IES CHAN DO MONTE
C.S. de Desarrollo de Aplicaciones Multiplataforma
Módulo Base de datos

UNIDAD 9: DESENCADENADORES

Índice

1. ¿Qué es un desencadenador?

Microsoft SQL Server proporciona dos mecanismos principales para exigir las reglas de negocios y la integridad de los datos: **restricciones** y **desencadenadores**.

Un desencadenador es un tipo especial de procedimiento almacenado que se invoca automáticamente cuando se ejecuta un evento de lenguaje.

SQL **Server incluye dos tipos generales de desencadenadores**: **DML** y **DDL.**

✓ **Los desencadenadores DML** :

- Se invocan cuando un **evento de lenguaje de manipulación de datos (DML)** tiene lugar en la base de datos.
- Los eventos DML incluyen instrucciones **INSERT, UPDATE o DELETE que** modifican datos en una tabla o vista especificada.
- Pueden **consultar otras tablas e incluir instrucciones Transact-SQL complejas**.
- El **desencadenador y la instrucción que lo activa** se tratan como **una sola transacción**, que puede revertirse desde el desencadenador.

  Si se detecta un error grave (por ejemplo, no hay suficiente espacio en disco), se revierte automáticamente toda la transacción.

✓ **Desencadenadores DDL:**

- Aparecieron a partir de SQL Server 2005.
- Se invocan cuando un **evento de lenguaje de definición de datos (DDL**) tiene lugar en el servidor o la base de datos.
- Estos eventos corresponden principalmente a instrucciones Transact-SQL que comienzan por las palabras clave **CREATE, ALTER y DROP**. Determinados procedimientos almacenados del sistema que realizan operaciones de estilo DDL también pueden activar desencadenadores DDL.
- Pueden utilizarse para tareas administrativas como auditar y regular las operaciones de base de datos como, por ejemplo:

  Evitar determinados cambios en el esquema de base de datos.

  Hacer ocurra algún evento en la base de datos como respuesta a un cambio realizado en el esquema de base de datos.

  Registrar cambios o eventos del esquema de base de datos.

2. Desencadenadores DML

2.1 Características de los desencadenadores

- **Asociado con una tabla**

  Se definen en **una tabla específica**.

- **Automáticamente invocado**

  Cuando se realiza un **intento de introducir, actualizar o eliminar los datos en una tabla** y se ha definido un desencadenador en la tabla para esa acción en concreto, el desencadenador se ejecuta automáticamente.

  Un desencadenador no se puede evitar.

- **No pueden ser llamados directamente**

  Al contrario que los procedimientos almacenados estándar, **los desencadenadores no pueden ser llamados directamente** y **no aceptan parámetros**.

- **Es una transacción**

  El d**esencadenador y la instrucción que hace que se dispare se consideran como una única transacción** que puede ser **deshecha desde cualquier lugar dentro del desencadenador**.

  Las definiciones de desencadenador pueden incluir una instrucción **ROLLBACK TRANSACTION** incluso si no existe una instrucción BEGIN TRANSACTION explícita. La instrucción que causa que se dispare el desencadenador, inicia una transacción implícita si no se ha ejecutado una instrucción BEGIN TRANSACTION explícita.

2.2 Utilidades de los desencadenadores

Los desencadenadores se utilizan:

- Para **mantener una integridad de datos**, no para devolver resultados de consulta.
- El beneficio principal es que pueden **contener una lógica de procesado compleja**.

Sólo se debería utilizar desencadenadores cuando las restricciones no proporcionen la funcionalidad que necesita.

Razones comunes para utilizar desencadenadores:

1. **Mantener la integridad de los datos**: se utilizan para mantener la integridad de los datos y no para devolver resultados de consulta.

   Pueden contener una lógica de procesamiento compleja y se utilizan cuando las restricciones no proporcionan la funcionalidad necesaria.
2. **Modificaciones en cascada**: Pueden asegurar que las modificaciones en cascada a través de tablas relacionadas en una base de datos tengan lugar a la vez.
3. **Aplicar integridad de datos más compleja**: Los desencadenadores pueden proteger contra operaciones INSERT, UPDATE y DELETE incorrectas o dañinas, y exigir otras restricciones que sean más complejas que las definidas con restricciones CHECK.
4. **Producción de mensajes de error personalizados**: Si tu aplicación requiere mensajes personalizados y un control de errores más complejo, debes utilizar un desencadenador.
5. **Mantener datos derivados**: Pueden ayudar a mantener datos derivados (como ventas del año hasta la fecha)
6. **Comparar o registrar antes y después de estados de datos**: Pueden evaluar el estado de una tabla antes y después de realizar una modificación de datos y actuar en función de esa diferencia.
7. **Varios desencadenadores DML del mismo tipo**: Varios desencadenadores DML del mismo tipo (INSERT, UPDATE o DELETE) en una tabla permiten realizar distintas acciones en respuesta a una misma instrucción de modificación.

