



TRIGGERS

Qué son los triggers?

- **Los triggers son también procedimientos almacenados en la base de datos que tienen la característica de ser implícitamente ejecutados o “disparados” cuando sucede algún evento en la Base de Datos.**
- **Existen triggers que se disparan asociados a las sentencias DML de las tablas, a algún suceso en la Base de Datos o en el esquema.**

TRIGGERS DE SENTENCIAS DML

- Los triggers DML están asociados a una tabla específica y son implícitamente ejecutados (“disparados”) cuando una tabla es modificada.
- Los triggers son utilizados para personalizar la reacción de un servidor de Base de Datos de Oracle ante eventos de aplicaciones

TRIGGERS

- **La definición de triggers DML se refiere al bloque o procedimiento que es disparado por las sentencias DML, incluyendo INSERT, UPDATE y/o DELETE. Un trigger está asociado con una y solo una tabla.**
- **Un trigger puede configurarse para que se dispare antes o después de la sentencia del trigger para proporcionar lógica específica de la aplicación**

SINTAXIS DEL TRIGGER:

```
CREATE OR REPLACE TRIGGER
<nombre_trigger>
BEFORE |AFTER operación [OR
operación... ] [OF campo, ... campo]
ON <nombre_tabla>
REFERENCING new| old AS <alias>
[FOR EACH ROW] [WHEN (condición) ]

DECLARE
BEGIN

END;
```

Donde:

OPERACIÓN = **INSERT | DELETE | UPDATE**

Campo = **nombre del campo de la tabla,**

indicada por nombre_tabla

Condición = **Condición por la que se**

dispara el trigger

EJEMPLO:

```
CREATE OR REPLACE TRIGGER
actualizar_stock AFTER INSERT ON
b_detalle_ventas
FOR EACH ROW WHEN (new.cantidad>0)
DECLARE
BEGIN
    update b_articulos set
    stock_actual=stock_actual -
    :new.cantidad
    where id=:new.id_articulo;
END;
/
```

Para identificar qué operación disparó el trigger se usa:

IF INSERTING THEN

IF DELETING THEN

IF UPDATING THEN

o bien

IF UPDATING (total_vendedor) THEN

Ejemplo: Log de las actualizaciones sobre la tabla vendedor

CREACION DE 2 TABLAS: VENDEDOR Y CONTROL_ABM

(Esta última para mantener el log de operaciones DML sobre la primera)

```
SQL> CREATE TABLE VENDEDOR  
  2  (ID NUMBER,  
  3   NOMBRE VARCHAR2(60));
```

```
SQL> CREATE TABLE CONTROL_ABM  
  2  (FECHA DATE,  
  3   TIPO  VARCHAR2(1),  
  4   USUARIO VARCHAR2(60));
```

Creación del trigger sobre la tabla ‘VENDEDOR’

```
CREATE OR REPLACE TRIGGER T_CONTROL_DML
    AFTER INSERT OR UPDATE OR DELETE ON
        vendedor
DECLARE
    tipo CHAR(1);
BEGIN
    IF INSERTING THEN
        tipo:= 'I' ;
    ELSIF UPDATING THEN
        tipo:= 'U';
    ELSIF DELETING THEN
        tipo:= 'D';
    END IF;
    INSERT INTO control_abm      VALUES (
        SYSDATE, Tipo, USER);
END T_CONTROL_DML;
/
```

TENGA EN CUENTA:

- El trigger garantiza que se ejecuta una acción (validación, auditoría,) cuando se produce una operación sobre la tabla.
- El trigger es válido para cualquier aplicación que acceda a la Base de Datos.
- No se deben definir triggers para operaciones que ya son controladas por reglas de integridad referencial o checks.
- No se deben crear trigger recursivos: Por ejemplo actualizar la tabla B_PERSONAS en un trigger que se dispara cuando se actualiza la propia tabla B_PERSONAS.
- No codifique triggers muy largos.

LA CLÁUSULA ‘FOR EACH ROW’

- Cuando se omite la cláusula ‘FOR EACH ROW’, el trigger se dispara como consecuencia de la aplicación de la sentencia: 1 única vez no importando cuántas filas se afecta.
- Cuando es incluida dicha sentencia, todas las filas afectadas por la sentencia DML disparan el trigger.

CUADRO DE EVENTOS

| TEMPORIZACION | ANTES | | | DESPUÉS | | |
|---------------|-------|---|---|---------|---|---|
| NIVEL | | | | | | |
| FILA | I | D | U | I | D | U |
| INSTRUCCIÓN | I | D | U | I | D | U |

ORDEN DE EJECUCIÓN DE DISPARADORES

1. Ejecución de los disparadores de tipo BEFORE a nivel de instrucción (se disparan 1 sola vez independientemente del n° de filas)
2. Para cada fila:
 - Ejecución de los disparadores BEFORE a nivel de fila
 - **Ejecución de la propia instrucción**
 - Ejecución de los disparadores AFTER a nivel de fila
3. Ejecución de los disparadores AFTER a nivel de instrucción.

