Funciones y Triggers
Introducción, Ejemplos

Álvaro Herrera

13 de noviembre de 2004

Contenidos

- Funciones: ¿para qué?
- Lenguajes
- Tipos de funciones
- Creación de funciones
  - tradicionales
  - SRFs
- Funciones para triggers
- Indices Funcionales

¿Para qué quiero funciones?

- Mantener lógica lejos de la aplicación
  - consistencia entre aplicaciones
  - reducción de funcionalidad duplicada
- Acceso predefinido a objetos restringidos
- Escoger herramienta adecuada a cada caso
- Algunas cosas necesitan lenguaje procedural
  - SQL es declarativo
  - se necesita poder expresivo distinto

Los lenguajes

- PostgreSQL soporta múltiples lenguajes procedurales
  - saber usar el apropiado en cada caso
  - conocer debilidades, fortalezas de cada uno
- Hackers:
  - agregar soporte a nuevos lenguajes
- Lenguajes confiables (trusted)
  - se puede “desconfiar” del usuario

Lenguajes: SQL

- Lenguaje más básico y simple
- No otorga poder adicional
- Permite simplificar consultas

Lenguajes: PL/pgSQL

- PL por excelencia
- Procedural
  - Sintaxis para loops, condicionales, etc
- Fácil de usar
  - portar desde Oracle PL/SQL
  - sintaxis simple y apropiada
- No particularmente veloz ···
  - cache de planes de ejecución

Lenguajes: C

- Muy potente, flexible
- Excelente rendimiento
- Tiene acceso a todo
  - ejecutar consultas, escribir archivos, etc
  - incluso a botar el proceso servidor
- Obtuso, pesado, complicado, peligroso
  - usarlo sólo en caso de necesidad!
- No es confiable (trusted)

Lenguajes: scripting

- Tcl
- Perl
- Python
- PHP
- otros

Lenguajes: PL/R

- Mención especial
  - brillantemente mantenido
- Propósito específico
  - manejo estadístico

Lenguajes: conclusiones

- Experimentar, usar, familiarizarse
- Puede alivianar el trabajo ···
  - si se usa la herramienta adecuada
- Puede mejorar rendimiento

"premature optimization is the root of all evil"
(Donald Knuth)

Tipos de funciones

- Según lo que retornan
  - Funciones “normales”
  - SRF (set-returning function)
  - Agregación
- Especiales
  - trigger
  - language_handler
- Privilegios durante invocación
  - security definer
  - security invoker

Funciones Tradicionales

- Retornan un solo resultado
  - escalar o tupla
- Es posible retornar un cursor abierto
- Pueden tener efectos secundarios
  - tabla temporal

Sintaxis de Creación de Funciones

```
CREATE [OR REPLACE] FUNCTION
([tipo argumento], …)
RETURNS [tipo resultado]
  { LANGUAGE lenguaje
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT
         | RETURNS NULL ON NULL INPUT
         | STRICT
    | [EXTERNAL] SECURITY INVOKER
         | [EXTERNAL] SECURITY DEFINER
    | AS 'definición'
    | AS 'archivo', 'símbolo'
  } …
```

Creación de Funciones: atributos

- IMMUTABLE
  - retorna lo mismo con los mismos argumentos
  - no hace búsquedas en la BD!
- STABLE
  - como immutable, pero puede cambiar con el estado de la BD
- VOLATILE
  - puede cambiar de una llamada a otra
  - si tiene efectos secundarios

Creación de Funciones: atributos (2)

- STRICT
- RETURNS NULL ON NULL INPUT
  - si algún argumento es NULL, no es necesario llamarla
- CALLED ON NULL INPUT
  - debe ser invocada

Ejemplo 1: Una fila, SQL

```
CREATE FUNCTION suma(INTEGER, INTEGER)
   RETURNS INTEGER
   IMMUTABLE STRICT LANGUAGE SQL
   AS 'SELECT $1 + $2';
regression=> select suma(10, 14);
 suma
------
   24
(1 row)
regression=> select * FROM suma(10, 14);
 suma
------
   24
(1 row)
```

Ejemplo 1: Una fila, SQL (cont.)

```
CREATE OR REPLACE FUNCTION suma(INT, INT)
  RETURNS INTEGER LANGUAGE SQL
  CALLED ON NULL INPUT
  AS '
    SELECT CASE
      WHEN $1 IS NULL THEN
        CASE
          WHEN $2 IS NULL THEN NULL
          ELSE $2
        END
      WHEN $2 IS NULL THEN $1
      ELSE $1 + $2
    END'
  IMMUTABLE;
```

