EJERCICIOS RESUELTOS DE PL-SQL

DECLARE...BEGIN...END

a) Ejemplos de creación de un bloque PL/SQL.

1.- Escribir un bloque PL/SQL que escriba el texto 'Hola'

```
SQL> BEGIN
2 DBMS_OUTPUT.PUT_LINE('HOLA');
3 END;
4 /
```

SAVE...

b) Ejemplos de como guardar un bloque PL/SQL.

2.- Escribir un bloque PL/SQL que cuente el número de filas que hay en la tabla productos, deposita el resultado en la variable v_num, y visualiza su contenido.

```
SQL>DECLARE
2 v_num NUMBER;
3 BEGIN
4 SELECT count(*) INTO v_num
5 FROM productos;
6 DBMS_OUTPUT.PUT_LINE(v_num);
7 END;
8 /
```

START...,GET...,RUN

c) Ejemplos como ejecutar o arrancar un bloque PL/SQL.

3.- Cargar y ejecutar el bloque guardado en el archivo PROG01.SQL de la unidad A.

```
SQL> START A:\PROG01.SQL
```

O bien:

```
SQL> GET A:\PROG01.SQL
SQL> RUN
```

CREATE OR REPLACE PROCEDURE...

1.- Ejemplos de creación procedimientos.

1) Escribir un procedimiento que reciba dos números y visualice su suma.

```
CREATE OR REPLACE PROCEDURE sumar_numeros (
num1 NUMBER,
num2 NUMBER)
IS
suma NUMBER(6);
BEGIN
suma := num1 + num2;
DBMS_OUTPUT.PUT_LINE('Suma: '|| suma);
END sumar_numeros;
```

2) Codificar un procedimiento que reciba una cadena y la visualice al revés.

```
CREATE OR REPLACE PROCEDURE cadena_reves(
vcadena VARCHAR2)
AS
vcad_reves VARCHAR2(80);
BEGIN
FOR i IN REVERSE 1..LENGTH(vcadena) LOOP
vcad_reves := vcad_reves || SUBSTR(vcadena,i,1);
END LOOP;
DBMS_OUTPUT.PUT_LINE(vcad_reves);
END cadena_reves;
```

3) Escribir una función que reciba una fecha y devuelva el año, en número, correspondiente a esa fecha.

```
CREATE OR REPLACE FUNCTION anio (
fecha DATE)
RETURN NUMBER
AS
v_anio NUMBER(4);
BEGIN
v_anio := TO_NUMBER(TO_CHAR(fecha, 'YYYY'));
RETURN v_anio;
END anio;
```

CREATE OR REPLACE FUNCTION....

2 .- Ejemplos de creación de funciones.

4) Escribir un bloque PL/SQL que haga uso de la función anterior.

```
DECLARE
n NUMBER(4);
BEGIN
n := anio(SYSDATE);
DBMS_OUTPUT.PUT_LINE('AÑO : '|| n);
END;
```

5) Dado el siguiente procedimiento:

```
CREATE OR REPLACE PROCEDURE crear_depart (
v_num_dept depart.dept_no%TYPE,
v_dnombre depart.dnombre%TYPE DEFAULT 'PROVISIONAL',
v_loc depart.loc%TYPE DEFAULT 'PROVISIONAL')
IS
BEGIN
INSERT INTO depart
VALUES (v_num_dept, v_dnombre, v_loc);
END crear_depart;
```

Indicar cuáles de las siguientes llamadas son correctas y cuáles incorrectas, en este último caso escribir la llamada correcta usando la notación posicional (en los casos que se pueda):

1º. crear_depart;
2º. crear_depart(50);
3º. crear_depart('COMPRAS');
4º. crear_depart(50,'COMPRAS');
5º. crear_depart('COMPRAS', 50);
6º. crear_depart('COMPRAS', 'VALENCIA');
7º. crear_depart(50, 'COMPRAS', 'VALENCIA');
8º. crear_depart('COMPRAS', 50, 'VALENCIA');
9º. crear_depart('VALENCIA', 'COMPRAS');
10º. crear_depart('VALENCIA', 50);

1º Incorrecta: hay que pasar al menos el número de departamento.

2º Correcta.

3º Incorrecta: hay que pasar también el número de departamento.

4º Correcta.

5º Incorrecta: los argumentos están en orden inverso. Solución:

```
crear_depart(50, 'COMPRAS');
```

6º Incorrecta: hay que pasar también el número.

7º Correcta.

8º Incorrecta: el orden de los argumentos es incorrecto.Solución:

```
crear_depart(50, 'COMPRAS', 'VALENCIA');
```

9º Incorrecta: hay que pasar también el número de departamento.

10º Incorrecta: los argumentos están en orden inverso.Solución:

```
crear_depart(50, NULL, 'VALENCIA');
```

6) Desarrollar una función que devuelva el número de años completos que hay entre dos fechas que se pasan como argumentos.

```
CREATE OR REPLACE FUNCTION anios_dif (
fecha1 DATE,
fecha2 DATE)
RETURN NUMBER
AS
v_anios_dif NUMBER(6);
BEGIN
v_anios_dif := ABS(TRUNC(MONTHS_BETWEEN(fecha2,fecha1)
/ 12));
RETURN v_anios_dif;
END anios_dif;
```

7) Escribir una función que, haciendo uso de la función anterior devuelva los trienios que hay entre dos fechas. (Un trienio son tres años completos).

```
CREATE OR REPLACE FUNCTION trienios (
fecha1 DATE,
fecha2 DATE)
RETURN NUMBER
AS
v_trienios NUMBER(6);
BEGIN
v_trienios := TRUNC(anios_dif(fecha1,fecha2) / 3);
RETURN v_trienios;
END;
```

PROCEDIMIENTOS Y FUNCIONES

3 .- Ejemplos de procedimientos y funciones.

8) Codificar un procedimiento que reciba una lista de hasta 5 números y visualice su suma.

```
CREATE OR REPLACE PROCEDURE sumar_5numeros (
Num1 NUMBER DEFAULT 0,
Num2 NUMBER DEFAULT 0,
Num3 NUMBER DEFAULT 0,
Num4 NUMBER DEFAULT 0,
Num5 NUMBER DEFAULT 0)
AS
BEGIN
DBMS_OUTPUT.PUT_LINE(Num1 + Num2 + Num3 + Num4 + Num5);
END sumar_5numeros;
```

9) Escribir una función que devuelva solamente caracteres alfabéticos sustituyendo cualquier otro carácter por blancos a partir de una cadena que se pasará en la llamada.

```
CREATE OR REPLACE FUNCTION sust_por_blancos(
cad VARCHAR2)
RETURN VARCHAR2
AS
nueva_cad VARCHAR2(30);
car CHARACTER;
BEGIN
FOR i IN 1..LENGTH(cad) LOOP
car:=SUBSTR(cad,i,1);
IF (ASCII(car) NOT BETWEEN 65 AND 90)
AND (ASCII(car) NOT BETWEEN 97 AND 122) THEN
car :=' ';
END IF;
nueva_cad := nueva_cad || car;
END LOOP;
RETURN nueva_cad;
END sust_por_blancos;
```

