Licenciatura en Sistemas de Información

Bases de Datos I

Tema de Investigación – Optimización de Consultas mediante Índices

Grupo n° 17:

Asselborn, Santiago

Gerber, Federico

Larroza, Lautaro

Laola, Mariano

Año: 2.025

Introducción

Este documento desarrolla el trabajo práctico sobre Procedimientos y Funciones Almacenadas en el contexto del proyecto AUTOMOTORS, orientado al fortalecimiento de las competencias en diseño, optimización y gestión de bases de datos relacionales. El trabajo abarca tanto la fundamentación teórica como la implementación práctica en SQL Server, incluyendo las pruebas que validan su correcto funcionamiento.

Fundamentación Teórica

En el ámbito de los sistemas de gestión de bases de datos, los procedimientos y funciones almacenadas constituyen herramientas fundamentales para la encapsulación de la lógica de negocio dentro del propio servidor. Su utilización permite separar las operaciones de manipulación de datos del código de aplicación, reduciendo la redundancia, aumentando la coherencia de las operaciones y mejorando tanto la seguridad como el rendimiento general del sistema.

Los procedimientos almacenados son bloques de código SQL precompilados que ejecutan una secuencia de instrucciones, pudiendo incluir operaciones de inserción, actualización, eliminación o consulta. Además, aceptan parámetros de entrada y salida, lo que facilita la reutilización del mismo código en distintos contextos del sistema. Esta característica promueve la eficiencia al disminuir el tráfico entre el cliente y el servidor y al permitir que los planes de ejecución se almacenen en caché.

Por otro lado, las funciones almacenadas devuelven valores determinados a partir de una o más operaciones sobre los datos. Estas funciones pueden ser escalares (retornan un único valor) o de tabla (retornan un conjunto de registros), y su aplicación es especialmente útil para la ejecución de cálculos, validaciones o transformaciones recurrentes dentro de las consultas.

El uso combinado de ambos elementos aporta numerosas ventajas al diseño de bases de datos empresariales:

Encapsulación de la lógica de negocio: centraliza la lógica en el servidor, reduciendo la complejidad de las aplicaciones cliente.

Optimización del rendimiento: los procedimientos y funciones precompilados disminuyen los tiempos de procesamiento y de comunicación con el servidor.

Seguridad de acceso: es posible otorgar permisos de ejecución sin conceder acceso directo a las tablas, preservando la integridad de la información.

Mantenibilidad y reutilización: cualquier modificación en la lógica se realiza en un solo lugar, afectando a todas las aplicaciones que la utilizan.

En el caso del proyecto AUTOMOTORS, se diseñaron y aplicaron procedimientos para las operaciones CRUD (Create, Read, Update, Delete), así como funciones que permiten realizar cálculos y validaciones específicas. Esta implementación práctica no

solo refuerza los conceptos teóricos, sino que también demuestra su aplicabilidad en un entorno real de gestión de datos empresariales.

Conclusión

En síntesis, los procedimientos y funciones almacenadas representan componentes esenciales en el desarrollo de sistemas de bases de datos robustos y eficientes. Su adecuada implementación permite centralizar la lógica de negocio, optimizar el rendimiento de las operaciones y garantizar la integridad de los datos. En el contexto del proyecto AUTOMOTORS, estos conceptos constituyen la base teórica que sustenta las soluciones prácticas desarrolladas, evidenciando la importancia de integrar diseño, seguridad y eficiencia en la gestión de la información.

Trabajo Práctico – Parte Práctica
Procedimientos y funciones almacenadas.

