PL/SQL

- **En caso de error**
- **Bloques**
- **Expresiones**
- **Registros**
- **Tablas**
- **Conversiones de datos**
- **Variables y constantes**
- **Cursores**
- **Excepciones**
- **Estructuras de control**

EN CASO DE ERROR

- **¿Qué hacer en caso de error?**
  - *"La base de datos no funciona bien, no hace lo que quiero"*
  - Proceso de resolución de errores:
    1. Revisar la tabla user_errors
       1. Show errors
       2. Select * from user_errors;
    2. ¿Sintaxis correcta?
    3. ¿Problema con una consulta?
       1. La consulta no devuelve resultados
       2. Fuera del disparador la consulta funciona pero dentro no
  - Comenta y vencerás.

INTRODUCCIÓN (I)

- **PL/SQL:** Lenguaje de programación procedimental estructurado en bloques que amplía el lenguaje estándar SQL.
- Permite:
  - Manipular datos de una BD Oracle.
  - Usar técnicas procedurales (bucles, ...)
  - Controlar las filas de una consulta una a una
  - Controlar errores (excepciones) definidas por el usuario o propios de Oracle (predefinidos)
  - Disparadores
- No diferencia las minúsculas de las mayúsculas

  CLIENTE == cliente

BLOQUES

- Es la unidad básica de cualquier programa PL/SQL.
- **Estructura básica** de un bloque (sólo es obligatorio el conjunto de sentencias ejecutables):

```
DECLARE
    /* Declaraciones de uso local: variables, cursores, y excepciones de
    usuario */
BEGIN
    /* Proceso: conjunto de sentencias ejecutables */
EXCEPTION
    /* Excepciones: zona de control de errores */
END;
```

BLOQUES (II)

- Las únicas instrucciones permitidas dentro de un bloque son: INSERT, UPDATE, DELETE Y SELECT + manipulación de datos + control de transacciones.
- La anidación de bloques se realiza entre las etiquetas *BEGIN Y EXCEPTION* y sólo se tiene una definición de variables.
- Instrucciones no permitidas son: DROP, CREATE, ALTER, …
- Los comentarios se ponen: /* Comentarios */
- No distingue mayúsculas y minúsculas

BLOQUES (III)

- Tipos:
  - ***Anónimo***: construido de manera dinámica y se ejecuta una vez.
    - Estructura básica se compilan cada vez que son ejecutados y no se guardan en la BD.
  - ***Nominado:*** bloque anónimo con etiqueta. Lo primero es la etiqueta. PE -> **<<*insertarEnTabla*>>**
    - ***Subprogramas:*** procedimientos, paquetes y funciones almacenadas en la BD ya compiladas. Se ejecutan múltiples veces mediante llamadas explícitas.
    - Remplazar la palabra clave *DECLARE* con las palabras ***CREATE OR REPLACE PROCEDURE nombre_proc AS***
  - ***Disparadores:*** bloques nominados almacenados en la BD (código fuente). Se ejecutan múltiples veces de forma implícita mediante eventos sobre una tabla (INSERT, UPDATE o DELETE). Se compila cada vez que se ejecuta. (se explicarán más adelante)

EXPRESIONES

- Tipos de expresiones:
  - Aritméticas : + - * /
  - Comparaciones : = != > > >= <=
  - Concatenación de caracteres ||
- Tipos de variables:
  - Escalares. Definidos por el lenguaje
    - NUMBER, CHAR, VARCHAR, VARCHAR2, DATE, BOOLEAN (TRUE, FALSE, NULL)
  - Compuestos: Definidos por el usuario
    - Registros
    - Tablas y matrices. Pueden almacenar registros y escalares

REGISTROS

- Registros. Definición

  TYPE tipo_reg IS RECORD (campo1tipo1[NOTNULL][:=expr];... campontipon[NOTNULL][:=expr];);
- Registro. Creación de variables

  Nombre_variable tipo_reg
- Referenciar las variables

  Nombre_variable.campo:=valor
- Ejemplo

```
DECLARE
  TYPE tcliente IS RECORD (
            DNI  NUMBER (8,0);
            Nombre VARCHAR (30);
            FECHA DATE;);
RCLIENTE   tcliente;
```

