INTRODUCCIÓN AL LENGUAJE PL/SQL

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.

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.

TIPOS DE DATOS

PL/SQL dispone de los mismos tipos de datos que SQL, además de otros propios. En la mayoría de los casos existe compatibilidad con los tipos correspondientes soportados por la base de datos pero también existen diferencias que deben tenerse en cuenta.

Podemos clasificar los tipos de datos soportados por PL/SQL en las siguientes categorías:

- **Escalares:** almacenan valores simples. A su vez pueden subdividirse en:
  - *Carácter/Cadena*: CHAR, NCHAR, VARCHAR2, NVARCHAR2, LONG, RAW, LONG RAW, ROWID, UROWID.
  - *Numérico*: NUMBER, BINARY_INTEGER, PLS_INTEGER, BINARY_DOUBLE, BINARY_FLOAT.
  - *Booleano*.
  - *Fecha/hora*: DATE, TIMESTAMP, INTERVAL.
  - *Otros*: ROWID, UROWID.
- **Compuestos:** Son tipos compuestos de otros simples. Los veremos en la unidad de programación avanzada. Por ejemplo:
  - *Tablas indexadas*.
  - *Tablas anidadas*.
  - *Varrays*.
  - *Objetos*.
- **Referencias:** Difieren de los demás por sus características de manejo y almacenamiento.
  - *REF CURSOR:* Son referencias a cursores.
  - *REF*: Son punteros a objetos.

En PL/SQL el programador puede definir sus propios tipos a partir de los anteriores.

TIPOS ESCALARES

CARÁCTER

Almacena **cadenas** de caracteres cuya longitud máxima es 32 KBytes. Al especificar las longitudes debemos tener en cuenta que, por defecto, cada carácter ocupa 1 byte, aunque hay juegos de caracteres que ocupan más (por ejemplo, el asiático).

CHAR[(L)]
 	
NCHAR[(L)]
	Almacena cadenas de caracteres de **longitud fija.** Opcionalmente se puede especificar, entre paréntesis, la longitud máxima; en caso contrario, el valor por defecto es 1 y no se pone el paréntesis. Las posiciones no utilizadas **se rellenan con blancos.**
	Ejemplo:
	`Nombre CHAR(15);`
	`Letra_nif CHAR;`
	`Situacion CHAR();--ERROR`
	
	La longitud de **CHAR** se expresa por defecto en **bytes** pero se puede expresar en caracteres del juego de caracteres nacional utilizando la opción CHAR:
	
	`Apellido CHAR(25 CHAR);`
	
	La longitud en **NCHAR** se expresa siempre en caracteres del juego de **caracteres** nacional. Equivale a usar la opción CHAR comentada arriba:
	
	`Apellido NCHAR(25); -- equivalente a la anterior`

VARCHAR2(L)
 	
NVARCHAR2(L)
 	
VARCHAR(L)
	Almacena cadenas de caracteres de **longitud variable** cuyo límite máximo se debe especificar.
	Ejemplos:
	`Apellidos VARCHAR2(25);`
	`Control VARCHAR2; --ERR falta la longitud`
	`Nombre VARCHAR2(15 CHAR);`
	`Nombre NVARCHAR2(25);`
	
	También está disponible el tipo VARCHAR por compatibilidad con el estándar SQL, aunque Oracle desaconseja su utilización recomendando utilizar en su lugar VARCHAR2 y NVARCHAR2 para beneficiarse de la evolución de estos tipos.

LONG[(L)]	Almacena cadenas de longitud variable. Es similar a VARCHAR2 pero no permite la opción CHAR.
	Ejemplos:
	`v_Observaciones LONG(5000);`
	`Resumen LONG;`
	
	Debemos tener en cuenta que el límite para este tipo en variables PL/SQL es de 32 KB, pero las columnas de este tipo admiten hasta 2GB.

