¿Qué es un *trigger?*

Un trigger o desencadenador es **una clase especial de procedimiento** almacenado que se ejecuta automáticamente cuando se producen determinados eventos en la base de datos.
La diferencia con los procedimientos almacenados del sistema es que los triggers:

- no pueden ser invocados directamente; al intentar modificar los datos de una tabla para la que se ha definido un disparador, el disparador se ejecuta automática mente.
- no reciben ni retornan parámetros.

Triggers y DRI

Las condiciones de integridad referencial que representan las DRI, son especificadas en el momento de la creación de las tablas y pueden manejar un buen número de situaciones. Su utilización, sin embargo, no permite resolver ciertos planteamientos en los que un análisis sofisticado de los datos sea necesario, y, sobre todo, poseen la limitación fundamental de que no permiten hacer referencia a los datos de otras bases de datos para validar los de la que nos ocupa. Esta posibilidad resulta especialmente necesaria cuando nos proponemos trabajar en entornos con aplicaciones distribuidas entre múltiples bases de datos o servidores.

Existen tres tipos de desencadenadores según los eventos:

Los desencadenadores DML se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.

Estos desencadenadores se activan cuando se desencadena cualquier evento válido, con independencia de que las filas de la tabla se vean o no afectadas.

Los desencadenadores DDL se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a

instrucciones CREATE, ALTER y DROP de Transact-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

Los desencadenadores logon se activan en respuesta al evento LOGON que se genera cuando se establece la sesión de un usuario. Los desencadenadores se pueden crear directamente a partir de instrucciones Transact-SQL.

Utilidad de los *triggers*

Los triggers son una *herramienta poderosa para centralizar en* la base de datos la toma de las decisiones de negocio que se asocian a los datos de la misma. De esta manera se descarga en gran medida a las aplicaciones cliente de la tarea de revisar las acciones de actualización de datos.

Supongamos el caso en el que existan dos tablas Existencias y Ventas de manera que cada vez que se produce una nueva venta deben actualizarse las unidades que aún quedan en existencias. La solución a este problema, si se sigue la filosofía de toma de decisiones mediante *triggers*, es dejar que sea la propia base de datos la que se encargue de realizar las actualizaciones de manera automática. De este modo, los triggers pueden utilizarse para resolver, entre otras, las siguientes situaciones:

- Establecimiento de condiciones complejas: los *triggers* permiten, especificar condiciones para que los datos de una o varias columnas sean considerados válidos.

- Respuesta a cambios en el estado de la tabla antes y después de que una determinada tarea de modificación haya sido llevada a cabo y obrar en consecuencia.

Creación de triggers (desencadenadores) DML

Aspectos básicos

Los triggers poseen fundamentalmente la utilidad de integrar en la base de datos las decisiones de negocio asociadas a los propios datos sin que sea necesario programar procedimientos en las aplicaciones *que respondan a los cambios. Por* decirlo de otro modo, permiten mantener coherente una base de datos.

La sentencia Transact-SQL que permite crear triggers es **CREATE trigger.** Como era de esperar, la creación de triggers sigue una sintaxis muy similar a la creación del resto de los objetos de la base de datos.

Para crear un trigger es necesario especificar la tabla cuyas modificaciones activarán su ejecución y las acciones que se llevarán a cabo en tal caso, en forma de una serie de sentencias Transact-SQL.

Los *triggers (DML)* están vinculados de manera inseparable a la tabla que motiva su ejecución, hasta tal punto que sólo el propietario de una tabla puede definir un *trigger* asociado a la misma.

El propietario de una tabla no puede otorgar el permiso de definición de triggers sobre la misma a ningún otro usuario.

Sólo pueden *crearse triggers* asociados a tablas de la base de datos actualmente seleccionada.

