

C.F.G.S.: DESARROLLO DE APLICACIONES WEB

Módulo: BASES DE DATOS

TEMA 08: PL/SQL Básico

U8. 01 Fundamentos del Lenguaje PL/SQL

Este lenguaje, basado en el lenguaje ADA, incorpora todas las características propias de los lenguajes de tercera generación: manejo de variables, estructura modular (procedimientos y funciones), estructuras de control (bifurcaciones, bucles y demás estructuras), control de excepciones. También incorpora un completo soporte para la Programación Orientada a Objetos (POO), por lo que puede ser considerado como un lenguaje procedimental y orientado a objetos.

Los programas creados con PL/SQL se pueden almacenar en la base de datos como cualquier otro objeto quedando así disponibles para su ejecución por los usuarios.

1. Tipos de Datos

Podremos trabajar con los principales tipos de datos de Oracle vistos en SQL, no existe una equivalencia exacta entre los tipos de PL/SQL y los mismos tipos de las columnas de Oracle, especialmente en lo relativo a las longitudes máximas que pueden almacenar.

CHAR	Cadena de caracteres de longitud fija. De 1 a 32767 caracteres. El dato se rellena con blancos a la derecha hasta alcanzar la longitud definida Si no damos longitud permite un único carácter
VARCHAR2	Cadena de caracteres de longitud variable De 1 a 32767 caracteres. Siempre hay que darle longitud. (Existe el tipo VARCHAR y funciona como VARCHAR2 pero Oracle recomienda el uso de este último por compatibilidad con futuras versiones)
NUMBER	Para datos numéricos, números enteros y reales. Admite hasta 38 dígitos Puede indicarse la precisión deseada NUMBER (p,s) “p” es el total de dígitos y “s” es el número de decimales.
DATE	Tipo de datos fecha Incluye día, mes, año, hora, minutos y segundos.
LONG	Cadena de caracteres de longitud variable. Longitud máxima 2Gb

INTEGER INT	Para números enteros
DECIMAL(p,s)	Como NUMBER
FLOAT REAL SMALLINT	Numéricos

Y otros propios de PL/SQL

BINARY_INTEGER	Numérico que se utiliza para índices, contadores Admite valores entre -21,47483647 y + 2747483647
BOOLEAN	Almacena valores TRUE, FALSE y NULL

Se pueden hacer conversiones entre datos utilizando las funciones correspondientes:

TO_CHAR, TO_NUMBER, TO_DATE

PL/SQL dispone además de tipos de datos más complejos como registros, tablas y arrays.

2. Identificadores

Para nombrar variables, constantes, funciones, procedimientos...

Pueden tener hasta 30 caracteres, empezando siempre por letra, que puede ir seguida por letras, números, \$, #, _

No diferencia entre minúsculas y mayúsculas.

3. Variables

Por cada variable se tiene que especificar el tipo y en algunos casos el tamaño.

No admite lista de variables separadas por comas y después el tipo como en otros lenguajes.

```
Importe NUMBER (6,2);
Nombre VARCHAR2(30);
Existe BOOLEAN;
```

A la vez que se declara se puede inicializar y determinar que no puede ser nula.

```
Contador NUMBER (3) := 0;
Suma NUMBER (3) NOT NULL := 0;
```

4. Uso de atributos %TYPE y %ROWTYPE

Permiten indicar el tipo de una variable para que sean del mismo tipo de otros objetos ya definidos.

- %TYPE el mismo tipo de otra variable ya definida, o que una columna de una tabla
- %ROWTYPE crea una variable de registro cuyos campos se corresponden con las columnas de una tabla

```
Importe NUMBER (6,2);
```

```
Total importe%TYPE;      -- declara total como del mismo tipo que importe
```

```
Vemple EMPLE%ROWTYPE ;
```

```
-- crea una variable que podrá contener una fila de la tabla EMPLE
```

```
-- para hacer referencia a cada uno de los campos variable.campo
```

```
Vemple.emp_no
```

5. Constantes

Se declaran como variables pero se le añade la palabra reservada CONSTANT y siempre hay que asignarles un valor en la declaración que después no se podrá cambiar.

```
PI CONSTANT REAL := 3.1416;
```

