



PL/SQL

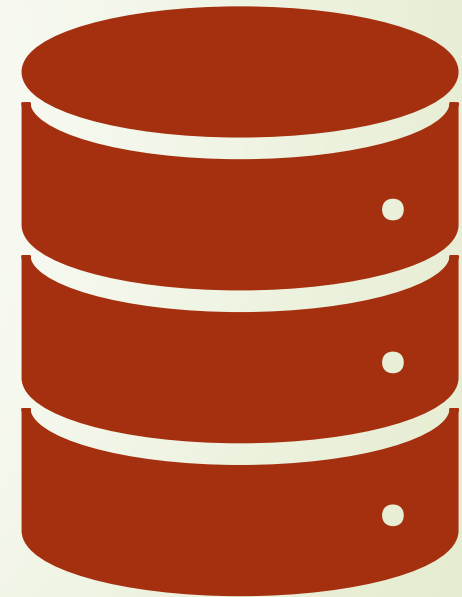
Francisco Javier González Brime



ÍNDICE

1. Fundamentos de programación en bases de datos
2. Salida y entrada de datos. El paquete DBMS_OUTPUT
3. Asignación de valores a variables a través de SELECT INTO
4. Estructura de control PL/SQL
5. Manejo de cursores
6. Gestión de excepciones
7. Diseño y uso de subprogramas

1. Fundamentos de programación en bases de datos



Lenguaje PL/SQL



- Lenguaje procedimental, dando a SQL nuevas posibilidades manipulando datos de la base de datos.
- Combina comandos SQL, y tratamiento de subprogramas.
- Mejora de rendimiento al reutilizar código, estar compilado y en memoria principal.
- Reutilización. Si cambia el código del subprograma, se hace una sola vez y no en cada aplicación, eliminando errores de codificación.
- Mejora de la seguridad.

Bloques de un programa PL/SQL

DECLARE -- (opcional) Se declaran variables, constantes, cursores y excepciones
v_mensaje VARCHAR2(50) := '¡Hola, PL/SQL!';

BEGIN -- (obligatorio) El bloque más importante
DBMS_OUTPUT.PUT_LINE(v_mensaje);

EXCEPTION -- (opcional) Tratamiento de errores
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Ocurrió un error.');

END; -- (obligatorio)

/ -- Especial, si se ejecuta desde un archivo en SQL Plus

Declaración de variables (bloque DECLARE)

nombre_variable [CONSTANT] tipo_dato [NOT NULL] [{DEFAULT | :=} valor]

dni VARCHAR2(10);

DECLARACIÓN

dni VARCHAR2(10) := '12345678A';

DECLARACIÓN + INICIALIZACIÓN

dni VARCHAR2(10) NOT NULL := '12345678A';

NOT NULL -> INICIALIZAR

dni CONSTANT VARCHAR2(10) := '12345678A';

CONSTANT -> INICIALIZAR

dni VARCHAR2(10) DEFAULT '12345678A';

VALOR POR DEFECTO

¡Sólo una variable por línea!

Tipos de datos en PL/SQL

Tipos de datos numéricos	BINARY_INTEGER , DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INT, INTEGER, NATURAL, NATURALN, NUMBER, NUMERIC, PLS_INTEGER , POSITIVE, POSITIVEN, REAL, SIGNTYPE, SMALLINT
Tipos de datos cadenas	CHAR, CHARACTER, LONG, LONG RAW, NCHAR, NVARCHAR2, RAW, ROWID , STRINGM UROWID, VARCHAR, VARCHAR2
Tipos de datos lógicos	BOOLEAN
Tipos de datos fecha	DATE, INTERVAL DAY TO SECOND, INTERVAL YEAR TO MONTH, TIMESTAMP, TIMESTAMP WITH LOCAL TIME ZONE, TIMESTAMP WITH TIME ZONE
Tipos de datos compuestos	RECORD , TABLE , VARRAY
Tipos de datos grandes	BFILE, BLOB, CLOB, NCLOB

Primer programa PL/SQL

Introducir Variable de Sustitución

Introduzca un valor para dni:

Aceptar Cancelar

Variable de sustitución (número -> sin comillas -> &dninumber)

```
1 DECLARE
2
3     vNumlibros NUMBER;
4     vDNIconсульта VARCHAR2(10) := '&dni';
5
6 BEGIN
7
8     SELECT COUNT(*) INTO vNumlibros
9     FROM LIBRO
10    WHERE DNI = vDNIconсульта;
11
12    DBMS_OUTPUT.PUT_LINE('El escritor con DNI ' || vDNIconсульта || ' tiene ' || vNumlibros || ' libros.');
```

Diagram illustrating the PL/SQL code execution flow:

- Line 4: Variable substitution (&dni) is used to assign a value to vDNIconсульта.
- Line 8: The SELECT statement uses vNumlibros to store the count of books.
- Line 12: The output statement concatenates the DNI value, the count, and the number of books.

Para concatenar la salida

Salida PL/SQL

El escritor con DNI 56789012E tiene 2 libros.

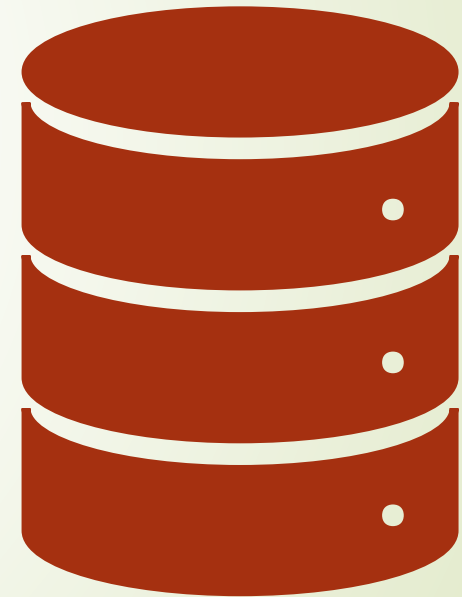
SET SERVEROUTPUT ON;

Salida SQL

```
SELECT COUNT(*)
FROM LIBRO
WHERE DNI = '&dni';
```

	COUNT(*)
1	2

2. Salida y entrada de datos. El paquete DBMS_OUTPUT



Salida PL/SQL

```
1 DECLARE
2
3     vNumlibros NUMBER;
4     vDNIconсульта VARCHAR2(10) := '&dni';
5
6 BEGIN
7
8     SELECT COUNT(*) INTO vNumlibros
9     FROM LIBRO
10    WHERE DNI = vDNIconсульта;
11
12    DBMS_OUTPUT.PUT_LINE('El escritor con DNI ' || vDNIconсульта || ' tiene ' || vNumlibros || ' libros.');
```

13

14 END;

Paquete Procedimiento

DBMS_OUTPUT.PUT('El escritor con DNI ' || vDNIconсульта); .PUT: Sin salto de línea
DBMS_OUTPUT.PUT_LINE(' tiene ' || vNumlibros || ' libros.');

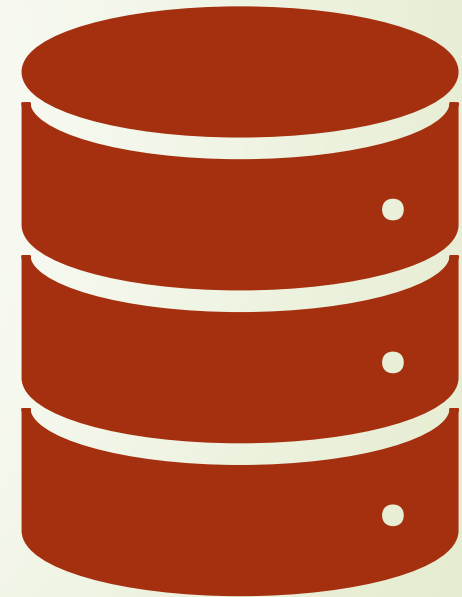
.PUT_LINE: Con salto de línea

El escritor con DNI 56789012E tiene 2 libros.

DBMS_OUTPUT.PUT_LINE('El escritor con DNI ' || vDNIconсульта || ' tiene ' || vNumlibros || ' libros.');

El escritor con DNI 56789012E tiene 2 libros.

