Postgres OnLine Journal: July 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 1.4 hot on the heels of PostgreSQL 8.4

PostgreSQL 8.4 has come out, and while I am a bit disappointed that PostGIS 1.4 has not come out for fear that we've missed a bit of the PostgreSQL 8.4 momentum, I am happy that we are nearing closer and just maybe we'll have it out by end next week. We now have a PostGIS 1.4RC1 http://postgis.refractions.net/download/postgis-1.4.0rc1.tar.gz tar ball as well as experimental binary builds of this for windows user's running PostgreSQL 8.3 http://postgis.refractions.net/download/windows/pg83/experimental/postgis/ or PostgreSQL 8.4 http://postgis.refractions.net/download/windows/pg84/experimental/postgis/. Please give both a try.

Working in the Cathedral Really?

As Paul duly noted in his blog entry Working in the Cathedral the model for PostGIS development is morphing, but I wouldn't call this morphing process one that is entirely toward the Cathedral model. Unlike the perceived Cathedral model, I would like to think we will have more frequent releases and beta releases, perhaps parallel experimental builds and most importantly, **more fun**. The main idea being making it much easier for mere mortals and fake mortals to taste test the cookies in the oven while they are cooking. By fake I mean unit tests, build bots, and computer generated people where the fear of destruction is removed. I feel this is the similar model PostgreSQL goes by or is trying to achieve.

Let me just say for one that I respect Eric Raymond a great deal, but I think that while most of his observations about Cathedral and Bazaar are accurate, some of his conclusions are a bit flawed. Hopefully I won't start too many fist fights from this statement. The main distinctions I see between the two and why Bazaar seems to many more efficient has nothing to do with the model at all. Its all about the people. If you get a group of people all passionate about the work they are doing it really doesn't matter which model you are using (Cathedral or Bazaar). Most likely you will succeed. A dispassionate mess of bazaar workers working together will create dysfunctional code that no one wants to use or work on. You'd be much better with a cathedral one where there is one passionate leader. I also think the success of the Bazaar and Open Source in general is probably over exaggerated. When no one is getting directly paid, its hard to quantify a loss. If 100,000 volunteers work on a project that is free and fun, most likely they aren't keeping a timesheet of how much time they are expending. In a cathedral model a lot of your utility is driven by monetary benefit, which means your time is either over exaggerated or at the very least precise. We know we lost $1,000,000 because we paid people $1,000,000 to build this thing.

First where did this cathedral model come from? it came from engineers and scientists. While I'm not a practicing engineer and I just have a degree in mechanical engineering, I still feel very close to my engineering roots. I'm not talking about that ill-defined term pompous people throw around "I've got a BS in Software Engineering", which to me is so badly abused not to be even worth talking about. I'm talking about real engineering; things like Mechanical Engineering, Civil Engineering, and Electrical Engineering. In engineering you do a lot of planning and have stringent requirements of what is acceptable and what is not. You have very seasoned engineers and technicians in the field who have amazing powers of intuition and new engineers and technicians straight out of school who are are honored to work with these people. You have a lot of smoke tests and things of that sort that stress test a system. This is because while building a bridge or plane might be fun, you can't afford to have bridges falling or planes crashing on people. One of those scandals and you are out of business. Yes some things also are not fun but need to be done. Having rules in place and stringent guidelines on what should and how should it be built is inherently not fun.

In engineering things like building virtual models are becoming more respected and more in vogue, because it allows you to test things out without killing real people or destroying real equipment or using real material in the process. It in essense allows you to be more care-free in designing thus making a better, cheaper and more fun project to work on.

In software the cathedral model was designed to get people to work on things that aren't fun. Even with "fun projects" you have a lot of stuff people consider as not fun and this will not change. **In software the cathedral is a self-fulfilling prophecy**. It managed to categorize tasks and projects such as building financial packages, documentation and testing as not fun such that the people who would have enjoyed these things decided -- software is not fun but pays the bills thus creating an institution of dispassionate people interested in the industry for the perceived money and scaring off passionate people on to other fulfilling things.

The bazaar model is an equally self-fulfilling prophecy. It encouraged a hacker mentality focused on only doing "fun stuff that scratches my own itch", that doesn't pay much but is fun. It too managed to give the vision of bleeding edge running code stuff is fun and documentation/testing is not important because people we care about will dig into the pie where they fit. These are the same people that whine about where did all the women go and why people aren't biting even though its free. Not to say that women are less worthy hackers, but that they are more apt to put a price on these harder to quantify things, thus making their utility function less compatible with open source or the bazaar model.

So what's my point. The point is that not only did these models define a concept of fun/not fun, they also defined the very meaning of fun not fun to people. No one wants to work on something they perceive of as not fun unless they are getting a ton of money for

it. Glory is also a kind of fun because no one wants to work on something that they don't get any glory for or monetary income from. You probably don't want to fill your coffers with people just interested in the monetary value and dispassionate about your product, because they will find ways to maximize profit while minimize level of effort they need to do. In the same vein you don't want to grossly undervalue the things that are hard to quantify.

This gets to the point of a more obnoxious problem. This is the devaluation of testing and documentation. These two things are things where people agree its important, but can't put a price on. A cathedral model has an easier time putting a price on this, because part of the documenation is the specification itself that drives the model. In a bazaar we may have a "this will be a cool feature lets throw it in". The tragedy of this is that people put a price of 0 on something they can't price and don't see standing on its own in a grocery list and thus the work of testers and documenters is seemed as useless which makes the work not glamorous and no fun and in a pure volunteer model, you also get no money.

