Programación de Bases de Datos
PL/SQL sobre Oracle

Índice

1. Introducción

En esta sesión se continúa usando el entorno de trabajo de las sesiones de prácticas anteriores. En este caso, no es preciso añadir nuevo software, puesto que simplemente se utilizará el gestor de base de datos ya instalado (Oracle Express Edition) y el entorno que facilita toda la gestión de consultas (SQL Developer).

- Oracle Database Express Edition - (*Ya instalado en prácticas anteriores*)
- Oracle SQL Developer - (*Ya instalado prácticas anteriores*)

2. Preparación del entorno

Para esta práctica se usa la misma base de datos que se desarrolló en sesiones prácticas anteriores.

Si se hubiesen borrado las tuplas y/o las tablas tras desarrollar las prácticas anteriores, habría que volver a crearlas e insertar las tuplas correspondientes, mediante los archivos facilitados en la práctica inicial. Así, se tendrán las siguientes tablas sobre la que trabajar:

```
SELECT * FROM departamentos;
SELECT * FROM empleados;
```

```
NUMERODPTO NOMBREDPTO            COSTE    PORCENT
---------- ---------------- ---------- ----------
         1 PERSONAL                  0          0
         2 PRODUCCIÓN                0          0
         3 DISEÑO                    0          0
         4 DESARROLLO                0          0


DNI       NOMBRE       FECHANAC   CP    S     SUELDO     NUMDEP
--------- ------------ ---------- ----- - ---------- ----------
111111111 Sánchez      15-11-1997 10005 M      35000          1
222222222 Martínez     12-12-1991 06800 M      40000          1
333333333 Álvarez      21-08-1990 10800 H      30000          1
444444444 González     12-09-1994 06002 H      28000          1
555555555 Martín       11-03-1989 10005 M      29000          2
666666666 Lagos        07-07-1991 06800 M      27000          2
777777777 Salazar      22-07-1993 06300 H      32000          2
888888888 López        10-11-1994 10300 H      32000          2
```

```
123456789 Pérez       15-11-1967 06400 H      36000          3
666884444 Ojeda       12-12-1991 06300 H      37000          3
666999333 Ruiz        01-02-1990 10300 H      25000          3
999999999 Simón       31-08-1988 10600 M      33000          3
333445555 Campos      12-04-1974 06002 M      50000          4
222447777 Torres      30-05-1988 10600 H      25000          4
987654321 Jiménez     10-04-1971 06400 M      40000          4
000000000 Sevilla     17-04-1980 10800 M      45000          4

16 filas seleccionadas.
```

Todas las explicaciones relativas al código de ejemplo de este guion se han ofrecido en las sesiones de teoría de la asignatura, por lo que no se repiten aquí. Esta documentación básicamente ilustra el código fuente y las salidas que origina, y no debe seguirse como ejemplo para la entrega que cada uno debe realizar: en las entregas de los estudiantes debe explicarse claramente cada acción del código que se entregue.

3. Código PL/SQL de ejemplo

A continuación, se presentan algunos de los ejemplos que se han visto en las clases de teoría.

Variables, tipos, comentarios, mostrar datos…

Como primer ejemplo, se presenta el código de las clases de teoría, incluido en el archivo **PLSQL_01.sql**. Principalmente se ilustra la declaración de variables, de tipos, comentarios, la forma de mostrar datos en pantalla…

```
SET SERVEROUTPUT ON
DECLARE
   v NUMBER := 2;
BEGIN
   DBMS_OUTPUT.ENABLE;
   v:=v*2;
   DECLARE
      z NUMBER := 3;
   BEGIN
      z:=v*3;
      DBMS_OUTPUT.PUT_LINE(v); --escribe 4
      DBMS_OUTPUT.PUT_LINE(z); --escribe 12
   END;
   DBMS_OUTPUT.PUT_LINE(v*2); --escribe 8
   --DBMS_OUTPUT.PUT_LINE(z); --error, ya no es accesible
END;
```

```
Salida de Script
4
12
8
```

En el archivo **PLSQL_02.sql** se incluye un nuevo ejemplo que ilustra la **declaración de variables** del tipo de algún atributo de la tabla EMPLEADOS, haciendo uso de `%TYPE`.

```
SET SERVEROUTPUT ON
DECLARE
   sueldo_min empleados.sueldo%TYPE;
   sueldo_max empleados.sueldo%TYPE;
   sueldo_med empleados.sueldo%TYPE;
   total_emp NUMBER(10,2);
   mensaje VARCHAR2(300);

BEGIN
   DBMS_OUTPUT.ENABLE;
   SELECT min(sueldo),max(sueldo),avg(sueldo),count(*) INTO sueldo_min, sueldo_max, sueldo_med,
total_emp
   FROM empleados;

   mensaje:='sueldo máximo: '||sueldo_max||' - sueldo mínimo: '||sueldo_min||' - sueldo medio:
'||sueldo_med||' - total empleados: '||total_emp;

   DBMS_OUTPUT.PUT_LINE(mensaje);
END;
```