NUMÉRICOS	

NUMBER(P, E)	Donde *P* es la precisión (número total de dígitos) y *E* es la escala (número de decimales).
	La precisión y la escala son opcionales, pero si se especifica la escala hay que indicar la precisión.
	Ejemplo:
	`Importe NUMBER(5,2); -- admite hasta 999.99`
	`Centimos NUMBER(,2); -- ERROR lo correcto es`
	`Centimos NUMBER(2,2); -- admite hasta .99`
	
	Si al asignar un valor se excede la escala, se producirá *truncamiento*. Si se excede la precisión, se producirá un *error*.
	Ejemplo:
	`Importe:= 1000; -- ERROR excede la escala`
	`Importe:= 234.344; -- guarda 234.34`
	
	PL/SQL dispone de subtipos de NUMBER que se utilizan por compatibilidad y/o para establecer restricciones. Son DECIMAL, NUMERIC, INTEGER, REAL, SMALLINT, etcétera.

BINARY_INTEGER	Es un tipo numérico entero que se almacena en memoria en formato binario para facilitar los cálculos. Puede contener valores comprendidos entre –2147483647 y + 2147483647. Se utiliza en contadores, índices, etcétera. Por ejemplo: `Contador BINARY_INTEGER;` Subtipos: NATURAL y POSITIVE.
PLS_INTEGER	Similar a BINARY_INTEGER y con el mismo rango de valores, pero tiene dos ventajas respecto al anterior: • Es más rápido. • Si se da desbordamiento en el cálculo se produce un error y se levanta la excepción correspondiente, lo cual no ocurre con BINARY_INTEGER. Por ejemplo: `indice PLS_INTEGER;`
BINARY_DOUBLE BINARY_FLOAT	Disponibles desde la versión 10gR1, su utilización se circunscribe al ámbito científico para realizar cálculos muy precisos y complejos conforme a la norma IEEE 754. Oracle no los soporta directamente como columnas en la base de datos. No los utilizaremos en este libro. Remitimos al lector a la documentación de Oracle para su eventual utilización.
BOOLEANO	

BOOLEAN	Almacena valores lógicos TRUE, FALSE y NULL. Para representar estos valores **no debemos utilizar comillas.**
	Ejemplo:
	`v_ocupado BOOLEAN;`
	`v_eliminado BOOLEAN DEFAUL FALSE;`
	
	La base de datos no soporta este tipo para definir columnas.

FECHA / HORA Almacenan valores de tiempo. Son idénticos a los tipos correspondientes de la base de datos.	
DATE	Almacena fechas incluyendo la hora. Los formatos en que muestran la información son los establecidos por defecto, aunque se pueden especificar máscaras de formato. Por defecto, sólo se muestra la fecha, pero también se almacena la hora: `día/mes/año horas:minutos:segundos`.
TIMESTAMP [(P)]	**TIMESTAMP** también almacena fecha y hora pero incluyendo fracciones de segundo. La precisión de estas fracciones se puede especificar como parámetro (por defecto es 6, es decir, 999999 millonésimas de segundo, pero puede variar entre 0 y 9). Normalmente el valor se toma de la variable del sistema SYSTIMESTAMP.
TIMESTAMP [(P)] **WITH TIME ZONE**	**TIMESTAMP WITH TIME ZONE** incluye el valor de la **zona horaria.**

TIMESTAMP [(P)] **WITH LOCAL TIME ZONE**

TIMESTAMP WITH LOCAL TIME ZONE igual que TIMESTAMP, pero:

- Los datos **se almacenan normalizados a la zona horaria donde se encuentra la base de datos.**
- Los datos recuperados **se muestran en el formato de la zona horaria correspondiente a la sesión** de los usuarios.

El siguiente ejemplo ilustra estos conceptos:

```
DECLARE
  fechaTI TIMESTAMP;
  fechaTZ TIMESTAMP WITH TIME ZONE;
  fechaTL TIMESTAMP WITH LOCAL TIME ZONE;
BEGIN /* incluiremos dos instrucciones en la misma linea */
  fechaTI := SYSTIMESTAMP;   DBMS_OUTPUT.PUT_LINE('fechaTI:'||fechaTI);
  fechaTZ := SYSTIMESTAMP;   DBMS_OUTPUT.PUT_LINE('fechaTZ:'||fechaTZ);
  fechaTL := SYSTIMESTAMP;   DBMS_OUTPUT.PUT_LINE('fechaTL:'||fechaTL);
END;
```

El resultado de la ejecución será:

```
fechaTI:02/04/06 10:07:40,196000
fechaTZ:02/04/06 10:07:40,196000 +02:00
fechaTL:02/04/06 10:07:40,196000
```

En este caso, FechaTL tiene el mismo valor que FechaTI porque la zona horaria de almacenamiento y recuperación es la misma.

