Procedimientos almacenados

Jesús Reyes Carvajal

¿Qué es un procedimiento almacenado?

- Es un lenguaje que combina la fortaleza del SQL con las mas típicas características de un lenguaje de programación.

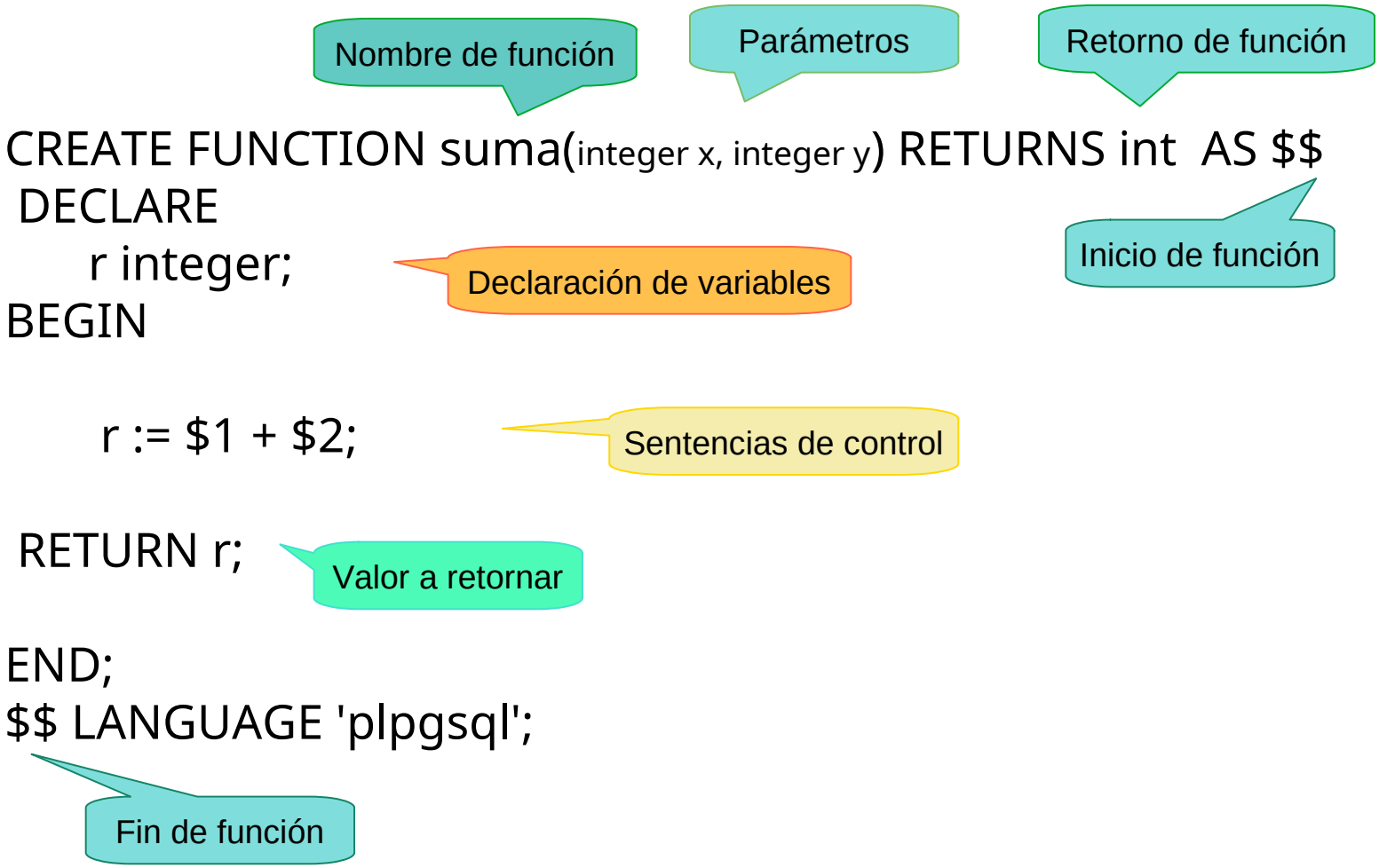
- Cuando se crea una función en PL/pgSQL, se puede incluir cualquiera de las sentencias del SQL, así como cualquier de los comandos procedimentales agregados por el PL/pgSQL.

