



Los desencadenantes

En muchas ocasiones deseamos que un proceso se ejecute automáticamente sobre los datos de una base de datos cuando una aplicación produce algún cambio en los datos. Para ello podemos definir en la base de datos código del lado del servidor a través de los desencadenantes ó triggers.

Esta página se ha dejado en blanco intencionalmente.

Capítulo 15

Los desencadenantes

Contenido

- ❑ *Desencadenantes DML*
 - ✓ *El desencadenante como transacción*
 - ✓ *Uso de los desencadenantes DML*
 - ✓ *Consideraciones para usar los desencadenantes DML*
 - ✓ *Tipos de desencadenantes DML*
 - *Desencadenantes AFTER*
 - *Desencadenantes INSTEAD OF*
 - ✓ *Creación de desencadenantes DML*
 - ✓ **Ejercicio 141:** *Preparación de las tablas*
 - ✓ **Ejercicio 142:** *Desencadenante AFTER INSERT*
 - ✓ *Mecanismo de un desencadenante AFTER INSERT – La tabla INSERTED*
 - ✓ **Ejercicio 143:** *Desencadenante AFTER DELETE*
 - ✓ *Mecanismo de un desencadenante AFTER DELETE – La tabla DELETED*
 - ✓ **Ejercicio 144:** *Desencadenante AFTER UPDATE*
 - ✓ *Mecanismo de un desencadenante AFTER UPDATE*
 - ✓ **Ejercicio 145:** *Desencadenante INSTEAD OF*
- ❑ *Desencadenantes DDL*
 - ✓ *Uso de los desencadenantes DDL*
 - ✓ *Creación de los desencadenantes DDL*
 - ✓ **Ejercicio 146:** *Desencadenante DDL con alcance para la base de datos actual*
 - ✓ **Ejercicio 147:** *Desencadenante DDL con alcance de servidor*

- ✓ *Eventos DDL*
 - *Sentencias DDL con alcance de base de datos*
 - *Sentencias DDL con alcance de servidor*
 - *Grupos de eventos*
- *Mantenimiento de los desencadenantes*
 - ✓ *Modificación de un desencadenante*
 - ✓ *Eliminación de un desencadenante*
 - ✓ *Deshabilitación/habilitación de un desencadenante*

Los desencadenantes

Los desencadenantes DML se ejecutan cuando una declaración DML (Data Manipulation Language) como INSERT, UPDATE ó DELETE ocurre en la base de datos. MS SQL Server 2005 incorpora los desencadenantes DDL (Data Definition Language) que se disparan al ejecutarse las siguientes declaraciones DDL: CREATE, ALTER y DROP.

Desencadenantes DML

- Un desencadenante DML es un clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce una modificación en el contenido de una tabla.
- Un desencadenante es definido específicamente para una tabla, y para ser ejecutado automáticamente ("disparado") cuando se produce una inserción de filas, una actualización de datos o una eliminación de filas.
- A diferencia de los procedimientos almacenados, un desencadenante no puede ser llamado directamente, y no acepta o retorna parámetros.
- Un desencadenante es tratado como una transacción que puede ser confirmada ó cancelada desde el interior del desencadenante.

El desencadenante como transacción

El desencadenante y la sentencia que lo dispara se tratan como si fueran una sola transacción en la que puede ejecutarse un ROLLBACK TRANSACTION en cualquier parte del desencadenante aún cuando no se haya declarado una sentencia BEGIN TRANSACTION de manera explícita. La declaración que invoca el desencadenante es considerada el principio de una transacción implícita, a menos que una declaración BEGIN TRANSACTION explícita sea incluida.

Un desencadenante que ejecuta una declaración ROLLBACK TRANSACTION desde dentro cancela no solo el desencadenante sino también el batch desde que el desencadenante se disparó.

Debe minimizar o debe evitar el uso de ROLLBACK TRANSACTION en el código de sus desencadenantes.

Uso de los desencadenantes DML

- Los desencadenantes se utilizan para definir lógica de aplicación compleja. Son adecuados para los procesos que se ejecutan en cascada en varias tablas.
- Los desencadenantes protegen contra la ejecución maliciosa ó incorrecta de INSERT, UPDATE, y DELETE y permiten definir restricciones más complejas que aquellas definidas con las restricciones CHECK. A diferencia de CHECK, un desencadenante DML puede hacer referencia a columnas de otras tablas. Por ejemplo, un desencadenante puede ejecutar un SELECT de otra tabla para comparar el dato insertado ó actualizado y ejecutar acciones adicionales, tales como modificar el dato ó mostrar un mensaje de error definido por el usuario.
- Como el desencadenante es reactivo, él puede comparar los datos antes y después que la data ha sido modificada y tomar una acción en base al resultado de la comparación.

Consideraciones para usar los desencadenantes DML

Tenga en cuenta lo siguiente al trabajar con desencadenantes:

- Los desencadenantes son reactivos; las restricciones son proactivas.

Los desencadenantes son ejecutados después de una sentencia INSERT, UPDATE, o DELETE que se ejecuta en la tabla donde se ha definido el desencadenante. Por ejemplo, una declaración UPDATE actualiza una fila en una tabla, y entonces el desencadenante por actualización en esa tabla se ejecuta automáticamente. Las restricciones se verifican antes de que una sentencia INSERT, UPDATE, o DELETE se ejecute.
- Las restricciones se verifican antes que los desencadenantes.

