PL/php – Linguagem Procedural em PHP para PostgreSQL

Conteúdo

O que é PL/php?

PL/php é uma linguagem procedural para o SGBD PostgreSQL, destinada a permitir que se escreva funções em PHP para ser usadas como funções dentro de bancos do PostgreSQL. Ela foi escrita pela

Command Prompt, Inc. E desde então teve seu códifo fonte aberto sob as licenças PHP e PosgreSQL (BSD).

Download e Instalação

Favor consultar a documentação de instalação para instruções sobre como instalar a versão 1.0 da PL/php. Para instalar o novo código, que somente funciona com as versões 8.0, 8.1 e seguintes e está atualmente em desenvolvimento, acesse ao contrário acesse esta página.

Criando a linguagem PL/php

Favor consultar a documentação em como criar a linguagem dentro de um banco de dados logo que a biblioteca esteja instalada. Se você está usando o PostgreSQL 8.1 você precisa, ao contrário, seguir essas outras instruções.

Funções e Argumentos

Para criar a função, use a sintaxe padrão:

```
CREATE FUNCTION funcname (argument-types) RETURNS return-type AS $$
   # plphp corpo da função aqui
```

```
$$ LANGUAGE 'plphp';
Argumentos são passados no array $args e o valor resultante é retornado com o a
declaração 'return'.
CREATE FUNCTION ola_mundo() RETURNS text AS $$
   $ola = 'Olá mundo!';
   return $ola;
$$ LANGUAGE 'plphp';
select ola_mundo();
CREATE FUNCTION ola_numero(integer) RETURNS integer AS $$
   $nr = 5;
   return $args[0] + $nr;
$$ LANGUAGE 'plphp';
select ola_numero(5)
Obs.: Os dois exemplos acima não constam na documentação original em inglês,
foram adicionados pelo tradutor.
CREATE FUNCTION plphp_max(integer, integer) RETURNS integer AS $$
   if ($args[0] > $args[1]) {
      return $args[0];
   } else {
      return $args[1];
   }
$$ STRICT LANGUAGE 'plphp'
```

NOTA: O uso da cláusula STRICT poupa-nos de ter que pensar em entradas com valor NULL para a nossa função. Se for passado um valor NULL, a função não será executada completamente, mas sim apenas retornará o resultado NULL automaticamente.

Em uma função que não use STRICT se o valor atual de um argumento for NULL o correspondente variável $args[n-1] deve ser setada para uma string vazia (unset).

Tipos de Dados e Retornos

Os argumentos passados para as funções em PL/php são convertidos para texto assim podemos manipular então com a ausência de tipos de scripts em PHP. Inversamente, o comando return deve aceitar qualquer string com formato aceitável para o tipo de retorno declarado das funções.

Variável Global Compartilhada

Esta é uma variável global que pode ser usada para armazenar dados entre chamadas de funções, denominada $_SHARED.

```
CREATE FUNCTION set_var(text) RETURNS text AS $$
   global $_SHARED;
   $_SHARED['primeiro']=$args[0];
   return 'ok';
$$ LANGUAGE 'plphp';
CREATE FUNCTION get_var() RETURNS text AS $$
   global $_SHARED;
   return $_SHARED['primeiro'];
$$ LANGUAGE 'plphp';
SELECT set_var('olá plphp');
SELECT get_var(); -- deve retornar 'olá plphp'
```

NOTA: A variável global compartilhada é connection-specific. Isto é útil para passar informações em torno da execução de um único script, mas esta variável é limpa quando a conexão é fechada.

Suporta Arrays do PostgreSQL

Existe suporte para arrays multidimensionais.

Por exemplo:

```
CREATE FUNCTION php_array() RETURNS text[][] AS $$
   $return = array(array("Steven", "Klassen"),
                   array("Jonathan", "Daugherty"));
   return $return;
$$ LANGUAGE 'plphp';
sklassen=# select php_array();
                 php_array
----------------------------------------
 {{Steven,Klassen},{Jonathan,Daugherty}}
(1 row)
```

Argumentos Polimórficos e Tipos de Retorno

As funções devem ser declaradas para aceitar e retornar os tipos polimórficos 'anyelement', 'anyarray' e 'anyrecord'. Consulte a seção 33.2.5 da documentação do PostgreSQL para uma explanação mais detalhada sobre funções polimórficas.

