Postgres OnLine Journal: April 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/

Many changes in PostgreSQL

Today was a very eventful day for PostgreSQL. We'll cover these changes in a bit.

Massive Forking of PostgreSQL project

First in PostgreSQL Announcements - David Fetter announces massive forking of the PostgreSQL project in several factions. We now have the following -- so take your pick:

1. Shizzle: High-performance and Feature-Free
2. MaryMary: Compiled with libhaltingproblem
3. Narcona: Painless installation and setup
4. OurThing: Lots of sources, based in Sicily
5. XPostgres: Everybody who's ever worked on Postgres code, back to UC Berkeley and Illustra.
6. Moon/PostgreSQL: Corporate support, as long as it lasts.

I feel this may be good for the community because it is hard to satisfy all these factions in one project. Now perhaps the newsgroups will be a bit calmer.

PgAdmin has come to an end -- make way for OpenPgAdmin

Dave Page announced today that the PgAdmin team received an offer they couldn't refuse from a very big software company yet to be announced. So they are closing PgAdmin and you will soon be able to purchase the services and support contracts from this new company.

Not to worry, Devrim Gunduz, has forked PgAdmin to form **OpenPgAdmin**. You can check out the site here http://openpgadmin.info.

I must say as much as we are saddened to see the PgAdmin group leave for more fun escapades, We are happy we finally have an administration tool that has the word **Open** in its name. If it starts with Open, its got to be open. Now all we need is an OpenPost PostgreSQL fork to go with it. OpenPost I think will be easier to pronounce than PostgreSQL and Postgres and it has Open in its name.

So to add to the list of PostgreSQL project forks, I would like to see another project fork

OpenPost: Its free, open source, fast, feature-rich, easy to use and best of all you can pronounce it and its open.

Back to Table Of Contents Many changes in PostgreSQL Reader Comments

Who needs sports when you have the database industry

The database industry is getting way too action packed for my blood. First we had Sun buying MySQL, then Oracle buying Sun (thus inheriting MySQL and Java) (recall they already owned InnoDb (the MySQL main transactional engine) and they also by the way own BerkelyDB which is the database engine underlying Subversion repository and they also own all those CRMS and ERPs (Peoplesoft and Seibel), and now we have IBM integrating EnterpriseDb into their DB so that it can look like Oracle Db. What next? Perhaps Microsoft will join the party to integrate EnterpriseDb into their SQL Server offering so SQL Server can look like Oracle and better yet a SQL Server for Linux/Unix to complete the circle.

Now why would such a thing be good for Microsoft to think of:

- Microsoft can't compete with Oracle on the Linux/Unix front and many companies are diversifying on their OS
- SQL Server is not the only portion of their selling - but helps sell their other offerings -- CRM/Development Stack/ etc.
- If you look at the portfolio of those running Oracle -- I would venture to bet there are a lot of SQL Server shops running both Oracle and SQL Server who would just love to ditch one to simplify maintenance. SQL Server Mag did a recent poll on what SQL Server users are running.
- What can be more appealing than a SQL Server that can play the Oracle dance and can still fit seamlessly with the vast network empire Microsoft has in place.

Back to Table Of Contents Who needs sports when you have the database industry Reader Comments

What is new in PostgreSQL 8.4

PostgresQL 8.4 beta will be out any day and 8.4 official release will hopefully not be too far behind. As we wait patiently for the official release, Robert Treat has summarized nicely all the new features you can expect in 8.4. PostgreSQL 8.4 is what I like to call an earth-shattering release because it has so many big ticket items in there, but also some long-needed usability features in it.

