Postgres OnLine Journal: July 2008

An in-depth Exploration of the PostgreSQL Open Source Database

Table Of Contents

From the Editors

More Database Comparisons

PostgreSQL Q & A

How to Inherit, Unherit and Merge Inherit *Intermediate*

Basics

YUM 2: Almost Idiot's Guide to upgrade from PostgreSQL 8.3.1 to 8.3.3 *Beginner*

SQL Idiom Design Patterns *Beginner*

Product Showcase

Portable GIS: PostgreSQL and PostGIS on a USB Stick

Special Feature

PostgreSQL Pg_dump Pg_Restore Cheatsheet Overview *Beginner*

Reader Comments

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

More Database Comparisons

As many people who know us know we sit on several camps especially when it comes to databases. The camps we sit on are growing rather than shrinking. While we do have our favorites, we understand that peoples needs and comfort levels are different from ours and we try to take that into consideration when making recommendations to people. The only thing that is generally true about the clientele we consult for is that they fit one of the following features:

- Very minimal bureaucratic structure - this generally rules out most fortune 500 companies and shall we say smaller companies who are too bureaucratic for their own good
- Dot com startups/Niche product developers who are looking to keep costs down to a minimum without too much fuss and are trying to produce a product to change the world
- Small companies who have a relatively low IT budget, but are predominantly windows-based
- Mid-sized companies predominantly windows-based or departments with decent IT staff, who are looking for something their staff can easily maintain rather than simply keeping licensing costs down

It has come up as a topic of discussion, now that SQL Server 2008 is coming out soon and with its new fangled geodetic spatial support, how does this change things?

The short answer is - not much except to increase awareness of spatial databases and to give us more business. As part of our due diligence work we have put together a comparison of the 3 databases spatial functionality - Cross Compare SQL Server 2008 Spatial, PostgreSQL/PostGIS 1.3-1.4, MySQL 5-6 to compliment our Cross Compare of SQL Server, MySQL, and PostgreSQL

We expect to be using both PostgreSQL and SQL Server 2008 databases for our spatial and non-spatial work. In fact we may be using both in the same project in some cases since each has complimentary functionality that the other is lacking and PostgreSQL has the advantage that its raw SQL functionality is on par with SQL Server and it can scale out/run/or for our product producing clients (install on client servers without additional cost) on more platforms via cloud/hosting without incurring software licensing costs.

We are predominantly SQL Server consultants. It must be noted we are less than we once were, but if truth be known 80% of our database revenue still comes from SQL Server Support and application development, and the remaining 20% is split between MySQL and PostgreSQL with MySQL still leading that chunk.

It is true that the predominant reason for us starting to use PostgreSQL was for the superior spatial functionality at zero cost and for that we put up with a lot of annoying things about PostgreSQL (which for the most part, most of those issues no longer exist in PostgreSQL 8+).

Along the way we discovered things about PostgreSQL that we liked a lot more than we liked in SQL Server and MySQL.

The fact that lots of rich .NET work surrounding the PostgreSQL and Linux community is happening Francisco's Npgsql group work, better overall support of PostgreSQL on windows, Mono.NET making it possible to run .NET on Linux, and various efforts of .NET in GIS world SharpMap.Net, ZigGIS, MapDotNet, Manifold, ArcGIS Server - each with a strong PostGIS support, the integration of SQL Server 2005+ with .NET and Windows platform is not as strong of a selling point as it once was a year or two ago.

Back to Table Of Contents

How to Inherit, Unherit and Merge Inherit *Intermediate*

A lot of this information is nicely tucked away in the PostgreSQL docs in http://www.postgresql.org/docs/8.3/interactive/ddl-inherit.html, but since the docs are so huge and rich, one may tend to miss these things.

While there are numerous interesting use cases for the PostgreSQL inheritance structure, one of the key reasons people use it is for table partitioning strategies.

How do you make a stand-alone table a child of another table?

The first question that comes to mind is why would you ever need a table to adopt another table. There are 2 reasons that come to mind.

