Departamento de Lenguajes y Sistemas Informáticos
E.T.S. Ingeniería Informática. Universidad de Sevilla
Avda Reina Mercedes s/n. 41012 Sevilla
Tlf/Fax 954 557 139 **E-mail** lsi@lsi.us.es **Web** www.lsi.us.es
Diseño de Bases de Datos

e.t.s. ingeniería informática

Diseño de bases de datos

Práctica-2

Triggers

Sevilla, enero 2008
V 2008.1.1

Indice

1 Introducción.

Un trigger ó disparador es un objeto procedimental asociado a una tabla.

Están constituidos por bloques PL/SQL y sentencias SQL.

Se almacena en la base de datos y Oracle lo ejecuta implícitamente (se dispara) cuando se modifican los datos de la tabla (comandos DML de SQL).

Los Triggers tienen múltiples aplicaciones por ej.:

- Forzar **reglas de integridad de los datos complejas** y que no se pueden definir declarando Restricciones de Integridad.
- Realizar cambios en la base de datos de forma transparente al usuario.
- Generar automáticamente valores de columnas derivadas en base a un valor proporcionado por una sentencia INSERT ó UPDATE.
- Sincronizar el mantenimiento de tablas duplicadas que están localizadas en nodos distintos de una base de datos distribuida.

Los Triggers frente a las Restricciones de Integridad, no se aplican a los datos almacenados en la base de datos antes de su definición; sólo se aplican cuando, una vez creados, se ejecutan comandos que manipulan las tablas sobre las que están definidos.

2 Creación de triggers.

Para definir los triggers se utiliza el comando SQL:

CREATE OR REPLACE TRIGGER

La definición de un trigger tiene tres partes:

- **Comando**:
  Se especifica el comando DML de SQL que dispara el Trigger (Delete, Insert ó Update), y la tabla asociada.
  Se puede especificar cuándo se dispara el Trigger:
  - BEFORE, antes de ejecutar el comando,
  - AFTER, después de ejecutar el comando.
- **Restricción:**
  Se define la condición que tiene que verificar cada fila de la tabla para que se ejecute la acción del trigger
- **Acción:**
  Se define mediante comandos y sentencias SQL y PL/SQL, la tarea específica que realiza el Trigger.

Cuando Oracle compila un trigger, guarda el código objeto en un área compartida de la SGA (System Global Area). Si un comando DML de SQL dispara un trigger, Oracle comprueba si el objeto está en la SGA. Si está, lo ejecuta; en caso contrario, lo compila, lo guarda en la SGA y finalmente lo ejecuta.

Por tanto, Oracle compila un trigger cada vez que se dispara y el código objeto no está en la SGA.

2.1 Expresiones.

Para referirse al valor nuevo y al antiguo de una columna de una fila de la tabla, se utilizan los prefijos **:OLD** y **:NEW**

- Cuando estamos haciendo una modificación (UPDATE) de una fila podemos referirnos al valor antes de ser modificado (:OLD) y al valor después de la modificación (:NEW).
- Al introducir valores nuevos (INSERT) podemos referenciar sólo el valor nuevo (:NEW).
- Al borrar (DELETE) podemos referenciar sólo el valor antiguo (:OLD).

Ejemplo:

```
:NEW.salario > (:OLD.salario*1.25)
                /* Compara si el salario ha aumentado en más del 25% de salario antiguo*/
```

2.2 Activación de triggers.

Un triggers puede estar activado (ENABLE) ó desactivado (DISABLE).
Cuando está desactivado no ejecuta ninguna acción.

Hay dos comandos SQL para activar un trigger es:

```
ALTER TRIGGER nombre_trigger ENABLE;

ALTER TABLE nombre_tabla ENABLE ALL TRIGGERS;
```

Cuando se crean están activos.

