Carrera: Técnico Superior en Programación
Materia: Laboratorio de Computación III
Tema: Triggers

Triggers

Los Triggers nos permiten ejecutar una serie de comandos de manera desatendida. Con esto nos referimos a que se ejecutará de manera automática cuando ocurra algún tipo de acción en la base de datos.
También son conocidos como desencadenadores ya que el código o procedimiento almacenado que se ejecuta surge tras la desencadenación de un evento. En SQL-Server, los desencadenadores pueden ser del tipo DML (Data Manipulation) y DDL (Data Definition).

Los desencadenadores pueden ser 'en lugar de' o 'luego' de una acción en una tabla en particular para los del tipo DML; o bien, 'para' una acción sobre una base de datos para los tipos DDL.

Desencadenadores DML

Estos desencadenadores se ejecutan cuando un usuario ejecuta alguna acción sobre los datos de una tabla. Las acciones que desencadenan el trigger son específicamente INSERT, UPDATE o DELETE.
Existen tres tipos de desencadenadores DML, de los cuales estudiaremos dos:

- AFTER: Las acciones que conforman el trigger se ejecutan después de que la acción INSERT, UPDATE o DELETE se ejecutó.

- INSTEAD OF: Las acciones que conforman el trigger se ejecutan en lugar de la acción INSERT, UPDATE o DELETE que lo dispara.

- .NET: Estos desencadenadores pueden ser del tipo AFTER ó INSTEAD OF pero en lugar de ejecutar sentencias T-SQL como los otros triggers ejecutan código desarrollado en C#.

Por ejemplo, podremos ejecutar una serie de acciones si un dato en particular figura en nuestra consulta de INSERT, realizar una baja lógica en lugar de un DELETE o tal vez tener una tabla de LOG que permita conocer ciertos cambios cuando se ejecuta un UPDATE. En resumen, podremos ejecutar código a medida que ocurran ciertos sucesos en nuestra base de datos.

Las tablas inserted y deleted

Dos tablas especiales intervienen dentro del código de un trigger, **inserted** y **deleted.** La gestión de estas tablas son responsabilidad del motor de base de datos de SQL Server. Las mismas tienen como objetivo contener los datos intermedios que surgen con el manejo de los desencadenadores.

Cuando se ejecuta una consulta del tipo DELETE, se almacenará una copia de los datos que se verán afectados por la eliminación dentro de la tabla **deleted.** Esto quiere decir que podremos

realizar una consulta de selección a la misma para poder conocer (entre otras cosas) cuáles son los ID de los registros a eliminar, cuántos registros son, etc.
Cuando se ejecuta una consulta del tipo INSERT, se almacenará una copia de los datos que se insertarán dentro de la tabla **inserted.**
Por último, cuando se ejecuta una consulta de UPDATE se almacenará una copia de la información en ambas tablas, los datos nuevos que estarán por modificarse por la consulta de update permanecerán en la tabla **inserted** mientras que los datos viejos que serán reemplazados por la consulta de update permanecerán en la tabla **deleted.**

Utilizaremos la base de datos del apunte Procedimientos almacenados para ejemplificar los desencadenadores.

Ejemplos:

Supongamos que queremos que sólo se puede realizar baja lógica de nuestra tabla Sucursales. Esto quiere decir que no aceptaremos las sentencias de tipo DELETE como tales. Cada vez que una consulta de DELETE se ejecute, se debería ejecutar una baja lógica del campo Estado a 0. Para ello podríamos crear un procedimiento almacenado que se llame spEliminarSucursal y que realice la baja lógica pero siempre quedaría la posibilidad de que se ejecute un DELETE en lugar de nuestro Stored Procedure y sí eliminar el registro.
Para evitar esto utilizaremos un desencadenador sobre la tabla Sucursales. Lo que hay que analizar antes de realizar el desencadenador es si queremos que se ejecute 'Después' de la consulta de DELETE o 'En lugar de' la consulta de eliminación. En este caso, queremos que se ejecute 'En lugar de'.

```
CREATE TRIGGER tr_Eliminar_Sucursal ON Sucursales
INSTEAD OF DELETE
AS
BEGIN
      UPDATE Sucursales SET estado = 0 WHERE idsucursal IN (SELECT idsucursal
FROM deleted)
END
```

