Tema 1: Procedimientos y Funciones Almacenadas

Este tema posee dos objetivos principales:

- Comprender la diferencia entre procedimientos y funciones almacenadas.
- Aplicar procedimientos y funciones en la implementación de operaciones CRUD.

Partamos por la base y definamos primero qué es un procedimiento almacenado y qué es una función almacenada.

Procedimiento Almacenado

Un ***procedimiento almacenado*** es un conjunto de sentencias SQL que se pueden guardar en una base de datos, pueden ejecutarse bajo demanda para **realizar tareas de manipulación y validación de datos**, reduciendo la necesidad de escribir código SQL repetitivo para operaciones comunes. Son útiles en la gestión de bases de datos porque promueven la eficacia y la reutilización. Además, permiten mejorar la seguridad y la mantenibilidad de las bases de datos. Entre las **ventajas** de los procedimientos almacenados de SQL se incluyen las siguientes:

- **Reutilización del código**: Una vez creado un procedimiento almacenado, se puede llamar tantas veces como sea necesario, eliminando la redundancia en el código SQL.
- **Rendimiento mejorado**: Los procedimientos almacenados suelen ejecutarse más rápido porque están pre compilados y almacenados en el servidor de la base de datos, lo que reduce la latencia de la red y el tiempo de compilación.
- **Seguridad**: Los procedimientos almacenados pueden mejorar la seguridad de los datos y el control del acceso a datos sensibles, concediendo a los usuarios permiso para ejecutar un procedimiento almacenado sin acceso directo a las tablas.

La **sintaxis** para crear un procedimiento almacenado puede variar ligeramente en función del sistema de base de datos. A continuación se muestra un ejemplo general utilizando la sintaxis de SQL Server:

```
CREATE PROCEDURE NombreProcedimiento
    @Parametro1 INT,
    @Parametro2 VARCHAR(50)
AS
BEGIN
    -- Cuerpo de la consulta
     SELECT * FROM NombreTabla WHERE Columna1 = @Parametro1 AND
Columna2 = @Parametro2;
END;
```

- **CREATE PROCEDURE**: Este comando se utiliza para definir un nuevo procedimiento almacenado.
- **NombreProcedimiento**: El nombre dado al procedimiento almacenado. Debe ser único dentro de la base de datos.
- **@Parametro1, @Parametro2**: Los parámetros son opcionales, permiten que el procedimiento reciba datos de entrada. Cada parámetro se define con un símbolo @ y un tipo de datos (por ejemplo: `INT`, `VARCHAR(50)`).
- **AS BEGIN … END**: Las sentencias SQL dentro de `BEGIN` y `END` forman el cuerpo del procedimiento, donde se ejecuta la lógica principal.

Crear un procedimiento almacenado en SQL Server implica definir el nombre del procedimiento, los parámetros y las sentencias SQL que componen su cuerpo. En SQL Server, el comando `EXEC` o `EXECUTE` llama a un procedimiento almacenado. Por ejemplo:

```
EXEC NombreProcedimiento @Parametro1 = 102, @Parametro2 =´test´;
```

Función Almacenada

Una ***función almacenada***, también conocida como simplemente función, es un conjunto de instrucciones o bloques de código SQL que se almacenan en una base de datos y pueden ser invocados o llamados en diferentes momentos para realizar tareas específicas. Puede recibir cero o varios parámetros y **siempre retorna un valor como**

resultado. Normalmente, se utiliza mediante una sentencia `SELECT` o dentro de una expresión.

Existen varios **tipos** de funciones, las **escalares** devuelven un único valor, como un número entero, una cadena de texto o un valor booleano. Las funciones **con valores de tabla** devuelven una tabla completa como resultado. Esto puede ser una tabla de varias instrucciones o una tabla en línea. Las **de agregado** toman un conjunto de valores de una columna y devuelven un único valor de resumen, como el `SUM()` o `AVG()`.

Las funciones dentro de SQL Server tienen las siguientes **características**:

- **Determinismo**: Las funciones pueden ser deterministas (siempre devuelven el mismo resultado para los mismos parámetros de entrada) o no deterministas (pueden devolver resultados diferentes).
- **Uso**: Se pueden usar en sentencias `SELECT`, en cláusulas `WHERE` y en otras partes de una consulta.
- **Reutilización**: Se crean para ejecutar un conjunto de instrucciones SQL repetidamente, lo que ayuda a mantener el código más limpio y fácil de mantener.

