PL/R extension in PostgreSQL 9.6

Document Author: arho.virkki@tyks.fi

For details, see the PL/R manual: http://www.joeconway.com/plr/doc/index.html

Prerequisites

One needs *super user* priviliges to ***create new R functions*** in PostgreSQL. Once the function is created, also non-privileged users can use it.

For deploying new functions, we set up a new **ktp_adm** user (having the same password as the ordinary *ktp* user).

```
CREATE ROLE ktp_adm LOGIN SUPERUSER PASSWORD '<passwd_here>';
```

Examples

R code can contain comments (starting with #) and empty lines. If empty lines produce an error, check that the function definition is passed to PostgreSQL in one batch (as *psql* `\i <filename.sql>` command does by default).

Vector argument, double precision output

```
-- The function is created into 'test' schema. It accepts
-- a double precision postgresql vector 'a' and returns
-- a double precision scalar

CREATE OR REPLACE FUNCTION test.array2double( a float8[] )
RETURNS float8 AS
$$
median ( a )
$$ LANGUAGE plr;

SELECT test.array2double( '{1,2,5}'::float8[] );
```

Single row output

```
CREATE OR REPLACE FUNCTION test.get_table()
RETURNS setof int4 AS $$
array(1:10)
$$ LANGUAGE plr;

SELECT test.get_table();
```

Table output from a data.frame

```
-- One option it to define a table and return a row set of that type

CREATE TABLE IF NOT EXISTS test.integertable (
  val1 int4,
  val2 int4 );

CREATE OR REPLACE FUNCTION test.get_table(n int4 default 10, m text default 'foo')
RETURNS setof test.integertable AS
$$
data.frame(1:n, n:1)
$$ LANGUAGE plr;

SELECT * FROM test.get_table(12);
SELECT * FROM test.get_table(m:='fifi');
```

```
-- Other option is to build the output record format in the
-- function definition

CREATE OR REPLACE FUNCTION test.get_table2(
  IN n int4,
  OUT val1 int4,
  OUT val2 int4)
RETURNS setof record AS $$
data.frame(1:n, n:1)
$$ LANGUAGE plr;

SELECT * FROM test.get_table(25);
```

Read table into a data.frame

```
CREATE OR REPLACE FUNCTION test.table_reader( table_name text )
RETURNS text AS
$$
#
# The queries follow RPostgreSQL package syntax with the
# exception that the connection object is neglected (and hence NA here)
#
X <- dbGetQuery( NA, paste("SELECT * FROM", table_name))
infotext <- paste0("nrow: ", nrow(X), " ncol: ", ncol(X), " colnames: ",
                   paste( colnames(X), collapse=",") )
#
return( infotext )
$$ LANGUAGE plr;

SELECT * FROM test.table_reader( 'test.koodisto_ods' );
```

Working with the Date type

```
DROP FUNCTION IF EXISTS delays.event_delays() ;
CREATE FUNCTION delays.event_delays(
  OUT id int4,                          -- Observe how
  OUT pid text,                         -- the output table is
  OUT event_idx int4,                   -- defined here
  OUT event text,
  OUT event_type text,
  OUT event_info text,
  OUT event_date date,                  -- Date type is 'date' in PosgreSQL
  OUT decision_date date,
  OUT first_treatment bool,
  OUT treatment_delay int4 )
RETURNS setof record AS
$$
con <- NA

# Detect first treatment events with plain SQL
events <- dbGetQuery(con, "
SELECT
    <QUERY HERE>
")
                                    ## In the following, we need to cast
                                    ## dates explicitly into 'Date' (as
                                    ## they where read as 'character')

events$decision_date <- as.Date(events$decision_date)
events$event_date <- as.Date(events$event_date)

   <R COMPUTATIONS>
                                    ## Finally, the internal R 'Date' types
                                    ## are cast back to 'character' for
                                    ## PostgreSQL to parse it as 'date'

events$decision_date <- as.character(events$decision_date)
events$event_date <- as.character(events$event_date)

return( events )                    ## Finally, return a data frame that
                                    ## Matches the FUNCTION definition

$$ language plr;
```

Installation

For compiling the extension, PostgreSQL command line tools need to be in path:

```
# this is a good place to put own customizations
cd /usr/local/bin/
sudo ln -s /usr/pgsql-9.6/bin/* .
```

Download the latest release of PL/L from https://github.com/postgres-plr/plr/releases

```
wget https://github.com/postgres-plr/plr/archive/REL8_3_0_17.tar.gz
tar xvzf REL8_3_0_17.tar.gz
cd plr-REL8_3_0_17
USE_PGXS=1 make

sudo bash -c "PATH=$PATH:/usr/local/bin/ USE_PGXS=1 make install"
```

Now we can install the extension to the desired databases:

```
sudo su - postgres
psql
\c ktp
CREATE EXTENSION plr;
-- Optionally, to get rid of the extension, issue:
-- DROP EXTENSION plr;
```

Now test the extension

```
SELECT * FROM plr_environ();
SELECT load_r_typenames();
SELECT * FROM r_typenames();
SELECT plr_array_accum('{23,35}', 42);
```

Test the installation

Example of a function (under test schema):

```
--
-- Create this function with 'kpt_adm' user
--
CREATE OR REPLACE FUNCTION test.r_max (a integer, b integer)
RETURNS integer AS '
    if (a > b)
       return(a)
    else
       return(b)
' LANGUAGE 'plr';
```

Test the function

```
-- Some test data
CREATE TABLE test.maxtest (
  id SERIAL,
  i INTEGER,
  j INTEGER);
INSERT INTO test.maxtest ( i, j ) VALUES
  ( 1, 3),
  ( 5, 2),
  (-9,-1);

-- Ordinary users can now use the function
SELECT id, test.r_max(i,j) AS maximum FROM test.maxtest;
```