3. Asignación de valores a variables a través de SELECT INTO



De consulta a variable: Consulta devuelve un ÚNICO valor

SELECT

Columna | Función de agregación

INTO

VARIABLE

Esta variable es de tipo simple y contendrá un único valor. Debe ser del mismo tipo que la columna o que pueda almacenar el valor que devuelve la función de agregación

```
1 DECLARE
2
3     vNumlibros NUMBER;
4     vDNIconсульта VARCHAR2(10);
5
6 BEGIN
7
8     vDNIconсульта := '56789012E';
9     SELECT COUNT(*) INTO vNumlibros
10    FROM LIBRO
11   WHERE DNI = vDNIconсульта;
12
13    DBMS_OUTPUT.PUT('El escritor con DNI ' || vDNIconсульта);
14    DBMS_OUTPUT.PUT_LINE(' tiene ' || vNumlibros || ' libros.');
```

```
15
16 END;
```

El atributo %TYPE

- ✓ Obtiene el mismo tipo de datos de la columna de una tabla

```
1 DECLARE
2
3     vNumlibros NUMBER;
4     vDNIconсульта LIBRO.DNI%TYPE;
5
6 BEGIN
7
8     vDNIconсульта := '&dni';
9     SELECT COUNT(*) INTO vNumlibros
10    FROM LIBRO
11   WHERE DNI = vDNIconсульта;
12
13    DBMS_OUTPUT.PUT_LINE('El escritor con DNI ' || vDNIconсульта || ' tiene ' || vNumlibros || ' libros.');

Atributo

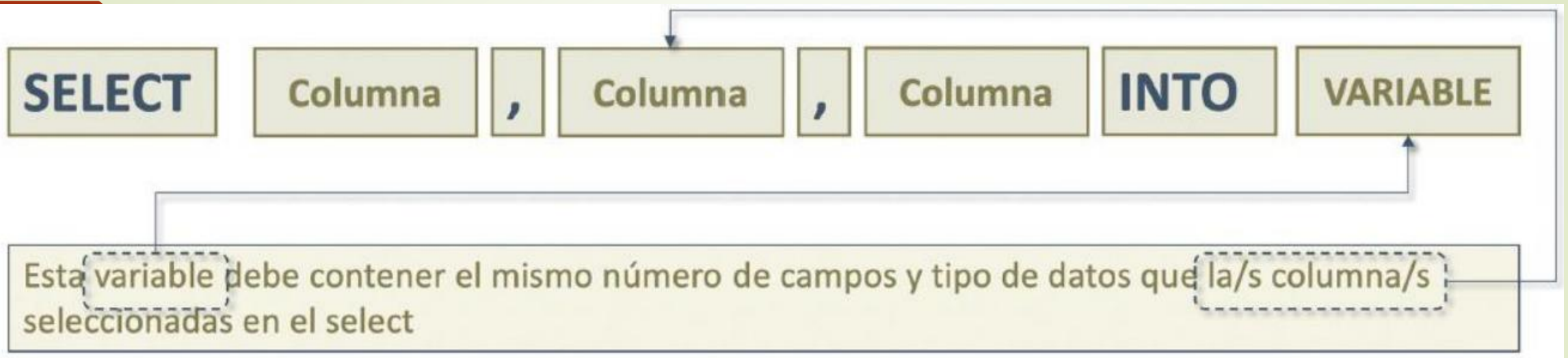


Tabla.columna


```

LIBRO.DNI%TYPE ≈ VARCHAR2(10)

De consulta a variable: Consulta devuelve ALGUNAS columnas



```
1 DECLARE
2
3     vNombre VARCHAR2(20);
4     vPrecio LIBRO.precio%TYPE;
5
6 BEGIN
7
8     SELECT nombre_1, precio INTO vNombre, vPrecio
9     FROM LIBRO
10    WHERE ISBN = 'ISBN1';
11
12    DBMS_OUTPUT.PUT('El libro con ISBN ISBN1');
13    DBMS_OUTPUT.PUT_LINE(' es ' || vNombre || ' y cuesta ' || vPrecio || ' €.');
14
15 END;
```

Un valor por columna

De consulta a variable: Consulta devuelve ALGUNAS columnas v2

```
TYPE nombreRegistro IS RECORD(  
    Campo1 tipo1,  
    Campo2 tipo2,  
    .... );
```

```
1 DECLARE
```

```
2
```

```
3     TYPE rlibro_completo IS RECORD(  
4         vNombre VARCHAR2(20),  
5         vPrecio LIBRO.precio%TYPE  
6     );  
7     vLibro rlibro_completo;
```

```
8
```

```
9 BEGIN
```

```
10
```

```
11     SELECT nombre_l, precio INTO vLibro  
12     FROM LIBRO  
13     WHERE ISBN = 'ISBN1';
```

```
14
```

```
15     DBMS_OUTPUT.PUT('El libro con ISBN ISBN1');
```

```
16     DBMS_OUTPUT.PUT_LINE(' es ' || vLibro.vNombre || ' y cuesta ' || vLibro.vPrecio || ' €.');
```

```
17
```

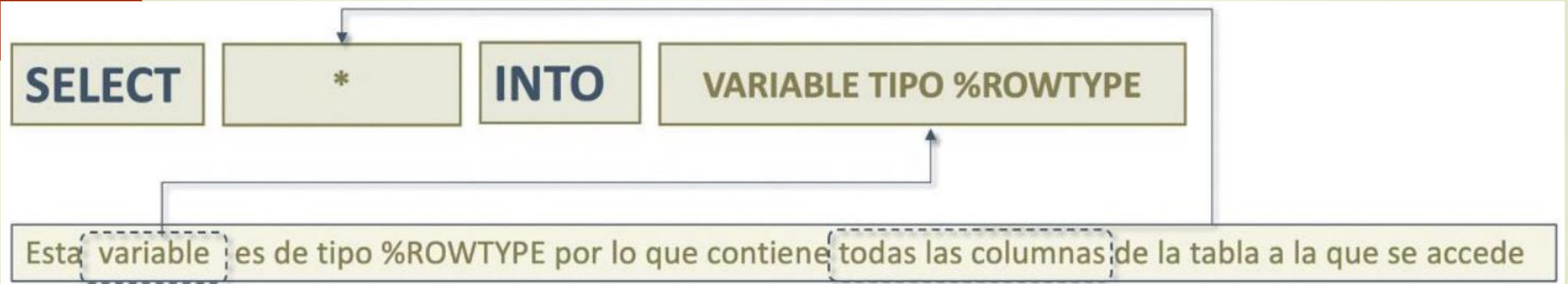
```
18 END;
```

Definir registro

Declaración variable de tipo rlibro_completo

Acceso a datos con variable.campo (notación punto)

De consulta a variable: Consulta devuelve UNA fila completa



```
1 DECLARE
2
3     vLibro LIBRO%ROWTYPE;
4
5 BEGIN
6
7     SELECT * INTO vLibro
8     FROM LIBRO
9     WHERE ISBN = 'ISBN1';
10
11     DBMS_OUTPUT.PUT('El libro con ISBN ISBN1');
12     DBMS_OUTPUT.PUT_LINE(' es ' || vLibro.nombre_1 || ' y cuesta ' || vLibro.precio || ' €.');
13
14 END;
```

