Postgres OnLine Journal: March 2008

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/

Journal Changes Etc

Welcome to our third Issue (March 2008) of Postgres Online Journal. We folded January and February together because February is a short month and we felt we had already covered quite a bit of ground. This also gives us time to fill March.

We have made a couple of changes to the underlying PDF output structure that we would like to cover.

- We now include Reader comments as an appendix at the end of the PDF version with a Bookmark back to the commented article since it seems people made a lot of useful comments that provided additional information to the topics we discussed.
- It turns out we have a lot of Non-U.S readers. Our stats suggest US readers make up about 25% of our population followed by Germany, Poland, Italy, Japan, France, Spain, UK, and Canada. I'm actually not sure what paper sizes the other nations use, but we have created an A4 version as well to make UK people happy.

Our January/February 2008 (Issue 2), can be downloaded from these links 8 1/2 by 11" and A4

In this issue as mentioned we hope to cover creating a lite Representational State Transfer (REST) application to showcase the new Full Text and XML features introduced in 8.3. Someone suggested we create a Java based server version to compliment our proposed ASP.NET and PHP versions since a lot of PostgreSQL users are Java Programmers. It has been a while since we've programmed with Java Servlets and JSP so not sure if we'll embarass ourselves too much by attempting such an example, but we'll think about it.

We are looking forward to PostgreSQL 8.3 and have started to test out 8.3 RC2 with our existing apps. So far so good, except for some little glitch we had using PostGIS in conjunction with UMN Mapserver. That particular problem seems to be more an issue with the UMN Mapserver Postgis driver with the way its reading the version number in PostgreSQL than anything else (the RC seems to be throwing it off). This issue only affects our more complicated view-based layers and we suspect will be a non-issue when PostgreSQL 8.3 has a bonafide version name e.g. 8.3.0 instead of 8.3 RC2. Aside from that we have noticed speed improvements among other things.

Check out Stefan Kaltenbrunner's Feature Matrix described here and displayed here. It is really quite a useful chart for those thinking of upgrading or wondering why they should or planning to move over from some other DBMS and waiting for a specific feature to be available in Postgres before they can make the jump.

Back to Table Of Contents

PostgreSQL 8.3 is out and the Project Moves On

PostgreSQL 8.3 is out

As many have said - PostgreSQL 8.3 was released on February 4th, 2008 and has numerous enhancements. Listing of features can be found at PostgreSQL 8.3 release notes, and has been mentioned ad-nauseum by several Postgres bloggers. Robert Treat has provided a nice round-up of blog entries that demonstrate various 8.3 enhancements in his PostgreSQL Blog's 8.3 Feature Round-Up. As a side note, the new EnterpriseDb funded Stack Builder feature for windows provides a nice complement for getting add-ons to PostgreSQL.

Horizon of PostgreSQL

Many PostgreSQL contributors are very proud of the fact that PostgreSQL is an open source project and therefore can not be bought like MySQL which is an open source product made by a commercial company. I'm not sure general PostgreSQL users really care that much about this. I suspect that many think

- yah - and Microsoft can be bought, Oracle can be bought, IBM can be bought - who is big enough to buy them and like they will kill off their prize cows
- and if Oracle, IBM or Microsoft one day were to give away non-crippled versions of Oracle 11G, SQL Server 2008, DB2 to leverage their other holdings (perhaps slightly unrealistic), what would that mean to PostgreSQL or MySQL?
- Can a community grow without money pumping into it and if it is not growing does it F*** matter that it is an open source project?

To many, the fact that PostgreSQL can't be bought and that its BSD licensed leaves some nagging questions?

- What is the incentive for one to cooperate and not create an incompatible fork with lots of money behind it to make it better than the main project?

  In the end I think what most users care about are 3 things.

  - Where is the project now? - things like what OSes does it support, if you are using shared hosting are their lots of hosters that provide it so you are not tied to one hoster, consulting support, training support, features, speed, ease of use, apps that can work with it etc.
  - Where is the project going? - future growth of aforementioned items
  - How fast is it getting there? - do I have to wait a year, 2 years, 3 years, my lifetime, in my dreams to see improvements I care about.

  How high the project ranks in each of these areas is the most important and will drive users to support the project and everything else is just debateable catalyst. When you think about PHP, .NET, Java, FireFox, Apache, do you even think about whether they can or can not be bought. I know we don't much. The success of those tools I think demonstrates that the little guy, even if he knows little about IT really does matter. Microsoft realized that a long time ago, MySQL realized that, which is why they have been so successful. If it does what we need it to do, is easy to use, and doesn't cost an arm and a leg, that's how we pick our tools. Everything else is just philosophical snobbery. MySQL still has an advantage in the hosting arena since pretty much any Linux host has it preinstalled. Its just there, low hanging fruit (aside from the confusing licensing), so just use it.

  In the past, I think PostgreSQL ranked pretty poorly in these areas - e.g. it didn't work natively on Windows, had to compile yourself, wasn't available on many shared hosts (still is case, but the story is getting better), and it had some pretty annoying user-friendly problems (it still does to some extent which we shall go into later, but is much better than when we started using it), and it had extremely limited funding which slowed its movement. Because it had so few users and marketing behind it - it was basically a geek-only friend and therefore few used it and cared to fund it. Today I would say PostgreSQL's pace of achievement is faster than any database (open source or commercial) and the more people relying on it for mission critical applications, the faster that accelaration will become. That key fact *faster acceleration*, future growth is perhaps the number one reason why we started to use it for projects where we had a choice of database over our past favorites. It may not do everything we need it to do today (heck none of the databases we have come across do everything we need or would like them to do), but it is getting there quicker than anything else.

  #### Why is BSD License good for PostgreSQL?

  I don't think the BSD license works for all projects, but for large commodity type projects such as PostgreSQL, I think it works well. The reason: When you have a large complex project with lots of contributors, and is so bread and butter to everything that anybody does, its a pain to fork and then try to reintegrate all the goodies other people have added in later versions especially if you are not in the DBMS building business. Its much easier to work right on the main path so you know when you get back the new version - its got all your enhancements and everyone else's enhancements in it and no need to cut in your changes each and every time. The BSD also means that if you choose to keep some piece of the pie for yourself (which is great for Integrators and DBMS builders) - you don't need to worry about the GPL police coming after you. For small projects in niche markets with few updates - the incentive to fork is much higher.

  #### Other incentive to cooperate

  PostgreSQL is beginning to garner quite a few high-profile company/org use - Skype, Sun, Affilias, SourceForge, Sony Games, and Federal Governements to name a few and these entities are not in the DBMS building business. This means their incentive to pool thier money and invest in missing parts of the PostgreSQL core, instead of handing over millions of licensing dollars to the likes of IBM, Microsoft, and Oracle is high. These companies provide a great marketing tool that no money can buy. The more features PostgreSQL piles on, the more of a no-brainer this decision is. This trickles down to consulting service money, training support, ISP and Integrator support, confidence in users *It is alive* and building a whole ecosystem of applications that work with it, funding for missing parts and the whole thing just snowballs into a huge feedback loop that grows with each new user as more people rely on this piece of software. The fact that it is BSD licensed is attractive to businesses and research institutions alike. One can see this in the numerous side projects going on that are built on top of the PostgreSQL core.

  #### What is the deal with DBMS providers contributing?

As many have noticed, there are a couple of DBMS providers that contribute to the PostgreSQL group - e.g. GreenPlum BizGres, and EnterpriseDB are perhaps two of the most glaring contributers. They build a DBMS that is not a forked version, but one built ontop of the main core, but with added features. What is in it for them to contribute? Two thoughts come to mind for us

- The idea of a DBMS only company is non-existent. Even if you look at the likes of Oracle, IBM, and Microsoft SQL Server, they have large consulting arms. Same case for EnterpriseDB and GreenPlum. In the case of EnterpriseDB and GreenPlum - I suspect their ratio between consulting/support contract vs. selling of the product is even higher than the big commercial arms. This means they have a vested interest in seeing PostgreSQL succeed.
- Even as a DBMS company - its always nice to take contributions from others and if a change you make affects the core, its easier to keep this in the main path and keep some goodies for yourself on the side.
- Another prediction, as Open source software becomes more and more pervasive the ratio between consulting dollars vs. money made from selling software will grow exponentially especially in high-end complex commodity products such as databases. This is a drastic shift in the IT ecosystem.

What does PostgreSQL 8.4 and future versions have in store?

One of the great things about PostgreSQL, like many other Open source projects/products - is that you don't need to wait 3-5 years to see changes and then have this gigantic thing that you need to engulf as you do with the likes of Microsoft, Oracle and IBM commercial databases. That is perhaps the most attractive thing about it. Nevertheless, there are still some annoying facets about it today that make it more difficult to use in certain cases than Microsoft SQL Server or MySQL for example.

- You can't use CREATE OR REPLACE VIEW when adding columns to a view or changing data types of a view. If you need to add a column or change the data type of a field in a view that other views depend on, its a real pain because you need to do a DROP CASCADE and recreate all the dependency views. Hopefully this will change.
- UPDATE: Rob Treat corrected us on this. Indeed PostgreSQL handles this nicely. It appears in version 8.0 PostgreSQL introduced the USING syntax to ALTER column of the form

  ```
  ALTER TABLE table ALTER COLUMN col TYPE new_data_type USING some_function_call_to_cast_with(col);
  ```

  . We'll add this to the Q&A section of this issue. While you can now change data types of columns in tables - you are somewhat limited as to what you can change them to instead of the Microsoft SQL Server way of - yah you can change it to anything, but if cast of existing data fails, it will rollback. This is a very touchy topic since there are so many ways where lazy casting can go wrong.
- The CASE sensitivity thing.
- Lack of In place upgrade for medium to major upgrades.
- Windowing functions - As Joshua Drake mentioned in PostgreSQL at Southern California Linux Expo. This will be a nice feature and looks like it might make it into 8.4, but frankly it is not something most people use on a day to day basis. For those unfamiliar with Windowing Functions which was introduced in SQL:2003 spec and currently supported by SQL Server 2005, (I presume Oracle and presume DB2 too) - these are things like ROW_NUMBER(), DENSE_RANK, RANK, CUME_DIST, MOVING AVERAGE using something like OVER() to define a window. General apps don't have much of a need for these unless they are computation intensive. On a related note - it would be nice if PostgreSQL supported ROLLUP and CUBE functions.

Hubert Lubaczewski had a similar rant a couple of months ago in What should be changed in PostgreSQL and while we didn't agree with half the things he said and were on the **Thank god PostgreSQL doesn't support that misfeature** side, we do feel his pain in other areas.

Check out the PostgreSQL 8.4 wishlist for items that are on the brains of Postgres developers.

Back to Table Of Contents

Moving tables from one schema to another *Intermediate*

Question: How do you move tables and views from one schema to another?

Often times when you start a new database, you put all your tables and views in the public schema. For databases with few tables and all relatively commonly grouped data, the benefits of using schemas for logical groupings vs. the downside of having to reference it with the schema name is more trouble than its worth.

As time goes by and with organic growth, this simple database you had that does one thing suddenly starts doing a lot of other things you hadn't initially planned for. Now to keep your sanity you really need the benefit of schemas for logical groupings. How do you retroactively do this? The answer is not quite as easy as one would hope. Ideally you would want to do a RENAME from public.sometable to newschema.sometable, but that doesn't work.

Answer:

UPDATE:
As Tom Lane and David Fetter have noted - 8.1 and above introduced a

```
ALTER TABLE name SET SCHEMA new_schema
```

command, which is documented in 8.1-8.3 ALTER TABLE docs so the below code is unnecessary for PostgreSQL 8.1 and above.
Fiddling directly with the raw PG Catalog is generally a bad thing to do and its very likely we forgot some steps below. Apologies if we encouraged any bad habits.

