Índice

1. PROCEDIMIENTOS

Los bloques vistos hasta ahora eran *bloques anónimos*, estos han de compilarse cada vez que son ejecutados. Un bloque anónimo no puede ser almacenado en la base de datos y no pueden ser llamados directamente desde otros bloques PL/SQL. Ahora empezaremos a ver los denominados ***bloques nominados*** (con nombre), estos bloques no tienen estas restricciones, pueden ser ejecutados en la base de datos para ser ejecutados cuando sea necesario. Entre estos bloques se encuentran los procedimientos, funciones, paquetes y disparadores. A los dos primeros se les denomina, de forma colectiva, ***subprogramas***.

Empecemos el tema con un ejemplo, supongamos que tenemos la siguiente tabla:

EMPLOYEE

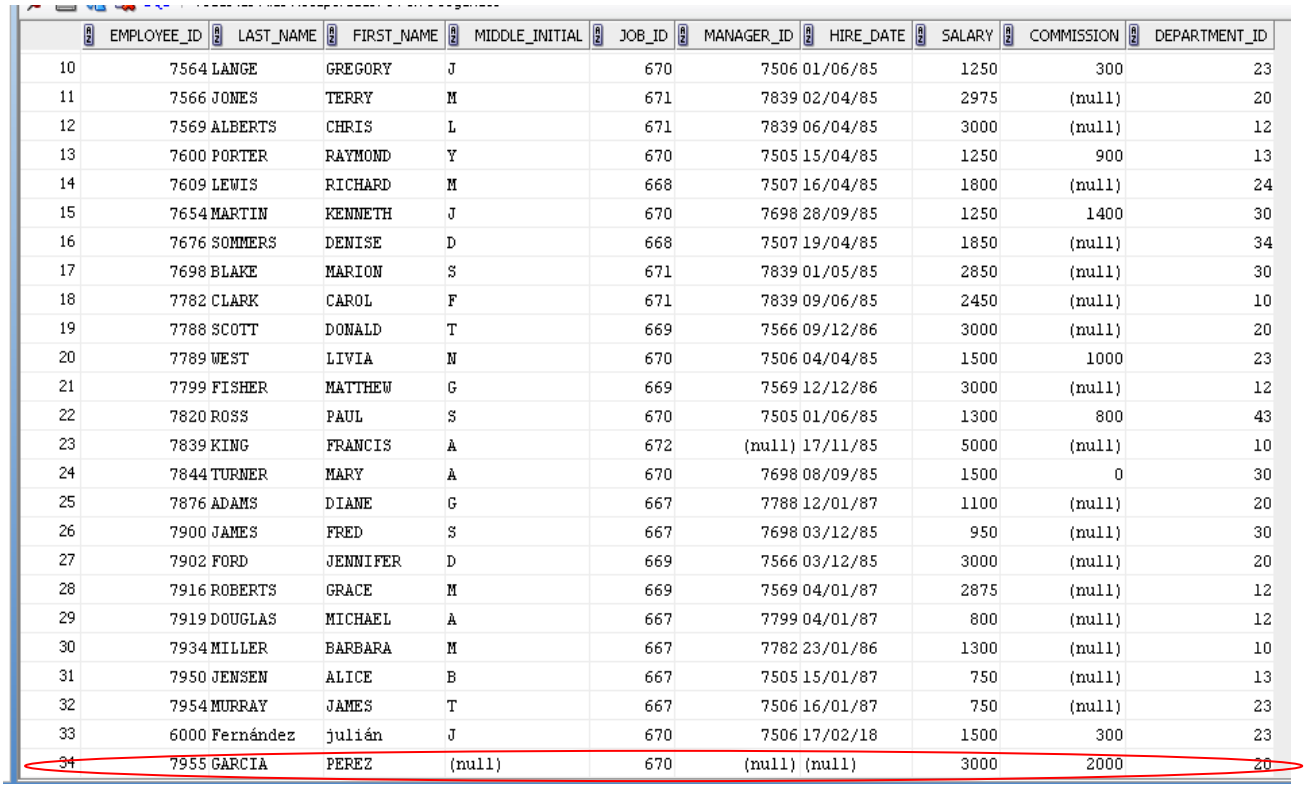
Ahora lo que queremos es crear un procedimiento para poder introducir nuevos empleados en esta tabla almacenada en la base de datos, tendremos que darle el nuevo employee_id nuevo y no repetido, apellido, nombre, tipo_trabajo, manager, salario, comision y departamento.

```
CREATE OR REPLACE PROCEDURE NUEVO_EMPLEADO (
        P_COD_EMPLEADO EMPLOYEE.EMPLOYEE_ID%TYPE,
        P_APELLIDO EMPLOYEE.LAST_NAME%TYPE,
        P_NOMBRE EMPLOYEE.FIRST_NAME%TYPE,
        P_FUNCION EMPLOYEE.JOB_ID%TYPE,
        P_SALARIO EMPLOYEE.SALARY%TYPE,
        P_COMISION EMPLOYEE.COMMISSION%TYPE,
        P_DEPARTAMENTO  EMPLOYEE.DEPARTMENT_ID%TYPE) IS
BEGIN
        INSERT INTO EMPLOYEE (EMPLOYEE_ID, LAST_NAME, FIRST_NAME, JOB_ID, SALARY, COMMISSION, DEPARTMENT_ID)
        VALUES (P_COD_EMPLEADO, P_APELLIDO, P_NOMBRE, P_FUNCION, P_SALARIO, P_COMISION, P_DEPARTAMENTO);
END NUEVO_EMPLEADO;
/
```

