TEMA7. PROCEDIMIENTO Y FUNCIONES EN PL/SQL.

1. PROCEDIMIENTOS.

Tienen la siguiente estructura:

```
PROCEDURE <nombreprocedimiento> [( <lista de parámetros> )]
IS
        <declaraciones>;    -- equivalente a la sección declare
BEGIN
        <instrucciones>;
EXCEPTION
        <excepciones>;
END [<nombreprocedimiento>];
```

Donde los parámetros tienen la siguiente sintaxis:

```
<nombrevariable> [ IN | OUT | IN OUT ] < tipodedato> [ { := | DEFAULT } <valor>]
```

Al indicar los parámetros debemos especificar el tipo, pero no el tamaño.

En el caso de que no tenga parámetros no se pondrán los paréntesis.

Las declaraciones se hacen después del **IS que equivale al DECLARE**, en este caso si se deberá indicar la longitud de las variables locales.

En la estructura general de un procedimiento se pueden apreciar dos partes:

La especificación (cabecera) del procedimiento: Comienza con la palabra PROCEDURE y termina después del último parámetro.

El cuerpo del procedimiento: Comienza con la palabra IS y termina con la palabra END opcionalmente seguida del nombre del procedimiento.

Ejemplo que utiliza un procedimiento para visualizar el dept con más empleados.

```
DECLARE
 REGISTRO DEPART%ROWTYPE;
 CON NUMBER(4);
     PROCEDURE VER(REG DEPART%ROWTYPE) IS
     BEGIN
       DBMS_OUTPUT.PUT_LINE('--LOS DATOS PEDIDOS --');
       DBMS_OUTPUT.PUT_LINE('NUM:'||TO_CHAR(REG.DEPT_NO));
       DBMS_OUTPUT.PUT_LINE('NOMBRE:' || REG.DNOMBRE);
       DBMS_OUTPUT.PUT_LINE('LOCALIDAD:' || REG.LOC);
      END VER;
BEGIN
  SELECT * INTO REGISTRO FROM DEPART  WHERE DEPT_NO =
     (SELECT DEPT_NO  FROM EMPLE
             GROUP BY DEPT_NO
             HAVING   COUNT(*)=(SELECT MAX(COUNT(*)) FROM EMPLE GROUP BY DEPT_NO) );
  VER(REGISTRO);
END;
```