10) Implementar un procedimiento que reciba un importe y visualice el desglose del cambio en unidades monetarias de 1, 5, 10, 25, 50, 100, 200, 500, 1000, 2000, 5000 Ptas. en orden inverso al que aparecen aquí enumeradas.

```
CREATE OR REPLACE PROCEDURE desglose_cambio(
importe NUMBER)
AS
cambio NATURAL := importe;
moneda NATURAL;
v_uni_moneda NATURAL;
BEGIN
DBMS_OUTPUT.PUT_LINE('***** DESGLOSE DE: ' || importe );
WHILE cambio > 0 LOOP
IF cambio >= 5000 THEN
moneda := 5000;
ELSIF cambio >= 2000 THEN
moneda := 2000;
ELSIF cambio >= 1000 THEN
moneda := 1000;
ELSIF cambio >= 500 THEN
moneda := 500;
ELSIF cambio >= 200 THEN
moneda := 200;
ELSIF cambio >= 100 THEN
moneda := 100;
ELSIF cambio >= 50 THEN
moneda := 50;
ELSIF cambio >= 25 THEN
moneda := 25;
ELSIF cambio >= 10 THEN
moneda := 10;
ELSIF cambio >= 5 THEN
moneda := 5;
ELSE
moneda := 1;
END IF;
v_uni_moneda := TRUNC(cambio / moneda);
DBMS_OUTPUT.PUT_LINE(v_uni_moneda ||
' Unidades de: ' || moneda || ' Ptas. ');
cambio := MOD(cambio, moneda);
END LOOP;
END desglose_cambio;
```

11) Codificar un procedimiento que permita borrar un empleado cuyo número se pasará en la llamada.

```
CREATE OR REPLACE PROCEDURE borrar_emple(
num_emple emple.emp_no%TYPE)
AS
BEGIN
DELETE FROM emple WHERE emp_no = num_emple;
END borrar_emple;
```

Nota: El procedimiento anterior devolverá el mensaje
<< Procedimiento PL/SQL terminado con éxito >> aunque no exista el número y, por tanto, no se borre el empleado. Para evitarlo se puede escribir:

```
CREATE OR REPLACE PROCEDURE borrar_emple(
num_emple emple.emp_no%TYPE)
AS
v_row ROWID;
BEGIN
SELECT ROWID INTO v_row FROM emple
WHERE emp_no = num_emple;
DELETE FROM emple WHERE ROWID = v_row;
END borrar_emple;
```

12) Escribir un procedimiento que modifique la localidad de un departamento. El procedimiento recibirá como parámetros el número del departamento y la localidad nueva.

```
CREATE OR REPLACE
PROCEDURE modificar_localidad(
num_depart NUMBER,
localidad VARCHAR2)
AS
BEGIN
UPDATE depart SET loc = localidad
WHERE dept_no = num_depart;
END modificar_localidad;
```

Nota: Lo indicado en la nota del ejercicio anterior se puede aplicar también a este.

13) Visualizar todos los procedimientos y funciones del usuario almacenados en la base de datos y su situación (valid o invalid).

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

Nota: También se puede utilizar la vista ALL_OBJECTS.

CURSOR...IS...

1.- Ejemplos de creación de procedimientos con cursores.

1) Desarrollar un procedimiento que visualice el apellido y la fecha de alta de todos los empleados ordenados por apellido.

```
CREATE OR REPLACE PROCEDURE ver_emple
AS
CURSOR c_emple IS
SELECT APELLIDO, FECHA_ALT
FROM EMPLE
ORDER BY APELLIDO;
v_apellido VARCHAR2(10);
v_fecha DATE;
BEGIN
OPEN c_emple;
FETCH c_emple into v_apellido, v_fecha;
WHILE c_emple%FOUND LOOP
DBMS_OUTPUT.PUT_LINE( v_apellido||' * '||v_fecha);
FETCH c_emple into v_apellido,v_fecha;
END LOOP;
CLOSE c_emple;
END ver_emple;
```

2) Codificar un procedimiento que muestre el nombre de cada departamento y el número de empleados que tiene.

```
CREATE OR REPLACE PROCEDURE ver_emple_depart
AS
CURSOR c_emple IS
SELECT dnombre, COUNT(emp_no)
FROM emple e, depart d
WHERE d.dept_no = e.dept_no(+)
GROUP BY dnombre;
v_dnombre depart.dnombre%TYPE;
v_num_emple BINARY_INTEGER;
BEGIN
OPEN c_emple;
FETCH c_emple into v_dnombre, v_num_emple;
WHILE c_emple%FOUND LOOP
DBMS_OUTPUT.PUT_LINE(v_dnombre||' * '||v_num_emple);
FETCH c_emple into v_dnombre,v_num_emple;
END LOOP;
CLOSE c_emple;
END ver_emple_depart;
```

3) Escribir un procedimiento que reciba una cadena y visualice el apellido y el número de empleado de todos los empleados cuyo apellido contenga la cadena especificada. Al finalizar visualizar el número de empleados mostrados.

```
CREATE OR REPLACE PROCEDURE ver_emple_apell(
cadena VARCHAR2)
AS
cad VARCHAR2(10);
CURSOR c_emple IS
SELECT apellido, emp_no FROM emple
WHERE apellido LIKE cad;
vr_emple c_emple%ROWTYPE;
BEGIN
cad :='%'||cadena||'%';
OPEN c_emple;
FETCH c_emple INTO vr_emple;
WHILE (c_emple%FOUND) LOOP
DBMS_OUTPUT.PUT_LINE(vr_emple.emp_no||' * '||vr_emple.apellido);
FETCH c_emple INTO vr_emple;
END LOOP;
DBMS_OUTPUT.PUT_LINE('NUMERO DE EMPLEADOS: '|| c_emple%ROWCOUNT);
CLOSE c_emple;
END ver_emple_apell;
```

4) Escribir un programa que visualice el apellido y el salario de los cinco empleados que tienen el salario más alto.

```
CREATE OR REPLACE PROCEDURE emp_5maxsal
AS
CURSOR c_emp IS
SELECT apellido, salario FROM emple
ORDER BY salario DESC;
vr_emp c_emp%ROWTYPE;
i NUMBER;
BEGIN
i:=1;
OPEN c_emp;
FETCH c_emp INTO vr_emp;
WHILE c_emp%FOUND AND i<=5 LOOP
DBMS_OUTPUT.PUT_LINE(vr_emp.apellido ||' * '|| vr_emp.salario);
FETCH c_emp INTO vr_emp;
i:=I+1;
END LOOP;
CLOSE c_emp;
END emp_5maxsal;
```

OPEN...FETCH...

2 .- Ejemplos de como como recorrer un cursor.

