10.1 Introducción

Hasta el momento, hemos trabajado con la base de datos de manera interactiva. Esta forma de trabajar, incluso con un lenguaje tan sencillo y potente como SQL, no resulta operativa en un entorno de producción, ya que supondría que todos los usuarios conocen y manejan SQL, y además está sujeta a frecuentes errores.

También hemos creado pequeños *scripts* de instrucciones SQL y SQL*Plus. Pero estos *scripts* tienen importantes limitaciones en cuanto al control de la secuencia de ejecución de instrucciones, el uso de variables, la modularidad, la gestión de posibles errores, etcétera.

Para superar estas limitaciones, Oracle incorpora un gestor PL/SQL en el servidor de la base de datos. Este lenguaje, basado en el lenguaje ADA, incorpora todas las características propias de los **lenguajes de tercera generación:** manejo de variables, estructura modular (procedimientos y funciones), estructuras de control (bifurcaciones, bucles y demás estructuras), control de excepciones. También incorpora un completo soporte para la **Programación Orientada a Objetos (POO)**, por lo que puede ser considerado como un lenguaje procedimental y orientado a objetos.

Los programas creados con PL/SQL se pueden almacenar en la base de datos como cualquier otro objeto quedando así disponibles para su ejecución por los usuarios. Esta forma de trabajo facilita la instalación, distribución y mantenimiento de los programas y reduce los costes asociados a estas tareas. Además, la ejecución de los programas en el servidor, conlleva un ahorro de recursos en los clientes y disminución del tráfico de red.

El uso del lenguaje PL/SQL es también imprescindible para construir disparadores de bases de datos que permiten implementar reglas complejas de negocio y auditoría en la base de datos.

10.2 Características del lenguaje

Por todo ello, el conocimiento de este lenguaje es imprescindible para poder trabajar en el entorno Oracle, tanto para administradores de la base de datos como para desarrolladores de aplicaciones. En las próximas unidades estudiaremos los fundamentos del lenguaje y su aplicación al servidor de la base de datos.

PL/SQL es un lenguaje procedimental diseñado por Oracle para trabajar con la base de datos. Está incluido en el servidor y en algunas herramientas de cliente. Soporta todos los comandos de consulta y manipulación de datos.

La unidad de trabajo es el **bloque**, constituido por un conjunto de declaraciones, instrucciones y mecanismos de gestión de errores y excepciones.

A. Bloques PL/SQL

El **bloque** es la estructura básica característica de todos los programas PL/SQL. Tiene tres zonas claramente definidas:

- Una **zona de declaraciones** donde se declaran objetos locales (variables, constantes, etcétera). Suele ir precedida por la cláusula DECLARE (o IS/AS en los procedimientos y funciones). Es opcional.
- Un **conjunto de instrucciones** precedido por la cláusula BEGIN.
- Una **zona de tratamiento de excepciones** precedido por la cláusula EXCEPTION. Esta zona, igual que la de declaraciones, es opcional.

El formato genérico del bloque es:

```
[ DECLARE
    <declaraciones> ]
BEGIN
    <órdenes>
[ EXCEPTION
    <gestión de excepciones> ]
END;
```

La zona de declaraciones comenzará con DECLARE o con IS, dependiendo del tipo de bloque. Las únicas cláusulas obligatorias son BEGIN y END.

Caso práctico

1 En el siguiente ejemplo se borra el departamento número 20, pero evitando posibles errores por violar restricciones de integridad referencial, pues el departamento tiene empleados asociados. Para ello crearemos antes un departamento provisional, al que asignamos los empleados del departamento 20 antes de borrar dicho departamento. El programa también informa del número de empleados afectados.

Antes de introducir el ejemplo deberemos introducir la instrucción SET SERVEROUTPUT ON o utilizar el menú de configuración de SQL*Plus para que muestre los mensajes.

```
DECLARE
    v_num_empleados NUMBER(2);
BEGIN
    INSERT INTO depart VALUES (99,'PROVISIONAL',NULL);
    UPDATE emple SET dept_no = 99
      WHERE dept_no = 20;
    v_num_empleados := SQL%ROWCOUNT;
    DELETE FROM depart
      WHERE dept_no = 20;
     DBMS_OUTPUT.PUT_LINE(v_num_empleados ||
         '  Empleados ubicados en PROVISIONAL');
```

(Continúa)

(Continuación)

```
    EXCEPTION
      WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20000, 'Error en aplicación');
    END;
    /
```

El resultado de la ejecución del programa será:

```
5  Empleados ubicados en PROVISIONAL

Procedimiento PL/SQL terminado con éxito.
```

La utilización de bloques supone una notable mejora de rendimiento ya que se envían los bloques completos al servidor para que sean procesados, en lugar de cada sentencia SQL. Así se ahorran muchas operaciones de E/S.

