Cursos SQL Server 2008 R2

Triggers

Cursos SQL Server 2008 R2

Índice

Introducción-Triggers

Es una clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de bases de datos.

Tipos de desencadenadores:

- Desencadenadores DML.
- Los desencadenadores DDL.

Desencadenadores DML

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.

Funcionalidad:

- Imponer las reglas de negocios
- Integridad de los datos

Los desencadenadores DML tienen varias utilidades:

- 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.
- Pueden hacer referencia a columnas de otras tablas. Por ejemplo, un desencadenador puede utilizar una instrucción SELECT de otra tabla para comparar con los datos insertados o actualizados y para realizar acciones adicionales, como modificar los datos o mostrar un mensaje de error definido por el usuario.
- Pueden evaluar el estado de una tabla antes y después de realizar una modificación de datos y actuar en función de esa diferencia.
- 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.

Tipos de triggers DML

- INSTEAD OF: se activan en lugar de la acción original, antes de que tenga lugar la modificación del objeto base.
- AFTER: se activan inmediatamente después de la modificación de los datos de la tabla base.

Una tabla puede tener un número ilimitado de triggers AFTER para cada acción peros solo un trigger INSTEAD OF por acción. Las vistas solo pueden tener triggers INSTEAD OF.

Recursos para el desarrollo de desencadenadores DML

Función UPDATE()

- Devuelve un valor booleano que indica si se intentó utilizar INSERT o UPDATE en una columna especificada de una tabla o vista. Para saber si dentro de un trigger si se ha modificado alguna columna usaremos la función UPDATE().

Utilizar la función COLUMNS_UPDATED

- Sirve para controlar los cambios en varias columnas en una sola instrucción.
- Devuelve una secuencia de bits, uno para cada columna, donde cada bit es 1 si la columna sufrió cambios o 0 en caso contrario.
- COLUMNS_UPDATED se utiliza en cualquier parte del cuerpo de un desencadenador INSERT o UPDATE de Transact-SQL para comprobar si el desencadenador debe ejecutar determinadas acciones.

Ejemplo:

10.ModColumnas.sql

Tablas inserted y deleted

Son tablas temporales residentes en memoria. Se utilizan para probar los efectos de determinadas modificaciones de datos y para establecer condiciones para las acciones del desencadenador DML.

En los desencadenadores DML, las tablas **inserted** y **deleted** se utilizan principalmente para realizar las siguientes tareas:

- Ampliar la integridad referencial entre tablas.
- Insertar o actualizar datos de tablas base subyacentes a una vista.
- Comprobar errores y realizar acciones en función del error.

Conocer la diferencia entre el estado de una tabla antes y después de realizar una modificación en los datos, y actuar en función de dicha diferencia.

La tabla **Deleted:**

- ✓ Almacena copias de las filas afectadas por las instrucciones DELETE y UPDATE.
- ✓ Durante la ejecución de una instrucción DELETE o UPDATE, las filas se eliminan de la tabla del desencadenador y se transfieren a la tabla **deleted**.
- ✓ La tabla **deleted** y la tabla del desencadenador no suelen tener filas en común.

La tabla **inserted:**

- ✓ Almacena copias de las filas afectadas durante las instrucciones INSERT y UPDATE.
- ✓ Durante una transacción de inserción o actualización, se agregan nuevas filas a la tabla **inserted** y a la tabla del desencadenador.
- ✓ Las filas de la tabla **inserted** son copias de las nuevas filas de la tabla del desencadenador.

Una transacción de actualización (UPDATE):

1. Se copian las filas antiguas en la tabla **deleted**
2. Se copian las filas nuevas en la tabla del desencadenador y en la tabla **inserted**.

Se pueden utilizar las tablas **inserted** y **deleted como condición:**

No se produce ningún error al hacer referencia a la tabla **deleted** cuando se prueba una instrucción INSERT, o bien al hacer referencia a la tabla **inserted** cuando se prueba una instrucción DELETE, estas tablas de prueba del desencadenador no contendrán filas en estos casos.

Ejemplos

3.Virtuales-1.sql

4.Virtuales-2.sql

Control de operaciones con varias filas

- La activación de un trigger solo se lleva a cabo como respuesta a una acción que modifica una sola fila o varias en una misma instrucción.
- Si queremos un trigger que solo se active con operaciones de una sola fila, debemos rechazar los cambios que afecten a más de una fila. Podemos controlarlo con la función de sistema @@ROWCOUNT.
- Si queremos un trigger que solo se active con operaciones de varias filas, podríamos usar funciones de agregación o cursores.
- La situación ideal sería crear un trigger condicional que pudiera trabajar con operaciones de una fila o de varias, de acuerdo con el valor devuelto por @@ROWCOUNT.

