

[Store](#) [About the Authors](#) [Consulting](#) [PostgreSQL](#) [PostGIS](#)

Thursday, december 27, 2007

Quicksearch

Postgres Online Journal

[The XML 1.0 Standard](#)
W3C XML Working Gr...
New \$12.95
Best \$12.95

[SQL Pocket Guide](#)
Jonathan Gennick
New \$10.19
Best \$5.81

[PostGIS in Action](#)
Regina Obe, Leo Hs...
New \$33.74
Best \$33.74

[Privacy Information](#)

CROSSTAB QUERIES IN POSTGRESQL USING TABLEFUNC CONTRIB

The generic way of doing cross tabs (sometimes called PIVOT queries) in an ANSI-SQL database such as PostgreSQL is to use CASE statements which we have documented in the article [What is a crosstab query and how do you create one using a relational database?](#).

In this particular issue, we will introduce creating crosstab queries using PostgreSQL **tablefunc** contrib.

Installing Tablefunc

Tablefunc is a contrib that comes packaged with all PostgreSQL installations - we believe from versions 7.4.1 up (possibly earlier). We will be assuming the one that comes with 8.2 for this exercise. Note in prior versions, tablefunc was not documented in the standard postgresql docs, but the new 8.3 seems to have it documented at <http://www.postgresql.org/docs/8.3/static/tablefunc.html>.

Often when you create crosstab queries, you do it in conjunction with GROUP BY and so forth. While the astute reader may conclude this from the docs, none of the examples in the docs specifically demonstrate that and the more useful example of **crosstab(source_sql,category_sql)** is left till the end of the documentation.

To install tablefunc simply open up the **share/contrib/tablefunc.sql** in pgadmin and run the sql file. Keep in mind that the functions are installed by default in the **public** schema. If you want to install in a different schema - change the first line that reads **SET search_path = public;**

Alternatively you can use **psql** to install tablefunc using something like the following command:

```
path\to\postgresql\bin\psql -h localhost -U someuser -d somedb -f "path\to\postgresql\share\contrib\tablefunc.sql"
```

We will be covering the following functions

1. **crosstab(source_sql, category_sql)**
2. **crosstab(source_sql)**
3. Tricking crosstab to give you more than one row header column
4. Building your own custom crosstab function similar to the crosstab3, crosstab4 etc. examples
5. Adding a total column to crosstab query

There are a couple of key points to keep in mind which apply to both crosstab functions.

1. Source SQL must always return 3 columns, first being what to use for row header, second the bucket slot, and third is the value to put in the bucket.
2. crosstab except for the example crosstab3 ..crosstabN versions return unknown record types. This means that in order to use them in a FROM clause, you need to either alias them by specifying the result type or create a custom crosstab that outputs a known type as demonstrated by the crosstabN flavors. Otherwise you get the common *a column definition list is required for functions returning "record" error*.
3. A corollary to the previous statement, it is best to cast those 3 columns to specific data types so you can be guaranteed the datatype that is returned so it doesn't fail your row type casting.
4. Each row should be unique for row header, bucket otherwise you get unpredictable results

Setting up our test data

For our test data, we will be using our familiar inventory, inventory flow example. Code to generate structure and test data is shown below.

```
CREATE TABLE inventory
(
    item_id serial NOT NULL,
    item_name varchar(100) NOT NULL,
    CONSTRAINT pk_inventory PRIMARY KEY (item_id),
    CONSTRAINT inventory_item_name_idx UNIQUE (item_name)
)
WITH (OIDS=FALSE);

CREATE TABLE inventory_flow
(
    inventory_flow_id serial NOT NULL,
    item_id integer NOT NULL,
    project varchar(100),
    num_used integer,
    num_ordered integer,
    action_date timestamp without time zone
        NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT pk_inventory_flow PRIMARY KEY (inventory_flow_id),
```

Calendar

July '10

| | | | | | |
|-----|-----|-----|-----|-----|-----|
| Mon | Tue | Wed | Thu | Fri | Sat |
| | | | 1 | 2 | 3 |
| 5 | 6 | 7 | 8 | 9 | 10 |
| 12 | 13 | 14 | 15 | 16 | 17 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 26 | 27 | 28 | 29 | 30 | 31 |