TABLAS

- Tablas definición.
  - TYPE tipo_tabla IS TABLE OF tipo
    INDEX BY BINARY_INTEGER;
- Tablas. Definición de variables: vcliente tipo_tabla;
- Tablas. Referencia: Tabla(indice).campo;
- Tablas. Ejemplo

```
DECLARE
  TYPE ttabla IS TABLE OF cliente%ROWTYPE
    INDEX BY BINARY_INTEGER;
  v_cliente ttable;
BEGIN
   SELECT * INTO v_cliente(2252480) FROM cliente WHERE  DNI=22522480;
DBMS_OUTPUT.PUT_LINE (vcliente(2252480).nombre);
END;
```


Tablas. Atributos

- Tablas. Atributos

Atributo	Tipo Devuelto	Descripción
COUNT	NUMBER	Devuelve el número de filas de la tabla
DELETE (nº)	N/A	Borra las filas de una tabla
EXITS (nº)	BOOLEAN	Devuelve TRUE si existe en la tabla la fila especificada
FIRST	BYNARY_ INTEGER	Devuelve el índice de la primera fila
LAST	BYNARY_ INTEGER	Devuelve el índice de la última fila
NEXT	BYNARY_ INTEGER	Devuelve el índice de la fila de la tabla que sigue a la fila especificada
PRIOR	BYNARY_ INTEGER	Devuelve el índice de la fila de la tabla que precede a la fila especificada

CONVERSIONES DE DATOS

- Entre escalares se pueden convertir datos de distintas familias excepto las propias restricciones de las variables (Ej. CHAR (10) no puede en VARCAHAR2(1))

Función	Descripción
TO_CHAR	Convierte el argumento en tipo VARCHAR2
TO_DATE	Convierte su argumento en tipo DATE
TO_NUMBER	Convierte el argumento en tipo NUMBER

Declaración de variables y constantes

- Variables: Se utilizan para almacenar valores devueltos por una consulta o para realizar cálculos intermedios.
- Constantes: Son campos definidos e inalterables
- Pasos a seguir
  - Definición Nombre_campo: nombre de la variable
  - Declaración **Nombre_campo tipo [CONSTANT][NOTNULL] [:=VALOR];**
  - Asignación Variable Tipo. Un identificador es el nombre de la vble.
  - Tipo: puede ser:
    - Tipo de datos: tipo de dato de la variable
    - Identificador%TYPE Se refiere al tipo usado por una columna)
    - Identificador%ROWTYPE Es una fila vble con los mismos nombres y tipos que las columnaas de una tabla o fila recuperada de un cursor)
    - [CONSTANT] Palabra reservada para def. de ctes
    - [NOTNULL]: Obliga a tener valor
    - [:=VALOR] Asigna como valor inicial con un valor cte

CTES Y VBLES. EJEMPLO

```
DECLARE
   DNI          NUMBER (8,0);
   Nombre       VARCHAR (30);
   Factor       CONSTANT   NUMBER(3,2):=0.10;
   DNI2         cliente.DNI%TYPE;
   Rcliente     cliente%ROWTYPE;
                (tendría los campos: Rcliente.DNI,  Rcliente.Nombre …)
   precio       NUMBER:= 300; (inicializa a un valor)
```


CURSORES

- Definición:
  - Cursor es un área de trabajo definida para las consultas que devuelven más de una fila.
  - Permite la manipulación de datos
  - Se le pueden pasar parámetros
- Tipos de cursores
  - **Cursor simple**
    - CURSOR nombre_cursor IS sentencia SELECT;
    - Ejemplo

```
DECLARE
….
CURSOR c_cliente  IS SELECT cliente.DNI, cliente.nombre FROM cliente
   WHERE cliente.tipo_trabajo=´jefe´;
```

CURSORES

- **Cursores con paso de parámetros**
  - CURSOR nombre_cursor (nombre_parametro tipo_parametro) IS Sentencia SELECT utilizando los parámetros
  - EJEMPLO

    CURSOR c_cliente (tipotrabajo VARCHAR2) IS SELECT cliente.DNI, cliente.Nombre FROM cliente WHERE cliente.tipo_trabajo=tipotrabajo;
- **Cursores con actualización**
  - Se dejan preparados para modificar las filas devueltas y se generan bloqueos exclusivos sobre las filas activas
  - SINTAXIS :CURSOR nombre_cursor IS sentencia SELECT FOR UPDATE [OF nombre_columna]
  - Ejemplo