Convertir a procedure local el proceso de visualizar los detalles de una compra, del segundo ejercicio de Ejercicio 2 de Cursores.

```
DECLARE
    CURSOR C2 IS SELECT NUM_COMPRA, NOMBRE, FECHA FROM COMPRAS JOIN ALMACENES USING (CODALMA);
    REGCOMPRAS C2%ROWTYPE;

        PROCEDURE VERDETALLECOMPRAS (NUMCOMPRA NUMBER) IS
          CURSOR C1 (COMP NUMBER) IS
                SELECT NUM_COMPRA, CODARTI, DENOMINACION, CATEGORIA, UNIDADES,
                PVP, UNIDADES*PVP IMPORTE, STOCK-UNIDADES STOCKACTUAL
                 FROM ARTICULOS JOIN DET_COMPRAS USING (CODARTI)
```

```
                    JOIN COMPRAS USING (NUM_COMPRA)
                    WHERE NUM_COMPRA = COMP;

  TOTALUNI NUMBER(6):=0;
  TOTALIMP NUMBER(8,2):=0;
  REG C1%ROWTYPE;

BEGIN
     OPEN C1(NUMCOMPRA);
     FETCH C1 INTO REG;
     -- COMPROBAMOS QUE LA COMPRA TIENE DETALLECOMPRAS
     IF C1%NOTFOUND THEN
       DBMS_OUTPUT.PUT_LINE('LA COMPRA '|| NUMCOMPRA ||' NO TIENE ARTICULOS');
     ELSE
      -- TRATAMIENTO DEL DETALLE DE ESA COMPRA
      -- VISUALIZAMOS LA CABECERA
          DBMS_OUTPUT.PUT_LINE('NUM  COMPRA  CODARTI  DENOMINACION  '||  '  CATEGORIA
    UNIDADES  PVP  IMPORTE STOCK-ACTUAL');
      DBMS_OUTPUT.PUT_LINE('---------- ------- --------------  ---------- ---------- ------ -------- ------------');
      TOTALUNI:=0;
      TOTALIMP:=0;
      WHILE C1%FOUND LOOP
        DBMS_OUTPUT.PUT_LINE(RPAD(TO_CHAR(REG.NUM_COMPRA),12)||
            RPAD(TO_CHAR(REG.CODARTI),8)||
            RPAD(REG.DENOMINACION,25)||
            RPAD(REG.CATEGORIA,6)||
            RPAD(TO_CHAR(REG.UNIDADES),9)||
            RPAD(TO_CHAR(REG.PVP),9)||
            RPAD(TO_CHAR(REG.IMPORTE),10)||
            RPAD(TO_CHAR(REG.STOCKACTUAL),10));
        TOTALIMP := TOTALIMP + REG.IMPORTE;
        TOTALUNI := TOTALUNI + REG.UNIDADES;
        FETCH C1 INTO REG;
      END LOOP;
      DBMS_OUTPUT.PUT_LINE('----------  ---------  -------------- '||
           '---------- ---------- ------ -------- ------------');
      DBMS_OUTPUT.PUT_LINE('TOTALES DE UNIDADES: '|| TOTALUNI);
      DBMS_OUTPUT.PUT_LINE('TOTAL IMPORTE: '||TOTALIMP);
     END IF;
     CLOSE C1;
END VERDETALLECOMPRAS;

BEGIN
OPEN C2;
FETCH C2 INTO REGCOMPRAS;
WHILE C2%FOUND LOOP
 --VISUALIZAR LA COMPRA
   DBMS_OUTPUT.PUT_LINE('----------  ---------  -------------- '||  '---------- ---------- ------ -------- ------------');
   DBMS_OUTPUT.PUT_LINE(' NUM COMPRA'|| REGCOMPRAS.NUM_COMPRA ||
        ' NOMBRE ALMACÉN: '  || REGCOMPRAS.NOMBRE ||
        ' FECHA DE COMPRA: ' || REGCOMPRAS.FECHA);
   VERDETALLECOMPRAS(REGCOMPRAS.NUM_COMPRA);
   FETCH C2 INTO REGCOMPRAS;
END LOOP;
END;
```

1.1. PROCEDIMIENTOS ALMACENADOS.

Podemos crear procedimientos almacenados, como un objeto más de la BD, de tal forma que se almacenará en nuestra BD, igual que una tabla, una vista o cualquier otro objeto, para ello pondremos **Create or replace:**

```
CREATE OR REPLACE
```

```
 PROCEDURE NOMBRE_PROCEDURE(PARÁMETROS) IS
 --Declaración de variables
BEGIN
  --Cuerpo de la procedure
EXCEPTION
  --Bloque de excepciones
END;
```

Ejemplo: procedimiento que visualiza por consola los datos de un departamento que recibe como parámetro:

```
CREATE OR REPLACE
   PROCEDURE VERDEPARTAMENTO(NUM NUMBER) IS
  REG DEPARTAMENTOS%ROWTYPE;
BEGIN
   SELECT * INTO REG FROM DEPARTAMENTOS
     WHERE DEPT_NO=NUM;
    DBMS_OUTPUT.PUT_LINE('--LOS DATOS PEDIDOS --');
    DBMS_OUTPUT.PUT_LINE('NUM:'||TO_CHAR(REG.DEPT_NO));
    DBMS_OUTPUT.PUT_LINE('NOMBRE:' || REG.DNOMBRE);
    DBMS_OUTPUT.PUT_LINE('LOCALIDAD:' || REG.LOC);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('NO EXISTE EL DEPARTAMENTO.');
  WHEN TOO_MANY_ROWS THEN
   DBMS_OUTPUT.PUT_LINE('DEVUELVE VARIAS FILAS.');
END VERDEPARTAMENTO;

EXEC VERDEPARTAMENTO(10);
EXEC VERDEPARTAMENTO(50);
```

