Capítulo XIV

Desencadenantes DML

Los desencadenantes DML ó triggers DML son objetos que definen código a ejecutarse automáticamente del lado del servidor cuando una aplicación produce cambios en los datos.

Un desencadenante es un tipo especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en SQL Server.

Un desencadenante DML (Data Manipulation Language) se ejecuta cuando se produce un evento DML debido a la ejecución de instrucciones INSERT, UPDATE ó DELETE.

Un desencadenante DDL (Data Definition Language) se ejecuta cuando se produce un evento que produce una modificación en el esquema de una base de datos ó en el contenido del servidor por la ejecución de sentencias CREATE, ALTER ó DROP.

En este capítulo veremos los desencadenantes DML.

1. DESENCADENANTES DML

Tenga en cuenta lo siguiente cuando programa sus desencadenantes:

- 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.
- 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.
- 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 desencadenate. Si se violan las restricciones, el desencadenante no se ejecuta.
- Las tablas pueden tener múltiples desencadenantes para cualquier acción. 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.

1.1. 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.

1.2. Tipos de desencadenantes DML

Podemos programar los siguientes tipos de desencadenantes DML:

- **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.

2. 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.

Sintáxis

```
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.
- SQL Server no permite usar como parte del código del desencadenante algunas instrucciones que producen cambios en el esquema de la base de datos ó en el contenido del servidor de base de datos. Algunas de ellas son:
    - CREATE DATABASE, ALTER DATABASE y DROP DATABASE
    - CREATE INDEX, ALTER INDEX y DROP INDEX
    - ALTER TABLE

Ejercicio 14.1: Preparación del escenario para los ejemplos de desencadenantes DML

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, '31/10/2012',
      'Comercial Gómez', NULL )
INSERT INTO Factura
   VALUES( 2, '02/11/2012',
      '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 14.2: Creación de desencadenante AFTER INSERT

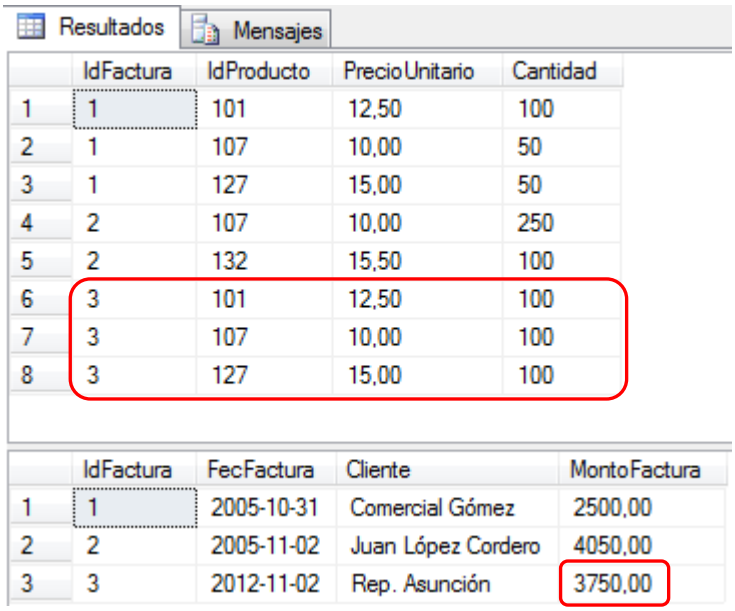
Crear un desencadenante AFTER INSERT que recalcule y actualice el monto de una factura cada vez que se inserta un detalle para dicha factura.