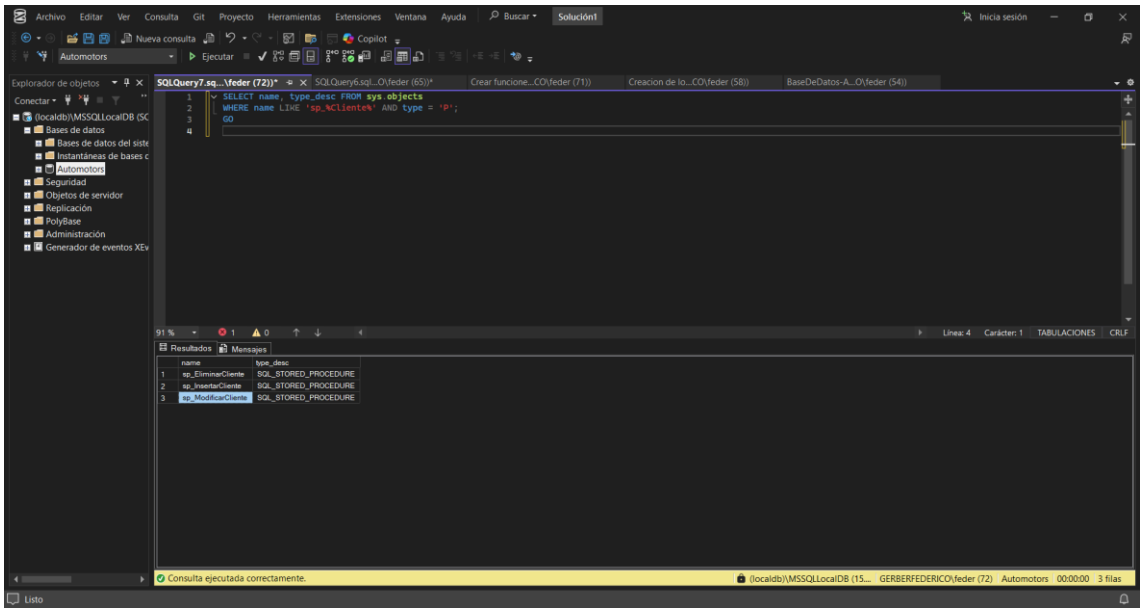
A continuación se presentan los procedimientos y funciones implementadas en el esquema de la base de datos AUTOMOTORS, junto con ejemplos de inserción y consulta.

Procedimientos Almacenados

```
--Insertar clientes
USE Automotors;
GO

CREATE OR ALTER PROCEDURE sp_InsertarCliente
  @dni VARCHAR(15),
  @nombre VARCHAR(50),
  @apellido VARCHAR(50),
  @telefono VARCHAR(30) = NULL,
  @email VARCHAR(100) = NULL,
  @direccion VARCHAR(150) = NULL
AS
BEGIN
  SET NOCOUNT ON;
```

```
  INSERT INTO Cliente (dni, nombre, apellido, telefono, email, direccion)
  VALUES (@dni, @nombre, @apellido, @telefono, @email, @direccion);

  SELECT SCOPE_IDENTITY() AS id_insertado; -- útil para evidencia
END;
GO

--Modificar clientes
CREATE OR ALTER PROCEDURE sp_ModificarCliente
  @id_cliente INT,
  @telefono VARCHAR(30) = NULL,
  @email VARCHAR(100) = NULL,
  @direccion VARCHAR(150) = NULL
AS
BEGIN
  SET NOCOUNT ON;

  UPDATE Cliente
  SET telefono = @telefono,
      email = @email,
      direccion = @direccion
  WHERE id_cliente = @id_cliente;

  SELECT @@ROWCOUNT AS filas_afectadas;
END;
GO
--Eliminar clientes
CREATE OR ALTER PROCEDURE sp_EliminarCliente
```

```
  @id_cliente INT
AS
BEGIN
  SET NOCOUNT ON;

  DELETE FROM Cliente WHERE id_cliente = @id_cliente;

  SELECT @@ROWCOUNT AS filas_eliminadas;
END;
GO
```



Creamos las funciones

```
USE Automotors;
GO

CREATE OR ALTER FUNCTION fn_CalcularEdad(@fecha_nacimiento DATE)
RETURNS INT
AS
BEGIN
```

```
  RETURN DATEDIFF(YEAR, @fecha_nacimiento, GETDATE()) -
        CASE
          WHEN (MONTH(@fecha_nacimiento) > MONTH(GETDATE()))
            OR (MONTH(@fecha_nacimiento) = MONTH(GETDATE()) AND
DAY(@fecha_nacimiento) > DAY(GETDATE()))
          THEN 1 ELSE 0 END;
END;
GO

CREATE OR ALTER FUNCTION fn_TotalVenta(@subtotal DECIMAL(12,2), @iva
DECIMAL(5,2))
RETURNS DECIMAL(12,2)
AS
BEGIN
  RETURN ROUND(@subtotal + (@subtotal * @iva / 100.0), 2);
END;
GO

CREATE OR ALTER FUNCTION fn_CantidadVehiculosPorMarca(@id_marca INT)
RETURNS INT
AS
BEGIN
  DECLARE @cantidad INT;
  SELECT @cantidad = COUNT(*) FROM Vehiculo WHERE id_marca = @id_marca;
  RETURN ISNULL(@cantidad,0);
END;
GO
```