Convertir en procedure almacenada la procedure local del ejercicio anterior

```
CREATE OR REPLACE PROCEDURE VERDETALLECOMPRAS (NUMCOMPRA NUMBER) IS
  CURSOR C1 (COMP NUMBER) IS
          SELECT NUM_COMPRA, CODARTI, DENOMINACION, CATEGORIA, UNIDADES,
          PVP, UNIDADES*PVP IMPORTE, STOCK-UNIDADES STOCKACTUAL
           FROM ARTICULOS JOIN DET_COMPRAS USING (CODARTI)
          JOIN COMPRAS USING (NUM_COMPRA)
          WHERE NUM_COMPRA = COMP;

         TOTALUNI NUMBER(6):=0;
         TOTALIMP NUMBER(8,2):=0;
         REG C1%ROWTYPE;

BEGIN
OPEN C1(NUMCOMPRA);
 FETCH C1 INTO REG;
 -- COMPROBAMOS QUE LA COMPRA TIENE DETALLECOMPRAS
 IF C1%NOTFOUND THEN
   DBMS_OUTPUT.PUT_LINE('LA COMPRA '|| NUMCOMPRA ||' NO TIENE ARTICULOS');
 ELSE
  -- TRATAMIENTO DEL DETALLE DE ESA COMPRA
```

```
  -- VISUALIZAMOS LA CABECERA
  DBMS_OUTPUT.PUT_LINE('NUM COMPRA CODARTI DENOMINACION '||
        ' CATEGORIA  UNIDADES   PVP   IMPORTE STOCK-ACTUAL');
  DBMS_OUTPUT.PUT_LINE('---------- ------- ---------- '||
        '---------- ---------- ------ -------- ----------');
                    TOTALUNI:=0;
                    TOTALIMP:=0;
  WHILE C1%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE(RPAD(TO_CHAR(REG.NUM_COMPRA),12)||
       RPAD(TO_CHAR(REG.CODARTI),8)||
       RPAD(REG.DENOMINACION,25)||
       RPAD(REG.CATEGORIA,6)||
       RPAD(TO_CHAR(REG.UNIDADES),9)||
       RPAD(TO_CHAR(REG.PVP),9)||
       RPAD(TO_CHAR(REG.IMPORTE),10)||
       RPAD(TO_CHAR(REG.STOCKACTUAL),10));
    TOTALIMP := TOTALIMP + REG.IMPORTE;
    TOTALUNI := TOTALUNI + REG.UNIDADES;
    FETCH C1 INTO REG;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('----------  ---------  ---------- '||
        '---------- ---------- ------ -------- ----------');
  DBMS_OUTPUT.PUT_LINE('TOTALES DE UNIDADES: '|| TOTALUNI);
  DBMS_OUTPUT.PUT_LINE('TOTAL IMPORTE: '||TOTALIMP);
 END IF;
 CLOSE C1;
END VERDETALLECOMPRAS;
```

Una vez que hemos realizado el procedimiento almacenado podemos ejecutarlo de dos formas:

```
EXEC VERDETALLECOMPRAS(cod_compra);
```

En un bloque PL/SQL

```
DECLARE
CURSOR C2 IS SELECT NUM_COMPRA, NOMBRE, FECHA
          FROM COMPRAS JOIN ALMACENES USING (CODALMA);
REGCOMPRAS C2%ROWTYPE;
BEGIN
OPEN C2;
FETCH C2 INTO REGCOMPRAS;
WHILE C2%FOUND LOOP
 --VISUALIZAR LA COMPRA
   DBMS_OUTPUT.PUT_LINE('----------  ---------  ---------- '||
        '---------- ---------- ------ -------- ----------');
   DBMS_OUTPUT.PUT_LINE(' NUM COMPRA'|| REGCOMPRAS.NUM_COMPRA ||
         ' NOMBRE ALMACÉN: '  || REGCOMPRAS.NOMBRE ||
         ' FECHA DE COMPRA: ' || REGCOMPRAS.FECHA);
  VERDETALLECOMPRAS(REGCOMPRAS.NUM_COMPRA);
  FETCH C2 INTO REGCOMPRAS;
END LOOP;
END;
```

Realizar un UNA PROCEDURE ALMACENADA que visualice por cada almacén el número de compras realizadas y el total importe de sus compras. Visualizar también el total de compras realizadas por los almacenes y el total importe de todos los almacenes.

