Índice

1. CREACIÓN DE DISPARADORES

Se asemejan a los procedimientos y funciones en que son bloques PL/SQL nominados con secciones declarativa, ejecutable y manejo de excepciones. Los disparadores deben almacenarse en la base de datos y no pueden ser locales a un bloque. Sin embargo, un procedimiento se ejecuta de manera explícita desde otro bloque mediante una llamada de procedimiento, que puede también pasar argumentos**; un disparador, por el contrario, se ejecuta de manera implícita cada vez que tiene lugar el suceso de disparo, y el disparador no admite argumentos.**

El acto de ejecutar un disparador se conoce como *'disparo'*. El suceso de disparo es una operación DML (INSERT, UPDATE o DELETE) sobre una tabla de la base de datos. Los disparadores pueden emplearse por muchas cosas diferentes, incluyendo:

- El mantenimiento de restricciones de integridad complejas, que no sean posibles con las restricciones declarativas definidas en el momento de crear la tabla.
- La auditoría de la información contenida en una tabla, registrando los cambios realizados y la identidad del que los llevó a cabo.
- El aviso automático a otros programas de que hay que llevar a cabo una determinada acción, cuando se realiza un cambio en una tabla.

La sintaxis general para un disparador es:

```
CREATE [ OR REPLACE ] TRIGGER nombre_disparador
{ BEFORE | AFTER } suceso_disparo ON referencia_tabla
[ FOR EACH ROW [ WHEN condición_disparo ]]
cuerpo_disparador;
```

Donde:

nombre_disparador	es el nombre del disparador
suceso_disparo	especifica cuándo se activa el disparador
referencia_tabla	es la tabla para la cual se define el disparador
cuerpo_disparador	incluida en la cláusula WHEN, si es que está presente. El cuerpo del disparador se ejecuta sólo cuando dicha condición se evalúa como verdadera.

2. TIPOS DE DISPARADORES

El suceso de disparo determina el tipo de disparador. Los disparadores pueden definirse para las operaciones INSERT, UPDATE o DELETE, y pueden dispararse antes o después de la operación. Finalmente, el nivel de los disparadores puede ser la fila o la orden. Los valores de la orden, de la temporización y del nivel determinan el tipo del disparador. Hay un total de 12 tipos posibles (3 órdenes por 2 opciones de temporización, por 2 niveles), estos 12 tipos pueden definirse en una tabla, uno de cada tipo, sin embargo, una tabla puede tener más de un disparador de cada tipo. Esta capacidad permite definir cuantos disparadores se quieran para una tabla. Un disparador puede también activarse para más de un tipo de orden de disparo. El disparador **UpdateMajorStats**, por ejemplo, se activa con las órdenes INSERT, UPDATE y DELETE. El suceso de disparo especifica una o varias operaciones DML que deben activar el disparador.

Tipos de disparadores		
Categoría	**Valores**	**Comentarios**
Orden	INSERT, DELETE, UPDATE	Define que tipo de orden DML provoca la activación del disparador
Temporización	BEFORE, AFTER	Define si el disparador se activa antes o después de que se ejecute la orden (disparador previo o posterior)
Nivel	Fila u Orden	Los disparadores con nivel de fila se activan una vez por cada fila afectada por la orden que provocó el disparo. Los disparadores con nivel de orden se activan sólo una vez, antes o después de la orden. Los disparadores con nivel de fila se identifican por la cláusula FOR EACH ROW en la definición del disparador

3. RESTRICCIONES

- Un disparador no puede emitir ninguna orden de control de transacciones: COMMIT, ROLLBACK o SAVEPOINT. El disparador se activa como parte de la ejecución de la orden que provocó el disparo, y forma parte de la misma transacción que dicha orden. Cuando la orden que provoca el disparo es confirmada o cancelada, se confirma o cancela también el trabajo realizado por el disparador.

- Por razones idénticas, ningún procedimiento o función llamado por el disparador puede emitir órdenes de control de transacciones.

- El cuerpo del disparador no puede contener ninguna declaración de variables LONG o LONG RAW. Asimismo, renuncia a columnas de tipo LONG o LONG RAW de la tabla sobre la que se define el disparador.

- Existen restricciones acerca de a qué tablas puede acceder el cuerpo de un disparador. Dependiendo del tipo de disparador y de las restricciones que afecten a las tablas, dichas tablas pueden ser **mutantes**.

