

DISPARADORES EN SQL (II)

MODELOS AVANZADOS DE BASES DE DATOS

Objetivo

Conocer mejor los disparadores en SQL. Se verá uno de los principales inconvenientes que se presentan al trabajar con disparadores, las tablas mutantes. También se verá un último tipo de disparadores (INSTEAD OF) que se utilizan en Oracle8 siempre asociados a vistas de la base de datos.

1. Cuerpo del disparador

En la práctica anterior ya se comentó lo que podía incluirse dentro del cuerpo de un disparador, como sentencias SQL. Pero también, en el cuerpo de un disparador pueden incluirse subprogramas escritos en PL/SQL.

En general, las sentencias de control utilizables por PL/SQL coinciden con las de cualquier lenguaje de alto nivel. Sin embargo, existen una serie de funciones utilizables dentro del cuerpo de un disparador que resultan de gran utilidad. Entre ellas, vamos a estudiar aquí, las funciones `inserting`, `deleting` y `updating`, y la función `RAISE_ERROR_APPLICATION`.

Funciones booleanas `inserting`, `deleting`, `updating`.

Estas funciones se utilizan cuando el evento de un disparador es compuesto, es decir, queremos que el disparador se active ante diferentes operaciones DML pero no queremos que haga lo mismo para cualquiera de los eventos activadores. Con lo visto hasta ahora, la única solución sería diseñar un disparador para cada una de las acciones DML de activación. Sin embargo, con las funciones `inserting`, `updating` y `deleting`, podremos hacer todo en un único disparador.

Ejemplo.

```
CREATE OR REPLACE TRIGGER ejemplo
BEFORE INSERT OR UPDATE OR DELETE ON tabla
BEGIN
    IF DELETING THEN
        Acciones asociadas al borrado
    ELSIF INSERTING THEN
        Acciones asociadas a la inserción
    ELSE
        Acciones asociadas a la modificación
    END IF;
END ejemplo;
/
```

Función `RAISE_APPLICATION_ERROR`

Procedimiento que devuelve mensajes de error al usuario cuando éste va a realizar una operación no permitida y que interrumpe la ejecución del disparador.

Sintaxis: RAISE_APPLICATION_ERROR(nro_erro, mensaje);

Dónde: nro_erro es un número asignado por el usuario que debe estar entre -20000 y -20999 y mensaje es el mensaje que aparecerá por pantalla al usuario cuando se produzca el error.

Ejemplo.

```
CREATE OR REPLACE TRIGGER ejemplo
BEFORE DELETE ON tabla
FOR EACH ROW
BEGIN
IF tabla.columna= valor_no_borrable THEN
    RAISE_APPLICATION_ERROR(-20000, 'La fila no se puede
    borrar');
END IF;
...
END ejemplo;
```

2. Declaración de variables

Dentro de un disparador podemos declarar variables que utilizaremos dentro del cuerpo del mismo.

Sintaxis:

```
CREATE...
BEFORE...
[FOR EACH ROW ...]
DECLARE
    Declaración de variables
BEGIN
    ...
```

La declaración de variables podrá ser:

```
nombre CONSTANT NUMBER:= valor;
(nombre=valor constante)
```

```
nombre TIPO;
(nombre como variable del tipo especificado)
```

```
nombre nombretabla%ROW TYPE;
ROWTYPE es el tipo de los registros OLD y NEW. Con esto crearemos una
variable nombre del tipo de la tabla. Para asignar el valor de una columna se
pondrá como: nombre.nombrecolumna, de esta forma podremos hacer
asignaciones del tipo nombre.nombrecolumna = :OLD.nombrecolumna)
```

nombre nombretabla.nombrecolumna%TYPE
Crea una variable nombre del mismo tipo que el de la columna nombrecolumna de la tabla nombretabla.

Para asignarle valor a una variable se utiliza el signo :=

3. Restricciones de los disparadores

Una tabla mutante es una tabla que está siendo modificada por una operación DELETE, INSERT o UPDATE, o una tabla que se verá afectada por los efectos de un DELETE CASCADE debido a la integridad referencial. Las tablas mutantes sólo deben aparecer en disparadores con nivel de fila. Una tabla sobre la que se define un disparador es una tabla mutante.