6. Operadores

- **Asignación:** :=
Por ejemplo importe:=40;
- **Lógicos:** AND, OR, NOT
- **Concatenación:** ||
- **Comparación:** =, !=, <>, <, >, <=, >=,
IS NULL, BETWEEN, LIKE, IN
- **Aritméticos:** +, -, *, /, **

Incluyendo que algunos se pueden utilizar con fechas:

f1 – f2 devuelve el número de días entre f1 y f2

f + n devuelve la fecha resultado de sumar n días a f

f - n devuelve la fecha resultado de restar n días a f



7. Estructuras de control IF

- **Alternativa simple:**

```
IF <condición> THEN
    Instrucciones;
    ...
END IF;
```

- **Alternativa doble:**

```
IF <condición> THEN
    Instrucciones;
    ...
ELSE
    Instrucciones;
    ...
END IF;
```

- **Alternativa múltiple:**

```
IF <condición1> THEN
    Instrucciones1;
    ...
ELSIF
    <condición2> THEN
    Instrucciones2;
    ...
ELSIF
    <condición3> THEN
    Instrucciones3;
    ...
    ...
    [ELSE
    Instrucciones;
    ...;]
END IF;
```

8. Bucles

- **LOOP**

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

También podemos salir del bucle con **EXIT WHEN <condición>**

- **WHILE <condición> LOOP**

```
Instrucciones;  
...  
END LOOP;
```

- **FOR <variablecontrol> IN <valorInicio> .. <valorFinal> LOOP**

```
Instrucciones;  
...  
END LOOP;
```

La <variablecontrol> es una variable que se declara de manera implícita como variable local al bucle de tipo BYNARY_INTEGER, por lo que nosotros ni podemos declararla ni utilizarla fuera del for. Se va incrementando de unidad en unidad en cada iteración del bucle. Se puede utilizar dentro del bucle pero no asignarle valor.

El incremento siempre es en una unidad pero puede ser negativo utilizando la opción **REVERSE**

```
FOR <variablecontrol> IN REVERSE <valorInicio> .. <valorFinal> LOOP  
    Instrucciones;  
    ...  
END LOOP;
```

9. BLOQUES PL/SQL

El bloque es la estructura básica característica de todos los programas PL/SQL.

Tiene tres zonas claramente definidas:

- Una zona de declaraciones donde se declaran objetos locales (variables, constantes, etc). Suele ir precedida por la cláusula **DECLARE** (o **IS/AS** en los procedimientos y funciones). Es opcional.
- Un conjunto de instrucciones precedido por la cláusula **BEGIN**.
- Una zona de tratamiento de excepciones precedido por la cláusula **EXCEPTION**. Es opcional.

Ejemplos:

Ejemplo 1

```

1 BEGIN
2   NULL;
3 END;
```

Esto es el bloque más pequeño que podemos tener. No tiene zona de declaración de variables ni de tratamiento de excepciones. La única instrucción que tiene es NULL, que no hace nada.

Ejemplo 2

```

1 BEGIN
2   DBMS_OUTPUT.PUT_LINE ('HOLA DAW');
3 END;
```

En este bloque tenemos una instrucción que sirve para mostrar mensajes.

Ejemplo 3

```

1 DECLARE
2   N1 NUMBER := 1;
3   TOTAL NUMBER := 0;
4 BEGIN
5   LOOP
6     TOTAL := TOTAL + N1;
7     IF N1 = 10 THEN
8       EXIT;
9     END IF;
10  END LOOP;
11  DBMS_OUTPUT.PUT_LINE ('LA SUMA DE LOS PRIMEROS 10 NÚMEROS NATURALES ES ' || TOTAL);
12 END;
```

En este bloque tenemos la declaración e inicialización de dos variables y el proceso para calcular la suma de los 10 primeros números naturales.

