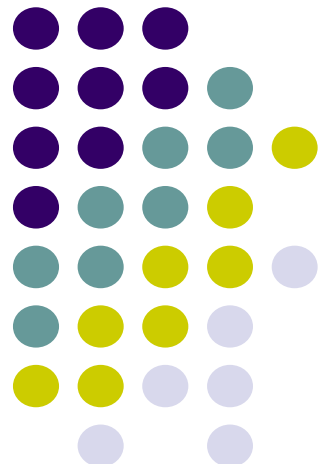
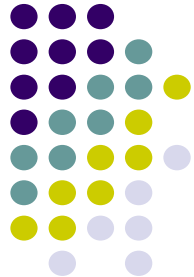
Triggers y PL/SQL



Triggers



Un trigger (disparador) define una acción que la base de datos siempre debería realizar cuando ocurre algún tipo de acontecimiento que la afecta.

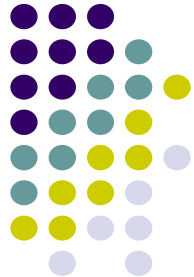
Se utilizan para

- mantener la integridad referencial,
- asegurar reglas de negocio complejas y
- auditar cambios en los datos.

Para definir un trigger se debe:

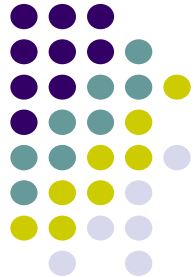
- Especificar las condiciones bajo las que el trigger será ejecutado.
- Especificar las acciones que se realizarán cuando el trigger se ejecute.

Triggers



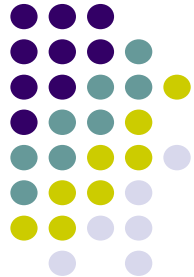
- Las acciones del trigger se definen mediante bloques de PL/SQL nominados con las secciones:
  - declarativa
  - ejecutable
  - manejo de excepciones
- Los triggers se almacenan en la BD (*user_triggers)* y están asociados a una tabla.
- Pueden afectar a *n* filas.
- Se ejecutan de manera **implícita** ante eventos (operación de inserción, modificación o borrado)
- Se compilan cada vez que se activan

Triggers: Sintaxis



create or replace trigger *NombreTrigger*
{before | after | instead of}
{delete | insert | update [of *NombreColumna*
    [, *NombreColumna*] ...] }
    [or {delete | insert | update [of *NombreColumna*
    [, *NombreColumna*] ...] }] ...
on {*NombreTabla* | *Nombre-Vista*}
[[referencing { old [as] *NombreViejo* | new [as] *NombreNuevo*} ...]
for each {row | statement} [when (*Condición*)]]
    Bloque pl/sql

Triggers: Sintaxis



- **Eliminación**

  DROP TRIGGER nombre_disparador;

- **Activación/Desactivación**

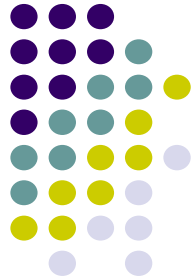
  ALTER TRIGGER nombre_disparador {DISABLE | ENABLE};

  ALTER TABLE nombre_tabla
  {ENABLE | DISABLE} ALL TRIGGERS;

- **Ver errores de compilación**

  SHOW ERRORS TRIGGER nombre_disparador;

Triggers: 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/Instead of:**
  - Instante de ejecución del disparador con respecto al evento que lo desencadena
- **Evento:**
  - Tipo de orden DML sobre una tabla que provoca la activación del disparador

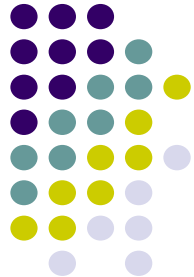
    INSERT | DELETE | UPDATE [OF <*lista de columnas*>]

Triggers: Componentes (2)

- **Nivel:**
  - FOR EACH ROW: disparadores a **nivel de fila**. Se activan una vez por cada fila afectada por el evento
  - FOR EACH STATEMENT: disparadores a **nivel de orden.** Se activan sólo una vez (antes o después de la orden).

- **When:** Sólo tiene sentido a **nivel de fila**. La *condición* se evalúa (*true o false).*

Triggers: Componentes (3)



- **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
  - No puede contener ninguna declaración de variables LONG o LONG RAW
  - No se puede acceder a cualquier tabla. Existen restricciones (Tablas Mutantes)
  - No puede modificar las columnas que son Primary Key

Registros *:old* y *:new*

- Un disparador a **nivel de fila** se ejecuta por 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.