Una tabla de restricción de la tabla del disparador es una tabla de la que el disparador puede tener necesidad de leer debido a una restricción de integridad referencial.

Las órdenes SQL del cuerpo de un disparador no pueden:

- Leer o modificar ninguna tabla mutante de la orden que provoca el disparo.
- Leer o modificar las columnas de clave primaria o ajena de una tabla de restricción de la tabla del disparador. Sin embargo pueden modificarse, si se quieren, las otras columnas. Es decir, no se podrá acceder a las claves primarias de las tablas unidas a la tabla sobre la que se define el disparador mediante integridad referencial.

Ejemplo de tabla mutante.

```
CREATE OR REPLACE TRIGGER chequear_salario
BEFORE INSERT OR UPDATE ON empleado
FOR EACH ROW
WHEN (new.trabajo<>'presidente')
DECLARE
    v_salariomin empleado.salario%TYPE;
    v_salariomax empleado.salario%TYPE;
BEGIN
    SELECT MAX(salario), MIN(salario) FROM empleado
        INTO v_salariomin, v_salariomax
    FROM empleado
    WHERE trabajo=:new.trabajo;
    IF :new.salario<v_salariomin OR :new.salario> v_salariomax THEN
        RAISE_APPLICATION_ERROR(-20001, 'Sueldo fuera de rango');
    END IF;
END chequear_salario;
/
```

Al ejecutar la orden

```
UPDATE empleado SET salario=1500
WHERE nombre_emp = 'Cortecero';
```

Dará un error de tabla mutante ya que el disparador está intentando obtener información de la tabla sobre la que está definido (tabla empleado) y que, por lo tanto, es mutante.

Ejemplo de tabla de restricción. Supongamos dos tablas, empleado y departamento, de forma que en la tabla empleado tenemos almacenado el número de departamento, definido como clave ajena que referencia a la tabla departamento.

```
CREATE TRIGGER ejemplo
AFTER UPDATE ON nrodep OF departamento
FOR EACH ROW
BEGIN
    UPDATE empleado
    SET empleado.dep = :NEW.nrodep
    WHERE empleado.dep= :OLD.nrodep;
END ejemplo;
/
```

Al ejecutar:

```
UPDATE departamento
SET nrodep= 1
WHERE nrodep=7;
```

Dará un error de tabla mutante ya que estamos intentando modificar una columna correspondiente a una restricción de integridad.

4. Disparadores de sustitución (INSTEAD OF)

Los disparadores de sustitución (INSTEAD OF) sólo pueden definirse sobre vistas en Oracle 8 y se activan en lugar de la orden DML que provocó el disparo. Los disparadores de sustitución deben estar a nivel de fila.

Ejemplo.

```
CREATE VIEW vista AS
SELECT edificio, sum(numero_asientos) FROM habitaciones
GROUP BY edificio;
```

Es ilegal hacer una operación de borrado directamente en la vista:

```
DELETE FROM vista WHERE edificio='edificio 7';
```

Sin embargo, se puede crear un disparador de sustitución que efectúe el borrado equivalente pero sobre la tabla habitaciones.

```
CREATE TRIGGER borra_en_vista
INSTEAD OF DELETE ON vista
FOR EACH ROW
BEGIN
    DELETE FROM habitaciones
    WHERE edificio = :OLD.edificio;
END borra_en_vista;
```

Bibliografía

- Owens, K.T. (1996) *Building Intelligent Databases with ORACLE PL/SQL, TRIGGERS & STORED PROCEDURES*, Ed. Prentice Hall. 1996
- *ORACLE Guía de aprendizaje*, Ed. ORACLE Press
- *ORACLE, Lenguaje de programación PL/SQL*, Ed. ORACLE Press
- *La biblia de ORACLE8*, Advanced Information Systems Inc., Ed. Anaya Multimedia. 1998.
- Cannan, S.J.,(1999), *The SQL Files, investigation into a database language*, Array Publications. 1999.

