

Unidad 6: Lenguaje PL/SQL

OBJETIVOS

- Adquirir una visión general del PL/SQL
- Conocer sus principales características
- Saber cuales son las posibilidades y limitaciones
- Manejar la estructura básica del lenguaje: el bloque y reconocer sus componentes
- Servirse del SQL*Plus para la edición, depuración y ejecución de programas sencillos en PL/SQL
- Conocer los tipos de datos que soporta el lenguaje, y como declararlos

Unidad 6: Lenguaje PL/SQL

OBJETIVOS

- Manejar operadores y funciones predefinidas
- Conocer las estructuras de control y diseñar programas
- Conocer los distintos tipos de datos compuestos: Registros, tablas.
- Utilizar cursores explícitos e implícitos para procesar la información contenida en la base de datos
- Diseñar programas capaces de recuperarse ante los posibles errores que puedan aparecer utilizando las excepciones
- Realizar procedimientos y funciones para desarrollar programas

Unidad 6: Lenguaje PL/SQL

OBJETIVOS

- Usar parámetros de distintos tipos
- Crear disparadores de bases de datos que se ejecuten automáticamente al modificar la tabla a la que han sido asociados
- Usar paquetes para almacenar los programas, así como otros objetos utilizados en las aplicaciones
- Manejar los paquetes suministrados por Oracle
- Construir programas que permitan crear nuevos objetos de bases de datos y modificar las características de los existentes utilizando los paquetes que permiten superar las limitaciones del SQL estático

Unidad 6:

Lenguaje PL/SQL

1. Introducción
2. Características del lenguaje
3. Interacción con el usuario en PL/SQL
4. Arquitectura
5. Bloques anónimos y procedimientos
6. Tipos básicos de datos
7. Identificadores, variables y operadores
8. Funciones
9. Estructuras de control

Unidad 6: Lenguaje PL/SQL

10. Tipos de datos compuestos: Registros, Tablas
11. Cursores. Tipos
12. Excepciones
13. Subprogramas: Procedimientos y Funciones
14. Triggers
15. Paquetes
16. SQL dinámico

Introducción

INTRODUCCIÓN

- Limitaciones del SQL: todos los usuarios debe conocer los comandos
- PL/SQL es una solución para usuarios que no conocen SQL
- Existencia de un gestor PL/SQL incorporado por el servidor de la BD y en las herramientas del PL/SQL (Forms, Reports, Graphics,.....)
- Los programas en PL/SQL son un objeto más en la base de datos, y como tal puede ser utilizado por cualquier usuario que tenga el correspondiente privilegio
- Los programas se ejecutan en el servidor

Características del Lenguaje

CARACTERÍSTICAS

- Es un lenguaje procedimental diseñado por Oracle para trabajar con la BD
- Incluido en el servidor y en algunas herramientas del cliente
- Soporta todos los comandos de consulta y manipulación de datos, añadiendo al SQL: estructuras de control y otros elementos propios de lenguajes procedimentales
- La unidad de trabajo es el BLOQUE
 - **BLOQUE:** *conjunto de declaraciones, instrucciones y mecanismos de gestión de errores y excepciones*

Características del Lenguaje

BLOQUES PL/SQL

- Es la estructura básica del PL/SQL
- Está formado por
 - **Zona de declaraciones:** donde se declaran los objetos (variables, constantes,...). Va precedido por DECLARE (o IS/AS en los procedimientos y funciones). Es opcional
 - **Un conjunto de instrucciones:** precedido por BEGIN
 - **Una zona de tratamientos de excepciones:** Precedido por EXCEPTION. Es opcional

Características del Lenguaje

BLOQUES PL/SQL

Formato:

```
[DECLARE  
    <declaraciones>  
BEGIN  
    sentencia1;  
    sentencia2;  
    .....  
[EXCEPTION  
    <gestión de excepciones>  
END;
```

Características del Lenguaje

BLOQUES PL/SQL

Ejemplo:

DECLARE

v_num_empleados NUMBER(2);

BEGIN

INSERT INTO depart VALUES (99,'PROVISIONAL',NULL);

UPDATE emple SET dept_no=99 where dept_no = 20;

v_num_empleados :=SQL%ROWCOUNT; --num filas afectadas

DELETE FROM depart WHERE dept_no= 20;

DBMS_OUTPUT.PUT_LINE (v_num_empleados || 'Empleados ubicados en PROVISIONAL');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE_APPLICATION_ERROR (-2000,'Error en la aplicación');

END;

Características del Lenguaje

DATOS COMPATIBLES CON SQL

- Dispone de datos compatibles con los utilizados en las columnas de las tablas (NUMBER, varchar2,...), además de otros propios como los BOOLEAN.
- Las declaraciones de datos se realizan en DECLARE

```
DECLARE
    importe number(8,2);
    contador NUMBER(2) DEFAULT 0;
    nombre VARCHAR2(25);
BEGIN
    ....
```

Características del Lenguaje

DATOS COMPATIBLES CON SQL

- Permite declarar una variable del mismo tipo que otra variable o que una columna de una tabla. Para ello se utiliza **%TYPE**

`v_nombreact` `empleados.nombre%TYPE`

- Se puede declarar una variable para guardar una fila completa de una tabla. **%ROWTYPE**

`v_mifila` `empleados %ROWTYPE`

Características del Lenguaje

ESTRUCTURAS DE CONTROL

Son las habituales : IF, WHILE, FOR, LOOP

ESTRUCTURAS DE CONTROL ALTERNATIVAS			
ALTERNATIVA SIMPLE	ALTERNATIVA DOBLE	ALTERNATIVA MULTIPLE (elsif)	ALTERNATIVA MULTIPLE (case)
IF <condición> THEN instrucción; END IF;	IF <condición> THEN instrucción;; ELSE instrucción;; END IF;	IF <condición1> THEN instrucción;; ELSIF <condicion2> THEN instrucción;; ELSIF <condicion3> THEN instrucción;; ELSE instrucción;; END IF;	CASE [expresión] WHEN <test1> THEN instrucciones1; WHEN <test2> THEN instrucciones2; WHEN <test3> THEN instrucciones3; [ELSE instrucciones n;] END CASE;

Características del Lenguaje

ESTRUCTURAS DE CONTROL

Son las habituales : IF, WHILE, FOR, LOOP

ESTRUCTURAS DE CONTROL REPETITIVAS		
MIENTRAS	PARA	ITERAR ... FIN ITERAR SALIR SI...
WHILE <condición> LOOP instrucciones;; END LOOP;	FOR variable IN <minimo>.. maximo> LOOP instrucciones;; END LOOP;	LOOP instrucciones;; EXIT WHEN <condicion>; instrucciones;; END LOOP;

Características del Lenguaje

ORDENES DE MANIPULACIÓN DE DATOS

- Desde SQL se puede ejecutar cualquier orden de manipulación de datos

....

```
DELETE from clientes WHERE nif = v_nif;
```

....

```
UPDATE PRODUCTOS SET STOCK := STOCK – Unidades_vendidas  
WHERE cod_producto= v_codigo;
```

.....

Características del Lenguaje

USO DE CURSORES

- En PL/SQL el resultado de una consulta no va directamente al terminal de usuario, sino a un área de memoria a la que se accede mediante una estructura denominada **CURSOR**.
- Los **CURSORES** sirven para guardar el resultado de una consulta.

```

SELECT COUNT (*) INTO v_numventas FROM ventas;

SELECT apellido, oficio INTO v_ape, v_ofi FROM emple WHERE
emp_no= 7900;
  
```


Características del Lenguaje

GESTIÓN DE EXCEPCIONES

- Las excepciones sirven para tratar errores
- Existen excepciones propias del ORACLE con algunos de los errores más frecuentes, que se disparan automáticamente al producirse los errores asociados:

NO_DATA_FOUND: producida cuando una **SELECT INTO** no ha devuelto ningún valor

TOO_MANY_ROWS: producida cuando una **SELECT INTO** devuelve más de una fila

Características del Lenguaje

EJEMPLO BLOQUE PL/SQL

DECLARE

```
v_ape VARCHAR2(10);  
v_ofi VARCHAR2(10);
```

BEGIN

```
SELECT apellido, oficio INTO v_ape, v_ofi FROM emple WHERE emp_no =7900;  
DBMS_OUTPUT.PUTLINE (v_ape||'*'||v_ofi);
```

EXCEPTION

```
WHEN NO_DATA_FOUND THEN
```

```
    insert into temp (col1) values ('ERROR no hay datos');
```

```
WHEN TOO_MANY_ROWS THEN
```

```
    insert into temp (col1) values ('ERROR demasiados datos');
```

```
WHEN OTHERS THEN
```

```
    RAISE_APPLICATION_ERROR (-20000,'Error en la aplicación');
```

```
END;
```

Características del Lenguaje

ESTRUCTURA MODULAR

Se distinguen los siguientes tipos de programas:

- **BLOQUES ANONIMOS:** No tienen ningún nombre y se utilizan poco. La zona de declaración comienza con **DECLARE**.
- **SUBPROGRAMAS:** Son bloques PL/SQL con un nombre. La zona de declaración comienza con la palabra **IS/AS**, y pueden ser:
 - **PROCEDIMIENTOS:** en cuyo caso la declaración irá precedida de la palabra **PROCEDURE**. Permite pasar y devolver más de un valor
 - **FUNCIONES:** similar a los procedimientos pero éstas pueden devolver solo un valor, y la declaración irá precedida de **FUNCTION**

Características del Lenguaje

ESTRUCTURA MODULAR

Se distinguen los siguientes tipos de programas:

ANONIMO	PROCEDIMIENTOS	FUNCIONES
[DECLARE] BEGIN ... [EXCEPTION] END;	PROCEDURE nombre IS BEGIN ... [EXCEPTION] END;	FUNCTION nombre RETURN tipo IS BEGIN ... [EXCEPTION] END;

Interacción con el usuario

DBMS_OUTPUT

- Oracle esta pensado para trabajar con la BD no con el usuario.
- El PL/SQL dispone de pocas órdenes para capturar datos por teclado o para visualizar datos en pantalla.
- Dispone, sin embargo de alguna instrucción para comunicarse con el usuario:
 - Dispone del paquete **DBMS_OUTPUT**: Con fines de depuración que contiene el procedimiento PUT_LINE para visualizar textos en pantalla.

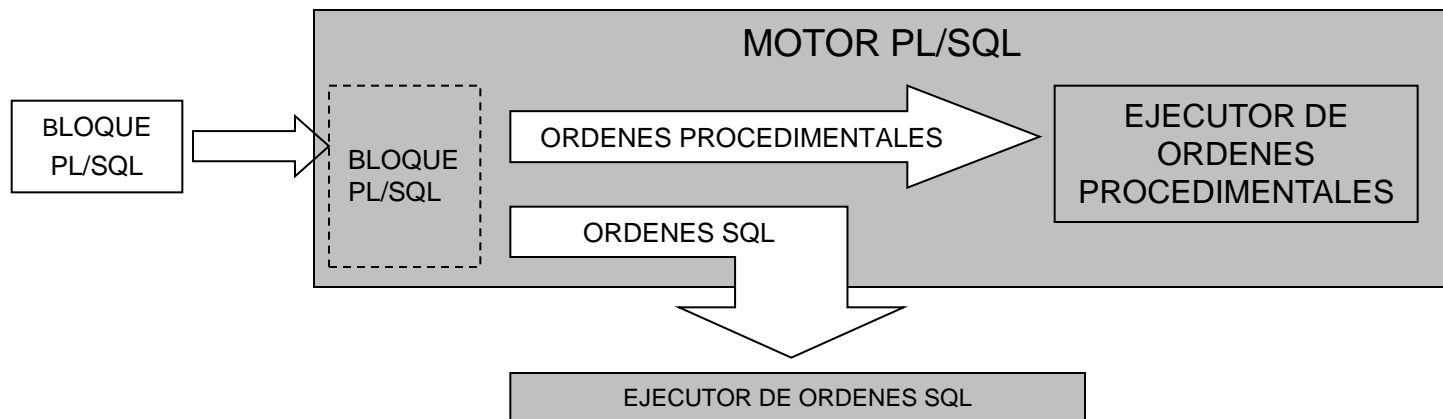
DBMS_OUTPUT.PUT_LINE (<expresion>)

Es necesario que la variable de entorno: SET SERVEROUTPUT ON

Arquitectura

Arquitectura

- PL/SQL está integrada en el servidor ORACLE junto con otras herramientas
- Es un motor o gestor capaz de ejecutar subprogramas y bloques PL/SQL en coordinación con el ejecutor de órdenes de SQL



Arquitectura

PL/SQL del servidor de ORACLE

Se pueden diferenciar tres tipos de bloques PL/SQL que trabajan con el motor del servidor:

- **BLOQUES PL/SQL ANONIMOS**
- **SUBPROGRAMAS ALMACENADOS**
- **DISPARADORES DE BASES DE DATOS.**

Arquitectura

PL/SQL del servidor de ORACLE

Se pueden diferenciar tres tipos de bloques PL/SQL que trabajan con el motor del servidor:

- **BLOQUES PL/SQL ANONIMOS:** generados con diversas herramientas: SQL*Plus,..., que se envían al servidor Oracle, donde serán compilados y ejecutados.

```
SQL> BEGIN  
        DBMS_OUTPUT.PUT_LINE('Hola mundo');  
END;
```


Arquitectura

PL/SQL del servidor de ORACLE

Se pueden diferenciar tres tipos de bloques PL/SQL que trabajan con el motor del servidor:

- **SUBPROGRAMAS ALMACENADOS:** procedimientos y funciones que se compilan y almacenan en la base de datos, donde quedarán disponibles para ser ejecutados.

```
CREATE OR REPLACE
PROCEDURE ver_depart(numdepart NUMBER)
AS
    v_dnombre    varchar2(14);
    v_loc        varchar2(14);
BEGIN
    select dnombre, loc INTO v_dnombre, v_loc from depart where dept_no=numdepart;
    DBMS_OUTPUT.PUT_LINE ('Num depart:'||numdepart ||
                           ' Nombre dep: ' || v_dnombre || ' Localidad: ' || v_loc);
END ver_depart;
```

Arquitectura

PL/SQL del servidor de ORACLE

Se pueden diferenciar tres tipos de bloques PL/SQL que trabajan con el motor del servidor:

- **DISPARADORES O TRIGGERS:** son programas almacenados en la base de datos que están asociados a un evento y que se ejecutan automáticamente cuando se detecta el evento asociado

```
CREATE OR REPLACE TRIGGER auditoria_borrado
  BEFORE DELETE
  ON emple FOR EACH ROW
BEGIN
  insert into TABLA_AUDITORIA values ('BORRADO EMPLEADO' , old.emp_no, old.apellido);
  DBMS_OUT_PUT.PUT_LINE( ' Se ha borrado el empleado '|| old.emp_no|| ' cuyo nombre es '||
  old.apellido);
END;
```

Arquitectura

PL/SQL DE LAS HERRAMIENTAS

- El ORACLE dispone de herramientas que contienen un motor PL/SQL capaz de procesar bloques PL/SQL, ejecutando las órdenes procedimentales en el cliente o donde se encuentra la herramienta, enviando solo las instrucciones SQL al servidor.
- Herramientas que contienen motor PL/SQL: FORMS, REPORTS, GRAPHICS,...

