ÍNDICE

Contenido

Antolín·Muñoz·Chaparro¶
Jefe·de·Proyecto·Sistemas·Informáticos·¶
División·III·de·Aplicaciones·de·Costes·de·Personal·y·Pensiones·Públicas¶
Oficina·de·Informática·Presupuestaria·(DIR3:·EA0027952)¶
Intervención·General·de·la·Administración·del·Estado¶

Antolín·Muñoz·Chaparro¶
Jefe·de·Proyecto·Sistemas·Informáticos·¶
División·III·de·Aplicaciones·de·Costes·de·Personal·y·Pensiones·Públicas¶
Oficina·de·Informática·Presupuestaria·(DIR3:·EA0027952)¶
Intervención·General·de·la·Administración·del·Estado¶

DISPARADORES (TRIGGERS)

El cuarto tipo de bloque PL/SQL nominado que se va a estudiar en este libro es el disparador o trigger.

Los disparadores se asemejan a los subprogramas debido a que son bloques PL/SQL nominados con secciones declarativa, ejecutable y de control de errores.

Al igual que los paquetes, los disparadores deben almacenarse en la base de datos y no pueden ser locales a un bloque.

La diferencia fundamental entre los disparadores y los subprogramas vistos hasta el momento es que mientras que un subprograma se ejecuta explícitamente, cuando es invocado desde una sección ejecutable de un código por su nombre, un disparador o trigger no puede ser invocado explícitamente.

Un disparador se ejecuta de manera implícita cada vez que tiene lugar el suceso del disparo para el que fue creado, mediante la asociación a una tabla.

Un disparador no admite argumentos y no devuelve valores.

El acto que provoca la ejecución del disparador (suceso del disparo), es una operación DML que provoque alteración de una tabla (INSERT, UPDATE o DELETE).

UTILIDAD DE LOS TRIGGERS

Los disparadores pueden emplearse para muchas cosas diferentes, entre las que se incluyen:

- Mantenimiento de restricciones de integridad complejas, que no sean posibles con las restricciones declarativas definidas al crear la tabla.
  Por ejemplo al crear una tabla y definir una columna de clave ajena (FOREIGN KEY), el propio SQL admite la opción DELETE CASCADE, lo que supone que se eliminen en la tabla dependiente todas las filas que contengan el valor que se borre en la tabla padre, para esa columna.
  Pero el lenguaje SQL de Oracle no soporta UPDATE CASCADE, que pertenece al SQL/92 estándar. Esta opción permite que cuando se cambie el valor de la tabla padre, se actualicen todos los registros en la tabla hija, cuya columna se vea afecta. Para poder implementar esto en Oracle hay que crear un trigger.
- Un trigger permite la auditoria de la información contenida en una tabla, registrando los cambios realizados y la identidad del que los llevó a cabo.
- Permiten el aviso automático a otros programas, de que hay que llevar a cabo una determinada acción cuando se realiza un cambio en una tabla.

Sintaxis

```
CREATE [OR REPLACE] TRIGGER nombre_disparador
{BEFORE | AFTER} suceso_disparo ON tabla
[FOR EACH ROW [WHEN condicion_disparo]]

cuerpo_disparador
```

INTEGRIDAD REFERENCIAL

El concepto de integridad referencial se aplica dentro de la creación de una tabla mediante la cláusula FOREIGN KEY. Es decir, cuando manejamos claves ajenas.

Este concepto nos va a permitir mantener una consistencia entre los datos relacionados en la tabla padre e hijo, de manera que las columnas que forman la clave ajena en la tabla hija no podrán tener valores distintos a sus homólogas referenciadas en la tabla padre.

En el lenguaje estándar SQL se aprobaron en su momento las posibilidades de integridad referencial que se podían dar entre columnas de una tabla padre e hijo, pero no todas estas opciones están implementadas en los SGBD de Oracle. A continuación, vemos un cuadro con las opciones que se aprobaron en el lenguaje SQL y las que contienen el SGBD de Oracle.