CREATE TRIGGER

Esta sentencia crea un *trigger* vinculado a una determinada tabla. Veamos su sintaxis:

DML Trigger

Trigger on an INSERT, UPDATE, or DELETE statement to a table or view

```
CREATE TRIGGER [ schema_name.] trigger_name
                  ON {table | view}
                  [ WITH <dml_trigger_option> [ ,...n ] ]
                        {FOR | AFTER | INSTEAD OF}
                        {[ INSERT] [,] [ UPDATE] [,] [ DELETE]}
                  [ NOT FOR REPLICATION]
                  AS { sql_statement [ ; ] [ ,...n ] [ ; ]
<dml_trigger_option> ::=
                   [ ENCRYPTION ]
                   [ EXECUTE AS Clause]
```

Simplificado:

CREATE TRIGGER Nombredisparador

ON NombreTabla

[**AFTER| INSTEAD OF] INSERT, UPDATE o DELETE**

AS

SENTENCIAS

En la primera parte de la sentencia encontramos la posibilidad de escoger entre **FOR, AFTER e INSTEAD OFF.** Esta elección es la que va a determinar la naturaleza del trigger es decir, si nos encontramos con un AFTER o un INSTEAD OF. En anteriores versiones la única cláusula posible era FOR. Ahora se añaden AFTER, que es sinónimo de FOR (que sólo se mantiene por compatibilidad con versiones anteriores) e INSTEAD OFF, es decir, si escogemos FOR o AFTER el trigger se ejecutará a posteriori, y si escogemos INSTEAD OF lo hará a priori.

Acciones que provocan la ejecución del *trigger*

El trigger se ejecuta como respuesta a la aplicación de ciertas sentencias de modificación sobre la tabla asociada al mismo. Estas sentencias se especifican en la cláusula UPDATE/INSERT/DELETE. Cuando se lleven a cabo la o las acciones que se especifiquen en la definición del trigger sobre la tabla, las sentencias de este se ejecutarán.

Como hemos dicho, las acciones que motivarán que un trigger se ponga en ejecución, es decir aquellas acciones que disparan el trigger, son modificaciones que puedan llevarse a cabo en los datos, tanto la adición de datos nuevos como la eliminación de datos existentes. Dichas modificaciones se llevan a cabo, evidentemente, mediante la ejecución de las sentencias UPDATE, DELETE o INSERT.

Los *triggers* se ejecutan una sola vez por sentencia.

TRIGGERS AFTER

Imaginemos el caso en el que queremos que cada vez que se inserta una nueva fila en una tabla se realice una acción de chequeo para comprobar que los datos que se han introducido en esa fila son correctos, o que al eliminar los datos de uno o varios registros se desea también eliminar o modificar los de ciertas filas relacionadas.

En estas situaciones los triggers se revelan como una herramienta poderosa y, sobre todo, automática. Es posible definir un trigger que realice las tareas descritas y que se ponga en ejecución inmediatamente después a la operación de borrado o inserción.

Cuando dicha acción de modificación se complete, el *trigger adecuado se d*isparará. Pueden definirse triggers que respondan a una o varias de estas acciones a la vez, es decir, puede definirse un *trigger* que responda tanto a una operación INSERT, como a una DELETE.

Un desencadenador AFTER se **ejecuta sólo después de ejecutar correctamente la instrucción SQL desencadenadora**.

AFTER especifica que el desencadenador DML sólo se activa cuando todas las operaciones especificadas en la instrucción SQL desencadenadora se han ejecutado correctamente. Además, todas las acciones referenciales en cascada y las comprobaciones de restricciones deben ser correctas para que este desencadenador se ejecute. **Si falla alguna el trigger no se desencadena.**

La sentencia que motiva la ejecución de un trigger y el propio trigger son considerados una única transacción que deberá ser, por tanto, completada (committed**) o descartada (**rolled back**) indivisiblemente. Si se descarta la operación en el trigger implica que se descarte la operación que lo desencadeno.**

Los desencadenadores AFTER no se pueden definir en las vistas, solo se pueden definir sobre tablas

Si existen restricciones en la tabla de desencadenadores, se comprueban antes de la de la ejecución del desencadenador AFTER. Si se infringen las restricciones, el desencadenador AFTER no se ejecuta.

