El lenguaje de programación PL/SQL

Objetivo

Practicar con el lenguaje de programación procedimental PL/SQL:
- Bloques, procedimientos y funciones.
- Declaración y uso de cursores.
- Uso de instrucciones de recorrido de cursores.
- Uso de instrucciones de control de PL/SQL.
- Uso de instrucciones de salida por pantalla.
- Manejo de excepciones.

Ejercicios

Como resultado de esta práctica se debe subir al CV un documento PDF con las **instrucciones usadas**, **procedimientos y funciones** escritos y **resultados** de las ejecuciones para cada uno de los apartados siguientes.

Si incluyes en el mismo *script* la creación de varios procedimientos y los bloques para probarlos, termina cada uno de ellos con la barra de división "/" (cosas de Oracle...)

Puedes usar Oracle SQL Live o SQL Developer. En este último caso, hay que seleccionar la conexión tania04 – db BDd (red interna) e iniciar sesión con tu usuario dg21_*idemail* (donde *idemail* es tu identificador del correo electrónico, como en ***idemail***@ucm.es). La contraseña es la misma que el usuario. Cámbiala a una nueva con:

ALTER USER *usuario* IDENTIFIED BY *contraseña_nueva*;

1) Ejecuta el script **`crear_tablas.sql`** en la consola de SQL Developer con:

```
@'ruta\crear_tablas.sql'
```

donde ***ruta*** es la carpeta en la que hayas descargado el script.

2) Escribe un bloque anónimo con la declaración del subtipo **`t_sueldo`** (**`BINARY_INTEGER`**, un rango entre 900 y 5.000, y no admite nulos). Define la variable **`v_sueldo`** de este tipo. Inserta en esta variable el sueldo del empleado con DNI 12345678L mediante una instrucción **`SELECT`**...**`INTO`**. Intenta asignar a esta variable su valor multiplicado por 4 y describe lo que ocurre. Cambia esta asignación para que se incremente el sueldo en un 10%. Asigna este valor al sueldo del empleado en la tabla. Finalmente haz un **`ROLLBACK`** para recuperar el valor original (date cuenta que si no hubiese un **`COMMIT`** en el *script* que has descargado, desaparecerían las filas de las tablas, aunque no las tablas en sí).

Solución:

```
@'C:\tmp\crear_tablas.sql'
```

```
declare
    subtype t_sueldo is BINARY_INTEGER range 900 .. 5000 NOT NULL;
    v_sueldo t_sueldo := 1500;
begin
    SELECT Sueldo
    INTO v_sueldo
    FROM Empleados
    WHERE DNI='12345678L';
--    v_sueldo := v_sueldo*4;
    v_sueldo := v_sueldo*1.1;
--    UPDATE Empleados
--    SET Sueldo=v_sueldo
--    WHERE DNI='12345678L';
end;

SELECT * FROM Empleados;

ROLLBACK;
```

3) Crea un procedimiento almacenado denominado **pr_empleados_tlf** que tenga un número de teléfono como parámetro de entrada e imprima por pantalla el nombre y DNI del empleado al que pertenezca. Si no se encuentra el número, o si hay más de un empleado para el mismo teléfono, se debe indicar con un mensaje. Usa un cursor en línea para resolver este apartado y pruébalo en tres ejecuciones distintas con los números '666666666', '611111111' y '913333333'. El resultado sería similar a:

Primera ejecución:
No se encontró el empleado con el teléfono 666666666.

Segunda ejecución:
El empleado con el teléfono 611111111 es: Carlota Cerezo, con DNI: 12345678C.

Tercera ejecución:
Hay más de un empleado con el teléfono 913333333.

Recuerda: en SQL Developer hay que habilitar la salida con:

```
SET SERVEROUTPUT ON SIZE 1000000;
```

Solución:

```
CREATE OR REPLACE PROCEDURE pr_empleados_tlf(p_teléfono Teléfonos.Teléfono%TYPE) AS
  v_DNI Empleados.DNI%TYPE;
  v_Nombre Empleados.Nombre%TYPE;
BEGIN
  SELECT DISTINCT DNI, Nombre
  INTO v_DNI, v_Nombre
  FROM Empleados NATURAL JOIN Teléfonos
  WHERE Teléfono=p_teléfono;
  DBMS_OUTPUT.put_line('El empleado con el teléfono ' || p_teléfono || ' es: ' ||
TRIM(v_Nombre) || ', con DNI: ' || v_DNI || '.');
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.put_line('No se encontró el empleado con el teléfono ' || p_teléfono
|| '.');
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.put_line('Hay más de un empleado con el teléfono ' || p_teléfono ||
'.');
END;
```

El resultado de su ejecución:

```
BEGIN
  pr_empleados_tlf('666666666');
  pr_empleados_tlf('611111111');
  pr_empleados_tlf('913333333');
END;
```

4) Crea un procedimiento almacenado denominado **pr_comprobar_poblaciones** que genere una excepción para la primera población que encuentre a la que le corresponda más de una provincia.

