TEMA 7 (PARTE IV). TRIGGERS EN PL/SQL.

1. INTRODUCCIÓN.

Son programas almacenados asociados a una tabla. Se disparan o ejecutan cuando se realiza una operación o una sentencia DELETE|INSERT|UPDATE sobre la tabla.

Se ejecuta el trigger automáticamente cuando alguna orden DML (INSERT, UPDATE, DELETE) afecta a la tabla. Se suelen utilizar para:

- ✓ Implementar restricciones complejas de seguridad o integridad.
- ✓ Auditar las actualizaciones, e incluso enviar alertas.
- ✓ Generar valores derivados.
- ✓ Prevenir transacciones erróneas.
- ✓ Gestión de réplicas remotas de la tabla

Para crear un trigger hay que tener privilegios de CREATE TRIGGER, así como los correspondientes privilegios sobre la/s tabla/s y otros objetos referenciados por el trigger ya que cuando un trigger se dispara, trabaja con los privilegios del propietario del trigger, no con los del usuario actual.

Existen 3 tipos de trigger:

- ✓ Asociados a tablas. Se disparan cuando se produce insert, update o delete sobre una tabla.
- ✓ Asociados a vistas. Se disparan cuando se produce insert, update o delete sobre una vista.
- ✓ Disparadores del sistema. Se disparan cuando se produce un evento del sistema, por ejemplo: create, alter, drop, connect.

2. TRIGGER ASOCIADOS A TABLAS.

El formato genérico para la creación de triggers es:

```
CREATE [OR REPLACE] TRIGGER NOMBRETRIGGER
{BEFORE|AFTER}{DELETE|INSERT|UPDATE [OF <LISTA_COLUMNAS> ]}
[OR { BEFORE|AFTER }  {DELETE|INSERT|UPDATE [OF <LISTA_COLUMNAS>] }] . . .
ON NOMBRE_TABLA
 [REFERENCING {OLD AS <NOMBREANT> | NEW AS <NOMBRENUEV> }]
 [FOR EACH {STATEMENT | ROW [WHEN (CONDICIÓN)]}]

DECLARE          -- opcional, si no hay que declarar variables no es necesario
        <declaraciones>
BEGIN
        <acciones>
EXCEPTION        -- opcional, si no hay que controlar excepciones, no se pone
        <gestión de excepciones>
END;
```

- ✓ Se puede ejecutar antes (BEFORE) o después (AFTER) de INSERT, UPDATE o DELETE que afecte a la tabla. En el caso del UPDATE se pueden especificar las columnas cuya actualización disparará el trigger.
- ✓ ON TABLA indica la tabla a la que se refiere.
- ✓ FOR EACH STATEMENT indica que el trigger se ejecutará a nivel de orden, o nivel de sentencia, es decir, se ejecutará sólo una vez, independiente del número de filas afectadas. Es la opción por defecto.
- ✓ FOR EACH ROW indica que se ejecutará una vez para cada fila afectada por la transacción.
- ✓ WHEN restringe la ejecución del trigger al cumplimiento de la condición. Sólo se puede utilizar con FOR EACH ROW. Se trata de una condición SQL, no PL/SQL. En el caso de

que queramos hacer aquí referencia a los valores new y old lo haremos sin poner los dos puntos.

- ✓ REFERENCING: cuando queremos hacer referencia a los valores anterior y posterior a una actualización, por defecto lo haremos respectivamente como :old.nobrecolumna y :new.nombrecolumna. Si queremos que en lugar de old y new aparezcan otras palabras lo indicaremos en esta cláusula.

Los valores :old.nombre_columna, y :new.nombre_columna, van a contener los datos almacenados de las columnas de las filas de la tabla sobre la que se dispara el trigger.

:old.nombre_columna, referencia a los valores antiguos (caso de borrar o de actualizar tendremos valores :old)

:new.nombre_columna, referencia a los valores nuevos (caso de modificar o insertar tendremos valores :new)

2.1. UTILIZACIÓN DE VALORES :OLD Y :NEW

SUCESO	:OLD	:NEW
INSERT	NULL	Nuevos valores
UPDATE	Valores almacenados	Nuevos valores
DELETE	Valores almacenados	NULL

Ejemplos.

