# PL/SQL

"Los límites de mi lenguaje son los límites de mi mente".

**(Ludwig Wittgenstein)**

## LENGUAJE DECLARATIVO vs LENGUAJE PROCEDURAL

Aunque PL/SQL y SQL están estrechamente integrados entre sí, existen varias diferencias en la forma en que funcionan. SQL es un lenguaje declarativo orientado a datos y por lo tanto no es procedimental. Es decir, el programador define qué se quiere obtener y no cómo (no especifica los pasos exactos para resolverlo). EL motor de base de datos es quien define exactamente como conseguir esos resultados. Por lo tanto, en este tipo de programación no se va a ver las estructuras que se usan en la programación secuencial, como bucles o ciclos condicionales. PL/SQL agrega procedimientos a SQL para crear aplicaciones que interactúan con las bases de datos de Oracle.


## ¿QUE ES PL/SQL?

PL/SQL (Procedural Language/Structured Query Language) es un lenguaje de programación desarrollado por Oracle en 1991 como extensión de SQL. Ambos son lenguajes de bases de datos relacionales, pero entre ellos existen varias diferencias. SQL es un lenguaje de consulta estructurado con el que podemos tratar con la información de una base de datos utilizando sentencias UPDATE, INSERT, SELECT, DELETE. Es un lenguaje declarativo. PL/SQL va más allá, es un lenguaje de programación por procedimientos, una ampliación de SQL que conserva la sintaxis SQL, pero expandiéndola. Incluye un conjunto de sentencias como variables, estructuras de control y de toma de decisiones usados por la mayoría de los lenguajes de alto nivel como Java, PHP o Python. Gracias a PL/SQL podremos escribir programas que almacena lo que se denomina la Lógica de Negocio (business Logic).

Un programa típico de PL/SQL podría validar por ejemplo el ingreso de un nuevo usuario antes de agregarlo a la base de datos de Oracle. Todo se ejecutará en el mismo servidor, lo que en teoría proporcionará una mejora en el rendimiento.

## VENTAJAS DE PL/SQL

Echemos un vistazo a las ventajas de usar PL/SQL:

- **Alto rendimiento:** PL/SQL admite el procesamiento masivo, lo que significa que permite el procesamiento eficiente de grandes cantidades de datos. Además, se almacena en forma compilada en la base de datos, lo que lo hace más rápido de ejecutar que los lenguajes interpretados como SQL.
- **Portabilidad:** PL/SQL es una buena opción para aplicaciones multiplataforma porque puede transferirse fácilmente entre varias plataformas y sistemas operativos.
- **Integración fluida con SQL:** PL/SQL es una extensión de SQL; por lo tanto, se integra a la perfección con SQL para permitir una fácil manipulación de datos.
- **Programación modular:** Mediante el uso de paquetes, soporta la programación modular. Por lo tanto, permite una mejor organización y mantenimiento del código.
- **Manejo de Excepciones:** Debido a su manejo de excepciones incorporado, permite la detección y manejo de errores de forma controlada.
- **Seguridad:** Para controlar el acceso a la base de datos y garantizar la integridad de los datos, PL/SQL incluye funciones de seguridad como privilegios y roles.

## ESTRUCTURA BÁSICA DE UN PROGRAMA PL/SQL

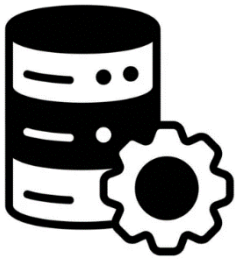
Una pieza de código PL/SQL a menudo se denomina programa. Un programa PL/SQL está estructurado en bloques. Un programa PL/SQL incluye varios bloques:

- Sección declarativa: aquí es donde se declaran las variables.
- Sección ejecutable: este es el código que se ejecuta como parte del programa.
- Sección de excepción: esto define lo que sucede si algo sale mal.

La única parte requerida de un programa PL/SQL es la sección ejecutable. Las otras dos secciones (declarativa y de excepción) son opcionales. La parte ejecutable de un programa PL/SQL comienza con la palabra clave BEGIN y termina con la palabra clave END.

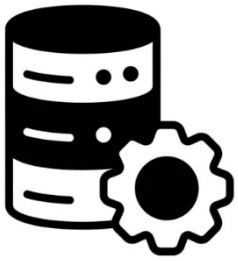
```
BEGIN
-- Acá ira el código de ejecución.
END;
```

La palabra clave BEGIN inicia la sección ejecutable de su programa. Se ejecuta todo lo que sigue a la sentencia BEGIN, hasta que se llega a la sentencia END.



**Nota**

*La palabra clave END termina con un punto y coma, pero la palabra clave BEGIN no necesita un punto y coma.*



**Nota**

*A diferencia de otros lenguajes de bases de datos, PL/SQL no distingue entre letras mayúsculas o minúsculas.*

## PRIMER PROGRAMA EN PL/SQL

Nuestro primer programa será construir una aplicación que muestre el clásico: Hola Mundo! (Hello World!). Para ello en el bloque de ejecución BEGIN-END usaremos la función PUT_LINE. Para llamar a la función PUT_LINE, necesitamos especificar dos cosas:

- El paquete en el que está contenida la función
- El texto que queremos mostrar.

La función PUT_LINE está contenida en un paquete llamado DBMS_OUTPUT. Un paquete es como una biblioteca o librería en otros lenguajes de programación. Es decir, contiene un conjunto de programas PL/SQL ya armados. El texto que queremos mostrar es "Hello World". Esto se especifica como un parámetro que está dentro de los paréntesis que aparecen después de la función PUT_LINE.

```
BEGIN
    dbms_output.put_line('Hello World');
END;
```

La función PUT_LINE incluye el texto 'Hello World' entre comillas simples, ya que es el estándar en SQL para trabajar con cadenas o valores de texto (VARCHAR2). También terminamos la línea con un punto y coma, para que la base de datos sepa que hemos llegado al final de la línea. Si escribe el código en SQL Developer y lo ejecuta como Script vera el siguiente mensaje:

```
Procedimiento PL/SQL terminado correctamente.
```

Para poder ver la salida, o sea el mensaje deberá activar desde el menú VER la opción Salida de DBMS. Esto agregara un nuevo panel denominado Salida de DBMS. En el podrá ver el botón Activar DBMS_OUTPUT para conexión, en el momento que realice un click sobre el mismo le solicitará la conexión de la Base de Datos. Seleccione la conexión y presione nuevamente el botón Ejecutar Script (F5) y vera la salida del programa:

```
Hello World
```

## LA SECCIÓN DECLARATIVA EN PL/SQL

La sección declarativa se define con la palabra clave DECLARE y se encuentra antes de la palabra clave BEGIN. ¿Para qué sirve la sección DECLARE? Lo usamos para declarar variables en PL/SQL. Una variable tiene un nombre, un tipo de datos y un valor. Por ejemplo:

```
DECLARE
v_text VARCHAR2(50)
```

En este caso se declaró la variable v_text (La v minúscula se usa como prefijo para indicar que es una variable no es un requisito agregar este prefijo, pero muchos desarrolladores lo recomiendan) a dicha variable se la definió como de tipo varchar2 de longitud 50. Ahora le asignaremos un valor agregando los dos puntos y el signo igual y el valor que tendrá al comienzo de la ejecución del programa:

```
DECLARE
    v_text VARCHAR2(50) := 'I am learning sql';
BEGIN
    dbms_output.put_line(v_text);
END;
```

Si ejecutamos este programa obtendremos la nueva salida:

```
I am learning sql
```

Fíjese que aplicamos una indentación al código para hacerlo más legible. La asignación también podría haberse realizado en el bloque de ejecución de esta forma:

```
DECLARE
  v_text VARCHAR2(50);
BEGIN
  v_text := 'I am learning sql';
  dbms_output.put_line(v_text);
END;
```

Ahora usaremos el operador de concatenación para hacer una salida más amplia:

```
DECLARE
  v_text VARCHAR2(50):='learning sql';
  v_new_text VARCHAR2(100);
BEGIN
  v_new_text := v_text || 'and also plsql';
  dbms_output.put_line(v_new_text);
END;
```

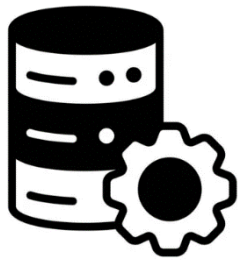
En esta nueva salida tenemos dos variables declaradas. La variable v_text usada anteriormente y la nueva variable denominada v_new_text. Esta nueva variable contiene la concatenación del texto asignado a v_text y del texto “and also sql”.