Observaciones

DETECCION DE ERRORES AL COMPILAR PL/SQL

- Cuando se escribe procedimiento, si al ejecutarlo el compilador encuentra un error, mostrará el correspondiente aviso. Para ver los errores:

SHOW ERRORS

Ejercicios

Ejercicios. Guardar los ejercicios en una carpeta llamada subprogramas. Usar las tablas EMPLE y DEPART y asegurarse de que no existen datos duplicados como dos veces el empleado SALA, si es así volver a lanzar el script

1. Escribir un bloque PL/SQL que almacena en una variable tu nombre y escriba el texto 'Hola, me llamo.....'.
2. ¿Qué hace el siguiente bloque PL/SQL?. Indicar cual sería el cursor


```

      DECLARE
          V_num          NUMBER;
      BEGIN
          SELECT count(*) INTO v_num FROM EMPLE;
          DBMS_OUTPUT.PUT_LINE (v_num);
      END;
```
3. Introducir el bloque anterior, autenticándose como U4 y guardarlo en un fichero llamado EJER2_U4.SQL, dentro de una carpeta llamada SUPROGRAMAS
4. Ejecutar el bloque anterior y comprobar el resultado
5. Diseñar un bloque que nos muestre, a partir de la tabla EMPLE el departamento al que pertenece el empleado cuyo apellido es 'SALA'

Tipos de Datos Básicos

TIPOS DE DATOS

- Dispone de los mismos datos que SQL, además de otros propios

caracter	
CHAR(longitud)	Cadenas de longitud fija. Si no se indica "longitud" por defecto es 1
VARCHAR2(longitud)	Cadena longitud variable
LONG(longitud)	Similar a varchar2, almacena hasta 3Gb

Tipos de Datos Básicos

TIPOS DE DATOS

numéricos	
NUMBER(p,e)	Valores numéricos, donde p: total de dígitos y e:número de decimales
	PL/SQL dispone de subtipos de NUMBER: DECIMAL, NUMERIC, INTEGER, REAL, SMALLINT,... <i>Importe number(5,2);</i>
BINARY_INTEGER	Numérico entero que se almacena en memoria en formato binario para facilitar los cálculos. NATURAL (0...), POSITIVE (1..) Admite valores entre -2147483647 y + 2147483647 Ej. <i>Contador BINARY_INTEGER;</i>
PLS_INTEGER	Similar al anterior, pero con las siguientes ventajas: es más rápido y si se da desbordamiento se produce un error y se levanta la excepción. <i>Indice PLS_INTEGER;</i>

Tipos de Datos Básicos

TIPOS DE DATOS

otros	
BOOLEAN	Almacena valores TRUE, FALSE y NULL
DATE	Almacena fechas en formato estándar: 'dd-mm-yyyy'. También almacena la hora
ROWID	Almacena identificadores de fila

Identificadores

IDENTIFICADORES

- Se utilizan para nombrar objetos que intervienen en un programa: variables, constantes, cursores, excepciones, procedimientos, funciones, etiquetas, etc.
- Pueden tener hasta 30 caracteres.
- Deben empezar por una letra, seguido de letra, número, \$, #, _
- No existe diferencia entre mayúsculas y minúsculas

Ejemplo: v_num_meses, v_num_filas, v_apel1

Variables

VARIABLES

- Deben ser declaradas en la sección **DECLARE** antes de su uso

Formato:

<nom_variable> <tipo> [NOT NULL] [{:= |DEFAULT} <valor>];

- Si no se inicializa el valor es NULL

Ejemplo:

```
DECLARE
  importe    NUMBER(8,2);
  nombre     VARCHAR2(20) NOT NULL := 'MIGUEL';
  nombre     VARCHAR2(20) NOT NULL DEFAULT 'MIGUEL';
  nombre     VARCHAR2(20) DEFAULT 'MIGUEL';
```

.....

Variables

VARIABLES: Uso de %TYPE y %ROWTYPE

- **%TYPE**: Declara una variable del mismo tipo que otra o que una columna de una tabla

Ejemplo:

<code>total</code>	<code>importe%TYPE;</code>
<code>nom_moroso</code>	<code>clientes.nombre%TYPE;</code>

- **%ROWTYPE**: Declara un registro cuyos campos se corresponden con las columnas de una tabla o vista

Ejemplo:

<code>moroso</code>	<code>cliente%ROWTYPE;</code>
---------------------	-------------------------------

Constantes

CONSTANTES

- Se le deberá de asignar siempre un valor en la declaración
- Formato:
`<nombre_constante> CONSTANT <tipo>:= <valor>;`

Ejemplo

```
pct_iva  CONSTANT  REAL:=18;
```

Etiquetas

ETIQUETAS

- Son textos que se ponen para conseguir mayor legibilidad, o permitir acceder a determinados puntos del bloque con la instrucción GOTO

<<etiqueta>>

EJEMPLO:

<<principal>>

DECLARE

...

BEGIN

...

END **principal;**

Variables

ÁMBITO DE LAS VARIABLES

- El ámbito de una variable es el bloque en el que se declara y los bloques hijos de dicho bloque.
- La variable es local para el bloque en el que se declara y global para los bloques hijos
- Las variables declaradas en los bloques hijos no son accesibles desde el bloque padre
- Distintos bloques pueden tener identificadores iguales, por defecto el nombre del identificador referencia a la variable local. Para acceder a la variable del mismo nombre de un bloque padre, será necesario especificar la etiqueta del bloque al que pertenece dicha variable. **ETIQUETA.VARIABLE**

Variables

ÁMBITO DE LAS VARIABLES

```

DECLARE                                -----> BLOQUE PADRE
  v1                                     CHAR;
BEGIN
  ....
  v1:=1;

```

```

DECLARE                                -----> bloque hijo
  v2                                     CHAR;
  BEGIN
    v2:=2;
    ....
    v1:=v2;
    ....
  END;                                -----> fin bloque hijo

```

```

  V2:=V1;                                -----> error, v2 es desconocida en el bloque padre!!!!
END;                                    ----->FIN BLOQUE PADRE

```

Variables

ÁMBITO DE LAS VARIABLES

```

<<padre>>      -----> ETIQUETA PARA REFERENCIAR VARIABLES DE ESTE BLOQUE
DECLARE        -----> BLOQUE PADRE
  v  CHAR;
BEGIN
  ....
  v:=1;

```

```

  DECLARE -----> bloque hijo
    v  CHAR;

  BEGIN

    v:=padre.v; ---- > v referecia la variable local y padre.v la variable v del bloque padre
    ....

  END; -----> fin bloque hijo

```

```

END padre; ----->FIN BLOQUE PADRE

```


Variables

LITERALES

- Los literales de fecha y de caracteres deben estar entre comillas simples '....'

Ejemplo:

```
v_name      := 'Casimiro';  
v_apellido  := 'Novee Naa';
```

Código de Comentarios

COMENTARIOS

- Si es en una línea precedido de dos guiones (--) a modo de prefijo
- Para comentarios de más de una línea, irá en los símbolos /*
... */

```
v_sal NUMBER(9,2);  
BEGIN  
    /* calcula el salario anual a partir del salario mensual  
       del usuario */  
    v_sal:=v_sal*12;  
END; -- Este es el final del bloque
```

Funciones SQL en PL/SQL

FUNCIONES SQL en PL/SQL

- La mayoría de las funciones de SQL son válidas en las expresiones de PL/SQL
- No están disponibles las funciones **GREATEST**, **LEAST**, **DECODE** en las sentencias procedurales de PL/SQL
- Igual sucede para las funciones de grupo (**avg**, **count**, **max**, **min**, **sum**,...) solo se pueden utilizar dentro del PL/SQL en sentencias de SQL, no en sentencias procedurales

Ejemplo:

```
v_nombre:=upper(v_nombre);
```

```
Select count(*) into v_total from emple where dept_no=30;
```

Conversión de Tipos

CONVERSION DE TIPOS

- Las funciones de conversión son las mismas que SQL:
 - TO_CHAR
 - TO_DATE
 - TO_NUMBER

Ejemplo

```
begin
  select to_char(fcha_alta,'dd MON YYYY') from emple;
end;

v_fecha:=TO_DATE('January 13, 1998','Month DD, YYYY');
```

ANEXO

ENTRADA DE DATOS ACCEPT y &

Formato:

ACCEPT variable PROMPT 'mensaje'

- No puede usarse dentro del cuerpo del bloque
- Una vez introducido la variable, para poder utilizar en el bloque tenemos que referenciarla

'&variable'

→ si es texto

&variable

→ en el caso de que sea numérico

ACCEPT nombre PROMPT 'Introduce el nombre:'

ACCEPT edad PROMPT 'Introduce tu edad'

Begin

dbms_out.put_line ('Tu nombre es : ' || '&nombre');

dbms_out.put_line ('Tu edad es : ' || &edad);

End;

Ejemplos

Ejemplos

```
SET SERVEROUTPUT ON
```

```
=====
```

```
ACCEPT nom PROMPT 'Introduce el nombre'
```

```
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE ('El nombre es : ' || ' &nom');
```

```
END;
```

```
=====
```

```
ACCEPT p_annual_sal PROMPT 'Introducir el salario anual';
```

```
DECLARE
```

```
    v_sal NUMBER(9,2) := &p_annual_sal;
```

```
BEGIN
```

```
    v_sal:=v_sal/12;
```

```
    DBMS_OUTPUT.PUT_LINE('Salario Mensual : ' || TO_CHAR(v_sal));
```

```
END;
```

Ejercicios

Ejercicios

- Indicar los errores que aparecen en las siguientes instrucciones y la forma de corregirlos:

```

DECLARE
    num1                number(8,2):=0
    num2                number(8,2) NOT NULL DEFAULT 0;
    num3                NUMBER(8,2) NOT NULL;
    cantidad            INTEGER(3);
    precio, descuento   NUMBER(6);
    num4                num1%ROWTYPE;
    dto                 CONSTANT      INTEGER;
BEGIN
    .....
END;
```

Ejercicios

Ejercicios

- Indicar los errores que aparecen en las siguientes instrucciones y la forma de corregirlos:

```

DECLARE
    num1          number(8,2):=0 ;
    num2          number(8,2) NOT NULL DEFAULT 0;
    num3          NUMBER(8,2) NOT NULL DEFAULT 0;
    cantidad      INTEGER;
    precio        NUMBER(6);
    descuento     NUMBER(6);
    num4          num1%TYPE;
    dto           CONSTANT    INTEGER:= 5;

BEGIN
    .....
END;
```


Ejercicios

Ejercicios. Guardar cada ejercicio con nombre u6_ejer_xx.sql

2. Introducir tu nombre por teclado y tu edad y visualizar el siguiente mensaje:
Hola, te llamas xxxxxxxx y tienes XX años
3. Introducir tu nombre por teclado y la fecha de nacimiento y visualizar el siguiente mensaje:
Hola, te llamas xxxxxxxx y tienes XX años
4. Introducir el apellido de un empleado por teclado y visualizar su salario. Probar para el usuario SALA y para el usuario ARROYO
5. Visualizar el nombre y el salario del empleado cuyo emp_no es 7521. El dato se introducirá por teclado.
6. Suma de los salarios de todos los miembros de un departamento que se introducirá por teclado

Ejercicios

Ejercicios

- Introducir tu nombre por teclado y tu edad y visualizar el siguiente mensaje:

Hola, te llamas xxxxxxxx y tienes XX años

```
ACCEPT      nom      PROMPT 'Introduce el nombre';
Accept      edad     PROMPT 'Introduce tu edad';
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hola, te llamas '||&nom' || ' y tienes '||&edad||' años');
END;
```

Ejercicios

Ejercicios

- Introducir tu nombre por teclado y la fecha de nacimiento y visualizar el siguiente mensaje:

Hola, te llamas xxxxxxxx y tienes XX años

```
alter session set nls_date_format='dd/mm/yyyy';
```

```
ACCEPT          nom          prompt 'Introduce el nombre';
```

```
Accept          fcha_nto     PROMPT 'Introduce año de nto ddmmyyyy';
```

```
Declare
```

```
    edad         number;
```

```
begin
```

```
    edad:= floor(MONTHS_BETWEEN(sysdate,to_date('&fcha_nto','ddmmyyyy'))/12);
```

```
    DBMS_OUTPUT.PUT_LINE('Hola, te llamas '||&nom' || ' y tienes '||edad||' años');
```


```
end;
```

Ejercicios

Ejercicios

4. Introducir el apellido de un empleado por teclado y visualizar su salario. Probar para el usuario SALA y para el usuario ARROYO

```
define apel=null;
accept apel    PROMPT 'Introduce el apellido del empleado';
declare
    v_salario EMPLE.SALARIO%TYPE;
begin
    select salario into v_salario from emple where apellido='&apel';
    DBMS_OUTPUT.PUT_LINE('El salario de '|| '&apel '|| 'es ' || v_salario);
end;
```