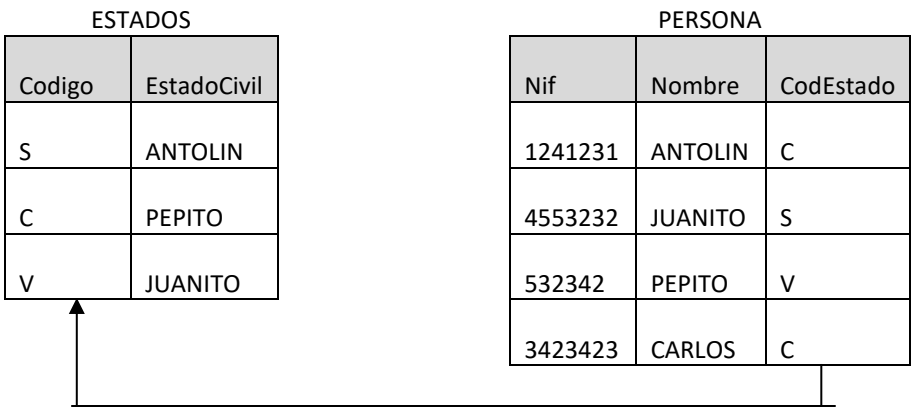
Enunciadas en SQL estándar	Implementadas en Oracle
`ON DELETE RESTRICT`	SI
`ON DELETE CASCADE`	SI
`ON UPDATE RESTRICT`	NO
`ON UPDATE CASCADE`	NO implementada directamente.

Vamos a ver a continuación el significado de cada una, y cómo afectan a los datos. Así mismo, podremos enunciar cómo se indican mediante instrucciones del lenguaje SQL de Oracle.

Antolín·Muñoz·Chaparro¶
Jefe·de·Proyecto·Sistemas·Informáticos·¶
División·III·de·Aplicaciones·de·Costes·de·Personal·y·Pensiones·Públicas¶
Oficina·de·Informática·Presupuestaria·(DIR3:·EA0027952)¶
Intervención·General·de·la·Administración·del·Estado¶
GOBIERNO DE ESPAÑA MINISTERIO DE HACIENDA

On delete restrict

Esta cláusula impide que se borren datos en la tabla padre, siempre y cuando existan en la tabla hija datos en las columnas referenciadas, que coincidan con dicho valor.



Tomando el ejemplo anterior vemos que existe una tabla padre que es `Estados` y una tabla hija `Persona` donde existe una clave ajena que es `Persona.CodEstado` sobre la columna `Estados.Codigo`. Para crear dicha clave ajena, mediante código SQL de Oracle, utilizaríamos un constraint al final de la definición de la tabla `Persona` de la siguiente manera:

```
CONSTRAINT FK_PERSONA_ESTADOS FOREIGN KEY (codestado)
REFERENCES estados(codigo)
```

Si ahora quisiéramos borrar la línea de la tabla `Estados` que contiene en la columna `codigo` el valor de '`C`' (casado), utilizaríamos la siguiente instrucción:

```
DELETE ESTADOS WHERE CODIGO = 'C';
```

El sistema nos devolverá un error indicándonos que no es posible llevar a cabo la operación, porque existe información de una tabla hija que referencia el valor de esta columna. En concreto, en nuestro ejemplo hay 2 columnas con el contenido '`C`' (casado) en la columna `CodEstado`, que son la primera y la última.

On update restrict

Esta cláusula impide que se modifiquen datos en la tabla padre, siempre y cuando existan en la tabla hija datos en las columnas referenciadas, que coincidan con dicho valor.

La indicación de este tipo de restricción mediante sentencia SQL de Oracle es la misma que en la opción anterior:

```
CONSTRAINT FK_PERSONA_ESTADOS FOREIGN KEY (codestado)
REFERENCES estados(codigo)
```

Si ahora quisiéramos actualizar la línea de la tabla `Estados` que contiene en la columna `codigo` el valor de 'V' (viudo), utilizaríamos la siguiente instrucción:

```
UPDATE ESTADOS SET CODIGO = 'X' WHERE CODIGO = 'V';
```

El sistema nos devolverá un error indicándonos que no es posible llevar a cabo la operación, porque existe información de una tabla hija que referencia el valor de esta columna. En concreto, en nuestro ejemplo la 3ª fila tiene el valor 'V' en la columna `CodEstado`.