While we all know about the Windowing functions and CTEs and Recursive CTEs, there are a couple of usability features that we always get beat up on, which I am glad to see will be in 8.4. these are

1. Ability to add new columns to a view with CREATE OR REPLACE without having to drop the view and all the view dependents
2. Case insensitivity module
3. Improved Vacuum performance
4. Common Table Expressions and Recursive Common Table Expressions (CTE), windowing functions - Hubert has an example of this in Waiting for 8.4 - window functions

Now the other niceties and usuability features which are nice but not quite as top of our list as the aforementioned. Note this far from an exhaustive list, but Robert Treat's 8.4 slide presentaton is pretty exhaustive:

- Variadic functions -- these are functions that have default values defined so can be called with varying arguments. To achieve this before you would have had to create a separate function that calls the first and passes in the default arg. NOte this can be done with any pl langauge and in fact we demonstrated its use in PL/Python PL/Python and default parameters.
- All plpgsql language and other non-sql/non-c proc languages that return sets to be called in the SELECT clause. To get around this problem before, you'd create your sophisticated set returning function in plpgsql or python or whatever and then wrap it in an SQL function. No need for that hack anymore. Again we demonstrated this feature in PL/Python Pl/Python for loops and returning sets
- pg_terminate_backend -- this kills a backend PostgreSQL process instead of just cancelling the query running on it as pg_cancel_backend did
- Column level priviledges - Hubert has a good example of this in Waiting for 8.4 - column level privileges .
- Faster Restore -- now Restore can use parallel threads
- RETURN QUERY EXECUTE support in plpgsql
- LIMIT clause can take a subquery -- SELECT a.field1, a.field2 FROM a LIMIT (SELECT COUNT(*)/10 FROM a)
- Make As alias in column SELECT optional as the ANSI SQL Standard allows. So you can now do - SELECT a field1, b field2 .... This is not something we would suggest since we find it makes code hard to read, but does make code that used this regrettable syntax more portably converted to PostgreSQL. It would be nice if this were a flag though in the config that can be turned on since I find it to be bad practice and encouraging bad habits.
- Numerours changes to EXPLAIN to show columns used, maintenance improvements such as dead-locking reporting

Back to Table Of Contents What is new in PostgreSQL 8.4 Reader Comments

Determining size of database, schema, tables, and geometry ***Beginner***

Even though others have blogged about this in the past and its well-documented in the docs, its a frequently enough asked question, that we thought we'd post it here again with a couple of additional twists.

How to determine the size of a database on disk

```
SELECT pg_size_pretty(pg_database_size('somedatabase')) As fulldbsize;
```

How to determine the size of a database table on disk

NOTE: There are two functions in PostgreSQL - *pg_relation_size* and *pg_total_relation_size*. The pg_relation_size just measures the size of the actual table where as the pg_total_relation_size includes both the table and all its toasted tables and indexes.

```
SELECT pg_size_pretty(pg_total_relation_size('someschema.sometable')) As fulltblsize,
pg_size_pretty(pg_relation_size('someschema.sometable')) As justthetblsize;
```

How to determine the size of a database schema

When you are doing a lot of data loading for later massaging to dump into your production tables, you will often generate junk tables in the process and worse yet other people may generate junk tables period. As a general practice, we like to create a schema just for junk. The reason for that is that it is extremely easy to exclude schemas from being backed up and to load individual schemas. So with this approach we create or move all tables we are not quite sure if they are needed or we only need temporarily into a schema called junk, scratch, possiblejunk or whatever you want to call the schema and exclude it from our backups.

Something of the form:

```
CREATE SCHEMA scratch;
ALTER TABLE someschema.joeyplaying SET SCHEMA scratch;
```

After saving the world from junk, you would like to know how much space you have saved your backups from backing up. So you would employ a query something of the form:

```
SELECT pg_size_pretty(pg_database_size('mycurrentdb')) As fullprod,
pg_size_pretty(CAST(pg_database_size('mycurrentdb') - (SELECT SUM(pg_total_relation_size
(table_schema || '.' || table_name)  )
FROM information_schema.tables WHERE table_schema = 'scratch') As bigint)) As  tobebackedup_size,
pg_size_pretty(CAST((SELECT SUM(pg_total_relation_size(table_schema || '.' || table_name) )
FROM information_schema.tables
        WHERE table_schema = 'scratch') As bigint) )  As junk_size;
```