Ejemplo 4

```

1  DECLARE
2  N1 NUMBER := 1;
3  TOTAL NUMBER := 0;
4  BEGIN
5    LOOP
6      TOTAL := TOTAL + N1;
7      IF N1 = 10 THEN
8        EXIT;
9      END IF;
10   END LOOP;
11   DBMS_OUTPUT.PUT_LINE ('LA SUMA DE LOS PRIMEROS 10 NÚMEROS NATURALES ES '||TOTAL);
12   EXCEPTION
13     WHEN OTHERS THEN
14       DBMS_OUTPUT.PUT_LINE ('SE HA PRODUCIDO UN ERROR INESPERADO');
15   END;
```

Es el mismo ejemplo que el anterior pero añadiendo la zona de excepciones.

El formato genérico del bloque es:

```

[ DECLARE
  <declaraciones>
  BEGIN
  <órdenes>
  [ EXCEPTION
    <gestión de excepciones>
  END;
```

Vamos a utilizar la función **DBMS_OUTPUT.PUT_LINE ()** para mostrar mensajes por pantalla.

Para que estos sean visibles hay que poner el parámetro **SERVEROUTPUT** a **ON**

Al comenzar la sesión haremos:

SET SERVEROUTPUT ON

Podemos realizar bloques anidados

```

DECLARE
<declaraciones>
BEGIN
<órdenes>
  DECLARE
  <declaraciones>
  BEGIN
  <órdenes>
  EXCEPTION
  <gestión de excepciones>
```

```

END;

...
EXCEPTION
<gestión de excepciones>
END;

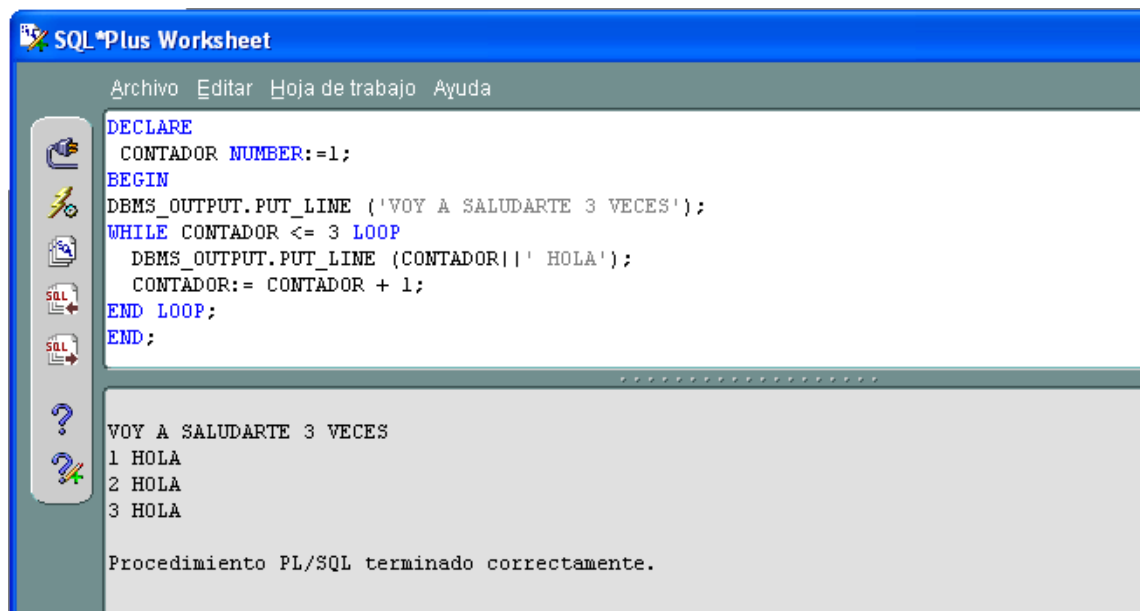
```

Podemos utilizar cualquiera de las funciones vistas en SQL (de cadenas de caracteres, fechas, numéricas...)

Podemos utilizar la instrucción **NULL** para indicar que no se realiza ninguna instrucción.