Si ahora llamamos al procedimiento:

```
DECLARE
        MAX_EMPLEADO VARCHAR2(4);
BEGIN
        SELECT MAX(EMPLOYEE_ID)+1 INTO MAX_EMPLEADO
        FROM EMPLOYEE;
        NUEVO_EMPLEADO(MAX_EMPLEADO,'GARCIA','PEREZ',670,3000,2000,20);
END;
/
```

La tabla quedaría:



	EMPLOYEE_ID	LAST_NAME	FIRST_NAME	MIDDLE_INITIAL	JOB_ID	MANAGER_ID	HIRE_DATE	SALARY	COMMISSION	DEPARTMENT_ID
10	7564	LANGE	GREGORY	J	670	7506	01/06/85	1250	300	23
11	7566	JONES	TERRY	M	671	7839	02/04/85	2975	(null)	20
12	7569	ALBERTS	CHRIS	L	671	7839	06/04/85	3000	(null)	12
13	7600	PORTER	RAYMOND	Y	670	7505	15/04/85	1250	900	13
14	7609	LEWIS	RICHARD	M	668	7507	16/04/85	1800	(null)	24
15	7654	MARTIN	KENNETH	J	670	7698	28/09/85	1250	1400	30
16	7676	SOMMERS	DENISE	D	668	7507	19/04/85	1850	(null)	34
17	7698	BLAKE	MARION	S	671	7839	01/05/85	2850	(null)	30
18	7782	CLARK	CAROL	F	671	7839	09/06/85	2450	(null)	10
19	7788	SCOTT	DONALD	T	669	7566	09/12/86	3000	(null)	20
20	7789	WEST	LIVIA	N	670	7506	04/04/85	1500	1000	23
21	7799	FISHER	MATTHEW	G	669	7569	12/12/86	3000	(null)	12
22	7820	ROSS	PAUL	S	670	7505	01/06/85	1300	800	43
23	7839	KING	FRANCIS	A	672	(null)	17/11/85	5000	(null)	10
24	7844	TURNER	MARY	A	670	7698	08/09/85	1500	0	30
25	7876	ADAMS	DIANE	G	667	7788	12/01/87	1100	(null)	20
26	7900	JAMES	FRED	S	667	7698	03/12/85	950	(null)	30
27	7902	FORD	JENNIFER	D	669	7566	03/12/85	3000	(null)	20
28	7916	ROBERTS	GRACE	M	669	7569	04/01/87	2875	(null)	12
29	7919	DOUGLAS	MICHAEL	A	667	7799	04/01/87	800	(null)	12
30	7934	MILLER	BARBARA	M	667	7782	23/01/86	1300	(null)	10
31	7950	JENSEN	ALICE	B	667	7505	15/01/87	750	(null)	13
32	7954	MURRAY	JAMES	T	667	7506	16/01/87	750	(null)	23
33	6000	Fernández	julián	J	670	7506	17/02/18	1500	300	23
34	7955	GARCIA	PEREZ	(null)	670	(null)	(null)	3000	2000	20

Teniendo en cuenta este ejemplo, podemos decir que:

1. Creamos el procedimiento con la orden CREATE OR REPLACE PROCEDURE. Cuando este se crea se compila en primer lugar y luego se almacena en la base de datos en forma compilada. Este código compilado puede ser posteriormente ejecutado desde un bloque PL/SQL

2. Cuando se llama al procedimiento le podemos pasar valores, en el ejemplo anterior se los pasamos en tiempo de ejecución, con lo cuál, los parámetros correspondientes tendrán los literales que hemos querido pasarle al momento de invocar el procedimiento.