Si existen restricciones en la tabla que contiene el desencadenante, ellas se verifican antes de la ejecución del desencadenante. Si se violan las restricciones, el desencadenante no se ejecuta.
- Las tablas pueden tener múltiples desencadenantes para cualquier acción.

Desde la versión 7 de SQL Server es posible definir múltiples desencadenantes en una sola tabla. Cada desencadenante puede definirse para una sola acción o para acciones múltiples. Los desencadenantes no se disparan en ningún orden en particular.

- Los dueños de las tablas pueden definir cuáles serán los desencadenantes que se dispararán primero y último.

El dueño de la tabla puede utilizar el procedimiento `sp_settriggerorder` para especificar el primer y el último desencadenante a disparar.

- Los dueños de las tablas deben tener el permiso para ejecutar todas las declaraciones definidas en los desencadenantes.

Sólo el dueño de la tabla puede crear y puede eliminar los desencadenantes para esa tabla. Estos permisos no pueden transferirse. Además, el dueño de la tabla también debe tener el permiso para ejecutar todas las declaraciones en todas las tablas afectadas. Si se niegan los permisos a cualquier porción de las declaraciones Transact-SQL dentro del desencadenante, la transacción entera realiza un ROLLBACK.

- Los dueños de las tablas no pueden crear desencadenantes AFTER sobre vistas o en las tablas temporales. Sin embargo, los desencadenantes pueden hacer referencia a las vistas y las tablas temporales.
- Los dueños de las tablas pueden crear desencadenantes INSTEAD OF sobre vistas y tablas extendiendo los tipos de actualizaciones que una vista puede soportar.
- Los desencadenantes no deben retornar un conjunto de filas como resultado.

Los desencadenantes contienen sentencias Transact-SQL, del mismo modo que los procedimientos almacenados. Como los procedimientos almacenados, los desencadenantes pueden contener declaraciones que devuelven un conjunto de filas como resultado. Sin embargo, incluso declaraciones que devuelven valores en los desencadenantes no se recomiendan porque los usuarios no esperan ver el resultado que el desencadenante devuelve cuando ellos ejecutan una declaración UPDATE, INSERT o DELETE.

Tipos de desencadenantes DML

Podemos programar los siguientes tipos de desencadenantes DML:

- Desencadenantes AFTER
- Desencadenantes INSTEAD OF

Desencadenantes AFTER

Son disparados después que una acción INSERT, UPDATE ó DELETE es ejecutada. Este tipo de desencadenantes solo se puede especificar para tablas.

Desencadenantes INSTEAD OF

Son ejecutados en lugar de la acción de disparo habitual. Los desencadenantes INSTEAD OF se pueden definir también sobre vistas que tienen una ó más tablas subyacentes, y extienden el tipo de actualizaciones que una vista puede soportar.

Creación de desencadenantes DML

Los desencadenantes son creados con la declaración CREATE TRIGGER. La declaración especifica la tabla en la que el desencadenante se define, los eventos que hacen que se ejecute, y las instrucciones particulares para el desencadenante.

Sintaxis

```
CREATE TRIGGER nombreDesencadenante
ON nombreTabla | nombreVista
FOR [ INSERT ] [,] [ UPDATE ] [,] [ DELETE ]
    | AFTER [ INSERT ] [,] [ UPDATE ] [,] [ DELETE ]
    | INSTEAD OF [ INSERT ] [,] [ UPDATE ] [,] [ DELETE ]
AS
sentenciasSQL
```

- FOR y AFTER definen desencadenantes AFTER. FOR se conserva por compatibilidad con versiones anteriores de SQL Server.
- Los dueños de la tabla, así como los miembros de los roles **db_owner** y **sysadmin**, tienen permiso para crear un desencadenante.

- SQL Server no permite usar las declaraciones siguientes en una definición de desencadenante:
 - CREATE DATABASE, ALTER DATABASE y DROP DATABASE
 - LOAD DATABASE y LOAD LOG
 - RESTORE DATABASE
 - RESTORE LOG
 - RECONFIGURE

Ejercicio 141: Preparación de las tablas

En este ejercicio creará una base de datos de pruebas que contendrá las tablas en la que se diseñarán y probarán los desencadenantes.

```
CREATE DATABASE Test
go

USE Test
go

-- Crear tablas MAESTRO-DETALLE
CREATE TABLE Factura(
    IdFactura int PRIMARY KEY,
    FecFactura datetime DEFAULT getdate(),
    Cliente varchar(30) not null,
    MontoFactura money null )
go

CREATE TABLE DetalleFactura(
    IdFactura int not null,
    IdProducto integer not null,
    PrecioUnitario money not null,
    Cantidad int not null )
go

ALTER TABLE DetalleFactura
    ADD CONSTRAINT pk_DetalleFactura
    PRIMARY KEY( IdFactura, IdProducto )
go
```

```
ALTER TABLE DetalleFactura
    ADD CONSTRAINT fk_DetalleFactura_Factura
    FOREIGN KEY( IdFactura )
    REFERENCES Factura
go

SET DATEFORMAT dmy
go

INSERT INTO Factura
    VALUES( 1, '31/10/2005', 'Comercial Gómez', NULL )
INSERT INTO Factura
    VALUES( 2, '02/11/2005', 'Juan López Cordero', NULL )
go

INSERT DetalleFactura VALUES( 1, 101, 12.5, 100 )
INSERT DetalleFactura VALUES( 1, 127, 15, 50 )
INSERT DetalleFactura VALUES( 1, 107, 10, 50 )
INSERT DetalleFactura VALUES( 2, 132, 15.5, 100 )
INSERT DetalleFactura VALUES( 2, 107, 10, 250 )
go

UPDATE Factura SET montoFactura = 2500
    WHERE idFactura = 1
UPDATE Factura SET montoFactura = 4050
    WHERE idFactura = 2
go
```

