

PL/SQL

Francisco Moreno
Universidad Nacional

Disparadores (*Triggers*)

- Los *triggers* son bloques PL/SQL que se ejecutan de manera **implícita** cuando se ejecutan operaciones (eventos DML) INSERT, DELETE, UPDATE a una tabla*.
- Se usan principalmente para establecer reglas **complejas de integridad** y **labores de auditoría**.



¿Por ejemplo?

- * También existen *triggers* que se disparan ante eventos de DDL (ej. BEFORE DDL) y eventos de *database* (ej. LOGON y LOGOFF) y sobre otros objetos de la BD (vistas, DATABASE, entre otros).

- Un *trigger* tiene asociado:
 - Un evento
 - Un momento
 - Un tipo
- El evento se refiere a la **operación** que se efectúa sobre la tabla (INSERT, DELETE o UPDATE).

- El momento, se refiere a **cuándo** se dispara el *trigger* en relación con el evento. Sus posibles valores son BEFORE y AFTER*.
- El tipo indica el **número de veces que el cuerpo del *trigger* se ejecuta**: por la operación en conjunto (*trigger de operación*) o por cada fila procesada (*trigger de fila*). En este último caso se debe adicionar la cláusula FOR EACH ROW

* Y también INSTEAD OF para vistas.

- Haciendo las diferentes combinaciones, para una tabla se pueden crear máximo doce **tipos** de *triggers*
- Se pueden tener **varios** *triggers* del **mismo** tipo asociados con una misma tabla
- Los *triggers* se ejecutan **implícitamente** cuando ocurre el evento
- Normalmente en los *triggers* **no** se puede^{*} usar ni COMMIT ni ROLLBACK

^{*} Es posible con *autonomous transactions*, por simplicidad se omitirán.

Sintaxis esencial de un *trigger*:

```
CREATE [OR REPLACE] TRIGGER nombre_trigger  
momento evento ON tabla  
[FOR EACH ROW → Para especificar  
                    triggers de fila  
[WHEN condición]]  
bloque de PL/SQL
```

Si el evento es UPDATE, se puede(n) especificar la(s) columna(s) que dispararán el *trigger* con la palabra OF (si no se especifican, el *trigger* se dispara con cualquier columna de la tabla que sea actualizada)

El bloque comienza con la palabra DECLARE o BEGIN

Si es un *trigger de fila*, se puede acceder al valor de una columna **antes de la actualización** con `:OLD.nomcolumna` y al valor **después de la actualización** con `:NEW.nomcolumna`

Evento	Valor anterior	Valor nuevo
INSERT	NULL	El valor que se va a insertar
DELETE	El valor antes del borrado	NULL
UPDATE	El valor antes de la actualización	El valor después de la actualización

Ejemplo: *Trigger* de operación

```
CREATE TABLE PRODUCTO(codigo NUMBER(6) PRIMARY KEY,  
                        precio NUMBER(6));
```

```
CREATE OR REPLACE TRIGGER control_insercion_productos  
BEFORE INSERT ON producto  
BEGIN  
    IF TO_CHAR(SYSDATE, 'D') = '7' THEN  
        RAISE_APPLICATION_ERROR(-20506, '¡No los domingos!');  
    END IF;  
END;  
/
```

Domingo (puede ser 1 o 7 según la config. del NLS)

SQLCODE y SQLERRM asignados por el usuario

La instrucción `RAISE_APPLICATION_ERROR` **abortará** la operación de inserción si esta se intenta hacer un domingo.


```
BEGIN
INSERT INTO producto VALUES (1, 10) ;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE (SQLCODE || | SQLERRM) ;
END ;
/
```

Ejemplo: *Trigger* de Fila

Sean las tablas:

```
CREATE TABLE cuenta(  
  nro_cta NUMBER(10) PRIMARY KEY,  
  saldo NUMBER(8) NOT NULL CHECK (saldo >= 0)  
);
```

```
INSERT INTO cuenta VALUES(1, 100);  
INSERT INTO cuenta VALUES(2, 200);
```

```
CREATE TABLE retiro(  
  nro_retiro NUMBER(10) PRIMARY KEY,  
  cta NUMBER(10) REFERENCES cuenta NOT NULL,  
  valor NUMBER(8) NOT NULL CHECK (valor > 0)  
);
```

Cuando se va a insertar un retiro:

- Si su valor es mayor al saldo de la cuenta, se debe rechazar la transacción.
- Si el retiro se puede hacer, se debe actualizar el saldo de la cuenta

```
CREATE OR REPLACE TRIGGER control_retiro
BEFORE INSERT ON retiro
FOR EACH ROW
DECLARE
    saldo_cta cuenta.saldo%TYPE;
BEGIN
    SELECT saldo INTO saldo_cta
    FROM cuenta WHERE nro_cta = :NEW.cta;
    IF saldo_cta < :NEW.valor THEN
        RAISE_APPLICATION_ERROR(-20505, '¡Fondos insuficientes!');
    ELSE
        UPDATE cuenta SET saldo = saldo - :NEW.valor
        WHERE nro_cta = :NEW.cta;
    END IF;
END;
/
```

¿Qué pasa si el valor del retiro fuese **negativo**?

```
INSERT INTO retiro VALUES (22, 1, 70) ;  
INSERT INTO retiro VALUES (28, 1, 15) ;  
INSERT INTO retiro VALUES (29, 1, 20) ;
```

Ejercicio:

Analizar las consecuencias de **actualizar** o **borrar** tanto en la tabla cuenta como en la tabla retiro.