```
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 cada uno de los siguientes batchs:

```
-- Registrando la factura 3
SET DATEFORMAT DMY
go

INSERT INTO Factura
   VALUES( 3, '02/11/2012',
      '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
```



Resultados | Mensajes

	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	Comercial Gómez	2500,00
2	2	2005-11-02	Juan López Cordero	4050,00
3	3	2012-11-02	Rep. Asunción	3750,00

2.1. 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 14.3: Creación de desencadenante AFTER DELETE

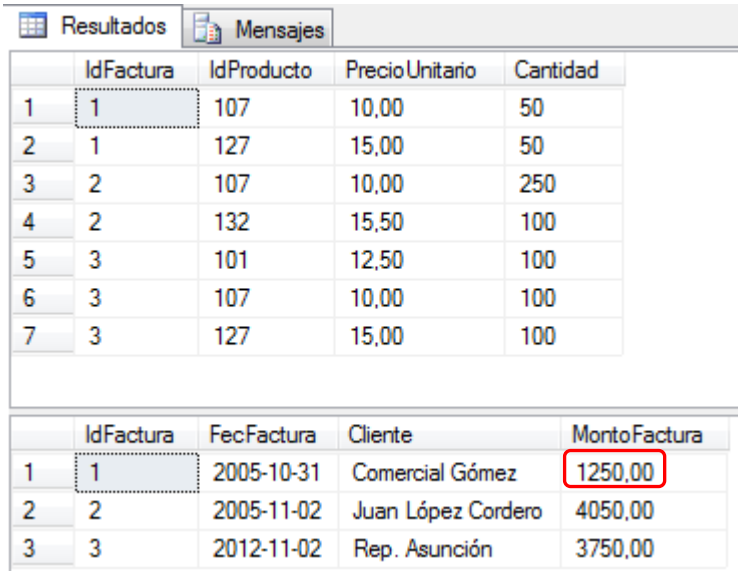
Crear un desencadenante AFTER DELETE que recalcule y actualice el monto de una factura cada vez que se elimina un detalle para dicha factura.

```
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
```



Resultados | Mensajes

	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	Comercial Gómez	1250,00
2	2	2005-11-02	Juan López Cordero	4050,00
3	3	2012-11-02	Rep. Asunción	3750,00

2.2. 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**
1	***101***	***12.50***	***100***

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
1	***31/10/2005***	***Comercial Gómez***	***1250.00***
2	02/11/2005	Juan López Cordero	4050.00
3	02/11/2005	Rep. Asunción	3750.00

Ejercicio 14.4: Creación de desencadenante AFTER UPDATE

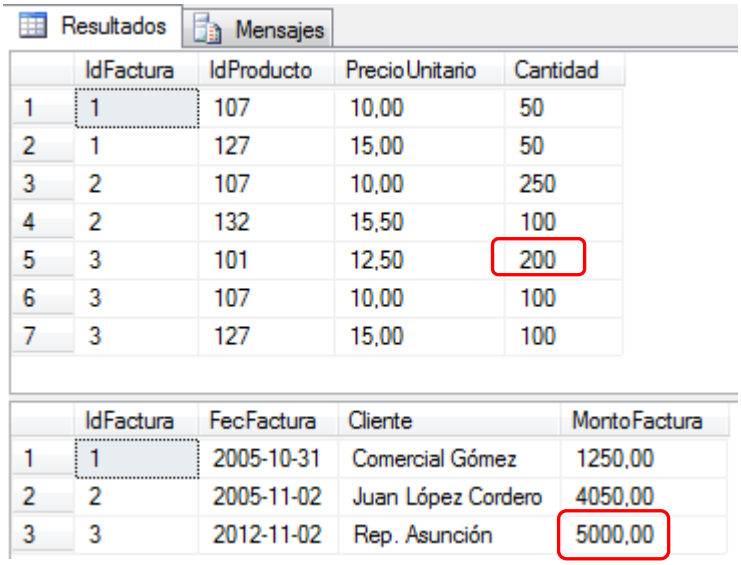
Crear un desencadenante AFTER UPDATE que recalcule el monto de una factura cada vez que se actualiza la cantidad ó el precio de un detalle de dicha factura.

```
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
```



Resultados | Mensajes

	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	Comercial Gómez	1250,00
2	2	2005-11-02	Juan López Cordero	4050,00
3	3	2012-11-02	Rep. Asunción	5000,00