Ejercicio 142: Desencadenante AFTER INSERT

Crear un desencadenante AFTER INSERT que recalcule y actualice el monto de una factura cada vez que se inserta un detalle para dicha factura.

```
USE Test
go

-- Desencadenante AFTER INSERT para DetalleFactura.
-- Recalcula el Monto de la Factura por cada detalle
-- insertado.
CREATE TRIGGER tg_insert_DetalleFactura
ON DetalleFactura AFTER INSERT
AS
DECLARE @factura int
DECLARE @suma money
SET @factura = ( SELECT idFactura FROM inserted )
SET @suma =
    ( SELECT SUM( precioUnitario * cantidad )
      FROM DetalleFactura
      WHERE DetalleFactura.idFactura = @factura )
UPDATE Factura
    SET montoFactura = @suma
    WHERE idFactura = @factura
go
```

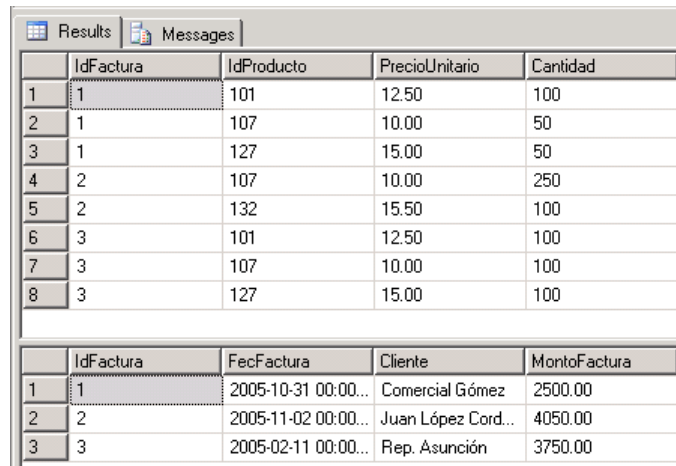
Para probar el desencadenante codifique y ejecute los siguientes batchs:

```
-- Registrando la factura 3
INSERT INTO Factura
    VALUES( 3, '02/11/2005', 'Rep. Asunción', NULL )
go

-- Probando el desencadenante
INSERT DetalleFactura VALUES( 3, 101, 12.5, 100 )
INSERT DetalleFactura VALUES( 3, 127, 15, 100 )
INSERT DetalleFactura VALUES( 3, 107, 10, 100 )
go
```

```
-- Verificando la data
SELECT * FROM DetalleFactura
go

SELECT * FROM Factura
go
```



The screenshot shows the SQL Server Enterprise Manager interface with two query result grids. The top grid displays the contents of the 'DetalleFactura' table, and the bottom grid displays the contents of the 'Factura' table.

	IdFactura	IdProducto	PrecioUnitario	Cantidad
1	1	101	12.50	100
2	1	107	10.00	50
3	1	127	15.00	50
4	2	107	10.00	250
5	2	132	15.50	100
6	3	101	12.50	100
7	3	107	10.00	100
8	3	127	15.00	100

	IdFactura	FecFactura	Cliente	MontoFactura
1	1	2005-10-31 00:00...	Comercial Gómez	2500.00
2	2	2005-11-02 00:00...	Juan López Cord...	4050.00
3	3	2005-02-11 00:00...	Rep. Asunción	3750.00

Mecanismo de un desencadenante AFTER INSERT – La tabla INSERTED

Cuando se ejecuta una declaración INSERT en una tabla que tiene definido un desencadenante AFTER INSERT, se crea automáticamente una tabla temporal INSERTED que tiene la misma estructura que la tabla con el desencadenante.

La tabla temporal INSERTED contiene una copia de la fila insertada por la declaración INSERT ejecutada en la tabla con el desencadenante.

Por ejemplo, antes de registrar la **factura 3**, las tablas **Factura** y **DetalleFactura** contienen los siguientes datos:

Tabla Factura			
IdFactura	FecFactura	Cliente	MontoFactura
1	31/10/2005	Comercial Gómez	2500.00
2	02/11/2005	Juan López Cordero	4050.00

Tabla DetalleFactura			
IdFactura	IdProducto	PrecioUnitario	Cantidad
1	101	12.50	100
1	107	10.00	50
1	127	15.00	50
2	107	10.00	250
2	132	15.50	100

Procedemos a registrar la cabecera de la **factura 3** (los datos de la factura 3 se muestran en negritas y cursivas):

```
INSERT INTO Factura
VALUES( 3, '02/11/2005', 'Rep. Asunción', NULL )
go
```

Tabla Factura			
IdFactura	FecFactura	Cliente	MontoFactura
1	31/10/2005	Comercial Gómez	2500.00
2	02/11/2005	Juan López Cordero	4050.00
3	02/11/2005	Rep. Asunción	NULL

Ahora, insertamos un detalle para la **factura 3** (estos datos también se muestran en negritas y cursivas):

```
INSERT DetalleFactura VALUES( 3, 101, 12.5, 100 )
```