INTERVAL YEAR [(PY)] **TO MONTH**

INTERVAL DAY [(PD)] **TO SECOND** [(PS)]

Los subtipos de INTERVAL representan intervalos de tiempo entre dos fechas. La diferencia entre ellos es la manera de expresar el intervalo:

- INTERVAL YEAR TO MONTH se expresa en años/meses. Opcionalmente se puede incluir la precisión para el número de años, entre 0 y 9 dígitos, por defecto es 2.

```
v_anios_meses2 INTERVAL YEAR TO MONTH := '1-06';
```

- INTERVAL DAY TO SECOND se expresa en días/segundos. Opcionalmente se puede incluir la precisión para el número de días (entre 0 y 9 dígitos, por defecto es 2), y para el número de segundos (entre 0 y 9 dígitos, por defecto es 6).

```
v_dias_segundos INTERVAL DAY TO SECOND DEFAULT '02 14:0:0.0';
```

Estos tipos se utilizan cuando se requiere manejar diferencias de tiempo con precisiones expresadas en fracciones de segundo. En la mayoría de los casos recurriremos a los tipos tradicionales junto con las funciones disponibles MONTHS_BETWEEN, etcétera.

OTROS TIPOS ESCALARES

RAW(L)	Almacena datos binarios en longitud fija. Se utiliza para almacenar cadenas de caracteres evitando problemas por las conversiones entre conjuntos de caracteres que realiza Oracle.
LONG RAW	Almacena datos binarios en longitud variable evitando conversiones entre conjuntos de caracteres.
ROWID **UROWID**[(L)]	Almacenan identificadores de direcciones de fila. Son idénticos a los tipos correspondientes de la base de datos. UROWID permite especificar la longitud de almacenamiento en número de bytes (4000 bytes es el máximo y el valor por defecto).

IDENTIFICADORES

Los **identificadores** se utilizan para nombrar los objetos que intervienen en un programa: variables, constantes, cursores, excepciones, procedimientos, funciones, etiquetas, etcétera.

En PL/SQL deben cumplir las siguientes características:

- Pueden tener entre 1 y 30 caracteres de longitud.
- El primer carácter debe ser una letra.
- Los restantes caracteres deben ser caracteres alfanuméricos o signos admitidos (letras, dígitos, los signos de dólar, almohadilla y subguión).
- No pueden incluir signos de puntuación, espacios, etcétera.

Se pueden saltar algunas de estas reglas utilizando identificadores entre comillas dobles (por ejemplo `"2ªvaria ble/"`) pero no es aconsejable.

Ejemplos de identificadores válidos	`v_`	`A#$`	`X2`	`anio`	`v_num.`
Ejemplos de identificadores no válidos	`_v`	`#A`	`2X`	`año`	`v-num.`

PL/SQL no diferencia entre mayúsculas y minúsculas en los identificadores ni en las palabras reservadas. Por ejemplo, podemos escribir `V_Num_Meses`, `v_num_meses` o `V_NUM_MESES`. En los tres casos se trata del mismo identificador.

VARIABLES

Las **variables** sirven para almacenar información cuyo valor puede cambiar a lo largo de la ejecución del programa.

A. Declaración e inicialización de variables

Todas las variables PL/SQL deben declararse en la *sección declarativa* antes de su uso. El formato genérico para declarar una variable es el siguiente:

```
<nombre_de_variable> <tipo> [NOT NULL] [{:= | DEFAULT}
                        <valor>];
```

La opción **DEFAULT** (o bien la asignación «:=») sirve para asignar valores por defecto a la variable desde el momento de su creación.

```
DECLARE
     importe NUMBER(8,2);
     contador NUMBER(2,0) := 0;
     nombre CHAR(20) NOT NULL := "MIGUEL";
...
```

Para cada variable se debe especificar el *tipo*. No se puede, como en otros lenguajes, indicar una lista de variables del mismo tipo y, a continuación, el tipo. PL/SQL no lo permite.

