Modularización de Programas Definiendo Subprogramas

Componentes de un Subprograma

Un subprograma se compone de una serie de secciones:

- una cabecera que indica el nombre y el tipo del bloque,
- una sección opcional de declaración, donde se indican los identificadores locales,
- una parte de código donde se indican las acciones,
- una sección opcional de manejo de excepciones, para tratamiento de excepciones.

Componentes de un Subprograma

```
CABECERA -- Obligatorio
    -- Nombre del subprograma, tipo y argumentos
DECLARACIONES -- Opcional
    -- Variables, constantes, cursores, excepciones definidas por el usuario.
BEGIN -- Obligatorio
    -- Sentencias SQL.
    -- Sentencias de control PL/SQL.
EXCEPTION -- Opcional
    -- Acciones que se llevarán a cabo al producirse algún error.
END; -- Obligatorio
```

Creación de Procedimientos

- Un procedimiento es un subprograma que ejecuta una acción específica.
- La sintaxis es la siguiente:

```
PROCEDURE nombre [(parámetro[,parámetro, ...])] IS
    [declaraciones locales]
BEGIN
    sentencias ejecutables
[EXCEPTION
    manejadores de excepciones]
END [nombre];
```

donde parámetro simboliza la siguiente sintaxis:

```
nombre _parámetro [IN | OUT | IN OUT] tipo_dato [{:= | DEFAULT} expresión]
```

Creación de Procedimientos

- Un procedimiento tiene dos partes: la especificación y el cuerpo.
  - La especificación del procedimiento comienza con la palabra reservada PROCEDURE y finaliza con el nombre del procedimiento o una lista de parámetros.
  - El cuerpo del procedimiento comienza con la palabra reservada IS y finaliza con la palabra reservada END seguida por el nombre del procedimiento, opcional.

Parámetros Procedurales

- Los subprogramas pasan información utilizando parámetros.
- Las variables o expresiones referenciadas en la lista de parámetros de una llamada a un subprograma son parámetros reales.
- Por ejemplo:

```
aumentar_salario(emp_num, merito + otros);
```

Parámetros Procedurales

- Las variables declaradas en una especificación de subprograma y referenciadas en el cuerpo del subprograma son parámetros formales.
- Por ejemplo, el siguiente procedimiento declara dos parámetros formales llamados emp_id e incremento:

```
PROCEDURE aumentar_salario (emp_id INTEGER, incremento REAL) IS
    salario_actual  REAL;
    ...
BEGIN
    SELECT salario INTO salario_actual FROM emp WHERE empno = emp_id;
    ...
    UPDATE emp SET salario = salario + incremento WHERE empno = emp_id;
END aumentar_salario;
```

Tipos de Parámetros Formales

- Se utilizan los tipos de parámetros para definir el comportamiento de los parámetros formales.
- Se pueden usar tres tipos de parámetros: IN (por defecto), OUT e IN OUT con cualquier subprograma.

Tipos de Parámetros Formales

- Tipo IN
  - Un parámetro IN permite pasar valores al subprograma que se está llamando.
  - En el subprograma, un parámetro IN actúa como una constante.
  - No se le puede asignar un valor.
  - El parámetro real que corresponde a un parámetro formal puede ser una constante, un literal, una variable inicializada o expresión.

Tipos de Parámetros Formales

- Tipo OUT
  - Un parámetro OUT permite devolver valores al ambiente de invocación de un subprograma.
  - En el subprograma, un parámetro OUT actúa como una variable no inicializada.
  - Su valor no puede ser asignado a otra variable o reasignado a si mismo.
  - El parámetro real que corresponde a un parámetro formal OUT debe ser una variable; no puede ser una constante o una expresión.

Tipos de Parámetros Formales

- Tipo IN OUT
  - Un parámetro IN OUT permite pasar valores iniciales a los subprogramas que se llaman y devolver valores actualizados al ambiente de invocación.
  - En el subprograma, un parámetro IN OUT actúa como una variable inicializada.
  - Se le puede asignar un valor y su valor puede ser asignado a otra variable.
  - El parámetro real que corresponde a un parámetro formal IN OUT debe ser una variable

Valores Por Defecto de Parámetros

- Se pueden inicializar parámetros IN a valores por defecto.
- De esa manera, se pueden pasar diferentes números de parámetros reales a un subprograma, aceptando o sobreescribiendo los valores por defecto como se necesite.
- Se pueden adicionar nuevos parámetros formales sin tener que cambiar cada llamada al subprograma.

