TRIGGERS

Introducción

Los **triggers** o **disparadores** son objetos de base de datos que se ejecutan automáticamente en respuesta a ciertos eventos en una tabla o vista. Estos eventos pueden ser inserciones (*INSERT*), actualizaciones (*UPDATE*) o borrados (*DELETE*) de filas. Permiten ejecutar la lógica definida cada vez que ocurre uno de estos eventos, proporcionando una capa adicional de integridad y automatización en el manejo de datos.

Concepto

Un **trigger** es una especie de "guardia" que está a la espera de un evento para activarse. Por ejemplo, cuando se inserta un nuevo registro en una tabla de ventas, un trigger podría generar un registro de auditoría para mantener un histórico de los cambios.

Cuándo usar triggers

Los triggers son útiles cuando se necesita garantizar la consistencia de los datos o ejecutar tareas automáticas que son difíciles de implementar desde la lógica de la aplicación. Algunos escenarios comunes donde usar triggers:

- Auditoría: registrar cambios realizados sobre los datos.
- Mantenimiento de Integridad: asegurar que las reglas de negocio se respeten, como el ajuste automático de inventarios al modificar ventas.
- Automatización: ejecutar tareas automáticas como actualizar totales o propagar cambios a otras tablas.

Los triggers se almacenan en la base de datos, dentro del esquema al que pertenecen. Son parte integral de la base de datos y se ejecutan directamente en el servidor, asegurando eficiencia y reduciendo la necesidad de enviar operaciones desde el cliente.

Para crear y modificar triggers, se requieren permisos de administrador o privilegios específicos sobre la base de datos. Sin embargo, para la ejecución de un trigger, los usuarios no necesitan permisos adicionales; el trigger se ejecuta automáticamente en respuesta a la operación desencadenante.

Dos aspectos a considerar:

- Independencia: los triggers se ejecutan de manera independiente a la aplicación cliente, lo cual asegura que las reglas de negocio se mantengan incluso si cambian las aplicaciones que interactúan con la base de datos.
- Impacto en el rendimiento: aunque son útiles, los triggers pueden tener un impacto negativo en el rendimiento si no se usan con cuidado, ya que agregan lógica adicional a las operaciones INSERT, UPDATE o DELETE.

Características Específicas de Implementación

PostgreSQL

Los triggers se crean con la instrucción "CREATE TRIGGER" y se pueden usar con funciones escritas en PL/pgSQL u otros lenguajes soportados. Los triggers en PostgreSQL pueden ser *BEFORE* o *AFTER*, dependiendo de si se necesita que se ejecuten antes o después de la operación.

Ejemplo:

```
CREATE TRIGGER nombre_trigger
AFTER INSERT OR UPDATE OR DELETE
ON producto.producto
FOR EACH ROW
EXECUTE FUNCTION funcion_trigger();
```

SQL Server

Los triggers se crean con "CREATE TRIGGER" y se utilizan principalmente para operaciones *AFTER* (después de la operación), aunque también existen los triggers *INSTEAD OF*, que reemplazan la operación desencadenante.

Ejemplo:

```
CREATE TRIGGER nombre_trigger
ON producto.producto
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    -- Lógica del trigger
END;
```

Principales diferencias:

- Momentos de ejecución: PostgreSQL permite "BEFORE" y "AFTER", mientras que SQL Server incluye "AFTER" e "INSTEAD OF".
- Función asociada: En PostgreSQL, los triggers se asocian a funciones, mientras que en SQL Server se define la lógica del trigger directamente.
- Almacenamiento temporal de datos modificados/nuevos/borrados.