10. Bloques anónimos

Son bloques que se crean y ejecutan en el momento pero no quedan almacenados.



```

SQL*Plus Worksheet
Archivo  Editar  Hoja de trabajo  Ayuda

DECLARE
  CONTADOR NUMBER:=1;
BEGIN
  DBMS_OUTPUT.PUT_LINE ('VOY A SALUDARTE 3 VECES');
  WHILE CONTADOR <= 3 LOOP
    DBMS_OUTPUT.PUT_LINE (CONTADOR||' HOLA');
    CONTADOR:= CONTADOR + 1;
  END LOOP;
END;

VOY A SALUDARTE 3 VECES
1 HOLA
2 HOLA
3 HOLA

Procedimiento PL/SQL terminado correctamente.

```

11. Subprogramas: procedimientos y funciones

Los subprogramas son bloques PL/SQL que tienen un nombre, pueden recibir parámetros y, en el caso de las funciones, también devolver un valor.

Se guardan en la base de datos y podemos ejecutarlos invocándolos desde otros subprogramas o herramientas.

En todo subprograma podemos distinguir:

- **La cabecera o especificación del subprograma**
 - Contiene el nombre del subprograma
 - La definición de los parámetros con sus tipos (nunca los tamaños)
 - En el caso de las funciones el tipo del valor de retorno

- **El cuerpo del subprograma**
Es un bloque PL/SQL que incluye declaraciones (opcional), instrucciones y manejo de excepciones (opcional)

PROCEDIMIENTOS

Tienen la siguiente estructura general:

```

PROCEDURE <nombreprocedimiento> [( <lista de parámetros> )]
AS (o IS)
    <declaraciones>;
BEGIN
    <instrucciones>;
[EXCEPTION
    <excepciones>;]
END [<nombreprocedimiento>;]
    
```

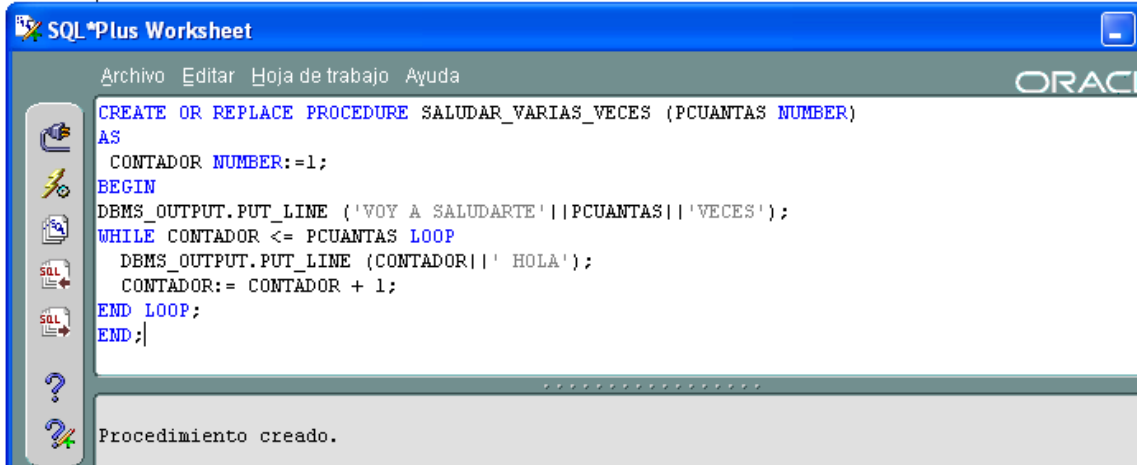
Para crear un procedimiento lo haremos utilizando la orden **CREATE**
CREATE [OR REPLACE] PROCEDURE <nombreprocedimiento>

```

CREATE OR REPLACE PROCEDURE SALUDAR_VARIAS_VECES (PCUANTAS NUMBER)
AS
    CONTADOR NUMBER:=1;
BEGIN
    DBMS_OUTPUT.PUT_LINE ('VOY A SALUDARTE VECES');
    WHILE CONTADOR <= PCUANTAS LOOP
        DBMS_OUTPUT.PUT_LINE (CONTADOR || ' HOLA');
        CONTADOR:= CONTADOR + 1;
    END LOOP;
END;
    
```

Para ejecutar un procedimiento hay que invocarlo desde cualquier herramienta Oracle, por ejemplo desde SQL

```
EXECUTE SALUDAR_VARIAS_VECES (5);
```

