

PL/SQL

IES SAN VICENTE



Unidad 6: PROGRAMACIÓN SGBD

Guiones PL/SQL

IES SAN VICENTE



Índice del Tema

- 1- ESTRUCTURA DE UN BLOQUE PL/SQL
- 2- ESTRUCTURAS DE CONTROL EN PL/SQL
- 3- SENTENCIAS DML EN PL/SQL
- 4- PROCEDIMIENTOS Y FUNCIONES
- 5- CURSORES
- 6- CONTROL DE EXCEPCIONES
- 7- INTRODUCCIÓN AL PL/SQL DINÁMICO



INDICE

- ☐ **1- ESTRUCTURA DE UN BLOQUE PL/SQL 9**
- ☐ Ejercicios en pág 45
- ☐ **2- ESTRUCTURAS DE CONTROL EN PL/SQL 51**
- ☐ Ejercicios en pág 79
- ☐ **3- SENTENCIAS DML EN PL/SQL 85**
- ☐ Ejercicios en pág 106
- ☐ **4- PROCEDIMIENTOS Y FUNCIONES 112**
- ☐ Ejercicios en pág 126
- ☐ **5- CURSORES 136**
- ☐ Ejercicios en pág 151
- ☐ **6- CONTROL DE EXCEPCIONES 153**
- ☐ Ejercicios en pág 170
- ☐ **7- INTRODUCCIÓN AL PL/SQL DINÁMICO 173**
- ☐ Ejercicios en pág 176



HISTORIA PL/SQL

- ¿Qué es PL/SQL?, PL/SQL significa Procedural Language extensions to the Structured Query Language usado por ORACLE, básicamente es la utilización de SQL dentro de un lenguaje procedural. Ya que SQL no
- tiene ningún elemento de programación, PL/SQL es un lenguaje basado en otro lenguaje de programación llamado ADA, el cual fue un lenguaje diseñado por el departamento de defensa de Estados Unidos.

Ada es un lenguaje que se enfoca en la abstracción, ocultamiento de información y otras estrategias de diseño. Gracias a este diseño PL/SQL es un lenguaje muy poderoso que incluye la mayoría de los elementos de los lenguajes procedurales (El programa se desarrolla como una

- secuencia de pasos que la computadora ejecuta para llegar al fin deseado).

Los desarrolladores deben codificar los flujos de control de las actividades a realizar, además de las actividades en si.



Elementos del PL/SQL

- ☐ Amplia variedad de tipos de datos para declarar números, cadenas de caracteres, registros, arreglos (colecciones en Oracle) y en las ultimas versiones se ha incluido manejar XML.
- ☐ Estructuras de control iterativas (ciclos), secuenciales, condicionales (if, case, etc).
- ☐ Manejo de excepciones para atrapar errores.
- ☐ Reutilización de código como procedimientos, funciones, triggers, objetos (POO) y paquetes

CARACTERÍSTICAS DE PL/SQL (I)

- Es una extensión de SQL con características típicas de los lenguajes de programación.
- Procedural language / structured query language.
- Las sentencias SQL de consulta y manipulación de datos pueden ser incluidas en unidades procedurales de código, pero no pueden usarse instrucciones DDL ni DCL.
- Se ejecuta en el lado del servidor y los procedimientos y funciones se almacenan en la BD.
- No tiene instrucciones de entrada por teclado o salida por pantalla.

CARACTERÍSTICAS DE PL/SQL (II)

- Incluye los tipos de datos y operadores de SQL.
- Los programas se pueden compilar desde SQL*Plus (comando /) o usar SQL Developer u otros IDEs.
- Los comentarios comienzan por - - o se colocan entre /* y */.
- Trae unas librerías con funciones predefinidas, se llaman paquetes.
- Para ejecutar los procedimientos almacenados desde SQL*Plus se usa el comando exec.



1- ESTRUCTURA BLOQUE PL/SQL

ESTRUCTURA DE UN BLOQUE PL/SQL

DECLARE **opcional**

variables, cursores, excepciones definidas por el usuario

BEGIN **obligatorio**

sentencias SQL

sentencias de control PL/SQL

EXCEPTION **opcional**

acciones a realizar cuando se producen errores

END; **obligatorio**

Sección	Descripción	Inclusión
Declarative	Contiene todas las variables, constantes, cursores y excepciones definidas para los usuarios en las secciones declarativas y ejecutables.	Opcional
Excecutable	Contiene sentencias SQL para manipular datos en la base de datos y sentencias PL/SQL para manipular datos en un bloque.	Obligatoria
Exception handling	Especifica las acciones para que funcione en caso de que existan errores y condiciones anormales aparezcan en la sección ejecutable.	Opcional

Estructura de un bloque PL/SQL

Ejemplo

DECLARE

```
v_usuario VARCHAR2(10);  
v_fecha   DATE;
```

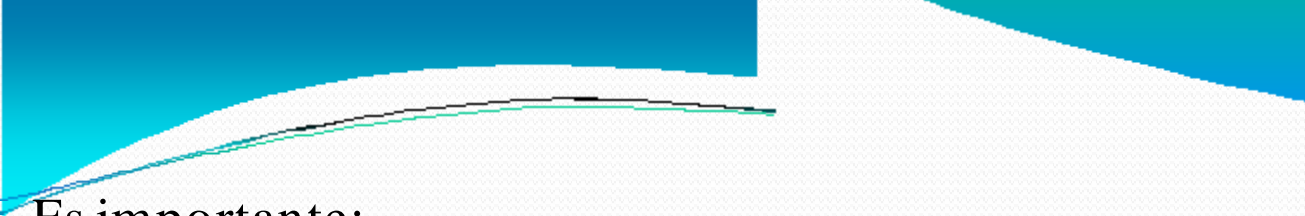
BEGIN

```
SELECT user_id, fecha  
INTO v_usuario, v_fecha  
FROM tabla;
```

EXCEPTION

```
WHEN nombre_excepcion then  
.....;
```

END;



Es importante:

- ❑ Poner punto y coma (;) al final de una sentencia SQL o sentencia de control PL/SQL.
- ❑ Cuando un bloque es ejecutado exitosamente sin errores de manejo y compilación, aparece el siguiente mensaje:

Procedimiento PL/SQL terminado correctamente.

- ❑ Las palabras claves de sección DECLARE, BEGIN y EXCEPTION no van seguidos de punto y coma.
- ❑ END y todas las otras sentencias PL/SQL requieren un punto y coma para terminar la sentencia.
- ❑ Se puede tener toda la expresión en una sola línea, pero este método no es recomendable para claridad o edición.



NOTA:

- ☐ En PL/SQL a un error se le llama excepción.
- ☐ Con la modularidad es fácil solucionar un problema complejo ya que se puede trabajar en módulos.

Tipos de Bloques

Anónimo

```
[DECLARE]

BEGIN
    -- statements

EXCEPTION

END;
```

Procedimiento

```
DECLARE name

IS

BEGIN
    --statements

[EXCEPTION]

END;
```

Función

```
FUNTION name
RETURN datatype

IS


BEGIN
    --statements

RETURN value;

[EXCEPTION]


END;
```

PL/SQL es un programa que tiene uno o más bloques. Estos bloques pueden estar separados o anidados unos con otros. Por lo tanto, un bloque puede representar una pequeña parte de otro bloque.



□ **Bloque anónimo.-** es un bloque secreto. Estos son declarados en un punto en una aplicación donde son ejecutados. Se puede enraizar en un bloque anónimo dentro de un programa pre-compilado y dentro de iSQL*Plus o un servidor director. Los triggers en los componentes de Oracle Developer constan de estos bloques.

□ **Sub-Programas.-** son bloques nombrados en PL/SQL que pueden aceptar parámetros y pueden ser invocados. Se los puede declarar como procedimientos o como funciones. Generalmente se usa un procedimiento para ejecutar una acción y una función para computar un valor.

- 
- Usando los componentes de Oracle Developer (formas, reportes, y gráficos), se pueden declarar procedimientos y funciones como parte de la aplicación (un formulario o reporte) y llamar a estos desde otros procedimientos, funciones y triggers dentro de una misma aplicación.

NOTA:

Una función es similar a un procedimiento, excepto que una función puede retornar un valor.

DECLARACIÓN DE VARIABLES EN PL/SQL

Sintaxis

Identificador [CONSTANT] tipo_dato [NOT NULL] [:= | DEFAULT expresion];

Ejemplos

v_fecha	DATE;
v_deptno	NUMBER(2) NOT NULL := 10;
v_localidad	VARCHAR2(13) := 'ATLANTA';
v_comision	CONSTANT NUMBER := 1400;



Identificador: es el nombre de la variable

CONSTANT: restringe la variable con el propósito de que su valor no pueda cambiar; las constantes deben ser inicializadas

Data type: puede ser un escalar, compuesto, referenciado o LOB.

NOT NULL: restringe a la variable para que tenga un valor .

Expr: es cualquier expresión PL/SQL que puede ser una expresión literal, otra variable o una expresión que involucra operadores y funciones.

Identificadores

Pueden contener 30 caracteres.

Deben empezar con un carácter alfabético.

Pueden contener números, signos de dólar, subrayar, y signos de números.

No pueden contener caracteres como guiones, slashes, y espacios.

No deben tener el mismo nombre de una columna de la tabla de la base de datos.

No deben haber palabras reservadas.

Tipos De Datos

Tipos de datos Oracle				
Alfanuméricos	Numéricos	Fecha	Binarios	Otros
CHAR VARCHAR2 VARCHAR NCHAR NVARCHAR2 LONG	NUMBER FLOAT	DATE	RAW LONG RAW BLOB CLOB NLOB BFILE	ROWID

Los más usados

- **Alfanuméricos:**

- **VARCHAR2(n).** Cuando se declara un dato de este tipo, éste puede contener caracteres (codificados según el juego de caracteres de la sesión) hasta una longitud total de n posiciones. La longitud máxima para las columnas de este tipo de datos es de 4000 posiciones. Es obligatorio especificar la longitud de una columna varchar2

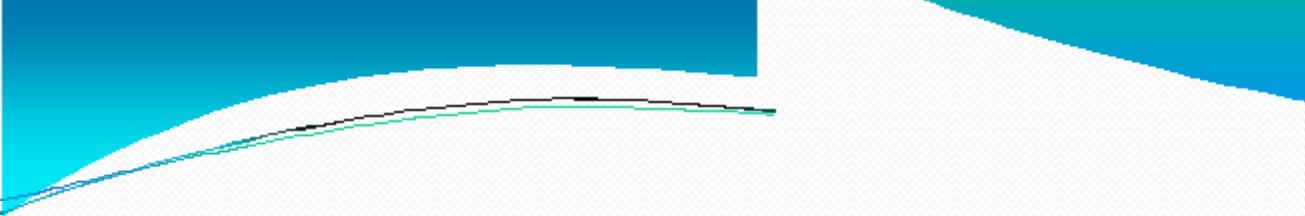
- **Numérico:**

- **NUMBER(p,s).**
 - Donde p indica el número total de dígitos (hasta 38),
 - s indica la escala, o número de dígitos a la derecha del punto decimal (hasta 127).
 - Si al almacenar un valor en una columna de tipo number se excede la precisión (número de dígitos), Oracle da un error, si se excede la escala (más decimales de los que soporta) Oracle redondea el valor a la capacidad de la columna.



□ Fecha:

- **Date:** Una columna de tipo date almacena información sobre fecha y hora. Para cada dato de tipo date, Oracle almacena información sobre siglo, año, mes, día, hora, minuto y segundo.
- Se puede especificar un valor de fecha mediante un literal (que Oracle intentará convertir) o mediante la función TO_DATE.
- '02/01/1975'
- TO_DATE('20/04/2003 09:20','DD/MM/YYYY HH:MI')

- 
- **LOB:** Los tipos de datos LOB (Large OBject), se usan para almacenar datos de gran tamaño, como ficheros (BFILE), texto (CLOB), vídeo o imágenes (BLOB).
 - **ROWID:** Toda fila de toda tabla tiene una dirección o identificación única en Oracle, conocida como ROWID

Atributos %TYPE y %ROWTYPE

- **%TYPE** sirve para declarar una variable basada en:
 - Otras previamente declaradas.
 - Una columna de la BD.
- Preceder %TYPE por:
 - La tabla y la columna de la BD.
 - El nombre de la variable definida con anterioridad
- Usar **%ROWTYPE** para tipos de datos compuestos (filas completas, registros...).



El atributo %ROWTYPE

- ❑ Declara un variable de acuerdo a una colección de columnas en una tabla de base de datos
- ❑ El prefijo %ROWTYPE con la tabla de base de datos
- ❑ Campos en el registro toma sus nombres y tipos de datos desde la columna de las tablas o vistas.

