SQL Programming

Módulo 6 - Trigger

Trigger

Trigger

Crea un desencadenador DML, DDL o logon.

Un **desencadenador** es un tipo especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de bases de datos.

Los desencadenadores DML se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.Estos desencadenadores se activan cuando se desencadena cualquier evento válido, con independencia de que las filas de la tabla se vean o no afectadas. Para más información, consulte DML Triggers.

TRIGGER

Los desencadenadores DDL se ejecutan como respuesta a diversos eventos del lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones CREATE, ALTER y DROP de Transact-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

Los usos más comunes para un TRIGGER son:

- Mantener la integridad de los datos.
- Mantener la integridad referencial
- Mantener auditoría de los cambios en registros de las tablas
- Invocar acciones externas, tales como enviar un mail...

Se pueden disparar los TRIGGERS cuando se modifican datos con alguna de las sentencias: INSERT, DELETE o UPDATE.

Triggers AFTER e INSTEAD-OF

Existen dos tipos de TRIGGERS:

- AFTER TRIGGER
- INSTEAD-OF TRIGGER

AFTER TRIGGER

Es el TRIGGER por defecto en SQL Server. Se pueden definir múltiples TRIGGERS para una tabla y para un mismo evento. Cada TRIGGER puede invocar varios STORES PROCEDURES.

Se puede crear un solo TRIGGER que se ejecute para alguno/s o para todos los eventos de modificación de una tabla: INSERT, UPDATE o DELETE.

SQL Server no ofrece la posibilidad de disparar un TRIGGER desde la sentencia SELECT. Estos TRIGGERS solo pueden existir para tablas y no para vistas.

SQL Server permite la ejecución de TRIGGER recursivos, siempre y cuando la opción recursive triggers este seteada en TRUE. Esto significa que si el TRIGGER modifica la tabla dentro del mismo TRIGGER esto dispararía una segunda ejecución del TRIGGER. Este procedimiento no puede entrar en un LOOP infinito porque SQL Server permite un máximo de 32 niveles de profundidad.

Triggers AFTER e INSTEAD-OF

INSTEAD-OF TRIGGERS

INSTEAD OF TRIGGERS son ejecutados en forma automática antes de que SQL SERVER realice los chequeos de PRIMARY KEY y FOREIGN KEY.

Mientras que los TRIGGERS tradicionales son ejecutados automáticamente después de chequear estas CONSTRAINTS. Solo se puede tener un instead-of trigger por cada acción (INSERT, UPDATE, y DELETE).

No se puede combinar instead-of triggers y claves foráneas que tengan definiciones de CASCADA.

Instead-of triggers están previstos para permitir actualizaciones sobre vistas, que normalmente no son modificables.

Triggers AFTER e INSTEAD-OF

Limitaciones de los desencadenadores

CREATE TRIGGER debe ser la primera instrucción en el proceso por lotes y solo se puede aplicar a una tabla.

Un desencadenador se crea solamente en la base de datos actual; sin embargo, un desencadenador puede hacer referencia a objetos que están fuera de la base de datos actual.

Si se especifica el nombre del esquema del desencadenador (para calificarlo), califique el nombre de la tabla de la misma forma.

La misma acción del desencadenador puede definirse para más de una acción del usuario (por ejemplo, INSERT y UPDATE) en la misma instrucción CREATE TRIGGER.

Los desencadenadores INSTEAD OF DELETE/UPDATE no pueden definirse en una tabla con una clave externa definida en cascada en la acción DELETE/UPDATE.

Triggers AFTER e INSTEAD-OF

En un desencadenador se puede especificar cualquier instrucción SET. La opción SET seleccionada permanece en efecto durante la ejecución del desencadenador y, después, vuelve a su configuración anterior.

Cuando se activa un trigger, los resultados se devuelven a la aplicación que llama, igual que con los procedimientos almacenados. Para impedir que se devuelvan resultados a la aplicación debido a la activación de un desencadenador, no incluya las instrucciones SELECT que devuelven resultados ni las instrucciones que realizan una asignación variable en un desencadenador.