- When you are loading huge amounts of data especially of a read only nature - its often convenient to not have that table be visible to your applications until you are done with the loading process. So you may want to make it a child after the loading.
- Your tables seemed fairly unrelated when you started out and then one day you realized you really were talking about apples and apples and need to report on them together at a higher level. One situation like this to give a somewhat real-world perspective - lets say you developed a timesheet app for an organization and each department insisted on having their own version of the app and each along with the basic fields needed to track some additional ones of their own. Then higher forces came in and said *I need to know what everyone is doing, but I don't need to see all that other crap they keep track of.*. Two options come to mind - create a bunch of views that union stuff together or institute a *round-up-the-children-and-adopt-them* program.

This fits into one of the categories of things that PostgreSQL lets you do that PgAdmin III doesn't have a graphical way to let you do it. If you try to inherit in PgAdmin III from a table that already exists, that option is just greyed out. So you have to resort to DDL SQL statements. Luckily its fairly trivial. Well this really only works for PostgreSQL 8.2+. I don't think PostgreSQL 8.1 and below supported INHERIT/NO INHERIT in the ALTER TABLE statement.

```
--Before Micromanagers
CREATE TABLE icecreammakers.timesheet
(
  ts_date date NOT NULL,
  ts_hours numeric(5,2) NOT NULL DEFAULT 0,
  ts_icecreameaten character varying(50),
  ts_employeename character varying(50) NOT NULL,
  CONSTRAINT pk_timesheet PRIMARY KEY (ts_date, ts_employeename)
);

--Micromanager comes on the scene
CREATE TABLE micromanagers.timesheet
(
  ts_date date NOT NULL,
  ts_hours numeric(5,2) NOT NULL DEFAULT 0,
  ts_employeename character varying(50) NOT NULL,
  CONSTRAINT pk_timesheet PRIMARY KEY (ts_date, ts_employeename)
)
WITH (OIDS=FALSE);

ALTER TABLE icecreammakers.timesheet INHERIT micromanagers.timesheet;

--Micromanagers still can't read the information
--unless we give them read-rights into the child tables
--this has some interesting use cases as well
--for implementing row level security
GRANT SELECT ON TABLE icecreammakers.timesheet TO micromanagers;



--Micromanagers discover that icecreammakers are
--more efficient than other employees
```

```
--they start to encourage eating of icecream during work
-- and then to keep track of which icecream flavors employees are eating

ALTER TABLE micromanagers.timesheet ADD COLUMN ts_icecreameaten character varying(50);

--Note the message you get when doing this
--NOTICE:  merging definition of column "ts_icecreameaten" for child "timesheet"


--Ice cream makers union goes on strike
--Ice cream production goes down - other employees unhappy
--Ice cream makers win to put in their contract
--not to be spied on by micro managers
ALTER TABLE icecreammakers.timesheet NO INHERIT micromanagers.timesheet;
```

Back to Table Of Contents

YUM 2: Almost Idiot's Guide to upgrade from PostgreSQL 8.3.1 to 8.3.3 *Beginner*

In our April Issue An Almost Idiot's Guide to PostgreSQL YUM we covered using the new PostgreSQL Yum repository to install the PostgreSQL 8.3.1 release on Fedora, RedHat Enterprise, and CentOS. We also received numerous useful feedback from others on issues they ran into and how they overcame them. The blog comments are definitely worth a read.

Now that 8.3.3 has come out, many of you should be considering upgrading if you haven't already since there are a couple of bug fixes as outlined in http://www.postgresql.org/docs/8.3/static/release-8-3-2.html, http://www.postgresql.org/docs/8.3/static/release-8-3-3.html, and for those running 8.3.0 you will need to reindex your tables after as noted in http://www.postgresql.org/docs/8.3/static/release-8-3-1.html. If you are running version 8.3.1 and above then this is a fairly painless upgrade that just requires you to backup your data as a precautionary measure, but doesn't require a dump reload.

In addition to the full dump or instead of the full dump if you are short on space or patience, you may want to selectively dump out important databases with a

- pg_dump -i -h localhost -p 5432 -U postgres - F c -b -v -f "\somepath\somedb.backup" somedb
- pg_dump -i -h localhost -p 5432 -U postgres - F c -b -v -f "\somepath\postgres.backup" postgres

With a compressed backup of the database vs. the pg_dumpall (well its good to have a pg_dumpall version too), you can restore portions of a database from the backup file. A pg_dumpall backup doesn't alot you that convenience, but does alot you the advantage of being able to restore your whole PgServer.

