PostgreSQL

for developers

Dimitri Fontaine dimitri@2ndQuadrant.fr

July, 2 2012

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*

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

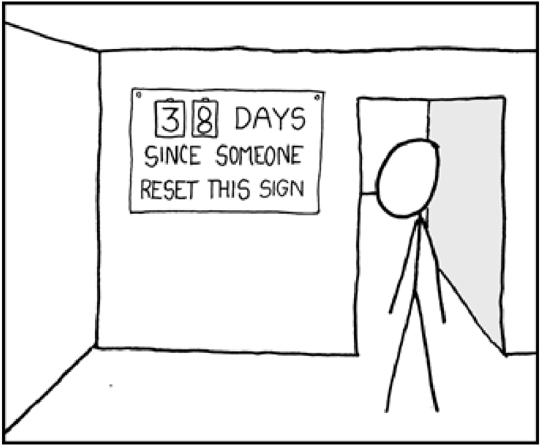
- `pgloader`, `prefix`, `skytools`, `debian`, ...
- `CREATE EXTENSION`
- `CREATE EVENT TRIGGER`
- *Bi-Directional Replication*
- *Partitioning*

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



Project definition and scope

Let's try and solve so,ething 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 so,ething 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 so,ething 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 so,ething 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 et 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)
```

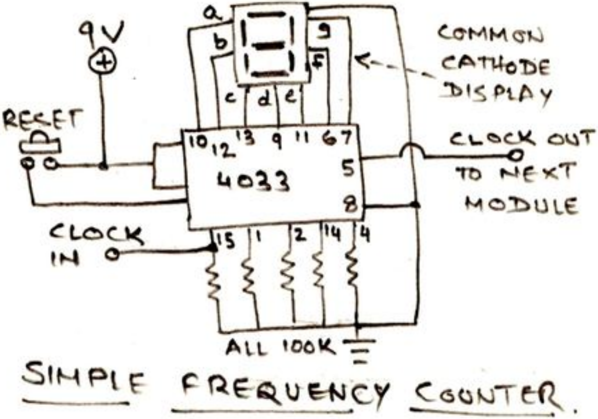
Window Functions and WHERE clause

```
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*

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*

```
 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                                 start | end | max
      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: in4range()

```
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)
```

Conclusion

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