Categories

| | |
|--------------------------|------------------------------|
| <input type="checkbox"/> | adobe flex (2) |
| <input type="checkbox"/> | adodb (2) |
| <input type="checkbox"/> | advanced (6) |
| <input type="checkbox"/> | application development (12) |
| <input type="checkbox"/> | webservices (6) |
| <input type="checkbox"/> | basics (34) |
| <input type="checkbox"/> | yum (5) |
| <input type="checkbox"/> | beginner (44) |
| <input type="checkbox"/> | cheatsheet (5) |
| <input type="checkbox"/> | contrib spotlight (9) |
| <input type="checkbox"/> | fuzzystmatch (|
| <input type="checkbox"/> | pgcrypto (1) |
| <input type="checkbox"/> | tablefunc (1) |
| <input type="checkbox"/> | tsearch (6) |
| <input type="checkbox"/> | cte (3) |
| <input type="checkbox"/> | demo dbs (1) |
| <input type="checkbox"/> | pagila (9) |
| <input type="checkbox"/> | usda (3) |
| <input type="checkbox"/> | editor note (22) |
| <input type="checkbox"/> | gis (19) |
| <input type="checkbox"/> | intermediate (55) |
| <input type="checkbox"/> | joke (2) |
| <input type="checkbox"/> | mono .NET (3) |
| <input type="checkbox"/> | new in postgres (11) |
| <input type="checkbox"/> | oobase (6) |
| <input type="checkbox"/> | other dbms (3) |
| <input type="checkbox"/> | db2 (11) |
| <input type="checkbox"/> | Dbase (2) |
| <input type="checkbox"/> | firebird (6) |
| <input type="checkbox"/> | informix (3) |
| <input type="checkbox"/> | ms access (10) |
| <input type="checkbox"/> | mysql (31) |
| <input type="checkbox"/> | netezza (1) |
| <input type="checkbox"/> | oracle (25) |
| <input type="checkbox"/> | sql server (42) |
| <input type="checkbox"/> | sqlite (3) |
| <input type="checkbox"/> | teradata (1) |
| <input type="checkbox"/> | pgadmin (13) |
| <input type="checkbox"/> | pgagent (2) |

```

CONSTRAINT fk_item_id FOREIGN KEY (item_id)
REFERENCES inventory (item_id)
ON UPDATE CASCADE ON DELETE RESTRICT
)
WITH (OIDS=FALSE);

CREATE INDEX inventory_flow_action_date_idx
ON inventory_flow
USING btree
(action_date)
WITH (FILLFACTOR=95);

INSERT INTO inventory(item_name) VALUES('CSCL (g)');
INSERT INTO inventory(item_name) VALUES('DNA Ligase (ul)');
INSERT INTO inventory(item_name) VALUES('Phenol (ul)');
INSERT INTO inventory(item_name) VALUES('Pipette Tip 10ul');

INSERT INTO inventory_flow(item_id, project, num_ordered, action_date)
SELECT i.item_id, 'Initial Order', 10000, '2007-01-01'
FROM inventory i;

--Simulate usage
INSERT INTO inventory_flow(item_id, project, num_used, action_date)
SELECT i.item_id, 'MS', n*2,
'2007-03-01'::timestamp + (n || ' day')::interval + ((n + 1) || ' hour')::interval
FROM inventory As i CROSS JOIN generate_series(1, 250) As n
WHERE mod(n + 42, i.item_id) = 0;

INSERT INTO inventory_flow(item_id, project, num_used, action_date)
SELECT i.item_id, 'Alzheimer's', n*1,
'2007-02-26'::timestamp + (n || ' day')::interval + ((n + 1) || ' hour')::interval
FROM inventory as i CROSS JOIN generate_series(50, 100) As n
WHERE mod(n + 50, i.item_id) = 0;

INSERT INTO inventory_flow(item_id, project, num_used, action_date)
SELECT i.item_id, 'Mad Cow', n*i.item_id,
'2007-02-26'::timestamp + (n || ' day')::interval + ((n + 1) || ' hour')::interval
FROM inventory as i CROSS JOIN generate_series(50, 200) As n
WHERE mod(n + 7, i.item_id) = 0 AND i.item_name IN('Pipette Tip 10ul', 'CSCL (g)');

vacuum analyze;