COD_ALMACEN	NOMBRE	NUM_COMPRAS	TOTAL IMPORTE
XXXXXXX	XXXXXXXXXXXXX	XXXXXX	XXXXXXXX
XXXXXXX	XXXXXXXXXXXXX	XXXXXX	XXXXXXXX
XXXXXXX	XXXXXXXXXXXXX	XXXXXX	XXXXXXXX

TOTAL COMPRAS: XXXXXXXXXXX (totalcompra)
TOTAL IMPORTE TODOS LOS ALMACENES: XXXXXXXX (totalimporte)

Dónde:

NUM_COMPRAS: es el número de compras que realiza cada almacén.

TOTAL IMPORTE: es la suma de los importes de las compras de cada almacén. Los importes se calculan multiplicando el PVP*UNIDADES de los artículos de las compras de los almacenes.

TOTAL COMPRAS: suma del número de compras de los almacenes.

TOTAL IMPORTE TODOS LOS ALMACENES: suma de los totales importe de cada almacén.

```
CREATE OR REPLACE PROCEDURE TOTALALMACENES
IS
CURSOR C1 IS SELECT CODALMA, NOMBRE, COUNT(DISTINCT NUM_COMPRA) NUMCOMPRAS,
             NVL(SUM(UNIDADES * PVP),0) TOTALIMPORTE
             FROM ALMACENES  LEFT JOIN COMPRAS USING (CODALMA)
             LEFT JOIN DET_COMPRAS USING(NUM_COMPRA)
             LEFT JOIN ARTICULOS USING(CODARTI)
             GROUP BY CODALMA, NOMBRE
             ORDER BY CODALMA;
REG C1%ROWTYPE;
TOTALCOMPRAS NUMBER(5) :=0;
TOTALIMPORTES NUMBER(10,2) :=0;
BEGIN
DBMS_OUTPUT.PUT_LINE('COD_ALMACEN NOMBRE NUM_COMPRAS TOTAL
IMPORTE');
DBMS_OUTPUT.PUT_LINE('------------  ------------ ------------  --------------');
OPEN C1;
FETCH C1 INTO REG;
WHILE (C1%FOUND) LOOP
 DBMS_OUTPUT.PUT_LINE(REG.CODALMA || ' ' || REG.NOMBRE
        || ' ' || REG.NUMCOMPRAS || ' ' || REG.TOTALIMPORTE);
  TOTALCOMPRAS:=TOTALCOMPRAS  +  REG.NUMCOMPRAS ;
  TOTALIMPORTES:=TOTALIMPORTES  + REG.TOTALIMPORTE;
  FETCH C1 INTO REG;
END LOOP;
DBMS_OUTPUT.PUT_LINE('TOTAL COMPRAS: ' ||TOTALCOMPRAS);
DBMS_OUTPUT.PUT_LINE('TOTAL IMPORTE TODOS LOS ALMACENES: ' ||TOTALIMPORTES);
CLOSE C1;
```

```
END TOTALALMACENES;
```

2. FUNCIONES.

```
FUNCTION <nombredefuncion> ([<lista de parámetros>])
        RETURN <tipo de valor devuelto >
IS
        -- <declaraciones>;
BEGIN
        -- <instrucciones>;
        RETURN <valor>;
        ...
EXCEPTION
        <excepciones>;
END <nombredefuncion>;
```

Donde los parámetros tienen la siguiente sintaxis:

```
<nombrevariable> [ IN | OUT | IN OUT ] < tipodedato> [ { := | DEFAULT } <valor>]
```

No se deben especificar tamaños ni de los valores de entrada ni de la salida.

