# Postgres OnLine Journal: August 2008 / September 2008

An in-depth Exploration of the PostgreSQL Open Source Database

## Table Of Contents

From the Editors

Fibonacci, Graphs and Recursive Queries *Advanced*

What's new and upcoming in PostgreSQL

CTEs and Windowing Functions in 8.4

PostgreSQL Q & A

How to determine which tables are missing indexes *Intermediate*

How to determine if text phrase exists in a table column *Intermediate*

Basics

How to restore select tables, select objects, and schemas from Pg Backup *Beginner*

PgAdmin III 1.9 First Glance *Beginner*

PL Programming

Build Median Aggregate Function in SQL *Intermediate*

More Aggregate Fun: Who's on First and Who's on Last *Intermediate*

Product Showcase

OpenJump for PostGIS Spatial Ad-Hoc Queries *Beginner*

*Reader Comments*

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

## Fibonacci, Graphs and Recursive Queries *Advanced*

One thing I'm really looking forward to have in the upcoming PostgreSQL 8.4 is the introduction of the WITH RECURSIVE feature that IBM DB2 and SQL Server 2005 already have. Oracle has it too but in a non-standard CONNECT BY so is much less portable. This is a feature that is perhaps more important to us for the kind of work we do than the much complained about lack of windowing functions.

I was recently taking a snoop at IBM DB2 newsletter. Why I read magazines and newsletters on databases I don't even use I guess is to see what I'm missing out on and to sound remotely educated on the topic when I run into one of *those people*. I also have a general fascination with magazines. In it their latest newsletter they had examples of doing Fibonacci and Graphs with Common Table Expressions (CTEs).

Robert Mala's Fibonacci CTE
Robert Mala's Graph CTE

Compare the above to David Fetter's Fibonacci Memoizing example he posted in our comments way back when.

I'd be interested in seeing what solutions David and others come out with using new features of 8.4. We can see a before 8.4 and after 8.4 recipe.

As a slightly off-topic side note - of all the Database magazines I have read - Oracle Magazine is the absolute worst. SQL Server Magazine and IBM DB2 are pretty decent. The real problem is that Oracle's magazine is not even a database magazine. Its a mishmash of every Oracle offering known to man squashed into a compendium that can satisfy no one. You would think that Oracle as big as their database is would have a magazine dedicated to just that. Perhaps there is another magazine besides Oracle Magazine, but haven't found it so I would be interested to know if I missed something.

Back to Table Of Contents  Fibonacci, Graphs and Recursive Queries Reader Comments

## CTEs and Windowing Functions in 8.4

As we mentioned in a previous article, RECURSIVE queries, often referred to in SQL ANSI specs and by DB2 and SQL Server as Common Table Expressions (CTE) will make it into the 8.4 release and can already be found in the dev source. Technically CTE is a superset and RECURSIVE queries are a subclass of CTE. Looks like basic windowing functionality will make it in 8.4 as well.

A summary of where your favorite patches are at can be found at the September 2008 PostgreSQL 8.4 commit-fest summary page http://wiki.postgresql.org/wiki/CommitFest:2008-09.

### What the hell is a RECURSIVE query and a common table expression (CTE) and why should I care?

CTE just specifies a way of defining a commonly used table expression (sort of like a view, but can also be used within a view (at least in SQL Server), we admittedly haven't experimented with 8.4 yet). A recursive CTE is a CTE that uses itself to define itself. There are two main reasons why people use CTEs (or rather why we use them).

- Simplify repetitively used select statements but that are not used outside of a specific body of work. True you could often break these out as SQL functions, but clutters the space if not used anywhere else and violates our general rule of thumb of keep code closest in contextual space to where it is most used so its purpose is obvious and it can be more easily extricated when it becomes obsolete.
- Create recursive queries - such as those defining tree structures. Again in many cases you can perform these tricks in current PostgreSQL versions already using recursive stored functions.

Some good examples on how this would work in 8.4 can be lifted off the recent hackers thread on WITH RECURSIVE.

### ALAS windowing functions

Well PostgreSQL 8.4 won't have complete support of Windowing Functions, but it looks like it will be on par or slightly better than what is available in SQL Server 2005, but not quite as good as Oracle and DB2. So to summarize from discussions read.

What will make it:

- windowed aggregates
- cooperate with GROUP BY aggregates
- Ranking and ROW_NUMBER()
- WINDOW clause

What will **NOT** make it:

- sliding window (window framing)
- lead(), lag(), etc. that reach for random rows
- user defined window functions

Details of what is coming and what's dropped and the general saga can be found at the following links:

http://umitanuki.net/pgsql/wfv04/design.html and also the hackers Windowing thread http://archives.postgresql.org/pgsql-hackers/2008-09/msg00001.php.

### YUM 8.4 snapshots

For those running RedHat EL, Fedora or CentOS and too lazy to compile yourself, check out Devrim's 8.4 RPM snapshots which will be released every week during commitfest via the new PostgreSQL Yum repository.

Back to Table Of Contents

## How to determine which tables are missing indexes ***Intermediate***

Every once in a while - particularly if you are using inherited tables, you forget to put an important index on one of your tables which bogs down critical queries. Its sometimes convenient to inspect the index catalog to see what tables are missing indexes or what tables are missing a critical index. Normally we try to stick with querying the information_schema because queries against that schema work pretty much the same in PostgreSQL as they do in SQL Server and MySQL. For most of the examples below we had to delve into pg_catalog schema territory since there was no view we could find in information_schema that would give us enough detail about indexes.

**Problem**: Return all non-system tables that are missing primary keys

**Solution**:

This will actually work equally well on SQL Server, MySQL and any other database that supports the Information_Schema standard. It won't check for unique indexes though.

```
SELECT c.table_schema, c.table_name, c.table_type
FROM information_schema.tables c
WHERE c.table_type = 'BASE TABLE' AND c.table_schema NOT IN('information_schema', 'pg_catalog')
AND
NOT EXISTS (SELECT cu.table_name
                                FROM information_schema.key_column_usage cu
                                WHERE cu.table_schema = c.table_schema AND
                                        cu.table_name = c.table_name)
ORDER BY c.table_schema, c.table_name;
```

**Problem**: Return all non-system tables that are missing primary keys and have no unique indexes

**Solution** - this one is not quite as portable. We had to delve into the pg_catalog since we couldn't find a table in information schema that would tell us anything about any indexes but primary keys and foreign keys. Even though in theory primary keys and unique indexes are the same, they are not from a meta data standpoint.

```
SELECT c.table_schema, c.table_name, c.table_type
FROM information_schema.tables c
WHERE  c.table_schema NOT IN('information_schema', 'pg_catalog') AND c.table_type = 'BASE TABLE'
AND NOT EXISTS(SELECT i.tablename
                                FROM pg_catalog.pg_indexes i
                        WHERE i.schemaname = c.table_schema
                                AND i.tablename = c.table_name AND indexdef LIKE '%UNIQUE%')
AND
NOT EXISTS (SELECT cu.table_name
                                FROM information_schema.key_column_usage cu
                                WHERE cu.table_schema = c.table_schema AND
                                        cu.table_name = c.table_name)
ORDER BY c.table_schema, c.table_name;
```

**Problem** - List all tables with geometry fields that have no index on the geometry field.

**Solution -**

```
SELECT c.table_schema, c.table_name, c.column_name
FROM (SELECT * FROM
        information_schema.tables WHERE table_type = 'BASE TABLE') As t  INNER JOIN
        (SELECT * FROM information_schema.columns WHERE udt_name = 'geometry') c
                ON (t.table_name = c.table_name AND t.table_schema = c.table_schema)
                LEFT JOIN pg_catalog.pg_indexes i ON
                        (i.tablename = c.table_name AND i.schemaname = c.table_schema
                                AND  indexdef LIKE '%' || c.column_name || '%')
WHERE i.tablename IS NULL
ORDER BY c.table_schema, c.table_name;
```

Back to Table Of Contents

## How to determine if text phrase exists in a table column *Intermediate*

**Common Case Scenario:**

You have a very aggravated person who demands you purge their email from any table you have in your system. You have lots of tables that have email addresses. How do you find which tables have this person's email address.

Below is a handy plpgsql function we wrote that does the following. Given a search criteria, field name pattern, table_name pattern, schema name pattern, data type pattern, and max length of field to check, it will search all fields in the database fitting those patterns and return to you the names of these schema.table.field names that contain the search phrase.

