImportarClientesDesdeCSV
4) Declaramos un parámetro de entrada **@RutaArchivo NVARCHAR(255)** (con nombre RutaArchivo y de tipo Nvarchar) para especificar la ruta completa del archivo CSV.
5) Iniciamos el **bloque BEGIN TRY - END TRY** para encapsular la operación de importación. Si ocurre algún error durante la importación, el control se transferirá al bloque CATCH.
6) Uso de la sentencia **BULK INSERT**:
   La sentencia `BULK INSERT` es una operación en SQL utilizada para importar grandes volúmenes de datos de un archivo externo a una tabla en una base de datos de forma rápida y eficiente. Esta sentencia es especialmente útil en escenarios donde se requiere cargar grandes cantidades de datos desde archivos, como archivos CSV, archivos de texto delimitado u otros formatos compatibles.
   Se puede definir el formato de los datos que se van a importar (como delimitadores de columnas y de filas), lo que permite flexibilidad en la importación de archivos con diferentes estructuras de datos.
7) Si la importación se realiza sin errores, se ejecuta la instrucción **PRINT** para mostrar el mensaje "Importación exitosa.".
8) Iniciamos el **bloque BEGIN CATCH - END CATCH** para el manejo de errores
9) Seleccionamos **"Ejecutar"** en la barra de herramientas para ejecutar la consulta. Y así se creará el procedimiento almacenado.

10) Para ejecutar el procedimiento almacenado, escribimos una instrucción **EXECUTE**, y proporcionamos los valores para los parámetros.
11) Y seleccionamos **“Ejecutar”**.

**Definición, Creación, y Ejecución de un Procedimiento Almacenado para la Inserción de lotes de datos a una tabla SIN la sentencia Bulk Insert:**

1) **CREATE PROCEDURE InsertarProducto** define un nuevo procedimiento llamado “InsertarProducto”.
2) Se declaran los parámetros de entrada con sus tipos de datos y longitudes, Estos parámetros capturan los valores que se utilizarán para crear el nuevo registro en la tabla "productos".
3) La instrucción **SET NOCOUNT ON** evita que el procedimiento devuelva la cantidad de filas afectadas por las instrucciones SELECT posteriores. Esto puede mejorar el rendimiento, especialmente cuando se trabaja con conjuntos de resultados grandes.
4) **`BEGIN TRY`**: Inicia un bloque TRY para encapsular la operación de inserción. Si ocurre algún error durante la inserción, el control se transferirá al bloque CATCH.
5) **`INSERT INTO [dbo].[productos]`**: Esta instrucción SQL insertará un nuevo registro en la tabla "productos".
6) Los valores entre paréntesis después de **`VALUES`** corresponden a los valores de los parámetros de entrada, que serán asignados a las columnas correspondientes en la tabla.
7) Si el producto se insertó correctamente se muestra el mensaje “Producto insertado correctamente”
8) Se cierra el bloque TRY de encapsulamiento y se inicia el **bloque CATCH** de captura y manejos de errores que ocurrieron en el bloque TRY
9) Si hubo algún error en la inserción muestra el mensaje de error “Error al insertar producto”
10) Fin del bloque CATCH
11) Fin de la definición del procedimiento.
12) Seleccionamos **“Ejecutar”** en la barra de herramientas para ejecutar la consulta.Y se creará el procedimiento almacenado.
13) Una vez creado el procedimiento, para hacer uso del mismo debemos escribir una instrucción EXECUTE en una nueva consulta, proporcionar valores para los parámetros y, a continuación, seleccionar Ejecutar.
14) Se insertará un nuevo registro en la tabla Producto.

**Definición, Creación, y Ejecución, de un Procedimiento Almacenado para modificar un registro:**

Este procedimiento almacenado en T-SQL, llamado `ActualizarProducto`, se utiliza para modificar los datos de un producto específico en una tabla llamada `productos`. La clave para identificar el producto a actualizar es el valor de `@id_producto`.

1) Definimos un nuevo procedimiento almacenado llamado “ActualizarProducto” con la siguiente sentencia: **CREATE PROCEDURE ActualizarProducto**
2) Declaramos los parámetros de entrada. Los parámetros tienen un valor por defecto de NULL, lo que significa que si no se proporciona un valor para un parámetro en particular, el valor actual en la base de datos no se modificará. Esto brinda flexibilidad al procedimiento.
3) Se inicia el bloque **BEGIN-TRY** que encapsula la operación de modificación. Si ocurre algún error durante la modificación, el control se transferirá al bloque CATCH.
4) La instrucción **UPDATE [dbo].[productos]** actualizará los registros en la tabla productos.
5) Con **SET**: Especificamos los nuevos valores para cada columna. La función COALESCE nos permite especificar un valor alternativo en caso de que el valor sea NULL, se utiliza para asignar el nuevo valor si se proporcionó, o el valor actual de la base de datos si no se proporcionó. Esto permite actualizar sólo las columnas cuyos valores se han pasado como parámetros.
6) Luego **`WHERE id_producto = @id_producto`:** Filtra los registros a actualizar según el valor de `@id_producto`.
7) Fin del bloque TRY
8) Se inicia el **bloque CATCH** y si se produjo un error en el bloque TRY, este lo captura y muestra el mensaje de error producido.
9) Fin del bloque CATCH .
10) Fin de la definición del Procedimiento “ActualizarProducto”
11) Seleccionamos **“Ejecutar”** en la barra de herramientas para ejecutar la consulta.Y se creará el procedimiento almacenado “ActualizarProducto”.
12) Una vez creado el procedimiento, para hacer uso del mismo debemos escribir una instrucción EXECUTE en una nueva consulta, proporcionar valores para los parámetros y, a continuación, seleccionar Ejecutar.
13) Se actualizará el registro en la tabla Producto.

