

PostgreSQL for Developers

Postgres Open, Chicago, 2014

Dimitri Fontaine `dimitri@2ndQuadrant.fr`

Sept. 19, 2014

2ndQuadrant France PostgreSQL Major Contributor

- pgloader
- prefix, skytools, debian, ...
- CREATE EXTENSION
- CREATE EVENT TRIGGER



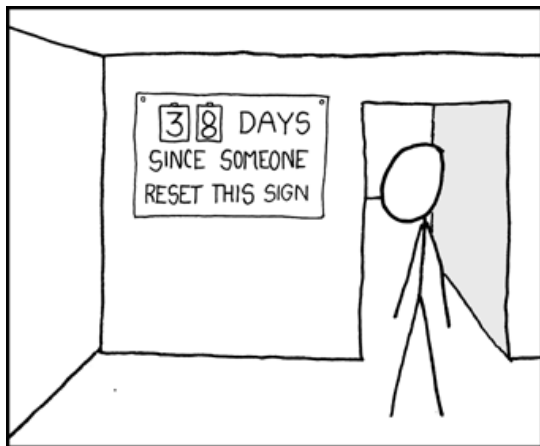
Tools and development languages

You're already using plenty of tools and languages already I'm sure, let's look at a typical web developer environment

- HTML
- Javascript
- *JQuery*
- **SQL**



A simple project



Project definition and scope

Let's try and solve something simple to get started:

- Managing a counter that can recycle
- Adding new measures in a time based fashion
- Do monthly reports to allow for invoicing
- Analyze the counter behavior

SQL: we start with DDLs

Joe Celko: 80% of the job is to define the schema

Example (DDL)

```
create table mesures(date timestamptz primary key,  
                    mesure integer);
```

```
dim=# \d mesures
```

```
\d mesures
```

```
Table "public.mesures"
```

Column	Type	Modifiers
date	timestamp with time zone	not null
mesure	integer	

```
Indexes:
```

```
"mesures_pkey" PRIMARY KEY, btree (date)
```

We take a very simple model for the presentation

```
create table measures(tick int, nb int);
```

```
insert into measures
```

```
values (1, 0), (2, 10), (3, 20), (4, 30), (5, 40),  
       (6, 0), (7, 20), (8, 30), (9, 60);
```



Testing data

Let's take some measures as if they came out of our counter, starting at 0, and with a *reset* in there. In that example, the global usage measured is $40 + 60 = 100$.

```
select * from measures;
```

tick	nb
1	0
2	10
3	20
4	30
5	40
6	0
7	20
8	30
9	60

(9 rows)



Aside: PostgreSQL knows about arrays

```
select array_agg(nb) from measures;  
       array_agg
```

```
-----  
{0,10,20,30,40,0,20,30,60}  
(1 row)
```



Finding the last counter value before *reset*

Write some *SQL* here

tick	nb	max
1	0	
2	10	
3	20	
4	30	
5	40	40
6	0	
7	20	
8	30	
9	60	60

(9 rows)



Window Functions: lead() over()

```
select tick,  
       nb,  
       lead(nb) over (order by tick)  
from measures;
```

tick	nb	lead
1	0	10
2	10	20
3	20	30
4	30	40
5	40	0
6	0	20
7	20	30
8	30	60
9	60	

(9 rows)

Window Functions and CASE

```
select tick, nb,  
       case when lead(nb) over w < nb  
            then nb  
            when lead(nb) over w is null  
            then nb  
            else null  
       end as max  
from measures  
window w as (order by tick);
```

tick	nb	max
1	0	
2	10	
3	20	
4	30	
5	40	40
6	0	
7	20	
8	30	
9	60	60

(9 rows)



Window Functions and WHERE clause

```
with t(tick, nb, max) as (  
    select tick, nb,  
           case when lead(nb) over w < nb then nb  
                when lead(nb) over w is null then nb  
                else null  
           end as max  
    from measures  
    window w as (order by tick)  
)  
select tick, nb, max from t where max is not null;  
tick | nb | max  
-----+-----+-----  
    5 | 40 | 40  
    9 | 60 | 60  
(2 rows)
```