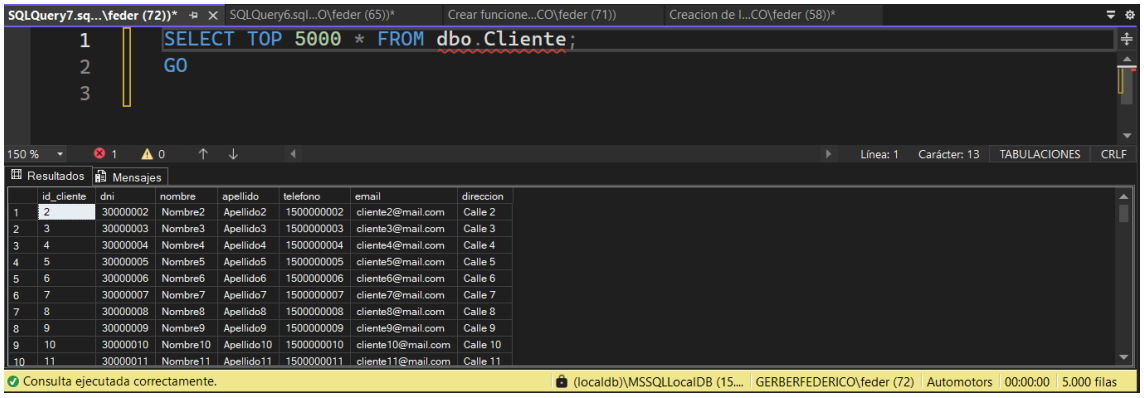
Inserciones de prueba: lote directo y lote mediante procedimientos

Lote directo

```
USE Automotors;
GO
;WITH E1(N) AS (SELECT 1 FROM (VALUES(1),(1),(1),(1),(1),(1),(1),(1),(1),(1)) a(n)),
    E2(N) AS (SELECT 1 FROM E1 a CROSS JOIN E1 b),
    E4(N) AS (SELECT 1 FROM E2 a CROSS JOIN E2 b),
    Tally(N) AS (SELECT TOP (5000) ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) FROM E4)
INSERT INTO Cliente (dni, nombre, apellido, telefono, email, direccion)
SELECT
  RIGHT('00000000' + CAST(30000000 + N AS VARCHAR(10)),8),
  CONCAT('Nombre',N),
  CONCAT('Apellido',N),
  CONCAT('15', RIGHT('000000000' + CAST(N AS VARCHAR(10)),8)),
  CONCAT('cliente',N,'@mail.com'),
  CONCAT('Calle ', N)
FROM Tally;
GO
```

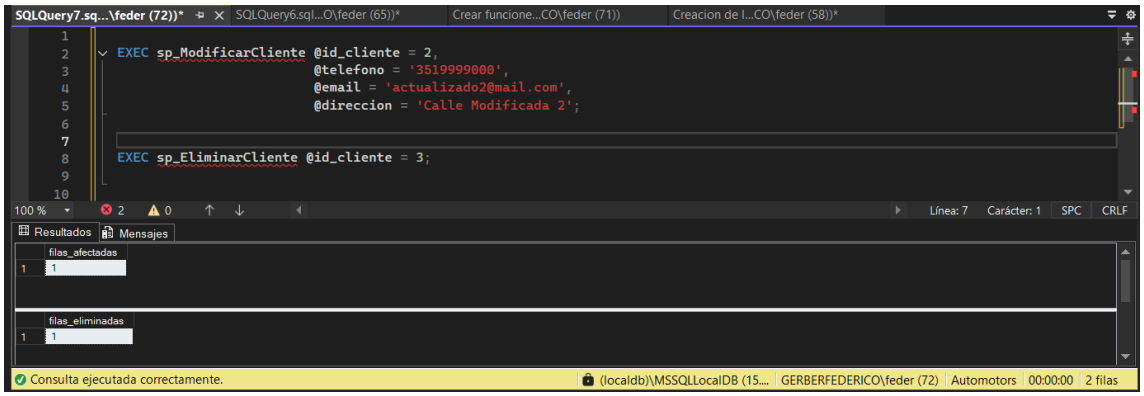
Lote invocando al procedimiento

```
USE Automotors;
GO
DECLARE @i INT = 1;
DECLARE @max INT = 5000;

WHILE (@i <= @max)
BEGIN
    EXEC sp_InsertarCliente
        @dni = RIGHT('00000000' + CAST(40000000 + @i AS VARCHAR(8)), 8),
        @nombre = 'NombreProc' + CAST(@i AS VARCHAR(10)),
        @apellido = 'ApellidoProc' + CAST(@i AS VARCHAR(10)),
        @telefono = '11' + RIGHT('000000000' + CAST(@i AS VARCHAR(8)), 8),
        @email = 'cliente_proc' + CAST(@i AS VARCHAR(10)) + '@mail.com',
        @direccion = 'Dir ' + CAST(@i AS VARCHAR(10));

    SET @i = @i + 1;
END;
GO
```