Por exemplo,

```
CREATE FUNCTION array_three_values(anyelement, anyelement, anyelement) RETURNS
anyarray AS '
   $ret[0] = $args[0];
   $ret[1] = $args[1];
   $ret[2] = $args[2];
   return $ret;
' LANGUAGE 'plphp';
SELECT array_three_values(3,2,1);
 array_three_values
--------------------
 {3,2,1}
(1 row)
CREATE OR REPLACE FUNCTION php_row(integer) RETURNS record AS '
   $ret[f1]=$args[0];
   $ret[f2]="hello";
   $ret[f3]="world";
   return $ret;
' LANGUAGE 'plphp';
select * FROM php_row(1) AS (f1 integer, f2 text, f3 text);
```

Acesso a Bancos de Dados (SPI)

SPI – Server Programming Interface

São oferecidas algumas funções de acesso a bancos de dados.

1. spi_exec - Executa uma consulta com opcional limite.

   ```
   resource spi_exec(string query[, int limit])
   ```

2. spi_fetch_row - Retorna um array associativo das linhas (rows) do resultado.

   ```
   array spi_fetch_row(resource result)
   ```

3. spi_status - Retorna o status de uma consulta executada anteriormente. Se o retorno for

SPI_OK_SELECT você pode obter tuplas do resultado usando spi_fetch_row.

```
string spi_status(resource result)
```

4. spi_processed - Retorna o número de tuplas do resultado.

   ```
   int spi_processed(resource result)
   ```

5. spi_rewind – Coloca o cursor row (linha) no início do resultado, assim spi_fetch_row deve continuar obtendo tuplas do início do resultado.

   ```
   void spi_rewind(resource result)
   ```

Por exemplo:

Este não é uma função particularmente útil, mas irá ilustrar as funções de acesso descritas acima. Você entra com um id inteiro e retorna um campo texto com o nome do usuário.

```
CREATE FUNCTION get_username(integer) RETURNS text AS $$
   # Assign the query to a variable.
   $query = "SELECT username FROM users WHERE id = " .$args[0];
   # Run the query and get the $result object.
   $result = spi_exec_query($query);
   # Fetch the row from the $result.
   $row = spi_fetch_row($result);
   return $row['username'];
$$ LANGUAGE 'plphp';
sklassen=# select get_username(1);
 get_username
--------------
 sklassen
(1 row)
```

Note que resultados de spi_exec_query são alocados na RAM como um inteiro, por isso se você precisa processar grandes conjuntos de resultados, você precisa usar um cursor. Isto não é possível com a atual interface.

Triggers

Quando uma função está sendo usada para retornar uma trigger, o array associativo $_TD contém o valor trigger-related.

$_TDnew?
: Um array associativo contendo os valores da nova linha da table para a ação INSERT/UPDATE, ou vazio para DELETE. O array é indexado pelo nome do campo. Nota importante: Campos que estão NULL não deverão aparecer no array!

$_TDold?
: Um array associativo contendo os valores das antigas linhas da tabela para ações UPDATE/DELETE, ou vazio para INSERT. O array é indexado pelo nome do campo. importante: Campos que estão NULL não deverão aparecer no array!

$_TDname?
: Contém o próprio nome da trigger.

$_TDevent?
: Contém um dos valores: "INSERT", "UPDATE", "DELETE".

$_TDwhen?
: Contém um dos valores: "BEFORE", "AFTER".

$_TDlevel?

Contém um dos valores: "ROW", "STATEMENT".

$_TDrelid?
: Contém a relação de ID da tabela em que a trigger ocorreu.

$_TDrelname?
: Contém o nome da relação.

$_TDargs?
: Um array de argumentos passado para a trigger, se existir. Ele pode ser accessado como $_TDargs?[idx]. Por exemplo, $_TDargs?[0].