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);
```

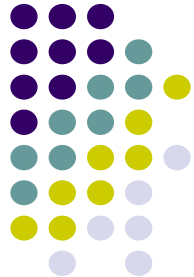
- La forma habitual de incluir información en la tabla sería:

```
insert into estudiante values
   (sec_estudiante.nextval);
```

- Con este trigger se ignoran los valores que un usuario pudiera introducir como código de estudiante y se inserta el siguiente valor de la secuencia.

```
create or replace trigger t_estudiante
  before insert on estudiante for each row
begin
  select sec_estudiante.nextval into :new.codigo from
  dual;
end;
/
```

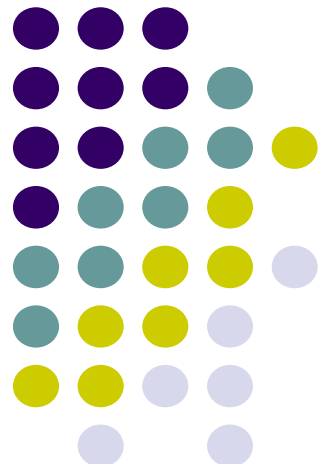
INSERTING, DELETING Y UPDATING



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

PL/SQL



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).
  - Definir disparadores.

- No diferencia las minúsculas de las mayúsculas

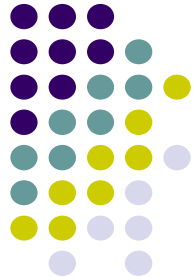
  CLIENTE == cliente

Bloques

- Es la unidad básica de cualquier programa PL/SQL.
- **Estructura básica** de un bloque: tiene tres partes, aunque sólo es obligatoria la parte del 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;
```

Expresiones y Tipos de Variable



- Tipos de expresiones:
  - Aritméticas : + - * /
  - Comparaciones : = != > > >= <=
  - Concatenación de caracteres ||
- Tipos de variables:
  - Escalares. Definidos por el lenguaje
    - VARCHAR2, DATE, LOB, LONG, NUMBER, etc.
  - Compuestos: Definidos por el usuario
    - Registros
    - Tablas y matrices. Pueden almacenar registros y escalares

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
  - ***Declaración***
    `<nom_variable> <tipo> [CONSTANT][NOT NULL] [:=VALOR];`
  - `[CONSTANT]` → Palabra reservada para definir constantes
  - `[NOT NULL]` → obliga a tener valor
  - `[:=VALOR]` → asigna el valor inicial (valor constante o expresión)
  - el <tipo> puede ser:
    - **Tipo de datos**: tipo de dato de la variable (de los tipos básicos SQL)
    - `campo%TYPE:` tipo de datos usado por una columna (o variable)
    - `tabla%ROWTYPE:` tipo de datos de la 'variable fila', con los mismos campos (y tipos) que los campos de la tabla

Declaración de Variables y Constantes

```
DECLARE
   DNI          NUMBER (8,0);
   Nombre       VARCHAR (30);
   Factor       CONSTANT   NUMBER(3,2):=0.10;
   DNI2         DNI%TYPE;
   Rcliente     cliente%ROWTYPE;
        /* (tendría los campos: Rcliente.DNI,  Rcliente.Nombre …) */
   precio       NUMBER:= 300; (inicializado a un valor)
   IVA          NUMBER CONSTANT :=16;
```

Estructuras De Control

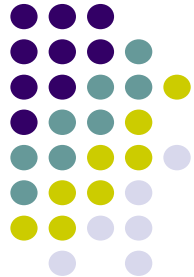
- Realizan el control del flujo del bloque.
- Son de varios tipos:

IF *(condicionales):*

Ejecuta una o varias sentencias dependiendo de una condición:

```
IF <condición>
    THEN <sentencias ejecutables>;
    [ELSIF <condición>
           THEN <sentencias_ejecutable>]
    [ELSE <sentencias_ejecutable>];
    ……
END IF;
```

Sentencia de cambio de control



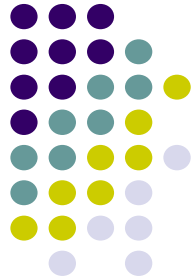
- **GOTO** y <etiquetas>: transfiere el control a un punto concreto dentro del bloque, la sentencia o bloque siguiente a la etiqueta indicada

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

  - La sentencia puede ir a otro bloque o sub-bloque pero nunca a la zona de excepciones.
  - No se pueden realizar saltos al interior de un bloque condicional o al interior de un bucle.
  - La sentencia siguiente a la etiqueta debe ser ejecutable

- **NULL :** sentencia ejecutable que no hace nada.