- Las funciones escritas en PL/pgSQL pueden ser invocadas desde otras funciones, incluso pueden ser usadas en disparadores(Trigger).

Configuración

- PostgreSQL soporta varios lenguajes procedimentales, antes de poder usar cada uno de ellos es necesario instalar estos en la Base de Datos.
- El comando del shell CREATELANG instala PL/pgSQL en la base de datos.
- Sintáxis: ***createlang plpgsql database-name***.

Estructura básica de un procedimiento



Ejecución y eliminación de un procedimiento o función

```
CREATE FUNCTION suma(integer x, integer y) RETURNS integer AS $$
DECLARE
r integer;
BEGIN
        -- Calculando la nueva fecha
r := $1 + $2;
RETURN r;
END;
$$ LANGUAGE 'plpgsql';
```

Llamado de la función

SELECT suma(3,5);

Eliminación de la función

DROP FUNCTION suma(int,int);

Uso de comentarios

Dos estilos de comentarios:

- Doble guión (--). Extiende el comentario desde la posición actual hasta el final de la línea.
- Estilo – Lenguaje C (/* */). Inicia con carácter /* y termina con */. Se expande en múltiples líneas.

Ejemplo de comentarios

```
CREATE FUNCTION suma(integer x, integer y) RETURNS integer AS $$
 DECLARE
        r integer;
BEGIN
        -- Asigna en r el resultado de sumar los valores de X  y  Y
        r := $1 + $2;
RETURN r;
END;
$$ LANGUAGE 'plpgsql';
```

Parámetros

- A cada parámetro de entrada se le asigna un nombre de forma automática.
- Al primer parámetro (Izquierda a Derecha) se le denomina $1, al segundo $2, y así sucesivamente.
- A cada parámetro se le define el tipo de dato en la definición de la función.

Declaración de variables

- EL nombre de una variable puede incluir caracteres alfabéticos (A-Z), underscores (_) y números.
- El nombre de una variable debe empezar con letras (A-Z ó a-z) ó un underscore.
- El nombre de una variable es insensible a mayúsculas o minúsculas.

Declaración de variables en declare

Sintaxis completa para declarar una variable:

var-name [CONSTANT] var-type [NOT NULL] [{ DEFAULT | := } expression];

CONSTANT: especifica que la variable es constante, por lo que el valor inicial asignado a ella se mantendrá,

NOT NULL: especifica que la variable no puede tener asignado un valor nulo.

DEFAULT (o :=): asigna un valor a la variable.

- Usar valores apropiados para iniciar variables.
- No se permite el uso de parámetros de la función para iniciar variables en la sección DECLARE.
- Una vez creada una variable, no puede volver a ser usada en la sección DECLARE.

Ejemplo de variables en declare

```
CREATE FUNCTION calc_geometria(REAL) RETURNS REAL AS '
DECLARE
    pi CONSTANT REAL := 3.1415926535;
    radio REAL := 3.0;
    diametro REAL := pi * ( radio * radio );
    Edad INTEGER:= 30;
    Cantidad NUMERIC(5);
    Etapa INTEGER  DEFAULT 32;
    Url varchar := "http ://www.quantamagazine.org";
     ....
```

Tipos de datos

Si declara una variable para recibir datos de una tabla, es mejor utilizar los siguientes tipos de datos ***%type, %rowtype, opaque, record.***

TYPE *permite definir una variable que tenga el tipo de datos idéntico al de otra.*

Variable se usa para almacenar un valor recuperado de una tabla, o copiar los datos de un parámetro(excepto en declare)

ROWTYPE: *Define una variable que tiene la misma estructura de una tabla.*

RECORD: *Usado para declarar una variable compuesta cuya estructura será determinada en tiempo de ejecución.*