NOTE: If you made custom changes to your postgresql startup service script /etc/rc.d/init.d/postgresql make sure to back it up. The upgrade wipes it out with the prepackaged version.

```
cp /etc/rc.d/init.d/postgresql /etc/rc.d/init.d/postgresql.080705
```

Now to upgrade your YUM install from 8.3.1 to 8.3.3 do the following:

1. **yum list postgresql**

   this should return output something of the form

```
rhel-i386-server-5        100% |=========================| 1.4 kB    00:00
pgdg83                    100% |=========================| 1.9 kB    00:00
rhel-i386-server-vt-5     100% |=========================| 1.4 kB    00:00
rhn-tools-rhel-i386-serve 100% |=========================| 1.2 kB    00:00
Reading repository metadata in from local files
primary.xml.gz            100% |=========================| 1.5 MB    00:01
################################################## 4206/4206
primary.xml.gz            100% |=========================|  25 kB    00:00
################################################## 90/90
primary.xml.gz            100% |=========================|  24 kB    00:00
################################################## 131/131
Installed Packages
postgresql.i686                           8.3.1-1PGDG.rhel5      installed
Available Packages
postgresql.i386                           8.3.3-1PGDG.rhel5      pgdg83
```

2. **yum update postgresql**

   It should then download the updates and say somethong to the effect Total download size: 8.3 M Is this ok [y/N]:

   y
3. **Note the below may fail if you have made custom changes to your postgresql service script such as the location of the database cluster.** Make sure to restore your original and then run the below.
   ```
   service postgresql restart
   ```
4. If you had installed pgagent as well before also do a
   ```
   service pgagent restart
   ```

As a final step - connect to your postgresql server via PgAdminIII or psql or whatever tool of choice you prefer and run the following to verify you are running the latest version:

```
SELECT version();
```

Back to Table Of Contents

SQL Idiom Design Patterns ***Beginner***

Programming Design Patterns define recommended approaches of solving common application problems. Within design patterns is a subset of design patterns called Idioms. Idioms you can think of as a strategy for expressing recurring constructs or if you will sub-problems and often take advantage of the special features of a language. They tend to be specific to a programming language and can not be reused in other languages they were not specifically designed for. To demonstrate the differences lets compare two design patterns we commonly use.

Macro Design Pattern

Design Pattern:

For core pieces of your application try to create a buffer between those components you can change and those you can't.

When developing applications, we often use industry standard abstraction layers and add our own abstraction layers on top of them. For example when building .NET applications, our pages never inherit directly from System.Web.UI.Page, but instead from an intermediary class we have control over.

The mind set behind this is that an industry standard abstraction layer is one that has taken into consideration all sorts of optimization concerns, been thoroughly tested by mass numbers of users. It in essense has a lot of good black box stuff we don't want to reinvent. Since it is one used by the industry, you have little control over its direction and it may not completely handle your particular use-case. The easiest way to customize it for your specific needs is to put your own abstraction wrapper around it and make it a rule never to call the inner abstraction. So we start off just putting a wrapper around the parts we plan to use that do nothing but wrap. We may then decide later on we need the wrapper to do more. The buffer also allows us to rip out the inner layer and replace with a more efficient one.

Case in point - Lets say you are trying to bullet-proof your application against SQL Injection attacks, if all SQL goes thru your abstraction layer - you can put a validation check around it that verifies the SQL conforms to the unique policies of your application before it gets passed to the sub abstraction layer. You can also do all sorts of interesting things like create your own dialect of SQL or translate Transact-SQL dialect to PSQL that then gets fed to your final database abstraction layer.

Micro Design Pattern: AKA the idiom

The way in which different languages choose to navigate and roll data constructs is different. Procedural languages use while loops, for loops and so forth. This is such a common thing to do for Relational Databases that SQL has constructs INNER JOIN, LEFT JOIN, CROSS JOIN, WHERE, GROUP BY etc. So something that seems so seemingly procedural in nature is completely abstracted away in SQL idioms.