Estructuras De Control



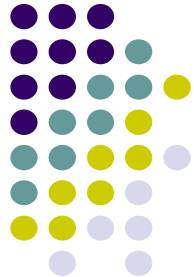
- **BUCLE:** bloque cuya ejecución se repite
  - **Tipos:**
    - Bucles simples (**`LOOP`**).
    - Bucles condicionales (**`WHILE`**).
    - Bucles numéricos (**`FOR`**).
    - Bucles sobre cursores.
    - Bucles sobre sentencias **`SELECT.`**
  - Para romper el bucle se debe usar la instrucción **`EXIT`** (aunque también se puede salir con **`GOTO`** o con **`RAISE`**)

Bucles: bucle simple

```
[etiqueta_nombre_bucle] LOOP
                              sentencias; ...
                          END LOOP;
```

```
SQL> create or replace procedure prueba is
  2   v_contador number :=1;
  3  begin
  4  loop
  5      insert into estudiante values (v_contador);
  6      v_contador := v_contador + 1;
  7     exit when v_contador > 50;
  8  end loop;
  9 end;
SQL> run
SQL> execute prueba
SQL> select * from estudiante;
```

Bucles: bucles condicionales



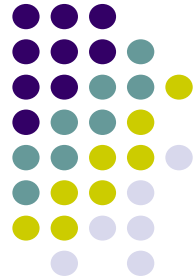
- *La condición se evalúa antes de entrar en el bucle*

```
[nombre_bucle] WHILE <condición> LOOP
                        sentencias; ...
                END LOOP;
```

  - **Ejemplo:**

```
DECLARE
   v_contador NUMBER :=1;
BEGIN
   WHILE v_contador <= 50 LOOP
       INSERT INTO estudiante
           VALUES (v_contador);
       v_contador:=v_contador+1;
   END LOOP;
END;
```

Bucles: numéricos (acotados)



- *Se ejecutan una sola vez por cada elemento de rango definido*

```
[nombre_bucle] FOR <indice>
        IN [REVERSE] <exp_n1>..<exp_n2> LOOP
        sentencias; ...
END LOOP;
```

- **índice**: variable de control empieza tomando el valor `exp_n1` y termina en `exp_n2,` en pasos de 1.
  Su declaración es implícita
- los extremos del intervalo (exp_n1 y exp_n2) pueden ser valores constantes o exp. numéricas
- REVERSE: el recorrido se hace al revés

- **Ejemplo:**

```
BEGIN
    FOR v_contador IN 1..50 LOOP
        INSERT INTO estudiante
        VALUES (v_contador);
    END LOOP;
END;
```

Bucles: definidos sobre cursores

- *Se ejecutan una vez por cada fila que se recupera del cursor.*
  - Pasos:
    - se abre el cursor
    - se realiza la lectura sobre un nombre de registro, y después se ejecutan las sentencias del bucle; hasta que no hay mas filas.
    - Se cierra el cursor

```
[nombre_bucle] FOR nombre_reg IN nombre_cursor
   LOOP sentencias
   END LOOP;
```

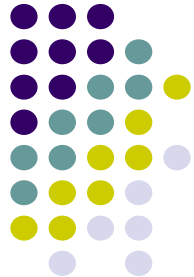
  - Se le pueden pasar parámetros a los cursores

Cursores en PL/SQL

- Cuando una consulta devuelve varias filas, es necesario declarar un **cursor** para procesar cada una de ellas individualmente.
- Un **cursor** es similar a una *variable de fichero* o *puntero de fichero*, que señala a una única fila del resultado obtenido.

- Se declaran en la parte declarativa del bloque PL/SQL y se manejan y controlan mediante tres sentencias:
- **OPEN**: Ejecuta la consulta asociada al cursor, se obtienen las filas y se sitúa el cursor en una posición antes de la primera fila: *fila en curso*.
- **FETCH**: Introduce los valores de la *fila en curso* en las variables del bloque declaradas en el bloque.
- **CLOSE**: Se libera el cursor.

