Programación de Bases de Datos

PL-pgSQL sobre PostgreSQL

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 (PostgreSQL) y el entorno que facilita toda la gestión de consultas (Pg Admin IV).

- PostgreSQL - (*Ya instalado en 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 la primera práctica, habría que volver a crearlas e insertar las tuplas correspondientes, mediante los archivos facilitados en la primera inicial. Así, se tendrán las siguientes tablas sobre la que trabajar:

```
SELECT * FROM empleados;
```

"dni"	"nombre"	"fechanac"	"cp"	"sexo"	"sueldo"	"numdep"		
"111111111"		"Sánchez"	"15-11-1997"		"10005"	"M"	35000.00	1
"222222222"		"Martínez"	"12-12-1991"		"06800"	"M"	40000.00	1
"333333333"		"Álvarez"	"21-08-1990"		"10800"	"H"	30000.00	1
"444444444"		"González"	"12-09-1994"		"06002"	"H"	28000.00	1
"555555555"		"Martín"	"11-03-1989"		"10005"	"M"	29000.00	2
"666666666"		"Lagos"	"07-07-1991"		"06800"	"M"	27000.00	2
"777777777"		"Salazar"	"22-07-1993"		"06300"	"H"	32000.00	2
"888888888"		"López"	"10-11-1994"		"10300"	"H"	32000.00	2
"123456789"		"Pérez"	"15-11-1967"		"06400"	"H"	36000.00	3
"666884444"		"Ojeda"	"12-12-1991"		"06300"	"H"	37000.00	3
"666999333"		"Ruiz"	"01-02-1990"		"10300"	"H"	25000.00	3
"999999999"		"Simón"	"31-08-1988"		"10600"	"M"	33000.00	3
"333445555"		"Campos"	"12-04-1974"		"06002"	"M"	50000.00	4
"222447777"		"Torres"	"30-05-1988"		"10600"	"H"	25000.00	4
"987654321"		"Jiménez"	"10-04-1971"		"06400"	"M"	40000.00	4
"000000000"		"Sevilla"	"17-04-1980"		"10800"	"M"	45000.00	4

```
SELECT * FROM departamentos;
```

"numerodpto"	"nombredpto"	"coste"	"porcent"
1	"PERSONAL" 0.00	0.00	
2	"PRODUCCIÓN"	0.00	0.00
3	"DISEÑO" 0.00	0.00	
4	"DESARROLLO"	0.00	0.00

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-pgSQL 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 **plpgSQL_01.sql**. Principalmente se ilustra la declaración de variables, de tipos, comentarios, la forma de mostrar datos en pantalla…

```
-- Primer ejemplo
CREATE OR REPLACE FUNCTION ejemplo01() RETURNS integer AS $$
DECLARE
  v integer := 2;
BEGIN
  v:=v*2;
  DECLARE
    z integer := 3;
  BEGIN
    z:=v*3;
    RAISE NOTICE 'v = %',v; -- escribe 4
    RAISE NOTICE 'z = %',z; -- escribe 12
  END;
  RAISE NOTICE 'v*2 = %',v*2; -- escribe 8
  --RAISE NOTICE 'z = %',z; -- error, ya no es accesible

  RETURN v;
END;
$$ LANGUAGE plpgsql;

select ejemplo01();

DROP FUNCTION ejemplo01();
```

La sintaxis es muy parecida a **PL/SQL**: debe declararse la cabecera de la función y a continuación la declaración de variables de esa función, comenzando por **DECLARE**. Hay algunas **diferencias con respecto a PL/SQL**:

- Como se observa, para mostrar un mensaje en la consola de mensajes es necesario especificarlo mediante `RAISE NOTICE 'Mensaje'`.
- No se pueden declarar variables globales en PL-pgSQL.
- Hay que especificar en la cabecera de la función el valor que va a devolver, y posteriormente un RETURN con algún valor del tipo indicado arriba. Si no se desea devolver nada, se indica un null en el RETURN.
- Tras el `END;` hay que indicar el lenguaje en el que se trabaja, ya que PostgreSQL tiene más de un lenguaje procedimental: `$$ LANGUAGE plpgsql;`
- En PL-pgSQL no hay procedimientos. Por tanto, cualquier codificación procedente de PL/SQL del tipo CREATE OR REPLACE PROCEDURE … hay que cambiarla a tipo función (CREATE OR REPLACE FUNCTION… con un RETURN null; al final de la función).