On delete cascade

Esta cláusula permite que se borren valores en la tabla padre aunque existan valores iguales al borrado en la hija. Pero para mantener la integridad referencial en los datos, también borra todas las filas de la tabla hija que contenga la clave de la fila borrada en el padre.

La indicación de este tipo de restricción mediante sentencia SQL de Oracle es la siguiente:

```
CONSTRAINT FK_PERSONA_ESTADOS FOREIGN KEY (codestado)
REFERENCES estados(codigo)
ON DELETE CASCADE
```

Si ahora quisiéramos borrar la línea de la tabla `Estados` que contiene en la columna `codigo` el valor de 'C' (casado), utilizaríamos la siguiente instrucción:

```
DELETE ESTADOS WHERE CODIGO = 'C';
```

El resultado que obtendremos en cuanto a las tablas de nuestro ejemplo sería el siguiente:

ESTADOS

Codigo	EstadoCivil
C	PEPITO
V	JUANITO

PERSONA

Nif	Nombre	CodEstado
4553232	JUANITO	S
532342	PEPITO	V

On update cascade

Esta cláusula permite que se actualicen valores en la tabla padre aunque existan valores iguales al borrado en la hija. Para mantener la integridad referencial en los datos, también se actualizan todas las filas de la tabla hija que contengan el valor actualizado en el padre.

Esta opción no tiene implementación en Oracle mediante una instrucción directa asociada a un CONSTRAINT, pero se puede llegar a realizar una implementación que posibilite la operación de ON UPDATE CASCADE, creando un TRIGGER asociado a la tabla padre.

ESPACIO DE NOMBRES DEL DISPARADOR

Los procedimientos, funciones, paquetes y tablas entre otros, comparten el mismo espacio de nombres, por lo que sólo puede haber un nombre distinto que se asigne a cualquiera de estos objetos.

Pero los triggers tienen su propio espacio de nombre, así pues podrán tener el mismo nombre que cualquier de los anteriores.

Ejemplo

```
CREATE TABLE PRUEBAS
(……

CREATE PROCEDURE PRUEBAS IS
……..
BEGIN
      ……..
END;
```

Este ejemplo provocaría un error al crear el procedimiento PRUEBAS porque ya existe un objeto denomina PRUEBAS (la tabla) que comparte el mismo espacio de nombres.

```
CREATE TABLE PRUEBAS
(……

CREATE TRIGGER PRUEBAS
……..
BEGIN
      ……..
END;
```

En cambio este segundo ejemplo no provocaría ningún error porque al tener espacios de nombres diferentes la tabla y el trigger, ambos objetos pueden denominarse de la misma forma.

MOMENTO DEL DISPARO

Un trigger se diseña para ejecutarse antes (BEFORE) o después (AFTER) de que se produzca una sentencia DML sobre la tabla a la que se ha asociado el trigger.

SUCESO DEL DISPARO

Los sucesos que provocan el disparo del trigger son sentencias DML que producen alteración en los datos contenidos en la tabla sobre la que se definió el trigger, y en concreto pueden ser 3:

- DELETE: cuando se borre una fila.
- INSERT: cuando se inserte una fila.
- UPDATE: cuando se modifique una fila.

NIVEL DE DISPARO

Nos define el número de veces que se va a ejecutar el trigger una vez que se produzca el suceso de disparo.

En concreto un disparador tiene 2 posibilidades de nivel de disparo:

- Una única vez por sentencia (es el valor por defecto).
- Una vez por fila de la tabla asociada que se vea modificada (FOR EACH ROW).

CONDICIÓN DE DISPARO

Un trigger por fila (FOR EACH ROW) a parte de ejecutarse cuando se produce el suceso de disparo, también se le pueden condicionar por otra circunstancia que se programa en el propio trigger, y que se conoce como condición de disparo.