Por cada tabla se pueden definir los triggers After que necesitemos para la misma acción, si conviene para el buen desarrollo del negocio.

TRIGGERS INSTEAD OF

SQL Server nos proporciona un mecanismo para poder construir triggers a priori. Los triggers INSTEAD OF provocan que **no se lleven** a cabo las acciones INSERT, UPDATE o DELETE que han lanzado el *trigger,* sino que, en su lugar, sea el *trigger* el que se ejecute. Por ejemplo, se puede definir un *trigger* INSTEAD OF para comprobar la idoneidad de los valores en una o más columnas y, a continuación, realizar acciones

adicionales antes de insertar un registro.

Como máximo, se puede definir un desencadenador INSTEAD OF por cada instrucción INSERT, UPDATE o DELETE en cada tabla o vista. No obstante, en las vistas es posible definir otras vistas que tengan su propio desencadenador INSTEAD OF.

Por ejemplo, s**i el borrado de un registro provoca la ejecución de uno de estos triggers INSTEAD OF, el borrado no llegaría a hacerse, sino que se dispararía el trigger.**

Los *triggers* INSTEAD OF pueden definirse en tablas o vistas. Esta es precisamente, su principal utilidad: hacer actualizables vistas que no lo serían *a priori.* Los *triggers* INSTEAD OF pueden proporcionar la lógica para modificar múltiples tablas de base de datos mediante una vista o para modificar tablas de base de datos que contengan estas columnas.

Tienen el mismo comportamiento transaccional que los AFTER.

Si existen restricciones en la tabla de desencadenadores, se comprueban después de la ejecución del desencadenador INSTEAD OF. Si se infringen las restricciones, se revierten las acciones del desencadenador INSTEAD OF.

Ejecución condicional del *trigger*

La funcion **UPDATE ()** devuelve un valor booleano que indica cuando un intento de Insert o Update se hace en una columna especifica de una tabla o vista. UPDATE () se usa en el cuerpo de un trigger de INSERT or UPDATE trigger para decidir cuando el trigger debe ejecutar ciertas acciones.

Las cláusulas **IF UPDATE ()** permiten que pueda conseguirse una ejecución condicional del trigger en inserción o actualización. Para cada cláusula IF UPDATE se especifica el nombre de una columna, y pueden especificarse tantas como se desee, separadas por operadores lógicos. No se utiliza con operaciones DELETE.

IF UPDATE devolverá **TRUE** si se ha realizado una inserción o si se ha realizado una actualización en la columna que se le pasa como parámetro.

Si la combinación de todas las IF UPDATE resulta ser cierta, se ejecutará el *trigger,* mientras que en caso contrario esta situación no se producirá.

Podemos utilizar ese patrón de bits para determinar si se han actualizado las columnas que nos interesan y proceder a una ejecución condicional.

El siguiente es un *trigger de* ejecución condicional sobre la tabla Empleados de la base de datos Almacen_Procedimientos_25.

```
CREATE TRIGGER tr_empleadosCambio
ON EMPLEADOS FOR INSERT, UPDATE
 AS
        IF UPDATE(IdEmpleado) OR UPDATE(Nombre)

                BEGIN
                        Print (' Se ha intentado actualizar, no está permitido')
                       ROLLBACK transaction
                 END
```

Como puede verse en el código, estamos generando un *trígger de* ejecución condicional que presenta un mensaje y descarta la operación de inserción o actualización siempre que se intente insertar un registro, o modificar las columnas IdEmpleado u Nombre

Si ejecutamos la sentencia:

```
UPDATE EMPLEADOS SET nombre= 'Rosa' WHERE nombre LIKE '%Rocio%'
```

SQL Server no realizará la modificación y nos da el siguiente mensaje:

```
Has intentado actualizar, piénsatelo bien
Mens. 3609, Nivel 16, Estado 1, Línea 1
La transacción terminó en el desencadenador. Se anuló el lote.
```

Si ejecutamos la sentencia:

```
UPDATE EMPLEADOS SET oficio= 'Analista' WHERE nombre LIKE '%Rocio%'
```

Lo modifica, ya que no es ni Id ni Nombre

La sentencia **TRÚNCATE TABLE** no desata la ejecución de los *triggers* destinados a responder a la acción DELETE.