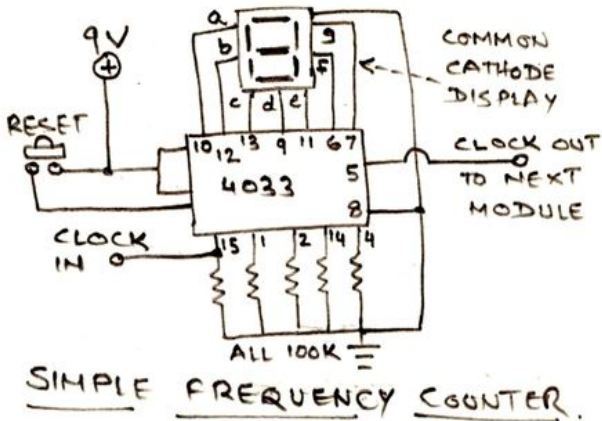


Common Table Expressions to complement WITH

```
with t(tops) as (  
    select case when lead(nb) over w < nb then nb  
              when lead(nb) over w is null then nb  
              else null  
            end as max  
    from measures  
    window w as (order by tick)  
)  
select sum(tops) from t;  
sum  
-----  
100  
(1 row)
```



Getting usage from the counter: done. SQL. 9 lines.



Let's test with more than one cycle

```
insert into measures
  values (10, 0), (11, 10), (12, 30), (13, 35), (14, 45),
         (15, 25), (16, 50), (17, 100), (18, 110);
```


Visualizing the cycles

```
with t(tick, nb, max) as (  
  select tick, nb,  
         case when lead(nb) over w < nb then nb  
              when lead(nb) over w is null then nb  
              else null  
         end as max  
  from measures  
  window w as (order by tick)  
)
```

```
select tick, nb, max from t where max is not null;
```

```
tick | nb  | max  
-----+-----+-----  
  5  |  40 |  40  
  9  |  60 |  60  
 14  |  45 |  45  
 18  | 110 | 110
```

(4 rows)

Resource usage, with several cycles

```
with t(tops) as (  
    select case when lead(nb) over w < nb then nb  
              when lead(nb) over w is null then nb  
              else null  
            end as max  
    from measures  
    window w as (order by tick)  
)  
select sum(tops) from t;  
sum  
-----  
255  
(1 row)
```



Limit measure taken into account



Limit measures period (time range)

```
select tick, nb
  from measures
 where tick >= 4 and tick < 14;
```

tick	nb
4	30
5	40
6	0
7	20
8	30
9	60
10	0
11	10
12	30
13	35



Limit measures period using first_value

```
select nb,
       first_value(nb) over w as first,
       case when lead(nb) over w < nb
            then nb
            when lead(nb) over w is null
            then nb
            else null
       end as max
from measures
where tick >= 4 and tick < 14
window w as (order by tick);
```

nb	first	max
30	30	
40	30	40
0	30	
20	30	
30	30	
60	30	60
0	30	
10	30	
30	30	
35	30	35

(10 rows)



Resource usage in a given period

```
with t as (  
  select tick,  
         first_value(nb) over w as first,  
         case when lead(nb) over w < nb then nb  
              when lead(nb) over w is null then nb  
              else null  
         end as max  
  from measures  
  where tick >= 4 and tick < 14  
  window w as (order by tick)  
)  
select sum(max) - min(first) as sum from t;  
sum  
-----  
105  
(1 row)
```

Counter behavior: *reset*

DC 24V



Range :0-99999 SourcingMap

Partitioning on the *reset*

```
with tops as (  
    select tick, nb,  
           case when lead(nb) over w < nb then nb  
                when lead(nb) over w is null then nb  
                else null  
           end as max  
    from measures  
    window w as (order by tick)  
)  
select tick, nb, max,  
       (select tick  
        from tops t2  
        where t2.tick >= t1.tick and max is not null  
        order by t2.tick  
        limit 1) as p  
from tops t1;
```



Partitioning on *reset*

tick	nb	max	p
1	0		5
2	10		5
3	20		5
4	30		5
5	40	40	5
6	0		9
7	20		9
8	30		9
9	60	60	9

tick	nb	max	p
10	0		14
11	10		14
12	30		14
13	35		14
14	45	45	14
15	25		18
16	50		18
17	100		18
18	110	110	18