$_TDargc?
: O número do argumento passado para a trigger, 0 se nenhum.

Exemplo de uma trigger AFTER INSERT

Suponha que você tenha uma tabela de usuários com colunas típicas e uma tabela atividade que você está usando para rastrear as páginas de acesso. No INSERT de registro para a tabela atividade, você deseja atualizar o campo last_seen do apropriado registro do usuário.

Considere a seguinte definição de tabela:

```
CREATE TABLE usuarios (
    id serial PRIMARY KEY NOT NULL,
    login text NOT NULL,
    email text,
    ultima_visita timestamp without time zone,
    ativo boolean DEFAULT true NOT NULL
);
CREATE TABLE atividade (
    id serial PRIMARY KEY NOT NULL,
    usuario_id integer NOT NULL,
    arquivo_acessado text NOT NULL,
    stamp timestamp without time zone DEFAULT now() NOT NULL,
    CONSTRAINT usuario_id_existe FOREIGN KEY (usuario_id) REFERENCES
usuarios(id)
);
CREATE FUNCTION update_ultimavisita() RETURNS trigger AS $$
    $new =& $_TD['new'];
    if (isset($new['usuario_id']) && isset($new['stamp'])) {
        $query = "UPDATE usuarios SET ultima_visita = '" .$new['stamp'].
                 "' WHERE id = " .$new['usuario_id'];
        $rv = spi_exec($query);
    }
    return;
$$ LANGUAGE 'plphp';
CREATE TRIGGER after_update_ultimavisita_trigger
    AFTER INSERT ON atividade FOR EACH ROW EXECUTE PROCEDURE
update_ultimavisita();
```

1. Inserir um novo registro em usuarios.

```
sklassen=# insert into usuarios (login, email) values ('sklassen','
 sklassen@commandprompt.comEste endereço de e-mail está sendo protegido de spam,
você precisa de Javascript habilitado para vê-lo
 ');
INSERT 1
sklassen=# select * from usuarios where login = 'sklassen';
 id | username |           email            | last_seen | active
----+----------+----------------------------+-----------+--------
  1 | sklassen |
```

```
 sklassen@commandprompt.comEste endereço de e-mail está sendo protegido de spam,
você precisa de Javascript habilitado para vê-lo
  |           | t
(1 row)
```

2. Insert a new row into the activity table.

```
sklassen=# insert into activity (users_id, file_accessed) values
(1,'index.html');
INSERT 1
```

3. Verificar a garantir que nossa trigger disparou como esperado.

```
sklassen=# select * from usuarios where login = 'sklassen';
 id | username |           email            |         last_seen          |
active
----+----------+----------------------------+----------------------------
+--------
  1 | sklassen |
 sklassen@commandprompt.comEste endereço de e-mail está sendo protegido de spam,
você precisa de Javascript habilitado para vê-lo
  | 2005-01-10 09:48:57.191595 | t
(1 row)
```

Exemplo de uma trigger BEFORE INSERT ou UPDATE

Deixe-me dizer que temos um usuário chamado admin que nós desejamos prevenir a aplicação de modificá-lo. Nós devemos criar uma trigger BEFORE DELETE que previne então de excluir o registro e uma trigger BEFORE UPDATE que previne então de modificações do login em que a trigger anterior depende.

```
CREATE TABLE usuarios (login text, email text);
INSERT INTO usuarios VALUES ('admin', '
 admin@commandprompt.comEste endereço de e-mail está sendo protegido de spam,
você precisa de Javascript habilitado para vê-lo
 ');
INSERT INTO usuarios VALUES ('darcy', '
 darcy@example.comEste endereço de e-mail está sendo protegido de spam, você
precisa de Javascript habilitado para vê-lo
 ');
CREATE OR REPLACE FUNCTION imortal() RETURNS trigger AS $$
   # O registro não deverá ser excluído se o login for "admin".
   if ($_TD['old']['login'] == 'admin') {
      pg_raise('notice', "Você não pode excluir o usuário admin");
      return 'SKIP';
   }
   return;
$$ LANGUAGE 'plphp';
CREATE TRIGGER before_delete_imortal_trigger BEFORE DELETE ON usuarios
FOR EACH ROW EXECUTE PROCEDURE imortal();
CREATE OR REPLACE FUNCTION protect_admin() RETURNS trigger AS $$
   # Não deixa então modificar o login da conta admin.
   $oldUsername = $_TD['old']['login'];
   $newUsername = $_TD['new']['login'];
   if ($oldUsername == 'admin' && ($oldUsername != $newUsername)) {
      pg_raise('notice', "Você não pode alterar o login admin.");
      return 'SKIP';
   } else {
      return;
   }
$$ LANGUAGE 'plphp';
```

```
CREATE TRIGGER before_update_protect_admin_trigger BEFORE UPDATE ON
usuarios FOR EACH ROW EXECUTE PROCEDURE protect_admin();
```