To demonstrate if you were to put a price on say introducing function X into a product - to figure out the value of it -- all you need to do is look at the price of that in a similar product. We need only look at say Oracle to figure out the value of x. However for a simple feature documentation, primarily user documentation, aside from its existence is not important to document it well and testing is a similar thing. How do you price the documentation of a complex feature X? The documentation/testing of X may be more valuable than the programming that went into X or it may not be needed at all. A pure model driven by volunteers has a bit of a harder time overcoming this problem than a paid project unless they can quantify the value of these or make the work more fun and glamorous. This means figuring out the importance of documentation on each item, glamorizing it, encouraging hackers who understand the function into documenting with cathedral like injections :).

Back to Table Of Contents PostGIS 1.4 hot on the heels of PostgreSQL 8.4 Reader Comments

PostGIS 1.4 is finally out and other news

I am very excited to report that we have finally released PostGIS 1.4. We are still preparing the windows binaries UPDATE: Windows binaries are now available and installers which will become available in the coming week for PostgreSQL 8.2,8.3, and 8.4. Below are the details excerpted from Paul Ramsey's postgis news announcement. We also recently came back from OSCON 2009 where we gave a talk on Tips and Tricks for writing PostGIS spatial queries. In that talk we showcased some of the new features of PostGIS 1.4, as well as demonstrating how the new Windowing and Common Table Expressions introduced in PostgreSQL 8.4 simplifies and provides more options for writing PostGIS spatial queries. We'll be making the slides and data available shortly.

RefCardz DZone PostgreSQL Essentials -- stay tuned

On another exciting note, not only are we working on our upcoming Manning book PostGIS in Action, but we have contracted with DZone RefCardZ to do a PostgreSQL Essentials. Recall we had discussed this a while back that how come there is one for MySQL, but none for PostgreSQL and that someone should write one up for PostgreSQL, preferrably someone who is writing a PostgreSQL related book. So I guess that someone would be us.

We are currently finalizing our first draft of this. Sadly we are a little behind on schedule, but hope to make the time up in the coming month. We'll provide more details on sponsorship and availability as the story unfolds. You can expect to see the general essential stuff like, backup, restore, the growing family of PostgeSQL PL/Languages and examples of them, basic architecture, common SQL constructs. In addition we will show case some of the new PostgreSQL 8.4 enhancements.

PostGIS 1.4 news announcement

This is excerpted from PostGIS newsgroup announcment. Yes there really was a lot of meditation and soul searching here, and a bit too much for my liking.

```
July 24, 2009

The PostGIS development team has, after a long period of meditation
and soul searching, released version 1.4.0 of the spatial data
extension for PostgreSQL.

 http://postgis.refractions.net/download/postgis-1.4.0.tar.gz

This new version of PostGIS includes substantial performance
enhancements, more detailed reference documentation, new output
formats (GeoJSON) and an improved internal testing system. PostGIS 1.4
also supports the recent PostgreSQL 8.4 release.

Thanks to everyone who helped by testing during the release candidate process!

Your PostGIS Team

-- Detailed PostGIS 1.4 Release Notes --

  - As of the 1.4 release series, the public API of PostGIS will not
               change during minor releases.

 - Compatibility
  - The versions below are the *minimum* requirements for PostGIS 1.4
  - PostgreSQL 8.2 and higher on all platforms
  - GEOS 3.0 and higher only
  - PROJ4 4.5 and higher only

 - New Features
  - ST_Union() uses high-speed cascaded union when compiled against
               GEOS 3.1+ (Paul Ramsey)
  - ST_ContainsProperly() requires GEOS 3.1+
  - ST_Intersects(), ST_Contains(), ST_Within() use high-speed cached
```

```
              prepared geometry against GEOS 3.1+ (Paul Ramsey)
  - Vastly improved documentation and reference manual
              (Regina Obe & Kevin Neufeld)
  - Figures and diagram examples in the reference manual (Kevin Neufeld)
  - ST_IsValidReason() returns readable explanations for validity
              failures (Paul Ramsey)
  - ST_GeoHash() returns a geohash.org signature for geometries
              (Paul Ramsey)
  - GTK+ multi-platform GUI for shape file loading (Paul Ramsey)
  - ST_LineCrossingDirection() returns crossing directions (Paul Ramsey)
  - ST_LocateBetweenElevations() returns sub-string based on Z-ordinate.
              (Paul Ramsey)
  - Geometry parser returns explicit error message about location of
              syntax errors (Mark Cave-Ayland)
  - ST_AsGeoJSON() return JSON formatted
  - Populate_Geometry_Columns() -- automatically add records to
              geometry_columns for TABLES and VIEWS (Kevin Neufeld)
  - ST_MinimumBoundingCircle() -- returns the smallest circle
              polygon that can encompass a geometry (Bruce Rindahl)

- Enhancements
  - Core geometry system moved into independent library, liblwgeom.
              (Mark Cave-Ayland)
  - New build system uses PostgreSQL "pgxs" build bootstrapper.
              (Mark Cave-Ayland)
  - Debugging framework formalized and simplified. (Mark Cave-Ayland)
  - All build-time #defines generated at configure time and placed in
              headers for easier cross-platform support (Mark Cave-Ayland)
  - Logging framework formalized and simplified (Mark Cave-Ayland)
  - Expanded and more stable support for CIRCULARSTRING,
              COMPOUNDCURVE and CURVEPOLYGON, better parsing,
              wider support in functions (Mark Leslie & Mark Cave-Ayland)
  - Improved support for OpenSolaris builds (Paul Ramsey)
  - Improved support for MSVC builds (Mateusz Loskot)
  - Updated KML support (Olivier Courtin)
  - Unit testing framework for liblwgeom (Paul Ramsey)
  - New testing framework to comprehensively exercise every
        PostGIS function (Regine Obe)
  - Performance improvements to all geometry aggregate
        functions (Paul Ramsey)
  - Support for the upcoming PostgreSQL 8.4 (Mark Cave-Ayland,
        Talha Bin Rizwan)
  - Shp2pgsql and pgsql2shp re-worked to depend on
        the common parsing/unparsing code in liblwgeom (Mark Cave-Ayland)
  - Use of PDF DbLatex to build PDF docs and preliminary
        instructions for build  (Jean David Techer)
  - Automated User documentation build (PDF and HTML) and
        Developer Doxygen Documentation (Kevin Neufeld)
  - Automated build of document images using ImageMagick
        from WKT geometry text files (Kevin Neufeld)
  - More attractive CSS for HTML documentation
        (Dane Springmeyer)
- Bug fixes
  - http://trac.osgeo.org/postgis/query?status=closed&milestone=postgis+1.4.0&order=priority
```