## CONSTANTES EN PLSQL

En PL/SQL podemos declarar constantes que son variables cuyo valor asignado no se puede modificar en el resto del programa. Para declarar una variable como constante, agregamos la palabra clave CONSTANT después del nombre de la variable y antes del tipo de datos. La constante debe definirse y asignarse al mismo tiempo. Para este ejemplo modificaremos el ejemplo anterior:

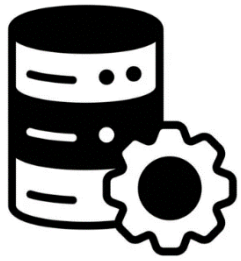
```
DECLARE
  c_text CONSTANT VARCHAR2(50):='learning sql';
  v_new_text VARCHAR2(100);
BEGIN
  v_new_text := c_text || 'and also plsql ';
  dbms_output.put_line(v_new_text);
END;
```

En este caso declaramos a la variable c_text como constante. Ese valor asignado en la declaración no puede luego modificarse en el resto del código.



**Nota**

*El error ORA-06550 significa que no puede asignar nuevamente un valor a esa variable porque se definió como constante.*



**Nota**

*Oracle posee un sitio para obtener información y ayuda sobre los errores de Oracle Database. Puede acceder fácilmente al sitio a través de la URL https://docs.oracle.com/error-help/db/.*

## COMENTARIOS EN PLSQL

Al programar en PL/SQL tenemos la posibilidad de escribir textos dentro del código que nos ayudan a describir qué hace o cómo funciona dicho codigo. Es a esto a lo que llamamos comentarios y el compilador de la base de datos de Oracle los pasa por alto sin afectar al funcionamiento del programa. Para insertar comentarios de una línea en PL/SQL, debe usar doble guion (--) al principio del comentario. Si en cambio queremos insertar comentarios de varias líneas en PL/SQL, debemos usar una barra y un asterisco (/*) al principio del comentario y un asterisco una barra (*/) al final del comentario. Esto puede ser un ejemplo:

```
/*
 Program: test_constant
 Author: Claudio Alonso
 Change history:
 10-06-2023 Incorporate new constants
 09-05-2023 Program created
*/
```

## INTEGRACION CON SQL

Uno de los aspectos más importantes de PL/SQL es su estrecha integración con SQL. No es necesario usar un lenguaje de Alto Nivel como C o JAVA para ejecutar sentencias de SQL Por ejemplo supongamos que queremos mostrar el nombre del empleado cuyo id es 100:

```
DECLARE
  v_fname VARCHAR2(20);
BEGIN
  SELECT  first_name INTO v_fname
  FROM    employees
  WHERE   employee_id=100;
  dbms_output.put_line('El nombre del empleado es ' || v_fname);
END;
```

El nombre del empleado es almacenado en la variable v_fname después de ejecutar la instrucción SELECT de SQL.
Ahora realizaremos otro ejercicio. En este ejercicio calcularemos el salario anual con una consulta SELECT del empleado cuyo id es 100. Luego dicho resultado se muestra por pantalla.

```
DECLARE
    v_annual_salary   NUMBER(9, 2);
    v_last_name       employees.last_name%TYPE;
BEGIN
    SELECT salary * 12, last_name
    INTO v_annual_salary, v_last_name
    FROM employees
    WHERE employee_id = 100;
    dbms_output.put_line('El empleado: ' || v_last_name ||
                         ' tiene un salario anual: ' || v_annual_salary);
END;
```

El atributo %TYPE es usado en la sección de declaración para declarar una variable que tendrá el mismo tipo que una columna de una tabla.

## ESTRUCTURA IF

La estructura más utilizada en la mayoría de lenguajes de programación es la estructura IF. Se emplea para tomar decisiones en función de una condición. Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro de un bloque. También puede evaluarse si la condición no se cumple (es decir, si su valor es false). Para este caso se debe crear otro bloque de instrucciones después de la palabra reservada else.

```
IF (condition) THEN
  your_code;
ELSE
  your_other_code;
END IF;
```

Probaremos un ejemplo muy sencillo:

```
DECLARE
  v_count number;
BEGIN
  SELECT count(*)
  INTO v_count
  FROM employees
  WHERE employee_id= 100;
  IF (v_count = 1) THEN
      dbms_output.put_line('El empleado Existe');
  ELSE
      dbms_output.put_line('El empleado No Existe');
  END IF;
END;
```

En este caso estamos buscando el empleado 100 si llega a existir muestra el mensaje: El empleado Existe. Caso contrario muestra el mensaje El empleado No Existe. Ahora evaluaremos el salario de un empleado:

```
DECLARE
    v_annual_salary   NUMBER(9, 2);
    v_last_name       employees.last_name%TYPE;
BEGIN
    SELECT salary * 12, last_name
    INTO v_annual_salary, v_last_name
    FROM employees
    WHERE employee_id = 100;
    dbms_output.put_line('El empleado: ' || v_last_name ||
                         ' tiene un salario anual: ' || v_annual_salary);
    IF (v_annual_salary<300000) THEN
        dbms_output.put_line('Debe recibir un bono');
    ELSE
        dbms_output.put_line('No debe recibir un bono');
    END IF;
END;
```

Para este caso se declaró una variable denominada v_annual_salary que contendrá el salario anual de un empleado. Con dicho valor entraremos dentro de una estructura IF donde comprobaremos si el valor es menor a 300000. En ese caso el mensaje que aparecerá será:

```
Debe recibir un bono.
```

Si el valor hubiera sido mayor o igual a 300000 el texto a mostrar seria:

```
No debe recibir un Bono.
```

Otra característica de las declaraciones IF de PL/SQL es la capacidad de tener múltiples condiciones dentro de una sola declaración IF. La sintaxis en PL/SQL seria la siguiente:

```
IF (condition) THEN
  your_code;
ELSIF (condition2) THEN
  your_second_code;
ELSE
  your_other_code;
END IF;
```

Por ejemplo, supongamos que nuestra empresa ahora quiere evaluar (además de los salarios anuales anteriores) si el salario anual es igual a 300000.

```
DECLARE
    v_annual_salary   NUMBER(9, 2);
    v_last_name       employees.last_name%TYPE;
BEGIN
    SELECT salary * 12, last_name
    INTO v_annual_salary, v_last_name
    FROM employees
    WHERE employee_id = 100;
    dbms_output.put_line('El empleado: ' || v_last_name || ' tiene un salario
                         anual: ' || v_annual_salary);
    IF (v_annual_salary<300000) THEN
        dbms_output.put_line('Debe recibir un Bono');
    ELSIF (v_annual_salary=300000) THEN
        dbms_output.put_line('No se decidió si recibe un Bono');
    ELSE
        dbms_output.put_line ('No debe recibir un Bono');
    END IF;
END;
```

Como se puede apreciar en el código se colocó un ELSEIF para evaluar si el salario anual es igual a 300000. Si sucede ese caso el mensaje será No se decidió si recibe un Bono.

## SENTENCIA CASE

Una sentencia CASE devuelve un resultado basado en una o más alternativas. Para devolver el resultado, la sentencia CASE utiliza un selector, que es una expresión cuyo valor se utiliza para devolver una de varias alternativas. El selector va seguido de una o más cláusulas WHEN que se comprueban secuencialmente. El valor del selector determina qué resultado se devuelve. Si el valor del selector es igual al valor de una expresión WHEN, se ejecuta esa cláusula y se devuelve ese resultado. La sintaxis básica de esta sentencia es:

```
CASE selector
   WHEN expression1 THEN result1
   [WHEN expression2 THEN result2
   ...
   WHEN expressionN THEN resultN]
   [ELSE resultN+1]
END;
```

Para desarrollar un ejemplo tomaremos el código base del ejemplo anterior:

```
DECLARE
    v_annual_salary   NUMBER(9, 2);
    v_last_name       employees.last_name%TYPE;
    v_message_txt     VARCHAR2(100);
BEGIN
    SELECT salary * 12, last_name
    INTO v_annual_salary, v_last_name
    FROM employees
    WHERE employee_id = 100;
      v_message_txt := CASE
      WHEN v_annual_salary<300000 THEN 'Debe recibir un Bono'
      WHEN v_annual_salary=300000 THEN 'No se decidió si recibe un Bono'
      ELSE 'No debe recibir un Bono'
      END;
    DBMS_OUTPUT.PUT_LINE ('El empleado : ' || v_last_name || ' ' ||
                          v_message_txt);
END;
```

En el ejemplo anterior, la expresión CASE consulta por el valor de la variable v_annual_salary. Según el valor calculado del salario anual retorna si va a recibir un bono dicho empleado.