Atributo %ROWTYPE

Tabla

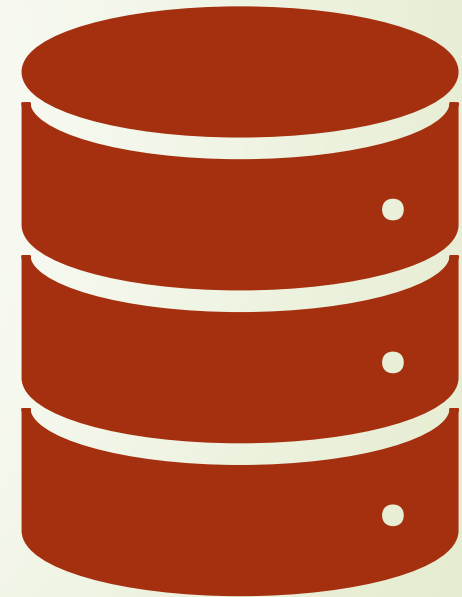
1 fila en vLibro

De consulta a variable: Consulta devuelve VARIAS filas completas

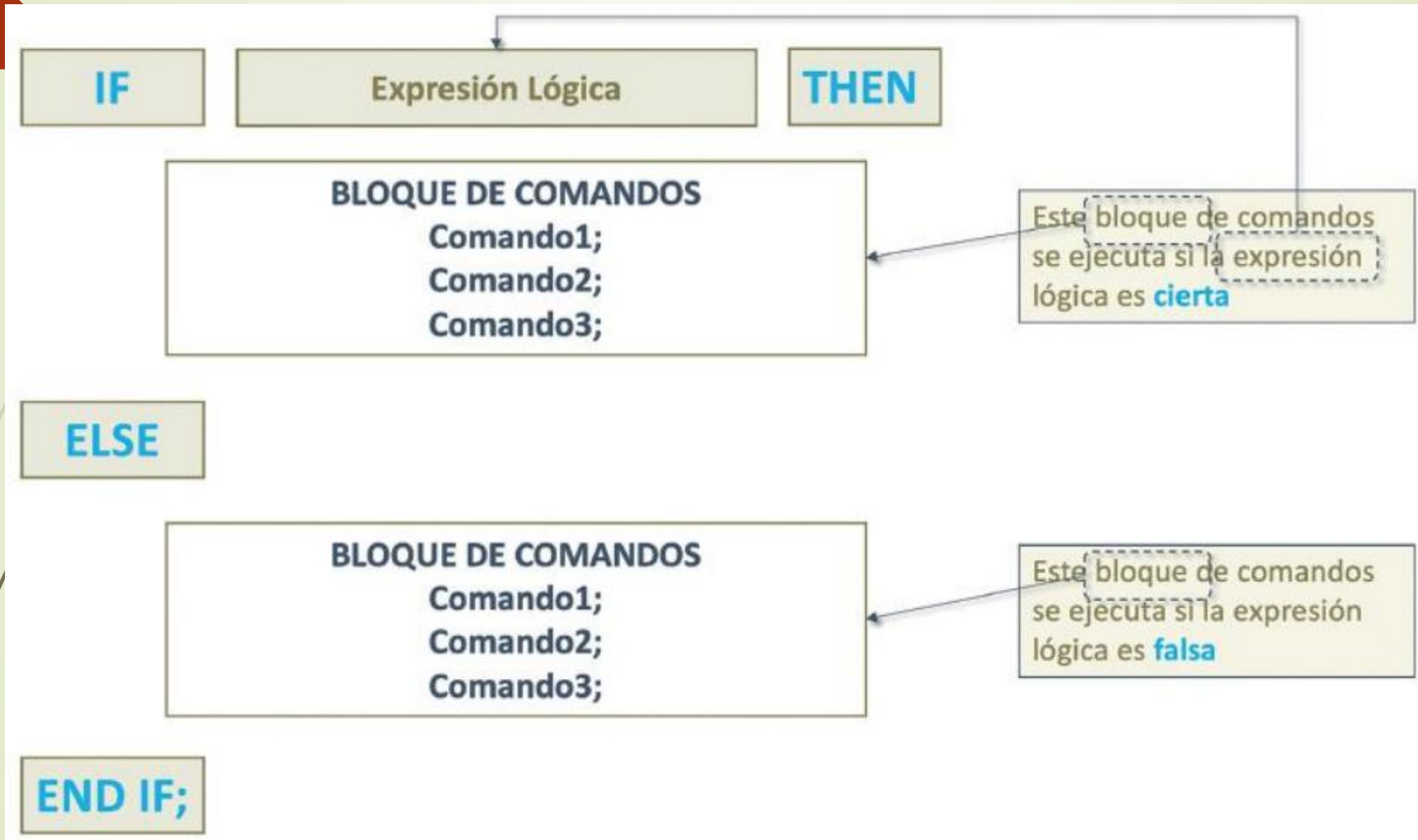


Uso optimizado de cursores, lo veremos en el
punto 5

4. Estructuras de control PL/SQL



Estructura selectiva IF



Estructura selectiva IF

¡El orden importa! Lectura hacia abajo

L
E
C
T
U
R
A

```
1 DECLARE
2
3     vNumLibros NUMBER;
4     vNumBarato NUMBER;
5     vNumCaro NUMBER;
6     vPrecio NUMBER := &valor;
7
8 BEGIN
9
10    SELECT COUNT(*) INTO vNumLibros FROM LIBRO;
11    SELECT COUNT(*) INTO vNumCaro FROM LIBRO
12    WHERE precio > vPrecio;
13    vNumBarato := vNumLibros - vNumCaro;
14
15    IF vNumBarato>5 THEN
16        DBMS_OUTPUT.PUT_LINE('La mayoría son baratos. ' || vNumBarato);
17    ELSIF vNumBarato>3 THEN
18        DBMS_OUTPUT.PUT_LINE('Hay algún libro barato. ' || vNumBarato);
19    ELSE
20        DBMS_OUTPUT.PUT_LINE('La mayoría son caros. ' || vNumBarato);
21    END IF;
22
23 END;
```

IF anidado

Por defecto

1 bloque cierto

n-1 alternativas

Estructura selectiva múltiple CASE

Expresión es obligatorio únicamente con valores

```
CASE [expresion]
  WHEN valor1|condicion1 THEN
    comandos
  WHEN valor2|condicion2 THEN
    comandos
  ...
  WHEN condicionN THEN
    comandos
  ELSE
    comandos
END CASE;
```


Estructura selectiva múltiple CASE

```
1 DECLARE
2
3     vNumLibros NUMBER;
4     vNumBarato NUMBER;
5     vNumCaro NUMBER;
6     vPrecio NUMBER := &valor;
7
8 BEGIN
9
10    SELECT COUNT(*) INTO vNumLibros FROM LIBRO;
11    SELECT COUNT(*) INTO vNumCaro FROM LIBRO
12    WHERE precio > vPrecio;
13    vNumBarato := vNumLibros - vNumCaro;
14
15    CASE
16        WHEN vNumBarato > 5 THEN
17            DBMS_OUTPUT.PUT_LINE('La mayoría son baratos. ' || vNumBarato);
18        WHEN vNumBarato > 3 THEN
19            DBMS_OUTPUT.PUT_LINE('Hay algún libro barato. ' || vNumBarato);
20        ELSE
21            DBMS_OUTPUT.PUT_LINE('La mayoría son caros. ' || vNumBarato);
22    END CASE;
23
24 END;
```

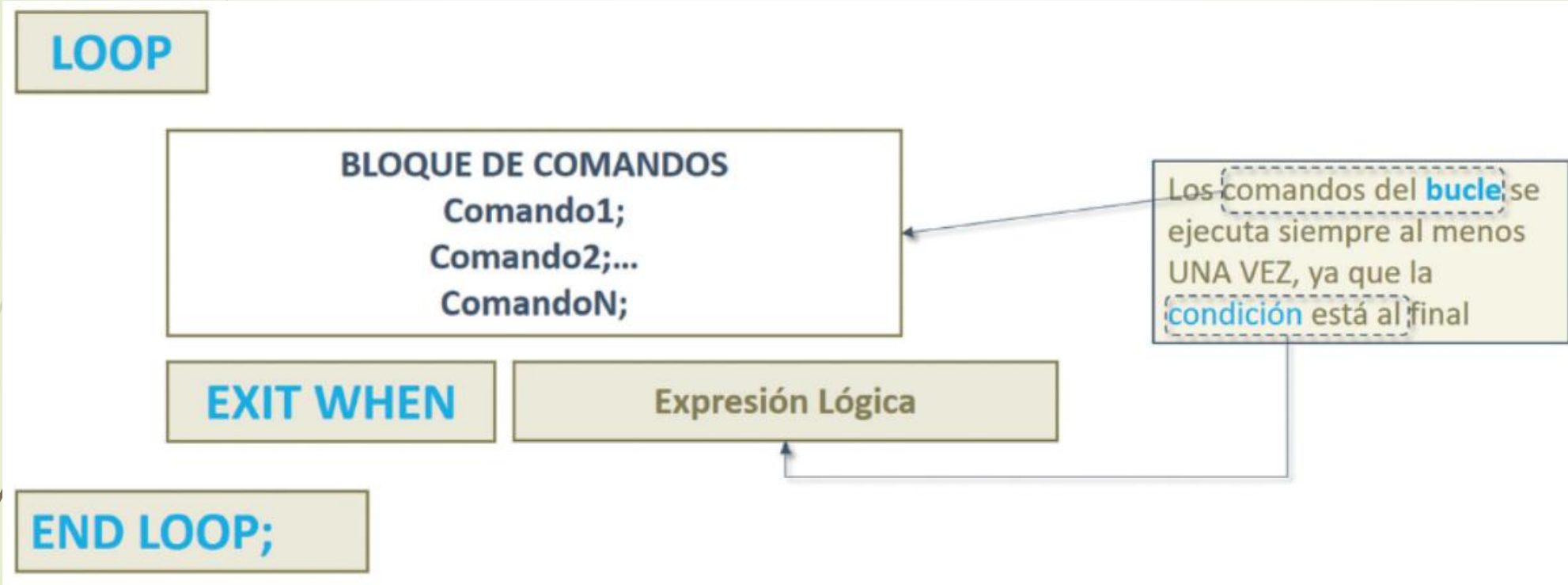
Con condiciones -> NO expresión

1 bloque cierto

n-1 alternativas

Estructura repetitiva **LOOP**

- ✓ nº iteraciones conocido
- ✓ Se ejecuta al menos una vez



Estructura repetitiva LOOP

```
1 DECLARE
2
3     vContador NUMBER := 1;
4     vTexto VARCHAR2(100);
5
6 BEGIN
7
8     LOOP
9
10        vTexto:='Este es el registro número ' || vContador;
11        INSERT INTO TABLAREGISTRO VALUES(vContador, vTexto);
12        vContador := vContador+1;
13        EXIT WHEN vContador>100;
14
15    END LOOP;
16
17 END;
```