ORDEN DE EJECUCIÓN DE DISPARADORES (Ejemplo)

```
update b_personas set es_cliente  
= 'S' where id_localidad=12;
```

→ Trigger BEFORE sentencia

| ID | NOMBRE | APELLIDO |
|----|---------|----------|
| 7 | Roberta | Pedrozo |
| 17 | Juan | Avalos |
| 18 | Akira | Nozaki |
| 19 | Víctor | Paez |

→ Trigger BEFORE fila

→ Trigger AFTER fila

...

→ Trigger BEFORE fila

→ Trigger AFTER fila

→ Trigger AFTER sentencia

Los identificadores :old y :new

- Los prefijos :old y :new permiten referirnos a las columnas de la tabla sobre la que se dispara el trigger. Ej: :new.ID
- :new no podría estar disponible en un disparador de una operación DELETE, así como :old no estará disponible en una operación INSERT.

Tablas Mutantes

Una tabla que está “**mutando**” es una tabla que **está siendo modificada por un INSERT, un DELETE o un UPDATE o una tabla que podría modificarse por los efectos de un ON DELETE CASCADE (integridad referencial)**.

Una tabla está “restringiendo” (tabla de restricción o tabla padre) si una sentencia activadora podría necesitar leerla directamente, por una sentencia SQL, o indirectamente, por una restricción de integridad referencial.

Tablas Mutantes

Hay dos restricciones importantes con respecto a las tablas que mutan o que restringen:

1. Las instrucciones de un trigger de fila no pueden leer ni modificar una tabla que está mutando.
2. Las instrucciones de un trigger de fila no pueden cambiar ni la clave primaria, ni claves foráneas, ni atributos únicos de tablas que estén restringiendo.

Esta restricción tiene una excepción: un **before row** trigger disparado por un INSERT de una sola fila de una tabla con una clave foránea, puede modificar cualquier columna de la tabla primaria siempre que no se viole ninguna de las restricciones de integridad referencial.

Tablas Mutantes

- Los ERRORES por Tablas Mutantes se detectan y se generan en Tiempo de Ejecución y no de Compilación (ORA-4091)

Ejemplo de tabla mutante

“Por una política de la empresa, NINGÚN jefe puede tener más de 3 empleados que dependan de él”.

Codifique el o los triggers necesarios para que este control se realice al INSERTAR un registro y/o MODIFICAR la cédula del jefe en la tabla B_EMPLEADOS. (Este control no se aplica cuando el empleado no tiene JEFE.)

Cómo podríamos codificar este control:

```
CREATE OR REPLACE TRIGGER T_BEF_EMP
BEFORE INSERT OR UPDATE ON B_EMPLEADOS
FOR EACH ROW WHEN (NEW.CEDULA_JEFE IS NOT NULL)
DECLARE
    V_CANTIDAD NUMBER;
BEGIN
    SELECT COUNT(CEDULA) INTO V_CANTIDAD
    FROM B_EMPLEADOS
    WHERE CEDULA_JEFE = :NEW.CEDULA_JEFE;
    DBMS_OUTPUT.PUT_LINE(V_CANTIDAD);
    IF NVL(V_CANTIDAD, 0) > 3 THEN
        RAISE_APPLICATION_ERROR (-20001, 'Jefe no
puede tener más de 3 dependientes');
    END IF;
END;
/
```

Probando el trigger

```
INSERT INTO B_EMPLEADOS  
VALUES  
(123456, 'EVER JOSE', 'RIVEROS VELEZ', SYSDATE,  
TO_DATE('10/02/1970','DD/MM/YYYY'), 952160, '1', 'A', 'B  
N', NULL);
```

(suponiendo que la cédula 952160 ya tenga 3 empleados)

ERROR en línea 1:

ORA-20001: Jefe no puede tener más de 3 dependientes

ORA-06512: en "BASEDATOS2.T_BEF_EMP", línea 9

ORA-04088: error durante la ejecución del disparador

'BASEDATOS2.T_BEF_EMP'

FUNCIONA!!!