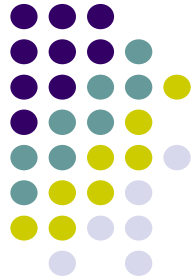
Cursores en PL/SQL (1)



- **Ejemplo**: Mostrar el NSS de los empleados cuyo salario es mayor que el salario de su supervisor.

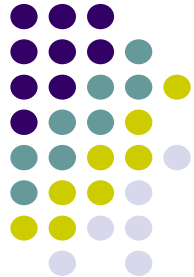
```
EJEMPLO2:
  salario_emp NUMBER;
  salario_superv_emp NUMBER;
  nss_emp VARCHAR2(9);
  nss_superv_emp VARCHAR2(9);
  CURSOR cursor_salario IS
  SELECT nss, salario, nss_superv FROM empleado;
```

Cursores en PL/SQL (2)



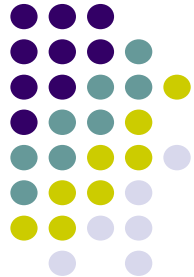
```
BEGIN
   OPEN cursor_salario;
   LOOP
      FETCH cursor_salario INTO nss_emp, salario_emp, nss_superv_emp;
      EXIT WHEN cursor_salario%NOTFOUND;
      IF nss_superv_emp is NOT NULL THEN
         SELECT salario INTO salario_superv_emp
         FROM empleado
         WHERE nss = nss_superv_emp;
         IF salario_emp > salario_superv_emp THEN
            dbms_output.put_line (nss_emp);
         END IF;
      END IF;
      END LOOP;
      IF cursor_salario%ISOPEN THEN
         CLOSE cursor_salario;
```

Cursores en PL/SQL (3)



```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
      dbms_outut.put_line ('Errores con el NSS ' ||
        nss_emp);
    IF cursor_salario%ISOPEN THEN
      CLOSE cursor_salario;
END;
```

Bucles: definidos sobre consultas



- Es el mismo esquema pero sin declaración de cursores

```
[nombre_bucle] FOR nombre_registro IN
                        <subquery>
               LOOP sentencias
               END LOOP;
```

- **Ejemplo:**

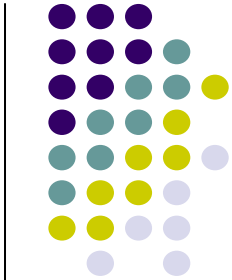
```
BEGIN
   FOR registro IN (SELECT DNI, nombre FROM cliente)
       LOOP
          INSERT into tabla_temporal
                 VALUES (registro.DNI, registro.nombre);
   END LOOP;
END;
```

Ejemplo de Procedimiento

- **Ejemplo**: Aumentar en un 10% el salario de aquellos empleados cuyo salario esté por debajo de la media. Recalcular la media e imprimirla si es mayor que 50.000€.

```
CREATE OR REPLACE PROCEDURE Aumento IS
salario_med NUMBER;
BEGIN
    SELECT avg(salario) INTO salario_med FROM empleado;
    UPDATE empleado
        SET salario = salario * 1,1
        WHERE salario < salario_med;
    SELECT avg(salario) INTO salario_med FROM empleado;
    IF salario_med > 50000 THEN
        dbms_output.put_line ('El salario medio es ' || salario_med);
    END IF;
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        dbms_output.put_line ('Error en la actualización');
    ROLLBACK;
END;
```

Excepciones



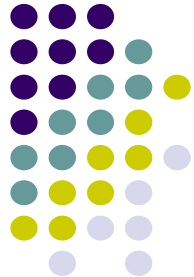
- Declaración de excepciones:
  - Se declaran por el usuario
  - Utilizan las funciones de error de SQL y PL/SQL

```
Nombre_excepción EXCEPTION;
```

  - Ejemplo:

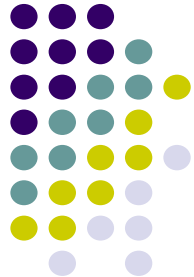
```
DECLARE
. . .
User_exception  EXCEPTION;
```

Tratamiento De Errores (1)



- *Se realiza mediante excepciones y gestión de excepciones*
  - Se tratan en tiempo de ejecución.
  - Cuando se produce un error, se cede el control a la zona `EXCEPTION`
  - Cada excepción sólo trata un error
  - Pueden ser
    - Definidas por el usuario
    - Definidas por el sistema

Tratamiento De Errores (2)



