Prácticas completas PL/SQL 12c

1. Introducción

- Este documento agrupa todos los ejemplos y prácticas que se hacen durante el curso.
- Están los enunciados y al final de cada capítulo se encuentran las soluciones,
- **En el curso, en cada capítulo dispones del código fuente de cada vídeo, para que podáis copiarlo con tranquilidad. Está en Recursos disponibles**.
- Las tablas que vamos a usar pertenecen al usuario HR, que es el usuario de ejemplo de Oracle del que hemos creado en la conexión que se usa durante el curso.
- Básicamente usaremos las siguientes:
    - EMPLOYEES: tabla de empleados
    - DEPARTMENTS: contiene lo departamentos
    - REGIONS: contiene las regiones
    - JOBS: contiene los tipos de trabajos
    - COUNTRIES: contiene los paises
- En las PRÁCTICAS tienes un fichero asociado con la solución
- **Cualquier duda que tengas no dudes en pedir ayuda¡¡¡¡**

2. Práctica ámbito de Variables / Bloques anidados

- Indicar que valores visualiza X en los 3 casos de este ejemplo?

```
SET SERVEROUTPUT ON
DECLARE
   X NUMBER:=10;
BEGIN
   DBMS_OUTPUT.PUT_LINE(X);
   DECLARE
      X NUMBER:=20;
   BEGIN
      DBMS_OUTPUT.PUT_LINE(X);
   END;

DBMS_OUTPUT.PUT_LINE(X);
END;
/
```

- ¿Es este bloque correcto? Si no es así ¿por qué falla?

```
BEGIN
   DBMS_OUTPUT.PUT_LINE(X);
   DECLARE
      X NUMBER:=20;
   BEGIN
      DBMS_OUTPUT.PUT_LINE(X);
   END;
   DBMS_OUTPUT.PUT_LINE(X);
END;
/
```

- ¿Es este bloque correcto? Si es así ¿qué valores visualiza X?

```
SET SERVEROUTPUT ON
DECLARE
   X NUMBER:=10;
BEGIN
   DBMS_OUTPUT.PUT_LINE(X);

   BEGIN
      DBMS_OUTPUT.PUT_LINE(X);
   END;
   DBMS_OUTPUT.PUT_LINE(X);
END;
/
```

Soluciones

- Indicar que valores visualiza X en los 3 casos de este ejemplo?

```
SET SERVEROUTPUT ON
DECLARE
   X NUMBER:=10;
BEGIN
```

```
   DBMS_OUTPUT.PUT_LINE(X);
   DECLARE
     X NUMBER:=20;
   BEGIN
     DBMS_OUTPUT.PUT_LINE(X);
   END;

DBMS_OUTPUT.PUT_LINE(X);
END;
/
```

Solución:
10
20
10

- ¿Es este bloque correcto? Si no es así ¿por qué falla?

```
BEGIN
   DBMS_OUTPUT.PUT_LINE(X);
   DECLARE
     X NUMBER:=20;
   BEGIN
     DBMS_OUTPUT.PUT_LINE(X);
   END;
   DBMS_OUTPUT.PUT_LINE(X);
END;
/
```

Falla porque la variable X está solo en el bloque anidado y por tanto no puede ser vista desde el bloque principal

- ¿Es este bloque correcto? Si es así ¿qué valores visualiza X?

```
SET SERVEROUTPUT ON
DECLARE
   X NUMBER:=10;
BEGIN
   DBMS_OUTPUT.PUT_LINE(X);

   BEGIN
     DBMS_OUTPUT.PUT_LINE(X);
   END;
   DBMS_OUTPUT.PUT_LINE(X);
END;
/
```

Es correcto porque la variable X definida en el padre es vista por el bloque hijo
Los valores visualizados son
10
10
10

3. PRÁCTICAS FUNCIONES PL/SQL

- **Práctica 1- Iniciales**
  - Crea un bloque PL/SQL con tres variables VARCHAR2: nombre, apellido1, apellido2
  - Debes visualizar las iniciales separadas por puntos. Además siempre en mayúscula
  - Por ejemplo alberto pérez García --> A.P.G

- **Práctica 2- Averiguar el nombre del día que naciste, por ejemplo "Martes"**
  - **PISTA (Función TO_CHAR)**

Soluciones

- **Práctica 1- Iniciales**

```
DECLARE
       NOMBRE VARCHAR2(20);
       APELLIDO1 VARCHAR2(20);
       APELLIDO2 VARCHAR2(20);
       INICIALES VARCHAR2(6);
BEGIN
   NOMBRE:='pedro';
   APELLIDO1:='garcia';
   APELLIDO2:='Rodrigiuez';

INICIALES:=SUBSTR(NOMBRE,1,1)||'.'||SUBSTR(APELLIDO1,1,1)||'.'||SUBST
R(APELLIDO2,1,1)||'.';
   DBMS_OUTPUT.PUT_LINE(UPPER(INICIALES));
END;
/
```

- **-- PRACTICA2- DIA DE NACIMIENTO**

```
DECLARE
   FEC_NAC DATE;
   DIA_SEMANA VARCHAR2(100);
BEGIN
   FEC_NAC:=TO_DATE('10/10/1965');
   DIA_SEMANA:=TO_CHAR(FEC_NAC,'DAY');
   DBMS_OUTPUT.PUT_LINE(DIA_SEMANA);
END;
/
```

4. PRÁCTICA COMANDO IF

- **PRÁCTICA 1**
  - Debemos hacer un bloque PL/SQL anónimo, donde declaramos una variable NUMBER y la ponemos algún valor.
  - Debe indicar si el número es PAR o IMPAR. Es decir debemos usar IF..... ELSE para hacer el ejercicio
  - Como pista, recuerda que hay una función en SQL denominada MOD, que permite averiguar el resto de una división.
  - Por ejemplo MOD(10,4) nos devuelve el resto de dividir 10 por 4.

- **PRÁCTICA 2**
  - Crear una variable CHAR(1) denominada TIPO_PRODUCTO.
  - Poner un valor entre "A" Y "E"
  - Visualizar el siguiente resultado según el tipo de producto
    - 'A' --> Electronica
    - 'B' --> Informática
    - 'C' --> Ropa
    - 'D' --> Música
    - 'E' --> Libros
  - Cualquier otro valor debe visualizar "El código es incorrecto"

- **SOLUCIONES**

-- PRACTICA PAR IMPAR

```
DECLARE
   VALOR NUMBER;
   RESULTADO NUMBER;
BEGIN
   VALOR :=10;
   RESULTADO := MOD(VALOR, 2);
   IF RESULTADO = 0 THEN
      DBMS_OUTPUT.PUT_LINE('PAR');
   ELSE
      DBMS_OUTPUT.PUT_LINE('IMPAR');
   END IF;
END;
/
```

--PRÁCTICA TIPO PRODUCTO

```
SET SERVEROUTPUT ON
DECLARE
  TIPO_PRODUCTO CHAR(1);
BEGIN
  TIPO_PRODUCTO:=UPPER('A');
```

```
IF TIPO_PRODUCTO='A' THEN
   DBMS_OUTPUT.PUT_LINE('ELECTRÓNICA');
ELSIF TIPO_PRODUCTO='B' THEN
   DBMS_OUTPUT.PUT_LINE('INFORMÁTICA');
ELSIF TIPO_PRODUCTO='C' THEN
   DBMS_OUTPUT.PUT_LINE('ROPA');
ELSIF TIPO_PRODUCTO='D' THEN
   DBMS_OUTPUT.PUT_LINE('MÚSICA');
ELSIF TIPO_PRODUCTO='E' THEN
   DBMS_OUTPUT.PUT_LINE('LIBRO');
ELSE
   DBMS_OUTPUT.PUT_LINE('TIPO DE PRODUCTO ');
END IF;
END;
```

5. PRÁCTICA CON CASE

- Vamos a crear una variable denominada "usuario", de tipo VARCHAR2(40)
- Vamos a poner dentro el nombre del usuario que somos, usando la función USER de ORacle que devuelve el nombre del usuario con el que estamos conectados (Recuerda que en Oracle no hace falta poner paréntesis si una función no tiene argumentos)
    - usuario:=user
- Luego hacermos un CASE para que nos pinte un mensaje del estilo:
    - Si el usuario es SYS ponemos el mensaje "Eres superadministrador"
    - Si el usuario es SYSTEM ponemos el mensaje "Eres un administrador normal"
    - Si el usuario es HR ponemos el mensaje "Eres de Recursos humanos"
    - Cualquier otro usuario ponemos "usuario no autorizado

SOLUCIONES

```
SET SERVEROUTPUT ON
DECLARE
  USUARIO VARCHAR2(30);
BEGIN
  USUARIO:=USER;
CASE USUARIO
  WHEN 'SYS' THEN DBMS_OUTPUT.PUT_LINE('ERES
SUPERADMINISTRADOR');
  WHEN 'SYSTEM' THEN DBMS_OUTPUT.PUT_LINE('ERES
ADMINISTRADOR NORMAL');
  WHEN 'HR' THEN DBMS_OUTPUT.PUT_LINE('ERES DE RECURSOS
HUMANOS');
  ELSE DBMS_OUTPUT.PUT_LINE('USUARIO NO AUTORIZADO');
END CASE;
END;
/
```

