Postgres OnLine Journal: February / March 2009

An in-depth Exploration of the PostgreSQL Open Source Database

Table Of Contents

A Product of Paragon Corporation
http://www.paragoncorporation.com/
http://www.postgresonline.com/

PostGIS Raster and More

The OSGEO Toronto Sprint

The OSGEO C-Camp Toronto Sprint was fun, although Leo and I couldn't stay for the whole event. I've never seen people close bugs so quickly. Paul Ramsey and Mark Cave-Ayland were on a marathon run in the PostGIS ring. Olivier Courtin was also following not too far behind with SVG bug fixes and so forth. We also discussed the possiblity of having ST_GeomFromGML, ST_GeomFromGeoJSON, ST_GeomFromKml and so forth and what that would entail. It was great to meet Pierre Racine of WKT Raster fame in person and chat with Mateusz and Sandro Santilli via IRC. Frank Warmerdam, the GDAL god came to our table to provide his big two cents about how WKT Raster meta data should be stored, dealing with large RASTERS and other things I didn't understand.

Mark Leslie in Australia did his part too, though he wasn't present -- he would come into IRC when others had fallen asleep. Such is the way with timezones. He has been working on beefing up the curved support in PostGIS. The FOSS 4G 2009 conference will be in Sydney, Australia.

It was nice to be able to put a face to these people I've talked via newsgroups. It was also strange since most of the clients and many of the people we work with we have never met, so the idea of meeting in person has become a very foreign concept for us.

What is in store for PostGIS

We sat at the PostGIS table and a lot of ground was covered. On the PostGIS side, we discussed plans for PostGIS 2.0 -- specifically

1. Making 0 the unknown SRID instead of -1 to conform with OGC standards
2. Introducing Geodetic support and possibly integrating Q3C for indexing support for geodetic.
3. More extensive projection support using CSMap so that we can deal with obscure projections and also utilize the well-known-text representation of spatial reference for importing shape files.
4. Bringing in WKT Raster more into the PostGIS family
5. Tweaking of storage and ability to expand structure to store new types -- e.g. all those X3D surface types needed for CityGML and so forth.
6. Using PostgreSQL 8.3+ typmod support so we can have geometry_columns as a view instead of a real table and encoding all we need in the typmod structure to simplify adding constrained geometry columns
7. Mateusz Loskot has been working on getting PostGIS to compile under Visual C++ in windows and has made a lot of headway. This will hopefully make it easier for windows users to try out cutting edge PostGIS versions and contribute more. Currently windows users wanting the latest and greatest are forced to use MingW, which is a bit of a pain to get going.

WKT Raster

I'm particularly looking forward to WKT Raster. When many think of Rasters in GIS, they think Aerial Imagery. I think aerial too, but what really excites me with Raster is its application in cellular automata, game grids, signal processing, computational biology, and all sorts of other sugar plums overlaid, intersected and differenced in space which admittedly don't necessarily have anything to do with GIS, but everything to do with spatial (by spatial I mean the use of space that need not be geographic). I think the real deal in GIS and more specifically the power of spatial analytical processing, is not GIS but the conventions and concepts it provides which are equally applicable in non-geographic problem domains. The analytical stuff planned and work already done for WKT Raster I find pretty exciting. I'm hoping we can start playing with it in the next month.

Getting more involved

On another note -- this year is going to be very busy for Leo and myself. We are going to be presenting at PGCon 2009 in Ottawa in May - doing a PostGIS Lightning talk and a PostGIS Spatial Query lecture, and another lecture in July at OSCON 2009 in San Jose, and some other new developments which we shall discuss later. So this year will be pretty interesting and exhausting. We are getting deeper into PostGIS on all fronts and if we can get thru this year all in one piece, I think it will be a major milestone for us and hopefully for PostGIS as well.