2.3 Comparación de los desencadenadores y las restricciones

Tanto las **restricciones como los desencadenadores DML ofrecen ventajas específicas** que resultan útiles en determinadas situaciones.

La **principal ventaja de los desencadenadores DML** consiste en que pueden **contener una lógica de proceso compleja** que utilice código de Transact-SQL.

Por tanto, **los desencadenadores DML permiten toda la funcionalidad de las restricciones**; sin embargo, **no son siempre el mejor método** para realizar una determinada característica.

Restricciones	Desencadenadores DML
Mantienen la **integridad de las entidades** a través de índices que forman parte de las restricciones **PRIMARY KEY y UNIQUE**.	Contienen una **lógica de proceso compleja** que utiliza código de Transact-SQL.
Mantienen la **integridad del dominio** mediante restricciones **CHECK.**	Permiten **toda la funcionalidad de las restricciones**.
Mantienen la **integridad referencial** mediante restricciones **FOREIGN KEY**.	Son útiles cuando las **características permitidas por las restricciones no cubren las necesidades funcionales** de la aplicación.
Las restricciones **FOREIGN KEY sólo pueden validar un valor de columna si coincide exactamente con un valor de otra columna**.	Pueden realizar **cambios en cascada mediante tablas relacionadas** de la base de datos.
Sólo pueden comunicar **la existencia de errores mediante mensajes de error estándar** del sistema.	Pueden **comunicar mensajes personalizados** y un **control de errores más complejo**.
No pueden revertir transacciones.	**Pueden revertir transacciones que infrinjan la integridad referencial.**
Si hay **restricciones en la tabla**, se comprobarán **después de la ejecución de la instrucción SQL**.	Si hay **restricciones en la tabla de desencadenadores**, se comprobarán **después de la ejecución del desencadenador INSTEAD OF**, pero **antes de la ejecución del desencadenador AFTER**.

2.4 Las tablas inserted y deleted

Hay **dos tablas especiales disponibles** dentro de los desencadenadores: la tabla **deleted** y la tabla **inserted**.

SQL Server **crea y administra automáticamente ambas tablas**.

- **Tabla deleted**: Almacena **copias de filas afectadas por las instrucciones DELETE y UPDATE**.

  Cuando se ejecuta una instrucción DELETE o UPDATE, las filas se eliminan de la tabla trigger y se transfieren a la tabla deleted.

- **Tabla inserted**: Almacena **copias de las filas afectadas por las instrucciones INSERT y UPDATE**.

  Cuando se ejecuta una instrucción INSERT o UPDATE, se agregan filas simultáneamente a la tabla inserted y a la tabla trigger.

- **No existe una tabla UPDATED.** Hacer **una actualización** es lo mismo que **borrar (deleted) e insertar los nuevos (inserted)**.

  La sentencia **UPDATE es la única en la que inserted y deleted tienen datos simultáneamente**

Uso de las tablas:

Se pueden utilizar las filas de las tablas deleted e inserted para hacer referencia a filas en tablas relacionadas o para probar valores en las filas que están siendo eliminadas o introducidas.

Restricciones:

No se puede alterar los datos de las tablas deleted o inserted directamente ni realizar en ellas operaciones de lenguaje de definición de datos (DDL), como CREATE INDEX.

ROLLBACK TRANSACTION:

Borra todas las modificaciones de datos realizadas desde el inicio de la operación o de un punto de retorno.

También **libera los recursos mantenidos** por la transacción.

Ejemplo:

- Creación de un trigger para que ***no permita insertar más departamentos*** en la base de datos compañía.
- Dentro de este trigger se visualiza el contenido de la tabla INSERTED.-
- En este caso se va utilizar **ROLLBACK TRANSACTION pues para desechar la inserción** del registro ya que el desencadenador y la instrucción que hace que se dispare se consideran como una única transacción que puede ser deshecha desde cualquier lugar dentro del desencadenador.
- ROLLBACK TRANSACTION: Borra todas las modificaciones de datos realizadas desde el inicio de la operación o de un punto de retorno. **También libera los recursos mantenidos por la transacción**.

```
USE CompañiaConsultas
SET NOCOUNT ON
IF EXISTS (SELECT name FROM sys.objects
                WHERE name = 'TR_NOINSERCCION_Departamento'
        AND type = 'TR')
  DROP TRIGGER  TR_NOINSERCCION_Departamento
GO
--Se crea un trigger de tipo AFTER, la operación de inserción que desencadena el trigger se realiza antes
CREATE TRIGGER TR_NOINSERCCION_Departamento
ON Departamentos
FOR INSERT
AS
--Visualizo el contenido de la tabla inserted donde aparece la información del registro a insertar
      SELECT * FROM INSERTED
      PRINT 'NO SE VA A REALIAZAR LA OPERACION DE INSERCCION
          DE UN NUEVO DEPARTAMENTO'
  --Se deshacen todos los cambios, es decir, se revierte la insercción
```

```
--NOTA:Como el desencadenador y la instrucción que hace que se dispare se consideran como una
-- única transacción que puede ser deshecha desde cualquier lugar dentro del desencadenador.
      ROLLBACK TRANSACTION
    GO
   INSERT Departamentos
   VALUES(11, 'Recursos Humanos',1, '11111111',getdate()-2)
   SELECT * FROM   Departamentos
```