Salida de Script

```
sueldo máximo: 50000 - sueldo mínimo: 25000 - sueldo medio: 34000 - total empleados: 16
```

Funciones, procedimientos, sentencias de bifurcación y estructuras de control

En el archivo **PLSQL_03.sql** se incluyen los ejemplos que ilustran la creación de **funciones** y **procedimientos**. En ellos, se muestran ejemplos de **estructuras de bifurcación** (`if`, `if`/`else`, `if`/`elsif`/`else`, `case`) y **estructuras de control** (`loop`, `while`, `for`, `for`/`reverse`).

```
SET SERVEROUTPUT ON

DECLARE
   mensaje VARCHAR(100);

/* ------------------------------------------------------------------------------------------ */

-- Ejemplo: Funcion para encontrar el mayor de 3 numeros
function Mayor3_1 (n1 number, n2 number, n3 number) return number is
   mayor number := 0;
begin
   DBMS_OUTPUT.PUT_LINE('Mayor3_1');
   if (n1>mayor) then
        mayor := n1;
   end if;
   if (n2>mayor) then
        mayor := n2;
   end if;
   if (n3>mayor) then
        mayor := n3;
   end if;
   return mayor;
End;

/* ------------------------------------------------------------------------------------------ */
```

```
-- Ejemplo: Funcion para encontrar el mayor de 3 numeros

function Mayor3_2 (n1 number, n2 number, n3 number) return number is
   mayor number;
begin
   DBMS_OUTPUT.PUT_LINE('Mayor3_2');
   if (n1>n2) then
        mayor := n1;
   else
        mayor := n2;
   end if;
   if (n3>mayor) then
        mayor := n3;
   end if;
   return mayor;
End;

/* ---------------------------------------------------------------------------------------- */

-- Ejemplo: Clasifica sueldo de empleados

function Clasif_sueldo (p_dni empleados.dni%type)
return varchar2 is
   v_sueldo empleados.sueldo%type;
   v_mensaje varchar2(30);
begin
   DBMS_OUTPUT.PUT_LINE('ClasifImp');
   select sueldo into v_sueldo from empleados where dni = p_dni;
   if (v_sueldo<=30000) then
          v_mensaje := 'sueldo bajo';
   elsif (v_sueldo<=40000) then
          v_mensaje := 'sueldo regular';
   else
          v_mensaje := 'sueldo bueno';
   end if;
   v_mensaje := to_char(v_sueldo) || ' - ' || v_mensaje;
   return v_mensaje;
end;

/* ---------------------------------------------------------------------------------------- */

-- Ejemplo: Clasifica un numero

function ClasifNum(n number) return varchar2 is
   respuesta varchar2(30);
begin
   DBMS_OUTPUT.PUT_LINE('ClasifNum');
   case n
          when 1 then
            respuesta := 'Campeón';
          when 2 then
            respuesta := 'Subcampeón';
          else
            respuesta := 'Resto';
          end case;
   return respuesta;
End;

/* ---------------------------------------------------------------------------------------- */

-- Ejemplo: Clasifica sueldo de empleados

function Clasif_sueldo2(p_dni empleados.dni%type)
return varchar2 is
   v_mensaje varchar2(40);
   v_sueldo empleados.sueldo%type;
begin
   DBMS_OUTPUT.PUT_LINE('ClasifImp2');
   select sueldo into v_sueldo from empleados where dni = p_dni;
   case
       when (v_sueldo > 0 and v_sueldo <= 30000) then
          v_mensaje := 'sueldo bajo';