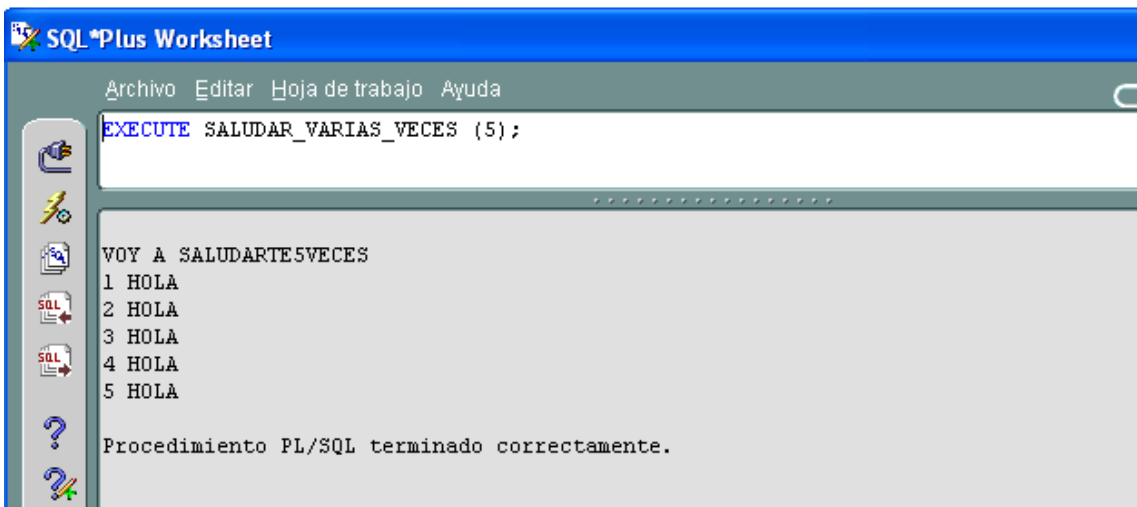



```

CREATE OR REPLACE PROCEDURE SALUDAR_VARIAS_VECES (PCUANTAS NUMBER)
AS
  CONTADOR NUMBER:=1;
BEGIN
  DBMS_OUTPUT.PUT_LINE ('VOY A SALUDARTE'||PCUANTAS||'VECES');
  WHILE CONTADOR <= PCUANTAS LOOP
    DBMS_OUTPUT.PUT_LINE (CONTADOR||' HOLA');
    CONTADOR:= CONTADOR + 1;
  END LOOP;
END;

```

Procedimiento creado.



```

EXECUTE SALUDAR_VARIAS_VECES (5);

```

VOY A SALUDARTE5VECES
 1 HOLA
 2 HOLA
 3 HOLA
 4 HOLA
 5 HOLA

Procedimiento PL/SQL terminado correctamente.

También se puede invocar a un procedimiento o función desde otro procedimiento o función.

Crearemos este procedimiento para que nos sea más cómodo mostrar datos por pantalla.

```

CREATE OR REPLACE PROCEDURE VER (A VARCHAR2)
AS
BEGIN
  DBMS_OUTPUT.PUT_LINE (A);
END;

```

Y ahora lo utilizamos en nuestro procedimiento anterior.

```

Archivo  Editar  Hoja de trabajo  Ayuda
CREATE OR REPLACE PROCEDURE SALUDAR_VARIAS_VECES (PCUANTAS NUMBER)
AS
  CONTADOR NUMBER:=1;
BEGIN
  VER ('VOY A SALUDARTE '||PCUANTAS||' VECES');
  WHILE CONTADOR <= PCUANTAS LOOP
    VER (CONTADOR||' HOLA');
    CONTADOR:= CONTADOR + 1;
  END LOOP;
END;
```

FUNCIONES

Tienen la siguiente estructura general:

```

FUNCTION <nombrefuncion> [( <lista de parámetros> )]
IS
RETURN <tipo del valor devuelto>
  <declaraciones>;
BEGIN
  <instrucciones>;
  RETURN <expresión>
[EXCEPTION
  <excepciones>;]
END [<nombreprocedimiento>;]
```

Para crear una función lo haremos utilizando la orden **CREATE**
CREATE [OR REPLACE] FUNCTION <nombrefuncion>

```

CREATE OR REPLACE FUNCTION HOY
RETURN VARCHAR2
IS
  DIA VARCHAR2(10);
BEGIN
  DIA:=TO_CHAR (SYSDATE,'DAY');
  RETURN DIA;
END;
```

Para ejecutar una función la llamaremos desde otro procedimiento o función.
 Podemos llamar al procedimiento que hemos creado, VER, para que nos muestre lo que devuelve la función.