To use the below you would do something like:

```
SELECT pc_search_tablefield('%john@hotmail%', '%email%', '%', '%', '%', null);
```

The above will return all database field names that have the phrase email in the field name and that contain the term *john@hotmail*

Code of the function looks like this:

```
CREATE OR REPLACE FUNCTION pc_search_tablefield(param_search text, param_field_like text, param_table_like
text,
    param_schema_like text, param_datatype_like text, param_max_length integer)
  RETURNS text AS
$$
DECLARE
    result text := '';
    var_match text := '';
    searchsql text := '';
BEGIN
    searchsql := array_to_string(ARRAY(SELECT 'SELECT ' || quote_literal(quote_ident(c.table_schema) || '.'
        || quote_ident(c.table_name) || '.' || quote_ident(c.column_name)) ||
                ' WHERE EXISTS(SELECT ' || quote_ident(c.column_name) || ' FROM '
                 || quote_ident(c.table_schema) || '.' || quote_ident(c.table_name) ||
                 ' WHERE ' || CASE WHEN c.data_type IN('character', 'character varying', 'text') THEN
                    quote_ident(c.column_name) ELSE 'CAST(' || quote_ident(c.column_name) || ' As text) ' END
                    || ' LIKE ' || quote_literal(param_search) || ') ' As subsql
                FROM information_schema.columns c
                WHERE c.table_schema NOT IN('pg_catalog', 'information_schema')
                    AND c.table_name LIKE param_table_like
                    AND c.table_schema LIKE param_schema_like
                    AND c.column_name LIKE param_field_like
                    AND c.data_type IN('"char"','character', 'character varying', 'integer', 'numeric', 'real', 'text')
                        AND c.data_type LIKE param_datatype_like
                    AND (param_max_length IS NULL OR param_max_length = 0
                        OR character_maximum_length <= param_max_length) AND
                        EXISTS(SELECT t.table_name
                            FROM information_schema.tables t
                            WHERE t.table_type = 'BASE TABLE'
                                AND t.table_name = c.table_name AND t.table_schema = c.table_schema)),
                    ' UNION ALL ' || E'\r');
    --do an exists check thru all tables/fields that match field table pattern
    --return those schema.table.fields that contain search pattern information
    RAISE NOTICE '%', searchsql;
    IF searchsql > '' THEN
        FOR var_match IN EXECUTE(searchsql) LOOP
            IF result > '' THEN
                result := result || ';' || var_match;
            ELSE
                result := var_match;
            END IF;
        END LOOP;
    END IF;
    RETURN result;
END;$$
  LANGUAGE 'plpgsql' VOLATILE SECURITY DEFINER;
```

Back to Table Of Contents  How to determine if text phrase exists in a table column Reader Comments

## How to restore select tables, select objects, and schemas from Pg Backup *Beginner*

One of the nice things about the PostgreSQL command-line restore tool is the ease with which you can restore select objects from a backup. We tend to use schemas for logical groupings which are partitioned by context, time, geography etc. Often times when we are testing things, we just want to restore one schema or set of tables from our backup because restoring a 100 gigabyte database takes a lot of space, takes more time and is unnecessary for our needs. In order to be able to accomplish such a feat, you need to create tar or compressed (PG custom format) backups. We usually maintain PG custom backups of each of our databases.

### Restoring a whole schema

Below is a snippet of how you would restore a schema including all its objects to a dev database or some other database.

```
psql -d devdbgoeshere -U usernamegoeshere -c "CREATE SCHEMA someschema"
pg_restore -d devdbgoeshere --format=c -U usernamegoeshere --schema="someschema" --verbose "/path/
to/somecustomcompressed.backup"
```

### Restoring a select set of objects

Now restoring a single table or set of objects is doable, but surprisingly more annoying than restoring a whole schema of objects. It seems if you try to restore a table, it doesn't restore the related stuff, so what we do is first create a table of contents of stuff we want to restore and then use that to restore.

To create a table of contents of stuff to restore do this:

```
pg_restore --list "/path/to/somecustomcompressed.backup" --file="mytoc.list"
```

Then simply open up the text file created from above and cut out all the stuff you don't want to restore. Then feed this into the below restore command.

```
pg_restore -v --username=usernamegoeshere --dbname=devdbgoeshere --use-list="mytoc.list" "/path/
to/somecustomcompressed.backup"
```

Sorry for the mix and match - note -U --username=, -d --dbname= etc. are interchangeable. For more details on how to use these various switches, check out our PostgreSQL Pg_dump Pg_Restore Cheatsheet.

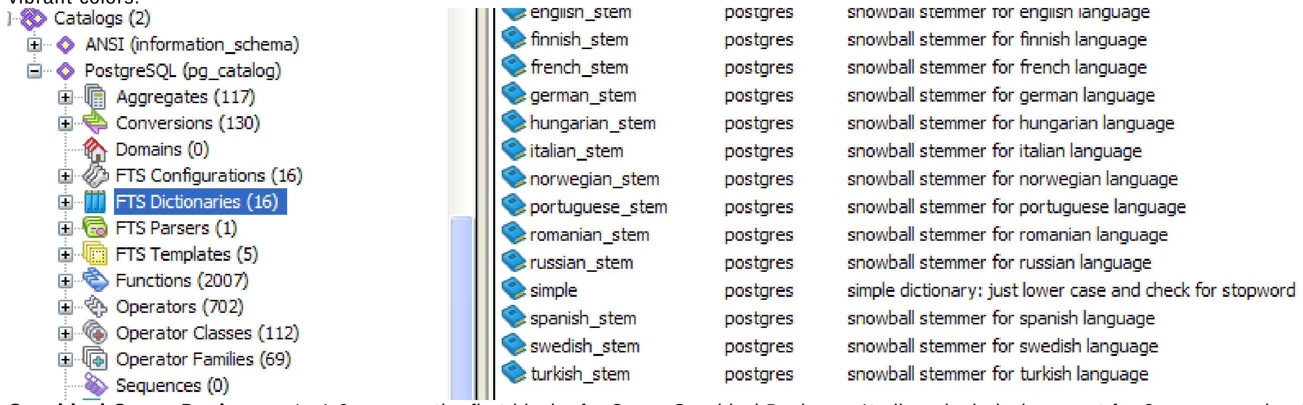
Back to Table Of Contents

## PgAdmin III 1.9 First Glance *Beginner*

We've been playing around with the snapshot builds of PgAdmin III 1.9 and would like to summarize some of the new nice features added. PgAdmin III 1.9 has not been released yet, but has a couple of neat features brewing.

For those interested in experimenting with the snapshot builds and src tarballs, you can download them from http://www.pgadmin.org/snapshots/

1. **Support for TSearch Free Text Search Engine.** If you have a 8.3 database and are using 1.9, then you will see the new kid on the block in vibrant colors.



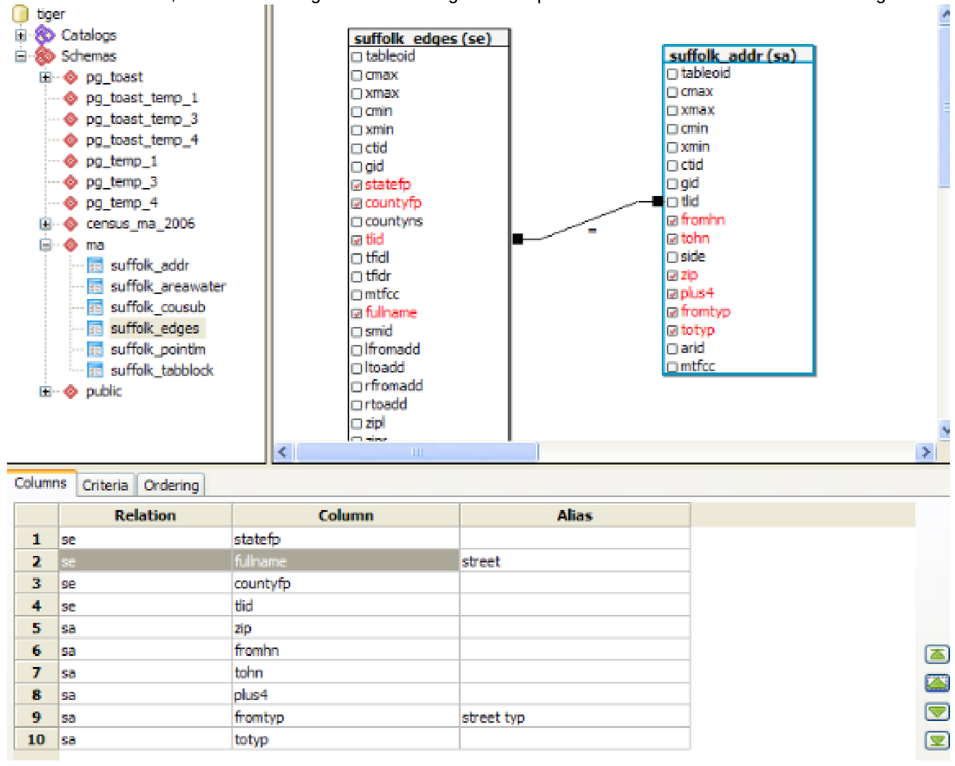
Catalogs (2)
- ANSI (information_schema)
- PostgreSQL (pg_catalog)
  - Aggregates (117)
  - Conversions (130)
  - Domains (0)
  - FTS Configurations (16)
  - FTS Dictionaries (16)
  - FTS Parsers (1)
  - FTS Templates (5)
  - Functions (2007)
  - Operators (702)
  - Operator Classes (112)
  - Operator Families (69)
  - Sequences (0)

| | | |
|---|---|---|
| english_stem | postgres | snowball stemmer for english language |
| finnish_stem | postgres | snowball stemmer for finnish language |
| french_stem | postgres | snowball stemmer for french language |
| german_stem | postgres | snowball stemmer for german language |
| hungarian_stem | postgres | snowball stemmer for hungarian language |
| italian_stem | postgres | snowball stemmer for italian language |
| norwegian_stem | postgres | snowball stemmer for norwegian language |
| portuguese_stem | postgres | snowball stemmer for portuguese language |
| romanian_stem | postgres | snowball stemmer for romanian language |
| russian_stem | postgres | snowball stemmer for russian language |
| simple | postgres | simple dictionary: just lower case and check for stopword |
| spanish_stem | postgres | snowball stemmer for spanish language |
| swedish_stem | postgres | snowball stemmer for swedish language |
| turkish_stem | postgres | snowball stemmer for turkish language |