FUNCTION CACHING

```
IMMUTABLE
STABLE
VOLATILE
```

SECURITY CONTEXT

```
SECURITY DEFINER
```

CONTROL FLOW

```
if foo =='blah':
 do_something()
for x in somearray:
 statements
[x['somefield'] for x in rv]
[1.1*x/(2 + x) for x in range(5)]

if some boolean expression:
 stuff happens

while b < n:
   stuff happens
```

RETURN constructs

```
return somevariable
return somearray_variable
```

Common Constructs

```
import somepackage
#this is a comment
a, b = b, a+b
alist =['Jim', 'Guru', 'x3456']
name, title, phone = alist
alist = [] #declares empty array
adict = {} #declare empty dictionary
adict = {'Jim': 'Guru', 'Jimmy' : 'Intern'}
```

Constants and functions

```
True
False
coerce(somestring, ther)
dict(..)
globals()
float(num_string)
hasattr(object, name)
hash(obj)
int(num_string)
len(somearray_ordict)
long(num_string)
map(function, sequence[, sequence, ...])
pow(x, y [, z])
range([start,] end [, step])
xrange(start [, end [, step]]) (#use for big lists)
round(x,[numdigits])
slice([start,] stop[, step])
split(pattern,string, [maxsplit])
str(object)
zip(seq1[, seq2,...])
somestring.split
```

Operators

```
+ - *
/ // *
** # power operator
& ^ | == #boolean operators
&= ^= |= #compound bool
<< >> #shift operators
<<= >>= #compound shift operators
```

EXCEPTION Handling

```
try:
 stuff happens
 return something
except (IOError, OSError):
 return "an error has happened"

try:
 stuff happens
 return something
except IOError:
 return "an IOError"
except RuntimeError:
 return "runtime error"
except:
 return "have no clue what happened"
```

Common Error States

```
Exception
 StandardError
 ArithmeticError
  FloatingPointError
  OverflowError
  ZeroDivisionError
 EnvironmentError
  IOError
  OSError
 EOFError
 ImportError
 RuntimeError
 SystemError
```

Built-in Objects

```
plpy
 execute(sql) #returns a python dictionary object
 prepare(sql) # returns a prepared plan object
TD["new"] # trigger new data
TD["old"] # trigger old data
TD["when"] # BEFORE, AFTER, UNKNOWN
SD
```

Common Packages and Package Functions

```
os -- chdir(path), chmod(path), listdir(path)
 mkdir(path,[mode]), rmdir(path), unlink(path),
 write(fp, str), path.exists()
math -- pi
sys -- argv, modules,path,stdin,stdout,
 stderr, version, exit(n)
time -- time(), clock(), strftime(format, timetuple), sleep(secs)
```

COSTING

```
COST cost metric
ROWS estimated number of rows
```

Official PostgreSQL 8.3 PL/Python Documentation URL: http://www.postgresql.org/docs/8.3/interactive/plpython.html
Official Python documentation: http://docs.python.org/
We cover only a subset of what we feel are the most useful constructs that we could squash in a single cheatsheet page
commonly used
[1] New in this release.

PLPYTHON FUNCTION SAMPLES

```
CREATE OR REPLACE FUNCTION readfile (param_filepath text)
  RETURNS text
AS $$
 import os
 if not os.path.exists(param_filepath):
  return "file not found"
 return open(param_filepath).read()
$$ LANGUAGE plpythonu;

SELECT readfile('/test.htm');
```

```
--Doing postgresql queries --
CREATE OR REPLACE FUNCTION getmydata (param_numitems integer)
  RETURNS SETOF mydata
AS $$
 #lets us grab the first param_numitems records
 rv = plpy.execute("SELECT pt_id, pt_observation FROM mydata",param_numitems);
 return rv
$$ LANGUAGE plpythonu;

SELECT *
 FROM getmydata(20) As d;
```

```
--Example Using custom classes and local classes --
CREATE OR REPLACE FUNCTION dofunkyplot (param_anum integer)
  RETURNS text
AS $$
 import aplotter
 import sys
 class WritableObject:
        def __init__(self):
                self.content = ''
        def write(self, string):
                self.content = self.content + string
 saveout = sys.stdout
 outbuffer = WritableObject()
 sys.stdout = outbuffer
 #range (param_anum) will return array
 #consisting of 0 to param_num - 1 and formula
 # gets applied to each element
 # [1.1*x/(2 + x) for x in range(2)] -> [0 1.1/3]
 aplotter.plot([1.1*x/(2 + x) for x in range(param_anum)])
 sys.stdout = saveout
 return outbuffer.content
$$ LANGUAGE plpythonu;

SELECT dofunkyplot(n)
FROM generate_series(5,20, 10) As n;
```