Los bloques PL/SQL se pueden anidar para conseguir distintas funcionalidades (por ejemplo, para evitar la propagación de excepciones) como veremos más adelante. A estos bloques se les llama **bloques anidados.**

```
DECLARE
   ...
BEGIN
   ...
   DECLARE -- comienzo del bloque interior anidado
      ...
   BEGIN
      ...
   EXCEPTION
      ...
   END;    -- fin del bloque interior anidado
   ...
EXCEPTION
   ...
END
```

Se pueden anidar bloques en las secciones ejecutable y de excepciones, pero no en la declarativa.

B. Definición de datos compatibles con SQL

PL/SQL dispone de tipos de datos compatibles con los tipos utilizados para las columnas de las tablas: NUMBER, VARCHAR2, DATE, etcétera, además de otros propios, como BOOLEAN.

Las declaraciones de los datos deben realizarse en la sección de declaraciones:

```
DECLARE
   importe  NUMBER(8,2);
```

```
    contador NUMBER(2);
    nombre   CHAR(20);
    nuevo    VARCHAR2(15);
BEGIN
 ...
```

PL/SQL permite declarar una variable del mismo tipo que otra variable o que una columna de una tabla mediante el atributo %TYPE.

Ejemplo: `nombre_act empleados.nombre%TYPE;`

Declara la variable *nombre_act*, del mismo tipo que la columna nombre de la tabla *empleados*.

También se puede declarar una variable para guardar una fila completa de una tabla mediante el atributo %ROWTYPE.

Ejemplo: `mi_fila empleados%ROWTYPE;`

Declara la variable *mi_fila* del mismo tipo que las filas de la tabla *empleados*.

C. Estructuras de control

Las **estructuras de control** de PL/SQL son las habituales de los lenguajes de programación estructurados: IF, CASE, WHILE, FOR y LOOP.

Estructuras de control alternativas

Alternativa simple

```
IF    <condición>
THEN
   instrucciones;
END IF;
```

Alternativa doble

```
IF <condición> THEN
   instrucciones;
ELSE
   instrucciones;
END IF;
```

Alternativa múltiple (elsif)

```
IF <condición> THEN
   instrucciones;
ELSIF <condición2>
THEN
   instrucciones;
ELSIF <condición3>
THEN
   instrucciones;
...
ELSE
   instrucciones;
END IF;
```

Alternativa múltiple (case)

```
CASE [<expresión>]
WHEN <test1> THEN
   <instrucciones1>;
WHEN <test2> THEN
   <instrucciones2>;
WHEN <test3> THEN
   <instrucciones3>;
[ELSE
   <otrasinstruccio-
nes>;]
END CASE;
```

Estructuras de control repetitivas

Mientras

```
WHILE <condición>
LOOP
   Instrucciones;
END LOOP;
```

Para

```
FOR <variable> IN

<mínimo>..<máximo>
LOOP
   instrucciones;
END LOOP;
```

Iterar... fin iterar salir si...

```
LOOP
   instrucciones;
EXIT WHEN <condi-
ción>;
   instrucciones;
END LOOP;
```

D. Soporte para órdenes de manipulación de datos

Desde PL/SQL se puede ejecutar cualquier orden de manipulación de datos. Ejemplos:

```
DELETE FROM clientes WHERE nif = v_nif;
```

Borra de la tabla *clientes* la fila correspondiente al cliente cuyo NIF se especifica en la variable *v_nif*.

```
UPDATE productos SET stock_disponible := stock_disponi-
ble - unidades_vendidas WHERE producto_no = v_producto;
```

Actualiza en la tabla *productos* la columna *stock_disponible*, correspondiente al código de producto especificado en la variable *v_producto*, restándole el valor que se especifica en la variable *unidades_vendidas*.

```
INSERT INTO clientes VALUES(v_num, v_nom, v_loc, ...);
```

Añade a la tabla *clientes* una fila con los valores contenidos en las variables que se especifican.

E. Soporte para consultas

PL/SQL permite ejecutar cualquier consulta admitida por la base de datos. Pero cuando se ejecuta la consulta, el resultado no se muestra automáticamente en el terminal del usuario, sino que queda en un área de memoria denominada **cursor** a la que accederemos utilizando variables.

Por ejemplo, para obtener en PL/SQL el número total de empleados de la tabla del mismo nombre no podemos utilizar directamente la instrucción SQL correspondiente pues dará error:

```
SELECT COUNT(*) FROM emple;          -- ERROR
```

Utilizaremos una o más variables (declaradas previamente) junto con la cláusula INTO para poder acceder a los datos devueltos por nuestra consulta, tal como figura en el siguiente ejemplo: `SELECT COUNT(*) INTO v_total_emple FROM emple;`

Ejecuta la consulta y deposita el resultado en la variable `v_total_emple`, que suponemos declarada previamente. A este tipo de cursores se les denomina **cursores implícitos**, ya que no hay que declararlos. Es el tipo más sencillo, pero tiene ciertas limitaciones: la consulta deberá devolver una única fila, pues en caso contrario se producirá un error catalogado como TOO_MANY_ROWS.