6. PRÁCTICA CON BUCLES

- Práctica 1
  - Vamos a crear la tabla de multiplicar del 1 al 10, con los tres tipos de bucles: LOOP, WHILE y FOR
- Práctica 2
  - Crear una variable llamada TEXTO de tipo VARCHAR2(100).
  - Poner alguna frase
  - Mediante un bucle, escribir la frase al revés, Usamos el bucle WHILE
- Práctica 3
  - Usando la práctica anterior, si en el texto aparece el carácter "x" debe salir del bucle. Es igual en mayúsculas o minúsculas.
  - Debemos usar la cláusula EXIT.
- Práctica 4
  - Debemos crear una variable llamada NOMBRE
  - Debemos pintar tantos asteriscos como letras tenga el nombre. Usamos un bucle FOR
  - Por ejemplo Alberto --> *******
- Práctica 5
  - Creamos dos variables numéricas, "inicio y fin"
  - Las inicializamos con algún valor:
  - Debemos sacar los números que sean múltiplos de 4 de ese rango

SOLUCIONES

- -- PRÁCTICA1- TABLAS DE MULTIPLICAR

```
DECLARE
  X NUMBER;
  Z NUMBER;
BEGIN
   X:=1;
   Z:=1;
   LOOP
      EXIT WHEN X=11;
      DBMS_OUTPUT.PUT_LINE('Tabla de multiplicar del :'||x);
      LOOP
        EXIT WHEN Z=11;
        DBMS_OUTPUT.PUT_LINE(X*Z);
        Z:=Z+1;
      END LOOP;
      Z:=0;
      X:=X+1;
   END LOOP;
END;
```

```
/

DECLARE
   X NUMBER;
   Z NUMBER;
BEGIN
    X:=1;
    Z:=1;
   WHILE X<11 LOOP
    DBMS_OUTPUT.PUT_LINE('Tabla de multiplicar del :'||x);
    WHILE Z<11 LOOP
       DBMS_OUTPUT.PUT_LINE(X*Z);
         Z:=Z+1;
    END LOOP;
    Z:=0;
       X:=X+1;
    END LOOP;
END;
/

BEGIN
   FOR X IN 1..10  LOOP
    DBMS_OUTPUT.PUT_LINE('Tabla de multiplicar del :'||x);
       FOR Z IN 1..10 LOOP
          DBMS_OUTPUT.PUT_LINE(X*Z);
       END LOOP;
    END LOOP;
END;
/
```

- --PRACTICA2- FRASE AL REVES

```
DECLARE
    FRASE VARCHAR2(100);
    LIMITE NUMBER;
    CONTADOR NUMBER;
    FRASE_AL_REVES VARCHAR2(100);
BEGIN
    FRASE:='ESTO ES UNA PRUEBA DE FRSE';
    LIMITE:=LENGTH(FRASE);
    WHILE LIMITE>0 LOOP
       FRASE_AL_REVES:=FRASE_AL_REVES||SUBSTR(FRASE,LIMITE,1);
       LIMITE:=LIMITE-1;
    END LOOP;
       DBMS_OUTPUT.PUT_LINE(FRASE_AL_REVES);
END;
/
```

- --PRACTICA 3. SALIR SI HAY UNA X

```
DECLARE
    FRASE VARCHAR2(100);
    LIMITE NUMBER;
    CONTADOR NUMBER;
    FRASE_AL_REVES VARCHAR2(100);
```

```
BEGIN
   FRASE:='ESTO ES UNA PRUEBA DE XRSE';
   LIMITE:=LENGTH(FRASE);
   WHILE LIMITE>0 LOOP
      EXIT WHEN UPPER((SUBSTR(FRASE,LIMITE,1)))='X';
      FRASE_AL_REVES:=FRASE_AL_REVES||SUBSTR(FRASE,LIMITE,1);
      LIMITE:=LIMITE-1;
   END LOOP;
      DBMS_OUTPUT.PUT_LINE(FRASE_AL_REVES);
END;
/
```

- --PRACTICA4- ASTERISCOS

```
DECLARE
   NOMBRE VARCHAR2(100);
   ASTERISCOS VARCHAR2(100);
BEGIN
   NOMBRE:='ALBERTO';
   FOR I IN 1..LENGTH(NOMBRE) LOOP
      ASTERISCOS:=ASTERISCOS||'*';

   END LOOP;
   DBMS_OUTPUT.PUT_LINE(NOMBRE ||'-->'||ASTERISCOS);
END;
/
```

- --PRACTICA 5- MULTIPLOS DE 4

```
DECLARE
   INICIO NUMBER;
   FINAL NUMBER;
BEGIN
  INICIO:=10;
  FINAL:=200;
  FOR I IN INICIO..FINAL LOOP
   IF MOD(I,4)=0 THEN
      DBMS_OUTPUT.PUT_LINE(I);
   END IF;
  END LOOP;
END;
/
```

7. PRÁCTICAS CON SELECT INTO

- Realiza los siguientes ejemplo. Usa %ROWTYPE U %TYPE si es posible

PRÁCTICA 1

- Crear un bloque PL/SQL que devuelva al salario máximo del departamento 100 y lo deje en una variable denominada salario_maximo y la visualice

PRÁCTICA2

- Visualizar el tipo de trabajo del empleado número 100

PRÁCTICA 3

- Crear una variable de tipo DEPARTMENT_ID y ponerla algún valor, por ejemplo 10.
- Visualizar el nombre de ese departamento y el número de empleados que tiene, poniendo. Crear dos variables para albergar los valores.

PRÁCTICA 4

- Mediante dos consultas recuperar el salario máximo y el salario mínimo e indicar su diferencia

SOLUCIONES

- PRÁCTICA 1- Crear un bloque PL/SQL que devuelva al salario máximo del departamento 100 y lo deje en una variable denominada salario_maximo y la visualice

```
SET SERVEROUTPUT ON
DECLARE
salario_maximo EMPLOYEES.SALARY%TYPE;
BEGIN
SELECT MAX(SALARY) INTO salario_maximo
FROM EMPLOYEES
WHERE DEPARTMENT_ID=100;
DBMS_OUTPUT.PUT_LINE('el salario máximo de ese departamento
es:'||salario_maximo);
END;
```

PRÁCTICA2

- Visualizar el tipo de trabajo del empleado número 100

```
set serveroutput on
DECLARE
   TIPO_TRABAJO employees.JOB_ID%TYPE;
BEGIN
   select job_id into tipo_trabajo from employees
   where employee_id=100;
       dbms_output.put_line('El tipo de trabajo del empleado 100
es:'||tipo_trabajo);
```

```
end;
/
El tipo de trabajo del empleado 100 es: AD_PRES
```

PRÁCTICA 3

- Crear una variable de tipo DEPARTMENT_ID y ponerla algún valor, por ejemplo 10.
- Visualizar el nombre de ese departamento y el número de empleados que tiene

```
set serveroutput on
DECLARE
   COD_DEPARTAMENTO DEPARTMENTS.DEPARTMENT_ID%TYPE:=10;
   NOM_DEPARTAMENTO DEPARTMENTS.DEPARTMENT_NAME%TYPE;
   NUM_EMPLE NUMBER;
BEGIN
  --RECUPERAR EL NOMBRE DEL DEPARTAMENTO
  SELECT DEPARTMENT_NAME INTO NOM_DEPARTAMENTO FROM
DEPARTMENTS WHERE DEPARTMENT_ID=COD_DEPARTAMENTO;
  --RECUPERAR EL NÚMERO DE EMLEADOS DEL DEPARTAMENTO
  SELECT COUNT(*) INTO NUM_EMPLE FROM EMPLOYEES WHERE
DEPARTMENT_ID=COD_DEPARTAMENTO;
  DBMS_OUTPUT.PUT_LINE('EL DEPARTAMENTO
'||NOM_DEPARTAMENTO||' TIENE '||NUM_EMPLE||' EMPLEADOS');
END;
/
```

PRÁCTICA 4

- Mediante dos consultas recuperar el salario máximo y el salario mínimo e indicar su diferencia

```
DECLARE
  MAXIMO NUMBER;
  MINIMO NUMBER;
  DIFERENCIA NUMBER;
BEGIN
   SELECT MAX(SALARY),MIN(SALARY) INTO MAXIMO,MINIMO
   FROM EMPLOYEES;
    DBMS_OUTPUT.PUT_LINE('EL SALARIO MÁXIMO ES:'||MAXIMO);
    DBMS_OUTPUT.PUT_LINE('EL SALARIO MÍNIMO ES:'||MINIMO);
    DIFERENCIA:=MAXIMO-MINIMO;
    DBMS_OUTPUT.PUT_LINE('LA DIFERENCIA ES:'||DIFERENCIA);
END;
/
```

8. PRÁCTICA CON INSERT, UPDATE Y DELETE

- Crear un bloque que inserte un nuevo departamento en la tabla DEPARTMENTS. Para saber el DEPARTMENT_ID que debemos asignar al nuevo departamento primero debemos averiguar el valor mayor que hay en la tabla DEPARTMENTS y sumarle uno para la nueva clave.
    - Location_id debe ser 1000
    - Manager_id debe ser 100
    - Department_name debe ser "INFORMATICA"
    - NOTA: en PL/SQL debemos usar COMMIT y ROLLBACK de la misma forma que lo hacemos en SQL. Por tanto, para validar definitivamente un cambio debemos usar COMMIT.
- Crear un bloque PL/SQL que modifique la LOCATION_ID del nuevo departamento a 1700. En este caso usemos el COMMIT dentro del bloque PL/SQL.
- Por último hacer otro bloque PL/SQL que elimine ese departamento nuevo.