Back to Table Of Contents

Window Functions Comparison Between PostgreSQL 8.4, SQL Server 2008, Oracle, IBM DB2 *Advanced*

PostgreSQL 8.4 has ANSI SQL:2003 window functions support. These are often classified under the umbrella terms of basic Analytical or Online Application Processing (OLAP) functions. They are used most commonly for producing cumulative sums, moving averages and generally rolling calculations that need to look at a subset of the overall dataset (a window frame of data) often relative to a particular row. For users who use SQL window constructs extensively, this may have been one reason in the past to not to give PostgreSQL a second look. While you may not consider PostgreSQL as a replacement for existing projects because of the cost of migration, recoding and testing, this added new feature is definitely a selling point for new project consideration.

If you rely heavily on windowing functions, the things you probably want to know most about the new PostgreSQL 8.4 offering are:

- What SQL window functionality is supported?
- How does PostgreSQL 8.4 offering compare to that of the database you are currently using?
- Is the subset of functionality you use supported?

To make this an easier exercise we have curled thru the documents of the other database vendors to distill what the SQL Windowing functionality they provide in their core product. If you find any mistakes or ambiguities in the below please don't hesitate to let us know and we will gladly amend.

For those who are not sure what this is and what all the big fuss is about, please read our rich commentary on the topic of window functions.

Window functions

There are two kinds of window constructs and they are very similar.

- Window rank type functions. There are a couple of built-in functions defined for these in the ANSI SQL specs. Most common are ROW_NUMBER() and RANK().
- Window aggregates. These are window constructs that use an aggregate function such as SUM, COUNT, AVG and aggregate data a window frame of data relative to the current record or some defined partition the current record belongs in. These are useful for doing rollup sums, moving averages, running totals, full totals that would appear in each child record of the grouping.

To our knowledge, PostgreSQL is the first open source database to introduce ANSI-2003 SQL Windowing support. Firebird database has window functions on their todo, but not released yet. We don't see this feature on MySQL's roadmap.

Both Oracle and IBM DB2 had windowing support before Microsoft SQL Server and PostgreSQL.

Below we have outlined the basic syntactic structure of each of the databases we will be discussing

Oracle 11G	(rank or agg window) Function([arguments]) OVER ([PARTITION BY value/expr, ...] [ORDER BY [SIBLINGS] expr [ASC|DESC], ... [ROWS | RANGE windowing_clause]]) windowing_clauses: INTERVAL 'nn' DAY PRECEDING INTERVAL 'nn' SECONDS FOLLOWING INTERVAL 'nn' MONTH PRECEDING BETWEEN x PRECEDING AND y FOLLOWING BETWEEN x PRECEDING AND y PRECEDING BETWEEN CURRENT ROW AND y FOLLOWING BETWEEN x PRECEDING AND CURRENT ROW BETWEEN x PRECEDING AND UNBOUNDED FOLLOWING BETWEEN UNBOUNDED PRECEDING AND y FOLLOWING column BETWEEN current.column +/- n AND current.column +/- m UNBOUNDED PRECEDING | FOLLOWING value/expr PRECEDING | FOLLOWING

	CURRENT ROW --where x and y are integers denoting position row of record relative to current --where n and m are value ranges in a column • Analytic functions by Example • Example cumulative sums, moving average, centered aggregates
IBM DB2 V. 9	Function([arguments]) OVER ([PARTITION BY value/expr, ..] [ORDER BY expr [ASC|DESC] [ROWS | RANGE windowing_clause]]) windowing clauses: BETWEEN x PRECEDING AND y FOLLOWING BETWEEN x PRECEDING AND y PRECEDING BETWEEN CURRENT ROW AND y FOLLOWING BETWEEN x PRECEDING AND CURRENT ROW BETWEEN x PRECEDING AND UNBOUNDED FOLLOWING BETWEEN UNBOUNDED PRECEDING AND y FOLLOWING UNBOUNDED PRECEDING | FOLLOWING value/expr PRECEDING | FOLLOWING CURRENT ROW --where x and y are integers position row of record relative to current • IBM DB2 example window aggregates
SQL Server 2008	SQL Server does not allow order by with aggregate functions and does not support window framing clauses so you can't do running sums and totals though you can accomplish similar things with grouping and ROLLUP and CUBE constructs. Ranking_Window_function([arguments]) OVER ([PARTITION BY value/expr, ...] [ORDER BY expr [ASC|DESC], ...) ---------- Agg_Window_function ([arguments]) OVER ([PARTITION BY value_expression , ...]) frame clauses: Does not support frame clauses • Rank window functions supported

PostgreSQL 8.4	PostgreSQL does support the window framing clauses but limited set than Oracle and IBM DB2. So you can't specify specific number ranges or row ranges. PostgreSQL supports definition of named windows and reusing these.
	