```

Using crosstab(source_sql, category_sql)

For this example we want to show the monthly usage of each inventory item for the year 2007 regardless of project. The crosstab we wish to achieve would have columns as follows: **item_name, jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec**

```

--Standard group by aggregate query before we pivot to cross tab
--This we use for our source sql
SELECT i.item_name::text As row_name, to_char(if.action_date, 'mon')::text As bucket,
SUM(if.num_used)::integer As bucketvalue
FROM inventory As i INNER JOIN inventory_flow As if
ON i.item_id = if.item_id
WHERE (if.num_used <> 0 AND if.num_used IS NOT NULL)
AND action_date BETWEEN date '2007-01-01' and date '2007-12-31 23:59'
GROUP BY i.item_name, to_char(if.action_date, 'mon'), date_part('month', if.action_date)
ORDER BY i.item_name, date_part('month', if.action_date);

```

```

--Helper query to generate lowercase month names - this we will use for our category sql
SELECT to_char(date '2007-01-01' + (n || ' month')::interval, 'mon') As short_mname
FROM generate_series(0,11) n;

```

--Resulting crosstab query --**Note: For this we don't need the order by month since the order of the columns is determined by the category_sql row order**

```

SELECT mthreport.*
FROM
crosstab('SELECT i.item_name::text As row_name, to_char(if.action_date, 'mon')::text As bucket,
SUM(if.num_used)::integer As bucketvalue
FROM inventory As i INNER JOIN inventory_flow As if
ON i.item_id = if.item_id
AND action_date BETWEEN date ''2007-01-01'' and date ''2007-12-31 23:59''
GROUP BY i.item_name, to_char(if.action_date, 'mon'), date_part('month', if.action_date)
ORDER BY i.item_name',
'SELECT to_char(date ''2007-01-01'' + (n || ' month')::interval, 'mon') As short_mname
FROM generate_series(0,11) n')
As mthreport(item_name text, jan integer, feb integer, mar integer,
apr integer, may integer, jun integer, jul integer,
aug integer, sep integer, oct integer, nov integer,

```

☐ [SQL](#) pl programming (18)
☐ [SQL](#) plperl (1)
☐ [SQL](#) plpgsql (10)
☐ [SQL](#) plpython (6)
☐ [SQL](#) PLR (3)
☐ [SQL](#) sql functions (1)
☐ [SQL](#) postgres (39)
☐ [SQL](#) postgresql versions (14)
☐ [SQL](#) 8.2 (20)
☐ [SQL](#) 8.3 (31)
☐ [SQL](#) 8.4 (41)
☐ [SQL](#) 9.0 (9)
☐ [SQL](#) product showcas (17)
☐ [SQL](#) q&a (34)
☐ [SQL](#) serendipity (1)
☐ [SQL](#) smarty (1)
☐ [SQL](#) special feature (5)
☐ [SQL](#) window function (5)

Go!

All categories

Archives

☐ July 2010
☐ June 2010
☐ May 2010
☐ Recent...
☐ Older...

Syndicate This Blog

☐ [SQL](#) RSS 0.91 feed
☐ [SQL](#) RSS 1.0 feed
☐ [SQL](#) RSS 2.0 feed
☐ [SQL](#) ATOM 0.3 feed
☐ [SQL](#) ATOM 1.0 feed
☐ [SQL](#) RSS 2.0 Comments
☐ [SQL](#) OPML 1.0 feed

Powered by

[serendipity](#)

Blog Administration

[Open login screen](#)

