Desencadenantes (Triggers)

- *Definición de trigger.*
- *El trigger como transacción.*
- *Ventajas de los triggers.*
- *Consideraciones para usar triggers.*
- *Creación de triggers.*
    - *Obtención de información sobre triggers.*
- *Modificación de un trigger.*
- *Eliminación de un trigger.*
- *Ejemplo.*
    - *Trigger por inserción.*
    - *Trigger por eliminación.*
    - *Trigger por actualización.*
    - *Trigger INSTEAD OF.*

Definición de trigger

- El **trigger** es un clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce una modificación en el contenido de una tabla.
- Un trigger 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 trigger no puede ser llamado directamente, y no aceptan o retorna parámetros.
- Un trigger es tratado como una transacción.

El trigger como transacción

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

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

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

Ventajas de los triggers

- Los triggers se utilizan para definir lógica de aplicación compleja. Son adecuados para los procesos que se ejecutan en cascada.
- Como el trigger es reactivo, él puede comparar los datos antes y después que la data ha sido modificada.

Consideraciones para usar triggers

Tenga en cuenta lo siguiente al trabajar con triggers:

- Los triggers son reactivos; las restricciones son proactivas.

  Los triggers son ejecutados después de una sentencia INSERT, UPDATE, o DELETE que se ejecuta en la tabla donde se ha definido el trigger. Por ejemplo, una declaración de UPDATE actualiza una fila en una tabla, y entonces el trigger 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 triggers.

  Si existen restricciones en la tabla que contiene el trigger, ellas se verifican antes de la ejecución del trigger. Si se violan las restricciones, el trigger no se ejecuta.

- Las tablas pueden tener múltiples triggers para cualquier acción.

  A partir de SQL Server 7 se puede definir múltiples triggers en una sola tabla. Cada trigger puede definirse para una sola acción o para acciones múltiples. Los triggers no se disparan en ningún orden en particular.

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

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

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

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

- Los dueños de las tablas no pueden crear triggers AFTER sobre vistas o en las tablas temporales. Sin embargo, los triggers pueden hacer referencia a vistas y las tablas temporales.

- Los dueños de las tablas pueden crear triggers INSTEAD OF sobre vistas y tablas extendiendo los tipos de actualizaciones que una vista puede soportar.

- Los triggers no deben retornar un conjunto de filas como resultado.

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

Creación de triggers

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

```
CREATE TRIGGER nombre_trigger
    ON nombre_tabla
    FOR INSERT | UPDATE | DELETE
AS
    sentencias_sql
```

Los dueños de la tabla, así como los miembros de los roles **db_owner** y **sysadmin**, tienen permiso para crear un trigger.

SQL Server no permite usar las declaraciones siguientes en una definición de trigger:

- CREATE DATABASE, ALTER DATABASE y DROP DATABASE
- LOAD DATABASE
- RESTORE DATABASE
- LOAD LOG
- RESTORE LOG
- RECONFIGURE
- DISK INIT y DISK RESIZE

Obtención de información sobre triggers

Procedimiento	Resultado
sp_depends *nombre_tabla*	Triggers que dependen de la tabla
sp_helptext *nombre_trigger*	Sentencia que definió el trigger
sp_helptrigger *nombre_tabla*	Triggers definidos para la tabla

Modificación de un trigger

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

ALTER TRIGGER *nombre_trigger*
ON *nombre_tabla*
FOR INSERT | UPDATE | DELETE
AS
sentencias_sql

Eliminación de un trigger

El permiso para eliminar un trigger 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 de la declaración en DROP TRIGGER.

DROP TRIGGER *nombre_trigger*

Ejemplo

Crear las siguientes tablas:

```
CREATE TABLE departamento(
    dep_cod  integer  not null  default 0 ,
    dep_nom  varchar(30) )

CREATE TABLE empleado(
    codigo  integer  not null ,
    paterno  varchar(20) ,
    sueldo  integer ,
    dep_cod  integer )
```

Luego, insertar los siguientes datos:

```
INSERT departamento VALUES( 1 , 'Gerencia' )
INSERT departamento VALUES( 2 , 'Planeamiento' )
INSERT departamento VALUES( 3 , 'Contabilidad' )

INSERT empleado VALUES( 101 , 'Vargas' , 1500 , 2 )
INSERT empleado VALUES( 102 , 'Duarte' , 1200 , 1 )
INSERT empleado VALUES( 103 , 'Soria' , 1500 , 2 )
INSERT empleado VALUES( 104 , 'Tang' , 1200 , 3 )
```

Inserte 20 empleados más.

Trigger por inserción

Para la tabla ***departamento*** crear un trigger de nombre ***tg_insert_departamento*** que no permita duplicados en el *código del departamento*, y que lo genere si el usuario no lo especifica.

```
CREATE TRIGGER tg_insert_departamento
    ON departamento
    FOR INSERT
AS
IF ( SELECT COUNT(*)
    FROM departamento INNER JOIN inserted
    ON departamento.dep_cod = inserted.dep_cod ) > 1
    BEGIN
        PRINT 'Código duplicado'
        ROLLBACK
    END
ELSE
    PRINT 'Inserción ejecutada'
    IF ( SELECT COUNT(*)
        FROM departamento
        WHERE departamento.dep_cod = 0 ) > 0
        BEGIN
            UPDATE departamento
                SET dep_cod =
                ( SELECT MAX( dep_cod )
                FROM departamento ) + 1
                WHERE dep_cod = 0
        END
```

Inserte en la tabla ***departamento***, una fila con *código de departamento* igual a 2 (un código duplicado), y con *nombre de departamento* igual a 'Finanzas'.

```
INSERT departamento VALUES( 2 , 'Finanzas' )

SELECT * FROM departamento
```

Inserte en la tabla ***departamento***, una fila con *nombre de departamento* igual a 'Finanzas', y sin especificar el *código de departamento*. Recuerde que para la columna ***dep_cod*** se definió una restricción *default* igual a 0.

```
INSERT departamento( dep_nom ) VALUES( 'Finanzas' )

SELECT * FROM departamento
```

Cómo trabaja un trigger por inserción

Cuando un trigger de INSERCIÓN se dispara, se agregan las nuevas filas a la tabla del trigger y a la tabla **inserted**. La tabla **inserted** es una tabla lógica que mantiene una copia de las filas que se han insertado. La tabla **inserted** permite obtener la referencia de los datos desde la inicialización de la sentencia INSERT. El trigger puede examinar la tabla **inserted** para determinar cómo las acciones del trigger deben llevarse a cabo.