3. Una llamada a un procedimiento es una orden PL/SQL por sí misma. La llamada no se produce como parte de una expresión. Cuando se llama a un procedimiento, el control pasa a la primera orden ejecutable dentro de él. Cuando el procedimiento termina, se devuelve el control a la orden que sigue a la llamada al procedimiento. A este respecto los procedimientos de PL/SQL se comportan de la misma forma que los de otros lenguajes de tercera generación (3GL).

4. Un procedimiento es un bloque PL/SQL, con una sección declarativa, una sección ejecutable y una sección de manejo de excepciones. Al igual que con los bloques anónimos, la única sección obligatoria es la sección ejecutable.

La sintaxis COMPLETA SERIA:

```
CREATE [ OR REPLACE ] PROCEDURE nombre_procedimiento
    [ (argumento [ { IN | OUT | IN OUT } ] tipo,
    . . .
    argumento [ { IN | OUT | IN OUT } ] tipo) ] { IS | AS }
    cuerpo_procedimiento
```

nombre_procedimiento .- Nombre del procedimiento que se quiere crear

argumento .- Nombre de un parámetro del procedimiento

tipo .- Tipo del parámetro asociado

cuerpo_procedimiento .- bloque PL/SQL que contiene el código del procedimiento

OR REPLACE.- Es para poder cambiar el código de un procedimiento que ya existía con anterioridad. Ya que esto es bastante común mientras se está desarrollando un procedimiento, para evitar eliminarlo y crearlo de nuevo, se utilizan estas palabras. Si esta creado se elimina sin generar ningún mensaje de aviso, si no está creado se crea. En el caso de que exista, pero no se han incluido las palabras clave OR REPLACE, la orden CREATE devolverá el error ORACLE:

ORA – 00955 : name is already used by an existing object

Los parámetros formales pueden tener tres modos: IN, OUT o IN OUT.

Si no se especifica el modo de un parámetro formal, se adopta, por defecto, el modo IN

Las diferencias entre los tres modos son:

IN .- a.- Permite pasar valores a un subprograma
b.- Dentro del subprograma, el parámetro actúa como una constante, es decir, no se le puede asignar ningún valor.
c.- El parámetro actual puede ser una variable, constante, literal o expresión.

OUT .- a.- Permite devolver valores al bloque que llamó al subprograma.
b.- Dentro del subprograma, el parámetro actúa como una variable no inicializada.
c.- No puede intervenir en ninguna expresión, salvo para tomar un valor.
d.- El parámetro actual debe ser una variable.

IN OUT .- a.- Permite pasar un valor inicial a un subprograma y devolver un valor actualizado.
b.- Dentro del subprograma actúa como una variable inicializada.
c.- Puede intervenir en otras expresiones y tomar nuevos valores.
d.- El parámetro actual debe ser una variable.

En el caso de que se declaren el modo de los argumentos, se les supone el modo INOUT por defecto. En el caso de que no haya parámetros en el procedimiento no se pondrán los paréntesis.

Ejemplo. Crear un procedimiento denominado mayor_de_dos que recibe dos números reales y visualiza cual es el mayor de los dos. En el caso de que los números sean iguales visualiza son iguales.

```
CREATE OR REPLACE PROCEDURE MAYOR_DE_DOS
    (N1 IN REAL,
    N2 IN REAL) IS
  BEGIN
     IF N1>N2
       THEN
         DBMS_OUTPUT.PUT_LINE(TO_CHAR(N1)||' ES MAYOR');
       ELSIF N2>N1
         THEN
           DBMS_OUTPUT.PUT_LINE(TO_CHAR(N2)||' ES MAYOR');
         ELSE
           DBMS_OUTPUT.PUT_LINE('SON IGUALES');
     END IF;
  END;
/

execute MAYOR_DE_DOS (3,5);
```