During our search, we did find a pg function in the newsgroups, submitted by Chris Traverse that moves tables from one schema to another and that more or less seems to do it. As many have noted in the comments of this article, this function is missing lots of parts. Below is our attempt to fill in these missing parts for pre-PostgreSQL 8.1 installs. PostgreSQL 8.1 should use the ALTER TABLE SET SCHEMA approach instead. We also needed a function that would also correct geometry_columns meta table for postgis spatial database and ALTER TABLE does not do that. Below are our 2 revised functions based on the above. NOTE we had to change it from LANGUAGE sql to LANGUAGE pgsql because we needed to verify geometry_columns table existed before trying to update it. In doing so since pgsql supports alias names and sql doesn't we were able to discard the $1, $2 etc and have more meaningful names for variables. We were also able to simplify the SQL update statements by declaring the old and new schema ids once.

```
-- Pre 8.1 version - USE at your own risk since we didn't have an old install to test this on
CREATE OR REPLACE FUNCTION cpmove_relation(param_tblname character varying,
        param_source_schema character varying,
        param_dest_schema character varying)
  RETURNS boolean AS
'
-- param_tblname is the table name
-- param_source_schema is the source schema
-- param_dest_schema is the destination schema
DECLARE
    new_schema_oid oid;
    old_schema_oid oid;
    tblname_oid oid;
BEGIN
    new_schema_oid := (SELECT oid FROM pg_catalog.pg_namespace
            WHERE nspname = param_dest_schema);
    old_schema_oid := (SELECT oid FROM pg_catalog.pg_namespace
            WHERE nspname = param_source_schema);

    tblname_oid := (SELECT oid FROM pg_catalog.pg_class
            WHERE relname = param_tblname AND relnamespace = old_schema_oid);

    IF new_schema_oid IS NULL or old_schema_oid IS NULL THEN
        RAISE NOTICE ''schema or table is invalid'';
        RETURN false;
    ELSE
        /**Correct table location **/
        UPDATE pg_catalog.pg_class
            SET relnamespace = new_schema_oid
            WHERE relnamespace = old_schema_oid
                AND relname = param_tblname;

        UPDATE pg_catalog.pg_type
            SET typnamespace = new_schema_oid
            WHERE typnamespace = old_schema_oid
                AND typname = param_tblname;

        /**update hidden table type **/
        UPDATE pg_catalog.pg_type
            SET typnamespace = new_schema_oid
            WHERE typnamespace = old_schema_oid
                AND typname = ''_'' || param_tblname;

        /**Correct schema dependency **/
        UPDATE pg_catalog.pg_depend
            SET refobjid = new_schema_oid WHERE refobjid = old_schema_oid AND
            objid = tblname_oid AND deptype = ''n'';

        /**Correct schema of all dependent constraints that were in old schema to new schema **/
        UPDATE pg_catalog.pg_constraint
            SET connamespace = new_schema_oid
            WHERE connamespace = old_schema_oid
                AND oid IN(SELECT objid FROM pg_depend WHERE refobjid = tblname_oid);

        /**Correct schema of type object of dependent constraint index that were in old schema  **/
        UPDATE pg_catalog.pg_class
            SET relnamespace = new_schema_oid
            WHERE relnamespace = old_schema_oid
                AND oid IN(SELECT d.objid
                    FROM pg_catalog.pg_depend As p INNER JOIN pg_catalog.pg_depend As d ON p.objid = d.refobjid
                        WHERE p.refobjid = tblname_oid AND d.deptype = ''i'');

        /**Correct schema of type object of dependent index that were in old schema  **/
        UPDATE pg_catalog.pg_class
            SET relnamespace = new_schema_oid
            WHERE relnamespace = old_schema_oid
                AND oid IN(SELECT d.objid
                    FROM pg_depend As d
                        WHERE d.refobjid = tblname_oid AND d.deptype = ''a'');

        /**Correct schema of class object of related sequence object that was in old schema **/
        UPDATE pg_catalog.pg_class
            SET relnamespace = new_schema_oid
            WHERE relnamespace = old_schema_oid
                AND relname
                    IN(SELECT  param_tblname || ''_'' || c.column_name || ''_seq''
                    FROM information_schema.columns c
                    WHERE c.table_name = param_tblname AND c.table_schema = param_source_schema);

        /**Correct schema of type object of related sequence object that was in old schema  **/
        UPDATE pg_catalog.pg_type
            SET typnamespace = new_schema_oid
            WHERE typnamespace = old_schema_oid
                AND typname
                IN(SELECT  param_tblname || ''_'' || c.column_name || ''_seq''
                    FROM information_schema.columns c
                    WHERE c.table_name = param_tblname AND c.table_schema = param_source_schema);

        /**Update schema dependency of related class object of sequence object **/
        UPDATE pg_catalog.pg_depend
            SET refobjid = new_schema_oid
            WHERE refobjid = old_schema_oid AND
            objid IN(SELECT c.oid
                    FROM pg_catalog.pg_class c
                        INNER JOIN information_schema.columns co
                        ON (c.relname = param_tblname || ''_'' || co.column_name || ''_seq'')
                        WHERE c.relnamespace = new_schema_oid AND co.table_name = param_tblname)
                    AND deptype = ''n'';


        /**Correct postgis geometry columns **/
        IF EXISTS(SELECT table_name
                FROM information_schema.tables
                WHERE table_name = ''geometry_columns'' AND table_schema = ''public'') THEN
            UPDATE public.geometry_columns SET f_table_schema = param_dest_schema
            WHERE f_table_schema = param_source_schema and f_table_name = param_tblname ;
            END IF;

        RETURN TRUE;
    END IF;
END
'
  LANGUAGE 'plpgsql' VOLATILE;


--Post 8.0 installs
CREATE OR REPLACE FUNCTION cpmove_relation(param_tblname character varying,
        param_source_schema character varying,
        param_dest_schema character varying)
  RETURNS boolean AS
$$
-- param_tblname is the table name
-- param_source_schema is the source schema
-- param_dest_schema is the destination schema
DECLARE
    new_schema_oid oid;
    old_schema_oid oid;
    tblname_oid oid;
BEGIN
```

```
   new_schema_oid := (SELECT oid FROM pg_catalog.pg_namespace
            WHERE nspname = param_dest_schema);
   old_schema_oid := (SELECT oid FROM pg_catalog.pg_namespace
            WHERE nspname = param_source_schema);

   tblname_oid := (SELECT oid FROM pg_catalog.pg_class
            WHERE relname = param_tblname AND relnamespace = old_schema_oid);

   IF new_schema_oid IS NULL or old_schema_oid IS NULL or tblname_oid IS NULL THEN
      RAISE NOTICE 'schema or table is invalid';
      RETURN false;
   ELSE
      EXECUTE('ALTER TABLE ' || param_source_schema || '.' || param_tblname || ' SET SCHEMA ' ||
param_dest_schema);


      /**Correct postgis geometry columns **/
      IF EXISTS(SELECT table_name
               FROM information_schema.tables
               WHERE table_name = 'geometry_columns' AND table_schema = 'public') THEN
         UPDATE public.geometry_columns SET f_table_schema = param_dest_schema
            WHERE f_table_schema = param_source_schema and f_table_name = param_tblname ;
         END IF;

      RETURN TRUE;
   END IF;
END
$$
  LANGUAGE 'plpgsql' VOLATILE;
```

To use do

```
SELECT cpmove_relation('mytable', 'public', 'mynewschema');
```

If you want to move multiple tables at once to a new schema - you can do something like this

```
SELECT cpmove_relation(table_name, table_schema, 'financials')
FROM information_schema.tables
WHERE table_name LIKE 'payment%' AND table_schema = 'public';
```

This routine we tested with inherited tables as well as regular tables and views and it repoints the indexes, constraints, rules and repoints the inherited child tables to the moved parent table. As a side benefit it corrects the definitions of views that reference moved tables to the new names. Note that we didn't need to explicitly move these in the above or rebuild the view because PostgreSQL is using the tables OID for referencing the actual table rather than the schema qualified name. The fact that views are automagically corrected may come as a surprise to many. This is possible we believe because PostgreSQL doesn't rely on the name of the table provided in the SQL definition of the view, but internally uses the OID of the table. This is different from say using Microsoft SQL Server or MySQL where renaming a table and so forth breaks your view.

This behavior of PostgreSQL is important to keep in mind, because it means you can't trick it by renaming a table and stuffing in a new table with the same original name (internally its using the OID of a table rather than the actual referenced name of the table). Your views will be magically changed to use the renamed bad table. To test this

1. Create a view
2. Rename a table that the view relies on.
3. Open up the definition of the view and you'll see something like

   ```
   SELECT oldname.field1, oldname.field2 ...
                           FROM newname As oldname
   ```

It is both great and scary at the same time. Sometimes you wish Postgres was not such a smart aleck, but for schema table movement, it is just what the doctor ordered.

Back to Table Of Contents

How to convert a table column to another data type *Beginner*

As Robert Treat pointed out in our PostgreSQL 8.3 is out and the Project Moves On, one of the features that was introduced in PostgreSQL 8.0 was the syntax of

```
ALTER TABLE sometable
    ALTER COLUMN somecolumn TYPE new_data_type
    USING some_function_call_to_cast(somecolumn);
```

The USING syntax is particularly handy because it allows you to control how casts are done. Let us suppose you have a text or varchar field that you realize later on should have been an integer and its padded on top of that because it comes from some stupid DBF or mainframe import.

So now you have this field called - fraddl which is of type CHAR(10). You want to change it to an integer. There are two issues you run into.

1. If you do something like

   ```
   ALTER TABLE ma_tiger ALTER COLUMN fraddl TYPE integer
   ```

   You get this rather unhelpful message:
   column "fraddl" cannot be cast to type "pg_catalog.int4"

2. Even if the above did work, you've got some stuff in there you don't really care about - letters and so forth or an empty string. So you want to control how the cast is done anyway

To resolve this issue - lets suppose we write a simple function like this which takes a string value and if it looks like a number, it converts it to a number otherwise it just returns NULL:

```
CREATE OR REPLACE FUNCTION pc_chartoint(chartoconvert character varying)
  RETURNS integer AS
$BODY$
SELECT CASE WHEN trim($1) SIMILAR TO '[0-9]+'
        THEN CAST(trim($1) AS integer)
    ELSE NULL END;

$BODY$
  LANGUAGE 'sql' IMMUTABLE STRICT;
```

Now with the USING syntax, we can solve this annoying issue with this command.

```
ALTER TABLE ma_tiger ALTER COLUMN fraddl TYPE integer USING pc_chartoint(fraddl);
```

Back to Table Of Contents

DML to generate DDL and DCL- Making structural and Permission changes to multiple tables *Intermediate*

Every once in a while you are tasked with an issue such as having to create logging fields in each of your tables or having to put like constraints on each of your tables or you need to Grant an X group or X user rights to a certain set of tables.

The nice thing about having an information_schema is that it provides an easy way to generate scripts to do just that with plain SELECT statements. In PostgreSQL its even easier with the array_to_string functions and ARRAY functions, you can get the script in a single field result. In the following sections we'll demonstrate some examples of this.

Data Definition Language (DDL): How to add an Added Date timestamp to all your tables that don't have them

For example you may decide one day it would be nice to have a date time stamp on all your table records, but you don't want to add these fields to tables that already have them and you are way too lazy to go thru each and every table with PgAdmin etc. to do this. The below code will generate a script to add these fields to all tables in a schema called **hr** that don't have the add_date field already and once the script is generated, you can selectively cut out the options you don't want.

```
SELECT array_to_string(ARRAY(SELECT 'ALTER TABLE ' || t.table_schema || '.'
    || t.table_name || ' ADD COLUMN add_date timestamp DEFAULT(current_timestamp);'
    FROM information_schema.tables t
        LEFT JOIN information_schema.columns c
        ON (t.table_name = c.table_name AND
            t.table_schema = c.table_schema AND c.column_name = 'add_date')
    WHERE t.table_schema = 'hr' AND c.table_name IS NULL) , '\r') As ddlsql
```

Data Control Language (DCL): Granting Table Permissions to Groups

This technique comes in pretty handy for granting permissions to groups and users based on some somewhat arbitrary requirement. Here is an example of that. The below SELECT statement will generate a script that gives read rights to group HR for all tables across all schemas that have a field called emp_number

```
SELECT array_to_string(ARRAY(SELECT 'GRANT SELECT ON TABLE ' || c.table_schema
    || '.' || c.table_name || ' TO hr;'
    FROM information_schema.columns c
    WHERE c.column_name = 'emp_number'), '\r') As ddlsql;
```

Back to Table Of Contents

Reading PgAdmin Graphical Explain Plans *Beginner*

One of our favorite features of PgAdmin is the graphical explain plan feature. While a graphical explain plan is not a complete substitute for EXPLAIN or EXPLAIN ANALYZE text plans, it does provide a quick and easy to read view that can be used for further analysis. In this article, we'll walk thru using the explain plan to troubleshoot query performance.

To use the graphical explain plan feature in PgAdmin III - do the following

1. Launch PgAdmin III and select a database.
2. Click the SQL icon
3. Type in a query or set of queries, and highlight the text of the query you want to analyse.
4. Click the F7 button or go under Query->Explain or click the Explain Query icon .
5. If you see no graphical explain plan, **make sure that Query->Explain options->Verbose is unchecked** - otherwise graphical explain will not work
6. In terms of Explain option under the Query->Explain options-> you can choose *Analyze* which will give you the actual Explain plan in use and actual time and will take longer to run. Unchecking this feature gives you the approximate explain plan and does not include time since its approximate. In terms of the graphical display - the raw display doesn't look too different between the 2, but if you click on a section of the graph, a little tip will pop up showing the stats for that part of the graph. For analyze, you will see time metrics in the tip.