Time range partitioning with PARTITION BY

```
with tops as ( <case lead() over()> ),
      parts as ( <self join limit 1> ),
      ranges as (
select
      first_value(tick) over w as start, -----+-----+-----
      last_value(tick) over w as end,          1 | 5 | 40
      max(max) over w                          6 | 9 | 60
from parts                                     10 | 14 | 45
window w as (PARTITION BY p                   15 | 18 | 110
              order by tick)                  (4 rows)
)
select * from ranges
where max is not null;
```



PostgreSQL knows about ranges: `int4range()`

```
with tops as ( <case lead() over()> ),
     parts as ( <self join limit 1> ),
     ranges as (
select int4range(
    first_value(tick) over w,
    last_value(tick) over w,
    '[]') as range,
    max(max) over w as compteur
from parts
window w as (partition by p
             order by tick)
)
select range, compteur
from ranges
where compteur is not null;
```

range	compteur
[1,6)	40
[6,10)	60
[10,15)	45
[15,19)	110
(4 rows)	



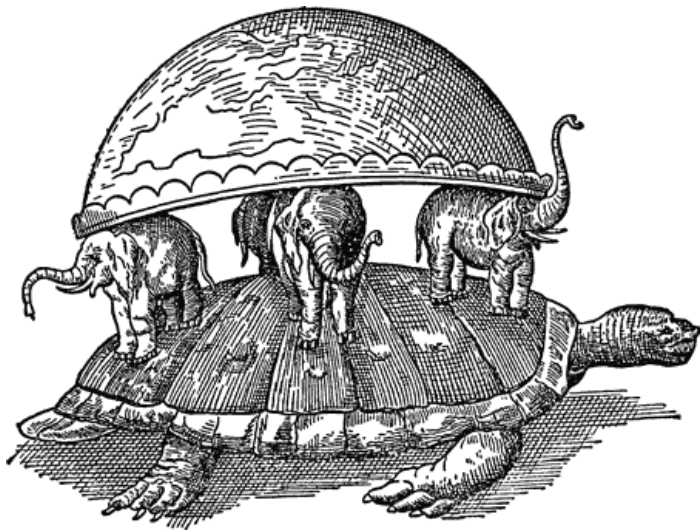
Usage by range using @>

```
with tops as ( <case lead() over()> ),
     parts as ( <self join limit 1> ),
     ranges as ( <int4range()
                 over (partition by
                       order by)> )
select range, compteur
   from ranges
  where compteur is not null
        and range @> 11;
```

range	compteur
[10,15)	45

(1 row)

Extensions and data types



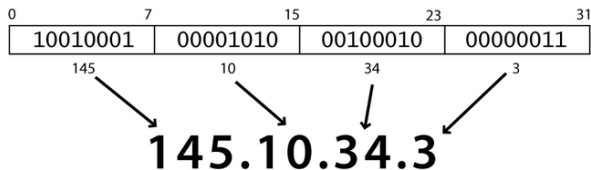
Some extensions example

46 Contribs, Community extensions, Private ones...

- **hll**
- **earthdistance**
- PostGIS
- pgcrypto
- cube
- pgq
- **ip4r**
- pgstattuple
- **ltree**
- **pg_trgm**
- **intarray**
- pg_buffercache
- citext
- wildspeed
- **prefix**
- pg_stat_statements
- **hstore**
- **plproxy**
- pgfincore
- **pgfincore**



IP Ranges, ip4r



IP Ranges, ip4r

```
table geolite.blocks limit 10;  
      iprange          | locid
```

```
-----+-----  
1.0.0.0/24            |    17  
1.0.1.0-1.0.3.255    |    49  
1.0.4.0/23           | 14409  
1.0.6.0/23           |    17  
1.0.8.0/21           |    49  
1.0.16.0/20          | 14614  
1.0.32.0/19          | 47667  
1.0.64.0/18          |   111  
1.0.128.0-1.0.147.255 |   209  
1.0.148.0/24         | 22537  
(10 rows)
```



IP Ranges, ip4r, Geolocation

PostgreSQL allows using SQL and JOINS to match IP4R with geolocation.

```
select *  
  from geolite.blocks  
  join geolite.location  
    using(locid)  
 where iprange  
        >>=  
        '74.125.195.147';
```



IP Ranges, ip4r, Geolocation

PostgreSQL allows using SQL and JOINS to match IP4R with geolocation.