	`Function([arguments]) OVER`
	`([PARTITION BY value/expr, ..]`
	`[ORDER BY expr [ASC|DESC]`
	`[ROWS | RANGE frame_clause]])`
	`----------`
	`Function([arguments]) OVER named_window`
	
	`frame clauses:`
	`RANGE UNBOUNDED PRECEDING`
	`RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`
	`RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`
	`ROWS UNBOUNDED PRECEDING`
	`ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`
	`ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`
	
	• Built-in window functions

Below is a break down of key window pieces and what is supported or not supported in each.

Item	Oracle 11G	IBM DB2 V.9	SQL Server 2008	PostgreSQL 8.4
Introduced in	Oracle 8i	DB2 6?	SQL Server 2005	8.4
Built-in Window supported functions	CUME_DIST DENSE_RANK FIRST FIRST_VALUE LAG LAST LAST_VALUE LEAD NTILE PERCENT_RANK PERCENTILE_CONT PERCENTILE_DISC RANK RATIO_TO_REPORT ROW_NUMBER	DENSE_RANK (DENSERANK) RANK ROW_NUMBER (ROWNUMBER)	DENSE_RANK NTILE RANK ROW_NUMBER	CUME_DIST DENSE_RANK FIRST_VALUE LAG LAST_VALUE LEAD NTH_VALUE NTILE PERCENT_RANK RANK ROW_NUMBER
Window aggregates with any aggregate (SUM, AVERAGE, COUNT) etc.	Yes	Yes	Yes	Yes
Custom Window Functions (ability to define custom window range like functions)	Yes	No	No	Yes - but only C functions currently supported
Basic Frame Clauses (no numbers)	Yes	Yes	No	Yes
Frame Clauses (with numbered ranges and rows)	Yes	Yes	No	No
Frame Clauses (between specific field values)	Yes	No	No	No
Ability to chain multiple frame clauses	Yes	No	No	No
ORDER BY within a range/ rank function	Yes	Yes	Yes	Yes

ORDER BY within an aggregation e.g. SUM(*) OVER (PARTITION BY ..ORDER BY ..)	Yes	Yes	No	Yes
Interval frame ranges for dates	Yes	No	No	No
Named windows - ability to name a window expression and reference it multiple times in the same query	No	No	No	Yes

Back to Table Of Contents

PostgreSQL 8.4 unnest and generate_series *Intermediate*

In this issue we shall be celebrating the arrival of PostgreSQL 8.4 by showcasing the new treats that PostgreSQL 8.4 has to offer. Although 8.4 has some nice big ticket items like Windowing Functions which we briefly covered numerous times and Common Table Expressions, it also has some small ticket items. These small ticket items while small, are perhaps more useful than even the big ticket ones because they are more commonly used constructs.

In this article we shall introduce the new unnest() function which makes converting an array to a table like structure not only easier, but much more efficient. We will also be covering the new enhancements to our favorite function the generate_series().

In the olden days (like before 8.4) when you wanted to convert an array to a table, you would do something like this.

```
SELECT myarray[i] as element
 FROM  (SELECT ARRAY['johnny','be','good','or','you be bad'] As myarray) as foo
  CROSS JOIN generate_series(1, 5) As i;
```

With the new unnest function we can do:

```
SELECT unnest(ARRAY['johnny','be','good','or','you be bad']) as element;
--or
SELECT element
 FROM unnest(ARRAY['johnny','be','good','or','you be bad']) As element;

--All of which return
```

```
 element
-----------
johnny
be
good
or
you be bad
```

Now while the unnest function has made transforming arrays into tables much easier, you lost the iterator with unnest, so generate_series is still useful under certain conditions. Wait a minute though generate_series has gotten a little bit nicer too. Now you have dates and timestamps.

In the olden days when you wanted a date series that iterates every 10 days up to a certain date, you'd first have to figure out how many sequences you needed or if you were lazy you would overshoot and limit like below:

```
SELECT CAST('2009-01-01' As date) + CAST((i || ' days') As interval) As aday
 FROM generate_series(0,1000,10) As i
 WHERE CAST('2009-01-01' As date) + CAST((i || ' days') As interval) <= CAST('2009-12-01' As date);
```

Now you can do this:

```
SELECT CAST(generate_series(CAST('2009-01-01' As date),
   CAST('2009-12-01' As date), '10 days') as date) As aday;

--Both of which return
```

```
    aday
------------
 2009-01-01
 2009-01-11
 2009-01-21
 2009-01-31
 2009-02-10
 2009-02-20
 2009-03-02
:
:
 2009-11-07
 2009-11-17
 2009-11-27
```

Back to Table Of Contents PostgreSQL 8.4 unnest and generate_series Reader Comments

PostgreSQL 8.4 Faster array building with array_agg *Intermediate*

One of the very handy features introduced in PostgreSQL 8.4 is the new aggregate function called **array_agg** which is a companion function to the **unnest** function we discussed earlier. This takes a set of elements similar to what COUNT, SUM etc do and builds an array out of them. This approach is faster than the old used array_append , array_accum since it does not rebuild the array on each iteration.