       when (v_sueldo > 30000 and v_sueldo <= 40000) then
```

```
            v_mensaje := 'sueldo regular';
        when (v_sueldo > 40000) then
            v_mensaje := 'sueldo bueno';
            else
                v_mensaje := 'Caso Desconocido';
    end case;
    v_mensaje := to_char(v_sueldo) || ' - ' || v_mensaje;
    return v_mensaje;
end;

/* ------------------------------------------------------------------------------------------ */

-- Ejemplo: Factorial de un número

function Factorial1 (n number) return number is
    f number := 1;
    cont number := n;
begin
    DBMS_OUTPUT.PUT_LINE('Factorial1');
    loop
          f := f * cont;
          cont := cont - 1;
          exit when (cont=0);
    end loop;
    return f;
end;

/* ------------------------------------------------------------------------------------------ */

-- Ejemplo: Escribe tabla de multiplicar

procedure TablaMulti1 (n number) is
    k number := 1;
begin
    DBMS_OUTPUT.PUT_LINE('TablaMulti1');
    dbms_output.put_line('Tabla de multiplicar del '||n);
    while (k<=10) loop
          dbms_output.put_line(n  || ' x ' || k || ' = ' || n*k);
          k := k + 1;
    end loop;
End;

/* ------------------------------------------------------------------------------------------ */

-- Ejemplo: Factorial de un número

function Factorial2 ( n number ) return number is
    f number := 1; -- No es necesario inicializar ni declarar k
begin
    DBMS_OUTPUT.PUT_LINE('Factorial2');
    for k in 1 .. n loop
          f := f * k;
    end loop;
    return f;
End;

/* ------------------------------------------------------------------------------------------ */

-- Ejemplo: Tabla de multiplicar, en orden inverso

procedure TablaMulti2 ( n number ) is
begin
    DBMS_OUTPUT.PUT_LINE('TablaMulti2');
    for k in reverse 1 .. 10 loop
          dbms_output.put_line(n  || ' x ' || k || ' = ' || n*k);
    end loop;
end;

/* ------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------ */
```

```
BEGIN
   --DBMS_OUTPUT.ENABLE;

   mensaje:='Funcion Mayor3_1(13,08,30): '||Mayor3_1 (13,08,30);
   DBMS_OUTPUT.PUT_LINE(mensaje);

   mensaje:='Funcion Mayor3_2(13,08,30): '||Mayor3_2 (13,08,30);
   DBMS_OUTPUT.PUT_LINE(mensaje);

   mensaje:='Funcion Mayor3_2(13,08,30): '||Mayor3_2 (13,08,30);
   DBMS_OUTPUT.PUT_LINE(mensaje);

   mensaje:='Funcion Clasif_sueldo(''123456789''): '||Clasif_sueldo('123456789');
   DBMS_OUTPUT.PUT_LINE(mensaje);

   mensaje:='Funcion ClasifNum(1): '||ClasifNum(1);
   DBMS_OUTPUT.PUT_LINE(mensaje);

   mensaje:='Funcion ClasifNum(2): '||ClasifNum(2);
   DBMS_OUTPUT.PUT_LINE(mensaje);

   mensaje:='Funcion Clasif_sueldo2(''123456789''): '||Clasif_sueldo2('123456789');
   DBMS_OUTPUT.PUT_LINE(mensaje);

   mensaje:='Funcion Factorial1(5): '||Factorial1(5);
   DBMS_OUTPUT.PUT_LINE(mensaje);

   mensaje:='Funcion TablaMulti1(8): ';
   DBMS_OUTPUT.PUT_LINE(mensaje);
   TablaMulti1(8);

   mensaje:='Funcion Factorial2(5): '||Factorial2(5);
   DBMS_OUTPUT.PUT_LINE(mensaje);

   mensaje:='Funcion TablaMulti2(7): ';
   DBMS_OUTPUT.PUT_LINE(mensaje);
   TablaMulti2(7);
END;
```