2. **Graphical Query Designer** - In 1.9, we see the first blush of a Query Graphical Designer. It all worked nicely except for 2 very annoying things. The way it makes joins - they are implemented as WHERE conditions instead of INNER JOIN, LEFT JOIN, RIGHT JOIN etc which makes its join type implementation <= and >= just incorrect. The other thing is that you can't write an SQL statement and then toggle to the graphical view as you can in *hmm MS Access* or *SQL Server Enterprise Manager*, although you can graphically design a query and toggle to the SQL View. As Leo likes to say - "Query graphical designers are over-rated. They breed bad habits. I only care about having a graphical relational designer. Call me when PgAdmin III has that." *And off Leo trotted, back to his Open Office Relational Designer.*

   I still find it quite beautiful and useful in its current incarnation. It at the very least saves a couple of keystrokes. Below are some key niceties it currently provides.

   - Ability to see all the schemas, tables at one glance and drag and drop them. Standard and the red-fonting of selected fields I find to be a



tiger
- Catalogs
- Schemas
  - pg_toast
  - pg_toast_temp_1
  - pg_toast_temp_3
  - pg_toast_temp_4
  - pg_temp_1
  - pg_temp_3
  - pg_temp_4
  - census_ma_2006
  - ma
    - suffolk_addr
    - suffolk_areawater
    - suffolk_cousub
    - suffolk_edges
    - suffolk_pointlm
    - suffolk_tabblock
  - public

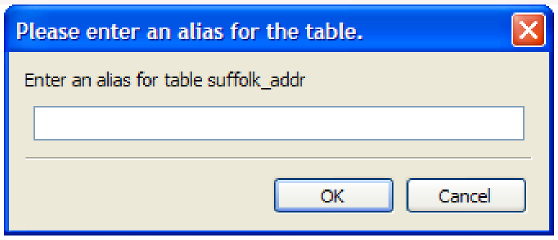
suffolk_edges (se): ☐ tableoid, ☐ cmax, ☐ xmax, ☐ cmin, ☐ xmin, ☐ ctid, ☐ gid, ☑ statefp, ☑ countyfp, ☐ countyns, ☑ tlid, ☐ tfidl, ☐ tfidr, ☐ mtfcc, ☑ fullname, ☐ smid, ☐ lfromadd, ☐ ltoadd, ☐ rfromadd, ☐ rtoadd, ☐ zipl

suffolk_addr (sa): ☐ tableoid, ☐ cmax, ☐ xmax, ☐ cmin, ☐ xmin, ☐ ctid, ☐ gid, ☐ tlid, ☑ fromhn, ☑ tohn, ☐ side, ☑ zip, ☑ plus4, ☑ fromtyp, ☑ totyp, ☐ arid, ☐ mtfcc

Columns | Criteria | Ordering

| | Relation | Column | Alias |
|---|---|---|---|
| 1 | se | statefp | |
| 2 | se | fullname | street |
| 3 | se | countyfp | |
| 4 | se | tlid | |
| 5 | sa | zip | |
| 6 | sa | fromhn | |
| 7 | sa | tohn | |
| 8 | sa | plus4 | |
| 9 | sa | fromtyp | street typ |
| 10 | sa | totyp | |

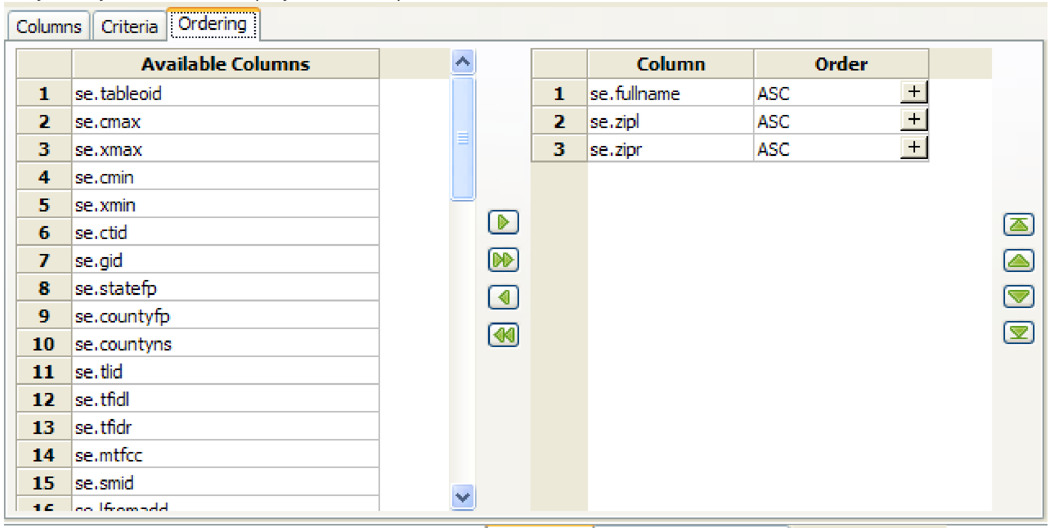
nice touch.



Please enter an alias for the table.

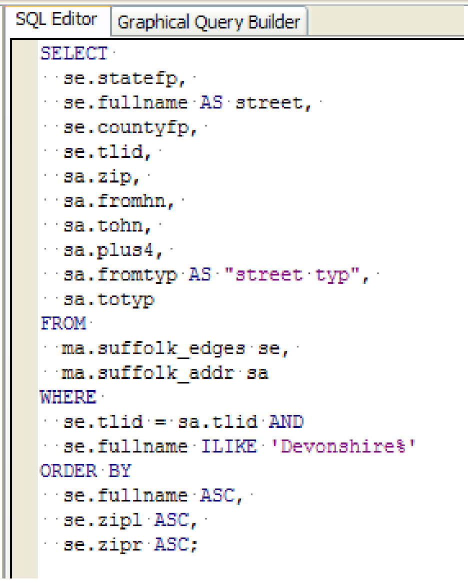
Enter an alias for table suffolk_addr

- Alias table names, fields with one click -
- Basic Criteria support that allows to navigate tree to pick a field - still a bit buggy when you try to delete criteria though
- Easy ability to sort field display order and put in order of fields.



| | Available Columns |
|---|---|
| 1 | se.tableoid |
| 2 | se.cmax |
| 3 | se.xmax |
| 4 | se.cmin |
| 5 | se.xmin |
| 6 | se.ctid |
| 7 | se.gid |
| 8 | se.statefp |
| 9 | se.countyfp |
| 10 | se.countyns |
| 11 | se.tlid |
| 12 | se.tfidl |
| 13 | se.tfidr |
| 14 | se.mtfcc |
| 15 | se.smid |
| 16 | se.lfromadd |

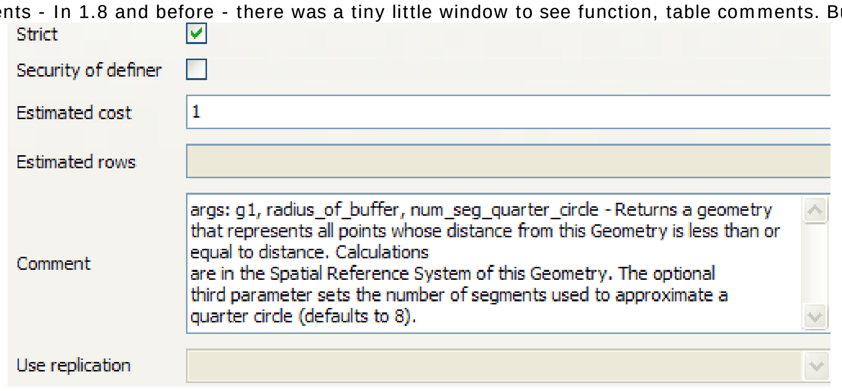
| | Column | Order |
|---|---|---|
| 1 | se.fullname | ASC |
| 2 | se.zipl | ASC |
| 3 | se.zipr | ASC |