El formato básico es:

```
SELECT <columna/s> INTO <variable/s> FROM <tabla> WHERE...];
```

La/s variable/s que siguen a INTO reciben el valor de la consulta. Por tanto, debe haber coincidencia en el tipo con las columnas especificadas en la cláusula SELECT.

Caso práctico

2 El siguiente bloque visualiza el apellido y el oficio del empleado cuyo número es 7900.

```
DECLARE
    v_ape VARCHAR2(10);
    v_oficio VARCHAR2(10);
BEGIN
    SELECT apellido, oficio INTO v_ape, v_oficio
      FROM EMPLE WHERE EMP_NO = 7900;
    DBMS_OUTPUT.PUT_LINE(v_ape||'*'||v_oficio);
END;
/
```

El resultado de la ejecución del programa será:

```
JIMENO*EMPLEADO
Procedimiento PL/SQL terminado con éxito.
```

F. Gestión de excepciones

Las **excepciones** sirven para tratar errores y mensajes de aviso. En Oracle están disponibles excepciones predefinidas correspondientes a algunos de los errores más frecuentes que se producen al trabajar con la base de datos, como por ejemplo:

- NO_DATA_FOUND. Una orden de tipo SELECT INTO no ha devuelto ningún valor.
- TOO_MANY_ROWS. Una orden SELECT INTO ha devuelto más de una fila.

Las excepciones se disparan automáticamente al producirse los errores asociados. La sección EXCEPTION es la encargada de gestionar mediante los manejadores (WHEN) las excepciones que puedan producirse durante la ejecución del programa.

Caso práctico

3 El ejemplo anterior, con gestión de excepciones, sería:

```
DECLARE
    v_ape VARCHAR2(10);
    v_oficio VARCHAR2(10);
BEGIN
    SELECT apellido, oficio INTO v_ape, v_oficio
       FROM EMPLE WHERE EMP_NO = 7900;
       DBMS_OUTPUT.PUT_LINE(v_ape||'*'||v_oficio);
```

(Continúa)

(Continuación)

```
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000,'ERROR no hay datos');
        WHEN TOO_MANY_ROWS THEN
            RAISE_APPLICATION_ERROR(-20000,'ERROR demasiados datos');
        WHEN OTHERS THEN
            RAISE_APPLICATION_ERROR(-20000,'Error en la aplicación');
    END;
```

Cuando PL/SQL detecta una excepción, automáticamente pasa el control del programa a la sección EXCEPTION del bloque PL/SQL. Allí buscará un manejador (cláusula WHEN) para la excepción producida, o uno genérico (WHEN OTHERS), y realizará el tratamiento establecido. Al finalizar este tratamiento, sale del bloque actual y devuelve el control al programa o a la herramienta que realizó la llamada.

PL/SQL permite que el programador defina sus propias excepciones.

G. Estructura modular

En esta primera aproximación podemos distinguir los siguientes tipos de programas PL/SQL que se ejecutan el servidor Oracle:

- **Bloques anónimos**
  - No tienen nombre. Se ejecutan en el servidor pero no se guardan en él.
  - La zona de declaraciones comienza con la palabra DECLARE.
  - Son las estructuras utilizadas fundamentalmente en las primeras unidades de este libro por razones didácticas pero su utilización real es escasa.
  - Todos los ejemplos de programas vistos hasta el momento en esta unidad son bloques anónimos.
- **Subprogramas: procedimientos y funciones**
  - Son bloques PL/SQL que tienen un nombre por el que son invocados desde otros programas o herramientas.
  - Se compilan, almacenan y ejecutan en la base de datos Oracle.
  - Tienen una cabecera que incluye el nombre del subprograma (indicando si se trata de un procedimiento o una función), los parámetros y el tipo de valor de retorno en el caso de las funciones.
  - La zona de declaraciones y el bloque o cuerpo del programa comienza con la palabra IS o AS.

- Pueden ser de dos tipos: **PROCEDIMIENTOS** y **FUNCIONES.** Su formato genérico es prácticamente el mismo pero las funciones devuelven un valor en algún punto de su código y deben declarar el tipo de valor devuelto en la cabecera:

Procedimiento	Función
PROCEDURE <nombreprocedimiento> [(<lista de parámetros>)] IS [<declarac objetos locales>;] BEGIN <instrucciones>; [EXCEPTION <excepciones>;] END [<nombreprocedimiento>];	FUNCTION <nombrefunción> [(<lista de parámetros>)] RETURN <tipo valor devuelto > IS [<declaración objetos locales>;] BEGIN <instrucciones>; RETURN <expresión>; [EXCEPTION <excepciones>;] END [<nombrefuncion>];

Al final de esta unidad podemos ver ejemplos de procedimientos y funciones.