La **sintaxis** para crear una función puede variar ligeramente según el sistema de base de datos. A continuación se muestra un ejemplo general utilizando la sintaxis de SQL Server:

```
CREATE FUNCTION NombreFuncion (
    @Parametro1 INT,
    @Parametro2 INT)
RETURNS INT
DETERMINISTIC
BEGIN
    -- Cuerpo de la consulta
    RETURN <valor que corresponda>;
END;
```

- **CREATE FUNCTION**: Este comando se utiliza para definir una nueva función.

- **NombreFuncion**: El nombre dado a la función. Debe ser único dentro de la base de datos.
- **@Parametro1, @Parametro2**: Los parámetros permiten que la función reciba datos de entrada. Cada parámetro se define con un símbolo @ y un tipo de datos (por ejemplo: `INT`, `VARCHAR(50)`).
- **DETERMINISTIC**: Es una palabra clave que especifica que la función que definimos es determinística (si retorna el mismo resultado si se la invoca de nuevo con los mismos valores de entrada).
- **BEGIN … END**: Las sentencias SQL dentro de `BEGIN` y `END` forman el cuerpo de la función, donde se ejecuta la lógica principal.
- **RETURN**: El return debe aparecer como mínimo una vez dentro del cuerpo de la función, ya que todas las funciones devuelven al menos un valor.

Crear una función en SQL Server implica definir el nombre de la función, los parámetros de entrada, el tipo de valor que devuelve la función, el comportamiento de la función y su cuerpo.

En SQL Server, se llama a una función mediante la sentencia `SELECT`. Por ejemplo:

```
SELECT NombreFuncion(2, 4);
```

Ahora resaltando los aspectos más importantes de cada uno:

Aspecto	Procedimiento Almacenado	Función Almacenada
Propósito	Ejecutar operaciones o procesos (puede modificar datos).	Calcular y devolver un valor o conjunto de valores.
Tipo de Retorno	Ninguno o varios (usando parámetros de salida).	Obligatoriamente devuelve un valor.
Uso de sentencias SQL	No puede usarse	Puede usarse dentro de un

	directamente en un `SELECT`.	`SELECT`, `WHERE`, etc.
Permite transacciones (`INSERT, UPDATE, DELETE`)	Sí	No
Ejemplo de uso	`EXEC InsertarSensor @modelo = 'Gardena 1188' ...;`	`SELECT CalcularPromedioHumedad(3);`

Habiendo sentado ya las bases de este tema pasemos a la práctica, recordemos que las tareas de este tema son:

- Realizar al menos tres procedimientos almacenados que permitan insertar, modificar y borrar registros de alguna de las tablas del proyecto.
- Insertar un lote de datos en las tablas mencionadas (guardar el script) con sentencias `INSERT` y otro lote invocando a los procedimientos creados.
- Realizar `UPDATE` y `DELETE` sobre algunos de los registros insertados en esas tablas invocando a los procedimientos.
- Desarrollar al menos tres funciones almacenadas.
- Comparar la eficiencia de las operaciones directas versus el uso de procedimientos y funciones.

Tarea N°1

Realizar al menos tres procedimientos almacenados que permitan insertar, modificar y borrar registros de alguna de las tablas del proyecto.

Para esta tarea, vamos a desarrollar tres procedimientos de operaciones CRUD para la tabla “ZonaRiego”:

Primer procedimiento - Insertar Registro:

```
CREATE PROCEDURE sp_insertar_zona_riego
```

```
    @nombre VARCHAR(15),
    @descripcion VARCHAR(30),
    @superficie FLOAT
AS
BEGIN
    INSERT INTO ZonaRiego (nombre, descripcion, superficie)
    VALUES (@nombre, @descripcion, @superficie);
END;
GO
```

Este procedimiento está definido para tomar tres parámetros de entrada, correspondientes al nombre, la descripción y superficie de la zona de riego, y dentro del cuerpo se encuentra la secuencia de inserción de los valores dentro de la tabla "ZonaRiego".

