PROCEDIMIENTOS Y FUNCIONES

Introducción

Los **procedimientos almacenados** y las **funciones** son dos componentes fundamentales en la programación dentro de bases de datos relacionales. Estos permiten encapsular lógica compleja que puede reutilizarse, manteniendo la integridad de los datos y mejorando la eficiencia del desarrollo.

¿Por qué usar procedimientos almacenados o funciones?

- Los procedimientos almacenados son ideales para realizar operaciones complejas que involucran modificaciones de datos y control de transacciones. Pueden agrupar una serie de instrucciones SQL y se ejecutan en el servidor, lo cual disminuye el tráfico entre el cliente y el servidor, mejorando el rendimiento en ciertas operaciones masivas o automatizadas.
- Las funciones, por otro lado, se utilizan principalmente para realizar cálculos y devolver valores. A diferencia de los procedimientos almacenados, las funciones se pueden integrar directamente en consultas `SELECT`, lo cual las hace útiles para operaciones de lectura y procesamiento de datos que requieren un resultado específico.

¿Dónde se almacenan y cómo se otorgan permisos?

- Tanto los procedimientos almacenados como las funciones se almacenan en el servidor de la base de datos, en el catálogo de objetos del esquema al que pertenecen. Esto permite que los usuarios autorizados los ejecuten sin necesidad de que se envíe toda la lógica desde la aplicación cliente cada vez.
- Los permisos se otorgan mediante comandos como `GRANT`. Por ejemplo, un administrador de bases de datos podría usar `GRANT EXECUTE ON PROCEDURE agregar_cliente TO usuario;` para otorgar permiso de ejecución de un procedimiento almacenado específico a un usuario. Esto permite un control detallado sobre quién puede ejecutar ciertos procesos en la base de datos, ayudando a mantener la seguridad.

Cuestiones conceptuales de interés

- En términos generales, los procedimientos almacenados se utilizan para realizar **acciones** (como insertar, actualizar o eliminar registros), mientras que las funciones se utilizan para **calcular y devolver valores**. Es importante recordar que las funciones no pueden controlar transacciones, lo que las hace menos adecuadas para operaciones que requieren cambios en varias tablas.
- El uso adecuado de procedimientos y funciones no solo facilita la reutilización de código, sino que también contribuye a la consistencia y el mantenimiento de la lógica de negocio en un solo lugar: la base de datos. Esto resulta especialmente valioso en sistemas donde la integridad de los datos y la eficiencia de las operaciones son críticas.

Disponibilidad de estas características

- En **PostgreSQL**, los procedimientos almacenados (`CREATE PROCEDURE`) se introdujeron a partir de la **versión 11** (2018). Antes de esto, PostgreSQL solo contaba con funciones para realizar lógica almacenada. Las funciones (`CREATE FUNCTION`) han estado presentes desde las primeras versiones de PostgreSQL y han evolucionado para incluir funcionalidades avanzadas como funciones de retorno de tabla y soporte para múltiples lenguajes de script.
- En **SQL Server**, los procedimientos almacenados y funciones han existido desde las primeras versiones del sistema. **SQL Server 2000** (lanzado en 2000) fue la versión en la que muchas de las funcionalidades modernas de funciones definidas por el usuario se consolidaron, mientras que los procedimientos almacenados ya eran una característica importante desde antes. Los procedimientos permiten control de transacciones y optimización de procesos complejos, mientras que las funciones permiten cálculos reutilizables. Sybase, motor de base de datos predecesor de SQL Server, ya contaba con procedimientos almacenados.

Procedimientos almacenados

PostgreSQL

En PostgreSQL, los *procedimientos almacenados* se crean con la instrucción "CREATE PROCEDURE". Estos no devuelven un valor directamente, pero permiten realizar operaciones como inserciones, actualizaciones y eliminaciones de datos.

Ejemplo: Procedimiento para agregar un cliente

```
CREATE OR REPLACE PROCEDURE agregar_cliente(
    nombre_cliente VARCHAR,
    direccion_cliente VARCHAR,
    email_cliente VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO persona.persona (tipo, codigo, fecha_alta, email, domicilio)
    VALUES ('FISICA', nextval('persona.persona_sequence'), NOW(), email_cliente,
direccion_cliente);

    -- Obtener el id generado para la persona y crear el cliente
    INSERT INTO persona.cliente (id_persona, codigo, fecha_alta)
    VALUES (currval('persona.persona_sequence'),
nextval('persona.persona_sequence'), NOW());
END;
$$;
```

Para ejecutar el procedimiento:

```
CALL agregar_cliente('Juan Perez', 'Calle Falsa 123',
'juan@example.com');
```

SQL Server

En SQL Server, los **procedimientos almacenados** se crean con la instrucción `CREATE PROCEDURE`. Se utilizan de manera similar a PostgreSQL.