Nombre	¿Nulo?	Tipo
ID	NOT NULL	NUMBER
TEXTO		VARCHAR2(100)

TABLAREGISTRO

Estructura repetitiva **WHILE LOOP**

- ✓ nº iteraciones desconocido.
- ✓ Se ejecuta ninguna o más de una



Estructura repetitiva **WHILE LOOP**

```
1 DECLARE
2
3     vContador NUMBER := 101;
4     vTexto VARCHAR2(100);
5
6 BEGIN
7
8     WHILE vContador <= 200 LOOP
9
10        vTexto := 'Este es el registro número ' || vContador;
11        INSERT INTO TABLAREGISTRO VALUES(vContador, vTexto);
12        vContador := vContador+1;
13
14    END LOOP;
15
16 END;
```

Nombre	¿Nulo?	Tipo
ID	NOT NULL	NUMBER
TEXTTO		VARCHAR2(100)

TABLAREGISTRO

Estructura repetitiva **FOR IN LOOP**

- ✓ nº iteraciones conocido.
- ✓ Se ejecuta $\text{ValorFin} - \text{ValorIni} + 1$ veces

Esta variable irá desde el valor inicial hasta el valor final.
El bloque de comandos se ejecutará tantas veces como el valor $\text{ValorFin} - \text{ValorIni} + 1$

FOR

VARIABLE contador

IN

ValorIni

..

ValorFin

LOOP

BLOQUE DE COMANDOS

Comando1;

Comando2;

.....

ComandoN;

END LOOP;

Si ValorIni es mayor que ValorFin se hace necesario usar **IN RESERVE**

Estructura repetitiva **FOR IN LOOP**

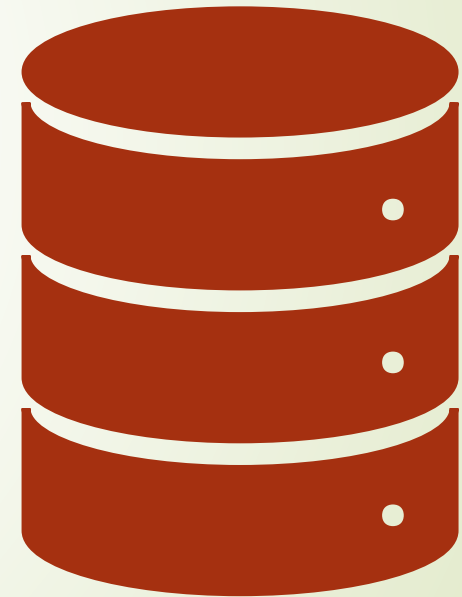
- ✓ El más simple, sin necesidad de declarar contador

```
1 DECLARE
2
3     vTexto VARCHAR2(100);
4
5 BEGIN
6
7     FOR vContador IN 201 .. 300 LOOP
8
9         vTexto := 'Este es el registro número ' || vContador;
10        INSERT INTO TABLAREGISTRO VALUES(vContador, vTexto);
11
12    END LOOP;
13
14 END;
```

Nombre	¿Nulo?	Tipo
ID	NOT NULL	NUMBER
TEXT0		VARCHAR2(100)

TABLAREGISTRO

5. Manejo de cursores



Cursores

Uso: Consultas SELECT que devuelven más de un resultado (varias filas), para acceder a cada fila.

► Procesamiento:

1. Declarar el cursor (DECLARE).

No hay INTO variable en la

```
CURSOR nombre IS sentenciaSELECT;
```

-
2. Abrir el cursor (BEGIN).

Reservar memoria, ejecutar SELECT y apuntar a la primera fila

```
OPEN cursor;
```

3. Procesar el cursor (BEGIN).

Recorrer el cursor registro a registro hasta el final, cargándolo en una(s) variable(s)

```
FETCH cursor INTO listaDeVariables;
```

4. Cerrar el cursor (BEGIN).

Liberar memoria

```
CLOSE cursor;
```

Cursores: Atributos

➡ **%ISOPEN**

Devuelve verdadero si el cursor ya está abierto. (Uso: bucle IF)

➡ **%NOTFOUND**

Devuelve verdadero si la última instrucción FETCH (procesamiento) no devolvió ningún valor. (Salir de un bucle LOOP)

➡ **%FOUND**

Devuelve verdadero si el último FETCH (procesamiento) devolvió una fila.

(Uso: Para continuar en un bucle WHILE e implica hacer al menos dos lecturas (FETCH), antes de entrar al WHILE y otra al final del bucle si tiene registros)

➡ **%ROWCOUNT**

Indica el número de filas que se han recorrido en el cursor (inicialmente vale cero). Indica cuántos FETCH se han aplicado sobre el cursor.

Cursores con LOOP

1- Declaración

CURSOR nombre **IS** sentenciaSELECT;

2- Apertura

3- Procesamiento

FETCH cursor **INTO** listaDeVariables;

4- Cierre

```
1 DECLARE
2
3     vDNI AUTOR.DNI%TYPE;
4     vNumLibros NUMBER := 0;
5
6     CURSOR cursorAutores IS
7         SELECT DNI, count(*) AS numlibros
8         FROM AUTOR JOIN LIBRO USING(DNI)
9         GROUP BY DNI
10        ORDER BY numlibros DESC;
11
12 BEGIN
13
14     OPEN cursorAutores;
15
16     LOOP
17
18         FETCH cursorAutores INTO vDNI, vNumLibros;
19         EXIT WHEN cursorAutores%NOTFOUND;
20         DBMS_OUTPUT.PUT_LINE(vDNI || ': ' || vNumLibros);
21
22     END LOOP;
23
24     CLOSE cursorAutores;
25
26 END;
```

Cursores con LOOP

1- Declaración

CURSOR nombre **IS** sentenciaSELECT;

```
1 DECLARE
2
3 CURSOR cursorAutores IS
4     SELECT DNI, count(*) AS numlibros
5     FROM AUTOR JOIN LIBRO USING(DNI)
6     GROUP BY DNI
7     ORDER BY numlibros DESC;
```

cursor_r cursorAutores%ROWTYPE;

Registro cursor_r,
%ROWTYPE del cursor

2- Apertura

```
11 BEGIN
12
13 OPEN cursorAutores;
```

3- Procesamiento

LOOP

FETCH cursorAutores INTO cursor_r;

EXIT WHEN cursorAutores%NOTFOUND;

DBMS_OUTPUT.PUT_LINE(cursor_r.DNI || ': ' || cursor_r.numLibros);

Acceso a
datos

4- Cierre

CLOSE cursorAutores;

```
24
25 END;
```

FETCH cursor **INTO** listaDeVariables;

Cursores con WHILE

Posibilidad de registro
registroAutor

- 1- DECLARACIÓN
- 2- APERTURA
- 3- PROCESAMIENTO
- 3.1- PROCESAMIENTO
- 4- CIERRE

n + 1 lecturas

```
1 DECLARE
2
3     TYPE registroAutor IS RECORD(
4         vDNI AUTOR.DNI%TYPE,
5         vNumLibros NUMBER := 0
6     );
7     librosAutor registroAutor;
8
9     CURSOR cursorAutores IS
10        SELECT DNI, count(*) AS numLibros
11        FROM AUTOR JOIN LIBRO USING(DNI)
12        GROUP BY DNI
13        ORDER BY numLibros DESC;
14
15 BEGIN
16
17     OPEN cursorAutores;
18     FETCH cursorAutores INTO librosAutor;
19
20     WHILE cursorAutores%FOUND LOOP
21
22         DBMS_OUTPUT.PUT_LINE(librosAutor.vDNI || ': ' || librosAutor.vNumLibros);
23         FETCH cursorAutores INTO librosAutor;
24
25     END LOOP;
26
27     CLOSE cursorAutores;
28
29 END;
```

