# PL/R – The Fusion of PostgreSQL and R

Joe Conway
joe.conway@credativ.com
mail@joeconway.com

credativ Group

June 15, 2012

# Intro to PL/R

What is R?

- An open source language and environment for statistical computing and graphics...

What is PostgreSQL?

- PostgreSQL is a powerful, open source object-relational database system. It has more than 25 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness.

What is PL/R?

- R Procedural Language Handler for PostgreSQL. Enables user-defined SQL functions to be written in the R language. Actively developed since early 2003.

`http://www.postgresql.org`

`http://www.joeconway.com/plr`

# Pros

- Leverage people's knowledge and skills
  - statistics/math, database, web are distinct specialties
- Leverage hardware
  - server better able to handle analysis of large datasets
- Processing/bandwidth efficiency
  - why send large datasets across the network?
- Consistency of analysis
  - ensure analysis done consistently once vetted
- Abstraction of complexity
  - keep system understandable and maintainable
- Leverage R
  - rich core functionality and huge ecosystem

# Cons

- PostgreSQL user
  - Slower than standard SQL aggregates and PostgreSQL functions for simple cases
  - New language to learn
- R user
  - Debugging more challenging than working directly in R
  - Less flexible for ad hoc analysis
  - New language to learn

# Creating PL/R Functions

- A little different from standard R functions

```
func_name <- function(myarg1 [,myarg2...]) {
  function body referencing myarg1 [, myarg2 ...]
}
```

- But similar to other PostgreSQL PLs

```
CREATE OR REPLACE FUNCTION func_name(arg-type1 [, arg-type2 ...])
RETURNS return-type AS $$
    function body referencing arg1 [, arg2 ...]
$$ LANGUAGE 'plr';

CREATE OR REPLACE FUNCTION func_name(myarg1 arg-type1
                                     [, myarg2 arg-type2 ...])
RETURNS return-type AS $$
    function body referencing myarg1 [, myarg2 ...]
$$ LANGUAGE 'plr';
```

# Example of Use

```
CREATE EXTENSION plr;

CREATE OR REPLACE FUNCTION test_dtup(OUT f1 text, OUT f2 int)
RETURNS SETOF record AS $$
  data.frame(letters[1:3],1:3)
$$ LANGUAGE 'plr';

SELECT * FROM test_dtup();
 f1 | f2
----+----
 a  |  1
 b  |  2
 c  |  3
(3 rows)
```

# Highlighted Features

- RPostgreSQL Compatibility
- Custom SQL aggregates
- Window functions
- R object $\Rightarrow$ bytea

# RPostgreSQL Compatibility

- Allows prototyping using R, move to PL/R for production
- Queries performed in current database
- Driver/connection parameters ignored; dbDriver, dbConnect, dbDisconnect, and dbUnloadDriver are no-ops

```
dbDriver(character dvr_name)
dbConnect(DBIDriver drv, character user, character password,
          character host, character dbname, character port,
          character tty, character options)
dbSendQuery(DBIConnection conn, character sql)
fetch(DBIResult rs, integer num_rows)
dbClearResult (DBIResult rs)
dbGetQuery(DBIConnection conn, character sql)
dbReadTable(DBIConnection conn, character name)
dbDisconnect(DBIConnection conn)
dbUnloadDriver(DBIDriver drv)
```

# RPostgreSQL Compatibility Example

- PostgreSQL access from R

```
tsp_tour_length<-function() {
  require(TSP)
  require(fields)
  require(RPostgreSQL)

  drv <- dbDriver("PostgreSQL")
  conn <- dbConnect(drv, user="postgres", dbname="plr", host="localhost")
  sql.str <- "select id, st_x(location) as x, st_y(location) as y,
              location from stands"
  waypts <- dbGetQuery(conn, sql.str)
  dist.matrix <-  rdist.earth(waypts[,2:3], R=3949.0)
  rtsp <- TSP(dist.matrix)
  soln <- solve_TSP(rtsp)
  dbDisconnect(conn)
  dbUnloadDriver(drv)

  return(attributes(soln)$tour_length)
}
```