Segundo procedimiento - Modificar Registro:

```
CREATE PROCEDURE sp_modificar_zona_riego
    @id_zona INT,
    @nombre VARCHAR(15),
    @descripcion VARCHAR(30),
    @superficie FLOAT
AS
BEGIN
    UPDATE ZonaRiego
    SET nombre = @nombre,
        descripcion = @descripcion,
        superficie = @superficie
    WHERE id_zona = @id_zona;
END;
GO
```

Este procedimiento, similar al anterior, está definido para tomar cuatro parámetros de entrada, correspondientes al identificador de la zona, su nombre, la

descripción y superficie, y dentro del cuerpo se encuentra la secuencia de actualización de los valores de la tupla correspondiente dentro de la tabla “ZonaRiego”.

Tercer Procedimiento - Eliminar Registro:

```
CREATE PROCEDURE sp_eliminar_zona_riego
    @id_zona INT
AS
BEGIN
    DELETE FROM ZonaRiego
    WHERE id_zona = @id_zona;
END;
GO
```

Este último procedimiento está definido para tomar solo un parámetro de entrada, correspondientes al identificador de la zona que se desea eliminar, y dentro del cuerpo se encuentra la secuencia de eliminación de la tupla correspondiente dentro de la tabla “ZonaRiego”.

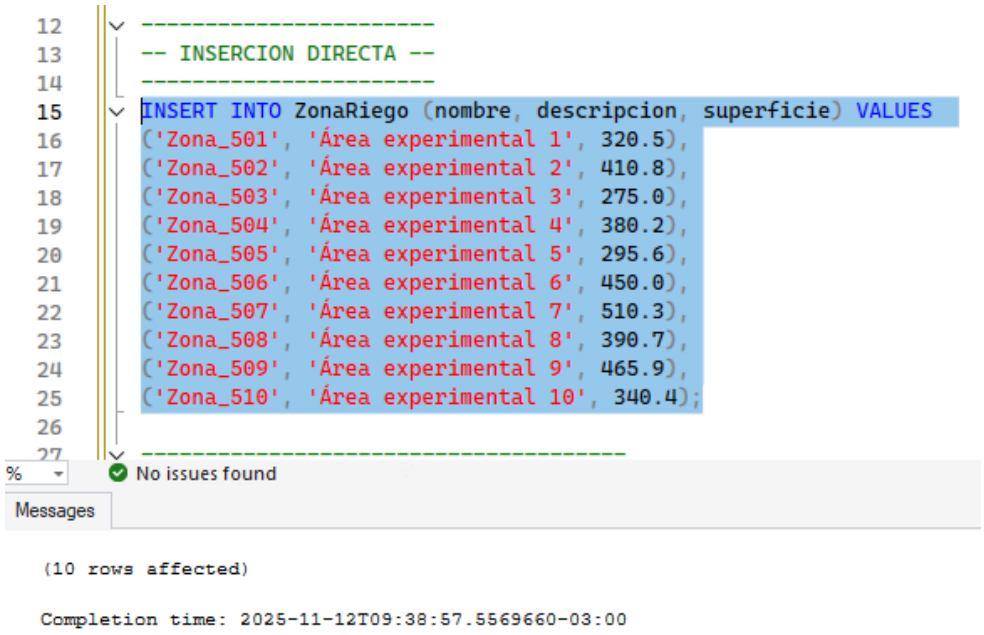
Tarea N°2

Insertar un lote de datos en las tablas mencionadas (guardar el script) con sentencias `INSERT` y otro lote invocando a los procedimientos creados.

Inserción directa:

```
INSERT INTO ZonaRiego (nombre, descripcion, superficie) VALUES
('Zona_501', 'Área experimental 1', 320.5),
('Zona_502', 'Área experimental 2', 410.8),
('Zona_503', 'Área experimental 3', 275.0),
('Zona_504', 'Área experimental 4', 380.2),
('Zona_505', 'Área experimental 5', 295.6),
('Zona_506', 'Área experimental 6', 450.0),
('Zona_507', 'Área experimental 7', 510.3),
('Zona_508', 'Área experimental 8', 390.7),
```

```
('Zona_509', 'Área experimental 9', 465.9),
('Zona_510', 'Área experimental 10', 340.4);
```