Tabla DetalleFactura			
IdFactura	IdProducto	PrecioUnitario	Cantidad
1	101	12.50	100
1	107	10.00	50
1	127	15.00	50
2	107	10.00	250
2	132	15.50	100
3	101	12.50	100

En este momento, SQL Server crea la tabla temporal INSERTED que contiene una copia de la fila insertada:

Tabla INSERTED			
IdFactura	IdProducto	PrecioUnitario	Cantidad
3	101	12.50	100

El código del desencadenante utiliza el **idFactura** en la tabla INSERTED para determinar cuál es la factura cuyo monto debe actualizar en la tabla **Factura**.

```
...
...
SET @factura = ( SELECT idFactura FROM inserted )
SET @suma =
    ( SELECT SUM( precioUnitario * cantidad )
      FROM DetalleFactura
      WHERE DetalleFactura.idFactura = @factura )
UPDATE Factura
    SET montoFactura = @suma
    WHERE idFactura = @factura
```

Tabla Factura			
IdFactura	FecFactura	Cliente	MontoFactura
1	31/10/2005	Comercial Gómez	2500.00
2	02/11/2005	Juan López Cordero	4050.00
3	02/11/2005	Rep. Asunción	1250.00

Ejercicio 143: Desencadenante AFTER DELETE

Crear un desencadenante AFTER DELETE que recalcule y actualice el monto de una factura cada vez que se elimina un detalle para dicha factura.

```
USE Test
go

-- Desencadenante por ELIMINACION para DetalleFactura
-- Recalcula el Monto de la Factura por cada detalle
-- eliminado
CREATE TRIGGER tg_delete_DetalleFactura
ON DetalleFactura AFTER DELETE
AS
DECLARE @factura int
DECLARE @suma money
SET @factura = ( SELECT idFactura FROM deleted )
SET @suma =
    ( SELECT SUM( PrecioUnitario * Cantidad )
      FROM DetalleFactura
      WHERE DetalleFactura.idFactura = @factura )
UPDATE factura
    SET MontoFactura = @suma
    WHERE idFactura = @factura
go
```

Pruebe el desencadenante eliminando el primer detalle de la **factura 1**.

```
-- Prueba del desencadenante mediante la eliminación
-- del primer detalle de la factura 1
DELETE FROM DetalleFactura
    WHERE idFactura = 1 AND idProducto = 101
go

-- Verificando la data
SELECT * FROM DetalleFactura
go

SELECT * FROM Factura
go
```


Results		Messages		
	IdFactura	IdProducto	PrecioUnitario	Cantidad
1	1	107	10.00	50
2	1	127	15.00	50
3	2	107	10.00	250
4	2	132	15.50	100
5	3	101	12.50	100
6	3	107	10.00	100
7	3	127	15.00	100

	IdFactura	FecFactura	Cliente	MontoFactura
1	1	2005-10-31 00:00...	Comercial Gómez	1250.00
2	2	2005-11-02 00:00...	Juan López Cord...	4050.00
3	3	2005-02-11 00:00...	Rep. Asunción	3750.00

Mecanismo de un desencadenante AFTER DELETE – La tabla DELETED

Cuando se ejecuta una declaración DELETE en una tabla que tiene definido un desencadenante AFTER DELETE, se crea automáticamente una tabla temporal DELETED que tiene la misma estructura que la tabla con el desencadenante.

La tabla temporal DELETED contiene a las filas que la declaración DELETE ejecutada en la tabla con el desencadenante ha eliminado.

Por ejemplo, antes de eliminar el primer detalle de la **factura 1**, las tablas **Factura** y **DetalleFactura** contienen los siguientes datos:

Tabla Factura			
IdFactura	FecFactura	Cliente	MontoFactura
1	31/10/2005	Comercial Gómez	2500.00
2	02/11/2005	Juan López Cordero	4050.00
3	02/11/2005	Rep. Asunción	3750.00

Tabla DetalleFactura			
IdFactura	IdProducto	PrecioUnitario	Cantidad
1	101	12.50	100
1	107	10.00	50
1	127	15.00	50
2	107	10.00	250
2	132	15.50	100
3	101	12.50	100
3	107	10.00	100
3	127	15.00	100

Procedemos a eliminar el primer detalle de la **factura 1**:

```
DELETE FROM DetalleFactura
      WHERE idFactura = 1 AND idProducto = 101
go
```

Tabla DetalleFactura			
IdFactura	IdProducto	PrecioUnitario	Cantidad
1	107	10.00	50
1	127	15.00	50
2	107	10.00	250
2	132	15.50	100
3	101	12.50	100
3	107	10.00	100
3	127	15.00	100

En este momento, SQL Server crea la tabla temporal DELETED que contiene a la fila eliminada por la declaración DELETE:

Tabla DELETED			
IdFactura	IdProducto	PrecioUnitario	Cantidad
<i>1</i>	<i>101</i>	<i>12.50</i>	<i>100</i>

El código del desencadenante utiliza el **idFactura** en la tabla DELETED para determinar cuál es la factura cuyo monto debe actualizar en la tabla **Factura**.

```

...
...
SET @factura = ( SELECT idFactura FROM deleted )
SET @suma =
    ( SELECT SUM( precioUnitario * cantidad )
      FROM DetalleFactura
      WHERE DetalleFactura.idFactura = @factura )
UPDATE Factura
    SET montoFactura = @suma
    WHERE idFactura = @factura

