Data Base Administrator PostgreSQL

Ernesto Quiñones A.
ernesto@eqsoft.net

Sesión 6

Funciones PL/PgSQL Parte 1

Funciones

Las funciones en PostgreSQL son el equivalente a los store procedures además de funcionar como funciones en si mismas.

PostgreSQL soporta una variedad muy grande de lenguajes de programación para escribir funciones (java, perl, python, php, ruby, c, etc.) sin embargo el lenguaje de programación más desarrollado es Pl/PgSql.

Desde la versión 8.3 – 8.4 PostgreSQL ya incluye como lenguaje por defecto el Pl/PgSQL cuando creamos una DB, pero si requerimos instalarlo manualmente existen 2 opciones:

Por consola del sistema operativo :

createlang plpgsql nombre_db <-- podría requerir especificar usuario y password

Por consola psql:

```
prueba5=# create language 'plpgsql';
```

Funciones - Estructura

```
prueba5=# create or replace function     <-- declaración de ingreso de una función
            f_prueba (parametro integer)<-- nombre y parámetros
                returns integer as       <-- declaración de tipo de dato que se retornará
Prueba5-# $$                             <-- delimitador de inicio/fin de código
prueba5$# Declare                        <-- zona de declaración de variables internas que
prueba5$#    variable integer := 10;         usará la función
prueba5$# Begin                          <-- inicio de código
prueba5$#    return parametro * variable;<-- código de la función
prueba5$# end;                           <-- fin de código
Prueba5$# $$
prueba5-# language plpgsql;              <-- lenguaje “pl” que se está usando
CREATE FUNCTION

prueba5=# select * from f_prueba(20);    <-- invocando a la función
 f_prueba
----------
      200
(1 fila)
```

Funciones - Estructura

Podemos tener “bloques” de código anidados:

```
create or replace function f_prueba (parametro integer) returns
integer as $$
Declare
   variable integer := 10;
Begin
   raise notice 'el resultado es: %', parametro * variable;
   -- iniciamos segundo bloque
   Declare
      variable integer := 20;
   Begin
      raise notice 'a estas alturas es: %', parametro * variable;
   end;
   -- fin del segundo bloque
   return parametro;
end;
$$
language plpgsql;
```

Los bloques anidados NO SON transacciones anidadas, solo se ponen con fines de agrupamiento.

Funciones – Declaraciones de variables / constantes

No se pueden usar variables/constantes dentro de una función sin que estas hayan sido declaradas, el tipo de una variable puede ser cualquier tipo de datos disponible en PostgreSQL, más 2 tipos de estructura de las tablas de una db.

```
prueba5=# create table tbl_personas (edad int, nombre varchar(100),
ciudad varchar(100));
```

Funciones – Declaraciones de variables / constantes

```
create or replace function f_prueba2 (p_edad integer, p_nombre
varchar) returns integer as
$$
Declare
   v_ciudad   CONSTANT varchar := 'LIMA';← este valor es inmutable
   v_registro tbl_personas%ROWTYPE;      ← esta variable es un type
                                           basado en un registro
   v_nombre   tbl_personas.nombre%TYPE;  ← esta variable es del tipo
                                           de dato de la columna que
                                           referencia
Begin
   insert into tbl_personas(edad, nombre, ciudad) values (p_edad,p_nombre, v_ciudad);

   select * into v_registro from tbl_personas where nombre = p_nombre;
   raise notice 'El registro guardo: % % %', v_registro.edad,
          v_registro.nombre, v_registro.ciudad;

   select nombre into v_nombre from tbl_personas where nombre = p_nombre;
   raise notice 'Solo el campo nombre: %', v_nombre;
   return 1;
end;
$$ language plpgsql;
```

Funciones – Declaraciones de variables / constantes

```
prueba5=# select f_prueba2(15,'juan');
NOTICE:  El registro guardo: 15 juan LIMA
NOTICE:  Solo el campo nombre: juan
 f_prueba2
-----------
          1
```