OPAQUE: *Puede ser usada para definir el tipo de datos de la variable de retorno de una función de tipo trigger. Cuando se devuelve un valor opaco se devuelve un renglón de la tabla del disparador(trigger).*

Ejemplo de tipos de datos

```
DECLARE
  mensaje VARCHAR2(100) := '¡Hola Mundo!';
  mensaje2 VARCHAR2(100) := mensaje || ' Mundo!';
  fechan DATE;
  Edad INTEGER;
  valor NUMBER;
  vcliente cliente%ROWTYPE;
  …..
```

Uso de alias

- Este comando crea un nombre alternativo para un parámetro de entrada.
- Solo se permite el uso de una variable para un parámetro de la función.

```
CREATE FUNCTION foo( INTEGER ) RETURNS INTEGER AS ‘
DECLARE
    param_1 ALIAS FOR $1;
    my_param ALIAS FOR $1;
    arg_1 ALIAS FOR $1;
BEGIN
    $1 := 42;
    ...
```

Comandos en Pl/pgSQL

- PL/pgSQL agrega un conjunto de instrucciones procedimentales a SQL.

- Incluye comandos: Ciclos, Excepciones y Manejo de Errores, asignaciones simples y ejecución condicional.

Manejo de errores

- Cuando PostgreSQL, decide que algo está mal, aborta la transacción y reporta un error.
- No se pueden interceptar los errores, no se puede corregir e intentar nuevamente y tampoco se puede traducir el error.
- Construir funciones que se anticipen a los errores.

Comando raise

Permite generar un error desde una función.

La sintaxis es:

RAISE severity 'message' [, variable [...]];

La severidad determina si la transacción debe ser abortada o no.

DEBUG: Mensaje grabado en el Log del Servidor e ignorado por éste. La función se completa, y la transacción no es afectada.

NOTICE: Mensaje grabado en el Log del Servidor y de allí enviada a la aplicación Cliente. La función se completa, y la transacción no es afectada.

EXCEPTION : Mensaje grabado en el Log del Servidor. La función termina, y la transacción es abortada.

Útil para depurar las funciones que se construyen.

Ejemplo comando raise

```
porcentaje := 20;
RAISE NOTICE ''Mayor (%)%", porcentaje;
```

Salida:

Mayor (20)%

Un ejemplo con exception:

```
SELECT * INTO myrec FROM EMP
WHERE empname =  myname;
IF NOT FOUND THEN
   RAISE EXCEPTION "empleado  % no  encontrado", myname;
ENDIF;
```

Comando select into

Este comando es otra forma de poner datos en una variable.

Sintaxis:

```
SELECT INTO destination [, ...] select-list FROM ...;
```

- La consulta NO debe devolver mas de un registro.
- Si obtiene varios registros solo toma el primero, los demás son desechados.

Ejemplo comando select into

```
CREATE OR REPLACE FUNCTION copia()
RETURNS text AS $h$
DECLARE
    copia clientes%ROWTYPE;
BEGIN
    SELECT * INTO copia FROM clientes WHERE
cod_cli='C0001';
    return copia.nomb_cli;
END;
$h$
LANGUAGE plpgsql;
```

Ejemplo de condicional

Ejemplo 1:

```
IF ( customers.balance > maximum_balance ) THEN
        RETURN( FALSE );
    ELSE
        rental_ok = TRUE;
    END IF;
```

Ejemplo 2:

```
IF  v_user_id  <> 0THEN
      UPDATE users SET email = v_email WHERE user_id = v_user_id;
ENDIF;
```

Ejemplo 3:

```
IF  v_count  > 0 THEN
  INSERT  INTO users_count(count) VALUES(v_count);
  RETURN "t";
ELSE
  RETURN "f";
ENDIF;
```

Ejemplo CASE

```
CREATE  OR REPLACE  FUNCTION  parimpar (x integer)
RETURNS  text AS $$
DECLARE
 msg text;
BEGIN
      CASE
            WHEN x IN(2,4,6,8,10)THEN
                msg :='Numero par';
           WHEN x IN(3,5,7,9,11)THEN
                msg :='Numero impar';
      END  CASE;
RETURN  msg;
END;
$$
LANGUAGE  plpgsql;
```