Ejemplo: Procedimiento para agregar un cliente**:

```
CREATE PROCEDURE agregar_cliente
    @nombre_cliente NVARCHAR(50),
    @direccion_cliente NVARCHAR(255),
    @email_cliente NVARCHAR(255)
AS
BEGIN
    INSERT INTO persona (tipo, codigo, fecha_alta, email, domicilio)
    VALUES ('FISICA', NEXT VALUE FOR persona_sequence, GETDATE(),
@email_cliente, @direccion_cliente);
    DECLARE @persona_id BIGINT = SCOPE_IDENTITY();
    INSERT INTO cliente (id_persona, codigo, fecha_alta)
    VALUES (@persona_id, NEXT VALUE FOR persona_sequence, GETDATE());
END;
```

Para ejecutar el procedimiento:

```
EXEC agregar_cliente 'Juan Perez', 'Calle PorAhi 678', 'juan@argentina.gov.ar';
```

Funciones

Las **funciones** devuelven un valor y se pueden usar en una consulta `SELECT`.

PostgreSQL
En PostgreSQL, las funciones pueden devolver valores escalares o compuestos.

Ejemplo: Función para calcular el total de ventas de un cliente:

```
CREATE OR REPLACE FUNCTION total_ventas_cliente(id_cliente BIGINT)
RETURNS NUMERIC
LANGUAGE plpgsql
AS $$
DECLARE
    total NUMERIC;
BEGIN
    SELECT SUM(f.total) INTO total
    FROM venta.factura f
    WHERE f.id_cliente = id_cliente;

    RETURN COALESCE(total, 0);
END;
$$;
```

Para utilizar la función:

```
SELECT total_ventas_cliente(1);
```

SQL Server
En SQL Server, las funciones también pueden devolver un valor escalar o compuesto.

```
CREATE FUNCTION total_ventas_cliente(@id_cliente BIGINT)
RETURNS DECIMAL(10, 2)
AS
BEGIN
    DECLARE @total DECIMAL(10, 2);

    SELECT @total = SUM(f.total)
    FROM venta.factura f
    WHERE f.id_cliente = @id_cliente;

    RETURN ISNULL(@total, 0);
END;
```

Para utilizar la función:

```
SELECT dbo.total_ventas_cliente(1);
```

Procedimiento Complejo con Transacciones

PostgreSQL

Imaginemos que queremos crear un procedimiento que registre una venta y sus detalles en las tablas `venta.factura` y `venta.factura_detalle`.

```
CREATE OR REPLACE PROCEDURE registrar_venta(
    id_cliente BIGINT,
    id_empleado BIGINT,
    id_producto BIGINT,
    cantidad NUMERIC,
    precio_unitario NUMERIC
)
LANGUAGE plpgsql
AS $$
DECLARE
    id_factura BIGINT;
BEGIN
    -- Iniciar la transacción
    BEGIN TRANSACTION;

    -- Insertar la factura
    INSERT INTO venta.factura (id_cliente, id_empleado, fecha, descuento, total)
    VALUES (id_cliente, id_empleado, CURRENT_DATE, 0, cantidad *
precio_unitario)
    RETURNING id INTO id_factura;

    -- Insertar el detalle de la factura
    INSERT INTO venta.factura_detalle (id_factura, item, id_producto, cantidad,
precio_unitario)
    VALUES (id_factura, 1, id_producto, cantidad, precio_unitario);

    EXCEPTION
        -- En caso de error, hacer rollback
        WHEN OTHERS THEN
            RAISE NOTICE 'Error al registrar la venta';
            ROLLBACK;
END;
$$;
```

SQL Server

El siguiente ejemplo hace lo mismo en SQL Server:

```
CREATE PROCEDURE registrar_venta
    @id_cliente BIGINT,
    @id_empleado BIGINT,
    @id_producto BIGINT,
    @cantidad DECIMAL(10, 2),
    @precio_unitario DECIMAL(10, 2)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        -- Insertar la factura
        DECLARE @id_factura BIGINT;
        INSERT INTO venta.factura (id_cliente, id_empleado, fecha, descuento,
total)
        VALUES (@id_cliente, @id_empleado, GETDATE(), 0, @cantidad *
@precio_unitario);

        SELECT @id_factura = IDENT_CURRENT('venta.factura');

        -- Insertar el detalle de la factura
        INSERT INTO venta.factura_detalle (id_factura, item, id_producto,
cantidad, precio_unitario)
        VALUES (@id_factura, 1, @id_producto, @cantidad, @precio_unitario);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        PRINT 'Error al registrar la venta';
    END CATCH;
END;
```

Comparación entre Procedimientos y Funciones

Característica	Procedimientos	Funciones
Devuelven un valor	No	Sí
Uso en consultas `SELECT`	No	Sí
Ejecución	`CALL` en PostgreSQL, `EXEC` en SQL Server	Directamente en consultas `SELECT`
Uso principal	Acciones complejas (modificaciones de datos)	Cálculos o devoluciones de datos
Transacciones	Sí	No

En resumen, los **procedimientos almacenados** se utilizan principalmente para realizar acciones que modifican el estado de la base de datos, incluyendo el uso de transacciones, mientras que las **funciones** son ideales para cálculos y lógica que necesitan ser reutilizadas en consultas. El uso adecuado de cada uno depende de la naturaleza de la operación: si se necesita modificar datos, lo mejor es un procedimiento almacenado; si solo se necesita devolver un valor sin efectos secundarios, una función es la opción ideal.

En el próximo tema, exploraremos el uso de **triggers** o disparadores, una herramienta poderosa para automatizar acciones en respuesta a eventos dentro de la base de datos.