- **Disparadores de base de datos.**
  - Son programas almacenados en la base de datos que se asocian a un evento.
  - Se ejecutan automáticamente al producirse determinados cambios (inserción, borrado o modificación) en la tabla a la que están asociados o en el sistema.
  - Son muy útiles para controlar los cambios que suceden en la base de datos, para implementar restricciones complejas, etcétera.

Hay un tipo de disparadores o *triggers* llamados **de sustitución** que se ejecutan en lugar de ciertas instrucciones de consulta sobre vistas.

Caso práctico

4 El siguiente código crea un *trigger* que se ejecutará automáticamente cuando se elimine algún empleado en la tabla correspondiente visualizando el número y el nombre de los empleados borrados:

```
CREATE OR REPLACE TRIGGER audit_borrado_emple
          BEFORE DELETE
          ON emple
       FOR EACH ROW
BEGIN
          DBMS_OUTPUT.PUT_LINE('BORRADO EMPLEADO'
                    || '*' ||:old.emp_no
                    ||'*'||:old.apellido);
END;
```

Estudiaremos con más detalle los disparadores o *triggers* de bases de datos en la unidad de programación avanzada en PL/SQL.

10.3 Interacción con el usuario en PL/SQL

PL/SQL es un lenguaje diseñado para trabajar con la base de datos y manejar grandes volúmenes de información de manera eficaz, pero no ha sido concebido para interactuar con el usuario. En efecto, PL/SQL no dispone de órdenes para la captura de datos introducidos por teclado, ni tampoco para visualizar datos en la pantalla. Para eso se utilizan otros lenguajes y herramientas. No obstante, Oracle incorpora el paquete DBMS_OUTPUT con fines de depuración. Éste incluye, entre otros, el procedimiento **PUT_LINE,** que permite visualizar textos en la pantalla.

Para aprender a programar necesitaremos probar frecuentemente los programas y visualizar sus resultados. Con este fin utilizaremos el procedimiento PUT_LINE, sabiendo que en un entorno de producción deberemos emplear otras herramientas especializadas para visualizar los resultados. El formato genérico para invocar a este procedimiento es el siguiente:

```
DBMS_OUTPUT.PUT_LINE(<expresión>);
```

Para que funcione correctamente, la variable de entorno **SERVEROUTPUT** deberá estar en **ON;** en caso contrario los programas no darán ningún error, pero no se visualizará nada.

Para cambiar el estado de la variable introduciremos al comienzo de la sesión:

```
SQL> SET SERVEROUTPUT ON
```

Para pasar datos a un programa podemos recurrir a una de las siguientes opciones:

- Introducir datos en una tabla desde SQL*Plus y, después, leerlos desde el programa.
- Pasar los datos como parámetros en la llamada (en procedimientos y funciones).
- Utilizar variables de sustitución SQL*Plus. Esta opción sólo puede utilizarse con bloques anónimos tal como se muestra en el ejemplo del Caso práctico 5.

10.4 Arquitectura

PL/SQL es una **tecnología integrada** en el servidor Oracle y también en algunas herramientas. Se trata de un motor o gestor que es capaz de ejecutar subprogramas y bloques PL/SQL, y que trabaja en coordinación con el ejecutor de órdenes SQL.

Por defecto, PL/SQL trabaja en modo **interpretado,** es decir, el código almacenado debe ser traducido a código ejecutable cada vez que ejecutamos un procedimiento. A partir de la versión 9iR1 se incluye la **posibilidad de compilación nativa:** los programas se traducen a C en tiempo de compilación y se almacena el código ejecutable. De esta forma los programas se ejecutan hasta 30 veces más rápido.

Con independencia del modo de trabajo, todos los programas PL/SQL son examinados por los analizadores lexicográficos y sintácticos antes de almacenarlos en el servidor (incluso cuando trabajamos en modo interpretado). En este proceso se comprueban también todas las referencias a objetos de la base de datos y se informa de los errores o problemas detectados.

La Figura 10.1 muestra el procesamiento de un bloque PL/SQL. Podemos apreciar que el motor PL/SQL envía las órdenes SQL que contiene el bloque al ejecutor de órdenes SQL, que se encargará de procesarlas.