Ejemplo de loop

```
row := 0;
LOOP
   IF ( row = 100 ) THEN
       EXIT;
   END IF;
   col := 0;
   LOOP
      IF ( col = 100 ) THEN
          EXIT;
      END IF;
       col := col + 1;
   END LOOP;
  row := row + 1;
END LOOP;
RETURN(0);
```

Ejemplo de while

```
row := 0;
WHILE ( row < 100 ) LOOP
    col := 0;
    WHILE ( col < 100 ) LOOP
          col := col + 1;
    END LOOP;
    row := row + 1;
  END LOOP;
RETURN( 0 );
```

Ejemplo de for

```
FOR i IN 1..10  LOOP
    -- algunas  expresiones  aquí
    RAISES  NOTICE "i is  %",i;
END LOOP;


FOR i IN REVERSE  10..1  LOOP
  --algunas  expresiones  aquí
END LOOP;
```

For con iterator

La segunda forma de uso del ciclo FOR es usado para procesar el resultado de una consulta.

Sintaxis:

```
FOR iterator IN query LOOP
        statements
END LOOP;
```

Se denomina FOR-IN-SELECT, los comandos dentro del ciclo son ejecutados una vez por cada renglón que devuelve la consulta.

Query debe ser un comando SELECT del SQL.
En cada ciclo el iterador contiene el siguiente renglón devuelto por query.
Si no devuelve renglones no entra al ciclo.

Debe ser tipo RECORD o %ROWTYPE, debe coincidir con el renglón devuelto por la consulta.

Ejemplo con for con iterator

```
CREATE OR REPLACE FUNCTION listado(
   n INTEGER DEFAULT 10
)
RETURNS VOID AS $$
DECLARE
    rec RECORD;
BEGIN
    FOR rec IN SELECT nomb_cli
          FROM clientes
          ORDER BY nomb_cli
          LIMIT n
    LOOP
   RAISE NOTICE '%', rec.nomb_cli;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

```
NOTICE:  Carolina
NOTICE:  Jorge
NOTICE:  Pablo
```

Comando exit

Finaliza el bloque actual, y la ejecución del programa continua con el comando posterior al final del bloque.

Sintáxis:

EXIT [label] [WHEN boolean-expression];

Un comando **EXIT** simple finaliza el bloque anidado más cercano.

Si se incluye con la condición **WHEN**, se convierte en una salida condicional.

La salida condicional ocurre solo si la condición es verdadera.

Ejemplo comando exit

```
FOR i IN 1 .. 12 LOOP
    balance := customer.customer_balances[i];
    EXIT WHEN ( balance = 0 );
    PERFORM check_balance( customer, balance );
END LOOP;
RETURN( 0 );
```

Comando return

Todas las funciones escritas en PL/pgSQL terminan con este comando.

Sintaxis:

RETURN expression;

Cuando se ejecuta este comando, 4 cosas suceden:

1) La expresión es evaluada y si es necesario, convertida al tipo de datos apropiados.
2) La función actual finaliza su ejecución. Y todas la variables destruidas.
3) El valor de retorno es devuelto al invocador.
4) El control de ejecución es devuelto al invocador.

- Si no incluye el comando **RETURN** en la función, se recibe el mensaje: “Control reaches end of function without **RETURN**”.
- Se permite la inclusión de múltiples **RETURN** pero solo uno se ejecuta: el que se alcance primero.

Ejemplo con return

```
CREATE OR REPLACE FUNCTION fibonacci (n INTEGER)
  RETURNS INTEGER AS $$
DECLARE
  counter INTEGER := 0 ;
  i INTEGER := 0 ;
  j INTEGER := 1 ;
BEGIN

  IF (n < 1) THEN
    RETURN 0 ;
  END IF;

  LOOP
    EXIT WHEN counter = n ;
    counter := counter + 1 ;
    SELECT j, i + j INTO i,   j ;
  END LOOP ;

  RETURN i ;
END ;
$$ LANGUAGE plpgsql;
```