Práctica 2: El lenguaje de programación PL/SQL

Objetivo

Estudiar el lenguaje de programación procedimental PL/SQL mediante una aplicación que permita listar y comprobar algunas condiciones de consistencia. Para ello se estudiará:
- Declaración de bloques y procedimientos almacenados.
- Declaración y uso de cursores.
- Uso de instrucciones de recorrido de cursores.
- Uso de instrucciones de control de PL/SQL.
- Uso de instrucciones de entrada/salida por pantalla.
- Manejo de excepciones.

Introducción

PL/SQL es un lenguaje de programación *persistente*, esto es, trabaja directamente con *datos* que son persistentes (perviven después de terminada la ejecución del programa), al contrario que los lenguajes tradicionales, que trabajan directamente con *archivos*. El acceso a los datos se hace mediante cursores (un objeto que representa a un conjunto de datos extraídos mediante una instrucción SELECT)

PL/SQL también se puede usar como un lenguaje con SQL incorporado. Es posible ejecutar sentencias SQL directamente en un bloque PL/SQL. Un bloque PL/SQL se define como:

```
DECLARE
  Declaraciones
BEGIN
  Instrucciones PL/SQL
EXCEPTION
  Tratamiento de excepciones
END;
```

(Si se usa la consola SQL*Plus hay que añadir una barra de división al final (/), para indicar el final de la entrada).

Es posible ejecutar sentencias SQL directamente en un bloque PL/SQL y asignar el resultado a una variable de programa, como en:

```
SELECT COUNT(*) FROM Clientes INTO var_número_de_clientes;
```

Un cursor es un objeto que almacena registros (filas de las tablas) que proceden de tablas mediante consultas. El recorrido de los registros se realiza mediante un bucle en cuyo cuerpo se usa una instrucción de lectura de registros (por ejemplo, FETCH). El siguiente es un ejemplo de uso de un cursor dentro de un bloque anónimo:

```
DECLARE
  v_StudentID    students.id%TYPE;
  v_FirstName    students.first_name%TYPE;
  v_LastName     students.last_name%TYPE;
  v_Major        students.major%TYPE := 'Computer Science';
  CURSOR c_Students IS
    SELECT id, first_name, last_name
      FROM students
      WHERE major = v_Major;
BEGIN
  OPEN c_Students;
  LOOP
    FETCH c_Students INTO v_StudentID, v_FirstName, v_LastName;
    EXIT WHEN c_Students%NOTFOUND;
  END LOOP;
  CLOSE c_Students;
END;
```

En este ejemplo los tipos se escogen directamente del esquema de la base de datos (***campo*%TYPE**), aunque también se pueden escribir explícitamente (como **VARCHAR(10)**), en particular los que soporta la base de datos y que se escriben en las sentencias **CREATE TABLE**.

Los procedimientos almacenados son grupos de instrucciones PL/SQL que se pueden llamar por su nombre. Son similares a los procedimientos conocidos de los lenguajes de tercera generación, pero a diferencia de éstos, se almacenan directamente en la base de datos (de ahí que sean persistentes). También admiten parámetros. Asimismo, existen funciones almacenadas.

Un procedimiento almacenado PL/SQL se define como:

```
CREATE OR REPLACE PROCEDURE Nombre [Lista_de_parámetros] AS
 Declaraciones
BEGIN
 Instrucciones PL/SQL
EXCEPTION
 Tratamiento de excepciones
END;
```

Se pueden usar estructuras de control como **IF THEN ELSE**. La parte **EXCEPTION** maneja el control de errores mediante excepciones. Para posibilitar la salida a pantalla se usa el paquete **DBMS_OUTPUT** y los métodos **PUT**, **PUT_LINE** y **NEW_LINE** (antes de poder escribir algo es necesario emitir la instrucción **SET SERVEROUTPUT ON SIZE 1000000**).

Ejemplo:

```
CREATE OR REPLACE PROCEDURE DetectarNulos AS
Declaraciones
BEGIN
 ...
  IF v_cp IS NULL THEN //  Si el código postal es nulo se genera una excepción
    RAISE excepcion_nulo;
  END IF;
 ...
EXCEPTION
  WHEN excepcion_nulo
    DBMS_OUTPUT.PUT_LINE ('Código postal nulo.');
    ...
END;
```

Para crear un procedimiento se puede hacer directamente en SQL Developer, o en la consola SQL o en la Consola de administración Enterprise Manager. Si se usa esta última se escoge la carpeta Esquema y se pulsa con el botón derecho sobre nuestro usuario y en Crear en el menú contextual que aparece. A partir de aquí se puede elegir el objeto a crear (en este caso, un procedimiento). Si se usa el primer método y al crear el procedimiento se obtiene un aviso de errores, se puede introducir SHOW ERRORS para mostrar la información acerca de los errores.

Apartado 0

Ejecuta en SQLDeveloper el siguiente bloque de PL/SQL:

```
SET SERVEROUTPUT ON SIZE 100000;
DECLARE
  V VARCHAR(10);
BEGIN
  V:='Hola';
  DBMS_OUTPUT.PUT_LINE(V);
END;
```

Observar que muestra en la ventana de salida del script:

Apartado 1

En esta práctica se llevará a cabo un ejercicio habitual en la empresa: la migración de datos. El objetivo es asegurar las restricciones de integridad definidas en el diseño de la base de datos que se va a cargar con datos de

una fuente externa. Se asume que estos datos no son fiables y pueden ser inconsistentes. En este caso se parte de datos para las tablas Domicilios y Códigos postales almacenados en ficheros ASCII delimitados por el carácter separador ";" que se han exportado desde alguna otra fuente. Se pide:

1) Generar manualmente los ficheros ASCII con los siguientes datos.