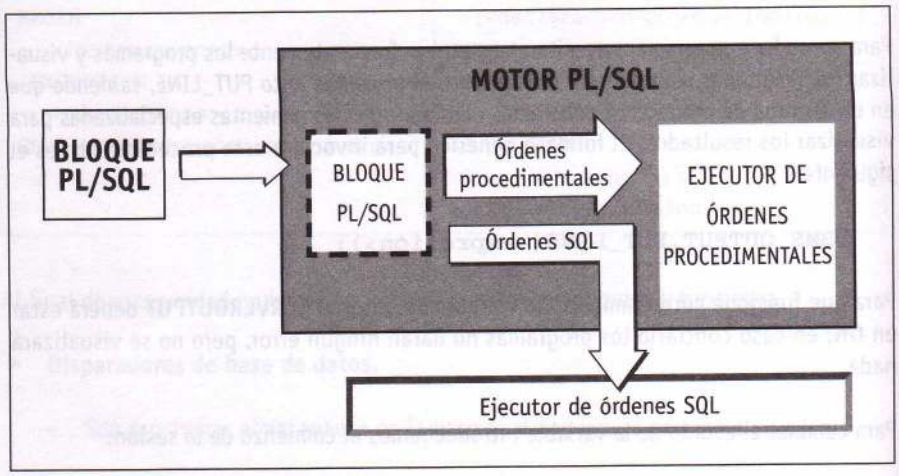

Fig. 10.1 *Procesamiento de un bloque PL/SQL.*

Algunas herramientas de Oracle (Forms, Reports, etcétera) contienen un motor PL/SQL capaz de procesar bloques, PL/SQL ejecutando las órdenes procedimentales en el cliente o en el lugar donde se encuentre la herramienta, enviando al servidor Oracle solamente las instrucciones SQL. En estas cuatro unidades dedicadas a PL/SQL nos ceñiremos exclusivamente al desarrollo de programas PL/SQL para el servidor Oracle.

10.5 Uso y ejemplos de distintos tipos de programas

Para crear nuestros programas podemos utilizar alguna de las herramientas específicas de desarrollo de Oracle, como JDeveloper, que incluyen facilidades gráficas de edición, depuración y compilación. Nosotros hemos optado por utilizar el **entorno SQL*Plus**, pues está disponible en todas las instalaciones Oracle y permite una plena utilización de todas las características del lenguaje.

A. Bloques anónimos

Se pueden generar con diversas herramientas, como SQL*Plus, y se envían al servidor Oracle, donde serán compilados y ejecutados.

SQL*Plus reconoce el comienzo de un bloque anónimo cuando encuentra la palabra reservada DECLARE o BEGIN. Cuando esto ocurre: limpia el buffer SQL, ignora los «;» y entra en modo INPUT.

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

HOLA MUNDO
Procedimiento PL/SQL terminado con éxito.

El ejemplo anterior muestra un bloque anónimo de código que escribe el texto 'HOLA MUNDO'. Para introducirlo se escribe la palabra reservada BEGIN desde el prompt de SQL*Plus seguida de un retorno de carro. Los números de línea se irán mostrando automáticamente. El símbolo «/» provoca que se guarde el bloque en el buffer y lo envía al servidor para su ejecución.

Las secciones declarativa y de excepciones son opcionales.

El bloque anterior carece de las secciones declarativas y de gestión de excepciones, pues la única sección obligatoria es la ejecutable. Ésta debe contener, al menos, **un comando ejecutable,** incluso aunque no haga nada tal como aparece en el siguiente ejemplo:

```
SQL> BEGIN
  2     NULL;
  3  END;
  4  /
Procedimiento PL/SQL terminado con éxito.
```

También podemos crear el programa y guardarlo en el buffer SQL pero sin ejecutarlo automáticamente. Para ello se utiliza un punto después de la última línea de código.

El siguiente programa muestra el precio de un producto especificado:

```
SQL> DECLARE
  2    v_precio NUMBER;
  3  BEGIN
  4    SELECT precio_actual INTO v_precio
  5      FROM productos
  6      WHERE PRODUCTO_NO = 70;
  7    DBMS_OUTPUT.PUT_LINE(v_precio);
  8  END;
  9  .
SQL>_
```

Ahora el programa se encuentra cargado en el buffer SQL dispuesto para ser ejecutado utilizando la barra oblicua (/) o el comando **RUN.**

```
SQL> /
450
Procedimiento PL/SQL terminado con éxito.
```

Podemos **guardar** el bloque del buffer en un fichero con la orden **SAVE** según el siguiente formato:

```
SQL > SAVE nombrefichero [REPLACE]
```

La opción REPLACE se usará si el fichero ya existía.

Para **cargar** un bloque de un fichero en el buffer SQL se hará mediante el comando **GET:**

```
SQL> GET nombrefichero
```

Una vez cargado se puede ejecutar tal como acabamos de ver con RUN o con «/».

También se puede **cargar** y **ejecutar** con una sola orden mediante el comando **START:**

```
SQL> START nombrefichero
```

El comando START puede ser sustituido por el símbolo @ con el mismo formato y resultado:

```
SQL> @nombrefichero
```

En PL/SQL podemos insertar comentarios de línea con "--". Todo lo que sigue a estos caracteres dentro de la línea será considerado comentario.