Funciones – Declaraciones de variables / constantes

```
prueba5=# create type tpy_persona as ( altura numeric, peso numeric);

create or replace function f_prueba3 (p_talla numeric, p_peso numeric)
returns integer as $$
Declare
   v_talla_peso tpy_persona;
   v_registro   record;
Begin
   v_talla_peso.altura := p_talla;
   v_talla_peso.peso   := p_peso;
   raise notice 'La talla es: %' , v_talla_peso.altura ;
   raise notice 'El peso es: %' , v_talla_peso.peso ;

   select * into v_registro from tbl_personas where nombre = 'juan';
   raise notice 'El registro guardado:% % %', v_registro.edad,
           v_registro.nombre, v_registro.ciudad;
   return 1;
end;
$$ language plpgsql;

prueba5=# select f_prueba3(1.84,103);
NOTICE:  La talla es: 1.84
NOTICE:  El peso es: 103
NOTICE:  El registro guardo: 15 juan LIMA
```

En una función podemos declarar variables basadas en un TYPE, y si no conocemos/definimos una estructura podemos utilizar el tipo RECORD

Funciones – Parámetros

En Pl/PgSql podemos declarar parametros de entrada (IN) y de salida (OUT), ó parámetros INOUT.

```
create or replace function f_prueba4 ( p_quienes varchar,
              OUT p_nombre varchar, OUT p_apellido varchar)
                  returns record as
$$
Begin
   p_nombre   := substr(p_quienes,1,8);
   p_apellido := substr(p_quienes,10,8);
end;
$$ language plpgsql;
```

Una función con parámetros de OUT es invocada solo referenciando a los parámetros IN (IN es por defecto no se necesita escribirlo).

```
prueba2=> select* from f_prueba4('cadena de texto');
p_nombre | p_apellido
----------+------------
cadena d |  texto

prueba2=> select* from f_prueba4('cadena de texto','parametro1', 'parametro2');
ERROR:  function f_prueba4(unknown, unknown, unknown) does not exist
LINE 1: select* from f_prueba4('cadena de texto','parametro1', 'para...
```

Funciones – Parámetros

```
create or replace function f_prueba5 ( p_quienes varchar) returns integer
as
$$
Declare
   nombre varchar; apellido varchar; datos record;
Begin
   select * into nombre, apellido from f_prueba4(p_quienes) ;
   raise notice 'El nombre   es:%', nombre;
   raise notice 'El apellido es:%', apellido;
   return 1;
end;
$$ language plpgsql;

prueba5=# select f_prueba5('ernesto  quiñones');
NOTICE:  El nombre   es:ernesto
NOTICE:  El apellido es:quiñones
 f_prueba5
-----------
         1
```

Como se puede ver en “select * into nombre, apellido from f_prueba4(p_quienes)” es necesario definir 2 variables (o tantas como parámetros OUT tengamos) para recibir los datos de salida.

Funciones – Parámetros

Creemos 2 tablas “tbl_personas” y “tbl_padre” (con cualquier estructura) e ingresemos al menos un par de registros en cada uno.

```
create or replace function f_prueba6 (id integer,
      OUT r1 refcursor, OUT r2 refcursor)
      returns record as $$
begin
  open r1 for select * from tbl_personas;
  open r2 for select * from tbl_padre;
  return;
End;
$$ language plpgsql;
```

Un parámetro de OUT puede devolver set de datos, esto se puede conseguir utilizando cursores.