Trigger definido a nivel de orden o a nivel de sentencia:

Crear un trigger que se ejecute cada vez que se inserta un registro en la tabla DEPART, el trigger insertará en la tabla TEM_DEPART una fila indicando la fecha, el usuario y que se ha insertado un registro en DEPART. La tabla TEM_DEPART tiene este formato:

```
--ANTES DE CREAR EL TRIGGER VAMOS A CREAR UNA NUEVA TABLA, TEM_DEPART.
--CREACIÓN DE LA TABLA TEM_DEPART:
DROP TABLE TEM_DEPART;
CREATE TABLE TEM_DEPART
 (INFORMACION  VARCHAR2(150));

--CREACIÓN DEL TRIGGER ALMACENADO:
CREATE OR REPLACE TRIGGER AUDIT_INSERT_DEPART AFTER INSERT ON DEPART
BEGIN
INSERT INTO TEM_DEPART VALUES ('SE HA INSERTADO UN REGISTRO, USUARIO: '||USER||' ,
FECHA: '||TO_CHAR(SYSDATE));
END;

--PRUEBA:
        INSERT INTO DEPART VALUES (12, '11111', '1111');

SE INSERTA UN REGISTRO:
EN LA TABLE TEM_DEPART SE INSERTARÁ UN REGISTRO:
SELECT * FROM TEM_DEPART;

--SE INSERTAN VARIOS REGISTROS EN DEPART:
        INSERT INTO DEPART SELECT * FROM DEPART;

EN LA TABLE TEM_DEPART SE INSERTARÁ TAMBIÉN UN REGISTRO PORQUE EL TRIGGER SE EJECUTA
A NIVEL DE ORDEN:
```

```
SELECT * FROM TEM_DEPART;
```

3. TRIGGER DEFINIDO A NIVEL DE FILA, FOR EACH ROW:

Se pide que cada vez que se inserte un registro en DEPART se inserte en la tabla TEM_DEPART una fila indicando la fecha, el usuario, que se ha insertado un registro en DEPART, el número de departamento y el nombre.
En este caso utilizaremos los valores :new, que contienen los datos del nuevo registro a insertar.

```
CREATE OR REPLACE TRIGGER AUDIT_INSERT2_DEPART AFTER INSERT ON DEPART FOR EACH ROW
BEGIN
INSERT INTO TEM_DEPART VALUES ('SE HA INSERTADO UN REGISTRO, USUARIO: '||USER||' ,
FECHA: '||TO_CHAR(SYSDATE)|| '. EL NÚMERO DEPARTAMENTO INSERTADO ES: '||
TO_CHAR(:NEW.DEPT_NO)|| ', Y EL NOMBRE ES : '||:NEW.DNOMBRE);
END;

PRUEBA:
Se inserta un registro:
INSERT INTO DEPART VALUES (22, '22222', '2222');
```

En la table TEM_DEPART se insertará un registro CON EL DETALLE DE LOS DATOS INSERTADOS y otro con la información del TRIGGER creado anteriormente ya que ha sido creado sobre la misma la tabla y sobre la misma operación:

```
SELECT * FROM TEM_DEPART;
Se insertan varios registros en DEPART:
INSERT INTO DEPART SELECT * FROM DEPART;
```

En la table TEM_DEPART se insertarán tantos registros como filas insertadas en DEPART, si se insertan 3 departamentos se crearán 3 filas en TEMP_DEPAR. Esto es así porque el trigger está definido a nivel de fila

```
SELECT * FROM TEM_DEPART;
```

Crear un trigger que se dispare cuando se inserte en la tabla emple.

