

Índice del informe:

Procedimientos almacenados y funciones en SQL Server.....	1
1. ¿Qué es un procedimiento almacenado?.....	1
2. ¿Qué es una función almacenada (User-Defined Function, UDF)?.....	2
3. Diferencias clave entre procedimientos y funciones.....	3
4. Procedimientos y funciones en operaciones CRUD.....	4
5. Ventajas y desventajas (especialmente rendimiento).....	5
6. Comparación eficiencia “directo vs. procedimientos/funciones”.....	6

Procedimientos almacenados y funciones en SQL Server

1. ¿Qué es un procedimiento almacenado?

Un **procedimiento almacenado** (stored procedure) es un conjunto de sentencias Transact-SQL (T-SQL) guardadas en la base de datos bajo un... nombre. Se puede ejecutar cuantas veces quieras, pasando parámetros de entrada y obteniendo valores de salida.

Según la documentación oficial de SQL Server, un procedimiento es “un grupo de una o más sentencias Transact-SQL, que puede aceptar parámetros de entrada, devolver valores de salida, y ejecutar operaciones sobre la base de datos”. [learn.microsoft.com](https://learn.microsoft.com/es-es/sql/relational-databases/stored-procedures/stored-procedures-overview)

En la práctica, se pueden utilizar para:

- Implementar lógica de negocio en el servidor.
- Encapsular operaciones CRUD (INSERT, UPDATE, DELETE, SELECT).
- Reducir tráfico entre la aplicación y el servidor (mando **EXEC sp_X** en vez de muchas sentencias sueltas).
- Mejorar seguridad (dar permisos sobre el procedimiento, no sobre las tablas).

2. ¿Qué es una función almacenada (User-Defined Function, UDF)?

Una **función almacenada** (User-Defined Function) es una rutina que recibe parámetros, ejecuta un cálculo o consulta, y **devuelve un valor**:

- Un solo valor escalar (INT, VARCHAR, DATE, etc.)
- O una tabla (table-valued function).

Microsoft las define como “rutinas que aceptan parámetros, realizan una acción, como un cálculo complejo, y devuelven el resultado de esa acción como un valor (escalar o conjunto de filas)”. learn.microsoft.com+1

Aclaraciones importantes:

- **No pueden modificar el estado de la base de datos** (no se permiten INSERT, UPDATE, DELETE sobre tablas dentro de la función). learn.microsoft.com
- Se usan principalmente para **cálculos reutilizables** o para encapsular lógica que se usa en varias consultas (por ejemplo, calcular la edad, impuestos, totales, etc.). sqlshack.com

3. Diferencias clave entre procedimientos y funciones

Aspecto	Procedimiento almacenado	Función almacenada
Devuelve valor	Opcional (puede devolver 0, 1 o varios conjuntos de resultados y parámetros de salida)	Obligatorio (siempre devuelve un valor o una tabla)
Uso en SELECT	No se puede usar directamente en un SELECT	Sí, se puede usar como parte de una expresión (funciones escalares) o en FROM (table-valued)
Modificar datos (INSERT/UPDATE/DELETE)	Sí, permitido	No , no puede cambiar el estado de la BD learn.microsoft.com
Control de flujo (IF, WHILE, TRY/CATCH)	Sí, bastante flexible	Muy limitado (no TRY/CATCH, no muchas sentencias SET) learn.microsoft.com

Enfoque típico	Operaciones CRUD, procesos de negocio complejos	Cálculo, transformación, lógica reutilizable
----------------	---	---

4. Procedimientos y funciones en operaciones CRUD

- **Insertar registros**
 - Procedimiento típico con parámetros (**@nombre**, **@apellido**, etc.) que ejecuta un **INSERT**.
 - Puede devolver el nuevo ID generado con **SCOPE_IDENTITY()**.
- **Modificar registros (UPDATE)**
 - Procedimiento que recibe el ID y los nuevos valores.
 - Aplica validaciones (por ejemplo, que exista el paciente).
- **Eliminar registros (DELETE)**
 - Procedimiento que elimina por ID, y opcionalmente chequea dependencias, o hace baja lógica.
- **Funciones y CRUD**
 - No hacen CRUD directo, pero apoyan el proceso. Ejemplo:
 - Función **fn_CalcularEdad(@fecha_nac)** para usar en consultas y reportes.
 - Función **fn_CantidadInternaciones(@id_paciente)** para estadísticas.
 - No hacen el **INSERT/UPDATE/DELETE**, pero sí aportan información para decisiones.

