9 .- CREACIÓN DE TRIGGERS EN UNA BASE DE DATOS

9 .1.- INTRODUCCIÓN

Un TRIGGER es un bloque PL/SQL que se ejecuta implícitamente cuando ocurre un evento.

Los TRIGGER pueden ser de dos tipos:

1. **De la base de datos**: los cuales se ejecutan de forma implícita cuando se lanza una sentencia DML contra una tabla.

2. **De aplicación:** se ejecutan de forma implícita cuando ocurre un evento particular en la aplicación.

Guía para diseñar triggers:

- ✓ Un TRIGGER garantiza que cuando se realiza una operación específica, se realicen también las acciones relacionadas.

- ✓ Utilizar TRIGGERS para operaciones globales.

- ✓ No definir TRIGGER para implementar reglas de integridad, que puedan hacerse mediante restricciones declarativas.

- ✓ No abusar de los TRIGGERS, pues pueden dar lugar a interdependencias complejas.

9 .2.- COMPONENTES DE UN TRIGGER

Para crear un TRIGGER hay que tener en cuenta:

1.- **Momento** en que se ejecuta el TRIGGER con relación al evento, admitiendo dos valores:

- **BEFORE**: el TRIGGER se ejecuta antes del evento (sentencia DML).
- **AFTER**: el TRIGGER se ejecuta después del evento.

2.- **Evento o disparador**: pueden ser tres:

- INSERT
- DELETE
- UPDATE

3.- **Tipos de TRIGGERS**: Los TRIGGERS pueden ser de dos **tipos** dependiendo de la ejecución del cuerpo del mismo:

- **De sentencia**: el cuerpo del TRIGGER se ejecuta una vez para el evento.
- **De fila**: el cuerpo del TRIGGER se ejecuta una vez para cada registro afectado por el evento.

4.- **Cuerpo del TRIGGER**: o la acción que debe realizar el TRIGGER, que es un bloque PL anónimo.

9.3.- SINTAXIS PARA LA CREACIÓN DE UN TRIGGER A NIVEL SENTENCIA.

Sintaxis.

```
CREATE [OR REPLACE] TRIGGER nb_trigger
{BEFORE | AFTER } evento1 [ OR  evento2 OR evento3 ...]
ON table_name
Bloque PL/SQL;
```

Donde evento es una sentencia INSERT, DELETE o UPDATE

Ejemplo:

Diseñar un TRIGGER llamado seguridad que restrinja la inserción de elementos los días 30 y 31 de todos los meses. Será a nivel sentencia y se ejecutará antes de producirse el evento.

```
CREATE OR REPLACE TRIGGER SEGURIDAD
BEFORE INSERT ON EMP
BEGIN
        IF(TO_CHAR(SYSDATE, 'DD') IN (30,31)) THEN
                RAISE_APPLICATION_ERROR (-20500,'NO SE PUEDE INSERTAR NI
                EL DIA 30 NI EL 31');
        END IF;
END;
/
```

Este TRIGGER se ejecutará cuando intentemos realizar una instrucción INSERT en la tabla EMP un 30 o un 31 de cualquier mes de cualquier año.

Supongamos que hoy es 30 de enero de 2002 si intentemos insertar un elemento sucede lo siguiente:

```
INSERT INTO EMP
VALUES (2000,'MARY','SALESMAN', 7902,'SYSDATE',100,200,10);
```

```
INSERT INTO EMP VALUES
       *
ERROR en línea 1:
ORA-20500: NO SE PUEDE INSERTAR NI EL DIA 30 NI EL 31
ORA-06512: en "SCOTT.SEGURIDAD",  línea 3
ORA-04088: error durante la ejecución del disparador 'SCOTT.SEGURIDAD'
```

La cláusula RAISE_APPLICATION_ERROR imprime el mensaje de usuario y provoca que falle. Su sintaxis es:

```
RAISE_APPLICATION_ERROR (- número, 'mensaje');
```

Cuando un trigger falla, Oracle hace ROLLBACK sobre todas las sentencias del TRIGGER.

Se pueden combinar varios eventos de TRIGGER en uno solo, aprovechando los siguientes predicados condicionales:

- INSERTING
- DELETING
- UPDATING

Ejemplo**:**

Modificar el ejemplo anterior, para que esto ocurra al realizar una inserción, modificación o eliminación de datos los días 30 y 31 de cada mes.

```
CREATE OR REPLACE TRIGGER SEGURIDAD
BEFORE INSERT OR UPDATE OR DELETE
ON EMP
BEGIN
IF (TO_CHAR (SYSDATE,'DD') in (30, 31)) THEN
 IF DELETING THEN
        RAISE_APPLICATION_ERROR (-20501, 'NO SE PUEDE BORRAR LOS DIAS
        30 Y 31');
 ELSIF INSERTING THEN
        RAISE_APPLICATION_ERROR (-20502, 'NO SE PUEDE INSERTAR LOS DIAS
        30 Y 31');
   ELSIF UPDATING ('SAL') THEN
        RAISE_APPLICATION_ERROR (-20503, 'NO SE PUEDE MODIFICAR EL
        SALARIO  LOS DIAS 30 Y 31');
     ELSE
                RAISE_APPLICATION_ERROR (-20504, 'NO SE PUEDE MODIFICAR
                LOS DIAS 30 Y 31');
 END IF;
END IF;
END;
/
```

Nota: La cláusula «*updating ('SAL')*» indica que la columna "*sal*" no puede ser modificada, si en vez de poner «*elsif updating ('SAL') then*» ponemos directamente «*else*» no podremos realizar ninguna modificación en ninguna columna, sino, sólo nos prohibiría modificar "*sal*" o para prohibir todas las modificaciones tendríamos que poner «*updating*» para todas las columnas.