Ejercicios

Ejercicios

5. Visualizar el nombre y el salario del empleado cuyo emp_no es 7521. El dato se introducirá por teclado.

```
ACCEPT cod_emp          prompt 'Introduce el codigo del empleado';
Declare
    v_nom                emple.APELLIDO%type;
    v_salario             emple.SALARIO%type;
begin
    select apellido,salario into v_nom,v_salario from emple where emp_no=&cod_emp;
    DBMS_OUTPUT.PUT_LINE('Cod_emple:'||&cod_emp || ' Nombre:'||v_nom||'
                          Salario: '||v_salario);
end;
```




Ejercicios

Ejercicios

6. Suma de los salarios de todos los miembros de un departamento que se introducirá por teclado

```
ACCEPT departamento    prompt 'Introduce el codigo de departamento';
Declare
    v_sum_salario      number;
begin
    select sum(salario) into v_sum_salario from emple where
    dept_no=&departamento;
    DBMS_OUTPUT.PUT_LINE('La suma de salarios del departamento
    '||&departamento || ' es ' || to_char(v_sum_salario,'999G999G999D99'));
end;
```



Ejercicios

Ejercicios

- Diseñar un bloque de programa que muestre el emp_no que le correspondería a un nuevo empleado, sabiendo que sería siguiente valor al mayor de todos los emp_no de la tabla EMPLE.
- Bloque que calcule cual es el departamento cuyo salario medio es menor. Visualizar el dept_no, y su salario medio
- Diseñar un bloque PL/SQL que permita insertar una fila con la siguiente información:
 - Emp_no: siguiente valor al mayor de todos los emp_no
 - Apellido: se introducirá por teclado tu nombre
 - Oficio: ANALISTA y se introducirá por teclado
 - Dept_no y salario: se calcularán a partir del ejercicio anterior
 - Jefe (DIR): emp_no del DIRECTOR de departamento donde se dará de alta
 - Fecha de alta: la fecha actual
 - El resto de los campos se dejan en blanco

Ejercicios

Ejercicios

- Diseñar un bloque en PL/SQL que permita incrementar el salario en un tanto por ciento que se introducirá por teclado para tu usuario.
- Modificar el programa para que se incremente el salario a todos los empleados de un departamento.
- Bloque que permita eliminar un usuario de la tabla. El usuario se deberá de introducir por teclado, a partir del campo que se considere oportuno. Mostrar la información del usuario que se borra. Comprobar el proceso con tu usuario.
- Bloque PL/SQL que muestre el departamento que más empleados tiene

Operadores

OPERADORES

Asignación	:=
Lógicos	AND, OR, NOT
Concatenación	
Comparación	=, !=, <,>,<=, >=, IS NULL, BETWEEN, LIKE, IN
Aritméticos	+, -, *, /, **

Operadores

ORDEN DE PRECEDENCIA EN LOS OPERADORES

Prioridad	Operador
1	** , NOT
2	* , /
3	+ , - ,
4	= , != , < , > , <= , >= , IS NULL , LIKE , BETWEEN , IN
5	AND
6	OR

Estructuras de Control

ESTRUCTURAS DE CONTROL

- Requieren evaluar una condición que puede dar como resultado: TRUE, FALSE o NULL.
- Cuando se cumple la condición devuelve TRUE, en caso contrario pues ser FALSE o NULL
- Permiten la utilización de etiquetas para facilitar la legibilidad

Estructuras de Control

ESTRUCTURAS DE CONTROL

- **ALTERNATIVA SIMPLE**

```
IF <condición> THEN  
    instrucción;
```

```
    ....  
END IF;
```

- **ALTERNATIVA DOBLE**

```
IF <condición> THEN  
    instrucción;
```

```
    ....;
```

```
ELSE
```

```
    instrucción;
```

```
    ....;
```

```
END IF;
```

Estructuras de Control

ESTRUCTURAS DE CONTROL

– ALTERNATIVA MULTIPLE (elsif)

```
IF <condición1> THEN
    instrucción;
    ....;
ELSIF <condicion2> THEN
    instrucción;
    ....;
....
[ELSE
    instrucción;
    ....;]
END IF;
```

Estructuras de Control

ESTRUCTURAS DE CONTROL

– ALTERNATIVA MULTIPLE (case)

```
CASE [<expresión>]
  WHEN <test1> THEN
    instrucciones1;
    ....;
  WHEN <test2> THEN
    instrucciones2;
    ....;
  [ELSE
    instrucciones;
    ....; ]

END CASE;
```

Estructuras de Control

ESTRUCTURAS DE CONTROL

- ITERAR... SALIR SI.... FIN ITERAR

```
LOOP
  instrucción;
  .....
  IF <condición> THEN
    EXIT;
  END IF;
  instrucción;
  .....
END LOOP;
```

Estructuras de Control

ESTRUCTURAS DE CONTROL

– MIENTRAS

```
WHILE <condicion> LOOP
    instrucción;
    ....
END LOOP;
```


Estructuras de Control

ESTRUCTURAS DE CONTROL

- **PARA:** donde la *var_control* se declara de manera implícita como var BINARY_INTEGER

```
FOR <var_control> IN <valor_inicial>..
```

Otra versión:

```
FOR <var_control> IN REVERSE<valor_inicial>..
```

Estructuras de Control

ESTRUCTURAS DE CONTROL

Ejemplos:

```
BEGIN
  FOR i IN 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;
END;
```

```
<<mibucle>>
LOOP
  ...
  LOOP
    ...
    EXIT mibucle WHEN ....
  END LOOP;
END LOOP miblucle;
```

Estructuras de Control

EJERCICIOS de estructuras repetitivas y de control

1. Crear una tabla llamada T_NUMEROS con la siguiente estructura:
numero number(3)
2. Diseñar un bloque PL/SQL que permita insertar en la tabla desde el número 1 hasta el 10.
3. Modificar el programa anterior para que solo inserte los valores múltiplos de 3 entre 11 y 30
4. Crear un bloque PL/SQL que modifique el salario y la comisión de los empleados dividiéndolo por 10
5. Crear un bloque PL/SQL que para un número de empleado introducido por teclado, visualice, el número de empleado, el apellido, el sueldo (salario+comision) y un asterísco por cada 500€
6. Modificar el salario de un empleado que se introduce por teclado , en función del número de empleados que tiene a su cargo: (utilizar CASE)
 - Si no tiene ningún empleado a su cargo la subida será de 50€
 - Si tiene 1 empleado la subida será de 80€
 - Si tiene 2 empleados la subida será de 100€
 - Si tiene mas de 3 empleados la subida será de 110€Además si el empleado es presidente se incrementará en 30€

Estructuras de Control

EJERCICIOS

1. Crear una tabla llamada T_NUMEROS con la siguiente estructura:
numero number(3)

```
CREATE TABLE t_numeros(  
numero number(3)  
);
```

2. Diseñar un bloque PL/SQL que permita insertar en la tabla NUMEROS, los números desde el número 1 hasta el 10.

```
BEGIN  
  FOR i IN 1..10 LOOP  
    INSERT INTO t_numeros values (i);  
  END LOOP;  
  COMMIT;  
END;
```

Estructuras de Control

EJERCICIOS

3. Modificar el programa anterior para que inserte en la tabla los múltiplos de 3 entre 11 y 30

```
BEGIN
  FOR i IN 11..30 LOOP
    if mod(i,3)=0 then
      INSERT INTO t_numeros values (i);
    end if;
  END LOOP;
  COMMIT;
END;
```

Estructuras de Control

EJERCICIOS

4. Crear un bloque PL/SQL que modifique el salario y la comisión de los empleados dividiéndolo por 10

```
BEGIN
```

```
    update emple set salario=salario/100,comision=comision/10;
```

```
END;
```

Estructuras de Control

EJERCICIOS

5. Crear un bloque PL/SQL que para un número de empleado introducido por teclado, visualice, el número de empleado, el apellido, el sueldo (salario+comision) y un asterísco por cada 500€

```
ACCEPT cod_emp          prompt 'Introduce el codigo del empleado';
Declare
    v_nom                emple.APELLIDO%type;
    v_salario             emple.SALARIO%type;
    v_comision            emple.COMISION%TYPE;
    v_final               number;
begin
    select apellido,salario,comision into v_nom,v_salario,v_comision from emple where
    emp_no=&cod_emp;
    DBMS_OUTPUT.PUT_LINE('Cod_emple:'||&cod_emp || ' Nombre:'||v_nom||' Salario:
    '||v_salario || ' Comisión: '||v_comision||' Estrellas: ');
    v_final:=trunc((v_salario+nvl(v_comision,0))/500);
    FOR i IN 1..v_final LOOP
        DBMS_OUTPUT.PUT_LINE (*);
    END LOOP;
end;
```

Estructuras de Control

EJERCICIOS

6. Modificar el salario de un empleado en función del números de empleados a su cargo.... (1/2)

```
ACCEPT cod_emp  prompt 'Introduce el codigo del empleado';
Declare
    v_apel          emple.apellido%type;
    v_total_emp     number(2);
    v_aumento       number(7) default 0;
    v_oficio         emple.oficio%type;
    v_salario        emple.salario%type;

begin
    select apellido,salario, oficio  into  v_apel, v_salario, v_oficio from emple where emp_no=&cod_emp;
    if v_oficio = 'PRESIDENTE' then
        v_aumento := 30;
    end if;

    select count(*) into v_total_emp from emple where dir = &cod_emp;
```


Estructuras de Control

EJERCICIOS

6. Modificar el salario de un empleado en función del números de empleados a su cargo.... (2/2)

```

CASE v_total_emp
  when 0 then
      v_aumento:=v_aumento+50;
  when 1 then
      v_aumento:=v_aumento+80;
  when 2 then
      v_aumento:=v_aumento+100;
  else
      v_aumento:=v_aumento+110;
END CASE;
update emple set salario=v_aumento+v_salario where emp_no=&cod_emp;

DBMS_OUTPUT.PUT_LINE('Cod_emple:'||&cod_emp || 'Nombre:'||v_apel|| 'Oficio: '||v_oficio);
dbms_output.put_line('Salario original: '||v_salario || ' Numero de empleados a su cargo: '||v_total_emp||'
Incremento salario '||v_aumento);

end;
```

Sentencia NULL

SENTENCIA NULL

NULL es una sentencia ejecutable que no realiza ninguna acción.

Se utiliza cuando es necesario que exista una sentencia ejecutable en alguna cláusula que así lo exija.

Ej1.

```
IF condición THEN NULL;
  ELSE Sentencias;
END IF;
```

Ej2.

.....

EXCEPTION

```
  WHEN condición THEN .....
  WHEN condición THEN .....
  WHEN OTHERS THEN NULL;
```

END;

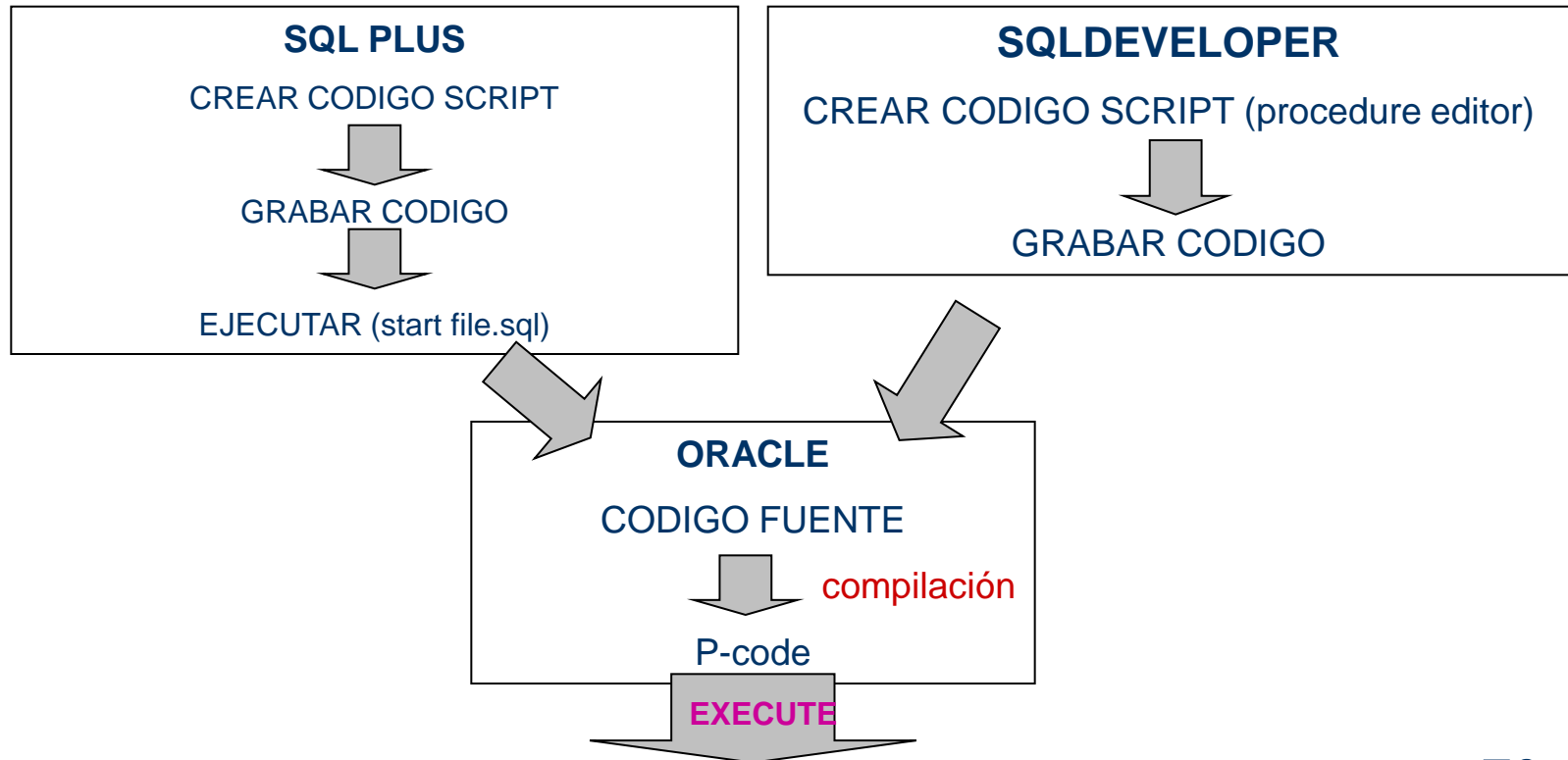
Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS

- Son bloques de PL/SQL que se identifican por un nombre y que realizan una acción.
- Se guardan en la BD con la extensión SQL y se pueden invocar desde cualquier subprograma o herramienta, siempre que se sea propietario o se tenga permiso de ejecución sobre él.
- Pueden ser de dos tipos:
 - PROCEDIMIENTOS
 - FUNCIONES
- Para ejecutar un subprograma:
`EXECUTE <nom_subprograma>[(<valores de parámetros>)]`

Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS

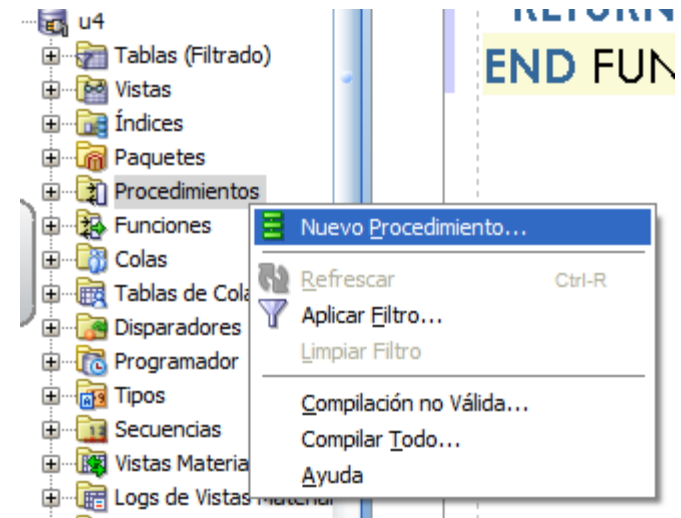


Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS. sqldeveloper

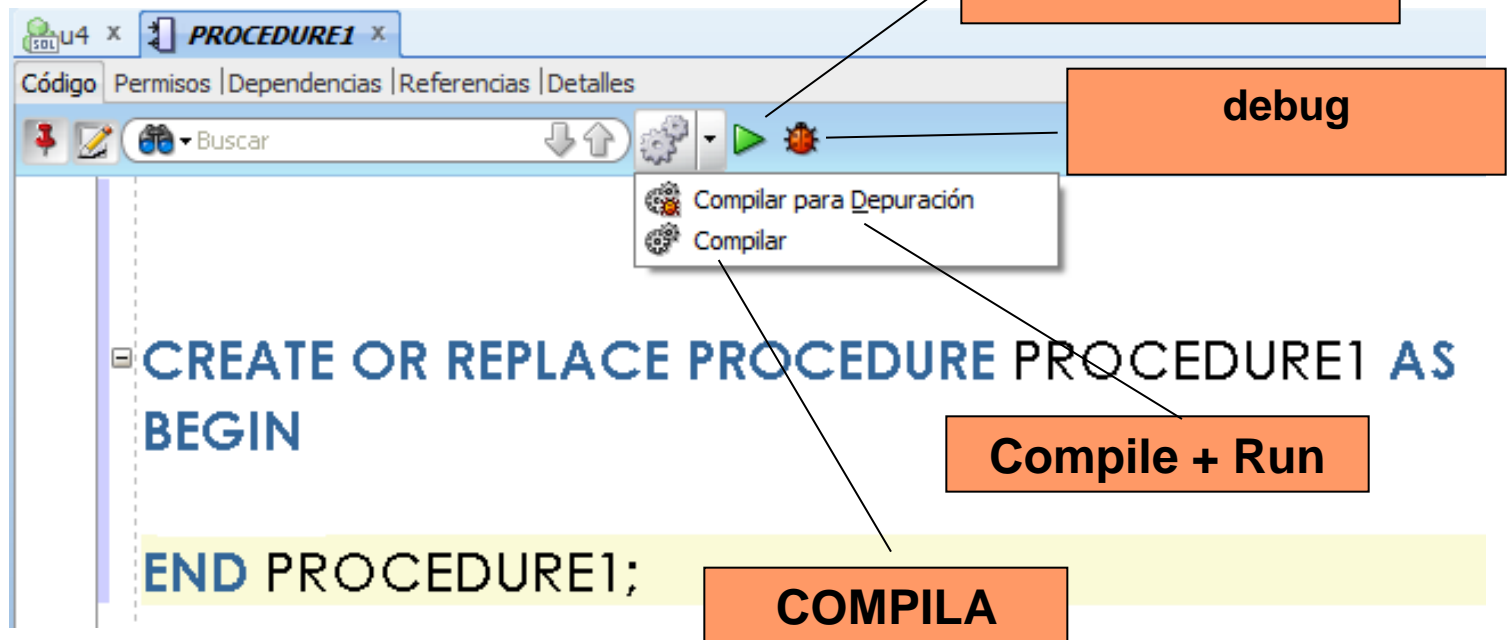
- Para crear un procedimiento o función en el sqlDeveloper, seleccionar el objeto a crear (procedimiento o función) y pulsando el botón izquierdo del ratón seleccionar nuevo procedimiento.
- Otra posibilidad es directamente escribiendo el comando

create o replace procedure



Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS. sqldeveloper

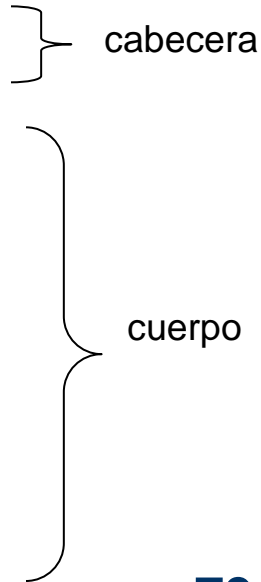


Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS

Tienen la siguiente estructura:

```
CREATE OR REPLACE  
  PROCEDURE <nom_procedimiento> [( <lista_parámetros> )]  
IS/AS  
  <declaraciones>  
  
BEGIN  
  instrucciones;  
  
EXCEPTION  
  <excepciones>;  
  
END [ <nom_procedimiento> ];
```



cabecera

cuerpo

Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS

- **REPLACE**: sustituye el procedimiento si ya existiera
- **[(<lista_parámetros>)]**
 - Son los parámetros de entrada o salida del procedimiento
 - Si existe más de un parámetro, estos van **separados por comas**
 - Si el procedimiento no tiene parámetros podemos omitir esta opción
 - El formato genérico es:

`<nom_variable> [IN | OUT | IN OUT] <tipo_dato> [{:=| DEFAULT}<valor>]`

- Al indicar los parámetros se debe de indicar **SOLO el tipo pero NO el tamaño**.
- Las opciones IN, OUT, IN OUT hacen referencia al tipo de parámetro (entrada, salida o entrada/salida)

Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS

- IS / AS
 <declaraciones>

Es el apartado donde se declararán las variables locales al procedimiento

Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS

- Los procedimientos NO pueden ser utilizados en expresiones

```
suma:= valor1 + mi_proced(param1, param2);
```

```
dbms_output.put_line('Nombre departamento es:'|| mi_proced(param1,param2));
```

Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS

- Los procedimientos NO pueden ser utilizados en expresiones
- Para crear un procedimiento o sustituir uno existente:


```
CREATE [OR REPLACE]
  PROCEDURE <nom_procedimiento>[(<lista_parámetros>)]
  .....;
  .....;
END [<nom_procedimiento>];
```

- Para volver a compilar un subprograma almacenado en la BD

```
ALTER  PROCEDURE nom_subprograma COMPILE;
```

Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS. CREACIÓN

SQL*PLUS	SQLDEVELOPER
<ol style="list-style-type: none"> 1. Introducir la sentencia CREATE PROCEDURE en un editor y salvarlo como un script (fichero .SQL) 2. Desde SQL*PLUS, ejecutar el script para almacenar el código fuente y compilar el procedimiento. 3. Usar SHOW ERRORS para ver los errores de compilación 4. Una vez compilado sin errores está preparado para ejecutarse y ser utilizado 	<ol style="list-style-type: none"> 1. Seleccionar el objeto Procedimiento en el esquema del usuario 2. Pulsar el botón izquierdo y crear nuevo procedimiento dándole nombre y agregando los parámetros si los conocemos ya 3. Compilar para comprobar errores de escritura y en su caso corregir 4. Si pinchamos  se abre una ventana con un bloque que contiene un programa con el procedimiento para que copiemos y lo probemos en una ventana nueva de edición

Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS

Ejemplo:

```
CREATE OR REPLACE
```

```
  PROCEDURE cambiar_oficio (num_empleado NUMBER, nuevo_oficio VARCHAR2)
```

```
AS
```

```
    old_oficio          emple.oficio%TYPE;
```

```
BEGIN
```

```
    SELECT oficio INTO old_oficio FROM EMPLE WHERE emp_no = num_empleado;
```

```
    UPDATE emple SET oficio=nuevo_oficio WHERE emp_no=num_empleado;
```

```
    DBMS_OUTPUT.PUT_LINE (num_empleado||'.....Anterior oficio:'||old_oficio || '.....Nuevo  
oficio:' || nuevo_oficio);
```

```
END cambiar_oficio;
```

Para ejecutarlo el anterior procedimiento:

```
EXECUTE cambiar_oficio(7902,'DIRECTOR');
```

O también generar un bloque u otro procedimiento que contenga el procedimiento **85**

Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS

Ejemplo, pero llamándolo desde otro bloque o función:

```
CREATE OR REPLACE
  PROCEDURE cam_ofi(v_apellido VARCHAR2, new_oficio VARCHAR2)
AS
  N_empleado emple.emp_no%TYPE;
BEGIN
  SELECT emp_no INTO N_empleado FROM emple WHERE apellido=v_apellido;
  cambiar_oficio(N_empleado, new_oficio);
END cam_ofi;
```

Para ejecutarlo: EXECUTE cam_ofi ('FERNANDEZ','ANALISTA');

O también

```
begin
```

```
  cam_ofi cam_ofi ('FERNANDEZ','ANALISTA');
```

```
end;
```

Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS

Ejemplo, pero llamándolo desde otro bloque o función:

```
CREATE OR REPLACE
  PROCEDURE cabecera
AS
BEGIN
    dbms_output.put_line(' ***** linea cabecera listado *****');
    dbms_output.put_line(' -----');
END cabecera;
```

Para ejecutarlo:

```
begin
    cabecera(); -- aunque el procedimiento no tenga parámetros tenemos que usar los
               --paréntesis para que pueda diferenciarlo de una variable
end;
```

Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS. PRIVILEGIOS A USUARIOS

- El procedimiento se ejecutará con los privilegios del propietario por defecto.
- Solo podrán utilizar un procedimiento el propietario y los usuarios con los privilegios EXECUTE ANY PROCEDURE
- Para asignar un privilegio de ejecución de un procedimiento a un usuario:
`GRANT EXECUTE ON nombre_procedure TO usuario;`
`GRANT EXECUTE ANY PROCEDURE TO usuario;`
- Para ejecutar un procedimiento propiedad de otro usuario:
`EXECUTE usuario.procedimiento;`

Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS. EJERCICIOS

- Escribir un procedimiento que escriba dos números y visualice su suma. Después escribir un bloque PL/SQL que introducidos dos números llame a dicho procedimiento para calcular el resultado.
- Escribir un procedimiento que permite calcular el doble y la mitad de un número
- Codificar un procedimiento que reciba una cadena y la visualice al revés.
- Crear una tabla llamada T_NUMEROS con un campo de tipo numérico (clave primaria).
- Escribir un procedimiento que permita insertar un número en la tabla T_NUMEROS
- Diseñar un bloque PL/SQL que pida un número y lo inserte en la tabla T_NUMEROS utilizando el procedimiento anterior