Here we look at the graphical explain plan of the two below queries querying against the demo **pagila** database. Note that with constraint_exclusion off, both these two queries have similar plans as shown in diagram below.

```
SELECT * FROM payment;
```

```
EXPLAIN ANALYZE
   SELECT * FROM payment;
```

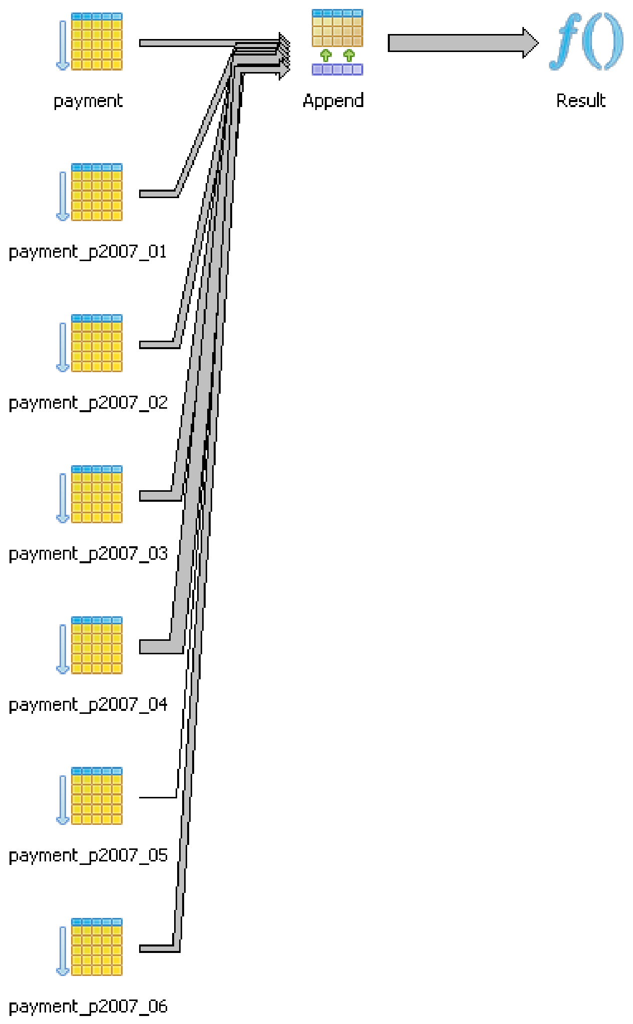
```
Result  (cost=0.00..327.09 rows=18709 width=31) (actual time=0.010..20.240 rows=16049 loops=1)
->  Append  (cost=0.00..327.09 rows=18709 width=31) (actual time=0.007..11.057 rows=16049 loops=1)
 ->  Seq Scan on payment  (cost=0.00..23.30 rows=1330 width=31) (actual time=0.001..0.001 rows=0 loops=1)
 ->  Seq Scan on payment_p2007_01 payment  (cost=0.00..20.57 rows=1157 width=28) (actual time=0.006..0.340 rows=1157 loops=1)
 ->  Seq Scan on payment_p2007_02 payment  (cost=0.00..40.12 rows=2312 width=28) (actual time=0.003..0.635 rows=2312 loops=1)
 ->  Seq Scan on payment_p2007_03 payment  (cost=0.00..98.44 rows=5644 width=28) (actual time=0.003..1.521 rows=5644 loops=1)
 ->  Seq Scan on payment_p2007_04 payment  (cost=0.00..117.54 rows=6754 width=28) (actual time=0.003..1.805 rows=6754 loops=1)
 ->  Seq Scan on payment_p2007_05 payment  (cost=0.00..3.82 rows=182 width=27) (actual time=0.003..0.051 rows=182 loops=1)
 ->  Seq Scan on payment_p2007_06 payment  (cost=0.00..23.30 rows=1330 width=31) (actual time=0.000..0.000 rows=0 loops=1)
Total runtime: 23.754 ms
```

AND

```
SET constraint_exclusion = 'off';
SELECT  * FROM payment
WHERE payment_date BETWEEN '2007-02-01' and '2007-02-15';
```

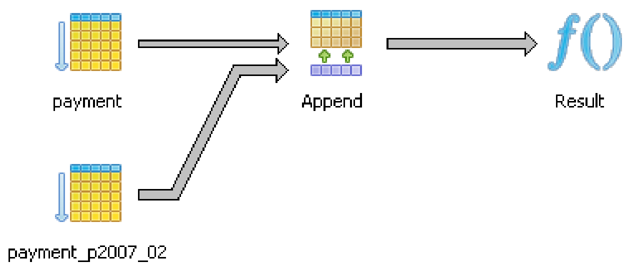
```
EXPLAIN ANALYZE
SELECT * FROM payment
WHERE payment_date
BETWEEN '2007-02-01' and '2007-02-15';
```

```
Result  (cost=0.00..420.63 rows=54 width=31) (actual time=0.262..5.193 rows=37 loops=1)
  ->  Append  (cost=0.00..420.63 rows=54 width=31) (actual time=0.261..5.173 rows=37 loops=1)
   ->  Seq Scan on payment  (cost=0.00..29.95 rows=7 width=31) (actual time=0.001..0.001 rows=0 loops=1)
         Filter: ((payment_date >= '2007-02-01 00:00:00'::timestamp without time zone)
                 AND (payment_date <= '2007-02-15 00:00:00'::timestamp without time zone))
   ->  Seq Scan on payment_p2007_01 payment  (cost=0.00..26.36 rows=1 width=28) (actual time=0.244..0.244 rows=0 loops=1)
         Filter: ((payment_date >= '2007-02-01 00:00:00'::timestamp without time zone)
                 AND (payment_date <= '2007-02-15 00:00:00'::timestamp without time zone))
   ->  Seq Scan on payment_p2007_02 payment  (cost=0.00..51.68 rows=36 width=28) (actual time=0.014..0.598 rows=37 loops=1)
          Filter: ((payment_date >= '2007-02-01 00:00:00'::timestamp without time zone)
                AND (payment_date <= '2007-02-15 00:00:00'::timestamp without time zone))
   ->  Seq Scan on payment_p2007_03 payment  (cost=0.00..126.66 rows=1 width=28) (actual time=1.461..1.461 rows=0 loops=1)
           Filter: ((payment_date >= '2007-02-01 00:00:00'::timestamp without time zone)
                       AND (payment_date <= '2007-02-15 00:00:00'::timestamp without time zone))
   ->  Seq Scan on payment_p2007_04 payment  (cost=0.00..151.31 rows=1 width=28) (actual time=2.683..2.683 rows=0 loops=1)
          Filter: ((payment_date >= '2007-02-01 00:00:00'::timestamp without time zone)
                  AND (payment_date <= '2007-02-15 00:00:00'::timestamp without time zone))
   ->  Seq Scan on payment_p2007_05 payment  (cost=0.00..4.73 rows=1 width=27) (actual time=0.052..0.052 rows=0 loops=1)
         Filter: ((payment_date >= '2007-02-01 00:00:00'::timestamp without time zone)
               AND (payment_date <= '2007-02-15 00:00:00'::timestamp without time zone))
   ->  Seq Scan on payment_p2007_06 payment  (cost=0.00..29.95 rows=7 width=31) (actual time=0.000..0.000 rows=0 loops=1)
          Filter: ((payment_date >= '2007-02-01 00:00:00'::timestamp without time zone)
               AND (payment_date <= '2007-02-15 00:00:00'::timestamp without time zone))
Total runtime: 5.287 ms
```



Now if we turn constraint_exclusion on - which, by the way you can make a global setting, in your PostgreSQL installation by modifying postgresql.conf or via PgAdmin->Tools->Server Configuration->postgresql.conf. If you make a global setting, make sure to restart your PostgreSQL service/daemon or reload the config.

```
SET constraint_exclusion = 'on';
SELECT  * FROM payment
WHERE payment_date
BETWEEN '2007-02-01' and '2007-02-15';
```



The graphical explain plan is displayed as shown below.

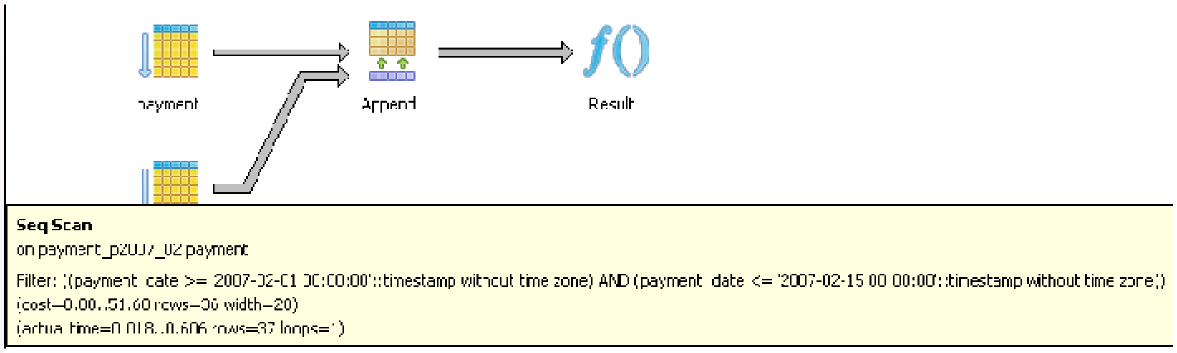
Note with constraint exclusion, the planner is analyzing the constraints in each of the tables and taking advantage of the fact that there is a constaint on each of the inherited tables of the form

```
ALTER TABLE payment_p2007_03
```

```
  ADD CONSTRAINT payment_p2007_03_payment_date_check
  CHECK (payment_date >= '2007-03-01' AND payment_date < '2007-04-01');
```

and similar constraints on the other inherited payment tables to exclude them from queries where they should never have results.

As said earlier, to look into the details of timings etc, we can click on a section of the graphical explain plan as shown here or just look at the raw explain plan text.



Seq Scan
on payment_p2007_02 payment
Filter: ((payment_date >= '2007-02-01 00:00:00'::timestamp without time zone) AND (payment_date <= '2007-02-15 00:00:00'::timestamp without time zone))
(cost=0.00..51.60 rows=36 width=20)
(actual time=0.018..0.606 rows=37 loops=1)

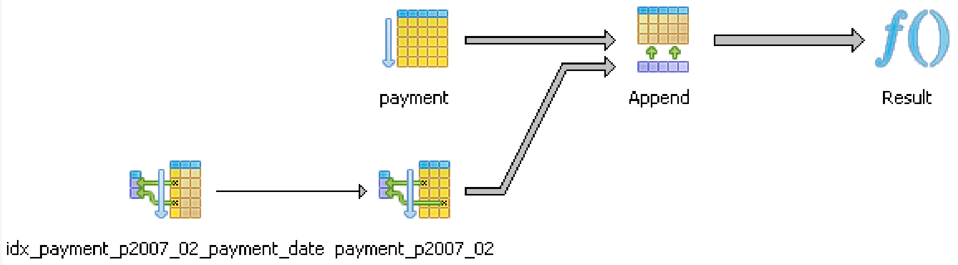
From the above we observe that with constrain exclusion on, our query has fewer tables to inspect. **Constraint Exclusion is turned off by default in PostgreSQL config, so if you use partitioned tables, make sure to enable it in your postgresql.conf file**. We also learn that our query is doing a sequential scan of the data instead of an index scan. What if we added an index on the payment_date field, like so

```
CREATE INDEX idx_payment_p2007_02_payment_date
  ON payment_p2007_02
  USING btree(payment_date);

VACUUM ANALYZE;
```

and then rerun the same query

```
SELECT  * FROM payment
WHERE payment_date
BETWEEN '2007-02-01' and '2007-02-15';
```

We see our query plan looks like this which is a bit different from before and is now using an indexed scan:

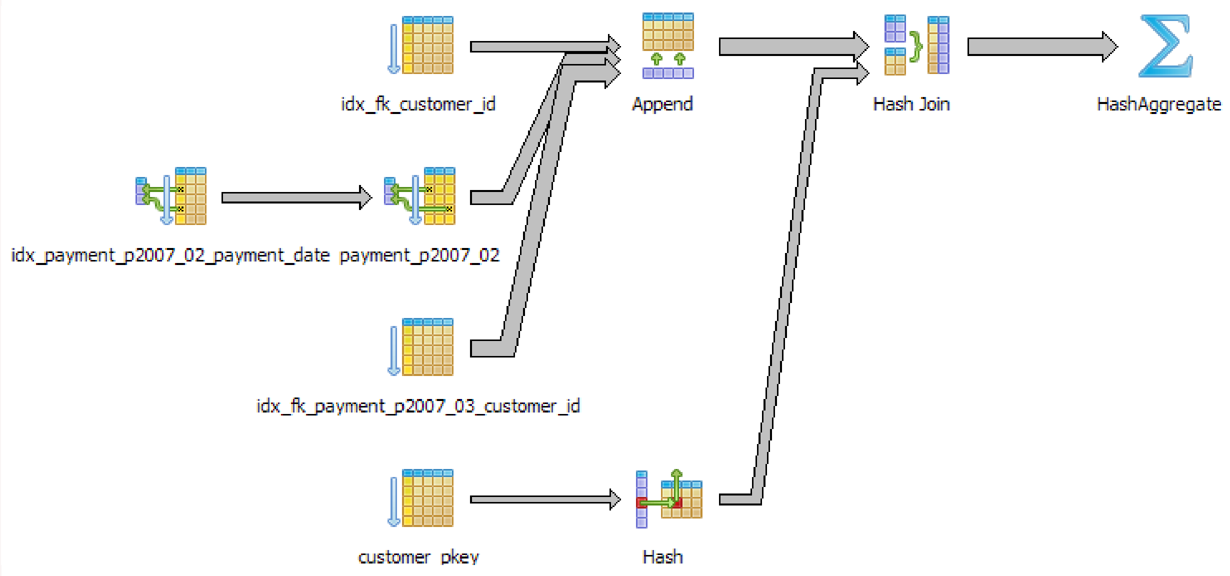


Below are some more interesting examples. Note: for our particular configuration, we can see that the planner has decided not to utilize the customer_id index.

```
 SELECT  c.first_name, c.last_name, sum(p.amount) as total
   FROM customer c INNER JOIN payment p  ON c.customer_id = p.customer_id
   WHERE payment_date BETWEEN '2007-02-01' and '2007-03-15'
   GROUP BY c.first_name, c.last_name;
```