Entry's Links

☐ [Show tagged entries](#)

```
dec integer)
```

The output of the above crosstab looks as follows:

| item_name text | jan integer | feb integer | mar integer | apr integer | may integer | jun integer | jul integer | aug integer | sep integer | oct integer | nov integer | dec integer |
|-------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| CSCl (g) | | 870 | 3884 | 9000 | 9792 | 11691 | 14625 | 15068 | 13230 | 7290 | | |
| DNA Ligase (ul) | | | 420 | 1650 | 3360 | 3364 | 3960 | 4860 | 5348 | 6600 | 3888 | |
| Phenol (ul) | | | 270 | 1096 | 2245 | 2361 | 2376 | 3210 | 3810 | 4410 | 2430 | |
| Pipette Tip 10ul | | | 196 | 1550 | 3800 | 4560 | 6432 | 6888 | 6440 | 3520 | 1952 | |

Using crosstab(source_sql)

crosstab(source_sql) is much trickier to understand and use than the crosstab(source_sql, category_sql) variant, but in certain situations and certain cases is faster and just as effective. The reason why is that crosstab(source_sql) is not guaranteed to put same named buckets in the same columns especially for sparsely populated data. For example - lets say you have data for CSCl for Jan Mar Apr and data for Phenol for Apr. Then Phenols Apr bucket will be in the same column as CSCl Jan's bucket. This in most cases is not terribly useful and is confusing.

To skirt around this inconvenience one can write an SQL statement that guarantees you have a row for each permutation of Item, Month by doing a cross join. Below is the above written so item month usage fall in the appropriate buckets.

```
--Code to generate the row tally - before crosstab
SELECT i.item_name::text As row_name, i.start_date::date As bucket,
       SUM(if.num_used)::integer As bucketvalue
FROM (SELECT inventory.*,
       date '2007-01-01' + (n || ' month')::interval As start_date,
       date '2007-01-01' + ((n + 1) || ' month')::interval + - '1 minute'::interval As end_date
FROM inventory CROSS JOIN generate_series(0,11) n) As i
LEFT JOIN inventory_flow As if
ON (i.item_id = if.item_id AND if.action_date BETWEEN i.start_date AND i.end_date)
GROUP BY i.item_name, i.start_date
ORDER BY i.item_name, i.start_date;

--Now we feed the above into our crosstab query to achieve the same result as
--our crosstab(source, category) example
SELECT mthreport.*
FROM crosstab('SELECT i.item_name::text As row_name, i.start_date::date As bucket,
               SUM(if.num_used)::integer As bucketvalue
FROM (SELECT inventory.*,
               date ''2007-01-01'' + (n || ' month')::interval As start_date,
               date ''2007-01-01'' + ((n + 1) || ' month')::interval + - ''1 minute''::interval As end_date
FROM inventory CROSS JOIN generate_series(0,11) n) As i
               LEFT JOIN inventory_flow As if
               ON (i.item_id = if.item_id AND if.action_date BETWEEN i.start_date AND i.end_date)
GROUP BY i.item_name, i.start_date
ORDER BY i.item_name, i.start_date;'
As mthreport(item_name text, jan integer, feb integer,
             mar integer, apr integer,
             may integer, jun integer, jul integer, aug integer,
             sep integer, oct integer, nov integer, dec integer)
```

In actuality the above query if you have an index on action_date is probably more efficient for larger datasets than the crosstab(source, category) example since it utilizes a date range condition for each month match.

There are a couple of situations that come to mind where the standard behavior of crosstab of not putting like items in same column is useful. One example is when its not necessary to distinguish bucket names, but order of cell buckets is important such as when doing column rank reports. For example if you wanted to know for each item, which projects has it been used most in and you want the column order of projects to be based on highest usage. You would have simple labels like **item_name**, **project_rank_1**, **project_rank_2**, **project_rank_3** and the actual project names would be displayed in project_rank_1, project_rank_2, project_rank_3 columns.

```
SELECT projreport.*
FROM crosstab('SELECT i.item_name::text As row_name,
               if.project::text As bucket,
               if.project::text As bucketvalue
FROM inventory i
               LEFT JOIN inventory_flow As if
               ON (i.item_id = if.item_id)
               WHERE if.num_used > 0
GROUP BY i.item_name, if.project
ORDER BY i.item_name, SUM(if.num_used) DESC, if.project')
As projreport(item_name text, project_rank_1 text, project_rank_2 text,
              project_rank_3 text)
```