    CURSOR c_cliente IS SELECT cliente.DNI, cliente.nombre FROM cliente WHERE cliente.tipo_trabajo=´jefe´ FOR UPDATE OF NOMBRE;

CURSORES:OPERACIONES

- Operaciones con cursores:
  - OPEN: abre los cursores
  - FETCH : lee los datos
  - CLOSE: cierra los cursores
- Pasos a seguir:

  1.- Declarar el cursor en la zona *DECLARE*

  2.- Abrir el cursor en la zona de procedimiento

  3.- Leer el cursor. Es necesario ejecutar un bucle para leer todos los cursores

  4.- Cerrar el cursor

CURSORES

- Atributos de los cursores. Se le añaden al nombre del cursor. No devuelven ningún tipo sino un valor que se pueda emplear como parte de una expresión.

Atributo	Descripción
%NOTFOUND	Devuelve TRUE si la ultima lectura falla porque no hay filas disponibles y FALSE si recupera. Se usaa para detectar el final
%FOUND	El contrario de %NOTFOUND
%ROWCOUNT	Devuelve el número de fila leida
%ISOPEN	Devuelve TRUE si el cursor esta abierto y FALSE si no lo está

EXCEPCIONES

- Declaración de excepciones:
  - Se declaran por el usuario
  - Utilizan las funciones de error de SQL y PL/SQL
  - Sintaxis:
    - Nombre_excepción EXCEPTION;
  - Ejemplo

    DECLARE

    ….

    User_exception EXCEPTION;

ESTRUCTURAS DE CONTROL

- Realizan el control del comportamiento del bloque. Son de varios tipos:
  - **CONDICIONALES (IF):** Ejecuta una o varias sentencias dependiendo de una condición:
    - SINTAXIS:

      IF condición THEN sentencias ejecutables;
      [ELSIF condición THEN sentencias_ejecutable]
      [ELSE sentencias_ejecutable];
      ……
      ENDIF

ESTRUCTURAS DE CONTROL

- **BUCLES**
  - Bucles simples (LOOP).
  - Bucles condicionales (WHILE)
  - Bucles numéricos (FOR)
  - Bucles sobre cursores
  - Bucles sobre sentencias SELECT
- Para romper el bucle se usa EXIT, GOTO o RAISE

BUCLES

- SIMPLE
  - Sintaxis: [nombre_bucle] LOOP
    sentencias;
    END LOOP;
  - Ejemplo

```
DECLARE
        V_contador BINARY_INTEGER:=1;
BEGIN
   LOOP
        INSERT INTO tabla_temporal
                VALUES (v_contador,'ejemplo');
        v_contador:=v_contador+1;
        EXIT WHEN v_contador>50;
   END LOOP;
END;
/
```


BUCLES

- CONDICIONAL. La condición se evalúa antes de entrar en el bucle
  - Sintaxis: [nombre_bucle] WHILE pl/sql_condición LOOP
    sentencias;
    END LOOP;
  - Ejemplo

```
DECLARE
        V_contador BINARY_INTEGER:=1;
BEGIN
   WHILE v_contador<=50; LOOP
        INSERT INTO tabla_temporal
        VALUES (v_contador,'ejemplo');
        v_contador:=v_contador+1;
        END LOOP;
END;
/
```

BUCLES

- NUMÉRICO: Se ejecutan una sola vez por cada elemento de rango definido
  - Sintaxis: [nombre_bucle] FOR indice IN [REVERSE]
    exp_n1..exp_n2 LOOP
    sentencias;
    END LOOP;
    - indice: variable numérica de control empieza en exp_n1 y termina en exp_n2 en pasos de 1 . Se declara implícitamente como un BINARY_INTEGER
    - [REVERSE]: La cuenta se hace al revés
    - Exp_n1, y exp_n2 pueden ser ctes o cualquier expresión que devuelva un número
  - Ejemplo

```
BEGIN
   FOR v_contador IN 1..50 LOOP
        INSERT INTO tabala_temporal
        VALUES (v_contador,'ejemplo');
        END LOOP;
END;
```


BUCLES

- BUCLES SOBRE CURSORES. Se ejecutan por cada fila que se recupera del cursor.
  - Pasos a seguir
    - Se abre el cursor
    - Se realiza la lectura y se ejecutan las sentencias del bucle hasta que no hay mas filas
    - Se cierra el cursor
  - Sintaxis: [nombre_bucle]
    FOR nombre_registro IN nombre_cursor
    LOOP sentencias
    END LOOP;