11.multifila.sql

Anidamiento de triggers

- Una tabla puede tener un trigger que modifica otra tabla que tiene asociado otro trigger.
- Se pueden llegar a anidar hasta 32 triggers.
- El anidamiento de triggers está habilitado de forma predeterminada.
- Si durante la ejecución de un trigger, un procedimiento almacenado o una función definida por el usuario, queremos saber la cantidad de niveles de anidamiento alcanzada, podemos utilizar la función @@NESTLEVEL.
- En una situación de anidamiento de triggers, todos se ejecutan dentro de la misma transacción. Un error en cualquiera de ellos cancela toda la Transacción.

Ejemplos:

15.Anidados.sql

15AnidadosIpurchase-upurchase

- Se puede cambiar esta opción a nivel de servidor configurando la opción "nested triggers" en 0 mediante el procedimiento sp_configure.

16.Evitar anidados.sql

Mas ejemplos de triggers DML AFTER:

1.Integridad_Compleja.sql

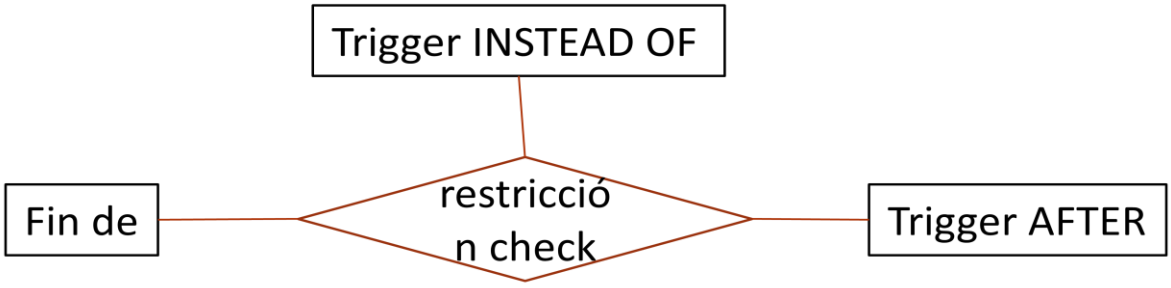
19.Ejemploafter.doc

Desencadenadores DML INSTEAD OF

Los desencadenadores INSTEAD OF se activan en lugar de la acción desencadenadora y antes del procesamiento de las restricciones.

Instead-1.sql

Si una tabla incluye desencadenadores AFTER, se activarán cuando finalice el procesamiento de las restricciones. Si se infringen las restricciones, se revertirán las acciones del desencadenador INSTEAD OF y no se ejecutará el desencadenador AFTER.



Funcionalidad:

Las vistas basadas en funciones de agregación son de solo lectura. Los desencadenadores INSTEAD OF resuelven este problema.

Ejemplo:

Instead-2.sql

Si una vista se define como una consulta que combina varias tablas, solo se puede ejecutar una instrucción UPDATE para realizar actualizaciones a través de la vista cuando las modificaciones afectan a columnas de una sola tabla.

Instead-3.sql

Habilitar y deshabilitar trigger

Existen dos motivos para deshabilitarlos:
1. Para acelerar ciertos procesos se puede deshabilitar temporalmente un trigger. Se consigue con la instrucción ALTER TABLE con la opción DISABLE TRIGGER.

2. Para evitar que se ejecuten con los triggers en respuesta a la recepción de datos provenientes de la duplicación. Se consigue con la instrucción CREATE o ALTER TRIGGER con la opción NOT FOR REPLICATION.

Ejemplo:
13.Deshabilitar.sql
Para volver a habilitar un trigger usamos la instrucción ALTER TABLE con la opción ENABLE TRIGGER.

14.Habilitar.sql

Triggers_recursivos

Existen dos tipos de recursividad:

1. Recursividad indirecta: una sola instrucción obliga a la ejecución repetida del mismo trigger mediante la ejecución de otros triggers. Por ejemplo, si un trigger sobre Products, el trigger definido para esta última tabla volverá a activarse.

2. Recursividad directa: una tabla base tiene un trigger que vuelve a modificar algunos datos en la tabla. En este caso, La configuración predeterminada de SQL Server hace que no vuelva a activar el trigger, evitando con ello la recursividad directa.