Agora o usuário admin não pode ser excluído, nem pode ser alterado:

```
pl_regression=# select * from usuarios;
 username |          email
----------+------------------------
 admin    |
 admin@commandprompt.comEste endereço de e-mail está sendo protegido de spam,
você precisa de Javascript habilitado para vê-lo

 darcy    |
 darcy@example.comEste endereço de e-mail está sendo protegido de spam, você
precisa de Javascript habilitado para vê-lo

(2 rows)
pl_regression=# update usuarios set login = 'foobar';
NOTICE:  plphp: Você não pode alterar o login admin
UPDATE 1
pl_regression=# select * from usuarios;
 username |          email
----------+------------------------
 admin    |
 admin@commandprompt.comEste endereço de e-mail está sendo protegido de spam,
você precisa de Javascript habilitado para vê-lo

 foobar   |
 darcy@example.comEste endereço de e-mail está sendo protegido de spam, você
precisa de Javascript habilitado para vê-lo

(2 rows)
pl_regression=# delete from usuarios;
NOTICE:  plphp:  Você não pode excluir o usuário admin
DELETE 1
pl_regression=# select * from usuarios;
 username |          email
----------+------------------------
 admin    |
 admin@commandprompt.comEste endereço de e-mail está sendo protegido de spam,
você precisa de Javascript habilitado para vê-lo

(1 row)
```

Confiáveis x não Confiáveis

Normalmente, PL/php é instalada como uma linguagem procedural confiável, nomeada de 'plphp'. Com esta configuração o PHP deve rodar em "safe mode". Leia mais sobre as restrições aqui:

http://www.php.net/manual/en/features.safe-mode.functions.php

Em geral, as operações que foram restritas são aquelas que interagem com o ambiente. Isso inclui operações do sistema de arquivos, require e use (para módulos externos).

Desde que não existe nenhuma maneira de ter acesso aos processos internos do servidor de bancos de dados ou ao sistema operacional em si, qualquer usuário do SGBD sem privilégios pode usar funções escritas nesta linguagem.

Um exemplo de uma função que não funciona devido às restrições de segurança:

```
CREATE FUNCTION ler_arquivo_senha() RETURNS text AS '
```

```
   readfile("/etc/passwd");
   return 0;
' LANGUAGE 'plphp';
```

Aparentemente ela deve executar, mas dependendo do seu nível de log, você deve ver alguma coisa como:

```
Warning: readfile(): SAFE MODE Restriction in effect. The script
whose uid is 500 is not allowed to access /etc/passwd owned by uid 0
in Command line code on line 3
Warning: readfile(/etc/passwd): failed to open stream: Success in
plphp function source on line 3
```

Algumas vezes é desejável escrever funções que não são restritas. Nesses casos, você pode criar a linguagem como 'plphpu' para habilitar as previamente indisponíveis funções.

```
CREATE LANGUAGE plphpu;
```

Argumentos de Tipos Compostos

Argumentos de tipos compostos são passados para as funções como arrays associativos. As chaves dos arrays são os nomes dos atributos do tipo composto. Aqui está um exemplo:

```
CREATE TABLE employee (
   name text,
   basesalary integer,
   bonus integer
);
CREATE FUNCTION empcomp(employee) RETURNS integer AS '
   return $args[0][''basesalary''] + $args[0][''bonus''];
' LANGUAGE 'plphp';
INSERT INTO employee values ('Josh', 1000, 10);
INSERT INTO employee values ('Darcy', 500, 20);
SELECT name, empcomp(employee) FROM employee;
```