Ejemplos de Triggers

1. Generar un Registro de Auditoría al Modificar/Insertar/Actualizar un Producto

PostgreSQL

```
CREATE OR REPLACE FUNCTION auditoria_producto() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO producto.auditoria_producto(id_producto, operacion, fecha)
    VALUES (NEW.id, TG_OP, NOW());
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auditoria_producto
AFTER INSERT OR UPDATE OR DELETE
ON producto.producto
FOR EACH ROW
EXECUTE FUNCTION auditoria_producto();
```

SQL Server

```
CREATE TRIGGER trigger_auditoria_producto
ON producto.producto
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    INSERT INTO producto.auditoria_producto(id_producto, operacion, fecha)
    SELECT
        CASE WHEN EXISTS(SELECT * FROM inserted) THEN inserted.id ELSE
deleted.id END,
        CASE WHEN EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT * FROM
deleted) THEN 'UPDATE'
             WHEN EXISTS(SELECT * FROM inserted) THEN 'INSERT'
             ELSE 'DELETE' END,
        GETDATE()
    FROM inserted
    FULL OUTER JOIN deleted ON inserted.id = deleted.id;
END;
```

2. Calcular el Total de Factura cada vez que se Inserta/Modifica/Borra en “factura_detalle”

PostgreSQL

```
CREATE OR REPLACE FUNCTION actualizar_total_factura() RETURNS TRIGGER AS $$
BEGIN
    UPDATE venta.factura
    SET total = (SELECT SUM(cantidad * precio_unitario) FROM
venta.factura_detalle WHERE id_factura = NEW.id_factura)
    WHERE id = NEW.id_factura;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_actualizar_total_factura
AFTER INSERT OR UPDATE OR DELETE
ON venta.factura_detalle
FOR EACH ROW
EXECUTE FUNCTION actualizar_total_factura();
```

SQL Server

```
CREATE TRIGGER trigger_actualizar_total_factura
ON venta.factura_detalle
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    UPDATE venta.factura
    SET total = (
        SELECT SUM(cantidad * precio_unitario)
        FROM venta.factura_detalle
        WHERE id_factura = inserted.id_factura
    )
    FROM inserted
    WHERE venta.factura.id = inserted.id_factura;
END;
```

Transacciones

Los triggers pueden tener un impacto significativo en el comportamiento de las transacciones. En PostgreSQL, cuando en un trigger se detecta una condición de falla, se puede utilizar **"**`RETURN NULL`**"** para evitar que se confirme la transacción (`COMMIT`), haciendo que toda la operación falle. Sin embargo, se debe tener cuidado al decidir si es conveniente o no ejecutar un `ROLLBACK` completo dentro de un trigger, ya que esto puede tener un impacto inesperado en el comportamiento de las aplicaciones que dependen de la base de datos.

En **SQL Server**, los triggers también se ejecutan dentro del contexto de la transacción que inició el evento desencadenante. Si ocurre un error dentro de un trigger, puede hacer que toda la transacción falle.

Es importante diseñar triggers que manejen adecuadamente los errores para evitar que las operaciones válidas sean revertidas innecesariamente. *En general, es mejor delegar el control de transacciones a la capa de aplicación cuando sea posible*.

Momento de ejecución y información de datos afectados

En *PostgreSQL* se utilizan los registros `NEW` y `OLD` para acceder a los valores antes y después de la operación que desencadena el trigger. Estos registros son útiles para comparar cambios o generar auditorías. En *SQL Server*, se utilizan las pseudo-tablas `inserted` y `deleted` para acceder a los registros afectados por la operación.

En *PostgreSQL*, a partir de la versión 14 se han introducido tablas temporales para almacenar estos valores, facilitando el acceso a múltiples filas afectadas por un mismo evento. Se las denomina "tablas de transición" y funcionan de modo similar a las tablas `inserted/deleted` de *SQL Server*

1. Ejemplo de trigger por cada fila: Este tipo de activación se utiliza cuando se desea manejar cada fila afectada de manera individual.

```
CREATE OR REPLACE FUNCTION auditoria_producto_fila()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO producto.auditoria_producto(id_producto, operacion, fecha)
        VALUES (NEW.id, 'INSERT', NOW());
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO producto.auditoria_producto(id_producto, operacion, fecha)
        VALUES (NEW.id, 'UPDATE', NOW());
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO producto.auditoria_producto(id_producto, operacion, fecha)
        VALUES (OLD.id, 'DELETE', NOW());
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trigger_auditoria_producto_fila
AFTER INSERT OR UPDATE OR DELETE
ON producto.producto
FOR EACH ROW
EXECUTE FUNCTION auditoria_producto_fila();
```

Los registros NEW y OLD se utilizan para acceder a los valores antes y después de la operación para cada fila afectada. La expresión “FOR EACH ROW” indica que el trigger se ejecuta una vez para cada fila afectada, permitiendo un manejo específico por cada registro.