Cursores con FOR

- ✓ El más simple de utilizar
- ✓ Sólo hay declaración
- ✓ librosAutor es una variable temporal del bucle FOR con los atributos de la consulta SELECT (cursor)

```
1 DECLARE
2
3     CURSOR cursorAutores IS
4         SELECT DNI, COUNT(*) AS numLibros
5         FROM AUTOR JOIN LIBRO USING(DNI)
6         GROUP BY DNI
7         ORDER BY numLibros DESC;
8
9 BEGIN
10
11     FOR librosAutor IN cursorAutores LOOP
12
13         DBMS_OUTPUT.PUT_LINE(librosAutor.DNI || ': ' || librosAutor.numLibros);
14
15     END LOOP;
16
17 END;
```

Apertura y cierre de
cursor implícito

FETCH implícito

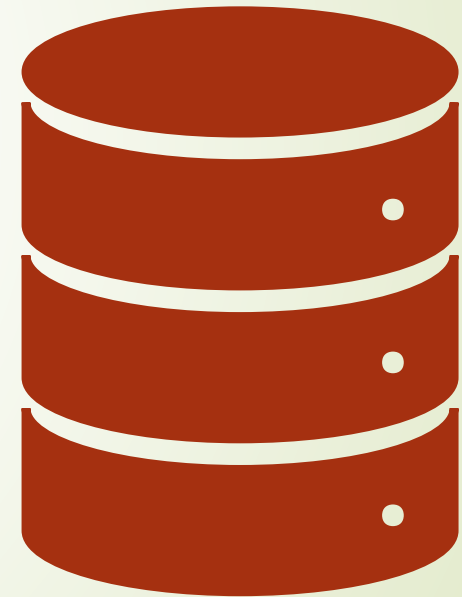
1- DECLARACIÓN

2- APERTURA

3- PROCESAMIENTO

4- CIERRE

6. Gestión de excepciones



Excepciones

EXCEPTION

WHEN excepción1 [**OR** excepción2 ...] **THEN**
instrucciones que se ejecutan si suceden esas excepciones
[**WHEN** excepción3 [**OR**...] **THEN**
instrucciones que se ejecutan si suceden esas excepciones]
[**WHEN OTHERS THEN**
instrucciones que se ejecutan si suceden otras excepciones]

- Muy importante en programación
- Acciones en una base de datos sujeto a numerosos errores
- Errores tienen un número único para cada error
- Tomar alternativas a que el programa aborte
- Aísla la gestión de errores del programa en su conjunto
- Bloque **EXCEPTION** para tomar acciones a un error

```
1 DECLARE -- (opcional) Se declaran las variables, constantes, cursores y excepciones
2
3 BEGIN MiBloque -- (obligatorio) El bloque más importante, con comandos SQL y sentencias de control PL
4
5 EXCEPTION -- (opcional) Definir que hacer en caso de error en BEGIN
6
7 END MiBloque; -- (obligatorio)
8 / -- Especial, si se ejecuta desde un archivo en sqlplus
```

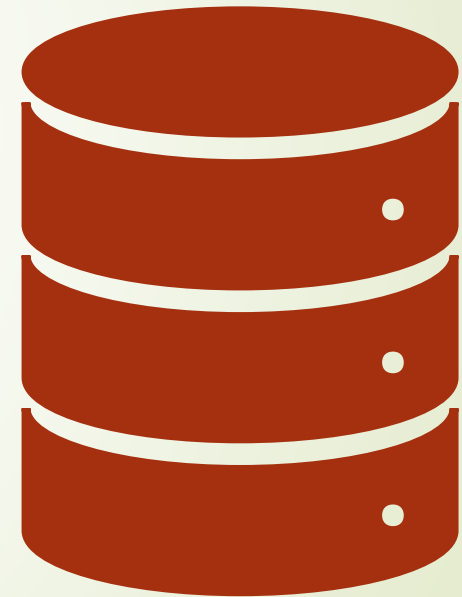
Excepciones predefinidas en Oracle

Nombre de excepción	Número de error	Ocurre cuando..
ACCESS_INTO_NULL	ORA-06530	Se intentan asignar valores a un objeto que no se había inicializado
CASE_NOT_FOUND	ORA-06592	Ninguna opción WHEN dentro de la instrucción CASE captura el valor, y no hay instrucción ELSE
COLLECTION_IS_NULL	ORA-06531	Se intenta utilizar un <i>varray</i> o una tabla anidada que no estaba inicializada
CURSOR_ALREADY_OPEN	ORA-06511	Se intenta abrir un cursor que ya se había abierto
DUP_VAL_ON_INDEX	ORA-00001	Se intentó añadir una fila que provoca que un índice único repita valores
INVALID_CURSOR	ORA-01001	Se realizó una operación ilegal sobre un cursor
INVALID_NUMBER	ORA-01722	Falla la conversión de carácter a número
LOGIN_DENIED	ORA-01017	Se intenta conectar con Oracle usando un nombre de usuario y contraseña inválidos
NO_DATA_FOUND	ORA-01403	El SELECT de fila única no devolvió valores
PROGRAM_ERROR	ORA-06501	Error interno de Oracle
ROWTYPE_MISMATCH	ORA-06504	Hay incompatibilidad de tipos entre el cursor y las variables a las que se intentan asignar sus valores
STORAGE_ERROR	ORA-06500	No hay memoria suficiente
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Se hace referencia a un elemento de un <i>varray</i> o una tabla anidada usando un índice mayor que los elementos que poseen
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Se hace referencia a un elemento de un <i>varray</i> o una tabla anidada usando un índice cuyo valor está fuera del rango legal
SYS_INVALID_ROWID	ORA-01410	Se convierte un texto en un número de identificación de fila (ROWID) y el texto no es válido
TIMEOUT_ON_RESOURCE	ORA-00051	Se consumió el máximo tiempo en el que Oracle permite esperar al recurso
TOO_MANY_ROWS	ORA-01422	El SELECT de fila única devuelve más de una fila
VALUE_ERROR	ORA-06502	Hay un error aritmético, de conversión, de redondeo o de tamaño en una operación
ZERO_DIVIDE	ORA-01476	Se intenta dividir entre el número cero.