Ejemplo

```
CREATE OR REPLACE TRIGGER comprueba_salario
BEFORE INSERT OR UPDATE salario, empleo ON emp
FOR EACH ROW WHEN (new.empleo <> 'PRESIDENTE')

DECLARE
      Minsalario NUMBER;
BEGIN
      ........
END;

INSERT INTO emp (empleo) VALUES ('PRESIDENTE');
```

Con esta instrucción el trigger COMPRUEBA_SALARIO no se dispararía, dado que la instrucción de inserción no cumple la condición de disparo necesaria (<> 'PRESIDENTE').

```
INSERT INTO emp (empleo) VALUES ('COMERCIAL');
```

Con esta instrucción sí que se dispararía el trigger COMPRUEBA_SALARIO al cumplirse la condición de disparo necesaria. Al ser un trigger del nivel FOR EACH ROW, se ejecutaría 1 sola vez, porque solo se inserta una fila, en caso de que hubiese más inserciones que cumpliesen la condición de disparo, se ejecutaría tantas veces como filas se insertasen.

SENTENCIAS DE BORRADO Y ALTERACIÓN DE TRIGGERS

Para borrar o alterar un trigger hay que ejecutar las sentencias DDL que se describen a continuación.

Borrado de un trigger

La sintaxis de borrado de un trigger es la siguiente:

```
DROP TRIGGER nombre_trigger;
```

Alteración de un trigger

Un trigger únicamente permite alterar mediante una instrucción DDL el estado en el que se encuentra, es decir: habilitado o deshabilitado. Para cambiar cualquier otro detalle del trigger: nivel de disparo, condición de disparo, tabla asociada, bloque de ejecución, etc., hay que volver a crear el trigger con la instrucción de creación vista al comienzo de este capítulo.

La sintaxis que permite deshabilitar o habilitar el funcionamiento de un trigger con el fin de que se pueda o no disparar cuando se produzca el evento del disparo, es la siguiente:

```
ALTER TRIGGER nombre_trigger {DISABLE | ENABLE}
```

La sintaxis que permite deshabilitar o habilitar todos los triggers asociados a una tabla son los siguientes:

```
ALTER TABLE nombre_trigger {DISABLE | ENABLE} ALL TRIGGERS;
```

USO DE LOS PREDICADOS :OLD Y :NEW

Cuando se produce el suceso que hace disparar el trigger entra en funcionamiento los conceptos de acceso a la fila que está siendo actualmente procesada mediante dos pseudo-registros que se denominan **:old** y **:new**.

Estos pseudo-registros sólo se pueden utilizar con disparadores a nivel de fila y su significado es el siguiente:

- :OLD Maneja los datos originales de la fila en la tabla antes de ser cambiados.
- :NEW Maneja los nuevos datos que van a sustituir a los que hay en :old para esa fila.

Orden de disparo	:OLD	:NEW
INSERT	NULL en todas las columnas	Los nuevos valores a insertar en las columnas
UPDATE	Los valores previos de las columnas antes de actualizar.	Los nuevos valores que se actualizan en las columnas.
DELETE	Los valores previos de las columnas antes de borrarlos.	NULL en todas las columnas.

USO DE LOS PREDICADOS BOOLEANOS

Para poder evaluar el tipo de suceso que ha provocado la ejecución de un trigger disponemos de una serie de predicados booleanos que interrogan al trigger para conocer el suceso que le ha acontecido.

Estos predicados son los que se indican a continuación con su significado:

- INSERTING: Verdadero si el trigger se ejecuta como consecuencia de un INSERT.
- UPDATING: Verdadero si el trigger se ejecuta como consecuencia de un UPDATE.
- DELETING: Verdadero si el trigger se ejecuta como consecuencia de un DELETE.

Ejemplo

```
CREATE OR REPLACE TRIGGER auditoria
BEFORE INSERT OR DELETE OR UPDATE ON emp
FOR EACH ROW
BEGIN
      IF INSERTING THEN
            INSERT INTO TEMPORAL VALURES
            ('Se ha producido una inserción en la tabla');
      ELSIF DELETING THEN
            INSERT INTO TEMPORAL VALURES
            ('Se ha producido un borrado en la tabla');
```

```
      ELSIF UPDATING THEN
            INSERT INTO TEMPORAL VALURES
            ('Se ha producido una inserción en la tabla');
      END IF;
END;
```

TABLAS MUTANTES

Se denomina tabla mutante a aquella que se está modificando actualmente por una orden DML.