Encriptación

Como vimos en la sintaxis de creación de procedimientos almacenados, es posible encriptar la descripción del *trigger* en la tabla syscomments, utilizando la cláusula WITH ENCRYPTION. Es común que SQL Server sea incluido como motor de bases de datos en ciertas aplicaciones comerciales. Los desarrolladores pueden utilizar la funcionalidad de encriptación para evitar que los usuarios de la aplicación puedan modificar el código creado por el programador.

Sentencias SQL del *trigger*

El cuerpo del *trigger* constará de una serie de sentencias SQL que realizarán las tareas que esperamos del mismo y una serie de condiciones que determinarán de manera adicional si las acciones del *trigger* deben ser llevadas o no cabo.

Cuando se indica más de una sentencia SQL a ejecutar, deben estar agrupadas en un bloque BEGIN ... END.

Sentencias admisibles

En general podemos decir que, en un *trigger,* como en cualquier procedimiento almacenado, pueden incluirse cualquier número y tipo de sentencias Transact_ Sql. Sin embargo, existen algunas restricciones aplicables a las sentencias ejecutables en un *trigger.* La mayoría de estas limitaciones están asociadas al hecho de que todas las modificaciones de un *trigger* deben poder ser reversibles, (en el sentido que puedan ser descartadas o rolled back, si la sentencia que dispara el *trigger* lo es). Todas aquellas sentencias, por tanto, que no puedan ser rolled back no podrán ser incluidas en un *trigger.*

Concretamente las siguientes sentencias no pueden utilizarse.

- Sentencias de selección: no está permitido utilizarla cláusula SELECT para devolver datos.

- Sentencias de creación: no puede incluirse ninguna sentencia CREATE. Recordemos que estas sentencias son:

CREATE DATABASE.
CREATE FUNCTION.
CREATE INDEX.
CREATE TABLE.
CREATE VIEW.
CREATE TRIGGER.
CREATE DEFAULT.
CREATE PROCEDURE.
CREATE RULE.

- **Sentencias de eliminación de objetos**: no pueden eliminarse objetos, por lo que no está permitido utilizar ninguna de ellas sentencias de la familia DROP.

- Sentencias de modificación de objetos: no puede alterarse la estructura de tablas ni bases de datos mediante las sentencias:

ALTER DATABASE.
ALTER TABLE.

- Sentencias de borrado de filas: no puede utilizarse la sentencia TRUNCATE TABLE para borrar todas las filas de una tabla.

- Permisos: no pueden otorgarse ni retirarse permisos, por lo que no son utilizables CRANT y REVOKE.

Existen otras sentencias que tampoco pueden utilizarse, como pueden ser UPDATE STATICS, o todas las sentencias DISK.

Tampoco se pueden utilizar objetos de los tipos TEXT o IMAGE.

Ejemplos

Vamos a presentar un ejemplo de *triggers* a priori. El código que se presenta a continuación imposibilita la operación de eliminación de registros de la tabla departamentos siempre que en ella se borre el departamento con id=1.

Crear esta tabla:

```
CREATE TABLE departamentos (
id int primary key,
nombre varchar(20)
)        .
--Meter datos:

INSERT INTO departamentos values (1, 'informática')
INSERT INTO departamentos values (2, 'contabilidad')
INSERT INTO departamentos values (3, 'administración')
INSERT INTO departamentos values (4, 'rrhh')


--Solo permite borrar de uno en uno:


CREATE OR ALTER trigger t_dep_io_delete
        on departamentos
        instead of DELETE
AS

        IF
        (SELECT id from deleted ) <>1
          DELETE departamentos where id in (SELECT id from deleted)

 ELSE

        PRINT ' No se puede borrar eldepartamento1'
```

Eliminación de *triggers*

Para eliminar un *trigger* puede utilizarse uno de los dos métodos siguientes:

Eliminación de *triggers* mediante la sentencia DROP trigger

La sentencia DROP trigger puede utilizarse de manera análoga al resto de sentencias de la familia DROP para eliminar un *trigger* simplemente pasándole como argumento el nombre del *trigger* a eliminar. Puede eliminarse más de un *trigger* de manera simultánea pasando a la sentencia sus nombres separados por comas.

