- **Procedimientos almacenados**
- **Funciones**

## Procedimientos almacenados

Un procedimiento almacenado, o stored procedure en inglés, es un conjunto de sentencias SQL, al cual se le asigna un nombre y se almacena como un objeto en el servidor de la base de datos, de manera que este pueda ser reutilizado y compartido con diferentes programas.

Al ejecutar un procedimiento por primera vez, SQL Server un plan de ejecución y lo guarda en el plan cache, de forma que en ejecuciones futuras se pueda reutilizar dicho plan y el procedimiento se realice muy rápido con rendimiento confiable.

**Ventajas**

- Reducción del tráfico en la red
- Encapsula lógica de negocio
- Añade seguridad para proteger contra ataques de SQL Injection
- Mantenibilidad

Sintaxis:

```
CREATE [OR ALTER] PROCEDURE nombre_procedimiento
[
@param1 datatype [, ...n]
]
AS
BEGIN
    sentencia_sql;
    [... n]
END
```

La información que se encuentra entre corchetes es opcional. La notación "**, … n**" indica que esa instrucción se puede repetir una o más veces. De esta manera podemos decir que el procedimiento puede recibir más de un parámetro y que puede ejecutar más de una sentencia SQL. Además, podemos ocupar "**OR ALTER**" para modificar el procedimiento si ya existe.

Ocupando la base de datos de la biblioteca, queremos una lista de los libros prestados, con su fecha de préstamo, su fecha de devolución y el nombre completo del usuario al que se le hizo el préstamo. Si prestamos atención, la información que queremos es una

consulta que hace JOIN entre cuatro tablas, por lo que escribir esa consulta una y otra vez puede ser tedioso y poco práctico. En lugar de eso, podemos generar un procedimiento almacenado que se encargue de eso por nosotros. La consulta del JOIN:

```
SELECT l.titulo Libro, p.fecha_prestamo 'Fecha de préstamo',
    p.fecha_devolucion 'Fecha de devolución', CONCAT(u.nombre, ' ', u.apellido) as Cliente
FROM PRESTAMO AS p
INNER JOIN USUARIO AS u ON p.codigo_usuario = u.codigo
INNER JOIN EJEMPLAR AS e ON e.codigo = p.codigo_ejemplar
INNER JOIN LIBRO AS l ON l.codigo = e.codigo_libro;
```

Lo único que debemos hacer en este caso es ingresar esa sentencia SQL de consulta en nuestra definición de procedimiento. Así:

```
CREATE PROCEDURE listaPrestamoUsuarioLibro
AS
BEGIN
    SELECT l.titulo Libro, p.fecha_prestamo 'Fecha de préstamo',
        p.fecha_devolucion 'Fecha de devolución', CONCAT(u.nombre, ' ', u.apellido) as Cliente
    FROM PRESTAMO AS p
    INNER JOIN USUARIO AS u ON p.codigo_usuario = u.codigo
    INNER JOIN EJEMPLAR AS e ON e.codigo = p.codigo_ejemplar
    INNER JOIN LIBRO AS l ON l.codigo = e.codigo_libro;
END;
```

Ahora que tenemos nuestro procedimiento, la forma de mandarlo a llamar es:

**`EXEC nombre_procedimiento`**

El cual en nuestro caso sería **`EXEC listaPrestamoUsuarioLibro`**.

## Parámetros

Como ya se mencionó, se pueden agregar varios parámetros a los procedimientos, pero además de eso se pueden definir también como parámetros de salida, de forma que se puedan almacenar en variables que declaremos.

```
/*
Modificando el procedimiento creado para que reciba un parámetro de entrada
y un parámetro de salida
*/
CREATE OR ALTER PROCEDURE listaPrestamoUsuarioLibro
@titulo VARCHAR(100),
@cantidad SMALLINT OUTPUT
AS
BEGIN
    SELECT l.titulo Libro, p.fecha_prestamo 'Fecha de préstamo',
        p.fecha_devolucion 'Fecha de devolución', CONCAT(u.nombre, ' ', u.apellido) as Cliente
    FROM PRESTAMO AS p
    INNER JOIN USUARIO AS u ON p.codigo_usuario = u.codigo
    INNER JOIN EJEMPLAR AS e ON e.codigo = p.codigo_ejemplar
    INNER JOIN LIBRO AS l ON l.codigo = e.codigo_libro
    WHERE l.titulo LIKE CONCAT('%', @titulo, '%');

    SELECT @cantidad = @@ROWCOUNT;
END;
```