5) Codificar un programa que visualice los dos empleados que ganan menos de cada oficio.

```
CREATE OR REPLACE PROCEDURE emp_2minsal
AS
CURSOR c_emp IS
SELECT apellido, oficio, salario FROM emple
ORDER BY oficio, salario;
vr_emp c_emp%ROWTYPE;
oficio_ant EMPLE.OFICIO%TYPE;
i NUMBER;
BEGIN
OPEN c_emp;
oficio_ant:='*';
FETCH c_emp INTO vr_emp;
WHILE c_emp%FOUND LOOP
IF oficio_ant <> vr_emp.oficio THEN
oficio_ant := vr_emp.oficio;
i := 1;
END IF;
IF i <= 2 THEN
DBMS_OUTPUT.PUT_LINE(vr_emp.oficio||' * '
||vr_emp.apellido||' * '
||vr_emp.salario);
END IF;
FETCH c_emp INTO vr_emp;
i:=I+1;
END LOOP;
CLOSE c_emp;
END emp_2minsal;
```

*6) Escribir un programa que muestre, en formato similar a las rupturas de control o secuencia vistas en SQL*plus los siguientes datos:*
- Para cada empleado: apellido y salario.
- Para cada departamento: Número de empleados y suma de los salarios del departamento.
- Al final del listado: Número total de empleados y suma de todos los salarios.

```
CREATE OR REPLACE PROCEDURE listar_emple
AS
CURSOR c1 IS
SELECT apellido, salario, dept_no FROM emple
ORDER BY dept_no, apellido;
vr_emp c1%ROWTYPE;
dep_ant EMPLE.DEPT_NO%TYPE;
cont_emple NUMBER(4) DEFAULT 0;
sum_sal NUMBER(9) DEFAULT 0;
tot_emple NUMBER(4) DEFAULT 0;
tot_sal NUMBER(10) DEFAULT 0;
BEGIN
OPEN c1;
FETCH c1 INTO vr_emp;
IF c1%FOUND THEN
dep_ant := vr_emp.dept_no;
END IF;
WHILE c1%FOUND LOOP

/* Comprobación nuevo departamento y resumen */
IF dep_ant <> vr_emp.dept_no THEN
DBMS_OUTPUT.PUT_LINE('*** DEPTO: ' || dep_ant ||
' NUM. EMPLEADOS: '||cont_emple ||
' SUM. SALARIOS: '||sum_sal);
dep_ant := vr_emp.dept_no;
tot_emple := tot_emple + cont_emple;
tot_sal:= tot_sal + sum_sal;
cont_emple:=0;
sum_sal:=0;
END IF;

/* Líneas de detalle */
DBMS_OUTPUT.PUT_LINE(RPAD(vr_emp.apellido,10)|| ' * '
||LPAD(TO_CHAR(vr_emp.salario,'9,999,999'),12));
```

```
/* Incrementar y acumular */
cont_emple := cont_emple + 1;
sum_sal:=sum_sal + vr_emp.salario;

FETCH c1 INTO vr_emp;
END LOOP;
CLOSE c1;

IF cont_emple > 0 THEN

/* Escribir datos del último departamento */
DBMS_OUTPUT.PUT_LINE('*** DEPTO: ' || dep_ant ||
' NUM EMPLEADOS: '|| cont_emple ||
' SUM. SALARIOS: '||sum_sal);
dep_ant := vr_emp.dept_no;
tot_emple := tot_emple + cont_emple;
tot_sal:= tot_sal + sum_sal;
cont_emple:=0;
sum_sal:=0;

/* Escribir totales informe */
DBMS_OUTPUT.PUT_LINE(' ****** NUMERO TOTAL EMPLEADOS: '
||tot_emple ||
' TOTAL SALARIOS: '|| tot_sal);
END IF;
END listar_emple;

/* Nota: este procedimiento puede escribirse de forma que la visualización de los
resultados resulte mas clara incluyendo líneas de separación, cabeceras de columnas,
etcétera. Por razones didácticas no se han incluido estos elementos ya que pueden
distraer y dificultar la comprensión del código. */
```

7) Desarrollar un procedimiento que permita insertar nuevos departamentos según las siguientes especificaciones:
Se pasará al procedimiento el nombre del departamento y la localidad.
El procedimiento insertará la fila nueva asignando como número de departamento la decena siguiente al número mayor de la tabla.
Se incluirá gestión de posibles errores.

```
CREATE OR REPLACE PROCEDURE insertar_depart(
nombre_dep VARCHAR2,
loc VARCHAR2)
AS
CURSOR c_dep IS SELECT dnombre
FROM depart WHERE dnombre = nombre_dep;
v_dummy DEPART.DNOMBRE%TYPE DEFAULT NULL;
v_ulti_num DEPART.DEPT_NO%TYPE;
nombre_duplicado EXCEPTION;
```

```
BEGIN
/* Comprobación de que el departamento no está duplicado */
OPEN c_dep;
FETCH c_dep INTO v_dummy;
CLOSE c_dep;
IF v_dummy IS NOT NULL THEN
RAISE nombre_duplicado;
END IF;

/* Captura del último número y cálculo del siguiente */
SELECT MAX(dept_no) INTO v_ulti_num FROM depart;

/* Inserción de la nueva fila */
INSERT INTO depart VALUES ((TRUNC(v_ulti_num, -1)+10)
, nombre_dep, loc);
EXCEPTION
WHEN nombre_duplicado THEN
DBMS_OUTPUT.PUT_LINE('Err. departamento duplicado');
RAISE;
WHEN OTHERS THEN
RAISE_APPLICATION_ERROR(-20005,'Err. Operación cancelada');
END insertar_depart;
```

8) Escribir un procedimiento que reciba todos los datos de un nuevo empleado procese la transacción de alta, gestionando posibles errores.

```
CREATE OR REPLACE PROCEDURE alta_emp(
num emple.emp_no%TYPE,
ape emple.apellido%TYPE,
ofi emple.oficio%TYPE,
jef emple.dir%TYPE,
fec emple.fecha_alt%TYPE,
sal emple.salario%TYPE,
com emple.comision%TYPE DEFAULT NULL,
dep emple.dept_no%TYPE)
AS
v_dummy_jef EMPLE.DIR%TYPE DEFAULT NULL;
v_dummy_dep DEPART.DEPT_NO%TYPE DEFAULT NULL;
BEGIN
/* Comprobación de que existe el departamento */
SELECT dept_no INTO v_dummy_dep
FROM depart WHERE dept_no = dep;

/* Comprobación de que existe el jefe del empleado */
SELECT emp_no INTO v_dummy_jef
FROM emple WHERE emp_no = jef;
```

```
/* Inserción de la fila */
INSERT INTO EMPLE VALUES
(num, ape, ofi, jef, fec, sal, com, dep);

EXCEPTION
WHEN NO_DATA_FOUND THEN
IF v_dummy_dep IS NULL THEN
RAISE_APPLICATION_ERROR(-20005,
'Err. Departamento inexistente');
ELSIF v_dummy_jef IS NULL THEN
RAISE_APPLICATION_ERROR(-20005,
'Err. No existe el jefe');
ELSE
RAISE_APPLICATION_ERROR(-20005,
'Err. Datos no encontrados(*)');
END IF;
WHEN DUP_VAL_ON_INDEX THEN
DBMS_OUTPUT.PUT_LINE
('Err.numero de empleado duplicado');
RAISE;
END alta_emp;
```

WHILE...FOUND...LOOP...

3 .- Ejemplos como procedimientos con cursores y parámetros de entrada.