"A **mutating table** is a table that is currently being modified by an update, delete, or insert statement. When a trigger tries to reference a table that is in state of flux (being changed), it is considered "mutating" and raises an error since Oracle should not return data that has not yet reached its final state"

Una tabla mutante es por tanto una tabla que está siendo modificada. Si un 'trigger' hace referencia a la propia tabla sobre la que se dispara se producirá una excepción con error ORA-04091. Según esto, cualquier 'trigger' que haga una simple consulta sobre la tabla que lo dispara generará dicho ORA-04091.

4. ELIMINACIÓN Y DESHABILITACIÓN

Al igual que los procedimientos y paquetes, también los disparadores pueden eliminarse. La sintaxis es:

```
DROP TRIGGER nombre_disparador;
```

Esta orden elimina el disparador de forma permanente del diccionario de datos. Al igual que con los subprogramas, puede incluirse la cláusula OR REPLACE en la orden CREATE cuando lo creamos. En este caso se elimina en primer lugar el disparador, si es que ya existe.

A diferencia de los procedimientos y paquetes, sin embargo, se puede deshabilitar un disparador sin necesidad de eliminarlo. Cuando deshabilitamos un disparador continúa existiendo en el diccionario de datos, pero no puede llegar a dispararse. La sintaxis para deshabilitar un disparador es:

```
ALTER TRIGGER nombre_disparador [DISABLE | ENABLE];
```

Todos los disparadores, en un principio, están habilitados en el momento de su creación. ALTER TRIGGER puede deshabilitar y luego volver a habilitar cualquier disparador.

5. ORDEN DE ACTIVACIÓN

Los disparadores se activan al ejecutarse la orden DML. El algoritmo de ejecución de una orden DML es el siguiente:

- Ejecutar, si existe, el disparador de tipo BEFORE con nivel de orden
- Para cada fila a la que afecta la orden:
  - Ejecutar, si existe, el disparador de tipo BEFORE con nivel de fila
  - Ejecutar la propia orden
  - Ejecutar, si existe, el disparador de tipo AFTER con nivel de fila
- Ejecutar, si existe, el disparador de tipo AFTER con nivel de orden

6. UTILIZACION DE :OLD Y :NEW en los DISPARADORES

Un disparador con nivel de fila se ejecuta una vez por cada fila procesada por la orden que provoca el disparo. Dentro del disparador, puede accederse a la fila que está siendo actualmente procesada utilizando, para ello, dos p*seudo-registros*, **:old** y **:new**.
Aunque sintácticamente se los trae como tales, no son verdaderamente registros. Realmente estos registros son útiles por las ventajas a la hora de mantener los datos actualizados y ahorrarnos trabajo

Sentencia Disparadora	:OLD	:NEW
INSERT	No Definida	Valores que se meterán cuando se ejecute la sentencia
UPDATE	Valores originales del registro antes de la modificación	Los valores nuevos de la modificación
DELETE	Los valores originales antes del borrado	No Definido

El problema es que esto solo funciona con las filas (rows), por lo que si se quisiera realizar esto con todas las filas (a nivel de bloque) nos daría un error de compilación. Para esto se utiliza la cláusula WHEN

WHEN *condición*

7. EJERCICIOS RESUELTOS

Ejercicio número: 1

Crear una tabla denominada borrado_empleados que contiene los campos cod_empleado, apellido, nombre y cod_departamento

```
CREATE TABLE BORRADO_EMPLEADOS (
COD_EMPLEADO NUMBER(4,0),
APELLIDO VARCHAR2 (15),
NOMBRE VARCHAR2 (15),
COD_DEPARTAMENTO NUMBER(2,0)
);
```

Una vez creada la tabla. Crear un disparador en el que antes de borrar de la tabla employee un empleado lo inserte en la tabla borrado_empleados

```
CREATE OR REPLACE TRIGGER TR_BORRADO_EMPLEADOS
  BEFORE DELETE
  ON EMPLOYEE
  FOR EACH ROW
BEGIN
  INSERT INTO BORRADO_EMPLEADOS
  VALUES (:OLD.EMPLOYEE_ID,:OLD.LAST_NAME,:OLD.FIRST_NAME,:OLD.DEPARTMENT_ID);
END;
/
```

Inserto dos empleados que no tengan jefe para que se puedan borrar

```
INSERT INTO EMPLOYEE
VALUES(500,'PEREZ','JUAN','Q',669,NULL,'17/12/85',1200,300,10);

INSERT INTO EMPLOYEE
VALUES(600,'PEREZ','JUAN','Q',669,NULL,'17/12/85',1200,300,23);
```