Size of Geometries in PostGIS tables

PostGIS has a companion function for measuring geometry size which is useful when you want to get a sense of how much space your geometries are taking up on disk for a set of records.

```
SELECT ST_Mem_Size(ST_GeomFromText('LINESTRING(220268 150415,220227 150505,220227 150406)'));
```

```
SELECT pg_size_pretty(CAST(SUM(ST_Mem_Size(the_geom)) As bigint) ) as totgeomsum
FROM sometable WHERE state = 'MA';
```

Back to Table Of Contents

How to force PostgreSQL to use a pre-calculated value *Intermediate*

This question is one that has come up a number of times in PostGIS newsgroups worded in many different ways. The situation is that if you use a function a number of times not changing the arguments that go into the function, PostgreSQL still insists on recalculating the value even when the function is marked IMMUTABLE. I have tested this on 8.2 and 8.3 with similarly awful results.

This issue is not so much a problem if function calculations are fast, but spatial function calculations relative to most other functions you will use are pretty slow especially when dealing with large geometries. As a result your query could end up twice as slow. Even setting the costs of these functions to relatively high does not help the situation.

To demonstrate here is a non-PostGIS version of the issue that everyone should be able to run and demonstrates its not a PostGIS only issue.

```
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;
```

Solution:

Our solution to this problem I find kind of ugly, hard to explain, and not ideal. The solution we use is to wrap in a subquery and put an ORDER BY in the subselect. It doesn't seem to matter what that ORDER BY is. You could do ORDER BY 1 and it works though if you have a preferred order, you should use that. The ORDER BY seems to trick the planner into materializing the subselect with the costly function so by the time it hits the main one, it sees the costly calculation as a constant.

Watch what happens when we throw in a meaningless ORDER BY clause

```
--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) As foo;
```

Note if you leave out the ORDER BY the planner may or may not materialize the subquery, but ORDER BY seems to almost guarantee it.

Note this is not optimal because for large datasets, you just want the cached result to be reused. You don't want the result to be materialized since you loose the usefulness of indexes.

If anyone has any thoughts on the matter I would love to hear them since this is a big cause of some frustration when you are trying to run spatial queries that have to return in 3 seconds or less.

Back to Table Of Contents How to force PostgreSQL to use a pre-calculated value Reader Comments

Where is my data and other stuff ***Beginner***

Different Linux distros have their preferred place of where stuff goes and of course the default location on windows is completely different from that too. So there isn't really one default location where you can find PostgreSQL data cluster. Of course user's can pick their locations as well. So what is a casual DBA supposed to do?

The pg_settings table

PostgreSQL has a convenient system

table view called **pg_settings** that stores a lot of information. It stores the location of the data cluster, the pg_hbafile and other conf files. In additon to that you can interogate it to find out information you will find in the postgresql.conf file. Why sift thru that postgresql.conf file (assuming you can already query your postgresql server) when you can find the answers you are looking for with an SQL query?

Below are some queries that list some of the things we find useful

Get listing of all File Locations

As Joe noted: this query only works if you are connected as a super user, though the queries after do not require super user access.

This will tell you where your data cluster is and all your conf files (postgresql.conf, pg_hba.conf, pg_ident.conf, external_pid if you have one).

```
SELECT name, setting
        FROM pg_settings
        WHERE category = 'File Locations';
```

If you are taking advantage of PostgreSQL support for tablespaces, the data file location will not be enough to tell you where all your stuff is. Another handy table to interrogate for this is the pg_tablespace table. If the location of a table space is not in the init cluster location, the spclocation will be filled in with the file path to it.

```
SELECT spcname, spclocation
        FROM pg_tablespace;
```

Listing of memory and buffer settings

This gives you information about all the different memory settings -- work_mem, maintenance_work_mem, shared_buffers, temp_buffers, wal_buffers and whether you have constraint exclusion enabled.

```
SELECT name, setting, unit, category
FROM pg_settings
WHERE name like '%mem%' or name LIKE 'constraint%' or name like '%buffer%'
```

```
ORDER BY name;
```