Veamos antes de ejecutar alguna consulta sobre la tabla Sucursales lo que contiene la misma:

	idsucursal	direccion	codpostal	estado
1	1	Mitre 1200	1	1
2	2	San Martín 233	4	1

Lo que haremos a continuación luego de compilar nuestro trigger es ejecutar una consulta de eliminación sobre nuestra tabla Sucursales.

```
DELETE FROM Sucursales
```

Ahora veremos que si volvemos a ejecutar nuestra consulta de selección sobre todos los registros de nuestra tabla Sucursales, originalmente luego de una consulta de DELETE como la anterior, no debería figurar ningún dato en nuestra tabla. Pero como en nuestra base de datos figura un trigger sobre la acción DELETE en nuestra tabla Sucursales lo que obtendremos será lo

siguiente:

	idsucursal	direccion	codpostal	estado
1	1	Mitre 1200	1	0
2	2	San Martín 233	4	0

Analizando nuestro trigger, podemos ver como cada vez que se ejecute una consulta de DELETE (ya que fue así como definimos nuestro trigger - sobre esa acción) se ejecutará 'en lugar de' dicha consulta, una consulta de UPDATE.
Es por eso que en ésta último lo que se realiza es una modificación del campo Estado (asignándole el valor cero) siempre que el código de sucursal se encuentre dentro de todos los idsucursales afectados en la consulta de DELETE, esto último se realiza mediante el uso de la tabla **deleted**.

Supongamos otro ejemplo, en el que el banco quisiera impedir que un cliente tenga más de una tarjeta de débito. Esto quiere decir que cuando insertamos un registro de tarjeta y si es de débito debemos sólo hacer el insert si el cliente no posee otra registrada.

```
CREATE TRIGGER tr_TarjetaDebito ON Tarjetas
INSTEAD OF INSERT
AS
BEGIN
IF(SELECT tipotarjeta FROM inserted) = 'D' BEGIN
 IF(SELECT COUNT(*) FROM Tarjetas WHERE tipotarjeta = 'D' AND idCliente IN
(SELECT idcliente FROM inserted)) = 1 BEGIN
  ROLLBACK TRANSACTION
  RETURN
 END
END
 INSERT INTO Tarjetas (nrotarjeta, idcliente, tipotarjeta, estado) SELECT
nrotarjeta, idcliente, tipotarjeta, estado FROM inserted
END
END
```

Analizando el código del trigger anterior, notamos que el mismo se ejecutará en lugar del insert que realizamos. De ésta manera, 'capturamos' la inserción del registro y podemos reescribir nuestro trigger de manera que ejecute el código que necesitemos.
En principio lo que debemos analizar es si la tarjeta que ingresamos es de tipo 'Débito'. Para ello lo que haremos será obtener mediante una consulta de SELECT el valor que se encuentra en el campo tipotarjeta de la tabla inserted. Si ocurre que la tarjeta es de débito, lo que debemos verificar es si la cantidad de tarjetas de débito que posee el cliente que ingresamos en nuestro insert sea igual a uno. En caso de ser así, significa que estaremos insertando una tarjeta de débito para un cliente que ya posee una, por lo tanto lo que deberemos hacer es un ROLLBACK TRANSACTION. Cabe destacar que los triggers siempre se ejecutan en el contexto de una transacción implícita y la ejecución de un ROLLBACK retraerá tanto las consultas que figuren dentro del trigger como la consulta que lo disparó. En el caso de que se ejecute un ROLLBACK en un trigger, indicará el siguiente mensaje 'La transacción terminó en el desencadenador. Se anuló el lote.', con nivel de severidad 16 y estado 1.
Si el cliente no posea una tarjeta de débito o que la tarjeta sea de crédito, se procederá a realizar

el insert manualmente obteniendo los datos de la tabla temporal inserted.