9) Codificar un procedimiento reciba como parámetros un numero de departamento, un importe y un porcentaje; y suba el salario a todos los empleados del departamento indicado en la llamada. La subida será el porcentaje o el importe indicado en la llamada (el que sea más beneficioso para el empleado en cada caso empleado).

```
CREATE OR REPLACE PROCEDURE subida_sal1(
num_depar emple.dept_no%TYPE,
importe NUMBER,
porcentaje NUMBER)
AS
CURSOR c_sal IS SELECT salario,ROWID
FROM emple WHERE dept_no = num_depar;
vr_sal c_sal%ROWTYPE;
v_imp_pct NUMBER(10);
```

```
BEGIN
OPEN c_sal;
FETCH c_sal INTO vr_sal;
WHILE c_sal%FOUND LOOP

/* Guardar en v_imp_pct el importe mayor */
v_imp_pct :=
GREATEST((vr_sal.salario/100)*porcentaje,
v_imp_pct);

/* Actualizar */
UPDATE EMPLE SET SALARIO=SALARIO + v_imp_pct
WHERE ROWID = vr_sal.rowid;

FETCH c_sal INTO vr_sal;
END LOOP;
CLOSE c_sal;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Err. ninguna fila actualizada');
END subida_sal1;
```

10) Escribir un procedimiento que suba el sueldo de todos los empleados que ganen menos que el salario medio de su oficio. La subida será del 50% de la diferencia entre el salario del empleado y la media de su oficio. Se deberá asegurar que la transacción no se quede a medias, y se gestionarán los posibles errores.

```
CREATE OR REPLACE PROCEDURE subida_50pct
AS
CURSOR c_ofi_sal IS
SELECT oficio, AVG(salario) salario FROM emple
GROUP BY oficio;
CURSOR c_emp_sal IS
SELECT oficio, salario FROM emple E1
WHERE salario <
(SELECT AVG(salario) FROM emple E2
WHERE E2.oficio = E1.oficio)
ORDER BY oficio, salario FOR UPDATE OF salario;

vr_ofi_sal c_ofi_sal%ROWTYPE;
vr_emp_sal c_emp_sal%ROWTYPE;
v_incremento emple.salario%TYPE;
```

```
BEGIN
COMMIT;
OPEN c_emp_sal;
FETCH c_emp_sal INTO vr_emp_sal;
OPEN c_ofi_sal;
FETCH c_ofi_sal INTO vr_ofi_sal;
WHILE c_ofi_sal%FOUND AND c_emp_sal%FOUND LOOP

/* calcular incremento */
v_incremento :=
(vr_ofi_sal.salario - vr_emp_sal.salario) / 2;

/* actualizar */
UPDATE emple SET salario = salario + v_incremento
WHERE CURRENT OF c_emp_sal;

/* siguiente empleado */
FETCH c_emp_sal INTO vr_emp_sal;

/* comprobar si es otro oficio */
IF c_ofi_sal%FOUND and
vr_ofi_sal.oficio <> vr_emp_sal.oficio THEN
FETCH c_ofi_sal INTO vr_ofi_sal;
END IF;
END LOOP;
CLOSE c_emp_sal;
CLOSE c_ofi_sal;
COMMIT;
EXCEPTION
WHEN OTHERS THEN
ROLLBACK WORK;
RAISE;
END subida_50pct;
```

11) Diseñar una aplicación que simule un listado de liquidación de los empleados según las siguientes especificaciones:

- El listado tendrá el siguiente formato para cada empleado:

```
**********************************************************************
Liquidación del empleado:.................(1)
Dpto:.............(2) Oficio:..........(3)

Salario : ...........(4)
Trienios :............(5)
Comp. Responsabil :............(6)
Comisión :............(7)
------------
Total :............(8)
**********************************************************************
```

Donde:

- 1 ,2, 3 y 4 Corresponden al apellido, departamento, oficio y salario del empleado.
- 5 Es el importe en concepto de trienios. Cada trienio son tres años completos desde la fecha de alta hasta la de emisión y supone 50€.
- 6 Es el complemento por responsabilidad. Será de 100€ por cada empleado que se encuentre directamente a cargo del empleado en cuestión.
- 7 Es la comisión. Los valores nulos serán sustituidos por ceros.
- 8 Suma de todos los conceptos anteriores.

El listado irá ordenado por Apellido.

```
CREATE OR REPLACE PROCEDURE liquidar
AS
CURSOR c_emp IS
SELECT apellido, emp_no, oficio, salario,
NVL(comision,0) comision, dept_no, fecha_alt
FROM emple
ORDER BY apellido;
vr_emp c_emp%ROWTYPE;
v_trien NUMBER(9) DEFAULT 0;
v_comp_r NUMBER(9);
v_total NUMBER(10);
```

```
BEGIN
FOR vr_emp in c_emp LOOP

/* Calcular trienios. Llama a la función trienios
creada en el ejercicio 11.8 */
v_trien := trienios(vr_emp.fecha_alt,SYSDATE)*50;

/* Calcular complemento de responsabilidad. Se
encierra en un bloque pues levantará NO_DATA_FOUND*/
BEGIN
SELECT COUNT(*) INTO v_comp_r
FROM EMPLE WHERE DIR = vr_emp.emp_no;
v_comp_r := v_comp_r *100;
EXCEPTION
WHEN NO_DATA_FOUND THEN
v_comp_r:=0;
END;

/* Calcular el total del empleado */
v_total := vr_emp.salario + vr_emp. comision +
v_trien + v_comp_r;

/* Visualizar datos del empleado */
DBMS_OUTPUT.PUT_LINE('************************************');
DBMS_OUTPUT.PUT_LINE(' Liquidacion de : '|| vr_emp.apellido
||' Dpto: ' || vr_emp.dept_no
|| ' Oficio: ' || vr_emp.oficio);
DBMS_OUTPUT.PUT_LINE(RPAD('Salario:',16)
||LPAD(TO_CHAR(vr_emp.salario,'9,999,999'),12));
DBMS_OUTPUT.PUT_LINE(RPAD('Trienios: ',16)
|| LPAD(TO_CHAR(v_trien,'9,999,999'),12));
DBMS_OUTPUT.PUT_LINE('Comp. Respons: '
||LPAD(TO_CHAR(v_comp_r,'9,999,999'),12));
DBMS_OUTPUT.PUT_LINE(RPAD('Comision: ' ,16)
||LPAD(TO_CHAR(vr_emp.comision,'9,999,999'),12));
DBMS_OUTPUT.PUT_LINE('------------------');
DBMS_OUTPUT.PUT_LINE(RPAD(' Total : ',16)
||LPAD(TO_CHAR(v_total,'9,999,999') ,12));
DBMS_OUTPUT.PUT_LINE('************************************');
END LOOP;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No se ha encontrado ninguna fila');
END liquidar;
```

/* Nota: También se puede utilizar una cláusula SELECT más compleja:

```
CURSOR c_emp IS
SELECT APELLIDO, EMP_NO, OFICIO,
(EMP_CARGO * 10000) COM_RESPONSABILIDAD,
SALARIO, NVL(COMISION, 0) COMISION, DEPT_NO,
TRIENIOS(FECHA_ALT, SYSDATE) * 5000 TOT_TRIENIOS
FROM EMPLE,(SELECT DIR,COUNT(*) EMP_CARGO FROM EMPLE
GROUP BY DIR) DIREC
WHERE EMPLE.EMP_NO = DIREC.DIR(+)
ORDER BY APELLIDO;
```

de esta forma se simplifica el programa y se evita la utilización de variables de trabajo.
*/