- La ejecución de las sentencias asociadas a un error se produce
  - **Automáticamente**: ORACLE detecta el error y pasa el control a la zona `EXCEPTION`, buscando (si existe) la excepción asociada al error
  - **Manualmente**: En la ejecución del proceso interesa ejecutar una excepción como si se hubiera producido un error
  - Sintaxis del control de errores:

```
WHEN nombre_exception THEN sentencias;
```

- Si no hay zona de excepción el proceso termina incorrectamente
- Las variables de control y recuperación de errores son: `SQLCODE` y `SQLERRM`

Tipos de Excepciones

- **Excepciones predefinidas:** no hace falta declararlas y están asociadas a errores comunes de ORACLE. Pueden asociarse a RAISE
- **Excepciones definidas por el usuario:**

  Hay que declararlas en la zona declarativa del bloque.

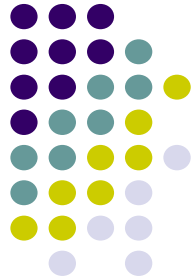
  Controlan los errores de ejecución de un programa, previstos por el programador. Pueden estar asociados a un error de datos.

  - Al generarse una excepción, el control pasa a la sección de excepciones y posteriormente se propaga al bloque de nivel superior. No se puede devolver el control a la zona ejecutable
  - Un mismo gestor puede utilizarse para mas de una excepción enumerándolas (separadas por un OR) en la cláusula WHEN
    - Ejemplo: `WHEN NO_DATA_FOUND OR TOO_MANY_ROWS THEN ...`

Algunas Excepciones Predefinidas

CURSOR_ALREADY_OPEN	Cursor ya abierto
DUP_VAL_ON_INDEX	Valor de un índice está duplicado
INVALID_CURSOR	Cursor no válido
INVALID_NUMBER	Número no válido
LOGIN_DENIED	Denegada conexión a Oracle
NO_DATA_FOUND	No recupera ninguna fila
NOT_LOGGED_ON	No conectado a Oracle
PROGRAM_ERROR	Problema interna
STORAGE_ERROR	Fuera de memoria o error de memoria
TIMEOUT_ON_RESOURCE	Exceso de recursos consumidos
TOO_MANY_ROWS	Mas de una fila
VALUE_ERROR	Valor incorrecto
ZERO_DIVIDE	División por 0
OTHERS	Cualquier otro error no especificado

Excepciones. Gestión de 'OTHERS'



Se ejecuta para todas las excepciones generadas.

- Debe ser el último del bloque
- Se definirá en el nivel superior de programa para que no quede ningún error sin procesar
- Se apoya en las funciones predeterminadas SQLERRM y SQLCODE para identificar los errores (mensaje y código del error, respectivamente).
- SQLCODE: Devuelve el número del error producido.
  - Sólo es válido con los errores de ORACLE
  - Sólo se habilita en la zona de excepciones
  - Ejemplo:

```
DECLARE
   ERROR number;
BEGIN
EXCEPTION
   WHEN OTHERS THEN Error:=SQLCODE;
END;
```

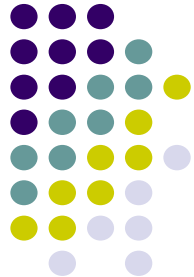
Excepciones

- SQLERRM: mensaje de error asociado al error producido
  - Solo es válido con errores de Oracle
  - La longitud del mensaje es de 512 caracteres. Si los mensajes son menores se debe usar la funcion SUBSTR para evitar el error
- PRAGMA EXCEPTION_INIT: directivas del compilador que sirven como instrucciones al PL/SQL
  - EXCEPTION_INIT es un PRAGMA que asocia una excepción nominada a un error Oracle.

```
PRAGMA EXCEPTION_INIT
        (nombre_exception,numero_error_oracle);
```

  - nombre_exception: definida antes del PRAGMA como excepción de usuario
  - numero_error_oracle El número de error que se quiere controlar (SQLCODE)

Tratamiento De Errores

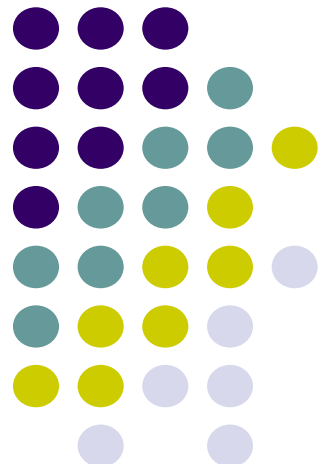


- RAISE_APLICATION_ERROR:
  Se puede usar para obtener mensajes de error personalizados

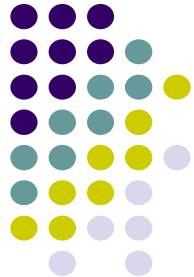
```
RAISE_APPLICATION_ERROR(numero_error, mensaje_error);
```

  donde:
  - *numero_error*: es un parámetro comprendido entre -20000 y -20999
  - *mensaje_error*: es el texto asociado al mensaje (menor de 512 cars.)