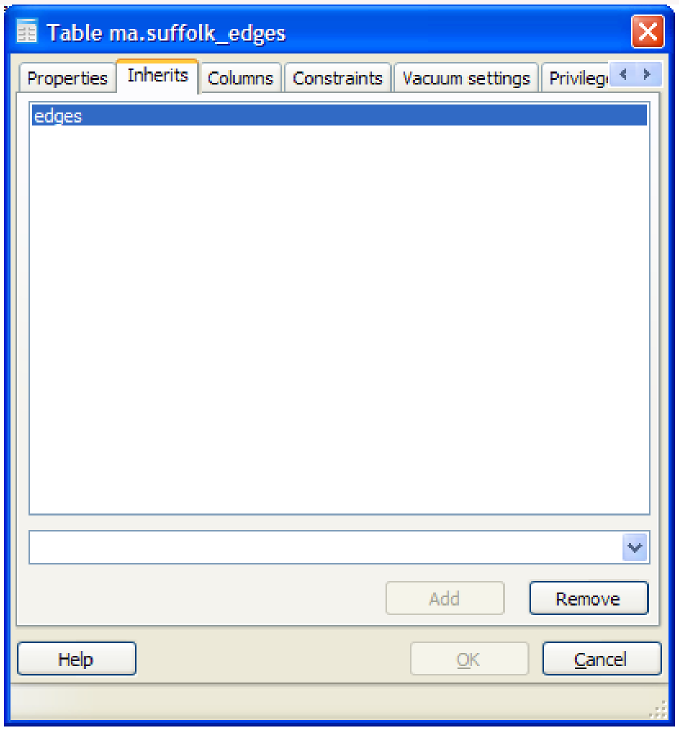
```
SELECT 
  se.statefp, 
  se.fullname AS street, 
  se.countyfp, 
  se.tlid, 
  sa.zip, 
  sa.fromhn, 
  sa.tohn, 
  sa.plus4, 
  sa.fromtyp AS "street typ", 
  sa.totyp
FROM 
  ma.suffolk_edges se, 
  ma.suffolk_addr sa
WHERE 
  se.tlid = sa.tlid AND
  se.fullname ILIKE 'Devonshire%'
ORDER BY
  se.fullname ASC, 
  se.zipl ASC, 
  se.zipr ASC;
```

- Toggle to the SQL View to see the generated SQL -

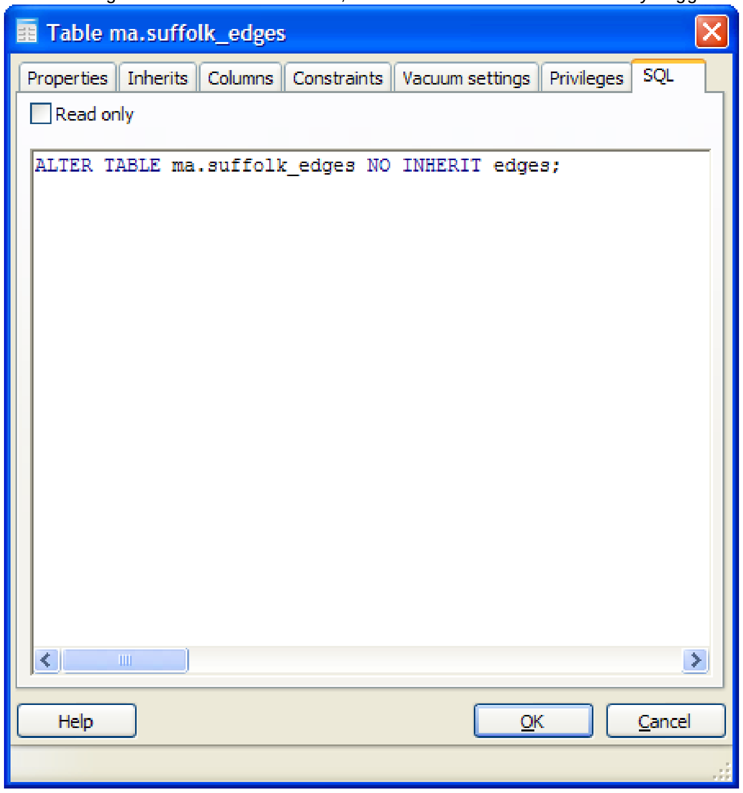
3. Finally you can see full comments - In 1.8 and before - there was a tiny little window to see function, table comments. But now its a nice big scrollable/expandable window.



| | |
|---|---|
| Strict | ✓ |
| Security of definer | ☐ |
| Estimated cost | 1 |
| Estimated rows | |
| Comment | args: g1, radius_of_buffer, num_seg_quarter_circle - Returns a geometry that represents all points whose distance from this Geometry is less than or equal to distance. Calculations are in the Spatial Reference System of this Geometry. The optional third parameter sets the number of segments used to approximate a quarter circle (defaults to 8). |
| Use replication | |



4. Ability to uninherit a table.
5. Just as in prior 1.8 after making a change to a table, you can toggle to the SQL tab before clicking OK to see what will be executed. This is an immeasurable learning tool. In addition to that, there is an additional Read Only toggle that allows you to uncheck and add additional SQL DDL of your own.



☐ Read only

```
ALTER TABLE ma.suffolk_edges NO INHERIT edges;
```

Back to Table Of Contents

## Build Median Aggregate Function in SQL *Intermediate*

One of the things we love most about PostgreSQL is the ease with which one can define new aggregate functions with even a language as succinct as SQL. Normally when we have needed a median function, we've just used the built-in median function in PL/R as we briefly demonstrated in Language Architecture in PostgreSQL.

If all you demand is a simple median aggregate function ever then installing the whole R statistical environment so you can use PL/R is overkill and much less portable.

In this article we will demonstrate how to create a Median function with nothing but the built-in PostgreSQL SQL language, array constructs, and functions.

### Primer on PostgreSQL Aggregate Structure

PostgreSQL has a very simple but elegant architecture for defining aggregates. Aggregates can be defined using any functions, built-in languages and PL languages you have installed. You can even mix and match languages if you want to take advantage of the unique speed optimization/library features of each language. Below are the basic steps to building an aggregate function in PostgreSQL.

1. Define a start function that will take in the values of a result set - this can be in the PL/built-in language of your choosing or you can use one that already exists.
2. Define an end function that will do something with the final output of the start function - this can be in the PL/built-in language of your choosing or you can use one that already exists.
3. If the intermediary type returned by your start function, does not exist, then create it
4. Now define the aggregate with steps that look something like this:

```
        CREATE AGGREGATE median(numeric) (
                  SFUNC=array_append,
                  STYPE=numeric[],
                  FINALFUNC=array_median
                );
```

5. NOTE: As Tom Lane pointed out in comments below, the following is not entirely true. Since all arrays can be cast to anyarray datatype. You can use anyarray to use the same function for all data types assuming you want all medians to behave the same regardless of data type. We shall demonstrate this in our next aggregate example
This part is a bit annoying. You need to define an aggregate for each data type you need it to work with that doesn't automatically cast to a predefined type. The above example will only work for numbers because all numbers can be automatically cast to a numeric. However if we needed a median for dates, we would also need to do

```
        CREATE AGGREGATE median(timestamp) (
                  SFUNC=array_append,
                  STYPE=timestamp[],
                  FINALFUNC=array_median
                );
```

and also define a array_median function for dates. Keep in mind that PostgreSQL supports function overloading which means we can have all these functions named the same as long as they take different data type inputs. This allows the final user of our median function not to worry about whether they are taking a median for dates or numbers and just call the aggregate median().

### Build our Median Aggregate

In the steps that follow we shall flesh out the FINALFUNC function. Please note that array_append is a built-in function in PostgreSQL that takes an array and keeps on appending elements to the array. So conveniently - we don't need to define an SFUNC as we would normally.

Now what makes creating a median aggregate function harder than say an Average is that it cares about order and needs to look at all items to determine what to return. This means that unlike average, sum, max, min etc - we need to look at all values passed to us, resort it based on the data type sorting rules of that data type and return the middle item. Here is where the beauty of array_append saves us.

Now lets get started. We have conveniently everything we need gratis from PostgreSQL. Now all we need are our array_median functions that will take in our array of items collected during the group process, junk the nulls and resort whats left and then return the middle item.

NOTES:

1. you can instead of using the array_append directly, create an intermediary that rejects nulls. That would probably perform better but require a bit more code.
2. When there are ties, the customary thing is to average the ties, for our particular use case, we wanted the result to be in the list, so we are simply taking the last in the average set.
3. You see the multiply by 2.0, that is needed because 1/2 is 0 in SQL because it needs to return the same data type as the inputs. To get around that we force the 2 to be a decimal.