La salida de la función anterior (**plpgSQL_01.sql**), pinchando en la pestaña "MESSAGES" de PgAdmin, sería:

```
Salida de Script
NOTICE:  v = 4
NOTICE:  z = 12
NOTICE:  v*2 = 8
DROP FUNCTION
```

En el archivo **PlpgSQL_02.sql** se incluye un nuevo ejemplo que ilustra la **declaración de variables** del tipo de algún atributo de la tabla EMPLEADO, haciendo uso de **%TYPE**.

```
CREATE OR REPLACE FUNCTION ejemplo02() RETURNS varchar AS $$
DECLARE
   sueldo_min empleados.sueldo%TYPE;
   sueldo_max empleados.sueldo%TYPE;
   sueldo_med empleados.sueldo%TYPE;
   total_pob NUMERIC(2);
   mensaje VARCHAR(300);

BEGIN
   SELECT min(sueldo),max(sueldo),avg(sueldo),count(*) INTO sueldo_min, sueldo_max, sueldo_med,
total_pob
   FROM empleados;

   mensaje:='sueldo maximo: '||sueldo_max||' - sueldo minimo: '||sueldo_min||' - sueldo medio:
'||sueldo_med||' - total empleados: '||total_pob;
   raise notice '%',mensaje;
   RETURN mensaje;
END;
$$ LANGUAGE plpgsql;

select ejemplo02();

DROP FUNCTION ejemplo02();
```

La salida sería:

```
Salida de Script
NOTICE: sueldo maximo: 50000.00 - sueldo minimo: 25000.00 - sueldo medio: 34000.00 - total
empleados: 16
```

Funciones, procedimientos, sentencias de bifurcación y estructuras de control

En el archivo **PlpgSQL_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`).

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

-- Ejemplo: Funcion para encontrar el mayor de 3 numeros
CREATE OR REPLACE function Mayor3_1 (n1 integer, n2 integer, n3 integer) RETURNS integer AS $$
DECLARE
   mayor integer:=0;
begin
   raise notice 'Mayor3_1';
   if (n1>mayor) then
       mayor := n1;
   end if;
   if (n2>mayor) then
       mayor := n2;
   end if;
   if (n3>mayor) then
       mayor := n3;
   end if;
   raise notice 'Mayor3_1: El mayor es %',mayor;
   return mayor;
End;
$$ LANGUAGE plpgsql;

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

-- Ejemplo: Funcion para encontrar el mayor de 3 numeros
CREATE OR REPLACE function Mayor3_2 (n1 integer, n2 integer, n3 integer) RETURNS integer AS $$
DECLARE
   mayor integer;
begin
    raise notice 'Mayor3_2';

   if (n1>n2) then
       mayor := n1;
   else
       mayor := n2;
   end if;
   if (n3>mayor) then
       mayor := n3;
   end if;
   raise notice 'Mayor3_2: El mayor es %',mayor;
   return mayor;
End;
$$ LANGUAGE plpgsql;

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

-- Ejemplo: Clasifica sueldo de empleados

CREATE OR REPLACE function Clasif_sueldo (p_dni empleados.dni%type) RETURNS varchar AS $$
DECLARE
   v_sueldo empleados.sueldo%type;
   v_mensaje varchar(30);
begin
   SELECT sueldo INTO v_sueldo FROM empleados WHERE dni = p_dni;
   if (v_sueldo<25) then
          v_mensaje := 'sueldo Bajo';
   elsif (v_sueldo<150) then
          v_mensaje := 'sueldo Regular';
   else
          v_mensaje := 'sueldo Bueno';
   end if;
   v_mensaje := v_sueldo || ' - ' || v_mensaje;
   raise notice 'Clasif_sueldo: %',v_mensaje;
   return v_mensaje;
end;
$$ LANGUAGE plpgsql;