Tanto si se encuentra como si no, se emitirá un mensaje con el resultado. Para probarlo, ejecútalo con los datos que tenga actualmente la tabla. Añade la tupla ('41008','Arganda','Sevilla') a la tabla "Códigos postales" y ejecútalo de nuevo. Finalmente, borra la tupla añadida. El resultado sería similar a:

Primera ejecución:
No hay dos o más provincias que compartan la misma población.

Segunda ejecución:
A la población Arganda no le corresponde siempre la misma provincia.

Solución:

```
CREATE OR REPLACE PROCEDURE pr_comprobar_poblaciones AS
  excepción_pob EXCEPTION;
  v_Población "Códigos postales".población%TYPE;
  CURSOR c_CódigosPostales IS
    SELECT población
    FROM
     (SELECT DISTINCT población, provincia
      FROM "Códigos postales")
    GROUP BY población
    HAVING COUNT(población)>1;
BEGIN
  OPEN c_CódigosPostales;
  LOOP
    FETCH c_CódigosPostales INTO v_Población;
    EXIT WHEN c_CódigosPostales%NOTFOUND;
    RAISE excepción_pob;
  END LOOP;
  CLOSE c_CódigosPostales;
  DBMS_OUTPUT.PUT_LINE('No hay dos o más provincias que compartan la misma
población.');
EXCEPTION
  WHEN excepción_pob THEN
    DBMS_OUTPUT.PUT_LINE('A la población '|| TRIM(v_Población) || ' no le corresponde
siempre la misma provincia.');
    CLOSE c_CódigosPostales;
END;
```

El resultado de su ejecución:

```
BEGIN
  pr_comprobar_poblaciones;
  INSERT INTO "Códigos postales" VALUES ('41008','Arganda','Sevilla');
  pr_comprobar_poblaciones;
  DELETE FROM "Códigos postales" WHERE "Código postal"='41008';
  ROLLBACK;
END;

No hay dos o más provincias que compartan la misma población.
A la población Arganda no le corresponde siempre la misma provincia.
```

5) Crea un procedimiento almacenado denominado **pr_empleados_CP** que liste el nombre, calle y sueldo de los empleados agrupados por código postal (solo deben aparecer códigos postales en los que resida al menos un empleado). Por cada código postal se mostrará el listado de los datos de sus empleados. Al final de cada grupo se indicará el número de empleados y el sueldo medio. Si algún empleado tiene más de un domicilio, se contabilizará tantas veces como domicilios tenga. Al final de todo el listado se indicará el total de empleados. El resultado sería similar a:

Solución:

```
CREATE OR REPLACE PROCEDURE pr_empleados_CP AS

  v_CódigoPostal Domicilios."Código postal"%TYPE;
  v_Empleados INTEGER;
  v_SueldoMedio NUMBER(8,2);
  v_Nombre Empleados.Nombre%TYPE;
```

```
  v_Sueldo Empleados.Sueldo%TYPE;
  v_Calle Domicilios.Calle%TYPE;
  v_TotalEmpleados INTEGER := 0;

  CURSOR c_CPs IS
    SELECT "Código postal",
           COUNT(*) AS Empleados,
           AVG(Sueldo) AS SueldoMedio
    FROM Empleados NATURAL JOIN Domicilios
    GROUP BY "Código postal"
    ORDER BY "Código postal";

  CURSOR c_Empleados(p_CP Domicilios."Código postal"%TYPE) IS
    SELECT Nombre, Calle, Sueldo
    FROM Empleados NATURAL JOIN Domicilios
    WHERE "Código postal"=p_CP;

BEGIN
  OPEN c_CPs;
  LOOP
    FETCH c_CPs INTO v_CódigoPostal, v_Empleados, v_SueldoMedio;
    EXIT WHEN c_CPs%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Código postal: ' || v_CódigoPostal);
    OPEN c_Empleados(v_CódigoPostal);
    LOOP
      FETCH c_Empleados INTO v_Nombre, v_Calle, v_Sueldo;
      EXIT WHEN c_Empleados%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('  ' || TRIM(v_Nombre) || ', ' || TRIM(v_Calle) || ', ' ||
v_Sueldo);
    END LOOP;
    CLOSE c_Empleados;
    DBMS_OUTPUT.PUT_LINE('  Nº empleados: ' || v_Empleados || ', Sueldo medio: ' ||
v_SueldoMedio);
    v_TotalEmpleados := v_TotalEmpleados+v_Empleados;
  END LOOP;
  CLOSE c_CPs;
  DBMS_OUTPUT.PUT_LINE('Total empleados: ' || v_TotalEmpleados);
END;
```

