BD
Bases de Datos

Disparadores (triggers)

UD9 – Triggers

ÍNDICE

1. Introducción

Los disparadores (o también conocidos como “triggers”) son tipos especiales de procedimientos almacenados que se activan automáticamente cuando sucede una operación DML sobre una tabla y ejecutan un bloque de instrucciones como respuesta.

Ejemplos reales de uso de triggers en una tienda:

- Cuando un cliente realiza un pedido de algún producto (cuando se inserte en la tabla DETALLE_PEDIDO), automáticamente se disminuirá la cantidad en stock del producto en la tienda en la cantidad que el cliente haya comprado. Dentro del disparador habría una operación de UPDATE.

- Si se realizara una devolución de un producto, se aumentaría la cantidad en stock del producto devuelto automáticamente.

- Si un producto se quedara sin stock, automáticamente podría avisarse por correo de para su reposición.

SQL Server proporciona **tres tipos de disparadores**:

- **DML**: activación al ejecutar INSERT, DELETE o UPDATE sobre una tabla.

- **DDL**: activación al ejecutar CREATE, ALTER o DROP sobre una tabla.

- **LOGON**: activación al iniciar sesión en el servidor de SQL Server. Se utiliza para auditorías y controlar las sesiones en el servidor como medida de seguridad.

2. Triggers DML

Como hemos visto anteriormente, los triggers DML se ejecutan cuando el usuario ejecuta una instrucción INSERT, UPDATE o DELETE sobre una tabla. Los triggers DML utilizan dos “pseudotablas” llamadas **INSERTED** y **DELETED**. Ambas tablas son gestionadas directamente por el SGBD y solo existen mientras se ejecuta el código del trigger.

La estructura de estas tablas será la misma que la tabla en la que se está ejecutando el trigger (por ejemplo, si el disparador actúa sobre la tabla PRODUCTOS y estamos eliminando datos de ella, la tabla DELETED será una **copia virtual** exacta **de dicha tabla**). Estas tablas son de **sólo lectura**, es decir, NO PODEMOS MODIFICAR DATOS SOBRE ELLAS, tan sólo consultarlos.

¿Y para qué consultar estas dos tablas? Principalmente por **dos motivos**:

1. Para comprobar qué valor tenía anteriormente el registro e insertarlo en otra tabla y si no fuera correcto impedir la actualización/borrado/inserción. Actualizar un valor en otra tabla en función de qué registro se ha insertado, modificado, etc.
2. Para insertar el valor del registro anterior en una tabla de HISTORICO de CAMBIOS (por si fuera necesario recurrir a ellos en el futuro, por ejemplo, por auditorías)

La tabla INSERTED: estará disponible sólo para las operaciones INSERT y UPDATE. En ella tendremos almacenado el registro **DESPUÉS** de la inserción o actualización.

La tabla DELETED: estará disponible sólo para las operaciones DELETE y UPDATE. En ella tendremos almacenado el registro **ANTES** de modificar o borrar.

*La tabla **UPDATED no existe**, puesto que actualizar un registro por otro es lo mismo que eliminarlo (DELETE) e insertarlo con los nuevos valores (INSERT).*

Creación de disparadores DML

```
CREATE OR ALTER TRIGGER TX_TABLA ON TABLA
AFTER | INSTEAD OF INSERT, UPDATE, DELETE
AS
BEGIN
     --Código del TRIGGER

END
```

- **AFTER | INSTEAD OF** (*debemos elegir una de las dos*)
  - **AFTER**: llama al código del trigger DESPUES de ejecutar la sentencia que lo ha invocado. Ejemplo, si se ejecuta la siguiente sentencia: `DELETE FROM CLIENTES WHERE idCliente = 1`; se eliminará el cliente directamente y después se invocará al código del trigger. Si falla la integridad referencial (por ejemplo, porque el idCliente tenga registros en otras tablas, la sentencia fallará y NO se invocará al trigger).

  - **INSTEAD OF**: reemplaza la acción que ha desencadenado el trigger sin ejecutarla directamente y pasa el control al código del trigger. Ejemplo, si se ejecuta la sentencia: `DELETE FROM CLIENTES WHERE idCliente = 1`; no se eliminará el cliente directamente, sino que tan solo invocará al trigger sin borrar nada. *Puede utilizarse para simular el borrado ON CASCADE de forma controlada*.
- INSERT | UPDATE | DELETE: indica qué instrucción o instrucciones harán activarse el trigger. Debe indicarse al menos una de ellas.