A person used to summarizing data with :

```
orders.sort('salesperson,customerid')
for each order in orders
        if order.salesperson != cursalesperson then
                if cursalesperson > '' then
                        tallystats[] = array(curtotal, custcount)
                end if
                curtotal = orders.total
                cursalesperson = order.salesperson
                curcustomer = order.customerid
                custcount = 1
        else
                curtotal += orders.total
                if curcustomer != order.customerid then
                        custcount += 1
                        curcustomer = order.customerid
                end if
        end if
loop
if cursalesperson > '' then
        tallystats[] = array(curtotal, custcount)
end if
```

May be puzzled to see the SQL idiom equivalent of

```
SELECT salesperson, SUM(total) As salestotal,
               COUNT(DISTINCT customerid) As custcount
        FROM orders
        GROUP BY  salesperson
        ORDER BY salesperson
```

As you demand solutions for more complicated versions of these questions, you quickly realize the benefit of these SQL idioms. At the end of the day we choose our languages because of the ecosystem built around them (e.g. what platforms they can run on, tools to build with, libraries to use) and fitness of the idioms it provides to solve our problems.

More SQL Idioms - the FOR 1..n Loop Idiom Equivalent

This one is by far one of our favorites. I'll call this the Dummy Number trick.

Basic Use Case:
You need a for loop that gives you a sequence of dates, numbers etc. SQL does not support FOR loops. To achieve this feat you use a table - lets call this the Dummy Number table that has nothing but a sequence of numbers say between 1 and 200000. PostgreSQL has the handy generate_series() function which creates this dummy number table for you on the fly.

Below are a couple of concrete examples:

Creating repeating labels - say business cards
Lets say you need to create a set of business cards, but you don't want to waste paper - so you want your cards to start at position 11, where you had left off and you want to print 10 labels for the employee 'Joey'. The below will generate blanks for the first 10 slots and repeating Joey fields for the next 10.

```
SELECT CASE WHEN n > 10 THEN e.first_name ELSE '' END As first_name,
        CASE WHEN n > 10 THEN e.last_name ELSE '' END As last_name,
        CASE WHEN n > 10 THEN e.title ELSE '' END As title,
        CASE WHEN n > 10 THEN e.phone ELSE '' END As phone
FROM employees e CROSS JOIN generate_series(1,20) n
WHERE e.first_name = 'Joey'
ORDER BY n;
```

Create a report that gives you sales for each day of year 2007 even for days that have no sales

```
SELECT pdays.doy, SUM(o.order_total) As sales_total
FROM (SELECT CAST('2007-01-01' As date) + n As doy
               FROM generate_series(0,364) n ) pdays
                       INNER JOIN orders o ON o.sale_date = pdays.doy;
GROUP BY pdays.doy
ORDER BY pdays.doy
```

PostGIS - this trick comes in very handy in geometry manipulation
Create a new data set with all lines in your table cut up such that no line is greater than 100 units. Assumes no line segment is longer than 10000x100 units long.

```
SELECT field1, field2, ST_Line_Substring(the_geom, 100.00*n/length,
  CASE
        WHEN 100.00*(n+1) < length THEN 100.00*(n+1)/length
        ELSE 1
  END) As the_geom
FROM
  (SELECT sometable.field1, sometable.field2, sometable.the_geom,
  ST_Length(sometable.the_geom) As length
  FROM sometable
  ) AS t
CROSS JOIN generate_series(0,10000) AS n
WHERE n*100.00/length < 1;
```

PostgreSQL Array Idioms

PostgreSQL has idioms that are fairly unique to it. Its extensive support of arrays as first class data-types introduces a lot of

interesting use-cases. Below is another example using PostGIS for nearest neighbor search.

For each building - list the 3 closest police precincts that are within 5000 units (our units are in feet so ~ 1 mile away)

```
        SELECT m.building_name, m.nn_precincts[1] As nn_precinct1,
                m.nn_precincts[2] As nn_precinct2,
                m.nn_precincts[3] As nn_precinct3
                FROM
                (SELECT b.building_name, b.the_geom,
                        ARRAY(SELECT p.precint_name
                                        FROM police_precints p
                WHERE ST_DWithin(b.the_geom, p.the_geom, 5000)
                ORDER BY ST_Distance(b.the_geom, p.the_geom)
                LIMIT 3) As nn_precincts
FROM buildings b  ) m
ORDER BY m.builidng_name
```

Back to Table Of Contents SQL Idiom Design Patterns Reader Comments

Portable GIS: PostgreSQL and PostGIS on a USB Stick

First this is a windows only package, but nevertheless sweet. In our article What can PostgreSQL learn from MySQL? we complained about the fact that there is nothing like Server2GO pre-packaged with PostgreSQL. Low and behold comes this thing called Portable GIS 1.2 which can be downloaded from http://www.archaeogeek.com/blog/portable-gis/. This is similar in architecture to Portable Apps. Its a suite of applications you can run from your USB drive without having to reboot your windows computer.