Borro el empleado cuya employe_id es 500 y compruebo que está en la tabla borrado_empleados

```
DELETE FROM EMPLOYEE
WHERE EMPLOYEE_ID=500;

SELECT *
FROM BORRADO_EMPLEADOS;
```

	COD_EMPLEADO	APELLIDO	NOMBRE	COD_DEPARTAMENTO
1	500	PEREZ	JUAN	10

Borro el empleado cuya employe_id es 600 y compruebo que ahora están en la tabla borrado_empleados el de id 500 y el de id 600

```
DELETE FROM EMPLOYEE
WHERE EMPLOYEE_ID=600;

SELECT *
FROM BORRADO_EMPLEADOS;
```

	COD_EMPLEADO	APELLIDO	NOMBRE	COD_DEPARTAMENTO
1	500	PEREZ	JUAN	10
2	600	PEREZ	JUAN	23

Ejercicio número: 2

Crear un disparador que nos guarde en una tabla denominada CONTROL, que contenga los campos USUARIO varchar2(30) y FECHA de tipo fecha, el nombre del usuario y la fecha cada vez que se modifica el precio de un producto

```
CREATE TABLE CONTROL (
USUARIO VARCHAR2 (30),
FECHA DATE
);

CREATE OR REPLACE TRIGGER TR_ACTUALIZA_PRECIO
 BEFORE UPDATE OF LIST_PRICE
 ON PRICE
 FOR EACH ROW
 BEGIN
 INSERT INTO CONTROL
 VALUES (USER, SYSDATE);
 END;
/
```

Actualizo en la tabla price el campo list_price a 20 para el product_id 100871

```
UPDATE PRICE
SET LIST_PRICE=20
WHERE PRODUCT_ID=100871;
```

Se me actualizan filas.

2 filas actualizadas

Visualizo el usuario y la fecha en la que se actualizó el precio de las dos filas

```
SELECT *
FROM CONTROL;
```

	USUARIO	FECHA
1	EMPLE	15/12/18
2	EMPLE	15/12/18

Ejercicio número: 3

Crea un disparador en el que cada vez que se modifique el pvp de un producto se inserte una fila en una tabla llamada historial_pvp con el código factura, código del producto, el precio anterior, y la fecha en la que ha sido modificado.

```
CREATE TABLE HISTORIAL_PVP
(
  ORDER_ID NUMBER (4,0),
  COD_PRODUCTO NUMBER(6,0),
  PVP NUMBER (8,2),
  FECHA DATE
);
CREATE OR REPLACE TRIGGER TR_HISTORIAL_PVP
        AFTER
        UPDATE OF ACTUAL_PRICE
        ON ITEM
        FOR EACH ROW
BEGIN
        INSERT INTO HISTORIAL_PVP VALUES (:OLD.ORDER_ID,:OLD.PRODUCT_ID,:OLD.ACTUAL_PRICE,SYSDATE);
END;
/
```

Actualizo en la tabla item el campo actual_price a 60 para el product_id 100890

```
UPDATE item
SET actual_price=60
WHERE product_id=100890;

6 filas actualizadas.
```

Se me actualizan 6 filas

Visualizo el código de factura, código de producto, precio antiguo del producto y fecha de la actualización.

```
SELECT *
FROM historial_pvp;
```

	OREDR_ID	COD_PRODUCTO	PVP	FECHA
1	610	100890	52	16/12/18
2	604	100890	52	16/12/18
3	616	100890	52	16/12/18
4	617	100890	52	16/12/18
5	609	100890	52	16/12/18
6	605	100890	52	16/12/18

🖉 **Ejercicio número: 4**

Realizar un disparador que impida a un departamento tener más de 5 empleados cuando se insertan empleados en la tabla empleados

```
CREATE OR REPLACE TRIGGER TR_EMP_DEP
 BEFORE

        INSERT ON EMPLOYEE
        FOR EACH ROW
DECLARE
        CONTADOR NUMBER(1);
BEGIN
        SELECT COUNT(*) INTO CONTADOR
        FROM EMPLOYEE
        WHERE DEPARTMENT_ID=:NEW.DEPARTMENT_ID
        GROUP BY DEPARTMENT_ID;

        IF CONTADOR>=5
          THEN
            RAISE_APPLICATION_ERROR(-20355,'MAS DE 5');
        END IF;
END;
/
```

Visualizo los empleados que tiene cada departamento ordenados por código de departamento.

```
SELECT *
FROM EMPLOYEE
ORDER BY DEPARTMENT_ID;
```

Observo que el departamento 12 tiene 4 empleados.