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

-- Ejemplo: Clasifica un numero

CREATE OR REPLACE function ClasifNum(n numeric) RETURNS varchar AS $$
DECLARE
```

```
   respuesta varchar(30);
begin
   case n
        when 1 then
            respuesta := 'Campeón';
        when 2 then
            respuesta := 'Subcampeón';
        else
            respuesta := 'Resto';
        end case;
   raise notice 'ClasifNum: %',respuesta;
   return respuesta;
End;
$$ LANGUAGE plpgsql;

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

-- Ejemplo: Clasifica sueldo de empleados

CREATE OR REPLACE FUNCTION Clasif_sueldo2(p_dni empleados.dni%type) RETURNS varchar AS $$
DECLARE
   v_mensaje varchar(40);
   v_sueldo empleados.sueldo%type;
BEGIN
   SELECT sueldo into v_sueldo from empleados where dni = p_dni;
   case
       when v_sueldo between 0 and  30000 then
          v_mensaje := 'sueldo Bajo';
       when v_sueldo between 30000 and 40000 then
          v_mensaje := 'sueldo Regular';
       when v_sueldo > 40000 then
          v_mensaje := 'sueldo Bueno';
          else
             v_mensaje := 'Caso Desconocido';
   end case;
   v_mensaje := v_sueldo || ' - ' || v_mensaje;
   raise notice 'Clasif_sueldo2: %',v_mensaje;
   return v_mensaje;
END;
$$ language plpgsql;

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

-- Ejemplo: Factorial de un número

CREATE OR REPLACE function Factorial1 (n integer) RETURNS integer AS $$
DECLARE
   f integer := 1;
   cont integer := n;
begin
   loop
         f := f * cont;
         cont := cont - 1;
         exit when (cont=0);
   end loop;
   raise notice 'Factorial1: El factorial de % es %',n,f;
   return f;
end;
$$ LANGUAGE plpgsql;

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

-- Ejemplo: Escribe tabla de multiplicar

CREATE OR REPLACE function TablaMulti1 (n numeric) RETURNS varchar AS $$
DECLARE
   mensaje varchar(500);
   k numeric := 1;
begin
   mensaje:= 'Tabla de multiplicar del '||n;
   raise notice '%',mensaje;
   while (k<=10) loop
         mensaje:= n || ' x ' || k || ' = ' || n*k;
         raise notice '%',mensaje;
```

```
        k := k + 1;
   end loop;
   return mensaje;
End;
$$ LANGUAGE plpgsql;

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

-- Ejemplo: Factorial de un número

CREATE OR REPLACE function Factorial2 ( n integer ) RETURNS integer AS $$
DECLARE
   f integer := 1; -- No es necesario inicializarlo ni declararlo
begin
   for k in 1 .. n loop
         f := f * k;
   end loop;
   raise notice 'Factorial2: El factorial de % es %',n,f;
   return f;
End;
$$ LANGUAGE plpgsql;

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

-- Ejemplo: Escribe tabla de multiplicar

CREATE OR REPLACE function TablaMulti2 (n numeric) RETURNS varchar AS $$
DECLARE
   mensaje varchar(500);
   k numeric;
begin
   mensaje:= 'Tabla de multiplicar del '||n;
   raise notice '%',mensaje;
   for k in reverse 10 .. 1 loop
        mensaje:= n || ' x ' || k || ' = ' || n*k;
        raise notice '%',mensaje;
   end loop;
   return mensaje;
End;
$$ LANGUAGE plpgsql;

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

SELECT Mayor3_1 (13,08,30);
SELECT Mayor3_2 (13,08,30);
SELECT Clasif_sueldo ('123456789');
SELECT Clasif_sueldo2('123456789');
SELECT ClasifNum(2);
SELECT Factorial1(3);
SELECT TablaMulti1(8);
SELECT Factorial2(4);
SELECT TablaMulti2(7);