Declarando registros con el atributo %ROWTYPE

□ Sintaxis

DECLARE

 identifier reference%ROWTYPE;

Donde: *identifier* es el nombre escogido por el registro como un todo
 reference es el nombre de una tabla o un tipo registro

Asignando valores y registros

Se puede asignar una lista de valores comunes a un registro usando sentencias SELECT o FETCH (como veremos más adelante). Asegúrese que el nombre de columnas aparezcan en el mismo orden como los campos en su registro. También se puede asignar un registro a otro si tienen el mismo tipo de datos.



Ventajas de usar %ROWTYPE

- El número de campos y tipo de datos de las columnas de base de datos no necesita ser conocida.
- El numero de campos y tipo de datos de las columnas de la base de datos se cambian en tiempo de ejecución, luego si variamos su número o tipo de campos no es necesario modificar nada, ya que al tener **%rowtype** crea la variable con el cambio efectuado.
- El atributo declarado es aprovechable cuando se recupera una fila entera con la sentencia **SELECT**.

Los atributos %ROWTYPE

- Ejemplos:
- Declare una variable para guardar la información acerca de un departamento de la tabla departamentos.

```
Dept_record  departtments%ROWTYPE;
```

- Declare una variable para guardar la información acerca de un empleado de la tabla employees.

```
Emp_record  employees%ROWTYPE;
```

Asignación de valores a variables

Sintaxis: **identificador := expresión;**

Ejemplos: **v_fecha := '31-OCT-2003';**
 v_apellido := 'López';

Desde consulta:

```
SELECT sal * 0.10 INTO v_comision  
FROM emp  
WHERE empno = 7082;
```

Dbms_output.put_line

Método de un paquete predefinido de ORACLE que sirve para que el servidor muestre información por pantalla, se usa fundamentalmente en la fase de depuración de los procedimientos.

begin

 dbms_output.put_line ('Mi 1º bloque PL/SQL
diseñado por '||user||' el día '||sysdate);

end;

/

Nota: si no nos imprime nada en pantalla, únicamente “Procedimiento PL/SQL terminado correctamente debemos añadir antes la siguiente línea:

set serveroutput on;

COMPONENTES DEL CUERPO

Literales

Caracteres y fechas deben estar encerrados en comillas simples.

v_name := `Henderson`

Los números pueden ser simples valores o notaciones científicas.

n_num:=5;

Un slash (/) se pone en bloques de PL-SQL en archivos de script o en herramientas tales como iSQL*PLUS. En el TOAD y en SQL Developer NO es necesario.

Comentarios En Código

Los comentarios en líneas simples se hacen con dos guiones (--)

Los comentarios para múltiples líneas se hace entre los símbolos /* y */. Ejemplo

```
DECLARE.....
```

```
V_sal number (9,2);
```

```
BEGIN
```

```
/*calcular el salario anual basado  
en la fecha de contrato*/
```

```
V_sal := g_monthly_sal *12;
```

```
END
```

Funciones De Sql En Pl/Sql: Ejemplo.

Convertir el nombre de empleado a
minúscula:

```
V_ename := LOWER(v_name);
```

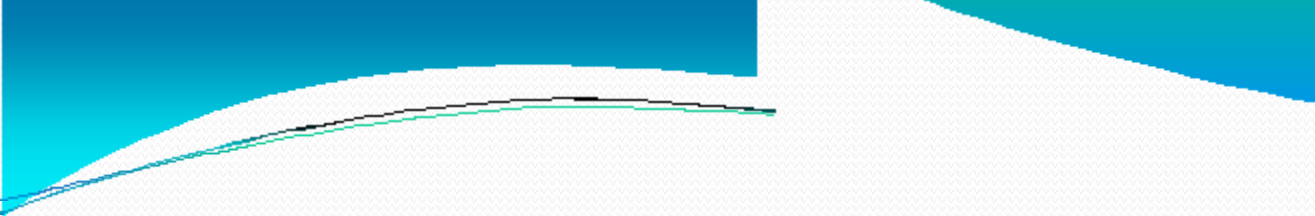
Conversión De Tipos De Datos

- Convertir datos a tipos de datos comparables.
- Mezclar tipos de datos puede resultar en un error y afectar al funcionamiento.
- Funciones de conversión son:
 - TO_CHAR
 - _TO_DATE
 - _TO_NUMBER

```
DECLARE
```

```
V_date date := TO_DATE('12-JAN-2001','DD-MON-  
YYYY');
```

```
BEGIN
```



Esta declaración produce un error en tiempo de ejecución si la variable `v_date` es declarada como un tipo de dato `DATE`.

```
V_date := `January 13, 2001`;
```

para corregir el error, use la función de conversión `TO_DATE`.

```
V_date := TO_DATE (`January 13, 2001`, `Month DD, YYYY`);
```

BLOQUES ANIDADOS Y ALCANCE DE LAS VARIABLES.

Los bloques de PL/SQL pueden ser anidados dondequiera que una declaración ejecutable sea permitida.

Una sección de excepción puede contener bloques anidados.

El alcance de un identificador es la región de una unidad de programa desde la cual se puede referenciar el identificador.



Alcance De Los Identificadores.

Un identificador es visible en la región donde se puede referenciar el identificador sin tener que caracterizar este:

- Un bloque puede buscar bloques encerrados.
- Un bloque no puede observar los bloques encerrados.

	Ámbito	Visibilidad
X Exterior	<div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>END;</div> <div>...</div> <div>END;</div> </div>	<div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>END;</div> <div>...</div> <div>END;</div> </div>
	<div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>END;</div> <div>...</div> <div>END;</div> </div>	<div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>END;</div> <div>...</div> <div>END;</div> </div>
X Interior	<div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>END;</div> <div>...</div> <div>END;</div> </div>	<div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>DECLARE X REAL; BEGIN</div> <div>...</div> <div>END;</div> <div>...</div> <div>END;</div> </div>

Ejemplo

```
DECLARE
  a CHAR;
  b REAL;
BEGIN
  -- Identificadores accesibles aquí: a (CHAR), b
  DECLARE
    a INTEGER;
    c REAL;
  BEGIN
    -- Identificadores accesibles aquí: a (INTEGER), b, c
  END;
  ...
END;
```




Capacitar Un Identificador

La capacitación puede etiquetar un bloque encerrado.

Capacitar un Identificador: puede usarse el prefijo de etiquetación de bloques.

```
DECLARE
    birthdate DATE;
BEGIN
    DECLARE
        birthdate DATE;
        BEGIN
            outer.birthdate := TO_DATE (03-AUG-1976, DD-
MON-YYYY)
        END
    END
END
```

Ejemplo

```
<<externo>>
```

```
DECLARE
```

```
    cumple DATE;
```

```
BEGIN
```

```
    DECLARE
```

```
        cumple DATE;
```

```
BEGIN
```

```
    ...
```

```
    IF cumple = exterior.cumple THEN ...
```



VARIABLES DE SUSTITUCIÓN

- Una primera forma de “obtener” información por parte del usuario para luego utilizarla dentro de un bloque o programa PL/SQL es empleando variables de sustitución.
- (Ejercicio) Escriba en PL/SQL un bloque que presente la mundialmente famosa frase: Hola Mundo en mayúsculas.
- Modifique el código anterior para que en lugar de Hola Mundo, el usuario vea una línea de salida con su nombre. Ejemplo: si el usuario ingresa el nombre Jose, la salida del programa será Hola Jose



Solución

DECLARE

```
aux_hola varchar2(50) := 'Hola';
```

```
aux_nombre varchar2(50) := '&nombre_usuario';
```

BEGIN

```
DBMS_OUTPUT.put_line(upper(aux_hola)||  
'||upper(aux_nombre));
```

END;

- El **&nombre_usuario** es una variable de sustitución, su valor será solicitado al usuario cuando se ejecute el bloque PL/SQL y el valor ingresado será empleado en las Sentencias de Ejecución

EJERCICIOS

- 1.- Determinar cuales de los siguiente Identificadores son equivalentes en PL/SQL
 - *Identificador1* NUMBER;
 - *Identificador_1* NUMBER;
 - *identificador1* NUMBER;
 - *IdEntificador_1* NUMBER;
 - *IDENTIFICADOR1* NUMBER;
- 2.- Determinar cual de los siguientes identificadores en PL/SQL es el válido:
 - *Primera variable* VARCHAR2(40);
 - *end* BOOLEAN;
 - *Una_variable* VARCHAR2(40);
 - *Otra-variable* VARCHAR2(40);
- 3.- ¿ Funcionaría la siguiente sentencia en PL/SQL?. En caso negativo proponer como resolverlo.

DECLARE

nombre VARCHAR2(80);

direccion VARCHAR2(80);

tipo empleado VARCHAR2(4);

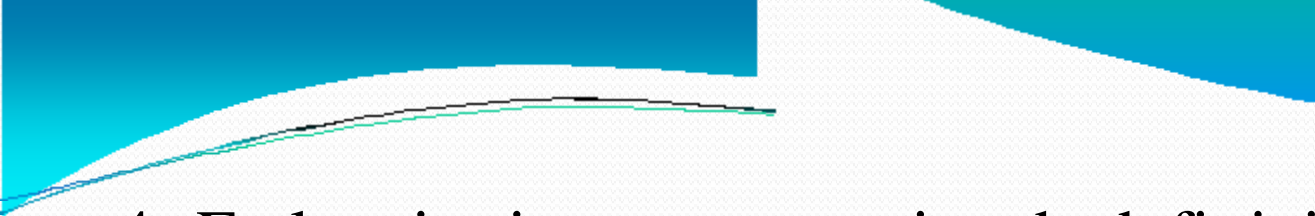
BEGIN

SELECT name, address, type

INTO nombre, direccion, tipo empleado

FROM emp;

END;



4- En las siguientes sentencias de definición de Subtipos, hay una que no es válida... Indicar cual es, y como solucionarlo.

DECLARE

SUBTYPE numero1 IS NUMBER;

SUBTYPE cadena IS VARCHAR2(10);

total numero1(6,2);



5.- Indicar del siguiente juego de declaraciones, cuales son correctas y cuales no, indicando además por qué no son correctas.

DECLARE

cierto BOOLEAN:=FALSE;

id_externo NUMBER(4) NOT NULL;

cosa NUMBER:=2;

*producto NUMBER:=2*cosa;*

suma NUMBER:=cosa+la_otra;

la_otra NUMBER:=2;

tipo_uno NUMBER(7,2) NOT NULL:=3;

tipo_otro tipo_uno%TYPE;

6.- Suponiendo que tenemos una tabla llamada EMP, en la cual existen tres campos: nombre VARCHAR2(40), direccion VARCHAR2(255), telefono NUMBER(10)... ¿Qué valores podrán tomar las variables que definimos en la siguiente declaración?

DECLARE

valor1 emp%ROWTYPE;

valor2 emp.nombre%TYPE;

valor3 emp.telefono%TYPE;

CURSOR c1 IS

SELECT nombre,direccion FROM emp;

valor4 c1%ROWTYPE;

valor5 emp%ROWTYPE;

7.- En base al enunciado anterior, ¿Sería correcta la siguiente asignación?

valor1:=valor5;

¿y esta?

valor4:=valor1;

Razonar el por qué en ambos casos.



8.-¿ Es esta declaración correcta en PL/SQL?

DECLARE

i, j NUMBER;

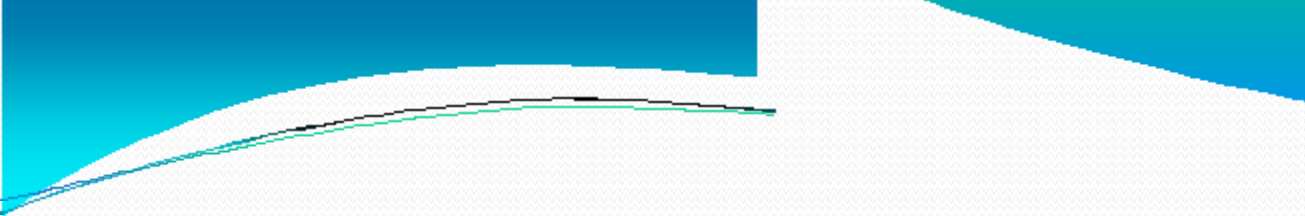
¿Y si la escribieramos así?

DECLARE

i, j NUMBER, NUMBER;

9- *Escribir un bloque PL/SQL que escriba el texto “Hola”.*

10– *Escribe un bloque PL/SQL que sume dos números y te diga que números ha sumado y cuál es el resultado.*

- 
- 11-Escribe un bloque PL/SQL donde declares una variable, inicializada a la fecha actual, que imprimes por pantalla.
 - 12-Escribe un boque PL/SQL que almacene en una variable numérica el año actual. Muestra por pantalla el año. Puedes usar las funciones to_number y to_char.