    Se le puede pasar parámetros a los cursores

BUCLES

- BUCLES SOBRE SENTENCIAS SELECT: Es el mismo concepto anterior pero sin declaración de cursores
  - Sintaxis: [nombre_bucle]

```
FOR nombre_registro IN (sentencia_select)
        LOOP sentencias
        END LOOP;
```

    Cuando se sale con exit o error del bucle el cursor interno que genera ORACLE se cierra.
  - Ejemplo

```
BEGIN
FOR registro IN (SELECT DNI, nombre FROM cliente)
LOOP
   INSERT INTO tabla_temporal (contador, texto) VALUES (registro.DNI,
   registro.nombre);
END LOOP;
END;
```


SENTENCIAS DE CONTROL

- GOTO y ETIQUETAS: transfiere el control a la sentencia o bloque siguiente a la etiqueta indicada
  - Sintaxis <<etiqueta>>

```
GOTO <<etiqueta>>;
```

  - La sentencia puede ir a otro bloque o sub-bloque pero nunca a la zona de excepciones.
  - La sentencia siguiente a la etiqueta debe ser un ejecutable.
  - No se puede realizar saltos al interior de un bloque interno, un IF o de un bucle.
- NULL. Sentencia ejecutable que no hace nada.

Disparadores

- **Definición**
- **Tablas Mutantes**

DISPARADORES

- Bloques de PL/SQL nominados, con las secciones:
  - declarativa
  - ejecutable
  - manejo de excepciones
- Almacenados en la BD (diccionario de datos: *user_triggers)* y asociados a una tabla.

DISPARADORES

- Puede afectar a *n* filas.
- Se ejecuta de manera **implícita** ante eventos (operación de inserción, modificación o borrado sobre una BD)
- Se compila cada vez que lo activa --> más lentos

DISPARADORES: Aplicaciones

- **Restricciones de Integridad complejas. IMPORTANTE:** no se deben usar para garantizar el cumplimiento de las RI a nivel de esquema !!! (el esquema ha de contener toda la semántica que permita sin utilizar disparadores)
- **Auditoría:** registro de los cambios realizados y quién los realizó
- **Aviso automático a otros programas** de llevar a cabo una determinada acción
- **Actualización en cascada**

DISPARADORES: Utilización

- **No disparadores recursivos:** agotan memoria.
- **Sólo se almacena el código fuente del disparador.** Se compila cada vez que se va a ejecutar (lectura del diccionario de datos). Por lo tanto si el disparador tiene más de 60 líneas de cuerpo conviene hacer un procedimiento.
- **No utilizar para RI simples (a nivel de esquema)**.
- **Identificador único** para cada elemento.

DISPARADORES: Sintaxis

- **Creación:** (se activan al crearlos)

```
CREATE [OR REPLACE] TRIGGER
  <nombre_disparador>
    {BEFORE | AFTER} evento ON referencia_tabla
    [ FOR EACH ROW [WHEN condición_evento]]
    cuerpo_disparador;

ALTER TABLE nombre_tabla
      {ENABLE | DISABLE} ALL TRIGGERS;
```

DISPARADORES: Sintaxis

- **Eliminación:**

  DROP TRIGGER nombre_disparador;

- **Activación/Desactivación:**

  ALTER TRIGGER nombre_disparador {DISABLE | ENABLE};

DISPARADORES: Componentes (1)

- **Nombre disparador:**
  - Siguen las mismas normas de nomenclatura que otros identificadores en la BD
- **Replace:**
  - Se utiliza para sobreescribir un disparador existente
- **Before/After:**
  - Instante de ejecución del disparador con respecto al evento

DISPARADORES: Componentes (2)

- **Evento:**
  - Tipo de orden DML sobre una tabla que provoca la activación del disparador {INSERT | DELETE | UPDATE [OF <*lista de columnas*>]}. La *lista de columnas* sólo tiene sentido en el evento UPDATE
- **Nivel:**
  - FOR EACH ROW: disparadores con **nivel de fila**. Se activan una vez por cada fila afectada por el evento
  - FOR EACH STATEMENT: disparadores con **nivel de orden.** Se activan sólo una vez (antes o después de la orden).

