Look Out The Window Functions

and free your SQL

Gianni Ciolli

2ndQuadrant Italia

PostgreSQL Conference Europe 2011
October 18-21, Amsterdam

Outline

1. Concepts
   - Aggregates
   - Different aggregations
   - Partitions
   - Window frames
2. Syntax
   - Frames from 9.0
   - Frames in 8.4
3. Other
   - A larger example
   - Question time

Aggregates 1

Example of an *aggregate*

Problem 1

How many rows there are in table `a`?

Solution

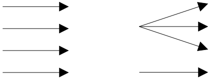
```
SELECT count(*) FROM a;
```

- Here `count` is an *aggregate function* (SQL keyword `AGGREGATE`).

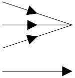
Aggregates 2

Functions and Aggregates

- FUNCTIONs:
  - input: *one* row
  - output: either *one* row or *a set* of rows:



- AGGREGATEs:
  - input: *a set* of rows
  - output: *one* row



Different aggregations 1

Without window functions, and with them

`GROUP BY` $col_1, \ldots, col_n$	*window functions*
any supported PostgreSQL version	only PostgreSQL version 8.4+
compute aggregates by creating *groups*	compute aggregates via *partitions* and *window frames*
output is one row *for each group*	output is one row *for each input row*

Different aggregations 2

Without window functions, and with them

`GROUP BY` $col_1, \ldots, col_n$	*window functions*
only one way of aggregating for each group	different rows in the same partition can have different window frames
only one way of grouping for each `SELECT`	aggregates in the same `SELECT` can use different partitions

Different aggregations

Dataset #1 for the next examples

```
SELECT *
FROM a
ORDER BY x;
```

```
 x  | y
----+---
  1 | 1
  2 | 2
  3 | 3
  4 | 1
  5 | 2
  6 | 3
  7 | 1
  8 | 2
  9 | 3
 10 | 1
```

Different aggregations

Example 1, with `GROUP BY`

```
SELECT y,sum(x)
FROM a
GROUP BY y
ORDER BY y;
```

```
 y | sum
---+-----
 1 |  22
 2 |  15
 3 |  18
```

Different aggregations

Example 2, with window functions

```
SELECT x,y,sum(x)
OVER (PARTITION BY y)
FROM a
ORDER BY y,x;
```

x	y	sum
1	1	22
4	1	22
7	1	22
10	1	22
2	2	15
5	2	15
8	2	15
3	3	18
6	3	18
9	3	18

Different aggregations

Example 3, with window functions (reordered)

```
SELECT x,y,sum(x)
OVER (PARTITION BY y)
FROM a
ORDER BY x;
```

x	y	sum
1	1	22
2	2	15
3	3	18
4	1	22
5	2	15
6	3	18
7	1	22
8	2	15
9	3	18
10	1	22

Partitions

Basics

- PARTITION BY $E_1, E_2, \ldots, E_n$
- each row belongs to one *partition*
- similar to GROUP BY
- with GROUP BY you can partition in one way only for each SELECT
- with window functions, you can partition in *several ways at once* for each SELECT (up to a different partitioning for each aggregate function)

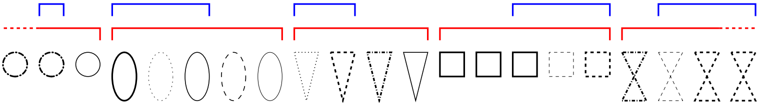
Window frames

Basics

- Each row has its *window frame*
- The window frame is included in that row's partition
- Default: the window frame is equal to that row's partition
- The result of the aggregate for that row is computed using all the rows in its window frame
- The frame can *move* as the row moves

Partitions and frames

In a picture



- red lines = partitions
- blue lines = frames

Window frames

Frame movement and `ORDER BY`

- Notions like “the current row plus the previous three rows” depend on a notion of *ordering*
- You can specify an ordering with `ORDER BY`
- If you don’t, all the rows will be *peers*