However if we force sequential scan off - it does

```
SET enable_seqscan = 'off';
SELECT  c.first_name, c.last_name, sum(p.amount) as total
 FROM customer c INNER JOIN payment p  ON c.customer_id = p.customer_id
  WHERE payment_date BETWEEN '2007-02-01' and '2007-03-15'
 GROUP BY c.first_name, c.last_name;
```



One may ask, that if the planner is capable of using 2 indexes simultaneously with bitmap heap index scans, why did it not do so in this case without forcing its hand? If we compare the timings between the two approaches, they turn out to be pretty much the same. Was the planner wrong not to use the customer index? Probably not. The reason is that even though we have an index in place for customer_id on both tables, the fact that the customer list and number of customers purchasing items is so small, doing a sequential table scan may be more efficient than using the index. Keep in mind that the planner thinks like an economist, it sees the index as a resource and a resource that takes energy to consume. Consuming that resource may be more costly than ignoring it. This is important to keep in mind for cases where you have small tables that don't expect to increase or fields such as boolean fields. It is often wasteful to put indexes on these since they will rarely be used since a table scan is more effective and indexes have cost in terms of needing to be updated during insert/update and having the planner even have to consider them. The planner uses table statistics to determine if an index is worthwhile to use. It is important that after large inserts/updates of data, one does a

```
VACUUM ANALYZE
```

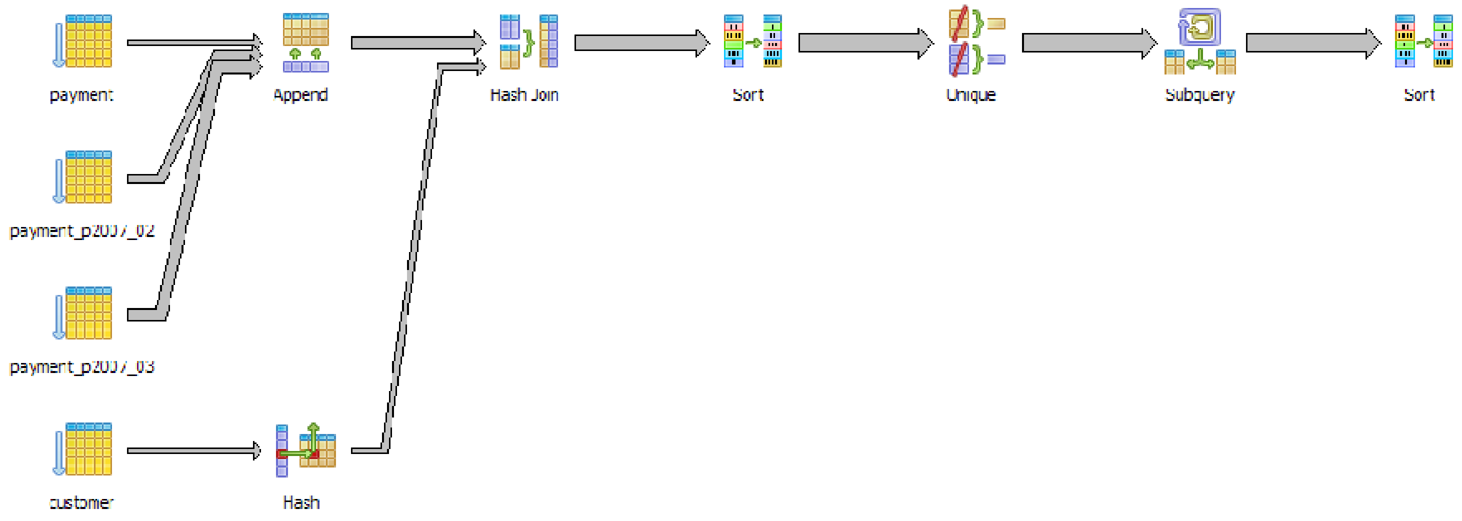
to update statistics.

In this case, over time, the number of customers will increase so while the planner determines it is not useful to use now, increase in data may change its mind.

There is another interesting thing about the graphical explain plan which is hard to see in these fairly simple examples. **There is meaning to the thickness of lines in the plan. A thicker line means more costly than a thinner line.**

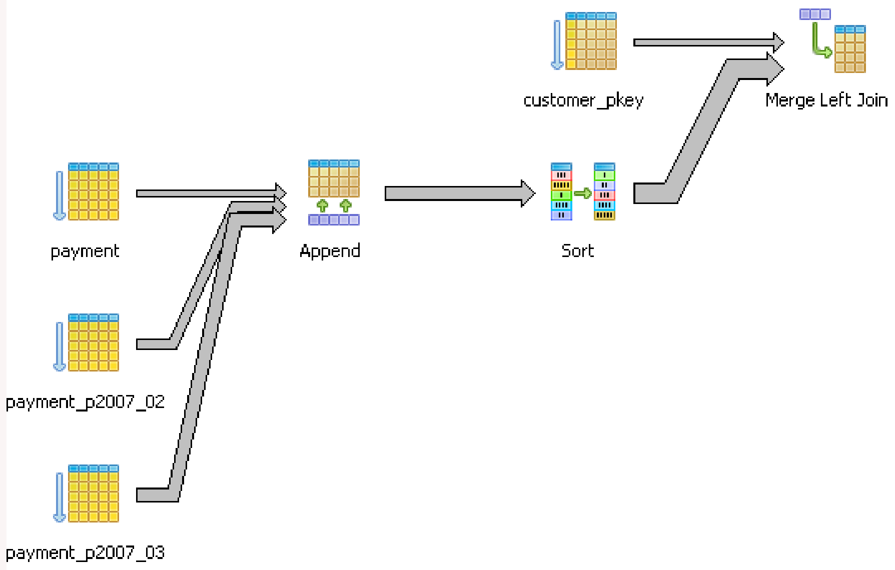
The graphical explain diagram also has a plethora of cute icons to display the various strategies in use which makes it easy to spot problematic areas - especially for fairly large plans. Below is a query that pulls the last sale for period February 2007 to March 15th 2007 for each customer who ordered during that period and sorts the customers by last name and first name.

```
SELECT cp.*
FROM (SELECT DISTINCT ON (c.customer_id)
c.first_name, c.last_name , p.amount, p.payment_date
   FROM customer c INNER JOIN payment p  ON c.customer_id = p.customer_id
   WHERE payment_date BETWEEN '2007-02-01' and '2007-03-15'
   ORDER BY c.customer_id, p.payment_date DESC) As cp
   ORDER BY cp.last_name, cp.first_name;
```



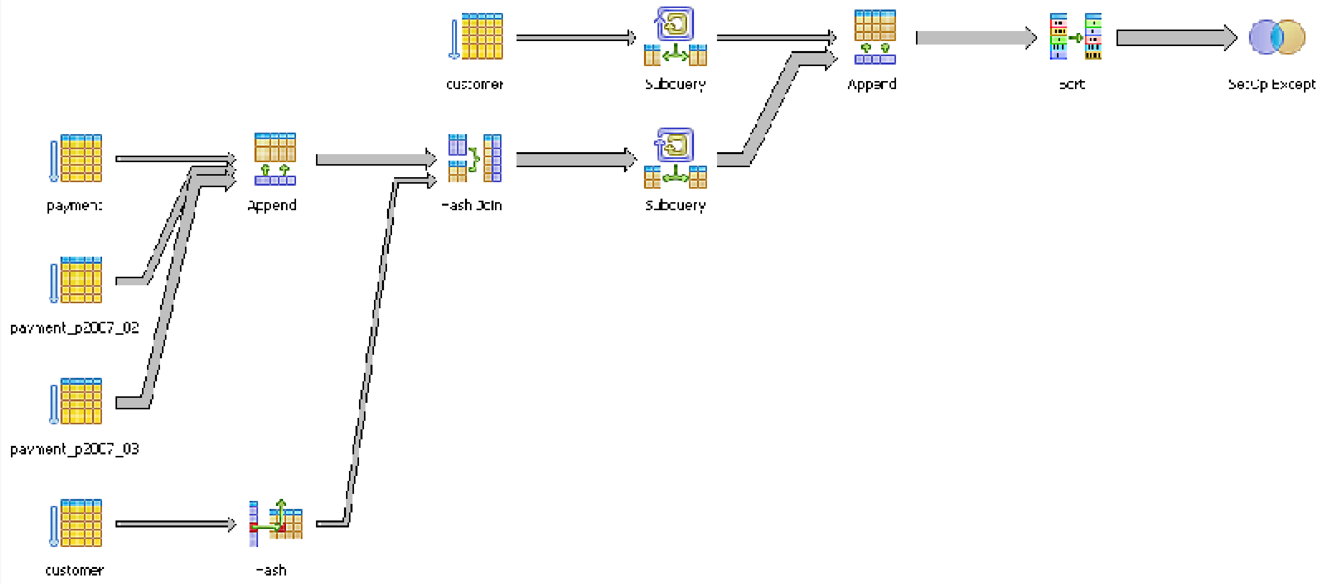
Here is another example that compares 2 different ways of selecting customers who did not purchase anything between the '2007-02-01' and '2007-03-15'.

```
SELECT c.customer_id, c.last_name, c.first_name
FROM customer c LEFT JOIN
    (SELECT customer_id
       FROM payment p
       WHERE p.payment_date BETWEEN '2007-02-01' and '2007-03-15') cp
       ON c.customer_id = cp.customer_id
WHERE cp.customer_id IS NULL;
```



vs

```
SELECT c.customer_id, c.last_name, c.first_name
FROM customer c
EXCEPT
SELECT c.customer_id, c.last_name, c.first_name
    FROM payment p
     INNER JOIN customer c ON p.customer_id = c.customer_id
     WHERE p.payment_date BETWEEN '2007-02-01' and '2007-03-15';
```



Back to Table Of Contents

New Features for PostgreSQL Stored Functions *Intermediate*

PostgreSQL 8.3 introduced a couple of new features that improves the processing of functions and makes plpgsql functions easier to write. These are as follows:

1. The new ROWS and COST declarations for a function. These can be used for any PostgreSQL function written in any language. These declarations allow the function designer to dictate to the planner how many records to expect and provide a hint as to how expensive a function call is. COST is measured in CPU cycles. A higher COST number means more costly. For example a high cost function called in an AND where condition will not be called if any of the less costly functions result in a false evaluation. The number of ROWs as well as COST will give the planner a better idea of which strategy to use.
2. RETURN QUERY functionality was introduced as well and only applies to plpgsql written functions. This is both an easier as well as a more efficient way of returning query results in plpgsql functions. Hubert Lubazeuwski provides an example of this in set returning functions in 8.3. We shall provide yet another example of this.
3. Server configuration parameters can now be set on a per-function basis. This is useful say in cases where you know a function will need a lot of work_mem, but you don't want to give all queries accessing the database that greater level of worker memory or you are doing something that index scan just works much better than sequential scan and you want to change the planners default behavior only for this function.
4. Scrollable Cursors in PL/pgSQL - this is documented in Declaring Cursor Variables
5. Plan Invalidation - Merlin Moncure covers this in PostgreSQL 8.3 Features: Plan Invalidation so we won't bother giving another example of this. Basic point to take away from this is that in procedures where you have stale plans floating dependent on tables being dropped by a function, those plans will be automagically deleted so you don't have ghost plans breaking your function.

COST

One very useful use case where allowing to specify the costliness of a function comes in handy is in **cost-based short-circuiting**. By that we mean instead of a standard orderly short-circuiting WHERE condition such as (a AND b AND c) evaluating in order and exiting when it reaches the first part that returns a false, the planner evaluates each based on order of cost. E.g. if evaluating c is cheaper than a and b, it would evaluate c and if c evaluates to false, then a and b will never be evaluated. Prior to 8.3 all internal functions were assumed to have a COST of 1 and regular stored functions COST of 100 where COST is cost per row. This of cause is often not true since for example some operations are faster in plpgsql than sql and vice versa and some other functions are more efficient done in PLPerl and you as a designer of a function know the internals of it and how resource hungry it really is e.g. not all C or plpgsql functions are made the same. The 8.3 feature allows you more granular control of how this cost-basing is done. There are two caveats that are not outlined in the 8.3 CREATE FUNCTION help docs that we feel are important to note since it left us scratching our heads for a bit. We thank Tom Lane for pointing these out to us.