2- ESTRUCTURAS DE CONTROL

ESTRUCTURAS DE CONTROL EN PL/SQL

SENTENCIAS IF

Sintaxis:

```
IF condicion THEN
    instrucciones;
[ELSIF condicion THEN
    instrucciones;]
[ELSE
    instrucciones;]
END IF;
```



Controlando Flujo de Ejecución de PL/SQL

- Utiliza la condición IF para cambiar la ejecución lógica de las sentencias. Las condiciones de las sentencias IF son:

IF-THEN-END IF

IF-THEN-ELSE-END IF

IF-THEN-ELSIF-END IF



Sentencias IF Simples

Las sentencias simples tienen la siguiente sintaxis:

```
IF condicion THEN sentencia  
END IF;
```



Sentencias IF Compuestas

Las sentencias IF compuestas usan operadores logicos como AND Y NOT

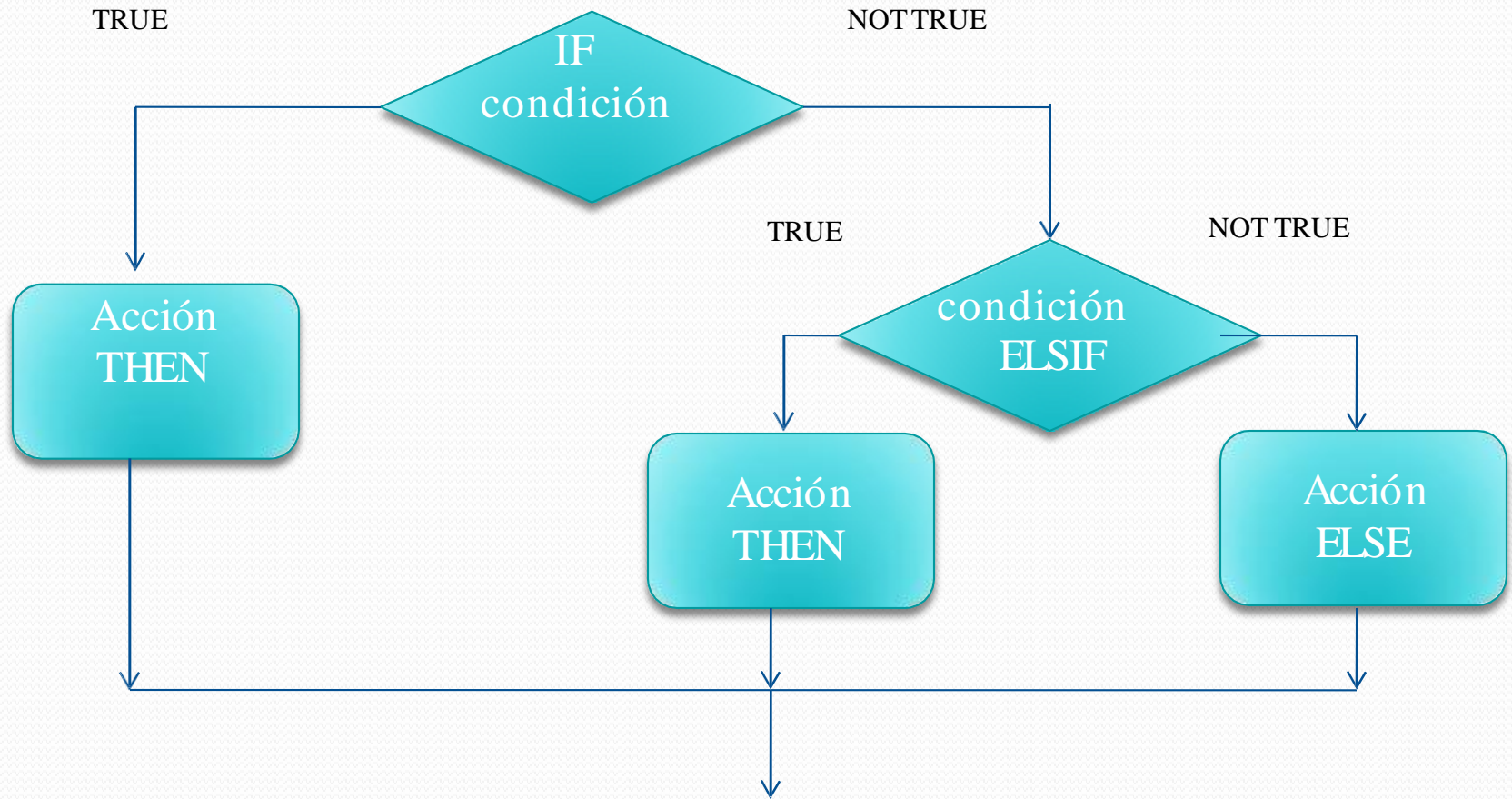
SINTAXIS:

IF condición AND condicion

THEN sentencia;

END IF;

Flujo De Ejecución De Sentencias IF-THEN-ELSEIF





Sentencias IF-THEN-ELSE

Para dar un valor, calcular un porcentaje del valor basado en una condicion.

Ejemplo.

```
IF          v_start > 100 THEN
            v_start := 0.2 v_start;
ELSIF      v_start >= 50 THEN
            v_start := 0.5 v_start;
ELSE
            v_start := 0.1 v_start
END IF;
```

ESTRUCTURAS DE CONTROL EN PL/SQL

SENTENCIAS CASE

Similar al switch de C, evalúa cada condición hasta encontrar alguna que se cumpla. La sintaxis es:

```
CASE [expresion]
WHEN [condicion1|valor1] THEN
    bloque_instrucciones_1
WHEN [condicion2|valor2] THEN
    bloque_instrucciones_2
....
ELSE
    bloque_instrucciones_por_defecto
END CASE;
```

Expresiones CASE

- Una expresión CASE selecciona un resultado y lo retorna.
- Para seleccionar el resultado , la expresión CASE usa una expresión cuyo valor es usado para seleccionar una o varias alternativas.

CASE selector

WHEN expression1 THEN result1

WHEN expression2 THEN result2

.....

WHEN expressionN THEN resultN

[ELSE resultN+1;]

END;

Ejemplos

CASE

WHEN $v_sal < 1000$ THEN

$v_sal := v_sal + 100;$

WHEN $v_sal > 2000$ THEN

$v_sal := v_sal - 100;$

END CASE;

CASE $tipo$

WHEN 1 THEN

$procesar_pedido_local;$

WHEN 2 THEN

$procesar_pedido_domicilio;$

WHEN 3 THEN

$procesar_pedido_extranjero;$

ELSE

$procesar_pedido_normal;$

END CASE;

Tablas Lógicas

- Construir una condición booleana simple con un operador de comparación

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Condiciones booleanas

*¿Cuál es el valor de **v_result** en cada caso?*

v_result := v_vari1 AND v_vari2;

v_result	v_vari1	v_vari2
TRUE	TRUE	TRUE
FALSE	FALSE	FALSE
NULL	TRUE	NULL
NULL	NULL	FALSE
FALSE	TRUE	FALSE

Ejercicio. Condiciones Booleanas

□ ¿Cuáles es el valor de V_FLAG en cada caso ?

```
v_flag := v_recorder_flag AND v_available_flag;
```

V_REORDER_FLAG	V_AVAILABLE_FLAG	V_FLAG
TRUE	TRUE	?
TRUE	FALSE	?
NULL	TRUE	?
FALSE	FALSE	?

Respuestas

1. TRUE 2. FALSE 3. NULL 4. FALSE

ESTRUCTURAS REPETITIVAS EN PL/SQL

- Los bucles repiten una sentencia o un grupo de sentencias varias veces.
- Hay 3 tipos de bucles:
 - Bucle básico. **LOOP**. Acciones repetitivas sin condiciones globales. Como si no existiera.
 - Bucle **FOR**. Acciones repetitivas basándose en un contador. Número conocido de vueltas.
 - Bucle **WHILE**. Basándose en una condición.

CONTROL ITERATIVO: SENTENCIAS LOOP

- LOOPS repite una sentencia o secuencia de múltiples sentencias
- Hay tres tipos de ciclos
 - Ciclo básico – ejecuta acciones repetitivas sin condiciones de conjunto
 - Ciclo for - ejecuta control iterativo de acciones basadas en un conteo
 - Ciclo while – ejecuta control iterativo de acciones basadas en una condición
 - Use la sentencia EXIT para terminar el ciclo.

Ciclos Básicos

LOOP	-- delimitado
statement1;	-- sentencias
...	
EXIT [WHEN condition]	-- Termina sentencia
END LOOP;	-- delimitador

Condicion: Es una variable booleana o expresión (TRUE, FALSE, o NULL)

Ciclos Básicos

La mas simple forma de las sentencias LOOP es la básica, la cual encierra una secuencia de sentencias entre las palabras reservadas LOOP y END LOOP. Un ciclo básico permite la ejecución de al menos una sentencia como mínimo.

La sentencia EXIT

Usted puede usar EXIT para terminar el ciclo.

CICLOS WHILE

```
WHILE condition LOOP
```

```
    statement1;
```

```
    statement2;
```

```
    ...
```

```
END LOOP;
```

Se utiliza el WHILE para repetir sentencias mientras una condición es verdadera

-- La condición es evaluada al comienzo
-- de cada iteración

- En la sintaxis:
 - *Condition* es una variable booleana (TRUE, FALSE, NULL)
 - *Statement* puede ser una o mas sentencias PL/SQL o SQL
- Si las variables involucradas en las condiciones no cambian durante el cuerpo del ciclo, la condición siempre es TRUE y el ciclo no termina
- Nota: Si la condición produce un NULL, el ciclo es desviado al control de la próxima sentencia

Ciclos WHILE: Ejemplo

□ DECLARE

```
V_country_id locations.country_id%TYPE:='CA';
V_location_id locations.location_id%TYPE;
V_city      locations.city%TYPE:='Montreal';
V_counter NUMBER := 1;
BEGIN
    SELECT MAX (location_id) INTO v_location_id
    FROM locations WHERE country_id = v_country_id;
    WHILE v_counter <= 3 LOOP
        INSERT INTO locations (location_id, city, country_id)
        VALUES ((v_location_id + v_counter), v_city, v_country_id);
        v_counter := v_counter + 1;
    END LOOP;
END;
```

Estructuras repetitivas en PL/SQL

Bucle WHILE. Ejemplo.

DECLARE

v_contador binary_NUMBER := 0;

BEGIN

WHILE v_contador <= 10 LOOP

INSERT INTO prueba (id, contador)

VALUES (v_id, v_contador);

v_contador := v_contador + 1;

END LOOP;

COMMIT;

END;

BUCLE FOR

```
FOR indice IN [REVERSE] valor_inicial .. valor_final LOOP
    instrucciones;
    .....
END LOOP;
```

- Usar bucle FOR para n° fijo de repeticiones.
- El índice se declara implícitamente. No se declarara.



Estructuras repetitivas en PL/SQL

Bucle FOR

- **El índice fuera del bucle no está definido, por lo tanto no se puede referenciar, estará indefinido.**
- **Usa una expresión para hacer referencia al valor actual de un índice.**
- **No hagas uso de un índice como objetivo de una asignación.**

Ciclos FOR. Sintaxis

FOR counter IN [REVERSE]

lower_bound .. Upper_bound LOOP

statement1;

-- de cada iteración

statement2;

...

END LOOP;

Use un ciclo FOR para conseguir un atajo para el numero de iteraciones

No declare un contador; este es declarado implícitamente como un entero

‘lower_bound .. Upper_bound’ es sintaxis requerida

- ❑ REVERSE causa al contador decrementar con cada iteración desde el salto alto al bajo
- ❑ Lower_bound especifica el salto mas bajo del rango de valores del contador
- ❑ upper_bound especifica el salto mas alto del rango de valores del contador

Ciclo FOR

- Inserte tres nuevas location_id para el código de la ciudad CA y la ciudad de Montreal

```
DECLARE
  V_country_id      locations.country_id%TYPE := 'CA';
  V_location_id     locations.location_id%TYPE ;
  v_city            locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX (locatio_id) INTO v_location_id
    FROM locations
   WHERE country_id = V_country_id;
  FOR i IN 1..3 LOOP
    INSERT INTO locations (location_id, city, country_id)
      VALUES ((v_location_id + i ), v_city, v_country_id);
  END LOOP;
END;
```



Estructuras repetitivas en PL/SQL

Bucle FOR. Ejemplo

DECLARE

valor_inicial NUMBER := 1;

valor_final NUMBER := 100;

FOR i IN valor_inicial .. valor_final LOOP

.....

END LOOP;

Estructuras repetitivas en PL/SQL

Bucle FOR. Ejemplo

Inserta las 10 primeras filas del pedido 601.

DECLARE

v_id prueba.id%TYPE := 601;

BEGIN

.....

FOR i IN 1 .. 10 LOOP

 INSERT INTO prueba (id, contador)

 VALUES (v_id, i);

END LOOP;

.....

END;

Líneas directivas mientras se usan ciclos

- ☐ Use el ciclo básico LOOP cuando las sentencias dentro del ciclo deban ejecutarse al menos una vez
- ☐ Use el ciclo WHILE si la condicion debe ser evaluada al comienzo de cada iteración.
- ☐ Use un ciclo FOR si se conoce el numero de iteraciones.