Las tablas que pueden necesitar ser actualizadas como resultado de restricciones de integridad referencial DELETE_CASCADE definidas en una cláusula FOREING KEY también se consideran tablas mutantes.

Tablas de restricción

Una tabla de restricción es aquella que se ser necesario leer para resolver una restricción de integridad referencial.

Por ejemplo:

```
      CREATE TABLE estu_registrados ( -- tabla mutante
          Id          NUMBER(5)    NOT NULL,
            Dpto  CHAR(3)     NOT NULL,
            Curso CHAR(3)     NOT NULL,
            Grado CHAR(1)     NOT NULL,
            CONSTRAINT fk_estudiante_id FOREIGN KEY (id)
REFERENCES estudiante(id), -- tabla de restricción
          CONSTRAINT fk_dpto_curso FOREIGN KEY (dpto,
curso)  REFERENCES  clase(dpto,  curso),--  tabla  de
restricción
          CONSTRAINT   fk_grado   FOREIGN   KEY   (grado)
      REFERENCES grados (grado), -- tabla mutante
          CONSTRAINT pk_estu_registrados PRIMARY KEY (id,
dpto, curso)
      );
```

Restricciones en triggers con nivel de fila

No se puede leer o modificar ninguna tabla mutante dentro del código ejecutable del trigger asociada a la tabla mutante.

No se puede leer o modificar columnas de clave primaria únicas, o externas de una tabla de restricción en el cuerpo del trigger asociado a la tabla mutante.

Por ejemplo tomemos en consideración el siguiente caso:

```
Tabla ESTUDIANTE (Clave: ID)
Tabla CLASE (Clave: ID, DPTO, CURSO)

CREATE OR REPLACE TRIGGER cascada
     BEFORE INSERT ON estu_registrados
     FOR EACH ROW
DECLARE
     V_creditos clase.num_creditos%TYPE;
BEGIN
/* La siguiente instrucción es correcta dado que no
lee la clave primaria de la tabla de restricción
 clase. */

SELECT num_creditos INTO v_creditos
FROM clase
WHERE dpto = :new.dpto
AND   curso = :new.curso;

/* Las siguientes instrucciones son también
 correctas, dado que no modifican elementos de clave
primaria de las tablas de restricción */

UPDATE estudiantes
SET creditos_actual = creditos_actual + v_creditos
WHERE id = :new.id;

UPDATE clase
SET num_estudiantes = num_estudiantes + 1
WHERE dpto = :new.dpto
AND curso = :new.curso;
```

Otro ejemplo que podemos considerar es el siguiente:

```
CREATE OR REPLACE TRIGGER incorrecto
BEFORE INSERT OR UPDATE OF dpto
ON estu_registrados FOR EACH ROW
DECLARE
     V_maxestu  NUMBER(3);
BEGIN
/* La siguiente instrucción es incorrecta, dada que
se intenta leer de una tabla mutante */

SELECT count(*)
INTO v_maxestu
FROM estu_registrados
WHERE dpto = :new.dpto;

/* La siguientes instrucción es incorrecta dado que
 actualiza un campo de la clave de la tabla de
restricción. */

UPDATE clase
SET dpto = :new.dpto
WHERE id = :old.id;

END;

SQL> UPDATE estu_registrados
     SET grado = 'A'
     WHERE dpto = 'HISTORIA';

A continuación nos aparecería el siguiente error:
ERROR at line 1:
ORA-04091:   table   ESTU_REGISTRADOS   is   muting,
trigger/fuction may not see it.
```