## BUCLES EN PL/SQL

Un bucle es una característica de los lenguajes de programación permiten ejecutar un bloque de código varias veces. Hay varias formas de crear bucles en PL/SQL. Ahora veremos de qué forma.

## LOOP BASICO

El tipo de bucle más básico en PL/SQL es el LOOP este iniciará un bucle y solo saldrá cuando se lo indique. Ahora veremos su sintaxis:

```
BEGIN
  LOOP
    your_code
    IF (condition) THEN
      EXIT;
    END IF;
  END LOOP;
END;
```

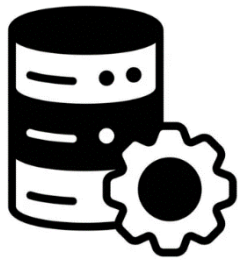
Dentro de las sentencias Loop y End Loop se indica todo el código que debe repetirse. El código se ejecuta línea por línea. Cuando se llega al End Loop salta de nuevo al comienzo del bloque. Si se llega a una instrucción EXIT, el bucle sale y se ejecuta cualquier código después del bucle. Veamos un ejemplo, donde queremos mostrar una salida cinco veces por pantalla.

```
DECLARE
  v_loop_counter NUMBER(3) := 0;
BEGIN
  LOOP
    v_loop_counter := v_loop_counter + 1;
    DBMS_OUTPUT.PUT_LINE('Write once, run everywhere');
    IF (v_loop_counter = 5) THEN
      EXIT;
    END IF;
  END LOOP;
END;
```

La salida muestra la línea "Write once, run everywhere" cinco veces, lo que cumple con los criterios de nuestro código. El contador del bucle comienza en 0, se incrementa en 1 cada vez que se ejecuta y, una vez que llega a 5, sale del bucle. Ahora consideraremos el siguiente ejercicio, necesitamos saber la cantidad de contratados por cada mes del año, independiente de cada uno. Para poder hacer este ejercicio necesitamos un bucle que inicie en valor 1 y termine en 12 por cada mes necesito contar la cantidad de empleados contratados usando una instrucción SELECT y una función de agregación AVG().

```
DECLARE
  v_c     NUMBER(2);
  v_loop_counter NUMBER(3) := 0;
BEGIN
  DBMS_OUTPUT.PUT_LINE (upper(rpad('Mes ',4))
                        || upper('Cantidad de Empleados '));
  LOOP
  v_loop_counter := v_loop_counter + 1;
    SELECT COUNT(*) INTO v_c
    FROM employees
    WHERE TO_CHAR(hire_date, 'MM') = v_loop_counter;
    DBMS_OUTPUT.PUT_LINE (rpad(TO_CHAR(v_loop_counter),4)|| TO_CHAR(v_c));
    IF (v_loop_counter = 12) THEN
      EXIT;
    END IF;
  END LOOP;
END;
```

La función rpad() es utilizada cuando se quiere colocar espacios en blanco (u otros caracteres) después del final de la cadena. Cuando se llama a esta función necesita primero indicar cuál es la cadena de caracteres y la longitud hasta la cual se desea completar con el carácter de separación. Si no se especifica ningún carácter de relleno ORACLE DATABASE utiliza en forma predeterminada el espacio en blanco. Si desea especificar otro carácter de relleno debe colocarse como argumento de la función después de la longitud.



**Nota**

*La función lpad() completa la cadena con espacios en blanco (u otros caracteres) por la izquierda hasta completar la longitud definida.*

## BUCLE FOR

PL/SQL ofrece otro tipo de bucle llamado bucle FOR. Este bucle FOR le permite definir los criterios del bucle al principio, lo que facilita ver cómo se ejecuta el bucle. La sintaxis de un bucle FOR se ve así:

```
FOR counter IN start_value ..  end_value LOOP
  your_code
END LOOP;
```

Dentro de la sintaxis podemos ver lo siguiente:

- La palabra clave FOR inicia el ciclo
- La variable contadora es una variable que se utiliza como contador para el ciclo. Se incrementa en 1 cada vez que se ejecuta el ciclo.
- start_value es el valor en el que se establece inicialmente el contador.
- end_value es el valor en el que termina el contador. Cuando el valor del contador es mayor que el valor final, se sale del ciclo y se ejecuta el código posterior al ciclo.

Aquí se muestra un ejemplo del bucle FOR:

```
BEGIN
FOR counter IN 2002..EXTRACT(YEAR FROM sysdate) LOOP
  DBMS_OUTPUT.PUT_LINE('Yo viví el año :' || counter);
END LOOP;
END;
```

En este caso listamos los años desde 2002 hasta el año actual.

## EL CICLO WHILE

Es un tipo de bucle que se ejecuta hasta que se cumple la condición especificada. Es similar al LOOP básico, pero incluye una condición cuando se la define:

```
WHILE (condition) LOOP
  your_code
END LOOP;
```

Esta sintaxis incluye una condición, lo que significa que el código dentro del bucle solo se ejecuta si la condición es verdadera. El código dentro del ciclo debe garantizar que en algún momento la condición se vuelva falsa de lo contrario, terminará con un ciclo infinito. El siguiente ejemplo muestra cómo podemos mostrar los nombres de los doce meses siguientes a partir del mes actual:

```
DECLARE
v_month DATE;
v_counter INTEGER;
BEGIN
v_counter:=0;
WHILE (v_counter<12) LOOP
  select add_months(sysdate,v_counter) into v_month from dual;
  dbms_output.put_line(to_char(v_month,'MONTH'));
  v_counter:=v_counter+1;
END LOOP;
END;
```

La variable v_counter va a almacenando temporariamente el valor que se va incrementando. De esta forma cuando supera el valor de 12 la condición se vuelve falsa y sale del bucle while. Ahora para completar todo lo visto haremos un ejercicio completo con estructuras de decisión y estructuras de repetición.

```
DECLARE
v_hd_month NUMBER;
v_hd_day NUMBER;
v_now_day NUMBER;
v_now_month NUMBER;
BEGIN
FOR R IN (SELECT HIRE_DATE,LAST_NAME FROM EMPLOYEES )
LOOP
    v_hd_month :=extract(MONTH from R.HIRE_DATE);
    v_hd_day := extract(DAY from R.HIRE_DATE);
    v_now_day := extract(DAY from SYSDATE);
    v_now_month := extract(MONTH from SYSDATE);
IF v_now_day=v_hd_day AND v_hd_month=v_now_month THEN
   DBMS_OUTPUT.PUT_LINE('Felicidades, gracias por estar con nosotros un año
   Más, : '|| R.LAST_NAME );
END IF;
END LOOP ;
END;
```

Este procedimiento visualiza aquellos empleados que cumplen un aniversario dentro de la empresa. Para ello se recorre con un bucle a todos los empleados obteniendo su día y mes de ingreso, luego se lo compara el día y mes actual a través de una estructura de decisión y en el caso de que sean iguales los valores se procede a informar mediante un mensaje Feliz aniversario.

## PROCEDIMIENTOS PL/SQL

Un procedimiento PL/SQL, o un procedimiento almacenado, es una unidad de programa reutilizable que encapsulan una serie de declaraciones PL/SQL para una tarea específica que además posee un nombre y se encuentra almacenado en la base de datos. Este procedimiento se puede ejecutar invocándolo desde otro código usando su nombre. Podríamos hacernos la siguiente pregunta ¿Por qué crear un procedimiento almacenado? La respuesta es simple este código se puede reutilizar en distintas partes de un programa. Por ejemplo, en lugar de confiar en que cada aplicación o desarrollador intente escribir una declaración INSERT que funcione, puede crear un procedimiento que acepte algunos parámetros y ejecute una declaración INSERT con ellos. Esto significa que las declaraciones INSERT son consistentes y los desarrolladores saben lo que deben proporcionar. Por otro lado, una función PL/SQL es una pieza de código que posee un nombre en la base de datos que también se le pueden proporcionar parámetros, realiza cálculos, pero solo puede devolver un resultado. La diferencia clave entre procedimientos y funciones es que los procedimientos no devuelven un valor, mientras que las funciones devuelven un único valor.