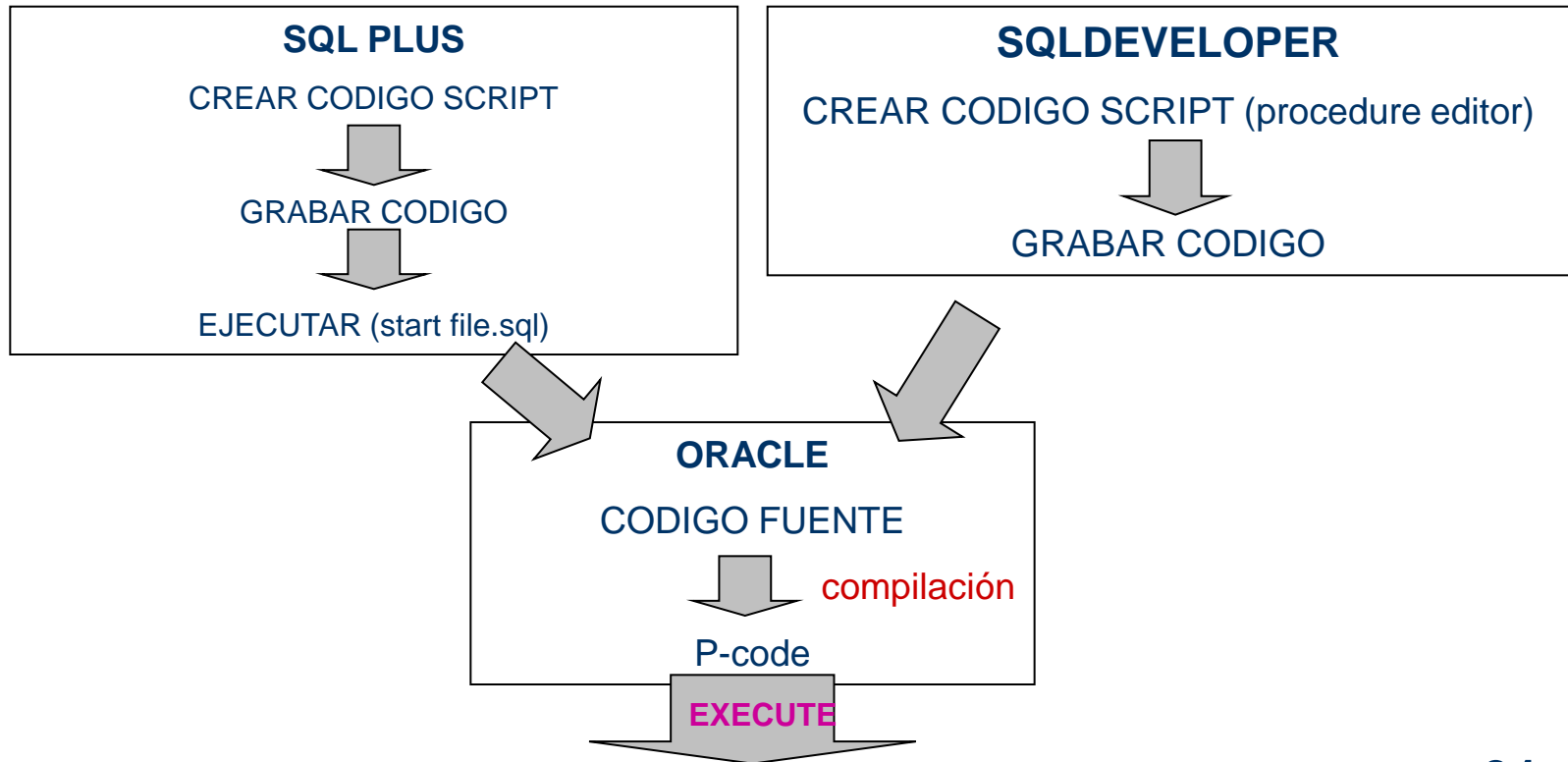
Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS. EJERCICIOS

- Crear un procedimientos que permita visualizar todos los datos de un usuario a partir del numero de empleado, visualizar además el nombre de departamento al que pertenece
- Realizar un bloque que una vez introducido por teclado el emp_no de un empleado, visualice los datos del empleado utilizando el procedimiento anterior.
- Crear un procedimiento que permita calcular la nómina de un empleado y visualizarla
- Realizar un bloque que una vez introducido por teclado el código de un empleado, visualice la nómina que cobrará en empleado utilizando el procedimiento anterior.

Subprogramas: Procedimientos y Funciones

FUNCIONES



Subprogramas: Procedimientos y Funciones

FUNCIONES

- Tienen una estructura y funcionalidad similar a los procedimientos, **pero devuelven siempre un valor.**
- Se diferencian de los procedimientos que **SI** se pueden utilizar como parte de una expresión.

```
suma := salario + nvl(comision,0) + mi_funcion(param1,param2);  
dbms_output.put_line('El valor es: '|| mi_funcion(param1,param2);
```

- Permite además utilizar parámetros IN, OUT, IN OUT

Subprogramas: Procedimientos y Funciones

FUNCIONES

- Tienen la siguiente estructura:

```
FUNCTION <nom_funcion>[(<lista de parámetros>)] RETURN <tipo_valor_devuelto>  
  IS/AS  
    <declaraciones>  
  BEGIN  
    instrucciones;  
    RETURN <expresión>;  
    ....  
  EXCEPTION  
    <excepciones>;  
END [<nom_funcion>;]
```

Subprogramas: Procedimientos y Funciones

FUNCIONES

- Para crear una función o sustituir una existente:

```
CREATE [OR REPLACE]
  FUNCTION <nom_funcion>[(<lista_parámetros>)]RETURN <tipo_valor_devuelto>
  .....
END [<nom_funcion>];
```

- Para ejecutar una función:
EXECUTE variable:=<nom_funcion>[(<valores de parámetros>)]

O bien crear un bloque para mostrar los resultados donde se utilice la función

Subprogramas: Procedimientos y Funciones

FUNCIONES

Ejemplo:

```
CREATE OR REPLACE
  FUNCTION encontrar_num_empleado(v_apellido VARCHAR2) RETURN  REAL
AS
  n_empleado  emple.emp_no%TYPE;
BEGIN
  SELECT emp_no INTO n_empleado FROM EMPLE WHERE apellido=v_apellido;
  RETURN n_empleado;
END encontrar_num_empleado;
```

Para ejecutarlo:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE (encontrar_num_empleado('MUÑOZ'));
END;
```

Subprogramas: Procedimientos y Funciones

FUNCIONES. Ejercicios

1. Escribir una función que reciba una fecha y devuelva el año, en número, correspondiente a esa fecha
2. Escribir un bloque PL/SQL que haga uso de la función anterior. Por ejemplo, a partir de la fecha de hoy visualizar el año.
3. Diseñar una función llamada FUNC_NUM_EMP que permita devolver el emp_no de un determinado empleado a partir de su apellido, el departamento y el oficio que tiene.
4. Diseñar un bloque que permita comprobar el ejercicio anterior

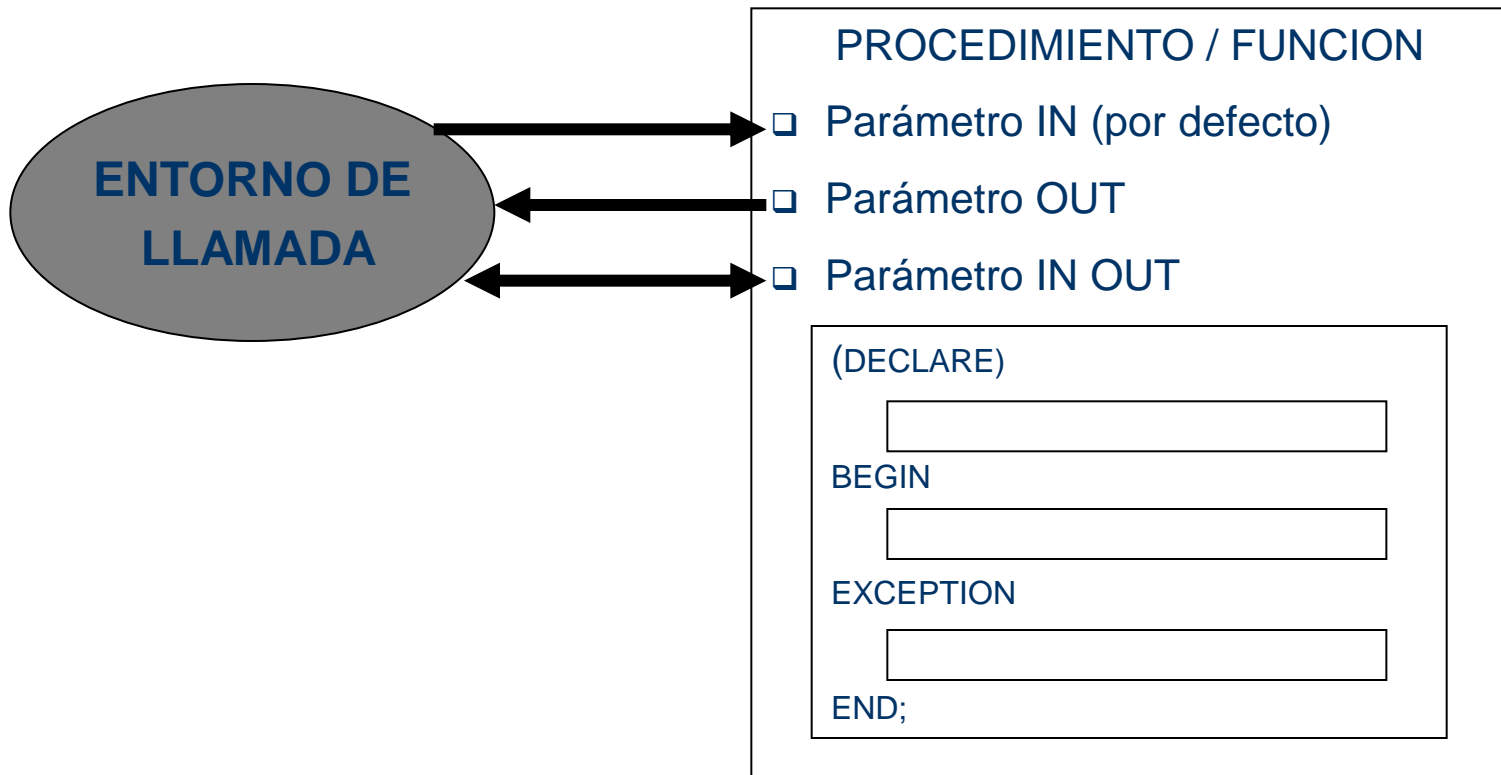
Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS: parámetros

- Se utilizan para pasar o recibir información en PROCEDURES y FUNCTION
- Pueden ser:
 - **Parámetros actuales o reales:** las variables o expresiones indicadas en la llamada al subprograma
 - **Parámetros formales:** variables declaradas en la especificación del subprograma
- El compilador asocia los parámetros actuales a los formales basándose en su posición

Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS: Modos de Parámetros Procedurales



Subprogramas: Procedimientos y Funciones

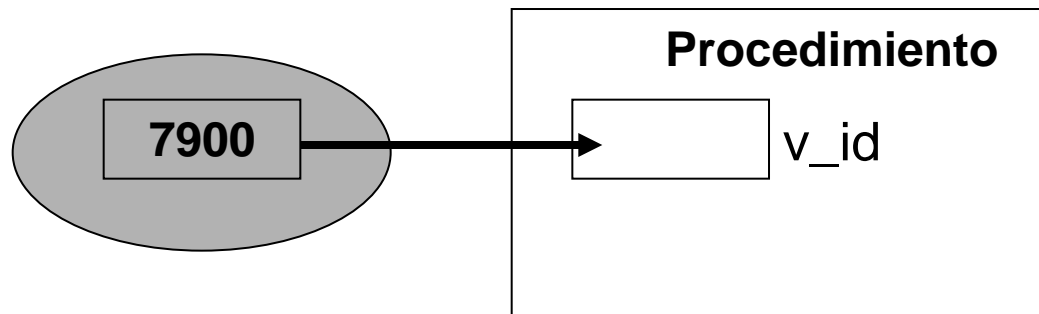
SUBPROGRAMAS: Modos de Parámetros Procedurales

TIPO	CARACTERÍSTICAS Y UTILIZACIÓN
IN	<ul style="list-style-type: none"> – Permite pasar valor a un subprograma – Dentro del subprograma no se puede modificar. Se comporta como constante – El parámetro actual puede ser una variable, constante, literal o expresión – Es la opción por defecto en el paso de parámetro
OUT	<ul style="list-style-type: none"> – Permite devolver valores al bloque que llamó al subprograma. Debe de especificarse – Dentro del subprograma, el parámetro actúa como una variable no inicializada – No puede intervenir en ninguna expresión, salvo para tomar un valor – El parámetro actual debe ser una variable
IN OUT	<ul style="list-style-type: none"> – Permite pasar valor inicial y devolver un valor actualizado. Debe de especificarse – Dentro del subprograma, el parámetro actúa como una variable inicializada – Puede intervenir en otras expresiones y puede tomar nuevos valores – El parámetro actual debe ser una variable

Subprogramas: Procedimientos y Funciones

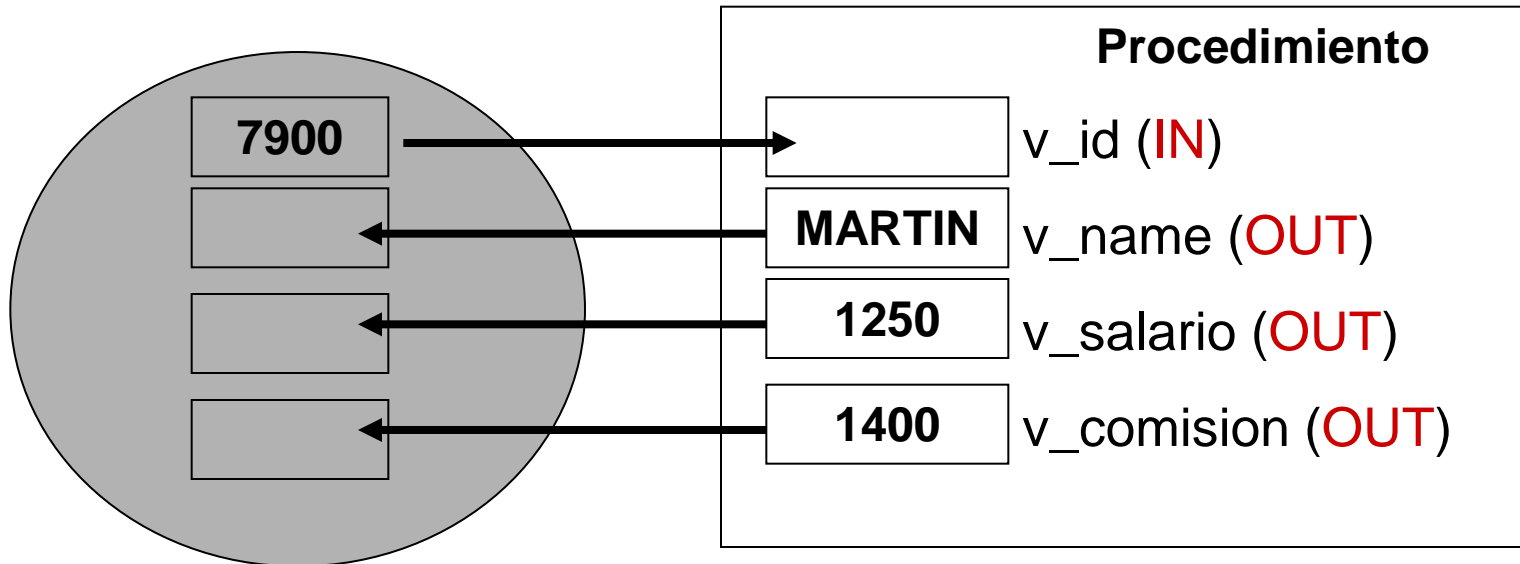
SUBPROGRAMAS: Modos de Parámetros Procedurales