Salida de Script

```
Mayor3_1
Funcion Mayor3_1(13,08,30): 30
Mayor3_2
Funcion Mayor3_2(13,08,30): 30
Mayor3_2
Funcion Mayor3_2(13,08,30): 30
ClasifImp
Funcion Clasif_sueldo('123456789'): 36000 - sueldo regular
ClasifNum
Funcion ClasifNum(1): Campeón
ClasifNum
Funcion ClasifNum(2): Subcampeón
ClasifImp2
Funcion Clasif_sueldo2('123456789'): 36000 - sueldo regular
Factorial1
Funcion Factorial1(5): 120
Funcion TablaMulti1(8):
TablaMulti1
Tabla de multiplicar del 8
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80
```

```
Factorial2
Funcion Factorial2(5): 120
Funcion TablaMulti2(7):
TablaMulti2
7 x 10 = 70
7 x 9 = 63
7 x 8 = 56
7 x 7 = 49
7 x 6 = 42
7 x 5 = 35
7 x 4 = 28
7 x 3 = 21
7 x 2 = 14
7 x 1 = 7
```

Funciones y procedimientos con valores de entrada y de salida

En **PLSQL_04a.sql** se ilustra el uso de parámetros de entrada y de salida. Se crea una función `suma_A` que recibe dos enteros y devuelve su suma en el `return` de la función. Esta función puede llamarse sin problemas desde el bloque principal, como se puede observar abajo del todo en el ejemplo. Por su parte, en procedimiento `suma_B` tiene tres parámetros, dos de entrada y un tercero de salida, que contiene la suma. En este caso, al haber parámetros de salida no es necesario especificar un valor de retorno.

```
SET SERVEROUTPUT ON

DECLARE
   mensaje VARCHAR2(100);
   x1 NUMBER;
   x2 NUMBER;
   x3 NUMBER;

/* ------------------------------------------------------------------------------------------ */
-- Ejemplo: Funcion para sumar dos numeros
FUNCTION suma_A (n1 NUMBER, n2 NUMBER) RETURN NUMBER IS
   resultado NUMBER:=0;
BEGIN
   resultado := n1 + n2;
   DBMS_OUTPUT.PUT_LINE('LOCAL - suma_A: ' || n1 || ' + ' || n2 || ' = ' || resultado);
   RETURN resultado;
END;

/* ------------------------------------------------------------------------------------------ */
-- Ejemplo: Funcion para sumar dos numeros
PROCEDURE suma_B (n1 IN NUMBER, n2 IN NUMBER, resultado OUT NUMBER) IS
BEGIN
   resultado := n1 + n2;
   DBMS_OUTPUT.PUT_LINE('LOCAL - suma_B: ' || n1 || ' + ' || n2 || ' = ' || resultado);
END;

/* ------------------------------------------------------------------------------------------ */

BEGIN
   x1:=12;
   x2:=6;
   mensaje:='PRINCIPAL - Funcion suma_A: ' || x1 || ' + ' || x2 || ' = ' || suma_A(x1,x2);
   DBMS_OUTPUT.PUT_LINE(mensaje);

    x1:=13;
    x2:=43;
    suma_B (x1,x2,x3);
    mensaje:='PRINCIPAL: Funcion suma_B ' || x1 || ' + ' || x2 || ' = ' || x3;
   DBMS_OUTPUT.PUT_LINE(mensaje);
END;
```

Salida de Script

```
LOCAL - suma_A: 12 + 6 = 18
PRINCIPAL - Funcion suma_A: 12 + 6 = 18
LOCAL - suma_B: 13 + 43 = 56
PRINCIPAL: Funcion suma_B 13 + 43 = 56
```

Funciones y procedimientos persistentes

El fichero **PLSQL_04b.sql** se corresponde con una versión del ejemplo anterior (**PLSQL_04a.sql**) donde las funciones y procedimientos se crean y almacenan de forma persistente, de modo que, si no se eliminan (con el correspondiente DROP…), permanecerían en el espacio del usuario para usos posteriores. OJO, que al definirse varios entornos diferentes **es obligatorio separarlos mediante la barra /**, para que se pueda ejecutar el script sin problemas.

```
SET SERVEROUTPUT ON
```

FUERA DEL BLOQUE DE CÓDIGO

```
/* ------------------------------------------------------------------------------------------ */
-- Ejemplo: Funcion para sumar dos numeros
CREATE OR REPLACE FUNCTION suma_A (n1 NUMBER, n2 NUMBER) RETURN NUMBER IS
   resultado NUMBER:=0;
BEGIN
   resultado := n1 + n2;
   DBMS_OUTPUT.PUT_LINE('LOCAL - suma_A: ' || n1 || ' + ' || n2 || ' = ' || resultado);
   RETURN resultado;
END;
/
```

```
/* ------------------------------------------------------------------------------------------ */
-- Ejemplo: Funcion para sumar dos numeros
CREATE OR REPLACE PROCEDURE suma_B (n1 IN NUMBER, n2 IN NUMBER, resultado OUT NUMBER) IS
BEGIN
   resultado := n1 + n2;
   DBMS_OUTPUT.PUT_LINE('LOCAL - suma_B: ' || n1 || ' + ' || n2 || ' = ' || resultado);
END;
/
```

```
DECLARE
   mensaje VARCHAR2(100);
   x1 NUMBER;
   x2 NUMBER;
   x3 NUMBER;
/* ------------------------------------------------------------------------------------------ */
BEGIN
   x1:=12;
   x2:=6;
   mensaje:='PRINCIPAL - Funcion suma_A: ' || x1 || ' + ' || x2 || ' = ' || suma_A(x1,x2);
   DBMS_OUTPUT.PUT_LINE(mensaje);

    x1:=13;
    x2:=43;
    suma_B (x1,x2,x3);
    mensaje:='PRINCIPAL: Funcion suma_B ' || x1 || ' + ' || x2 || ' = ' || x3;
   DBMS_OUTPUT.PUT_LINE(mensaje);
END;
/
```

```
DROP FUNCTION suma_A;
DROP PROCEDURE suma_B;
```

Salida de Script

```
Function SUMA_A compilado
Procedure SUMA_B compilado

LOCAL - suma_A: 12 + 6 = 18
PRINCIPAL - Funcion suma_A: 12 + 6 = 18
LOCAL - suma_B: 13 + 43 = 56
PRINCIPAL: Funcion suma_B 13 + 43 = 56

Procedimiento PL/SQL terminado correctamente.

Function SUMA_A borrado.
Procedure SUMA_B borrado.
```

Básicamente, se trata de crear la función y el procedimiento antes del bloque de código (antes del DECLARE con el que se inicia el bloque de código). Y, tras el bloque de código, deben incluirse las sentencias las sentencias DROP correspondientes para borrar estas funciones (de no eliminarse, quedarían almacenadas de forma persistente, en el espacio del usuario).