I'm not sure if a similar thing exists for Linux, but would be nice to know if it does. Note: all the packages this portable tool set comes with work on Linux and most started life on Linux, so it seems to me it should not be too hard to make a Linux port of this if it doesn't already exist. Also most of these tools work on Mac OSX as well so a similar package can be made for Mac OSX.

The package is a bit hefty at 460 MB and is still alpha quality, but comes packaged with sample data and all any beginning or experienced GIS user needs to carry along with them. To boot it would provide a good intro to increasing the PostgreSQL family. Unfortunately we don't have a virgin windows computer that doesn't have some of these packages installed to try this out on.

Now this bag of treats comes with the following packages:

- PostgreSQL 8.2 - would be nice if this were bumped up to 8.3 and they packaged PgAdmin III in here.
- PostGIS 1.1 - would be nice if they could bump this to 1.3.3. We already covered PostGIS in PostGIS for geospatial analysis and mapping in our first issue.
- Desktop - GRASS - which is a popular GIS analysis and visualization tool. Works with PostGIS among other formats.
- Desktop - QGIS (version 0.10 with GRASS plugin) - this is a popular GIS visualization and editing tool very handy for viewing and editing PostGIS geometries
- Desktop - gvSIG (version 1.1) - this is another popular open source GIS viewing/editing tool funded by governement of Spain with a similar feel we are told to ESRI ArcView. It supports editing and viewing of PostGIS layers similar to QGIS. The other thing that is interesting about this desktop app is that it totes a mobile edition for running on Mobile devices.
- There are some other noteworthy open source GIS desktop tools left out of this package - uDig and OpenJump come to mind, but those tend to cater to more advanced users.
- Command Line - FWTools (GDAL and OGR toolkit, version 2.10). This is a data conversion, data transformation, data loading tool that deals with both relational spatial data and raster data. We already described this tool and how to use it in GDAL OGR2OGR for Data Loading.
- Web Apps - Mapserver - this is a very popular open source mapserver originally developed by University of Minnesota and adopted over the years by other developers. It can run on Apache or IIS , is pretty light-weight, and conforms to OpenGIS standards such as Web Map Service and Web Feature Service standards.
- Web Apps - OpenLayers - this is a javascript mapping toolkit developed by local MetaCarta that can be used for querying multiple disparate mapping services such as Google Maps, Virtual Earth, standard WMS, loading KML and displaying in a single map view. Very handy if you have your own custom data layers you want to dish out via MapServer, GeoServer, or FeatureServer and overlay on top of commodity layers such as those provided by Google, Yahoo, Virtual Earth. Its our favorite mapping toolkit because of its versatility and relative lightness.
- Tilecache, FeatureServer - these are also MetaCarta open source contributions - TileCache caches image tiles to allow you to develop your own google map like tile server. FeatureServer is basically a REST-based GIS Feature server that allows you to feed various disparate datasources and edit them - e.g. PostGIS, Oracle, etc. Its a nice complement to OpenLayers. We briefly described the concepts behind REST in Showcasing REST in PostgreSQL - The PreQuel.
- Geoserver - this is a very popular Open Source Web mapping server. It is built on top of Java and Java Servlets. There is a lot of overlap between what Geoserver and Mapserver do. Both support WMS and WFS standards. Geoserver is a bit harder to configure and needs a servlet engine such as Tomcat to work and in general is not as light. It does torte WFS-T capability which means you can edit geometries with it e.g. via OpenLayers which you can't do with Mapserver.

Back to Table Of Contents Portable GIS: PostgreSQL and PostGIS on a USB Stick Reader Comments

PostgreSQL Pg_dump Pg_Restore Cheatsheet Overview ***Beginner***

Backup and Restore is probably the most important thing to know how to do when you have a database with data you care about.

The utilities in PostgreSQL that accomplish these tasks are pg_restore, pg_dump, pg_dumpall, and for restore of plain text dumps - psql.