2.3. 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 temporal es 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 14.5: Preparación del escenario para la creación y prueba de un desencadenante INSTEAD OF

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

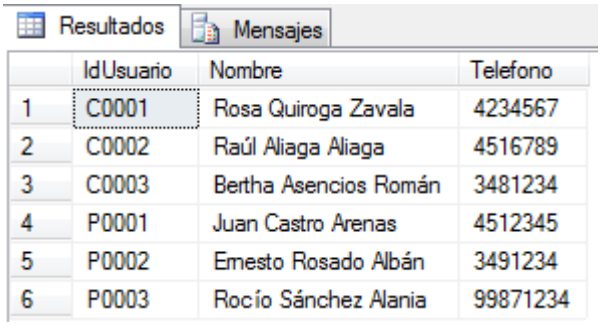
```
USE Test
go
```

```
-- 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
```



Resultados | Mensajes

	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

Ejercicio 14.6: Creación de un desencadenante INSTEAD OF

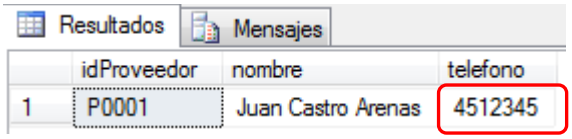
Creamos sobre la vista **v_Usuarios**, un desencadenante INSTEAD OF que cuando se actualiza el telefono de un usuario a través de la vista se actualiza el dato en la tabla correspondiente.

```
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
```

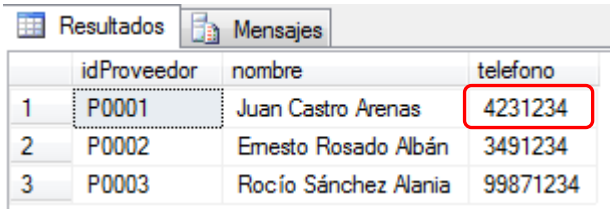
Probamos el desencadenante actualizando el teléfono del proveedor de código **P0001**.

```
-- Verificamos el teléfomo registrado para P0001
SELECT * FROM Proveedor
   WHERE idProveedor = 'P0001'
go
-- Su telefono es 4512345
```



	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	4231234
2	P0002	Ernesto Rosado Albán	3491234
3	P0003	Rocío Sánchez Alania	99871234

3. MODIFICACIÓN DE LA DEFINICIÓN 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.

Sintáxis

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

4. 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.

Sintáxis

```
DROP TRIGGER nombreEsquema.nombreDesencadenante
```

5. OBTENER LA METADATA DE UN DSENCADENANTE DML

Ejercicio 14.7: Obtener la definición de un desencadenante DML

Ejecute las siguientes instrucciones para obtener la definición del desencadenante **tg_Update_DetalleFactura**.

```
USE Test
go

-- Ver la definición de un desencadenante
SELECT definition
```

```
FROM sys.sql_modules
WHERE object_id =
     OBJECT_ID('tg_Update_DetalleFactura')
go

SELECT OBJECT_DEFINITION (
     OBJECT_ID('tg_Update_DetalleFactura'))
go

sp_helptext 'tg_Update_DetalleFactura'
go
```

Resultados | Mensajes

	Text
1	
2	CREATE TRIGGER tg_update_DetalleFactura
3	ON DetalleFactura AFTER UPDATE
4	AS
5	IF UPDATE(precioUnitario) OR UPDATE(cantidad)
6	BEGIN
7	DECLARE @factura int
8	DECLARE @suma money
9	SET @factura = (SELECT idFactura FROM inserted)
10	SET @suma =
11	(SELECT SUM(PrecioUnitario * Cantidad)
12	FROM DetalleFactura
13	WHERE DetalleFactura.idFactura = @factura)
14	UPDATE factura
15	SET MontoFactura = @suma
16	WHERE idFactura = @factura
17	END