Greg Smith suggested this query which gives a prettier more understandable look to setting (makes it show in kb/MB) and also included effective cache size -

```
SELECT name, setting, current_setting(name),unit, category
FROM pg_settings
WHERE name like '%mem%' or name LIKE 'constraint%' or name like '%buffer%' or name like 'effective
%'
ORDER BY name;
```

Listing of all categories of settings

Menu of all categories of settings

```
SELECT DISTINCT category
FROM pg_settings
ORDER BY category;
```

Back to Table Of Contents Where is my data and other stuff Reader Comments

Loading and Processing GPX XML files using PostgreSQL *Intermediate*

Simon Greener, wrote an article on how to load GPX xml files into Oracle XMLDB. That got me thinking that I haven't really explored all the XML features that PostgreSQL has to offer and to some extent I've been reticent about XML processed in any database for that matter.

In this article we shall attempt to perform the same feats that Simon did, but with PostgreSQL instead of Oracle XMLDB. Note while we are demonstrating this with a GPX file, the same XPath approach can be used to process any XML file.

PostgreSQL since 8.3 has had ANSI SQL 2003 XML functionality built in. Before 8.3, you could use the xml2 contrib module to achieve the same effect in a not so standards compliant sort of way. In this example we shall demonstrate the built in functionality in 8.3 and above. The key function we will use is the xpath function. XPath is a language used to query XML data and PostgreSQL supports the XPath 1.0 version. The following is a quick primer on XPath that seems useful http://www.zvon.org/xxl/XPathTutorial/General/examples.html. You can also refer to the PostgreSQL XML section of the documentation.

Getting the data

We will use the same sample data Simon used, except sadly we had to change it further because the schema wasn't defined in such a way that PostgreSQL liked or rather I was too stupid to construct the XPath statement in such a fashion that would satisfy the PostgreSQL XML thingy. PostgreSQL seems to require that the schema have a name in addition to a location or at least that is what we concluded. The GPX example Simon had a location but no name. The docs have an example of the form. Where the second argument is an array of 2 dimensional arrays with the first item being the schema name and second the URI for the schema. Also note that the xpath function always returns an array even if there is only one element.

```
SELECT xpath('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>',
                ARRAY[ARRAY['my', 'http://example.com']]);

 xpath
--------
 {test}
(1 row)
```

Suffice it to say, our version is like Simon's revised version except we also stripped off the namespace references since the PostgreSQL XML parser seemed unhappy that the name space defined was never referenced in format xsi:.... or something of that sort . We will be using the xpath version that takes no schema references.

Our revision of Simon's revision can be downloaded from here

The change made is very subtle. Simon had this as the first part

```
<?xml version="1.0"?>
<gpx version="1.1" creator="Toshihiro Hiraoka" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.topografix.com/GPX/1/1"
 xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/1/gpx.xsd">
```

and we changed it to:

```
<?xml version="1.0"?>
<gpx version="1.1" creator="Toshihiro Hiraoka">
```

Getting the data in the database

Simon used Oracle's get LOB fileopen to get the xml file into the db which he calls from an Oracle stored function. When you think about the closest parallel in PostgreSQL, I would say its the lo_* functions that allow import export of files into the db, though that only allows you to import and export files and not read the file. There is also the **pg_read_file** which does what we want, but can only read files from the PostgreSQL init cluster. Of course their are other ways. You could use perl or python or some other language such as PLPerlU that has system file access.