Output of the above looks like:

| item_name text | project_rank_1 text | project_rank_2 text | project_rank_3 text |
|-------------------|------------------------|------------------------|------------------------|
| CSCL (g) | MS | Mad Cow | Alzheimer's |
| DNA Ligase (ul) | MS | Alzheimer's | |
| Phenol (ul) | MS | Alzheimer's | |
| Pipette Tip 10ul | Mad Cow | MS | Alzheimer's |

Tricking crosstab to give you more than one row header column

Recall we said that crosstab requires exactly 3 columns output in the sql source statement. No more and No less. So what do you do when you want your month crosstab by Item, Project, and months columns. One approach is to stuff more than one Item in the item slot by either using a delimiter or using an Array. We shall show the array approach below.

```
SELECT mthreport.row_name[1] As project, mthreport.row_name[2] As item_name,
       jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
FROM
  crosstab('SELECT ARRAY[if.project::text, i.item_name::text] As row_name,
               to_char(if.action_date, 'mon')::text As bucket, SUM(if.num_used)::integer As bucketvalue
  FROM inventory As i INNER JOIN inventory_flow As if
    ON i.item_id = if.item_id
    AND action_date BETWEEN date '2007-01-01' and date '2007-12-31 23:59'
  WHERE if.num_used <> 0
  GROUP BY if.project, i.item_name, to_char(if.action_date, 'mon'),
           date_part('month', if.action_date)
  ORDER BY if.project, i.item_name',
  'SELECT to_char(date '2007-01-01' + (n || ' month')::interval, 'mon') As short_mname
  FROM generate_series(0,11) n')
  As mthreport(row_name text[], jan integer, feb integer, mar integer,
               apr integer, may integer, jun integer, jul integer,
               aug integer, sep integer, oct integer, nov integer,
               dec integer)
```

Result of the above looks as follows:

| project text | item_name text | jan integer | feb integer | mar integer | apr integer | may integer | jun integer | jul integer | aug integer | sep integer | oct integer | nov integer | dec integer |
|-----------------|-------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| Alzheimer's | CSCL (g) | | | | 666 | 2295 | 864 | | | | | | |
| Alzheimer's | DNA Ligase (ul) | | | | 330 | 1140 | 480 | | | | | | |
| Alzheimer's | Phenol (ul) | | | | 226 | 775 | 291 | | | | | | |
| Alzheimer's | Pipette Tip 10ul | | | | 162 | 608 | 192 | | | | | | |
| Mad Cow | CSCL (g) | | | | 666 | 2295 | 2954 | 4035 | 4935 | 3990 | | | |
| Mad Cow | Pipette Tip 10ul | | | | 684 | 2156 | 2940 | 4320 | 4620 | 3780 | | | |
| MS | CSCL (g) | | | 870 | 2552 | 4410 | 5974 | 7656 | 9690 | 11078 | 13230 | 7290 | |
| MS | DNA Ligase (ul) | | | 420 | 1320 | 2220 | 2884 | 3960 | 4860 | 5348 | 6600 | 3888 | |
| MS | Phenol (ul) | | | 270 | 870 | 1470 | 2070 | 2376 | 3210 | 3810 | 4410 | 2430 | |
| MS | Pipette Tip 10ul | | | 196 | 704 | 1036 | 1428 | 2112 | 2268 | 2660 | 3520 | 1952 | |

Building your own custom crosstab function

If month tabulations are something you do often, you will quickly become tired of writing out all the months. One way to get around this inconvenience - is to define a type and crosstab alias that returns the well-defined type something like below:

```
CREATE TYPE tablefunc_crosstab_month AS
  (row_name text[], jan integer, feb integer, mar integer,
   apr integer, may integer, jun integer, jul integer,
   aug integer, sep integer, oct integer, nov integer,
   dec integer);

CREATE OR REPLACE FUNCTION crosstabmonthint(text, text)
  RETURNS SETOF tablefunc_crosstab_month AS
  '$libdir/tablefunc', 'crosstab_hash'
  LANGUAGE 'c' STABLE STRICT;
```