SOLUCIONES

- **PRÁCTICA 1**

```
DECLARE
   MAX_ID NUMBER;
BEGIN
   SELECT MAX(DEPARTMENT_ID) INTO MAX_ID FROM DEPARTMENTS;
   MAX_ID:=MAX_ID+1;
   INSERT INTO departments
(department_id,department_name,manager_id,location_id)
   VALUES (MAX_ID,'INFORMATICA',100,1000);
   COMMIT;
END;
/
```

- **PRÁCTICA 2**

```
BEGIN
UPDATE DEPARTMENTS SET LOCATION_ID=1700
WHERE DEPARTMENT_NAME='INFORMATICA';
COMMIT;
END;
```

- **PRÁCTICA 3**

```
BEGIN
DELETE DEPARTMENTS
WHERE DEPARTMENT_NAME='INFORMATICA';

COMMIT;
END;
```

9. PRÁCTICAS CON EXCEPCIONES

Crea los siguientes ejemplos:

- 1- Crear una SELECT (no un cursor explícito) que devuelva el nombre de un empleado pasándole el EMPLOYEE_ID en el WHERE,
    - Comprobar en primer lugar que funciona pasando un empleado existente
    - Pasar un empleado inexistente y comprobar que genera un error
    - Crear una zona de EXCEPTION controlando el NO_DATA_FOUND para que pinte un mensaje como "Empleado inexistente"
- 2-Modificar la SELECT para que devuelva más de un empleado, por ejemplo poniendo EMPLOYEE_ID> 100. Debe generar un error. Controlar la excepción para que genere un mensaje como "Demasiados empleados en la consulta"
- 3-Modificar la consulta para que devuelva un error de división por CERO, por ejemplo, vamos a devolver el salario en vez del nombre y lo dividimos por 0. En este caso, en vez de controlar la excepción directamente, creamos una sección WHEN OTHERS y pintamos el error con SQLCODE y SQLERR
- 4-El error -00001 es clave primaria duplicada. Aunque ya existe una predefinida (DUP_VAL_ON_INDEX) vamos a crear una excepción no -predefinida que haga lo mismo. o Vamos a usar la tabla REGIONS para hacerlo más fácil o Usamos PRAGMA EXCEPTION_INIT y creamos una excepción denominada "duplicado". Cuando se genere ese error debemos pintar "Clave duplicada, intente otra". o Insertamos una fila en la tabla REGIONS que esté duplicada y vemos que se controla el error.

SOLUCIONES

- PRÁCTICA 1

```
DECLARE
nombre_empleado employees.first_name%TYPE;
BEGIN
   SELECT first_name INTO nombre_empleado FROM employees
   WHERE employee_id=1000;   --EMPLEADO INEXISTENTE
  DBMS_OUTPUT.PUT_LINE(nombre_empleado);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE('No existe el empleado.');

END;
/
```

- PRÁCTICA 2

```
DECLARE

nombre_empleado employees.first_name%TYPE;
BEGIN
   SELECT first_name INTO nombre_empleado FROM employees
   WHERE employee_id=1000;
```

```
    DBMS_OUTPUT.PUT_LINE(nombre_empleado);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No existe el empleado.');
   WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Empleado duplicado.');
END;
/
```

- PRÁCTICA 3

```
DECLARE
   nombre_empleado   employees.first_name%TYPE;
   salario number;
BEGIN
   SELECT  salary iNTO   salario
   FROM
      employees
   WHERE
      employee_id = 100;
   salario:=salario/0;
   dbms_output.put_line(salario);
EXCEPTION
   WHEN no_data_found THEN
      dbms_output.put_line('No existe el empleado.');
   WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('Empleado duplicado.');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('CODIGO:'||SQLCODE);
      DBMS_OUTPUT.PUT_LINE('MENSAJE:'||SQLERRM);
END;
/
```

- PRÁCTICA 4

```
set serveroutput on
DECLARE
duplicado EXCEPTION;
PRAGMA EXCEPTION_INIT(duplicado,-00001);
BEGIN
INSERT INTO REGIONS VALUES (1,'PRUEBA');
COMMIT;
EXCEPTION
when duplicado then
dbms_output.put_line('Registro duplicado');
End;
```

10. PRÁCTICA CON EXCEPCIONES DE USUARIO

- Crear una Excepción personalizada denominada CONTROL_REGIONES.
    - Debe dispararse cuando al insertar o modificar una región queramos poner una clave superior a 200. Por ejemplo usando una variable con ese valor.
    - En ese caso debe generar un texto indicando algo así como “Codigo no permitido. Debe ser inferior a 200”
    - Recordemos que las excepciones personalizadas deben dispararse de forma manual con el RAISE.

SOLUCIONES

- PRÁCTICA1

```
SET SERVEROUTPUT ON
DECLARE
CONTROL_REGIONES EXCEPTION;
CODIGO NUMBER:=201;
BEGIN
if codigo > 200 then
raise control_regiones;
else
INSERT INTO REGIONS VALUES (CODIGO,'PRUEBA');
end if;
exception
when control_regiones then
dbms_output.put_line('El codigo debe ser inferior a 200');
when others then
dbms_output.put_line(SQLcode);
dbms_output.put_line(SQLERRM);
END;
/
```

11. PRÁCTICA CON RAISE_APPLICATION_ERROR

- Modificar la practica anterior para disparar un error con RAISE_APPLICATION en vez de con PUT_LINE.
- Esto permite que la aplicación pueda capturar y gestionar el error que devuelve el PL/SQL

SOLUCIONES

- PRÁCTICA 1

```
SET SERVEROUTPUT ON
DECLARE
CONTROL_REGIONES EXCEPTION;
CODIGO NUMBER:=201;
BEGIN
if codigo > 200 then
raise control_regiones;
else
INSERT INTO REGIONS VALUES (CODIGO,'PRUEBA');
end if;
exception
when control_regiones then
RAISE_APPLICATION_ERROR(-20001,'El codigo debe ser inferior a 200');

when others then
dbms_output.put_line(SQLcode);
dbms_output.put_line(SQLERRM);
END;
/
```

12. PRÁCTICAS DE COLECCIONES Y RECORDS

- Creamos un TYPE RECORD que tenga las siguientes columnas

```
NAME VARCHAR2(100),
SAL EMPLOYEES.SALARY%TYPE,
COD_DEPT EMPLOYEES.DEPARTMENT_ID%TYPE);
```

- Creamos un TYPE TABLE basado en el RECORD anterior
- Mediante un bucle cargamos en la colección los empleados. El campo NAME debe contener FIRST_NAME y LAST_NAME concatenado.
- Para cargar las filas y siguiendo un ejemplo parecido que hemos visto en el vídeo usamos el EMPLOYEE_ID que va de 100 a 206
- A partir de este momento y ya con la colección cargada, hacemos las siguientes operaciones, usando métodos de la colección.
    - Visualizamos toda la colección
    - Visualizamos el primer empleado
    - Visualizamos el último empleado
    - Visualizamos el número de empleados
    - Borramos los empleados que ganan menos de 7000 y visualizamos de nuevo la colección
    - Volvemos a visualizar el número de empleados para ver cuantos se han borrado

SOLUCIONES

13. PRÁCTICAS CON CURSORES

- 1-Hacer un programa que tenga un cursor que vaya visualizando los salarios de los empleados. Si en el cursor aparece el jefe (Steven King) se debe generar un RAISE_APPLICATION_ERROR indicando que el sueldo del jefe no se puede ver.
- 2-Hacemos un bloque con dos cursores. (Esto se puede hacer fácilmente con una sola SELECT pero vamos a hacerlo de esta manera para probar parámetros en cursores)
    - El primero de empleados
    - El segundo de departamentos que tenga como parámetro el MANAGER_ID
    - Por cada fila del primero, abrimos el segundo curso pasando el ID del MANAGER
    - Debemos pintar el Nombre del departamento y el nombre del MANAGER_ID
    - Si el empleado no es MANAGER de ningún departamento debemos poner "No es jefe de nada"
- 3-Crear un cursor con parámetros que pasando el número de departamento visualice el número de empleados de ese departamento
- 4-Crear un bucle FOR donde declaramos una subconsulta que nos devuelva el nombre de los empleados que sean ST_CLERCK. Es decir, no declaramos el cursor sino que lo indicamos directamente en el FOR.
- 5-Creamos un bloque que tenga un cursor para empleados. Debemos crearlo con FOR UPDATE.
    - Por cada fila recuperada, si el salario es mayor de 8000 incrementamos el salario un 2%
    - Si es menor de 800 lo hacemos en un 3%
    - Debemos modificarlo con la cláusula CURRENT OF
    - Comprobar que los salarios se han modificado correctamente.