En la estructura general de una función se pueden apreciar dos partes:

La especificación de la función: Comienza con la palabra FUNCTION y termina después del RETURN.

El cuerpo de la función: Comienza con la palabra IS y termina con la palabra END opcionalmente seguida del nombre de la función.

En la parte ejecutable (entre BEGIN y EXCEPTION - o end si no hay exception-) debe aparecer RETURN valor. Esta cláusula devuelve el control al programa que llamó a la función asignando el valor devuelto por la función al identificador de la misma en el punto de la llamada.

```
/* Utiliza una función para leer un número */
/* suma el número 10 veces.          */
set serveroutput on
DECLARE
  NUME INT:=0;
  SUMA INT:=0;
  FUNCTION LEER RETURN NUMBER
    IS
       NUM NUMBER(3);
    BEGIN
       NUM :=&N;
       RETURN NUM;
    END LEER;
BEGIN
  NUME:=LEER;
  FOR CON IN 1..10 LOOP
     SUMA:=SUMA+NUME;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('RESUL:' || TO_CHAR(SUMA));
END;
```

El formato habitual de llamada a una función es utilizando la función como parte de una expresión:

```
<variable> :=  <nombredefuncion>(parámetros);
```

En el ejemplo anterior:

nume:=leer; o también nume:=leer();

2.1. FUNCIONES ALMACENADAS.

Al igual que las procedures, las funciones pueden almacenarse como elementos de la base de datos:

```
CREATE OR REPLACE
   FUNCTION nombre_función (parámetros)
            RETURN <tipo de valor devuelto >
IS
            -- <declaraciones>;
BEGIN
           -- <instrucciones>;
           RETURN <valor>;
           ...
EXCEPTION
           <excepciones>;
END <nombredefuncion>;
```

Ejemplo.

Función que devuelve la media de salario de un departamento que recibe como parámetro:

```
CREATE OR REPLACE FUNCTION MEDIA (DEP NUMBER)    RETURN NUMBER IS
  MEDIA NUMBER(9,2);
BEGIN
  SELECT AVG(SALARIO) INTO MEDIA
     FROM EMPLE WHERE DEPT_NO=DEP;
  RETURN MEDIA;
END MEDIA;
```

Ejemplo de uso en bloque anónimo:

```
DECLARE
  MEDIA_SALARIO NUMBER(9,2);
  D NUMBER(4);
BEGIN
   D:=20;
   MEDIA_SALARIO := MEDIA(D);
        DBMS_OUTPUT.PUT_LINE('La media de salario del departamento:' || D || ' es: ' ||
TO_CHAR(MEDIA_SALARIO));
END;
```

Crear una función que reciba un oficio y devuelva el número de empleados de ese oficio. Hacer un bloque PL que utilice la función.

```
CREATE OR REPLACE FUNCTION CUENTAEMPLEOFI   (OFI VARCHAR2)   RETURN NUMBER IS
  CONTADOR NUMBER(9);
BEGIN
 SELECT COUNT(*) INTO CONTADOR
   FROM EMPLE WHERE UPPER(OFICIO) = UPPER(OFI);
 RETURN CONTADOR;
END CUENTAEMPLEOFI;
```

```
EJEMPLO DE USO:
BEGIN
 DBMS_OUTPUT.PUT_LINE('EMPLEADOS DEL OFICIO PROFESOR:' ||
          CUENTAEMPLEOFI('PROFESOR'));
END;
```

3. PARÁMETROS.

Los subprogramas utilizan parámetros para pasar/recibir información. Existen 2 tipos:

Parámetros actuales son las variables o expresiones indicadas en la **llamada** a un subprograma.

Parámetros formales son las variables declaradas en la **especificación** del subprograma.

El paso de parámetros se puede hacer utilizando las notaciones posicional, nominal o mixta (ambas):

Notación posicional: el compilador asocia los parámetros actuales a los formales en base a su posición.