Códigos postales I
08050;Parets;Barcelona;
14200;Peñarroya;Córdoba;
14900;Lucena;Córdoba;
;Arganda;Sevilla;
08050;Zaragoza;Zaragoza;
28040;Arganda;Madrid;
28004;Madrid;Madrid;

Domicilios I
12345678A;Avda. Complutense;28040;
12345678A;Cántaro;28004;
12345678P;Diamante;14200;
12345678P;Carbón;14901;

2) Crear las tablas "Domicilios I" y "Códigos postales I" con el mismo esquema que Domicilios y " Códigos postales" pero sin restricciones de integridad.
3) Importar con Oracle Loader las tablas del punto 1.
4) Escribir un bloque que permita la detección de valores nulos en la tabla "Códigos postales I" y que emita un error por pantalla que identifique el problema (usar para ello la instrucción RAISE).
5) Crear un bloque que permita detectar la violación de clave primaria en la tabla la tabla "Códigos postales I" y que emita por pantalla un error que identifique el problema.
6) Crear un bloque que permita detectar la violación de dependencia funcional en la tabla "Códigos postales I" (a una población siempre le corresponde una misma provincia) y que emita por pantalla un error que identifique el problema.
7) Crear un bloque que permita detectar la violación de integridad referencial en la tabla Domicilios I y que emita un error por pantalla que identifique el problema.

Nota: Para que este procedimiento funcione correctamente es necesario que ningún valor del campo "Código postal" en "Códigos postales I" sea nulo. En caso contrario, la condición "NOT IN" devuelve el valor indefinido y no se encuentran las tuplas erróneas. Para solucionarlo rellenamos el nulo encontrado en el apartado 4.

```
UPDATE "Códigos postales I" SET "Código postal"='14900' WHERE Población='Arganda' AND Provincia='Sevilla';
```

8) **Opcional:** Crear procedimientos almacenados a partir del bloque definido en el paso 4 añadiendo la siguiente funcionalidad:
   a) Procesamiento por lotes de todas las comprobaciones (en lugar de parar en cada error se deben procesar todas las comprobaciones). El tipo de procesamiento se debe poder elegir mediante un parámetro.

Nota:
Para poder probar el procedimiento añadimos dos tuplas con valores nulos a la tabla "Códigos postales I":

```
INSERT INTO "Códigos postales I" VALUES (NULL,'Toledo',NULL);
INSERT INTO "Códigos postales I" VALUES (NULL,'Segovia',NULL);
```

   b) Escritura de los informes de error en una tabla Errores.

Sugerencia:
Se crea la tabla:

   c) Escritura en un archivo de informe en el servidor denominado informe*XX*.txt (donde *XX* son los dígitos del usuario) con el resultado de la comprobación de la restricción de integridad. Para ello se debe usar el paquete UTL_FILE, que permite la escritura de archivos en el cliente. Consúltense las funciones y procedimientos FOPEN (apertura de archivos), FCLOSE (cierre de archivos), PUT (escritura de cadenas de caracteres), PUT_LINE (escritura de cadenas de caracteres y salto de línea), NEW_LINE (escritura de un salto de línea). Consúltese también la conversión explícita de tipos (por ejemplo, la función TO_CHAR). El directorio donde ubicar el archivo de informe es 'UTL' (alojado en el servidor). Crear un procedimiento que lea el archivo de informe y lo muestre por pantalla.

Pistas:
Antes de abordar este apartado y para poder escribir en un directorio del servidor, es necesario que el administrador cree el directorio UTL y conceda permisos a los usuarios:

```
CREATE DIRECTORY UTL AS 'd:\oracle\EMPRESAGIICxx\utl';
```

Ahora los permisos para ambos:

```
GRANT WRITE ON DIRECTORY UTL TO PUBLIC;
```

```
GRANT READ ON DIRECTORY UTL TO PUBLIC;
```

Con los dos siguientes procedimientos, el primero para crear el archivo y el segundo para leerlo, se puede probar que todo ha ido bien. Hay que poner en mayúsculas el nombre del directorio.

Procedimiento de prueba escritura:

```
DECLARE
  v_Archivo UTL_FILE.FILE_TYPE;
BEGIN
  v_Archivo := UTL_FILE.FOPEN('UTL','informe00.txt','w');
  UTL_FILE.PUT_LINE(v_Archivo,'Prueba');
  UTL_FILE.FCLOSE(v_Archivo);
END;
/
```

Procedimiento de prueba de lectura:

```
DECLARE
  v_Archivo UTL_FILE.FILE_TYPE;
  v_Buffer VARCHAR2(300);
BEGIN
  v_Archivo := UTL_FILE.FOPEN('UTL','informe00.txt','r');
  LOOP
    BEGIN
      UTL_FILE.GET_LINE(v_Archivo, v_Buffer);
    EXCEPTION
      WHEN NO_DATA_FOUND THEN EXIT;
    END;
    DBMS_OUTPUT.PUT_LINE(v_Buffer);
  END LOOP;
  UTL_FILE.FCLOSE(v_Archivo);
END;
/
```

Se pueden borrar archivos con:

```
DECLARE
BEGIN
  UTL_FILE.FREMOVE('UTL','informe00.txt');
END;
/
```

9) **Opcional:** Repetir los procedimientos del apartado 8 pero recorriendo los cursores con un bucle FOR