Podemos observar que al parámetro cantidad después de su tipo de dato le especificamos que es de salida con “OUTPUT”. Luego vemos como la última sentencia

SQL es una consulta en la que asignamos el valor de @@ROWCOUNT al parámetro de salida cantidad.

En T-SQL, @@**ROWCOUNT** es una variable global que el servidor modifica. Esta variable devuelve un entero que significa el número de filas que fueron afectadas en la sentencia SQL ejecutada previamente. Esto puede ser un INSERT, UPDATE, DELETE O SELECT.

Ahora, para ejecutar nuestro procedimiento:

```
/*
Ahora creamos una variable para almacenar la salida del procedimiento
*/
DECLARE @count SMALLINT;

-- Ejecutando el procedimiento
EXEC listaPrestamoUsuarioLibro
    @titulo = 'a',
    @cantidad = @count OUTPUT;

SELECT @count;
```

Es importante destacar que para que estas instrucciones funcionen, debe seleccionarse todo desde la instrucción **`DECLARE`** hasta **`SELECT @count;`** y ejecutarse todo al mismo tiempo.

## Estructuras de control

### IF…ELSE

T-SQL también nos permite utilizar estructuras de control de selección, como lo son if y else.

```
-- ESTRUCTURAS DE CONTROL
IF (expresion_booleana)
BEGIN
    -- Sentencia(s) que se ejecuta(n) si la expresion es verdadera
END
ELSE
BEGIN
    -- Sentencia(s) que se ejecuta(n) si la expresion es falsa
END
```

Ejemplo:

```
BEGIN
    DECLARE @x INT = 10,
            @y INT = 20;
    IF (@x > 0)
    BEGIN
        IF (@x < @y)
        BEGIN
            PRINT '0 < x < y';
        END
        ELSE
        BEGIN
            PRINT '0 < y <= x';
        END
    END
END
```

## WHILE

De igual forma, podemos hacer estructuras de control iterativas. La sintaxis:

```
WHILE expresion_booleana
BEGIN
    -- Sentencias sql a ejecutar
END
```

Ejemplo:

```
-- EJEMPLO
BEGIN
    DECLARE @a INT = 0, @b INT = 1, @tmp INT, @i INT = 1;
    WHILE @i < 10
    BEGIN
        SELECT @tmp = @b, @b = @a + @b, @a = @tmp, @i = @i + 1;
        PRINT 'b = ' + CAST(@b AS VARCHAR);
    END
END
```

Este ejemplo imprime en pantalla los 10 primeros números calculados de la serie de Fibonacci.

Tanto dentro de los procedimientos almacenados como de las funciones, pueden utilizarse estructuras de control para añadir lógica a conveniencia, como se mostrará más adelante.

# Funciones

De forma similar a los procedimientos almacenados, las funciones son conjuntos de sentencias SQL que se almacenan como objetos en la base de datos que suelen encapsular procesos lógicos para su reutilización.

Sin embargo, la principal diferencia con los procedimientos almacenados es que es obligatorio que las funciones devuelvan un valor, al contrario de los procedimientos que pueden o no tener valores de retorno. Otras diferencias entre procedimientos y funciones son:

- Las funciones pueden tener sólo parámetros de entrada, mientras que los procedimientos pueden tener de entrada y salida.
- Las funciones pueden ser llamadas desde un procedimiento, pero no se puede llamar un procedimiento desde una función.
- Las funciones escalares pueden usarse como campo en una consulta.
- Sólo se permiten instrucciones de consulta, no es posible insertar, actualizar o eliminar datos de tablas.