SOLUCIONES

- Práctica 1

```
DECLARE
CURSOR C1
IS SELECT first_name,last_name,salary from EMPLOYEES;
BEGIN
for i IN C1
LOOP
IF i.first_name='Steven' AND i.last_name='King'
THEN
raise_application_error(-20300,'El salario del jefe no puede ser visto');
ELSE
```

```
DBMS_OUTPUT.PUT_LINE(i.first_name ||' ' || i.last_name || ': '|| i.salary ||
'DLS');
END IF;
END LOOP;
END;
```

- Práctica 2

```
SET SERVEROUTPUT ON
DECLARE
DEPARTAMENTO DEPARTMENTS%ROWTYPE;
jefe DEPARTMENTS.MANAGER_ID%TYPE;
CURSOR C1 IS SELECT * FROM EMployees;
CURSOR C2(j DEPARTMENTS.MANAGER_ID%TYPE)
IS SELECT * FROM DEPARTMENTS WHERE MANAGER_ID=j;
begin
for EMPLEADO in c1 loop
open c2(EMPLEADO.employee_id) ;
FETCH C2 into departamento;
if c2%NOTFOUND then
DBMS_OUTPUT.PUT_LINE(EMPLEADO.FIRST_NAME ||' No es JEFE de
NADA');
ELSE
DBMS_OUTPUT.PUT_LINE(EMPLEADO.FIRST_NAME || 'ES JEFE DEL
DEPARTAMENTO '|| DEPARTAMENTO.DEPARTMENT_NAME);
END IF;
CLOSE C2;
END LOOP;
END;
```

- Práctica 3

```
SET SERVEROUTPUT ON
DECLARE
   CODIGO DEPARTMENTS.DEPARTMENT_ID%TYPE;
   CURSOR C1(COD DEPARTMENTS.DEPARTMENT_ID%TYPE ) IS
SELECT COUNT(*) FROM employees
WHERE DEPARTMENT_ID=COD;
NUM_EMPLE NUMBER;
BEGIN
CODIGO:=10;
OPEN C1(CODIGO);
FETCH C1 INTO NUM_EMPLE;
DBMS_OUTPUT.PUT_LINE('numero de empleados de ' ||codigo||' es
'||num_emple);
end;
```

- Práctica 4

```
BEGIN

  FOR EMPLE IN(SELECT * FROM EMPLOYEES WHERE
JOB_ID='ST_CLERK') LOOP
    DBMS_OUTPUT.PUT_LINE(EMPLE.FIRST_NAME);
```

```
    END LOOP;
END;
```

- Práctica 5

```
SET SERVEROUTPUT ON
DECLARE
CURSOR C1 IS SELECT * FROM EMployees for update;
begin
for EMPLEADO IN C1 LOOP
IF EMPLEADO.SALARY > 8000 THEN
UPDATE EMPLOYEES SET SALARY=SALARY*1.02
WHERE CURRENT OF C1;
ELSE
UPDATE EMPLOYEES SET SALARY=SALARY*1.03
WHERE CURRENT OF C1;
END IF;
END LOOP;
COMMIT;
END ;
/
```

14. PRÁCTICAS CON PROCEDIMIENTOS Y PARÁMETROS

- 1- Crear un procedimiento llamado visualizar que visualice el nombre y salario de todos los empleados.
- 2- Modificar el programa anterior para incluir un parámetro que pase el número de departamento para que visualice solo los empleados de ese departamento
    - Debe devolver el número de empleados en una variable de tipo OUT
- 3- crear un bloque por el cual se de formato a un nº de cuenta suministrado por completo, por ej , 11111111111111111111
    - Formateado a: 1111-1111-11-1111111111
    - Debemos usar un parámetro de tipo IN-OUT

SOLUCIONES

- Práctica 1

```
CREATE OR REPLACE PROCEDURE visualizar IS
CURSOR C1 IS SELECT first_name,salary FROM EMPLOYEES;
v_nombre employees.first_name%TYPE;
v_salario employees.salary%TYPE;
BEGIN
FOR i IN C1
LOOP
DBMS_OUTPUT.PUT_LINE(i.first_name || ' ' || i.salary);
END LOOP;
END;
/
EXECUTE visualizar;
```

- Práctica 2

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE visualizar(departamento NUMBER) IS
CURSOR C1 IS SELECT first_name,salary FROM EMPLOYEES WHERE
department_id=departamento;
v_nombre employees.first_name%TYPE;
v_salario employees.salary%TYPE;
v_conteo NUMBER;
BEGIN
FOR i IN C1
LOOP
DBMS_OUTPUT.PUT_LINE(i.first_name || ' ' || i.salary);
v_conteo:=v_conteo+1;
END LOOP;
DBMS_OUTPUT.PUT_LINE(v_conteo);
END;
```

```
EXECUTE visualizar(60,v_conteo);
DBMS_OUTPUT.PUT_LINE(visualizar.v_conteo);
DECLARE
x NUMBER;
BEGIN
visualizar(60,x);
DBMS_OUTPUT.PUT_LINE(x);
END;
/
```

- Práctica 3

```
CREATE OR REPLACE PROCEDURE formateo_cuenta (numero IN OUT
VARCHAR2)
IS
guardar1 VARCHAR2(20);
guardar2 VARCHAR2(20);
guardar3 VARCHAR2(20);
guardar4 VARCHAR2(20);
BEGIN
guardar1:=substr(numero,1,4);
guardar2:=substr(numero,5,4);
guardar3:=substr(numero,9,2);
guardar4:=substr(numero,10);
numero:=guardar1 || '-' || guardar2 || '-' || guardar3 || '-' || guardar4;
END;
/



DECLARE
x varchar2(30):='15210457901111111111';
BEGIN
formateo_cuenta(x);
dbms_output.put_line(x);
END;
```

15. PRACTICA DE FUNCIONES

- 1-Crear una función que tenga como parámetro un número de departamento y que devuelve la suma de los salarios de dicho departamento. La imprimimos por pantalla.
    - Si el departamento no existe debemos generar una excepción con dicho mensaje
- 2-Modificar el programa anterior para incluir un parámetro de tipo OUT por el que vaya el número de empleados afectados por la query. Debe ser visualizada en el programa que llama a la función. De esta forma vemos que se puede usar este tipo de parámetros también en una función
- 3-Crear una función llamada CREAR_REGION, donde se pase un nombre de región que debe ser insertada en la tabla REGIONS y que devuelva un número.
    - De forma automática debe calcular el código de región más alto, añadir 1 e insertar un registro con el nuevo número y la región que se ha pasado.
    - Si la región no existe debe arrojar un error de control.
    - El valor devuelto es el número que ha asignado a la región

SOLUCIONES

- Práctica 1

```
create or replace FUNCTION salarios_dept(dep_id NUMBER) RETURN
NUMBER
IS
TOTAL_SAL NUMBER;
dept DEPARTMENTS.department_id%TYPE;
BEGIN
   --COMPROBAR QUE EL DEPARTAMENTO EXISTE. SI NO EXISTE SE
DISPARA LA EXCEPCIÓN
   SELECT DEPARTMENT_ID INTO DEPT FROM DEPARTMENTS WHERE
DEPARTMENT_ID=DEP_ID;
   --SI EL DEPARTAMENTO EXISTE CALCULAR EL TOTAL
   SELECT sum(salary) INTO TOTAL_SAL from employees where
department_id=dep_id group by department_id;
   RETURN TOTAL_SAL;
EXCEPTION
WHEN NO_DATA_FOUND THEN
   --SI EL DEPARTAMENTO NO EXISTE DEVUELVE ERROR
   RAISE_APPLICATION_ERROR(-20001,'ERROR, DEPARTAMENTO
'||DEP_ID|| ' NO EXISTE');
END;


--PROBAR LA FUNCIÓN
SET SERVEROUTPUT ON
DECLARE
SAL NUMBER;
DEPT NUMBER:=100;
BEGIN
```

```
SAL:=salarios_dept(DEPT);
DBMS_OUTPUT.PUT_LINE('El salario total del departamento ' || DEPT || ' es: '
|| SAL);
END;
/
```

- Práctica 2

```
CREATE OR REPLACE FUNCTION salarios_dept1(dep_id NUMBER,
n_empleados OUT NUMBER) RETURN NUMBER
IS
sal NUMBER;
BEGIN
        --COMPROBAR QUE EL DEPARTAMENTO EXISTE. SI NO EXISTE
SE DISPARA LA EXCEPCIÓN
   SELECT DEPARTMENT_ID INTO DEPT FROM DEPARTMENTS WHERE
DEPARTMENT_ID=DEP_ID;
        --SI EL DEPARTAMENTO EXISTE CALCULAR TOTALES
        SELECT sum(salary),count(salary) INTO SAL,n_empleados from
employees where department_id=dep_id group by department_id;
        RETURN sal;
END;
/

--PROBAR LA FUNCIÓN
set serveroutput on
declare
TOTAL_SAL NUMBER;
NUM_EMPLE NUMBER;
DEPT NUMBER:=110;
BEGIN
TOTAL_SAL:=salarios_dept1(DEPT,NUM_EMPLE);
DBMS_OUTPUT.PUT_LINE('El salario total del departamento ' || DEPT || ' es: '
|| TOTAL_SAL);
DBMS_OUTPUT.PUT_LINE('El número total de empleados recabados es : ' ||
NUM_EMPLE);
END;
```

- Práctica 3

```
create or replace FUNCTION CREAR_REGION (nombre varchar2)
RETURN NUMBER IS
regiones NUMBER;
NOM_REGION VARCHAR2(100);
BEGIN
        --AVERIGUAR SI EXISTE LA REGIÓN. SI YA EXISTE DAMOS
ERROR. SI NO EXISTE PASAMOS A EXCEPTION Y SEGUIMOS CON EL
PROGRAMA
        SELECT REGION_NAME INTO NOM_REGION FROM REGIONS
WHERE REGION_NAME=UPPER(NOMBRE);
        raise_application_error(-20321,'Esta región ya existe!');
EXCEPTION
```