EJERCICIOS

Crear las siguientes tablas con el siguiente contenido:

```
CREATE TABLE empleado
(nss NUMBER,
 nombre CHAR(10),
 dep NUMBER,
 PRIMARY KEY (nss));
```

```
CREATE TABLE departamento
(nro_dep NUMBER,
 nombre CHAR(20),
 PRIMARY KEY (nro_dep));
```

```
INSERT INTO empleado VALUES(1, 'Arturo', 30);
INSERT INTO empleado VALUES(2, 'Eva', 20);
INSERT INTO empleado VALUES(3, 'Miguel', 10);
INSERT INTO empleado VALUES(4, 'Alejandro', 10);
INSERT INTO empleado VALUES(5, 'Elena', 50);
INSERT INTO empleado VALUES(6, 'Carmen', 40);
INSERT INTO empleado VALUES(7, 'Mario', 30);
INSERT INTO empleado VALUES(8, 'Esteban', 10);
INSERT INTO empleado VALUES(9, 'Paco', 20);
INSERT INTO empleado VALUES(10, 'Pili', 30);
```

```
INSERT INTO departamento VALUES(10, 'Informática');
INSERT INTO departamento VALUES(20, 'Ventas');
INSERT INTO departamento VALUES(30, 'Compras');
INSERT INTO departamento VALUES(40, 'Publicidad');
INSERT INTO departamento VALUES(50, 'Recursos Humanos');
```

1. Crear un disparador para controlar la no inserción de una fila en la tabla empleado cuando no sea horario laboral.

Intentar ejecutar la siguiente orden y comprobar si ha habido algún cambio en la tabla:

```
INSERT INTO empleado VALUES (11, 'Pepe', 30);
```

Notas:

Se supone que el horario laboral es de nueve de la mañana a cinco de la tarde y de lunes a viernes.

Cambiar la fecha del ordenador para que sea sábado o domingo.

2. Crear el mismo disparador pero utilizando un procedimiento en el cuerpo del disparador que controle si la hora y el día son correctos.

Un procedimiento se crea utilizando la estructura

```
CREATE OR REPLACE PROCEDURE nombre
IS
BEGIN
    Cuerpo del procedimiento
END nombre;
/
```

Desde un disparador se puede llamar a un procedimiento mediante su nombre en el cuerpo del disparador.

Intentar ejecutar la siguiente orden y comprobar si ha habido algún cambio en la tabla:

```
INSERT INTO empleado VALUES (11, 'Pepe', 30);
```

3. Crear un disparador que impida cualquier operación en la tabla empleado fuera del horario laboral dando una indicación específica de la operación que no es realizable y porqué no es realizable.
4. Añadir a la tabla empleado una nueva columna salario dónde se almacenará el valor del departamento al que un empleado pertenezca multiplicado por 100. Crear una tabla (cambios) dónde se almacenarán, mediante la utilización de un disparador, los cambios que se han llevado a cabo en la tabla empleado de forma que ante cada cambio de departamento o de salario de un empleado, la nueva tabla almacenará el identificador del empleado(nss), el antiguo y el nuevo salario (v_salario, n_salario), o el antiguo y el nuevo departamento (v_dep, n_dep) junto con la fecha en que el cambio tuvo lugar.

Utilizar variables para la construcción del disparador.

Probar las siguientes consultas y mostrar los resultados obtenidos.

```
UPDATE empleado SET salario= salario*2 WHERE dep=10;
UPDATE empleado SET dep = salario/200 where salario>2000;
```

5. Añadir a la tabla empleado una nueva columna (med) que contendrá la media del salario que el trabajador ha tenido en la empresa. Esta columna tendrá el salario actual del trabajador hasta que este salario varíe y en ese momento pasará a valer la suma del salario anterior, el salario actual y la media de salario del empleado dividido entre tres.
¿Qué ocurre?. ¿Porqué?

6. Crear una vista (vista_seis) dónde se almacenen el identificador y la media de los salarios de cada departamento.

La estructura general de una vista en Oracle 8 es:

```
CREATE OR REPLACE VIEW vista
AS SELECT columnas_de_la_vista
FROM tabla
WHERE condición_de_selección_de_las_columnas;
```

Realizar lo necesario para conseguir ejecutar la orden:

```
DELETE FROM vista_seis WHERE iddep=20;
```