```
CREATE OR REPLACE
  PROCEDURE incrementa_salario (v_id IN emple.emp_no%TYPE)
IS
BEGIN
  UPDATE emple SET sal:=sal*1.10 where emp_no=v_id;
END incrementa_salario;
=====
EXECUTE incrementa_salario(7900);
```



Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS: Modos de Parámetros Procedurales



Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS: Modos de Parámetros Procedurales

```

CREATE OR REPLACE PROCEDURE consulta_emp (
    v_id          IN          emple.emp_no%TYPE,
    v_nombre      OUT         emple.apellido%TYPE,
    v_salario     OUT         emple.salario%TYPE,
    v_comision    OUT         emple.comision%TYPE)
IS
BEGIN
    SELECT apellido, salario, comision INTO v_nombre,v_salario,v_comision FROM emple
    WHERE emp_no=v_id;
END consulta_emp;

=====
Para comprobar el funcionamiento
declare
    g_nombre emple.apellido%type;
    g_salario emple.salario%type;
    g_comision emple.comision%type;
begin
    consulta_emp(7900,g_nombre,g_salario,g_comision); --llamada al procedimiento
    dbms_output.put_line(g_nombre||' '||g_salario||' '||g_comision);
end;
```

Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS: Modos de Parámetros Procedurales

```
CREATE OR REPLACE PROCEDURE cambiar_divisas(  
    cantidad          IN NUMBER,  
    cambio_actual     IN NUMBER,  
    comision          IN OUT  NUMBER,  
    divisas           OUT NUMBER)  
AS  
    porcen_comision   NUMBER(3,2)   DEFAULT 0.2;  
    min_comision      NUMBER(6)      DEFAULT 3;  
BEGIN  
    IF comision IS NULL THEN  
        comision:=GREATEST(cantidad/100*porcen_comision,min_comision);  
    END_IF;  
    divisas:=(cantidad – comision)/cambio_actual;  
END;
```

Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS: Parámetros. Modos

```
CREATE OR REPLACE PROCEDURE probar_cambio_divisas(
    dolares          NUMBER,
    cambio           NUMBER)
AS
    v_comision       NUMBER(9);
    v_divisas        NUMBER(9);
BEGIN
    cambiar_divisas(ptas,cambio,v_comision,v_divisas);
    DBMS_OUTPUT.PUT_LINE('Dolares          :'|| TO_CHAR (dolares,'999,999,999.999'));
    DBMS_OUTPUT.PUT_LINE('Precio Divisas   :'|| TO_CHAR (cambio,'999,999,999.999'));
    DBMS_OUTPUT.PUT_LINE('€ Comision      :'|| TO_CHAR (v_comision,'999,999,999.999'));
    DBMS_OUTPUT.PUT_LINE('Cantidad divisas :'|| TO_CHAR (v_divisas,'999,999,999.999'));
END;
```

Para probar: EXECUTE probar_cambio_divisas(1500, 1.17);

Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS: Parámetros **in**, **out**, **in out** en funciones

Ejemplo:

```
CREATE OR REPLACE
FUNCTION SUMA(n1 IN NUMBER,n2 IN OUT NUMBER,n3 OUT NUMBER) RETURN
    NUMBER
AS
    rtdo number;
BEGIN
    rtdo := n1+n2;
    n2:= n2+1;
    n3:=4;
    RETURN rtdo;
END SUMA;
```

Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS: Parámetros **in**, **out**, **in out** en funciones

Ejemplo: Bloque de prueba de la función con parámetros in, out, in out

```
declare
```

```
  n1 number:=1;
```

```
  n2 number:=1;
```

```
  n3 number;
```

```
Begin
```

```
  dbms_output.put_line('suma = '||suma(n1,n2,n3));
```

```
  dbms_output.put_line('n2 = '||n2); -- variable de tipo in out
```

```
  dbms_output.put_line('n3 = '||n3); -- variable de tipo out
```

```
end;
```

```
set serveroutput on
suma = 2
n2 = 2
n3 = 4
```

Subprogramas: Procedimientos y Funciones

SUBPROGRAMAS. BORRADO

- **BORRADO** de un subprograma

DROP {PROCEDURE | FUNCTION} nombresubprograma;

- **Vistas** para comprobar el estado de un subprograma. Este puede estar **VALID** o **INVALID**

USER_OBJECTS

- Para volver a compilar un subprograma almacenado en la BD

ALTER {PROCEDURE|FUNCTION} nom_subprograma **COMPILE;**

Subprogramas: Procedimientos y Funciones

PROCEDIMIENTOS FRENTA FUNCIONES

PROCEDIMIENTO	FUNCIÓN
Se ejecuta como una sentencia PL/SQL	Son llamadas como parte de una expresión
No devuelve un tipo de dato asociado al nombre del procedimiento	Deben contener RETURN <tipo_dato>
Puede devolver 0, uno o más valores según la declaración de los parámetros realizada	Devuelve siempre un valor asociado al nombre de la función, aunque podemos usar también parámetros de forma IN, OUT, IN OUT

Subprogramas: Procedimientos y Funciones

RECURSIVIDAD

Permite que un subprograma se llame a sí mismo hasta que se cumpla una condición de parada, momento a partir de la cual se pueda empezar a realizar los cálculos pendientes en memoria de las sucesivas llamadas

Ejemplo

```
CREATE OR REPLACE FUNCTION factorial (n NATURAL) RETURN INTEGER
AS
BEGIN
    IF n=0 THEN
        RETURN 1;
    ELSE
        RETURN n* factorial(n-1);
    END IF;
END factorial;
```

Cursores

CURSOR SQL

- Un cursor SQL es un área de memoria que se abre para analizar y ejecutar sentencias SQL
- Siempre aparece cuando se ejecuta una sentencias SQL
- Tipos de cursores:
 - **CURSORES IMPLÍCITOS**: Creado por el servidor Oracle para analizar y ejecutar las sentencias SQL. Se crean automáticamente para todas las sentencias DML y el SELECT ... INTO... del PL/SQL
 - **CURSORES EXPLÍCITOS**: son los definidos por el programador.

Cursores

CURSOR IMPLICITO. ATRIBUTOS

- Permiten evaluar el resultado de las sentencias SQL
- No pueden ser utilizados en sentencias SQL
- Se pueden utilizar en la sección de EXCEPCIONES de un bloque para reunir información sobre la ejecución de un sentencia

SQL%ROWCOUNT	Devuelve el número de filas afectadas por la sentencia SQL
SQL%FOUND	Devuelve TRUE si la sentencia SQL afecta a una o más filas
SQL%NOTFOUND	Devuelve TRUE si la sentencia no afecta a ninguna fila, es decir ha fallado
SQL%ISOPEN	Para los cursores implícito siempre devuelve FALSE, ya que se cierran automáticamente después de ejecutar una SQL

Cursores

CURSOR IMPLICITO. ATRIBUTOS. Ejercicios

- Diseñar un bloque PL/SQL que para un departamento introducido por teclado, incremente el sueldo los empleados de dicho departamento en 1€. Visualizar el número de filas afectadas.
- Modificar el bloque anterior para que en el caso de no existir empleados del departamento introducido, devuelva el mensaje de 'NO HAY FILAS DE EMPLEADOS PARA EL DEPARTAMENTO

Cursores

CURSOR IMPLICITO. ATRIBUTOS. Ejercicios

- Diseñar un bloque PL/SQL que para un departamento introducido por teclado, incremente el sueldo los empleados de dicho departamento en 1€. Visualizar el número de filas afectadas.

```
ACCEPT DEP PROMPT 'Introduce el número de departamento: ';  
DECLARE  
    NUM NUMBER;  
BEGIN  
    UPDATE EMPL SET EMPL.SALARIO=EMPL.SALARIO+1 WHERE  
    DEPT_NO=&DEP;  
    DBMS_OUTPUT.PUT_LINE('COLUMNAS ACTUALIZADAS '|| SQL%ROWCOUNT);  
    COMMIT;  
END;
```

Cursores

CURSOR IMPLICITO. ATRIBUTOS. Ejercicios

- Modificar el bloque anterior para que en el caso de no existir empleados del departamento introducido, devuelva el mensaje de 'NO HAY FILAS DE EMPLEADOS PARA EL DEPARTAMENTO

```
ACCEPT DEP PROMPT 'Introduce el número de departamento: ';
DECLARE
    NUM NUMBER;
BEGIN
    UPDATE EMPL SET EMPL.SALARIO=EMPL.SALARIO+1 WHERE DEPT_NO=&DEP;
    IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('No hay filas de empleados para el
                               departamento '|| &dep);
    ELSE
        DBMS_OUTPUT.PUT_LINE('COLUMNAS ACTUALIZADAS '||
                               SQL%ROWCOUNT);
    END IF;
    COMMIT;
END;
```

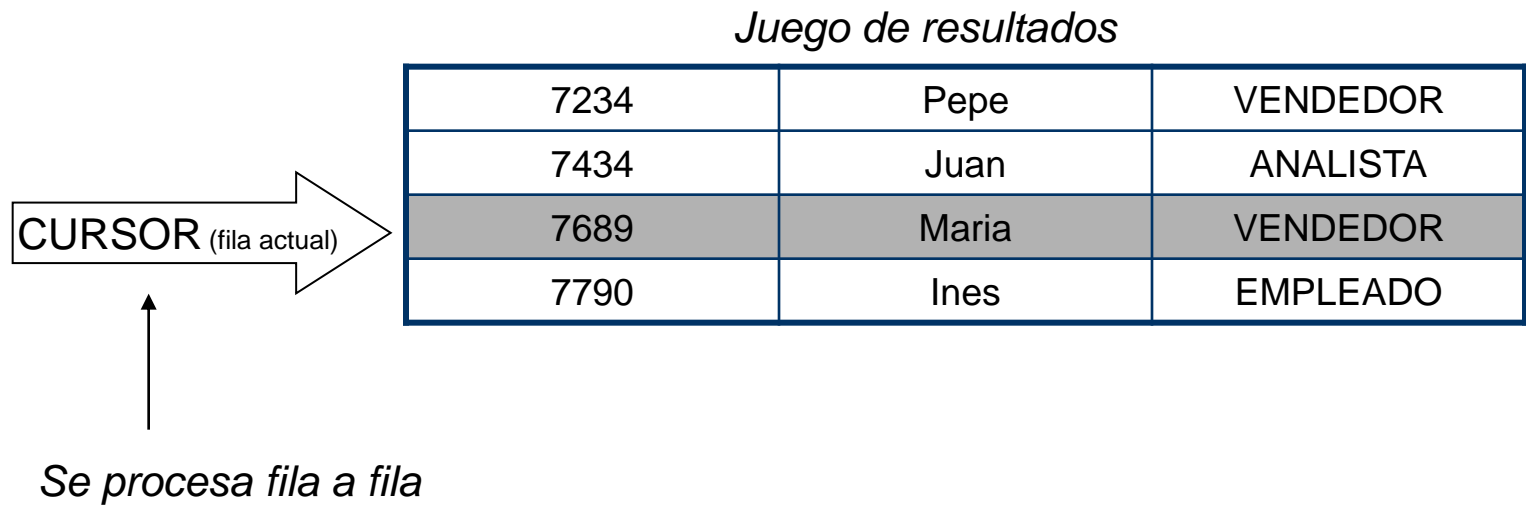
Cursores

CURSOR EXPLICITO

- Son declarados y definidos por el programador
- Un programa PL/SQL: abre el cursor, procesa filas devueltas por la consulta, y después cierra el cursor.
- Se utilizar normalmente para consultas que devuelven más de una fila (**juego de resultados**)
- Permite **procesar individualmente las filas devueltas por una sentencia *SELECT* que devuelve más de una fila.**
- El tamaño del cursor explícito es el número de filas que cumplen los criterios de búsqueda de la sentencia **SELECT**
- El cursor marca la posición actual en el juego de resultados

Cursores

CURSOR EXPLICITO



Cursores

CURSOR EXPLICITO. Funciones

- Puede procesar más allá de la primera fila devuelta por la consulta
- La manipulación de las filas devueltas (JUEGO DE RESULTADOS) se realiza fila a fila.
- Comprueba la fila que está siendo procesada actualmente
- Permiten al programador controlar la fila manualmente

Cursores

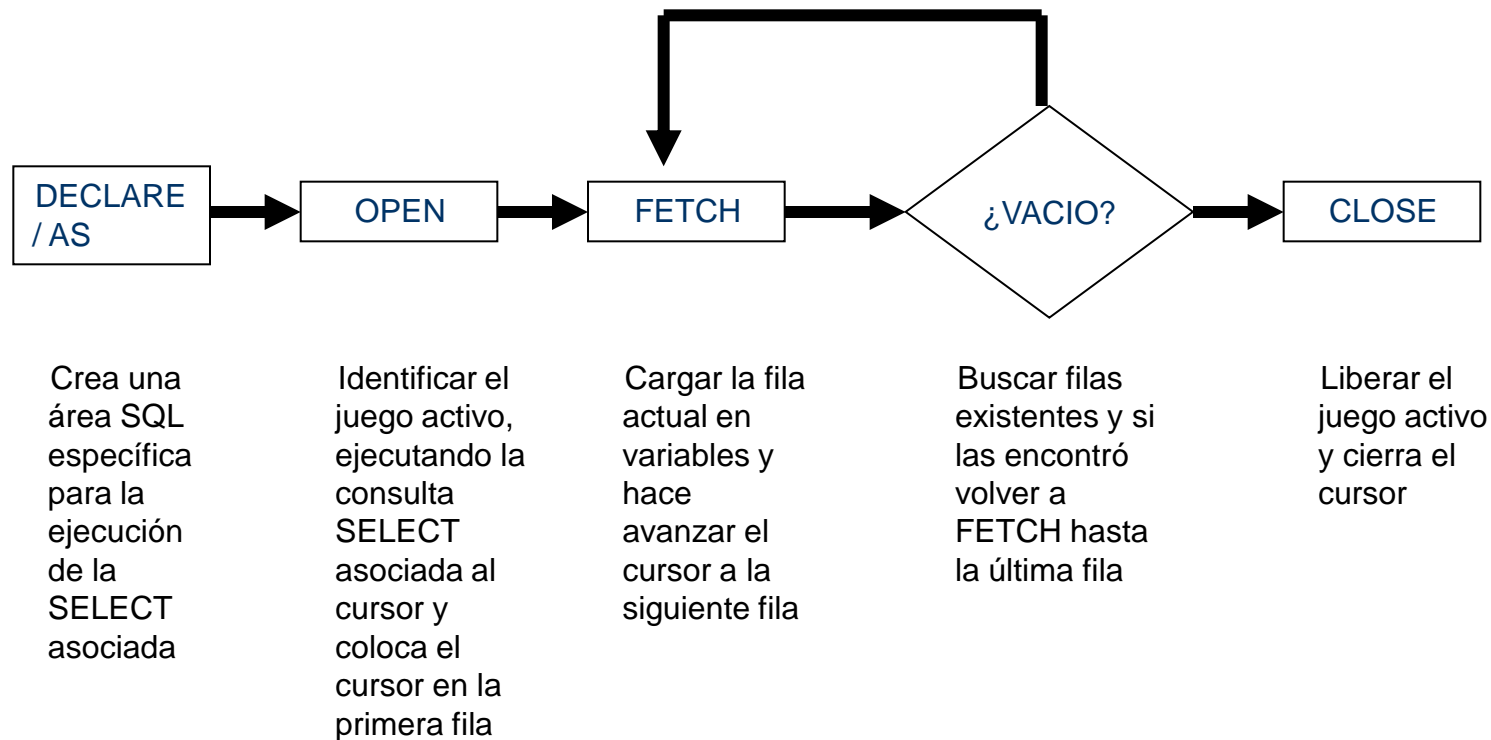
CURSOR EXPLICITO. CONTROL DE CURSORES

Pasos a dar para la utilización de cursores:

- **Declarar el cursor** dándole un nombre y definiendo la estructura de la consulta **SELECT** que se va a ejecutar en él
- **Abrir el cursor** con la sentencia **OPEN**, que ejecuta la consulta y resuelve las variables a las que hace referencia. Las filas devueltas por la consulta se denominan **JUEGO ACTIVO**, y están disponibles para ser recuperadas.
- **Recuperar datos del cursor**. La sentencia **FETCH** toma la fila actual del cursor y la carga en variables. Cada recuperación hace que el cursor se mueva hacia la siguiente fila del juego activo.
- **Cierre del cursor**. Cuando se llega al final se la sentencia **CLOSE** libera el juego activo de filas. En este momento es posible volver a abrir el cursor para establecer un juego activo refrescado.

Cursores

CURSOR EXPLICITO. CONTROL DE CURSORES



Cursores

CURSOR EXPLICITO. Declaración del cursor

SINTAXIS:

DECLARE

.....

CURSOR nom_cursor IS sentencia_SELECT;

- No incluir la cláusula INTO en la declaración del cursor

Ejemplo:

DECLARE

.....

CURSOR emp_cursor IS SELECT emp_no, apellido FROM emp;

CURSOR dept_cursor IS SELECT * FROM depart WHERE dept_no=10;

Cursores

CURSOR EXPLICITOS. Apertura del cursor

SINTAXIS:

BEGIN

.....

OPEN nom_cursor;

OPEN es una sentencia que realiza las siguientes operaciones:

- Asigna memoria dinámicamente
- Analiza la sentencia SELECT
- Identifica el juego de resultados que cumplen la SELECT, pero no se cargan en las variables de memoria hasta que se ejecute FETCH
- Posiciona el puntero antes de la 1ª fila del juego activo

NOTA: *Si la consulta no devuelve ninguna fila no devuelve ninguna excepción, sin embargo se puede comprobar el estado del cursor después de una recuperación.*

Cursores

CURSOR EXPLICITO. Recuperación de datos

SINTAXIS:

BEGIN

.....

FETCH nom_cursor INTO [variable1, variable2,... | nom_reg];

- Recupera los valores de la fila actual y los almacena en las variables
- Las variables se definirán en el DECLARE
- Se deben incluir el mismo número de variables que columnas devuelve la sentencia SELECT (los tipos han de ser compatibles)
- Como alternativa se puede definir un registro para el cursor y hacer referencia al registro en la cláusula FETCH INTO
- Cada vez que se ejecuta FETCH el puntero se desplaza a la siguiente fila del juego activo
- Se utiliza en combinación de un bucle, para saber cuando hemos llegado al final del juego de resultados

Cursores

CURSOR EXPLICITO. Recuperación de datos