## ¿CÓMO CREAR UN PROCEDIMIENTO PL/SQL?

Para crear un procedimiento PL/SQL se usa la sentencia CREATE PROCEDURE. La sintaxis de la declaración es la siguiente:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[ ( parameter_name [ IN | OUT | IN OUT ] parameter_type [, ...] ) ]
{ IS | AS }
BEGIN
  procedure_body
END procedure_name;
```

Hay algunas cosas a tener en cuenta aquí:

- El procedimiento tiene como nombre procedure_name (el nombre del procedimiento a crear).
- La palabra clave OR REPLACE es opcional. Si lo incluye, significa que cuando ejecute esta declaración, reemplazará el procedimiento con el mismo nombre. Si no lo incluye y ya existe un procedimiento con este nombre, obtendrá un error.

También se pueden proporcionar uno o más parámetros:

- El parameter_name es el nombre del parámetro, que se utiliza en el procedimiento.
- Un parámetro puede ser un parámetro IN (El modo por defecto, donde el parámetro se usa solo para la entrada. El programa que invoca a ese procedimiento proporciona un valor y el procedimiento/función no puede modificarlo) un parámetro OUT (el parámetro se utiliza solo para la salida. El procedimiento/función establece el valor y el programa que lo invoca puede usarlo después de la ejecución del procedimiento/función), un parámetro IN OUT (El parámetro se utiliza tanto para la entrada como para la salida. El programa que lo invoca proporciona un valor inicial y el procedimiento/función puede modificarlo).
- Un parámetro tiene un tipo que es un tipo de datos SQL, como NUMBER.
- Se pueden agregar más parámetros y separarlos por comas.

Después de definir los parámetros:

- Especifique la palabra AS para iniciar el código de procedimiento.
- BEGIN se especifica describir el bloque de ejecución del procedimiento.
- El procedimiento termina con END procedure_name.

Supongamos que tenemos que visualizar el nombre completo de un empleado basado en su código de empleado. Para ello solicitaremos como parámetro IN el código del empleado y en un parámetro OUT pondremos el nombre completo del empleado.

```
CREATE or REPLACE PROCEDURE get_employee_info(p_employee_id IN
NUMBER,p_full_name OUT VARCHAR2) AS
BEGIN
  SELECT first_name || ' ' || last_name
  INTO p_full_name
  FROM employees
  WHERE employee_id = p_employee_id;
END get_employee_info;
```

Si ejecutamos este procedimiento, este es el resultado que obtenemos:

```
Procedure GET_EMPLOYEE_INFO compilado
```

Luego podemos generar el mensaje con el nombre del empleado usando otro código PL/SQL

```
DECLARE
  v_output_message VARCHAR2(100);
BEGIN
  get_employee_info(101,v_output_message);
  DBMS_OUTPUT.PUT_LINE(v_output_message);
END;
```

En este código, hemos declarado una nueva variable para el mensaje de salida. En el bloque BEGIN, ejecutamos la función get_employee_info con nuestro parámetro con la información del código del empleado. El segundo parámetro es la variable v_output_message, que se declara dentro del procedimiento. Luego mostramos ese valor con la línea:

```
DBMS_OUTPUT.PUT_LINE(v_output_message);
```

El mensaje obtenido finalmente es:

```
Neena Kochhar
```

Si necesito listar todos los procedimientos almacenados debo escribir la siguiente instrucción:

```
SELECT object_name FROM user_procedures;
```

Si necesita editar el procedimiento recién creado en SQL DEVELOPER para poder hacer un cambio de código debe sobre el Navegador de conexiones seleccionar la conexión actual y mostrar el objeto procedimiento y sobre el nombre del procedimiento con el botón derecho escoger la opción EDITAR. Al momento se abrirá todo el código del procedimiento sobre la ventana de código.

## ATRIBUTO %TYPE y %ROWTYPE

Normalmente, las variables que creemos en PL/SQL van a hacer referencias a campos de una tabla de la base de datos. Como tendríamos que usar el mismo tipo que del campo para no tener problemas podemos usar el atributo %type, esto hace que el dato del campo sea del mismo tipo que la variable. Ejemplo:

```
DECLARE
v_hire_date employees.hire_date%type;
```

De esta forma creamos una variable denominada v_hire_date que será del mismo tipo que el de la columna hire_date de la tabla employees. También tenemos otro atributo más en PL/SQL llamado %rowtype que es como él atributo %type pero en lugar de un campo hace referencia a toda una fila o registro de una tabla (por ello en este caso no es necesario indicar el campo, pero si la tabla). Vamos a mostrar el siguiente ejemplo:

```
DECLARE
v_employee employees%rowtype;
```

Ahora realizaremos un ejercicio más completo desarrollando toda esta teoría. La idea es construir un procedimiento donde le pasemos el número del empleado y nos devuelva su apellido, correo electrónico y su salario:

```
CREATE OR REPLACE PROCEDURE emp_sal_query (p_empnro IN
employees.employee_id%TYPE)
IS
    r_emp employees%ROWTYPE;
BEGIN
    SELECT employee_id, last_name, email, salary INTO r_emp.employee_id,
    r_emp.last_name, r_emp.email, r_emp.salary
    FROM employees WHERE employee_id = p_empnro;
    DBMS_OUTPUT.PUT_LINE('Employee # : ' || p_empnro);
    DBMS_OUTPUT.PUT_LINE('Name       : ' || r_emp.last_name);
    DBMS_OUTPUT.PUT_LINE('Job        : ' || r_emp.email);
    DBMS_OUTPUT.PUT_LINE('Hire Date  : ' || r_emp.salary);
END;
```

Para poder practicar, realizaremos un stored procedure más. En este caso listaremos los nombres y apellidos de los empleados que fueron contratados en un determinado año y en un departamento dado.

```
CREATE OR REPLACE PROCEDURE emp_full_name( p_depto_name IN
departments.department_name%TYPE,p_year IN NUMBER) IS
v_first_name employees.first_name%TYPE;
v_last_name employees.last_name%TYPE;
BEGIN
   FOR D IN (SELECT e.first_name, e.last_name into v_first_name,v_last_name
               FROM employees e
               join departments d
                 on e.department_id = d.department_id
               where d.department_name = p_depto_name
                and extract(year from e.hire_date) = p_year)
   LOOP
        DBMS_OUTPUT.PUT_LINE(D.FIRST_NAME || ' ' || D.LAST_NAME);
   END LOOP;
END emp_full_name;
```

## FUNCIONES EN PL/SQL

El otro tipo de objeto que puede crear con PL/SQL es una función. Una función PL/SQL ejecuta un bloque de código y devuelve un valor. Crear una función es similar a crear un procedimiento:

```
CREATE [OR REPLACE] FUNCTION function_name
[ ( parameter_name [ IN | OUT | IN OUT ] parameter_type [, ...] ) ]
RETURN return_datatype
{ IS | AS }
BEGIN
  function_body
END function_name;
```

La principal diferencia con un procedimiento es la palabra clave return. Esto significa que cada vez que se llama a esta función, devuelve el valor del return.

```
CREATE OR REPLACE FUNCTION emp_count(p_deptno in number)
RETURN number is
cnt number(2) := 0;
BEGIN
SELECT count(*)  INTO cnt
FROM   employees e
WHERE  e.department_id = p_deptno;
RETURN cnt;
END;
```

Como retornaremos el valor dentro de la variable cnt. Hay que definirla con la palabra reservada IS. Para probar la función debemos probarlo dentro de otro bloque PL/SQL:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(emp_count(9));
END;
```

Debido a que este código es una función, también podemos llamarlo desde SQL de esta forma:

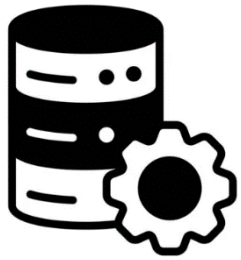
```
SELECT emp_count(9) FROM DUAL;
```

O simplemente:

```
SELECT emp_count(9);
```

## USO DE VARRAYS EN PL/SQL

Sin duda, PL/SQL es un lenguaje poderoso para usarlo con distintos tipos de datos. Una colección es un grupo de elementos de tipos de datos homogéneos. Un tipo de colección especial son los VARRAY. Un VARRAY (Arreglo de tamaño variable) se utiliza para almacenar varios datos simultáneamente donde cada información almacenada es del mismo tipo (Por ejemplo, puede servir para almacenar números de departamentos, legajos de empleados, etc.). Los datos se pueden acceder a través de un índice de tipo entero.