- ✓ Si la comisión es nula asignarla el valor 100.
- ✓ Si el departamento no existe en la tabla DEPART, asignar el departamento 10.

```
CREATE OR REPLACE TRIGGER INSER_EMPLE BEFORE INSERT ON EMPLE FOR EACH ROW
DECLARE
 D NUMBER(2);
BEGIN
  IF :NEW.COMISION IS NULL THEN
        :NEW.COMISION := 100;
  END IF;
  SELECT DEPT_NO INTO D FROM DEPART WHERE DEPT_NO=:NEW.DEPT_NO;
  DBMS_OUTPUT.PUT_LINE('REG OK.');
EXCEPTION
 WHEN NO_DATA_FOUND THEN
   :NEW.DEPT_NO:=10;
   DBMS_OUTPUT.PUT_LINE('DEP ASIGNADO 10.');
 WHEN TOO_MANY_ROWS THEN
   NULL;
END;

INSERT INTO EMPLE  VALUES(1,'11111','OFI', 1, SYSDATE, 111,NULL,33);
SELECT * FROM EMPLE;
```

Cuando creamos un trigger sobre una modificación, a diferencia, de cuando lo creamos sobre una inserción o un borrado, podemos realizar el trigger indicando del campo sobre el que se activará. Es

decir, sólo se activará el disparador cuando se modifique el campo para el que hemos creado el disparador.

Vamos a ver un ejemplo:

Crear un trigger que se ejecute cuando se actualice el OFICIO de un empleado. Si el oficio nuevo es VENDEDOR, asignarle una comisión de 200.

```
CREATE OR REPLACE TRIGGER OFIVENDEDOR BEFORE UPDATE OF OFICIO ON EMPLE FOR EACH ROW
 WHEN (UPPER(NEW.OFICIO) = 'VENDEDOR')
BEGIN
  :NEW.COMISION:=200;
END;

--comprobacion:
SELECT * FROM EMPLE;
UPDATE EMPLE SET OFICIO = 'VENDEDOR' WHERE EMP_NO = 3;
SELECT * FROM EMPLE;
UPDATE EMPLE SET OFICIO = 'VENDEDOR' WHERE EMP_NO = 1133;
```

Podemos observar como el trigger ha sido creado para cuando se modifica el campo oficio, en el caso de que se modifique otro campo, el disparador no se activará.

4. ORDEN DE EJECUCIÓN DE LOS DISPARADORES.

Cuando una tabla lleva asociados varios disparadores, el orden de ejecución es el siguiente:

- ✓ Antes de comenzar a ejecutar la orden
  BEFORE ….. FOR EACH STATEMENT
- ✓ Para cada fila afectada por la orden

1. Se ejecuta **BEFORE ….. FOR EACH ROWS**
2. Se ejecuta **INSERT, UPDATE o DELETE**. Se bloquea la fila hasta confirmar la transacción.
3. Se ejecuta **AFTER ….. FOR EACH ROWS**

- ✓ Después de actualizar se ejecutan los disparadores
  AFTER ….. FOR EACH STATEMENT

Un trigger puede estar activado o desactivado. Cuando se crea está activado pero podemos variar esta situación:

```
ALTER TRIGGER nombretrigger {ENABLE | DISABLE | COMPILE }
```

Para eliminar un trigger:

```
DROP TRIGGER NOMBRETRIGGER;
```

Ejemplos:

Crear un trigger que se dispare cuando insertemos en la tabla emple, de tal forma que simplemente tiene que visualizar un mensaje.

```
CREATE OR REPLACE TRIGGER INSEREMPLE AFTER INSERT ON EMPLE
BEGIN
  DBMS_OUTPUT.PUT_LINE('REG EMPLE INSERTADO');
END;

INSERT INTO EMPLE ( EMP_NO, APELLIDO, OFICIO, DIR, FECHA_ALT, SALARIO, COMISION, DEPT_NO )
  VALUES  ( 1122, 'RAMOS', 'VENDEDOR', 1, SYSDATE, 1000, 1000, 10);

SELECT * FROM EMPLE;
INSERT  INTO EMPLE  SELECT * FROM EMPLE;
SELECT * FROM EMPLE;
```

Modificar el trigger anterior para visualizar los datos insertados. Utilizamos :old y :new y for each row