Ciclos Anidados y etiquetas

- ☐ Ciclos anidados para múltiples etiquetas
- ☐ Use etiquetas para distinguir entre bloques y ciclos.
- ☐ Salga del ciclo exterior con la sentencia EXIT que referencia la etiqueta
- ☐ Se pueden anidar ciclos a múltiples niveles. Se puede anidar ciclos básicos LOOP, FOR, While dentro de otro.
- ☐ La finalización de un ciclo anidado no determina la culminación del ciclo al menos que una excepción fuese levantada. Sin embargo, se puede etiquetar ciclos y salir del ciclo externo con la sentencia EXIT.

Ciclos Anidados y etiquetas

```
...  
BEGIN  
  <<outer_loop>>  
  LOOP  
    v_counter := v_counter +1;  
    EXIT WHEN v_counter>10;  
    <<Inner_loop>>  
    LOOP  
      ...  
      EXIT outer_loop WHEN total_done = 'YES';  
      -- Leave both loops  
      EXIT WHEN inner_done = 'YES';  
      -- Leave inner loop only  
      ...  
    END LOOP Inner_loop;  
    ...  
  END LOOP outer_loop;  
END
```

EJERCICIOS

1.- ¿Es correcta la siguiente sintaxis General de la sentencia IF-THEN ELSE?, ¿Por qué?, ¿Cómo la escribirías?.

BEGIN

IF condicion1 THEN

BEGIN

secuencia_de_instrucciones1;

ELSE

secuencia_de_instrucciones2;

ENDIF;

END;

2.- ¿Qué resultado nos daría la siguiente comparación?

DECLARE

identificador1 VARCHAR2(10):='Hola Pepe';

identificador2 VARCHAR2(10):='Hola pepe';

BEGIN

IF identificador1<>identificador2 THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END;

3.- Indicar que errores existen en el siguiente código fuente:

DECLARE

a NUMBER:=1;

b NUMBER:=6;

salida_bucle BOOLEAN;

BEGIN

salida_bucle:='FALSE';

WHILE NOT salida_bucle

LOOP

BEGIN

IF a>=b THEN

salida_bucle:='TRUE';

ELSE

a:=(a+1);

END IF;

END LOOP;

END;



4- ¿Qué valor contendrá la variable 'sumador' al salir del bucle?, ¿Por qué?

DECLARE

sumador *NUMBER*;

BEGIN

FOR i IN 1..100 LOOP

sumador:=sumador+i;

END LOOP;

END;

5.- ¿Qué resultado dará la ejecución del siguiente código?

DECLARE

temp *NUMBER*(1,0);

SUBTYPE *numero* *IS* *temp*%*TYPE*;

valor *numero*;

BEGIN

WHILE *valor*<20 *LOOP*

valor:=*valor*+1;

END LOOP;

END;



6- ¿Funcionaría el siguiente trozo de código?, ¿Por qué?, ¿Cómo arreglarlo?

DECLARE

mi_valor *NUMBER*;

cierto *BOOLEAN*:=*FALSE*;

BEGIN

WHILE NOT *cierto*

LOOP

IF *mi_valor*=*NULL* *THEN*

mi_valor:=*1*;

ELSE

mi_valor:=*mi_valor*+*1*;

END IF;

IF *mi_valor*>*100* *THEN* *cierto*:=*TRUE*; *END IF*;

EXIT WHEN *cierto*;

END LOOP;

END;

7.- Escribir la sintaxis general de un código que evalúe si se cumple una condición, en caso de cumplirse que ejecute una serie de sentencias, en caso contrario que evalúe otra, que de cumplirse ejecute otras instrucciones, si ésta no se cumple que evalúe una tercera condición.. y así N veces. En caso de existir varias soluciones, comentarlas y escribir la más óptima o clara.

8.- Implementar en PL/SQL un bucle infinito que vaya sumando valores en una variable de tipo NUMBER.

9.- En base al bucle anterior, añadirle la condición de que salga cuando la variable sea mayor que 10.000.

10.- Implementar un bucle en PL/SQL mediante la sentencia WHILE, en el cual vayamos sumando valores a una variable mientras ésta sea menor que 10, y asegurándonos de que el bucle se ejecute por lo menos una vez.

11.- Implementar en PL/SQL, el código necesario de un programa que al final de su ejecución haya almacenado en una variable llamada 'cadena',

el siguiente valor:

*cadena:='10*9*8*7*6*5*4*3*2*1'*



3- SENTENCIAS DML EN PL/SQL




Con PL/SQL podemos:

- ☐ Escribir en PL/SQL una sentencia SELECT exitosamente.
- ☐ Escribiremos en PL/SQL sentencias DML
- ☐ Controlar transacciones en PL/SQL
- ☐ Determinar el resultado de las sentencias del Lenguaje de Manipulación de datos. (DML)



SENTENCIAS SQL EN PL/SQL

□ Cuando se desea extraer información o aplicar cambios a la base de datos usted debe usar SQL. PL/SQL soporta el Lenguaje de Manipulación de datos (DML) y los comandos del control de transacciones de SQL. Se puede usar una sentencia SELECT para incrementar una variable con valores consultados de una fila o una tabla.

- 
- Se pueden usar comandos DML para modificar los datos en una tabla de la base de datos. Sin embargo hay que recordar los siguientes puntos acerca de los bloques PL/SQL mientras está usando sentencias DML y comandos de control de transacciones.
 - La palabra clave END significa el fin de un bloque PL/SQL, no el fin de una transacción. Justo como un bloque puede contener múltiples transacciones, una transacción puede tener múltiples bloques.
 - PL/SQL no soporta directamente el Lenguaje de Definición de Datos (DDL), así como CREATE TABLE, ALTER TABLE o DROP TABLE. Requeriremos PL/SQL Dinámico.
 - PL/SQL no soporta sentencias del Lenguaje de Control de Datos (DCL) así como GRANT o REVOKE.

RECUPERAR DATOS DE LA BD CON SELECT

Sintaxis:

SELECT select_lista

INTO {variable_nombre[,]...
| nombre_registro}

FROM nombre_tabla

WHERE condicion;

- **Nombre_tabla:** Especifica el nombre de la tabla de la base de datos.
- **condición:** está compuesta de un nombre de columna, expresión, constante, y operadores de comparación incluyendo variables y constantes PL/SQL.
- **select_lista :** es una lista de al menos una columna y puede incluir expresiones SQL, filas, funciones, o grupo de funciones.
- **variable_nombre:** es la variable escalar que retiene el valor recuperado.
- **Nombre_registro:** es el registro PL/SQL que alberga el valor recuperado.



Guía para recuperar datos en PL/SQL

- ❑ Terminar cada sentencia SQL con un punto y coma (;).
- ❑ La cláusula INTO se requiere para la sentencia SELECT cuando esta es insertada en PL/SQL.
- ❑ La cláusula WHERE es opcional y puede ser usada para especificar variables de entrada, constantes, literales o expresiones PL/SQL.
- ❑ Especificar el mismo número de variables en la cláusula INTO como columnas en la base de datos en la cláusula SELECT. Debe asegurarse que se correspondan posicionalmente y que sus tipos de
- ❑ datos sean compatibles.

Use funciones de grupo, así como SUM, en una sentencia SQL. porque las funciones de grupo se aplican a filas en una tabla.



Cláusula INTO

- ☐ La cláusula INTO es obligatoria y ocurre entre las cláusulas SELECT y FROM. Esta es usada para especificar los nombres de variables que contienen los valores que SQL retorna de una cláusula SELECT. Usted debe especificar una variable para cada ítem seleccionado, y el orden de las variables debe corresponderse con los ítems seleccionados.
- ☐ Use la cláusula INTO para llenar variables o variables host.

□ Las consultas deben retornar una y sólo una fila.

□ La sentencia SELECT dentro de un bloque PL/SQL está dentro de la clasificación ANSI de insertar SQL, para lo cual se aplican las siguientes reglas: Las consultas deben retornar una y sólo una fila, una consulta que retorna mas de una fila o ninguna genera un error.

□ PL/SQL maneja estos errores levantando excepciones, las cuales usted puede atrapar en la sección de excepciones del bloque con las excepciones NO_DATA_FOUND y TOO_MANY_ROWS.

Ejemplo 1

☐ DECLARE

☐ v_deptno NUMBER(4);

☐ v_location_id NUMBER(4);

☐ BEGIN

☐ SELECT department_id, location_id

☐ INTO v_deptno, v_location_id

☐ FROM departments

☐ WHERE department_name = 'Sales';

☐ END;

☐ /

Ejemplo 2

```
DECLARE
    v_hire_date employees.hire_date%TYPE;
    v_salary      employees.salary%TYPE;
BEGIN
    SELECT  hire_date, salary
    INTO    v_hire_date, v_salary
    FROM    employees
    WHERE   employee_id = 100;
END;
/
```


□ En el ejemplo anterior, la variable `v_hire_date` y `v_salary` son declarados en la sección `DECLARE` del bloque `PL/SQL`.

□ En la sección ejecutable, los valores de las columnas `HIRE_DATE` y `SALARY` para el empleado con el número de empleado 100 son recuperados de la tabla `Empleados` y almacenados en las variables `v_hire_date` y `v_salary` respectivamente.

□ Observe como la cláusula `INTO`, a lo largo con la sentencia `SELECT`, recupera los valores de las columnas de la base de datos dentro de las variables `PL/SQL`.

Ejemplo 3

```
SET SERVEROUTPUT ON  
DECLARE
```

```
    v_sum_sal    NUMBER(10, 2);  
    v_deptno     NUMBER NOT NULL := 60;
```

```
BEGIN
```

```
    SELECT      SUM(salary)      - -- función de grupo
```

```
    INTO        v_sum_sal
```

```
    FROM        employees
```

```
    WHERE       department_id = v_deptno;
```

```
    DBMS_OUTPUT.PUT_LINE ('La suma del salario es'  
                           ||TO_CHAR(v_sum_sal));
```

```
END;
```

```
/
```

En el ejemplo 3, las variables `v_sum_sal` y `v_deptno` son declaradas en la sección `DECLARE` del bloque `PL/SQL`.

□ En la sección ejecutable, el total del salario para el departamento con el número de departamento 60 es calculado usando la función de grupo `SUM` de `SQL`, y asignado a la variable `v_sum_sal`.

□ Nótese que la función de grupo no puede ser usada en la sintaxis `PL/SQL`. Ellas son usadas en la sentencia `SQL` dentro del bloque `PL/SQL`.

□ La salida del bloque `PL/SQL` se muestra a continuación:

- La suma del salario es 28800
- `PL/SQL procedure successfully completed.`



INSERTAR DATOS

- En el ejemplo que se muestra a continuación, una declaración INSERT es usada dentro del bloque PL/SQL para insertar un registro dentro de la tabla Empleados. Mientras este usando el comando INSERT en un bloque PL/SQL usted puede:
- Usar funciones como USER y SYSDATE.
- Generar valores de claves primarias usando secuencias de base de datos.
- Derivar valores en el bloque PL/SQL
- Añadir valores por defecto de columna

Ejemplo

- Añada nueva información de empleados a la tabla Empleados.

BEGIN

INSERT INTO employees

(employee_id, first_name, last_name, email, hire_date, job_id,
salary)

VALUES

(employees_seq.NEXTVAL, 'Ruth', 'Cores,' 'RECORES', sysdate,
'AD_ASST', 4000);

END;

/

□ **NOTA:**

- No hay posibilidad de ambigüedad con identificadores y nombres de columnas en la sentencia **SELECT**. Ningún identificador en la cláusula **INSERT** debe ser el nombre de una columna de la base de datos.

Inserción de datos en PL/SQL. Ejemplo 2

Ej.: *Añadir información sobre un nuevo empleado en la tabla emp.*

BEGIN

```
INSERT INTO emp(empno, ename, job, deptno)
VALUES(empno_sequence.nextval, 'HARDING',
'CLERK', 10);
```

END;

ACTUALIZACIÓN DE DATOS EN PL/SQL

- Puede haber ambigüedad en la cláusula SET de una declaración UPDATE porque aunque el identificador en la parte izquierda del operador de asignación es siempre una columna de la base de datos, el identificador de la parte derecha no puede ser una columna de la base de datos o variable PL/SQL.
- Recuerde que la cláusula WHERE es usada para determinar que filas son afectadas. Si no hay filas modificadas, no ocurren errores, a diferencia de una sentencia SELECT en PL/SQL.



Ejemplo

Aumenta el salario de todos los empleados de la tabla emp que son analistas.

DECLARE

v_incre_sal emp.sal%TYPE := 2000;

BEGIN

UPDATE emp

SET sal = sal + v_incre_sal

WHERE job = 'ANALYST';

.....

END;

BORRAR DATOS EN PL/SQL

La sentencia **DELETE** quita filas no deseadas de una tabla. Sin el uso de una cláusula **WHERE**, los contenidos enteros de una tabla pueden ser quitados, a condición de que no hay restricciones de integridad.