Por ejemplo:

¿sería válido **actualizar** el valor de un retiro?

La cláusula WHEN

- Sirve para agregar una condición adicional para activar el *trigger*
- Dentro de la condición WHEN para referirse a NEW y a OLD no se usan los dos puntos (:)

Ejemplo. Sea:

```
CREATE TABLE auditoria  
(usuario VARCHAR2(20), cuando DATE, cta NUMBER(6),  
aumento NUMBER(10));
```

Usando WHEN:

```
CREATE or REPLACE TRIGGER
quien_sube_retiro
BEFORE UPDATE OF valor ON retiro
FOR EACH ROW
WHEN (NEW.valor > OLD.valor)
BEGIN
    INSERT INTO auditoria
    VALUES (USER, SYSDATE, :old.cta,
            :NEW.valor - :OLD.valor );
END;
/
```

Sin usar WHEN:

```
CREATE or REPLACE TRIGGER
quien_sube_retiro
BEFORE UPDATE OF valor ON retiro
FOR EACH ROW
BEGIN
    IF :NEW.valor > :OLD.valor
    THEN
        INSERT INTO auditoria
        VALUES (USER, SYSDATE, :old.cta,
                :NEW.valor - :OLD.valor );
    END IF;
END;
/
```

Conclusión: WHEN evita el uso de un IF *explícito dentro de la sección ejecutable* del trigger

- La ejecución de un *trigger* es **transaccional**, es decir, si un *trigger* de fila afecta *n* registros y si *uno solo de ellos aborta*, entonces todo el proceso se aborta.
- El orden de ejecución de los *triggers* es el siguiente:
 1. Se ejecutan (si los hay*) los *triggers* BEFORE de operación
 2. Se ejecutan sucesivamente **para cada** fila:
 - (Si los hay*) los *triggers* BEFORE de fila
 - Se ejecuta la operación DML correspondiente **
 - (Si los hay*) los *triggers* AFTER de fila
 3. Se ejecutan (si los hay*) los *triggers* AFTER de operación

* Si hay varios *triggers* con idéntica definición (a partir de Oracle 8i), el orden de ejecución *entre ellos* es “aleatorio” → [Ver diapositiva final](#).

** Tarea: analizar en relación con los *triggers* cuando se realizan las verificaciones de integridad (clave primaria, foránea, *checks*, entre otras).


```
CREATE TABLE jefe(cc NUMBER(8) PRIMARY KEY,  
salario NUMBER(8));  
INSERT INTO jefe VALUES(10, 400);  
INSERT INTO jefe VALUES(20, 200);
```

```
CREATE OR REPLACE TRIGGER bef_upd_row1  
BEFORE UPDATE ON jefe  
FOR EACH ROW  
BEGIN  
DBMS_OUTPUT.PUT_LINE('Bef Upd fila1 cc:' || :NEW.cc);  
DBMS_OUTPUT.PUT_LINE(:NEW.salario);  
IF :NEW.salario > 100 THEN  
    :NEW.salario := 0;  
END IF;  
END;  
/
```

```
CREATE OR REPLACE TRIGGER bef_upd_row2
BEFORE UPDATE ON jefe
FOR EACH ROW
BEGIN
DBMS_OUTPUT.PUT_LINE(' Bef Upd fila2 cc:' || :NEW.cc);
    DBMS_OUTPUT.PUT_LINE(:NEW.salario);
    :NEW.salario := :NEW.salario + 50;
END;
/
```

Ahora para comprobar el funcionamiento:

```
UPDATE jefe set salario = 100;
```

Resultados

Bef Upd fila2 cc:10

100

Bef Upd fila1 cc:10

150

*Triggers de fila afectando a
la fila con cc 10*

Bef Upd fila2 cc:20

100

Bef Upd fila1 cc:20

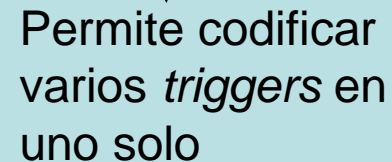
150

*Triggers de fila afectando a
la fila con cc 20*

Ejercicio: intercambiar los nombres de bef_upd_row1 y de bef_upd_row2 y ejecutar todo de nuevo y ver que sucede

- Note que en un *trigger* de fila BEFORE se pueden cambiar los valores NEW.
- En un *trigger* de fila AFTER no se pueden cambiar dichos valores.
- Así, si hay varios *triggers* de fila BEFORE y se desea hacer una validación, con respecto al valor final con que el quedan los valores NEW, esta validación se debe hacer en un *trigger* de fila AFTER (o en el último *trigger* de fila BEFORE si hay un orden establecido → ver [FOLLOWS](#)).
- Los *triggers* de fila AFTER también son adecuados para grabar los valores NEW en tablas de auditoría (ya que YA han pasado las verificaciones de integridad, su posibilidad de ser rechazados es baja...)

- Tarea: consultar las opciones:
IF INSERTING ..., IF UPDATING..., IF DELETING...
- A partir de Oracle 11g se soporta la opción *FOLLOWS*.
- También se soportan *Compound Triggers*.



Permite codificar
varios *triggers* en
uno solo

Ver:

<https://oracle-base.com/articles/11g/trigger-enhancements-11gr1>