So the code looks like this:

```
CREATE OR REPLACE FUNCTION array_median(numeric[])
  RETURNS numeric AS
$$
    SELECT CASE WHEN array_upper($1,1) = 0 THEN null ELSE asorted[ceiling(array_upper(asorted,1)/2.0)] END
    FROM (SELECT ARRAY(SELECT ($1)[n] FROM
generate_series(1, array_upper($1, 1)) AS n
    WHERE ($1)[n] IS NOT NULL
            ORDER BY ($1)[n]
) As asorted) As foo ;
$$
  LANGUAGE 'sql' IMMUTABLE;


CREATE OR REPLACE FUNCTION array_median(timestamp[])
  RETURNS timestamp AS
$$
    SELECT CASE WHEN array_upper($1,1) = 0 THEN null ELSE asorted[ceiling(array_upper(asorted,1)/2.0)] END
    FROM (SELECT ARRAY(SELECT ($1)[n] FROM
generate_series(1, array_upper($1, 1)) AS n
    WHERE ($1)[n] IS NOT NULL
            ORDER BY ($1)[n]
) As asorted) As foo ;
$$
  LANGUAGE 'sql' IMMUTABLE;


CREATE AGGREGATE median(numeric) (
  SFUNC=array_append,
  STYPE=numeric[],
  FINALFUNC=array_median
);

CREATE AGGREGATE median(timestamp) (
  SFUNC=array_append,
  STYPE=timestamp[],
  FINALFUNC=array_median
);
```

Now the tests

```
----TESTS numeric median - 16ms
SELECT m, median(n) As themedian, avg(n) as theavg
FROM generate_series(1, 58, 3) n, generate_series(1,5) m
WHERE n > m*2
GROUP BY m
ORDER BY m;

--Yields
m | themedian |       theavg
---+-----------+---------------------
 1 |        31 | 31.0000000000000000
 2 |        31 | 32.5000000000000000
 3 |        31 | 32.5000000000000000
 4 |        34 | 34.0000000000000000
 5 |        34 | 35.5000000000000000

 SELECT m, n
FROM generate_series(1, 58, 3) n, generate_series(1,5) m
WHERE n > m*2 and m = 1
ORDER BY m, n;
--Yields


 m | n
---+----
 1 |  4
 1 |  7
 1 | 10
 1 | 13
 1 | 16
 1 | 19
 1 | 22
 1 | 25
 1 | 28
 1 | 31
 1 | 34
 1 | 37
 1 | 40
 1 | 43
 1 | 46
 1 | 49
 1 | 52
 1 | 55
 1 | 58
```

```
--Test to ensure feeding numbers out of order still works
SELECT avg(x), median(x)
FROM (SELECT 3 As x
   UNION ALL
   SELECT - 1 As x
   UNION ALL
   SELECT 11 As x
   UNION ALL
   SELECT 10 As x
   UNION ALL
   SELECT 9 As x) As foo;

--Yields -


avg | median
--+------------
6.4 | 9

---TEST date median -NOTE: average is undefined for dates so we left that out. 16ms
SELECT m, median(CAST('2008-01-01' As date) + n) As themedian
   FROM generate_series(1, 58, 3) n, generate_series(1,5) m
   WHERE n > m*2
GROUP BY m
ORDER BY m;


 m |      themedian
---+--------------------
 1 | 2008-02-01 00:00:00
 2 | 2008-02-01 00:00:00
 3 | 2008-02-01 00:00:00
 4 | 2008-02-04 00:00:00
 5 | 2008-02-04 00:00:00

SELECT m, (CAST('2008-01-01' As date) + n) As thedate
FROM generate_series(1, 58, 3) n, generate_series(1,5) m
WHERE n > m*2 AND m = 1
ORDER BY m,n;

--Yields


m |  thedate
--+------------
1 | 2008-01-05
1 | 2008-01-08
1 | 2008-01-11
1 | 2008-01-14
1 | 2008-01-17
1 | 2008-01-20
1 | 2008-01-23
1 | 2008-01-26
1 | 2008-01-29
1 | 2008-02-01
1 | 2008-02-04
1 | 2008-02-07
1 | 2008-02-10
1 | 2008-02-13
1 | 2008-02-16
1 | 2008-02-19
1 | 2008-02-22
1 | 2008-02-25
1 | 2008-02-28
```

Back to Table Of Contents  Build Median Aggregate Function in SQL Reader Comments

## More Aggregate Fun: Who's on First and Who's on Last *Intermediate*

Microsoft Access has these peculiar set of aggregates called **First** and **Last**. We try to avoid them because while the concept is useful, we find Microsoft Access's implementation of them a bit broken. MS Access power users we know moving over to something like MySQL, SQL Server, and PostgreSQL often ask - *where's first and where's last?* First we shall go over what exactly these aggregates do in MS Access and how they are different from MIN and MAX and what they should do in an ideal world. Then we shall create our ideal world in PostgreSQL.

### Why care who's on First and who's on Last?

This may come as a shock to quite a few DBAs, but there are certain scenarios in life where you want to ask for say an Average, Max, Min, Count etc and you also want the system to give you the First or last record of the group (this could be based on physical order or some designated order you ascribe). Even more shocking to DB Programmer type people who live very orderly lives and dream of predictability where there is none, some people don't care which record of the group is returned, just as long as all the fields returned are for a specific record. Not Care, You ask?

Here is a somewhat realistic scenario. Lets say you want to generate a mailing, but you have a ton of people on your list and you only want to send to one person in each company where the number of employees in the company is greater than 100. The boss doesn't care whether that person is *Doug Smith* or *John MacDonald*, but if you start making people up such as a person called *Doug MacDonald*, that is a reason for some concern. So your mandate is clear - *Save money on stamps, Inventing people is not cool, DO NOT INVENT ANYONE IN THE PROCESS*. So you see why MIN and MAX just does not work in this scenario. Yah Yah you say, I'm a top notch database programmer, I can do this in a hard to read but efficient SQL statement, that is portable across all databases. Good for you.

With First or Last function, your query would look like this:

```
SELECT First(LastName) As LName, First(FirstName) As FName, COUNT(EmployeeID) As numEmployees
FROM CompanyRoster
GROUP BY CompanyID
HAVING COUNT(EmployeeID) > 100;
```

The above is all fine and dandy and MS Access will help you nicely. What if you care about order though? This is where Access fails you because even if you do something like below in hopes of sending to the oldest person in the company, Access will completely ignore your attempts at sorting and return to you the first person entered for that company. This is where we will improve on Access's less than ideal implementation of First and Last.

```
SELECT First(LastName) As LName, First(FirstName) As FName, COUNT(EmployeeID) As numEmployees
FROM (SELECT * FROM
        CompanyRoster
        ORDER BY CompanyID, BirthDate DESC) As foo
GROUP BY CompanyID
HAVING COUNT(EmployeeID) > 100;
```

### Creating our First and Last Aggregates

Creating a First and Last Aggregate is much simpler than our Median function example. The First aggregate will simply look at the first entry that comes to it and ignore all the others. The Last aggregate will continually replace its current entry with whatever new entry is passed to it. The last aggregate is very trivial. The first aggregate is a bit more complicated because we don't want to throw out true nulls, but since our initial state is null, we want to ignore our initial state as well.

This time we shall also use Tom Lane's suggestion from our median post of using anyelement to make this work for all data types.

```
CREATE OR REPLACE FUNCTION first_element_state(anyarray, anyelement)
  RETURNS anyarray AS
$$
    SELECT CASE WHEN array_upper($1,1) IS NULL THEN array_append($1,$2) ELSE $1 END;
$$
  LANGUAGE 'sql' IMMUTABLE;

CREATE OR REPLACE FUNCTION first_element(anyarray)
  RETURNS anyelement AS
$$
    SELECT ($1)[1] ;
$$
  LANGUAGE 'sql' IMMUTABLE;

CREATE OR REPLACE FUNCTION last_element(anyelement, anyelement)
  RETURNS anyelement AS
$$
    SELECT $2;
$$
  LANGUAGE 'sql' IMMUTABLE;

CREATE AGGREGATE first(anyelement) (
  SFUNC=first_element_state,
  STYPE=anyarray,
  FINALFUNC=first_element
  )
```

```
;

CREATE AGGREGATE last(anyelement) (
  SFUNC=last_element,
  STYPE=anyelement
);
--Now some sample tests
--pick the first and last member from each family arbitrary by order of input
SELECT max(age) As oldest_age, min(age) As youngest_age, count(*) As numinfamily, family,
    first(name) As firstperson, last(name) as lastperson
FROM (SELECT 2 As age , 'jimmy' As name, 'jones' As family
    UNION ALL SELECT 50 As age, 'c' As name , 'jones' As family
    UNION ALL SELECT 3 As age, 'aby' As name, 'jones' As family
    UNION ALL SELECT 35 As age, 'Bartholemu' As name, 'Smith' As family
    ) As foo
GROUP BY family;
--Result
 oldest_age | youngest_age | numinfamily | family | firstperson | lastperson
------------+--------------+-------------+--------+-------------+-----------
         50 |            2 |           3 | jones  | jimmy       | aby
         35 |           35 |           1 | Smith  | Bartholemu  | Bartholemu


--For each family group list count of members,
--oldest and youngest age, and name of oldest and youngest family members
SELECT max(age) As oldest_age, min(age) As youngest_age, count(*) As numinfamily, family,
    first(name) As youngest_name, last(name) as oldest_name
FROM (SELECT * FROM (SELECT 2 As age , 'jimmy' As name, 'jones' As family
    UNION ALL SELECT 50 As age, 'c' As name , 'jones' As family
    UNION ALL SELECT 3 As age, 'aby' As name, 'jones' As family
    UNION ALL SELECT 35 As age, 'Bartholemu' As name, 'Smith' As family
    ) As foo ORDER BY family, age) as foo2
    WHERE age is not null
GROUP BY family;

--Result
 oldest_age | youngest_age | numinfamily | family | youngest_name | oldest_name
------------+--------------+-------------+--------+---------------+-------------
         35 |           35 |           1 | Smith  | Bartholemu    | Bartholemu
         50 |            2 |           3 | jones  | jimmy         | c
```

Back to Table Of Contents  More Aggregate Fun: Who's on First and Who's on Last Reader Comments

## OpenJump for PostGIS Spatial Ad-Hoc Queries *Beginner*

OpenJump is a Java Based, Cross-Platform open source GIS analysis and query tool. We've been using it a lot lately, and I would say out of all the open source tools (and even compared to many commercial tools) for geospatial analysis, it is one of the best out there.