```

DECLARE
    v_emp_no  emple.emp_no%type;
    v_apellido emple.apellido%type;
    CURSOR    emp_cursor IS SELECT emp_no,apellido from emple;
BEGIN
    ....
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO v_emp_no,v_apellido; --si devuelve filas carga la 1ª
        EXIT when emp_cursor%NOTFOUND; --si no devuelve filas → TRUE y sale
        DBMS_OUTPUT.PUT_LINE(v_emp_no || ' ' ||v_apellido);
    END LOOP;
    CLOSE emp_cursor;
END;
```

Cursores

CURSOR EXPLICITO. Cierre

SINTAXIS:

BEGIN

.....

CLOSE nom_cursor;

- Una vez completado el procesamiento de la sentencia SELECT es necesario cerrar el cursor, para liberar el área de contexto en memoria.
- Una vez cerrado el cursor no se pueden recuperar datos.
- Existe un número máximo de cursores abiertos por usuario determinado por el parámetro OPEN_CURSORS. OPEN_CURSORS=50 por defecto.

Ejemplo: **CLOSE emp_cursor;**

Cursores

CURSOR EXPLICITOS. Atributos de Cursores

- Obtiene información del estado de un cursor

ATRIBUTO	TIPO	DESCRIPCION
<i>Nom_cursor%ISOPEN</i>	Booleano	TRUE si el cursor está abierto
<i>Nom_cursor%NOTFOUND</i>	Booleano	TRUE si la recuperación más reciente no devuelve una fila
<i>Nom_cursor%FOUND</i>	Booleano	TRUE si la recuperación más frecuente devuelve una fila
<i>Nom_cursor%ROWCOUNT</i>	Número	Da el número total de filas procesadas hasta ese momento

Cursores

CURSOR EXPLICITOS. Ejercicios

- Realizar un bloque PL/SQL que permite definir un cursor para visualizar el emp_no, apellido y dept_no de todos los empleados

<u>Num Emple</u>	<u>nombre</u>	<u>departamento</u>
.....

- Modificar el bloque anterior para visualizar solo los empleados del departamento 30

EMPLEADOS DEL DEPARTAMENTO: 30

<u>Num Emple</u>	<u>nombre</u>	<u>departamento</u>
------------------	---------------	---------------------

Cursores

CURSOR EXPLICITOS. Ejercicios

- Realizar un bloque PL/SQL que permite definir un cursor para visualizar el emp_no, apellido y dept_no de todos los empleados

DECLARE

v_emp_no emple.emp_no%type;

v_apellido emple.apellido%type;

v_dept_no emple.dept_no%type;

CURSOR emp_cursor IS SELECT emp_no,apellido,dept_no from emple;

BEGIN

OPEN emp_cursor; *--abro el cursor*

dbms_output.put_line(' EMPLEADOS DEPARTAMENTO ');

dbms_output.put_line(' NumEmple Nombre Departamento');

-- proceso la sentencia

LOOP

FETCH emp_cursor INTO v_emp_no,v_apellido,v_dept_no;

EXIT when emp_cursor%NOTFOUND;

DBMS_OUTPUT.PUT_LINE(v_emp_no || ' '||v_apellido||' '||v_dept_no);

END LOOP;

--cierro el cursor

CLOSE emp_cursor;

END;

Cursores

CURSOR EXPLICITOS. Ejercicios

- Realizar un bloque PL/SQL que permite definir un cursor para visualizar el emp_no, apellido y dept_no de todos los empleados del dept_no=30

DECLARE

```
v_emp_no     emple.emp_no%type;
v_apellido   emple.apellido%type;
v_dept_no    emple.dept_no%type;
CURSOR       c_emp_cursor IS SELECT emp_no,apellido,dept_no from emple where dept_no=30;
```

BEGIN

```
OPEN c_emp_cursor; --abro el cursor
dbms_output.put_line('      EMPLEADOS DEPARTAMENTO  30');
dbms_output.put_line('      NumEmple      Nombre      Departamento');
-- proceso la sentencia
FETCH c_emp_cursor INTO v_emp_no,v_apellido,v_dept_no;
WHILE c_emp_cursor%found LOOP
    DBMS_OUTPUT.PUT_LINE(v_emp_no || ' '||v_apellido||' '||v_dept_no);
    FETCH c_emp_cursor INTO v_emp_no,v_apellido,v_dept_no;
END LOOP;
--cierro el cursor
CLOSE c_emp_cursor;
```


END;

Cursores

CURSOR EXPLICITO y REGISTROS

- Corresponde a la situación en la que se define un cursor y a continuación se define una variable registro (RECORD) a partir de las filas que devuelve el cursor utilizando %ROWTYPE

```
DECLARE
    CURSOR    c_emp_cursor IS SELECT emp_no, apellido FROM emple;
    v_emp_reg  c_emp_cursor%ROWTYPE; -- variable de tipo record
BEGIN
    OPEN c_emp_cursor;
    LOOP
        FETCH c_emp_cursor INTO v_emp_reg;
        ...
        DBMS_OUTPUT.PUT_LINE(v_emp_reg.emp_no|| ' ' ||v_emp_reg.apellido);
        ....
    
```



Cursores

CURSOR EXPLICITO. Bucles FOR de CURSOR

Permite procesar filas en un cursor explícito, es otra alternativa al FETCH, mucho mas sencilla de usar

SINTAXIS:

```
FOR nom_registro IN nom_cursor LOOP
    instrucc1;
    ....
END LOOP;
```

Donde ***nom_cursor*** debe estar declarada anteriormente en DECLARE

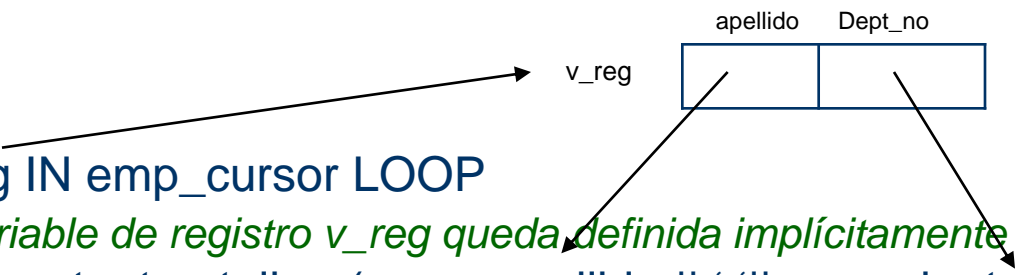
- Facilita el procesamiento de cursores explícitos
- Abre el cursor, recupera filas en cada iteración y cierra automáticamente el cursor.
- La variable registro (***nom_registro***) queda declarada implícitamente con el FOR, no siendo necesario definirla en DECLARE y desplazando automáticamente el cursor a la siguiente fila.

Cursores

CURSOR EXPLICITO. Bucles FOR de CURSOR

```
DECLARE
    CURSOR emp_cursor IS select apellido,dept_no from emple;

BEGIN
    for v_reg IN emp_cursor LOOP
        -- la variable de registro v_reg queda definida implícitamente
        dbms_output.put_line (v_reg.apellido || ' ' || v_reg.dept_no);
        ....
    end loop;
END;
```



Cursores

CURSOR EXPLICITO. Ejercicios

- Realizar procedimiento que permite definir un cursor con todas las columnas/campos de la tabla DEPART. Visualizar del juego de resultados de dicho cursor utilizando un registro y un bucle FOR de CURSOR
- Realizar procedimiento que permita visualizar el número del empleado, el nombre y el sueldo de los empleados de un determinado departamento que se introducirá por teclado.

Cursores

CURSOR EXPLICITO. Ejercicios

- Realizar procedimiento que permite definir un cursor con todas las columnas de la tabla DEPART. Visualizar dicho cursor utilizando un registro y un bucle FOR de CURSOR

```
CREATE OR REPLACE PROCEDURE ver_departamentos
AS
    CURSOR depart_cursor IS select * from depart;
BEGIN
    --no hace falta ABRIR el cursor por ser una operación implícita en el FOR CURSOR
    FOR reg_depart IN depart_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('Cod. Departamento: '|| reg_depart.dept_no ||
                               ' Nom. Departamento: '||reg_depart.dnombre);
    END LOOP;
    --no hace falta CERRAR el cursor por ser una operación implícita en el FOR CURSOR
END ver_departamentos;
```

Cursores

CURSOR EXPLICITO. Ejercicios

- Realizar un procedimiento que permita visualizar el número del empleado, el nombre y el sueldo de los empleados de un determinado departamento que se introducirá por teclado.