Ejemplo

Borre filas que pertenezcan al departamento 10 de la tabla empleados.

```
DECLARE
    v_deptno employees.department_id%TYPE := 10;
BEGIN
    DELETE FROM employees
    WHERE department_id = v_deptno;
END;
/
```




Borrado de datos en PL/SQL. Ejemplo

Suprimir los empleados que trabajan en el DPTO 10.

DECLARE

v_deptno dept.deptno%TYPE;

BEGIN

DELETE FROM emp

WHERE deptno = v_deptno;

END;

EJERCICIOS

1.- ¿Funcionaria el siguiente código?. Explicar por qué y cómo solucionarlo (si se os ocurren varias formas de arreglarlo, explicarlas todas).

DECLARE

base NUMBER:=100;

BEGIN

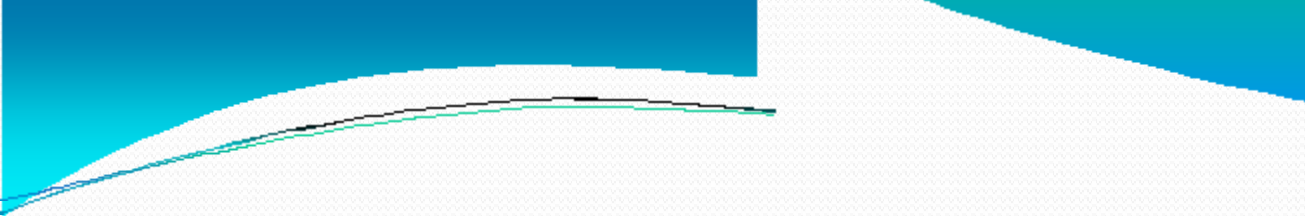
FOR dept IN 1..10 LOOP

UPDATE dept SET nom_dept=base+dept

WHERE cod_dept=base+dept;

END LOOP;

END;



2- ¿Qué ocurriría al ejecutar el siguiente código?, ¿Por qué?, ¿Cómo arreglarlo? (Si existen varias posibilidades, comentarlas e indicar cuál sería más eficiente).

DECLARE

apl_emp VARCHAR2(40):='Fernandez';

BEGIN

DELETE FROM emp WHERE apl_emp=apl_emp;

COMMIT;

END;

3.- ¿Es correcto el siguiente código en PL/SQL?, ¿Por qué? Nota: Ignorar el funcionamiento del código (no hace nada), ceñirse exclusivamente a la sintaxis válida en PL/SQL.

```
FOR ctr IN 1..10 LOOP
    IF NOT fin THEN
        INSERT INTO temp
        VALUES (ctr, 'Hola');
        COMMIT;
        factor:=ctr*2;
    ELSE
        ctr:=10;
    END IF;
END LOOP;
```

4.- ¿Qué resultado provoca la ejecución del siguiente código PL/SQL?

```
DECLARE
    variable NUMBER:=1;
    almacenamos NUMBER:=1;
BEGIN
    FOR i IN 5..variable LOOP
        almacenamos:=almacenamos+i;
    END LOOP;
    INSERT INTO traza VALUES(TO_CHAR(almacenamos));
    COMMIT;
END;
```



5- ¿Qué da este bloque?

```
DECLARE
```

```
    V_num NUMBER;
```

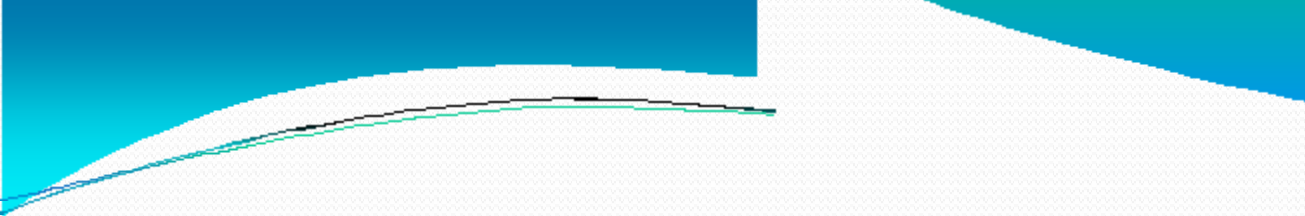
```
BEGIN
```

```
    SELECT COUNT(*) INTO V_num
```

```
    FROM productos;
```

```
    DBMS_OUTPUT.PUT_LINE(V_num);
```

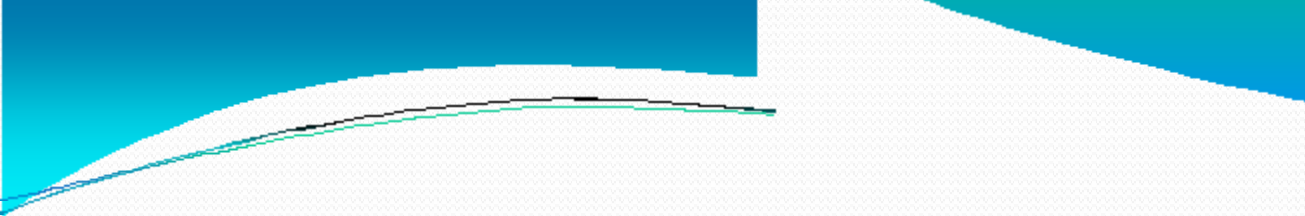
```
END;
```



- 6- Crea una tabla MENSAJES con un solo campo VALOR, de tipo VARCHAR2(5). Crea un bloque que inserte 8 elementos en la tabla con valores del 1 al 10, excepto el 4 y 5.
- 7- Vuelve a usar la tabla anterior con los datos ya rellenos, y si el número que insertas es menor que 4 entonces debes sacar un mensaje de error si ya existe, si el número es mayor o igual que 4, debes insertarlo si no existe y actualizarlo sumándole 1 si ya existe.

8- Crea una tabla PRODUCTO(CODPROD, NOMPROD, PRECIO), usando SQL (no uses un bloque PL/SQL).

- ☐ Añade un producto a la tabla usando una sentencia insert dentro de un bloque PL/SQL.
- ☐ 9. Añade otro producto, ahora utilizando una lista de variables en la sentencia insert.
- ☐ 10. Añade, ahora usando un registro PL/SQL, dos productos más.
- ☐ 11. Borra el primer producto insertado, e incrementa el precio de los demás en un 5%.
- ☐ 12. Obtén y muestra por pantalla el número de productos que hay almacenados, usando select ... Into y el mensaje “Hay n productos”.
- ☐ 13. Obtén y muestra todos los datos de un producto (busca a partir de la clave primaria, escogiendo un código de producto existente. Usa select ... into y un registro PL/SQL.
- ☐ 14- Haz un bloque PL/SQL, de forma que se indique que no hay productos, que hay pocos (si hay entre 1 y 3) o el número exacto de productos existentes (si hay más de 3).



4- PROCEDIMIENTOS Y FUNCIONES

PROGRAMACIÓN MODULAR EN PL/SQL

En PL/SQL se pueden escribir cuatro tipos de bloques de código:

- **Bloques anónimos:** No se almacenan en la BD. Se ejecutan tras escribirlos.
- **Procedimientos:** Se almacenan en el DD y son invocados. Pueden recibir y devolver múltiples parámetros.
- **Funciones:** Se almacenan en el DD y son invocadas. Pueden recibir parámetros y devuelven un valor “en su nombre”.
- **Triggers:** Se almacenan en el DD y se ejecutan automáticamente cuando ocurre algún evento.

PROCEDIMIENTOS EN PL/SQL

```
CREATE OR REPLACE PROCEDURE [esquema].nomproc  
    (nombre-parámetro {IN, OUT, IN OUT} tipo de dato, ..)  
{IS, AS}
```

Declaración de variables;
Declaración de constantes;
Declaración de cursores;

```
BEGIN
```

Cuerpo del subprograma PL/SQL;

```
EXCEPTION
```

Bloque de excepciones PL/SQL;

```
END;
```



Ejemplo Procedure

```
CREATE OR REPLACE PROCEDURE Muestra (Cad IN Varchar2)
IS
BEGIN
    DBMS_OUTPUT.PUT_LINE (Cad);
END Muestra;
```

```
DECLARE
    Mens varchar2(30) := '&Mensaje';
BEGIN
    Muestra(Mens);
END;
```

FUNCIONES EN PL/SQL

```
CREATE OR REPLACE FUNCTION[esquema].nombre-funcion  
    (nombre-parámetro tipo-de-dato,...)  
    RETURN tipo-de-dato  
{IS, AS}
```

Declaración de variables;
Declaración de constantes;
Declaración de cursores;

BEGIN

Cuerpo del subprograma PL/SQL;

EXCEPTION

Bloque de excepciones PL/SQL;

END;



Ejercicio

- Pasa el procedure de ejemplo a función, añadiéndole al principio el literal ‘La cadena es: ‘.



Ejemplo de Función

```
CREATE OR REPLACE FUNCTION Factorial (N IN NUMBER) RETURN NUMBER
AS
BEGIN
    IF N<0 THEN
        Return -1;
    ELSIF N=0 THEN
        Return 1;
    ELSE
        Return N*Factorial(N-1);
    END IF;
END Factorial;

DECLARE
    v NUMBER := &valor;
    f NUMBER;
BEGIN
    f := Factorial(v);
    DBMS_OUTPUT.PUT_LINE ('Factorial de ' || v || ' = ' || f);
END;
```

Aclaraciones sintaxis procedimientos y funciones

Nombre-parámetro: es el nombre que nosotros queramos dar al parámetro. Podemos utilizar múltiples parámetros. En caso de no necesitarlos podemos omitir los paréntesis.

IN: especifica que el parámetro es de entrada y que por tanto dicho parámetro tiene que tener un valor en el momento de llamar a la función o procedimiento. Si no se especifica nada, los parámetros son por defecto de tipo entrada.

OUT: especifica que se trata de un parámetro de salida. Son parámetros cuyo valor es devuelto después de la ejecución el procedimiento al bloque PL/SQL que lo llamó. Las funciones PLSQL no admiten parámetros de salida.

IN OUT: Son parámetros de entrada y salida a la vez.

Ejemplo de procedimiento PL/SQL

```
CREATE ORREPLACE PROCEDURE procedimiento1
    (a IN NUMBER, b IN OUT NUMBER)
IS
    vmax NUMBER;
BEGIN
    SELECT salario, maximo INTO b, vmax
    FROM empleados
    WHERE empleado_id=a;
    IF b < vmax THEN
        b:=b+100;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        b:= -1;
        RETURN;
END;
```


Ejemplo de llamada a un procedimiento PL/SQL

```
DECLARE
    vsalario NUMBER;
BEGIN
    procedimiento1 (3213, vsalario);
    dbms_output.put_line ('El salario del empleado 3213
es ' || vsalario);
END;
```

CREACIÓN DE PAQUETES

- Los paquetes sirven para agrupar bajo un mismo nombre funciones y procedimientos. Facilitan la modularización de programas y su mantenimiento.
- Los paquetes constan de dos partes:
 - **Especificación.** Que sirve para declarar los elementos de los que consta el paquete. En esta especificación se indican los procedimientos, funciones y variables **públicos** del paquete
 - **Cuerpo.** En la que se especifica el funcionamiento del paquete. Consta de la definición de los procedimientos indicados en la especificación. Además se pueden declarar y definir variables y procedimientos **privados**



Especificación. Sintaxis

CREATE [OR REPLACE] PACKAGE nombrePaquete
{IS|AS}

variables, constantes, cursores y excepciones públicas
cabecera de procedimientos y funciones

END nombrePaquete;

Ejemplo:

CREATE OR REPLACE PACKAGE paquete1 **IS**

v_cont **NUMBER** := 0;

PROCEDURE reset_cont(v_nuevo_cont **NUMBER**);

FUNCTION devolver_cont

RETURN NUMBER;

END paquete1;

Cuerpo. Sintaxis

CREATE[OR REPLACE] PACKAGE BODY nombrePaquete

IS|AS

variables, constantes, cursores y excepciones privadas

cuerpo de los procedimientos y funciones

END nombrePaquete;

Ejemplo:

```
CREATE OR REPLACE PACKAGE BODY paquete1 IS
    PROCEDURE reset_cont(v_nuevo_cont NUMBER)
    IS
    BEGIN
        v_cont:=v_new_cont;
    END reset_cont;
    FUNCTION devolver_cont
    IS
    BEGIN
        RETURN v_cont;
    END devolver_cont;
END paquete1
```



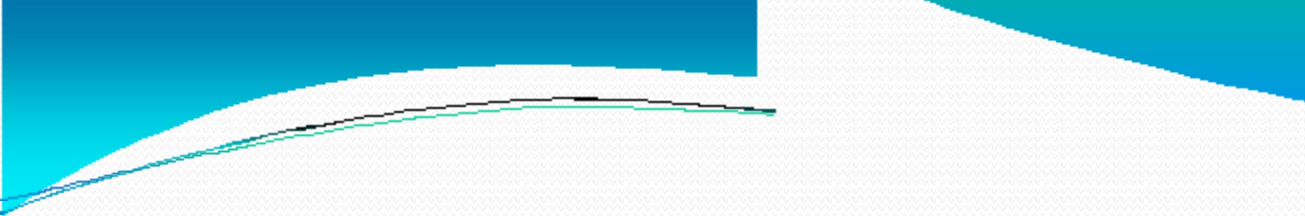
Ejecución

- Desde dentro del paquete, para utilizar otra función o procedimiento o variable dentro del mismo paquete, basta con invocarla por su nombre.
- Si queremos utilizar un objeto de un paquete, fuera del mismo, entonces se antepone el nombre del paquete a la función. Por ejemplo ***paquete1.reset_cont(4)*** (en el ejemplo anterior).