Ejercicio 14.8: Ver las dependencias de un desencadenante DML

Ejecute la siguiente consulta para ver los objetos que tienen dependencia con el desencadenante **tg_Update_DetalleFactura**.

```
USE Test
go

SELECT OBJECT_NAME(referencing_id)
          AS nombreDesencadenante,
```

```
    o.type_desc AS tipo,
    referenced_entity_name AS tablaReferenciada
FROM sys.sql_expression_dependencies AS sed
     INNER JOIN sys.objects AS o
     ON sed.referencing_id = o.object_id
WHERE referencing_id =
          OBJECT_ID('tg_Update_DetalleFactura')
go
```

Resultados | Mensajes

	nombreDesencadenante	tipo	tablaReferenciada
1	tg_update_DetalleFactura	SQL_TRIGGER	DetalleFactura
2	tg_update_DetalleFactura	SQL_TRIGGER	factura

Ejercicio 14.9: Desencadenantes presentes en la base de datos

```
USE Test
go

SELECT  name, parent_id, create_date,
     modify_date, is_instead_of_trigger
FROM sys.triggers
WHERE type = 'TR'
go
```

Resultados | Mensajes

	name	parent_id	create_date	modify_date	is_instead_of_trigger
1	tg_insert_DetalleFactura	293576084	2013-06-16 14:33:10.830	2013-06-16 14:33:10.830	0
2	tg_delete_DetalleFactura	293576084	2013-06-16 14:47:34.037	2013-06-16 14:47:34.037	0
3	tg_update_DetalleFactura	293576084	2013-06-16 14:58:13.040	2013-06-16 14:58:13.040	0
4	tg_update_v_Usuarios	453576654	2013-06-16 15:13:27.263	2013-06-16 15:13:27.263	1

```
SELECT  name, object_id, schema_id,
     parent_object_id, type_desc, create_date,
     modify_date, is_published
FROM sys.objects
WHERE type = 'TR'
go
```

Resultados | Mensajes

	name	object_id	schema_id	parent_object_id	type_desc	create_date	modify_date	is_published
1	tg_insert_DetalleFactura	341576255	1	293576084	SQL_TRIGGER	2013-06-16 14:33:10.830	2013-06-16 14:33:10.830	0
2	tg_delete_DetalleFactura	357576312	1	293576084	SQL_TRIGGER	2013-06-16 14:47:34.037	2013-06-16 14:47:34.037	0
3	tg_update_DetalleFactura	373576369	1	293576084	SQL_TRIGGER	2013-06-16 14:58:13.040	2013-06-16 14:58:13.040	0
4	tg_update_v_Usuarios	469576711	1	453576654	SQL_TRIGGER	2013-06-16 15:13:27.263	2013-06-16 15:13:27.263	0

Ejercicio 14.10: Eventos que disparan un desencadenante DML

Para ver qué acción dispara el desencadenante tg_Update_DetalleFactura ejecute la siguiente consulta:

```
USE Test
go

SELECT object_id, type, type_desc,
     is_trigger_event, event_group_type,
     event_group_type_desc
FROM sys.events
WHERE object_id =
          OBJECT_ID('tg_Update_DetalleFactura')
go
```

Resultados | Mensajes

	object_id	type	type_desc	is_trigger_event	event_group_type	event_group_type_desc
1	373576369	2	UPDATE	1	NULL	NULL

6. EJERCICIOS PROPUESTOS

1. Para la base de datos Test de este capítulo, escriba un desencadenante que envíe un mensaje a la aplicación cliente según las siguientes especificaciones:
    - Si se intenta eliminar una factura, y el intento de eliminación genera un error, el mensaje debe decir: 'Intento de eliminación de factura causó error'
    - Si se produce la eliminación de la factura, el mensaje debe decir: 'Se ha eliminado la factura'
2. Para la base de datos **Test**, elabore un desencadenante de modo tal que si se elimina un detalle de una factura, y éste detalle es el único que tenía la factura, se elimine automáticamente la factura.