Ejemplo de deshacer la inserción con una instrucción delete.

```
   USE CompañiaConsultas
   SET NOCOUNT ON
   IF EXISTS (SELECT name FROM sys.objects
                    WHERE name = 'TR_NOINSERCCION_Departamento'          AND type = 'TR')
     DROP TRIGGER  TR_NOINSERCCION_Departamento
   GO
   --Se crea un trigger de tipo AFTER, la operacion de inserccion que desencadena el trigger se realiza antes
   CREATE TRIGGER TR_NOINSERCCION_Departamento
   ON Departamentos
   FOR INSERT
   AS
--Visualizo el contenido de la tabla inserted donde aparece la información del registro a insertar
        SELECT * FROM INSERTED
         PRINT 'NO SE VA A REALIAZAR LA OPERACION DE INSERCCION
              DE UN NUEVO DEPARTAMENTO'
   --Se borra el registro insertado.
      DELETE FROM departamentos
      WHERE NumDep IN  (SELECT Numdep FROM INSERTED)
    GO

   INSERT Departamentos
   VALUES(11, 'Recursos Humanos',1, '11111111',getdate()-2)
   SELECT * FROM   Departamentos
```

2.5 Tipos de desencadenadores DML

Hay dos tipos de desencadenadores: los **AFTER** y los **INSTEAD OF**. Puede aplicarse a cualquier subconjunto de las tres acciones de usuario (**INSERT, UPDATE, DELETE**).

- **Desencadenadores AFTER o FOR**:
  - Se ejecutan **después de llevar a cabo la acción** de las instrucciones INSERT, UPDATE o DELETE. La especificación de AFTER produce el mismo efecto que especificar FOR.
  - El desencadenador AFTER sólo puede especificarse en tabla y pueden tener varios desencadenadores AFTER de un mismo tipo, siempre que tengan nombres distintos
- **Desencadenadores INSTEAD OF**:
  - Se ejecutan **en lugar de la acción habitual** de desencadenamiento.
  - También se pueden definir desencadenadores INSTEAD OF en vistas con una o más tablas y en una tabla.
  - **Una tabla sólo puede tener un desencadenador INSTEAD OF de un tipo determinado.**

Característica	Desencadenadores AFTER(FOR)	Desencadenadores INSTEAD OF
Cuando se ejecutan	**Después de las instrucciones** INSERT, UPDATE o DELETE.	**En lugar de la acción habitual** de desencadenamiento.
Dónde se pueden especificar	Sólo en tablas.	En tablas y en vistas con una o más tablas base.
Relación con las restricciones	**No se ejecutan si se produce una infracción de restricción**.	**Se ejecutan antes que las restricciones, permitiendo tareas de procesamiento** previo que complementan a las acciones de restricción.
Recursividad	No aplicable.	Si un desencadenador INSTEAD OF ejecuta una instrucción que normalmente activaría el mismo desencadenador, este no se llama de forma recursiva.

Ejemplo: **Creación de un trigger INSTEAD OF** para no permitir inserciones en la tabla departamentos

Este desencadenador se activará **en lugar de la operación de inserción** estándar en la tabla Departamentos. Cuando se intenta insertar un nuevo registro, el desencadenador se activa y muestra el contenido del registro que se intentó insertar (usando la tabla inserted), pero no realiza la inserción. En su lugar, imprime un mensaje que indica que no se realizará la operación de inserción.

```
USE CompañiaConsultas
SET NOCOUNT ON
IF EXISTS (SELECT name FROM sys.objects
      WHERE name = 'TR_NOINSERCCION_Departamento'
        AND type = 'TR')
  DROP TRIGGER  TR_NOINSERCCION_Departamento
GO
--Se crea un trigger de tipo INSTEAD OF, la operacion de inserccion que desencadena el trigger no se realiza
--si no que se ejecuta  el trigger
CREATE TRIGGER TR_NOINSERCCION_Departamento
ON Departamentos
INSTEAD OF INSERT
AS
--Visualizo el contenido de la tabla inserted donde aparece la información  del registro a insertar
      SELECT * FROM inserted
      PRINT 'NO SE VA A REALIAZAR LA OPERACION DE INSERCCION
        DE UN NUEVO DEPARTAMENTO'
 --No hay que hacer ninguna acción por que no se ha insertado el registro en la tabla departamentos

GO
INSERT Departamentos
VALUES(11, 'Recursos Humanos',1, '11111111',getdate()-2)
SELECT * FROM   Departamentos
```

Ejemplo: Ejemplo de un desencadenador **INSTEAD OF**, donde si el registro que está en la tabla inserted cumple la condición (año de inicio del gerente es superior a 2002), entonces se inserta en la tabla departamentos.