Actividades

1. Crear una función pl/sql que duplica la cantidad recibida como parámetro .
2. Crear una función pl/sql llamada factorial que devuelva el factorial de un número, por ejemplo $5! = 1 * 2 * 3 * 4 * 5 = 120$.
3. Crear un procedimiento pl/sql que muestra los números desde el 1 hasta el valor pasado como parámetro.
4. Modificar el procedimiento del Ejercicio 1 para que muestre números desde un valor inferior hasta uno superior con cierto salto, que por defecto será 1.
5. Modificar el procedimiento del Ejercicio 1 para que inserte los números en una tabla

- 
- 6- Crea una función llamada, PVP que toma como argumento un código de producto, una descripción y un coste del producto, y realice una inserción en una tabla PRODUCTOS si el código de producto (PK) no existe y en caso de existir actualice los datos de descripción y coste y devuelva el precio de venta al público, que resulta de aplicarle a ese precio de coste un margen comercial del 20%.

7- Dado este PL/SQL:


```
CREATE OR REPLACE PROCEDURE modificar_precio_producto (codigoprod NUMBER,  
nuevo NUMBER)
```

```
AS  
    Precioant NUMBER(5);  
BEGIN  
    SELECT precio_uni INTO precioant  
    FROM productos  
    WHERE cod_producto =codprod;  
    IF (precioant *0.20) > ABS(    precioant – nuevoprecio) then  
        UPDATE productos  
        SET precio_uni = nuevoprecio  
        WHERE cod_producto =codigoprod;  
    ELSE  
        DBMS_OUTPUT.PUT_LINE('Error, modificación superior a 20%');  
    END IF;  
EXCEPTION  
WHEN NO_DATA_FOUND THEN  
    DBMS_OUTPUT.PUT_LINE ('No encontrado el producto '|| codigoprod);  
END modificar_precio_producto;
```



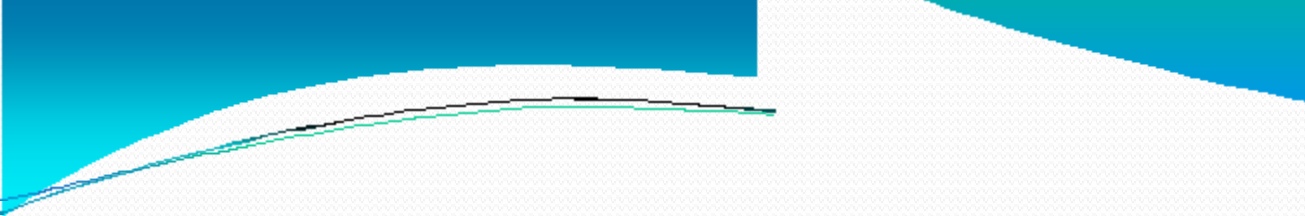

Responde a las siguientes preguntas:

- a) ¿Se trata de una función o de un procedimiento, por qué?, ¿Qué habría que cambiar para que fuera lo otro?
- b) ¿Cuál es la cabecera del procedimiento?
- c) ¿Qué es el precioant?
- d) ¿Qué es el nuevoprecio?
- e) ¿Qué es el precio_uni?
- f)) ¿Cuáles son los parámetros del procedimiento?
- g) ¿Qué es NO_DATA_FOUND?
- h) ¿Cuál es el nombre del procedimiento?
- i) ¿Dónde comienza el bloque?
- j) ¿Qué hace la cláusula into?
- k) ¿qué hace la condición del IF?
- l) ¿Porque no tiene la cláusula declare? ¿Qué tiene en su lugar?



8- Corregir los errores de sintaxis en esta función Esta función PL/SQL devuelve el número PI (3,141592653589793238462...). Calculado mediante el algoritmo que ideó John Wallis en 1665

```
CREATE FUNCTION piWallis(pIteraciones number) number
IS
    vCont number
    vRet number
    vCont = 0
    vRet = 1
    loop
        vCont = vCont + 1;
        when vCont > pIteraciones exit loop;
        if (vCont % 2) = 0
            vRet := vRet * vCont / (vCont + 1);
        else
            vRet := vRet * (vCont + 1) / vCont;
        endif;
    end loop
    return (2 * vRet);
END;
```

- 
- 9. Introduce todas las funciones y procedimientos de los ejercicios anteriores en un PACKAGE llamado FUNCIONES_PROCE
 - 10. Ejecuta los procedimientos y funciones del nuevo Paquete desde un bloque anónimo.



5- CURSORES



DEFINICIÓN DE CURSOR

- ☐ El conjunto de filas resultantes de una consulta con la sentencia `SELECT`, puede estar compuesto por ninguna, una o varias filas, dependiendo de la condición que define la consulta.
- ☐ Para poder procesar individualmente cada fila de la consulta debemos definir un cursor (que es un área de trabajo de memoria) que contiene los datos de las filas de la tabla consultada por la sentencia `SELECT`



PASOS A REALIZAR

- ☐ Definir el cursor, especificando la lista de parámetros con sus correspondientes tipos de datos y estableciendo la consulta a realizar con la sentencia `SELECT`
- ☐ Abrir el cursor para inicializarlo, siendo éste el momento en que se realiza la consulta.
- ☐ Leer una fila del cursor, pasando sus datos a las variables locales definidas a tal efecto
- ☐ Repetir el proceso fila a fila hasta llegar a la última
- ☐ Cerrar el cursor una vez que se terminó de procesar su última fila

TIPOS DE CURSORES

Los **cursores** son áreas de memoria que almacenan datos extraídos de la base de datos. Hay dos tipos:

Implícitos: Declarados implícitamente para todas las sentencias del DML y SELECT de PL/SQL, consultas que devuelven una sola fila.

Explícitos: Declarados y nombrados por el programador. Manipulados por sentencias específicas en las instrucciones ejecutables del bloque. Se usan para consultas que devuelven más de 1 fila.



CURSORES IMPLÍCITOS

a) **SELECT** de una sola fila.

Si no devuelve ninguna fila o devuelve más de una se origina un error.

Lo controlaremos en la zona **EXCEPTION**, mediante los manejadores :

NO_DATA_FOUND Y **TOO_MANY_ROWS**.

b) **Sentencias** del **DML**, de más de una fila.

CURSORES EXPLÍCITOS

Declaración de Cursor:

Sintaxis:

```
CURSOR      nombre_cursor      IS  
Sentencia select ;      /* sin INTO */
```

Ejemplo:

```
DECLARE  
    v_empno    emp.empno%TYPE;  
    v_ename    emp.empno%TYPE;  
  
CURSOR emp_cursor IS  
SELECT empno, ename  
FROM emp  
WHERE deptno = 30;
```



Apertura del cursor

- Al abrir el cursor se ejecuta realmente la consulta.
- Si la consulta no devuelve ninguna fila no se producirá ninguna EXCEPTION.
- Hay que utilizar los atributos del cursor para comprobar los resultados obtenidos tras una recuperación.

OPEN nombre_cursor;

Recuperar datos del cursor

FETCH nombre_cursor **INTO** [var1,var2,...] | nom_registro;

- Recuperar la fila actual e introducir los valores en las variables de salida.
- Incluir el mismo n° de variables.
- Relacionar posicionalmente variables y columnas.
- Comprobar si el cursor tiene filas

Cierre del Cursor

Sintaxis:

CLOSE nombre_cursor

- Desactiva el cursor y libera la memoria reservada.
- Cerrar cursor tras el procesamiento de las filas.
- Volver a abrir el cursor si fuese necesario. Hay un máximo número de cursores que pueden estar abiertos a la vez en la BD.
- No se pueden recuperar datos del cursor una vez cerrado.

Cursores Explícitos

Ejemplo

```
DECLARE
    CURSOR emp_cursor IS
        SELECT empno, ename FROM emp;

BEGIN
    OPEN emp_cursor;
    FETCH emp_cursor INTO v_empno, v_ename;
    WHILE emp_cursor%FOUND LOOP
        // Procesar información
        FETCH emp_cursor INTO v_empno, v_ename;
    END LOOP;
    CLOSE emp_cursor;
END;
```

Cursores explícitos. Atributos

%ISOPEN

booleano

TRUE si está abierto

%NOTFOUND

booleano

TRUE si la recuperación
más reciente no
devuelve fila

%FOUND

booleano

TRUE si la recuperación
más reciente
devuelve fila

%ROWCOUNT

número

nº de filas devueltas
hasta ese momento

Atributos explícitos de los cursores

- Obtener el estado de información acerca del cursor

Atributo	Tipo	Descripción
%ISOPEN	Boolea na	Evaluar VERDADERO si el cursor el abierto
%NOTFOUND	Boolea na	Evaluar V si el mas reciente fetch no retorna una fila
%FOUND	Boolea na	Evalúa Verdadero si el mas reciente fetch retorna una fila; complementa %NOTFOUND
%ROWCOUNT	Numer al	Evalúa el numero total de filas retornadas a distancia

Apertura. Comprobación

```
BEGIN
```

```
    IF NOT cursor_emp%ISOPEN THEN
```

```
        DBMS_OUTPUT.PUT_LINE ('El cursor está  
        cerrado. Vamos a abrirlo');
```

```
    OPEN cursor_emp;
```

```
END IF;
```

```
-- Cierre del cursor
```

```
CLOSE cursor_emp;
```

```
END
```


Cursor con variable registro

```
DECLARE
    CURSOR emp_cursor IS
        SELECT ename, sal FROM emp WHERE deptno = 10;
    v_reg_cursor          emp_cursor%ROWTYPE;
BEGIN
    OPEN emp_cursor;    // Abrir cursor
    FETCH emp_cursor INTO v_reg_cursor; // Leer primera fila
    WHILE emp_cursor%FOUND LOOP // mientras haya filas
        DBMS_OUTPUT.PUT_LINE(v_reg_cursor.ename||' * '||
            v_reg_cursor.sal); // procesar
        FETCH emp_cursor INTO v_reg_cursor; // leer siguiente
    END LOOP;
    CLOSE emp_cursor; // cerrar cursor
    // presentar resultados finales (si procede)
END;
```

Los atributos %NOTFOUNDy % ROWCOUNT

- ☐ Usar el atributo del cursor %ROWCOUNT para recuperar un numero exacto de filas
- ☐ Use el atributo del cursor %NOTFOUND para determinar cuando existe el ciclo.

Ejemplo:

```
□ DECLARE
□     v_empno employees.employee_id%TYPE;
□     v_ename employees.last_name%TYPE;
□     CURSOR emp_cursor IS
□         SELECT employee_id last_name
□         from employees
□
□     BEGIN
□         OPEN emp_cursor;
□         LOOP
□             FETCH emp_cursor into v_empno, v_ename;
□             EXIT WHEN emp_cursor %ROWCOUNT > 10 OR
□             emp_cursor %NOTFOUND;
□             DBMS_OUTPUT.PUT_LINE(''||TO_CHAR(v_empno)||' '||
v_ename);
□             END LOOP;
□         CLOSE emp_cursor;
□     END
□
```

Bucles FOR de cursor

FOR nombre_registro **IN** nombre_cursor **LOOP**

instrucción_1;

instrucción_2;

....

