Procedimientos y Funciones PL/SQL

INTRODUCCION

- Los procedimientos y funciones son subprogramas que pueden ser invocados por los usuarios.
- En PL/SQL el desarrollador puede definir sus propios subprogramas o bien utilizar funciones predefinidas en PL/SQL.
- Los subprogramas pueden ser de dos tipos:
  - Funciones: subprogramas que devuelven un valor
  - Procedimientos: subprogramas que ejecutan una secuencia de instrucciones pero que el nombre del subprograma en si mismo no devuelve un valor.

FUNCIONES PREDEFINIDAS PL/SQL

PL/SQL proporciona un gran número de funciones muy útiles para ayudar a manipular la información y permite incorporar en sus expresiones casi todas las funciones disponibles en SQL. Se pueden agrupar en categorías:

- caracteres
- numéricas
- fechas
- conversión de tipo de datos
- manejo de nulos
- misceláneas
- error-reporting

Las funciones de agrupación de SQL como por ejemplo AVG, MIN, MAX, COUNT, SUM, STDDEV, y VARIANCE, no están implementadas en PL/SQL, sin embargo se pueden usar en sentencias SQL. Tampoco se pueden usar algunas otras como DECODE, DUMP, y VSIZE.

A continuación vamos a ver algunas de las más utilizadas.

FUNCIONES PREDEFINIDAS CARACTERES

- LENGTH: Devuelve la longitud de un tipo CHAR.

```
resultado := LENGTH('HOLA MUNDO’); -- Devuelve 10
```

- INSTR

Busca una cadena de caracteres (la que se indica en el segundo parámetro pasado) dentro de otra (la que se indica en el primer parámetro) y devuelve la posición de la ocurrencia de la cadena buscada dentro de la cadena. En el tercer parámetro se indica la posición desde la que se comienza a buscar (opcional) y en el cuarto el número de ocurrencia que se busca (opcional).

Su sintaxis es la siguiente: INSTR(<char>, <search_string>, <startpos>, <occurrence>)

```
resultado := INSTR('AQUI ES DONDE SE BUSCA', 'BUSCA', 1, 1); -- Devuelve 18
```

FUNCIONES PREDEFINIDAS CARACTERES

- REPLACE: Reemplaza un texto por otro en una cadena de caracteres.

REPLACE(<expresion>, <busqueda>, <reemplazo>)

El siguiente ejemplo reemplaza la palabra 'HOLA' por 'VAYA' en la cadena 'HOLA MUNDO'.

```
resultado := REPLACE ('HOLA MUNDO','HOLA', 'VAYA'); -- devuelve VAYA MUNDO
```

- SUBSTR: Obtiene una parte de una cadena de caracteres, desde una posición de inicio hasta una determinada longitud.

SUBSTR(<expresion>, <posicion_ini>, <longitud>)

```
resultado := SUBSTR('HOLA MUNDO', 6, 5); -- Devuelve MUNDO
```

FUNCIONES PREDEFINIDAS CARACTERES

- UPPER: Convierte una cadena alfanumérica a mayúsculas.

```
resultado := UPPER('hola mundo'); -- Devuelve HOLA MUNDO
```

- LOWER: Convierte una cadena alfanumérica a minúsculas.

```
resultado := LOWER('HOLA MUNDO'); -- Devuelve hola mundo
```

- RTRIM: Elimina los espacios en blanco a la derecha de una cadena de caracteres.

```
resultado := RTRIM ('Hola Mundo ');
```

- LTRIM: Elimina los espacios en blanco a la izquierda de una cadena de caracteres.

```
resultado := LTRIM (' Hola Mundo');
```

- TRIM: Elimina los espacios en blanco a la izquierda y derecha de una cadena de caracteres.

```
resultado := TRIM (' Hola Mundo ');
```

FUNCIONES PREDEFINIDAS NUMERICAS

- MOD: Devuelve el resto de la división entera entre dos números.

MOD(<dividendo>, <divisor>)

```
resultado := MOD(20,15); -- Devuelve el modulo de dividir 20/15
```

- TRUNC: Trunca un número y devuelve la parte entera.

```
resultado := TRUNC(9.99); -- Devuelve 9
```

- ROUND: Devuelve el entero más próximo.

```
resultado := ROUND(9.99); -- Devuelve 10
```

FUNCIONES PREDEFINIDAS FECHAS

- SYSDATE: Devuelve la fecha del sistema.

```
resultado := SYSDATE;
```

- TRUNC: Trunca una fecha, elimina las horas, minutos y segundos de la misma.

```
resultado := TRUNC(SYSDATE);
```

FUNCIONES PREDEFINIDAS CONVERSION TIPO DATOS

- TO_DATE: Convierte una expresión al tipo fecha.