For now we'll just create a folder called **gpxdir** in the PostgreSQL cluster. You can determine the location of your cluster by running as super user

```
SELECT name, setting FROM pg_settings WHERE name='data_directory';
```

Now we'll create a function to mirror Simon's **getClobDocument**, except instead of calling it getClobDocument, we'll call it **getXMLDocument** because it will return an **XML** object instead of a **CLOB** object. Please note -- our getXMLDocument function is marked as **SECURITY DEFINER** because only super users can use the pg_read_file, so to allow regular users access to this, we have this run in the postgres context and then can give rights to this function to those users we want to who may not have super user rights.

```
--create the function to load xml doc
CREATE OR REPLACE FUNCTION getXMLDocument(p_filename character varying)
  RETURNS xml AS
$$
---we set the end read to some big number
-- because we are too lazy to grab the length
-- and it will cut of at the EOF anyway
SELECT CAST(pg_read_file(E'gpxdir/' || $1 ,0, 100000000) As xml);
$$
  LANGUAGE 'sql' VOLATILE SECURITY DEFINER;
ALTER FUNCTION getxmldocument(character varying) OWNER TO postgres;
```

Now to use this function we simply do:

Copy the gpxtestrevised.gpx file into the **gpxdir** and call the below

```
SELECT getXMLDocument('gpxtestrevised.gpx');
```

Next we'll create a table similar to what Simon has called gpx and stuff our xml in there

```
--create table to store  xml docs
CREATE TABLE gpx
(
  object_name character varying(50) NOT NULL PRIMARY KEY,
  object_value xml
);


--insert xml doc
INSERT INTO gpx(object_name, object_value)
VALUES ('gpxtestrevised.gpx', getXMLDocument('gpxtestrevised.gpx'));
```

Unfortunately PostgreSQL even in 8.4 is not as rich as Oracle's offering for XMLDB and doesn't have all that fancy validation schema stuff, though if you try to pull an obviously malformed XML document with getXMLDocument it will tell you you are missing tags and so forth. So we are skipping that section of Simon's and moving straight to the fun part.

```
--Get the metadataname
SELECT (xpath('/gpx/metadata/name/text()', g.object_value))[1] As metadataname
FROM GPX As g;

          metadataname
----------------------
 Manila to Mt. Pinatubo


--Full meta data
SELECT (xpath('/gpx/metadata/name/text()', g.object_value))[1] as Name,
           (xpath('/gpx/metadata/desc/text()', g.object_value))[1] as Description,
 (xpath('/gpx/metadata/copyright/year/text()', g.object_value))[1] as Copyright_Year,
 (xpath('/gpx/metadata/copyright/license/text()', g.object_value))[1] as Copyright_License,
(xpath('/gpx/metadata/link/@href', g.object_value))[1] as Hyperlink,
(xpath('/gpx/metadata/link/text/text()', g.object_value))[1] as Hyperlink_Text ,
(xpath('/gpx/metadata/link/time/text()', g.object_value))[1] as Document_DateTime ,
(xpath('/gpx/metadata/link/keywords/text()', g.object_value))[1] as keywords ,
(xpath('/gpx/metadata/bounds/@minlon', g.object_value))[1] as MinLong,
(xpath('/gpx/metadata/bounds/@minlat', g.object_value))[1] as MinLat,
(xpath('/gpx/metadata/bounds/@maxlon', g.object_value))[1] as MaxLong,
(xpath('/gpx/metadata/bounds/@maxlat', g.object_value))[1] as MaxLat
 FROM GPX AS g;


 --the same stuff Simon got
                  name          |          description           | copyright_year |
  copyright_license       |           hyperlink            |  hyperlink_text   |
document_datetime | keywords | minlong | minlat | maxlong | maxlat
----------------------+------------------------------+---------------+-----
-----------------------+------------------------------+------------------+-
----------------+---------+---------+--------+---------+--------
 Manila to Mt. Pinatubo | This is test data for gpx2shp. | 2004           | http
://gpx2shp.sourceforge.jp | http://gpx2shp.sourceforge.jp | Toshihiro Hiraoka |
                                  |          | -180.0  | -90.0  | 179.9   | 90.0
(1 row)
```