Este desencadenador se activa en lugar de la operación de inserción estándar en la tabla Departamentos. Cuando se intenta insertar un nuevo registro, el desencadenador verifica si el año de inicio del gerente es superior a 2002. Si es así, se realiza la inserción; de lo contrario, se imprime un mensaje y no se realiza ninguna inserción.

```
USE CompañiaConsultas
SET NOCOUNT ON
IF EXISTS (SELECT name FROM sys.objects
                WHERE name = 'TR_NOINSERCCION_Departamento'   AND type = 'TR')
  DROP TRIGGER  TR_NOINSERCCION_Departamento
GO
--Se crea un trigger de tipo INSTEAD OF, la operacion de inserccion que desencadena el trigger no se realiza
--si no que se ejecuta el trigger
CREATE TRIGGER TR_NOINSERCCION_Departamento
ON Departamentos
INSTEAD OF INSERT
AS
  --Si el año de inicio del gerente es superior al 2002 entonces se inserta este registro en la tabla
--departamentos por que es correcto


  IF 2002<=(SELECT year(FechaInicGerente) FROM INSERTED  )
    INSERT INTO DEPARTAMENTOS
     SELECT *  FROM INSERTED
--En este insert no se activa el desencadenador, si lo hiciera estaría siempre ejecutándose de forma
 --recursiva
  ELSE
      PRINT 'NO SE VA A REALIAZAR LA OPERACION DE INSERCCION
        DE UN NUEVO DEPARTAMENTO'
   --No hay que hacer ninguna acción por que no se ha insertado el registro en la tabla departamentos
    GO
INSERT Departamentos
VALUES(11, 'Recursos Humanos',1, '11111111',getdate())
SELECT * FROM   Departamentos
```

3. Cómo crear desencadenadores

- **Creación del desencadenador DML:**
  - La instrucción CREATE TRIGGER **debe ser la primera del lote**.
  - Los nombres de los desencadenadores deben seguir las reglas de los identificadores en SQL Server.
  - Los desencadenadores DML se crean en la base de datos actual.
  - No se pueden crear desencadenadores DML en tablas temporales o del sistema.
  - Limitaciones específicas, como no poder definir desencadenadores INSTEAD OF DELETE o INSTEAD OF UPDATE en tablas con claves externas con acciones DELETE o UPDATE.

- **Especificaciones al crear el desencadenador:**
  - Se debe especificar **el nombre del desencadenador**.
  - **La tabla** en la que se define el desencadenador.
  - El momento de activación del desencadenador (**AFTER o INSTEAD OF**).
  - Las **instrucciones de modificación de datos que activarán el desencadenador** (INSERT, UPDATE, DELETE).
  - Las instrucciones de programación que realizan la acción desencadenadora.

Sintaxis:

```
CREATE TRIGGER [ esquema . ] nombre_desencadenador
ON { tabla | vista }
[ WITH [ ENCRYPTION ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS { instrucción_sql [ ; ] [ ,...n ] }
```

- **Esquema**: Es el nombre del esquema al que pertenece un desencadenador DML. Los desencadenadores DML se limitan al esquema de la tabla o vista en la que se crearon.
- **Nombre del desencadenador**: Debe cumplir con las reglas de los identificadores, excepto que no puede comenzar con los símbolos # o ##.
- **Tabla | Vista**: Es la tabla o vista en la que se ejecuta el desencadenador DML. No se pueden definir desencadenadores DML en tablas temporales locales o globales.
- **FOR | AFTER**: Especifica que el desencadenador sólo se activa cuando todas las operaciones especificadas en la instrucción SQL desencadenadora se han ejecutado correctamente.
- **INSTEAD OF**: Especifica que se ejecuta el desencadenador DML en lugar de la instrucción SQL desencadenadora.
- **{ [DELETE] [,] [INSERT] [,] [UPDATE] }:** Especifica las instrucciones de modificación de datos que activan el desencadenador DML.
- **Instrucción SQL**: Son las condiciones y acciones del desencadenador. Las condiciones del desencadenador especifican los criterios adicionales que determinan si los intentos de los eventos DML hacen que se lleven a cabo las acciones del desencadenador.
- **Tablas lógicas deleted e inserted**: Los desencadenadores DML usan estas tablas que guardan los valores antiguos o nuevos de las filas que la acción del usuario puede cambiar.
- **Restricciones:** No se pueden utilizar ciertas instrucciones en un desencadenador DML, como ALTER DATABASE, CREATE DATABASE, DROP DATABASE, LOAD DATABASE, LOAD LOG, RECONFIGURE, RESTORE DATABASE, RESTORE LOG.

3.1 Cómo ver información acerca de los desencadenadores

Puede utilizar los procedimientos almacenados de sistema de la siguiente tabla para encontrar información adicional acerca de los desencadenadores.