Ejemplo:

17.Recursivos1.sql

18.Recursivos2.sql

Triggers DDL

Antes era dificultoso auditar los cambios en el esquema subyacente (no auditar los datos).

Un trigger DDL puede ser definido a nivel de servidor o de base de datos.

Pueden ser lanzados para la creación, modificación o borrado de cualquier objeto de SQL Server.

Para saber qué evento lanzó el trigger se utiliza la función ***eventdata()***. No se utiliza la función *Update().*

Triggers DDL – Creación y modificación

No se definen sobre objetos de bases de datos y no se permite INSTEAD OF.

Se pueden especificar a nivel de servidor (ALL SERVER) o a nivel de base de datos (DATABASE).

Eventos a nivel Servidor:

• CREATE|ALTER|DROP LOGIN

• CREATE|DROP HTTP ENDPOINT

• GRANT|DENY|REVOKE SERVER ACCESS

• CREATE|ALTER|DROP CERT

El resto de eventos que pueden ser usados por los triggers DDL son a nivel de base de datos:

- CREATE|ALTER|DROP TABLE, CREATE|ALTER|DROP TRIGGER

Un evento interesante es DDL_DATABASE_LEVEL_EVENTS. Incluye todos los eventos que pueden ocurrir en la base de datos, y es satisfactorio para situaciones en las que el DBA podría desear registrar o bloquear cualquier cambio realizado en la base de datos.

Ejemplo: deshacemos cualquier modificación DDL:

```
CREATE TRIGGER SinCambios
ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS
SELECT 'DDL's no están permitidas en la base de datos actual!'
SELECT 'Para permitir DDL's, elimina el trigger SinCambios.'
ROLLBACK
```

Triggers DDL – Borrado

Se debe especificar el ámbito del trigger en el mandato. Si no lo hacemos, no lo podemos eliminar.

DROP TRIGGER *trigger_name [,...n]*

ON { DATABASE | ALL SERVER } [;]

Triggers DDL – Activado / Desactivado

Su sintaxis es similar a la del borrado.

{ ENABLE | DISABLE } TRIGGER *trigger_name*

ON { DATABASE | SERVER } [;]

Triggers DDL – Vistas de catálogo

Tenemos dos vistas: *sys.triggers* y *sys.triggers_events*.

Los triggers de nivel de servidor pueden enumerarse con sys.server_triggers y sys.server_triggers_events.

La columna parent_class_desc se puede usar para diferenciar los triggers DDL de los DML al consultar la vista sys.triggers.

Esta consulta devuelve el nombre y la fecha de creación de todos los triggers DDL de la base de datos actual:

```
SELECT name, create_date

FROM sys.triggers

WHERE parent_class_desc = 'DATABASE'
```

Triggers DDL – Vistas de catálogo

La columna object_id relaciona las vistas de triggers con las vistas de eventos.

Ejemplo:

```
SELECT tr.name, ev.type_desc

FROM sys.server_triggers tr

JOIN sys.server_trigger_events ev ON tr.object_id = ev.object_id

WHERE tr.is_disabled = 0
```

Triggers DDL – Programación con la función eventdata()

Esta función devuelve un documento XML conteniendo información sobre el elemento que lanzó el trigger.

Cada evento puede devolver datos usando un schema diferente, pero todos comparten schemas base comunes.

Los eventos a nivel de servidor usan el siguiente schema base:

```
<EVENT_INSTANCE>

<EventType>name</EventType>

<PostTime>date-time</PostTime>

<SPID>spid</SPID>
```

<ServerName>server_name</ServerName>

<LoginName>login</LoginName>

</EVENT_INSTANCE>

Los eventos a nivel de base de datos añaden el elemento UserName:

<EVENT_INSTANCE>

<EventType>name</EventType>

<PostTime>date-time</PostTime>

<SPID>spid</SPID>

<ServerName>server_name</ServerName>

<LoginName>login</LoginName>

<UserName>user</UserName>

</EVENT_INSTANCE>

Los objetos basados en eventos (CREATE_TABLE, ALTER_PROCEDURE,etc.) añaden elementos DatabaseName, SchemaName, ObjectName, y ObjectType, y un elemento TSQLCommand que contiene los elementos CommandText y SetOptions.

Al consulta el documento XML, es posible determinar todos los aspectos del evento que lanzó el trigger.

Ejemplo:

TriggerDDL_1.sql

- T-SQL DBA's

Más información

C/ Miracruz, 10 (Bº de Gros) 20001 Donostia

Telf.: 943 275819

email: s e i m @ c e n t r o s e i m . c o m

S.E.I.M.