Caso práctico

5 El siguiente programa solicitará la introducción de un número de cliente y visualizará el nombre del cliente correspondiente con el número introducido. Para introducir el número de cliente recurriremos a las variables de sustitución de SQL*Plus.

```
SQL> DECLARE
  2       v_nom CLIENTES.NOMBRE%TYPE; --(ejemplo uso %TYPE)
  3     BEGIN
  4       SELECT nombre INTO v_nom
  5         FROM clientes
  6         WHERE CLIENTE_NO=&vn_cli;
  7       DBMS_OUTPUT.PUT_LINE(v_nom);
  8     END;
  9  .
SQL>
```

Para ejecutar el programa utilizaremos RUN o /. Solicitará un valor para la variable de sustitución, y una vez introducido el valor sustituirá a la variable y se enviará el bloque al servidor para su ejecución.

```
SQL> /

Introduzca valor para vn_cli: 102
antiguo   6:        WHERE CLIENTE_NO=&vn_cli;
nuevo     6:        WHERE CLIENTE_NO=102;
LOGITRONICA S.L

   Procedimiento PL/SQL terminado con éxito.
```

Actividades propuestas

1 **Desde el entorno SQL*Plus introduce el bloque del ejemplo anterior, depura los posibles errores y realiza diversas pruebas de ejecución.**

Las variables de sustitución solamente son operativas en los bloques anónimos preferentemente al comienzo y siempre fuera de estructuras condicionales o repetitivas. No podemos usarlas en los procedimientos ya que SQL*Plus realizará la sustitución antes de enviar el bloque al servidor. De esta forma, cuando se compila y almacena el procedimiento se hace con un valor que será siempre el mismo para futuras ejecuciones.

Las variables de sustitución no se pueden usar en procedimientos o funciones.

B. Uso de subprogramas almacenados: procedimientos y funciones

Los **procedimientos** y **funciones** se almacenan en la base de datos, donde quedarán disponibles para ser ejecutados, por ello reciben el nombre de subprogramas almacenados.

Para su creación nos remitimos a lo expuesto en el apartado anterior para los bloques anónimos teniendo en cuenta que, a diferencia de estos, los procedimientos y funciones, una vez compilados, se almacenan en la base de datos y quedan disponibles para su ejecución mediante los comandos apropiados sin necesidad de volver a cargarlos.

Caso práctico

6 **Introduciendo estas líneas desde el indicador de SQL*Plus dispondremos de un procedimiento PL/SQL sencillo para consultar los datos de un cliente:**

```
SQL> CREATE OR REPLACE
  2  PROCEDURE ver_depart (numdepart NUMBER)
  3  AS
  4     v_dnombre VARCHAR2(14);
  5     v_localidad VARCHAR2(14);
  6  BEGIN
  7     SELECT dnombre, loc INTO v_dnombre, v_localidad
  8       FROM depart
  9       WHERE dept_no = numdepart;
 10     DBMS_OUTPUT.PUT_LINE('Num depart:'||numdepart|| ' * Nombre dep:'|| v_dnombre ||
 11                          ' * Localidad:'||v_localidad);
 12  EXCEPTION
 13     WHEN NO_DATA_FOUND THEN
 14     DBMS_OUTPUT.PUT_LINE('No encontrado departamento ');
 15  END ver_depart;
 16 /

Procedimiento creado.
```

En el ejemplo anterior podemos observar:

- La **cabecera del procedimiento** (línea 2) indica el nombre del mismo y el parámetro que se utilizará para pasarle valores.
- La palabra **AS** especifica el comienzo del cuerpo del programa y del bloque, comenzando con la zona de declaraciones donde se indicarán los objetos que intervendrán en el programa: variables, constantes, etcétera.
- La palabra **BEGIN** indica el comienzo de la zona de instrucciones.
- La instrucción **SELECT** deposita el resultado de la consulta en las variables *v_dnombre, v_localidad* que siguen a INTO tal como comentamos al hablar del soporte para consultas del lenguaje. La cláusula WHERE contiene el parámetro con el que se llamó al programa, que será el número de departamento cuyos datos se requieren.
- El procedimiento DBMS_OUTPUT.PUT_LINE visualiza el contenido de las variables y los literales. Para que funcione correctamente la variable SERVEROUTPUT debe estar en ON antes de ejecutar el programa.
- La palabra EXCEPTION indica el comienzo de la zona de tratamiento de excepciones. Esta zona sólo se ejecutará si se produzce alguna excepción. Por ejemplo, si al ejecutarse la cláusula SELECT no se encuentra ninguna fila que cumpla la condición, se levantará la excepción NO_DATA_FOUND, se bifurcará el control del programa a esta sección.
- El manejador WHEN NO_DATA_FOUND «caza» la excepción, en el caso de que se produzca, y ejecuta las instrucciones que siguen a la cláusula THEN dando por finalizado el programa.
- El símbolo «/» indica a SQL*Plus el final del procedimiento. Es una abreviatura del comando RUN. Compila y guarda en la base de datos el programa introducido.