Funciones – Parámetros

```
create or replace function f_prueba7 () returns integer as $$
declare
  r1T refcursor; r2T refcursor;
  edad integer;  nombre varchar; ciudad varchar; cuenta varchar;
begin
  -- jalamos la data
  select * into r1T, r2T from f_prueba6(1);
  --abrimos primera tabla
  LOOP
    fetch r1T into edad,nombre, ciudad;
    IF NOT FOUND THEN exit; END IF;
    raise notice 'Tabla 1, data: % % %',edad, nombre,ciudad;
  end loop;  close r1T;
  --abrimos segunda tabla
  select * into r1T, r2T from f_prueba6(1);
  LOOP
    fetch r2T into cuenta;   IF NOT FOUND THEN  exit; END IF;
    raise notice 'Tabla 2, data: %',cuenta;
  end loop;   close r2T;
  return 1;
End; $$ language plpgsql;
```

Funciones – Parámetros

```
prueba5=# select * from f_prueba7();
NOTICE:  Tabla 1, data: 15 juan LIMA
NOTICE:  Tabla 2, data: 00.01
NOTICE:  Tabla 2, data: 10.01
NOTICE:  Tabla 2, data: 10.02
NOTICE:  Tabla 2, data: 40.01
NOTICE:  Tabla 2, data: 540.01
 f_prueba7
-----------
         1
(1 fila)
```

Funciones – Parámetros

Si la función se vuelve a declarar con el mismo nombre pero con diferentes parámetros o valor de retorno entonces PostgreSQL dará por entendido que son diferentes y se mantendrán todas en la dbms.

Es valido hacer esto:

```
Create function prueba1() returns integer as ....

Create function prueba1(parametro varchar) returns integer as ....

Create function prueba1(parametro int) returns date as ....
```

No es valido hacer esto:

```
Create function prueba2(parametro timestamp) returns int as ….

Create function prueba2(parametro timestamp) returns date as ….
```

En este caso, si dos funciones tienen el mismo parámetro de entrada y diferente salida entonces PostgreSQL no sabrá cual de las dos tomar, se enviará un mensaje de error al tratar de crear la 2da función.

Funciones – Parámetros

Podemos crear funciones con N número de parámetros usando arreglos (pero todos serán siempre del mismo tipo), se soporta anyelement, anyarray, anynonarray ó anyenum.

```
create or replace function f_prueba23( parametros anyarray) returns integer as
$$
BEGIN
  raise notice 'Elementos :%', parametros;
  raise notice 'Elemento 1:%', parametros[1];
  raise notice 'Elemento 2:%', parametros[2];
  return 1;
END;
$$ language plpgsql;

prueba4=# select * from f_prueba23(ARRAY[1,3,5,6,7]);
NOTICE:  Elementos :{1,3,5,6,7}
NOTICE:  Elemento 1:1
NOTICE:  Elemento 2:3

prueba4=# select * from f_prueba23(ARRAY['ernesto','juan']);
NOTICE:  Elementos :{ernesto,juan}
NOTICE:  Elemento 1:ernesto
NOTICE:  Elemento 2:juan

prueba4=# select * from f_prueba23(ARRAY['ernesto','juan',2]);
ERROR:  invalid input syntax for integer: "ernesto"
LINE 1: select * from f_prueba23(ARRAY['ernesto','juan',2]);
```

Funciones – Perform

Permite ejecutar querys/funciones sin necesidad de procesar devolución de datos

```
create or replace function f_prueba8 (numero integer) returns integer as
$$
BEGIN
  update tbl_prueba set id = id * numero; return 1;
END;
$$ language plpgsql;

create or replace function f_prueba9 () returns integer as
$$
BEGIN
  perform f_prueba8(10); return 1;
END;
$$ language plpgsql;

prueba4=# select * from tbl_prueba;
 id
----
  1

prueba4=# select f_prueba9();
 f_prueba9
-----------
         1

prueba4=# select * from tbl_prueba;
 id
----
 10
```

Funciones – STRICT

Permite la evaluación estricta de los datos que se invocan.

```
prueba2=> create table tbl_prueba(id int, nombre varchar);
CREATE TABLE

prueba2=> insert into tbl_prueba values (10,'Dato 10'),
               (20, 'Dato 20'),
               (23, 'Dato 23'),
               (10, 'Dato 10'),
               (10, 'Dato 10');
INSERT 0 5
```