Borrado de disparadores

El código para borrar triggers es el siguiente:

```
DROP TRIGGER TX_TABLA;
```

Transacciones aplicadas a los TRIGGERS

Los triggers, generalmente se encuentran dentro de la transacción principal, por lo que, si dentro del trigger ejecutamos un ROLLBACK, no solo estaremos echando atrás las modificaciones realizadas dentro del trigger sino también todas las anteriores. Por lo tanto: está desaconsejado en general utilizar ROLLBACK en un trigger porque quien debe realizar dicho ROLLBACK debe ser la transacción inicial.

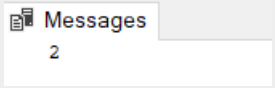
IMPORTANTE: La tabla INSERTED o DELETED contendrá tantos registros como se hayan insertado, actualizado o eliminado.

Ejemplo 0 – Trigger que devuelve el número de registros que lo han invocado

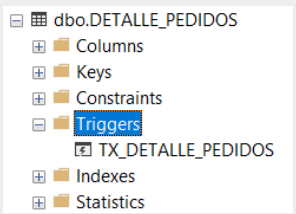
Demostración de que si insertamos varios registros simultáneamente la tabla INSERTED contendrá varios registros.

```
USE JARDINERIA
GO
-- Creación del trigger
CREATE OR ALTER TRIGGER TX_DETALLE_PEDIDOS ON DETALLE_PEDIDOS AFTER INSERT
AS
      BEGIN
            DECLARE @cantidad INT

            SET @cantidad = (SELECT COUNT(1)
                             FROM INSERTED)

            PRINT @cantidad
END

-- Ejecutamos una sentencia de inserción de varios registros
INSERT INTO DETALLE_PEDIDOS
VALUES (131, 'FR-67', 1, 6.99, 3, NULL, NULL),
       (131, 'FR-64', 1, 6.99, 4, NULL, NULL);
```

Resultado



Los triggers se ubican dentro de las propias tablas



Ejemplo 1 – Trigger que mantiene automáticamente otra tabla (solo una insert)

Implementa un trigger para que cuando se inserte un producto en la tabla DETALLE_PEDIDO se disminuya en una unidad la cantidad en stock de dicho producto.

```
USE TIENDA_DB
GO
-- Creación del trigger
CREATE OR ALTER TRIGGER TX_DETALLE_PEDIDOS ON DETALLE_PEDIDOS AFTER INSERT
AS
BEGIN
      DECLARE @cantidad INT, @codProducto VARCHAR(15)

      -- Obtenemos los valores de la pseudotabla INSERTED
      SELECT @cantidad = cantidad,
             @codProducto = codProducto
        FROM INSERTED;

      -- Disminuimos la cantidad en stock del producto
      UPDATE PRODUCTOS
         SET cantidad_en_stock = cantidad_en_stock - @cantidad
       WHERE codProducto = @codProducto;

END

SELECT codProducto, cantidad_en_stock
  FROM PRODUCTOS
 WHERE codProducto IN ('FR-67');
```

Antes

codProducto	cantidad_en_stock
FR-67	50

Ejecutamos la sentencia de inserción

```
INSERT INTO DETALLE_PEDIDOS
VALUES (131, 'FR-67', 1, 6.99, 3, NULL, NULL);
```

Después

Messages

```
(1 row affected)

(1 row affected)
```

codProducto	cantidad_en_stock
FR-67	49

Ejemplo 2 – Trigger que mantiene automáticamente otra tabla (multivalor)

Implementa un trigger para que cuando se inserte un producto en la tabla DETALLE_PEDIDO se disminuya en una unidad la cantidad en stock de dicho producto.

```
USE TIENDA_DB
GO
CREATE OR ALTER TRIGGER TX_DETALLE_PEDIDOS ON DETALLE_PEDIDOS AFTER INSERT
AS
BEGIN

      UPDATE PRODUCTOS
         SET cantidad_en_stock = cantidad_en_stock - INSERTED.cantidad
        FROM INSERTED
       WHERE PRODUCTOS.codProducto = INSERTED.codProducto;

END

Antes
SELECT codProducto, cantidad_en_stock
  FROM PRODUCTOS
 WHERE codProducto IN ('FR-67', 'FR-64');
```

codProducto	cantidad_en_stock
FR-64	15
FR-67	50

```
Ejecutamos la sentencia de inserción múltiple
INSERT INTO DETALLE_PEDIDOS
VALUES (131, 'FR-67', 1, 6.99, 3, NULL, NULL),
       (131, 'FR-64', 1, 6.99, 4, NULL, NULL);

Después
```