```

Tabla Factura			
IdFactura	FecFactura	Cliente	MontoFactura
<i>1</i>	<i>31/10/2005</i>	<i>Comercial Gómez</i>	<i>1250.00</i>
2	02/11/2005	Juan López Cordero	4050.00
3	02/11/2005	Rep. Asunción	3750.00

Ejercicio 144: Desencadenante AFTER UPDATE

Crear un desencadenante AFTER UPDATE que recalcule el monto de una factura cada vez que se actualiza la cantidad ó el precio de un detalle de dicha factura.

```
USE Test
go

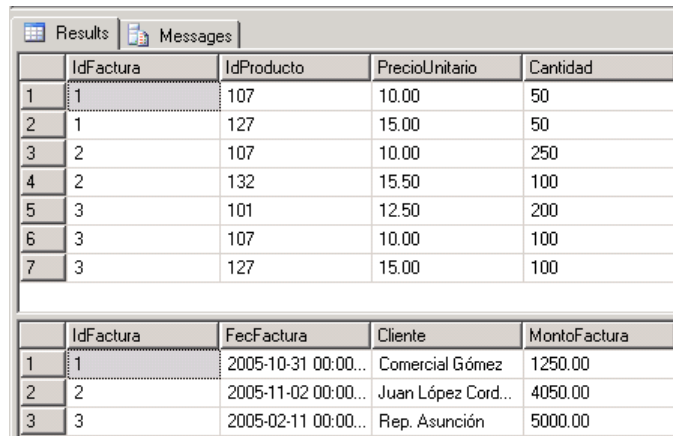
CREATE TRIGGER tg_update_DetalleFactura
ON DetalleFactura AFTER UPDATE
AS
IF UPDATE(precioUnitario) OR UPDATE(cantidad)
BEGIN
    DECLARE @factura int
    DECLARE @suma money
    SET @factura = ( SELECT idFactura FROM inserted )
    SET @suma =
        ( SELECT SUM( PrecioUnitario * Cantidad )
          FROM DetalleFactura
          WHERE DetalleFactura.idFactura = @factura )
    UPDATE factura
    SET MontoFactura = @suma
    WHERE idFactura = @factura
END
go
```

Pruebe el desencadenante duplicando la cantidad para el primer detalle de la **factura 3**.

```
-- Prueba del desencadenante duplicando la cantidad en
-- el primer item de la factura 3
UPDATE DetalleFactura
    SET cantidad = 200
    WHERE idFactura = 3 AND idProducto = 101
go
```

```
-- Verificando la data
SELECT * FROM DetalleFactura
go

SELECT * FROM Factura
go
```



The screenshot shows the SQL Server Enterprise Manager interface with two tabs: 'Results' and 'Messages'. The 'Results' tab is active, displaying two tables of query results. The first table, 'DetalleFactura', has columns: IdFactura, IdProducto, PrecioUnitario, and Cantidad. The second table, 'Factura', has columns: IdFactura, FecFactura, Cliente, and MontoFactura.

	IdFactura	IdProducto	PrecioUnitario	Cantidad
1	1	107	10.00	50
2	1	127	15.00	50
3	2	107	10.00	250
4	2	132	15.50	100
5	3	101	12.50	200
6	3	107	10.00	100
7	3	127	15.00	100

	IdFactura	FecFactura	Cliente	MontoFactura
1	1	2005-10-31 00:00...	Comercial Gómez	1250.00
2	2	2005-11-02 00:00...	Juan López Cord...	4050.00
3	3	2005-02-11 00:00...	Rep. Asunción	5000.00

Mecanismo de un desencadenante AFTER UPDATE

Cuando se ejecuta una declaración UPDATE en una tabla que tiene definido un desencadenante AFTER UPDATE, se crean automáticamente las tablas temporales INSERTED y DELETED que tienen la misma estructura que la tabla con el desencadenante.

La tabla temporal DELETED contiene las filas afectadas por la declaración UPDATE, pero con los valores anteriores a la ejecución de UPDATE. La tabla temporal INSERTED contiene las filas afectadas pero con los valores actualizados.

Ejercicio 145: Desencadenante INSTEAD OF

Procedemos a crear las tablas de prueba para el desencadenante INSTEAD OF.

```
USE Test
go

-- Creación de las tablas PROVEEDOR y CLIENTE
CREATE TABLE Proveedor(
    idProveedor char(5) PRIMARY KEY,
    nombre varchar(30) not null,
    telefono varchar(20) null )
go

INSERT INTO Proveedor
VALUES('P0001', 'Juan Castro Arenas', '4512345')
INSERT INTO Proveedor
VALUES('P0002', 'Ernesto Rosado Albán', '3491234')
INSERT INTO Proveedor
VALUES('P0003', 'Rocío Sánchez Alania', '99871234')
go

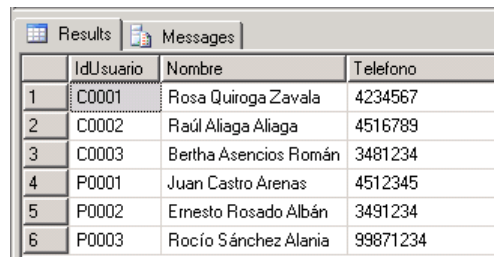
CREATE TABLE Cliente(
    idCliente char(5) PRIMARY KEY,
    nombre varchar(30) not null,
    telefono varchar(20) null )
go

INSERT INTO Cliente
VALUES('C0001', 'Rosa Quiroga Zavala', '4234567')
INSERT INTO Cliente
VALUES('C0002', 'Raúl Aliaga Aliaga', '4516789')
INSERT INTO Cliente
VALUES('C0003', 'Bertha Asencios Román', '3481234')
go
```