Funciones – STRICT

```
create or replace function f_prueba10 (p_numero integer) returns integer as
$$
DECLARE
  numero1 tbl_prueba.id%TYPE;  numero2 tbl_prueba.id%TYPE;
BEGIN
  BEGIN
    RAISE NOTICE 'Primer Query';
    SELECT id INTO numero1 FROM tbl_prueba WHERE id = p_numero;
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        RAISE NOTICE '>No hay datos B: % %', p_numero, numero1;
      WHEN TOO_MANY_ROWS THEN
        RAISE NOTICE '>Muchos datos';
  END;
  BEGIN
    RAISE NOTICE 'Segundo Query';
    SELECT id INTO STRICT numero2 FROM tbl_prueba WHERE id = p_numero;
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        RAISE NOTICE ')No hay datos B: % %', p_numero, numero2;
      WHEN TOO_MANY_ROWS THEN
        RAISE NOTICE ')Muchos datos';
  END;
  return -1;
END;
$$ language plpgsql;
```

Funciones – STRICT

```
prueba2=> select * from f_prueba10(1);
NOTICE:  Primer Query
NOTICE:  Segundo Query
NOTICE:  )No hay datos B: 1 <NULL>
f_prueba10
------------
         -1
(1 row)

prueba2=> select * from f_prueba10(10);
NOTICE:  Primer Query
NOTICE:  Segundo Query
NOTICE:  )Muchos datos
f_prueba10
------------
         -1
(1 row)
```

Funciones – Retorno de Valores y Set de datos

La declaración del tipo de dato en la sección RETURNS de una función especifica de que tipo será, son soportados los tipos de datos convencionales más estos:

- **TYPES <-- declarados por el usuario**
- **RECORD <-- estructura no definida de datos**
- **SETOF RECORD <-- conjunto de estructuras de datos**
- **(TABLA) <-- podemos especificar el nombre de una tabla**
- **REFCURSOR <-- como SETOF RECORD**

Funciones – Retorno de Valores y Set de datos

Retornando un TYPE

```
prueba4=# create type tpy_prueba as( id integer, nombre varchar(100));

create or replace function f_prueba11 (p_numero integer, p_nombre
varchar) returns tpy_prueba as
$$
DECLARE
  data tpy_prueba;
BEGIN
  data.id := p_numero;  data.nombre := p_nombre;  return data;
END;
$$language plpgsql;

prueba4=# select * from f_prueba11(10,'ernesto');
 id | nombre
----+---------
 10 | ernesto
```

Funciones – Retorno de Valores y Set de datos

Retornando un Record

```
create or replace function f_prueba12(p_id integer) returns record as
$$
DECLARE
  datos record;
BEGIN
  select * into datos from tbl_prueba where id = p_id;
  return datos;
END;
$$ language plpgsql;

prueba4=# select * from f_prueba12(20) as (id integer, nombre
varchar);
 id | nombre
----+---------
 20 | Dato 20
```

Como el Record no tiene una estructura definida es necesario especificar cual será la estructura de salida.

En este caso que varios registros cumplan la condición retornará solo uno de ellos, sin preferencia alguna.

Funciones – Retorno de Valores y Set de datos

Retornando un Setof Record (devuelve un conjunto de datos)

```
create or replace function f_prueba13() returns setof record as
$$
BEGIN
  return query select * from tbl_prueba;
END;
$$
language plpgsql;
```

```
prueba4=# select * from f_prueba13() as (id integer, nombre
varchar);
 id | nombre
----+---------
 10 | Dato 10
 20 | Dato 20
 23 | Dato 23
 10 | Dato 10
 10 | Dato 10
```

Como el Record no tiene una estructura definida es necesario especificar cual será la estructura de salida.

