pgloader
prefix, skytools, debian, ...
CREATE EXTENSION
CREATE EVENT TRIGGER
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
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
Joe Celko: 80% of the job is to define the schema

Example (DDL)

```sql
create table mesures(date timestamp with time zone primary key,
                      mesure integer);
```

dim=# \d mesures
\d mesures

```
Table "public.mesures"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>timestamp with time zone</td>
<td>not null</td>
</tr>
<tr>
<td>mesure</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:
"mesures_pkey" PRIMARY KEY, btree (date)
```
We take a very simple model for the presentation

```sql
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

```sql
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

<table>
<thead>
<tr>
<th>tick</th>
<th>nb</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

(9 rows)
Window Functions: \texttt{lead()} \texttt{over()}

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

<table>
<thead>
<tr>
<th>tick</th>
<th>nb</th>
<th>lead</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>9</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

(9 rows)
Window Functions and CASE

```sql
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);
```

<table>
<thead>
<tr>
<th>tick</th>
<th>nb</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

(9 rows)
Window Functions and WHERE clause

```sql
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;
```

<table>
<thead>
<tr>
<th>tick</th>
<th>nb</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>9</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

(2 rows)
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

```sql
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;
```

<table>
<thead>
<tr>
<th>tick</th>
<th>nb</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>9</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>14</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>18</td>
<td>110</td>
<td>110</td>
</tr>
</tbody>
</table>

(4 rows)
```
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;
```

```
<table>
<thead>
<tr>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
</tr>
</tbody>
</table>

(1 row)
Limit measure taken into account
Limit measures period (time range)

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

<table>
<thead>
<tr>
<th>tick</th>
<th>nb</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>13</td>
<td>35</td>
</tr>
</tbody>
</table>
Limit measures period using first_value

```sql
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);
```

<table>
<thead>
<tr>
<th>nb</th>
<th>first</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>30</td>
<td>35</td>
</tr>
</tbody>
</table>

(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
Partitionning 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

<table>
<thead>
<tr>
<th>tick</th>
<th>nb</th>
<th>max</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>40</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>60</td>
<td>60</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tick</th>
<th>nb</th>
<th>max</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>30</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>35</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>45</td>
<td>45</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>25</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>50</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>17</td>
<td>100</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>18</td>
<td>110</td>
<td>110</td>
<td>18</td>
</tr>
</tbody>
</table>
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,
    max(max) over w
  from parts
  window w as (PARTITION BY p
    order by tick)
)
select * from ranges
where max is not null;

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>45</td>
</tr>
<tr>
<td>15</td>
<td>18</td>
<td>110</td>
</tr>
</tbody>
</table>

(4 rows)
PostgreSQL knows about ranges: \texttt{in4range()}

with tops as ( \texttt{<case lead() over()> } ),
    parts as ( \texttt{<self join limit 1> } ),
    ranges as ( 
        select \texttt{int4range(}
            first\_value(tick) \texttt{over w},
            last\_value(tick) \texttt{over w},
            '[]') \texttt{as range,}
            max(max) \texttt{over w as compteur}
        from parts
        window \texttt{w as (partition by p}
            order by tick)
    )

select range, compteur
    from ranges
where compteur is not null;

range | compteur

\[
\begin{array}{|c|c|}
\hline
[1,6) & 40 \\
[6,10) & 60 \\
[10,15) & 45 \\
[15,19) & 110 \\
\hline
\end{array}
\]

(4 rows)
Usage by range using @>

```sql
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;

<table>
<thead>
<tr>
<th>range</th>
<th>compteur</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10,15)</td>
<td>45</td>
</tr>
</tbody>
</table>
(1 row)
```

Dimitri Fontaine dimitri@2ndQuadrant.fr
PostgreSQL for Developers March 11, 2015 28 / 43
Extensions and data types
Some extensions example

46 Contribs, Community extensions, Private ones...

- hll
- cube
- ltree
- citext
- hstore
- earthdistance
- pgq
- pg_trgm
- wildspeed
- plproxy
- PostGIS
- ip4r
- intarray
- prefix
- pgfincore
- pgcrypto
- pg_statstuple
- pg_buffercache
- pg_stat_statements
- pgfincore
IP Ranges, ip4r