Registros

Para ilustrar el uso de registros, siguiendo con los ejemplos vistos en las clases de teoría, se incluye el archivo **PLSQL_05.sql**.

```
SET SERVEROUTPUT ON

DECLARE
   mensaje VARCHAR2(100);

/* ------------------------------------------------------------------------------------------ */

-- Ejemplo: Procedimiento para consultar el nombre y sueldo de una tupla de empleados
procedure ConsultaEMP (pdni empleados.dni%type) is

   type regproy is record (
          rnombre empleados.nombre%type,
          rsueldo empleados.sueldo%type
   );
   r regproy;

begin
   dbms_output.put_line('Procedimiento ConsultaEMP');
   select nombre, sueldo into r from empleados where dni = pdni;
   dbms_output.put_line('Nombre: ' || r.rnombre );
   dbms_output.put_line('sueldo: ' || r.rsueldo );
End;

/* ------------------------------------------------------------------------------------------ */

-- Ejemplo: Procedimiento para consultar los datos de un dpto
procedure ConsultaDEP(pnumerodpto departamentos.numerodpto%type ) is
   r departamentos%rowtype;
begin
   dbms_output.put_line('Procedimiento ConsultaDEP');
   select * into r from departamentos where numerodpto = pnumerodpto;
   dbms_output.put_line('Número dpto: ' || r.numerodpto);
   dbms_output.put_line('Nombre dpto: ' || r.nombredpto);
   dbms_output.put_line('Coste dpto: ' || r.coste);
end;

/* ------------------------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------------------------ */
```

```
/* ---------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------- */

BEGIN
   DBMS_OUTPUT.ENABLE;

   ConsultaEMP('123456789');
   ConsultaDEP('3');

   mensaje := 'Fin de programa';
   dbms_output.put_line(mensaje);
END;
```

Salida de Script

```
Procedimiento ConsultaEMP
Nombre: Pérez
sueldo: 36000
Procedimiento ConsultaDEP
Número dpto: 3
Nombre dpto: DISEÑO
Coste dpto: 0
Fin de programa
```

Cursores

El archivo **PLSQL_06.sql** incluye ejemplos vistos en las clases, relacionados con los cursores.

```
SET SERVEROUTPUT ON

DECLARE
   mensaje VARCHAR2(100);

/* ---------------------------------------------------------------------------------------------- */

-- Ejemplo: Procedimiento para consultar el nombre y sueldo de una tupla de empleados
procedure ConsultaEMP (pdni empleados.dni%type) is

   type regproy is record (
          rnombre empleados.nombre%type,
          rsueldo empleados.sueldo%type
   );
   r regproy;

begin
   dbms_output.put_line('Procedimiento ConsultaEMP');
   select nombre, sueldo into r from empleados where dni = pdni;
   dbms_output.put_line('Nombre: ' || r.rnombre );
   dbms_output.put_line('sueldo: ' || r.rsueldo );
End;

/* ---------------------------------------------------------------------------------------------- */

-- Ejemplo: Procedimiento para consultar los datos de un dpto
procedure ConsultaDEP(pnumerodpto departamentos.numerodpto%type ) is
   r departamentos%rowtype;
begin
   dbms_output.put_line('Procedimiento ConsultaDEP');
   select * into r from departamentos where numerodpto = pnumerodpto;
   dbms_output.put_line('Número dpto: ' || r.numerodpto);
   dbms_output.put_line('Nombre dpto: ' || r.nombredpto);
   dbms_output.put_line('Coste dpto: ' || r.coste);
end;

/* ---------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------- */
```

```
/* ---------------------------------------------------------------------------------------------- */
/* ---------------------------------------------------------------------------------------------- */

BEGIN
   DBMS_OUTPUT.ENABLE;

   ConsultaEMP('123456789');
   ConsultaDEP('3');

   mensaje := 'Fin de programa';
   dbms_output.put_line(mensaje);
END;
```

Salida de Script

```
Número departamento: 1
Nombre departamento: PERSONAL
Coste: 0
111111111 - Sánchez - 1
222222222 - Martínez - 1
333333333 - Álvarez - 1
444444444 - González - 1
555555555 - Martín - 2
666666666 - Lagos - 2
777777777 - Salazar - 2
888888888 - López - 2
123456789 - Pérez - 3
666884444 - Ojeda - 3
666999333 - Ruiz - 3
999999999 - Simón - 3
333445555 - Campos - 4
222447777 - Torres - 4
987654321 - Jiménez - 4
000000000 - Sevilla - 4
Fin de programa
```