Then you can write the above query as

```
SELECT mthreport.row_name[1] As project, mthreport.row_name[2] As item_name,
       jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
FROM
  crosstabmonthint('SELECT ARRAY[if.project::text, i.item_name::text] As row_name, to_char(if.action_date, 'mon')::text As bucket,
  SUM(if.num_used)::integer As bucketvalue
  FROM inventory As i INNER JOIN inventory_flow As if
    ON i.item_id = if.item_id
    AND action_date BETWEEN date '2007-01-01' and date '2007-12-31 23:59'
  WHERE if.num_used <> 0
  GROUP BY if.project, i.item_name, to_char(if.action_date, 'mon'), date_part('month', if.action_date)
  ORDER BY if.project, i.item_name',
  'SELECT to_char(date '2007-01-01' + (n || ' month')::interval, 'mon') As short_mname
  FROM generate_series(0,11) n')
  As mthreport;
```

Adding a Total column to the crosstab query

Adding a total column to a crosstab query using crosstab function is a bit tricky. Recall we said the source sql should have exactly 3 columns (row header, bucket, bucketvalue). Well that wasn't entirely accurate. The **crosstab(source_sql, category_sql)** variant of the function allows for a source that has columns **row_header**, **extraneous columns**, **bucket**, **bucketvalue**. Don't get extraneous columns confused with row headers. They are not the same and if you try to use it as we did for creating multi row columns, you will be leaving out data. For simplicity here is a fast rule to remember.
Extraneous column values must be exactly the same for source rows that have the same row header and they get inserted right before the bucket columns.

We shall use this fact to produce a total column.

```
--This we use for our source sql
SELECT i.item_name::text As row_name,
      (SELECT SUM(sif.num_used)
       FROM inventory_flow sif
        WHERE action_date BETWEEN date '2007-01-01' and date '2007-12-31 23:59'
          AND sif.item_id = i.item_id)::integer As total,
      to_char(if.action_date, 'mon')::text As bucket,
      SUM(if.num_used)::integer As bucketvalue
FROM inventory As i INNER JOIN inventory_flow As if
  ON i.item_id = if.item_id
WHERE (if.num_used <> 0 AND if.num_used IS NOT NULL)
  AND action_date BETWEEN date '2007-01-01' and date '2007-12-31 23:59'
GROUP BY i.item_name, total, to_char(if.action_date, 'mon'), date_part('month', if.action_date)
ORDER BY i.item_name, date_part('month', if.action_date);

--This we use for our category sql
SELECT to_char(date '2007-01-01' + (n || ' month')::interval, 'mon') As short_mname
FROM generate_series(0,11) n;

--Now our cross tabulation query
SELECT mthreport.*
FROM crosstab('SELECT i.item_name::text As row_name,
      (SELECT SUM(sif.num_used)
       FROM inventory_flow sif
        WHERE action_date BETWEEN date ''2007-01-01'' and date ''2007-12-31 23:59''
          AND sif.item_id = i.item_id)::integer As total,
      to_char(if.action_date, ''mon'')::text As bucket,
      SUM(if.num_used)::integer As bucketvalue
FROM inventory As i INNER JOIN inventory_flow As if
  ON i.item_id = if.item_id
WHERE (if.num_used <> 0 AND if.num_used IS NOT NULL)
  AND action_date BETWEEN date ''2007-01-01'' and date ''2007-12-31 23:59''
GROUP BY i.item_name, total, to_char(if.action_date, ''mon''), date_part(''month'', if.action_date)
ORDER BY i.item_name, date_part(''month'', if.action_date)',
'SELECT to_char(date ''2007-01-01'' + (n || ' month')::interval, ''mon'') As short_mname
FROM generate_series(0,11) n'
)
As mthreport(item_name text, total integer, jan integer, feb integer,
mar integer, apr integer,
may integer, jun integer, jul integer, aug integer,
sep integer, oct integer, nov integer, dec integer)
```