Otra solución con cursores implícitos:

```
CREATE OR REPLACE PROCEDURE pr_empleados_CP
IS
  v_contador INTEGER;
  v_total INTEGER := 0;
  v_suma empleados.Sueldo%TYPE;
  v_SueldoMedio empleados.Sueldo%TYPE;
BEGIN
  FOR c_CP IN (SELECT DISTINCT "Código postal" FROM Domicilios ORDER BY "Código
postal") LOOP
    v_suma := 0;
    v_contador := 0;
    DBMS_OUTPUT.PUT_LINE('Código postal: ' || c_CP."Código postal");
    FOR c_Empleado IN (SELECT Nombre, Calle, Sueldo FROM domicilios NATURAL JOIN
empleados
WHERE "Código postal" = c_CPs."Código postal" ORDER BY Nombre) LOOP
      DBMS_OUTPUT.PUT_LINE('  '||e.nombre || ', ' || c_Empleado.calle || ', ' ||
c_Empleado.sueldo);
      v_suma := v_suma + c_Empleado.sueldo;
      v_contador := v_contador + 1;
    END LOOP;
    v_SueldoMedio:= v_suma / v_contador;
    DBMS_OUTPUT.PUT_LINE('    Nº empleados: ' || v_contador || ', Sueldo medio: ' ||
v_avg);
    v_total := v_total + v_contador;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('Total empleados: ' || v_total);
END;
```

El resultado de su ejecución:

```
   EXECUTE pr_empleados_CP;
```

```
Código postal: 14200
  Laura López, Diamante, 1500
  Pedro Pérez, Diamante, 2000
    Nº empleados: 2, Sueldo medio: 1750
Código postal: 14900
  Pedro Pérez, Carbón, 2000
    Nº empleados: 1, Sueldo medio: 2000
Código postal: 28004
  Antonio Arjona, Cántaro, 5000
    Nº empleados: 1, Sueldo medio: 5000
Código postal: 28040
  Antonio Arjona, Avda. Complutense, 5000
    Nº empleados: 1, Sueldo medio: 5000
Total empleados: 5
```

Crea una función denominada **f_e_x(p_potencia INT)** que devuelva el número *e* (que puedes obtener con la expresión **exp(1)**) elevado a su argumento **p_potencia**. A continuación, redefine esta función cambiando cómo se obtiene el número *e*: en este caso llamando a una función **f_e** (sin argumentos) que a su vez llame a otra función recursiva **f_e_taylor(n,v)** que devuelva las sucesivas aproximaciones al número *e* para el **n** -ésimo término de la serie.

Solución:

```
CREATE OR REPLACE FUNCTION f_e_x(p_potencia INT) RETURN FLOAT IS
BEGIN
    RETURN power(exp(1),p_potencia);
END;

SET SERVEROUTPUT ON SIZE 1000000;

DECLARE
    v_potencia := 2 INT;
BEGIN
    DBMS_OUTPUT.PUT_LINE(' El número e elevado a '||v_potencia||' es: '|| f_e_x(2));
END;

CREATE OR REPLACE FUNCTION f_e_x(p_potencia INT) RETURN FLOAT IS
BEGIN
    RETURN power(f_e,p_potencia);
END;

CREATE OR REPLACE FUNCTION f_e RETURN FLOAT IS
BEGIN
    RETURN f_e_taylor();
END;
```

6) ¡Noticia bomba! Unos jóvenes matemáticos han descubierto una constante (a la que han denominado constante de Buenos Aires) que permite determinar con complejidad $O(n)$ los n primeros números primos. Su valor es el número irracional c_{BA} = 2.920050977316... Para calcular el n-ésimo número primo se debe aplicar la fórmula $f_{n+1} = \lfloor f_n \rfloor (f_n - \lfloor f_n \rfloor + 1)$ con $f_1 = c_{BA}$. El n-ésimo número primo es la parte entera de f_n. Así, ya que $f_1 = c_{BA}$, el primer número primo resulta ser 2. El segundo es la parte entera de $f_2 = \lfloor f_1 \rfloor (f_1 - \lfloor f_1 \rfloor + 1) = 2(2.920050977316...-2+1) = 3{,}840101955...$; es decir, 3, que es el segundo número primo. Y así sucesivamente. Declara la constante como una función **f_c_ba** (sin argumentos) que devuelva el valor de c_{BA}. Crea una función recursiva **f_primo(p_n)** que devuelva el n-ésimo término de la serie. Escribe un bloque anónimo para mostrar los 10 primeros números primos.

