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.

Introducción

Oracle es un gestor de bases de datos que data de finales de la década de 1970 y ha evolucionado en gran medida. Actualmente se usa en muchas empresas e instituciones (administración pública, banca, operadores de telefonía...). PL/SQL es su lenguaje de programación de cuarta generación, es decir, un lenguaje de propósito general y procedimental que tiene integrados los tipos y sus operaciones asociadas correspondientes a un SGBD concreto; en este caso, a Oracle. Además es posible ejecutar sentencias SQL directamente sin necesidad de usar ningún API. El acceso a los datos de las tablas se hace mediante cursores (un objeto que representa a un conjunto de datos extraídos mediante una instrucción **`SELECT`**) o incluso directamente en variables a partir de una instrucción **`SELECT`** (en este caso se habla de denominados cursores implícitos o en línea). Las instrucciones de PL/SQL se pueden escribir en determinados contextos: bloques, procedimientos, funciones y disparadores.

Ejercicios

Como resultado de esta práctica se debe subir al CV un documento PDF con las instrucciones usadas, procedimientos 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...) Inicia sesión en SQL Developer con tu usuario (el identificador del correo electrónico, como en **usuario**@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).

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: hay que habilitar la salida con:

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

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 y se terminará. 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.
```

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:

```
Código postal: 14200
  Laura López, Diamante, 1500
  Pedro Pérez, Diamante, 2000
    N° empleados: 2, Sueldo medio: 1750
Código postal: 14900
  Pedro Pérez, Carbón, 2000
    N° empleados: 1, Sueldo medio: 2000
Código postal: 28004
  Antonio Arjona, Cántaro, 5000
    N° empleados: 1, Sueldo medio: 5000
Código postal: 28040
  Antonio Arjona, Avda. Complutense, 5000
    N° empleados: 1, Sueldo medio: 5000
Total empleados: 5
```

6) ¡Noticia bomba! Unos 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.

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 (intenta) crear un procedimiento no recursivo **`pr_jefes_nr(p_DNI)`** que implemente una consulta recursiva y un recorrido por cursor para hacer lo mismo (nota: no podrás hacer que funcione porque Oracle no deja hacer casi nada con las consultas recursivas y te devolverá: " ORA-32034: unsupported use of WITH clause". Ni siquiera te deja crear una vista con una vista recursiva... ver para creer).