Messages

```
(2 rows affected)

(2 rows affected)
```

codProducto	cantidad_en_stock
FR-64	14
FR-67	49

Ejemplo 3 – Trigger que inserta en una tabla de Histórico cuando se borran

Implementa un trigger que inserte los clientes ANTES de ser modificados o borrados en una tabla llamada HISTORICO_CLIENTES con la misma estructura que CLIENTES. Se deberá agregar una columna llamada fechaCambio de tipo SMALLDATETIME a la tabla HISTORICO_CLIENTES.

```
USE TIENDA_DB
GO

-- Creación de la tabla a partir de la instrucción SELECT INTO
SELECT *
  INTO CLIENTES_HISTORICO
  FROM CLIENTES
 WHERE 1=0; -- Condición FALSA para que no copie registros, solo la estructura


-- Agregamos la columna fechaCambio
ALTER TABLE CLIENTES_HISTORICO
ADD fechaCambio SMALLDATETIME;


-- Creación del trigger
CREATE OR ALTER TRIGGER TX_CLIENTES_HIST ON CLIENTES AFTER UPDATE, DELETE
AS
BEGIN

      -- Insertamos los datos en la tabla CLIENTES_HISTORICO
      INSERT INTO CLIENTES_HISTORICO
      SELECT *, GETDATE()
        FROM DELETED;

END

-- Eliminamos un cliente que no tenga relaciones con otras tablas, funciona para
DELETE de varios registros (inserta todo el contenido de la tabla DELETED)
DELETE FROM CLIENTES
 WHERE codCliente = 37;
```

ANTES

codCliente	nombre_cliente	nombre_contacto	apellido_contacto	telefono	fax	linea_direccion1	linea_direccion2	ciudad	region	pais	codigo_postal	codigo_empleado_rep_ventas	limite_credito	fechaCambio

DESPUES

Messages

```
(1 row affected)

(1 row affected)
```

codCliente	nombre_cliente	nombre_contacto	apellido_contacto	telefono	fax	linea_direccion1	linea_direccion2	ciudad	region	pais	codigo_postal	codigo_empleado_rep_ventas	limite_credito	fechaCambio
37	The Magic Garden	Richard	Mcain	926523468	9364875882	Lihgting Park	NULL	London	NULL	NULL	NULL	18	10000.00	2023-02-08 16:55:00

Ejemplo 4 – Trigger que inserta en una tabla de Histórico cuando se borran (contemplando el borrado en cascada)

Implementa un trigger que inserte los clientes ANTES de ser modificados o borrados en una tabla llamada HISTORICO_CLIENTES con la misma estructura que CLIENTES. Se deberá agregar una columna llamada fechaCambio de tipo SMALLDATETIME a la tabla HISTORICO_CLIENTES.

Utiliza INSTEAD OF para borrar todas las tablas relacionadas simulando el borrado en cascada.

```
USE TIENDA_DB
GO

-- Creación del trigger
CREATE OR ALTER TRIGGER TX_CLIENTES_BORRADO ON CLIENTES INSTEAD OF DELETE
AS
BEGIN
      DECLARE @codCliente INT = (SELECT codCliente FROM DELETED)

      -- Borramos los productos de los pedidos del cliente
      DELETE FROM DETALLE_PEDIDOS
         WHERE codPedido IN (SELECT codPedido
                               FROM PEDIDOS
                              WHERE codCliente = codCliente);

      -- Borramos los pedidos del cliente
      DELETE FROM PEDIDOS
         WHERE codCliente = @codCliente;

      -- Borramos los pagos del cliente
      DELETE FROM PAGOS
         WHERE codCliente = @codCliente;

      -- Por último, borramos el cliente (al ser INSTEAD OF no se ha ejecutado)
      DELETE FROM CLIENTES
        WHERE codCliente = @codCliente;
END

-- Eliminamos un cliente que SI tenga relaciones con
otras tablas
DELETE FROM CLIENTES
 WHERE codCliente = 1;
```

Messages

```
(324 rows affected)
(11 rows affected)
(2 rows affected)
(1 row affected)
(1 row affected)
(1 row affected)
```

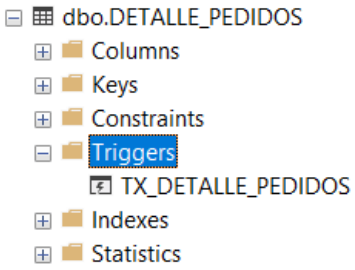
```
Comprobamos el estado final de las tablas para dicho cliente:
SELECT *
  FROM CLIENTES
 WHERE codCliente = 1;
```

codCliente	nombre_cliente	nombre_contacto	apellido_contacto	telefono	fax	linea_direccion1	linea_direccion2	ciudad	region	pais	codigo_postal	codigo_empleado_rep_ventas	limite_credito