EJERCICIO RESUELTO PARA COMPROBAR FUNCIONAMIENTO TRIGGER

Comprobar el funcionamiento de un TRIGGER ejecutando un código dado (omitiendo las instrucciones correspondientes a los comentarios):

```
SQL> /* Creamos una tabla denomina ESTUDIANTE */
SQL> CREATE TABLE ESTUDIANTE
  2  (NOMBRE VARCHAR2(10));

Tabla creada.

SQL> /* Creamos una tabla denomina SUCESOS */
SQL> CREATE TABLE SUCESOS
  2  (NOMBRE VARCHAR2(50),
  3  EN_TABLA VARCHAR2(10),
  4  DIA DATE);

Tabla creada.

SQL> /* Creamos un trigger asociado a la tabla estudiante
  2  que se dispara cuando se produce una operación de INSERT
  3  sobre dicha tabla y que tiene como misión insertar una
fila */
SQL> CREATE OR REPLACE TRIGGER GRABA_SUCESO
  2  BEFORE INSERT ON ESTUDIANTE
  3  BEGIN
  4             INSERT       INTO       SUCESOS       VALUES
('TRIGGER','ESTUDIANTE',SYSDATE);
  5  END GRABA_SUCESO;
  6  /

Disparador creado.

SQL> /* Seleccionamos todas las filas de la tabla ESTUDIANTE
*/
SQL> SELECT * FROM ESTUDIANTE;

ninguna fila seleccionada
```

```
SQL> /* Seleccionamos todas las filas de la tabla SUCESOS */
SQL> SELECT * FROM SUCESOS;

ninguna fila seleccionada

SQL> /* Realizamos un select sobre la tabla del sistema
USER_OBJECTS
  2  que nos devuelve el nombre, tipo y estado del trigger
cuyo nombre
  3  empieza por GRA para ver en que estado se encuentran */
SQL>  SELECT  OBJECT_NAME,  OBJECT_TYPE,  STATUS  FROM
USER_OBJECTS
  2  WHERE OBJECT_NAME LIKE 'GRA%';

OBJECT_NAME
------------------------------------------------------------
OBJECT_TYPE   STATUS
------------- -------
GRABA_SUCESO
TRIGGER       VALID

SQL> /* Insertamos una fila en la tabla ESTUDIANTE lo cual
provoca que
  2  se dispare nuestro trigger GRABA_SUCESO */
SQL> INSERT INTO ESTUDIANTE VALUES ('PEPITO');

1 fila creada.

SQL> /* Seleccionamos todas las filas de la tabla ESTUDIANTE
para comprobar
  2  que se ha ejecutado el INSERT anterior. */
SQL> SELECT * FROM ESTUDIANTE;

NOMBRE
----------
PEPITO

SQL> /* Seleccionamos todas las filas de la tabla SUCESOS
para comprobar
  2  que se ha disparado nuestro trigger GRABA SUCESO */
```

```
SQL> SELECT * FROM SUCESOS;

NOMBRE                                                  EN_TABLA
           DIA
-------------------------------------------------- ----------
TRIGGER                                                 ESTUDIANTE
           22/11/00

SQL> /* Nos arrepentimos y deshacemos el INSERT sobre la
tabla ESTUDIANTES
  2    ¿que ocurre entonces con la tabla SUCESOS y su
información generada
  3  por el trigger GRABA_SUCESO? */
SQL> ROLLBACK;

Rollback terminado.

SQL> /* Seleccionamos todas la filas de la tabla ESTUDIANTE
y vemos que tras
  2  el ROLLBACK volvemos al estado inicial que era una tabla
sin filas */
SQL> SELECT * FROM ESTUDIANTE;

ninguna fila seleccionada

SQL>  /*  El  ROLLBACK  deshace  cualquier  operación  de
actualización realizada o sea
  2    que también deja la tabla SUCESOS como estaba al
principio, sin filas
  3  para ello ejecutamos la select que devuelve todos los
datos y vemos que
  4  está vacía */
SQL> SELECT * FROM SUCESOS;

ninguna fila seleccionada

SQL> /* Volvemos a ejecutar la operación de inserción sobre
tabla ESTUDIANTE */
SQL> INSERT INTO ESTUDIANTE VALUES ('PEPITO');

1 fila creada.
```

```
SQL> /* Confirmamos la inserción y la hacemos definitiva */
SQL> COMMIT;

Validación terminada.

SQL> /* Consultamos las filas de la tabla ESTUDIANTE */
SQL> SELECT * FROM ESTUDIANTE;

NOMBRE
----------
PEPITO

SQL> /* Consultamos las filas de la tabla SUCESOS y vemos que
se ha ejecutado
  2  el trigger */
SQL> SELECT * FROM SUCESOS;

NOMBRE                                             EN_TABLA
           DIA
-------------------------------------------------- ----------
TRIGGER                                            ESTUDIANTE
           22/11/00

SQL> /* Borramos la tabla ESTUDIANTE */
SQL> DROP TABLE ESTUDIANTE;

Tabla borrada.

SQL> /* Consultamos la tabla SUCESOS para ver si su estado
ha variado al borrar
  2  la tabla ESTUDIANTE y vemos que no ha variado */
SQL> SELECT * FROM SUCESOS;

NOMBRE                                             EN TABLA
           DIA
-------------------------------------------------- ----------
TRIGGER                                            ESTUDIANTE
           22/11/00

SQL> /* Consultamos la tabla del sistema USER OBJECTS para
ver que ha ocurrido
```

```
  2  con el trigger al borrar la tabla sobre la que estaba
asociado el mismo. El
  3  resultado es que se ha borrado también porque un trigger
depende de una tabla
  4   y si la misma se borra, se eliminan automáticamente
todos los elementos asociados
  5  a ella: Indices, Vistas y Triggers */
SQL> SELECT * FROM USER_OBJECTS WHERE OBJECT_NAME LIKE 'GR%';

ninguna fila seleccionada

SQL> /* Borramos la tabla sucesos */
SQL> drop table sucesos;

Tabla borrada.

SQL> /* Fin del ejemplo de utilización de TRIGGERS */
.
```