Ahora, creamos una vista que une las tablas PROVEEDOR y CLIENTE.

```
-- Creación de una vista que une las tablas PROVEEDOR y
CLIENTE
CREATE VIEW v_Usuarios
AS
SELECT IdProveedor AS IdUsuario, Nombre, Telefono
      FROM Proveedor
UNION
SELECT IdCliente, Nombre, Telefono
      FROM Cliente
go

SELECT * FROM v_Usuarios
go
```



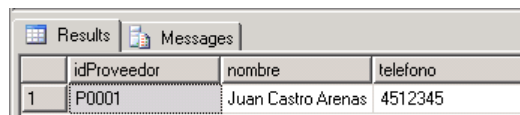
	IdUsuario	Nombre	Telefono
1	C0001	Rosa Quiroga Zavala	4234567
2	C0002	Raúl Aliaga Aliaga	4516789
3	C0003	Bertha Asencios Román	3481234
4	P0001	Juan Castro Arenas	4512345
5	P0002	Ernesto Rosado Albán	3491234
6	P0003	Rocío Sánchez Alania	99871234

A continuación, creamos sobre la vista **v_Usuarios**, un desencadenante INSTEAD OF que cuando se actualiza el telefono de un usuario a través de la vista se actualiza el dato en la tabla correspondiente.


```
CREATE TRIGGER tg_update_v_Usuarios
ON v_Usuarios
INSTEAD OF UPDATE
AS
DECLARE @tipo char(1)
SET @tipo = (SELECT left(idUsuario, 1) FROM inserted)
IF @tipo = 'C'
BEGIN
    UPDATE Cliente
    SET Cliente.telefono = inserted.telefono
    FROM Cliente INNER JOIN inserted
    ON Cliente.idCliente = inserted.idUsuario
END
ELSE
IF @tipo = 'P'
BEGIN
    UPDATE Proveedor
    SET Proveedor.telefono = inserted.telefono
    FROM Proveedor INNER JOIN inserted
    ON Proveedor.idProveedor = inserted.idUsuario
END
go
```

Probamos el desencadenante actualizando el teléfono del proveedor de código **P0001**.

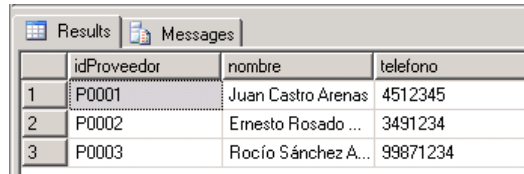
```
-- Probando el desencadenante
SELECT * FROM Proveedor
    WHERE idProveedor = 'P0001'
go
-- Su telefono es 4512345
```



Results		Messages	
	idProveedor	nombre	telefono
1	P0001	Juan Castro Arenas	4512345

```
-- Usamos la vista para cambiar el telefono de P0001
UPDATE v_Usuarios
    SET telefono = '4231234'
    WHERE idUsuario = 'P0001'
go

SELECT * FROM Proveedor
go
-- el telefono de P0001 fue actualizado en la tabla
correspondiente
```



	idProveedor	nombre	telefono
1	P0001	Juan Castro Arenas	4512345
2	P0002	Ernesto Rosado ...	3491234
3	P0003	Rocío Sánchez A...	99871234

Desencadenantes DDL

Un desencadenante DDL es una clase especial de desencadenante que se dispara en respuesta a la ejecución de una sentencia DDL (Data Definition Language). El desencadenante DDL se puede utilizar para ejecutar tareas administrativas en la base de datos, tales como auditoría y control de las operaciones de bases de datos.

Uso de los desencadenantes DDL

Use desencadenantes DDL cuando:

- Desea impedir que ciertos cambios se produzcan en el esquema de la base de datos.
- Desea que algo ocurra en la base de datos en respuesta a un cambio en su esquema.
- Desea registrar los cambios ó eventos en el esquema de la base de datos.

Creación de los desencadenantes DDL

Sintaxis

```
CREATE TRIGGER nombreDesencadenante  
ON ALL SERVER | ON DATABASE  
FOR | AFTER eventoDDL, eventoDDL, ...  
AS  
sentenciasSQL
```

- Un desencadenante DDL se puede definir solo para la base de datos actual ó para todo el servidor. Esto determina el alcance del desencadenante: ALL SERVER ó DATABASE.
- Un desencadenante DDL es siempre de tipo AFTER. La cláusula FOR se conserva por compatibilidad con versiones anteriores de SQL Server.

Ejercicio 146: Desencadenante DDL con alcance para la base de datos actual

Este ejercicio muestra cómo utilizar un desencadenante DDL para impedir que una tabla se pueda eliminar ó modificar su definición.

```
USE Test
go

CREATE TRIGGER tg_seguridad
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
    PRINT 'Debe deshabilitar el desencadenante tg_seguridad
para eliminar ó modificar una tabla.'
    ROLLBACK
go
```

Probamos el desencadenante.

```
DROP TABLE Cliente
go
```

Se recibe el siguiente mensaje:

```
Debe deshabilitar el desencadenante tg_seguridad para
eliminar ó modificar una tabla.
Msg 3609, Level 16, State 2, Line 1
The transaction ended in the trigger. The batch has been
aborted.
```