DROP function Mayor3_1 (integer,integer,integer);
DROP function Mayor3_2 (integer,integer,integer);
DROP function Clasif_sueldo (varchar);
DROP function Clasif_sueldo2 (varchar);
DROP function ClasifNum (numeric);
DROP function Factorial1 (integer);
DROP function TablaMulti1 (numeric);
DROP function Factorial2 (integer);
DROP function TablaMulti2 (numeric);
```

Salida de Script

```
NOTICE: la referencia al tipo empleados.dni%TYPE convertida a character varying
NOTICE: la referencia al tipo empleados.dni%TYPE convertida a character varying
NOTICE: Mayor3_1
NOTICE: Mayor3_1: El mayor es 30
NOTICE: Mayor3_2
```

```
NOTICE: Mayor3_2: El mayor es 30
NOTICE: Clasif_sueldo: 36000.00 - sueldo Bueno
NOTICE: Clasif_sueldo2: 36000.00 - sueldo Regular
NOTICE: ClasifNum: Subcampeón
NOTICE: Factorial1: El factorial de 3 es 6
NOTICE: Tabla de multiplicar del 8
NOTICE: 8 x 1 = 8
NOTICE: 8 x 2 = 16
NOTICE: 8 x 3 = 24
NOTICE: 8 x 4 = 32
NOTICE: 8 x 5 = 40
NOTICE: 8 x 6 = 48
NOTICE: 8 x 7 = 56
NOTICE: 8 x 8 = 64
NOTICE: 8 x 9 = 72
NOTICE: 8 x 10 = 80
NOTICE: Factorial2: El factorial de 4 es 24
NOTICE: Tabla de multiplicar del 7
NOTICE: 7 x 10 = 70
NOTICE: 7 x 9 = 63
NOTICE: 7 x 8 = 56
NOTICE: 7 x 7 = 49
NOTICE: 7 x 6 = 42
NOTICE: 7 x 5 = 35
NOTICE: 7 x 4 = 28
NOTICE: 7 x 3 = 21
NOTICE: 7 x 2 = 14
NOTICE: 7 x 1 = 7
```

Funciones con valores de entrada y de salida

En **PlpgSQL_04.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 un `SELECT` en el bloque principal, como se puede observar abajo del todo en el ejemplo.

Por su parte, la función `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 (no hay `RETURNS` en la cabecera de la función). Por otra parte, esta función no puede ser llamada desde el bloque principal, porque no se pueden declarar variables para obtener el resultado de una posible llamada a `suma_B`(3,5,x). Por tanto, debe crearse una función que haga las veces de módulo principal (aquí la hemos llamado función main). Desde esta función puede llamarse a otras funciones. En el ejemplo, se aprovecha para llamar a las dos funciones de suma (A y B). Es de destacar la llamada a la función `suma_B`: como se ha comentado, esta función tiene sus dos primeros parámetros de entrada y el tercero de salida, por lo que en la llamada `SELECT` se deben pasar dos parámetros, y en el `INTO` el tercer parámetro. Aparece destacado en fondo amarillo.

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

-- Ejemplo: Funcion para sumar dos numeros
CREATE OR REPLACE function suma_A (n1 integer, n2 integer) RETURNS integer AS $$
DECLARE
   resultado integer:=0;
begin
   raise notice 'Funcion suma_A';
   resultado := n1 + n2;
```

```
   raise notice 'suma_A: % + % = %',n1,n2,resultado;
   return resultado;
End;
$$ LANGUAGE plpgsql;

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

-- Ejemplo: Funcion para sumar dos numeros
-- En este caso, como hay parametros OUT, no permite que se retorne ningun valor (no hay
RETURNS)
CREATE OR REPLACE function suma_B (IN n1 integer, IN n2 integer, OUT resultado integer) AS $$
begin
   raise notice 'Funcion suma_B';
   resultado := n1 + n2;
   raise notice 'suma_B: % + % = %',n1,n2,resultado;

End;
$$ LANGUAGE plpgsql;

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

-- Ejemplo: Funcion para llamar a otras funciones
CREATE OR REPLACE function main () RETURNS integer AS $$
DECLARE
   resultado integer:=0;
   x1 integer;
   x2 integer;
   x3 integer:=0;
begin
   raise notice 'Funcion main';

   x1:=12;
   x2:=6;
   x3:=suma_A (x1,x2);

   x1:=13;
   x2:=43;
   select suma_B (x1,x2) into x3;
   raise notice 'Funcion main: % + % = %',x1,x2,x3;
   return NULL;
End;
$$ LANGUAGE plpgsql;


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

-- Desde el modulo principal se puede llamar a la funcion suma_A
select suma_A (12,42);
-- Pero no se puede llamar a la funcion suma_B, porque retorna un parametro de salida y no se
puede declarar previamente
-- select suma_B (12,42,¿n3?);
-- No queda mas remedio que crear una funcion (main) que llame a otras funciones...
select main();

drop function suma_A(integer,integer);
drop function suma_B(integer,integer,out integer);
drop function main();
```