Resulting output of our cross tabulation with total column looks like this:

| item_name text | total integ | jan integ | feb integ | mar integ | apr integ | may integ | jun integ | jul integ | aug integ | sep integ | oct integ | nov integ | dec integ |
|-------------------|----------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| CsCl (g) | 85450 | | | 870 | 3884 | 9000 | 9792 | 11691 | 14625 | 15068 | 13230 | 7290 | |
| DNA Ligase (ul) | 33450 | | | 420 | 1650 | 3360 | 3364 | 3960 | 4860 | 5348 | 6600 | 3888 | |
| Phenol (ul) | 22208 | | | 270 | 1096 | 2245 | 2361 | 2376 | 3210 | 3810 | 4410 | 2430 | |
| Pipette Tip 10ul | 35338 | | | 196 | 1550 | 3800 | 4560 | 6432 | 6888 | 6440 | 3520 | 1952 | |

If per chance you wanted to have a total row as well you could do it with a union query in your source sql. Unfortunately PostgreSQL does not support windowing functions that would make the row total not require a union. We'll leave that one as an exercise to figure out.

Another not so obvious observation. You can define a type that say returns 20 bucket columns, but your actual crosstab need not return up to 20 buckets. It can return less and whatever buckets that are not specified will be left blank. With that in mind, you can create a generic type that returns generic names and then in your application code - set the heading based on the category source. Also if you have fewer buckets in your type definition than what is returned, the right most buckets are just left off. This allows you to do things like list the top 5 colors of a garment etc.

Posted by Leo Hsu and Regina Obe in contrib spotlight, intermediate, tablefunc at 03:57 | Comments (10) | Trackbacks (0)

30

TRACKBACKS

Trackback specific URI for this entry

No Trackbacks

**COMMENTS**

Display comments as ([Linear](#) | [Threaded](#))

This is nice, but as I see it always presumes that you know your data before you do the crosstab. Even if you specify your columns with an sql statement you still need to enumerate all resulting columns individually in the As mthreport(...) part. I have searched extensively but could not find a plpgsql based solution for the situation where you don't know what the categories will be. If you have any solution for that please let me know.

Thx.
SWK

#1 SunWuKung on 2008-01-09 07:52 ([Reply](#))

Good question. Sadly I don't think there is an easy answer. To get around the issue say in PHP and so forth, we use dummy names in our sql statement or a dummy type with lots of output columns and then in PHP to display the header, loop thru our category sql recordset.

The problem is not so much with crosstab as with PostgreSQL inability to deal with dynamic record types or ability to do record introspection. This has been discussed as a future enhancement for example here

<http://archives.postgresql.org/pgsql-patches/2005-07/msg00458.php>
But unfortunately haven't heard any recent talk of it.

#1.1 Regina on 2008-01-09 22:10 ([Reply](#))

I'm pretty new in RDBMS and in PostGreSQL, and I recently discovered crosstab utility so maybe I'm wrong but, as you say "I have searched extensively but could not find a plpgsql based solution for the situation where you don't know what the categories will be", did you have a look at

<http://www.ledscripts.com/tech/article/view/5.html> ?

I am also looking for a solution for dynamical categories...

Cheers,
--
Denis

#1.2 Denis Bitouzé on 2008-01-13 07:06 ([Reply](#))

Hi, can you help me please!

I need create sql that returns fils an no colums.
There is some function similar to PIVOT of ORACLE for EnterpriseDB?

For example I want a structrue this
paciente vacuna1 vacuna2 vacunaN
jhon 01/01/08 01/02/08 01/03/08
daniel 03/01/08 03/02/08 03/03/08

but sql returns this:
paciente vacuna1 vacuna2 vacunaN
jhon 01/01/08
jhon 01/02/08
jhon 01/03/08
daniel 03/01/08
daniel 03/02/08
daniel 03/03/08

I'm from El Salvador.
thank, wake up your help!

#1.2.1 sandra on 2008-08-12 18:06 ([Reply](#))

Hello,

I've been looking for such a "generic" crosstab for monthes. Does anyone know whether there were recent progress, on this point?

Cheers. And happy new year!
A+
Pierre

#1.2.2 Pierre Chevalier on 2010-01-18 09:05 ([Reply](#))

Maybe you could dynamically write the crosstab query in a plpgsql procedure.
You would first query your data to get the categories, write the record types string, then write the query itself ?