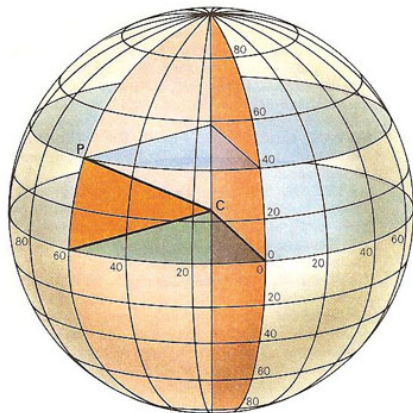
```
select *
  from geolite.blocks
  join geolite.location
    using(loid)
 where iprange
        >=
        '74.125.195.147';
```

```
-[ RECORD 1 ]-----
loid          | 2703
iprange       | 74.125.189.24-74.125.
country       | US
region        | CA
city          | Mountain View
postalcode    | 94043
location      | (-122.0574,37.4192)
metrocode    | 807
areacode      | 650

Time: 1.335 ms
```



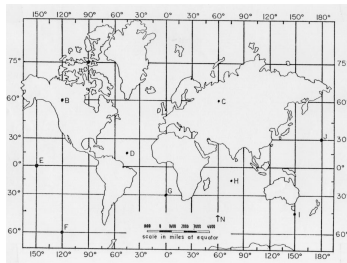
Earth Distance



How Far is The Nearest Pub

The point datatype is in-core

```
# CREATE TABLE pubnames  
(  
  id    bigint,  
  pos   POINT,  
  name  text  
);
```



How Far is The Nearest Pub

```
select name, pos
  from pubnames
order by pos <-> point(-6.25,53.346)
  limit 3;
```

Pub Name	pos
Ned's	(-6.2519967,53.3458267)
Sub Lounge	(-6.2542332,53.3469085)
O'Neill's of Pearse Street	(-6.2524389,53.3448589)

(3 rows)

Time: 18.679 ms



How Far is The Nearest Pub

```
CREATE INDEX on pubnames USING GIST(pos);
```

```
select name,  
       pos  
  from pubnames  

```

name	pos
Ned's	(-6.25,53.34)
Sub Lo	(-6.25,53.34)
O'Neil	(-6.25,53.34)

(3 rows)

Time: 0.849 ms



How Far is The Nearest Pub, in Miles please.

```
create extension cube;  
create extension earthdistance;
```

```
select name,  
       pos <@> point(-6.25,53.34) miles  
       from pubnames  
order by pos <-> point(-6.25,53.34)  
limit 3;
```

name	miles
Ned's	0.06
Sub Lo	0.07
O'Neil	0.12

(3 rows)

Time: 1.335 ms



Some pubs far away from here...

```
select c.name as city,  
pos <@> point(-6.25,53.34) as miles  
from pubnames p,  
    lateral (select name  
             from cities c  
             order by c.pos <-> p.pos  
             limit 1) c  
order by pos <-> point(-6.25,53.34)  
        desc  
limit 5;
```

city		miles
-----+-----		
Canterbury		399.44
Canterbury		378.91
Canterbury		392.08
Canterbury		397.30
Canterbury		379.68

(5 rows)

Time: 636.445 ms



Geolocation: ip4r meets earthdistance



Some pubs nearby... some place...

```
with geoloc as
(
  select location as l
    from location
   join blocks using(locid)
  where iprange
        >>=
        '212.58.251.195'
)
select name,
       pos <@> 1 miles
   from pubnames, geoloc
order by pos <-> 1
 limit 10;
```

name	miles
Blue Anchor	0.299
Dukes Head	0.360
Blue Ball	0.337
Bell (aka The Rat)	0.481
on the Green	0.602
Fox & Hounds	0.549
Chequers	0.712
Sportsman	1.377
Kingswood Arms	1.205
Tattenham Corner	2.007

(10 rows)

Time: 3.275 ms



Conclusion

You are already using SQL, make the best out of it!