# RPostgreSQL Compatibility Example

- Same function from PL/R

```
CREATE OR REPLACE FUNCTION tsp_tour_length() RETURNS float8 AS $$
  require(TSP)
  require(fields)
  require(RPostgreSQL)

  drv <- dbDriver("PostgreSQL")
  conn <- dbConnect(drv, user="postgres", dbname="plr", host="localhost")
  sql.str <- "select id, st_x(location) as x, st_y(location) as y,
              location from stands"
  waypts <- dbGetQuery(conn, sql.str)
  dist.matrix <-  rdist.earth(waypts[,2:3], R=3949.0)
  rtsp <- TSP(dist.matrix)
  soln <- solve_TSP(rtsp)
  dbDisconnect(conn)
  dbUnloadDriver(drv)

  return(attributes(soln)$tour_length)
$$ LANGUAGE 'plr' STRICT;
```

# RPostgreSQL Compatibility Example (cont.)

- Output from R

```
tsp_tour_length()
[1] 2804.581
```

- Same function from PL/R

```
SELECT tsp_tour_length();
 tsp_tour_length
------------------
 2804.58129355858
(1 row)
```

# Aggregates

- Aggregates in PostgreSQL are extensible via SQL commands
- State transition function and possibly a final function are specified
- Initial condition for state function may also be specified

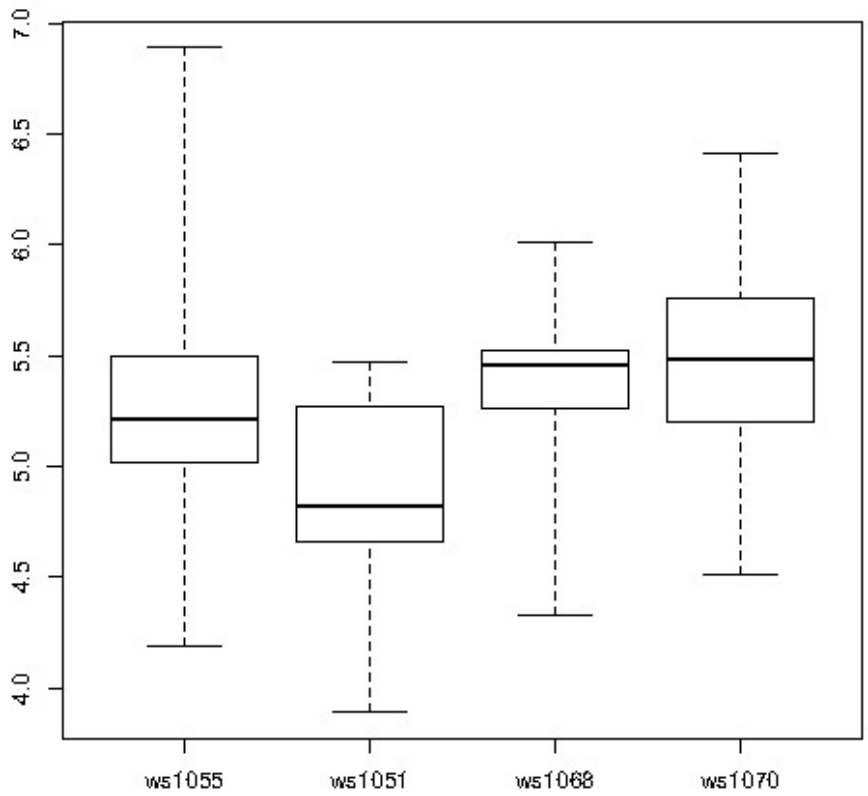
# Aggregates Example

```
CREATE OR REPLACE FUNCTION r_quartile(ANYARRAY) RETURNS ANYARRAY AS $$
  quantile(arg1, probs = seq(0, 1, 0.25), names = FALSE)
$$ LANGUAGE 'plr';

CREATE AGGREGATE quartile (ANYELEMENT) (
  sfunc = array_append,
  stype = ANYARRAY,
  finalfunc = r_quantile,
  initcond = '{}');

SELECT workstation, quartile(id_val) FROM sample_numeric_data
WHERE ia_id = 'G121XB8A' GROUP BY workstation;
 workstation |            quantile
-------------+--------------------------------
 1055        | {4.19,5.02,5.21,5.5,6.89}
 1051        | {3.89,4.66,4.825,5.2675,5.47}
 1068        | {4.33,5.2625,5.455,5.5275,6.01}
 1070        | {4.51,5.1975,5.485,5.7575,6.41}
(4 rows)
```