Captura de excepciones

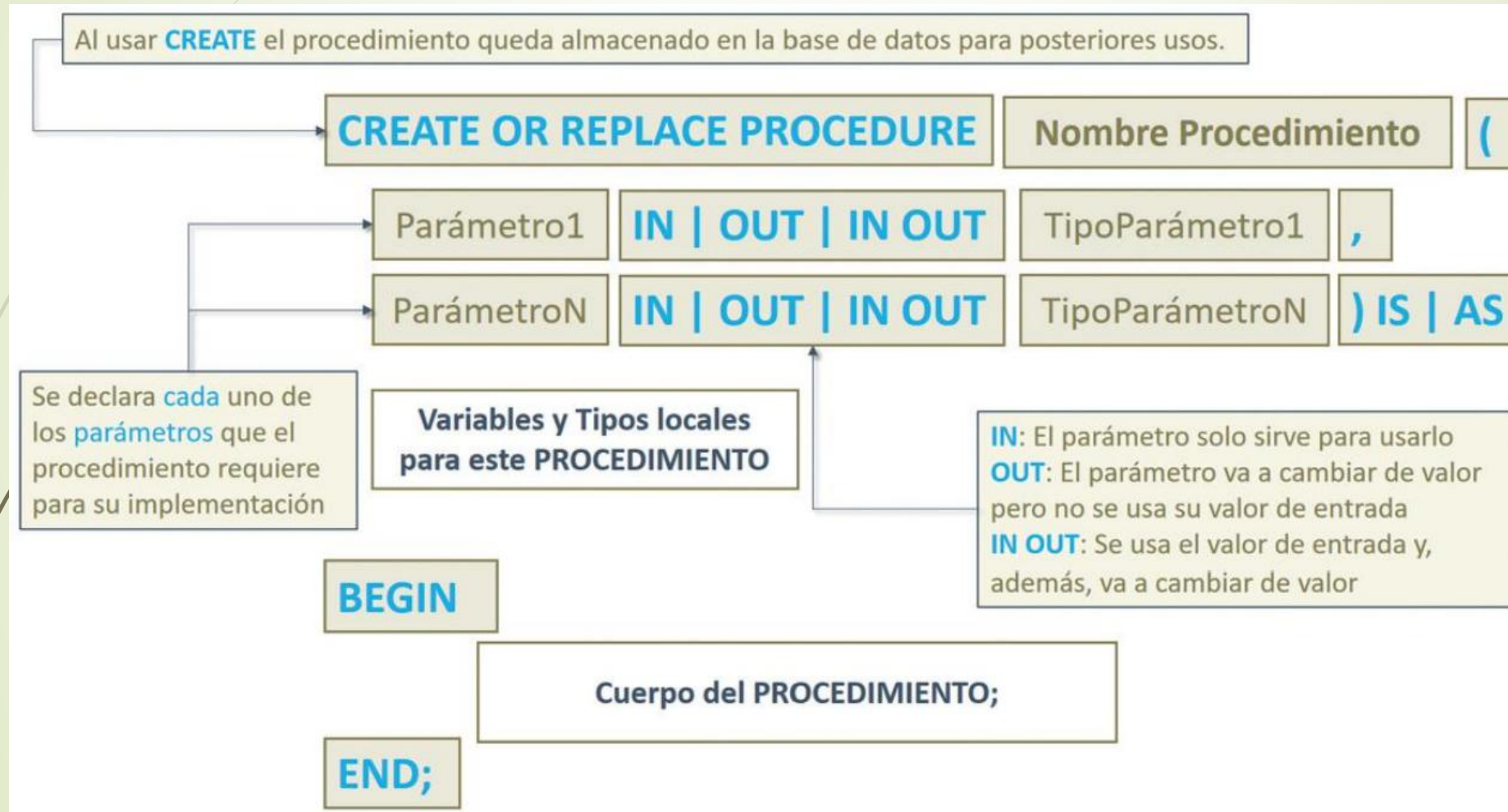
```
1 DECLARE
2
3     vNombre AUTOR.nombre_a%TYPE;
4     vDNI AUTOR.DNI%TYPE;
5
6 BEGIN
7
8     vDNI := '&DNI';
9     SELECT nombre_a INTO vNombre
10    FROM AUTOR JOIN LIBRO USING(dni)
11   WHERE dni = vDNI;
12    DBMS_OUTPUT.PUT_LINE('El escritor con DNI ' || vDNI || ' es ' || vNombre);
13
14 EXCEPTION
15
16     WHEN NO_DATA_FOUND THEN
17
18         DBMS_OUTPUT.PUT_LINE('La consulta no ha devuelto ninguna fila');
19
20     WHEN TOO_MANY_ROWS THEN
21
22         DBMS_OUTPUT.PUT_LINE('La consulta devuelve demasiadas filas');
23
24 END;
```

7. Diseño y uso de subprogramas



Procedimientos en PL/SQL

- ✓ Abstrae comandos de código para reutilizarse llamando al procedimiento
- ✓ NO devuelve nada, solo lee y/o modifica variables



DROP PROCEDURE nombreProcedimiento;

Procedimientos en PL/SQL

Solo el tipo sin el número de caracteres, dígitos

```
1 CREATE OR REPLACE PROCEDURE consultarLibro (  
2     vDNI IN VARCHAR2,  
3     vNombre_l OUT VARCHAR2,  
4     vPrecio OUT NUMBER  
5 ) IS  
6 BEGIN  
7  
8     SELECT nombre_l, precio INTO vNombre_l, vPrecio  
9     FROM LIBRO  
10    WHERE dni = vDNI;  
11  
12 EXCEPTION  
13     WHEN NO_DATA_FOUND THEN  
14         DBMS_OUTPUT.PUT_LINE('La consulta no ha devuelto ninguna fila');  
15     WHEN TOO_MANY_ROWS THEN  
16         DBMS_OUTPUT.PUT_LINE('La consulta devuelve demasiadas filas');  
17 END;
```

Parámetro 1: Usar
Parámetro 2: Procesar para salida
Parámetro 3: Procesar para salida

Procedimientos en PL/SQL

Creación

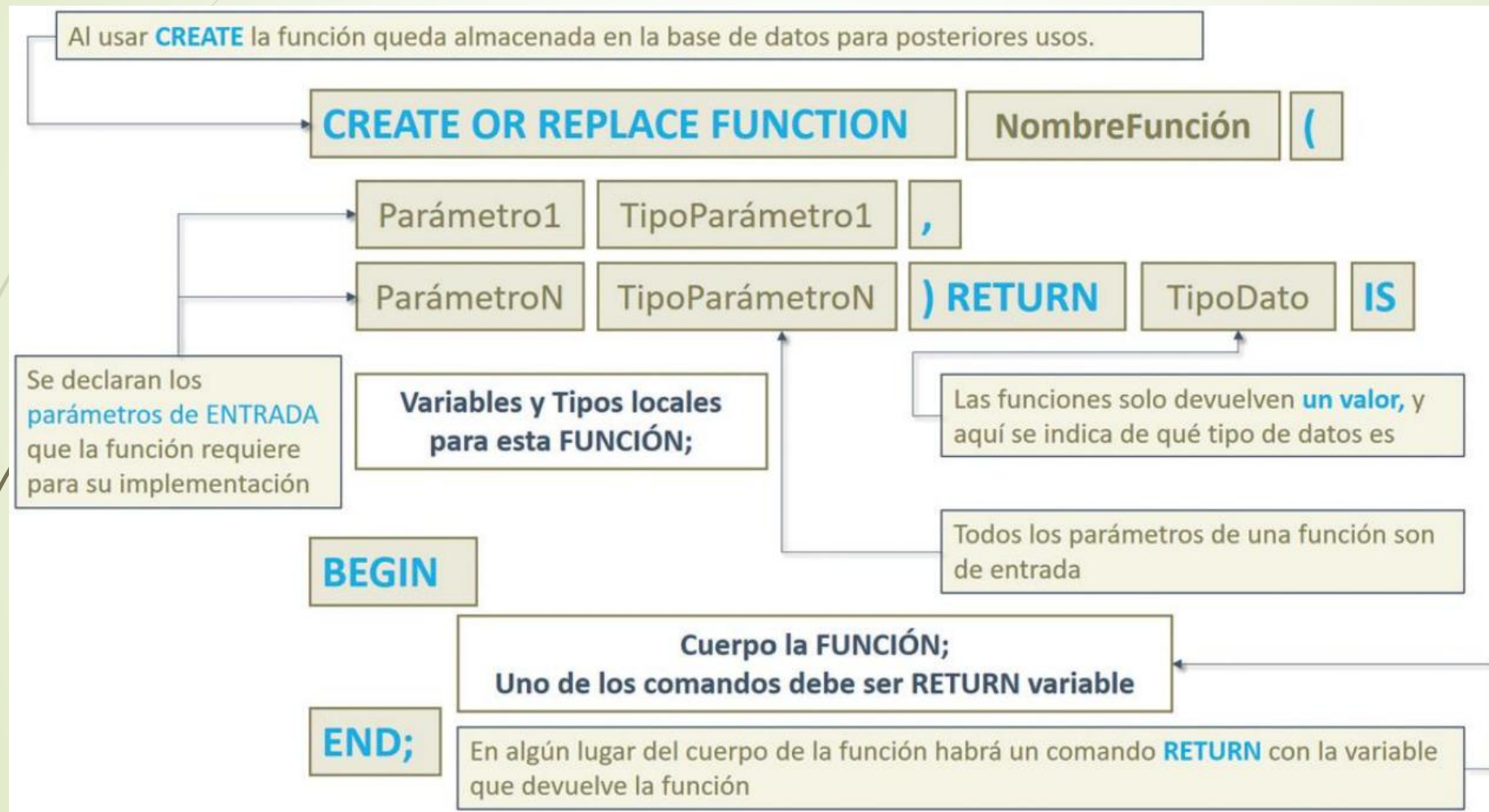
```
1 CREATE OR REPLACE PROCEDURE consultarLibro (  
2     vDNI IN VARCHAR2,  
3     vNombre_l OUT VARCHAR2,  
4     vPrecio OUT NUMBER  
5 ) IS  
6 BEGIN  
7  
8     SELECT nombre_l, precio INTO vNombre_l, vPrecio  
9     FROM LIBRO  
10    WHERE dni = vDNI;  
11  
12 EXCEPTION  
13     WHEN NO_DATA_FOUND THEN  
14         DBMS_OUTPUT.PUT_LINE('La consulta no ha devuelto ninguna fila');  
15     WHEN TOO_MANY_ROWS THEN  
16         DBMS_OUTPUT.PUT_LINE('La consulta devuelve demasiadas filas');  
17 END;
```