- **sp_helptext nombre_desencadenador:** Muestra el texto del desencadenador especificado si éste no está cifrado.
- **sp_depends nombre_desencadenador**: Lista los objetos que están referenciados por un desencadenador.
- **sp_helptrigger nombre_tabla**: Devuelve una lista de los desencadenadores definidos en la tabla especificada.

4. Cómo modificar y eliminar desencadenadores

Puede modificar o eliminar un desencadenador existente. También puede deshabilitar temporalmente un desencadenador y después habilitarlo de nuevo.

4.1 Cómo modificar un desencadenador

Si debe cambiar la definición de un desencadenador existente, puede modificarlo sin tener que eliminarlo.

La definición modificada reemplaza la definición del desencadenador existente con la nueva definición. La acción del desencadenador también puede ser modificada. Por ejemplo, si crea un desencadenador para INSERT y después modifica el desencadenador para UPDATE, el desencadenador ya no se dispara cuando se introducen filas en la tabla, sino que se dispara cuando se actualizan las filas de la tabla.

Sintaxis

```
ALTER TRIGGER nombre_desencadenador
ON tabla [WITH ENCRYPTION]
{FOR {[INSERT][,][UPDATE][,][DELETE]}
AS
instrucción_sql [ ,... nl]}
```

4.2 Cómo habilitar o deshabilitar un desencadenador

Puede habilitar o deshabilitar un desencadenador específico o todos los desencadenadores en una tabla. Cuando se deshabilita un desencadenador, todavía está definido para la tabla; sin embargo, cuando se ejecuta una instrucción INSERT, UPDATE o DELETE en la tabla el desencadenador no se dispara.

Utilice la instrucción ALTER TABLE para habilitar o deshabilitar desencadenadores específicos o todos los desencadenadores en una tabla.

Sintaxis parcial

```
ALTER TABLE tabla
{ENABLE | DISABLE} TRIGGER
{ALL | trigger_name[,...n]}
```

4.3 Cómo eliminar un desencadenador

Puede quitar un desencadenador eliminándolo. Los desencadenadores se eliminan automáticamente si se elimina su tabla asociada.

El permiso para eliminar un desencadenador se establece de manera predeterminada en el propietario de la tabla y no es transferible. Los miembros de las funciones sysadmin, db_owner y db_ddladmin pueden eliminar cualquier objeto especificando el propietario en la instrucción DROP TRIGGER.

Sintaxis

```
DROP TRIGGER {desencadenador} [,... n]
```

5. Ejemplos de desencadenadores

Ejemplo1: Ejemplo de creación de un desencadenador AFTER para que **no se permita tener asignado en un proyecto a más de tres empleados**

```
SET NOCOUNT ON
IF EXISTS (SELECT name FROM sysobjects
                WHERE name = 'NumEmpleadosPorProyecto' AND type = 'TR')
  DROP TRIGGER NumEmpleadosPorProyecto
GO
CREATE TRIGGER NumEmpleadosPorProyecto
ON Participacion
FOR INSERT, UPDATE
AS
      IF EXISTS (SELECT * FROM Participacion
              WHERE NumProy IN (SELECT inserted.Numproy
                                FROM inserted)
              GROUP BY Numproy
              HAVING COUNT(*) > 3)
      BEGIN
     RAISERROR( 'En un proyecto no pueden participar más de 3 empleados', 16, 1)
      ROLLBACK TRANSACTION
    END
GO
INSERT Participacion
VALUES('44444444', 1, 40)
SELECT * FROM Participacion
```

Otra manera de comprobar que un proyecto no tenga más de tres empleados asignados

```
SET NOCOUNT ON
IF EXISTS (SELECT name FROM sysobjects
                WHERE name = 'NumEmpleadosPorProyecto' AND type = 'TR')
  DROP TRIGGER NumEmpleadosPorProyecto
GO
CREATE TRIGGER NumEmpleadosPorProyecto
ON Participacion
FOR INSERT, UPDATE
AS
      IF (SELECT count(*) FROM Participacion
     WHERE Numproy IN (SELECT Numproy FROM Inserted))>3
      BEGIN
        RAISERROR( 'En un proyecto no pueden participar más de 3 empleados',  16, 1)
         ROLLBACK TRANSACTION
      END
GO
INSERT Participacion
VALUES('44444444', 1, 40)
SELECT * FROM Participacion
--ERROR! Y NO SE INSERTARÍA POR QUE EL PROYECTO 1 TIENE YA 3 EMPLEADOS
```