**Definición, Creación, y Ejecución, de un Procedimiento Almacenado para eliminar un registro**:

Este procedimiento está diseñado específicamente para eliminar registros de una tabla llamada "productos" en una base de datos.

1) Definimos un nuevo procedimiento almacenado llamado “EliminarProducto” con la siguiente sentencia: **CREATE PROCEDURE EliminarProducto**
2) Se define un parámetro de entrada llamado **`@id_producto`** de tipo entero. Este parámetro servirá como identificador único para el producto que queremos eliminar.
3) Se inicia el bloque **BEGIN-TRY** que encapsula la operación de modificación. Si ocurre algún error durante la modificación, el control se transferirá al bloque CATCH.
4) La instrucción **`DELETE FROM productos WHERE id_producto = @id_producto`** SQL elimina uno o más registros de la tabla "productos".

5) Con la condición **`WHERE id_producto = @id_producto`** especifica que solo se eliminarán los registros cuyo campo "id_producto" coincida con el valor proporcionado en el parámetro de entrada.
6) Fin del bloque TRY
7) Se inicia el **bloque CATCH** y si se produjo un error en el bloque TRY, este lo captura y muestra el mensaje de error producido.
8) Fin del bloque CATCH .
   Fin de la definición del Procedimiento “EliminarProducto”
9) Seleccionamos **“Ejecutar”** en la barra de herramientas para ejecutar la consulta.Y se creará el procedimiento almacenado “EliminarProducto”.
10) Una vez creado el procedimiento, para hacer uso del mismo debemos escribir una instrucción EXECUTE en una nueva consulta, proporcionar valores para los parámetros y, a continuación, seleccionar Ejecutar.
11) Se eliminará el registro en la tabla Producto.

**Definición, Creación, y Ejecución, de una Función Almacenada definida por el usuario, de tipo escalar:**

Se requiere el permiso `CREATE FUNCTION` en la base de datos y el permiso `ALTER` en el esquema en el que se va a crear la función. Si la función especifica un tipo definido por el usuario, requiere el permiso `EXECUTE` en el tipo.

Pasos a seguir para la Función: **CalcularTotalPedidos**

**Objetivo:** Esta función está diseñada para calcular el total de todos los pedidos realizados por un cliente específico en una base de datos.

1) Conectarse a una instancia de SQL Server en el Motor de BD SQL Server Management Studio.
2) Escribir el código que va a definir la sintaxis de una función almacenada:
   i) **CREATE FUNCTION CalcularTotalPedidos** define una nueva función almacenada que se va a llamar “CalcularTotalPedidos”
   ii) **(@id_cliente INT)** especifica un parámetro de entrada @id_cliente de tipo INT. Este parámetro representará el identificador único del cliente para el cual se desea calcular el total de pedidos.
   iii) **RETURNS FLOAT** especifica que la función va a devolver UN VALOR de salida de tipo decimal de precisión simple (float). Este valor representará el total de todos los pedidos del cliente.
   iv) dentro del bloque **BEGIN-END** se escribe la lógica de la función
   v) **DECLARE @total_pedidos FLOAT** declara una variable local llamada "@total_pedidos" de tipo float para almacenar el resultado del cálculo del total de pedidos.
   vi) **SELECT @total_pedidos = SUM(monto_total)**

**FROM pedido**
**WHERE id_cliente = @id_cliente**

Esta línea realiza una consulta a la tabla "pedido" para calcular el total de todos los pedidos del cliente especificado. La función `SUM(monto_total)` suma el campo "monto_total" de todos los registros donde el campo "id_cliente" coincide con el valor pasado como parámetro. El resultado de esta suma se almacena en la variable "@total_pedidos".

vii) **RETURN ISNULL(@total_pedidos, 0)** Esta línea devuelve el valor de la variable "@total_pedidos". Si el cliente no tiene pedidos asociados, la función `ISNULL` reemplaza el valor nulo de "@total_pedidos" por 0, asegurando que siempre se devuelva un valor numérico.

3) Seleccionamos **"Ejecutar"** en la barra de herramientas para ejecutar la consulta.Y se creará la función almacenada "CalcularTotalPedidos".
4) Una vez creado el procedimiento, para hacer uso del mismo debemos escribir una instrucción **SELECT CalcularTotalPedidos** en una nueva consulta, proporcionar valores para los parámetros y, a continuación, seleccionar Ejecutar.
5) La función devolverá el total de pedidos realizados por el cliente o devolverá 0 en caso de que el cliente no haya realizado ningún pedido