2.3 Desactivación de triggers.

Es conveniente desactivar los triggers sobre una tabla cuando se hace una carga masiva de datos.

```
ALTER TRIGGER nombre_trigger DISABLE;
ALTER TABLE nombre_tabla DISABLE ALL TRIGGERS;
```

2.4 Borrado de triggers.

```
DROP  TRIGGER nombre_trigger;
```

2.5 Documentación de triggers.

Se pueden consultar las vistas:

- USER_TRIGGERS
- ALL_TRIGGERS
- DBA_TRIGGERS

2.6 Ejemplo de triggers.

Regla: Empleados que supervisa un jefe <=5

```
CREATE TRIGGER jefes
        BEFORE INSERT OR UPDATE OF cojefe ON Empleado
        FOR EACH ROW
        DECLARE supervisa INTEGER;
        BEGIN
                /*Seleccionar empleados que supervisa el jefe*/
                /*Comprobar si supervisa >  5 empleados*/
END;
```

En este ejemplo, se define un trigger para forzar que los datos de la tabla Empleado verifiquen la siguiente regla: "Un jefe no puede supervisar a más de cinco empleados".

Para asegurar que cuando se modifique la tabla Empleado se cumpla esta restricción, tenemos que definir un trigger asociado a la tabla Empleado (ON Empleado).
Este trigger se dispara antes de insertar (Before Insert) un nuevo empleado ó antes de actualizar el atributo cojefe (Before Update Of cojefe) de un empleado que ya existe.

Cuando se dispara el trigger para cada fila (For Each Row), Oracle ejecuta implícitamente la acción: se comprueba si el número de empleados que supervisará el jefe si se realiza la modificación no supera a cinco. La modificación de la tabla Empleado (insertar un nuevo empleado o modificar el atributo cojefe de uno que ya existe) se llevará a cabo, si después de realizarla los datos verifican la regla.

En el bloque PL/SQL que define la acción, podemos acceder a los valores anteriores y nuevos de las columnas: OLD.columna y NEW.columna.

Esta regla de integridad de los datos de la tabla Empleado no podemos forzarla declarando Restricciones de Integridad.

Ejercicio 1.

Crear un trigger sobre la tabla empleados para que no se permita que un empleado sea jefe de más de cinco empleados.

```
DROP TABLE empleados;
CREATE TABLE empleados
(dni                    char(4),
nomemp                  varchar2(15),
cojefe                  char(4),
PRIMARY KEY  (dni),
FOREIGN KEY (cojefe) references empleados on delete cascade);
```

```
CREATE OR REPLACE TRIGGER jefes
        BEFORE
        INSERT ON empleados
        FOR EACH ROW
DECLARE
        supervisa INTEGER;
BEGIN
        SELECT count(*) INTO supervisa
        FROM empleados
        WHERE cojefe = :new.cojefe;

        IF supervisa > 4
                THEN raise_application_error
                (-20600,:new.cojefe||'no se puede supervisar mas de 5');
        END IF;
END;
/
--
-- Inserta datos de ejemplo en la tabla
INSERT INTO empleados VALUES ('D1','Director',null);
INSERT INTO empleados VALUES ('D2','D.Comercial','D1');
INSERT INTO empleados VALUES ('D3','D.Producción','D1');
INSERT INTO empleados VALUES ('D4','Jefe Ventas','D2');
INSERT INTO empleados VALUES ('D5','Jefe Marketing','D2');
INSERT INTO empleados VALUES ('D6','Vendedor 1','D4');
INSERT INTO empleados VALUES ('D7','Vendedor 2','D4');
INSERT INTO empleados VALUES ('D8','Vendedor 3','D4');
INSERT INTO empleados VALUES ('D9','Vendedor 4','D4');
INSERT INTO empleados VALUES ('D10','Obrero 1','D3');
INSERT INTO empleados VALUES ('D11','Obrero 2','D3');
INSERT INTO empleados VALUES ('D12','Obrero 3','D3');
INSERT INTO empleados VALUES ('D13','Secretaria','D5'
```

Para contar los empleados a los que supervisa cada jefe:

```
SELECT cojefe, count(*) FROM empleados GROUP BY cojefe;
```

Ejercicio 2.