```
CREATE OR REPLACE PROCEDURE ver_emple_departamento(v_dep number)
AS
  CURSOR emp_cursor IS select * from emple;
  --podria hacer CURSOR emp_cursor IS select emp_no,apellido,salario,comision FROM emple where
  dept_no=v_dept;
  sueldo emple.salario%type;
BEGIN
  --no hace falta abrir el curso, pues el FOR cursor abre automáticamente el cursor
  FOR reg_emple IN emp_cursor LOOP
    if reg_emple.dept_no=v_dep then
      sueldo:=reg_emple.salario+nvl(reg_emple.comision,0);
      DBMS_OUTPUT.PUT_LINE('Emp_no: '||reg_emple.emp_no||' Nombre:
        '||reg_emple.apellido||' Sueldo:'||sueldo);
    end if;
  END LOOP;
  --no hace falta CERRAR el cursor por ser una operación implícita en el FOR CURSOR
END;
```

Cursores

CURSOR EXPLICITO. Parámetros

- Permite pasar valores a un cursor cuando éste es abierto para que se utilicen cuando se evalúa la sentencia SELECT
- Permite devolver un juego activo distinto cada vez

SINTAXIS:

**CURSOR nombre_cursor [(param1 tipo_dato, param2 tipo_dato,...)]
IS sentencia_SELECT;**

Tipo_dato: corresponde solo al tipo de dato, no al tamaño

Para utilizar el cursor:

OPEN nombre_cursor(valor1, valor2,...);

*Donde valor1, valor2,... Son los valores con los que se va a ejecutar la consulta
SELECT*

Cursores

CURSOR EXPLICITO. Parámetros

```
DECLARE
```

```
.....
```

```
CURSOR      emp_cursor (v_dept_no NUMBER, v_oficio VARCHAR) IS  
SELECT apellido, salario FROM emple WHERE dept_no=v_dept_no and oficio=v_oficio;
```

```
...
```

```
BEGIN
```

```
.....
```

```
OPEN emp_cursor(10,'ANALISTA'); -- se ejecuta el SELECT con estos datos
```

```
....
```

```
END;
```


Cursores

CURSOR EXPLICITO. Ejercicios

- Realizar un procedimiento que permita visualizar el número del empleado, el nombre y el departamento de los empleados de un determinado departamento que se introducirá por teclado. Realizar el ejercicio utilizando OPEN, FETCH y CLOSE, utilizando paso de parámetros.
- Realizar el mismo ejercicio utilizando un bucle FOR de cursor

Cursores

CURSOR EXPLICITO. Ejercicios Solucion (I)

- Realizar un procedure que permita visualizar el número del empleado, el nombre y el departamento de los empleados de un determinado departamento que se introducirá por teclado.

Create or replace procedure ver_empleados_departamento (v_dep number)

AS

CURSOR emp_cursor (v_dept_no NUMBER) IS select emp_no,apellido,dept_no from
emple where dept_no=v_dept_no;

emp_reg emp_cursor%rowtype;

BEGIN

OPEN emp_cursor(v_dep);

FETCH emp_cursor INTO emp_reg;

WHILE emp_cursor%FOUND LOOP

DBMS_OUTPUT.PUT_LINE('Num emple: '||emp_reg.emp_no||' Nombre:
'||emp_reg.apellido||' Departamento: '||emp_reg.dept_no);

FETCH emp_cursor INTO emp_reg;

END LOOP;

CLOSE emp_cursor;

END ver_empleados_departamento;

Cursores

CURSOR EXPLICITO. Ejercicios. Solución (II)

- Realizar un procedimiento que permita visualizar el número del empleado, el nombre y el departamento de los empleados de un determinado departamento que se introducirá por teclado.

```
Create or replace procedure ver_empleados_departamento (v_dep number)
AS
```

```
    CURSOR emp_cursor (v_dept_no NUMBER) IS select
        emp_no,apellido,dept_no from
            emple where dept_no=v_dept_no;
```

```
BEGIN
```

```
    FOR emp_reg IN emp_cursor(v_dep) LOOP
```

```
        DBMS_OUTPUT.PUT_LINE('Num emple: '||emp_reg.emp_no||' Nombre:
        '||emp_reg.apellido||' Departamento: '||emp_reg.dept_no);
```

```
    END LOOP;
```

```
END ver_empleados_departamento;
```

Cursores

CURSOR EXPLICITO: CURSOR FOR UPDATE

- Permite actualizar o borrar las filas seleccionadas por un cursor explícito
- Indica que las filas actualizadas por el cursor van a ser actualizadas o borradas, quedando bloqueadas tan pronto se abra el cursor y serán desbloqueadas al terminar las actualizaciones.
- Para crearlos será necesario añadir **FOR UPDATE** al final de la declaración del cursor, y en la sentencia de actualización UPDATE o DELETE será necesario añadir la cláusula **WHERE CURRENT OF nom_cursor** para actualizar **SOLO la fila recuperada con FETCH**

SINTAXIS:

CURSOR nombre_cursor **IS** sentencia_SELECT **FOR UPDATE;**

En el cuerpo para realizar la actualización solo de la fila donde esta el cursor

...
{UPDATE | DELETE} **WHERE CURRENT OF nom_cursor;**

Cursores

CURSOR EXPLICITO. Ejercicios (1/2)

- Procedimiento que permita subir el salario de todos los empleados de un departamento un tanto por ciento. Ambos valores se indicarán en la llamada al procedimiento

```
Create or replace procedure subir_salario_dpto ( v_num_dpto NUMBER,  
v_subida NUMBER)
```

```
AS
```

```
    CURSOR c_emple IS
```

```
        select oficio, salario from emple where dept_no = v_num_dpto FOR UPDATE;
```

```
    v_reg c_emple%rowtype;
```

```
    v_inc  number;
```

Cursores

CURSOR EXPLICITO. Ejercicios (2/2)

- Procedimiento que permita subir el salario de todos los empleados de un departamento un tanto por ciento. Ambos valores se indicarán en la llamada al procedimiento

Begin

OPEN c_emple;

FETCH c_emple into v_reg;

WHILE c_emple%found LOOP

 v_inc:=(v_reg.salario/100)* v_subida;

 update emple SET salario = salario + v_inc

 where **CURRENT OF c_emple**; -- (al actual)

 FETCH c_emple into v_reg;

END LOOP;

End subir_salario_dpto ;

Cursores

CURSOR EXPLICITO. Ejercicios

- Realizar un procedimiento que para cada departamento (dept_no, dnombre) de la tabla DEPART visualice los empleados (emp_no, apellido, fecha_alt y sueldo (salario+comision)). Utilizar un cursor para la consulta de la tabla DEPART y un cursor con parámetro para la consulta de la tabla de EMPLE de ese departamento. El listado deberá estar ordenado por Número de Departamento

Departamento Num: 99	Nombre Departamento: xxxxxxxx
9999 XXXXXX	dd/mm/aaaa 99.999.999,99
9999 XXXXXX	dd/mm/aaaa 99.999.999,99

Departamento Num: 99	Nombre Departamento: xxxxxxxx
9999 XXXXXX	dd/mm/aaaa 99.999.999,99
9999 XXXXXX	dd/mm/aaaa 99.999.999,99

Cursores

CURSOR EXPLICITO. Ejercicios Solucion (1/2)

Create or replace procedure listado_emple_por_departamento

AS

CURSOR cursor_depart IS SELECT dept_no,dnombre FROM depart;

CURSOR cursor_emple(cod_dep emple.dept_no%type) IS

SELECT * FROM emple WHERE dept_no=cod_dep;

v_dep_num_actual depart.dept_no%type;

v_dep_nom_actual depart.dnombre%type;

v_reg_emple_actual emple%rowtype;

v_cabecera varchar2(100):='COD_EMP NOMBRE FECHA_ALTA SUELDO';

v_linea varchar2(100):='-----';

sueldo number;

BEGIN

 OPEN cursor_depart;

Cursores

CURSOR EXPLICITO. Ejercicios Solucion (2/2)

```

FETCH cursor_depart INTO v_dep_num_actual,v_dep_nom_actual;
WHILE cursor_depart%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE('Numero Departamento: '|| v_dep_num_actual||' Nombre
                          Departamento: ' ||v_dep_nom_actual);
    DBMS_OUTPUT.PUT_LINE(' '); --escribo linea en blanco
    DBMS_OUTPUT.PUT_LINE(v_cabecera);
    DBMS_OUTPUT.PUT_LINE(v_linea);
    -- para este departamento miro los empleados que hay
    OPEN cursor_emple(v_dep_num_actual);
    FETCH cursor_emple INTO v_reg_emple_Actual;
    WHILE cursor_emple%FOUND LOOP
        sueldo:=v_reg_emple_actual.salario+nvl(v_reg_emple_actual.comision,0);
        DBMS_OUTPUT.PUT_LINE(' '||v_reg_emple_actual.emp_no||' '
                              ||RPAD(v_reg_emple_actual.apellido,10,' ')||' '||v_reg_emple_actual.fecha_alt||' '||sueldo);
        FETCH cursor_emple INTO v_reg_emple_actual;
    END LOOP;
    CLOSE cursor_emple;
    -- para este departamento he finalizado los empleados y paso al siguiente departamento
    DBMS_OUTPUT.PUT_LINE(' '); --deja linea en blanco para el nuevo departamento
    FETCH cursor_depart INTO v_dep_num_actual,v_dep_nom_actual;
END LOOP;
CLOSE cursor_depart;
END listado_emple_por_departamento ;

```

Cursores

CURSOR EXPLICITO. Ejercicios Otra solución con FOR de CURSOR, quizás más fácil de entender

```

BEGIN
  OPEN cursor_depart;
  FETCH cursor_depart INTO v_dep_num_actual,v_dep_nom_actual;
  WHILE cursor_depart%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE('Numero Departamento: || v_dep_num_actual|
                          |' Nombre Departamento: ||v_dep_nom_actual);
    DBMS_OUTPUT.PUT_LINE(' '); --escribo linea en blanco
    DBMS_OUTPUT.PUT_LINE(v_cabecera);
    DBMS_OUTPUT.PUT_LINE(v_linea);

    --con el FOR automáticamente abre, lee y cierra el cursor
    FOR v_reg_emple_actual IN cursor_emple(v_dep_num_actual) LOOP
      sueldo:=v_reg_emple_actual.salario+nvl(v_reg_emple_actual.comision,0);
      DBMS_OUTPUT.PUT_LINE(' ||v_reg_emple_actual.emp_no||' '
                          ||RPAD(v_reg_emple_actual.apellido,10,' ')||' '||v_reg_emple_actual.fecha_alt||' '||sueldo);
    END LOOP;

    DBMS_OUTPUT.PUT_LINE(' ');
    FETCH cursor_depart INTO v_dep_num_actual,v_dep_nom_actual;
  END LOOP;
  CLOSE cursor_depart;
END listado_emple_por_departamento ;

```

Cursores

CURSOR EXPLICITO. Ejercicios

- Realizar un procedimiento que visualice los empleados que hay por cada oficio

Oficio: xxxxxx

9999	Xxxxxxxxxxxxxxx	departamento: xxxxx
9999	Xxxxxxxxxxxxxxx	departamento: xxxxx
.....

Oficio: xxxxxx

9999	Xxxxxxxxxxxxxxx	departamento: xxxxx
9999	Xxxxxxxxxxxxxxx	departamento: xxxxx
.....

Cursores

CURSOR VARIABLES

- REF CURSOR es un cursor que no tiene fijada una consulta SQL asociada a ella. Son referencias/punteros a cursores
- Aportan mayor flexibilidad
- Se pueden pasar como parámetros variables de cursor para transferir juegos de resultados de consultas entre los subprogramas almacenados PL/SQL y distintos clientes
- Pasos:
 1. Básicamente, se declara un *TIPO* de tipo REF CURSOR.
 2. A continuación, definir una variable de ese tipo.
 3. A continuación, se asigna una consulta a la variable de cursor.
 4. Después se puede utilizar la variable de cursor como cualquier otro cursor.

Cursores

CURSOR VARIABLES. Definir un tipo REFCURSOR y una variable de dicho tipo

Define a REF CURSOR type

```
TYPE ref_type_name IS REF CURSOR [RETURN return_type];
```

Declarar una variable de cursor de dicho tipo.

```
ref_cv ref_type_name;
```

Ejemplo:

DECLARE

```
TYPE DeptCurTyp IS REF CURSOR RETURN depart%ROWTYPE;
```

```
dept_cv DeptCurTyp;
```

Cursores

CURSOR VARIABLES. Asignar una consulta a la variable cursor

Una vez definido el cursor variable debemos asociarlo a una consulta (notar que esto no se hace en la parte declarativa, sino dinámicamente en la parte de ejecución) y esto lo hacemos con la sentencia OPEN-FOR utilizando la siguiente sintaxis:

OPEN nombre_variable_cursor FOR sentencia_select;

*Ej: OPEN cAgentes FOR SELECT * FROM agentes WHERE oficina = 1;*

Un cursor variable no puede tomar parámetros. Podemos usar los atributos de los cursores para cursores variables.

Además, podemos usar varios **OPEN-FOR para abrir el mismo cursor variable para diferentes consultas. No** necesitamos cerrarlo antes de reabrirlo. Cuando abrimos un cursor variable para una consulta diferente, la consulta previa se pierde.

Una vez abierto el cursor variable, su manejo es idéntico a un cursor. Usaremos **FETCH para traernos las filas**, usaremos sus atributos para hacer comprobaciones y lo cerraremos cuando dejemos de usarlo.

Cursores

CURSOR VARIABLES. Definir un tipo REFCURSOR

```
DECLARE
```

```
    TYPE EmpCurTyp IS REF CURSOR;
```

```
    cursor_emp EmpCurTyp;
```

```
    reg emple%ROWTYPE;
```

```
    sql_stmt VARCHAR2(200);
```

```
    oficio VARCHAR2(10) := 'VENDEDOR';
```

```
BEGIN
```

```
    sql_stmt := 'SELECT * FROM emple WHERE OFICIO = :j' ;
```

```
    OPEN cursor_emp FOR sql_stmt USING oficio;
```

```
    LOOP
```

```
        FETCH cursor_emp INTO reg;
```

```
        EXIT WHEN cursor_emp%NOTFOUND;
```

```
        dbms_output.put_line(reg.apellido||' '||reg.oficio);
```

```
    END LOOP;
```

```
    CLOSE cursor_emp;
```

```
END;
```