Uso

```
19 DECLARE  
20  
21     vDNI LIBRO.dni%TYPE;  
22     vNombre_l LIBRO.nombre_l%TYPE;  
23     vPrecio LIBRO.precio%TYPE;  
24  
25 BEGIN  
26  
27     vDNI := '&dni';  
28     consultarLibro(vDNI, vNombre_l, vPrecio);  
29     DBMS_OUTPUT.PUT_LINE('El libro es ' || vNombre_l || ' y cuesta ' || vPrecio);  
30  
31 END;
```

Funciones en PL/SQL

- ✓ Similar a los procedimientos
- ✓ Si devuelve, **UN VALOR**, con RETURN
- ✓ numero := contar('12345678A'); ≈ numero := 3;



DROP FUNCTION nombreFunción;

Funciones en PL/SQL

IN implícito (en las funciones todos los parámetros son de tipo IN)

```
1 CREATE OR REPLACE FUNCTION consultarPrecio(  
2     vPrecio NUMBER)  
3     RETURN NUMBER  
4 IS  
5  
6     vNumero NUMBER := 0;  
7  
8 BEGIN  
9  
10    SELECT COUNT(*) INTO vNumero  
11    FROM LIBRO  
12    WHERE precio > vPrecio;  
13  
14    RETURN vNumero;  
15  
16 END;
```

Devuelve UN valor de tipo numérico, puede ser entero (este caso) o decimal

Declaración de variable local para esta función. Se almacena el resultado de la SELECT y se devuelve con el RETURN

Funciones en PL/SQL

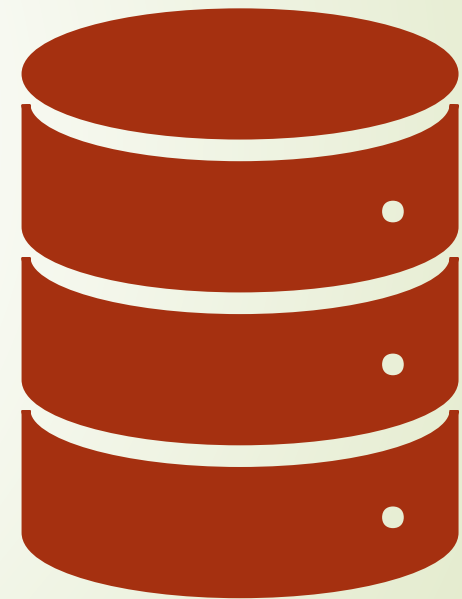
Creación

```
1 CREATE OR REPLACE FUNCTION consultarPrecio(  
2     vPrecio NUMBER)  
3     RETURN NUMBER  
4 IS  
5  
6     vNumero NUMBER := 0;  
7  
8 BEGIN  
9  
10    SELECT COUNT(*) INTO vNumero  
11    FROM LIBRO  
12    WHERE precio > vPrecio;  
13  
14    RETURN vNumero;  
15  
16 END;
```

Uso

```
18 DECLARE  
19  
20     vPrecio LIBRO.precio%TYPE;  
21     vNumeroFuncion NUMBER;  
22  
23 BEGIN  
24  
25     vPrecio := &precio;  
26     vNumeroFuncion := consultarPrecio(vPrecio);  
27     DBMS_OUTPUT.PUT_LINE('Hay ' || vNumeroFuncion || ' libros con un precio superior a ' || vPrecio || ' €.');  
28  
29 END;
```

7. Disparadores



Disparadores

Introducción.

Un disparador o trigger es un bloque PL/SQL que se ejecuta implícitamente cuando ocurre un evento o acción detectada por el programa. Los disparadores pueden ser de dos tipos:

- De la base de datos: se ejecutan de forma implícita cuando se lanza una sentencia DML (INSERT, DELETE o UPDATE) hacia la tabla.
- De aplicación: se ejecutan de forma implícita cuando ocurre un evento particular en la aplicación.

Disparadores

Componentes de un disparador.

Los disparadores garantizan que cuando se realiza una operación específica se realicen también las acciones relacionadas. Se deben utilizar para operaciones globales y no es conveniente definirlos para implementar reglas de integridad.

Al crear un disparador se tendrá en cuenta:

- El momento de ejecución en relación con el evento, que puede ser antes (BEFORE) de producirse el evento o sentencia DML, o después (AFTER) de producirse el evento.
- El evento propiamente dicho, que puede ser INSERT, DELETE o UPDATE.
- Los disparadores pueden ser de dos tipos dependiendo de la ejecución del cuerpo de este:
 - De sentencia: el cuerpo del disparador se ejecuta una vez para el evento.
 - De fila o registro: el cuerpo del disparador se ejecuta una vez para cada registro afectado por el evento.

Disparadores



Gestión de disparadores. A nivel de sentencia.

El cuerpo del disparador se ejecuta una vez producido el evento.

Cuando un disparador falla, el gestor de la base de datos hace ROLLBACK sobre las sentencias que se hayan hecho en el disparador.

Se pueden combinar varios eventos de un disparador en uno solo, aprovechando los predicados condicionales INSERTING, DELETING y UPDATING.

Disparadores

Ejemplos

Diseñar un disparador llamado SEGURIDAD que restrinja la inserción de elementos el día 1 de todos los meses en la tabla CURSO. Será a nivel sentencia y se ejecutará antes de producirse el evento.

```
CREATE OR REPLACE TRIGGER SEGURIDAD
BEFORE INSERT ON CURSO
BEGIN
    IF (TO_CHAR(SYSDATE,'DD') IN (01)) THEN RAISE_APPLICATION_ERROR(-20500, 'NO SE INSERTA EL DÍA 1');
END IF;
END;
```

La cláusula RAISE_APPLICATION_ERROR imprime el mensaje de usuario y provoca que falle.

Disparadores

Ejemplos

Modificar el ejemplo anterior para que esto ocurra al realizar una inserción, modificación o eliminación de datos el día 1 de cada mes.

```
CREATE OR REPLACE TRIGGER SEGURIDAD
BEFORE INSERT OR UPDATE OR DELETE
ON CURSO
BEGIN
    IF (TO_CHAR(SYSDATE,'DD') IN (01)) THEN
        IF DELETING THEN RAISE_APPLICATION_ERROR(-20501, 'NO SE BORRA EL DÍA 1');
        ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(-20502, 'NO SE INSERTA EL DÍA 1');
        ELSIF UPDATING THEN RAISE_APPLICATION_ERROR(-20503, 'NO SE ACTUALIZA EL DÍA 1');
    END IF;
END IF;
END;
```

Disparadores

Gestión de disparadores. A nivel de registro.

El disparador se ejecuta una vez para cada registro afectado por el evento.

Los trigger se pueden:

- activar o desactivar mediante la orden ALTER TRIGGER.
- Compilar mediante la orden CREATE TRIGGER.
- Borrar mediante la orden DROP TRIGGER
- Ver en el DD en las vistas: USER_OBJECTS, USER_TRIGGERS y DBA_TRIGGERS.

Disparadores

Ejemplos

Crear un disparador llamado SUBIDA_SAL_PROF que se ejecute después de cada modificación en la columna SALPRO en la tabla PROFESOR. Se guardarán los valores insertados en otra tabla llamada AUDITAR.

```
DROP TABLE AUDITAR CASCADE CONSTRAINTS;
```

```
/
```

```
CREATE TABLE AUDITAR
```

```
(COLUMNA1 VARCHAR2(200));
```

```
/
```

```
CREATE OR REPLACE TRIGGER SUBIDA_SAL_PROF
```

```
AFTER UPDATE OF SALPRO ON PROFESOR FOR EACH ROW
```

```
BEGIN
```

```
    INSERT INTO AUDITAR VALUES ('SUBIDA DEL SALARIO DEL PROFESOR ' || :OLD.NUMPRO || ' QUE ERA DE ' || :OLD.SALPRO || ' PASA  
    A SER DE ' || :NEW.SALPRO);
```

```
END;
```


Disparadores

Pruebas sobre disparadores.