  Si el compilador detecta errores veremos el siguiente mensaje:

```
Aviso: Procedimiento creado con errores de compilación.
```

La orden **SHOW ERRORS** permite ver los errores detectados. Invocaremos al editor y subsanaremos el error. Hecho esto el programa se compilará y almacenará de nuevo, introduciendo «/» seguido de INTRO.

Ahora, el procedimiento se encuentra almacenado en el servidor de la base de datos y puede ser invocado desde cualquier estación por cualquier usuario autorizado y con cualquier herramienta; por ejemplo, desde SQL*Plus mediante la orden EXECUTE:

```
SQL> SET SERVEROUTPUT ON
SQL>  EXECUTE ver_depart(20)
Num depart:20 * Nombre dep: INVESTIGACION * Localidad:
MADRID
```

También podemos invocar al procedimiento desde un bloque PL/SQL, por ejemplo de esta forma:

```
SQL> BEGIN ver_depart(30); END;
  2  /
Num depart:30 * Nombre dep:VENTAS * Localidad: BARCELONA
```

En realidad, la orden EXECUTE es una utilidad de SQL*Plus que crea un bloque PL/SQL anónimo añadiendo un BEGIN y un END al principio y al final, respectivamente, de la llamada al procedimiento.

Caso práctico

7 Ejemplos de aplicación:

1. En el siguiente procedimiento se visualiza el precio de un producto cuyo número se pasa como parámetro.

```
SQL> CREATE OR REPLACE
  2      PROCEDURE ver_precio(v_num_producto NUMBER)
  3  AS
  4      v_precio NUMBER;
  5  BEGIN
  6      SELECT precio_actual INTO v_precio
  7        FROM productos
  8        WHERE producto_no = v_num_producto;
  9      DBMS_OUTPUT.PUT_LINE('Precio = '||v_precio);
 10  END;
 11  /

Procedimiento creado.

SQL> EXECUTE VER_PRECIO(50);
   Precio = 1050

  Procedimiento PL/SQL terminado con éxito.
```

2. Escribiremos un procedimiento que modifique el precio de un producto pasándole el número del producto y el nuevo precio. El procedimiento comprobará que la variación de precio no supere el 20 por 100:

```
SQL> CREATE OR REPLACE
  2  PROCEDURE modificar_precio_producto
  3     (numproducto NUMBER, nuevoprecio NUMBER)
  4  AS
  5     v_precioant NUMBER(5);
  6  BEGIN
  7     SELECT precio_actual INTO v_precioant
  8        FROM productos
  9          WHERE producto_no = numproducto;
```

(Continúa)

(Continuación)

```
10    IF (v_precioant * 0.20) > (nuevoprecio - v_precioant) THEN
11       UPDATE productos SET precio_actual = nuevoprecio
12         WHERE producto_no = numproducto;
13    ELSE
14         DBMS_OUTPUT.PUT_LINE('Error, modificación supera 20%');
15    END IF;
16
17 EXCEPTION
18    WHEN NO_DATA_FOUND THEN
19    DBMS_OUTPUT.PUT_LINE('No encontrado producto '|| numproducto);
20 END modificar_precio_producto;
21 /
```

Ejemplos de ejecución:

```
SQL> SET SERVEROUTPUT ON

SQL>  EXECUTE MODIFICAR_PRECIO_PRODUCTO(60,300)
Procedimiento PL/SQL terminado con éxito.

SQL>  SELECT PRECIO_UNI FROM PRODUCTOS WHERE COD_PRODUCTO=60;
PRECIO_UNI
----------
      300

SQL> EXECUTE MODIFICAR_PRECIO_PRODUCTO(60,10000)
Error, modificación supera 20%
Procedimiento PL/SQL terminado con éxito.

SQL>  SELECT PRECIO_UNI FROM PRODUCTOS WHERE COD_PRODUCTO=3;
PRECIO_UNI
----------
       300
```

Observamos que en el segundo caso no se ha producido la modificación deseada. Aun así, el procedimiento ha terminado con éxito, ya que no se ha producido ningún error no tratado.