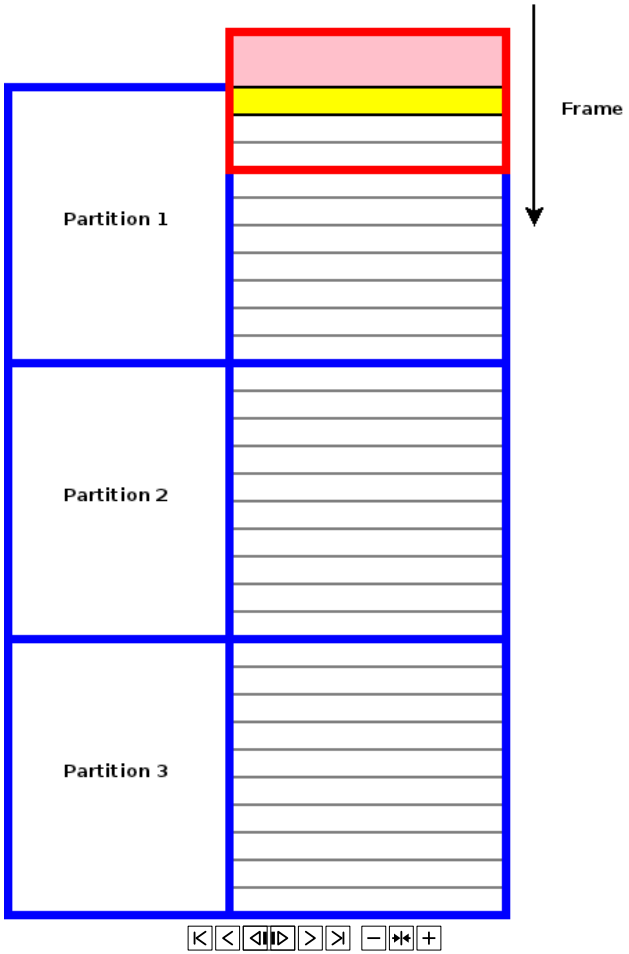
# Aggregates Example - Quartile Boxplot Output



# Window Functions

- Window Functions are available as of PostgreSQL 8.4
- Provide ability to calculate across sets of rows related to current row
- Similar to aggregate functions, but does not cause rows to become grouped
- Able to access more than just the current row of the query result

# Window Functions



# Window Function Example

```
CREATE TABLE test_data
  (fyear integer,firm float8,eps float8);
INSERT INTO test_data
SELECT (b.f + 1) % 10 + 2000 AS fyear,
       floor((b.f+1)/10) + 50 AS firm,
       f::float8/100 + random()/10 AS eps
FROM generate_series(-500,499,1) b(f);

-- find slope of the linear model regression line
CREATE OR REPLACE FUNCTION r_regr_slope(float8, float8)
RETURNS float8 AS $BODY$
  slope <- NA
  y <- farg1
  x <- farg2
  if (fnumrows==9) try (slope <- lm(y ~ x)$coefficients[2])
  return(slope)
$BODY$ LANGUAGE plr WINDOW;
```

# Window Function Example

```
SELECT *, r_regr_slope(eps, lag_eps) OVER w AS slope_R
FROM (SELECT firm AS f, fyear AS fyr, eps,
  lag(eps) OVER (PARTITION BY firm ORDER BY firm, fyear) AS lag_eps
FROM test_data) AS a WHERE eps IS NOT NULL
WINDOW w AS (PARTITION BY firm ORDER BY firm, fyear ROWS 8 PRECEDING);
 f |  fyr |       eps          |     lag_eps        |     slope_r
---+------+--------------------+--------------------+-------------------
 1 | 1991 | -4.99563754182309  |                    |
 1 | 1992 | -4.96425441872329  | -4.99563754182309  |
 1 | 1993 | -4.96906093481928  | -4.96425441872329  |
 1 | 1994 | -4.92376988714561  | -4.96906093481928  |
 1 | 1995 | -4.95884547665715  | -4.92376988714561  |
 1 | 1996 | -4.93236254784279  | -4.95884547665715  |
 1 | 1997 | -4.90775520844385  | -4.93236254784279  |
 1 | 1998 | -4.92082695348188  | -4.90775520844385  |
 1 | 1999 | -4.84991340579465  | -4.92082695348188  | 0.691850614092383
 1 | 2000 | -4.86000917562284  | -4.84991340579465  | 0.700526929134053
```

# Stock Data Example

- get Hi-Low-Close data from Yahoo for any stock symbol
- plot with Bollinger Bands and volume
- requires extra R packages - from R:

```
install.packages(c('xts','Defaults','quantmod','cairoDevice','RGtk2'))
```