**Nota**

*Un VARRAY es como una matriz unidimensional en otros lenguajes de programación.*

La instrucción CREATE TYPE crea un tipo de dato VARRAY. Se debe especificar el tamaño máximo y el tipo de elementos almacenados en dicho VARRAY. La sintaxis es la siguiente:

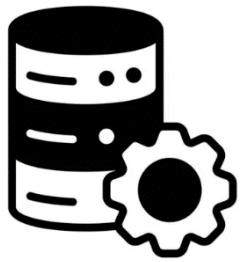
```
CREATE OR REPLACE TYPE varray_type_name IS VARRAY(n) of <element_type>;
```

Por ejemplo, crearemos el VARRAY denominado emp_type de 3 elementos con el tipo de dato number(8,2) para cada elemento del array.

```
CREATE OR REPLACE TYPE emp_type IS varray(3) of number(8,2);
```

Al tiempo vera el siguiente mensaje en la ventana de salida:

```
Type EMP_TYPE compilado
```



**Nota**

*Todos los elementos de un VARRAY están identificados con un índice. A diferencia de otros lenguajes de programación el primer índice del array en PL/SQL comienza en uno y no en cero.*

Ahora usaremos el VARRAY recién creado para almacenar el salario de todos los empleados, el salario máximo encontrado y el mínimo.

```
CREATE OR REPLACE FUNCTION emp_sal
RETURN emp_type is
    emp emp_type := emp_type();
    v_salary employees.salary%TYPE;
    v_max employees.salary%TYPE;
    v_min employees.salary%TYPE;
BEGIN
    SELECT sum(salary),max(salary),min(salary) INTO v_salary,v_max,v_min
    FROM EMPLOYEES;
    emp := emp_type(v_salary,v_max,v_min);
    RETURN emp;
END;
```

Tenemos que inicializar una variable de tipo VARRAY antes de usarla, de lo contrario, se producirá un error. Para inicializar un VARRAY se debe indicar el nombre del VARRAY que se usara, el tipo de VARRAY (en este caso emp_type, que es el tipo de dato creado anteriormente) y se le asigna los elementos. Para este caso no le asignamos ningún elemento. Una vez obtenido la suma de los salarios, el valor máximo y el valor mínimo se almacena en distintas variables. Luego dichos valores se le asigna al VARRAY creado. Finalmente, como es una función podemos invocarla de la siguiente manera:

```
SELECT emp_sal FROM dual;
```

## REGISTROS DEFINIDOS POR EL USUARIO

Un registro es un tipo de dato compuesto, lo que significa que puede contener más de un dato, a diferencia de un tipo de dato escalar, como un número o una cadena. Ya ha visto cómo declarar una variable de registro mediante el atributo %ROWTYPE. Pero también puede declarar sus propios tipos de registro mediante la instrucción TYPE...RECORD. Estos tipos de registro definidos por el usuario resultan útiles al declarar conjuntos de variables individuales, por ejemplo, en vez de declarar esto:

```
DECLARE
   v_first_name  employees.first_name%TYPE;
   v_last_name   employees.last_name%TYPE;
   v_hire_date   employees.hire_date%TYPE;
   v_salary      employees.salary%TYPE;
```

En lugar de tener varias variables escalares separadas, ¿Por qué no creamos nuestro propio tipo de registro de esta manera?

```
DECLARE
TYPE r_employee_type IS RECORD
(
       v_first_name  employees.first_name%TYPE,
       v_last_name   employees.last_name%TYPE,
       v_hire_date   employees.hire_date%TYPE,
       v_salary      employees.salary%TYPE,
);
v_employee  r_employee_type;
```

Observe la ventaja de usar registros propios: permite escribir código simple, limpio y fácil de mantener. En lugar de trabajar con largas listas de variables o parámetros, se puede trabajar con un registro que contiene toda esa información. Ahora desarrollaremos el siguiente ejercicio:

```
create or replace procedure statistics_dept(p_dept_name IN
departments.department_name%TYPE) is
TYPE r_dept IS RECORD (
        department_name     departments.department_name%TYPE,
        department_id       departments.department_id%TYPE,
        employees_total     NUMBER,
        salary_total        NUMBER,
        salary_min          NUMBER,
        salary_max          NUMBER
 );
 v_info_depto r_dept;
 BEGIN
        SELECT d.department_name, d.department_id,count(*) as cantidad,
        sum(e.salary),max(e.salary), min(e.salary)INTO
        v_info_depto.department_name, v_info_depto.department_id,
        v_info_depto.employees_total, v_info_depto.salary_total,
        v_info_depto.salary_max,v_info_depto.salary_min
        FROM departments d
        INNER JOIN employees e ON e.department_id = d.department_id
        WHERE UPPER(d.department_name) = UPPER(p_dept_name)
        GROUP BY d.department_name,d.department_id;
```

```
    DBMS_OUTPUT.PUT_LINE('');
    DBMS_OUTPUT.PUT_LINE('=== ESTADÍSTICAS DEL DEPARTAMENTO ===');
    DBMS_OUTPUT.PUT_LINE('Nombre de Departamento: ' ||
    v_info_depto.department_name);
    DBMS_OUTPUT.PUT_LINE('ID del Departamento: ' ||
    v_info_depto.department_id);
    DBMS_OUTPUT.PUT_LINE('Total de empleados: ' ||
    v_info_depto.employees_total);
    DBMS_OUTPUT.PUT_LINE('Salario total del departamento: $' ||
    TO_CHAR(v_info_depto.salary_total, '999,999,999.99'));
    DBMS_OUTPUT.PUT_LINE('Salario mínimo: $' ||
    TO_CHAR(v_info_depto.salary_min, '999,999.99'));
    DBMS_OUTPUT.PUT_LINE('Salario máximo: $' ||
    TO_CHAR(v_info_depto.salary_max, '999,999.99'));
 END;
```

El procedimiento statistics_dept está diseñado para generar estadísticas de empleados por departamento. Devuelve el nombre del departamento, su id, el total de empleados, el total de salario, su valor mínimo y máximo. En vez de usar variables escalares usamos un registro denominado r_dept. Ahora definiremos lo que es una tabla anidada. En Oracle una tabla anidada es un tipo de columna que almacena una cantidad no especificada de filas sin ningún orden en particular. Ejemplo:

```
DECLARE
TYPE r_employee_type IS RECORD (
     v_first_name  employees.first_name%TYPE,
     v_last_name   employees.last_name%TYPE,
     v_hire_date   employees.hire_date%TYPE,
     v_salary      employees.salary%TYPE,
);
TYPE t_employees IS TABLE OF r_employee_type;
```

La última línea crea una tabla anidada llamada t_employees que puede contener múltiples elementos del tipo r_employee_type. Esto significa que t_employees es una colección que puede almacenar múltiples registros de empleados, donde cada elemento tiene la estructura definida en el RECORD anterior. Ahora realizaremos el siguiente ejemplo práctico:

```
CREATE OR REPLACE PROCEDURE get_employees_department(p_dept_id IN NUMBER)
IS
    TYPE t_employee_id IS TABLE OF NUMBER;
    v_employee_ids t_employee_id;
    v_dept_name departments.department_name%TYPE;
    v_total_employees NUMBER;

BEGIN

    SELECT department_name
    INTO v_dept_name
    FROM departments
    WHERE department_id = p_dept_id;

    v_employee_ids := t_employee_id();

    FOR emp_rec IN (
        SELECT employee_id
        FROM employees
        WHERE department_id = p_dept_id
        ORDER BY employee_id
    ) LOOP
        v_employee_ids.EXTEND;
        v_employee_ids(v_employee_ids.COUNT) := emp_rec.employee_id;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('===  DEPARTAMENTO: ' || v_dept_name || ' ===');
    DBMS_OUTPUT.PUT_LINE('Total de empleados: ' || v_employee_ids.COUNT);
    DBMS_OUTPUT.PUT_LINE('IDs de empleados:');

    FOR i in 1..v_employee_ids.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE('  - ID: ' || v_employee_ids(i));
    END LOOP;

END;
```

En este ejemplo se crea una tabla anidada llamada t_employee_id que va a contener múltiples elementos que serán de tipo NUMBER donde almacenara los id de los empleados que trabajan en el departamento pasado por parámetro.

## EXCEPCIONES EN PL/SQL

Las excepciones son errores en tiempo de ejecución o eventos inesperados que ocurren durante la ejecución de un bloque de código PL/SQL Algunos errores en su código se denominan error de sintaxis ya que son errores de incumplimiento de reglas sintácticas como olvidarse de colocar un punto y coma. Sin embargo, se producen las excepciones cuando no se pueden predecir el error antes de ejecutar el código, es decir son errores en tiempo de ejecución del programa PL/SQL como agregar un valor de texto a un tipo de datos numérico o tratar de dividir por cero.