Para probar el funcionamiento de los disparadores, es conveniente realizar lo siguiente:

- Probar cada una de las operaciones de datos sobre los disparadores, así como las operaciones sobre otros datos.
- Probar cada posibilidad de la cláusula WHEN (antes del bloque PL/SQL)
- Provocar el disparo.
- Probar el efecto del disparador sobre otros disparadores que provocarán un evento hacia la misma tabla.
- Probar el efecto de otros disparadores sobre el actual.

Disparadores

Ejemplos

Crear un disparador llamado BORRADO que se ejecute antes de borrar un profesor. Se guardarán sus datos en la tabla AUDITAR.

```
CREATE OR REPLACE TRIGGER BORRADO
BEFORE DELETE ON PROFESOR
FOR EACH ROW
BEGIN
    INSERT INTO AUDITAR VALUES ('BORRADO PROFESOR : ' || :OLD.NUMPRO || ' * ' || :OLD.NOMPRO);
END;
```

Disparadores

Ejemplos

Crear un disparador en la tabla PROFESOR llamado CALCULO_COM para calcular la comisión de un profesor de la especialidad WEB cuando se añade un registro a la tabla PROFESOR o cuando se modifica el salario de esta especialidad, 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_COM
BEFORE INSERT OR UPDATE OF SALPRO ON PROFESOR
FOR EACH ROW
WHEN (NEW.ESPPRO='WEB')
BEGIN
    IF INSERTING THEN :NEW.COMPRO:=0;
    ELSE
        IF :OLD.COMPRO IS NULL THEN :NEW.COMPRO:=0;
        ELSE :NEW.COMPRO:=:OLD.COMPRO*(:NEW.SALPRO/:OLD.SALPRO);
        END IF;
    END IF;
END;
```

Disparadores

Reglas e implementación.

Las reglas fundamentales que rigen los disparadores son las siguientes:

- No leer los datos de una tabla que se está transformando. Una tabla mutante es aquella que está siendo actualizada mediante sentencias DML, funciones o los efectos de una acción de integridad referencial (FOREIGN KEY).
- Combinar datos de una tabla restrictiva. Una tabla es restrictiva cuando un disparador tiene la necesidad de leer directamente una sentencia SQL o indirectamente mediante una restricción de integridad.

Disparadores

Ejemplos

Crear un disparador CHEQUEO que garantice que siempre que se añade un profesor a la tabla PROFESOR o se cambie un salario, el salario se encuentre dentro del establecido por el máximo y el mínimo. La condición del disparador será que el nuevo profesor sea distinto del profesor 101.

```
CREATE OR REPLACE TRIGGER CHEQUEO
BEFORE INSERT OR UPDATE OF SALPRO, ESPPRO ON PROFESOR
FOR EACH ROW
WHEN (NEW.NUMPRO<>101)
DECLARE
    V_MINSAL PROFESOR.SALPRO%TYPE;
    V_MAXSAL PROFESOR.SALPRO%TYPE;
BEGIN
    SELECT MIN(SALPRO), MAX(SALPRO) INTO V_MINSAL, V_MAXSAL
    FROM PROFESOR
    WHERE ESPPRO=:NEW.ESPPRO;
    IF (:NEW.SALPRO<V_MINSAL) OR (:NEW.SALPRO>V_MAXSAL) THEN RAISE_APPLICATION_ERROR(-20550, 'FUERA DE RANGO');
    END IF;
END;
```

Disparadores

Reglas e implementación.

Los disparadores se implementan por las siguientes razones:

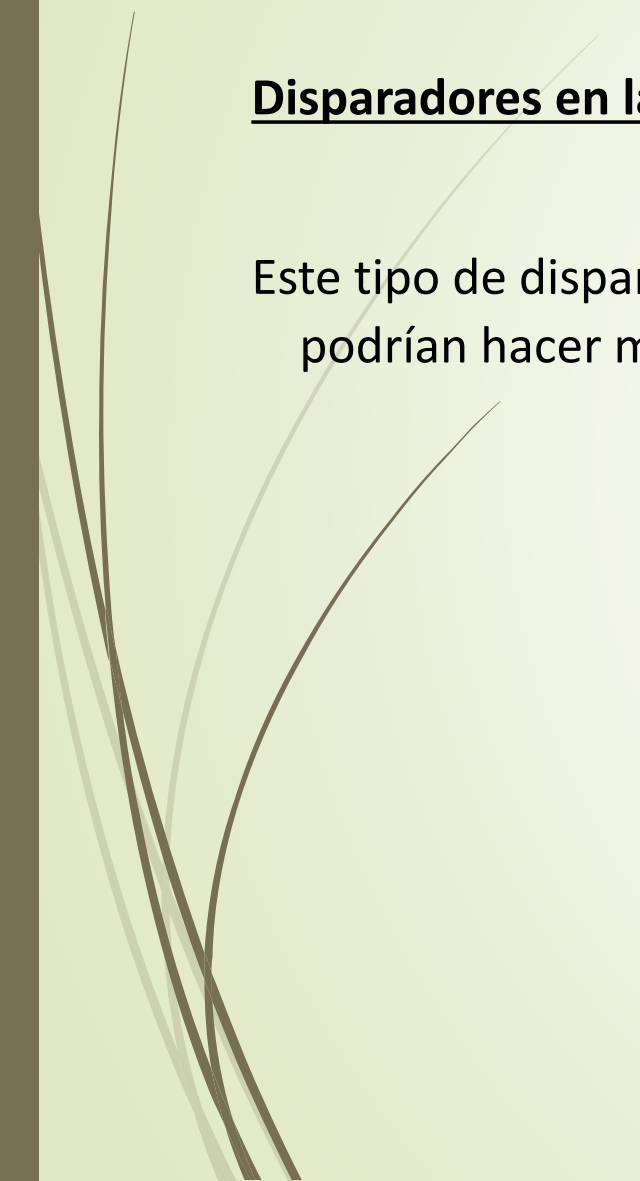
- Seguridad: permiten el acceso a las tablas según el valor de los datos.
- Auditoría: guardan los valores en otras tablas.
- Integridad de datos: implementan reglas complejas de integridad.
- Integridad referencial: implementan funcionalidad no estándar.
- Replicación de datos: copian tablas de manera sincronizada a través de réplicas.
- Datos derivados: calcular automáticamente valores que se derivan de otros datos.
- Control de eventos: controlan los eventos de forma transparente.

Disparadores



Disparadores en las vistas.

Este tipo de disparadores se usan para proporcionar modificaciones de vistas que no se podrían hacer mediante sentencias SQL.



Disparadores

Ejemplos

Construir un disparador que permita realizar operaciones de actualización en la tabla PROFESOR a partir de la vista W_PROFES. El disparador se llamará GEST_PROFES y permitirá insertar, borrar y modificar la especialidad del profesor.

```
CREATE OR REPLACE VIEW W_PROFES AS
SELECT PROFESOR.NUMPRO, NOMPRO, ESPPRO, COUNT(CURSO.NUMCUR) TOT_CURSOS
FROM PROFESOR, CURSO
WHERE CURSO.NUMPRO=PROFESOR.NUMPRO
GROUP BY PROFESOR.NUMPRO, NOMPRO, ESPPRO;
```

```
/* SI SE INTENTA INTRODUCIR UN NUEVO REGISTRO A LA VISTA, NO DEJA, DEBIDO A QUE LA VISTA ES COMPLEJA. */
```

Disparadores

```
CREATE OR REPLACE TRIGGER GEST_PROFES
INSTEAD OF DELETE OR INSERT OR UPDATE ON W_PROFES
FOR EACH ROW
BEGIN
    IF DELETING THEN DELETE PROFESOR
        WHERE NUMPRO=:OLD.NUMPRO;
    ELSIF INSERTING THEN INSERT INTO PROFESOR (NUMPRO, NOMPRO, ESPPRO)
        VALUES (:NEW.NUMPRO, :NEW.NOMPRO, :NEW.ESPPRO);
    ELSIF UPDATING THEN UPDATE PROFESOR
        SET ESPPRO=:NEW.ESPPRO
        WHERE NUMPRO=:OLD.NUMPRO;
    ELSE
        RAISE_APPLICATION_ERROR(-20501, 'ERROR');
    END IF;
END;
```

FIN