

**plphp - PHP jako
język proceduralny w PostgreSQL.**

PHP od kilku lat z powodzeniem zdobywa wszelkie odmiany aplikacji webowych. I jak to zwykle bywa w takich przypadkach ma on zarówno gorących zwolenników jak i zaciekłych przeciwników. Po którejkolwiek stronie barykady się nie stanie, faktem jest, że PHP jest wszechobecne i przynajmniej na razie żaden nowy język programowania nie zapowiada zmiany tej sytuacji.

PHP od dłuższego czasu istnieje jako język skryptowy, choć wydają się, że przynajmniej w tym przypadku nie udało mu się zdeponować perla i języków skryptowych opartych na shellach.

Od roku można także wykorzystywać PHP jako język proceduralny w PostgreSQL. PIPHP – co wydaje się brzmieć jak herezja – wkrada się w miejsca zupełnie nowe i jakby zupełnie nieprzeznaczone dla tego języka. Czy tak jest w istocie?

Języki proceduralne.

Języki proceduralne (PL - procedural languages) w PostgreSQL pozwalają na pisanie funkcji i procedur wykonywanych po stronie bazy danych. Dzięki nim część logiki aplikacji może zostać przeniesiona do bazy danych. Pozwala to na ograniczenie ilości pomyłek w danych przechowywanych w bazie danych, dodatkową kontrolę ich integralności i możliwość znacznego rozszerzenia funkcjonalności RDBMS.

Interpretacją funkcji napisanych w dowolnym PL nie zajmuje się Postgres. W nim przechowywany jest jedynie handler do funkcji, której przekazywana jest treść funkcji plPHP, a zadaniem funkcji podpiętej do handlera jest wykonanie kodu funkcji PL.

W PostgreSQL dostępne są obecnie następujące języki proceduralne:

- plpgsql - pierwszy dostępny język proceduralny w PGSQL stworzony przez Iana Wiecka, wykorzystuje własną składnię i język
- pltcl - PL oparty na TCL
- plperl - oczywiście perl i oczywiście wyznawcy perla tylko wzruszą ramionami kiedy usłyszą o plphp:)
- plpython - kolejna możliwość wykorzystania popularnego i wygodnego języka jako PL

plphp nie jest jeszcze oficjalną częścią PostgreSQL, ale pewnie należy się tego spodziewać być może już w wersji 7.5 PostgreSQL.

Instalacja plphp.

Źródła plPHP są dostępne pod adresem <http://plphp.commandprompt.com/>. Sama kompilacja i instalacja jest świetnie opisana w pliku INSTALL i nie ma potrzeby jej tutaj powtarzać. Warto zwrócić jedynie uwagę na drobny szczegół, mianowicie na to, że plPHP nie daje się skompilować pod starszymi gcc. Żeby uniknąć problemów warto postarać się o

względnie aktualną wersję (nie udało mi się skompilować pPHP z GCC 2.95.4, a z GCC 3.3.3 bez problemów).

Po zaaplikowaniu łątki pPHP na źródła PostgreSQL w katalogu `./postgresql-x.y.z/src/pl/plphp` znajduje się skompilowany handler pPHP, w katalogu `./tests/` kilka przykładowych funkcji do przetestowania, a w katalogu `./doc/` dokumentacja.

Samo tworzenie języka jest dość proste. Najpierw należy utworzyć funkcję-handler, która będzie interpretować zawartość funkcji PL, a następnie utworzyć język i powiązać go z naszym handlerem. Istnieją dwie rzeczy, na które trzeba zwrócić uwagę. Po pierwsze, jeśli utworzymy język w bazie `template1`, każda baza która będzie później tworzona, będzie już zawierała dostęp do tego języka. `template1` jest wzorem dla każdej nowo utworzonej bazy. I sprawa druga, na którą nie zwrócono uwagi w dokumentacji pPHP, składnia tworzenia funkcji handlera musi zawierać dokładną ścieżkę do pliku zawierającego handler funkcji.

```
template1=# CREATE FUNCTION plphp_call_handler() RETURNS
LANGUAGE_HANDLER AS '/usr/lib/postgresql/lib/plphp.so' LANGUAGE C;
CREATE FUNCTION
template1=# CREATE TRUSTED LANGUAGE plphp HANDLER plphp_call_handler;
CREATE LANGUAGE
```

Sprawdźmy teraz czy PostgreSQL "widzi" nowy język.

```
template1=# select * from pg_language where lanname like 'pl%';
 lanname | lanispl | lanpltrusted | lanplcallfoid | lanvalidator | lanacl
-----+-----+-----+-----+-----+-----
 plpgsql | t       | t             | 17883         | 0             |
 plphp   | t       | t             | 30853         | 0             |
```

Jak widać utworzony został język pPHP.

Język pPHP został utworzony z dodatkowym parametrem - TRUSTED. Ogranicza to w znaczny sposób możliwości działania funkcji. Funkcje utworzone za pomocą języka zaufanego będą mogły wykonywać jedynie te akcje, do których nie potrzeba praw administratora, czyli dostępu do bazy danych czy systemu plików. W pPHP funkcje dostępne w "trybie" TRUSTED są także ograniczone przez "safe mode" samego PHP. Spis funkcji dostępnych w trybie "safe mode" znajduje się pod następującym adresem: <http://www.php.net/manual/en/features.safe-mode.functions.php>

Tworzenie języka jako untrusted jest prawie identyczne z tworzeniem języka trusted, różnica występuje jedynie w nazwie tworzonego języka:

```
template1=# CREATE LANGUAGE plphpu HANDLER plphp_call_handler;
CREATE LANGUAGE
template1=# select * from pg_language where lanname like 'pl%';
 lanname | lanispl | lanpltrusted | lanplcallfoid | lanvalidator | lanacl
-----+-----+-----+-----+-----+-----
 plpgsql | t       | t             | 17883         | 0             |
 plphp   | t       | t             | 30853         | 0             |
```

Jak widać pojawił się dodatkowy język z flagą "lanpltrusted" ustawioną na "false".