Notación nominal: El símbolo => antes del parámetro actual y después del formal indica al compilador la correspondencia (Parámetro formal=>Parámetro actual).

Notación mixta: Consiste en usar ambas notaciones con la siguiente restricción: la notación posicional debe preceder a la nominal.

Ejemplo

```
CREATE OR REPLACE PROCEDURE CREAR_DEPT (
   N_DNOM  VARCHAR2 DEFAULT 'NUEVO NOM',
   N_LOC   VARCHAR2 DEFAULT 'NUEVA LOC') IS
   NUDEP NUMBER(4);
BEGIN
  SELECT MAX(DEPT_NO)+1 INTO NUDEP FROM DEPART;
  INSERT INTO DEPART
    VALUES (NUDEP,N_DNOM, N_LOC);
END CREAR_DEPT;

LLAMADAS:
EXEC CREAR_DEPT;
EXEC CREAR_DEPT ('MARKETING');
EXEC CREAR_DEPT ('MARKETING', 'NEW YORK');
EXEC CREAR_DEPT ('NEW YORK');       -- > INCORRECTO
EXEC CREAR_DEPT ( , 'NEW YORK');    -- > ILEGAL
/* EN ESTOS DOS ÚLTIMOS CASOS SE PODRÍA USAR LA NOTACIÓN NOMINAL*/
EXEC CREAR_DEPT (N_LOC => 'NEW YORK');
EXEC CREAR_DEPT (N_LOC => 'TOLEDO', N_DNOM => 'COMERCIAL');
```

3.1. TIPOS DE PARÁMETROS

IN Permite pasar valores a un subprograma. No se le puede asignar ningún valor. El parámetro actual puede ser variable, constante, literal o expresión.

OUT Permite devolver valores al bloque que llamó al subprograma. No puede intervenir en ninguna expresión salvo para tomar un valor. El parámetro actual debe ser una variable.

IN OUT Permite pasar un valor inicial y devolver un valor actualizado. Dentro del subprograma actúa como una variable inicializada, puede intervenir en otras expresiones y

puede tomar nuevos valores. El parámetro actual debe ser una variable.

Utilizando los tipos de parámetros.

Procedure que reciba un numero de departamento y devuelva el salario máximo y el nombre de empleado con ese salario.

```
CREATE OR REPLACE PROCEDURE SALMAX (  NUM NUMBER,  SAL_MAX  OUT EMPLE.SALARIO
%TYPE,  N_MAX  OUT EMPLE.APELLIDO%TYPE) IS
  NUDEP NUMBER(4);
BEGIN
   SELECT SALARIO, APELLIDO INTO SAL_MAX, N_MAX FROM EMPLE WHERE SALARIO = (SELECT
MAX(SALARIO) FROM EMPLE WHERE DEPT_NO=NUM) AND DEPT_NO=NUM;
EXCEPTION
 WHEN TOO_MANY_ROWS THEN
                    DBMS_OUTPUT.PUT_LINE(' EXISTEN VARIOS REGISTROS CON MAX SALARIO.');
 WHEN NO_DATA_FOUND THEN
                    DBMS_OUTPUT.PUT_LINE('NO HAY DATOS DE ESE DEPARTAMENTO.');
END SALMAX;
DECLARE
  S_MAX  EMPLE.SALARIO%TYPE ;
  NO_MAX   EMPLE.APELLIDO%TYPE;
BEGIN
 SALMAX(&DEP,S_MAX,NO_MAX);
 DBMS_OUTPUT.PUT_LINE('SALARIO MÁXIMO='||TO_CHAR(S_MAX));
 DBMS_OUTPUT.PUT_LINE('NOMBRE MÁXIMO='||NO_MAX);
END;
```

4. UTILIZANDO SUBPROGRAMAS LOCALES

Se pueden declarar y definir:

Dentro de un bloque PL/SQL.

Dentro de un script SQL+.

Dentro de un package.