Ejercicio 147: Desencadenante DDL con alcance de servidor

En este ejercicio se muestra un desencadenante que se dispara cuando en el servidor ocurre un evento CREATE LOGIN, ALTER LOGIN ó DROP LOGIN.

```
CREATE TRIGGER tg_login
ON ALL SERVER
FOR DDL_LOGIN_EVENTS
AS
    PRINT 'Evento Login.'
    RAISERROR('Se ha registrado la modificación en la BD de
seguridad.', 16, 1)
go
```

Para probar el desencadenante vamos a crear una nueva cuenta de inicio de sesión (login name).

```
-- Probando el desencadenante
CREATE LOGIN pacherez
    WITH PASSWORD = 'cachete'
go
```

Se genera el siguiente mensaje:

```
Evento Login.
Msg 50000, Level 16, State 1, Procedure tg_login, Line 6
Se ha registrado la modificación en la BD de seguridad.
```

Eventos DDL

Las tablas siguientes muestran los eventos DDL que se pueden utilizar con los desencadenantes DDL.

Sentencias DDL con alcance de base de datos

CREATE_APPLICATION_ROLE	ALTER_APPLICATION_ROLE	DROP_APPLICATION_ROLE
CREATE_ASSEMBLY	ALTER_ASSEMBLY	DROP_ASSEMBLY
ALTER_AUTHORIZATION_DATABASE		
CREATE_CERTIFICATE	ALTER_CERTIFICATE	DROP_CERTIFICATE
CREATE_CONTRACT	ALTER_CONTRACT	DROP_CONTRACT
GRANT_DATABASE	DENY_DATABASE	REVOKE_DATABASE
CREATE_EVENT_NOTIFICATION	DROP_EVENT_NOTIFICATION	
CREATE_FUNCTION	ALTER_FUNCTION	DROP_FUNCTION
CREATE_INDEX	ALTER_INDEX	DROP_INDEX
CREATE_MESSAGE_TYPE	ALTER_MESSAGE_TYPE	DROP_MESSAGE_TYPE
CREATE_PARTITION_FUNCTION	ALTER_PARTITION_FUNCTION	DROP_PARTITION_FUNCTION
CREATE_PARTITION_SCHEME	ALTER_PARTITION_SCHEME	DROP_PARTITION_SCHEME
CREATE_PROCEDURE	ALTER_PROCEDURE	DROP_PROCEDURE
CREATE_QUEUE	ALTER_QUEUE	DROP_QUEUE
CREATE_REMOTE_SERVICE_BINDING	ALTER_REMOTE_SERVICE_BINDING	DROP_REMOTE_SERVICE_BINDING
CREATE_ROLE	ALTER_ROLE	DROP_ROLE

CREATE_ROUTE	ALTER_ROUTE	DROP_ROUTE
CREATE_SCHEMA	ALTER_SCHEMA	DROP_SCHEMA
CREATE_SERVICE	ALTER_SERVICE	DROP_SERVICE
CREATE_STATISTICS	DROP_STATISTICS	UPDATE_STATISTICS
CREATE_SYNONYM	DROP_SYNONYM	
CREATE_TABLE	ALTER_TABLE	DROP_TABLE
CREATE_TRIGGER	ALTER_TRIGGER	DROP_TRIGGER
CREATE_TYPE	DROP_TYPE	
CREATE_USER	ALTER_USER	DROP_USER
CREATE_VIEW	ALTER_VIEW	DROP_VIEW
CREATE_XML_SCHEMA_COLLECTION	ALTER_XML_SCHEMA_COLLECTION	DROP_XML_SCHEMA_COLLECTION

Sentencias DDL con alcance de servidor

ALTER_AUTHORIZATION_SERVER		
CREATE_ENDPOINT	DROP_ENDPOINT	
CREATE_LOGIN	ALTER_LOGIN	DROP_LOGIN
GRANT_SERVER	DENY_SERVER	REVOKE_SERVER

Grupos de eventos

El siguiente diagrama muestra los grupos de eventos que se pueden utilizar para disparar desencadenantes DDL. las sentencias Transact-SQL que cada grupo cubre, y su alcance.

El diagrama se ha tomado de la documentación del sistema: **SQL Server Books Online – Microsoft SQL Server 2005 CTP April 2005.**