Ejemplo 1a, PL/pgSQL

```
CREATE OR REPLACE FUNCTION suma_null(INT, INT, INT)
  RETURNS INTEGER CALLED ON NULL INPUT
  LANGUAGE plpgsql AS '
        DECLARE
            acum INTEGER;
        BEGIN
            acum := 0;
            IF $1 IS NOT NULL THEN
                acum := acum + $1;
            END IF;
            IF $2 IS NOT NULL THEN
                acum := acum + $2;
            END IF;
            IF $3 IS NOT NULL THEN
                acum := acum + $3;
            END IF;
            RETURN acum;
        END';
```

Ejemplo 1b: Un operador

```
CREATE OPERATOR @+
    (PROCEDURE = suma,
     LEFTARG = INTEGER,
     RIGHTARG = INTEGER);
SELECT 1 @+ 2 @+ NULL @+ 4;
 ?column?
----------
         7
(1 fila)
```

Ejemplo 2: Una fila, PL/pgSQL

Números a letras

```
CREATE OR REPLACE FUNCTION test_plpgsql(INT)
  RETURNS TEXT LANGUAGE plpgsql STRICT
  IMMUTABLE AS '
    DECLARE
      num ALIAS FOR $1;
      ret TEXT;
    BEGIN
      IF num = 1 THEN
        ret := ''uno'';
      ELSIF num = 2 THEN
        ret := ''dos'';
      END IF;
      RETURN ret;
    END ';
```

Ejemplo 3: Una fila, PL/perl

```
CREATE OR REPLACE FUNCTION num2pal(INTEGER)
  RETURNS TEXT AS '
    $num = shift;
    return undef if ($num < 1 || $num > 9);
    $num--;
    return (qw(uno dos tres cuatro cinco seis
               siete ocho nueve))[$num];
' LANGUAGE plperl;
regression=> select num2pal(5);
 num2pal
---------
 cinco
(1 fila)
```

Ejemplo 4: Una fila, C

```
#include "postgres.h"
#include "fmgr.h"
PG_FUNCTION_INFO_V1(digitoVer);
Datum digitoVer(PG_FUNCTION_ARGS) {
    int rut = PG_GETARG_INT32(0);
    text *ret;
    int M=0, S=1;
    for (; rut; rut = rut / 10)
        S = (S + rut % 10 * (9 - M++ % 6)) % 11;
    ret = (text *) palloc(VARHDRSZ + 1);
    VARATT_SIZEP(ret) = 5;
    sprintf(VARDATA(ret), "%c", S ? S + '0' - 1 : 'K
    PG_RETURN_TEXT_P(ret);
}
```

Ejemplo 4: Una fila, C (cont.)

```
gcc -Wall -O2 -c -fpic
  -I‘pg_config --includedir‘
  -I‘pg_config --includedir‘/server
  pglib.c
gcc -shared pglib.o -o pglib.so
CREATE OR REPLACE FUNCTION
  digito_verificador(INTEGER)
  RETURNS TEXT
  STRICT IMMUTABLE
  AS ’/usr/local/lib/pgsql/pglib.so’,
  ’digitoVer’ LANGUAGE C;
```

Ejemplo 4: Una fila, C (cont.)

```
#include "postgres.h"
#include "fmgr.h"
PG_FUNCTION_INFO_V1(digitoVer);
Datum digitoVer(PG_FUNCTION_ARGS) {
    int rut = PG_GETARG_INT32(0);
    text *ret;
    int M=0, S=1;
    for (; rut; rut = rut / 10)
        S = (S + rut % 10 * (9 - M++ % 6)) % 11;
    ret = (text *) palloc(VARHDRSZ + 1);
    VARATT_SIZEP(ret) = VARHDRSZ + 1;
    sprintf(VARDATA(ret), "%c", S ? S+'0'-1 : 'K');
    PG_RETURN_TEXT_P(ret);
}
```

Ejemplo 4: para que?

```
CREATE TABLE clientes (
    …
    rut INTEGER,
    dv TEXT CHECK (dv = digito_verificador(rut)),
    …
 );
```

Set-returning Functions (SRF)

- Funciones que “retornan tablas”
- En cláusula FROM
  - hacer JOIN
    - con otras tablas
    - otras funciones
    - subconsultas
- Definir tipo de retorno

SRF: Ejemplos de uso

```
SELECT * FROM
    una_srf(arg1, arg2),
    otra_srf(arg1, arg2)
    WHERE una_srf.uno = otra_srf.uno
SELECT * FROM
    una_srf(arg1, ···)
    JOIN otra_srf( ··· ) USING (columna)
    JOIN una_tabla ON (condicion ···)
```