Dentro de otros subprogramas. En este caso se declararán y definirán al final de la sección DECLARE después de declarar todos los demás objetos del programa.

EJEMPLOS:

Bloque PL que define una procedure local que visualiza el departamento con más empleados.

```
DECLARE
   REGISTRO DEPART%ROWTYPE;
   CON NUMBER(4);
   PROCEDURE VER(REG DEPART%ROWTYPE) IS
    BEGIN
     DBMS_OUTPUT.PUT_LINE('----LOS DATOS PEDIDOS ----');
     DBMS_OUTPUT.PUT_LINE('NUM:' || TO_CHAR(REG.DEPT_NO));
     DBMS_OUTPUT.PUT_LINE('NOMBRE:' || REG.DNOMBRE);
     DBMS_OUTPUT.PUT_LINE('LOCALIDAD:' || REG.LOC);
```

```
    END VER;
BEGIN
  SELECT * INTO REGISTRO FROM DEPART  WHERE DEPT_NO =
  (SELECT DEPT_NO FROM EMPLE GROUP BY DEPT_NO  HAVING
  COUNT(*)=(SELECT MAX(COUNT(*)) FROM EMPLE GROUP BY DEPT_NO) );
  VER(REGISTRO);
END;
-- ESTE EJEMPLO INSERTA UN NUEVO REGISTRO EN EMPLE
-- UTILIZA UNA PROCEDURE QUE RECIBE LOS DATOS A INSERTAR.
DECLARE
  REGISTRO EMPLE%ROWTYPE;
  PROCEDURE INSERTAR(REG EMPLE%ROWTYPE)
     IS
   BEGIN
      INSERT INTO EMPLE VALUES  (REG.EMP_NO,
      REG.APELLIDO, REG.OFICIO, REG.DIR,
      REG.FECHA_ALT, REG.SALARIO, REG.COMISION,
      REG.DEPT_NO);
   END INSERTAR;
BEGIN
    REGISTRO.EMP_NO:=&NUMERO_EMPLE;
    REGISTRO.APELLIDO:='&NOMBRE_EMPLE';
    REGISTRO.OFICIO:='&OFICIO_EMPLE';
    REGISTRO.DIR:=&NUMERO_DIRECTOR;
    REGISTRO.FECHA_ALT:=SYSDATE;
    REGISTRO.SALARIO:=12000;
    REGISTRO.COMISION:=0;
    REGISTRO.DEPT_NO:=20;
    INSERTAR(REGISTRO);
END;
/*EJEMPLO QUE UTILIZA UNA FUNCIÓN PARA MODIFICAR EL SUELDO DE LOS EMPLEADOS CON
OFICIO EMPLEADO, EL SUELDO SE SUBIRÁ UN 15 %. LA FUNCIÓN RECIBE EL SUELDO Y DEVUELVE EL
SUELDO NUEVO. PROBAR QUÉ OCURRE */
DECLARE
    REGISTRO EMPLE%ROWTYPE;
    NUSUEL  EMPLE.SALARIO%TYPE;
    FUNCTION NUEVSUEL (SUELDO EMPLE.SALARIO%TYPE)
      RETURN  EMPLE.SALARIO%TYPE
      IS
        NUEVOSUELDO EMPLE.SALARIO%TYPE;
      BEGIN
        NUEVOSUELDO:=SUELDO+0.15*SUELDO;
        RETURN NUEVOSUELDO;
    END NUEVSUEL;
BEGIN
   UPDATE EMPLE SET SALARIO=NUEVSUEL(SALARIO)
         WHERE OFICIO='EMPLEADO';
     /* ESTA LLAMADA DA ERROR PUES LA FUNCIÓN DEBE DE ESTAR CREADA Y ALMACENADA EN LA
BASE DE DATOS */
     DBMS_OUTPUT.PUT_LINE('SE HAN ACTUALIZADO : ' ||SQL%ROWCOUNT);
```

```
END;
-- PARA VERLO
SELECT * FROM EMPLE WHERE OFICIO='EMPLEADO';
```