Sadly it does not appear to be completely swappable with array_append as there does not seem to be a mechanism to use it to build your own custom aggregate functions that need to maintain the set of objects flowing thru the aggregate without venturing into C land. This we tried to do in our median example but were unsuccessful.

In PostGIS 1.4 Paul borrowed some of this array_agg logic to make the PostGIS spatial aggregates much much faster with large numbers of geometries. So collecting polygons or making a line out of say 30,000 geometries which normally would have taken 2 minutes or more (just accumulating), got reduced to under 10 seconds in many cases. That did require C code even when installed against PostgreSQL 8.4. Though in PostGIS you reap the benefits as far as geometries go even if you are running lower than 8.4.

We had originally thought array_agg was a PostgreSQL only creation, but it turns out that array_agg is a function defined in the ANSI SQL:2008 specs and for one appears to exist in IBM DB2 as well. I don't think Oracle or any other database supports it as of yet.

As we had demonstrated in the other article, we shall demonstrate the olden days and what array_agg brings to the table to make your life easier.

In the olden days prior to 8.4, when you wanted to get the list of employees that reports to a supervisor as an array and also the salaries of these people as an array so you could pass it to a funky R-statistical function to do interesting calculations with it, you would have had one of two choices.

- Create an aggregate function for said R-statistical function, which would be slow if you had a lot of numbers
- Or do an ugly ARRAY subselect thing

As mentioned the aggregate solution is personally nicer and elegant if its a function you call often, but was slow when accumulating large numbers of records in each grouping. Below is a very contrived example that uses our favorite system catalog to demonstrate its use and we are doing it because we are too lazy to create dummy data and this query will work in all databases. In the below for each table in our database we are getting two sets of arrays -- one that has the columns in the table and another that has the distinct types of columns in each table.

```
--The olden days
--Note this is not a terribly interesting example since an array_accum
-- built with array_append would work fine here as the arrays don't get that big for tables
--but just imagine we had something that had 10,000 elements in each grouping
SELECT t.table_schema, t.table_name,
 ARRAY(SELECT CAST(c.column_name AS text)
       FROM information_schema.columns AS c
    WHERE c.table_schema = t.table_schema AND c.table_name = t.table_name) as col_names,
  ARRAY(SELECT DISTINCT CAST(c.data_type AS text)
          FROM information_schema.columns AS c
       WHERE c.table_schema = t.table_schema
       AND c.table_name = t.table_name) As dat_types
 FROM information_schema.tables AS t
GROUP BY t.table_schema, t.table_name;


--The 8.4 way
SELECT c.table_schema, c.table_name,
 array_agg(CAST(c.column_name AS text)) as col_names,
 array_agg(DISTINCT CAST(c.data_type AS text)) As dat_types
 FROM information_schema.columns AS c
GROUP BY c.table_schema, c.table_name;
```

PostgresQL 8.4: Common Table Expressions (CTE), performance improvement, precalculated functions revisited

Intermediate

Common table expressions are perhaps our favorite feature in PostgreSQL 8.4 even more so than windowing functions. Strangely enough I find myself using them more in SQL Server too now that PostgreSQL supports it.

CTEs are not only nice syntactic sugar, but they also produce better more efficient queries. To our knowledge only Firebird (see note below), PostgreSQL,SQL Server, and IBM DB2 support this, though I heard somewhere that Oracle does too or is planning too
UPDATE: As noted below Oracle as of version 9 supports non-recursive CTEs. For recursion you need to use the Oracle proprietary corresponding by syntax.

As far as CTEs go, the syntax between PostgreSQL, SQL Server 2005/2008, IBM DB2 and Firebird is pretty much the same when not using recursive queries. When using recursive queries, PostgreSQL and Firebird use WITH RECURSIVE to denote a recursive CTE where as SQL Server and IBM DB2 its just WITH.

All 4 databases allow you to have multiple table expressions within one WITH clause anda RECURSIVE CTE expression can have both recursive and non-recursive CTEs. This makes writing complex queries especially where you have the same expressions used multiple times in the query, a lot easier to debug and also more performant.

In our article on How to force PostgreSQL to use a pre-calculated value we talked about techniques for forcing PostgreSQL to cache a highly costly function. For PostgreSQL 8.3 and below, the winning solution was using OFFSET which is not terribly cross platform and has the disadvantage of materializing the subselect. David Fetter had suggested for 8.4, why not try CTEs. Yes CTEs not only are syntactically nice, more portable, but they help you write more efficient queries. To demonstrate, we shall repeat the same exercise we did in that article, but using CTEs instead.