SRF: Ejemplos de uso (cont.)

```
SELECT * FROM
     una_tabla,
     una_srf(…),
     (SELECT tres, cuatro
           FROM otra_srf(…)
           WHERE …)
     WHERE …
```

SRF: Tipos de retorno

- Lo especial de una SRF
- RETURNS SETOF foo
- foo puede ser
  - un tipo
  - una tabla
  - una definición anónima

SRF: Sintaxis para retorno

```
CREATE FUNCTION una_srf (…)
        RETURNS SETOF tabla …;
SELECT * FROM una_srf (foo, bar, …);

CREATE TYPE foo (a int, b int);
CREATE FUNCTION otra_srf (…)
        RETURNS SETOF foo …;
SELECT * FROM otra_srf(…);

CREATE FUNCTION tercera_srf(…)
        RETURNS SETOF RECORD …;
SELECT * FROM tercera_srf(…)
    AS tercera(definicion de tipo);
```

SRF en plgpsql: sintaxis

- RETURN NEXT foo
  - acumula la tupla foo para retornar
- RETURN termina la función
  - y devuelve todas las tuplas retornadas

```
FOR SELECT ··· LOOP
    ...
    RETURN NEXT ···
END LOOP;
RETURN;
```

Ejemplo 5: SRF en plpgsql

- Todo el mes en intervalos de 30 minutos

```
DECLARE
    inicio  ALIAS FOR $1;
    valor   TIMESTAMP WITH TIME ZONE;
    final   ALIAS FOR $2;
    paso    ALIAS FOR $3;
    ret     RECORD;
```

Ejemplo 5: SRF en plpgsql (cont.)

```
BEGIN
valor := inicio;
LOOP
    IF valor >= final THEN
        RETURN;
    END IF;
    SELECT INTO ret valor, valor + paso;
    RETURN NEXT ret;
    valor := (valor + paso);
END LOOP;
END;
```

Ejemplo 5: SRF en plpgsql (cont.)

```
SELECT * FROM intervalos
    ('2003-01-01', '2003-01-02', '3 hour')
    AS foo(inicio TIMESTAMP WITH TIME ZONE,
           fin TIMESTAMP WITH TIME ZONE);
```

inicio	*fin*
2003-01-01 00:00:00-03	2003-01-01 03:00:00-03
2003-01-01 03:00:00-03	2003-01-01 06:00:00-03
2003-01-01 06:00:00-03	2003-01-01 09:00:00-03
2003-01-01 09:00:00-03	2003-01-01 12:00:00-03
2003-01-01 12:00:00-03	2003-01-01 15:00:00-03
2003-01-01 15:00:00-03	2003-01-01 18:00:00-03
2003-01-01 18:00:00-03	2003-01-01 21:00:00-03
2003-01-01 21:00:00-03	2003-01-02 00:00:00-03

Ejemplo 6: SRF en SQL

```
CREATE FUNCTION getfoo(int)
    RETURNS setof foo AS ’
        SELECT * FROM foo
        WHERE fooid = $1;’
    LANGUAGE SQL;
SELECT * FROM getfoo(1);
```

Privilegios

- Otorgar privilegios: GRANT
- Funciones: EXECUTE
- Lenguajes: USAGE
  - untrusted: sólo usuarios confiables!
- SECURITY INVOKER
  - normal; permisos del que hace SELECT
- SECURITY DEFINER
  - equivalente a *setuid*

Ejemplo 7: Privilegios

```
CREATE TABLE blah ( ··· );
GRANT SELECT ON TABLE blah TO usuario1;
SET SESSION AUTHORIZATION usuario1;
CREATE FUNCTION lee_blah AS '··· /* lee tabla blah */
GRANT EXECUTE ON lee_blah TO usuario2;
SET SESSION AUTHORIZATION usuario2;
SELECT * FROM blah;
Error: permiso denegado
SELECT * FROM lee_blah();
```

- puede otorgar una visión restringida
- puede otorgar privilegios limitados de modificación

Índices Funcionales

- Crear un índice siguiendo una función
- Agiliza consultas que usen la función
- Función debe ser IMMUTABLE

Ejemplo 9: Índice Funcional

```
CREATE INDEX indice_func
    ON tabla date_part('month', campo_fecha);
SELECT * FROM tabla WHERE
  date_part('month', campo_fecha) = ...
SELECT * FROM tabla WHERE
  date_part('month', campo_fecha)
     BETWEEN ... AND ...
```

Referencias:

- http://www.postgresql.org
- http://techdocs.postgresql.org
- http://www.postgresql.cl
- http://www.varlena.com/GeneralBits
- http://archives.postgresql.org/pgsql-es-ayuda

¿Preguntas?