UT10. SQL MODO PROGRAMACIÓN

Módulo: BASES DE DATOS

Curso 2022/2023. 1° DAM

Ruth Lospitao Ruiz

CONTENIDOS

- Introducción. Conceptos generales
- Variables, operadores, expresiones
- Bloques instrucciones de código
- Estructuras de control.
- Procedimientos y funciones almacenados.
- Cursores
- Manejo de errores
- Triggers

INTRODUCCIÓN

CONCEPTO

- Los triggers, también llamados desencadenadores o disparadores, son programas almacenados que se ejecutan ("disparan") automáticamente en respuesta a algún suceso que ocurre en la base de datos.
- En MySQL ese tipo de suceso se corresponde con alguna instrucción DML (INSERT, UPDATE, DELETE) sobre alguna tabla.
- Suponen un mecanismo para asegurar la integridad de los datos.
- Se emplean también como un método para realizar operaciones de auditoría sobre la base de datos.
- No hay que abusar de su utilización pues ello puede traer consigo una sobrecarga del sistema y por tanto un bajo rendimiento del mismo.

SINTAXIS

```
CREATE [DEFINER={cuenta_usuario | CURRENT_USER}] TRIGGER
nombre_trigger
    {BEFORE | AFTER}
    {UPDATE | INSERT | DELETE}
  ON tabla
  FOR EACH ROW
  cuerpo_del_trigger
```

Para poder crear triggers, en versiones anteriores a la 5.16 se necesita el privilegio SUPER. A partir de esta el privilegio es el de TRIGGER. No se pueden crear ni sobre una tabla temporal ni sobre una vista, solamente sobre tablas

SINTAXIS

- DEFINER ={cuenta_usuario | CURRENT_USER } indica con qué privilegios se ejecutan las instrucciones del trigger. La opción por defecto es CURRENT_USER, que indica que las instrucciones se ejecutan con los privilegios del usuario que lanzó la instrucción de creación del trigger. La opción cuenta_usuario por otro lado hace que el trigger se ejecute con los privilegios de dicha cuenta.
- Nombre del trigger. Sigue las mismas normas que para nombrar cualquier objeto de la base de datos.
- BEFORE | AFTER. Señala cuando se ejecuta el trigger, antes (before) o después (alter) de la instrucción DML que lo provocó.
- UPDATE | INSERT | DELETE. Define la operación DML asociada al trigger.
- ON tabla. Define la tabla base asociada al trigger.

```
CREATE [DEFINER={cuenta_usuario | CURRENT_USER}] TRIGGER
nombre_trigger
    {BEFORE | AFTER}
    {UPDATE | INSERT | DELETE}
  ON tabla
  FOR EACH ROW
  cuerpo_del_trigger
```

SINTAXIS

- FOR EACH ROW. Indica que el trigger se ejecutará por cada fila de la tabla afectada por la operación DML. Esto es, si tenemos asociado un trigger a la operación de borrado de una tabla y se eliminan con una sola instrucción 6 filas de ésta última, el trigger se ejecutará 6 veces, una por cada fila eliminada. Otros gestores de bases de datos (así como futuras implementaciones de MySQL) consideran también el otro estándar de ANSI, la cláusula FOR EACH STATEMENT. Con esta segunda opción, el trigger se ejecutaría por cada operación DML realizada; en el ejemplo anterior, la instrucción de borrado daría lugar a que sólo se ejecutara el trigger una sola vez en lugar de 6 (filas afectadas) con esta futura cláusula
- Cuerpo del trigger: El conjunto de instrucciones que forman este programa almacenado. Si son más de una irán en un bloque BEGIN … END. No pueden contener una instrucción CALL de llamada a un procedimiento almacenado

```
CREATE [DEFINER={cuenta_usuario | CURRENT_USER}] TRIGGER
nombre_trigger
    {BEFORE | AFTER}
    {UPDATE | INSERT | DELETE}
  ON tabla
  FOR EACH ROW
  cuerpo_del_trigger
```