La buena noticia en PL/SQL es que puede escribir código para trabajar con estas excepciones. Esto se llama "manejo de excepciones", y es donde su código puede ver que ha ocurrido un error y tomar una acción diferente. Sin el manejo de excepciones, sus errores se informan al programa que lo llama y se muestran en su IDE o directamente en la aplicación.

Hay dos tipos de excepciones:

1) Excepciones del sistema (predefinidas)
2) Excepciones definidas por el usuario

Las excepciones del sistema son las más comunes y están identificadas por Oracle con un nombre o un número. Por ejemplo, la excepción NO_DATA_FOUND sucede cuando no se encuentra registros
La sintaxis de las excepciones del sistema es la siguiente:

```
BEGIN
  executable_code;
EXCEPTION
  WHEN exception_type THEN
    exception_code;
  WHEN OTHERS THEN
    exception_code;
END;
```

En la sección BEGIN va su bloque de código de ejecución como se venía trabajando hasta ahora. Se agrega una sección de EXCEPTION después de su código ejecutable (antes de la instrucción END). Esta sección EXCEPTION contiene código para manejar cualquier excepción que ocurra. Dentro de la sección EXCEPTION hay una serie de instrucciones WHEN/THEN.
Cada una de estas declaraciones se relaciona con una posible excepción que se puede encontrar como si fuera una instrucción IF. Cuando se encuentra una excepción al ejecutar el código, se lanza la excepción Si no podemos atrapar la excepción ya que no coincide con ninguna de las declaraciones WHEN/THEN entonces se ejecuta la excepción WHEN OTHERS THEN.
Ahora probaremos el siguiente procedimiento. A este procedimiento se le pasa un parámetro que es el código del empleado luego en base a ese código debe devolver la información del empleado:

```
CREATE OR REPLACE PROCEDURE employee_data(p_empnro NUMBER)
IS
v_out VARCHAR2(255);
BEGIN
SELECT initcap(first_Name)||' , '||initcap(last_name)
INTO v_out
FROM employees
WHERE employee_id = p_empnro;
dbms_output.put_line('El empleado es : ' || v_out);
EXCEPTION
WHEN no_data_found THEN
     dbms_output.put_line('No se ha encontrado ningún empleado');
END;
```

Si el parámetro introducido no devuelve ninguna fila la excepción muestra el mensaje "No se ha encontrado ningún empleado". Por ejemplo, pruebe con el siguiente código:

```
BEGIN
employee_data(500);
END;
```

El mensaje que aparece será el siguiente:

```
No se ha encontrado ningún empleado
```

En cualquier programa, existe la posibilidad de que se produzcan una serie de errores que pueden no ser considerados como excepciones por Oracle. En ese caso, el programador puede definir una excepción mientras escribe el código. Este tipo de excepciones se denominan excepciones definidas por el usuario. Las excepciones definidas por el usuario se definen en general para manejar casos especiales en los que nuestro código puede generar una excepción debido a nuestra lógica de código. Para realizar esto en nuestra sintaxis se debe usar la palabra clave RAISE y luego se maneja dentro del bloque EXCEPTION. A continuación, se muestra la sintaxis para ello:

```
DECLARE
        <exception name> EXCEPTION
BEGIN
        <sql sentence>
        If <test_condition> THEN
                RAISE <exception_name>;
        END IF;
        EXCEPTION
                WHEN <exception_name> THEN
                        -- some action
END;
```

Para ello veremos el siguiente ejemplo:

```
CREATE OR REPLACE PROCEDURE employee_data(p_empnro NUMBER)
IS
v_out VARCHAR2(255);
ex_invalid_id  EXCEPTION;
BEGIN
IF p_empnro <= 0 THEN
      RAISE ex_invalid_id;
ELSE
SELECT initcap(first_Name)||' , '||initcap(last_name)
INTO v_out
FROM employees
WHERE employee_id = p_empnro;
dbms_output.put_line('El empleado es : ' || v_out);
END IF;
EXCEPTION
WHEN ex_invalid_id THEN
     dbms_output.put_line('ID Debe ser mayor a cero !');
WHEN no_data_found THEN
     dbms_output.put_line('No se ha encontrado ningún empleado');
WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Otro Error -> '||SQLERRM);
END;
```

En este ejemplo se usa una excepción definida por el usuario llamada ex_invalid_id que se genera cuando el número de empleado pasado por parámetro es menor o igual a cero. Podemos probar el siguiente procedimiento y ver qué sucede:

```
BEGIN
    employee_data(-1);
END;
```

Como el parámetro es menor a 0 se lanzará la excepción y mostrará el siguiente mensaje:

```
ID Debe ser mayor a cero !
```

## CURSORES EN PL/SQL

Un cursor es un puntero al área de trabajo o al área de contexto, utilizado por el motor de Oracle para ejecutar sentencias SQL. El cursor contiene varios datos, incluidas las filas que ha devuelto la declaración (si es una declaración SELECT) y atributos sobre el conjunto de resultados, como el número de filas afectadas por la declaración. Hay dos tipos de cursores en Oracle SQL: cursores implícitos y cursores explícitos.

## CURSORES IMPLÍCITOS

Un cursor implícito es un tipo de cursor creado automáticamente por Oracle cuando se ejecuta una instrucción SQL. Se llama cursor implícito porque se crea automáticamente (no necesita hacer nada para que se cree). Cada vez que ejecuta una instrucción DML (INSERT, UPDATE, DELETE) o DQL (SELECT), se crea un cursor implícito:

- Para declaraciones INSERT, el cursor contiene los datos que se están insertando.
- Para las instrucciones UPDATE o DELETE, el cursor identifica las filas que se verán afectadas
- Para las declaraciones SELECT, las filas que se recuperan se pueden almacenar en un objeto.

¿Cómo pueden ayudarnos los cursores implícitos? Para las sentencias INSERT, UPDATE y DELETE, podemos ver varios atributos de las sentencias para entender si han funcionado o no. Veamos un ejemplo de esto:

```
BEGIN
   INSERT INTO COUNTRIES (country_id,country_name,region_id) VALUES
   ('PE','Perú',2);
   DBMS_OUTPUT.PUT_LINE('Registros Insertados : ' || sql%rowcount);
END;
```

La salida será:

```
Registros Insertados : 1
```

Este código incluye el atributo sql%rowcount que contiene el número de filas afectadas por la declaración SQL. Podemos hacer más con este valor, como almacenarlo en una variable:

```
DECLARE
  v_total_rows NUMBER(10);
BEGIN
  INSERT INTO COUNTRIES (country_id,country_name,region_id) VALUES
('BO','Bolivia',2);
  v_total_rows := sql%rowcount;
  IF (v_total_rows > 0) THEN
    DBMS_OUTPUT.PUT_LINE('Registros Insertados : ' || v_total_rows);
  ELSE
    DBMS_OUTPUT.PUT_LINE('No se ha insertado registros.');
  END IF;
END;
```

## CURSORES EXPLICITOS

¿Qué sucedería si ejecutamos el siguiente código?:

```
DECLARE
  v_last_name VARCHAR2(50);
BEGIN
  SELECT last_name
  INTO v_last_name
  FROM employees;
  DBMS_OUTPUT.PUT_LINE('The apellido is ' || v_last_name);
END;
```

Esta será la salida. El mensaje de error que recibimos es:

```
Informe de error -
ORA-01422: la recuperación exacta devuelve un número mayor de filas que el
solicitado
```

Este error ocurre porque tenemos varios valores devueltos por nuestra consulta, pero solo una variable para almacenarlos. ¿Cómo podemos manejar situaciones como estas? Cuando la ejecución de una consulta SELECT devuelve más de una fila debe usarse un cursor explícito. El uso de un cursor explícito implica varios pasos:

1. Declarar el cursor como una variable en la sección DECLARE.
2. Abrir el cursor.
3. Ejecutar la instrucción SELECT y almacenar los datos en el cursor.
4. Cerras el cursor y liberar la memoria asignada.