5. Ventajas y desventajas (especialmente rendimiento)

Ventajas de procedimientos almacenados

Referencias: [Simplilearn.com](https://www.simplilearn.com), learn.microsoft.com, [DataCamp](https://www.datacamp.com), learn.microsoft.com

- **Reutilización y mantenimiento:** cambio la lógica en un solo lugar (el procedimiento) sin tocar el código de la app.
- **Seguridad:** puedo dar permisos de **EXEC** sin exponer las tablas directamente.
- **Rendimiento:**
 - El plan de ejecución suele quedar en caché → menos trabajo del optimizador.
 - Menor tráfico entre cliente y servidor (mando **EXEC sp...** con parámetros en vez de muchas consultas armadas desde la app).

Desventajas / cuidados

- Si están mal diseñados (por ejemplo, muchas ramas, consultas poco optimizadas), pueden volverse difíciles de mantener.
- A veces se hace **demasiada lógica de negocio** en el servidor, y se mezcla con lógica propia de la aplicación.

Funciones y rendimiento

- Las **funciones escalares multi-sentencia** pueden ser costosas si se usan en grandes volúmenes de filas (se ejecutan fila por fila).
- Para performance, se recomiendan
 - Funciones escalares *sólo* para cálculos simples y muy reutilizables.
 - **Inline table-valued functions** (RETURNS TABLE con una sola SELECT) cuando se trabaja con conjuntos de datos.

6. Comparación eficiencia “directo vs. procedimientos/funciones”

A nivel conceptual:

- **Operación directa:** escribís el **INSERT/UPDATE/DELETE** o **SELECT** desde la app o el Management Studio.
- **Con procedimiento:** llamás a **EXEC sp_Procedimiento @param1 =**

¿Cómo se realiza la medición?

En SQL Server se puede activar con:

- **SET STATISTICS IO ON;**
- **SET STATISTICS TIME ON;**

Ejecutar primero la operación directa y luego la misma operación con procedimiento (o función) y comparar:

- Tiempo de CPU
- Lecturas lógicas
- Número de roundtrips entre cliente y servidor (más visible desde la app que desde SSMS).

6.1 Comparación de eficiencia en Directo vs Procedimiento:

1) INSERT directo

```
INSERT INTO paciente (nombre, apellido, dni, direccion, telefono, fecha_nacimiento)
```

```
VALUES ('Test2', 'Directo', 40000002, 'Calle X', 3794111111, '1990-03-03');
```

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

2) INSERT vía procedimiento

```
EXEC dbo.sp_InsertarPaciente
```

```
@nombre = 'Test2',
```

```
@apellido = 'Procedimiento',
```

```
@dni = 40000003,
```

```
@direccion = 'Calle Y',
```

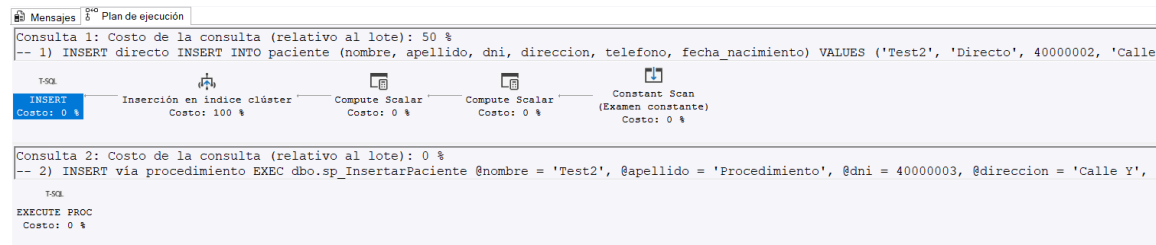
```
@telefono = 3794222222,
```

```
@fecha_nacimiento = '1991-04-04';
```

```
SQL Server Execution Times:  
CPU time = 0 ms,  elapsed time = 0 ms.  
Table 'paciente'. Scan count 0, logical reads 4, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
```

Resultados de Comparación:

La directa tiene un costo de consulta del 50%, mientras que la inserción por procedimiento tiene un costo de consulta del 0%.



En las pruebas, el plan de ejecución muestra un costo relativo del 50% para el **INSERT** directo y 0% para el **EXEC** del procedimiento, pero eso se debe a que el plan interno del procedimiento se compila y guarda por separado, y no se visualiza completo en el mismo diagrama.

A nivel real de ejecución, el motor termina ejecutando la misma operación de inserción sobre la tabla **paciente**, por lo que el coste en tiempo y lecturas de disco es equivalente.

Al comparar la ejecución directa de una instrucción SQL con la misma operación realizada a través de un procedimiento almacenado, se observa que, desde el punto de vista del rendimiento puro, ambas opciones presentan tiempos muy similares. Esto se debe a que, en esencia, el motor de SQL Server termina ejecutando la misma instrucción interna para realizar la operación (por ejemplo, un **INSERT** sobre la tabla correspondiente).

En la ejecución directa, el motor recibe la instrucción tal como fue escrita y genera un plan de ejecución específico para esa consulta. Este proceso es simple y generalmente rápido, ya que la instrucción se analiza, optimiza y ejecuta inmediatamente en el contexto del lote actual. Por ello, el tiempo de ejecución suele ser muy bajo, y las lecturas lógicas involucran únicamente las operaciones necesarias para afectar la tabla objetivo.