EXECUTE VER (HOY);

Podemos borrar un procedimiento o una función con la orden **DROP**;

DROP PROCEDURE <nombreprocedure>;
 DROP FUNCTION <nombrefuncion>;

Para ver los subprogramas almacenados de un usuario:

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
FROM USER_OBJECTS
WHERE OBJECT_TYPE IN ('PROCEDURE','FUNCTION');
```

Para ver el código fuente:

```
SELECT LINE, SUBSTR(TEXT,1,60)
FROM USER_SOURCE
WHERE NAME = 'SALUDAR_VARIAS_VECES';
```

U8. 02 Acceso a BD en PL/SQL: SELECT, INSERT, UPDATE, DELETE

PL/SQL permite acceder a los datos almacenados en las tablas de la base de datos Oracle y manipularlos.

Podemos realizar **INSERT, UPDATE o DELETE** de una o varias filas de una tabla.

Ejemplos:

Procedimiento para dar de alta un nuevo departamento en la tabla DEPART. Recibe el número de departamento, el nombre y la localidad e inserta una nueva fila con estos datos.

```
CREATE OR REPLACE PROCEDURE
    ALTA_DEPART (PNUM NUMBER, PNOMBRE VARCHAR2, PLOC VARCHAR2)
AS
BEGIN
    INSERT INTO DEPART VALUES (PNUM,PNOMBRE,PLOC);
END;
```

Procedimiento para subir una cantidad al salario de un empleado. El procedimiento recibe la cantidad y el apellido del empleado.

```
CREATE OR REPLACE PROCEDURE
    SUBE_SALAR (PCANTIDAD NUMBER, PAPELL VARCHAR2)
AS
BEGIN
    UPDATE EMPL
    SET SALARIO = SALARIO + PCANTIDAD
    WHERE APELLIDO = PAPELL;
END;
```

Procedimiento para dar de baja un departamento. El procedimiento recibe el número de departamento.

```
CREATE OR REPLACE PROCEDURE BORRA_DEPART (PNUM NUMBER)
AS
BEGIN
    DELETE DEPART
    WHERE DEPT_NO = PNUM;
END;
```

Probar los procedimientos creados con los siguientes datos:

1. ALTA_DEPART con el departamento 50 de nombre PUBLICIDAD en MADRID
 2. SUBE_SALAR con 100 € al que se apellida SALA
 3. BORRA_DEPART con el departamento 50 que acabamos de dar de alta.
- ¿Qué sucederá si queremos dar de alta un departamento que ya exista?
 ¿Y si le intentamos subir el salario a un empleado que no existe?
 ¿Y si queremos borrar un departamento con datos?

Cuando desde PL/SQL intentamos dar de alta un registro con clave duplicada o cuando queremos borrar un registro con datos relacionados en otra tabla, se produce un error. Y el procedimiento termina su ejecución con un error.

Para que esto no suceda deberíamos comprobar primero mediante las select necesarias si esto va a suceder o no.

SELECT

PL/SQL permite ejecutar cualquier consulta admitida por la base de datos. Pero cuando se ejecuta la consulta, el resultado no se muestra automáticamente en el terminal del usuario, sino que queda en un área de memoria denominada cursor a la que accederemos utilizando variables.

La sintaxis es la siguiente:

```
SELECT <columna/s> INTO <variable/s>
FROM <tabla/s>
WHERE ...
```

Por ejemplo:

```
SELECT EMP_NO, SALAR INTO V_EMP_NO, V_SALAR
FROM EMPL
WHERE APELLIDO = 'SALA';
```

V_EMP_NO Y V_SALAR serán dos variables que se han tenido que declarar antes y que deben ser del mismo tipo que las columnas de la tabla.