DISPARADORES: Componentes (3)

- **When:**

  Sólo tiene sentido a **nivel de fila**. La *condición* se evalúa (*true o false).* No se pueden utilizar consultas anidadas.

DISPARADORES: Componentes (4)

- **Cuerpo:** bloque PL/SQL con las siguientes restricciones:
  - Un disparador no puede emitir ninguna orden de control de transacciones (COMMIT, ROLLBACK o SAVEPOINT)
  - Ningún procedimiento o función llamada por el disparador puede emitir órdenes de control de transacciones.

DISPARADORES: Componentes (5)

- No puede contener ninguna declaración de variables LONG o LONG RAW
- Restricciones en tablas a las que se puede acceder (Tablas Mutantes)
- No puede modificar las columnas de clave primaria.

Registros *:old* y *:new*

- Un disparador con **nivel de fila** se ejecuta en cada fila en la que se produce el suceso.
- *:old y :new* son registros que nos permiten acceder a los datos de la fila actual
- Tipo de los registros: *nombre_tabla%ROWTYPE;*

Suceso	:old	:new
INSERT	NULL	Nuevos valores
UPDATE	Valores almacenados	Nuevos valores
DELETE	Valores almacenados	NULL

Ejemplo *:old* y *:new*

```
CREATE SEQUENCE sec_estudiante start with 2;
CREATE TABLE estudiante ( codigo number(2) primary key);
CREATE OR REPLACE TRIGGER t_estudiante
 BEFORE INSERT ON estudiante FOR EACH ROW
BEGIN
 SELECT sec_estudiante.nextval INTO :new.codigo FROM dual;
END t_estudiante;
```

- NOTAS:
  - se ignoran los valores que se introducen como código de estudiante → se inserta el siguiente de la secuencia

INSERTING, DELETING Y UPDATING

- Predicados de los disparadores (booleanos), empleadas para determinar qué operación se está realizando en un disparador.

```
CREATE OR REPLACE TRIGGER Cambios
  BEFORE INSERT OR DELETE ON Alumnos
  FOR EACH ROW
DECLARE
  Cambio_tipo  CHAR(1);
```


INSERTING, DELETING Y UPDATING

```
BEGIN
    /* Usa 'I' para INSERT y 'D' Para DELETE */
    IF INSERTING THEN
        Cambio_tipo := 'I';
    ELSE
        Cambio_tipo := 'D';
    END IF;
END Cambios;
```

Tablas Mutantes

Qué son

- **Tablas que están siendo modificadas por una operación DML (INSERT, DELETE, UPDATE):**
  - En un disparador, la tabla sobre la que está definido
  - Tablas que serán actualizadas como consecuencia de la integridad referencial (P.e.: DELETE CASCADE)

En los disparadores (1)

- **A nivel de FILA, dentro del cuerpo de un disparador, no pueden existir:**
  - **lecturas o modificaciones** de tablas mutantes
  - **cambios de clave primaria, claves ajenas o claves alternativas** de las tablas que restringen (el resto de las columnas sí se pueden cambiar)

En los disparadores (2)

- A nivel de **SENTENCIA** no existen problemas de tablas mutante

En los disparadores (2)

TIPO DE DISPARADOR	ERROR DE TABLA MUTANTE
BEFORE INSERT ROW	NO
AFTER INSERT ROW	NO
BEFORE INSERT STATEMENT	NO
AFTER INSERT STATEMENT	NO
BEFORE DELETE ROW	SI
AFTER DELETE ROW	SI
BEFORE DELETE STATEMENT	NO
AFTER DELETE STATEMENT	NO
BEFORE UPDATE ROW	SI
AFTER UPDATE ROW	SI
BEFORE UPDATE STATEMENT	NO
AFTER UPDATE STATEMENT	NO

Ejemplo (1)

- **"Una zona tiene uno o varios departamentos y un departamento trabaja en una o ninguna zona".**

ZONA(Cod_Zona, Nom_Zona)

DC/UC

DEPARTAMENTO(Cod_Dep, Presupuesto, Cod_Zona)

```
CREATE SEQUENCE Secuencia_Departamento
    START WITH 100000
    INCREMENT BY 1;
```