	Server Scope	Database Scope
DDL_ENDPOINT_EVENTS (CREATE ENDPOINT, ALTER ENDPOINT, DROP ENDPOINT)	X	
DDL_SERVER_SECURITY_EVENTS	X	
DDL_LOGIN_EVENTS (CREATE LOGIN, ALTER LOGIN, DROP LOGIN)	X	
DDL_GDR_SERVER_EVENTS (GRANT SERVER, DENY SERVER, REVOKE SERVER)	X	
DDL_AUTHORIZATION_SERVER_EVENTS (ALTER AUTHORIZATION SERVER)	X	
DDL_DATABASE_LEVEL_EVENTS		X
DDL_TABLE_VIEW_EVENTS		X
DDL_TABLE_EVENTS (CREATE TABLE, ALTER TABLE, DROP TABLE)		X
DDL_VIEW_EVENTS (CREATE VIEW, ALTER VIEW, DROP VIEW)		X
DDL_INDEX_EVENTS (CREATE INDEX, ALTER INDEX, DROP INDEX)		X
DDL_STATISTICS_EVENTS (CREATE STATISTICS, UPDATE STATISTICS, DROP STATISTICS)		X
DDL_SYNONYM_EVENTS (CREATE SYNONYM, DROP SYNONYM)		X
DDL_FUNCTION_EVENTS (CREATE FUNCTION, ALTER FUNCTION, DROP FUNCTION)		X
DDL_PROCEDURE_EVENTS (CREATE PROCEDURE, ALTER PROCEDURE, DROP PROCEDURE)		X
DDL_TRIGGER_EVENTS (CREATE TRIGGER, ALTER TRIGGER, DROP TRIGGER)		X
DDL_EVENT_NOTIFICATION_EVENTS (CREATE EVENT NOTIFICATION, DROP EVENT NOTIFICATION)		X
DDL_ASSEMBLY_EVENTS (CREATE ASSEMBLY, ALTER ASSEMBLY, DROP ASSEMBLY)		X
DDL_TYPE_EVENTS (CREATE TYPE, DROP TYPE)		X
DDL_DATABASE_SECURITY_EVENTS		X
DDL_CERTIFICATE_EVENTS (CREATE CERTIFICATE, ALTER CERTIFICATE, DROP CERTIFICATE)		X
DDL_USER_EVENTS (CREATE USER, ALTER USER, DROP USER)		X
DDL_ROLE_EVENTS (CREATE ROLE, ALTER ROLE, DROP ROLE)		X
DDL_APPLICATION_ROLE_EVENTS (CREATE APPROLE, ALTER APPROLE, DROP APPROLE)		X
DDL_SCHEMA_EVENTS (CREATE SCHEMA, ALTER SCHEMA, DROP SCHEMA)		X
DDL_GDR_DATABASE_EVENTS (GRANT DATABASE, DENY DATABASE, REVOKE DATABASE)		X
DDL_AUTHORIZATION_DATABASE_EVENTS (ALTER AUTHORIZATION DATABASE)		X
DDL_SSB_EVENTS		X
DDL_MESSAGE_TYPE_EVENTS (CREATE MSGTYPE, ALTER MSGTYPE, DROP MSGTYPE)		X
DDL_CONTRACT_EVENTS (CREATE CONTRACT, DROP CONTRACT)		X
DDL_QUEUE_EVENTS (CREATE QUEUE, ALTER QUEUE, DROP QUEUE)		X
DDL_SERVICE_EVENTS (CREATE SERVICE, ALTER SERVICE, DROP SERVICE)		X
DDL_ROUTE_EVENTS (CREATE ROUTE, ALTER ROUTE, DROP ROUTE)		X
DDL_REMOTE_SERVICE_BINDING_EVENTS (CREATE REMOTE SERVICE BINDING, ALTER REMOTE SERVICE BINDING, DROP REMOTE SERVICE BINDING)		X
DDL_XML_SCHEMA_COLLECTION_EVENTS (CREATE XML SCHEMA COLLECTION, ALTER XML SCHEMA COLLECTION, DROP XML SCHEMA COLLECTION)		X
DDL_PARTITION_EVENTS		X
DDL_PARTITION_FUNCTION_EVENTS (CREATE PARTITION FUNCTION, ALTER PARTITION FUNCTION, DROP PARTITION FUNCTION)		X
DDL_PARTITION_SCHEME_EVENTS (CREATE PARTITION SCHEME, ALTER PARTITION SCHEME, DROP PARTITION SCHEME)		X

Mantenimiento de los desencadenantes

Modificación de un desencadenante

Para modificar un desencadenante sin tener que eliminarlo, ejecute la sentencia ALTER TRIGGER. La definición previa del desencadenante será reemplazada por la definición establecida en ALTER TRIGGER.

Sintaxis para desencadenante DML

```
ALTER TRIGGER nombreDesencadenante
ON nombreTabla | nombreVista
FOR [ INSERT ] [,] [ UPDATE ] [,] [ DELETE ]
    | AFTER [ INSERT ] [,] [ UPDATE ] [,] [ DELETE ]
    | INSTEAD OF [ INSERT ] [,] [ UPDATE ] [,] [ DELETE ]
AS
sentenciasSQL
```

Sintaxis para desencadenante DDL

```
ALTER TRIGGER nombreDesencadenante
ON ALL SERVER | ON DATABASE
FOR | AFTER eventoDDL, eventoDDL, ...
AS
sentenciasSQL
```

Eliminación de un desencadenante

El permiso para eliminar un desencadenante lo tiene el dueño de la tabla y no es transferible. Sin embargo, miembros de los roles **sysadmin** y **db_owner** pueden eliminar cualquier objeto especificando al dueño en la declaración DROP TRIGGER.

Sintaxis para desencadenante DML

```
DROP TRIGGER nombreEsquema.nombreDesencadenante
```

Sintaxis para desencadenante DDL

DROP TRIGGER nombreDesencadenante

```
DROP TRIGGER nombreDesencadenante  
ON DATABASE | ON ALL SERVER
```

- ON DATABASE ú ON ALL SERVER se debe especificar si fue especificado al momento de crear el desencadenante.

Deshabilitación/habilitación de un desencadenante

Un desencadenante se puede deshabilitar de modo tal que no se dispare cuando ocurrra el evento para el que fue programado. La deshabilitación no elimina el desencadenante, por lo que puede volver a ser habilitado.

Sintaxis

```
DISABLE TRIGGER | ENABLE TRIGGER  
nombreDesencadenante1, nombreDesencadenante2, ... | ALL  
ON nombreTablaóVista | DATABASE | ALL SERVER
```

Esta página se ha dejado en blanco intencionalmente.