La opción **NOT NULL** fuerza a que la variable tenga siempre un valor. Si se usa, deberá inicializarse la variable en la declaración con DEFAULT o con := para la inicialización. Por ejemplo:

```
nombre CHAR(20) NOT NULL DEFAULT "MIGUEL";
```

También se puede inicializar una variable que no tenga la opción NOT NULL. Por ejemplo:

```
nombre CHAR(20) DEFAULT "MIGUEL";
```

Si no se inicializan las variables en PL/SQL, se garantiza que su valor es NULL.

B. Uso de los atributos %TYPE y %ROWTYPE

En lugar de indicar explícitamente el tipo y la longitud de una variable existe la posibilidad de utilizar los atributos %TYPE y %ROWTYPE para declarar variables que sean del mismo tipo que otros objetos ya definidos.

- **%TYPE** declara una variable del mismo tipo que otra, o que una columna de una tabla. Su formato es:

```
nombre_variable  nombre_objeto%TYPE;
```

- **%ROWTYPE** declara una variable de registro cuyos campos se corresponden con las columnas de una tabla o vista de la base de datos. Su formato es:

```
nombre_variable  nombre_objeto%ROWTYPE;
```

Ejemplos:

- `total importe%TYPE;`

  Declara la variable `total` del mismo tipo que la variable `importe` que se habrá definido previamente.

- `nombre_moroso clientes.nombre%TYPE;`

  Declara la variable `nombre_moroso` del mismo tipo que la columna `nombre` de la tabla `clientes`.

- `moroso clientes%ROWTYPE;`

  Declara la variable `moroso` que podrá contener una fila de la tabla `clientes`.

Para hacer referencia a cada uno de los campos indicaremos el nombre de la variable, un punto y el nombre del campo que coincide con el de la columna correspondiente. Por ejemplo:

```
DBMS_OUTPUT.PUT_LINE(moroso.nombre);
```

Al declarar una variable del mismo tipo que otro objeto usando los atributos %TYPE y %ROWTYPE, se hereda el tipo y la longitud, pero no los posibles atributos NOT NULL ni los valores por defecto que tuviese definidos el objeto original.

Las **variables** se crean al comienzo del bloque y dejan de existir una vez finalizada la ejecución del bloque en el que han sido declaradas. El ámbito de una variable incluye el bloque en el que se declara y sus bloques «hijos».

Una variable declarada en un bloque será **local** para el bloque en el que ha sido declarada y **global** para los bloques hijos de éste. Las variables declaradas en los bloques hijos no son accesibles desde el bloque padre.

En el siguiente ejemplo, podemos observar que `v1` es accesible para los dos bloques (es local al bloque padre y global para el bloque hijo), mientras que `v2` solamente es accesible para el bloque hijo.

```
DECLARE      -------------------------Bloque PADRE
    v1 CHAR;
BEGIN
    v1 := 27;

    DECLARE  ------------------Bloque HIJO
        v2 CHAR;
    BEGIN
        v2:= 2;
        v1:= v2;
        ...
    END;  ---------------------Fin bloque HIJO

    v2:= v1; --> Error: v2 no existe en este ámbito.

END;  --------------------------------Fin bloque PADRE
```

En el caso de que un identificador local coincida con uno global, si no se indica más, se referencia el local. Es decir, el identificador local dentro de su ámbito oculta la visibilidad del global. No obstante, se pueden utilizar etiquetas y cualificadores para deshacer ambigüedades.

```
<<padre>>    -- etiqueta que identifica al bloque padre.
DECLARE
    v CHAR;
BEGIN
    ...
    DECLARE
        v CHAR;
    BEGIN
        ...
        v := padre.v;-- el primero es el identificador local
        ...
    END;
END;
```

En caso de coincidencia, los identificadores de columnas tienen precedencia sobre las variables y parámetros formales; éstos, a su vez, tienen precedencia sobre los nombres de tablas.

CONSTANTES Y LITERALES

Para representar valores que no variarán a lo largo de la ejecución del programa disponemos de dos elementos: *constantes* y *literales*.

Aunque PL/SQL no diferencia entre mayúsculas y minúsculas en los identificadores, sí lo hace en los literales y con el contenido de las variables y constantes (tal como ocurre en SQL).