We first start by creating our super slow function.

```
CREATE OR REPLACE FUNCTION fn_very_slow(IN param_sleepsecs numeric) RETURNS numeric AS
$$
BEGIN
	PERFORM pg_sleep(param_sleepsecs);
	RETURN param_sleepsecs;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE STRICT ;


--runs in 4524 ms
SELECT fn_very_slow(i*0.5) As firstcall
FROM generate_series(1,5,2) As i;


--runs in 9032 ms - no cache, but in spatial functions (say ST_Distance)
-- we have tried this does sometimes cache and return in 4524ms
SELECT fn_very_slow(i*0.5) As firstcall,
	fn_very_slow(i*0.5)  As secondcallsame
FROM generate_series(1,5,2) As i;



--runs in 9032 ms - no cache
SELECT firstcall,
	firstcall + 1 As secondcalldifferent
FROM (SELECT fn_very_slow(i*0.5) As firstcall
FROM generate_series(1,5,2) As i
) As foo;


--the olden days hack
--runs in 4524 ms - caches
```

```
SELECT firstcall,
      firstcall + 1 As secondcalldifferent
FROM (SELECT fn_very_slow(i*0.5) As firstcall
FROM generate_series(1,5,2) As i
ORDER BY 1 OFFSET 0) As foo;


--The world with 8.4 CTEs
--runs in 4524 ms
WITH
 ctefn AS (
      SELECT fn_very_slow(i*0.5) As slow_calc
            FROM generate_series(1,5,2) As i)

SELECT slow_calc As firstcall,
      slow_calc + 1  As secondcallsame
FROM ctefn;
```

The beauty of even non-recursive CTEs really shows itself when you start doing complex self joins or using the same complex subselect in multiple locations of a bigger query. Observe:

```
--doing self joins the olden days
--the olden days hack
--runs in 9012 ms - caches
SELECT foo1.i As foo1_i, foo2.i As foo2_i, foo1.slow_calc As foo1_call,
      foo2.slow_calc + 1 As foo_2_call
FROM (SELECT fn_very_slow(i*0.5) As slow_calc, i
FROM generate_series(1,5,2) As i
ORDER BY 1 OFFSET 0) As foo1 LEFT JOIN
 (SELECT fn_very_slow(i*0.5) As slow_calc, i
FROM generate_series(1,5,2) As i
ORDER BY 1 OFFSET 0) As foo2 ON foo1.i >= foo2.i;

--doing self join in 8.4 with CTES
-- runs in 4512 ms
WITH
 ctefn AS (
      SELECT fn_very_slow(i*0.5) As slow_calc, i
            FROM generate_series(1,5,2) As i)

SELECT foo1.i As foo1_i, foo2.i As foo2_i, foo1.slow_calc As foo1_call,
      foo2.slow_calc + 1 As foo_2_call
FROM ctefn As foo1 LEFT JOIN ctefn AS foo2 ON foo1.i >= foo2.i;
```

Back to Table Of Contents PostgresQL 8.4: Common Table Expressions (CTE), performance improvement, precalculated functions revisited Reader Comments

Use of OUT and INOUT Parameters *Intermediate*

PostgreSQL has supported what are called Out (output) parameters since version 8.1. We were surprised it has been that long since we always thought of it as a feature from 8.2+ until it recently came up for discussion on PostGIS newsgroup and we decided to investigate how long it has been supported.

What are OUT parameters? These are parameters you define as part of the function argument list that get returned back as part of the result. When you create functions, the arguments are defaulted to IN parameters when not explicitly specified (which means they are passed in and not returned) which is why you sometimes see PgAdmin do something like *IN somevariable variabletype* when you use the function wizard.

You can have INOUT parameters as well which are function inputs that both get passed in, can be modified by the function and also get returned.

As a side note - In 8.4, PostgreSQL was enhanced to allow dynamic sql RETURN QUERY using RETURN QUERY EXECUTE syntax for plpgsql queries and also allow set returning functions being called in the SELECT part for any pl language. In prior versions, this was only a feature of PL functions written in SQL. 8.3 introduced RETURN query which required a static sql statement, but did make things a bit easier.

One of the common use cases for using OUT parameters is to be able to return multiple outputs from a function without having to declare a PostgreSQL type as output of the function. In this article we shall cover all variants of this. We'll just focus on sql and plpgsql for this discussion, since we are not sure to what extent other pl languages (if at all) support IN OUT.

SQL and PLPGSQL examples of OUTPUT parameters - return single record

Below are some examples in plain SQL and their outputs.

```
--returning a single record using SQL
CREATE OR REPLACE FUNCTION fn_sqltestout(param_subject text,
   OUT subject_scramble text, OUT subject_char text)
  AS
$$
   SELECT  substring($1, 1,CAST(random()*length($1) As integer)), substring($1, 1,1)
   $$
 LANGUAGE 'sql' VOLATILE;

SELECT  (fn_sqltestout('This is a test subject')).subject_scramble;

-- Output
 subject_scramble
------------------
 This is a test

 SELECT  (fn_sqltestout('This is a test subject')).*;
 --Output
   subject_scramble   | subject_char
----------------------+--------------
 This is a test subje | T

--Same function but written in plpgsql
--PLPGSQL example -- return one record
CREATE OR REPLACE FUNCTION fn_plpgsqltestout(param_subject text,
   OUT subject_scramble text, OUT subject_char text)
  AS
$$
BEGIN
   subject_scramble := substring($1, 1,CAST(random()*length($1) As integer));
   subject_char := substring($1, 1,1);
END;
   $$
```

```
 LANGUAGE 'plpgsql' VOLATILE;
```