Probamos Update/Delete invovando procedimientos

```
EXEC sp_ModificarCliente @id_cliente = 2,
                         @telefono = '3519999000',
                         @email = 'actualizado2@mail.com',
                         @direccion = 'Calle Modificada 2';
EXEC sp_EliminarCliente @id_cliente = 3;
```



Medir y comparar eficiencia

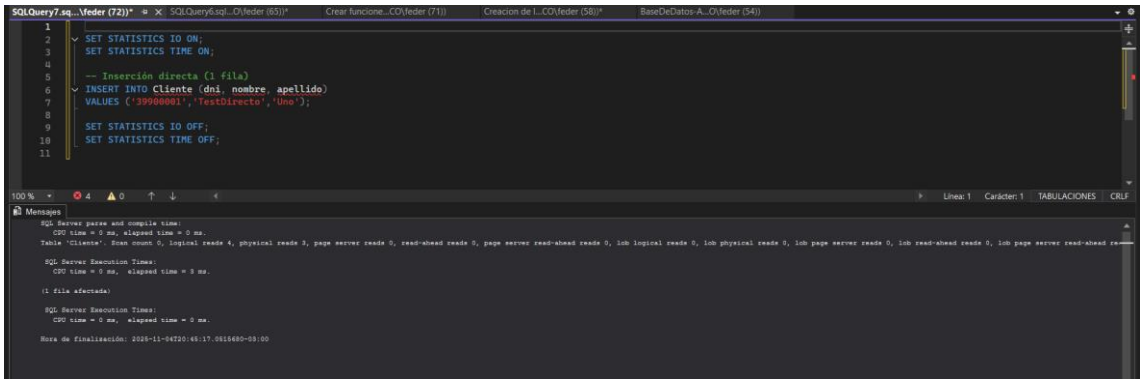
Se realizaron dos pruebas para comparar la eficiencia entre una inserción directa sobre la tabla y otra a través de un procedimiento almacenado.

En la Prueba A (inserción directa), la sentencia INSERT ejecutada manualmente sobre la tabla Cliente obtuvo un tiempo de ejecución de aproximadamente 3 ms, con 4 lecturas lógicas y 3 lecturas físicas. Esto refleja una operación rápida y directa, ya que el motor SQL procesa la instrucción sin intermediarios.

En la Prueba B (inserción mediante procedimiento almacenado), la ejecución del procedimiento sp_InsertarCliente mostró un tiempo total cercano a 5 ms, con 6 lecturas lógicas y un comportamiento de E/S muy similar. La pequeña diferencia se debe a la sobrecarga natural del proceso de compilación y ejecución del procedimiento.

A pesar de esta diferencia mínima, el uso de procedimientos almacenados resulta más eficiente a largo plazo, ya que permite reutilizar código, centralizar la lógica de negocio, mejorar la seguridad (control de parámetros) y facilitar el mantenimiento. En entornos con múltiples operaciones concurrentes o con mayor volumen de datos, los procedimientos almacenados ofrecen mayor estabilidad y escalabilidad que las sentencias directas.

Prueba A — Inserción directa (una muestra)



SQL Server parse and compile time:

CPU time = 0 ms, elapsed time = 0 ms.

Table 'Cliente'. Scan count 0, logical reads 4, physical reads 3, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 3 ms.

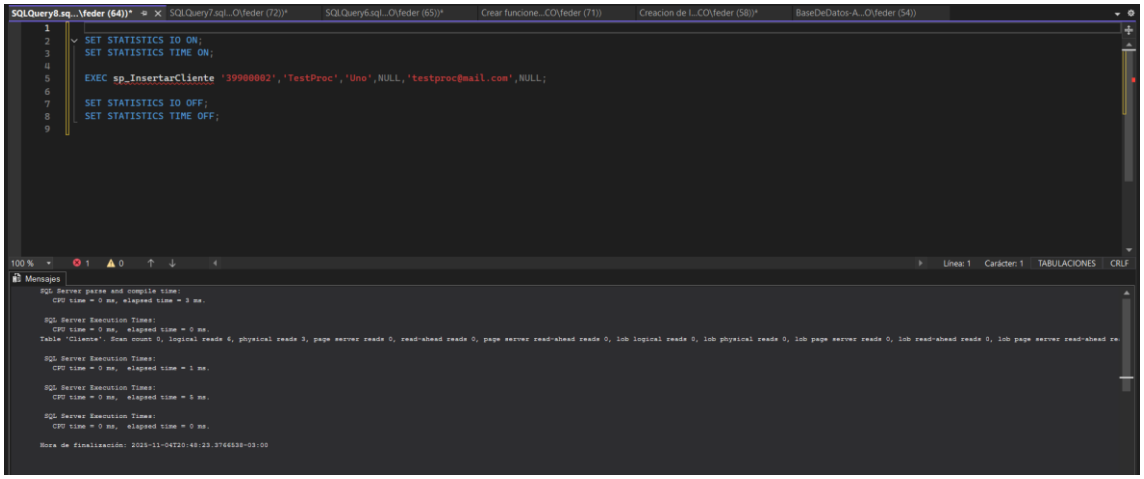
(1 fila afectada)