Salida de Script

```
NOTICE:  Funcion suma_A
NOTICE:  suma_A: 12 + 42 = 54
NOTICE:  Funcion main
NOTICE:  Funcion suma_A
NOTICE:  suma_A: 12 + 6 = 18
NOTICE:  Funcion suma_B
NOTICE:  suma_B: 13 + 43 = 56
NOTICE:  Funcion main: 13 + 43 = 56
DROP FUNCTION
```

Registros

Para ilustrar el uso de registros, siguiendo con los ejemplos vistos en las clases de teoría, se incluye el archivo **PlpgSQL_05.sql**.

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

-- Ejemplo: Procedimiento para consultar el nombre y sueldo de una tupla de empleados
CREATE OR REPLACE FUNCTION ConsultaEMP (pdni empleados.dni%type) RETURNS varchar AS $$
DECLARE
   r record;

   salida varchar(100);
BEGIN
   SELECT nombre, sueldo INTO r FROM empleados WHERE dni = pdni;
   salida:=r.nombre || ' -- ' || r.sueldo;
   raise notice 'ConsultaEMP: %',salida;
   RETURN salida;
END;
$$ LANGUAGE plpgsql;

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

-- Ejemplo: Procedimiento para consultar los datos de un servidor
CREATE OR REPLACE FUNCTION ConsultaDEP(pnumerodpto departamentos.numerodpto%type) RETURNS
varchar AS $$
DECLARE
   r departamentos%rowtype;
    --r record; // también se podría definir simplemente como record
   salida varchar(100);
BEGIN
   SELECT * INTO r FROM departamentos WHERE numerodpto = pnumerodpto;
   salida := r.nombredpto || ' -- ' || r.numerodpto || ' -- ' || r.coste || ' -- ' || r.porcent;
   raise notice 'ConsultaDEP: %',salida;
   RETURN salida;
END;
$$ LANGUAGE plpgsql;

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

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

DROP FUNCTION ConsultaEMP(varchar);
DROP FUNCTION ConsultaDEP(numeric);
```

Salida de Script

```
NOTICE: la referencia al tipo empleados.dni%TYPE convertida a character varying
NOTICE: la referencia al tipo departamentos.numerodpto%TYPE convertida a numeric
NOTICE: ConsultaEMP: Pérez -- 36000.00
NOTICE: ConsultaDEP: DISEÑO -- 3 -- 0.00 -- 0.00
```

Cursores

El archivo **PlpgSQL_06.sql** incluye ejemplos vistos en las clases de teoría, relacionados con los cursores.

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

-- Ejemplo de cursor
CREATE OR REPLACE FUNCTION EjCursor() RETURNS varchar AS $$
```

```
DECLARE
   c_sect CURSOR FOR SELECT * FROM departamentos;
   r departamentos%rowtype;
   mensaje varchar;
BEGIN
   open c_sect;
   fetch c_sect into r;
   close c_sect;
   mensaje:= r.numerodpto || ' -- ' || r.nombredpto || ' -- ' || r.coste || ' -- ' || r.porcent;
   raise notice 'EjCursor: %',mensaje;
   return null;
END;
$$ LANGUAGE plpgsql;

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

-- Ejemplo de cursor loop
CREATE OR REPLACE FUNCTION ListarEmp() RETURNS varchar AS $$
DECLARE
       c_Emp CURSOR FOR SELECT * FROM Empleados;
       r Empleados%rowtype;
      mensaje varchar(300):='';
BEGIN
       raise notice 'ListarEmp';
       OPEN c_Emp;
       LOOP
              FETCH c_Emp INTO r;
           IF r IS NULL THEN EXIT;
           END IF;
             mensaje:= r.dni || ' - ' || r.nombre || ' - ' || r.numdep;
             raise notice '%',mensaje;
       END LOOP;
       CLOSE c_Emp;
       return null;
END;
$$ LANGUAGE plpgsql;

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

SELECT EjCursor();
SELECT ListarEmp();

DROP FUNCTION EjCursor();
DROP FUNCTION ListarEmp();
```