Przykładowe funkcje

Najprostsza funkcja wzięta wprost z dokumentacji plPHP wygląda następująco:

```
CREATE OR REPLACE FUNCTION sum(integer, integer) RETURNS integer AS
    'return $args[0]+$args[1];'
LANGUAGE 'plphp';
```

Powyższe polecenie spowoduje utworzenie lub wymianę funkcji na nową. Po nazwie funkcji "sum" określamy jej parametry, a następnie typ zwracanego wyniku. Pomiedzy znakami " " znajduje się tekst funkcji i na końcu język, jaki ma być użyty do zinterpretowania zawartości funkcji.

```
test=# SELECT sum(11,12);
sum
-----
 23
(1 row)
```

Spróbujmy wykorzystać z języka PHP trochę więcej niż sam operator dodawania:

```
CREATE FUNCTION plphp_max (integer, integer) RETURNS integer AS '
if ($args[0] > $args[1]){
    return $args[0];
} else return $args[1];
' LANGUAGE 'plphp' WITH (isStrict);
```

Jeśli wykorzystujemy "isStrict", a w argumencie funkcji pojawi się wartość NULL, zawartość całej funkcji jest pomijana. Zwracana jest od razu wartość NULL. Jeśli "isStrict" nie zostanie użyte, argument, który miał wartość NULL zostaje przekazany w tablicy \$args[] z pustym łańcuchem (unset).

Od czasu do czasu na listach dotyczących postgresa pojawia się pytanie jak wysłać i czy w ogóle da się wysłać maila bezpośrednio z postgresa. Dzięki wprowadzeniu języków unrestricted jest to możliwe, dzięki plphp jest to banalne (czy ma to sens, to już zupełnie inna sprawa:)):

```
CREATE OR REPLACE FUNCTION mail(text, text, text) RETURNS text AS '
    if (mail ($args[0],$args[1],$args[2])) return $args[0];
    else return NULL;
' LANGUAGE 'plphpu';
```

Spróbujmy teraz przetestować naszą nową funkcję:

```
template1=# select mail('mazek@netsync.pl','sss','sssagwe');
ERROR: plphp: fatal error...
```

Ważną rzeczą, na którą trzeba zwrócić uwagę jest to, że nazwa funkcji

plphp nie może być identyczna z funkcją już istniejącą w php. Po drobnej przeróbce wszystko gra, jak poniżej:

```
CREATE OR REPLACE FUNCTION maz_mail(text,text,text) RETURNS text AS '  
    if (mail($args[0],$args[1],$args[2])) return $args[0];  
    else return NULL;  
' LANGUAGE 'plphpu';
```

I test samej funkcji:

```
SELECT maz_mail('m.mazurek@netsync.pl','alarm z pgsq1','wywolana zostaa...');
```

Za chwilę stanie się coś, przy czym purystom bazodanowym zadrży ręka: z funkcji napisanej w języku proceduralnym utworzony zostanie plik na zawnętrzym systemie plików.

```
CREATE OR REPLACE FUNCTION maz_zapisz(text) RETURNS text AS '  
    $f=fopen('/tmp/plik.txt','w+');  
    fwrite($f,'some data');  
    fclose($f);  
    return NULL;  
' LANGUAGE 'plphpu';
```

Funkcja `maz_zapisz` zapisuje podany argument w pliku, który jest argumentem funkcji PHP `fopen()`. Można sobie wyobrazić bardzo wiele zastosowań takiej funkcji. Może ona być przydatna na potrzeby backupowania, przy wprowadzeniu nowych pozycji do cennika. Można też łatwo uzyskać jego zrzut automatycznie, zamiast wykonywać krok dodatkowy.

Do tej pory wykorzystywaliśmy funkcje, które są dostępne w PHP natywnie, ale czy da się skorzystać z funkcji, które są dostępne jako rozszerzenie PHP? Oczywiście tak! Każdy kto zna PHP i jego rozbudowany zestaw funkcji zostanie zauroczony tą możliwością. Przy tworzeniu biblioteki PHP w procesie instalacji plPHP należy dodać jedynie rozszerzenie, które chcemy do niej dodać. Dla przykładu dodajmy proste funkcje zestawu "calendar":

```
./configure --enable-calendar  
make libphp4.la
```

Spróbujmy wykorzystać jedną z tych funkcji:

```
CREATE OR REPLACE FUNCTION wielkanoc(text) RETURNS text AS '  
    $j=$args[0];  
    $i=date('M-d-Y', easter_date($j));  
    return $i;  
' LANGUAGE 'plphpu';
```

```
SELECT wielkanoc('2004');
```

Funkcja "wielkanoc" jako argument przyjmuje rok i dzięki funkcji PHP `easter_date()` zwraca datę Wielkanocy w podanym roku. Użyteczność tej funkcji – może nieszczerólnie imponująca – pokazuje w jaki sposób można znacznie rozbudować możliwości języków proceduralnych.

Co dalej?

W kolejnej części tekstu o plPHP postaram się przedstawić możliwości wykorzystania go jako języka do tworzenia triggerów, czyli akcji wyzwalanych wewnątrz w bazie danych po wykonaniu określonych działań na danych. Prostotę ich tworzenia i możliwości doceni każdy administrator baz danych i programista.

Linki:

<http://plphp.commandprompt.com/> - oficjalne repozytorium plphp

<http://www.postgresql.org/docs/7.4/static/xplang.html> - opis wykorzystania języków proceduralnych w PostgreSQL

<http://www.php.net/>