Excepciones

El archivo **PLSQL_07.sql** contiene algunas versiones de los ejemplos expuestos en clases.

```
SET SERVEROUTPUT ON

DECLARE
   mensaje VARCHAR2(100);

/* ---------------------------------------------------------------------------------------------- */

-- Procedimiento para consultar el sueldo de una tupla de Empleados
PROCEDURE FindEmp(pdni Empleados.dni%TYPE) IS

   v_sueldo Empleados.sueldo%TYPE;

BEGIN
   SELECT sueldo INTO v_sueldo FROM Empleados WHERE dni = pdni;
   DBMS_OUTPUT.PUT_LINE( 'FindEmp - sueldo: ' || v_sueldo );
   EXCEPTION
      WHEN No_Data_Found THEN DBMS_OUTPUT.PUT_LINE( 'FindEmp - EXCEPCION: dni no existe.' );
END;

/* ---------------------------------------------------------------------------------------------- */
```

```
-- Permite generar una excepcion (lanzar una excepción)
PROCEDURE UpdatesueldoEmp (pdni Empleados.dni%TYPE, p_sueldo Empleados.sueldo%TYPE) IS
   Cont Number;
BEGIN
   SELECT Count(*) INTO Cont FROM Empleados WHERE dni = pdni;
   IF (Cont=0) THEN
      RAISE No_Data_Found; -- lanza la excepcion
   END IF;
   UPDATE Empleados SET sueldo = p_sueldo WHERE dni = pdni;
   Commit;
   DBMS_OUTPUT.PUT_LINE( 'UpdatesueldoEmp - Proceso OK' );
   EXCEPTION -- captura la excepción
      WHEN No_Data_Found THEN DBMS_OUTPUT.PUT_LINE( 'UpdatesueldoEmp - EXCEPCION: dni no
existe.' );
END;

/* ---------------------------------------------------------------------------------------- */

-- Mismo procedimiento, pero con excepcion de usuario
PROCEDURE UpdatesueldoEmp2 (pdni Empleados.dni%TYPE, p_sueldo Empleados.sueldo%TYPE) IS
   Cont Number;
   Excep1 EXCEPTION;
BEGIN
   SELECT Count(*) INTO Cont FROM Empleados WHERE dni = pdni;
   IF (Cont=0) THEN
      RAISE Excep1; -- lanza la excepcion
   END IF;
   UPDATE Empleados SET sueldo = p_sueldo WHERE dni = pdni;
   Commit;
   DBMS_OUTPUT.PUT_LINE( 'UpdatesueldoEmp2 - Proceso OK' );
   EXCEPTION -- captura la excepción
      WHEN Excep1 THEN DBMS_OUTPUT.PUT_LINE( 'UpdatesueldoEmp2 - EXCEPCION: dni no existe.' );
END;

/* ---------------------------------------------------------------------------------------- */

-- Mismo procedimiento, pero con mensaje de error
PROCEDURE UpdatesueldoEmp3 (pdni Empleados.dni%TYPE, p_sueldo Empleados.sueldo%TYPE) IS
   Cont Number;
BEGIN
   SELECT Count(*) INTO Cont FROM Empleados WHERE dni = pdni;
   IF (Cont=0) THEN
      RAISE_APPLICATION_ERROR( -20000, 'UpdatesueldoEmp3 - EXCEPCION: dni no existe.' );
   END IF;
   UPDATE Empleados SET sueldo = p_sueldo WHERE dni = pdni;
   Commit;
   DBMS_OUTPUT.PUT_LINE( 'UpdatesueldoEmp3 - Proceso OK' );
   -- ahora no se atiende la excepcion
END;

/* ---------------------------------------------------------------------------------------- */

BEGIN
   DBMS_OUTPUT.ENABLE;

   FindEmp('123456789'); -- Ok
   FindEmp('MAL---MAL'); -- Excepcion

   UpdatesueldoEmp('123456789',4500); -- Ok
   UpdatesueldoEmp('MAL---MAL',4500); -- Excepcion

   UpdatesueldoEmp2('123456789',5000); -- Ok
   UpdatesueldoEmp2('MAL---MAL',5000); -- Excepcion

   UpdatesueldoEmp3('123456789',5500); -- Ok - se vuelve a actualizar con el valor original
   UpdatesueldoEmp3('MAL---MAL',5500); -- Excepcion

   mensaje := 'Fin de programa';
   DBMS_OUTPUT.PUT_LINE(mensaje);

   COMMIT;
END;
```