A lot of the switches used by pg_dump, pg_restore, pg_dumpall are common to all three and on rare cases, the switches used by each overlap but mean different things. pg_dump and pg_restore are complementary. You use pg_dump to do hot backups of a database and pg_restore to restore it either to another database or to recover portions of a database.

Rather than trying to keep track of which switch works with which, we decided to combine all into a single cheat sheet with a column denoting which utility the switch is supported in. Pretty much all the text is compiled from the --help switch of each.

Below is a Thumbnail view of the PostgreSQL 8.3 Dump Restore cheat sheet that covers PostgreSQL 8.3 pg_dump, pg_dumpall, pg_restore utilities.

PDF Portrait version 8.5 x 11" of this cheatsheet is available at PostgreSQL 8.3 Dump Restore 8.5 x 11 and also available in PDF A4 format and HTML.

Back to Table Of Contents

POSTGRESQL 8.3 PG_DUMP, PG_DUMPALL, PG_RESTORE CHEAT SHEET

pg_dump, pg_dump_all, pg_restore are all located in the bin folder of the PostgreSQL install and PgAdmin III install.

pg_dump dumps a database as a text file or to other formats.

Usage: `pg_dump [OPTION]... [DBNAME]`

pg_dumpall extracts a PostgreSQL database cluster into an SQL script file.

Usage: `pg_dumpall [OPTION]...`

pg_restore restores a PostgreSQL database from an archive created by pg_dump.

Usage: `pg_restore [OPTION]... [FILE]`

General options: (D - pg_dump, R - pg_restore , A - pg_dumpall)

R	-d, --dbname=NAME	connect to database name (pg_dump uses this to mean inserts)
D R A	-f, --file=FILENAME	output file name
D R	-F, --format=c|t|p (p only for pg_dump, psql to restore p)	specify backup file format (c = compressed, t = tar, p = plain text)
D R A	-i, --ignore-version	proceed even when server version mismatches
R	-l, --list	print summarized TOC of the archive
D R	-v, --verbose	verbose mode
D R A	--help	show this help, then exit
D R A	--version	output version information, then exit
D	-Z, --compress=0-9	compression level for compressed formats

Options controlling the dump / restore: (D - pg_dump, R - pg_restore, A - pg_dumpall)

D R A	-a, --data-only	restore only the data, no schema
D	-b, --blobs	include large objects in dump
D R A	-c, --clean	clean (drop) schema prior to create (for pg_dumpall drop databases prior to create)
D	-C, --create	(D) include commands to create database, (R) create the target database
D A	-d, --inserts	dump data as INSERT commands, rather than COPY
D A	-D, --column-inserts	dump data as INSERT commands with column names
D	-E, --encoding=ENCODING	dump the data in encoding ENCODING
A	-g, --globals-only	dump only global objects, no databases
R	-I, --index=NAME	restore named index
R	-L, --use-list=FILENAME	use specified table of contents for ordering output from this file
D R	-n, --schema=NAME	dump/restore only objects in this schema
D	-N, --exclude-schema=SCHEMA	do NOT dump the named schema(s)
D A	-o, --oids	include OIDs in dump
R	-O, --no-owner	skip restoration of object ownership
R	-P, --function=NAME(args)	restore named function
A	-r, --roles-only	dump only roles, no databases or tablespaces
D R	-s, --schema-only	dump/restore only the schema, no data
D R A	-S, --superuser=NAME	specify the superuser user name to use for disabling triggers/and dumping in plain text
D R	-t, --table=NAME	(D) dump the named table(s), (R) restore named table
A	-t, --tablespaces-only	dump only tablespaces, no databases or roles
R	-T, --trigger=NAME	(R) restore named trigger
D	-T, --exclude-table=TABLE	(D) do NOT dump the named table(s)
D R A	-x, --no-privileges	(D) do not dump privileges (R) skip restoration of access privileges (grant/revoke)
D R A	--disable-triggers	disable triggers during data-only restore
D R A	--use-set-session-authorization	use SESSION AUTHORIZATION commands instead of OWNER TO commands
R	--no-data-for-failed-tables	do not restore data of tables that could not be created
R	-1, --single-transaction	restore as a single transaction
D A	--disable-dollar-quoting	disable dollar quoting, use SQL standard quoting