1. The new COST functionality provides additional cost-based short-circuiting only in AND conditions, not OR conditions. Presumably PostgreSQL really only applies cost-based short-circuiting in AND clauses except for the case of constants
2. Again cost-based short-circuiting applies only in WHERE clauses, not in the SELECT part.

To demonstrate the above - we provide here a fairly trivial example that makes clear the above points.

```
--Setup code
CREATE TABLE log_call
(
  fn_name character varying(100) NOT NULL,
  fn_calltime timestamp with time zone NOT NULL DEFAULT now()
)
WITH (OIDS=FALSE);

CREATE OR REPLACE FUNCTION fn_pg_costlyfunction()
  RETURNS integer AS
$$
BEGIN
 INSERT INTO log_call(fn_name) VALUES('fn_pg_costlyfunction()');
 RETURN 5;
END$$
  LANGUAGE 'plpgsql' VOLATILE
  COST 1000000;

CREATE OR REPLACE FUNCTION fn_pg_cheapfunction()
  RETURNS integer AS
$$
BEGIN
 INSERT INTO log_call(fn_name) VALUES('fn_pg_cheapfunction()');
 RETURN 5;
END$$
  LANGUAGE 'plpgsql' VOLATILE
  COST 1;

--- Now for the tests -
--No cost-based short-circuiting - planner evaluates in sequence the more
--costly fn_pg_costlyfunction() and stops
TRUNCATE TABLE  log_call;
SELECT (fn_pg_costlyfunction() > 2 OR fn_pg_cheapfunction() > 2 OR 5 > 2);
```

```
--Pseudo Cost-based short-circuiting
-- planner realizes 5 > 2 is a constant and processes that
-- No cheap or costly functions are run.
-- No functions are forced to work in this test.
TRUNCATE TABLE log_call;
SELECT true
WHERE fn_pg_costlyfunction() > 2 OR fn_pg_cheapfunction() > 2 OR 5 > 2;

--Again planner goes for low hanging fruit - processes 2 > 5 first
-- no point in processing the functions.
--No functions are forced to work in this test.
TRUNCATE TABLE log_call;
SELECT true
WHERE fn_pg_costlyfunction() > 2 AND fn_pg_cheapfunction() > 2 AND 2 > 5;


--No Cost-based short-circuiting  planner processes in order
--and stops at first true
-- only fn_pg_costlyfunction() is run
TRUNCATE TABLE log_call;
SELECT true
WHERE fn_pg_costlyfunction() > 2 OR fn_pg_cheapfunction() > 2 ;

--Cost-based short-circuiting
-- planner realizes fn_pg_costlyfunction() is expensive
-- fn_pg_cheapfunction() is the only function run even though it is second in order
TRUNCATE TABLE log_call;
SELECT true as value
WHERE (fn_pg_costlyfunction() > 2 AND fn_pg_cheapfunction() > 5 );
```

ROWS

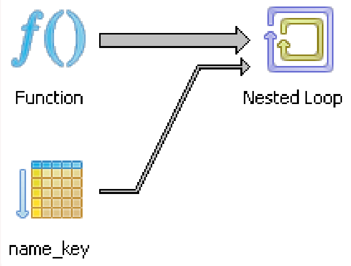
Another parameter one can specify in defining a function is the number of expected ROWS. This is covered a little in Hans-Jürgen SCHÖNIG's Optimizing function calls in PostgreSQL 8.3. We feel more examples are better than fewer, so we will provide yet another example of how this can affect plan decisions.

```
--Create dummy people with dummy names
CREATE TABLE people
(
  first_name character varying(50),
  last_name character varying(50),
  mi character(1),
  name_key serial NOT NULL,
  CONSTRAINT name_key PRIMARY KEY (name_key)
)
WITH (OIDS=FALSE);


INSERT INTO people(first_name, last_name, mi)
SELECT a1.p1 || a2.p2 As fname, a3.p3 || a1.p1 || a2.p2 As lname, a3.p3 As mi
FROM
    (SELECT chr(65 + mod(CAST(random()*1000 As int) + 1,26)) as p1
           FROM generate_series(1,30)) as a1
    CROSS JOIN
        (SELECT chr(65 + mod(CAST(random()*1000 As int) + 1,26)) as p2
           FROM generate_series(1,20)) as a2
    CROSS JOIN
        (SELECT chr(65 + mod(CAST(random()*1000 As int) + 1,26)) as p3
           FROM generate_series(1,100)) as a3;

CREATE INDEX idx_people_last_name
  ON people
  USING btree
  (last_name)
  WITH (FILLFACTOR=98);
ALTER TABLE people CLUSTER ON idx_people_last_name;



-- The tests
CREATE OR REPLACE FUNCTION fn_get_peoplebylname_key(lname varchar)
  RETURNS SETOF int AS
$$
SELECT name_key FROM people WHERE last_name LIKE $1;
$$
  LANGUAGE 'sql' ROWS 5 STABLE;

--The Test
VACUUM ANALYZE;
```

```
SELECT p.first_name, p.last_name, nkey
 FROM fn_get_peoplebylname_key('M%') as nkey
    INNER JOIN people p ON p.name_key = nkey
WHERE p.first_name <> 'E';
```

```
Nested Loop  (cost=0.00..42.75 rows=5 width=11)
(actual time=10.171..22.140 rows=2560 loops=1)
  ->  Function Scan on fn_get_peoplebylname_key nkey  (cost=0.00..1.30 rows=5 width=4)
  (actual time=10.153..10.841 rows=2560 loops=1)
  ->  Index Scan using name_key on people p  (cost=0.00..8.28 rows=1 width=11)
  (actual time=0.002..0.003 rows=1 loops=2560)
        Index Cond: (p.name_key = nkey.nkey)
        Filter: ((p.first_name)::text <> 'E'::text)
Total runtime: 22.806 ms
```

The pgAdmin graphical explain plan shows nicely that a Nested loop strategy is taken.

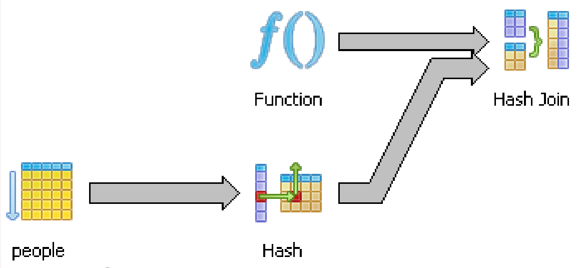


Now we try the same test again after setting ROWS to 3000.

```
CREATE OR REPLACE FUNCTION fn_get_peoplebylname_key(lname varchar)
  RETURNS SETOF int AS
$$
SELECT name_key FROM people WHERE last_name LIKE $1;
$$
  LANGUAGE 'sql' ROWS 3000 STABLE;
```

```
Hash Join  (cost=1822.49..2666.14 rows=2990 width=11)
(actual time=66.680..70.367 rows=2560 loops=1)
  Hash Cond: (nkey.nkey = p.name_key)
  ->  Function Scan on fn_get_peoplebylname_key nkey  (cost=0.00..780.00 rows=3000 width=4)
  (actual time=10.308..10.942 rows=2560 loops=1)
  ->  Hash  (cost=1075.00..1075.00 rows=59799 width=11)
  (actual time=56.317..56.317 rows=60000 loops=1)
        ->  Seq Scan on people p  (cost=0.00..1075.00 rows=59799 width=11)
                (actual time=0.007..27.672 rows=60000 loops=1)
              Filter: ((first_name)::text <> 'E'::text)
Total runtime: 71.229 ms
```

By specifying the rows to a higher value, the planner changes strategies to a Hash Join from a nested loop. Why the hash join performs so much worse than the nested loop and it totally rejects using the index key in this case (even though the rows match closer to reality) is a little odd. Seems to suggest sometimes its best to lie to the planner, which is somewhat counter-intuitive.



RETURN QUERY

One of the advantages of writing functions in plpgsql over writing it in say sql is that you can run dynamic sql statements in plpgsql and you can use named values. The downside was that you had to use the convoluted RETURN NEXT syntax which is both awkward and less efficient than the new RETURN QUERY. Below are two examples of using RETURN QUERY.

```
CREATE OR REPLACE FUNCTION fnpgsql_get_peoplebylname_key(lname varchar)
  RETURNS SETOF int AS
$$
BEGIN
   RETURN QUERY SELECT name_key
```

```
        FROM people WHERE last_name LIKE lname;
END
$$
  LANGUAGE 'plpgsql' STABLE;

CREATE OR REPLACE FUNCTION fnpgsql_get_peoplebylname(lname varchar, only_count boolean)
  RETURNS SETOF int AS
$$
BEGIN
    IF only_count = true THEN
        RETURN QUERY SELECT COUNT(name_key)::int
            FROM people WHERE last_name LIKE lname;
    ELSE
        RETURN QUERY SELECT name_key
            FROM people WHERE last_name LIKE lname;
    END IF;
END;
$$
  LANGUAGE 'plpgsql' STABLE;

--To use --
SELECT *
FROM fnpgsql_get_peoplebylname('E%', true);

SELECT *
FROM fnpgsql_get_peoplebylname('E%', false);
```

Server Configuation Parameters per function

There are too many server configuration parameters one can use in functions to enumerate them. To demonstrate how these settings can be done, we revised our prior query and modified it mindlessly. NOTE don't construe any meaning to this function or the settings we chose. It is all in the name of "What goofy exercises can we concoct to demonstrate postgres features".

```
CREATE OR REPLACE FUNCTION fnpgsql_get_loop(lname varchar, numtimes integer)
  RETURNS SETOF int AS
$$
DECLARE i integer := 0;
BEGIN
    WHILE i < numtimes LOOP
        RETURN QUERY SELECT p.name_key
            FROM people p INNER JOIN people a ON p.name_key = (a.name_key + 1)
        WHERE (p.last_name LIKE lname OR a.last_name LIKE lname);
        i := i + 1;
    END LOOP;
END;
$$
  LANGUAGE 'plpgsql' STABLE
  SET work_mem = 64
  SET enable_hashjoin = false
  SET enable_indexscan = true;


SELECT p.first_name, p.last_name, nkey
 FROM fnpgsql_get_loop('M%',100) as nkey
    INNER JOIN people p ON p.name_key = nkey
WHERE p.first_name <> 'E';
```

Back to Table Of Contents

TSearch Primer *Beginner*

What is TSearch?

TSearch is a Full-Text Search engine that is packaged with PostgreSQL. The key developers of TSearch are Oleg Bartunov and Teodor Sigaev who have also done extensive work with GiST and GIN indexes used by PostGIS, PgSphere and other projects. For more about how TSearch and OpenFTS got started check out A Brief History of FTS in PostgreSQL. Check out the TSearch Official Site if you are interested in related TSearch tips or interested in donating to this very worthy project.

Tsearch is different from regular string searching in PostgreSQL in a couple of key ways.

1. It is well-suited for searching large blobs of text since each word is indexed using a Generalized Inverted Index (GIN) or Generalized Search Tree (GiST) and searched using text search vectors. GIN is generally used for indexing. Search vectors are at word and phrase boundaries.
2. TSearch has a concept of Linguistic significance using various language dictionaries, ISpell, thesaurus, stop words, etc. therefore it can ignore common words and equate like meaning terms and phrases.
3. TSearch is for the most part case insensitive.
4. While various dictionaries and configs are available out of the box with TSearch, one can create new ones and customize existing further to cater to specific niches within industries - e.g. medicine, pharmaceuticals, physics, chemistry, biology, legal matters.

Prior to PostgreSQL 8.3, it was a contrib module located in the shared/contribs folder. As of PostgreSQL 8.3 it is now fully integrated into the PostgreSQL core. The official documents for using TSearch in 8.3 are located in Chapter 12. Full Text Search of the official PostgreSQL documentation.

In this article we shall provide a quick primer to using TSearch in 8.3. In the next month's issue of the Postgres OnLine Journal we shall provide a TSearch cheat sheet similar to our PostgreSQL 8.3 cheat sheet.

Getting Started with Using TSearch

While you can still use TSearch without creating indices, for large fields or huge tables it is highly adviced you create indices before performing TSearch queries. Below are the general steps to take to use TSearch.

Create the Index

In terms of creating indexes you have the choice of GIN or GIST indexes. Pros and cons are itemized in the Chapter 12. Full Text Search: 12.9. GiST and GIN Index Types. In a nutshell - GIST indexes are better when doing weighted queries while GIN indexes are better for standard word queries and larger texts that don't require weighting. GIST indexes are also lossy and produce more false positives thus requiring rechecking of raw data, but faster to build than GIN indexes.