Probando el trigger

```
UPDATE B_EMPLEADOS SET CEDULA_JEFE = 952160  
WHERE CEDULA = 4433245;
```

(suponiendo que la cédula 952160 ya tenga 3 empleados)

ERROR en línea 1:

ORA-04091: la tabla BASEDATOS2.B_EMPLEADOS está mutando, puede que el disparador/la función no puedan verla

ORA-06512: en "BASEDATOS2.T_BEF_EMP", línea 4

ORA-04088: error durante la ejecución del disparador 'BASEDATOS2.T_BEF_EMP'

- Las tablas mutantes surgen básicamente en los disparadores a nivel de Fila. Como en estos no puede accederse a las tablas mutantes, la solución es crear dos disparadores:
 - **A Nivel de Fila:** En este guardamos los valores importantes en la operación, pero no accedemos a tablas mutantes. Estos valores pueden guardarse en:
 - Tablas de la BD especialmente creadas para esta operación.
 - Variables o tablas PL/SQL de un paquete: Como cada sesión obtiene su propia instancia de estas variables no tendremos que preocuparnos de si hay actualizaciones simultáneas en distintas sesiones.
 - **A Nivel de sentencia BEFORE o AFTER:** Utiliza los valores guardados en el disparador a Nivel de Fila para acceder a las tablas que ya no son mutantes.

Ejemplo: Solución con variables de paquete

➤ **Paso 1: Declarar una variable de paquete:**

```
CREATE OR REPLACE PACKAGE PACK_EMP
IS
    TYPE T_PER IS TABLE OF NUMBER INDEX BY
    BINARY_INTEGER;
    V_PER T_PER;
END;
/
```

Ejemplo: Solución con variables de paquete

- **Paso 2: Codificar un trigger a nivel de SENTENCIA y guardar, por cada jefe, la cantidad de empleados que dependen de él**

```
CREATE OR REPLACE TRIGGER T_BIU_EMP_INST
BEFORE UPDATE OR INSERT ON B_EMPLÉADOS
DECLARE
    CURSOR C_EMP IS
        SELECT CEDULA_JEFE, COUNT(CEDULA) CANTIDAD
        FROM B_EMPLÉADOS WHERE CEDULA_JEFE IS NOT
        NULL
        GROUP BY CEDULA_JEFE;
BEGIN
    PACK_EMP.V_PER.DELETE;
    FOR REG IN C_EMP LOOP
        PACK_EMP.V_PER(REG.CEDULA_JEFE) := 
        REG.CANTIDAD;
    END LOOP;
END;
/
```

Ejemplo: Solución con variables de paquete

- **Paso 3: Creación de un trigger a nivel de fila para actualizar y verificar la cantidad de empleados por jefe**

```
CREATE OR REPLACE TRIGGER T_BIU_EMPLEADOS
BEFORE INSERT OR UPDATE OF CEDULA_JEFE ON B_EMPLEADOS
FOR EACH ROW
BEGIN
    IF :NEW.CEDULA_JEFE IS NOT NULL THEN
        IF UPDATING THEN
            PACK_EMP.V_PER(:OLD.CEDULA_JEFE) :=
                PACK_EMP.V_PER(:OLD.CEDULA_JEFE) - 1;
        END IF;
        PACK_EMP.V_PER(:NEW.CEDULA_JEFE) :=
            PACK_EMP.V_PER(:NEW.CEDULA_JEFE) + 1;
        IF PACK_EMP.V_PER(:NEW.CEDULA_JEFE) > 3 THEN
            RAISE_APPLICATION_ERROR(-20000, 'NO PUEDE
TENER MAS DE 3 EMPLEADOS POR JEFE');
        END IF;
    END IF;
END;
/
```

COMPOUND TRIGGERS: Simplificación del tratamiento de tablas mutantes (a partir de ORACLE 11g)

Un trigger compuesto (**compound trigger**), permite que se pueda codificar, en un solo trigger, diferentes momentos del tiempo de ejecución (AFTER | BEFORE) que afectan a un objeto.

Todos los triggers para un evento específico, pueden compartir una misma zona global de declaración de variables, cuyo estado es mantenido durante la vida del trigger. Una vez que la sentencia termina, se limpia el estado de estas variables.

En versiones anteriores de ORACLE, este tipo de funcionalidad sólo era posible combinando varios triggers (a nivel de sentencia y filas) que combian variables globales a nivel de un paquete.

```
CREATE OR REPLACE TRIGGER BIU_TEST_COMPUESTO
```

```
FOR INSERT OR UPDATE ON B_EMPLEADOS
```

```
COMPOUND TRIGGER
```

```
TYPE T_PER IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```