```
        -- SI LA REGION NO EXISTE LA INSERTAMOS. ES UN EJEMPLO DE
COMO PODEMOS USAR LA EXCEPCION PARA HACER ALGO CORRECTO
        WHEN NO_DATA_FOUND THEN
        SELECT MAX(REGION_ID)+1 INTO REGIONES from REGIONS;
        INSERT INTO REGIONS (region_id,region_name) VALUES
(regiones,upper(nombre));
        RETURN REGIONES;
END;
/

--PROBAR LA FUNCIÓN

DECLARE
N_REGION NUMBER;
BEGIN
N_REGION:=crear_region('NORMANDIA');
DBMS_OUTPUT.PUT_LINE('EL NUMERO ASIGNADO ES:'||N_REGION);
END;
/
```

16. PRÁCTICA CON PAQUETES

- **1-Crear un paquete denominado REGIONES que tenga los siguientes componentes:**
  - **PROCEDIMIENTOS:**
    - ALTA_REGION, con parámetro de código y nombre Región. Debe devolver un error si la región ya existe. Inserta una nueva región en la tabla. Debe llamar a la función EXISTE_REGION para controlarlo.
    - BAJA_REGION, con parámetro de código de región y que debe borrar una región. Debe generar un error si la región no existe, Debe llamar a la función EXISTE_REGION para controlarlo
    - MOD_REGION: se le pasa un código y el nuevo nombre de la región Debe modificar el nombre de una región ya existente. Debe generar un error si la región no existe, Debe llamar a la función EXISTE_REGION para controlarlo
  - **FUNCIONES**
    - CON_REGION. Se le pasa un código de región y devuelve el nombre
    - EXISTE_REGION. Devuelve verdadero si la región existe. Se usa en los procedimientos y por tanto es PRIVADA, no debe aparecer en la especificación del paquete

- **2-Crear un paquete denominado NOMINA que tenga sobrecargado la función CALCULAR_NOMINA de la siguiente forma:**
  - CALCULAR_NOMINA(NUMBER): se calcula el salario del empleado restando un 15% de IRPF.
  - CALCULAR_NOMINA(NUMBER,NUMBER): el segundo parámetro es el porcentaje a aplicar. Se calcula el salario del empleado restando ese porcentaje al salario
  - CALCULAR_NOMINA(NUMBER,NUMBER,CHAR): el segundo parámetro es el porcentaje a aplicar, el tercero vale 'V' . Se calcula el salario del empleado aumentando la comisión que le pertenece y restando ese porcentaje al salario siempre y cuando el empleado tenga comisión.

SOLUCIONES

- Práctica 1

```
--PACKAGE HEAD
CREATE OR REPLACE PACKAGE regiones IS
PROCEDURE alta_region ( codigo NUMBER,nombre VARCHAR2 );
PROCEDURE baja_region ( id NUMBER );
PROCEDURE mod_region ( id NUMBER,nombre VARCHAR2 );
FUNCTION con_regiom ( codigo NUMBER ) RETURN VARCHAR2;
END regiones;
/
--BODY / PROCEDURE ALTA REGIÓN
```

```
CREATE OR REPLACE PACKAGE BODY regiones IS
--FUNCIÓN EXISTE_REGION
FUNCTION existe_region ( id NUMBER,nombre VARCHAR2 ) RETURN
BOOLEAN IS
CURSOR c1 IS
SELECT
region_name,
region_id
FROM
regions;
y VARCHAR2(20);
BEGIN
FOR i IN c1 LOOP
IF
i.region_name = nombre AND i.region_id = id
THEN
RETURN true;
END IF;
END LOOP;
RETURN false;
END;
PROCEDURE alta_region ( codigo NUMBER,nombre VARCHAR2 ) IS
devuelto BOOLEAN;
BEGIN
devuelto := existe_region(codigo,nombre);
IF
devuelto = false
THEN
INSERT INTO regions ( region_id,region_name ) VALUES ( codigo,nombre );
dbms_output.put_line('Inserción correcta');
ELSE
dbms_output.put_line('La región ya existe.');
END IF;
EXCEPTION
WHEN OTHERS THEN
dbms_output.put_line('La ID YA EXISTE (duplicada)');
END;
--PROCEDURE BAJA_REGION
PROCEDURE baja_region ( id NUMBER ) IS
devuelto BOOLEAN;
otro VARCHAR2(20);
BEGIN
SELECT
region_name
INTO
otro
FROM
regions
WHERE
region_id = id;
devuelto := existe_region(id,otro);
IF
devuelto = true
THEN
```

```
DELETE FROM regions WHERE
region_id = id;
dbms_output.put_line('Región con ID ' || id || ' borrada');
END IF;
EXCEPTION
WHEN no_data_found THEN
dbms_output.put_line('La región no existe!');
END;
--PROCEDURE MOD_REGION
PROCEDURE mod_region ( id NUMBER,nombre VARCHAR2 ) IS
devuelto BOOLEAN;
BEGIN
devuelto := existe_region(id,nombre);
IF
devuelto = true
THEN
UPDATE regions
SET
region_name = nombre
WHERE
region_id = id;
dbms_output.put_line('La región ha sido actualizada.');
ELSE
dbms_output.put_line('La región no existe.');
END IF;
END;
--FUNCIÓN CON_REGIOM
FUNCTION con_regiom ( codigo NUMBER ) RETURN VARCHAR2 IS
nombre_devolver VARCHAR2(20);
BEGIN
SELECT
region_name
INTO
nombre_devolver
FROM
regions
WHERE
region_id = codigo;
RETURN nombre_devolver;
END;
END regiones;
/
EXECUTE mod_region(7,'pikachutotal');
EXECUTE regiones.baja_region(10);
EXECUTE regiones.alta_region(10,'Prueba');
SELECT
*
FROM
regions;
DELETE FROM regions WHERE
region_id > 4;
ROLLBACK;
```

- Práctica 2

```
CREATE OR REPLACE PACKAGE NOMINA IS
function calcular_nomina(id number) RETURN NUMBER;
function calcular_nomina(id number,porcentaje varchar) RETURN NUMBER;
function calcular_nomina(id number,porcentaje number, tercero varchar2:='V')
RETURN NUMBER;
END NOMINA;
/
CREATE OR REPLACE PACKAGE BODY NOMINA IS
--PRIMERA FUNCION
•
FUNCTION calcular_nomina ( id NUMBER ) RETURN NUMBER IS
salario_final NUMBER;
salario NUMBER;
BEGIN
SELECT salary INTO salario from employees where employee_id=id;
salario_final := salario - ( salario * 0.15 );
RETURN salario_final;
END;
--SEGUNDA FUNCION
FUNCTION calcular_nomina ( id NUMBER,porcentaje varchar ) RETURN
NUMBER IS
salario_final NUMBER;
salario NUMBER;
BEGIN
SELECT salary INTO salario from employees where employee_id=id;
salario_final := salario - ( salario * (to_number(porcentaje)/100 ));
RETURN salario_final;
END;
--TERCERA FUNCION
FUNCTION calcular_nomina (
id NUMBER,
porcentaje NUMBER,
tercero VARCHAR2 := 'V'
) RETURN NUMBER IS
salario_final NUMBER;
comision NUMBER;
salario NUMBER;
BEGIN
SELECT salary into salario from employees where employee_id=id;
SELECT commission_pct into comision from employees where
employee_id=id;
salario_final := salario - ( salario * (porcentaje/100 )) + (salario*comision);
RETURN salario_final;
END;
END NOMINA;
/
declare
x number;
BEGIN
x:=NOMINA.CALCULAR_NOMINA(150,'8');
DBMS_OUTPUT.PUT_LINE(x);
```

```
END;
/
desc nomina;
```

17. PRÁCTICAS CON TRIGGERS

- 1- Crear un TRIGGER BEFORE DELETE sobre la tabla EMPLOYEES que impida borrar un registro si su JOB_ID es algo relacionado con CLERK
- 2- Crear una tabla denominada AUDITORIA con las siguientes columnas:

```
CREATE TABLE AUDITORIA (
USUARIO VARCHAR(50),
FECHA DATE,
SALARIO_ANTIGUO NUMBER,
SALARIO_NUEVO NUMBER);
```

- 3-Crear un TRIGGER BEFORE INSERT de tipo STATEMENT, de forma que cada vez que se haga un INSERT en la tabla REGIONS guarde una fila en la tabla AUDITORIA con el usuario y la fecha en la que se ha hecho el INSERT
- 4-Crear un TRIGGER BEFORE INSERT en la tabla DEPARTMENTS que al insertar un departamento compruebe que el código no esté repetido y luego que si el LOCATION_ID es NULL le ponga 1700 y si el MANAGER_ID es NULL le ponga 200

SOLUCIONES

- Práctica 1

```
CREATE OR REPLACE TRIGGER t1 BEFORE
DELETE ON employees FOR EACH ROW
BEGIN
IF
:old.job_id LIKE ( '%CLERK' )
THEN
raise_application_error(-20320,'NADA');
END IF;
END;
/
select * from employees;
delete from employees where job_id LIKE ('%CLERK');
```

- Práctica 2

```
CREATE TABLE AUDITORIA (
USUARIO VARCHAR(50),
FECHA DATE,
SALARIO_ANTIGUO NUMBER,
SALARIO_NUEVO NUMBER);
```

- Práctica 3