Ejemplo 2: Vamos a realizar lo mismo pero utilizando un trigger **INSTEAD OF**. En estos tipos de desencadenadores solo se puede poner una acción. **Solo se puede definir un desencadenador INSTEAD OF por cada instrucción INSERT, UPDATE o DELETE en cada tabla o vista**

```
SET NOCOUNT ON
IF EXISTS (SELECT name FROM sysobjects
                WHERE name = 'NumEmpleadosPorProyecto' AND type = 'TR')
  DROP TRIGGER NumEmpleadosPorProyecto
GO
CREATE TRIGGER NumEmpleadosPorProyecto
ON Participacion
INSTEAD OF INSERT
AS
      IF  EXISTS (SELECT * FROM Participacion
            WHERE NumProy IN (SELECT inserted.Numproy FROM inserted)
            GROUP BY Numproy
            HAVING COUNT(*) =3)
      BEGIN
       PRINT ' NO SE INSERTA. UN PROYECTO NO PUEDE TENER MÁS DE 3    EMPLEADOS'
   END
   ELSE
 --Tenemos que insertar en la tabla participacion la insercción cuando se cumple la condición
     INSERT INTO PARTICIPACION
      SELECT *  FROM INSERTED
 GO
--No se inserta por que el proyecto 1 ya tiene 3 participantes
INSERT Participacion
VALUES('44444444', 1, 40)
--Si que se inserta por que el proyecto 2 solo tiene 2 participantes
INSERT Participacion
VALUES('44444444', 2, 40)
```

Ejemplo 3: Tenemos creada una tabla llamada HistoricoJefesDepartamentos, donde se irá guardando la información de los directores antiguos cuando se cambia el director del departamento. Sus campos son NumDep, NSSGerente, FechaInicGerente, NombreDep , FechaFinGerente. Para ello creamos un trigger para la acción UPDATE. En la **fechaFinGerente guardamos la fecha actual** y en la **fecha de inicio del nuevo director también ponemos la fecha actual**.

```
USE CompañiaConsultas
SET NOCOUNT ON
IF EXISTS (SELECT * FROM sys.objects
                WHERE name = 'ModificarDirector' AND type = 'TR')
  DROP TRIGGER  ModificarDirector
GO
CREATE TRIGGER ModificarDirector
ON Departamentos
FOR UPDATE
AS
--Se inserta los datos que estan en DELETED en el histórico
    insert  HistoricoJefesDepartamentos
    select Numdep,NSSGerente,FechaInicGerente, NombreDep,getdate()  from deleted
    Print 'SE ha modificado el director y sus datos están en el  historico'
--Se actualiza la fechaInicGerente por la actual de la tabla departamentos  de aquellos departamentos
-- que se ha modificado  el  director y que están en INSERTED
--El trigger es de tipo AFTER or lo que se realiza primero la -modificacion y despues se ejecuta el trigger.
```

```
    update departamentos
    set FechainicGerente=getdate()
    where numdep in (select numdep from inserted)
go
```

Situación antes de ejecutarse el trigger.

```
NumDep NombreDep           NumEmpdep NSSGerente FechaInicGerente
------ ------------------------------------------------------------ --------- ---------- -----------
1      Finanzas                20        11111111   2001-02-16 23:25:00
2      Marketing               15        11111111   1995-05-05 00:00:00
3      Informática             4         22222222   1998-12-07 00:00:00
4      Procesos                17        44444444   2002-11-08 00:00:00
```

Ahora modificamos el director del departamento 1.

```
select * from departamentos
update Departamentos
set NSSGerente='44444444'   where Numdep=1
```

```
NumDep NombreDep           NumEmpdep NSSGerente FechaInicGerente
------ ------------------------------------------------------------ --------- ---------- -----------
1      Finanzas                20        44444444   2011-03-16 13:15:00
2      Marketing               15        11111111   1995-05-05 00:00:00
3      Informática             4         22222222   1998-12-07 00:00:00
4      Procesos                17        44444444   2002-11-08 00:00:00
```

Y el histórico tendrá:

```
NumDep NSSGerente FechaInicGerente        NombreDep             FechaFinGerente
------ ---------- ---------------------- ------------------------------------------------------------
1      11111111   2011-03-16 23:25:00    Finanzas              2011-03-16 13:13:00
```

NOTA: Si NSSGerente tiene una **restrición de clave foránea en la tabla empleados,** si al modificar se infrige esta regla, entonces *el trigger no se ejecuta por que es de tipo AFTER.*