```
V_PER T_PER;
```

```
CURSOR C_EMP IS
```

```
SELECT CEDULA_JEFE, COUNT(CEDULA) CANTIDAD
```

```
FROM B_EMPLEADOS WHERE CEDULA_JEFE IS NOT NULL
```

```
GROUP BY CEDULA_JEFE;
```

```
BEFORE STATEMENT IS
```

```
BEGIN
```

```
V_PER.DELETE;
```

```
FOR REG IN C_EMP LOOP
```

```
    V_PER(REG.CEDULA_JEFE) := REG.CANTIDAD;
```

```
END LOOP;
```

```
END BEFORE STATEMENT;
```

```
BEFORE EACH ROW IS
```

```
BEGIN
```

```
IF :NEW.CEDULA_JEFE IS NOT NULL THEN
```

```
    V_PER(:NEW.CEDULA_JEFE) := V_PER(:NEW.CEDULA_JEFE) + 1;
```

```
    IF V_PER(:NEW.CEDULA_JEFE) > 3 THEN
```

```
        RAISE_APPLICATION_ERROR(-20000,
```

```
            'No puede tener mas de 3 empleados');
```

```
    END IF;
```

```
END IF;
```

```
END BEFORE EACH ROW;
```

```
END;
```

TRIGGERS DE SUSTITUCIÓN (INSTEAD OF)

- Los disparadores DML actúan además de la operación INSERT, DELETE, UPDATE.
- Los disparadores de sustitución en cambio, actúan en lugar de la operación DML y se usan principalmente para permitir la modificación de una vista

Triggers de sustitución

- Para aplicar un trigger de sustitución la vista tiene que ser ‘modificable’; lo que significa que no debe contener operaciones de agregación (SUM, AVG, GROUP BY), operaciones de conjuntos (UNION, MINUS...), el operador DISTINCT.
- Las opciones AFTER y BEFORE no se aplican para operaciones de sustitución.

Ejemplo de Trigger de sustitución

-- Primero se crea la vista V_PER_LOCAL

```
CREATE OR REPLACE VIEW V_PER_LOCAL
AS
SELECT bp.id,
bp.tipo_persona, bp.nombre, bp.apellido,
bp.direccion, bp.telefono, bp.correo_electronico,
bp.es_cliente, bp.es_proveedor, bp.fecha_nacimiento,
bp.id_localidad, lo.nombre nombre_localidad
FROM b_personas bp, b_localidad lo
WHERE bp.id_localidad=lo.id;
```

Ejemplo de Trigger de sustitución

```
CREATE OR REPLACE TRIGGER T_INSTEAD_DML
INSTEAD OF INSERT OR UPDATE
ON V_PER_LOCAL
FOR EACH ROW
DECLARE
    V_DPTO NUMBER;
    FUNCTION F_VER_LOCALIDAD(DPTO NUMBER) RETURN BOOLEAN
    IS
        V_ID_LOCALIDAD NUMBER;
    BEGIN
        BEGIN
            SELECT ID INTO V_ID_LOCALIDAD FROM b_localidad
            WHERE ID = :NEW.ID_LOCALIDAD;
        RETURN TRUE;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN FALSE;
    END;
END;
```

Ejemplo de Trigger de sustitución

```
BEGIN
```

```
    IF INSERTING      THEN
        IF (:NEW.ID_LOCALIDAD IS NOT NULL) THEN
            -- COMPRUEBA SI EXISTE LA LOCALIDAD --
            IF NOT F_VER_LOCALIDAD(:NEW.ID_LOCALIDAD) THEN
                INSERT INTO b_localidad(ID, NOMBRE)
                VALUES (:NEW.ID_LOCALIDAD, :NEW.NOMBRE_LOCALIDAD);
            END IF;
        END IF;
        INSERT INTO b_personas
        (id, tipo_persona, nombre, apellido, direccion, telefono,
        correo_electronico, es_cliente, es_proveedor,
        fecha_nacimiento, id_localidad)
        VALUES (:NEW.id, :NEW.tipo_persona, :NEW.nombre, :NEW.apellido,
        :NEW.direccion, :NEW.telefono, :NEW.correo_electronico,
        :NEW.es_cliente, :NEW.es_proveedor,
        :NEW.fecha_nacimiento, :NEW.id_localidad);
```

Ejemplo de Trigger de sustitución

```
-- ACTUALIZA DATOS DEL EMPLEADO
ELSIF UPDATING THEN
UPDATE b_personas
SET tipo_persona = :NEW.tipo_persona,
nombre = :NEW.nombre, apellido = :NEW.apellido,
direccion = :NEW.direccion, telefono = :NEW.telefono,
correo_electronico = :NEW.correo_electronico,
es_cliente = :NEW.es_cliente,
es_proveedor = :NEW.es_proveedor,
fecha_nacimiento = :NEW.fecha_nacimiento,
id_localidad = :NEW.id_localidad
WHERE ID = :NEW.ID;
END IF;
END;
```