Inserción mediante procedimientos:

```
EXEC sp_insertar_zona_riego 'Zona_511', 'Zona agregada mediante SP 1', 410.5;
EXEC sp_insertar_zona_riego 'Zona_512', 'Zona agregada mediante SP 2', 355.0;
EXEC sp_insertar_zona_riego 'Zona_513', 'Zona agregada mediante SP 3', 430.3;
EXEC sp_insertar_zona_riego 'Zona_514', 'Zona agregada mediante SP 4', 298.7;
EXEC sp_insertar_zona_riego 'Zona_515', 'Zona agregada mediante SP 5', 500.0;
EXEC sp_insertar_zona_riego 'Zona_516', 'Zona agregada mediante SP 6', 475.2;
EXEC sp_insertar_zona_riego 'Zona_517', 'Zona agregada mediante SP 7', 382.9;
EXEC sp_insertar_zona_riego 'Zona_518', 'Zona agregada mediante SP 8', 445.4;
```

```
EXEC sp_insertar_zona_riego 'Zona_519', 'Zona agregada mediante SP 9', 410.8;
EXEC sp_insertar_zona_riego 'Zona_520', 'Zona agregada mediante SP 10', 390.1;
```

```
27  ---------------------------------------
28  -- INSERCION MEDIANTE PROCEDIMIENTO --
29  ---------------------------------------
30  EXEC sp_insertar_zona_riego 'Zona_511', 'Zona agregada mediante SP 1', 410.5;
31  EXEC sp_insertar_zona_riego 'Zona_512', 'Zona agregada mediante SP 2', 355.0;
32  EXEC sp_insertar_zona_riego 'Zona_513', 'Zona agregada mediante SP 3', 430.3;
33  EXEC sp_insertar_zona_riego 'Zona_514', 'Zona agregada mediante SP 4', 298.7;
34  EXEC sp_insertar_zona_riego 'Zona_515', 'Zona agregada mediante SP 5', 500.0;
35  EXEC sp_insertar_zona_riego 'Zona_516', 'Zona agregada mediante SP 6', 475.2;
36  EXEC sp_insertar_zona_riego 'Zona_517', 'Zona agregada mediante SP 7', 382.9;
37  EXEC sp_insertar_zona_riego 'Zona_518', 'Zona agregada mediante SP 8', 445.4;
38  EXEC sp_insertar_zona_riego 'Zona_519', 'Zona agregada mediante SP 9', 410.8;
39  EXEC sp_insertar_zona_riego 'Zona_520', 'Zona agregada mediante SP 10', 390.1;
```

```
00 %   No issues found
Messages
```

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2025-11-12T09:41:46.4779317-03:00
```

Ahora vamos a hacer un `SELECT` a la tabla de “ZonaRiego” para ver si efectivamente se insertaron los registros.

```
47  SELECT * FROM ZonaRiego;
```

100 % No issues found

Results | Messages | Client Statistics

	id_zona	nombre	descripcion	superficie
134	134	Zona_134	Área de cultivo número 134	140,68
135	135	Zona_135	Área de cultivo número 135	1089,97
136	136	Zona_501	Área experimental 1	320,5
137	137	Zona_502	Área experimental 2	410,8
138	138	Zona_503	Área experimental 3	275
139	139	Zona_504	Área experimental 4	380,2
140	140	Zona_505	Área experimental 5	295,6
141	141	Zona_506	Área experimental 6	450
142	142	Zona_507	Área experimental 7	510,3
143	143	Zona_508	Área experimental 8	390,7
144	144	Zona_509	Área experimental 9	465,9
145	145	Zona_510	Área experimental 10	340,4
146	146	Zona_511	Zona agregada mediante SP 1	410,5
147	147	Zona_512	Zona agregada mediante SP 2	355
148	148	Zona_513	Zona agregada mediante SP 3	430,3
149	149	Zona_514	Zona agregada mediante SP 4	298,7
150	150	Zona_515	Zona agregada mediante SP 5	500
151	151	Zona_516	Zona agregada mediante SP 6	475,2
152	152	Zona_517	Zona agregada mediante SP 7	382,9
153	153	Zona_518	Zona agregada mediante SP 8	445,4
154	154	Zona_519	Zona agregada mediante SP 9	410,8
155	155	Zona_520	Zona agregada mediante SP 10	390,1

Tarea N°3

Realizar `UPDATE` y `DELETE` sobre algunos de los registros insertados en esas tablas invocando a los procedimientos.

Actualización:

```
EXEC sp_modificar_zona_riego @id_zona = 137, @nombre = 'Zona_Modificada', @descripcion = 'Descripción actualizada', @superficie = 500.0;
```

```
-----------------
-- ACTUALIZACION --
-----------------
SELECT * FROM ZonaRiego
WHERE id_zona = 137;
GO

EXEC sp_modificar_zona_riego @id_zona = 137, @nombre = 'Zona_Modificada', @descripcion = 'Descripción actualizada', @superficie = 500.0;
GO

SELECT * FROM ZonaRiego
WHERE id_zona = 137;
GO
```

100 % — No issues found

Results | Messages

	id_zona	nombre	descripcion	superficie
1	137	Zona_502	Área experimental 2	410,8

	id_zona	nombre	descripcion	superficie
1	137	Zona_Modificada	Descripción actualizada	500

Eliminación:

```
EXEC sp_eliminar_zona_riego @id_zona = 150;
```

```
-----------------
-- ELIMINACION --
-----------------
SELECT * FROM ZonaRiego
WHERE id_zona = 150;
GO

EXEC sp_eliminar_zona_riego @id_zona = 150;
GO

SELECT * FROM ZonaRiego
WHERE id_zona > 145;
GO
```

100 % — No issues found

Results | Messages

	id_zona	nombre	descripcion	superficie
1	150	Zona_515	Zona agregada mediante SP 5	500

	id_zona	nombre	descripcion	superficie
1	146	Zona_511	Zona agregada mediante SP 1	410,5
2	147	Zona_512	Zona agregada mediante SP 2	355
3	148	Zona_513	Zona agregada mediante SP 3	430,3
4	149	Zona_514	Zona agregada mediante SP 4	298,7
5	151	Zona_516	Zona agregada mediante SP 6	475,2
6	152	Zona_517	Zona agregada mediante SP 7	382,9
7	153	Zona_518	Zona agregada mediante SP 8	445,4
8	154	Zona_519	Zona agregada mediante SP 9	410,8
9	155	Zona_520	Zona agregada mediante SP 10	390,1

Tarea N°4

Desarrollar al menos tres funciones almacenadas.

Función N°1 - Cálculo del promedio de humedad de una zona:

```
CREATE FUNCTION fn_promedio_humedad_zona (@id_zona INT)
RETURNS FLOAT
AS
BEGIN
    DECLARE @promedio FLOAT;
    SELECT @promedio = AVG(humedad)
    FROM LecturasSensor L
    JOIN Sensores S ON L.id_sensor_lectura = S.id_sensor
    WHERE S.zona_asignada = @id_zona;
    RETURN @promedio;
END;
GO
```

Función N°2 - Obtención de la cantidad de sensores activos:

```
CREATE FUNCTION fn_cant_sensores_activos ()
RETURNS INT
AS
BEGIN
    DECLARE @cantidad INT;
          SELECT  @cantidad  =  COUNT(*)  FROM  Sensores  WHERE
estado_sensor = 1;
    RETURN @cantidad;
END;
GO
```

Función N°3 - Cálculo de la eficiencia de riego de una zona:

```
CREATE FUNCTION fn_eficiencia_riego_zona (@id_zona INT)
RETURNS FLOAT
AS
BEGIN
    DECLARE @total INT, @exitosos INT;
        SELECT @total = COUNT(*) FROM RegistroRiego WHERE id_zona_registro = @id_zona;
        SELECT @exitosos = COUNT(*) FROM RegistroRiego WHERE id_zona_registro = @id_zona AND resultado = 1;
     RETURN CASE WHEN @total = 0 THEN 0 ELSE CAST(@exitosos AS FLOAT)/@total * 100 END;
END;
GO
```

Tarea N°5

Comparar la eficiencia de las operaciones directas versus el uso de procedimientos y funciones.

Para esta tarea, usaremos los comandos `SET STATISTICS TIME ON` y `SET STATISTICS IO ON`, que nos permite ver el tiempo necesario para analizar, compilar y ejecutar cada declaración de la consulta (`STATISTICS TIME`) y ver tiempo de actividad en disco generado por la consulta (`STATISTICS IO`).