Ahora supongamos que necesitamos actualizar la regla de negocio que se sugirió en el apunte de Transacciones y que cada vez que se ingrese un cliente se le genere automáticamente una tarjeta de débito y una cuenta de tipo Caja de Ahorro.
Esto quiere decir que en este caso no vamos a querer impedir la inserción del registro de cliente sino que después de insertar dicho registro se procederá a ingresar una tarjeta y una cuenta.
El trigger se realizará sobre la tabla Clientes y se realizará después de realizar el insert. El código quedaría así:

```
CREATE TRIGGER tr_AgregarCliente ON Clientes
AFTER INSERT
AS
BEGIN
  --Agregar tarjeta de débito al cliente
  DECLARE @idCliente BIGINT
  SELECT @idCliente = idcliente FROM inserted
  INSERT INTO Tarjetas(nrotarjeta, idcliente, tipotarjeta, estado)
VALUES('D-'+CAST(@idCliente AS VARCHAR(10)+'-1', @idCliente, 'D', 1)
  --Agregar caja de ahorro al cliente
  DECLARE @nrocuenta VARCHAR(10)
  SELECT @nroCuenta = MAX(nrocuenta) FROM cuentas
  INSERT INTO Cuentas(nrocuenta, idcliente, idtipocuenta, saldo,
limite_descubierto, fecha_alta, fecha_baja, estado)
  VALUES(CAST(@nroCuenta) AS BIGINT) + 1, @idCliente, 1, 0, 0, GETDATE(), NULL,
1)
END
```

Antes de ver el funcionamiento del trigger lo que haremos será obtener los datos de las tablas Clientes, Tarjetas y Cuentas. Si realizamos la ejecución de dichos SELECT obtendremos los siguientes datos:

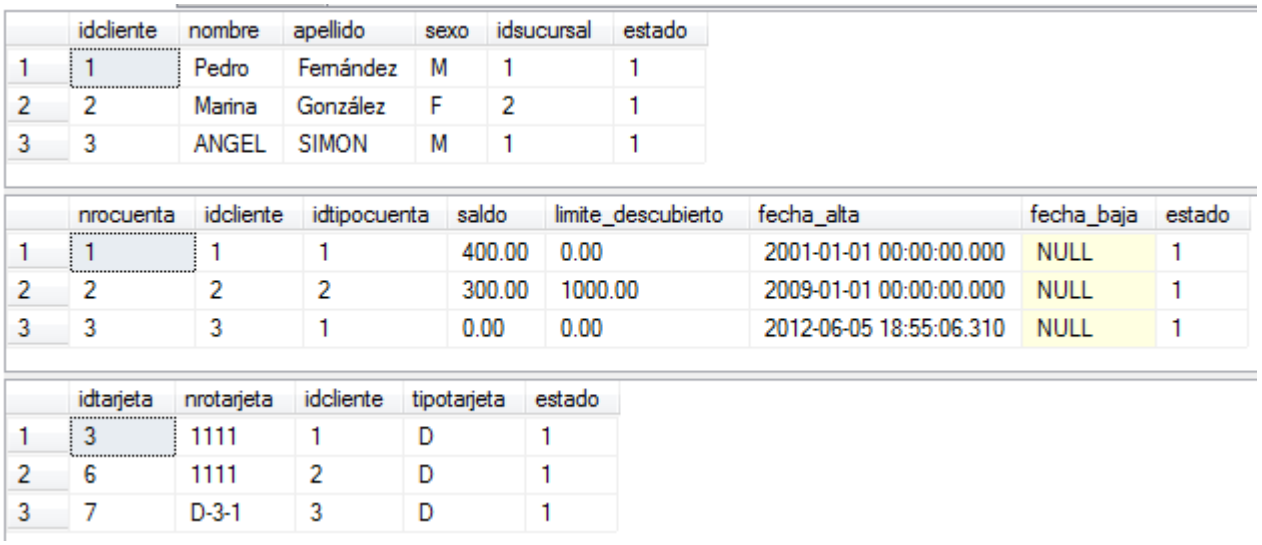
	idcliente	nombre	apellido	sexo	idsucursal	estado
1	1	Pedro	Fernández	M	1	1
2	2	Marina	González	F	2	1