A. Constantes

Las declaramos con el siguiente formato:

```
<nombre_de_constante> CONSTANT <tipo> := <valor>;
```

Cuando declaramos una constante siempre se deberá asignar un valor. Por ejemplo:

```
Pct_iva CONSTANT REAL := 16;
```

B. Literales

Los **literales** representan valores constantes directamente (sin recurrir a identificadores). Se utilizan para visualizar valores, hacer asignaciones a variables, constantes u otros objetos del programa. Pueden ser de varios tipos:

Carácter

Constan de un único carácter alfanumérico o especial introducido entre comillas simples. Ejemplos : `'A', 'a', 'j', '5', '%', '*'`

Debemos ser muy cuidadosos con estos delimitadores, especialmente si utilizamos editores distintos del SQL*Plus pues muchos editores inducen a confusión o tienden a cambiar estos signos.

Cadena

Son conjuntos de caracteres introducidos entre comillas simples. Ejemplos: `'Hola Mundo', 'Cliente Nº: ', 'Introduzca un valor...'`

En ocasiones necesitaremos incluir el carácter utilizado como delimitador en la cadena o carácter que queremos representar. Por ejemplo: `DBMS_OUTPUT.PUT_LINE('Peter's hotel');`.

Dará el error: `ORA-01756: quoted string not properly terminated.`

En estos casos emplearemos dos comillas simples en el lugar donde debe aparecer una:

```
DBMS_OUTPUT.PUT_LINE('Peter''s hotel'); END;
```

El resultado será el deseado: `Peter's hotel.`

Numérico

Representan valores numéricos enteros o reales. Se pueden representar con notación científica. Ejemplos: `4, -7, 567.45, -458.456, 2.234E4, 5.25e-5, -8,75E+2`

Booleano

Representan los valores lógicos verdadero, falso y nulo. Se representan mediante las palabras **TRUE, FALSE** y **NULL** escritas directamente en el código y sin comillas (no son literales sino palabras reservadas del lenguaje). Se pueden utilizar para asignar valores a variables y constantes, o para comprobar el resultado de expresiones lógicas. Ejemplos: `v_cobrado := TRUE;  v_enviado := FALSE;`

Fecha/hora

Se utilizan para representar valores de fecha y hora según los distintos formatos estudiados anteriormente. Se representan mediante la expresión que indica el tipo DATE o TIMESTAMP seguida de la cadena delimitada que indica el valor que queremos representar. Por ejemplo: `DATE '2005-11-09'   TIMESTAMP '2005-11-09 13:50:00'.`

También podemos representar valores correspondientes a los **subtipos TIMESTAMP WITH TIME ZONE** y **TIMESTAMP WITH LOCAL TIME ZONE** utilizando la palabra TIMESTAMP y la cadena en el formato correspondiente al subtipo que queremos representar como se muestra en los siguientes ejemplos:

```
TIMESTAMP '2005-11-09 13:50:00 +06:00' (para TIMESTAMP WITH
TIME ZONE)

TIMESTAMP '2005-11-09 13:50:00' (para TIMESTAMP WITH LOCAL
TIME ZONE)
```

OPERADORES Y DELIMITADORES

Los operadores se utilizan para asignar valores y formar expresiones. PL/SQL dispone de operadores de:

- **Asignación.** := Asigna un valor a una variable.
  Por ejemplo: `Edad := 19;`
- **Concatenación.** || Une dos o más cadenas.
  Por ejemplo: `'buenos' || 'días'` dará como resultado `'buenosdías'`.
- **Comparación.** `=, !=, <, >, <=, >=, IN, IS NULL, LIKE, BETWEEN,...`
  Funcionan igual que en SQL.
- **Aritméticos.** `+, -, *, /, **`. Se emplean para realizar cálculos. Algunos de ellos se pueden utilizar también con fechas.

  f1 - f2. Devuelve el número de días que hay entre las fechas *f1* y *f2*.

  f + n. Devuelve una fecha que es el resultado de sumar *n* días a la fecha *f*.

f - n. Devuelve una fecha que es el resultado de restar *n* días a la fecha *f*.