Existen 2 tipos principales de funciones. Las **funciones escalares** que devuelven un único valor y las **funciones de tabla**, que retornan tablas generadas dentro de la misma.

## Funciones Escalares

Las funciones escalares reciben varios parámetros y devuelven un único valor. Sintaxis:

```
CREATE [OR ALTER] FUNCTION dbo.[nombre_funcion] (
    @param1 datatype [, ...n]
)
RETURNS datatype
AS
BEGIN
    [sentencias sql]
    [... n]
    RETURN @variable_de_retorno
END
```

Como puede observarse, la sintaxis en muy similar a la de un procedimiento, con algunas diferencias: los parámetros (que serán únicamente de entrada) van encerrados en paréntesis. RETURNS indica el tipo de dato que se devolverá y RETURN devuelve el valor en sí. De igual forma como con procedimientos, se puede sustituir CREATE por ALTER para modificar la función una vez haya sido creada.

A continuación, algunos ejemplos de funciones, siempre sobre la base de datos de la biblioteca.

## Ejemplo

Dado el código de un libro, se desea obtener el total recaudado por los prestamos de sus ejemplares. Si el código dado no existe, devolver -1.

```
CREATE FUNCTION recaudado(@c_libro int)
RETURNS money
AS
BEGIN
    DECLARE @total money;
    SELECT @total = SUM(precio)
        FROM LIBRO l INNER JOIN EJEMPLAR e
        ON l.codigo = e.codigo_libro
        INNER JOIN PRESTAMO p
        ON p.codigo_ejemplar = e.codigo
        WHERE codigo_libro = @c_libro;
    IF(@total IS NULL)
        SET @total = -1
    RETURN @total;
END
```

Como puede observarse, dentro de las funciones pueden incluirse estructuras de control para añadir lógica según convenga.

Prestar atención a que **RETURNS** no es igual a **RETURN**.

**RETURNS** especifica el tipo de dato que la función va devolver, que puede ser cualquiera de los tipos validos en SQL Server. **RETURN** es el que devuelve el valor.

Para ejecutar la función una vez creada:

```
SELECT dbo.recaudado(1) AS 'Recaudado';
```

Las funciones brindan la posibilidad que pueden ser incluidas en una consulta como si de otra columna se tratara. Si se quisiera obtener el total recaudado de cada uno de los libros, se puede ejecutar:

```
SELECT codigo, titulo, dbo.recaudado(codigo) AS Recaudado FROM LIBRO;
```

De esta forma veríamos cada uno de los títulos registrados junto a su total recaudado.

## Funciones de tabla

Este tipo de funciones brinda como valor de retorno una tabla generada dentro de la misma. Estas brindan la posibilidad de incluir estructuras de control como IF o WHILE. Pueden ser utilizadas en el FROM de una consulta y también en las sentencias JOIN.

Para ejemplo, se puede recrear el ejemplo hecho en la parte de Procedimientos, pero ahora como una función que devuelve una tabla.

```
CREATE FUNCTION lista()
RETURNS table
AS
    RETURN
    SELECT l.titulo Libro, p.fecha_prestamo 'Fecha de préstamo',
        p.fecha_devolucion 'Fecha de devolución', CONCAT(u.nombre, ' ', u.apellido) as Cliente
    FROM PRESTAMO AS p
    INNER JOIN USUARIO AS u ON p.codigo_usuario = u.codigo
    INNER JOIN EJEMPLAR AS e ON e.codigo = p.codigo_ejemplar
    INNER JOIN LIBRO AS l ON l.codigo = e.codigo_libro;
```

La tabla generada por esta función puede ser generada total o parcialmente, de la siguiente forma:

```
SELECT * FROM dbo.lista();

SELECT Libro, Cliente FROM dbo.lista();
```

Asimismo, la tabla generada puede ser tomada para sentencias JOIN.