2. Ejemplo de trigger para múltiples filas : Se usa cuando una operación afecta a múltiples filas, y se desea manejar todas las filas afectadas de una sola vez. En este caso, se utilizan las tablas de transición: “NEW TABLE” y “OLD TABLE” para acceder a todos los registros afectados.

```
CREATE OR REPLACE FUNCTION auditoria_producto_transicion()
RETURNS TRIGGER AS $$
BEGIN
    -- Insertar registros de auditoría para todas las nuevas filas
    IF TG_OP = 'INSERT' THEN
        INSERT INTO producto.auditoria_producto(id_producto, operacion, fecha)
        SELECT id, 'INSERT', NOW() FROM new_table;
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO producto.auditoria_producto(id_producto, operacion, fecha)
        SELECT id, 'UPDATE', NOW() FROM new_table;
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO producto.auditoria_producto(id_producto, operacion, fecha)
        SELECT id, 'DELETE', NOW() FROM old_table;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auditoria_producto_transicion
AFTER INSERT OR UPDATE OR DELETE ON producto.producto
REFERENCING NEW TABLE AS new_table OLD TABLE AS old_table
FOR EACH STATEMENT
EXECUTE FUNCTION auditoria_producto_transicion();
```

Las tablas de transición (“new_table”, “old_table”) almacenan todas las filas afectadas por una operación de inserción, actualización o eliminación. La condición “FOR EACH STATEMENT” indica que el trigger se ejecuta una vez por cada sentencia, y las tablas `new_table` y `old_table` contienen todas las filas afectadas.

El trigger “for each statement” es más adecuado para operaciones masivas que afectan a muchas filas, ya que permite trabajar con todas las filas de una sola vez, lo cual mejora el rendimiento.
El trigger “for each row” más fino, ejecutándose una vez por cada registro afectado, lo cual es útil cuando cada fila necesita una lógica específica.

Ambos métodos tienen sus propias ventajas dependiendo del caso de uso y la cantidad de registros que se espera afectar.

Conclusiones y Recomendaciones

Los triggers son una herramienta poderosa para garantizar la integridad de los datos y automatizar tareas repetitivas dentro de la base de datos. Sin embargo, deben usarse con cuidado, ya que pueden impactar el rendimiento y hacer que los cambios en los datos sean menos predecibles si no se documentan correctamente. Se recomienda utilizarlos para tareas críticas que deban ejecutarse siempre que ocurra un evento específico en la base de datos, como la auditoría y el cálculo automático de valores.

Se recomienda:

1. Evitar la recursividad involuntaria: Los triggers pueden desencadenar otros triggers, lo cual puede generar un ciclo infinito de ejecuciones. Es importante evitar este comportamiento utilizando condiciones claras dentro del trigger o deshabilitando temporalmente los triggers cuando se realicen ciertas operaciones masivas.

2. Mantener la simplicidad: La lógica dentro de un trigger debe ser lo más sencilla posible. Triggers complejos pueden ser difíciles de depurar y aumentar significativamente el tiempo de ejecución de las operaciones de la base de datos.

3. Documentar: debido a que los triggers se ejecutan automáticamente y de manera independiente, es importante documentar claramente qué hacen y cuándo se activan. Esto facilita la comprensión y el mantenimiento del sistema, especialmente cuando hay cambios en el equipo de desarrollo.

4. Limitar su uso: los triggers son ideales para garantizar que ciertas reglas de negocio siempre se cumplan. Sin embargo, no deberían usarse para lógica de negocio que podría implementarse de manera más eficiente desde la capa de aplicación.

5. Evitar el uso para cálculos complejos: Si un trigger necesita realizar cálculos complejos o consultar muchas tablas, es posible que esa lógica deba implementarse en un procedimiento almacenado en lugar de un trigger para mejorar la legibilidad y el rendimiento.