- **Lógicos. AND, OR y NOT.** Permiten operar con expresiones que devuelven valores booleanos (comparaciones, etcétera). A continuación, se detallan las tablas de valores de los operadores lógicos AND, OR y NOT, teniendo en cuenta todos los posibles valores, incluida la ausencia de valor (NULL).

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Tabla 9.2. *Valores de los operadores lógicos.*

- **Otros indicadores y delimitadores.**

( )	Delimitador de expresiones.
''	Delimitador de literales de cadena.
" "	Delimitador de identificadores (utilización desaconsejable, en general).
<< >>	Etiquetas.
/* */	Delimitador de comentarios de varias líneas.
--	Indicador de comentario de una línea.
%	Indicador de atributo (TYPE, ROWTYPE, FOUND,...).
:	Indicador de variables de transferencia (*bind*).
,	Separador de ítem de lista.
;	Terminador de instrucción.
@	Indicador de enlace de base de datos.

A. Orden de precedencia en los operadores

El **orden de precedencia** o **prioridad de los operadores** determina el orden de evaluación de los operandos de una expresión. Por ejemplo, la siguiente expresión: 12+18/6, dará como resultado 15, ya que la división se realizará primero. En la siguiente tabla se muestran los operadores disponibles agrupados y ordenados de mayor a menor, por orden de precedencia.

Prioridad	Operador	Operación
1	**, NOT	Exponenciación, negación.
2	*, /	Multiplicación, división.
3	+, -, ||	Suma, resta, concatenación.
4	=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	Comparación.
5	AND	Conjunción.
6	OR	Disyunción.

Tabla 9.3. *Orden de precedencia en los operadores.*

Debemos tener cuidado con expresiones del tipo:

```
100 > x > 0  -- ilegal
```

Ya que primero se ejecutará 100 > x y devolverá un valor que será VERDADERO, FALSO o NULL, y ése será el primer término de la siguiente comparación, lo cual causará un error. Para evitarlo, podemos indicar la expresión de la siguiente forma:

```
(100 > x) AND (x > 0)
```

FUNCIONES PREDEFINIDAS

En PL/SQL se pueden utilizar todas las funciones de SQL. No obstante, algunas, como las de agrupamiento (AVG, MIN, MAX, COUNT, SUM, STDDEV, etcétera), solamente se pueden usar dentro de cláusulas de selección (SELECT).

Respecto a la utilización de funciones, también debemos tener en cuenta que:

- La función no modifica el valor de las variables o expresiones que se pasan como argumento, sino que devuelve un valor a partir de dicho argumento.
- Si a una función se le pasa un valor nulo en la llamada, normalmente devolverá un valor nulo.

Veamos un ejemplo de funciones: escribiremos un bloque PL/SQL que muestre la fecha y la hora con minutos y segundos.

```
SQL> BEGIN
  2      DBMS_OUTPUT.PUT_LINE(TO_CHAR(SYSDATE,
  3 'DAY " día " DD " de " MONTH " de" YYYY " a las ":HH24
    :MI:SS'));
  4  END;
  5  /
JUEVES    día  02  de  MARZO     de 2006  a las :13:06:10
Procedimiento PL/SQL terminado con éxito.
```

COMENTARIOS

Los comentarios se utilizan para documentar datos generales y particulares de los programas (el autor, la fecha, el objeto del programa, aspectos relevantes o hitos en el programa, función de determinados objetos o comentarios explicativos) e incluso para añadir legibilidad y organización a nuestros programas: líneas de separación, etcétera. En PL/SQL, podemos insertar comentarios en cualquier parte del programa. Estos comentarios pueden ser:

- **De línea con "--".** Todo lo que le sigue en esa línea será considerado comentario. Todos los comentarios incluidos en el código hasta el momento son de este tipo.
- **De varias líneas con "/*" <comentarios> "*/"** (igual que en C). Se pueden incluir en cualquier sección del programa. Es aconsejable, aunque no obligatorio, que incluyan una o varias líneas completas.

```
/* Este programa de prueba ha sido realizado por F. MONTERO
para documentar la realización de comentarios en PL/SQL
*/
  ------------------------------------------------------
  BEGIN    -- aquí comienza la sección ejecutable
```