Sample indexes are shown below:

```
--Single field index
CREATE INDEX idx_sometable_somefield
    ON sometable
    USING gin(to_tsvector('english', somefield));
--Multi-field index
CREATE INDEX idx_sometable_ts
    ON sometable
    USING gin(to_tsvector('english', COALESCE(somefield1,'') || ' ' || COALESCE(somefield2,'')));

CREATE INDEX idx_sometable_ts_gist
    ON sometable
    USING gist(to_tsvector('english', COALESCE(somefield1,'') || ' ' || COALESCE(somefield2,'')));
--Index on field of type tsvector
CREATE INDEX idx_sometable_ts
    ON sometable
    USING gin(tsvector_field);
```

In the above examples we are indexing based on the English language. There are 16 options pre-packaged with PostgreSQL and TSearch is flexible enough that you can define your own custom ones - say catered for certain niche scientific or medical research disciplines. The choice of languages is listed in **pg_catalog.ts_config** and the default option selected if none is specified is based on your locale.

As demonstrated above, you can create an index on a tsvector type column, but that would require creating yet another column and a trigger to update it when data changes. This does provide efficiency of not having to recalculate a tsvector or specify it in your query each time you run a ts query. To not have to respecify it, you can also use a view. Much of this is well documented in 12.2. Full Text Search: Table Text Search. A good example of using Triggers to update fulltext fields and storing fulltext fields is provided in the Pagila database (check the film table) as well as the Full Text Search: 12.4.3. Triggers for Automatic Updates chapter of official docs.

Using TSearch - The Query

There are 2 concepts in TSearch that are most important - The TSearch query condition which evaluates to either **true** or **false** and the ranking which is a measure of how good the match is.

A TSearch condition uses the match operators (@@ or @@@ -- @@ is used for general searches and weighted GIST searches, and @@@ is slower but must be used for weighted GIN searches). @@ is the more common and always of the form (*some_ts_vector* @@ *some_ts_query*). The ts vector is simply what you have indexed as shown above and the secret in the sauce is the *some_ts_query*. The key operators in a TS Query are & (AND), | (OR) and ! (NOT)

Below are some examples of using TS Query:

```
--Snippet two  - examples using TQuery
--We want to check if the provided phrase contains the words dog and sick.
--This returns true
SELECT to_tsvector('english', 'My dog is sick')
   @@ to_tsquery('english', 'dog & SICK');

--This one is false because doggy is not a word boundary for dog
SELECT to_tsvector('english', 'My doggy is sick')
   @@ to_tsquery('english', 'dog & SICK');

--However dogs and dog are lexically equivalent so this is true
SELECT to_tsvector('english', 'I want a dog')
   @@@ to_tsquery('english', 'want & dogs');

--This one is also true
--because ski and skiing
--are derived from same word (lexeme)
SELECT to_tsvector('english', 'I like to ski')
   @@ to_tsquery('english', 'like & skiing');

--This uses the default locale
SELECT to_tsvector('My dog is sick')
   @@ to_tsquery('dog & SICK');

--This searches for all views that have SUM and order in them
SELECT *
FROM information_schema.views
WHERE to_tsvector(view_definition)  @@ to_tsquery('sum & order');

--Search all views that have SUM or FILM
SELECT *
FROM information_schema.views
WHERE to_tsvector(view_definition)  @@ to_tsquery('sum | film');

--Search all view definitions that have sum but are not about films
SELECT *
FROM information_schema.views
WHERE to_tsvector(view_definition)  @@ to_tsquery('sum & !film');

--Search all view definitions with sum and store but not about film
SELECT *
FROM information_schema.views
WHERE to_tsvector(view_definition)  @@ to_tsquery('(sum & store) & !film ');
```

Ranking Results

TSearch provides 2 ranking functions - **ts_rank** and **ts_rank_cd**. The **CD** in ts_rand_cd stands for *Cover Density*. The ranking functions rank the relevance of a ts vector of a document to a ts query based on proximity of words, length of document, and weighting of terms. The higher the rank, the more relevant the document. The ts_rand_cd ranking function penalizes results where the search terms are further apart. The ts_rand does not penalize based on distance. One can further control weight based on position or section in a record using **setweight**. Some examples are shown below:

```
--Weight positions are demarcated by the letters A, B, C, D
--Create a fulltext field where the title is
--marked as weight position A and description is weight position B
ALTER TABLE film
   ADD COLUMN ftext_weighted tsvector;
UPDATE film
   SET ftext_weighted = (setweight(to_tsvector(title), 'A')
       || setweight(to_tsvector(description), 'B'));
CREATE INDEX idx_books_ftext_weighted
   ON film
   USING gin(ftext_weighted);

--List top 3 films about Mysql that are epic, documentary or chocolate
--NOTE: the {0,0,0.10,0.90} corresponds
--to weight positions D,C, B, A and the sum of the array should be 1
-- which means
--weight the title higher than the summary
--NOTE: we are doing a subselect here because if we don't the expensive
--highlight function gets called all the results that match the WHERE, not just the highest 3
SELECT title, description, therank,
ts_headline(title || ' ' || description, q,
   'StartSel = <em>, StopSel = </em>, HighlightAll=TRUE') as htmlmarked_summary
FROM (SELECT title, description,
    ts_rank('{0,0,0.10,0.90}', ftext_weighted, q) as therank, q
FROM film, to_tsquery('(epic | documentary | chocolate) & mysql') as q
WHERE ftext_weighted @@ q
ORDER BY therank DESC
LIMIT 3) As results;

--List top 3 films with (chocolate, secretary , or mad) and (mysql or boring) in the title or description
--Note the {0,0,0.90,0.10} corresponds to weight positions
--D,C, B, A which means based on how we weighted our index
--weight the title higher than the summary.
--This time we are using ts_rank_cd which will penalize
--query words that are further apart
--For highlighting this uses the default ts_headline which is to make terms bold
SELECT title, description,  therank,
    ts_headline(title || ' ' || description, q) as htmlmarked_summary
FROM (SELECT title, description,
    ts_rank_cd('{0,0,0.9,0.10}', ftext_weighted, q) as therank, q
    FROM film,
        to_tsquery('(chocolate | secretary | mad) & (mysql | boring)') as q
    WHERE ftext_weighted @@ q
    ORDER BY therank DESC
    LIMIT 3) As results;

--This is same as previous except in our filtering
--we only want to count secretary and mad (:A)
-- if it appears in the title of the document
--NOTE: Since we are using a GIN index, we need to use the slower @@@
SELECT title, description,  therank, ts_headline(title || ' ' || description, q) as htmlmarked_summary
FROM (SELECT title, description,
    ts_rank_cd('{0,0,0.9,0.10}', ftext_weighted, q) as therank, q
    FROM film,
        to_tsquery('(chocolate | secretary:A | mad:A) & (mysql | boring)') as q
    WHERE ftext_weighted @@@ q
    ORDER BY therank DESC
    LIMIT 3) As results;
```

Additional Features

TSearch also provides some added functions such as:

- highlighting locations of matches with html markup, using the **ts_headline** function as we have demonstrated above.
- Gathering statistics about your documents (e.g which words are filler) with **ts_stat** to better cater the searching to your specific document set.
- Debugging functions for showing debugging information

We'll provide more examples in the upcoming cheat sheet.

Support of FullText in Other Databases

As a side note: MySQL and the major commercial offerings have Full Text Search capabilities as well, but all have different full text search query syntax so there is no real standard one can rely on as far as portability goes. MySQL has integrated FullText search and has for some time, but is limited to only MyISAM tables. MySQL 5.1 seems to have some enhanced features in Full Text Search over prior versions that make it more configurable than prior versions and easier to integrate plugins. SQL Server also has FullText search, but from experience has been always somewhat awkward to use, and it relied on an additional service. The upcoming SQL Server 2008 Full Text search is supposed to be more integrated. Not sure about the other popular DBMSs Oracle, IBM DB2 etc.

We are also not sure about the speed comparisons between the various offerings. Osku Salerma's Masters thesis Design of a Full Text Search index for a database management system written in 2006, provides a cursory comparison of how Full Text Search is implemented in Oracle, MySQL, and PostgreSQL as well as a description of full text terminology and indexing strategies and other fulltext non-database search engines such as Lucerne. I think both MySQL and PostgreSQL have changed a great deal in terms of their Full Text Search speed and offerings so applying the comparison to current version of each is probably a bit unfair.

MySQL has an interesting feature called Full Text Query Expansion which allows results of a first level query to be used to find other related results. I'm not sure how good this is or if it produces garbage results and not sure if its natural language mode has improved since the above article was written. GIST and GIN indexes have definitely improved so TSearch is probably faster than it was 2 years ago.

Other related Resources

- Ian Barwick Converting tsearch2 to 8.3
- Robert Treat's PostgreSQL full text search testing PART II - has links to other part.
- Esoteric Curio's related blog entry to Robert Treat's Benchmarking Lucene

Back to Table Of Contents

Showcasing REST in PostgreSQL - The PreQuel *Intermediate*

What is REST?

Representationl State Transfer (REST) is a term to describe an architectural style of sharing information with consumers using already existing protocols such as HTTP. In the strictest sense of the term, the transport protocol need not be HTTP.

REST was first coined by Roy Fielding in his year 2000 doctoral thesis. Unlike things like Simple Object Access Protocol (SOAP), Distributed Component Object Model (DCOM), Common Object Request Broker Architecture (CORBA), it is not an architecture nor a protocol but rather a style of architecture. As a result, a lot of things that don't call themselves **RESTFUL** full under that umbrella or use some of the same concepts, or can arguably fall under that umbrella.

What is probably so alluring about REST is that it gives a catchy name to something a lot of people were doing already and describes how much of the web operates. Contrary to some popular belief, it is not NEW technology but rather a grouping of mainstream technology with a flashier name. Part of this confusion is that the cult following of REST is a fairly new phenomenon although the underpinnings are relatively old. The REST movement reflects a return back to the basics that prioritizes simplicity and accessibility over complexity and formality. Nowadays REST is most often used to refer to web services where resources are requested via plain URIs and GET requests, representations are returned in simple XML or JSON format and resources are created using POST, updated using PUT and deleted using DELETE HTTP verbs. This is similar to XML-RPC except that XML-RPC has a concept of state and everything is generally encoded in an XML message envelop. XML-RPC also uses POST for both updating and accessing resources unlike REST which tends to use GETS and URIS for resource access. The advantage of using get is that resources can be bookmarked. SOAP is similar to XML-RPC and in fact was born from the XML-RPC standard except the XML message streams are more complicated and formal, but arguably richer in functionality.

Contrary to some popular belief, REST is not a silver bullet nor was it designed to be. It doesn't work for all problems and web applications. Our personal opinion: REST is well suited for transporting data that will be consumed by various kinds of clients, but is not well suited for updating of data or where authenticated transactions are needed.

REST has 4 basic features that differentiate/and it shares with similar Architectural styles.

1. **Client Stateless Server** - As the name suggests - the state of an object is part of the message, and is commonly referred to as a stateless communication. It is not done with things like Session cookies where the server maintains some stateful view of the client and the client passes a session cookie saying (here is my ticket - give me my state). The server does not hold information about state, only the client. Right away one can tell - this can not work for all modes of communication that require immense amounts of state information to be maintained, but does have the advantage of should the application server hiccup or connection to the server times out or breaks only the current message is lost. It also works well for Web-Farms that are simply outputing data since the need for such Web-Farms to replicate state is not needed (think image caching networks such as Akamai).
2. **Client-Cache** - The idea of client caching. The server can dictate certain requests as being cacheable and if cacheable a client can use the cache request to satisfy future similar requests instead of going back to the server. This saves on band-width but has disadvantage of possibly resulting in stale results. Keep in mind again this concept is not new and most webservers are designed to work that way and pass this info via http headers.
3. **Layered System** - two way interaction. In a REST style architecture, there is a client and a server. The client is only dependent on the server it communicates with. It has no knowledge of the components the server uses to fulfill its request. That server can be a client in another REST interaction and keep its own cache to serve up like requests. This does not break the *client keeps the cache* rule as the server is acting as a client in this context. Think **DNS**. DNS is a perfect example of such a style where intermediaries cache requests for a certain period of time and act as clients to DNS servers further up the root and behave as servers to DNS and client computers below.
4. **Resource and Resource Identifiers** - REST is predominantly a mechanism for accessing resources although it can be used for editing as well. The key element of it is a mechanism for defining resources, how a resource or grouping of resources are requested via a Resource Identifier (URL or URN), transfer of representation via (HTML, XML, Jpeg etc.), representational metadata (e.g. media type, last modified), control data (such as how long it can be cached). Yes this is pretty much a common concept in web interfaces.

What we will cover

This series will flow into our next issue of the journal since it will be long.

Our exploration into REST will have three components.

In Part 1 article we will demonstrate setting up the database and creating stored functions to support the service to highlight new features in PostgreSQL 8.3.

1. New XML features
2. Integrated TSearch
3. Enums - maybe

In Part 2 we'll build the REST Web Services. We hope to demonstrate 4 implementations of our back-end service

1. ASP.NET - Compatible with both Microsoft ASP.NET and Mono.NET - C#
2. ASP.NET with MonoBasic/VB.NET
3. PHP 5
4. Java Servlet

Keep in mind this is not a pure REST architecture as the connection to the PostgreSQL Server is not RESTFUL. In practice its very rare to have an architecture where every interaction is RESTFUL.