And now for the finale -- we shall pull the way points just as Simon did

```
--Lets extract way points (Simon's is a bit shorter)
-- (the offset here is an ugly hack to force Postgres to use our xml value instead of recopying it as
-- suggested by a commenter to our previous post
--note we were using order by before but OFFSET though still ugly seems cleaner --
--With the offset hack -- this finishes in 895ms.  Without offset hack it takes about 3182 ms (~3 seconds)
SELECT CAST((xpath('/wpt/name/text()', wayp.pt))[1] As varchar(20)) As Name,
        CAST(CAST((xpath('/wpt/@lon', wayp.pt))[1] As varchar) As numeric) As longitude,
        CAST(CAST((xpath('/wpt/@lat', wayp.pt))[1] As varchar) As numeric) As latitude,
        CAST(CAST((xpath('/wpt/ele/text()', wayp.pt))[1] As varchar) As numeric) As Elevation
FROM (SELECT (xpath('/gpx/wpt',g.object_value))[it.i] As pt
FROM (SELECT generate_series(1,array_upper(xpath('/gpx/wpt',g.object_value),1)) As i
FROM GPX As g WHERE object_name = 'gpxtestrevised.gpx') As it
 CROSS JOIN (SELECT object_value
        FROM GPX WHERE object_name = 'gpxtestrevised.gpx') As g OFFSET 0) As wayp;
```

```
  name  |   longitude    |   latitude   | elevation
--------+----------------+--------------+------------
 001    |  121.043382715 | 14.636015547 |  45.307495
 002    |  121.042653322 | 14.637198653 |  50.594727
 003    |  121.043165457 | 14.640581002 |  46.989868
```

```
 004    |  120.155537082 | 14.975596117 |  38.097656
 005    |  120.236538453 | 15.037303017 | 147.687134
 006    |  120.236548427 | 15.037305867 | 145.043579
 007    |  120.237012533 | 15.038105585 | 160.905151
 008    |  120.237643858 | 15.038478328 | 165.231079
 009    |  120.238984879 | 15.038991300 | 173.882935
 010    |  120.239190236 | 15.039099846 | 166.192383
 011    |  120.241263332 | 15.040223943 | 175.324829
 012    |  120.247956365 | 15.042621084 | 186.860474
 013    |  120.253084749 | 15.043179905 | 208.730347
 014    |  120.254095523 | 15.043297336 | 211.374023
 015    |  120.254105665 | 15.043296246 | 213.296631
 016    |  120.247880174 | 15.042568864 | 189.984863
 017    |  120.246971911 | 15.042486135 | 187.100830
 018    |  120.245966502 | 15.042233923 | 185.418579
 019    |  120.244808039 | 15.041693626 | 181.092651
 020    |  120.244476954 | 15.041558258 | 179.410400
 021    |  120.243841019 | 15.041360026 | 178.689453
 022    |  120.241488637 | 15.040351683 | 176.526489
:
:
```

Back to Table Of Contents

Reader Comments

Many changes in PostgreSQL

greg
why not just rename postgresql fork to 'open' :D
we would 'own3' virtually all projects ;)

Gurjeet Singh
This article really made my heart miss a beat! What would I do without Postgres?

A good 'April fool's' prank I must say.

Who needs sports when you have the database industry

Robert Young
SQLServer, since the 2005 version, can run functionally equivalent to Oracle. M$ added SNAPSHOT isolation, which is functionally the same (although some will argue about it) as MVCC, the concurrency mechanism used by Oracle/ Postresql.

The IBM/Postgresql relationship is clearly defensive by IBM. Rather than adding SNAPSHOT or declarative MVCC to 9.x, they did a quick and dirty IMS port, calling it pureXML. M$ did the smart thing.

IBM is gambling that their mainframe DB2 will win. It won't. Mainframe DB2 is a thin veneer over VSAM, and is good only as a sql parser between COBOL and VSAM files. It is rarely used by COBOL programmers to do anything relational. I know, I had to deal with such folks for some years.