```
CREATE TRIGGER T2 BEFORE INSERT ON REGIONS
BEGIN
INSERT INTO AUDITORIA (usuario, fecha)
VALUES (user,sysdate);
END;
/
```

```
INSERT INTO REGIONS VALUES (20,'Prueba');
SELECT USER FROM DUAL;
select * from auditoria;
```

- Práctica 4

18. Práctica completa

18.1. Crear las tablas y datos necesarios

- Vamos a crear cuatro tablas. Podemos ejecutar el script "**creación_ddl.sql**" que se encuentra en los recursos del capítulo o bien podéis copiarlo y pegarlo de este documento

FACTURAS
LINEAS_FACTURA
LOG_CONTROL
PRODUCTOS

- Las columnas de cada tabla con la siguientes:

FACTURAS

COD_FACTURA	**NUMBER**
FECHA	DATE
DESCRIPCIÓN	VARCHAR2(100)

- Clave primaria: COD_FACTURA

LINEAS_FACTURAS

COD_PRODUCTO	**NUMBER**
COD_FACTURA	NUMBER
PVP	NUMBER
UNIDADES	NUMBER
FECHA	DATE

- Clave Primaria: COD_FACTURA+COD_PRODUCTO
- Referencia a la tabla FACTURAS y a la tabla PRODUCTOS

PRODUCTOS

COD_PRODUCTO	**NUMBER**
NOMBRE_PRODUCTO	**VARCHAR2(100)**
PVP	**NUMBER**

TOTAL_VENDIDO	**NUMBER**

- Clave primaria: COD_PRODUCTO

CONTROL_LOG

COD_EMPLEADO	**NUMBER**
FECHA	**DATE**
TABLA_AFECTADA	**VARCHAR2(50)**
COD_OPERACÍON	**CHAR(1)**

- La columna COD_OPERACION debe valer:(I, U, D → INSERT, UPDATE ,DELETE)

SCRIPTS DE CREACIÓN

- Los scripts son los siguientes:

```
--------------------------------------------------------
--  DDL for Table LINEAS_FACTURA
--------------------------------------------------------

  CREATE TABLE "HR"."LINEAS_FACTURA"
   (    "COD_FACTURA" NUMBER,
        "COD_PRODUCTO" NUMBER,
        "PVP" NUMBER,
        "UNIDADES" NUMBER,
        "FECHA" DATE
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Table FACTURAS
--------------------------------------------------------

  CREATE TABLE "HR"."FACTURAS"
```

```
  (   "COD_FACTURA" NUMBER(5,0),
      "FECHA" DATE,
      "DESCRIPCION" VARCHAR2(100 BYTE)
  ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Table PRODUCTOS
--------------------------------------------------------

  CREATE TABLE "HR"."PRODUCTOS"
  (   "COD_PRODUCTO" NUMBER,
      "NOMBRE_PRODUCTO" VARCHAR2(50 BYTE),
      "PVP" NUMBER,
      "TOTAL_VENDIDOS" NUMBER
  ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Table CONTROL_LOG
--------------------------------------------------------

  CREATE TABLE "HR"."CONTROL_LOG"
  (   "COD_EMPLEADO" NUMBER,
      "FECHA" DATE,
      "TABLA" VARCHAR2(20 BYTE),
      "COD_OPERACION" CHAR(1 BYTE)
  ) SEGMENT CREATION IMMEDIATE
```

```
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
REM INSERTING into HR.LINEAS_FACTURA
SET DEFINE OFF;
--------------------------------------------------------
--  DDL for Index LINEAS_FACTURA_PK
--------------------------------------------------------


  CREATE UNIQUE INDEX "HR"."LINEAS_FACTURA_PK" ON "HR"."LINEAS_FACTURA" ("COD_FACTURA", "COD_PRODUCTO")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
----


--------------------------------------------------------
--  Constraints for Table LINEAS_FACTURA
--------------------------------------------------------


  ALTER TABLE "HR"."LINEAS_FACTURA" ADD CONSTRAINT "LINEAS_FACTURA_PK" PRIMARY KEY ("COD_FACTURA", "COD_PRODUCTO")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS"  ENABLE;
  ALTER TABLE "HR"."LINEAS_FACTURA" MODIFY ("COD_PRODUCTO" NOT NULL ENABLE);
  ALTER TABLE "HR"."LINEAS_FACTURA" MODIFY ("COD_FACTURA" NOT NULL ENABLE);
```

```
REM INSERTING into HR.PRODUCTOS
SET DEFINE OFF;
Insert into HR.PRODUCTOS (COD_PRODUCTO,NOMBRE_PRODUCTO,PVP,TOTAL_VENDIDOS) values ('1','TORNILLO','1',null);
Insert into HR.PRODUCTOS (COD_PRODUCTO,NOMBRE_PRODUCTO,PVP,TOTAL_VENDIDOS) values ('2','TUERCA','5',null);
Insert into HR.PRODUCTOS (COD_PRODUCTO,NOMBRE_PRODUCTO,PVP,TOTAL_VENDIDOS) values ('3','ARANDELA','4',null);
Insert into HR.PRODUCTOS (COD_PRODUCTO,NOMBRE_PRODUCTO,PVP,TOTAL_VENDIDOS) values ('4','MARTILLO','40',null);
Insert into HR.PRODUCTOS (COD_PRODUCTO,NOMBRE_PRODUCTO,PVP,TOTAL_VENDIDOS) values ('5','CLAVO','1',null);
```

Práctica final PL/SQL 12c

19. Objetivos

- Esta práctica pretende hacer un repaso de los componentes más importantes de PL/SQL: procedimintos, funciones, paquetes y triggers.

20. Crear las tablas y datos necesarios

- Vamos a crear cuatro tablas. Podemos ejecutar el script "**creación_ddl.sql**" que se encuentra en los recursos del capítulo o bien podéis copiarlo y pegarlo de este documento

FACTURAS
LINEAS_FACTURA
LOG_CONTROL
PRODUCTOS

- Las columnas de cada tabla con la siguientes:

FACTURAS

COD_FACTURA	**NUMBER**
FECHA	DATE
DESCRIPCIÓN	VARCHAR2(100)

- Clave primaria: COD_FACTURA

LINEAS_FACTURAS

COD_PRODUCTO	**NUMBER**
COD_FACTURA	NUMBER
PVP	NUMBER
UNIDADES	NUMBER
FECHA	DATE

- Clave Primaria: COD_FACTURA+COD_PRODUCTO
- Referencia a la tabla FACTURAS y a la tabla PRODUCTOS

PRODUCTOS

COD_PRODUCTO	**NUMBER**
NOMBRE_PRODUCTO	**VARCHAR2(100)**
PVP	**NUMBER**
TOTAL_VENDIDO	**NUMBER**

- Clave primaria: COD_PRODUCTO

CONTROL_LOG

COD_EMPLEADO	**NUMBER**
FECHA	**DATE**
TABLA_AFECTADA	**VARCHAR2(50)**
COD_OPERACÍON	**CHAR(1)**

- La columna COD_OPERACION debe valer:(I, U, D → INSERT, UPDATE ,DELETE)

SCRIPTS DE CREACIÓN

- Los scripts son los siguientes:

```
--------------------------------------------------------
--  DDL for Table LINEAS_FACTURA
--------------------------------------------------------


  CREATE TABLE "HR"."LINEAS_FACTURA"
   (    "COD_FACTURA" NUMBER,
        "COD_PRODUCTO" NUMBER,
        "PVP" NUMBER,
        "UNIDADES" NUMBER,
        "FECHA" DATE
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
```

```
--------------------------------------------------------
--  DDL for Table FACTURAS
--------------------------------------------------------


  CREATE TABLE "HR"."FACTURAS"
   (    "COD_FACTURA" NUMBER(5,0),
        "FECHA" DATE,
        "DESCRIPCION" VARCHAR2(100 BYTE)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Table PRODUCTOS
--------------------------------------------------------


  CREATE TABLE "HR"."PRODUCTOS"
   (    "COD_PRODUCTO" NUMBER,
        "NOMBRE_PRODUCTO" VARCHAR2(50 BYTE),
        "PVP" NUMBER,
        "TOTAL_VENDIDOS" NUMBER
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Table CONTROL_LOG
--------------------------------------------------------


  CREATE TABLE "HR"."CONTROL_LOG"
   (    "COD_EMPLEADO" NUMBER,
```

```
      "FECHA" DATE,
      "TABLA" VARCHAR2(20 BYTE),
      "COD_OPERACION" CHAR(1 BYTE)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
REM INSERTING into HR.LINEAS_FACTURA
SET DEFINE OFF;
--------------------------------------------------------
--  DDL for Index LINEAS_FACTURA_PK
--------------------------------------------------------


  CREATE UNIQUE INDEX "HR"."LINEAS_FACTURA_PK" ON "HR"."LINEAS_FACTURA" ("COD_FACTURA", "COD_PRODUCTO")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
----


--------------------------------------------------------
--  Constraints for Table LINEAS_FACTURA
--------------------------------------------------------


  ALTER TABLE "HR"."LINEAS_FACTURA" ADD CONSTRAINT "LINEAS_FACTURA_PK" PRIMARY KEY ("COD_FACTURA", "COD_PRODUCTO")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS"  ENABLE;
```

```
  ALTER TABLE "HR"."LINEAS_FACTURA" MODIFY ("COD_PRODUCTO" NOT NULL ENABLE);
  ALTER TABLE "HR"."LINEAS_FACTURA" MODIFY ("COD_FACTURA" NOT NULL ENABLE);


