Jun 27th, 2008

PL/Perl

Los Angeles PostgreSQL User's Group

by Juan J. Natera

Outline

1. Perl
2. Installing PL/Perl
    - From source
    - From binaries
3. Using PL/Perl
    - Necessary Perl
    - Writing functions
    - Using SQL inside your functions
    - Sharing data between function calls
4. More Information

Perl

. Programming language.

Perl

. Programming language.
. Created by Larry Wall in 1986.

Perl

. Programming language.
. Created by Larry Wall in 1986.
. Ubiquitous, it's available in all modern Unix-like operating systems.

Perl

. Programming language.
. Created by Larry Wall in 1986.
. Ubiquitous, it's available in all modern Unix-like operating systems (and Windows too).

Perl

. Programming language.
. Created by Larry Wall in 1986.
. Ubiquitous, it's available in all modern Unix-like operating systems (and Windows too).
. CPAN, more libraries than you can shake a stick at.

Perl

. Programming language.
. Created by Larry Wall in 1986.
. Ubiquitous, it's available in all modern Unix-like operating systems (and Windows too).
. CPAN, more libraries than you can shake a stick at.
. Currently at version 5.10, includes many features brought from Perl 6.

Installing PL/Perl
From source

```
$ tar -xvzf postgresql-8.3.3.tar.gz
$ ./configure --prefix=/usr/local --with-perl
$ make
# make install
```

Installing PL/Perl
From binaries (Debian)

```
# aptitude install postgresql-8.1 \
                   postgresql-client-8.1 \
                   postgresql-plperl-8.1
```

Enabling PL/Perl

. Enabling it globally:

```
$ createlang plperl template1
```

Enabling PL/Perl

. Enabling it globally:

```
$ createlang plperl template1
```

. Enabling it per database:

```
$ createlang plperl lapugdemo
```

Enabling PL/Perl

. Enabling it globally:

```
$ createlang plperl template1
```

. Enabling it per database:

```
$ createlang plperl lapugdemo
```

Or:

```
$ psql lapugdemo
lapugdemo# CREATE LANGUAGE plperl
```

Necessary Perl

. Perl Datatypes
. Subroutines
. References

Perl Datatypes

```
# Scalars

my $i = 0;
my $city = 'Pasadena';
```

Perl Datatypes

```
# Scalars

my $i = 0;
my $city = 'Pasadena';

# Arrays

my @universities = ('Caltech', 'UCLA', 'USC');
print $universities[0];
```

Perl Datatypes

```
# Scalars

my $i = 0;
my $city = 'Pasadena';

# Arrays

my @universities = ('Caltech', 'UCLA', 'USC');
print $universities[0];

# Hashes

my %students = ('Caltech' => 23456, 'UCLA' => 12345);
print $students{'Caltech'};
```

Perl Subroutines

```
0    # find number in an array
1    sub find {
2      my ($number, @array) = @_;
3      foreach my $i (@array) {
4        return $number if ($number == $i);
5      }
6      return undef;
7    }
```

Perl References

```
0  # creating a reference to a scalar
1  my $foo = 5;
2  my $ref = \$foo;
3  print “$foo\n”;
4  print “$ref\n”;
```

Perl References

```
0  # creating a reference to a scalar
1  my $foo = 5;
2  my $ref = \$foo;
3  print “$foo\n”;
4  print “$ref\n”;
```

Prints:

```
5
SCALAR(0x504f60)
```

Perl References

```
0   # creating a reference to a scalar
1   my $foo = 5;
2   my $ref = \$foo;
3   print “$foo\n”;
4   print “$ref\n”;
5   # Dereference it
6   print “${$ref}\n”;
```

Prints:

```
5
SCALAR(0x504f60)
5
```

Perl References

```
0  # creating a reference to an array
1  my @foo = (5, 7, 9, 11);
2  my $ref = \@foo;
3  print “@foo\n”;
4  print “$ref\n”;
5  # Dereference it
6  print “${$ref}\n”;
```