12) Crear la tabla T_liquidacion con las columnas apellido, departamento, oficio, salario, trienios, comp_responsabilidad, comisión y total; y modificar la aplicación anterior para que en lugar de realizar el listado directamente en pantalla, guarde los datos en la tabla. Se controlarán todas las posibles incidencias que puedan ocurrir durante el proceso.

```
CREATE TABLE t_liquidacion (
APELLIDO VARCHAR2(10),
DEPARTAMENTO NUMBER(2),
OFICIO VARCHAR2(10),
SALARIO NUMBER(10),
TRIENIOS NUMBER(10),
COMP_RESPONSABILIDAD NUMBER(10),
COMISION NUMBER(10),
TOTAL NUMBER(10)
);

CREATE OR REPLACE PROCEDURE liquidar2
AS
CURSOR c_emp IS
SELECT apellido, emp_no, oficio, salario,
NVL(comision,0) comision, dept_no, fecha_alt
FROM emple
ORDER BY apellido;
vr_emp c_emp%ROWTYPE;
v_trien NUMBER(9) DEFAULT 0;
v_comp_r NUMBER(9);
v_total NUMBER(10);
```

```
BEGIN
COMMIT WORK;
FOR vr_emp in c_emp LOOP

/* Calcular trienios. Llama a la función trienios
creada en el ejercicio 11.8 */
v_trien := trienios(vr_emp.fecha_alt,SYSDATE)*5000;

/* Calcular complemento de responsabilidad. Se
encierra en un bloque pues levantará NO_DATA_FOUND*/
BEGIN
SELECT COUNT(*) INTO v_comp_r
FROM EMPLE WHERE DIR = vr_emp.emp_no;
v_comp_r := v_comp_r *10000;
EXCEPTION
WHEN NO_DATA_FOUND THEN
v_comp_r:=0;
END;

/* Calcular el total del empleado */
v_total := vr_emp.salario + vr_emp. comision +
v_trien + v_comp_r;

/* Insertar los datos en la tabla T_liquidacion */
INSERT INTO t_liquidacion
(APELLIDO, OFICIO, SALARIO, TRIENIOS,
COMP_RESPONSABILIDAD, COMISION, TOTAL)
VALUES
(vr_emp.apellido, vr_emp.oficio, vr_emp.salario,
v_trien, v_comp_r, vr_emp.comision, v_total);
END LOOP;
EXCEPTION
WHEN OTHERS THEN
ROLLBACK WORK;
END liquidar2;
```

CREATE OR REPLACE TRIGGER...

a) Ejemplo de como crear un trigger.

1.- Construir un disparador de base de datos que permita auditar las operaciones de inserción o borrado de datos que se realicen en la tabla emple según las siguientes especificaciones:
*- En primer lugar se creará desde SQL*Plus la tabla auditaremple con la columna col1 VARCHAR2(200).*
- Cuando se produzca cualquier manipulación se insertará una fila en dicha tabla que contendrá:
- Fecha y hora
- Número de empleado
- Apellido
- La operación de actualización INSERCIÓN o BORRADO

```
CREATE TABLE auditaremple (
col1 VARCHAR2(200)
);
```

```
CREATE OR REPLACE TRIGGER auditar_act_emp
BEFORE INSERT OR DELETE
ON EMPLE
FOR EACH ROW
BEGIN
IF DELETING THEN
INSERT INTO AUDITAREMPLE
VALUES(TO_CHAR(sysdate,'DD/MM/YY*HH24:MI*')
|| :OLD.EMP_NO|| '*' || :OLD.APELLIDO || '* BORRADO ');
ELSIF INSERTING THEN
INSERT INTO AUDITAREMPLE
VALUES(TO_CHAR(sysdate,'DD/MM/YY*HH24:MI*')
|| :NEW.EMP_NO || '*' || :NEW.APELLIDO||'* INSERCION ');
END IF;
END;
```

CREATE OR REPLACE TRIGGER....

b) Ejemplo de como crear un trigger cuando actualizamos en la tabla datos.

2.- Escribir un trigger de base de datos un que permita auditar las modificaciones en la tabla empleados insertado en la tabla auditaremple los siguientes datos:
- Fecha y hora
- Número de empleado
- Apellido
- La operación de actualización: MODIFICACIÓN.
- El valor anterior y el valor nuevo de cada columna modificada. (solo las columnas modificadas)

```
CREATE OR REPLACE TRIGGER audit_modif
BEFORE UPDATE ON EMPLE
FOR EACH ROW
DECLARE
v_cad_inser auditaremple.col1%TYPE;
BEGIN
v_cad_inser := TO_CHAR(sysdate,'DD/MM/YY*HH24:MI*') ||:OLD.EMP_NO ||'*
MODIFICACION *';

IF UPDATING ('EMP_NO') THEN
v_cad_inser := v_cad_inser
||:OLD.EMP_NO|| '*'|| :NEW.EMP_NO;
END IF;

IF UPDATING ('APELLIDO') THEN
v_cad_inser := v_cad_inser
||:OLD.APELLIDO|| '*'||:NEW.APELLIDO;
END IF;

IF UPDATING ('OFICIO') THEN
v_cad_inser := v_cad_inser
||:OLD.OFICIO|| '*'||:NEW.OFICIO;
END IF;

IF UPDATING ('DIR') THEN
v_cad_inser := v_cad_inser
||:OLD.DIR|| '*'||:NEW.DIR;
END IF;

IF UPDATING ('FECHA_ALT') THEN
v_cad_inser := v_cad_inser
||:OLD.FECHA_ALT||:NEW.FECHA_ALT;
END IF;
```

```
IF UPDATING ('SALARIO') THEN
v_cad_inser := v_cad_inser
||:OLD.SALARIO|| '*'||:NEW.SALARIO;
END IF;

IF UPDATING ('COMISION') THEN
v_cad_inser := v_cad_inser
||:OLD.COMISION|| '*'||:NEW.COMISION;
END IF;

IF UPDATING ('DEPT_NO') THEN
v_cad_inser := v_cad_inser
||:OLD.DEPT_NO|| '*'||:NEW.DEPT_NO;
END IF;

INSERT INTO AUDITAREMPLE VALUES(v_cad_inser);
END;
```

3.- Escribir un disparador de base de datos que haga fallar cualquier operación de modificación del apellido o del número de un empleado, o que suponga una subida de sueldo superior al 10%.

```
CREATE OR REPLACE TRIGGER fallo_modif
BEFORE UPDATE OF apellido, emp_no, salario
ON emple
FOR EACH ROW
BEGIN
IF UPDATING('emp_no') OR UPDATING('apellido')
OR (UPDATING ('salario') AND
:new.salario>:old.salario*1.1)
THEN
RAISE_APPLICATION_ERROR
(-20001,'Err. Modificacion no permitida');
END IF;
END;
```

CREATE OR REPLACE TRIGGER...

c) Ejemplo de como crear un trigger a partir de una vista.