While it is fairly rich in functionality in terms of doing statistical analysis on ESRI shapefile as well as PostGIS and other formats and also has numerous geometry manipulation features and plugins in its tool belt, we like the ad-hoc query ability the most. The ease and simplicity of that one tool makes it stand out from the pack. People not comfortable with SQL may not appreciate that feature as much as we do though.

In this excerpt we will quickly go thru the history of project and the ties between the PostGIS group and OpenJump group, how to install, setup a connection to a PostGIS enabled PostgreSQL database and doing some ad-hoc queries.

### Quick History Lesson

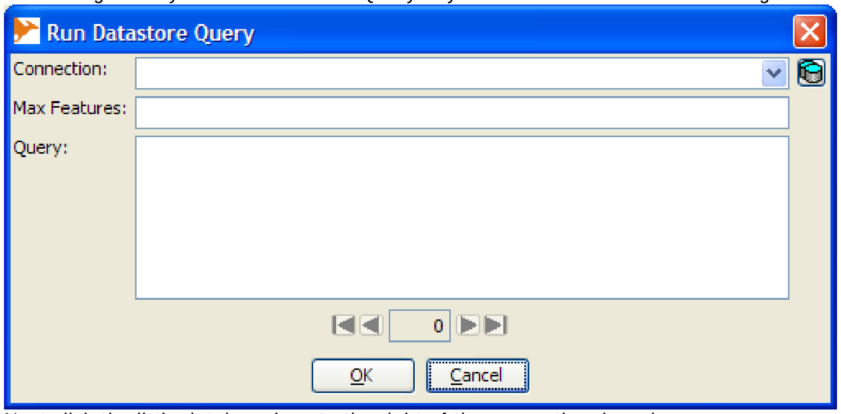
1. OpenJump is descended from Java Unified Mapping Platform - JUMP which was incubated by Vivid Solutions.
2. OpenJump and the whole JUMP family tree have Java Topology Suite (JTS) as a core foundation of their functionality.
3. GEOS which is a core foundation of PostGIS functionality and numerous other projects, is a C++ port of JTS. New Enhancements often are created in JTS and ported to GEOS and a large body of GEOS work has been incubated by Refractions Research, the PostGIS incubation company.
4. For more gory details about how all these things are intertwined, check out Martin Davis' recount of the history of GEOS and JTS.

### Installing

1. Install JRE 1.5 or above if you don't have it already.
2. You can then choose either the stable release version from http://sourceforge.net/project/showfiles.php?group_id=118054 or go with a nightly snapshot build. We tend to go with the nightly snapshot since there have been a lot of speed enhancements made that are not in the current production release.
3. For snapshot releases, no install is necessary - you can simply extract the zip and launch the openjump.bat (for Windows) or openjump.sh for Linux/ Unix based to launch the program. The production release includes an installer for windows.
4. Note - OpenJump uses a Plug-In architecture. For our particular exercises, you won't need any plug-ins not in Core. Many plugins are not included in the core, so to get those download them from http://sourceforge.net/project/showfiles.php?group_id=118054. Details on how to install plugins is http://openjump.org/wiki/show/Installing+PlugIns Installing PlugIns
5. For those who don't know anything about PostGIS and have no clue how to load spatial data into PostgreSQL, please check out our Almost Idiot's Guide to PostGIS that demonstrates quickly installing and loading using Mass Town data as an example. Also check out our pgsql2shp and shp2pgsql cheat sheet for dumping and loading spatial data from PostgreSQL.

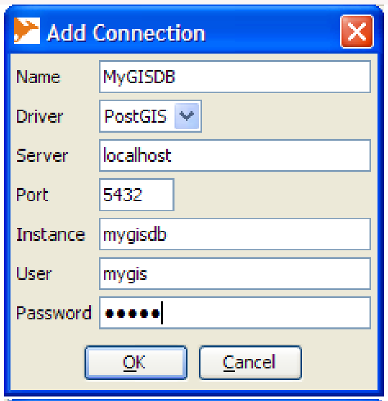
### Connecting to a PostGIS enabled PostgreSQL database

1. Launch the bin/openjump.bat (for windows), bin/openjump.sh (for Linux)
2. On Menu go to Layer->Run Datastore Query -- your screen should look something like this

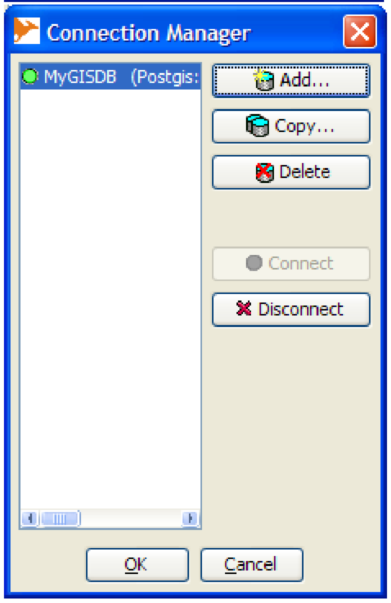


3. Next click the little database icon to the right of the connection drop down
4. Click "Add"
5. Fill in connection info and then click okay
   - **Name** field can have any name you want to give the connection
   - **Driver** PostGIS
   - **Server** - the hostname of the PostgreSQL Server
   - **Port** 5432 (or if you have a non-standard port whatever that is)
   - **Instance** This is the name of the database you want to connect to.
   - **User, Password** - The username and password of the PostgreSQL user you want to connect as.

   When you are done, your screen should look something like this.


Add Connection

Name: MyGISDB
Driver: PostGIS
Server: localhost
Port: 5432
Instance: mygisdb
User: mygis
Password: •••••

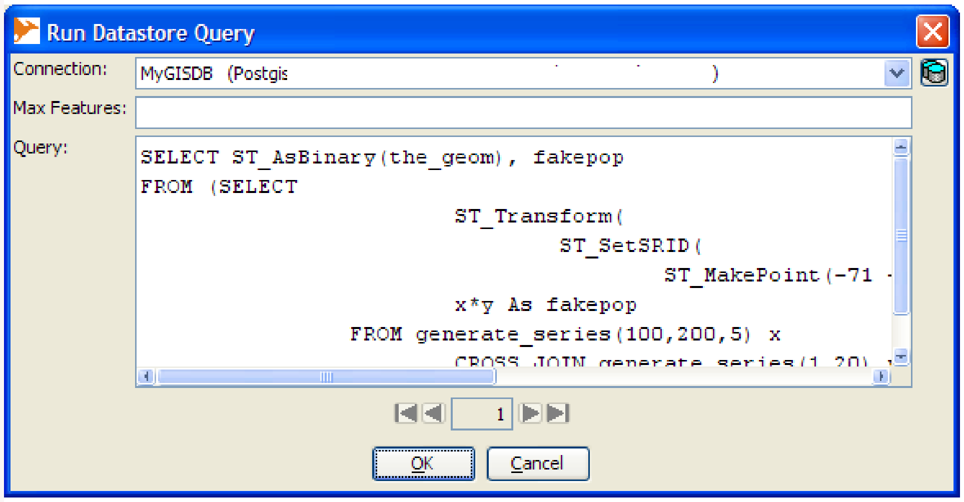

Connection Manager

MyGISDB (Postgis:

**Writing spatial queries and viewing them**

Now we shall create a very trivial query that magically needs no real data. Simply select the connection you just made, and in the query box type the following:

```
SELECT ST_AsBinary(the_geom), fakepop
FROM (SELECT
        ST_Transform(
           ST_SetSRID(
              ST_MakePoint(-71 + x*random()*0.01,42 + y*0.1*random()),4326),2163) As the_geom,
        x*y As fakepop
     FROM generate_series(100,200,5) x
        CROSS JOIN generate_series(1,20) y
    ) As foo;
```


Run Datastore Query

Connection: MyGISDB (Postgis )
Max Features:
Query:

```
SELECT ST_AsBinary(the_geom), fakepop
FROM (SELECT
                        ST_Transform(
                                ST_SetSRID(
                                        ST_MakePoint(-71 -
                        x*y As fakepop
                FROM generate_series(100,200,5) x
                        CROSS JOIN generate_series(1,20) y
```