END LOOP;

- Apertura, recuperación y cierre implícitos.
- Implícito Open, fetch, exit, y close
- La variable registro se declara implícitamente

Bucle FOR de cursor.Ejemplo

```
DECLARE
CURSOR emp_cursor IS
SELECT ename, sal, TRUNC((SYSDATE – hiredate)/365) as antigüedad
      FROM emp
      WHERE deptno = 10;
BEGIN
FOR v_reg_cursor IN emp_cursor LOOP
      DBMS_OUTPUT.PUT_LINE(v_reg_cursor.ename || ' * ' ||
v_reg_cursor.sal || ' * ' ||
v_reg_cursor.antigüedad);
END LOOP;
END;
```

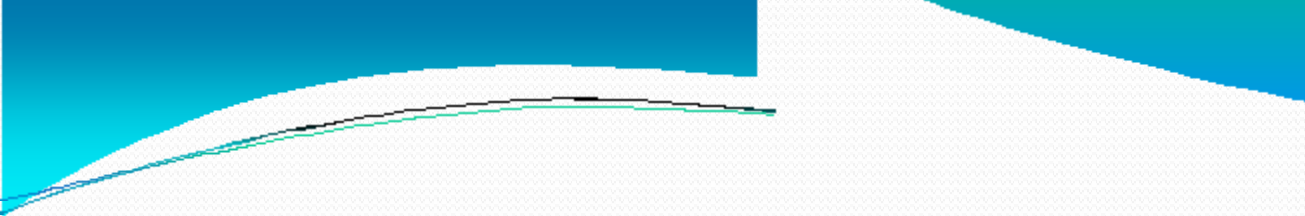
Ciclos FOR para cursores usando subconsultas

- No se necesita declarar el cursor
- Ejemplo:

```
BEGIN
    FOR emp_record IN (SELECT last_name, department_id FROM
employees) LOOP
        IF emp_record.department_id = 80 THEN
            ...
        END LOOP; -- implicit close occurs
END;
```

EJERCICIOS

- 1- Realiza un cursor sobre la tabla Bancos y sucursales, donde indiques el nombre del banco y el de cada sucursal. Hazlo con Fetch y con FOR.
- 2- Crea un campo en la tabla Cuenta, llamándolo “Rentable”, recorre la tabla Cuenta y si el haber es mayor que el debe actualizara S de lo contrario a N, sumando las rentables y las no rentables y sacándolas por pantalla al final. Usa ROWID para hacerlo
- (identificador único para cada registros de una BD).
- 3- Mira de realizar lo anterior con una UPDATE Masiva.



- ☐ 4- Utiliza un cursor y un bucle LOOP simple para recuperar y mostrar los datos de todos Los productos. Indica al final cuantos productos hay.
- ☐ 5- Usa el cursor del loop directamente en el bucle
- ☐ 6- Usa ahora un while

7. Dadas las siguientes tablas:

```
-- tabla para almacenar todos los alumnos de la
BD CREATE TABLE Alumnos
(numMatricula NUMBER PRIMARY KEY,
 nombre VARCHAR2(15),
 apellidos VARCHAR2(30),
 titulacion VARCHAR2(15),
 precioMatricula NUMBER);

-- tabla para los alumnos de informática
CREATE TABLE AlumnosInf
(IDMatricula NUMBER PRIMARY KEY,
 nombre_apellidos VARCHAR2(50),
 precio NUMBER);
```

Inserte los siguientes datos de prueba en la tabla ALUMNOS:

<i>numMatricula</i>	<i>nombre</i>	<i>apellidos</i>	<i>titulacion</i>	<i>precioMatricula</i>
1	Juan	Álvarez	Administrativo	1000
2	José	Jiménez	Informatica	1200
3	Maria	Pérez	Administrativo	1000
4	Elena	Martínez	Informatica	1200

Construya un cursor que inserte sólo los alumnos de informática en la tabla ALUMNOSINF, teniendo en cuenta la estructura de esta tabla, así por ejemplo, debe tener en cuenta que el atributo *nombre_apellidos* resulta de la concatenación de los atributos *nombre* y *apellidos*. Antes de la inserción de cada tupla en la tabla ALUMNOSINF debe mostrar por pantalla el nombre y el apellido que va a insertar.

8. Dadas las siguientes tablas:

```
CREATE TABLE Tabla_Departamento (
  Num_Depart Number(2) PRIMARY KEY,
  Nombre_Depart VARCHAR2(15),
  Ubicación VARCHAR2(15),
  Presupuesto Number(10,2),
  Media_Salarios Number(10,2),
  Total_Salarios Number(10,2));

CREATE TABLE Tabla_Empleado(
  Num_Empleado Number(4) PRIMARY KEY,
  Nombre_Empleado VARCHAR(25),
  Categoria VARCHAR(10), -- Gerente, Comercial, ...
  Jefe Number(4),
  Fecha_Contratacion DATE,
  Salario Number(7,2),
  Comision Number(7,2),
  Num_Depart Number(2),
  FOREIGN KEY (Jefe) REFERENCES Tabla_Empleado,
  FOREIGN KEY (Num_Depart) REFERENCES Tabla_Departamento);
```

a) Construya un bloque que calcule el presupuesto del departamento para el año próximo. Se almacenará el mismo en la tabla **Tabla_Departamento** en la columna **Presupuesto**. Hay que tener en cuenta las siguientes subidas de sueldo:

Gerente	+	20%
Comercial	+	15%

Los demás empleados que no estén en ninguna de las categorías anteriores se les subirá el sueldo un 10%.

- b) Construya un bloque que actualice el campo **Total_Salarios** y el campo **Media_Salarios** de la tabla **Tabla_Departamento**, siendo el total la suma del salario de todos los empleados, igualmente con la media. Para ello:
- o Cree un cursor **C1**, que devuelva todos los departamentos
 - o Cree un cursor **C2**, que devuelva el salario y el código de todos los empleados de su departamento.

9. Cree una tabla NOTAS con los atributos que estime necesarios y construya un bloque que inserte en la tabla NOTAS cuatro filas para cada alumno matriculado, estas filas corresponderán a las tres convocatorias ordinarias y la última para la convocatoria extraordinaria de junio. Antes de insertar se comprobará que no están ya creadas. Las filas deberán inicializarse a nulo.

10. Cree un bloque que almacene en la tabla AUX_ARTICULOS (debe crearse previamente) un número dado de los artículos con mayor precio de los existentes en la tabla Tabla_Articulos. El número de artículos a almacenarse debe ser dado por teclado.

11. Escriba un bloque que recupere todos los proveedores por países. El resultado debe almacenarse en una nueva tabla Tabla_Aux que permita almacenar datos del tipo: ***Proveedor: ONCE – País: España***

Utilice un cursor para recuperar cada país de la tabla Tabla_Proveedores y pasar dicho país a un cursor que obtenga el nombre de los proveedores en él. Una vez que se tiene el nombre del proveedor y su país, debe añadirse a la nueva tabla en el formato especificado.

NOTA: En primer lugar, debe crear las tablas Tabla_Proveedores y Tabla_Aux e insertar en las mismas datos de prueba.

12. Crea un cursor que obtenga por pantalla los siguiente datos de vuelos del modelo AEROPUERTO: Nombre aeropuerto de origen, nombre aeropuerto de destino, director de cada uno si lo tuviera (sino que aparezca en blanco), año que se ha realizado y número de veces que se efectúan ese mismo vuelo por cada año.



6- CONTROL DE EXCEPCIONES

EXCEPCIONES EN PL/SQL

- ¿**Qué es una excepción?**
 - Identificador PL/SQL que surge durante la ejecución provocado por un error.
- ¿**Cómo surge?**
 - Se produce un error Oracle.
 - Es provocado explícitamente por el programador.
- ¿**Como se gestiona?**
 - Tratándola con un manejador en el propio bloque.
 - Propagándola al proceso padre.

Gestión de excepciones en PL/SQL

Captura de una excepción:

Cuando se produce una excepción en la sección ejecutable, esta se ramifica a la zona de EXCEPTION, donde se puede capturar y tratar programando su correspondiente manejador.

Propagación de una excepción:

Cuando se produce una excepción en un bloque y no hay el correspondiente manejador para ella, el bloque termina con un error. Podremos gestionarla en el entorno de llamada de dicha bloque.

Tipos de excepciones.

- Del servidor Oracle. Provocadas implícitamente.
 - Con nombre. Ej: NO_DATA_FOUND
 - Sin nombre. ORA-12560
- Definidas por el usuario. Provocadas explícitamente
 - Con nombre. (se declaran previamente)
 - Sin nombre. (RAISE_APPLICATION_ERROR)

Manejadores de excepciones. Sintaxis.

EXCEPTION

WHEN exception1 [**OR** exception2...] **THEN**

instruccion1;

instruccion2;

[.....]

WHEN OTHERS THEN

instruccion3;

instruccion4;]

Reglas para interrumpir excepciones

- EXCEPTION, inicia la sección de gestión de excepciones.
- Se permiten varios manejadores de excepciones.
- Solo se procesa un manejador de excepciones antes de salir del bloque.
- WHEN OTHERS es la última cláusula

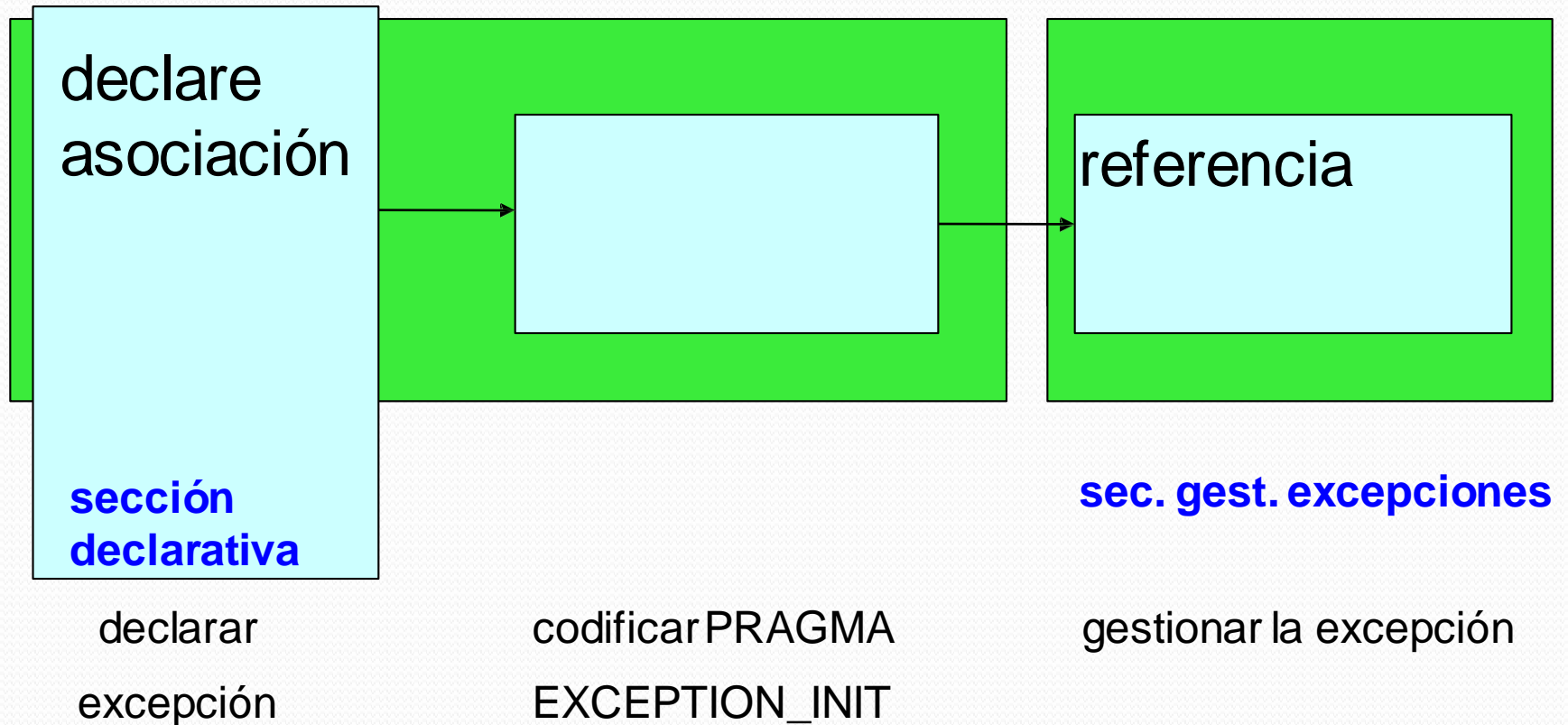
PREDEFINIDAS POR ORACLE Y CON NOMBRE. EJEMPLOS.

Excepcion	Nº error	Descripción
NO_DATA_FOUND	ORA-01403	SELECT NO DEVUELVE DATOS
TOO_MANY_ROWS	ORA-01422	SELECT DEVUELVE MÁS DE 1 FILA
INVALID_CURSOR	ORA-01001	OPERACIÓN ILEGAL DE CURSOR
ZERO_DIVIDE	ORA-01476	DIVIDIR POR 0
DUP_VAL_ON_INDEX	ORA-01017	INSERTAR UN REGISTRO DUPLICADO

Manejando excepciones con PL/SQL

- ☐ Una excepción es un identificador en PL/SQL que es levantada durante la ejecución.
- ☐ ¿Como es levantada?
 - ☐ Un error Oracle ocurre
 - ☐ Se levanta explícitamente
- ☐ ¿Como se manipulan los errores?
 - ☐ Atraparlo con un manipulador
 - ☐ Propagarlo en un entorno de invocación

Predefinidas de ORACLE, sin nombre. Método 1



Predefinidas de ORACLE, sin nombre. Método 1.Ejemplo

DECLARE

// declaración y asociación de la excepción

e_restri_integri_empEXCEPTION;

PRAGMAEXCEPTION_INIT(e_restri_integri_emp, -2292);

v_deptno dept.deptno%type:=10;

BEGIN

DELETE FROM dept WHERE deptno = v_deptno;

COMMIT;

EXCEPTION

WHEN e_restri_integri_emp THEN

 dbms_output.put_line('no se puede borrar depto,
 existen empleados');

END;

Predefinidas de ORACLE, sin nombre. Método 2

Cuando se produce una excepción sin nombre, hay dos pseudovariantes que almacenan información del error:

SQLCODE: N° del código del error.