#2 Arnaud on 2008-07-10 06:38 (Reply)

Thank you for the great howto to crosstab functionality. However, I am not able to follow your last hint about having a type with more buckets than the crosstab function returns.
For example, I create the following type and method:

```
CREATE TYPE tablefunc_crosstab_year AS
(row_name text, jan integer, feb integer, mar integer,
apr integer, may integer, jun integer, jul integer,
aug integer, sep integer, oct integer, nov integer,
dec integer, jan1 integer);

CREATE OR REPLACE FUNCTION crosstabyear(text, text)
RETURNS SETOF tablefunc_crosstab_year AS
'$libdir/tablefunc', 'crosstab_hash'
LANGUAGE 'c' STABLE STRICT;
```

and when I call the function like this

```
SELECT mthreport.*
FROM
crosstabyear('SELECT if.project::text As row_name, to_char(if.action_date, "mon")::text As bucket,
SUM(if.num_used)::integer As bucketvalue
FROM inventory As i INNER JOIN inventory_flow As if
ON i.item_id = if.item_id
AND action_date BETWEEN date "2007-01-01" and date "2007-12-31 23:59"
WHERE if.num_used = 0
GROUP BY if.project, to_char(if.action_date, "mon"), date_part("month", if.action_date)
ORDER BY if.project',
'SELECT to_char(date "2007-01-01" + (n || " month")::interval, "mon") As short_mname
FROM generate_series(0,11) n')
As mthreport;
```

I get the error:
ERROR: invalid return type
DETAIL: Query-specified return tuple has 14 columns but crosstab returns 13.

Could you give me some pointer what I'm doing wrong here?

Thanks a lot,
Andreas

#3 Andreas on 2009-04-08 09:36 (Reply)

Andreas,

Could be a postgresql version issue. Which version of PostgreSQL are you using?

Just tried your example and it worked for me under 8.3 and a beta version of 8.4. Don't have 8.2 and lower lying around. The only thing is that your query seems to be missing an = sign -- but that would have generated a different error or got stripped for some reason by the commenter.

So the query I tried is

```
SELECT mthreport.*
FROM
crosstabyear('SELECT if.project::text As row_name, to_char(if.action_date, "mon")::text As bucket,
SUM(if.num_used)::integer As bucketvalue
FROM inventory As i INNER JOIN inventory_flow As if
ON i.item_id = if.item_id
AND action_date BETWEEN date "2007-01-01" and date "2007-12-31 23:59"
WHERE if.num_used = 0
GROUP BY if.project, to_char(if.action_date, "mon"), date_part("month", if.action_date)
ORDER BY if.project',
'SELECT to_char(date "2007-01-01" + (n || " month")::interval, "mon") As short_mname
FROM generate_series(0,11) n')
As mthreport;
```

#3.1 Regina on 2009-04-09 09:48 (Reply)

Hello Regina,

thanks for your answer.

I am using a vanilla installation of postgres 8.3 on Ubuntu 8.10.

Actually, my query missed the not equal '!=' sign. If I write the line in question like

```
WHERE if.num_used != 0
```

I get my stated Error.

If the result of the query is empty (which happens if you use the equal sign), there is no error, but that is not what I want.

Andreas

#3.1.1 Andreas on 2009-04-16 10:27 ([Reply](#))

Andreas,

Okay I get the same error you do so I guess I was wrong that this was possible. I'll have to investigate and see if there is a work-around for this issue.

#3.1.1.1 Regina on 2009-04-19 02:34 ([Reply](#))

ADD COMMENT

Name
Email
Homepage
In reply to

Standard emoticons like :-) and ;-) are converted to images.

Comment E-Mail addresses will not be displayed and will only be used for E-Mail notifications

To prevent automated Bots from commentspamming, please enter the string you see in the image below in the appropriate input box. Your comment will only be submitted if the strings match. Please ensure that your browser supports and accepts cookies, or your comment cannot be verified correctly.

H J W K

Enter the string from the spam-prevention image above:

☐ Remember Information?

☐ Subscribe to this entry