Ejemplo 2. Crear un procedimiento en el que introducimos el número de un empleado y visualizamos el número de clientes que tiene.

```
CREATE OR REPLACE PROCEDURE VER_CLIENTES_EMPLEADO (CODIGO IN NUMBER)
 IS
    CURSOR C IS
    SELECT CUSTOMER_ID, NAME
    FROM EMPLOYEE, CUSTOMER
    WHERE EMPLOYEE.EMPLOYEE_ID=CUSTOMER.SALESPERSON_ID AND EMPLOYEE_ID=CODIGO;
    COD_CLIENTE CUSTOMER.CUSTOMER_ID%TYPE;
    NOM_CLIENTE CUSTOMER.NAME%TYPE;
BEGIN
   OPEN C;
   DBMS_OUTPUT.PUT_LINE('EL EMPLEADO  '||CODIGO||'  TIENE COMO CLIENTES');
   FETCH C INTO COD_CLIENTE, NOM_CLIENTE;
   WHILE C%FOUND LOOP
      DBMS_OUTPUT.PUT_LINE(COD_CLIENTE||'  '||NOM_CLIENTE);
      FETCH C INTO COD_CLIENTE, NOM_CLIENTE;
   END LOOP;
   CLOSE C;
END VER_CLIENTES_EMPLEADO;
/

execute ver_clientes_empleado(7521);
```

ejemplo con parámetro de salida out

```
CREATE OR REPLACE PROCEDURE ASIGNA (K OUT NUMBER)
AS
BEGIN
        K:=4;
END ASIGNA;
/
```

¿Qué pasa cuando se invoca?

```
DECLARE
        N NUMBER;
BEGIN
        ASIGNA(N);
        DBMS_OUTPUT.PUT_LINE('DESPUES:   '||N);
END;
/
```

Ejemplo con parámetro de entrada/salida in out

```
CREATE OR REPLACE PROCEDURE INCREMENTA (K  IN OUT NUMBER)
AS
BEGIN
        K:=K+1;
END INCREMENTA;
/
```

¿Qué pasa cuando se invoca?

```
DECLARE
        N NUMBER;
BEGIN
        N:=&N;
        DBMS_OUTPUT.PUT_LINE('ANTES:     '||N);
        INCREMENTA(N);
        DBMS_OUTPUT.PUT_LINE('DESPUES:   '||N);
END;
/
```

Ejemplo con los parámetros IN, OUT e IN/OUT

```
CREATE OR REPLACE PROCEDURE EMPL_SAL (
        COD_EMPLEADO IN EMPLOYEE.EMPLOYEE_ID%TYPE,
        NOMBRE OUT VARCHAR2,
        P_X IN OUT VARCHAR2)
AS
        SALARIO EMPLOYEE.SALARY%TYPE;
BEGIN
        SELECT FIRST_NAME, SALARY INTO NOMBRE, SALARIO
        FROM EMPLOYEE
        WHERE EMPLOYEE_ID=COD_EMPLEADO;
        P_X:='NOMBRE';
        DBMS_OUTPUT.PUT_LINE('SALARIO ACTUAL: '||SALARIO);
END EMPL_SAL;
/

DECLARE
V_NOMBRE EMPLOYEE.FIRST_NAME%TYPE;
V_X VARCHAR2(40);
BEGIN
V_X:='NAME';
EMPL_SAL(7555,V_NOMBRE,V_X);
DBMS_OUTPUT.PUT_LINE(V_X||'  '||V_NOMBRE);
END;
/
```

RESTRICCIONES SOBRE LOS PARÁMETROS FORMALES

Cuando se llama a un procedimiento, se pasan los valores de los parámetros reales. Dentro del cuerpo del procedimiento, la manera de hacer referencia a dichos valores es a través de los parámetros formales. El mecanismo de paso de parámetros no solo pasa los valores, sino también las restricciones que afectan a las variables. En una declaración de procedimiento, es ilegal restringir un parámetro CHAR o VARCHAR2 con una determinada longitud, o un parámetro NUMBER con un valor de precisión y/o escala. Como conclusión sacamos que:

A la hora de definir los parámetros no es necesario indicar los tamaños, ya que heredan las restricciones de las variables que se les asocian, por lo que si una variable es de tipo VARCHAR2(3) y al parámetro se le asigna el siguiente valor "abcdefg" lo que se obtiene es un error (ORA – 6502). El único modo de que los parámetros adquieran restricciones completas es por medio de %TYPE.

VALORES POR DEFECTO (PARÁMETROS)

De igual manera que con las variables, los parámetros de un procedimiento o función, pueden asignárseles valores por defecto. Su sintaxis es:

Nombre_del_parámetro [modo] tipo_de_parámetro { := | DEFAULT} valor_defecto.

