PostgreSQL
for developers

Dimitri Fontaine dimitri@2ndQuadrant.fr

July, 2 2012
pgloader, prefix, skytools, debian, ...

CREATE EXTENSION

CREATE EVENT TRIGGER

Bi-Directional Replication

Partitioning
pgloader, prefix, skytools, debian, ...

CREATE EXTENSION

CREATE EVENT TRIGGER

Bi-Directional Replication

Partitioning
Dimitri Fontaine

2ndQuadrant France
PostgreSQL Major Contributor

- pgloader, prefix, skytools, debian, ...
- CREATE EXTENSION
- CREATE EVENT TRIGGER
- Bi-Directional Replication
- Partitioning
Dimitri Fontaine

2ndQuadrant France
PostgreSQL Major Contributor

- pgloader, prefix, skytools, debian, ...
- CREATE EXTENSION
- CREATE EVENT TRIGGER
- Bi-Directional Replication
- Partitioning
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

38 DAYS
SINCE SOMEONE
RESET THIS SIGN
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
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
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
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
**Joe Celko:** 80% of the job is to define the schema

**Example (DDL)**

```sql
create table mesures(date timestamptz 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 \(40 + 60 = 100\).

```sql
select * from measures;

<table>
<thead>
<tr>
<th>tick</th>
<th>nb</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
</tr>
<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>
</tbody>
</table>

(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: lead() 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 et 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></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 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;

<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

```sql
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)
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;
```

<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></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>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

```sql
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

```sql
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;
```

Dimitri Fontaine
dimitri@2ndQuadrant.fr

PostgreSQL

July, 2 2012 24 / 29
### 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>
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)
    (4 rows)
) select * from ranges
  where max is not null;
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;

---

[1,6] | 40
[6,10) | 60
[10,15) | 45
[15,19) | 110
(4 rows)
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)
Conclusion

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