```
0  7  15  23  31
10010001  00001010  00100010  00000011
145  10  34  3
145.10.34.3
```
### IP Ranges, ip4r

```sql
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)
PostgreSQL allows using SQL and JOINs to match IP4R with geolocation.

```sql
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(locid)
where iprange
  >>>=  
'74.125.195.147';
```

```
- [ RECORD 1 ]------------------------
  locid          | 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

```sql
# CREATE TABLE pubnames
(
    id    bigint,
    pos   POINT,
    name  text
);
```
How Far is The Nearest Pub

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

<table>
<thead>
<tr>
<th>Pub Name</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ned’s</td>
<td>(-6.2519967,53.3458267)</td>
</tr>
<tr>
<td>Sub Lounge</td>
<td>(-6.2542332,53.3469085)</td>
</tr>
<tr>
<td>O’Neill’s of Pearse Street</td>
<td>(-6.2524389,53.3448589)</td>
</tr>
</tbody>
</table>

(3 rows)

Time: 18.679 ms
How Far is The Nearest Pub

CREATE INDEX on pubnames USING GIST(pos);

```
select name, pos
from pubnames
order by pos <-> point(-0.12,51)
limit 3;
```

<table>
<thead>
<tr>
<th>name</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ned’s</td>
<td>(-6.25,53.34)</td>
</tr>
<tr>
<td>Sub Lo</td>
<td>(-6.25,53.34)</td>
</tr>
<tr>
<td>O’Neil</td>
<td>(-6.25,53.34)</td>
</tr>
</tbody>
</table>

(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;
```

<table>
<thead>
<tr>
<th>name</th>
<th>miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ned’s</td>
<td>0.06</td>
</tr>
<tr>
<td>Sub Lo</td>
<td>0.07</td>
</tr>
<tr>
<td>O’Neil</td>
<td>0.12</td>
</tr>
</tbody>
</table>

(3 rows)

Time: 1.335 ms
Some pubs far away from here...

```sql
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;
```

<table>
<thead>
<tr>
<th>city</th>
<th>miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canterbury</td>
<td>399.44</td>
</tr>
<tr>
<td>Canterbury</td>
<td>378.91</td>
</tr>
<tr>
<td>Canterbury</td>
<td>392.08</td>
</tr>
<tr>
<td>Canterbury</td>
<td>397.30</td>
</tr>
<tr>
<td>Canterbury</td>
<td>379.68</td>
</tr>
</tbody>
</table>

(5 rows)

Time: 636.445 ms
Geolocation: ip4r meets earthdistance
Some pubs nearby… some place…

```sql
with geoloc as 
(
    select location as l 
    from location 
    join blocks using(locid) 
    where iprange >>>= '212.58.251.195'
)

    select name, 
    pos <@> l miles 
    from pubnames, geoloc 
order by pos <-> l 
limit 10;
```

<table>
<thead>
<tr>
<th>name</th>
<th>miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Anchor</td>
<td>0.299</td>
</tr>
<tr>
<td>Dukes Head</td>
<td>0.360</td>
</tr>
<tr>
<td>Blue Ball</td>
<td>0.337</td>
</tr>
<tr>
<td>Bell (aka The Rat) on the Green</td>
<td>0.481</td>
</tr>
<tr>
<td>Fox &amp; Hounds</td>
<td>0.549</td>
</tr>
<tr>
<td>Chequers</td>
<td>0.712</td>
</tr>
<tr>
<td>Sportsman</td>
<td>1.377</td>
</tr>
<tr>
<td>Kingswood Arms</td>
<td>1.205</td>
</tr>
<tr>
<td>Tattenham Corner</td>
<td>2.007</td>
</tr>
</tbody>
</table>

(10 rows)

Time: 3.275 ms
Conclusion

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