Salida de Script

```
NOTICE: EjCursor: 1 -- PERSONAL -- 0.00 -- 0.00
NOTICE: ListarEmp
NOTICE: 111111111 - Sánchez - 1
NOTICE: 222222222 - Martínez - 1
NOTICE: 333333333 - Álvarez - 1
NOTICE: 444444444 - González - 1
NOTICE: 555555555 - Martín - 2
NOTICE: 666666666 - Lagos - 2
NOTICE: 777777777 - Salazar - 2
NOTICE: 888888888 - López - 2
NOTICE: 123456789 - Pérez - 3
NOTICE: 666884444 - Ojeda - 3
NOTICE: 666999333 - Ruiz - 3
NOTICE: 999999999 - Simón - 3
NOTICE: 333445555 - Campos - 4
NOTICE: 222447777 - Torres - 4
NOTICE: 987654321 - Jiménez - 4
NOTICE: 000000000 - Sevilla – 4
```

Excepciones

El archivo **PlpgSQL_07.sql** contiene algunas versiones de los ejemplos expuestos en clases de teoría.

Postgresql tiene un tratamiento de excepciones bastante pobre, como se pone de manifiesto en los siguientes ejemplos:

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

-- Procedimiento para consultar el salario de un empleado
CREATE OR REPLACE FUNCTION FindEmp(pdni Empleados.dni%type ) RETURNS varchar AS $$
DECLARE
   v_sueldo Empleados.sueldo%Type;
   salida varchar(50);
   Cont numeric;

Begin
   Select sueldo Into v_sueldo From Empleados Where dni = pdni;
   salida:='FindEmp - sueldo: ' || v_sueldo;
   if (salida != NULL) then
        raise notice '%',salida;
    end if;
    return salida;
   EXCEPTION
      When No_Data_Found Then salida:='FindEmp - dni no existe.';
   raise notice '%',salida;
   return salida;
End;
$$ LANGUAGE plpgsql;

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

-- Permite generar una excepcion (lanzar una excepción)
-- Procedimiento para actualizar el salario de una tupla de Empleados
CREATE OR REPLACE FUNCTION UpdatesueldoEmp(pdni Empleados.dni%Type, p_sueldo
Empleados.sueldo%Type) RETURNS varchar AS $$
DECLARE
   Cont numeric;
   salida varchar(50);
Begin
   Select Count(*) Into Cont From Empleados Where dni = pdni;
   If (Cont=0) Then
      Raise No_Data_Found; -- lanza la excepcion
    End If;
   Update Empleados Set sueldo = p_sueldo Where dni = pdni;
   salida:='UpdatesueldoEmp - Proceso OK';
   raise notice '%',salida;
    return salida;
   EXCEPTION -- captura la excepción
      When No_Data_Found Then salida:='UpdatesueldoEmp - dni no existe.';
   raise notice '%',salida;
   return salida;
End;
$$ LANGUAGE plpgsql;

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

-- En PgPlSQL no es posible declarar una variable de tipo "exception"

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

-- Mismo procedimiento, pero con mensaje de error
CREATE OR REPLACE FUNCTION UpdatesueldoEmp3 (pdni Empleados.dni%Type, p_sueldo
Empleados.sueldo%Type)RETURNS varchar AS $$
DECLARE
   Cont numeric;
   salida varchar(150);
Begin
   Select Count(*) Into Cont From Empleados Where dni = pdni;
```