4.- Suponiendo que disponemos de la vista

```
CREATE VIEW DEPARTAM AS
SELECT DEPART.DEPT_NO, DNOMBRE, LOC, COUNT(EMP_NO) TOT_EMPLE
FROM EMPLE, DEPART
WHERE EMPLE.DEPT_NO (+) = DEPART.DEPT_NO
GROUP BY DEPART.DEPT_NO, DNOMBRE, LOC;
```

Construir un disparador que permita realizar operaciones de actualización en la tabla depart a partir de la vista dptos, de forma similar al ejemplo del trigger t_ges_emplead. Se contemplarán las siguientes operaciones:
- Insertar departamento.
- Borrar departamento.
- Modificar la localidad de un departamento.

```
CREATE OR REPLACE TRIGGER ges_depart
INSTEAD OF DELETE OR INSERT OR UPDATE
ON DEPARTAM
FOR EACH ROW
BEGIN
IF DELETING THEN
DELETE FROM depart WHERE dept_no = :old.dept_no;
ELSIF INSERTING THEN
INSERT INTO depart
VALUES(:new.dept_no, :new.dnombre, :new.loc);
ELSIF UPDATING('loc') THEN
UPDATE depart SET loc = :new.loc
WHERE dept_no = :old.dept_no;
ELSE
RAISE_APPLICATION_ERROR
(-20001,'Error en la actualización');
END IF;
END;
```

CREATE OR REPLACE PACKAGE...

1.- Ejemplo de cómo crear un paquete.

Escribir un paquete completo para gestionar los departamentos. El paquete se llamará ***gest_depart*** *y deberá incluir, al menos, los siguientes subprogramas:*

- ***insertar_nuevo_depart****: permite insertar un departamento nuevo. El procedimiento recibe el nombre y la localidad del nuevo departamento. Creará el nuevo departamento comprobando que el nombre no se duplique y le asignará como número de departamento la decena siguiente al último número de departamento utilizado.*
- ***borrar_depart****: permite borrar un departamento. El procedimiento recibirá dos números de departamento de los cuales el primero corresponde al departamento que queremos borrar y el segundo al departamento al que pasarán los empleados del departamento que se va eliminar. El procedimiento se encargará de realizar los cambios oportunos en los números de departamento de los empleados correspondientes.*
- ***modificar_loc_depart****: modifica la localidad del departamento. El procedimiento recibirá el número del departamento a modificar y la nueva localidad, y realizará el cambio solicitado.*
- ***visualizar_datos_depart****: visualizará los datos de un departamento cuyo número se pasará en la llamada. Además de los datos relativos al departamento, se visualizará el número de empleados que pertenecen actualmente al departamento.*
- ***visualizar_datos_depart****: versión sobrecargada del procedimiento anterior que, en lugar del número del departamento, recibirá el nombre del departamento. Realizará una llamada a la función buscar_depart_por_nombre que se indica en el apartado siguiente.*
- ***buscar_depart_por_nombre****: función local al paquete. Recibe el nombre de un departamento y devuelve el número del mismo.*

```
/************ Cabecera o especificación del paquete **********/
CREATE OR REPLACE PACKAGE gest_depart AS
PROCEDURE  insert_depart
(v_nom_dep    VARCHAR2,
v_loc VARCHAR2);
PROCEDURE  borrar_depar
(v_dep_borrar    NUMBER,
v_dep_nue NUMBER);
PROCEDURE cambiar_localidad
(v_num_dep NUMBER,
v_loc VARCHAR2);
PROCEDURE visualizar_datos_depart
(v_num_dep NUMBER);
PROCEDURE visualizar_datos_depart
(v_nom_dep VARCHAR2);
END gest_depart;
/
```

```
/****************** Cuerpo del paquete *********************/
CREATE OR REPLACE PACKAGE BODY gest_depart AS
FUNCTION buscar_depart_por_nombre /* Función privada */
(v_nom_dep VARCHAR2)
RETURN NUMBER;
/*************************************************************/

PROCEDURE insert_depart(
v_nom_dep VARCHAR2,
v_loc VARCHAR2)
AS
ultimo_dep DEPART.DEPT_NO%TYPE;
nombre_repetido EXCEPTION;
BEGIN

/*Comprobar dpt repetido(Puede levantar NO_DATA_FOUND)*/
DECLARE
nom_dep depart.DNOMBRE%TYPE;
nombre_repetido EXCEPTION;
BEGIN
SELECT dnombre INTO nom_dep FROM depart
WHERE dnombre = v_nom_dep;
RAISE insert_depart.nombre_repetido;
EXCEPTION
WHEN NO_DATA_FOUND THEN
NULL;
WHEN TOO_MANY_ROWS THEN
RAISE insert_Depart.nombre_repetido;
END; /* Fin del bloque de comprobación
de departamento repetido */

/* Calcular el número de departamento e insertar */
SELECT MAX(DEPT_NO) INTO ultimo_dep FROM DEPART;
INSERT INTO DEPART VALUES ((TRUNC(ultimo_dep, -1) +10),
v_nom_dep,v_loc);
EXCEPTION
WHEN nombre_repetido THEN
DBMS_OUTPUT.PUT_LINE
('Err. Nombre de departamento duplicado');
WHEN NO_DATA_FOUND THEN /* Si no había ningún departamento */
INSERT INTO DEPART VALUES (10,v_nom_dep,v_loc);
END insert_depart;
/*************************************************************/
```

```
PROCEDURE borrar_depar
(v_dep_borrar NUMBER,
v_dep_nue NUMBER)
AS
BEGIN
UPDATE emple SET dept_no = v_dep_nue
WHERE DEPT_NO=v_dep_borrar;
DELETE FROM depart WHERE dept_no = v_dep_borrar;
END borrar_depar;
/*************************************************************/

PROCEDURE visualizar_datos_depart
(v_num_dep NUMBER)
AS
vr_dep depart%ROWTYPE;
v_num_empleados NUMBER(4);
BEGIN
SELECT * INTO vr_dep FROM depart
WHERE DEPT_NO=v_num_dep;
SELECT COUNT(*) INTO v_num_empleados FROM
EMPLE WHERE DEPT_NO=v_num_dep;

DBMS_OUTPUT.PUT_LINE
('Número de departamento: '||vr_dep.dept_no);
DBMS_OUTPUT.PUT_LINE
('Nombre del departamento: '||vr_dep.dnombre);
DBMS_OUTPUT.PUT_LINE
('Localidad : '||vr_dep.loc);
DBMS_OUTPUT.PUT_LINE
('Numero de empleados : '||v_num_empleados);
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Err departamento no encontrado');
END visualizar_datos_depart;
/*************************************************************/

PROCEDURE visualizar_datos_depart /* Versión sobrecargada */
(v_nom_dep VARCHAR2)
AS
v_num_dep depart.dept_no%TYPE;
vr_dep depart%ROWTYPE;
v_num_empleados NUMBER(4);
BEGIN
v_num_dep:=buscar_depart_por_nombre(v_nom_dep);
SELECT * INTO vr_dep FROM depart
WHERE dept_no=v_num_dep;
```

```
SELECT COUNT(*) INTO v_num_empleados FROM EMPLE
WHERE dept_no=v_num_dep;

DBMS_OUTPUT.PUT_LINE
('Número de departamento: '||vr_dep.dept_no);
DBMS_OUTPUT.PUT_LINE
('Nombre del departamento: '||vr_dep.dnombre);
DBMS_OUTPUT.PUT_LINE
('Localidad : '||vr_dep.loc);
DBMS_OUTPUT.PUT_LINE
('Numero de empleados : '||v_num_empleados);

EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Err departamento no encontrado');
END visualizar_datos_depart;
/*************************************************************/

FUNCTION buscar_depart_por_nombre
(v_nom_dep VARCHAR2)
RETURN NUMBER
AS
v_num_dep depart.dept_no%TYPE;
BEGIN
SELECT dept_no INTO v_num_dep FROM depart
WHERE DNOMBRE = v_nom_dep;
RETURN v_num_dep;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Err departamento no encontrado');
END buscar_depart_por_nombre;
/*************************************************************/

PROCEDURE cambiar_localidad(
v_num_dep NUMBER,
v_loc VARCHAR2)
AS
BEGIN
UPDATE depart
SET LOC=v_loc
WHERE dept_no=v_num_dep;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Err departamento no encontrado');
END cambiar_localidad;
END gest_depart;
```

CREATE OR REPLACE PACKAGE BODY....