Solución:

```
CREATE OR REPLACE FUNCTION f_c_ba RETURN FLOAT IS
BEGIN
    RETURN 2.920050977316;
END;

CREATE OR REPLACE FUNCTION f_primo(p_n INT) RETURN FLOAT IS
    v_primo_n_1 FLOAT;
BEGIN
    IF p_n = 1 THEN
        RETURN f_c_ba;
```

```
    ELSE
        v_primo_n_1 := f_primo(p_n-1);
        RETURN FLOOR(v_primo_n_1)*(v_primo_n_1 - FLOOR(v_primo_n_1) + 1);
    END IF;
END;

SET SERVEROUTPUT ON SIZE 1000000;

BEGIN
  FOR N IN 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE(FLOOR(f_primo(N)));
  END LOOP;
END;
```

7) Crea un procedimento almacenado **pr_jefes(p_DNI)** que escriba en pantalla el nombre y DNI de cada uno de los jefes del empleado con DNI **p_DNI**. Si el empleado no existe se debe generar una excepción e informar al usuario de que ese empleado no existe. Resuélvelo con un procedimiento recursivo. A continuación crea un procedimiento no recursivo **pr_jefes_nr(p_DNI)** que implemente una consulta recursiva y un recorrido por cursor para hacer lo mismo.

Solución:

```
CREATE OR REPLACE PROCEDURE pr_jefes(p_DNI CHAR) AS
  v_dni_jefe  Empleados.Jefe%TYPE;
  v_nombre    Empleados.Nombre%TYPE;
  e_no_existe EXCEPTION;
BEGIN
  SELECT Jefe INTO v_dni_jefe FROM Empleados WHERE DNI=p_DNI;
  IF v_dni_jefe IS NOT NULL THEN
    SELECT Nombre INTO v_nombre FROM Empleados WHERE DNI=v_dni_jefe;
    DBMS_OUTPUT.PUT_LINE('Nombre: '||v_nombre||', DNI: '||v_dni_jefe);
    pr_jefes(v_dni_jefe);
  END IF;
EXCEPTION
  WHEN e_no_existe THEN
    DBMS_OUTPUT.PUT_LINE('No existe ningún empleado con DNI '|| p_DNI);
END;

EXECUTE pr_jefes('12345678C');
```

Resultado de la ejecución:

```
Nombre: Laura López, DNI: 12345678L
Nombre: Pedro Pérez, DNI: 12345678P
Nombre: Antonio Arjona, DNI: 12345678A
```

Procedimiento con consulta recursiva:

```
CREATE OR REPLACE PROCEDURE pr_jefes_nr(p_DNI Empleados.DNI%TYPE) IS
  v_vacío CHAR(1) := 'S';
  e_no_existe EXCEPTION;
  CURSOR c_jefes (p_c_dni empleados.dni%TYPE) IS
    WITH rec_jefe (
        jefe
    ) AS (
        SELECT jefe
        FROM Empleados
        WHERE jefe IS NOT NULL AND DNI=p_DNI
        UNION ALL
        SELECT Empleados.jefe
        FROM rec_jefe JOIN Empleados ON rec_jefe.jefe = Empleados.DNI
        WHERE rec_jefe.jefe IS NOT NULL
    )
    SELECT rec_jefe.jefe dni, Empleados.Nombre nombre
    FROM rec_jefe JOIN Empleados ON rec_jefe.jefe=Empleados.DNI;

BEGIN
  FOR v_dni_nombre IN c_jefes(p_dni) LOOP
    DBMS_OUTPUT.PUT_LINE('Nombre: '||v_dni_nombre.nombre||', DNI: '||
v_dni_nombre.dni);
    v_vacío := 'N';
```

```
  END LOOP;
  IF v_vacío = 'S' THEN RAISE e_vacío; END IF;

EXCEPTION
  WHEN e_no_existe THEN
    DBMS_OUTPUT.PUT_LINE('No existe ningún empleado con DNI '||p_DNI);
END;
```

Resultados:

```
EXECUTE pr_jefes_nr('12345678C');

Nombre: Laura López, DNI: 12345678L
Nombre: Pedro Pérez, DNI: 12345678P
Nombre: Antonio Arjona, DNI: 12345678A

EXECUTE pr_jefes_nr('1245678C');

No existe ningún empleado con DNI 1245678C
```