Un desencadenador que incluya instrucciones SELECT que devuelven resultados al usuario o instrucciones que realizan asignaciones de variables requiere un tratamiento especial; estos resultados devueltos tendrían que escribirse en cada aplicación en la que se permiten modificaciones a la tabla del desencadenador. Si es preciso que existan asignaciones de variable en un trigger, utilice una instrucción SET NOCOUNT al principio del mismo para impedir la devolución de cualquier conjunto de resultados.

Triggers AFTER e INSTEAD-OF

Una instrucción TRUNCATE TABLE es de hecho una instrucción DELETE, pero no activa un desencadenador porque la operación no registra eliminaciones de filas individuales. Sin embargo, sólo los usuarios con permisos para ejecutar una instrucción TRUNCATE TABLE tienen que ocuparse de cómo sortear un desencadenador de DELETE de esta manera.

Creamos la tabla **Prueba** y **CopiaPrueba** para realizar los siguientes ejemplos.

Sintaxis

```
IF OBJECT_ID (N'dbo.Prueba', N'U') IS NOT NULL
           DROP TABLE dbo.Prueba;
GO
CREATE TABLE dbo.Prueba(Codigo INT, Nombre
VARCHAR(50));
GO
IF OBJECT_ID (N'dbo.CopiaPrueba', N'U') IS NOT NULL
           DROP TABLE dbo.CopiaPrueba;
GO
CREATE TABLE [dbo].[CopiaPrueba](Codigo INT, Nombre
VARCHAR(50));
```

Triggers AFTER e INSTEAD-OF

Crear un trigger que se dispara ante un evento INSERT

El siguiente ejemplo crea un trigger llamado TR_Prueba sobre la taba Prueba, al dispararse ante una operación INSERT guarda un histórico en la tabla CopiaPrueba.

Sintaxis

```
IF OBJECT_ID (N'dbo.TR_Prueba', N'TR') IS NOT NULL

            DROP TRIGGER dbo.TR_Prueba;
GO

CREATE TRIGGER dbo.TR_Prueba ON dbo.Prueba
AFTER INSERT AS
BEGIN
        INSERT INTO dbo.CopiaPrueba
        SELECT * FROM inserted;
END

INSERT INTO dbo.Prueba
VALUES (1,'GABRIEL'),(2,'CARLOS'),(3,'JUAN');

SELECT * FROM dbo.CopiaPrueba;
```

Triggers AFTER e INSTEAD-OF

Crear un trigger que se dispara ante un evento UPDATE

El siguiente ejemplo crea un trigger llamado TR_Prueba sobre la taba Prueba, al dispararse ante una operación UPDATE modifica el nombre en la tabla de históricos CopiaPrueba.

Sintaxis

```
IF OBJECT_ID (N'dbo.TR_Prueba', N'TR') IS NOT NULL

            DROP TRIGGER dbo.TR_Prueba;
GO

CREATE TRIGGER dbo.TR_Prueba ON dbo.Prueba
AFTER UPDATE AS
BEGIN
        UPDATE p SET Nombre=i.Nombre
        FROM dbo.CopiaPrueba p INNER JOIN Inserted i
  ON p.Codigo=i.Codigo;
END
GO

      UPDATE dbo.Prueba SET Nombre='PEDRO'
      WHERE codigo = 1;
SELECT Nombre FROM dbo.CopiaPrueba WHERE codigo = 1;
```

Triggers AFTER e INSTEAD-OF

Crear un trigger que se dispara ante un evento DELETE

El siguiente ejemplo crea un trigger llamado TR_Prueba sobre la taba Prueba,
al dispararse ante una operación DELETE elimina el dato del histórico.

Sintaxis

```
IF OBJECT_ID (N'dbo.TR_Prueba', N'TR') IS NOT NULL
            DROP TRIGGER dbo.TR_Prueba;
GO
CREATE TRIGGER dbo.TR_Prueba ON dbo.Prueba
AFTER DELETE AS
BEGIN
        DELETE FROM dbo.CopiaPrueba
        FROM dbo.CopiaPrueba p INNER JOIN deleted d
        ON p.Codigo=d.Codigo;
END
GO
DELETE FROM dbo.Prueba
WHERE Nombre LIKE '%PEDRO%';
SELECT * FROM dbo.CopiaPrueba;
```

¡Muchas gracias!

¡Sigamos trabajando!