```
SET STATISTICS TIME ON;         -- Este comando muestra el tiempo necesario para analizar, compilar y ejecutar cada declaracion de la consulta
SET STATISTICS IO ON;           -- Este comando muestra el tiempo de actividad en disco generado por la consulta
GO

-- Para testear vamos a usar los comandos de la Tarea 2
-------------------------
-- INSERCION DIRECTA --
-------------------------
INSERT INTO ZonaRiego (nombre, descripcion, superficie) VALUES
('Zona_501', 'Área experimental 1', 320.5),
('Zona_502', 'Área experimental 2', 410.8),
('Zona_503', 'Área experimental 3', 275.0),
('Zona_504', 'Área experimental 4', 380.2),
('Zona_505', 'Área experimental 5', 295.6),
('Zona_506', 'Área experimental 6', 450.0),
('Zona_507', 'Área experimental 7', 510.3),
('Zona_508', 'Área experimental 8', 390.7),
('Zona_509', 'Área experimental 9', 465.9),
('Zona_510', 'Área experimental 10', 340.4);
GO

-------------------------------------
-- INSERCION MEDIANTE PROCEDIMIENTO --
-------------------------------------
EXEC dbo.sp_insertar_zona_riego 'Zona_511', 'Zona agregada mediante SP 1', 410.5;
EXEC dbo.sp_insertar_zona_riego 'Zona_512', 'Zona agregada mediante SP 2', 355.0;
EXEC dbo.sp_insertar_zona_riego 'Zona_513', 'Zona agregada mediante SP 3', 430.3;
EXEC dbo.sp_insertar_zona_riego 'Zona_514', 'Zona agregada mediante SP 4', 298.7;
EXEC dbo.sp_insertar_zona_riego 'Zona_515', 'Zona agregada mediante SP 5', 500.0;
EXEC dbo.sp_insertar_zona_riego 'Zona_516', 'Zona agregada mediante SP 6', 475.2;
EXEC dbo.sp_insertar_zona_riego 'Zona_517', 'Zona agregada mediante SP 7', 382.9;
EXEC dbo.sp_insertar_zona_riego 'Zona_518', 'Zona agregada mediante SP 8', 445.4;
EXEC dbo.sp_insertar_zona_riego 'Zona_519', 'Zona agregada mediante SP 9', 410.8;
EXEC dbo.sp_insertar_zona_riego 'Zona_520', 'Zona agregada mediante SP 10', 390.1;
GO
```

```
(10 rows affected)
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'ZonaRiego'. Scan count 0, logical reads 2, physical reads (

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

(1 row affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 1 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'ZonaRiego'. Scan count 0, logical reads 2, physical reads (

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

(1 row affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'ZonaRiego'. Scan count 0, logical reads 2, physical reads (

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

(1 row affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'ZonaRiego'. Scan count 0, logical reads 2, physical reads (

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

(1 row affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'ZonaRiego'. Scan count 0, logical reads 2, physical reads (
```

En el caso de la inserción directa vs. la inserción con procedimientos, vemos que la directa es más rápida, pero al ser directa no posee control de errores ni validaciones,

en cambio el procedimiento almacenado es más lento, pero se le pueden agregar verificaciones para mejorar la seguridad, además de ser reutilizable.

```
-- Y ahora para las funciones, usando la de humedad promedio
SELECT dbo.fn_promedio_humedad_zona(1) AS Promedio_Humedad;

SELECT
    AVG(humedad) AS Promedio_Humedad
FROM LecturasSensor AS L
INNER JOIN Sensores AS S ON L.id_sensor_lectura = S.id_sensor
WHERE S.zona_asignada = 1;
GO
```

```
(1 row affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads (
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads
Table 'LecturasSensor'. Scan count 1, logical reads 27, physical reads 0, page server reads 0, read-ahead
Table 'Sensores'. Scan count 1, logical reads 2, physical reads 0, page server reads 0, read-ahead reads (

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 2 ms.

(1 row affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads (
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads
Table 'LecturasSensor'. Scan count 1, logical reads 27, physical reads 0, page server reads 0, read-ahead
Table 'Sensores'. Scan count 1, logical reads 2, physical reads 0, page server reads 0, read-ahead reads (

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 1 ms.
```

Y en el caso de la consulta directa vs. la función almacenada, vemos que la consulta directa es más rápida a comparación de la función, pero la ventaja de la función radica en poder usarla de manera repetitiva y sencilla, además de poder llamarla dentro de otras consultas, mejorando la legibilidad.