Connection options:

```
  -h, --host=HOSTNAME      database server host or socket directory
  -p, --port=PORT          database server port number
  -U, --username=NAME      connect as specified database user
  -W, --password           force password prompt (should happen automatically)
  -e, --exit-on-error      exit on error, default is to continue

If no input file name is supplied, then standard input is used.
```

pg_restore Example Use

restore whole database
```
pg_restore --host=localhost --dbname=db_to_restore_to --username=someuser /path/to/somedb.backup
```
restore only the schema (no objects)
```
pg_restore --schema-only=someschema --dbname=db_to_restore_to --username=someuser /path/to/somedb.backup
```
restore only a specifically named schema's data: note the schema has to exist before hand
```
pg_restore --schema=someschema --dbname=db_to_restore_to --username=someuser /path/to/somedb.backup
```

Get a listing of items in backup file and pipe to text file (only works for tar and compressed formats)
```
pg_restore --list backupfilepath --file=C:/somedb_list.txt
```

pg_dump, pg_dumpall Example Use

dump database in compressed include blobs show progress
```
pg_dump -i -h someserver -p 5432 -U someuser -F c -b -v -f "/somepath/somedb.backup" somedb
```
dump database in sql_ascii encoding
```
pg_dump -i -h someserver -p 5432 -U someuser -E sql_ascii -F c -b -v -f "/somepath/somedb.backup" somedb
```
backup pgagent schema of postgres db in plain text copy format, maintain oids
```
pg_dump -i -h someserver -p 5432 -U postgres -F p -o -v -n pgagent -f "C:/pgagent.sql" postgres
```
dump all databases - note pg_dumpall can only output to plain text
```
pg_dumpall -i -h someserver -p 5432 -U someuser -c -o -f "/somepath/alldbs.sql"
```

http://www.postgresonline.com

Reader Comments

SQL Idiom Design Patterns

David Fetter

Re: generating reports over all the days in a year, the LEFT JOIN idiom below is clearer and more efficient:

```
BEGIN;
CREATE TABLE foo AS SELECT date '2007-01-01' + floor(365 * random())::int AS d, md5(s.i::text) AS v FROM generate_series
(1,1000) AS s(i);
SELECT s.i, count(foo.*) FROM generate_series(timestamptz '2007-01-01', timestamptz '2007-12-31', interval '1 day') AS s(i)
LEFT JOIN foo ON s.i=foo.d GROUP BY s.i;
ROLLBACK;
```

Also, re: arrays, the array_accum() aggregate is handy for a lot of situations :)

Portable GIS: PostgreSQL and PostGIS on a USB Stick

Archaeogeek

As the author of the package, many thanks for taking the time to download it and try it out!

I would like to respond to one or two points though- firstly all the packages are designed to run on the stick with no installation on your hard drive. This means that a few packages, such as pgadmin3 probably won't work. Not sure about uDig or OpenJump. Adding extra packages wouldn't help the download size much either!

I have thought about making versions for linux and macosx, but, at least at this stage, it's not top priority. That's partly because I believe linux users (and mac users to some extent) are more used to open source software, and are familiar with somewhat esoteric installation and configuration methods- so they don't need someone to do the hard work for them. I thought of this as getting open source GIS to the windows masses without them being scared off during the installation part. I know how that feels- I nearly gave up on a number of occasions early on!

Thirdly- about the alpha-quality- feedback is always appreciated and I hope that people will report any problems or improvements to me. I do have some ideas for making it look more professional, but it's not my day job though so these things take time...

Regina

Jo,

You are welcome. It worked fine for me so far, but I wanted to test it on a virgin install to be fair.

Are you sure you need PgAdmin III to be installed to work. I remember way back I would just extract the files to a file share and run them from there but that was a couple years ago.

I just uninstalled my PgAdmin III and tried running it from a file folder and it worked fine. But this isn't a fair test since I may have the whole dependency libraries etc. on my machine. I'm going to find someone nearby that I can stick this USB stick in that I know doesn't use open source and see if it works. I'm also going to copy over the PgAdmin III files to the USB to test if it can work without installation.

It would be a nice addition since windows people tend to be command line scared.

This is truly a great addition and I appreciate the great generosity involved in putting in the time to do this.

inglese

how did you manage to make postgresql portable ?