TO_DATE(<expresion>, [<formato>])

El parámetro opcional formato indica el formato de entrada de la expresión no el de salida.

En este ejemplo se convierte la cadena de caracteres '01/12/2006' a una fecha (tipo DATE). El formato indica que la fecha está escrita como día/mes/año, de forma que la fecha sea el uno de diciembre y no el doce de enero.

```
resultado := TO_DATE('01/12/2006', 'DD/MM/YYYY');
```

El siguiente ejemplo muestra la conversión con formato de día y hora.

```
resultado := TO_DATE('31/12/2006 23:59:59', 'DD/MM/YYYY HH24:MI:SS');
```

FUNCIONES PREDEFINIDAS CONVERSION TIPO DATOS

- TO_CHAR: Convierte una expresión al tipo CHAR.

TO_CHAR(<expresion>, [<formato>])

El parámetro opcional formato indica el formato de salida de la expresión.

```
resultado := TO_CHAR(SYSDATE, 'DD/MM/YYYY HH24:MI:SS');
```

- TO_NUMBER: Convierte una expresión alfanumérica en numérica, se puede especificar el formato de salida (opcional).

TO_NUMBER(<expresion>, [<formato>])

```
resultado := TO_NUMBER ('10.21', ’99.99’); -- resultado: 10,21 (el separador decimal es ,)
```

FUNCIONES PREDEFINIDAS MANEJO DE NULOS

- NVL: Devuelve el valor recibido como parámetro en el caso de que expresión sea NULL o el valor de la expresión en caso contrario.

NVL(<expresion>, <valor>)

El siguiente ejemplo devuelve 0 si el precio es nulo, y el precio cuando está informado:

```
SELECT CO_PRODUCTO, NVL(PRECIO, 0) FROM PRECIOS;
```

FUNCIONES PREDEFINIDAS MISCELANEAS

- DECODE: Proporciona la funcionalidad de una sentencia de control de flujo if-elseif-else.

DECODE(<expr>, <cond1>, <val1>[, ..., <condN>, <valN>], <default>)

Esta función evalúa una expresión "<expr>", si se cumple la primera condición "<cond1>" devuelve el valor1 "<val1>", en caso contrario evalúa la siguiente condición y así hasta que una de las condiciones se cumpla. Si no se cumple ninguna condición se devuelve el valor por defecto (el último parámetro).

Es muy común escribir la función DECODE indentada como si se tratase de un bloque IF.

```
SELECT DECODE (co_pais, /* Expresion a evaluar */
               'ESP', 'ESPAÑA', /* Si co_pais = 'ESP' ==> 'ESPAÑA' */
               'MEX', 'MEXICO', /* Si co_pais = 'MEX' ==> 'MEXICO' */
               'PAIS '||co_pais)/* ELSE ==> concatena */
FROM PAISES;
```

FUNCIONES PREDEFINIDAS

Carácter	Numéricas	Fecha	Conversión	Manejo Nulos	Misceláneas	Error
ASCII	ABS	ADD_MONTHS	CHARTOROWID	NVL	DECODE	SQLCODE
CHR	ACOS	CURRENT_DATE	CONVERT		DUMP	SQLERRM
CONCAT	ASIN	CURRENT_TIMESTAM	HEXTORAW		GREATEST	
INITCAP	ATAN	LAST_DAY	NLS_CHARSET_ID		GREATEST_LB	
INSTR	ATAN2	LOCALTIMESTAMP	NLS_CHARSET_NAME		LEAST	
INSTRB	CEIL	MONTHS_BETWEEN	RAWTOHEX		LEAST_UB	
LENGTH	COS	NEW_TIME	ROWIDTOCHAR		UID	
LENGTHB	COSH	NEXT_DAY	TO_CHAR		USER	
LOWER	EXP	ROUND	TO_DATE		USERENV	
LPAD	FLOOR	SYSDATE	TO_LABEL		VSIZE	
LTRIM	LN	SYSTIMESTAMP	TO_MULTI_BYTE			
NLS_INITCAP	LOG	TRUNC	TO_NUMBER			
NLS_LOWER	MOD		TO_SINGLE_BYTE			
NLS_UPPER	POWER					
NLSSORT	ROUND					
REPLACE	SIGN					
RPAD	SIN					
RTRIM	SINH					
SOUNDEX	SQRT					
SUBSTR	TAN					
SUBSTRB	TANH					
TRANSLATE	TRUNC					
UPPER						

PROCEDIMIENTOS Y FUNCIONES DEFINIDOS POR EL DESARROLLADOR

Los bloques de código anónimos BEGIN .. END, son un mecanismo básico para la programación en PL/SQL, pero no están orientados a la reutilización de SCRIPTS. Por ejemplo, en caso de que se tenga un algoritmo para algún cálculo según determinados parámetros tendríamos que repetirlo cuantas veces sea necesario.