A continuación, se incluye el reporte de actividad que ofrece ORACLE. Cabe destacar que **no se muestra el mensaje 'Fin de programa'**, puesto que en el procedimiento UpdateParPer3 **el sistema ejecuta una excepción invocando a un mensaje de error del sistema**. Por tanto, el programa no continúa: **no imprime el mensaje final de 'Fin de programa'**:

```
Salida de Script

FindEmp - sueldo: 36000
FindEmp - EXCEPCION: dni no existe.
UpdatesueldoEmp - Proceso OK
UpdatesueldoEmp - EXCEPCION: dni no existe.
UpdatesueldoEmp2 - Proceso OK
UpdatesueldoEmp2 - EXCEPCION: dni no existe.
UpdatesueldoEmp3 - Proceso OK


Error que empieza en la línea: 3 del comando :

FindUsu - megabits: 550
FindUsu - EXCEPCION: cod no existe.
UpdatemegabitsUsu - Proceso OK
UpdatemegabitsUsu - EXCEPCION: cod no existe.
UpdatemegabitsUsu2 - Proceso OK
UpdatemegabitsUsu2 - EXCEPCION: cod no existe.
UpdatemegabitsUsu3 - Proceso OK


Error que empieza en la línea: 3 del comando :
[ ... CÓDIGO FUENTE ...]
Informe de error -
ORA-20000: UpdatesueldoEmp3 - EXCEPCION: dni no existe.
ORA-06512: en línea 61
ORA-06512: en línea 88
20000. 00000 -  "%s"
*Cause:    The stored procedure 'raise_application_error'
           was called which causes this error to be generated.
*Action:   Correct the problem as described in the error message or contact
           the application administrator or DBA for more information.
```

Excepciones predefinidas

Excepción	Descripción
ACCESS_INTO_NULL	El programa intentó asignar valores a los atributos de un objeto no inicializado -6530.
COLLECTION_IS_NULL	El programa intentó asignar valores a una tabla anidada aún no inicializada -6531.
CURSOR_ALREADY_OPEN	El programa intentó abrir un cursor que ya se encontraba abierto. Recuerde que un cursor de ciclo FOR automáticamente lo abre y ello no se debe especificar con la sentencia OPEN -6511.
DUP_VAL_ON_INDEX	El programa intentó almacenar valores duplicados en una columna que se mantiene con restricción de integridad de un índice único (unique index) -1.
INVALID_CURSOR	El programa intentó efectuar una operación no válida sobre un cursor -1001.
INVALID_NUMBER	En una sentencia SQL, la conversión de una cadena de caracteres hacia un número falla cuando esa cadena no representa un número válido -1722.
LOGIN_DENIED	El programa intentó conectarse a Oracle con un nombre de usuario o password inválido -1017.
NO_DATA_FOUND	Una sentencia SELECT INTO no devolvió valores o el programa referenció un elemento no inicializado en una tabla indexada 100.
NOT_LOGGED_ON	El programa efectuó una llamada a Oracle sin estar conectado -1012.
PROGRAM_ERROR PL/SQL	Tiene un problema interno -6501.

ROWTYPE_MISMATCH	Los elementos de una asignación (el valor a asignar y la variable que lo contendrá) tienen tipos incompatibles. También se presenta este error cuando un parámetro pasado a un subprograma no es del tipo esperado -6504.
SELF_IS_NULL	El parámetro SELF (el primero que es pasado a un método MEMBER) es nulo -30625.
STORAGE_ERROR	La memoria se terminó o está corrupta -6500.
SUBSCRIPT_BEYOND_COUNT	El programa está tratando de referenciar un elemento de una colección indexada que se encuentra en una posición más grande que el número real de elementos de la colección -6533.
SUBSCRIPT_OUTSIDE_LIMIT	El programa está referenciando un elemento de una tabla utilizando un número fuera del rango permitido (por ejemplo, el elemento "-1") -6532.
SYS_INVALID_ROWID	La conversión de una cadena de caracteres hacia un tipo ROWID falló porque la cadena no representa un número -1410.
TIMEOUT_ON_RESOURCE	Se excedió el tiempo máximo de espera por un recurso en Oracle -51.
TOO_MANY_ROWS	Una sentencia SELECT INTO devuelve más de una fila -1422.
VALUE_ERROR	Ocurrió un error aritmético, de conversión o truncamiento. Por ejemplo, sucede cuando se intenta introducir un valor muy grande dentro de una variable más pequeña -6502.
ZERO_DIVIDE	El programa intentó efectuar una división por cero -1476.