# Stock Data Example

```
CREATE OR REPLACE FUNCTION plot_stock_data(sym text) RETURNS bytea AS $$
  library(quantmod)
  library(cairoDevice)
  library(RGtk2)

  pixmap <- gdkPixmapNew(w=500, h=500, depth=24)
  asCairoDevice(pixmap)

  getSymbols(c(sym))
  chartSeries(get(sym), name=sym, theme="white",
                TA="addVo();addBBands();addCCI()")

  plot_pixbuf <- gdkPixbufGetFromDrawable(NULL, pixmap,
                pixmap$getColormap(),0, 0, 0, 0, 500, 500)
  buffer <- gdkPixbufSaveToBufferv(plot_pixbuf, "jpeg",
                character(0),character(0))$buffer

  return(buffer)
$$ LANGUAGE plr;
```

# Stock Data Example

- Need screen buffer on typical server:

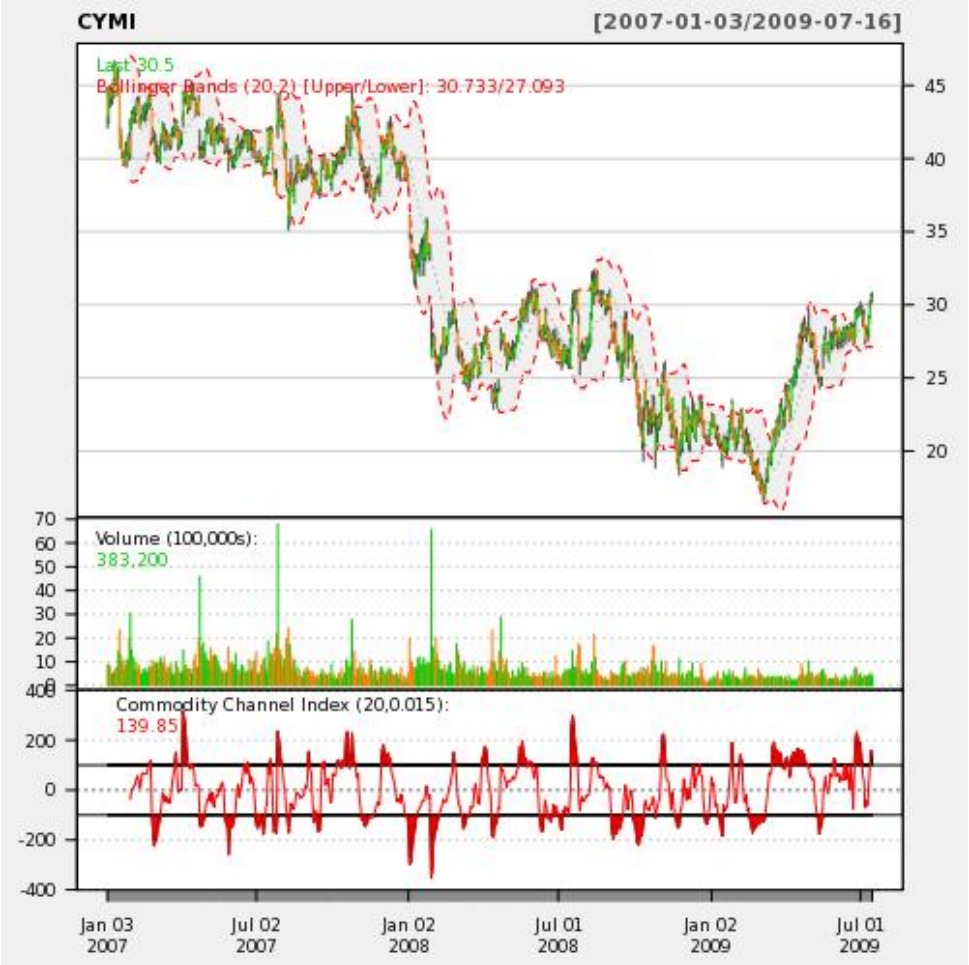
```
Xvfb :1 -screen 0 1024x768x24
export DISPLAY=:1.0
```

- Calling it from PHP for CYMI

```
<?php
$dbconn = pg_connect("...");
$rs = pg_query( $dbconn,
      "select plr_get_raw(plot_stock_data('CYMI'))");
$hexpic = pg_fetch_array($rs);
$cleandata = pg_unescape_bytea($hexpic[0]);

header("Content-Type: image/png");
header("Last-Modified: " .
       date("r", filectime($_SERVER['SCRIPT_FILENAME'])));
header("Content-Length: " . strlen($cleandata));
echo $cleandata;
?>
```

# Stock Data Example - Output



# Questions?

Thank You!

joe.conway@credativ.com
mail@joeconway.com