SQL OUTPUT parameters - return multiple records

```
--SQL returning multiple records
CREATE OR REPLACE FUNCTION fn_sqltestmulti(param_subject varchar,
   OUT test_id integer,
   OUT test_stuff text)
   RETURNS SETOF record
  AS
$$
   SELECT test_id, test_stuff
      FROM testtable where test_stuff LIKE $1;
$$
 LANGUAGE 'sql' VOLATILE;

--example
SELECT * FROM fn_sqltestmulti('%stuff%');
--example
--OUTPUT--
 test_id |     test_stuff
---------+--------------------
       1 | this is more stuff
       2 | this is new stuff


--PLPGSQL same using 8.3+ syntax
--OUT takes precendence which is why we prefix the table columns
CREATE OR REPLACE FUNCTION fn_plpgsqltestmulti(
   param_subject varchar,
   OUT test_id integer,
   OUT test_stuff text)
   RETURNS SETOF record
  AS
$$
BEGIN
   RETURN QUERY SELECT t.test_id , t.test_stuff
      FROM testtable As t
   WHERE t.test_stuff LIKE param_subject;
END;
$$
 LANGUAGE 'plpgsql' VOLATILE;
```

INOUT parameters - return multiple records

```
--INOUT return multiple records SQL
CREATE OR REPLACE FUNCTION fn_sqltestmulti_inout(param_subject varchar,
   INOUT test_id integer,
   INOUT test_stuff text)
   RETURNS SETOF record
  AS
$$
   SELECT $2 + test_id, $3 || test_stuff
      FROM testtable
      WHERE test_stuff LIKE $1;
$$
 LANGUAGE 'sql' VOLATILE;

--Example
 SELECT * FROM fn_sqltestmulti_inout('%stuff%',1, 'test');

 --OUTPUT
 test_id |    test_stuff
--------------+----------------------
       2 | testthis is more stuff
       3 | testthis is new stuff

--INOUT same function in plpgsql 8.3+
```

```
--Note its a little odd to look at -- the INOUT param takes
--precendence in naming which is why we prefix the table columns
CREATE OR REPLACE FUNCTION fn_plpgsqltestmulti_inout(
    param_subject varchar,
    INOUT test_id integer,
    INOUT test_stuff text)
    RETURNS SETOF record
   AS
$$
BEGIN
    RETURN QUERY SELECT t.test_id + test_id , test_stuff || t.test_stuff
        FROM testtable As t
    WHERE t.test_stuff LIKE param_subject;
END;
$$
  LANGUAGE 'plpgsql' VOLATILE;
```

Back to Table Of Contents

Reader Comments

PostGIS 1.4 hot on the heels of PostgreSQL 8.4

Joe
There are some people who do value the importance of documentation and testing, without it being glamorized. To wit, early in the Linux project people wrote How-To's to share their knowledge and allow others to participate. Did Bruce Momjiam write the first PostgreSQL book because he foresaw making a ton of money from it or because he innately enjoys teaching? On the testing front, look at the work of David Wheeler.

Open source software development has some characteristics of the "economic calculation problem" (see http://mises.org/econcalc.asp) postulated by Ludwig von Mises with regard to socialism. However, not being able to put a monetary price on an item to be produced doesn't mean it will not be produced (or will only be produced under coercion). And it's not just "glory" (or recognition) that leads humans to act (see http://mises.org/rothbard/mes/chap1b.asp#5._Further_Implications, in particular, the paragraph that starts with "All action is an attempt to exchange a less satisfactory state of affairs for a more satisfactory one.")

Regina
Joe,

all good points.
Regarding my comment about glory. That may be too strong of a term. My point is if you are doing a lot of work for a project and you don't get the sense that your work is being valued by the key people in a project, chances are you are going to leave and stop working for it.

In a paying project you stil care but care a little less because you are getting a monetary value from it.

Bruce documented because he knew he needed to to get buy in from people. When you think about Bruce though -- you don't think of him so much in the light of the guy who first documented how to use PostgreSQL so the common folk can use it. In fact that was probably one of his biggest contributions.

In a similar light Paul Ramsey did most of the initial documentation for PostGIS, but he is not known as the guy who first cared enough to document how to use PostGIS. That is also perhaps one of his biggest as well, because I know for one I wouldn't have started using it. I would have dismissed the project as one too early on or for reckless hackers who don't care about a user community.

I guess my point is even though some people feel these things are important and especially the people that cradle the project while its forming, its not seen as a critical component by most -- because its kind of buried down there. Its sort of a hidden line item that is needed and expected but not valued like raw features. Presumably because a feature can exist without documentation, but documentation about something that doesn't exist is not terribly useful unless it is the specification that programmers go by to build the work and specifications can be so easily flipped into user documentation because they encode in them the test cases to test the work when done :). Such a concept I don't believe exists in a pure bazaar model.

In an open source project often times, if you are not a hacker -- you are simply documenting, you are basically seen as valueless even though your value is probably more important than people coding. This drives away people who enjoy just tinkering with new tools. Sure you'll still have some of these -- but they probably have a mixed personality so are credited for their non-documenter work. You will lose all the purer breeds and that is a real shame since others who find this work less enjoyable will be forced to step up to the plate more.

Then again I would venture to bet there is no such thing as a successful pure bazaar model and even Linux has cathedral like qualities built into it at least nowadays but they are sufficiently light not to be offensive or get in the way of creativity.

Joe
Once any enterprise (as in endeavor, not necessarily a business) reaches a certain magnitude, people tend to specialize and thus it becomes more necessary to coordinate activities (e.g., developers' meetings for planning). Thus, as you say, the bazaar model acquires cathedral-like qualities.

However, what I think still differentiates an open source project from a closed one is the freedom of the individual contributors to pursue their vision of what matters to the product. If someone at Oracle or Microsoft thinks of a cool new feature, they will not be able to add it at their discretion. They'll have to get buy-in from product management or whoever else decides what gets in the next release. And they won't be able to work on it at their leisure if they've been assigned some other priority. Maybe some enterprising souls will still work on their dream feature, but they'll lack any incentives from the company or the project manager to do so.