Cuando se llame al procedimiento en el caso de que no se especifique el valor para el parámetro, éste cogerá el valor por defecto. Por comodidad es recomendable colocar los parámetros con valores por defecto al final de los argumentos para que a la hora de declarar la llamada del procedimiento y no se pasen valores no existan errores o confusiones. Ejemplo:

```
CREATE OR REPLACE PROCEDURE   DefaultTest (
        Parametro1   NUMBER DEFAULT 10,
        Parametro2   VARCHAR2 DEFAULT 'ABCDEFG',
        Parametro3   DATE DEFAULT SYSDATE)  AS

BEGIN
…
END Ejemplo_N;
```

Llamadas:

```
        DefaultTest (Parametro1 => 7, Parametro3 => '30 – DEC – 95');
```

Si llamásemos así al procedimiento, el Parametro2 toma el valor por defecto.

Si hubiésemos llamado así:

```
        DefaultTest (7, '30 – DEC – 95');
```

Al ejecutarse mostraría un error, ya que colocaría el segundo valor en el segundo parámetro, cuando este es de tipo VARCHAR2 y no DATE

2. FUNCIONES

Una función es bastante similar a un procedimiento. Ambos aceptan argumentos, y estos pueden ser de cualquiera de los modos reseñados. Ambos son formas diferentes de bloque PL/SQL, con sus secciones declarativa, ejecutable y de excepciones. Ambos pueden ser almacenados en la base de datos o ser declarados dentro de un bloque. Sin embargo, **una llamada a un procedimiento es una orden PL/SQL en sí misma, mientras que una llamada a función se realiza como parte de su expresión.**

La sintaxis es:

CREATE [OR REPLACE] FUNCTION *nombre_función*

[(*argumento* **[{IN | OUT | INOUT}]** *tipo*,

…

argumento **[{IN | OUT | INOUT }] tipo }]**

RETURN *return_tipo* **{ IS | AS }**

Cuerpo de la función

Dentro del cuerpo de la función existirá la sentencia RETURN que devolverá el valor de la variable deseada, esta será del mismo tipo que el asignado en la cláusula: RETURN tipo IS… . Su sintaxis es:

RETURN *variable*

Nota.- Podremos eliminar un procedimiento o una función mediante:

DROP PROCEDURE *procedimiento;*

DROP FUNCTION *función;*

EJECUCIÓN DE FUNCIONES

A.- MEDIANTE UNA VARIABLE HOST:

1. Definir una variable host: VARIABLE nb_variable tipo_dato;
2. Ejecutar de la siguiente forma: EXECUTE nb_variable:= nb_función (valor_de_entrada);
3. Imprimir la variable host: PRINT nb_variable.

B.- INVOCARLA DESDE UN BLOQUE PL CARGÁNDOLA EN UNA VARIABLE.

<VARIABLE> := nb_ funcion (parámetro)

Ejemplo - Parte 1: Diseñar una función que devuelva el número de años comprendidos entre dos fechas, las cuales son introducidas como parámetros.

```
CREATE OR REPLACE FUNCTION FUNC_FECHA
        (FECHA1 DATE, FECHA2 DATE)
        RETURN NUMBER
        AS
        V_ANNOS NUMBER(6);
BEGIN
        V_ANNOS := ABS(TRUNC(MONTHS_BETWEEN(FECHA2,FECHA1)/12));
        RETURN (V_ANNOS);
END FUNC_FECHA;
```

Ejemplo - Parte 2: Crear un bloque anónimo que llama a la función y devuelva su valor.

```
DECLARE
        V_RESP NUMBER;
BEGIN
        V_RESP:= FUNC_FECHA ('03/12/1995', '08/08/2020');
        DBMS_OUTPUT.PUT_LINE(V_RESP);
END;
```

Una función de usuario puede ser invocada desde una orden SQL:

- Como columna de una SELECT
- En las condiciones WHERE y HAVING
- En las cláusulas ORDER BY y GROUP BY
- Cláusulas VALUES en el comando INSERT
- En la cláusula SET del comando UPDATE.