Ejemplo (2)

```
CREATE TABLE Zona (
    Cod_Zona NUMBER(6) CONSTRAINT pk_zona PRIMARY KEY,
     Nom_Zona VARCHAR2(40) NOT NULL
);

CREATE TABLE Departamento (
    Cod_Dep NUMBER(6) CONSTRAINT pk_departamento PRIMARY KEY,
    Presupuesto NUMBER(8) NOT NULL,
    Cod_Zona NUMBER(2) NOT NULL
                CONSTRAINT fk_departamento_zona REFERENCES
                 Zona(Cod_Zona) ON DELETE CASCADE
);
```


Ejemplo 1 (1)

- EJEMPLO 1:

```
CREATE OR REPLACE TRIGGER Disparador1
     AFTER INSERT ON Zona FOR EACH ROW
    BEGIN
      INSERT INTO Departamento VALUES
      (Secuencia_Departamento.NEXTVAL, 10000000, :new.Cod_Zona);
    END Disparador1;
    /
```

Operación:

```
INSERT INTO Zona VALUES (1, 'CENTRO');
```

Ejemplo 1 (2)

- EJEMPLO 1. Comentarios:
  - La tabla *departamento* referencia a la tabla *zona* (FK).
  - En Oracle 9i NO da error de tabla mutante. En Oracle 8i, cada vez que se inserta un nuevo dato en la tabla *departamento*, Oracle controla la integridad referencial (el código *Departamento.Cod_Zona* ha de existir en la tabla *Zona* --> Realiza una lectura de la tabla *Zona*, que está mutando !!)

Ejemplo 2 (1)

- EJEMPLO 2:

```
CREATE OR REPLACE TRIGGER Disparador2
        BEFORE DELETE ON Departamento FOR EACH ROW
  DECLARE
    Var Zona%ROWTYPE;
  BEGIN
    SELECT * INTO Var FROM Zona WHERE Cod_Zona=:old.Cod_Zona;
END Disparador2;
 /
```

Operación1: DELETE FROM Departamento WHERE Cod_Zona=1;

Operación2: DELETE FROM Zona WHERE Cod_Zona=2;

Ejemplo 2 (2)

- EJEMPLO 2. Comentarios:
  - **Operación 1:** NO da error de tabla mutante: *departamento* referencia a la tabla *zona*, que sí se puede consultar, ya que no está mutando.
  - **Operación 2:** SI da error de tabla mutante, ya que, al borrar en la tabla *zona* (tabla mutante), se borran todas las las tuplas de la tabla *departamento* que referencian a la *zona* borrada. Esto activa el *disparador2* de *departamento*, que consulta la tabla *zona*, que en este caso sí esta mutando.

Ejemplo 3 (1)

- EJEMPLO 3:

```
CREATE OR REPLACE TRIGGER Disparador3
        BEFORE UPDATE ON Departamento FOR EACH ROW
  DECLARE
    Var Zona%ROWTYPE;
  BEGIN
    SELECT * INTO Var FROM Zona WHERE Cod_Zona=:old.Cod_Zona;
END Disparador3;
 /
```

Operación1: UPDATE Departamento SET Presupuesto = 0 WHERE Cod_Zona=1;
Operación2: UPDATE Zona SET NomZona = 'G' WHERE Cod_Zona=1;
Operación3: UPDATE Zona SET Cod_Zona = 10 WHERE Cod_Zona=1;

Ejemplo 3 (2)

- EJEMPLO 3. Comentarios:
  - **Operación 1:** NO da error de tabla mutante: *departamento* referencia a la tabla *zona*, que sí se puede consultar, ya que no está mutando.
  - **Operación 2:** NO da error de tabla mutante, la actualización en cascada no está implementada, así que la modificación del nombre de la zona no afecta para nada a la tabla *departamento*.
  - **Operación 3:** NO da error de tabla mutante, pero se produce un error debido a que estamos tratando de modificar el valor de una clave de la tabla Zona. Conviene recordar que Oracle no implementa la actualización en cascada.

En caso de error de Tabla Mutante

- CÓMO SOLUCIONAR UN ERROR DE TABLA MUTANTE
  - REVISAR PROPIEDADES DEL TRIGGER, ¿PUEDEN MODIFICARSE?
  - REVISAR ESQUEMA RELACIONAL, ¿PUEDE MODIFICARSE? → OBJETIVO, ELIMINAR ACCESOS A LA PROPIA TABLA SOBRE LA QUE SE DEFINE EL DISPARADOR