  REM INSERTING into HR.PRODUCTOS
SET DEFINE OFF;
Insert into HR.PRODUCTOS (COD_PRODUCTO,NOMBRE_PRODUCTO,PVP,TOTAL_VENDIDOS) values ('1','TORNILLO','1',null);
Insert into HR.PRODUCTOS (COD_PRODUCTO,NOMBRE_PRODUCTO,PVP,TOTAL_VENDIDOS) values ('2','TUERCA','5',null);
Insert into HR.PRODUCTOS (COD_PRODUCTO,NOMBRE_PRODUCTO,PVP,TOTAL_VENDIDOS) values ('3','ARANDELA','4',null);
Insert into HR.PRODUCTOS (COD_PRODUCTO,NOMBRE_PRODUCTO,PVP,TOTAL_VENDIDOS) values ('4','MARTILLO','40',null);
Insert into HR.PRODUCTOS (COD_PRODUCTO,NOMBRE_PRODUCTO,PVP,TOTAL_VENDIDOS) values ('5','CLAVO','1',null);
```

21. Componentes de la práctica

- La práctica pretende realizar los componentes necesarios para gestionar esas tablas. En concreto:
    - Paquete para gestionar las facturas
    - Paquete para gestionar las líneas de factura
    - Triggers para controlar el acceso a las tablas
    -

PAQUETE FACTURAS

PROCEDIMIENTOS

- ALTA_FACTURA (COD_FACTURA, FECHA,DESCRIPCIÓN).
    - Debe dar de alta una factura con los valores indicados en los parámetros
    - Debe comprobar que no se duplica
- BAJA_FACTURA (cod_factura).
    - Debe borrar la factura indicada en el parámetros
    - Debe borrar también las líneas de facturas asociadas en la tabla LINEAS_FACTURA.
- MOD_DESCRI(COD_FACTURA, DESCRIPCION).
    - Debe modificar la descripción de la factura que tenga el código del parámetro con la nueva descripción
- MOD_FECHA (COD_FACTURA,FECHA).
    - Debe modificar la descripción de la factura que tenga el código del parámetro con la nueva fecha
    -

FUNCIONES

- NUM_FACTURAS(FECHA_INICIO,FECHA_FIN).
    - Devuelve el número de facturas que hay entre esas fechas
- TOTAL_FACTURA(COD_FACTURA.)
    - Devuelve el total de la factura con ese código. Debe consultar el campo TOTAL_VENTAS de la tabla LINEAS_FACTURA

PAQUETE LINEA_FACTURAS

PROCEDIMIENTOS

- ALTA_LINEA (COD_FACTURA, COD_PRODUCTO, UNIDADES, FECHA)
    - Procedimiento para insertar una línea de Factura
    - Debe comprobar que existe ya la factura antes de insertar el registro.
    - También debemos comprobar que existe el producto en la tabla de PRODUCTOS.
    - El PVP debemos seleccionarlo de la tabla PRODUCTOS
- BAJA_LINEA (cod_factura, COD_PRODUCTO
    - Damos de baja la línea con esa clave primaria)
- MOD_PRODUCTO(COD_FACTURA,COD_PRODUCTO,PARAMETRO)
    - Se trata de 2 métodos sobrecargados, es decir el segundo parámetro debe admitir los siguientes valores:
        - MOD_PRODUCTO(COD_FACTURA,COD_PRODUCTO, UNIDADES)
        - MOD_PRODUCTO(COD_FACTURA,COD_PRODUCTO, FECHA)
    - Por tanto, debe modificar o bien unidades si se le pasa un NUMBER o bien la fecha si se le pasa un DATE

FUNCIONES

- NUM_LINEAS(COD_FACTURA)
    - Devuelve el número de líneas de la factura

TRIGGERS

Triggers de tipo sentencia

- Creamos 2 triggers de tipo SENTENCIA, uno para la tabla FACTURAS y otro para la tabla LINEAS_FACTURA
- Cada cambio en alguna de las tablas (Insert, update, delete), debe generar una entrada en la tabla CONTROL_LOG con los datos siguientes:
    - Tabla (FACTURAS O LONEAS_FACTURA)
    - Fecha → usamos la función SYSDATE
    - Usuario que lo ha realizado → función USER
    - Operación realizada (I-U-D)

Trigger de tipo fila

- La columna TOTAL_VENDIDO, de la tabla PRODUCTOS mantiene el total de ventas de un determinado producto.

- Prac controlaro, creamos un Trigger de tipo fila sobre la tabla LINEAS_FACTURA, de forma que cada vez que se añada, cambie o borre una línea se actualice en la tabla PRODUCTOS la columna TOTAL_VENDIDO.
- Si se inserta una nueva línea con ese producto, se debe añadir el total al campo.
- Si se borra la línea debemos restar el total
- Si se modifica, debemos comprobar si el valor antiguo era superior al nuevo y sumamos o restamos dependiendo del resultado

SOLUCIONES A LA PRÁCTICA FINAL

- Páquete FACTURAS

```
create or replace PACKAGE FACTURASP AS
PROCEDURE ALTA_FACTURA (CODIGO NUMBER, FECHA DATE, DESCRIP VARCHAR2);
PROCEDURE BAJA_FACTURA (CODIGO NUMBER);
PROCEDURE MOD_DESCRI(CODIGO NUMBER,DESCRI VARCHAR2);
PROCEDURE MOD_FECHA(CODIGO NUMBER, FECHA_MOD DATE);
FUNCTION NUM_FACTURAS (FECHA_INICIO DATE,FECHA_FIN DATE) RETURN NUMBER;
FUNCTION TOTAL_FACTURA(CODIGO NUMBER) RETURN NUMBER;
END FACTURASp;
/



create or replace PACKAGE BODY facturasp AS


--FUNCIÓN ADICIONAL PARA COMPROBAR SI EXISTE UNA FACTURA, Se trata de una función PRIVADA


   FUNCTION existe (
      codigo NUMBER
   ) RETURN BOOLEAN IS
      cont   NUMBER := 0;
      cod_f facturas.cod_factura%type;


   BEGIN
 --Comprobar si existe la factura:
      SELECT
         cod_factura into cod_f FROM  facturas
         where cod_factura=codigo;
         RETURN true;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         return false;
      WHEN OTHERS THEN
          RAISE_APPLICATION_ERROR(-20000,'ERROR'|| sqlcode);
```

```
   END;

--PROCEDURE ALTA_FACTURA

   PROCEDURE alta_factura (
      codigo    NUMBER,
      fecha     DATE,
      descrip   VARCHAR2
   ) AS
      devuelto   BOOLEAN;
      erroryaexiste EXCEPTION;
   BEGIN
      devuelto := existe(codigo);
      IF not devuelto   THEN
         INSERT INTO facturas VALUES (codigo,fecha, descrip );
         commit;

      ELSE
         RAISE erroryaexiste;
      END IF;

   EXCEPTION
      WHEN erroryaexiste THEN
         RAISE_APPLICATION_ERROR(-20001,'ERROR, ESE NÚMERO DE
FACTURA YA EXISTE, NO SE ADMITEN DUPLICADOS');
      WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20000,'ERROR'|| sqlcode);
   END;

--PROCEDURE BAJA_FACTURA.

   PROCEDURE baja_factura (
      codigo NUMBER
   ) AS
      errornoexiste EXCEPTION;
      devuelto   BOOLEAN;
   BEGIN
```

```
    devuelto := existe(codigo);
    IF
      devuelto = true
    THEN
      DELETE FROM lineas_factura WHERE
        cod_factura = codigo;

      DELETE FROM facturas WHERE
        cod_factura = codigo;
        commit;

    ELSE
      RAISE errornoexiste;
    END IF;

  EXCEPTION
    WHEN errornoexiste THEN
      RAISE_APPLICATION_ERROR(-20001,'ERROR, ESE NÚMERO DE FACTURA NO EXISTE');
    WHEN OTHERS THEN
      RAISE_APPLICATION_ERROR(-20000,'ERROR'|| sqlcode);
  END;

--PROCEDURE MOD_DESCRI

  PROCEDURE mod_descri (
    codigo NUMBER,
    descri VARCHAR2
  ) AS
    errornoexiste EXCEPTION;
    devuelto   BOOLEAN;
  BEGIN
    devuelto := existe(codigo);
    IF
      devuelto = true
    THEN
      UPDATE facturaS
```

```
            SET
                descripcion = descri
            WHERE
                cod_factura = codigo;
        commit;
        ELSE
            RAISE errornoexiste;
        END IF;

    EXCEPTION
         WHEN errornoexiste THEN
          RAISE_APPLICATION_ERROR(-20001,'ERROR, ESE NÚMERO DE FACTURA NO EXISTE');
        WHEN OTHERS THEN
          RAISE_APPLICATION_ERROR(-20000,'ERROR'|| sqlcode);
    END;



--Procedure MOD_FECHA

    PROCEDURE mod_fecha (
        codigo NUMBER,
        fecha_mod DATE
    ) IS
        devuelto   BOOLEAN;
        errornoexiste EXCEPTION;
    BEGIN
        devuelto := existe(codigo);
        IF
            devuelto = true
        THEN
            UPDATE facturas
                SET
                    fecha = fecha_mod
            WHERE
                cod_factura = codigo;
            commit;
```

```
      ELSE
        RAISE errornoexiste;
      END IF;