Modificar la tabla empleados añadiendo el campo ***salario*** (integer). Añadirle un valor por defecto para dicho campo.

```
ALTER TABLE empleados ADD salario integer DEFAULT 1000;
```

Crear un trigger para impedir que se aumente el salario de un empleado en más de un 20%. Es necesario comparar los valores :old.salario y :new.salario cuando se modifica (BEFORE UPDATE).

```
IF :NEW.salario > :OLD.salario*1.20          THEN raise…
```

Ejercicio 3.

Crear una tabla empleados_baja con la siguiente estructura:

```
CREATE TABLE empleados_baja
(dni            char(4) PRIMARY KEY,
nomemp          varchar2(15),
cojefe          char(4),
salario         integer,
usuario         varchar2(15),
fecha           date );
```

Crear un trigger que inserte una fila en la tabla empleados_baja cuando se borre una fila en la tabla empleados. Los datos que se insertan son los correspondientes al empleado que se da de baja en la tabla empleados.
En las columnas usuario y fecha se grabarán las variables del sistema USER y SYSDATE que almacenan el usuario y fecha actual.
El comando que dispara el trigger es AFTER DELETE.

```
CREATE OR REPLACE TRIGGER bajas
        AFTER DELETE ON empleados
        FOR EACH ROW
BEGIN
        INSERT INTO empleados_baja VALUES (:old.dni, …., USER, SYSDATE)
```

Ejercicio 4.

Visualizar los trigger definidos sobre una tabla consultando la vista ALL_TRIGGERS.

```
DESC ALL_TRIGGERS
SELECT trigger_name, status FROM ALL_TRIGGERS WHERE table_name = 'empleados';
```

Desactivar (DISABLE) y activar (ENABLE) los trigger definidos sobre una tabla:

```
ALTER TABLE empeados DISABLE ALL TRIGGERS;
```

Activar y desactivar un trigger especifico: ALTER TRIGGER jefes DISABLE;

Ver la descripción de un trigger:

```
SELECT description FROM USER_TRIGGERS WHERE trigger_name = 'JEFES';
```

Ver el cuerpo de un trigger:

```
SELECT trigger_body  FROM USER_TRIGGERS WHERE trigger_name = 'JEFES';
```

Ejercicio 5.

Modificar la tabla empleados añadiendo el campo ***departamento*** (integer) (ALTER TABLE ..).

Crear un trigger para impedir que el salario total por departamento (suma de los salarios de los empleados por departamento) sea superior a 10.000.

Será necesario distinguir si se trata de una modificación o de una inserción.

Cuando se trate de una inserción (**IF INSERTING**...) se comprobará que el salario del empleado a insertar (:**NEW**.salario) más el salario total del departamento al que pertenece dicho empleado no es superior a 10.0000.

Cuando se trate de una modificación (**IF UPDATING**..), al salario total del departamento se le sumará el :**NEW**.salario y se le restará el :**OLD**.salario.

Ejercicio 6.

Crear un trigger para impedir que un empleado y su jefe pertenezcan a departamentos distintos.

3 Tablas mutantes.

3.1 Concepto de tabla mutante

Cuando se define un trigger a nivel de fila (cláusula FOR EACH ROW), con eventos BEFORE o AFTER y dentro del cuerpo PL/SQL se referencia la misma tabla a la que está vinculado el trigger se presenta el error:

ORA-04091: **table** DBD.emp is **mutating**, trigger/function may not see **ORA**-06512: at "DBD.emp", line 4 **ORA**-04088: **error** during execution of trigger 'DBD.TriggerName'.

Una tabla en esta situación, en el entorno ORACLE SERVER, es una tabla mutante. El funcionamiento del trigger en cuestión será anómalo.