SQLERRM: Mensaje asociado al n° del error.

Así, se pueden gestionar en el manejador WHEN OTHERS:

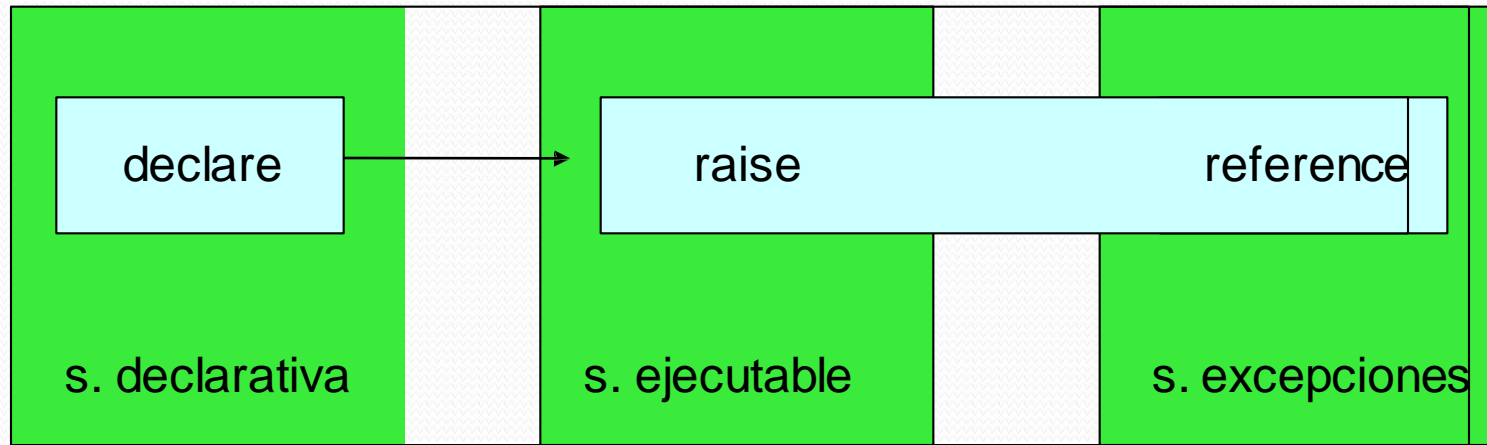
```
WHEN OTHERS THEN
```

```
IF SQLCODE= - 2292 THEN
```

```
    dbms_output.put_line ('Error de integridad referencial');
```

```
END IF;
```

DEFINIDAS POR EL USUARIO, CON NOMBRE



Se especifica la
excepción

Se provoca
explícitamente
la excepción
con RAISE

Se gestiona la
excepción

Definidas por el usuario, con nombre. Ejemplo

```
DECLARE
    e_invalid_product    EXCEPTION;
BEGIN
    UPDATE product
    SET descript = p_descripcion_product
    WHERE prodid = p_product_number;
    IF SQL%NOTFOUND THEN
        RAISE e_invalid_product;
    END IF;
    COMMIT;
EXCEPTION
    WHEN e_invalid_product THEN
        DBMS_OUTPUT.PUT_LINE('nº del producto invalidado');
END;
```


Funciones para atrapar excepciones

```
DECLARE

    v_error_code      NUMBER;
    v_error_message   VARCHAR2(255);

BEGIN
    ...
EXCEPTION
    ...

    WHEN OTHERS THEN
        ROLLBACK;
        v_error_code := SQLCODE;
        v_error_mesage := SQLERRM;
        INSERT INTO errors
        VALUES (v_error_code , v_error_message);

END;
```

Definidas por el usuario, sin nombre

RAISE_APPLICATION_ERROR

- Procedimiento que permite emitir mensajes de error definidos por el usuario desde subprogramas almacenados.
- Solo se puede llamar desde un subprograma almacenado.
- Se usa en la sección ejecutable o en la de excepciones.
- Devuelve condiciones de error al usuario de forma consistente con otros errores del servidor Oracle.

Sintaxis:

`RAISE_APPLICATION_ERROR(num_error, mensaje);`
donde num_error entre -20000 y -20999

Definidas por el usuario, sin nombre
RAISE_APPLICATION_ERROR. Ejemplo.

.....

EXCEPTION

```
WHEN NO_DATA_FOUND THEN
```

```
    RAISE_APPLICATION_ERROR (-20201, 'manager, no es un  
    empleo válido,');
```

.....

```
DELETE FROM emp
```

```
WHERE mgr = v_mgr;
```

```
IF SQL%NOTFOUND THEN
```

```
    RAISE_APPLICATION_ERROR(-20202, 'este no es un jefe válido');  
END IF;
```

.....

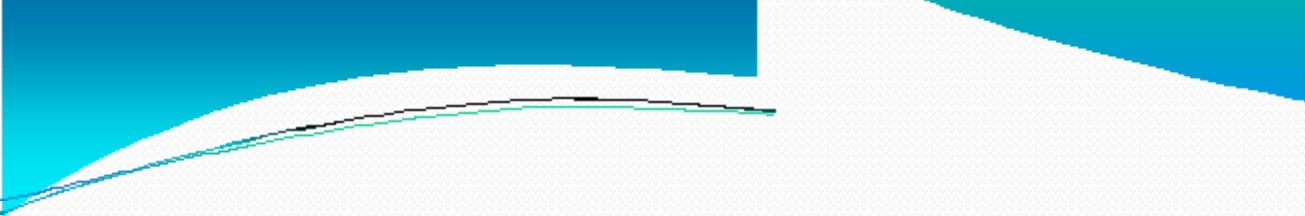
Propagando excepciones

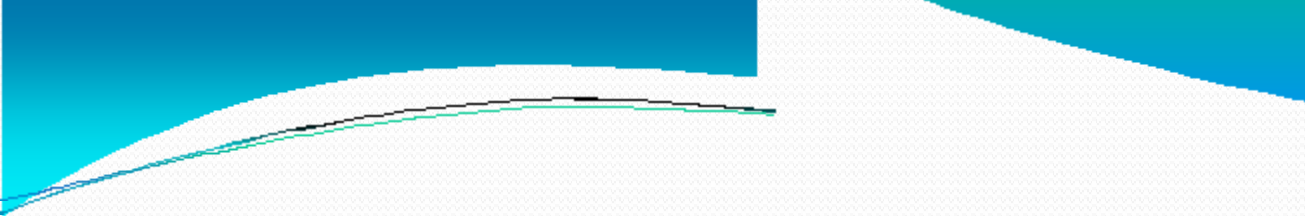
```
DECLARE
    e_no_rows EXCEPTION;
    e_integrity EXCEPTION;
    PRAGMA EXCEPTION_INIT (e_integrity, -2292)
BEGIN
    FOR c_record IN emp_cursor LOOP
        BEGIN
            SELECT ...
            UPDATE ...
            IF SQL%NOTFOUND THEN
                RAISE e_no_rows;
            END IF;
        END;
    END LOOP;
    EXCEPTION
    WHEN e_integrity THEN ...
    WHEN e_no_rows THEN ...
END;
```



Actividades

- ☐ 1– Realiza el ejercicio 6 de funciones, usando excepciones.
- ☐ 2- Realiza el ejercicio anterior dando un mensaje de error en caso de que ya exista. (Sin hacer el update)
- ☐ 3- Realiza el ejercicio anterior dando un mensaje de error en caso de que no existe. (Sin hacer el insert)
- ☐ 4- Provoca un error para que salte una excepción y captúrala antes de salir de la función.
- ☐ 5- Anida todas las excepciones posibles de los ejercicios anteriores en la misma función.

- 
- 6-Muestra el número de productos, de forma que indique “No hay productos” en vez de “Hay 0 productos”.
 - 7- Crea un bloque PL/SQL en el que insertas un producto con código 3, ya existente. Haz el control de excepciones de forma que un error de clave primaria duplicada se ignore (no se inserta el producto pero el bloque PL/SQL se ejecuta correctamente). Si intentamos insertar un producto con precio negativo, debe aparecer un error

- 
- 8- Escribe un bloque PL/SQL donde usas un registro para almacenar los datos de un producto. El bloque insertará ese producto, pero si el precio no es positivo generará una excepción de tipo `precio_prod_invalido`, que deberás declarar previamente, en vez de realizar la inserción.
 - Modifica el ejercicio anterior, de forma que ahora capture la excepción elevada y produces otra, con `SQLCODE` y mensaje de error específico, usando `RAISE_APPLICATION_ERROR`



7- INTRODUCCIÓN PL/SQL DINÁMICO

Consiste en

- PL/SQL ofrece la posibilidad de ejecutar sentencias SQL a partir de cadenas de caracteres. Para ello debemos emplear la instrucción **EXECUTE IMMEDIATE**.
- Podemos obtener información acerca de número de filas afectadas por la instrucción ejecutada por **EXECUTE IMMEDIATE** utilizando **SQL%ROWCOUNT**.

Ejemplo

```
□ DECLARE
ret NUMBER;
FUNCTION fn_execute RETURN NUMBER IS
sql_str VARCHAR2(1000);
BEGIN
sql_str := 'UPDATE DATOS SET NOMBRE = "NUEVO
NOMBRE"
WHERE CODIGO = 1';
EXECUTE IMMEDIATE sql_str;
RETURN SQL%ROWCOUNT;
END fn_execute;
BEGIN
ret := fn_execute();
dbms_output.put_line(TO_CHAR(ret));
END;
```

- Podemos además parametrizar nuestras consultas a través de variables host. Una variable host es una variable que pertenece al programa que está ejecutando la sentencia SQL dinámica y que podemos asignar en el interior de la sentencia SQL con la palabra clave **USING** . Las variables host van precedidas de dos puntos

- **DECLARE**

ret **NUMBER**;

FUNCTION fn_execute (nombre **VARCHAR2**, codigo **NUMBER**)

RETURN NUMBER

IS

sql_str **VARCHAR2**(1000);

BEGIN

sql_str := 'UPDATE DATOS SET NOMBRE =:new_nombre

WHERE CODIGO = :codigo';

EXECUTE IMMEDIATE sql_str **USING** nombre, codigo;

RETURN SQL%ROWCOUNT;

END fn_execute ;**BEGIN**

ret := fn_execute('Devjoker',1);

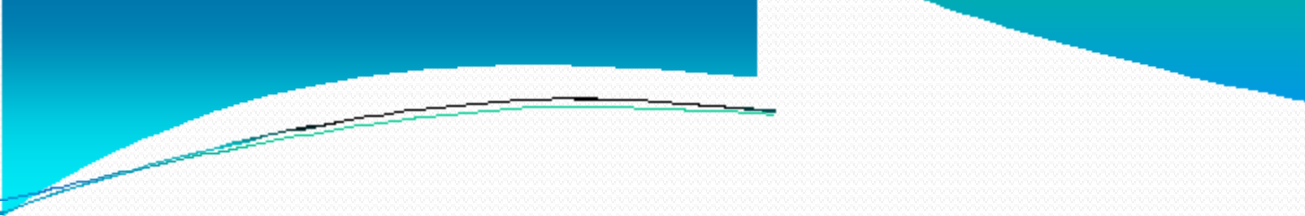
dbms_output.put_line(**TO_CHAR**(ret));

END;



Ejercicios

- ☐ 1- Crea un procedimiento que dado un nombre y una contraseña te cree un usuario y te de permisos.
- ☐ 2- Cambia el ejercicio anterior capturando todos los fallos posibles que se puedan dar.
- ☐ 3- Crea dinámicamente un tabla de 5 campos, y creale una clave primaria
- ☐ 4- Realiza un procedimiento llamado Busqueda, que entren por parámetros los 5 campos dados y busque dinámicamente todos los datos de la tabla anterior por todo los criterios que no se le pase a NULL a la función, dando un error si todos se pasan a NULL.

- 
- 5- Crea la tabla usando SQL dinámico y EXECUTE IMMEDIATE dentro de un bloque PL/SQL. Comprueba que efectivamente la tabla fue creada, con una consulta simple sobre el catalogo de Oracle. Finalmente, elimina (DROP) la tabla
 - 6- De nuevo en un bloque PL/SQL, crea una tabla T2 con un campo S alfanum' erico, e inserta una fila en esa tabla
 - 7- Recupera e imprime de nuevo los datos de todos los productos, ahora usando SQL dinámico
 - 8- Altera la tabla de productos, añadiendo una restricción c_precio_positivo que compruebe que el precio de un producto es mayor que cero