

# Indice

1	INTRODUCCIÓN.....	3
2	CREACIÓN DE TRIGGERS.....	3
2.1	EXPRESIONES. ....	4
2.2	ACTIVACIÓN DE TRIGGERS.....	4
2.3	DESACTIVACIÓN DE TRIGGERS.....	4
2.4	BORRADO DE TRIGGERS.....	4
2.5	DOCUMENTACIÓN DE TRIGGERS.....	4
2.6	EJEMPLO DE TRIGGERS.....	5
	EJERCICIO 1.....	6
	EJERCICIO 2.....	7
	EJERCICIO 3.....	7
	EJERCICIO 4.....	7
	EJERCICIO 5.....	8
	EJERCICIO 6.....	8
3	TABLAS MUTANTES.....	9
3.1	CONCEPTO DE TABLA MUTANTE.....	9
3.2	EJEMPLO .....	9
3.3	SOLUCIÓN PROPUESTA TABLAS MUTANTES .....	10

## 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  $\leq 5$

```
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 empleados DISABLE ALL TRIGGERS;
```

Activar y desactivar un trigger específico: `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.