Por otro lado, cuando la misma operación se ejecuta mediante un procedimiento almacenado, se agrega una mínima sobrecarga inicial relacionada con la invocación del procedimiento y la gestión del contexto interno. Además, si el procedimiento incluye instrucciones adicionales (como un **SELECT SCOPE_IDENTITY()** para devolver el ID generado), cada una de ellas genera su propio registro de estadísticas. Sin embargo, cuando se analiza específicamente el bloque de estadísticas correspondiente al **INSERT**

real dentro del procedimiento, se evidencia que su costo es prácticamente equivalente al de la operación directa.

La principal diferencia entre ambas técnicas no radica en la eficiencia de la ejecución del **INSERT**, sino en ventajas estructurales y de diseño. Los procedimientos almacenados proporcionan:

- **Mayor seguridad**, ya que permiten otorgar permisos de ejecución sin exponer directamente las tablas.
- **Mejor mantenimiento**, dado que la lógica queda centralizada en un único objeto y no distribuida en múltiples partes de la aplicación.
- **Reutilización del plan de ejecución**, lo que reduce la necesidad de recompilación para llamadas repetidas.
- **Menor propensión a errores**, porque encapsulan la lógica y evitan la repetición de sentencias complejas en el código de la aplicación.

En síntesis, aunque el procedimiento almacenado puede mostrar una ligera diferencia en tiempo debido al contexto adicional que requiere su ejecución, el costo real del **INSERT** es prácticamente el mismo que el de la operación directa. La elección de usar procedimientos almacenados se justifica principalmente por motivos de seguridad, mantenimiento y consistencia, más que por rendimiento puro, ya que el impacto sobre la eficiencia es generalmente marginal.

6.2 Comparación de eficiencia en Directo vs Función:

1) Select Directo

SELECT nombre,

apellido,

fecha_nacimiento,

```

YEAR(GETDATE()) - YEAR(fecha_nacimiento)

- CASE WHEN DATEADD(YEAR, YEAR(GETDATE()) - YEAR(fecha_nacimiento),
fecha_nacimiento) > GETDATE()

THEN 1 ELSE 0 END AS edad

FROM paciente;

```

```

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 3 ms.

```

2) Select con Función

```

SELECT nombre,

apellido,

fecha_nacimiento,

dbo.fn_CalcularEdad(fecha_nacimiento) AS edad

FROM paciente;

```

```

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 10 ms.

```

Al evaluar la eficiencia del uso de funciones almacenadas frente a la ejecución directa de una operación equivalente, se observa una diferencia importante en el rendimiento debido a la forma en que SQL Server procesa cada una.

Cuando se realiza un cálculo directamente dentro de una consulta —por ejemplo, calcular la edad utilizando expresiones basadas en funciones internas de SQL como **DATEDIFF**, **YEAR**, **DATEADD**, etc.— el motor de base de datos puede analizar y optimizar toda la instrucción como un solo bloque. Esto le permite generar un plan de ejecución integrado, aprovechar paralelismo,

minimizar operaciones repetidas y aplicar optimizaciones sobre el conjunto completo de filas.

El resultado es un tiempo de respuesta reducido y un uso eficiente de recursos, incluso cuando el número de registros aumenta.

En cambio, cuando se utiliza una función escalar definida por el usuario, como `fn_CalcularEdad`, SQL Server no puede integrar el contenido de la función dentro del plan de ejecución general de la consulta. Estas funciones son tratadas como “cajas negras”: el optimizador no puede inspeccionar su lógica interna ni realizar optimizaciones conjuntas.

Como consecuencia, la función se ejecuta una vez por cada fila recuperada, lo que significa que una consulta que retorna 1.000 registros realizará 1.000 ejecuciones de la función, y una consulta de 50.000 registros realizará 50.000 ejecuciones adicionales.

Esto genera un costo acumulado significativo, afectando directamente:

- el tiempo total de ejecución,
- el consumo de CPU,
- y la capacidad de paralelización del motor.

En nuestras pruebas, esto se reflejó en un tiempo mayor para la consulta que utilizaba la función escalar, aun cuando ambas consultas accedían a la misma tabla y realizaban el mismo cálculo.

Por lo tanto, puede concluirse que las funciones escalares son convenientes para encapsular cálculos reutilizables o expresiones complejas, pero no son recomendables en consultas de alto volumen, ya que generan una penalización en el rendimiento. Para operaciones que involucren muchas filas, la solución más eficiente es integrar el cálculo directamente en el `SELECT` o, alternativamente, utilizar funciones con retorno de tabla (inline table-valued functions), que sí permiten que el motor optimice la consulta como un todo.