Columnas de emple: EMP_NO, APELLIDO, OFICIO, DIR, FECHA_ALT, SALARIO, COMISION, DEPT_NO

```
CREATE OR REPLACE TRIGGER INSEREMPLE  AFTER INSERT ON EMPLE  FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('REG EMPLE INSERTADO');
  DBMS_OUTPUT.PUT_LINE('NUMERO EMPLE:'||:NEW.EMP_NO);
 DBMS_OUTPUT.PUT_LINE('APELLIDO: '||:NEW.APELLIDO);
 DBMS_OUTPUT.PUT_LINE('OFICIO: '||:NEW.OFICIO);
 DBMS_OUTPUT.PUT_LINE('FECHA_ALTA: '||:NEW.FECHA_ALT);
 DBMS_OUTPUT.PUT_LINE('SALARIO :'||:NEW.SALARIO);
 DBMS_OUTPUT.PUT_LINE('NUM DEPART :'||:NEW.DEPT_NO);
END;

INSERT INTO EMPLE  ( EMP_NO,  APELLIDO,  OFICIO,  DIR,  FECHA_ALT,  SALARIO,
   COMISION,  DEPT_NO  )
 VALUES  ( 1133, 'RAMOS', 'VENDEDOR', 1, SYSDATE, 1000, 1000, 10);


INSERT  INTO EMPLE  SELECT * FROM EMPLE;
SELECT * FROM EMPLE;
```

Crear un trigger que guarde en la tabla TEMP las actualizaciones de más de 100 € en el salario de la tabla EMPLE. Guardaremos el apellido antiguo, el sueldo antiguo y el nuevo sueldo.

```
DROP TABLE TEMP;
CREATE TABLE TEMP
 (EMPLE  VARCHAR2(15),
  SUCESO VARCHAR2(20),
  SUELDO_ANTE NUMBER(9,2),
  SUELDO_NUE NUMBER(9,2));

CREATE OR REPLACE TRIGGER AUDIT_SUBI_SUELDO AFTER UPDATE OF SALARIO ON EMPLE
 FOR EACH ROW
 WHEN ((NEW.SALARIO - OLD.SALARIO) > 100)
BEGIN
INSERT INTO TEMP VALUES (:OLD.APELLIDO,'SUBIDA SUELDO', :OLD.SALARIO, :NEW.SALARIO);
END;
```

Prueba, subimos el salario 101€ a los empleados del departamento 10,

```
UPDATE EMPLE SET SALARIO=SALARIO+101 WHERE DEPT_NO=10;
```

Visualizar la tabla TEMP y ver su contenido:

```
SELECT * FROM TEMP;
```

5. INSERTING - UPDATING - DELETING

Cuando un mismo trigger pueda ser disparado por distintas operaciones o eventos de disparo utilizaremos **or** para separar los eventos y para tratar cada situación podremos utilizar inserting, deleting y updating dependiendo de la operación, ejemplo:

```
CREATE OR REPLACE TRIGGER Cambios_DEPART   AFTER INSERT OR DELETE OR UPDATE ON EMPLE
. . . . . .
DECLARE
 . . . . . .
BEGIN
IF INSERTING THEN
 . . . . . .
ELSIF UPDATING THEN
    . . . . .. . .
   ELSIF UPDATING('SALARIO') THEN
```

```
      . . . . . .
      ELSIF DELETING THEN
        . . . . . . .
END IF;
. . . . . . .
END;
```

EJEMPLOS:

Primero vamos a creamos la tabla AUDITA_DEPART, con los datos:

```
CREATE TABLE AUDITA_DEPART
 (DEPT_NO NUMBER(2),

  OPERACION VARCHAR2(1),
  NOMBRE VARCHAR2(14),
  USUARIO VARCHAR2(14),
  FECHA    DATE  );
```