9.4.- CREACIÓN DE TRIGGER A NIVEL REGISTRO.

Sintaxis:

```
CREATE [OR REPLACE] TRIGGER nb_triger
{BEFORE|AFTER} evento 1 [OR evento2 OR evento3 ...]
ON table_name
[REFERENCES {OLD AS old | NEW AS new}]
FOR EACH ROW
[WHEN condición]
Bloque PL/SQL;
```

Donde:

EVENTO: si el evento es UPDATE tiene la siguiente sintaxis: UPDATE [OF nb_col].
FOR EACH ROW: indica que el TRIGGER es a nivel registro.
WHEN condición: evalúa la condición del TRIGGER.
REFERENCES...: permite cambiar los prefijos OLD y NEW por otros.

Ejemplo**:**

Crear un disparador llamado auditar_subida_salario que se ejecute después de cada modificación en la columna SAL en la tabla EMP. Se guardarán los valores insertados en otra tabla llamada AUDITAREMPLE.

En primer lugar creamos la tabla:

```
CREATE TABLE AUDITAREMPLE
(COL1 VARCHAR2(200));
```

En segundo lugar creámos el TRIGGER:

```
CREATE OR REPLACE TRIGGER AUDITAR_SUBIDA_SALARIO
AFTER UPDATE OF SAL
ON EMP
FOR EACH ROW
BEGIN
INSERT INTO AUDITAREMPLE VALUES ('SUBIDA DE SALARIO EMPLEADO
'||:OLD.EMPNO||' QUE ERA DE '||:OLD.SAL||' ES DE '||:NEW.SAL);
END;
/
```

En tercer lugar se intentan modificar los salarios:

```
UPDATE EMP SET SAL=900
WHERE EMPNO=7788;

1 fila actualizada.
```

Finalmente observamos el contenido de la tabla AUDITAREMPLE:

```
SELECT * FROM AUDITAREMPLE;

COL1
--------------------------------------------------
SUBIDA SALARIO EMPLEADO 7788 1140 900
```

Para restringir la condición del TRIGGER a solo aquellos registros que satisfagan cierta condición lo proporciona la cláusula WHEN. En la cláusula WHEN el calificador NEW no necesita ir precedido de dos puntos.

9.5.- DIFERENCIAS ENTRE TRIGGER Y PROCEDURE

TRIGGER	PROCEDIMIENTO
CREATE TRIGGER	CREATE PROCEDURE
Invocación implícita.	Invocación explicita.
No admite COMMIT, SAVEPOINT y ROLLBACK	Si admite COMMIT, SAVEPOINT y ROLLBACK .

9.6.- GESTIÓN Y BORRADO DE TRIGGERS

- Para activar o desactivar un TRIGGER se hace con la orden:

```
ALTER TRIGGER nb_trigger {ENABLE|DISABLE};
```

- Para activar o desactivar los TRIGGERS asociados a una tabla:

```
ALTER TRIGGER nb_trigger {ENABLE|DISABLE} ALL TRIGGERS;
```

- Para volver a compilar un TRIGGER:

```
ALTER TRIGGER nb_trigger COMPILE;
```

- Para borrar un TRIGGER:

```
DROP TRIGGER nb_trigger;
```

- En el diccionario de datos se ven en:
  - USER _ OBJECTS
  - USER_TRIGGERS;
  - DBA_TRIGGERS;

9.7.- PRUEBAS SOBRE TRIGGERS

1. Probar cada una de las operaciones de datos sobre los TRIGGERS, así como las operaciones sobre otros datos.
2. Probar cada posibilidad de la cláusula WHEN.
3. Provocar el disparo.
4. Probar el efecto del TRIGGER sobre otros TRIGGERS que provocarán un evento hacia la misma tabla.
5. Probar el efecto de otros TRIGGERS sobre el actual.