EJERCICIO RESUELTO PARA COMPROBAR FUNCIONAMIENTO TRIGGER

Comprobar el funcionamiento de un TRIGGER ejecutando un código dado (omitiendo las instrucciones correspondientes a los comentarios):

```
SQL> /* Creamos una tabla estudiantes */
SQL> CREATE TABLE estudiantes
  2  (ID  NUMBER PRIMARY KEY,
  3  NOMBRE VARCHAR2(25),
  4  APELLIDOS VARCHAR2(30));

Table created.

SQL> /* Creamos una secuencia */
SQL> CREATE SEQUENCE contador
  2  START WITH 1
   3  INCREMENT BY 1
   4  NOCACHE;

Sequence created.
```

```
SQL> CREATE OR REPLACE TRIGGER GeneraID
  2  BEFORE INSERT OR UPDATE ON estudiantes
  3  FOR EACH ROW
  4  BEGIN
  5  SELECT CONTADOR.NEXTVAL INTO :NEW.ID FROM DUAL;
  6  END;
  7  /

Trigger created.

SQL> /* Hacemos una primera inserción en la tabla estudiantes
*/
SQL>  INSERT  INTO  estudiantes  VALUES(10,'Juan','Gabriel
Sanchez');

1 row created.


SQL> /* Vemos que se ha insertado en realidad en tabla
ESTUDIANTES */
SQL> SELECT * FROM estudiantes;

       ID NOMBRE                     APELLIDOS
--------- -------------------------- ----------------------
        1 Juan                       Gabriel Sanchez

SQL> /* Realizamos otro insert en tabla ESTUDIANTES */
SQL>  INSERT  INTO  estudiantes  (NOMBRE,  APELLIDOS)  VALUES
('Pepe','Domingo Castro');

1 row created.

SQL> /* Vemos que se ha insertado */
SQL> SELECT * FROM estudiantes;

       ID NOMBRE                     APELLIDOS
--------- -------------------------- ----------------------
        1 Juan                       Gabriel Sanchez
        2 Pepe                       Domingo Castro
```

```
SQL> /* Como se puede ver por las 2 inserciones el trigger
lo que hace
  2> es dar un valor para el campo ID (el valor del siguiente
número de la secuencia)
  3>independientemente de que se inserte o no un valor para
dicho campo */
SQL> /* Fin de la ejecución del trigger */
.
```