Un cursor se declara de esta forma:

```
CURSOR cursor_name IS select_statement;
```

.
A continuación, tenemos que abrir el cursor. Esto se hace en el bloque BEGIN:

```
DECLARE
  CURSOR c_employee IS
  SELECT last_name FROM employees;
BEGIN
  OPEN c_employee;
END;
```

Ahora, almacenamos los datos del cursor en una variable:

```
DECLARE
  v_last_name VARCHAR2(50);
  CURSOR c_employee IS
  SELECT last_name FROM employees;
BEGIN
  OPEN c_employee;
  FETCH c_employee INTO v_last_name;
END;
```

Entonces ahora tenemos que cerrar el cursor:

```
DECLARE
  v_last_name VARCHAR2(50);
  CURSOR c_employee IS
  SELECT last_name FROM employees;
BEGIN
  OPEN c_employee;
  FETCH c_employee INTO v_last_name;
  CLOSE c_employee;
END;
```

Luego imprimimos la variable por pantalla:

```
DECLARE
  v_last_name VARCHAR2(50);
  CURSOR c_employee IS
  SELECT last_name FROM employees;
BEGIN
  OPEN c_employee;
  FETCH c_employee INTO v_last_name;
  DBMS_OUTPUT.PUT_LINE('El apellido del empleado es : ' || v_last_name);
  CLOSE c_employee;
END;
```

La salida que se muestra es:

```
El apellido del empleado es : King
```

Sin embargo, hay más registro para mostrar de la tabla EMPLOYEES. ¿Cómo podemos manejar esto los cursores?

## CURSORES Y BUCLES EXPLÍCITOS

Si nuestra consulta SELECT devuelve varias filas de un cursor explícito, necesitamos usar un bucle para obtener todos los datos. Para ello la sintaxis es la siguiente:

```
LOOP
  FETCH cursor_name INTO variable;
  EXIT WHEN cursor_name%notfound;
  your_code;
END LOOP;
```

La parte importante de este código es que la instrucción FETCH obtendrá la siguiente fila del cursor. Esto significa que obtiene una fila a la vez.
La expresión EXIT WHEN cursor_name%notfound significa que saldrá del ciclo cuando no encuentre más filas en el cursor.

```
DECLARE
  v_last_name VARCHAR2(50);
  CURSOR c_employee IS
  SELECT last_name FROM employees;
BEGIN
  OPEN c_employee;
  LOOP
  FETCH c_employee INTO v_last_name;
  EXIT WHEN c_employee%notfound;
  DBMS_OUTPUT.PUT_LINE('El apellido del empleado es : ' || v_last_name);
  END LOOP;
  CLOSE c_employee;
END;
```

Como ve se visualiza todos los apellidos de la tabla EMPLOYEES. Los cursores explícitos admiten el uso de parámetros, estos son útiles cuando no queremos que nos devuelva siempre el mismo resultado ya que dependerán del parámetro pasado en el momento de apertura del cursor. Para ello construiremos un procedimiento denominado show_employees donde pasamos como parámetro un código de departamento. Luego con ese código se recorre la tabla employees para devolver un listado de todos los empleados que trabajan en ese departamento.

```
CREATE OR REPLACE PROCEDURE show_employees
    (p_id  employees.department_id%type)
    IS
    CURSOR c_emp_cursor(p_id employees.department_id%type)  IS
    SELECT employee_id, last_name FROM employees WHERE department_id = p_id;
BEGIN
    FOR v_emp_record IN c_emp_cursor(p_id)
    LOOP
        DBMS_OUTPUT.PUT_LINE(c_emp_cursor%rowcount|| ' ' ||
        v_emp_record.employee_id || ' ' ||v_emp_record.last_name);
    END LOOP;
END;
```

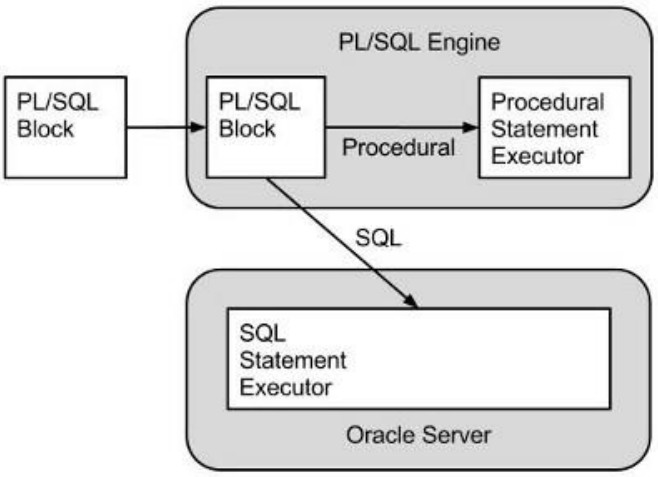
Cuando se define el cursor debe establecerse el nombre del parámetro (que es el mismo que pasamos en el procedimiento almacenado) junto con el tipo de dato de ese parámetro.
A continuación, desarrollaremos un ejercicio donde aplicaremos varios conceptos vistos como cursores, registros y varrays en forma conjunta dentro de un bloque anónimo. La idea central es trabajar con la tabla countries para mostrar el nombre del país y su código de región. Para ello empiezo a recorrer la tabla con un cursor a medida que tengo información de cada país lo agrego como registro a un varray denominado v_arr_countries:

```
DECLARE
    TYPE arr_countries is varray(1000) of countries%rowtype;
    CURSOR cur_countries is select * from countries;
    v_arr_countries arr_countries:=arr_countries();
BEGIN
FOR v_countries_record IN cur_countries
LOOP
 v_arr_countries.extend;
 v_arr_countries(v_arr_countries.count):=v_countries_record;
 dbms_output.put_line(rpad(v_arr_countries.count,4)||
 rpad(v_arr_countries(v_arr_countries.count).country_name,25) ||
 v_arr_countries(v_arr_countries.count).region_id);
END LOOP;
END;
```

El método extend de PL/SQL permite agregar un elemento al final del varray. El método count devuelve la cantidad de elementos del varray.

## BULK COLLECT

Casi todos los programas PL/SQL incluyen sentencias PL/SQL y SQL. Las sentencias PL/SQL son ejecutadas por el PL/SQL Statement Executor (ejecutor de sentencias PL/SQL); las sentencias SQL son ejecutadas por el SQL Statement Executor (ejecutor de sentencias SQL). Cuando el motor de ejecución PL/SQL encuentra una sentencia SQL, se detiene y la transfiere al motor SQL. El motor SQL ejecuta la sentencia SQL y devuelve información al motor PL/SQL. Esta transferencia de control se denomina context switch o cambio de contexto, y cada uno de estos cambios genera una sobrecarga que ralentiza el rendimiento general de los programas.



Supongamos que desarrolláramos un procedimiento para que actualice el salario de los empleados ingresados en el último año con la posibilidad de que el procedimiento también informe en una lista los empleados afectados a dicho aumento salarial. Con nuestros conocimientos podríamos haber hecho lo siguiente:

```
CREATE or REPLACE PROCEDURE increase_salary (p_increase_pct IN NUMBER,
p_list_employee OUT VARCHAR2) IS
v_list VARCHAR2(4000);
BEGIN
   FOR c_employee IN (SELECT employee_id, first_name, last_name,salary FROM
                      employees WHERE hire_date >= SYSDATE - 365)
   LOOP
      UPDATE employees SET salary = salary + salary * p_increase_pct WHERE
      employee_id = c_employee.employee_id;
      v_list := v_list || to_char(c_employee.employee_id) || ',';
   END LOOP;
   IF v_list IS NOT NULL THEN
      p_list_employee := rtrim(v_list, ',');
   END IF;
END;
```

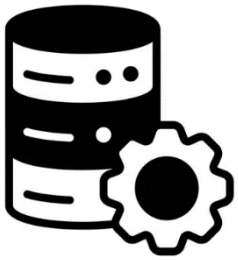
La ejecución de ese procedimiento podría ser el siguiente:

```
DECLARE
   v_out varchar2(255);
BEGIN
  increase_salary(0.05,v_out);
  dbms_output.put_line(v_out);
END;
```