The Intel chip (and may be AMD tagging along), with *nix and multi-core/processor boards will blow the mainframe away. Only legacy dinosaur companies (does Financial Services industry sound familiar) will go down that road. In time, they will be made obsolete by true innovators. IBM/DB2 is in deep doodoo.

What is new in PostgreSQL 8.4

(s)zymon
typos: "LIMIT calsue can take[...]"
calsue ? ;> "[...]default arg. NOte this can[...]" ;)

Leo
Thanks - fixed

How to force PostgreSQL to use a pre-calculated value

Heikki Linnakangas
It would be straightforward to put a caching function in front of the real function. The caching function would check if there's a precalculated result for the given arguments in the cache, and return the result from the cache if so. Otherwise, run the real function, and put the result in the cache.

Regina
Heikki,

That's an interesting thought. So are you thinking creating a function such as

fn_cache_me(somekey,fn_slow_function(args))

and the cache would keep say a max of 1000 records and pull out if it sees a match or is there an easier way to do this. somekey the users would guarantee is unique foa given call.

I guess what I'm finding strange is that sometimes it cahces and sometimes it doesn't and not quite sure what controls that.

For example as one users pointed out the construct in PostGIS

ST_Distance(a1,b1) As dist1, ST_Distance(a1,b1) As dist2

almost always caches, though for the above example I gave with pg_sleep it doesn't. Though not sure if that is because ST_Distance is implemented as a C function.

However ST_Distance(a1,b1) + 1 as dist1, ST_Distance(a1,b1) + 2 As dist2 doesn't seem to cache the first call.

David Fetter
Checking for immutability and repeated calls with the same arguments could be an optimization target. Other factors might include the function cost, which I'm guessing is (or should be) set high for at least some of the PostGIS functions.

In 8.4, you'll be able to use CTEs as a way to materialize whole result sets including such function calls.

Regina
David,
I tried the immutable and also set function cost high to about 1000 and that didn't seem to help at all in the tests I have run. though it does help in use of && verses costly intersects vs I think what index it applies first when btree indexes are options.

moltonel
Using "OFFSET 0" instead of "ORDER BY 1" has the same effect, without the overhead of sorting. It's still an undesirable trick-the-planner hack, though :(

Lars
The main use of the caching is for repeated calls over large sets. So the caching is performed across rows!

\timing
select fn_very_slow(1) from generate_series(1,100) as i;
Time: 1002.019 ms

Even though fn_very_slow is called 100 times the overall time is still just about 1s.

select fn_very_slow(1), fn_very_slow(1) from generate_series(1,100) as i;
Time: 2003.386 ms

Here the function is executed twice (although we are selecting from a 100 row table)

Oracle (9i and 10g at least) behaves exactly the same.

Postgres OnLine Journal
Simon Greener, wrote an article on how to load GPX xml files into Oracle XMLDB. That
got me thinking that I haven't really explored all the XML features that PostgreSQL has to offer
and to some extent I've been reticent about XML processed in any datab

---

Where is my data and other stuff

Joe
The first query with the 'File Locations' only seems to work if you're connected as 'postgres'.

Greg Smith
For the memory query, that will show you all the values in terms of the server units, which can be confusing--you have to pay attention to whether it's in 1k or 8k units. You can improve that to show the human-readable versions as well, like this:

SELECT name, setting, current_setting(name),unit, category FROM pg_settings WHERE name like '%mem%' or name LIKE 'constraint%' or name like '%buffer%' or name like 'effective%' ORDER BY name;

Note that I also added in effective_cache_size, which is one of the more critical memory-related parameters missing from the list you had.

Melvin Davidson
A slight correction here. pg_settings is a system VIEW, not a table. That is clearly stated in the documentation. The structure is detailed in the following url.

http://www.postgresql.org/docs/8.3/interactive/view-pg-settings.html

Leo
Melvin, thanks we've updated

Leo
Thanks Joe, Greg, and Melvin for all the constructive comments. We've incorporated these into the article.