   EXCEPTION
       WHEN errornoexiste THEN
        RAISE_APPLICATION_ERROR(-20001,'ERROR, ESE NÚMERO DE FACTURA NO EXISTE');
      WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20000,'ERROR'|| sqlcode);
   END;

--Función NUM_FACTURAS

   FUNCTION num_facturas (
      fecha_inicio DATE,
      fecha_fin DATE
   ) RETURN NUMBER AS
      cont   NUMBER := 0;
   BEGIN
      SELECT COUNT(*) INTO CONT FROM FACTURAS WHERE fecha BETWEEN fecha_inicio AND fecha_fin;

      RETURN cont;
   END;

--FUNCION TOTAL_FACTURA

   FUNCTION total_factura (
      codigo NUMBER
   ) RETURN NUMBER AS
      total_a_devolver   NUMBER;
   BEGIN
      SELECT
        SUM(pvp*UNIDADES)
      INTO
        total_a_devolver
      FROM
```

```
        lineas_factura
    WHERE
        cod_factura = codigo;

    RETURN total_a_devolver;
  END;

END facturasp;
```

- Líneas Factura

```
create or replace PACKAGE lineas_FACTURASP AS
PROCEDURE ALTA_linea (COdigo number,cod_prod NUMBER,unid number,FEC DATE);
PROCEDURE BAJA_linea (CODIGO number,cod_prod NUMBER);
PROCEDURE MOD_producto(CODIGO NUMBER,cod_prod number,unid number);
PROCEDURE mod_producto(CODIGO NUMBER, cod_prod number, FECHA_MOD DATE);
FUNCTION NUM_lineas (CODIGO number) RETURN NUMBER;

END lineas_FACTURASp;
/



create or replace PACKAGE BODY lineas_facturasp AS

--FUNCIÓN ADICIONAL PARA COMPROBAR SI EXISTE UNA FACTURA, Se trata de una función PRIVADA

  FUNCTION existe (
    codigo NUMBER,
    cod_prod number
  ) RETURN BOOLEAN IS
    cont   NUMBER := 0;
    cod_f lineas_factura.cod_factura%type;
```

```
   BEGIN
 --Comprobar si existe la factura:
      SELECT
          cod_factura into cod_f FROM  lineas_factura
          where cod_factura=codigo and cod_producto=cod_prod;
          RETURN true;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
          return false;
      WHEN OTHERS THEN
           RAISE_APPLICATION_ERROR(-20000,'ERROR'|| sqlcode);
   END;


--PROCEDURE ALTA_FACTURA


   PROCEDURE ALTA_linea (COdigo number,cod_prod NUMBER, unid
number,FEC DATE)
    AS
      devuelto   BOOLEAN;
      erroryaexiste EXCEPTION;


      PRECIO NUMBER;
   BEGIN
      devuelto := existe(codigo,COD_PROD);
      IF not devuelto THEN
          select PVP into PRECIO from productos where cod_producto=cod_prod;
          INSERT INTO LINEAS_factura VALUES
(CODIGO,COD_PROD,PRECIO,UNID,fec );
          commit;
      ELSE
          RAISE erroryaexiste;
      END IF;


   EXCEPTION
      WHEN erroryaexiste THEN
          RAISE_APPLICATION_ERROR(-20001,'ERROR, ESA FACTURA-
PRODUCTO YA EXISTE, NO SE ADMITEN DUPLICADOS');
      WHEN NO_DATA_FOUND THEN
```

```
        RAISE_APPLICATION_ERROR(-20002,'ERROR, EL PRODUCTO NO
EXISTE');
      WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20000,'ERROR'|| sqlcode);
  END;

--PROCEDURE BAJA_FACTURA.

  PROCEDURE baja_LINEA (
    codigo NUMBER,COD_PROD NUMBER
  ) AS
    errornoexiste EXCEPTION;
    devuelto   BOOLEAN;
  BEGIN
    devuelto := existe(codigo,cod_prod);
    IF
      devuelto = true
    THEN
      DELETE FROM lineas_factura WHERE
        cod_factura = codigo AND COD_PRODUCTO=COD_PROD;



        commit;


    ELSE
      RAISE errornoexiste;
    END IF;

  EXCEPTION
    WHEN errornoexiste THEN
      RAISE_APPLICATION_ERROR(-20001,'ERROR, ESE NÚMERO DE
FACTURA -PRODUCTO NO EXISTE');
    WHEN OTHERS THEN
      RAISE_APPLICATION_ERROR(-20000,'ERROR'|| sqlcode);
  END;

--Procedure MOD_FECHA
```

```
PROCEDURE mod_producto (
   codigo NUMBER,
   cod_prod number,
   fecha_mod DATE
) IS
   devuelto   BOOLEAN;
   errornoexiste EXCEPTION;
BEGIN
   devuelto := existe(codigo,cod_prod);
   IF
      devuelto = true
   THEN
      UPDATE lineas_factura
         SET
            fecha = fecha_mod
      WHERE
         cod_factura = codigo and cod_producto=cod_prod;
      commit;
   ELSE
      RAISE errornoexiste;
   END IF;

EXCEPTION
    WHEN errornoexiste THEN
      RAISE_APPLICATION_ERROR(-20001,'ERROR, ESE NÚMERO DE
FACTURA NO EXISTE');
    WHEN OTHERS THEN
      RAISE_APPLICATION_ERROR(-20000,'ERROR'|| sqlcode);
END;

 PROCEDURE mod_producto (
   codigo NUMBER,
   cod_prod number,
   unid number
) IS
   devuelto   BOOLEAN;
   errornoexiste EXCEPTION;
```

```
   BEGIN
      devuelto := existe(codigo,cod_prod);
      IF
         devuelto = true
      THEN
         UPDATE lineas_factura a
            SET
               unidades=unid, pvp=(select pvp from productos where
cod_producto=a.cod_producto)
         WHERE
            cod_factura = codigo and cod_producto=cod_prod;
         commit;
      ELSE
         RAISE errornoexiste;
      END IF;

   EXCEPTION
       WHEN errornoexiste THEN
        RAISE_APPLICATION_ERROR(-20001,'ERROR, ESE NÚMERO DE
FACTURA NO EXISTE');
      WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20000,'ERROR'|| sqlcode);
   END;

--Función NUM_FACTURAS

   FUNCTION num_lineas (
      CODIGO NUMBER
   ) RETURN NUMBER AS
      cont   NUMBER := 0;
   BEGIN
      SELECT COUNT(*) INTO CONT FROM lineas_FACTURA  WHERE
COD_FACTURA=CODIGO;

      RETURN cont;
   END;

--FUNCION TOTAL_FACTURA
```

```
END lineas_facturasp;
```

- Triggers

```
--TRIGGER PARA CONTROLAR LA TABLA FACTURAS
create or replace TRIGGER T_FACTURAS
BEFORE DELETE OR INSERT OR UPDATE ON FACTURAS
DECLARE
OPERACION CHAR(1);
BEGIN
  IF INSERTING THEN
     OPERACION:='I';
  END IF;
IF UPDATING THEN
     OPERACION:='U';
  END IF;
  IF DELETING THEN
     OPERACION:='D';
  END IF;
  INSERT INTO CONTROL_LOG VALUES (USER,SYSDATE,'FACTURAS',OPERACION);
END;

/


--TRIGGER PARA CONTROLAR LA TABLA LINEAS_FACTURA
create or replace TRIGGER T_LINEAS_FACTURAS
BEFORE DELETE OR INSERT OR UPDATE ON LINEAS_FACTURA
DECLARE
OPERACION CHAR(1);
BEGIN
  IF INSERTING THEN
     OPERACION:='I';
  END IF;
IF UPDATING THEN
```

```
    OPERACION:='U';
  END IF;
  IF DELETING THEN
    OPERACION:='D';
  END IF;
  INSERT INTO CONTROL_LOG VALUES (USER,SYSDATE,'LINEAS_FACTURA',OPERACION);
END;
/



--TRIGGER PARA CONTROLAR LA COLUMNA TOTAL_VENDIDOS DE LA TABLA PRODUCTOS
create or replace TRIGGER T_LINEAS_FACTURAS_PRECIO
AFTER DELETE OR INSERT OR UPDATE ON LINEAS_FACTURA
FOR EACH ROW
DECLARE
  --- VARIABLE PARA CONTROLA REL UPDATE
   TOTAL_CAMBIO NUMBER:=0;
BEGIN
  IF INSERTING THEN
    UPDATE PRODUCTOS SET TOTAL_VENDIDOS=TOTAL_VENDIDOS+(:NEW.PVP*:NEW.UNIDADES)
     WHERE COD_PRODUCTO=:NEW.COD_PRODUCTO;
  END IF;
IF UPDATING THEN
   TOTAL_CAMBIO:=(:OLD.PVP*:OLD.UNIDADES)-(:NEW.PVP*:NEW.UNIDADES);
   UPDATE PRODUCTOS SET TOTAL_VENDIDOS=TOTAL_VENDIDOS+TOTAL_CAMBIO
    WHERE COD_PRODUCTO=:NEW.COD_PRODUCTO;
  END IF;
  IF DELETING THEN
   UPDATE PRODUCTOS SET TOTAL_VENDIDOS=TOTAL_VENDIDOS-(:NEW.PVP*:NEW.UNIDADES)
     WHERE COD_PRODUCTO=:NEW.COD_PRODUCTO;
  END IF;
END;
```