

pPHP – PHP jako język proceduralny w PostgreSQL

Nie tylko Perl

Język PHP to język skryptowy, jednak nie udało mu się zdezonizować Perla i języków skryptowych powłoki. Niniejszy artykuł przedstawia możliwość zastosowania PHP w zupełnie nowym dla niego środowisku - jako języka proceduralnego dla PostgreSQL.

MARCIN MAZUREK

PHP od wielu lat jest z powodzeniem stosowany we wszelkiego rodzaju aplikacjach WWW. I jak to zwykle bywa w takich przypadkach, ma on zarówno gorących zwolenników, jak i zaciekle przeciwników. Fakty są jednak takie, że PHP jest wszechobecny i przynajmniej na razie żaden nowy język programowania nie zapowiada zmiany tej sytuacji.

Od roku można także wykorzystywać PHP jako język proceduralny w PostgreSQL. Język pPHP – co wydaje się brzmieć jak herezja – wkrada się w miejsca całkiem nowe i jakby zupełnie nie przeznaczone 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 sam PostgreSQL. W nim przechowywana jest jedynie funkcja obsługi (handler), która przekazuje treść wywołania pPHP do funkcji/procedury bezpośrednio ją realizującej. Język pPHP nie jest jeszcze oficjalną częścią PostgreSQL, ale zapewne należy się tego spodziewać, być może już w wersji 7.5 serwera PostgreSQL.

Instalacja

Źródła pPHP są dostępne pod adresem [2]. Sama kompilacja i instalacja jest bardzo do-

kładnie opisana w pliku INSTALL i nie ma potrzeby jej tutaj powtarzać. Warto zwrócić jedynie uwagę na drobny szczegół, mianowicie na to, że pPHP nie daje się skompilować pod starszymi wersjami kompilatora 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 nie było problemów).

Po zaaplikowaniu łatki pPHP na źródła PostgreSQL w katalogu `./postgresql-x.y.z/src/pl/p1php` znajduje się skompilowana funkcja obsługi pPHP, w katalogu `./tests/` jest kilka przykładowych funkcji do przetestowania, natomiast w katalogu `./doc/` dokumentacja.

Samo tworzenie języka jest dość proste. Najpierw należy utworzyć funkcję obsługi, która będzie interpretować zawartość konkretnej funkcji PL, a następnie stworzyć język i powiązać go z utworzoną funkcją obsługi. Na dwie sprawy trzeba jednak zwrócić uwagę. Po pierwsze, jeśli utworzymy język w bazie `template1`, każda później tworzona baza będzie także zawierała dostęp do tego języka. Baza `template1` jest w PostgreSQL wzorem dla każdej nowej bazy danych. I sprawa druga, na którą nie zwrócono uwagi w dokumentacji pPHP – składnia two-

żenia funkcji obsługi musi zawierać dokładną ścieżkę do pliku zawierającego funkcje obsługi pPHP.

```
template1=# CREATE FUNCTION ↵
p1php_call_handler() RETURNS
LANGUAGE_HANDLER AS '/usr/lib/↵
postgresql/lib/p1php.so' ↵
LANGUAGE C;
CREATE FUNCTION
template1=# CREATE TRUSTED ↵
LANGUAGE p1php HANDLER ↵
p1php_call_handler;
CREATE LANGUAGE
```

Sprawdźmy teraz, czy PostgreSQL „widzi” nowy język.

```
template1=# select * from ↵
pg_language where lanname ↵
like 'p1%';
lanname | lanispl | lanpltrusted ↵
| lanplcallfoid | lanvalidator ↵
| lanac1
-----+-----+-----↵
-+-----+-----+-----↵
-----
p1pgsql | t | t | 17883 | 0 |
p1php | t | t | 30853 | 0 |
```

W powyższym przykładzie utworzony został język pPHP z dodatkowym parametrem – TRUSTED. Funkcje utworzone za pomocą języka zaufanego (TRUSTED) 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. Opis funkcji dostępnych w trybie *safe mode* znajduje się pod adresem [3].

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

```
template1=# CREATE LANGUAGE ↵
p1php HANDLER p1php_call_handler;
CREATE LANGUAGE
template1=# select * from ↵
pg_language where lanname ↵
like 'p1%';
lanname | lanispl | ↵
lanpltrusted | lanplcallfoid | ↵
lanvalidator | lanac1
-----+-----+-----↵
-+-----+-----+-----↵
-----
```

Języki proceduralne dostępne w PostgreSQL [1]

p1pgsql	pierwszy dostępny język proceduralny w PGSQL, stworzony przez Iana Wiecka, wykorzystuje własną składnię i język
pl1cl	PL oparty na TCL
p1perl	czyli opisywany przez nas p1PHP
p1python	kolejna możliwość wykorzystania popularnego i wygodnego języka jako PL

```
plpgsql | t | t | 17883 | 0 |
plphp | t | t | 30853 | 0 |
plphpu | t | f | 30853 | 0 |
```

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

Przykładowe funkcje

Najprostsza funkcja, wzięta wprost z dokumentacji pPHP, wygląda następująco:

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

Powyzsze polecenie spowoduje utworzenie lub wymianę funkcji na nową. Po nazwie funkcji *sum* podajemy jej parametry, a następnie typ zwracanego wyniku. Między 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 „wycisnąć” z języka PHP trochę więcej niż tylko 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 PostgreSQL pojawia się pytanie: jak wysłać i czy w ogóle da się wysłać maila bezpośrednio z PostgreSQL. Dzięki wprowadzeniu języków unrestricted jest to możliwe, zaś dzięki pPHP 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';
```

Przetestujmy teraz naszą nową funkcję:

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

Ważną sprawą, na którą trzeba zwrócić uwagę jest to, że nazwa funkcji pPHP nie może być identyczna z funkcją już istniejącą w PHP. Na szczęście po drobnej przeróbce wszystko działa prawidłowo:

```
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';
```

Oto test samej funkcji:

```
SELECT maz_mail('m.mazurek@>
netsync.pl','alarm' pgsq',>
'wywolana zosta?a...');
```

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

```
CREATE OR REPLACE FUNCTION >
maz_zapisz(text) RETURNS text >
AS '
$f=fopen('/tmp/plik.txt',>
'w+');
write($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 np. przydatna na potrzeby wykonywania kopii zapasowych, przy wprowadzeniu no-

wych pozycji do cennika itp.

Do tej pory wykorzystywaliśmy funkcje, które są dostępne w PHP natywnie, ale czy możliwe jest skorzystanie z funkcji dostępnych 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 pPHP należy określić jedynie rozszerzenie, które chcemy dodać. Dla przykładu dodajmy proste funkcje zestawu *calendar*:

```
./configure --enable-calendar
make libphp4.1a
```

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 nieszczerze imponująca – pokazuje, w jaki sposób można znacznie rozbudować możliwości języków proceduralnych.

Co dalej?

W kolejnej części tekstu o pPHP postaram się przedstawić możliwości wykorzystania go jako języka do tworzenia triggerów, czyli akcji wyzwalanych wewnętrznie w bazie danych po wykonaniu określonych działań na danych. Prostotę ich tworzenia i możliwości doceniają administratorzy baz danych i programiści. ■

INFO

- [1] Opis wykorzystania języków proceduralnych w PostgreSQL:
<http://www.postgresql.org/docs/7.4/static/xplang.html>
- [2] Oficjalne repozytorium pPHP:
<http://plphp.commanprompt.com>
- [3] Opis trybu safe mode PHP:
<http://www.php.net/manual/en/features.safe-mode.functions.php>