El uso de procedimientos en PL/SQL es un buen mecanismo para la reutilización de código, además de que permite dividir el código en partes funcionales más pequeñas.

Los procedimientos pueden ser declarados en bloques anónimos o almacenarnos en la base de datos.

DECLARACION DE PROCEDIMIENTOS

La creación de un procedimiento en PL/SQL es similar a la creación de un bloque anónimo. La sintaxis de un procedimiento es la siguiente:

```
PROCEDURE nom_proc[(param1[,param2 ...])]
IS
  declaraciones locales;
BEGIN
  sentencias;
[EXCEPTION
         tratamiento_de_excepciones]
END [nom_proc];
```

- nom_proc: Es el nombre del procedimiento, se usará para identificarlo.
- param: son como variables, contienen datos que se pueden especificar en el momento de llamar al procedimiento.
- declaraciones locales: Como en un bloque anónimo se pueden crear variables que sólo pueden ser usadas en código dentro del procedimiento.
- sentencias: Es el código que se ejecuta cuando se llama al procedimiento y que puede hacer uso de las variables declaradas así como de los parámetros.

DECLARACION DE PROCEDIMIENTOS

El IS es el equivalente a DECLARE en los bloques anónimos.

En el IS sí debemos indicar la longitud de las variables locales.

PARAMETROS DE LOS PROCEDIMIENTOS

Tienen la siguiente sintaxis:

Nom_param [IN|OUT|IN OUT] tipo_dato[{:=|DEFAULT }Valor]

- Cuando no se indica, los parámetros se definen por defecto de tipo IN.
- En tipo_dato sólo se especifica el tipo, sin indicar su longitud ni restricciones.
- Si un procedimiento no tienen parámetros, no es necesario poner los paréntesis en la cabecera.

- Un parámetro de entrada permite que pasemos valores al subprograma y no puede ser modificado en el subprograma. El parámetro pasado puede ser una constante o una variable.
- Un parámetro de salida permite devolver valores y en el subprograma actúa como variable no inicializada. El parámetro pasado debe ser una variable.
- Un parámetro de entrada-salida se utiliza para pasar valores al subprograma y/o para recibirlos, por lo que un parámetro formal que actúe como parámetro pasado debe ser una variable.

PROCEDIMIENTOS DENTRO DE UN BLOQUE ANONIMO

En estos casos el procedimientos se debe crear dentro de la sección DECLARE ... BEGIN.

```
DECLARE
-- El procedimiento debe ser declarado dentro de la sección DECLARE .. BEGIN
  PROCEDURE registrar_cliente(P_ID NUMBER,
                              P_NOMBRE VARCHAR2,
                              P_APELLIDOS VARCHAR2)
  IS
    declaraciones locales;
  BEGIN
    sentencias;
  [EXCEPTION
           tratamiento_de_excepciones]
  END [nom_proc];
BEGIN
-- Sentencias, código de bloque anónimo
 ...
  REGISTRAR_CLIENTE(1,'Juan', 'Rosales');
  REGISTRAR_CLIENTE(2,'Luis', 'Cabrera');
 ...
END;
/
```

La declaración de procedimientos debe ir al final de la sección DECLARE correspondiente.

PROCEDIMIENTOS ALMACENADOS

PL/SQL permite almacenar los procedimientos para ser usados desde cualquier bloque anónimo (sin que haya la necesidad de declararlo) y también desde otros procedimientos.

Para crear un procedimiento almacenado debemos poner la palabra reservada CREATE y ejecutar el código como si se tratase de un bloque PL/SQL.

El procedimiento almacenado es compilado previamente por el motor PL/SQL y si no da errores quedará almacenado y se podrá llamar.

```
CREATE PROCEDURE registrar_cliente (P_ID NUMBER,
                                    P_NOMBRE VARCHAR2,
                                    P_APELLIDOS VARCHAR2)
IS
  declaraciones locales;
BEGIN
  sentencias;
[EXCEPTION
         tratamiento_de_excepciones]
END [nom_proc];
```

PROCEDIMIENTOS ALMACENADOS

Adicionalmente, se puede añadir las palabras reservadas **OR REPLACE** para evitar errores al intentar compilar un procedimiento que ya ha sido compilado :

CREATE OR REPLACE PROCEDURE REGISTRAR_CLIENTE….

Con esta sentencia creamos un procedimiento. Si ya existía lo reemplaza.

En el caso de los procedimientos almacenados ya no es necesario declarar el procedimiento dentro de la sección DECLARE .. BEGIN de los bloques anónimos.

Un procedimiento **se puede invocar desde un** bloque u otro procedimiento/ función de PL/SQL **llamándolo** simplemente **por su nombre** y pasándole los parámetros.