Trigger por eliminación

Para la tabla ***departamento*** crear un trigger de nombre ***tg_delete_departamento*** que permita eliminar un *departamento*. Si existen *empleados* para el departamento eliminado, que les asigne el valor 999 para su *departamento*.

```
CREATE TRIGGER tg_delete_departamento
    ON departamento
    FOR DELETE
AS
IF EXISTS( SELECT empleado.dep_cod
    FROM empleado INNER JOIN deleted
    ON empleado.dep_cod = deleted.dep_cod )
    BEGIN
        UPDATE empleado SET dep_cod = 999
            FROM deleted
            WHERE empleado.dep_cod =
            deleted.dep_cod
        PRINT 'Transacción procesada'
    END
```

Elimine de la tabla ***departamento***, el *departamento* cuyo *código* es 2.

```
DELETE FROM departamento WHERE dep_cod = 2

SELECT * FROM empleado
```

Cómo trabaja un trigger por eliminación

Cuando un trigger DELETE se dispara, los registros eliminados se transfieren a la tabla **deleted**. La tabla **deleted** es una tabla lógica que mantiene una copia de las filas que se han eliminado. La tabla **deleted** permite obtener la referencia de los datos desde la inicialización de la sentencia DELETE.

Cuando una fila se añade a la tabla **deleted**, ésta ya no existe en la tabla de la base de datos; por consiguiente, la tabla **deleted** y la tabla de la base de datos no tienen ninguna fila en común.

Trigger por actualización

Para la tabla ***departamento*** crear un trigger de nombre ***tg_update_departamento*** que no permita modificar el *código de departamento*.

```
CREATE TRIGGER tg_update_departamento
    ON departamento
    FOR UPDATE
AS
IF UPDATE( dep_cod )
    BEGIN
        PRINT 'Operación no permitida'
        ROLLBACK
    END
ELSE
    PRINT 'Actualización procesada'
```

Modifique el *código* del departamento 'Planeamiento'. El nuevo *código* será 4.

```
UPDATE departamento SET dep_cod = 4
WHERE dep_nom = 'Planeamiento'
```

Cómo trabaja un trigger por actualización

Una declaración UPDATE puede pensarse en función de dos pasos - el paso DELETE que captura la imagen “antes” de los datos, y el paso INSERT que captura la imagen “después” de los datos. Cuando una declaración UPDATE se ejecuta en una tabla que tiene un trigger definida en él, las filas originales (“antes”) se pasan a la tabla **deleted** y las filas actualizadas (“después”) se insertan en la tabla **inserted**.

El trigger puede examinar las tablas **deleted** e **inserted**, así como la tabla actualizada, para determinar si se han actualizado las múltiples filas, y cómo las acciones del trigger deben llevarse a cabo.

Trigger INSTEAD OF

Este ejemplo crea una tabla con los empleados del departamento 2, y otra tabla con los empleados del departamento 3. Luego se crea una vista con los datos de ambas tablas. En la vista se crea un trigger INSTEAD OF de manera que cuando se actualice una fila en la vista, esta actualización se redireccione a la tabla correspondiente. La actualización en la tabla ocurre en vez de la actualización en la vista.

```
-- Crear dos tablas con datos de los empleados

SELECT * INTO empleados2
     FROM empleado
     WHERE dep_cod = 2

SELECT * INTO empleados3
     FROM empleado
     WHERE dep_cod = 3
GO


-- Crear una vista con los datos de las tablas anteriores

CREATE VIEW v_Empleados2y3
AS
SELECT * FROM empleados2
UNION
SELECT * FROM empleados3
GO


-- Crear un trigger INSTEAD OF sobre la vista

CREATE TRIGGER tg_update_v_Empleados2y3
ON v_Empleados2y3
INSTEAD OF UPDATE
AS
DECLARE @depa integer
SET @depa = (SELECT dep_cod FROM inserted)
IF @depa = 2
     BEGIN
     UPDATE empleados2
          SET empleados2.sueldo = inserted.sueldo
          FROM empleados2 JOIN inserted
          ON empleados2.codigo = inserted.codigo
     END
ELSE
```

```
IF @depa = 3
      BEGIN
      UPDATE empleados3
            SET empleados3.sueldo = inserted.sueldo
            FROM empleados3 JOIN inserted
            ON empleados3.codigo = inserted.codigo
      END
GO

-- Prueba del trigger

SELECT * FROM empleados3
      WHERE codigo = 104
-- Empleado Tang del departamento 3 con sueldo de 1200

-- Actualiza en la vista los datos del empleado Tang

UPDATE v_Empleados2y3
      SET sueldo = 1800
      WHERE codigo = 104
GO

SELECT * FROM empleados3
      WHERE codigo = 104
GO

-- El sueldo fue cambiado.
-- Los datos de Tang fueron actualizados en la tabla.
```

Esta página se ha dejado en blanco intencionalmente.