In an open source project, that same person can pursue development of their cool feature without worrying (as much) about project or product management (if those exist) approval. They will still have to get approval from committers and others who decide on released features, but the bar will be generally much lower.

In a sense it's analogous to traditional warfare vs. guerrilla or fourth-generation warfare. In the former, a general dictates the strategy and the tactical targets. A battalion leader may do it "his way" and even succeed but that's the exception. In 4GW the individual guerrilla units pursue the objectives they deem most desirable.

On the subject of coding vs. documenting/testing being more or less valuable, the current agile or XP approach takes a different tack.

In the traditional approach, you may be required to document first (as in writing a detailed design), but in many (most?) instances people will code first (to the design in their heads) and document later (and any tests will be developed even later, maybe by a QA group). In test-driven development, the emphasis is to first write tests to document expected behavior. I don't know to what extent this practice is prevalent in open source projects or individual contributor habits, but it does change your outlook on what is more valuable.

Regina
Joe,

I totally agree with your above statements. Part of my argument was that while PostGIS is garnering more cathedral like qualities, that doesn't mean we've become super austere and require the level of dictatorship that a hmm Microsoft or Oracle does. We still want to maintain the spirit of freedom and allow people to work on what they want to.

Many patches we will reject because they don't jive well with the rest of our code base or require additional dependencies we don't want to introduce. Just like PostgreSQL does the same. Tom Lane is very critical of patches,and that is a good thing. It makes users confident that there is a certain level of quality/cohesiveness we demand. I would say the same for Linux -- Linux was successful early on because while there was freedom -- people knew there was a Linus Torvalds out there monitoring the patches for their suitability -- e.g. doesn't break an patent laws, doesn't have holes etc. His rejection reasons I would imagine were very well defined such that people couldn't accuse him of being unfair. So he in essence added the light handed Cathedral to the Bazaar. Also most open source projects only allow a few committers (gatekeepers) who accept patches from others.

But my more important point is that I think we are also moving toward a more bazaar model (something that was missing in Paul's blog entry). By that I mean by coming up with better ways of stress testing the system, we can be more reckless in our changes and still maintain a fairly stable code base that people won't be afraid to try. By strengthening our tests we will more likely catch a line of code that breaks 20 other areas of functionality or breaks on one OS. By making documentation of a new feature immediately available as soon as a feature is added and prominent in our documentation, we can get more people excited about testing out these new features.

Coming up with ingenious ways of testing and documenting is a very fun and fulfilling job I think and sadly that is what is missing in many open source projects when people are only respected for the amount of code/features they add to a codebase.

PostgreSQL 8.4 unnest and generate_series

Scott Bailey
Don't forget unnest()'s counterpart array_agg(). They go hand in hand.

Leo
We won't. We were going to cover that in a separate article. Still a ton of stuff to talk about.

Rose
guest blogger invitation

Hello,

This is Rose writing from www.huliq.com. I visited your blog and liked your content.

Would you be interested to send us a guest post on any of the issues related to the topics that you cover in your blog. We will publish it in our site www.huliq.com

In return with each guest blog we will give one link in the author's byline back to your blog. We only ask that the guest post (we prefer it be a news coverage, sources can be Google News, CNN, MSNBC, Yahoo News, BBC and others) be a unique story and not be published in your blog.

HULIQ is indexed by Google News and Google requires that the length of the unique news is at least 5 paragraphs. We desire it to be at least 6 paragraphs if possible. And that all need to be a unique content. Once you send us a new story totally unique we will immediately publish it with you link in it, and within 15 minutes it should be indexed by Google News.

Also, please structure author byline as follows:

author's name:
author's e-mail:
author's blog url:

Please let me know if you may have any questions about www.huliq.com.

If you want to consult the topic with me first that's perfectly fine as well.

Many thanks
ruzik.mail@gmail.com

Postgres OnLine Journal
One of the very handy features introduced in PostgreSQL 8.4 is the new aggregate function called array_agg which is a companion function to the unnest function we discussed earlier. This
takes a set of elements similar to what COUNT, SUM etc do and buil

PostgreSQL 8.4 Faster array building with array_agg

Joe
The "olden days" example gives an error:

ERROR: missing FROM-clause entry for table "c"
LINE 1: SELECT c.table_schema, c.table_name,

I think you just need to "SELECT table_schema, table_name ...".

Regina
Thanks. Corrected. I guess I never learned the lesson to never change an example without retesting it.

Pythian Group Blog
Welcome to the 154th edition of Log Buffer, the weekly review of database blogs. Let's dive right in, shall we?
Oracle
On Radio Free Tooting, Andrew Clarke says, "No SQL, so what?" taking as his keynote something Nuno Souto said: ...

PostgresQL 8.4: Common Table Expressions (CTE), performance improvement, precalculated functions revisited

Kristian Natalis
FirebirdSQL does support CTE since its 2.1 version, months earlier than final 8.4 released

Regina
Kristian,

Thanks for the input. We have corrected the article and also provided links to the Firebird CTE documentation.

Jake Rocheleau
This is great, I haven't learned a lot about PostGres yet but it is something I have a large interest in Database-wise.

Thomas
Oracle supports CTEs as well (since 9.0), but not recursive ones.
The syntax is also more or less the same as for the others.

@Jake:
The name of the product ist "PostgreSQL" or simply "Postgres". The g is never written in capitals