4. Ejercicios de PL/SQL

Realizar los siguientes ejercicios:

A) Implementar las funciones o procedimientos necesarios para realizar las acciones básicas sobre bases de datos: **altas**, **bajas**, **modificaciones** y **consultas**. Realizar estas tareas sobre la tabla **Departamentos**, de modo que se siga la siguiente secuencia:

- **A.1)** Inserción de un nuevo registro en la tabla (`InsertarDepartamentos(...)`), y ver todos los registros de esa tabla, incluido el que se acaba de insertar (`VerDepartamentos()`).
- **A.2)** Modificación del registro recién insertado (`ModificarDepartamentos (...)`), y ver todos los registros de esa tabla, incluido el modificado (`VerDepartamentos()`).
- **A.3)** Declarar las variables adecuadas:

```
DECLARE
     vnombre VARCHAR2(20);
     vcoste NUMBER(9,2);
     vporcent NUMBER(3,2);
```

  Implementar una función o procedimiento de consulta que retorne en estas variables (como parámetros de salida) los valores de Departamentos cuyo `numerodpto` se pase como parámetro de entrada. Por ejemplo, retornar los datos del departamento 3.

- **A.4)** Borrado del registro que se ha insertado y modificado, y ver todos los registros de la tabla, donde ya no estará el que se ha insertado (`VerDepartamentos()`).

Por si sirve de ayuda... se ofrece el siguiente guion:

```
SET SERVEROUTPUT ON;

DECLARE
```

```
    vnombre VARCHAR2(20);
    vcoste NUMBER(9,2);
    vporcent NUMBER(3,2);
    mensaje VARCHAR2(300);

/* ---------------------------------------------------------------------- */
-- Procedimientos... (A.1, A.2, A.3, A.4)
/* ---------------------------------------------------------------------- */

BEGIN
    DBMS_OUTPUT.ENABLE;

    -- A.1)
    InsertarDepartamentos (0,'CONTABILIDAD',0,0);
    VerDepartamentos();

    -- A.2)
    ModificarDepartamentos (0,'FINANZAS',0,0);
    VerDepartamentos();

    -- A.3)
    ConsultarDepartamentos (0, vnombre, vcoste, vporcent);
    dbms_output.put_line('Nombre departamento: ' || vnombre);
    dbms_output.put_line('Coste: ' || vcoste);
    dbms_output.put_line('Porcent: ' || vporcent);

    -- A.4)
    BorrarDepartamentos (0);
    VerDepartamentos();

    mensaje := 'Fin de programa';
    DBMS_OUTPUT.PUT_LINE(mensaje);
END;
```

B) Realizar versiones de los procedimientos anteriores, de modo que se gestionen las excepciones de forma adecuada. Llamar a los procedimientos/funciones con el mismo nombre, pero esta vez terminados en 2 (`InsertarDepartamentos2`, `ModificarDepartamentos2`, `ConsultarDepartamentos2` y `BorrarDepartamentos2`).

5. Estudio a realizar

- Probar el código facilitado y desarrollar los ejercicios propuestos en esta sesión.

EXTRA POINT: Con carácter voluntario, se presenta la posibilidad de obtener un *EXTRA POINT* a quienes deseen profundizar más en los temas presentados en este guion práctico. Para ello, en la memoria a entregar, se deberá incluir (además de los apartados obligatorios del guion) un nuevo apartado denominado EXTRA POINT e identificado con el logo BONUS, donde se presente algún aspecto novedoso o diferenciador con respecto a la propuesta práctica base.

- **Entregables**: debe entregarse

- **OPCIÓN (1)** – OBLIGATORIO

  (1) El código fuente correspondiente a los ejercicios debidamente resueltos.
  Los ejercicios deben implementarse y documentarse **para todos los SGBD**

  (2) Un PDF en el que se documente:
  a) Un anexo completo con **las salidas obtenidas tras la ejecución del script completo del código desarrollado**. **Las salidas obtenidas deben incluirse en la documentación en modo texto y no como una captura de imágenes.**

- **OPCIÓN (2)** – OPTATIVO, además del punto (1) obligatorio:

  a) Todo el **código fuente** desarrollado y entregado en esta sesión práctica, ampliamente **comentado y documentado**, con las explicaciones necesarias y las **salidas parciales** obtenidas por cada fragmento de código. Si alguna función/procedimiento del código fuente es idéntica en todos los SGBD, es suficiente con explicar esa función/procedimiento una única vez. No es preciso documentar los ejercicios resueltos en este guion y que no formen parte del código del estudiante, de modo que solo es preciso comentar las funciones usadas en los ejercicios propuestos. **El código fuente debe incluirse en la documentación en modo texto y no como una captura de imágenes, tal y como se presenta en los guiones prácticos de la asignatura**.
  b) Particularmente, debe explicarse detalladamente todos los **aspectos novedosos** que incluyen el tema que se presenta.
  c) Todas las capturas, salidas, código… que se incluyan en la documentación deben ser **perfectamente legibles**, presentándose con la máxima claridad y corrección.