2 .- Ejemplo de como crear un paquete.

Escribir un paquete completo para gestionar los empleados. El paquete se llamará ***gest_emple*** *e incluirá, al menos los siguientes subprogramas:*

- ***insertar_nuevo_emple***
- ***borrar_emple****. Cuando se borra un empleado todos los empleados que dependían de él pasarán a depender del director del empleado borrado.*
- ***modificar_oficio_emple***
- ***modificar_dept_emple***
- ***modificar_dir_emple***
- ***modificar_salario_emple***
- ***modificar_comision_emple***
- ***visualizar_datos_emple****. También se incluirá una versión sobrecargada del procedimiento que recibirá el nombre del empleado.*
- ***buscar_emple_por_nombre****. Función local que recibe el nombre y devuelve el número.*

Todos los procedimientos recibirán el número del empleado seguido de los demás datos necesarios. También se incluirán en el paquete cursores y declaraciones de tipo registro, así como siguientes procedimientos que afectarán a todos los empleados:

- ***subida_salario_pct****: incrementará el salario de todos los empleados el porcentaje indicado en la llamada que no podrá ser superior al 25%.*
- ***subida_salario_imp****: sumará al salario de todos los empleados el importe indicado en la llamada. Antes de proceder a la incrementar los salarios se comprobará que el importe indicado no supera el 25% del salario medio.*

```
/****************** Cabecera del paquete *********************/
CREATE OR REPLACE PACKAGE gest_emple AS

CURSOR c_sal RETURN EMPLE%ROWTYPE;

PROCEDURE insertar_nuevo_emple(
v_num_emp EMPLE.EMP_NO%TYPE,
v_apell EMPLE.APELLIDO%TYPE,
v_oficio EMPLE.OFICIO%TYPE,
v_dir EMPLE.DIR%TYPE,
v_fecha_al EMPLE.FECHA_ALT%TYPE,
v_sal EMPLE.SALARIO%TYPE,
v_comision EMPLE.COMISION%TYPE DEFAULT NULL,
v_num_dep EMPLE.DEPT_NO%TYPE);

PROCEDURE borrar_emple(
v_num_emple NUMBER);
```

```
PROCEDURE modificar_oficio_emple(
v_num_emp EMPLE.EMP_NO%TYPE,
v_oficio EMPLE.OFICIO%TYPE);

PROCEDURE  modificar_dept_emple(
v_num_emp  EMPLE.EMP_NO%TYPE,
v_dept EMPLE.DEPT_NO%TYPE);

PROCEDURE modificar_dir_emple(
v_num_emp EMPLE.EMP_NO%TYPE,
v_direccion EMPLE.DIR%TYPE);

PROCEDURE  modificar_salario_emple(
v_num_emp    EMPLE.EMP_NO%TYPE,
v_salario EMPLE.SALARIO%TYPE);

PROCEDURE modificar_comision_emple(
v_num_emp EMPLE.EMP_NO%TYPE,
v_comis EMPLE.COMISION%TYPE);

PROCEDURE visualizar_datos_emple(
v_num_emp EMPLE.EMP_NO%TYPE);

PROCEDURE visualizar_datos_emple(
v_nombre_emp EMPLE.APELLIDO%TYPE);

PROCEDURE subida_salario_pct(
v_pct_subida NUMBER);

PROCEDURE subida_salario_imp(
v_imp_subida NUMBER);
END gest_emple;

/******************** Cuerpo del paquete **********************/

CREATE OR REPLACE PACKAGE BODY gest_emple AS

CURSOR c_sal RETURN EMPLE%ROWTYPE
IS SELECT * FROM EMPLE;

FUNCTION buscar_emple_por_nombre
(n_emp VARCHAR2)
RETURN NUMBER;
/************************************************************/
```

```
PROCEDURE insertar_nuevo_emple(
v_num_emp EMPLE.EMP_NO%TYPE,
v_apell EMPLE.APELLIDO%TYPE,
v_oficio EMPLE.OFICIO%TYPE,
v_dir EMPLE.DIR%TYPE,
v_fecha_al EMPLE.FECHA_ALT%TYPE,
v_sal EMPLE.SALARIO%TYPE,
v_comision EMPLE.COMISION%TYPE DEFAULT NULL,
v_num_dep EMPLE.DEPT_NO%TYPE)
IS
dir_no_existe EXCEPTION;
BEGIN
DECLARE
v_num_emple EMPLE.EMP_NO%TYPE;
BEGIN
SELECT EMP_NO INTO v_num_emple FROM EMPLE
WHERE EMP_NO=v_dir;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RAISE insertar_nuevo_emple.dir_no_existe;
END;
INSERT INTO EMPLE VALUES (v_num_emp, v_apell, v_oficio,
v_dir, v_fecha_al, v_sal, v_comision, v_num_dep);
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
DBMS_OUTPUT.PUT_LINE('Err. Número de empleado duplicado');
WHEN dir_no_existe THEN
DBMS_OUTPUT.PUT_LINE('Err. No existe el director');
END insertar_nuevo_emple;

/*************************************************************/
PROCEDURE borrar_emple(
v_num_emple NUMBER)
IS
emp_dir EMPLE.DIR%TYPE;
BEGIN
SELECT DIR INTO emp_dir FROM EMPLE
WHERE EMP_NO = v_num_emple;
DELETE FROM EMPLE WHERE EMP_NO = v_num_emple;
UPDATE EMPLE SET DIR = emp_dir WHERE DIR = v_num_emple;
END borrar_emple;

/*************************************************************/
```

```
PROCEDURE modificar_oficio_emple(
v_num_emp EMPLE.EMP_NO%TYPE,
v_oficio EMPLE.OFICIO%TYPE)
IS
BEGIN
UPDATE EMPLE SET OFICIO = v_oficio
WHERE EMP_NO = v_num_emp;
END modificar_oficio_emple;

/************************************************************/
PROCEDURE modificar_dept_emple(
v_num_emp EMPLE.EMP_NO%TYPE,
v_dept EMPLE.DEPT_NO%TYPE)
IS
BEGIN
UPDATE EMPLE SET DEPT_NO = v_dept WHERE EMP_NO = v_num_emp;
END modificar_dept_emple;

/************************************************************/
PROCEDURE modificar_dir_emple(
v_num_emp EMPLE.EMP_NO%TYPE,
v_direccion EMPLE.DIR%TYPE)
IS
BEGIN
UPDATE EMPLE SET DIR = v_direccion WHERE EMP_NO = v_num_emp;
END modificar_dir_emple;

/************************************************************/
PROCEDURE modificar_salario_emple(
v_num_emp EMPLE.EMP_NO%TYPE,
v_salario EMPLE.SALARIO%TYPE)
IS
BEGIN
UPDATE EMPLE SET SALARIO = v_salario WHERE EMP_NO = v_num_emp;
END modificar_salario_emple;

/************************************************************/
PROCEDURE modificar_comision_emple(
v_num_emp EMPLE.EMP_NO%TYPE,
v_comis EMPLE.COMISION%TYPE)
IS
BEGIN
UPDATE EMPLE SET COMISION = v_comis WHERE EMP_NO = v_num_emp;
END modificar_comision_emple;
/************************************************************/
```

```
PROCEDURE visualizar_datos_emple(
v_num_emp EMPLE.EMP_NO%TYPE)
IS
reg_emple EMPLE%ROWTYPE;
BEGIN
SELECT * INTO reg_emple FROM EMPLE WHERE EMP_NO = v_num_emp;
DBMS_OUTPUT.PUT_LINE('NUMERO EMPLEADO: '||reg_emple.EMP_NO);
DBMS_OUTPUT.PUT_LINE('APELLIDO: '||reg_emple.APELLIDO);
DBMS_OUTPUT.PUT_LINE('OFICIO: '||reg_emple.OFICIO);
DBMS_OUTPUT.PUT_LINE('DIRECTOR: '||reg_emple.DIR);
DBMS_OUTPUT.PUT_LINE('FECHA ALTA): '||reg_emple.FECHA_ALT);
DBMS_OUTPUT.PUT_LINE('SALARIO: '||reg_emple.SALARIO);
DBMS_OUTPUT.PUT_LINE('COMISION: '||reg_emple.COMISION);
DBMS_OUTPUT.PUT_LINE('NUMERO DEPARTAMENTO: '||reg_emple.DEPT_NO);
END visualizar_datos_emple;
/*************************************************************/

PROCEDURE visualizar_datos_emple(
v_nombre_emp EMPLE.APELLIDO%TYPE)
IS
v_num_emp EMPLE.EMP_NO%TYPE;
reg_emple EMPLE%ROWTYPE;
BEGIN
v_num_emp:=buscar_emple_por_nombre(v_nombre_emp);
SELECT * INTO reg_emple FROM EMPLE WHERE EMP_NO = v_num_emp;
DBMS_OUTPUT.PUT_LINE('NUMERO EMPLEADO: '||reg_emple.EMP_NO);
DBMS_OUTPUT.PUT_LINE('APELLIDO : '||reg_emple.APELLIDO);
DBMS_OUTPUT.PUT_LINE('OFICIO : '||reg_emple.OFICIO);
DBMS_OUTPUT.PUT_LINE('DIRECTOR : '||reg_emple.DIR);
DBMS_OUTPUT.PUT_LINE('FECHA ALTA: '||reg_emple.FECHA_ALT);
DBMS_OUTPUT.PUT_LINE('SALARIO : '||reg_emple.SALARIO);
DBMS_OUTPUT.PUT_LINE('COMISION : '||reg_emple.COMISION);
DBMS_OUTPUT.PUT_LINE('NUM DEPART: '||reg_emple.DEPT_NO);
END visualizar_datos_emple;
/*************************************************************/

FUNCTION buscar_emple_por_nombre(
n_emp VARCHAR2)
RETURN NUMBER
IS
numero EMPLE.EMP_NO%TYPE;
BEGIN
SELECT EMP_NO INTO numero FROM EMPLE WHERE APELLIDO = n_emp;
RETURN numero;
END buscar_emple_por_nombre;
/*************************************************************/
```

```
PROCEDURE subida_salario_pct(
v_pct_subida NUMBER)
IS
subida_mayor EXCEPTION;
BEGIN
IF v_pct_subida > 25 THEN
RAISE subida_mayor;
END IF;
FOR vr_c_sal IN c_sal LOOP
UPDATE EMPLE SET SALARIO = SALARIO +
(SALARIO * v_pct_subida / 100)
WHERE EMP_NO = vr_c_sal.emp_no;
END LOOP;
EXCEPTION
WHEN subida_mayor THEN
DBMS_OUTPUT.PUT_LINE('Subida superior a la permitida');
END subida_salario_pct;
/************************************************************/

PROCEDURE subida_salario_imp(
v_imp_subida NUMBER)
IS
subida_mayor EXCEPTION;
sueldo_medio NUMBER(10);
BEGIN
SELECT AVG(SALARIO) INTO sueldo_medio FROM EMPLE;
IF v_imp_subida>sueldo_medio THEN
RAISE subida_mayor;
END IF;
FOR vr_c_sal in c_sal LOOP
UPDATE EMPLE SET SALARIO = SALARIO + v_imp_subida
WHERE EMP_NO = vr_c_sal.emp_no;
END LOOP;
EXCEPTION
WHEN subida_mayor THEN
DBMS_OUTPUT.PUT_LINE('Subida superior a la permitida');
END subida_salario_imp;
END gest_emple;
```