DROP **trigger** *[prop.] nombretrigger1 [[prop.] nombretrigger2...]*

Sólo el propietario de una tabla puede eliminar los *triggers* asociados a la misma, y sus permisos para hacerlo no pueden transferirse.

Evidentemente, todos los *trigger* asociados a una tabla se eliminan

automáticamente cuando dicha tabla es eliminada.

Obtención de información acerca de los *triggers* existentes

Como el resto de los objetos de la base de datos, los *triggers* aparecen en la tabla sysobjects, con el identificador de tipo TR, por lo que basta con consultar esta tabla con una sentencia SELECT para obtener los *triggers* definidos.

```
Select  *  from  Sysobjects  where xtype = 'TR'
```

Su definición, como la del resto de objetos, se almacena en la tabla syscomments, y puede ser consultada si es que en el momento de definir el *trígger* no se especificó la cláusula WITH ENCRIPTION.

Por otra parte, análogamente a lo presentado en el apartado anterior, existe un procedimiento almacenado que proporciona información acerca de un determinado trigger:

```
sp_help t_dep_io_delete
sp_helptext  t_dep_io_delete
```

Las tablas inserted y deleted

En multitud de ocasiones las operaciones de actualización de la tabla que suponen inserción y borrado de filas se llevan a cabo en cadena, normalmente "por la propia naturaleza de las sentencias implicadas.

En este tipo de situaciones el *trigger* que deba responder a estas acciones puede necesitar valorar qué cambios se han producido sobre la tabla de manera que se pueda obrar en consecuencia. Para ello se necesitaría disponer de algún modo de información del estado de la tabla antes y después de las modificaciones efectuadas.

Para ello SQL Server proporciona dos tablas temporales denominadas inserted y deleted. **Solo es posible acceder a los datos de estas tablas mediante la sentencia SELECT.**

Son de estructura similar a la tabla en que se define el desencadenador, es decir, la tabla en que se intenta la acción del usuario. Las tablas **deleted** e **inserted** guardan los valores antiguos o nuevos de las filas que la acción del usuario puede cambiar.

Inserción múltiple

La tabla INSERTED almacena una copia de las filas que se han añadido durante la ejecución de una sentencia INSERT o UPDATE sobre una tabla que tiene asociado un trigger.

Para conocer el número de filas modificadas por una sentencia de actualización determinada es posible acudir a la variable global @@**rowcount**. Esta variable contiene el número de filas modificadas.

Tras una inserción, **las filas añadidas se encuentran en la tabla inserted antes de que el *trigger que* pueda estar asociado a la tabla en la que se han insertado los datos inicie su ejecución.**

Borrado múltiple

La tabla DELETED almacena una copia de las filas eliminadas durante una sentencia DELETE o UPDATE.

Evidentemente, una vez realizada la operación de borrado, las filas desaparecerán de la tabla asociada al trigger y solo aparecerán en la tabla DELETED

SQL Server crea y administra automáticamente ambas tablas. Puede utilizar estas tablas temporales residentes en memoria para probar los efectos de determinadas modificaciones de datos y para establecer condiciones para las acciones del desencadenador DML. No puede modificar directamente los datos de estas tablas ni realizar en ellas operaciones de lenguaje de definición de datos (DDL).

En los desencadenadores DML, las tablas **inserted** y **deleted** se utilizan principalmente para realizar las siguientes tareas:

- Ampliar la integridad referencial entre tablas.
- Insertar o actualizar datos de tablas base subyacentes a una vista.
- Comprobar errores y realizar acciones en función del error.
- Conocer la diferencia entre el estado de una tabla antes y después de realizar una modificación en los datos, y actuar en función de dicha diferencia.