Ejemplo, suponemos que **no existe el empleado 44447777 en la tabla empleados** e incumple la restricción de clave foranea.Si lo insertamos:

```
select * from departamentos
update Departamentos
set NSSGerente='44447777'  where Numdep=1
```

```
Mens. 547, Nivel 16, Estado 0, Línea 2
Instrucción UPDATE en conflicto con la restricción FOREIGN KEY
"FK_DEPARTAMENTOS_EMPLEADOS". El conflicto ha aparecido en la base de datos
"CompañiaConsultas", tabla "dbo.EMPLEADOS", column 'NSS'.
Se terminó la instrucción.
```

```
USE CompañiaConsultas
SET NOCOUNT ON
IF EXISTS (SELECT *
            FROM sys.objects
             WHERE name = 'ModificarDirector' AND type = 'TR')
  DROP TRIGGER  ModificarDirector
GO
```

```
CREATE TRIGGER ModificarDirector
ON Departamentos
INSTEAD OF UPDATE
AS
--Se inserta los datos que estan en DELETED en el historico
   insert  HistoricoJefesDepartamentos
   select Numdep,NSSGerente,FechaInicGerente, NombreDep,getdate()    from deleted
   Print 'SE ha modificado el director y sus datos están en el historico'
 --Se actualiza el campo NSSGerente y la fechaInicGerente por la actual de la tabla departamentos  de
--aquellos departamentos que se ha modificado  el director y que están en INSERTED
select * from inserted
update departamentos
 set NSSGerente=(select NSSGerente from inserted), FechainicGerente=getdate()
 where numdep in (select numdep from inserted)
```

Situación antes de ejecutarse el trigger.

```
NumDep NombreDep           NumEmpdep NSSGerente FechaInicGerente
------ ------------------------------------------------------- --------- ---------- -----------
1      Finanzas                20        11111111   2001-02-16 23:25:00
2      Marketing               15        11111111   1995-05-05 00:00:00
3      Informática             4         22222222   1998-12-07 00:00:00
4      Procesos                17        44444444   2002-11-08 00:00:00
   ..............
```

Ahora modificamos el director del departamento 1.

```
select * from departamentos
update Departamentos
set NSSGerente='44444444'  where Numdep=1
```

```
NumDep NombreDep           NumEmpdep NSSGerente FechaInicGerente
------ ------------------------------------------------------- --------- ---------- -----------
1      Finanzas                20        44444444   2011-03-16 13:25:00
2      Marketing               15        11111111   1995-05-05 00:00:00
3      Informática             4         22222222   1998-12-07 00:00:00
4      Procesos                17        44444444   2002-11-08 00:00:00

   ..............
```

Y el histórico tendrá:

```
NumDep NSSGerente FechaInicGerente        NombreDep             FechaFinGerente
------ ---------- ---------------------- ------------------------------------------------------------
1      11111111   2011-03-16 23:25:00    Finanzas              2011-03-16 13:23:00
```

Si en la **actualización se infrige una restricción, entonces <u>se revierten todas las acciones del trigger y no se actualiza</u>**.

Ejemplo 4: Queremos impedir que se borre de la tabla participacion un empleado que tenga un sueldo superior a 1500.

```
USE CompañiaConsultas
SET NOCOUNT ON
IF EXISTS (SELECT *
                FROM sys.objects
                 WHERE name = 'BorrarParticipacion' AND type = 'TR')
  DROP TRIGGER BorrarParticipacion
GO
CREATE TRIGGER BorrarParticipacion
ON PARTICIPACION
FOR DELETE
AS
--nO PERMITE BORRAR A UN EMPLEADO EN PARTICIPACION QUE TENGA UN
--SUELDO superior A 1500
```

```
IF (SELECT Salario FROM EMPLEADOS
    WHERE NSS IN (select NSS from deleted)) >1500
 BEGIN
   PRINT 'NO SE PUEDE BORRAR EL EMPLEADO DE LA TABLA PARTICIPACION'
   PRINT 'TIENE UN SUELDO SUPERIOR A 1500'
 --Hay que deshacer el borrado. TB vale con ROLLBACK TRANSACTION
  INSERT PARTICIPACION
    SELECT * FROM DELETED
END
```

Ejemplo 5: Idem pero con desecadenadores INSTEAD OF

```
USE CompañiaConsultas
SET NOCOUNT ON
IF EXISTS (SELECT *
                FROM sys.objects
                 WHERE name = 'BorrarParticipacion' AND type = 'TR')
  DROP TRIGGER BorrarParticipacion
GO
CREATE TRIGGER BorrarParticipacion
ON PARTICIPACION
INSTEAD OF DELETE
AS
--nO PERMITE BORRAR A UN EMPLEADO EN PARTICIPACION QUE TENGA UN SUELDO superior A 1500
 IF (SELECT Salario FROM EMPLEADOS
      WHERE NSS IN (select NSS from deleted)) >1500
   BEGIN
     PRINT 'NO SE PUEDE BORRAR EL EMPLEADO DE LA TABLA PARTICIPACION'
   END
ELSE
  BEGIN
    --TEnemos que borrarlo por que es de tipo INSTEAD OF
    PRINT 'PROCEDEMOS AL BORRADO'
    DELETE FROM PARTICIPACION
     WHERE NSS IN (select NSS from deleted)
  END
```

OPERACIÓN CON MULTIPLES REGISTROS

Ejemplo 6: Hasta ahora solo nos valía para borrar un registro, pero que pasa si ejecutamos el siguiente código que la operación afecta a más de una fila.

```
DELETE FROM Participacion
WHERE (nss='11111111' and Numproy=1) Or(nss='22222222' and Numproy=4)
```