	nrocuenta	idcliente	idtipocuenta	saldo	limite_descubierto	fecha_alta	fecha_baja	estado
1	1	1	1	400.00	0.00	2001-01-01 00:00:00.000	NULL	1
2	2	2	2	300.00	1000.00	2009-01-01 00:00:00.000	NULL	1

	idtarjeta	nrotarjeta	idcliente	tipotarjeta	estado
1	3	1111	1	D	1
2	6	1111	2	D	1

Luego realizaremos un INSERT sobre la tabla Clientes para entender el funcionamiento de este trigger y luego obtendremos, nuevamente, los datos de las tablas Clientes, Tarjetas y Cuentas.

```
INSERT INTO clientes (apellido, nombre, sexo, idsucursal, estado) VALUES
('SIMON', 'ANGEL', 'M', 1, 1)
```



	idcliente	nombre	apellido	sexo	idsucursal	estado
1	1	Pedro	Fernández	M	1	1
2	2	Marina	González	F	2	1
3	3	ANGEL	SIMON	M	1	1

	nrocuenta	idcliente	idtipocuenta	saldo	limite_descubierto	fecha_alta	fecha_baja	estado
1	1	1	1	400.00	0.00	2001-01-01 00:00:00.000	NULL	1
2	2	2	2	300.00	1000.00	2009-01-01 00:00:00.000	NULL	1
3	3	3	1	0.00	0.00	2012-06-05 18:55:06.310	NULL	1

	idtarjeta	nrotarjeta	idcliente	tipotarjeta	estado
1	3	1111	1	D	1
2	6	1111	2	D	1
3	7	D-3-1	3	D	1

Podemos ver como luego de realizar el insert en la tabla Clientes tenemos también los registros nuevos en las tablas de Tarjetas y Cuentas. Estos registros fueron creados dentro del trigger. Los datos relacionados al cliente se encontraban en la tabla temporal inserted por lo que para evitar acceder constantemente a esta tabla se creó una variable llamada @idCliente a la cual se le asignó el valor que se necesitaba.

Desencadenadores DDL

Los desencadenadores DDL, a diferencia de los DML tienen alcance de base de datos o de servidor. Reaccionan ante ciertos eventos que se disparan en dichos ámbitos.
Por ejemplo:

```
CREATE TRIGGER tr_Alterar_Tabla
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
BEGIN
  PRINT 'No se puede modificar la tabla. Deshabilitá el trigger
tr_Alterar_Tabla'
  ROLLBACK
END
```

El siguiente desencadenador se activará cuando querramos hacer una modificación en nuestras tablas de la base de datos. Por lo que, en este caso en particular no permitirá realizar un DROP o ALTER a una tabla.

Eliminar y modificar triggers

Una vez funcionando un trigger es posible modificar su comportamiento o bien eliminarlo por completo, para ello utilizaremos las sentencias T-SQL ALTER TRIGGER y DROP TRIGGER respectivamente.
La sintaxis general para modificar un trigger existente es:

```
ALTER TRIGGER nombre_trigger
ON nombre_tabla
{ AFTER | INSTEAD OF }
{ INSERT | UPDATE | DELETE }
AS
BEGIN
  sentencia_sql_1
  sentencia_sql_2
  sentencia_sql_N
END
```

La sintaxis general para eliminar un trigger existente es:

```
DROP TRIGGER nombre_trigger
```

Habilitar y deshabilitar triggers

Es posible que en algunos casos querramos deshabilitar momentáneamente nuestros triggers. Sería muy tedioso tener que eliminar el trigger y luego crearlo nuevamente cuando lo necesitemos. Por lo que SQL Server ofrece la posibilidad de deshabilitarlo mediante una instrucción y luego poder volver a ponerlo en funcionamiento.
La sintaxis para deshabilitar un desencadenador es la siguiente:

```
DISABLE TRIGGER nombre_trigger ON nombre_objeto
```

Por ejemplo:
```
DISABLE TRIGGER tr_AgregarCliente ON Clientes
```

Mientras que para poder habilitarlo nuevamente es:

```
ENABLE TRIGGER nombre_trigger ON nombre_objeto
```

Por ejemplo:
```
ENABLE TRIGGER tr_AgregarCliente ON Clientes
```