Tablas Mutantes

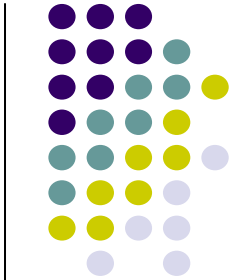


Qué son



- **Tablas que están siendo modificadas por una operación DML (INSERT, DELETE, UPDATE):**
  - En un disparador, la tabla sobre la que está definido el mismo
  - 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 puede existir:**
  - **lecturas o modificaciones** de tablas mutantes
  - **cambio de clave primaria, claves ajenas o claves alternativas** de las tablas que restringen (el resto de las columnas sí se pueden cambiar)
  - EXCEPCIÓN: no se dan las tablas mutantes en los disparadores con nivel de fila BEFORE INSERT
- A nivel de **SENTENCIA** no existen problemas de tablas mutantes
  - EXCEPCIÓN: si el disparador se activa como consecuencia de un BORRADO EN CASCADA (problemas de tablas mutantes)

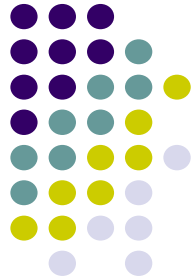
En los disparadores (2)

TIPO DE DISPARADOR	ERROR DE TABLA MUTANTE
BEFORE INSERT ROW	NO[1]
AFTER INSERT ROW	SI
BEFORE INSERT STATEMENT	NO
AFTER INSERT STATEMENT	NO
BEFORE DELETE ROW	SI
AFTER DELETE ROW	SI
BEFORE DELETE STATEMENT	NO[2]
AFTER DELETE STATEMENT	NO[2]
BEFORE UPDATE ROW	SI
AFTER UPDATE ROW	SI
BEFORE UPDATE STATEMENT	NO
AFTER UPDATE STATEMENT	NO

[1] Siempre que la inserción que provoque la activación del disparador sea una inserción simple (se inserte una única fila).

[2] Siempre que el disparador no se active como consecuencia de un borrado en cascada. En ese caso, aparecerá también un error de tabla mutante.

Ejemplo 1



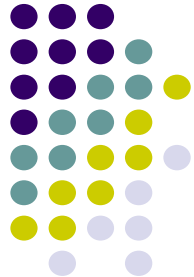
- **“Una zona tiene uno o varios departamentos y un departamento trabaja en una o ninguna zona”**

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

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)



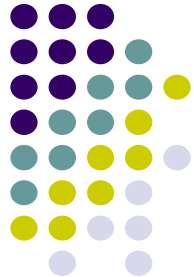
- EJEMPLO 1:

```
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:

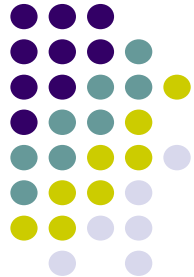
```
INSERT INTO Zona VALUES (1, 'CENTRO');
```

Ejemplo 1 (2)



- EJEMPLO 1. Comentarios:
  - La tabla *departamento* referencia a la tabla *zona* (FK).
  - Cada vez que se inserta un nuevo dato en la tabla *departamento*, se 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
  AFTER INSERT ON Departamento FOR EACH ROW
DECLARE
  Var Departamento%ROWTYPE;
BEGIN
  UPDATE Zona SET Nom_Zona='U' WHERE Cod_Zona=:new.Cod_Zona;
END Disparador2;
/
```

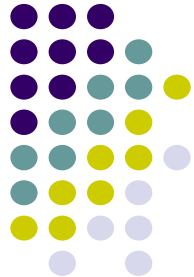
Operación:

```
INSERT INTO Departamento VALUES
(Secuencia_Departamento.NEXTVAL, 20000000, 1);
```

Ejemplo 2 (2)

- EJEMPLO 2. Comentario:
  - No existe error de tabla mutante: la tabla *departamento* referencia a la tabla *zona*, por lo que no se pueden modificar sus claves, pero sí se pueden modificar las demas columnas.