MAS SOBRE FUNCIONES

Para borrar funciones se utiliza la siguiente orden:

```
DROP FUNCTION nb_funcion;
```

3. EJERCICIOS RESUELTOS

Ejercicio número: 1

Función que pasándole un departamento nos devuelva el sueldo total del mismo

```
CREATE OR REPLACE FUNCTION     SALARIO_DEPT
        (CODIGO      DEPARTMENT.DEPARTMENT_ID % TYPE)
         RETURN     NUMBER
         IS
         SALARIO     NUMBER (8);
BEGIN

        SELECT SUM (salary) INTO   SALARIO
        FROM EMPLOYEE
        WHERE DEPARTMENT_ID  =  CODIGO;
        RETURN ( SALARIO);

END SALARIO_DEPT;


DECLARE
        SALARIO NUMBER;
        BEGIN
        SALARIO:= SALARIO_DEPT (13);
        DBMS_OUTPUT.PUT_LINE(SALARIO);
END;
```

Comprobamos que la salida es la misma que si hacemos la select:

```
SELECT SUM(SALARY)
FROM EMPLOYEE
WHERE DEPARTMENT_ID=13;
```

Vemos ahora que hace esta select:

```
SELECT
   department_id     cod_dep,
   department.name   nombre,
   location_id       cod_loc,
   salario_dept(department_id) sal_dep
FROM
   department
ORDER BY
   department_id;
```

✎ **Ejercicio número: 2** Crear una función que dada una "función de trabajo" (ó JOB.FUNCTION) muestre el "nombre del departamento" (ó DEPARTMENT.NAME) que contienen el mayor número de empleados que realice esa "función de trabajo".

```
CREATE OR REPLACE FUNCTION GET_DEP_JOB
        (NAME_JOB VARCHAR2)
        RETURN VARCHAR2         IS
        DEP VARCHAR2(14);
        CANTIDAD NUMBER(6);
        CURSOR C_DEP IS
         SELECT DEPARTMENT.NAME DEP,  COUNT(*) TOTAL
        FROM JOB , EMPLOYEE , DEPARTMENT
        WHERE     DEPARTMENT.DEPARTMENT_ID     =     EMPLOYEE.DEPARTMENT_ID     AND
        JOB.JOB_ID = EMPLOYEE.JOB_ID AND JOB.FUNCTION = NAME_JOB
        GROUP BY DEPARTMENT.NAME
        ORDER BY TOTAL DESC;
BEGIN
        OPEN C_DEP;
        FETCH C_DEP INTO DEP, CANTIDAD;
        IF C_DEP%FOUND
                THEN
                        RETURN DEP;
                ELSE
                        RETURN 'NO HAY DEPARTAMENTO';
        END IF;
        CLOSE C_DEP;
END GET_DEP_JOB ;
/


DECLARE
  DEP VARCHAR2(15);
BEGIN
  DEP := GET_DEP_JOB('ANALYST');
  DBMS_OUTPUT.PUT_LINE(DEP);
END;
/
```

4.- EJERCICIOS PROPUESTOS

1º.- Crear un procedimiento que se denomine ***ordenación_tres*** al que se le pasen tres números como parámetros y nos salgan ordenados de mayor a menor

2º.- Crear un procedimiento **Pcalcularsueldosdep** al que se le pasa un número de departamento y muestra cuantos empleados hay en el departamento, el total de sueldos y el total de comisiones del departamento

3º.- Desarrolla un procedimiento denominado **pverApellidos** que visualice el apellido y la fecha de alta de todos los empleados ordenados por apellido

4º.- Crear un procedimiento **pverempleoficio** al que le paso un oficio y muestra el nombre, apellidos, oficio, código de departamento y nombre del departamento de los empleados que tienen ese oficio

5º.- Crear un procedimiento denominado ***ver_productos*** al que se le pase como parámetro el nombre de un cliente y muestre todos los productos de dicho cliente.

6º.- Crear un procedimiento denominado ***clientes_productos*** que muestre todos los productos de todos los clientes. Para ello utiliza el procedimiento ***ver_productos*** del ejercicio anterior.

7º.- Crear una función denominada ***cuentablancos*** a la que se le pasa una cadena de caracteres y nos devuelve el número de espacios que tiene dicha cadena de caracteres

8º.- Crear una función denominada ***suma_salarial*** en la que se le pasa el nombre de un departamento y nos devuelve la suma de salarios de los empleados de dicho departamento.

9º.- Escribir una función denominada ***trienios*** que haciendo uso de la función ***func_fecha*** del ejemplo resuelto en el tema devuelva los trienios que hay entre dos fechas. (Un trienio son tres años completos).

10º.- Crear una función que permita visualizar la ciudad que más órdenes de venta ha realizado.

11º.- Crear una función en la que dado un departamento devuelva el empleado (apellido) de dicho departamento que cuenta con mayor salario