En el siguiente código anterior del trigger nos **daría un error: por que la subconsulta nos devuelve mas de una fila** y no se puede **utilizar ningún signo de comparación**

```
IF (SELECT Salario FROM EMPLEADOS
      WHERE NSS IN (select NSS from deleted)) >1500  --ERROR
```

SOLUCION 1: Utilizar Cursor para acceder individualmente a las filas de la tabla DELETED y ir viendo si cumplen la condición o no.

```
USE CompañiaConsultas
SET NOCOUNT ON
IF EXISTS (SELECT *  FROM sys.objects
      WHERE name = 'BorrarParticipacion' AND type = 'TR')
  DROP TRIGGER BorrarParticipacion
GO
CREATE TRIGGER BorrarParticipacion
ON PARTICIPACION
INSTEAD OF DELETE
AS
--nO PERMITE BORRAR A UN EMPLEADO EN PARTICIPACION QUE TENGA UN SUELDO superior A 1500
DECLARE deleted_cursor CURSOR FOR
     SELECT * FROM DELETED

DECLARE @NSS char(8),@Numproy int,@NumHoras tinyint

OPEN deleted_cursor
FETCH NEXT FROM Deleted_cursor INTO @NSS,@Numproy,@NumHoras
WHILE @@FETCH_STATUS =0
BEGIN
   IF (SELECT Salario FROM EMPLEADOS  WHERE NSS=@NSS) >1500
    BEGIN
      PRINT 'NO SE PUEDE BORRAR EL EMPLEADO '+@NSS+ 'DE LA TABLA PARTICIPACION'
    END
   ELSE
    BEGIN
    --TEnemos que borrarlo por que es de tipo INSTEAD OF
    PRINT 'PROCEDEMOS AL BORRADO DEL EMPLEADO '+@NSS
    DELETE FROM PARTICIPACION
     WHERE NSS=@NSS and NumProy=@Numproy
    END
 FETCH NEXT FROM Deleted_cursor INTO @NSS,@Numproy,@NumHoras
END
CLOSE deleted_cursor
DEALLOCATE deleted_cursor
```

El **uso de cursores en SQL Server puede ser necesario en ciertos casos donde se necesita procesar filas individualmente.**

Sin embargo, los cursores pueden ser costosos en términos de rendimiento, especialmente cuando se trabaja con grandes conjuntos de datos, ya que procesan las filas una por una.

Una alternativa a los cursores **podría ser el uso de operaciones basadas en conjuntos**, que son generalmente más eficientes en SQL. En tu caso, podrías utilizar una subconsulta que devuelva un conjunto de resultados y luego operar sobre ese conjunto.

Un ejemplo de cómo podrías reescribir el anterior desencadenador utilizando una subconsulta en lugar de un cursor:

SQL

```
CREATE TRIGGER BorrarParticipacion
ON PARTICIPACION
INSTEAD OF DELETE
AS
BEGIN
   SET NOCOUNT ON;

   -- Identificar los empleados con salario > 1500
   DECLARE @Empleados TABLE (NSS char(8));
   INSERT INTO @Empleados (NSS)
   SELECT NSS FROM EMPLEADOS WHERE Salario > 1500;

   -- Borrar solo los empleados que no están en @Empleados
   DELETE FROM PARTICIPACION
   WHERE NSS NOT IN (SELECT NSS FROM @Empleados) AND NSS IN (SELECT NSS FROM DELETED);
END
```

6. IF UPDATE (columna)

Devuelve un **valor booleano que indica si se intentó utilizar INSERT o UPDATE en una columna especificada de una tabla o vista**. UPDATE() se utiliza en cualquier lugar del cuerpo de un desencadenador INSERT o UPDATE de Transact-SQL para probar si el desencadenador debe ejecutar ciertas acciones.

UPDATE() devuelve TRUE independientemente de si un intento de INSERT o UPDATE tiene éxito.

IF UPDATE() en el cuerpo de un desencadenador DML devuelve TRUE si estas columnas se actualizaron.

UPDATE(columna) se puede usar en cualquier lugar del cuerpo de un desencadenador Transact-SQL

Ejemplo: Se crea un desencadenador que imprime un mensaje si se intenta actualizar la columna CodigoPostal de la tabla Ciudad.

```
IF EXISTS (SELECT name FROM sys.objects
      WHERE name = 'NoActualizacionCodigoPostal' AND type = 'TR')
   DROP TRIGGER NoActualizacionCodigoPostal

CREATE TRIGGER NoActualizacionCodigoPostal
ON Persona.Ciudad
INSTEAD OF UPDATE
AS
IF (UPDATE (CodigoPostal) )
BEGIN
    RAISERROR ('No se puede modificar el código Postal', 16, 10)
END;
GO
-- NO PERMITE CAMBIAR EL CODIGO POSTAL
UPDATE Persona.Ciudad
SET CodigoPostal = '99999'
WHERE CodigoPostal = '12345';
```