We shall be using the Pagila demo database as the backend database.

In Part 3 we shall build a simple client to consume our REST Service. We shall demonstrate using Adobe Flex to consume this REST service. If there is enough interest, we'll consider demonstrating other clients such as Silverlight, MoonLight, PHP, ASP.NET, Java clients in a later issue.

The User Story

The Pagila database, for those unfamiliar with it, is a BSD licensed sample database, and one of several demo database for PostgreSQL to show off PostgreSQL features. Other demo databases exist and can be found at PostgreSQL Sample Databases http://pgfoundry.org/frs/?group_id=1000150&release_id=998. The pagila database was ported from the MySQL **Sakila** database.

The Pagila database features a fictious Film Rental company who rents out films with very amusing titles. These films are indeed hard to find. The database keeps track of things such as what films are available in its inventory, film categories, actors, list of customers, orders placed, items ordered, payment, store locations. Basically everything that a respectable film rental business should keep track of.

Related Articles

- Roy Fielding's UC Irvine Doctoral Dissertation - http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- The SOAP/XML-RPC/REST Saga, Chap. 51 - http://www.tbray.org/ongoing/When/200x/2003/05/12/SoapAgain
- REST Ajax Patterns - http://ajaxpatterns.org/RESTful_Service

Back to Table Of Contents

REST in PostgreSQL Part 1 - The DB components *Intermediate*

In this section we'll go over creating the views and stored functions that our REST Server service will rely on.

Our REST service will be a thin wrapper around a pgsql function that accepts film queries and returns results in XML.

Loading the database

Step 1: Download the Pagila 0.10.0 database from http://pgfoundry.org/frs/?group_id=1000150&release_id=570 and load the Pagila database using the following commands:

Note: For windows users - psql is usually located at *"C:\Program Files\PostgreSQL\8.3\bin\psql"*

```
psql -h localhost -p 5433 -U postgres -c "CREATE DATABASE pagila ENCODING 'UTF8'"
psql -h localhost -p 5433 -U postgres -c "CREATE ROLE pagila_app LOGIN PASSWORD 'pg@123'"
psql -h localhost -p 5433 -U postgres -d pagila -f "pagila-schema.sql"
psql -h localhost -p 5433 -U postgres -d pagila -f "pagila-data.sql"
```

Writing our view

We are creating a view to make querying the data easier, but we don't need to give our pagila_app account rights to the view since all access will be thru our stored function. The view is mostly there to provide a namespace for the XML output.

```
CREATE OR REPLACE VIEW vwfilms AS
 SELECT film.film_id AS fid, film.title, film.description, category.name AS category,
    film.rental_rate AS price, film.length, film.rating, film.fulltext
   FROM category
   LEFT JOIN film_category
    ON category.category_id = film_category.category_id
   LEFT JOIN film
    ON film_category.film_id = film.film_id
```

Writing our search stored function

Now we shall write a stored function that takes a search criteria and returns an XML result. Note that we are designating the function is accessed with **SECURITY DEFINER** which means the executer of the function has all rights to items used within the function as long as the creator of the function has rights. Keep in mind we didn't really think long and hard about how to do this efficiently so our implementation of converting to Tsearch format is probably highly inefficient and is a brain dead implementation of converting a simple search phrase into suitable Tsearch format. Feel free to comment as to a better way of doing this or bash us for our stupid implementation. We'll try hard not to be offended.

This is shown below:

```
CREATE OR REPLACE FUNCTION fnget_film_search_results(arg_search character varying,
    arg_num_results integer, arg_start_at integer)
  RETURNS xml AS
$BODY$
       --#GIVEN A users search criteria - convert to TSearch Suitable format
       --#Return results as XML to be consumed by REST client;
DECLARE
    var_sql text;
    var_count integer;
    var_count_xml xml;
    var_tsearch_query text;
BEGIN
    var_tsearch_query := REPLACE(REPLACE(REPLACE(array_to_string(ARRAY(SELECT lower(foo)
        FROM
            regexp_split_to_table(arg_search, E'\\s+') As foo
            WHERE foo SIMILAR TO E'\\w+'), '&'),  '|and', '&'), '|or', '|') , 'not&', '!');
    RAISE NOTICE '"%"', var_tsearch_query;
    SELECT count(fid) INTO var_count FROM vwfilms WHERE fulltext @@ to_tsquery(var_tsearch_query);
    var_count_xml := '<?xml version="1.1"?><resultsummary><count>' || CAST(var_count As varchar(20))
          || '</count></resultsummary>';
    var_sql := 'SELECT fid, title, description, category, price, length, rating
        FROM vwfilms WHERE fulltext @@ to_tsquery(' || quote_literal(var_tsearch_query) || ')';
    var_sql := var_sql || ' LIMIT ' || CAST(arg_num_results As varchar(20))
       || ' OFFSET ' || CAST(arg_start_at as varchar(20));
    RETURN '<results>' || xmlconcat(var_count_xml, query_to_xml(var_sql, false, false, 'vwfilms'))
        || '</results>';
END
$BODY$
  LANGUAGE 'plpgsql' VOLATILE SECURITY DEFINER
  COST 100;
GRANT EXECUTE ON FUNCTION fnget_film_search_results(character varying, integer, integer) TO pagila_app;
```

To test out our function we do this:

```
SELECT fnget_film_search_results('not epic but has dogs',10,1);
SELECT fnget_film_search_results('not epic but has dogs',10,20);
```

In the next issue we will cover consuming this via a .NET REST Service. We will show both C# and VB.NET implementations which are compatible with Microsoft.NET ASP.NET 2.0 as well as Mono.NET 1.2.6 (C# and Mono Basic).

Back to Table Of Contents

GDAL OGR2OGR for Data Loading ***Beginner***

What is FWTools and OGR GDAL?

FWTools GIS Toolkit is a freely available open source toolkit for Windows and Linux that can do more than GIS tricks. It is a precompiled bundle of Open Source GIS tools. The FW comes from the initials of Frank Warmerdam, the originator of the toolkit and current President of the Open Source Geospatial Foundation (OSGEO).

One key component of the GIS Toolkit is the GDAL/OGR library. Parts of the library have been enhanced by several in the OSGEO community. GDAL is a basic foundation of countless Open source GIS as well as commercial GIS applications. Here are Listings of commercial and open source software that use it and GDAL sponsors.

This is a library which historically has been developed and maintained by Frank Warmerdam, but has started to garner quite a few developers. GDAL is X/MIT licensed (similar to BSD license), therefore the licensing is very generous for commercial use. The toolkit can be downloaded from http://fwtools.maptools.org/

In this article we shall talk a little bit about the OGR2OGR commandline data loader tool packaged in here, which should be of interest to many even those who do not know or care about GIS data. We shall demonstrate some non-GIS uses of this light-weight tool.

If you want to see some GIS example uses of this tool such as loading data into Postgis tables from various GIS formats or exporting data out and brief install instructions, check out our OGR2OGR Tips page.

Although GDAL/OGR was designed to deal with GIS data, a side-effect of the design is that it also handles regular attribute and relational data. The reason is simple. Spatial Data is not an island; it most often accompanies standard relational and attribute data to make it more meaningful. If you load mapping objects and geometries, you want to load the attributes of them as well.

When using OGR2OGr for loading non-spatial data ,which it really wasn't designed for, it does have some additional annoying consequences, but we feel the *almost fit in your pocket* feel of it makes it a convenient swiss army knife even for non-spatial data.

I would say we are not alone in being addicted to this tool. The ability to load data from various data sources with a simple line of script code is breath-taking. I think Bill Dollins says it well in his A One-Touch (Almost) Data Loader With FWTools and PostGIS. One only needs to look at the OGR supported formats to get a sense of just how powerful this tool is. Enough talk about the greatness of this tool; I'm beginning to make myself nauseous. On with the show.

OGR2OGR can read from any read supported format and write to any write supported format. You can imagine the permutations of data loading you can do here. To get a list of the current drivers your install supports. Launch your FW Tools command line and type:

```
ogrinfo --formats
```

(NOTE: some drivers such as the Oracle Spatial and IBM Informix Spatial Datablade require proprietary libraries so you must compile yourself)

Some Caveats - Rought Spots

1. We had hoped the -nlt NONE would prevent geometry columns from being created, but doesn't appear to or rather we haven't been successful in getting this to work, so blank geometry columns are created.
2. The overwrite also doesn't appear to work if you also include a SCHEMA. Works if you don't include a SCHEMA option.
3. Data Types such as varchar are brought in as char, integers as numeric.
4. Sometimes OGR fails if the Client Encoding is not LATIN1 and you've got high-end characters in your data. To get around this we usually set the database client_encoding to LATIN1 at least while we are loading the database with a command of the form
   `ALTER DATABASE` *`pgdbname`* `SET client_encoding=latin1;`
5. Views are brought in as if they were regular tables. Which in a lot of cases is a feature, but it would be nice to override this.
6. In older version, if you are trying to append to a schema qualified table, it seems to try to recreate the table even if you don't designate update.
7. Primary keys are not preserved and a dummy key is always created when in create mode.
8. It can't create schemas or databases
9. When using the -sql tag, the field lengths and data types are lost

For the following examples we will be a bit verbose and spell out the full connection strings to the PostgreSQL database. Keep in mind any arguments left out will use their default parameters. For example if no host or user or password or port is specified, OGR will use localhost and logged in user and 5432 port. For us this is usually wrong since our PostgreSQL server is on a separate box from our OGR2OGR scripts (e.g. we are using PgAgent that is sitting on a separate box) or is running on a non-standard port so we spell it out. We spell it out too because few tutorials spell these out, leaving many to read deeply in the docs to figure this basic stuff out. Other note - for the below - we put these on multiple lines so they fit on a page. In practice, each ogr2ogr call should be a one liner.

Getting listings of tables/views in a data source

To get a listing of tables/views (layers in GIS lingo), the ogrinfo command-line tool comes in handy. Simply pass in the connection info for the database. Below is an example of querying the tables of an ODBC datasource, PostgreSQL, Dbase:

```
ogrinfo "ODBC:some_dsn"
ogrinfo PG:"host=pghost user=pgloginname dbname=pgdbname password=pgpassword port=5432"
ogrinfo "/path/to/somefile.csv"
ogrinfo "/path/to/somefile.dbf"
```

MySQL to PostgreSQL

For starters, lets say you need to do a nightly or one-time load from MySQL to PostgreSQL. In the past, writing to MySQL with OGR was not possible, but as of OGR 1.3.2, read and write to MySQL is possible (so MySQLers 4+ are now in luck and can now load ESRI shape files among other data directly into MySQL). The PostgreSQL driver in OGR has always supported both directions. Below is a simple example of dumping general non-spatial table from MySQL to PostgreSQL. To dump a spatial MYSQL table, the command is pretty much the same.

The below statement will copy the tables customers and orders from a MySQL database to a PostgreSQL database into the schema called *mysqldump*.. All the below should be on a single line. I wrapped to save space.

--Pull customers and orders into the *mysqldump* schema

```
ogr2ogr -overwrite -update -f "PostgreSQL" PG:"host=pghost user=pgloginname dbname=pgdbname
password=pgpassword"
        MYSQL:"mydb,host=myhost,user=mylogin,password=mypassword,port=3306"
                 -lco OVERWRITE=yes -lco SCHEMA=mysqldump customers orders
```

--Pull customers and orders into the public schema, and overwrite if it exists

```
ogr2ogr -overwrite -update -f "PostgreSQL" PG:"host=pghost user=pgloginname dbname=pgdbname
password=pgpassword"
        MYSQL:"mydb,host=myhost,user=mylogin,password=mypassword,port=3306" customers orders
```

--Pull all the tables from mysql into the *mysqldump* schema

```
ogr2ogr -overwrite -update -f "PostgreSQL" PG:"host=pghost user=pgloginname dbname=pgdbname
password=pgpassword"
        MYSQL:"mydb,host=myhost,user=mylogin,password=mypassword,port=3306"
```

--For nightly loads where you already have the structure of the table -- you can do a truncate table with a psql script and then append with OGR2OGR
-- note: nln means new name for layer. In this case we want to take the mysql table orders and load into our mysqldump.orders table.

```
psql -U pgloginname -d pgdbname -h pghost -c "TRUNCATE TABLE orders"
ogr2ogr -append -f "PostgreSQL" PG:"host=pghost user=pgloginname dbname=pgdbname
password=pgpassword"
        MYSQL:"mydb,host=myhost,user=mylogin,password=mypassword,port=3306" -nln "mysqldump.orders"
orders
```

Microsoft SQL Server (and other ODBC) to PostgreSQL

OGR has a driver that supports ODBC DSN connections. Unfortunately this will only work on Windows. Below is an example that imports all tables from a SQL Server database to PostgreSQL. In this example, the DSN I have is using NT Authentication or has the name and password encoded in the System DSN. For this you need to setup a DSN connection to the SQL Server database in ODBC Manager (we described setting up a DSN in Using MS Access with PostgreSQL, except the ODBC driver seems to only work with System DSNs not File DSNs. For more details on referencing the ODBC DSN - check http://www.gdal.org/ogr/drv_odbc.html

```
ogr2ogr -f "PostgreSQL" PG:"host=pghost user=pgloginname dbname=pgdbname password=pgpassword"
        "ODBC:system_dsn_goes_here" -update -lco OVERWRITE=yes -nlt NONE
```

As noted, geometry fields seem to be created and right field types are not necessarily created. SQL Server text come in as VARCHARS and VARCHARS come in as CHARS etc. One way to get around this is to create your table structures before hand and use OGR2OGR to just load the data.