Crear un disparador que se ejecute cada vez que se haga un insert, update o delete sobre la tabla depart. Guardaremos en audita_depart el número de departamento, la operación realizada (i,u,d), el nombre , el usuario que realiza la operación y la fecha.

```
CREATE OR REPLACE TRIGGER AUDITDEPART AFTER INSERT OR UPDATE OR DELETE ON DEPART FOR
EACH ROW
BEGIN
 IF INSERTING THEN
   INSERT INTO AUDITA_DEPART VALUES (:NEW.DEPT_NO, 'I', :NEW.DNOMBRE, USER, SYSDATE);
 ELSIF DELETING THEN
   INSERT INTO AUDITA_DEPART VALUES (:OLD.DEPT_NO, 'D', :OLD.DNOMBRE, USER, SYSDATE);
 ELSE
  -- UPDATING
   INSERT INTO AUDITA_DEPART VALUES (:OLD.DEPT_NO, 'U', :OLD.DNOMBRE, USER, SYSDATE);
 END IF;
END;

INSERT INTO DEPART (DEPT_NO,   DNOMBRE,   LOC,   NUM_EMPLE)
 VALUES ( 60,'DEP60','TALAVERA',0  );
SELECT * FROM AUDITA_DEPART;

DELETE DEPART WHERE DEPT_NO = 45;
SELECT * FROM AUDITA_DEPART;

UPDATE DEPART SET DNOMBRE = 'NUEVO' WHERE DEPT_NO = 42;
SELECT * FROM AUDITA_DEPART;
```

Crear un disparador que se ejecute cuando se actualice en la tabla depart. Se deben visualizar por pantalla sólo las columnas que se actualizan

```
CREATE OR REPLACE TRIGGER VERCAMBIOSDEPAR AFTER UPDATE ON DEPART FOR EACH ROW
BEGIN
 IF UPDATING('DEPT_NO') THEN
   DBMS_OUTPUT.PUT_LINE ('NÚMERO DEPAR CAMBIA:'||
   ' NUEVO =' || :NEW.DEPT_NO ||', ANTIGUO = '|| :OLD.DEPT_NO);
 END IF;
 IF UPDATING('DNOMBRE') THEN
   DBMS_OUTPUT.PUT_LINE ('NOMBRE DEP CAMBIA:'||
   ' NUEVO =' || :NEW.DNOMBRE ||', ANTIGUO = '|| :OLD.DNOMBRE);
 END IF;
 IF UPDATING('LOC') THEN
   DBMS_OUTPUT.PUT_LINE ('LOCALIDAD DEP CAMBIA:'||
   ' NUEVO =' || :NEW.LOC ||', ANTIGUO = '|| :OLD.LOC);
```

```
 END IF;
END;

UPDATE DEPART SET DNOMBRE='CAMBIO' WHERE DEPT_NO=40;
UPDATE DEPART SET DNOMBRE='NUEVONOM', LOC='NUEVA' WHERE DEPT_NO=40;

UPDATE DEPART SET DNOMBRE='NUEV3', LOC='NUEVA3', DEPT_NO = 81 WHERE DEPT_NO=40;
```

Esta es otra solución, que diferencia habría entre ambas soluciones:

```
CREATE OR REPLACE TRIGGER VERCAMBIOSDEPAR AFTER UPDATE ON DEPART FOR EACH ROW
BEGIN
 IF :NEW.DEPT_NO != :OLD.DEPT_NO THEN
   DBMS_OUTPUT.PUT_LINE ('NÚMERO DEPAR CAMBIA:'||
   ' NUEVO =' || :NEW.DEPT_NO ||', ANTIGUO = '|| :OLD.DEPT_NO);
 END IF;
 IF :NEW.DNOMBRE != :OLD.DNOMBRE THEN
   DBMS_OUTPUT.PUT_LINE ('NOMBRE DEP CAMBIA:'||
   ' NUEVO =' || :NEW.DNOMBRE ||', ANTIGUO = '|| :OLD.DNOMBRE);
 END IF;
 IF :NEW.LOC != :OLD.LOC THEN
   DBMS_OUTPUT.PUT_LINE ('LOCALIDAD DEP CAMBIA:'||
   ' NUEVO =' || :NEW.LOC ||', ANTIGUO = '|| :OLD.LOC);
 END IF;
END;
```