Inserto un nuevo empleado en el departamento 12.

```
INSERT INTO EMPLOYEE
VALUES (9000,'JULIA','GARCIA','A',670,7782,'4/2/87',385,100,12);
```

Me dice

1 FILA INSERTADA

Inserto un nuevo empleado en el departamento 12.

```
INSERT INTO EMPLOYEE
VALUES (9500,'MARIO','CANO','A',670,7782,'4/2/87',385,100,12);
```

Salta el trigger porque ya tiene más de 5 empleados el departamento 12 y no le inserta.

Informe de error:
Error SQL: ORA-20355: MAS DE 5
ORA-06512: at "EMPLE.EMP_DEP", line 11
ORA-04088: error during execution of trigger 'EMPLE.EMP_DEP'

Ejercicio número: 5

Escribir un disparador de base de datos que haga fallar cualquier operación de modificación del número de un empleado, del apellido o que suponga una subida de sueldo superior al 10%.

```
CREATE OR REPLACE TRIGGER TR_FALLO_MODIF
        BEFORE
        UPDATE OF EMPLOYEE_ID, LAST_NAME, SALARY
        ON EMPLOYEE
        FOR EACH ROW
BEGIN
        IF UPDATING('EMPLOYEE_ID') OR UPDATING ('LAST_NAME')
        OR (UPDATING ('SALARY') AND :NEW.SALARY>:OLD.SALARY*1.1)
                THEN
                        RAISE_APPLICATION_ERROR(-20001,'ERROR, MODIFICACION NO PERMITIDA');
        END IF;
 END;
 /
```

Vamos a modificar el employee_id de un empleado

```
UPDATE employee
SET employee_id=1000
WHERE employee_id=7505;
```

Salta el trigger

```
Error que empieza en la línea 1 del comando:
UPDATE employee
SET employee_id=1000
WHERE employee_id=7505
Informe de error:
Error SQL: ORA-20001: ERROR, MODIFICACION NO PERMITIDA
ORA-06512: at "EMPLE.TR_FALLO_MODIF", line 5
ORA-04088: error during execution of trigger 'EMPLE.TR_FALLO_MODIF'
```

Vamos a modificar el apellido de un empleado

```
UPDATE employee
SET last_name='CANO'
WHERE employee_id=7521;
```

Salta el trigger

```
Error que empieza en la línea 1 del comando:
UPDATE employee
SET last_name='CANO'
WHERE employee_id=7521
Informe de error:
Error SQL: ORA-20001: ERROR, MODIFICACION NO PERMITIDA
ORA-06512: at "EMPLE.TR_FALLO_MODIF", line 5
ORA-04088: error during execution of trigger 'EMPLE.TR_FALLO_MODIF'
```

Vamos a subir un 5% el salario a un empleado

```
UPDATE employee
SET salary=salary*1.05
WHERE employee_id=7934;
```

```
1 filas actualizadas.
```

Vamos a subir un 15% el salario a un empleado

```
UPDATE employee
SET salary=salary*1.15
WHERE employee_id=7789;
```

Salta el trigger

```
Error que empieza en la línea 1 del comando:
UPDATE employee
SET salary=salary*1.15
WHERE employee_id=7789
Informe de error:
Error SQL: ORA-20001: ERROR, MODIFICACION NO PERMITIDA
ORA-06512: at "EMPLE.TR_FALLO_MODIF", line 5
ORA-04088: error during execution of trigger 'EMPLE.TR_FALLO_MODIF'
```

8.- EJERCICIOS PROPUESTOS

1º.- Crear un disparador que impida borrar de la tabla **ITEM** el día 11 de cualquier mes.

2º.- Crear un disparador que impida que un departamento quede con más de 7 empleados cuando se inserte un nuevo empleado en la tabla **EMPLOYEE** o se modifique el departamento a un empleado en dicha tabla.

3º.- Crear un disparador que impida la inserción de productos (description) en la tabla productos si ya existe el producto (description).

4º.- Crear un disparador que impida que se inserten nombres con todas las letras minúsculas en la tabla clientes.

5º.- Crear un trigger para impedir que, al insertar un empleado, el empleado y su jefe puedan pertenecer a departamentos distintos.

6º.- Crear un disparador que impida la inserción de pedidos con fecha anterior a la actual.