Valores Por Defecto de Parámetros

```
PROCEDURE crear_dept (
    nuevo_dnombre  CHAR DEFAULT 'TEMP',
    nueva_localidad  CHAR DEFAULT 'TEMP') IS
BEGIN
    INSERT INTO dept
        VALUES (deptno_seq.NEXTVAL, nuevo_dnombre, nueva_localidad);
```

```
crear_dept;
crear_dept('MARKETING');
crear_dept('MARKETING', 'NUEVA YORK');
```

```
crear_dept(nueva_localidad => 'NUEVA YORK');
```

Creación de Funciones

- Una función es un subprograma que calcula un valor.
- La sintaxis es la siguiente:

```
FUNCTION nombre [(parámetro[, parámetro, ...])] RETURN tipo_dato IS
    [declaraciones locales]
BEGIN
    sentencias ejecutables
[EXCEPTION
    manejadores de excepciones]
END [nombre];
```

donde parámetro simboliza:

```
nombre _parámetro [IN | OUT | IN OUT] tipo_dato [{:= | DEFAULT} expresión]
```

Creación de Funciones

- Una función tiene dos partes:
  - La especificación de la función, que comienza con la palabra reservada FUNCTION y finaliza con la cláusula RETURN, que especifica el tipo de dato del valor resultado.
  - El cuerpo de la función, que comienza con la palabra reservada IS y finaliza con la palabra reservada END seguida por el nombre de la función.

La Sentencia RETURN

- La sentencia RETURN finaliza inmediatamente la ejecución de un subprograma y devuelve el control al ambiente de invocación.
- Un subprograma puede contener varias sentencias RETURN, ninguna de las cuales necesita ser la última sentencia léxica. Ejecutando cualquiera de ellas, el subprograma finaliza inmediatamente.

La Sentencia RETURN y la cláusula RETURN

- la cláusula RETURN determina el tipo de dato del valor resultante en una especificación de función.

```
FUNCTION salario_ok (salario REAL, cargo REAL) RETURN BOOLEAN IS
    salario_min    REAL;
    salario_max    REAL;
BEGIN
    SELECT salario_bajo, salario_alto INTO salario_min, salario_max
        FROM salarios
        WHERE tarea = cargo;
    RETURN (salario >= salario_min) AND (salario <= salario_max);
END salario_ok;
```

Invocación de Subprogramas

- Se pueden llamar procedimientos y funciones desde cualquier herramienta o lenguaje que soporte PL/SQL.
- Desde PL/SQL, se puede llamar un procedimiento con una invocación directa y se puede invocar una función como parte de una expresión.

Invocación de Procedimientos desde Aplicaciones de SQL

- Se ingresa el nombre del procedimiento y cualquier parámetro real, si corresponde, en el prompt del Intérprete de la aplicación. Se ejecutará la acción especificada en el procedimiento.
- Por ejemplo,

```
aumentar_salario(17, 1000);
```

Invocando Procedimientos desde otros Procedimientos

- Un procedimiento puede llamar a otro procedimiento.
- Por ejemplo,

```
PROCEDURE procesar_salario (
    v_emp_id      IN INTEGER,
    v_incremento IN REAL) IS
BEGIN
    aumentar_salario(v_emp_id, v_incremento);
    ...
END procesar_salario;
```

Utilización de Funciones en una Sentencia SQL

- Se pueden llamar funciones PL/SQL dentro de cualquiera de las siguientes cláusulas SQL:
  - La lista select del comando SELECT.
  - La condición de una cláusula WHERE y HAVING.
  - Las cláusulas START WITH, ORDER BY y GROUP BY.
  - Las cláusulas VALUES del comando INSERT.
  - La cláusula SET del comando UPDATE.

La Sentencia RETURN

- En los procedimientos, una sentencia RETURN no puede contener una expresión. La sentencia simplemente devuelve el control al ambiente de invocación antes que se alcance el fin normal del procedimiento.
- En las funciones, una sentencia RETURN debe contener una expresión, que se evalúa cuando se ejecuta la sentencia RETURN.