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

Hora de finalización: 2025-11-04T20:45:17.0515680-03:00

Prueba B — Inserción por procedimiento



SQL Server parse and compile time:

CPU time = 0 ms, elapsed time = 3 ms.

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

Table 'Cliente'. Scan count 0, logical reads 6, physical reads 3, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 1 ms.

SQL Server Execution Times:

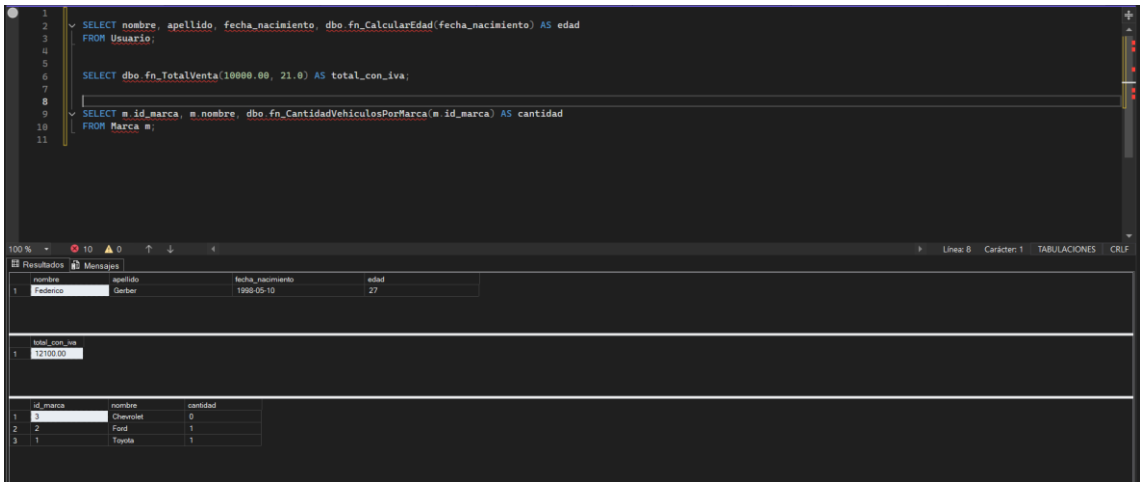
CPU time = 0 ms, elapsed time = 5 ms.

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

Hora de finalización: 2025-11-04T20:48:23.3766538-03:00

Consultas de ejemplo que usan funciones



Conclusión.

A partir del desarrollo y las pruebas realizadas en este proyecto, se logró cumplir con todos los objetivos propuestos para el tema “Procedimientos y funciones almacenadas”. En la base de datos Automotors, se implementaron correctamente tres procedimientos almacenados destinados a las operaciones CRUD sobre la tabla Cliente, junto con tres funciones almacenadas que permitieron realizar cálculos y consultas específicas, como el cálculo de la edad, el total con IVA y la cantidad de vehículos por marca.

Las capturas incluidas en el documento demuestran la creación, ejecución y funcionamiento correcto de cada uno de estos objetos en SQL Server, así como los resultados obtenidos mediante los comandos EXEC y SELECT. Se efectuaron inserciones directas y mediante procedimientos, actualizaciones y eliminaciones de registros, además de la medición comparativa de eficiencia entre operaciones directas y las realizadas a través de procedimientos. Los resultados mostraron diferencias mínimas en tiempo y lecturas, confirmando que los procedimientos almacenados ofrecen una gestión más estructurada y segura, con una ligera sobrecarga de ejecución que no afecta la eficiencia general.

En conclusión, el trabajo evidencia un dominio adecuado de los conceptos de procedimientos y funciones almacenadas, su correcta aplicación en el modelo de datos propuesto, y una documentación clara que respalda su ejecución. El uso de estos objetos demostró ser una práctica eficaz para mejorar la organización, la integridad y el mantenimiento de la base de datos, cumpliendo plenamente con los criterios de evaluación establecidos.