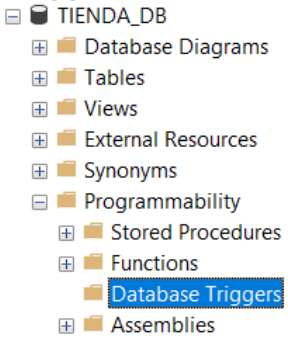
3. Triggers DDL

Los triggers DDL se ejecutan automáticamente cuando se produce una operación DDL (CREATE TABLE, ALTER TABLE o DROP TABLE) sobre cualquier tabla. La diferencia principal con los triggers DML es que los triggers DML se almacenan en las propias tablas (un trigger DML solo afecta a una tabla) y los triggers DDL afectan a la base de datos completamente y por lo tanto se ubican en lugares diferentes:

Trigger DML



Trigger DDL



Ejemplo* – *<u>Auditoría de cambios en las tablas de la BD</u>

Guardar los cambios de esquema que se realizan al modificar la estructura de una tabla de la BD.

```
USE TIENDA_DB
GO
-- Tabla que contendrá los cambios realizados en las tablas
CREATE TABLE AuditoriaCambiosBD (
      estructuraModif   XML,
      fechaModif        DATETIME
);

-- Creamos el trigger de modificación de tablas
CREATE TRIGGER TR_AUDITORIA_BD ON DATABASE FOR ALTER_TABLE, DROP_TABLE,
CREATE_TABLE
AS
BEGIN
      INSERT INTO AuditoriaCambiosBD
      SELECT EVENTDATA(), GETDATE()
END

-- Modificamos la estructura de una tabla y comprobamos las acciones del trigger
ALTER TABLE VENDEDOR
ADD fechaAlta DATE;
```

Messages

(1 row affected)

```
-- Tabla AuditoriaCambiosBD
```

estructuraModif	fechaModif
<EVENT_INSTANCE><EventType>ALTER_TABLE</EventType...	2023-02-09 14:20:45.343

Si se hace clic encima del campo XML comprobamos que tenemos toda la información necesaria para saber qué ha ocurrido:

```xml
<EVENT_INSTANCE>
  <EventType>ALTER_TABLE</EventType>
  <PostTime>2023-02-09T14:20:45.340</PostTime>
  <SPID>51</SPID>
  <ServerName>5c0a77ff5140</ServerName>
  <LoginName>sa</LoginName>
  <UserName>dbo</UserName>
  <DatabaseName>TIENDA_DB</DatabaseName>
  <SchemaName>dbo</SchemaName>
  <ObjectName>VENDEDOR</ObjectName>
  <ObjectType>TABLE</ObjectType>
  <AlterTableActionList>
    <Drop>
      <Columns>
        <Name>fechaAlta</Name>
      </Columns>
    </Drop>
  </AlterTableActionList>
  <TSQLCommand>
    <SetOptions ANSI_NULLS="ON" ANSI_NULL_DEFAULT="ON" ANSI_PADDING="ON" QUOTED_IDENTIFIER="ON" ENCRYPTED="FALSE" />
    <CommandText>alter table VENDEDOR drop column fechaAlta</CommandText>
  </TSQLCommand>
</EVENT_INSTANCE>
```

Si creamos una nueva tabla:

```
CREATE TABLE PRUEBA (
      idPrueba    INT IDENTITY(1,1),
      nombre      VARCHAR(120)
      CONSTRAINT PK_PRUEBA PRIMARY KEY (idPrueba)
);
```

estructuraModif	fechaModif
<EVENT_INSTANCE><EventType>ALTER_TABLE</EventType...	2023-02-09 14:20:45.343
<EVENT_INSTANCE><EventType>CREATE_TABLE</EventTyp...	2023-02-09 14:26:10.640

Si eliminamos una tabla:

```
DROP TABLE PRUEBA;
```

	estructuraModif	fechaModif
1	<EVENT_INSTANCE><EventType>ALTER_TABLE</EventType...	2023-02-09 14:20:45.343
2	<EVENT_INSTANCE><EventType>CREATE_TABLE</EventTyp...	2023-02-09 14:26:10.640
3	<EVENT_INSTANCE><EventType>DROP_TABLE</EventType>...	2023-02-09 14:27:09.787

4. Activación / desactivación de triggers

Podemos activar o desactivar los triggers con las siguientes instrucciones:

```
-- Desactivar el trigger TR_TABLA
DISABLE TRIGGER TR_TABLA ON TABLA;

-- Activar el trigger TR_TABLA
ENABLE TRIGGER TR_TABLA ON TABLA;
```