Este exemplo resulta na seguinte saída:

```
 name  | empcomp
-------+---------
 Josh  |    1010
 Darcy |     520
(2 rows)
```

Retornando uma Linha (Row) (tipo composto)

Para retornar uma linha ou um valor de sipo composto de uma função na linguagem PL/php, você pode usar um array indexado:

```
CREATE TYPE php_row AS (f1 integer, f2 text, f3 text);
CREATE OR REPLACE FUNCTION php_row(integer) RETURNS php_row AS $$
   $ret['f1'] = $args[0];
   $ret['f3'] = "world";
   $ret['f2'] = "hello";
   return $ret;
$$ LANGUAGE 'plphp';
SELECT * FROM php_row(1);
```

Deve retornar:

```
 f1 |  f2   |  f3
----+-------+-------
  1 | hello | world
```

Retornando múltiplas linhas (funções SRF)

SRF são "set-returning functions". Isto permite que você possa retornar múltiplas linhas, como no exemplo seguinte:

```
CREATE TYPE php_row AS (f1 integer, f2 text, f3 text);
CREATE OR REPLACE FUNCTION php_set(integer) RETURNS SETOF php_row AS $$
   $ret['f1'] = $args[0];
   $ret['f2'] = "hello";
   $ret['f3'] = "world";
   return_next($ret);
   $ret['f1'] = 2 * $args[0];
   $ret['f2'] = "hello";
   $ret['f3'] = "postgres";
   return_next($ret);
   $ret['f1'] = 3 * $args[0];
   $ret['f2'] = "hello";
   $ret['f3'] = "plphp";
   return_next($ret);
$$ LANGUAGE 'plphp';
```

Isto deve resultar em:

```
pl_regression=# SELECT * FROM php_set(1);
 f1 |  f2   |    f3
----+-------+----------
  1 | hello | world
  2 | hello | postgres
  3 | hello | plphp
(3 rows)
```

O que é importante ressaltar aqui é a função return_next(). Esta função precisa ser chamada com o parâmetro simples, que precisa ser um array com um elemento por atributo do tipo de retorno. Note que atualmente os nomes dos elementos dos arrays são ignorados; os elementos são atribuídos em ordem. Se você precisa retornar um valor nulo, atribua um valor nulo para um elemento do array. Isto deve ser corrigido em futuras versões.

Finalizando a SRF é causada por uma ou outra chamada return ou por falha no final da função. Neste ponto todos os resultados são entregues de volta para o exterior da função,

At that point the entire results are delivered back to the outer query, sendo diretamente para o usuário ou para fazer junção (join) com outras tabelas/SRFs/etc.

As tuplas retornadas são escritas para um "tuplestore" do Postgres , que é levado de volta para o disco e não necessita ficar inteiramente na RAM.

Nomes de Argumentos de Funções

Você pode atribuir nomes para os argumentos de funções em pl/php:

```
CREATE FUNCTION plphp_match(document text, word text) RETURNS BOOL AS
$$
   if (strstr($document, $word))
       return TRUE;
   else
```

```
       return FALSE;
$$ LANGUAGE 'plphp';
Example output:
template1=# select plphp_match('PostgreSQL', 'SQL');
 plphp_match
-------------
 t
(1 row)
Podemos usar tanto o nome do argumento ($nome) quanto a variável
$args[position], pois ambos são suportados.
```

Limitações

Funções em PL/php não podem chamar outras funções em PL/php diretamente porque seus nomes são danificados (mangled).

Chamadas SPI têm todo o resultado alocado em memória, assim processando grandes resultados deve ser problemático. Também não é possível usar cursores com SPI.

Original em Inglês: https://projects.commandprompt.com/public/plphp/wiki/Documentation

Tradução: Ribamar FS – http://postgresql.ribafs.org