Supongamos que hay 100 empleados que recibirán dicha actualización salarial. El motor PL/SQL cambiará al motor SQL 100 veces, una por cada fila que se actualice. Podríamos referirnos a este cambio fila por fila como un procesamiento lento que consume mucho recurso, es algo que definitivamente debe evitarse. Para ello usaremos la función de procesamiento masivo BULK COLLECT y la sentencia FORALL. La sentencia BULK COLLECT permite usar la instrucción SELECT para recuperar múltiples filas con un solo cambio de contexto. Mientras que la sentencia FORALL se usa cuando necesita ejecutar sentencia DML (como INSERT, UPDATE, DELETE) repetidamente para diferentes valores de variables de enlace. La sentencia UPDATE usada anteriormente se adapta a este escenario ya que en caja ejecución se debe pasar un nuevo ID de empleado. Ahora veamos como puedo hacer el mismo procedimiento, pero usando BULK COLLECT y FORALL:

```
CREATE or REPLACE PROCEDURE increase_salary_2 (p_increase_pct IN NUMBER,
p_list_employee OUT VARCHAR2) IS

v_list VARCHAR2(4000);
TYPE emp_rec_type IS RECORD (
employee_id employees.employee_id%TYPE,
first_name employees.first_name%TYPE,
last_name employees.last_name%TYPE,
salary employees.salary%TYPE
);

TYPE emp_rec_list IS TABLE OF emp_rec_type;
emp_data emp_rec_list;

BEGIN
SELECT employee_id, first_name, last_name,salary
BULK COLLECT INTO emp_data
FROM employees
WHERE hire_date >= SYSDATE - 365;

FORALL i IN 1..emp_data.COUNT
UPDATE employees
SET salary = salary * 1.1
WHERE employee_id = emp_data (i).employee_id;

FOR i IN 1..emp_data.COUNT LOOP
       IF i = 1 THEN
            v_list := TO_CHAR(emp_data (i).employee_id);
        ELSE
            v_list := v_list || ',' || TO_CHAR(emp_data (i).employee_id);
        END IF;
END LOOP;

p_list_employee := v_list;

END;
```

En este ejemplo, la consulta recupera datos de empleados contratados durante el último año y los almacena en la colección emp_data mediante BULK COLLECT. Fíjese el uso de BULK COLLECT, siempre se usa antes de la palabra clave INTO para luego proporcionar una o más colecciones después de esta. Oracle Database también ofrece una cláusula LIMIT para BULK COLLECT. Esto permite limitar la cantidad de datos a procesar, debido al consumo excesivo de memoria. A continuación, se utiliza la instrucción FORALL para actualizar el salario de cada empleado de la colección con el porcentaje pasado por parámetro, en lugar de alternar entre los motores PL/SQL y SQL para actualizar cada fila, FORALL agrupa todas las actualizaciones y las pasa al motor SQL con un único cambio de contexto. El resultado es una mejora extraordinaria del rendimiento. Este enfoque mejora significativamente el rendimiento en comparación con la situación anterior.



**Nota**

*Cada sentencia FORALL puede contener solo una sentencia DML. Si su bucle contiene dos actualizaciones y una eliminación, deberá escribir tres sentencias FORALL.*

## PAQUETES EN PL/SQL

Un paquete en PL/SQL es un objeto que agrupa código PL/SQL relacionado lógicamente. Este paquete contiene procedimiento y funciones que se puede invocar desde otro código PL/SQL. Los paquetes se compilan y se almacenan en la base de datos. Tienen dos partes: una especificación de paquete que define la interfaz pública para el paquete y un cuerpo que contiene el código requerido para cada procedimiento/función dentro del paquete. ¿Por qué usar Paquetes? Hay varias razones por las que debería usar paquetes para su código PL/SQL:

**Encapsulación:** Los paquetes encapsulan procedimientos, funciones y otras construcciones PL/SQL relacionadas, haciéndolos más fáciles de entender, administrar y modificar.

**Aplicaciones más fáciles de diseñar:** Puede escribir primero la especificación (la definición), y trabajar luego más tarde en el cuerpo (los detalles de implementación o el código).

**Mejor rendimiento:** En relación a su ejecución, cuando un procedimiento o función que está definido dentro de un paquete es llamado por primera vez, todo el paquete es ingresado a memoria. Por lo tanto, posteriores llamadas al mismo u otros sub-programas dentro de ese paquete realizarán un acceso a memoria en lugar de a disco. Esto no sucede con procedimientos y funciones estándares.

**Ocultar los detalles:** Puede proporcionar acceso a la especificación sin proporcionar los detalles de cómo funcionan. Puede cambiar los detalles cuando lo desee, y el código que se refiere al paquete no necesita cambiar.

**Desarrollo modular:** Puede combinar todo el código relacionado en un paquete para facilitar el desarrollo de su aplicación. Se pueden desarrollar interfaces entre paquetes y cada paquete puede ser autónomo, similar a cómo funcionan las clases en la programación orientada a objetos de otros lenguajes.

Los Paquetes están divididos en dos partes: La especificación (obligatoria) y el cuerpo (no obligatoria). La especificación o encabezado es la interfaz entre el Paquete y las aplicaciones que lo utilizan y es allí donde se declaran los tipos, variables, constantes, excepciones, cursores, procedimientos y funciones que podrán ser invocados desde fuera del paquete. En el cuerpo del paquete se implementa la especificación del mismo. El cuerpo contiene los detalles de implementación y declaraciones privadas, manteniéndose todo esto oculto a las aplicaciones externas, siguiendo el conocido concepto de “caja negra”. Sólo las declaraciones hechas en la especificación del paquete son visibles y accesibles desde fuera del paquete (por otras aplicaciones o procedimientos

almacenados) quedando los detalles de implementación del cuerpo del paquete totalmente ocultos e inaccesibles para el exterior.
Para acceder a los elementos declarados en un paquete basta con anteceder el nombre del objeto referenciado con el nombre del paquete donde está declarado y un punto, de esta manera:

```
<nombre_paquete>.<nombre_objeto>
```

Una especificación de paquete en PL/SQL es donde se declara el código público para el paquete. Contiene todas las variables, funciones y procedimientos que desea que otros conozcan y accedan. Las funciones y procedimientos contienen los nombres y tipos de parámetros. Esto es para que otros usuarios y programas puedan acceder a ellos y ejecutarlos, sin conocer los detalles de cómo funcionan. La sintaxis de la especificación de un paquete es la siguiente:

```
CREATE OR REPLACE PACKAGE <package_name> IS/AS
         FUNCTION <function_name> (<list of arguments>)
         RETURN <datatype>;
         PROCEDURE <procedure_name> (<list of arguments>);
         -- code statements
END <package_name>;
```

Por ejemplo, crearemos la siguiente especificación de un paquete denominado employee_pkg:

```
CREATE OR REPLACE PACKAGE employee_pkg AS
       FUNCTION get_department(p_emp_id NUMBER) RETURN VARCHAR2;
       FUNCTION get_employee_count(p_dept_id NUMBER) RETURN NUMBER;
END employee_pkg;
```

Al terminar de crear la especificación del paquete aparecerá el siguiente mensaje:

```
Package EMPLOYEE_PKG compilado
```

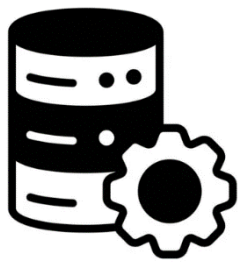
En esta especificación se crea un nuevo paquete que contiene dos funciones (get_department y get_employee_count. Estas funciónes tiene cada una un parámetro,
Por otro lado, tenemos el cuerpo del paquete que contiene los detalles de implementación que requiere la especificación del paquete. Esto incluye el código para cada uno de los procedimientos o funciones, y detalles de los cursores .
Aquí es donde va el código dentro de cada uno de los procedimientos y funciones. El cuerpo de un paquete se crea con la sentencia CREATE PACKAGE BODY. Las declaraciones de los objetos en el cuerpo deben coincidir con las de la especificación (nombres de procedimientos, parámetros). Siguiendo con el ejemplo anterior, vamos a escribir el cuerpo del paquete con los detalles de cada función:

Y este sería el cuerpo:

```
CREATE OR REPLACE PACKAGE BODY employee_pkg AS

    -- Implementation of get_departmen function
    FUNCTION get_department(p_emp_id NUMBER) RETURN VARCHAR2 IS
         v_department_name VARCHAR2(30);
    BEGIN
        SELECT d.department_name INTO v_department_name FROM departments d
        INNER JOIN employees e ON d.department_id=e.department_id
        WHERE e.employee_id = p_emp_id;
        RETURN v_department_name;
    END get_department;

    -- Implementation of get_employee_count function
    FUNCTION get_employee_count(p_dept_id NUMBER) RETURN NUMBER IS
          v_count NUMBER;
    BEGIN
       SELECT COUNT(*) INTO v_count FROM employees WHERE department_id =
        p_dept_id;
       RETURN v_count;
    END get_employee_count;

END employee_pkg;
```

Al terminar de crear el cuerpo del paquete aparecerá el siguiente mensaje:

```
Package Body EMPLOYEE_PKG compilado
```



**Nota**

*Si desea eliminar un paquete ejecute el comando DROP PACKAGE <nombre_paquete>*

Para ejecutar el código del paquete, debemos invocarlo de la siguiente forma:

```
SELECT employee_pkg.get_employee_count(9) FROM DUAL;
```

```
SELECT employee_pkg.get_department(100) FROM DUAL;
```