CONSIDERACIONES

- Dentro del cuerpo del trigger se puede hacer referencia a los valores de las columnas que están siendo modificadas con una sentencia DML (la que provocó que se disparara el trigger), incluso pueden cambiarse si así se considerara.
- Para ello se utilizan los **objetos NEW y OLD**. De esta manera, en un trigger de tipo BEFORE UPDATE afectando a una columna micolumna, se utilizará la expresión OLD.micolumna para conocer el valor de la columna antes de ser modificado y NEW.micolumna será el nuevo valor de la columna después de la modificación.
- Estos dos valores sólo tienen sentido los dos juntos en una modificación, pues ante una inserción (INSERT) no existe valor antiguo (OLD) y ante un borrado (DELETE) no existe un valor nuevo (NEW) pues el que existe se elimina.
- Dentro de un trigger de tipo BEFORE, puede cambiarse el nuevo valor mediante una sentencia de asignación SET por lo que anularía por completo el efecto de la instrucción DML que provocó el trigger

EVENTOS DE DISPARO

- Como se ha indicado, un trigger se ejecuta automáticamente (dispara) antes o después de una instrucción DML: INSERT, UPDATE o DELETE.
- Además de la forma explícita anterior, pueden también ejecutarse las instrucciones del cuerpo del trigger si la modificación se produce de forma implícita, como sería el caso de una instrucción REPLACE pues en realidad equivale a una instrucción DELETE seguida de una INSERT (por lo tanto aquí se ejecutarían los triggers asociados a estas dos operaciones).

¿TRIGGERS TIPO BEFORE O AFTER?

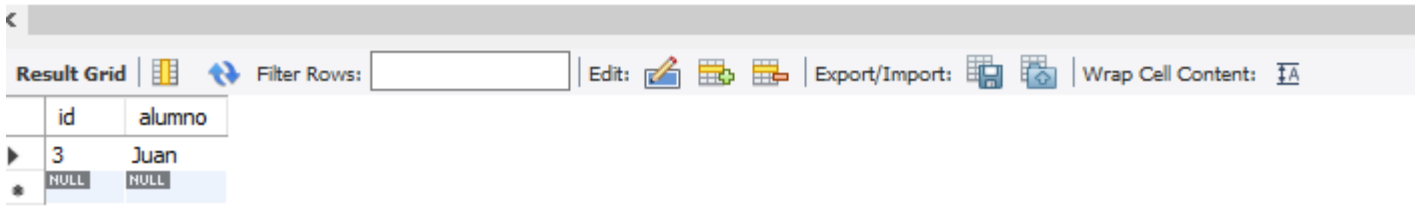
- Prácticamente no hay diferencia. La ventaja que puede suponer utilizar los triggers de tipo BEFORE es que pueden cambiarse los valores modificados inicialmente por una instrucción UPDATE o INSERT mientras que con los triggers de tipo AFTER daría un error de ejecución.
- Teniendo en cuenta los 2 tipos de triggers y las tres operaciones DML distintas, podríamos tener tener 6 triggers por tabla: BEFORE INSERT, AFTER INSERT, BEFORE UPDATE, AFTER UPDATE, BEFORE DELETE y AFTER DELETE.
- La vista TRIGGERS del diccionario de datos contendrá toda la información sobre cada trigger, existiendo una entrada (fila) en esta vista por cada trigger en el servidor; la información que aparece en las columnas es la que se ha tenido en cuenta a la hora de crearlo.

EJEMPLO

- Mantener sincronizada una copia de seguridad de unos datos.

```sql
use curso;
select * from alumnos; -- para ver los registros tiene la tabla original
drop table if exists alumnos_replica;
create table alumnos_replica
(
    id int primary key,
    alumno varchar(50)
);

-- Creación trigger que mantiene sincronizada una copia de seguridad
DROP TRIGGER IF EXISTS trigger1;
DELIMITER //
CREATE TRIGGER trigger1 before insert on alumnos for each row
begin
    insert into alumnos_replica values (new.id, new.alumno);
end //
delimiter ;
select * from alumnos_replica; -- no tenemos alumnos
-- Insertamos un nuevo alumno
insert into alumnos values (3, 'Juan');
select * from alumnos_replica; -- aparece el nuevo registro
```



BORRADO TRIGGER Y CONOCER LOS TRIGGERS

```
DROP TRIGGER nombre_trigger
```

```
1    show triggers from curso;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer	character_set_client	collation_connection	Database Collation
trigger1	INSERT	alumnos	begin insert into alumnos_replica values (new.id...	BEFORE	2023-05-09 12:49:33.87	ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLE...	root@localhost	utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci

UT10. SQL MODO PROGRAMACIÓN

Módulo: BASES DE DATOS

Curso 2022/2023. 1° DAM

Ruth Lospitao Ruiz