Ejemplo 3 (1)

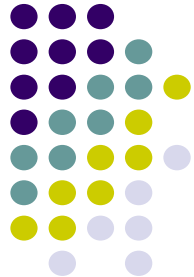


- EJEMPLO 3:
  - Creación de una tabla independiente a las anteriores:

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

CREATE TABLE Mensaje_Departamento (
    Cod_Mensaje NUMBER(6) CONSTRAINT pk_error PRIMARY KEY,
    Cod_Dep NUMBER(6) NOT NULL,
    Tipo VARCHAR2(255) NOT NULL,
    Fecha DATE NOT NULL
);
```

Ejemplo 3 (2)



- EJEMPLO 3. Disparador:

```
CREATE OR REPLACE TRIGGER Disparador3
AFTER INSERT ON Departamento
FOR EACH ROW
  BEGIN
INSERT INTO Mensaje_Departamento VALUES
  (Secuencia_Mensaje.NEXTVAL,  :new.Departamento,  'Presupuesto elevado',  SYSDATE);
  END Disparador3;
/
```

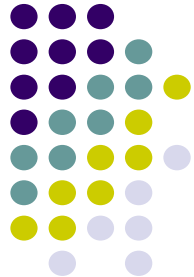
Operación:

```
INSERT INTO Departamento VALUES (Secuencia_Departamento.NEXTVAL, 70000, 1);
```

Ejemplo 3 (3)

- EJEMPLO 3. Comentarios:
  - No existe error de tabla mutante: la tabla *mensaje_departamento* es independiente de la tabla *departamento*.

Ejemplo 4 (1)



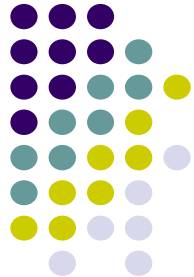
- EJEMPLO 4:

```
CREATE OR REPLACE TRIGGER Disparador4
      BEFORE DELETE ON Departamento FOR EACH ROW
  DECLARE
   Var Zona%ROWTYPE;
  BEGIN
   SELECT * INTO Var FROM Zona WHERE
Cod_Zona=:old.Cod_Zona;
END Disparador4;
 /
```

Operación1: DELETE FROM Departamento WHERE Cod_Zona=1;

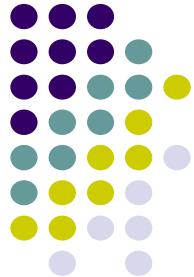
Operación2: DELETE FROM Zona WHERE Cod_Zona=1;

Ejemplo 4 (2)



- EJEMPLO 4. 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:** Da error de tabla mutante, ya que, al borrar en la tabla *zona* (tabla mutante), se borran todas las tuplas de la tabla *departamento* que referencian a la *zona* borrada. Esto activa el disparador4 de *departamento*, que consulta la tabla *zona*, que en este caso sí esta mutando.

Ejemplo 5 (1)



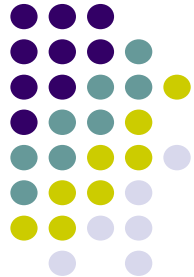
- EJEMPLO 5:

```
CREATE OR REPLACE TRIGGER Disparador5
                AFTER DELETE ON Departamento
DECLARE
  Var Zona%ROWTYPE;
BEGIN
    SELECT * INTO Var FROM Zona WHERE Cod_Zona=1;
END Disparador5;
/
```

Operación 1: DELETE FROM Zona WHERE Cod_Zona = 1;

Operación 2: DELETE FROM Departamento WHERE Cod_Zona = 1;

Ejemplo 5 (2)



- EJEMPLO 5. Comentarios:
  - Operación 1: Error de tabla mutante.
    - La excepción en los disparadores a **nivel de sentencia** se encuentra en los **borrados en cascada**, al **leer o modificar una tabla mutante.**
    - Al borrar de la tabla *zona*, se desencadena un borrado en cascada en la tabla *departamento* (ambas tablas mutantes), y al mismo tiempo intenta leer de la tabla *zona*.
  - Operación 2: NO hay error.
    - Al borrar de la tabla *departamento* no se desencadena ningún borrado en cascada (sólo borra de la tabla departamento)