3. Escribiremos una función que devuelva el valor con IVA de una cantidad que se pasará como primer parámetro. La función también podrá recoger un segundo parámetro opcional, que será el tipo de IVA siendo el valor por defecto 16.

```
SQL> CREATE OR REPLACE FUNCTION con_iva (
  2      cantidad NUMBER,
  3      tipo NUMBER DEFAULT 16)
  4
  5      RETURN NUMBER
  6  AS
  7      v_resultado NUMBER (10,2) DEFAULT 0;
  8  BEGIN
```

(Continú

(Continuación)

```
 9      v_resultado := cantidad * (1 + (tipo / 100));
10      RETURN(v_resultado);
11  END con_iva;
12  /

Función creada.
```

Ahora podemos usar la función creada para realizar cálculos dentro de un bloque o programa PL/SQL:

```
SQL>  BEGIN  DBMS_OUTPUT.PUT_LINE(con_iva(200)); END;
  2  /
232
```

Debemos recordar que hay que HACER ALGO con el valor devuelto por la función: visualizarlo, usarlo como parte de una expresión, etcétera. También podemos usarla en instrucciones SQL:

```
SQL> SELECT producto_no, precio_actual, con_iva(precio_actual) FROM productos;

PRODUCTO_NO    PRECIO_ACTUAL    CON_IVA(PRECIO_ACTUAL)
-----------    -------------    ----------------------
         10              550                       638
         20              670                     777,2
         30              460                     533,6
         40              340                     394,4
         50             1050                      1218
         60              280                     324,8
         70              450                       522
         80              550                       638
```

En los ejemplos anteriores se muestra el código tal como aparece en el entorno SQL*Plus, con los números de línea y el indicador SQL.

En lo sucesivo prescindiremos de esos elementos para centrarnos en el código.

Conceptos básicos

- PL/SQL es un lenguaje de programación que soporta:
  - Instrucciones SQL de consulta y manipulación de datos.
  - Estructuras de control y otros elementos de lenguajes de tercera generación.
  - Programación Orientada a Objetos.
- El bloque es la unidad básica de todos los programas PL/SQL. Su formato básico incluye tres secciones:
  - [**DECLARE** - Sección declarativa: se declaran las variables y otros objetos
    <declaraciones>]
  - **BEGIN** -- Sección ejecutable: se incluyen las instrucciones del programa.
    <órdenes>
  - [**EXCEPTION** -- Sección de excepciones: se tratan las posibles excepciones
    <gestión de excepciones>]
  - **END;**
- En PL/SQL podemos distinguir los siguientes tipos de programas:
  - **Bloques anónimos.** Sin nombre. Se ejecutan en el servidor pero no se guardan.
  - **Subprogramas:** *procedimientos* y *funciones*. Se guardan y ejecutan en el servidor.
  - **Disparadores de base de datos.** Se ejecutan al producirse un evento en la BD.
- Para crear y ejecutar bloques anónimos usaremos:
  - Para crear un bloque desde SQL*Plus simplemente introduciremos las palabras reservadas DECLARE o BEGIN y, a continuación, el resto del bloque.
  - Para guardar un bloque en un fichero utilizaremos el comando SAVE *nombredefichero*.
  - Para cargar un bloque guardado en un fichero usaremos GET *nombredefichero*.
  - Para ejecutar un bloque cargado en memoria usaremos RUN o «/».
  - Para cargar y ejecutar un fichero que contiene un bloque usaremos START *nobrefichero* o @*nombrefichero*.
  - Para visualizar los errores que puedan producirse durante la compilación usaremos SHOW ERRORS.
- Para visualizar datos desde un programa PL/SQL usaremos DBMS_OUTPUT.PUT_LINE. Al ejecutar el programa debemos asegurarnos de que la variable del sistema SEVEROUTPUT está en ON (SET SERVEROUTPUT ON).

Actividades complementarias

1 **Construye un bloque PL/SQL que escriba el texto 'Hola'.**

2 **¿Qué hace el siguiente bloque PL/SQL?**

```
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 /
```

3 **Introduce el bloque anterior desde SQL*Plus y guardarlo en un fichero.**

4 **Ejecuta la orden SELECT especificada en el bloque anterior desde SQL*Plus sin la cláusula INTO.**

5 **Carga y ejecuta el bloque de nuevo, y comprueba que el resultado aparece en pantalla.**

6 **Escribe desde SQL*Plus el ejemplo número 1 del epígrafe «Uso de subprogramas almacenados» y prueba a ejecutarlo con distintos valores.**

7 **Identifica en el ejemplo número 2 del epígrafe «Uso de subprogramas almacenados» los siguientes elementos:**

- La cabecera del procedimiento.
- El nombre del procedimiento.
- Los parámetros del procedimiento.
- Las variables locales.
- El comienzo y el final del bloque PL/SQL.
- El comienzo y el final de las secciones declarativa, ejecutable y de gestión de excepciones.
- ¿Qué hace la cláusula INTO?
- ¿Qué hace WHEN NO_DATA_FOUND?
- ¿Por qué no tiene la cláusula DECLARE? ¿Qué tiene en su lugar?