3.2 Ejemplo

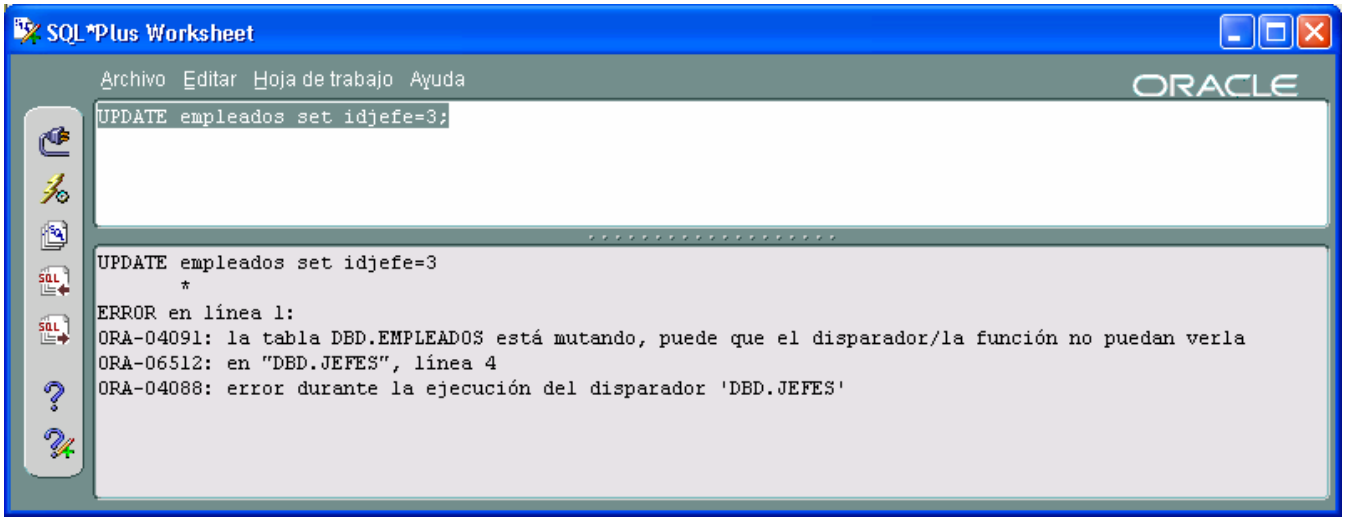
```
DROP TABLE empleados;
CREATE TABLE  empleados
  (     idempleado              NUMBER,
        empleado                VARCHAR2(4000),
        idjefe          NUMBER,
        salario         NUMBER(9,2),
        dpto                    VARCHAR2(20),
         PRIMARY KEY (idempleado),
         FOREIGN KEY (idjefe)
                REFERENCES  empleados (idempleado) ON DELETE CASCADE ENABLE   );

CREATE OR REPLACE TRIGGER jefes
        BEFORE INSERT OR UPDATE OF idjefe ON empleados
        FOR EACH ROW
DECLARE
        V_supervisados INTEGER;
BEGIN
        SELECT count(*) INTO V_supervisados
        FROM empleados
WHERE       idjefe  = :new.idjefe;

        IF V_supervisados > 4
                THEN raise_application_error
                (-20600,:new.idjefe||'no se puede supervisar mas de 5');
        END IF;
END;
/
```

- Si se hacen inserciones fila a fila el trigger no da el error de tabla mutante.

- Si se intenta insertar un conjunto de filas (INSERT INTO...SELECT...) o se ejecuta una modificación de un conjunto (Ej.UPDATE empleados SET idjefe = '2' ;) se produce error debido a la referencia a **:new.idjefe** en el cuerpo del trigger, ya que **idjefe** es una columna que está siendo modificada por las instrucciones (INSERT o UPDATE) que causan el disparo del trigger.