Ejemplo 1:

Crear un TRIGGER, que se dispare antes de borrar un empleado, guardando sus datos en la tabla AUDITAREMPLE.

```
CREATE OR REPLACE TRIGGER AUDITAR_BORRADO_EMPLE
BEFORE DELETE ON EMP
FOR EACH ROW
BEGIN
        INSERT INTO AUDITAREMPLE VALUES ('BORRADO EMPLEADO ' || '*'
        ||:OLD.EMPNO || '*' ||:OLD.ENAME || '*' ||:OLD.DEPTNO);
END;
/

DELETE EMP;
13 FILAS BORRADAS.

SELECT * FROM AUDITAREMPLE;

COL1
-----------------------------------------------------------------
BORRADO EMPLEADO *7499*SMITH*30
...
```

Ejemplo 2:

Crear un TRIGGER en la tabla EMP para calcular la comisión de un vendedor (SALESMAN) cuando se añade un registro a la tabla EMP o cuando se modifica el salario de un vendedor, teniendo en cuenta que al insertar la comisión valdrá 0 y cuando se modifica si esta es nula vale cero y si no es nula es la antigua comisión por el nuevo salario entre el antiguo.

```
CREATE OR REPLACE TRIGGER CALCULO_COMM
 BEFORE INSERT OR UPDATE OF SAL ON EMP
 FOR EACH ROW
 WHEN (NEW.JOB='SALESMAN')
 BEGIN
  IF INSERTING THEN :NEW.COMM:=0;
   ELSE
    IF :OLD.COMM IS NULL THEN
      :NEW.COMM:=0;
```

```
   ELSE
     :NEW.COMM:=:OLD.COMM*(:NEW.SAL/:OLD.SAL);
   END IF;
  END IF;
 END;
/

INSERT INTO EMP
VALUES (8888, 'ANTONIO', 'SALESMAN',7900,'',600,'',20);

1 fila creada.

UPDATE EMP
SET SAL = 3000
WHERE EMPNO=7521;

1 fila actualizada.

ROLLBACK;

Rollback terminado.
```

9.8.- REGLAS QUE GOBIERNAN LOS TRIGGERS.

1.- **No leer los datos de una tabla mutante.** Una tabla mutante es aquella que está siendo actualizada mediante sentencias DML, funciones o los efectos de una acción de integridad referencial (FOREIGN KEY)

Ejemplo:

Crear un TRIGGER que garantice que siempre que se añaden un empleado a una tabla EMP o se cambie un salario, el salario caiga dentro del establecido por el máximo y por el mínimo. La condición del TRIGGER será que el nuevo trabajo sea distinto de presidente.

```
CREATE OR REPLACE TRIGGER CHECK_SAL
BEFORE INSERT OR UPDATE OF SAL, JOB
ON EMP
FOR EACH ROW
WHEN (NEW.JOB<>'PRESIDENT')
DECLARE
        V_MINSAL EMP.SAL%TYPE;
        V_MAXSAL EMP.SAL%TYPE;
BEGIN
        SELECT MIN(SAL), MAX(SAL) INTO
        V_MINSAL,V_MAXSAL
        FROM EMP WHERE JOB= :NEW.JOB;
        IF (:NEW.SAL<V_MINSAL) OR (:NEW.SAL>V_MAXSAL) THEN
                RAISE_APPLICATION_ERROR(-20550,'FUERA DE RANGO');
        END IF;
END;
/
```

```
UPDATE EMP SET SAL= 9000
WHERE EMPNO=7782;
UPDATE EMP SET SAL= 9000
       *
ERROR en línea 1:
ORA-04091: la tabla SCOTT.EMP está mutando, puede que el disparador/la función no
puedan verla
ORA-06512: en "SCOTT.CHECK_SAL",  línea 5
ORA-04088: error durante la ejecución del disparador 'SCOTT.CHECK_SAL'
```

2**.- Combinando datos de una tabla restrictiva**. Una tabla es restrictiva cuando un TRIGGER tiene la necesidad de leer directamente una sentencia SQL o indirectamente mediante una restricción de integridad referencial.

Ejemplo:

Crear un TRIGGER que intente actualizar en cascada la clave foránea para los registros hijos de la tabla EMP, a partir del cambio de la clave primaria de la tabla DEPT.

```
CREATE OR REPLACE TRIGGER CASCADA_CAMBIOS
AFTER UPDATE OF DEPTNO ON DEPT
FOR EACH ROW
BEGIN
UPDATE EMP SET EMP.DEPTNO = :NEW.DEPTNO
WHERE EMP.DEPTNO = :OLD.DEPTNO;
END;
/


UPDATE DEPT SET DEPTNO=1
WHERE DEPTNO = 30;
UPDATE DEPT SET DEPTNO=1
       *
ERROR en línea 1:
ORA-04091: la tabla SCOTT.DEPT está mutando, puede que el disparador/la función no
puedan verla
ORA-06512: en "SCOTT.CASCADA_CAMBIOS",  línea 2
ORA-04088: error durante la ejecución del disparador 'SCOTT.CASCADA_CAMBIOS'
```

9.9.- IMPLEMENTACIÓN DE TRIGGERS

Los TRIGGERS se crean, entre otras razones, por lo siguiente:

- **Seguridad**: permiten el acceso a las tablas según el valor de los datos.
- **Auditorias**: guardan los valores en otras tablas.
- **Integridad de datos**: implementan reglas complejas de integridad.

- **Integridad referencial**: implementan funcionalidad no estándar.
- **Replicación de datos:** copian tablas de manera sincronizada a través de replicas.
- **Datos derivados**: calculan automáticamente valores que se derivan de otros datos.
- **Control de eventos:** controla los eventos de forma transparente.