PROCEDIMIENTOS ALMACENADOS

Se puede invocar un procedimiento también desde SQL*PLUS:

Sql> **execute** registrar_cliente (7902,'Antonio','Alvarez Sánchez');

Si alguno de los parámetros fuera de salida (OUT o IN OUT) se debe invocar con una variable que debe ser definida previamente:

```
// Creación del procedimiento, el segundo parámetro es de salida
CREATE OR REPLACE PROCEDURE calcular_cuadrado_procedure(P_NUMERO NUMBER, P_CUADRADO OUT NUMBER)
IS
BEGIN
  P_CUADRADO := P_NUMERO*P_NUMERO;
END calcular_cuadrado_procedure;
```

```
// Llamada al procedimiento desde consola SQL*PLUS
SQL> var num_cuadrado NUMBER -- Se define la variable Host necesaria para parámetro de salida

SQL> exec calcular_cuadrado_procedure(5,:num_cuadrado) -- Llamada a método, : antes de variable Host

SQL> PRINT num_calculo -- Muestra por pantalla la variable Host
```

METODOS DE PASO DE PARAMETROS

Notación Posicional: Se pasan los valores de los parámetros en el mismo orden en que el procedure los define.

```
BEGIN
  REGISTRAR_CLIENTE(1,'Juan', 'Rosales');
END;
```

Notación Nominal: Se pasan los valores en cualquier orden nombrando explícitamente el parámetro y su valor separados por el símbolo =>.

```
BEGIN
  REGISTRAR_CLIENTE(P_ID => 1,P_NOMBRE => 'Juan', P_APELLIDOS => 'Rosales');
END;
```

Notación Mixta: Combina las dos anteriores.

```
BEGIN
  REGISTRAR_CLIENTE(1,P_NOMBRE => 'Juan', P_APELLIDOS => 'Rosales');
END;
```

DECLARACION DE FUNCIONES

La creación de una función tiene una sintaxis similar a la de un procedimiento:

```
FUNCTION nom_funcion([param1[,param2 ...]])
RETURN [tipo de valor devuelto]
IS
  declaraciones locales;
BEGIN
  sentencias;
  RETURN(expresión);
[EXCEPTION
         tratamiento_de_excepciones]
END [nom_funcion];
```

- Los parámetros tienen la misma sintaxis que en los procedimientos.
- La cláusula RETURN de la cabecera indica el tipo de datos que devuelve la función.
- La cláusula RETURN del cuerpo hace efectivo ese retorno.

FUNCIONES ALMACENADAS

Como para los procedimientos, para crear una función almacenada:

```
CREATE OR REPLACE FUNCTION nom_funcion(([param1[,param2 ...]])
RETURN [tipo de valor devuelto]
IS
   declaraciones locales;
BEGIN
   sentencias;
   RETURN(expresión);
[EXCEPTION
        tratamiento_de_excepciones]
END [nom_funcion];
```

En el caso de las funciones almacenadas ya no es necesario declararla dentro de la sección DECLARE .. BEGIN de los bloques anónimos.

LLAMADAS A FUNCIONES

Una función **se puede invocar desde un** bloque u otro procedimiento / función de PL/SQL **llamándola** simplemente **por su nombre** y pasándole los parámetros requeridos, asignando el valor (mediante :=) a una variable del mismo tipo que devuelve la función:

```
BEGIN
  ...
  num_calculo := calcular_cuadrado(3);
  ...
END;
```

Se puede invocar también desde SQL*PLUS

- Consulta genérica:

```
SQL> select year_of_date(start_date) FROM DUAL;
```

- Utilizando exec y variables Host:

```
SQL> var num_calculo NUMBER -- Se define la variable de Host necesaria para asignar valor a la función
SQL> exec :num_calculo := calcular_cuadrado(3) -- Llamada a la función, usar : antes de variable Host

SQL> PRINT num_calculo -- Muestra por pantalla la variable Host
```

PROCEDIMIENTOS Y FUNCIONES

PL/SQL permite la **sobrecarga** en los nombres de subprogramas (aplica a procedimientos y funciones), es decir, podemos llamar a dos subprogramas con el mismo nombre y los distingue porque sus parámetros deben tener o distinto número o distintos tipo. La sobrecarga de los subprogramas se usa generalmente cuando conceptualmente se ejecuta la misma tarea (o similar) pero con un conjunto de parámetros ligeramente diferente (o con diferente definición).

También permite **programación recursiva** (ejemplo típico cálculo del factorial de un número).

PROCEDIMIENTOS Y FUNCIONES

Se pueden conocer los procedimientos y funciones definidos mediante la siguiente consulta

```
SELECT object_name FROM user_procedures;
```

Se puede eliminar un procedimiento ejecutando

```
drop procedure nombre_procedimiento;
```

Se puede eliminar una función ejecutando

```
drop function nombre_función;
```