3.3 Solución propuesta tablas mutantes

Hay que evitar la referencia a las variables en modificación en el cuerpo del trigger. Se propone crear dos triggers y un paquete para almacenar en memoria las variables referenciadas:

a) **Trigger BEFORE a nivel de fila**. En el trigger a nivel de fila (for each row) se almacenan los datos que se referencian (los que provocan el error de tabla mutante). La alternativa propuesta es almacenarlos en variables en memoria públicas (visibles por distintos elementos de código PL/SQL); en concreto, se propondrá crear un paquete para recoger dichas variables (aunque pueden diseñarse otras alternativas).

b) **Trigger AFTER a nivel de instrucción.** En el trigger se realiza la referencia a las variables que se han almacenado en el paquete.

Ejemplo:

Crear la tabla de empleados con las siguientes restricciones:
- Un empleado no puede supervisar a más de dos empleados
- El presupuesto total por departamento no puede ser superior a 1000.

```
DROP TABLE Empleados;
CREATE TABLE  "EMPLEADOS"
    (  "IDEMPLEADO"      NUMBER,
       "EMPLEADO"        VARCHAR2(4000),
       "IDJEFE"          NUMBER,
       "SALARIO"         NUMBER(9,2),
       "DPTO"            VARCHAR2(20),
        CONSTRAINT "EMPLEADOS_PK" PRIMARY KEY ("IDEMPLEADO") ENABLE,
        CONSTRAINT "EMPLEADOS_CON" FOREIGN KEY ("IDJEFE")
             REFERENCES  "EMPLEADOS" ("IDEMPLEADO") ON DELETE CASCADE ENABLE   )
/

CREATE OR REPLACE PACKAGE  "RI_EMP" as
   NewEmp Empleados%rowtype;
end;
/

CREATE OR REPLACE TRIGGER  "EMPLEADOS_JEFES_AFTER"
AFTER delete or insert or update on "EMPLEADOS"
DECLARE V_Supervisados INTEGER; VMsg VARCHAR(50);
begin
      SELECT count(*) INTO V_Supervisados FROM empleados
```

```
      WHERE IdJefe = RI_Emp.Newemp.IdJefe;
      IF INSERTING THEN
            VMsg:=CONCAT('Insertando :',RI_Emp.Newemp.IdEmpleado);
      ELSE
            IF UPDATING THEN
                  VMsg:=CONCAT('Modificando :',RI_Emp.Newemp.IdEmpleado);
            ELSE
                  VMsg:=CONCAT('Borrando :',RI_Emp.Newemp.IdEmpleado);
                  DBMS_OUTPUT.PUT_LINE(Vmsg);
            END IF;
      END IF;
      IF V_Supervisados>2 THEN
            RAISE_APPLICATION_ERROR(-20001,VMsg||' Error cuenta supervisados de: '
            ||RI_Emp.Newemp.IdJefe);
      END IF;
end;
/
ALTER TRIGGER  "EMPLEADOS_JEFES_AFTER" ENABLE
/

CREATE OR REPLACE TRIGGER  "Empleados_Before"
BEFORE delete or insert or update on "EMPLEADOS"
for each row
begin
      RI_Emp.Newemp.IdEmpleado := :new.IdEmpleado;
      RI_Emp.Newemp.IdJefe := :new.IdJefe;
      RI_Emp.Newemp.Dpto := :new.Dpto;
      RI_Emp.Newemp.Salario := :new.Salario;
end;
/
ALTER TRIGGER  "Empleados_Before" ENABLE
/


CREATE OR REPLACE TRIGGER  "PRESUPUESTO_DPTOS"
AFTER insert or update on "EMPLEADOS"
DECLARE Vpresupuesto Empleados.salario%type;
      Vmsg VARCHAR(50) DEFAULT 'Presupuesto excedido para Dpto: ';
begin
      SELECT SUM(salario) INTO VPresupuesto FROM Empleados
      WHERE Dpto=RI_Emp.Newemp.Dpto;

      Vmsg:= CONCAT(Vmsg,RI_Emp.Newemp.Dpto);

      IF Vpresupuesto >1000 THEN
            RAISE_APPLICATION_ERROR(-20001,Vmsg);
      END IF;

end;
/
ALTER TRIGGER  "PRESUPUESTO_DPTOS" ENABLE
/
```