SQL DINÁMICO – Paquete DBMS.SQL

1.- Crear un procedimiento que permita consultar todos los datos de la tabla depart a partir de una condición que se indicará en la llamada al procedimiento.

```
CREATE OR REPLACE PROCEDURE consultar_depart
(condicion VARCHAR2,
valor VARCHAR2)
AS
id_cursor INTEGER;
v_comando VARCHAR2(2000);
v_dummy NUMBER;
v_dept_no depart.dept_no%TYPE;
v_dnombre depart.dnombre%TYPE;
v_loc depart.loc%TYPE;
BEGIN
id_cursor := DBMS_SQL.OPEN_CURSOR;
v_comando := ' SELECT dept_no, dnombre, loc
FROM depart
WHERE ' || condicion || ':val_1';
DBMS_OUTPUT.PUT_LINE(v_comando);
DBMS_SQL.PARSE(id_cursor, v_comando, DBMS_SQL.V7);
DBMS_SQL.BIND_VARIABLE(id_cursor, ':val_1', valor);
/* A continuación se especifican las variables que recibirán los valores de la selección*/
DBMS_SQL.DEFINE_COLUMN(id_cursor, 1, v_dept_no);
DBMS_SQL.DEFINE_COLUMN(id_cursor, 2, v_dnombre,14);
DBMS_SQL.DEFINE_COLUMN(id_cursor, 3, v_loc, 14);
v_dummy := DBMS_SQL.EXECUTE(id_cursor);
/* La función FETCH_ROWS recupera filas y retorna el número de filas que quedan */
WHILE DBMS_SQL.FETCH_ROWS(id_cursor)>0 LOOP
/* A continuación se depositarán los valores recuperados en las variables PL/SQL */
DBMS_SQL.COLUMN_VALUE(id_cursor, 1, v_dept_no);
DBMS_SQL.COLUMN_VALUE(id_cursor, 2, v_dnombre);
DBMS_SQL.COLUMN_VALUE(id_cursor, 3, v_loc);
DBMS_OUTPUT.PUT_LINE(v_dept_no || '*' || v_dnombre
|| '*' || v_loc);
END LOOP;
DBMS_SQL.CLOSE_CURSOR(id_cursor);
EXCEPTION
WHEN OTHERS THEN
DBMS_SQL.CLOSE_CURSOR(id_cursor);
RAISE;
END consultar_depart;
```