```
DECLARE
V_EMP_NO NUMBER;
V_SALAR NUMBER;
BEGIN
SELECT EMP_NO, SALARIO INTO V_EMP_NO, V_SALAR
FROM EMPL
WHERE APELLIDO = 'SALA';
DBMS_OUTPUT.PUT_LINE ('SU NÚMERO ES EL '||V_EMP_NO||' Y GANA '||V_SALAR||'€');
END;
```

Una select de este tipo puede devolver 0 filas, 1 fila o varias filas. En el caso de que devuelva 0 filas o varias filas se provoca un error y el procedimiento interrumpe su ejecución.

Estos errores se pueden capturar en la zona de EXCEPTION de tratamiento de errores. En el siguiente capítulo veremos más sobre tratamiento de errores, pero por ahora vamos a capturar los siguientes:

```

DECLARE
V_EMP_NO NUMBER;
V_SALAR NUMBER;
BEGIN
SELECT EMP_NO, SALARIO INTO V_EMP_NO, V_SALAR
FROM EMPL
WHERE APELLIDO <> 'SALA';
DBMS_OUTPUT.PUT_LINE ('SU NÚMERO ES EL '||V_EMP_NO||' Y GANA '||V_SALAR||'€');
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE ('LA CONSULTA NO HA DEVUELTO NINGUNA FILA');
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE ('LA CONSULTA DEVUELVE VARIAS FILAS');
END;
```

Vamos ahora a completar los procedimientos anteriores.

En ALTA_DEPART primero comprobamos si ya existe el departamento que queremos dar de alta.

Podemos “contar” cuantos departamentos hay con ese número. Si hay 0 lo damos de alta y si hay más de 0 damos un mensaje de error.

Una SELECT COUNT(*) siempre devuelve un valor, si no hay nada devuelve 0

```

-- VERSIÓN 1
CREATE OR REPLACE PROCEDURE
ALTA_DEPART (PNUM NUMBER, PNOMBRE VARCHAR2, PLOC VARCHAR2)
AS
CUANTOS NUMBER;
BEGIN
SELECT COUNT(*) INTO CUANTOS
FROM DEPART
WHERE DEPT_NO = PNUM;
IF CUANTOS = 0 THEN
INSERT INTO DEPART VALUES (PNUM,PNOMBRE,PLOC);
ELSE
DBMS_OUTPUT.PUT_LINE ('DEPARTAMENTO YA EXISTE');
END IF;
END;
```

En esta otra versión hacemos la select que comprueba si existe el departamento. Si esta select no devuelve ninguna fila, se produce un error que es capturado en la zona EXCEPTION con WHEN NO_DATA_FOUND, y es ahí donde daremos de alta el nuevo departamento.

```
-- VERSIÓN 2
CREATE OR REPLACE PROCEDURE
    ALTA_DEPART (PNUM NUMBER, PNOMBRE VARCHAR2, PLOC VARCHAR2)
AS
EXISTE NUMBER;
BEGIN
    SELECT DEPT_NO INTO EXISTE
    FROM DEPART
    WHERE DEPT_NO = PNUM;
    DBMS_OUTPUT.PUT_LINE ('DEPARTAMENTO YA EXISTE');
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO DEPART VALUES (PNUM,PNOMBRE,PLOC);
END;
```

Vamos ahora a modificar el procedimiento que borra departamentos para comprobar primero que no tiene empleados-

```
CREATE OR REPLACE PROCEDURE BORRA_DEPART (PNUM NUMBER)
AS
CUANTOS NUMBER;
BEGIN
    SELECT COUNT(*) INTO CUANTOS
    FROM EMPL
    WHERE DEPT_NO = PNUM;
    IF CUANTOS = 0 THEN
        DELETE DEPART
        WHERE DEPT_NO = PNUM;
    ELSE
        DBMS_OUTPUT.PUT_LINE ('DEPARTAMENTO TIENE EMPLEADOS, NO SE PUEDE BORRAR');
    END IF;
END;
```