Funciones – Retorno de Valores y Set de datos

Retornando un Setof TABLE (retorna un conjunto de datos con estructura de una tabla)

```
create or replace function f_prueba15() returns setof tbl_prueba
as
$$
BEGIN
  return query select * from tbl_prueba;
END;
$$
language plpgsql;
```

```
prueba4=# select * from f_prueba15();
 id | nombre
----+---------
 10 | Dato 10
 20 | Dato 20
 23 | Dato 23
 10 | Dato 10
 10 | Dato 10
```

Funciones – Retorno de Valores y Set de datos

Retornando un Refcursor (es necesario abrir una Transacción).

```
create or replace function f_prueba16(datos refcursor) returns setof refcursor
as $$
BEGIN
  open datos for select * from tbl_prueba;
  return next datos;
END;
$$ language plpgsql;

prueba4=# begin; select * into resultado from f_prueba16('data'); fetch all
from data;

SELECT 1
 id | nombre
----+---------
 10 | Dato 10
 20 | Dato 20
 23 | Dato 23
 10 | Dato 10
 10 | Dato 10
(5 rows)
prueba4=# commit;
COMMIT
```

Esta no es una forma formal de devolver datos, es una “alternativa” para obtener un set de datos del cual desconocemos la estructura que devolverá.

Funciones – Retorno de Valores y Set de datos

Retornando un Setof TYPE

```
prueba4=#create type tpy_data (id integer, nombre varchar(100));

create or replace function f_prueba13_3() returns setof tpy_data as
$$
BEGIN
  return query select id,nombre from tbl_prueba;
  return;
END;
$$
language plpgsql;

prueba4=# select * from f_prueba13_3();
 id | nombre
----+---------
 10 | Dato 10
 20 | Dato 20
 23 | Dato 23
 10 | Dato 10
 10 | Dato 10
(5 rows)
```

Esta es la mejor forma de retornar un set de datos, declaramos la estructura que vamos a devolver con nuestra consulta y la función puede ser utilizada por cualquier aplicación de forma transparente.

Funciones – Retorno de Valores y Set de datos

Retornando un Setof RECORD pero especificando la estructura de devolución como parámetros de OUT.

```
create or replace function f_prueba13_4( out p_id integer, out
p_nombre varchar) returns setof record as
$$
BEGIN
  return query select id,nombre from tbl_prueba;
  return;
END;
$$ language plpgsql;

prueba4=# select * from f_prueba13_4();
 p_id | p_nombre
------+----------
   10 | Dato 10
   20 | Dato 20
   23 | Dato 23
   10 | Dato 10
   10 | Dato 10
(5 rows)
```

Funciones – Los RETURNs

Return query ó return query execute nos permiten devolver set de datos de la ejecución directa de un query, en el segundo caso puede ser un query dinámico.

```
create or replace function f_prueba20() returns setof tpy_data2 as
$$
BEGIN
  return query select a.id,a.fecha,b.id,b.monto from factura_cab a,
                       factura_det b where a.id = b.fac_id;
  return;
END;
$$ language plpgsql;

create or replace function f_prueba20_2() returns setof tpy_data2 as
$$
BEGIN
  return query execute 'select a.id,a.fecha,b.id,b.monto from
             factura_cab a, factura_det b where a.id = b.fac_id';
  return;
END;
$$ language plpgsql;
```

Funciones – Los RETURNs

Return Next lo usamos en entornos donde necesitamos pre-procesar la data ejecutando código más complejo.

```
create or replace function f_prueba22() returns setof tpy_data3 as
$$ DECLARE data tpy_data3;
BEGIN
  FOR data IN select a.id,a.fecha,b.id,b.monto, '' from factura_cab a,
                      factura_det b where a.id = b.fac_id
  LOOP
    if data.monto > 200 then  data.mensaje = 'Muy caro';  end if;
    RETURN NEXT data;
  END LOOP;
  return;
END;
$$ language plpgsql;
```

```
prueba4=# select * from f_prueba22();
 id |   fecha    | id_det |  monto  | mensaje
----+------------+--------+---------+----------
  1 | 2010-01-01 |      1 |  100.20 |
  1 | 2010-01-01 |      2 |  100.30 |
  2 | 2010-01-02 |      3 |  200.30 | Muy caro
  3 | 2010-01-03 |      4 | 2500.30 | Muy caro
```