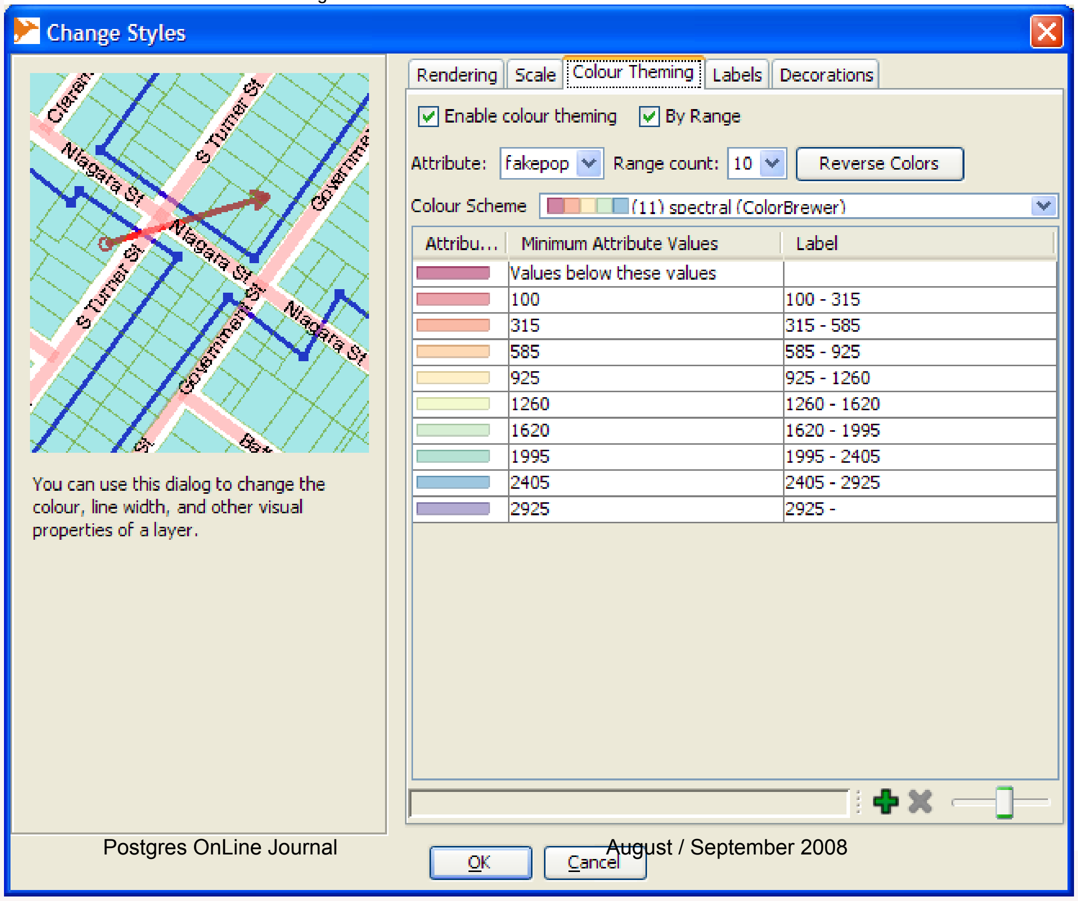
Click okay and when you are done, you should have a breath-taking map that looks something like this and will change each time you run the query:

**NOTE:** There are options on the file menu that will allow you to save the view as PNG, JPG or SVG.

Now if we want to change the colors of the points based on our fakepop, we do this.

1. Select the Layer
2. Right mouse-click and select **Change Styles**
3. Switch to Color Theming tab and click to Enable Color Theming and also by Range.

Your screen should look something like this:



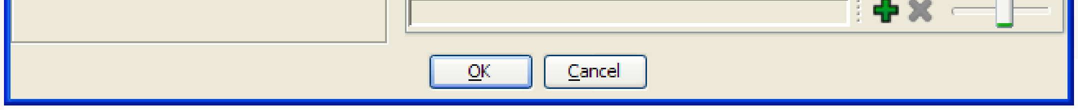
You can use this dialog to change the colour, line width, and other visual properties of a layer.

☑ Enable colour theming ☑ By Range

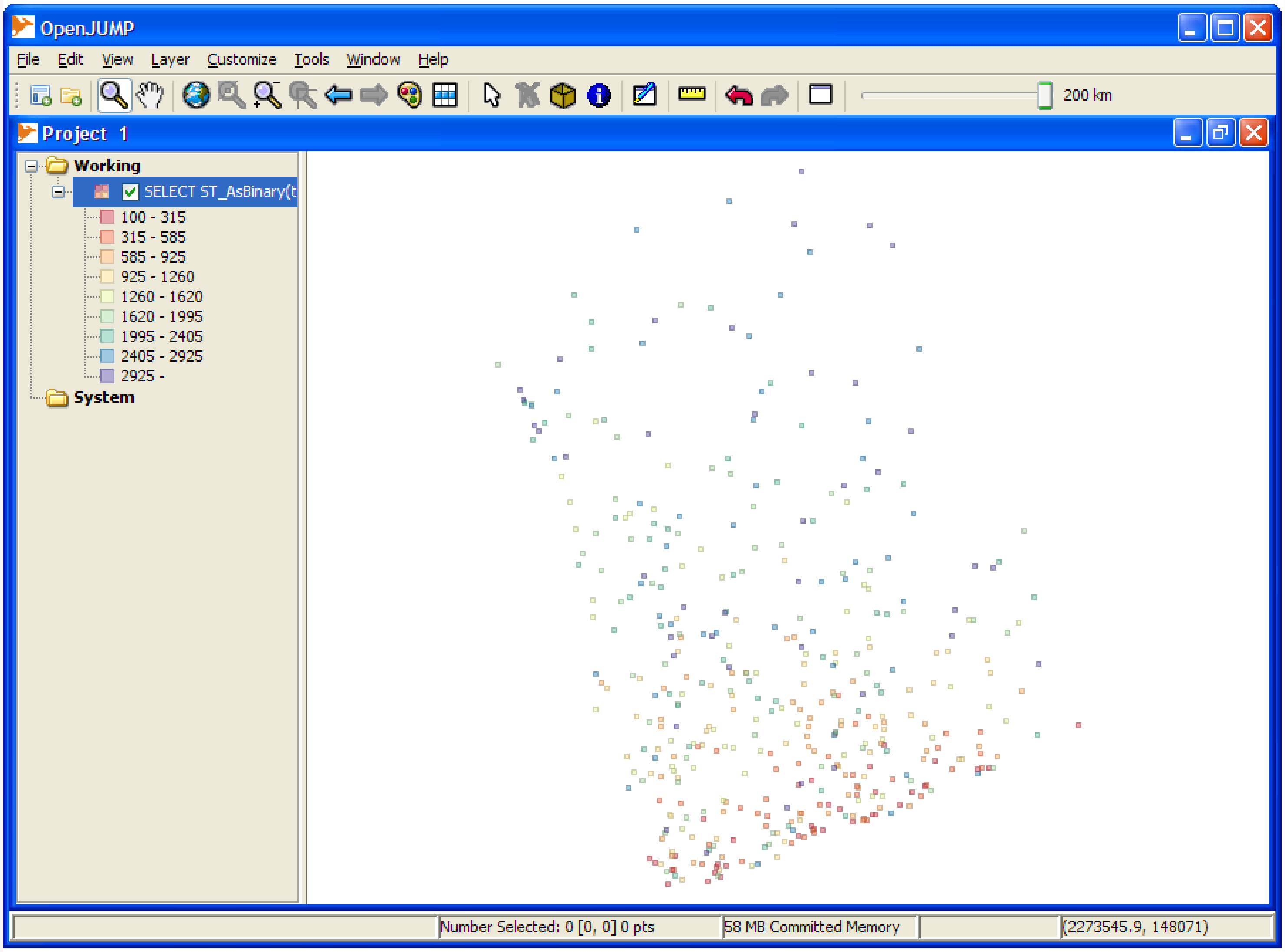
Attribute: fakepop Range count: 10

Colour Scheme (11) spectral (ColorBrewer)

| Attribu... | Minimum Attribute Values | Label |
|---|---|---|
| 🟥 | Values below these values | |
| 🟥 | 100 | 100 - 315 |
| 🟧 | 315 | 315 - 585 |
| 🟧 | 585 | 585 - 925 |
| 🟨 | 925 | 925 - 1260 |
| 🟨 | 1260 | 1260 - 1620 |
| 🟩 | 1620 | 1620 - 1995 |
| 🟩 | 1995 | 1995 - 2405 |
| 🟦 | 2405 | 2405 - 2925 |
| 🟪 | 2925 | 2925 - |



And the result of your hard work should look something like this



| | |
|---|---|
| 🟥 | 100 - 315 |
| 🟧 | 315 - 585 |
| 🟧 | 585 - 925 |
| 🟨 | 925 - 1260 |
| 🟨 | 1260 - 1620 |
| 🟩 | 1620 - 1995 |
| 🟩 | 1995 - 2405 |
| 🟦 | 2405 - 2925 |
| 🟪 | 2925 - |

Number Selected: 0 [0, 0] 0 pts | 58 MB Committed Memory | (2273545.9, 148071)

Back to Table Of Contents

## Reader Comments

### Fibonacci, Graphs and Recursive Queries

***Pythian Group Blog***
It’s time again for another edition of the weekly review of database blogs, Log Buffer. Since it was a big week for SQL Server, let’s start there, shall we? The big news — SQL Server 2008 is released, as reported by SqlServer-qa.net,...

***Jorge Gustavo Rocha***
Hi!

After reading your post, I've checked the Postgresql feature matrix, and I found nothing related with recursive queries.

http://www.postgresql.org/about/featurematrix

You know more details about the support for recursive queries? Is it really included in the 8.4 version?