E.g.

--This will append to a table called **orders** reading from the SQL Server (ODBC) table orders and customers. It will only append like-named fields.

```
psql -h pghost -p 5432 -U pguser -d pgdbname -c "TRUNCATE TABLE orders;TRUNCATE TABLE customers;"
ogr2ogr -append -f "PostgreSQL" PG:"host=pghost user=pgloginname dbname=pgdbname
```

```
password=pgpassword port=5432"
        "ODBC:system_dsn_goes_here" orders customers
```

Or correct the structures afterword by dropping unnecessary fields using a script generation technique similar to what we described in DML to Generate DDL and DCL. So your script builder SQL to drop all the ogc_fid and wkb_geometry fields created by OGR2OGR would look something like this.

```
SELECT 'ALTER TABLE ' || t.table_schema || '.' || t.table_name || ' DROP COLUMN ogc_fid;'
FROM information_schema.tables t
   INNER JOIN information_schema.columns c
   ON (t.table_name = c.table_name AND t.table_schema = c.table_schema AND c.column_name = 'ogc_fid')
UNION
SELECT 'ALTER TABLE ' || t.table_schema || '.' || t.table_name || ' DROP COLUMN wkb_geometry;'
FROM information_schema.tables t
INNER JOIN information_schema.columns c
ON (t.table_name = c.table_name AND t.table_schema = c.table_schema AND c.column_name =
'wkb_geometry';
```

Loading Data from DBase (DBF) files

Below is an example of loading a single dbase file. Although the formats do not list DBase as a supported format, the ESRI Shapefile format includes several files (.shp, .dbf, .shx), so the OGR utility uses the ESRI shapefile driver to accomplish this.

```
ogr2ogr -f "PostgreSQL"
        PG:"host=pghost user=pgloginname dbname=pgdbname password=pgpassword port=5432"
                somefile.dbf  -nln mynewschema.mynewtable
```

Loading Data from Microsoft Accesss

For loading Microsoft Access, if you have a regular old Access database, it is best to use the ODBC driver. While the PGeo driver (ESRI Personal Geodatabase) does work against an MS Access Database, it relies on certain ESRI meta data tables to work so is not really useable for pure Access database use. One advantage of using OGR2OGR to export MS Access tables is that, unlike the MS Access export which requires you to export each table individually, one can do a bulk export of all the Microsoft Access tables or a subset of tables. The downside of using it over the Microsoft Access export feature is that its implementation of datatypes as mentioned, is impoverished, therefore things that should be text or varchar come in as char or varchar. Integers come in as numerics. Below is an example of copying Microsoft Access tables directly into PostgreSQL. To set it up, you would register a System DSN as we did for our SQL Server example. We'll show a slightly different feature here

This example apppends data from a table called **company** from an MS Access database into a table in accessdump.company in PostgreSQL, but only copies the records with company_group=2

```
ogr2ogr -append -f "PostgreSQL" PG:"host=pghost user=pgloginname dbname=pgdbname
password=pgpassword port=5432"
ODBC:"someaccessdsn" -nln "accessdump.company" -where "company_group=2" company
```

Exporting data out of PostgreSQL

As mentioned, OGR2OGR can be used to create a data dump in some supported writable format. Below is a simple example to export a table to DBase.

This example will create a table called test in the folder C:/datadbf and in addition to the dbf file an annoying .shp and a .shx that you can discard if you know you are not outputing geometries.

```
ogr2ogr -overwrite -f "ESRI Shapefile" "C:/datadbf/"
        PG:"host=pghost user=pgloginname dbname=pgdbname password=pgpassword port=5432" -nln test
somepgtable
```

If you want to output only a subset of the data in a table you can use the -sql, -select, -where properties of OGR2OGR - such as shown below

```
ogr2ogr -overwrite -f "ESRI Shapefile" "C:/datadbf/" -select "company,sector_id" -where
"company_group=2"
        PG:"host=pghost user=pgloginname dbname=pgdbname password=pgpassword port=5432"
"msaccessdump.company"
```

If you needed to use an SQL statement to rename fields and so forth, you would do it something like the below. The main issue we have with this is that the OGR tool is not smart enough to figure out the data types and lengths to output the fields so lengths etc are wrong. In general if we need to rename fields, we create a view in PostgreSQL and output the view.

```
ogr2ogr -overwrite -f "ESRI Shapefile" "C:\datadbf"
        -sql "SELECT company_name As company,sector_id FROM accessdump.company"
        PG:"host=pghost user=pgloginname dbname=pgdbname password=pgpassword port=5432" -nln
compsec2
```

Back to Table Of Contents

Reader Comments

Journal Changes Etc

Peter Manchev
Hi there,

I can give you a hand in preparing the Java/Struts2 version for the magazine. Just let me know, OK?

Pete

Leo
Pete,

Thanks. We'll let you know when we have our examples prepared and of course we'll give you due credit.

PostgreSQL 8.3 is out and the Project Moves On

Robert Treat
"While you can now change data types of columns in tables - you are somewhat limited as to what you can change them to"

Actually I am quite certain that PostgreSQL offers more flexibility in this regard than any other database (commercial or floss) given that you are not limited to casting to change data types, you can use user defined functions as well. And when you think about the idea that you can make those functions in any language (perl, java, python, php, tcl and on and on) really the capabilities are close to limitless if you really need it.

Now, what would be a nice improvement in this area would be an optimization to allow transparent data type changes to occur solely at the catalog level, rather than requiring a table rewrite, but that's a different rant all-together. :-)

Vincenzo Romano
I'll repeat it ad nauseam!

Leo and Regina
Indeed you are right. This is one of those duh moments. I guess we can take that off the list of rants and put it in our Q & A section. So I guess the

ALTER TABLE table ALTER COLUMN col TYPE new_data_type USING CAST(col as new_data_type);

was introduced in 8.0 or at least that's when the docs first mention it?

Thanks for the tip.

Jan Ischebeck
Windowing Functions are very important in the ETL area. LAG / LEAD functions in Oracle are f.e. essential to calculate a valid_to out of a valid_from date.

I hope that basic windowing functions will make it into 8.4.

PS: MS SQL, Oracle, DB2 and Terradata support windowing
functions.

Postgres OnLine Journal
As Robert Treat pointed out in our PostgreSQL 8.3 is out and the Project Moves On,
one of the features that was introduced in PostgreSQL 8.0 was the syntax of
ALTER TABLE sometable
ALTER COLUMN somecolumn TYPE new_data_type
USING some_function_c

LewisC
Quote: One of the great things about PostgreSQL, like many other Open source projects/products - is that you don't need to wait 3-5 years to see changes and then have this gigantic thing that you need to engulf as you do with the likes of Microsoft, Oracle and IBM commercial databases. That is perhaps the most attractive thing about it. :End Quote

From an enterprise perspective, that is not really attractive. Large companies usually have many databases and upgrades are time consuming and expensive. Postgres can be a painful upgrade and rather than frequent patches, it has frequent major upgrades. It's kind of like saying Fedora is better than Redhat because it is updated frequently.

LewisC

Regina
True - but from experience upgrading from something like SQL Server 2000 to SQL Server 2005 was much more painful than the PostgreSQL pseudo major upgrades (in terms of cost of fixing applications broken by it, switching DTS packages to SSIS etc - though you could run in downward compatibility mode, but then in downward mode you couldn't take advantage of a lot of new features). Granted lots of work needs to be put in to make this much less painful to make PostgreSQL more attractive since ,gasp - a lot of ISPs that claim to support PostgreSQL are running only 7.4 or 8.0. What's the fun in that.

Sure the sum of each pseudo major upgrade may be more than the pain of one super upgrade. If you are lacking some key nice feature you want you don't want to wait 5 years to get it. It is not a choice one would or should always take advantage of, but I think the fact the choice is there is good. Technology is moving way too quickly and competition as well to have to always wait for 3-5 years.

Anonymouse
I make heavy use of windowing and analytic functions in Oracle, to the point where I hardly know how I lived without them. In fact, it's the one thing keeping me from moving to PostgreSQL, and I'd love to see them supported in 8.4. BTW, MS SQL's support in woefully incomplete. Oracle's is much more flexible (don't know about DB2 or Teradata).

Moving tables from one schema to another

Tom Lane
Uh ... why not ALTER TABLE SET SCHEMA? That function seems to be a holdover from 8.0 or before
(and it doesn't fix everything it should, either).

David Fetter
Just generally, it's a bad idea to fool directly with the catalog, especially when there are user land commands that do the same thing.

Regina
Oops I guess I missed that development. So was ALTER TABLE SET SCHEMA introduced in 8.0 or 8.1. There is no mention of it until the 8.1 docs
8.0 docs
http://www.postgresql.org/docs/8.0/interactive/sql-altertable.html

8.1 docs
http://www.postgresql.org/docs/8.1/interactive/sql-altertable.html

I presume it doesn't correct postgis geometry_columns table, but I could be wrong. Anyrate I'll update the above article to note the introduction of this new option and that it is the preferred way.

Bernd Helmle
Using you're suggested way will likely lead to a catalog completely messed up, since it doesn't handle implicit sequences, indexes and (more worse) pg_depend. ALTER TABLE SET SCHEMA is the *only* way to do that safely.

Tom Lane
Right, pg_depend was what I was worried about. Although failing to move the table's indexes to the new schema would probably confuse some things too.

The business about geometry_columns is worrisome too --- it looks like postgis might have built in a foot-gun. However, since this code is obviously several releases old, perhaps the postgis guys have fixed that since then.

Regina
I think the next plan for postgis as far as geometry_columns was to make it a view or put in a trigger to update automagically. To my knowledge that has not been done yet.

Regarding the index issue. Where exactly is schema referenced there that it would be an issue. As far as I could tell, the index is only dependent on the table it is on and the schema it is assumed to live in is the same as the table that owns it. At list that is the way the pg_indexes view looks like it is written to reference the schema of the table that owns the index. If I query the pg_indexes view in pg_catalog, it looks correct and when I look at pg_index table, I see no reference to a schema.

Bernd is right that sequences didn't move over and pg_depends is messed up as a result, presumably its still got dependency info for the old schema.

Bernd Helmle
Indexes references schemas in pg_class, at least. There are places where we assume that an index lives in the same namespace than the table it belongs to, so we are likely to confuse things if this isn't going to be true anymore...

Regina
Ah yes I see it. At the very least this whole exercise has been very educational for us.

I think part of the reason we didn't notice this ALTER TABLE SET SCHEMA feature was present is that we use PgAdmin a lot for easy administration and in PgAdmin - it doesn't allow you to change the schema of a table (at least not in 1.8.1 Jan 2008), but it does allow you to move it to a different tablespace, so we assumed this was a feature not implemented yet.

Seems like it would be a good thing to add to PgAdmin since I imagine we are not alone in using it as an administrative tool. I'll bring that up in pgAdmin group.

DML to generate DDL and DCL- Making structural and Permission changes to multiple tables

David Fetter
That SQL is pretty Rube Goldberg. Try removing the array_to_string(array()) stuff and tacking a ';' to the ends.

Leo and Regina
Actually thats the original way we had it, but didn't like that since it would give one row per command. For our purposes - since we are using pgAdmin a lot or dumping this in a single EXECUTE(..) call, it was easier to have a single field that has the full script rather than multiple records.

I guess you have a point though that it does make things look a little bit more complicated than they really are.

Reading PgAdmin Graphical Explain Plans

Steinam's Blog
Folgender Blog beschreibt sehr schön das explain-Statement von Postgres mit Hilfe des Admin-Tools pgadmin

PGAdmin graphical Explain

Postgres OnLine Journal
What is Constraint Exclusion?
Constraint Exclusion is a feature introduced in PostgreSQL 8.1 which is used in conjunction with Table Inheritance to implement
table partitioning strategies. The basic idea is you put check constraints on tables to limit

---

New Features for PostgreSQL Stored Functions

Gurjeet Singh
In the plan you obtained after setting function with 'ROWS 3000', it is showing 'HASH JOIN' whereas you say in the following text: 'planner changes strategies to a Merge Join'. There seems to be a mismatch.

Regina
Indeed you are right. I think the first time I did it , it was a merge join, but then I was experimenting with some other settings on my final run and assumed the general plan hadn't changed so didn't resnapshot the graphical explain. I'll rerun the test again and update the article.

Thanks for the catch!