Perl References

```
0  # creating a reference to an array
1  my @foo = (5, 7, 9, 11);
2  my $ref = \@foo;
3  print “@foo\n”;
4  print “$ref\n”;
5  # Dereference it
6  print “${$ref}\n”;
```

Prints:

```
5 7 9 11
ARRAY(0x504f60)
Not a SCALAR reference at line X # Oops!
```

Perl References

```
0  # creating a reference to an array
1  my @foo = (5, 7, 9, 11);
2  my $ref = \@foo;
3  # Dereference the whole array
4  print “@{$ref}\n”;
5  # Dereference a single element
6  print $ref->[0], “\n”;
```

Prints:

```
5 7 9 11
5
```

Perl References

```
0  # creating a reference to an hash
1  my %foo = ('Los Angeles' => 'LAX', \
              'New York' => 'JFK');
2  my $ref = \%foo;
3  print $ref->{'Los Angeles'}, “\n”;
```

Prints:

```
LAX
```

Perl References

```
# anonymous arrays
my $arrayref = [1, 2, 3, 4, 5];

# anonymous hashes
my $hashref = { 'Foo' => 1, 'Bar' => 2};
```

PL/Perl

```
CREATE FUNCTION funcname (argument-types) \
RETURNS return-type AS $$
    # PL/Perl function body
$$ LANGUAGE plperl;
```

PL/Perl

A sample function:

```
CREATE FUNCTION reverse (TEXT) RETURNS TEXT AS
$$
  my ($string) = @_;
  my @symbols = split //, $string;
  return join ('', reverse @symbols);
$$ LANGUAGE plperl;
```

PL/Perl

A composite type function:

```
CREATE OR REPLACE FUNCTION tax (orders) RETURNS
float AS $$

  my ($o) = @_;
  return sprintf("%.2f",
    $o->{totalamount} - $o->{netamount}
  );
$$ LANGUAGE plperl;
```

PL/Perl

Returning a composite type:

```
CREATE TYPE order_with_tax AS (orderid integer,
tax numeric(12,2));

CREATE OR REPLACE FUNCTION owtax (orders)
RETURNS  order_with_tax AS $$

  my ($o) = @_;
  return {
    orderid => $o->{orderid},
    tax     => sprintf("%.2f",
      $o->{totalamount} - $o->{netamount}
    )
  };
$$ LANGUAGE plperl;
```

PL/Perl

Returning a set of rows:

```
CREATE OR REPLACE FUNCTION order_set()

RETURNS SETOF order_w_tax AS $$
    return_next({ orderid => 1, tax => 12.27});
    return_next({ orderid => 2, tax => 3.89});
    return_next({ orderid => 3, tax => 4.65});
    return_next({ orderid => 4, tax => 9.78});
    return undef;
$$ LANGUAGE plperl;
```

PL/Perl

Running queries inside your functions:

```
CREATE OR REPLACE FUNCTION nlargest(INTEGER) RETURNS SETOF order_w_tax AS $$
return undef unless(@_ && $_[0] > 0);
my $result = spi_exec_query('SELECT * FROM orders ORDER BY totalamount DESC LIMIT '.$_[0]);
foreach (@{$result->{rows}}) {
  return_next({
    orderid  => $_->{order_id},
    tax      => sprintf("%.2f",
      $_->{totalamount} - $_->{netamount}
    )
  });
}
return undef;
$$ LANGUAGE plperl;
```

PL/Perl

Sharing data between functions calls:

```
CREATE FUNCTION set_it(TEXT) RETURNS TEXT AS $$

my ($it) = @_;

$_SHARED{'it'} = $it;

return $it;

$$ LANGUAGE plperl;


CREATE FUNCTION get_it() RETURNS TEXT AS $$

return $_SHARED{'it'};

$$ LANGUAGE plperl;
```

More information

http://www.postgresql.org/docs/8.3/interactive/plperl.html

http://www.postgresql.org/docs/8.3/interactive/xplang.html

http://www.cpan.org/modules/by-module/DBD/APILOS/