Regards,

Jorge

***Regina***
Jorge,

Yes its planned for 8.4 and if you look on the hackers list a lot of that work is in the 8.4 dev source and there is a lot of talk back and forth on the topic and what parts of the spec will make it.

Below is a good recent thread chuck full of some test case examples.

http://archives.postgresql.org/pgsql-hackers/2008-08/msg00708.php

Hope that helps,
Regina

---

### How to determine if text phrase exists in a table column

***Caleb Cushing (xenoterracide)***
does this take into account that the email address might be in someone elses data?

say you were storing all emails in pg. john wants all references to him deleted but jack and john have been talking. you can just delete all emails that jack has that has john's email in it headers.

***Robert Treat***
What it really doesn't take into account is that in many countries it is now illegal to delete any emails that have passed through your system within the last 5-10 years. (Yeah, that's a business issue, not so much a code issues).

One solution we developed at OmniTI was to use plperl and some perl modules to ingest, parse, and then store a breakdown of all email messages. One example portion of that search's all parts of the email (envelope, headers, and body) to stores any email addresses found in a table linked back to the original message. Hmm... I wonder if that code is hiding on labs...

***Regina***
Caleb,
Actually this script just returns the table field names that the email address appears in your database (or search phrase appears in) and will limit the search to only the pattern of tables, schemas, fields you specify.

It also by the way doesn't do a case insensitive search, so would miss JOHN@hotmail. That would be easy enough to change by doing an ILIKE or upper, lower check or a regular expression check at a potentially significant speed penalty.

As Robert said it is illegal to delete emails in many countries and it gets even more messy as each government agency/company has thier own rules too.

I guess the main point of this exercise for us - was to say are we culpable (e.g. is this person simply blaming us because someone is using our address to spam people which annoyingly happens a lot and a lot of mail servers ignore SPF rules).

If we find him in our system then we are likely to blame, but if not - we should investigate further or kindly tell him he is wrong.

***David Fetter***
I know it's not very databasey, but why not just grep the output of pg_dump, if you're trying to find whether something exists in your database?

***Regina***
David,
You would say that wouldn't you? Its a very fetterish thing to say :).

I think people are all caught up in the stupid example I provided. The reason I wouldn't grep my pg_dump is because
1) I only have grep on my linux box not my windows box.
2) I'm not a grep hacker and I only want to search certain specific fields in my database. For example I may not care to check bodies of messages and only check short fields we use to spam people with.

3) Not really using this for email that much. I just thought that was an example people would be more likely to relate to than my real sinister intensions :).

---

### Build Median Aggregate Function in SQL

***Tom Lane***
Uh, you don't need multiple implementations if you use polymorphism.

```
CREATE OR REPLACE FUNCTION array_median(anyarray)
RETURNS anyelement AS
$$
SELECT CASE WHEN array_upper($1,1) = 0 THEN null ELSE asorted[ceiling(array_upper(asorted,1)/2.0)] END
FROM (SELECT ARRAY(SELECT ($1)[n] FROM
generate_series(1, array_upper($1, 1)) AS n
WHERE ($1)[n] IS NOT NULL
ORDER BY ($1)[n]
) As asorted) As foo ;
$$
LANGUAGE 'sql' IMMUTABLE;

CREATE AGGREGATE median(anyelement) (
SFUNC=array_append,
STYPE=anyarray,
FINALFUNC=array_median
);
```

***Regina***
Thanks Tom. That does make it much easier.

***Pavel Stehule***
This is example, where aggregate function is relative slow.

```
postgres=# select median(a) from foo;
median
--------
503
(1 row)

Time: 90,089 ms

postgres=# select array_median(array(select a from foo order by 1));
array_median
--------------
503
(1 row)

Time: 30,101 ms
```

***Regina***
Pavel,

This is very interesting. I thought your example difference was because of the presorting, but evidentally the aggregation process is adding a lot of overhead as well. What could that be?

I assumed there was a benefit to presorting especially if you have an index on the presorted column, but strangely I am not seeing the benefit in my sample data.

I have an index on the table on the length column census2000tiger_arc. Looking at these 3 - I would have expected the sorted ones to perform better, but it doesn't by much if at all. Looking at the explain I realize its not using the length index I put in but instead the zip index. So I'm guessing presorting only helps if your data has an index on that field and that index is actually used.

```
--This one returns in 2907 ms
-- midlength = 134.34012, count = 19,084 (without count 2891 ms)
SELECT median(length) as midlength, count(zipl)
from census2000tiger_arc
where zipl between 2000 and 2109;

--This one returns in 2964 ms (without count 2953 ms)
--midlength = 134.34012, count = 19,084

SELECT median(length) as midlength, count(zipl)
from (SELECT * FROM census2000tiger_arc
where zipl between 2000 and 2109 ORDER BY length) as foo;

--Equivalent to yours - 1844 ms, result = 134.34012
select array_median(array(select length FROM census2000tiger_arc
where zipl between 2000 and 2109 ORDER BY length));
```

***Pavel Stehule***
Hello,

simply repeated calling array_append function is expensive - this function isn't optimalised for using as aggregate. Array(subselect) is much faster.

***Postgres OnLine Journal***
Microsoft Access has these peculiar set of aggregates called First and Last. We try to avoid them because while the concept is useful, we find Microsoft Access's implementation of them
a bit broken. MS Access power users we know moving over to somethin

***Mike***
There is an error with this algorithm, as it doesn't consider the odd/even rule in the calculation. Although not a major error, the results using this function are different than all other implementations of the median function (e.g., R, Excel, etc.).

When there is an even number of items, the median is the average of the inner two middle values (since there is no symmetry to provide a middle value). If there is an odd number of items, the median is the middle index (as applied above). The examples above have odd set counts, so they work as expected. However, even set count examples fail with a "left" bias.

Building on Tom's suggestion above, consider using the following (note that this only works for numeric-like inputs, or other types that can be divided by two):

```
CREATE OR REPLACE FUNCTION array_median(anyarray)
RETURNS anyelement AS
$$
SELECT CASE
WHEN array_upper($1,1) = 0 THEN null
WHEN mod(array_upper($1,1),2) = 1 THEN asorted[ceiling(array_upper(asorted,1)/2.0)]
```

```
ELSE ((asorted[ceiling(array_upper(asorted,1)/2.0)] + asorted[ceiling(array_upper(asorted,1)/2.0)+1])/2.0) END
FROM (SELECT ARRAY(SELECT ($1)[n] FROM
generate_series(1, array_upper($1, 1)) AS n
WHERE ($1)[n] IS NOT NULL
ORDER BY ($1)[n]
) As asorted) As foo ;
$$
LANGUAGE 'sql' IMMUTABLE;
```

***Regina***
Mike,

Thanks for the example. Yes I was thinking about providing something like that and ideally for this one, I think you would use numeric[], numeric instead of anyelement.

As I recall things working you can have both definitions of median working and the one that is the closest match to the function is the one that gets used. So if you define this - then it would get used for numerics and the other would get used for everything else. I'll have to try that though since I'm not absolutely sure that's how it works.

---

## More Aggregate Fun: Who's on First and Who's on Last

***Mike***
This aggregate really helpful, and is exactly what I needed to find "hot spot" records.

However, it is also useful to know how many "first" matches have been found (using the examples above, "how many jones' have an age of two"; the example above will result with one).

I've been wracking my brain on this one (aggregate functions are relatively new to me), and I've come up with these functions to solve this issue:

```
CREATE OR REPLACE FUNCTION explode_array(anyarray)
RETURNS SETOF anyelement AS
$$
SELECT ($1)[s] from generate_series(1,array_upper($1, 1)) AS s;
$$
LANGUAGE 'sql' IMMUTABLE
ROWS 1000;

CREATE OR REPLACE FUNCTION count_first_element(anyarray)
RETURNS integer AS
$$
SELECT COUNT(*)::integer FROM explode_array($1) AS e WHERE e=$1[1];
$$
LANGUAGE 'sql' IMMUTABLE;

CREATE AGGREGATE count_first(anyelement) (
SFUNC=array_append,
STYPE=anyarray,
FINALFUNC=count_first_element
);

--and a similar (but modified) test:

SELECT max(age) As oldest_age, min(age) As youngest_age, count_first(age) As youngest_count, count(*) As numinfamily, family,
first(name) As youngest_name, last(name) as oldest_name
FROM (SELECT * FROM (SELECT 2 As age , 'jimmy' As name, 'jones' As family
UNION ALL SELECT 2 As age, 'c' As name , 'jones' As family
UNION ALL SELECT 3 As age, 'aby' As name, 'jones' As family
UNION ALL SELECT 35 As age, 'Bartholemu' As name, 'Smith' As family
) As foo ORDER BY age, family, name) as foo2
WHERE age is not null
GROUP BY family;
```

I haven't prepared a similar "count_last" aggregate, nor have I thoroughly tested this function. Also, I don't think that I handled NULL records gracefully as first_element_state above.

------------------

Also of importance is that the ORDER BY needs to have the first/last index before anything else (i.e., age needs to appear first). Failure to sort on this first will yield errors (e.g., try "ORDER BY family, name, age" which will incorrectly place aby as the youngest). The above example should have sorted in this order to reinforce this point.

---