```
   If (Cont=0) Then
      salida:= 'UpdatesueldoEmp3 - No existe Empleados con dni '||pdni;
      RAISE EXCEPTION '%',salida;
    else
      Update Empleados Set sueldo = p_sueldo Where dni = pdni;
      salida:= 'UpdatesueldoEmp3 - Proceso OK';
      raise notice '%',salida;
   end if;
   -- ahora no se atiende la excepcion
    return salida;
End;
$$ LANGUAGE plpgsql;

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

select FindEmp('123456789'); -- Ok
select FindEmp('MAL---MAL'); -- Excepcion
select UpdatesueldoEmp('123456789',45000); -- Ok
select UpdatesueldoEmp('MAL---MAL',45000); -- Excepcion
select UpdatesueldoEmp3('123456789',50000); -- Ok
select UpdatesueldoEmp3('MAL---MAL',50000); -- Excepcion

-- Se vuelve a actualizar con sueldo original (36000)
select UpdatesueldoEmp('123456789',36000);

DROP FUNCTION FindEmp(varchar);
DROP FUNCTION UpdatesueldoEmp(varchar,numeric);
DROP FUNCTION UpdatesueldoEmp3(varchar,numeric);
```

Salida de Script

```
NOTICE: la referencia al tipo empleados.dni%TYPE convertida a character varying
NOTICE: la referencia al tipo empleados.dni%TYPE convertida a character varying
NOTICE: la referencia al tipo empleados.sueldo%TYPE convertida a numeric
NOTICE: la referencia al tipo empleados.dni%TYPE convertida a character varying
NOTICE: la referencia al tipo empleados.sueldo%TYPE convertida a numeric
NOTICE: UpdatesueldoEmp - Proceso OK
NOTICE: UpdatesueldoEmp - dni no existe.
NOTICE: UpdatesueldoEmp3 - Proceso OK

ERROR: UpdatesueldoEmp3 - No existe Empleados con dni MAL---MAL
CONTEXT: función PL/pgSQL updatesueldoemp3(character varying,numeric) en la línea 9
en RAISE

SQL state: P0001
```

4. Ejercicios de PL-pgSQL

Realizar los siguientes ejercicios:

A) Implementar las funciones necesarias 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 ...;
      vcoste ...;
      vporcent ...;
```

  Implementar una función de consulta que retorne en estas variables (como parámetros de salida) los valores del Departamentos cuyo `numerodpto` se pase como parámetro de entrada. Por ejemplo, retornar los datos del Servidor 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... seguir el siguiente guion:

```
/* ---------------------------------------------------------------------- */
-- Ejercicios... (A.1, A.2, A.3, A.4)
/* ---------------------------------------------------------------------- */
-- A.1)
Select InsertarDepartamentos (0,'CONTABILIDAD',0.0,0.0);
Select VerDepartamentos();

-- A.2)
Select ModificarDepartamentos (0,'FINANZAS',0.0,0.0);
Select VerDepartamentos();

-- A.3)
Select ConsultarDepartamentos (0,vnombre,vcoste, vporcent);

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

En este caso, la llamada a `ConsultarDepartamentos` desde otra función sería algo así como:

```
SELECT (ConsultarDepartamentos (3)).* INTO vnombre, vcoste, vporcent;

SELECT * FROM ConsultarDepartamentos (3) INTO vnombre, vcoste, vporcent;

salida:='Servidor : ' || vnombre || ' -- 3 -- ' || vcoste || ' -- ' || vporcent;
RAISE NOTICE '%',salida;
```

B) Realizar versiones de los procedimientos anteriores, de modo que se gestionen las excepciones de forma adecuada. Llamar a las 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:
  - 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:

  - 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**.
  - Particularmente, debe explicarse detalladamente todos los **aspectos novedosos** que incluyen el tema que se presenta.
  - Todas las capturas, salidas, código… que se incluyan en la documentación deben ser **perfectamente legibles**, presentándose con la máxima claridad y corrección.