Window frames

`RANGE` or `ROWS`

- Frames are of two kinds:
  - `RANGE` frames, according to the ordering
  - `ROWS` frames, according to the actual row number
- Difference:
  - in `ROWS` frames, two rows will never be peers
  - in `RANGE` frames, they might be

Window frames

Examples of possible frames

- Some interesting frames:
  - the current row and all its peers
  - from the start of the partition to the current row
  - the previous row only (the first row has an empty frame)
  - the next row only (the last row has an empty frame)
  - from two rows before the next row only (the last row has an empty frame)
  - ...
- The frame is *trimmed* to fit into the partition

Window frames

Example 4: trivial partition and trivial window frame

```
SELECT
  sum(x) OVER ()
FROM a;
```

- The whole table is one partition
- The window frame is the whole partition
- Very similar to:

```
SELECT sum(x)
FROM a;
```

Window frames

Example 5: frame is current row plus previous row

```
SELECT x,y,sum(x)
OVER (PARTITION BY y
  ORDER BY x
  ROWS 1 PRECEDING)
FROM a
ORDER BY x;
```

```
 x  | y | sum
----+---+-----
  1 | 1 |   1
  2 | 2 |   2
  3 | 3 |   3
  4 | 1 |   5
  5 | 2 |   7
  6 | 3 |   9
  7 | 1 |  11
  8 | 2 |  13
  9 | 3 |  15
 10 | 1 |  17
```

Window frames

Example 6: frame is current row plus previous row and next row

```
SELECT x,y,sum(x)
OVER (PARTITION BY y
  ORDER BY x
  ROWS BETWEEN 1 PRECEDING
       AND 1 FOLLOWING)
FROM a
ORDER BY x;
```

```
 x  | y | sum
----+---+-----
  1 | 1 |   5
  2 | 2 |   7
  3 | 3 |   9
  4 | 1 |  12
  5 | 2 |  15
  6 | 3 |  18
  7 | 1 |  21
  8 | 2 |  13
  9 | 3 |  15
 10 | 1 |  17
```

Syntax for Window Frames from 9.0

`[ RANGE | ROWS ] BETWEEN` **`FrameStart`** `AND` **`FrameEnd`**

- **`FrameStart`** and **`FrameEnd`** can be chosen among:
  - `UNBOUNDED PRECEDING`
  - *n* `PRECEDING` (only `ROWS` for now)
  - `CURRENT ROW`
  - *n* `FOLLOWING` (only `ROWS` for now)
  - `UNBOUNDED FOLLOWING`
- abbreviation:

  `[ RANGE | ROWS ]` **`FrameStart`**

  meaning

  `[ RANGE | ROWS ] BETWEEN` **`FrameStart`**
  `AND CURRENT ROW`

Syntax for Window Frames before 9.0

- no window functions before 8.4
- in 8.4, only these frames allowed:
  - `[ RANGE | ROWS ] BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`
    (the window frame is from the start of the partition to the current row)
  - `[ RANGE | ROWS ] BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`
    (the window frame is the whole partition)

Heat diffusion on a 2D grid

Window functions over a larger dataset

- We consider a square $N \times N$ grid
- Heat flows along each horizontal or vertical segment
- The speed on a given segment is proportional to the temperature gap between its two extremes
- Our plan:
  1. we produce a dataset which simulates heat flow using window functions on PostgreSQL;
  2. snapshots of that dataset are plotted with *gnuplot* and assembled into a movie using *mencoder*;
  3. we view the movie to confirm that heat flows as we would expect.

The discrete 2D heat equation

- The following equation encodes our model:

$$\begin{aligned}\Delta z(x,y) = C\Big[& z(x-1,y) - 2z(x,y) + z(x+1,y) \\ & + z(x,y-1) - 2z(x,y) + z(x,y+1)\Big] \qquad (1)\end{aligned}$$

- $z(x,y)$ is the temperature at the point (x,y)
- Coefficient C controls the speed of the simulation
- Equation (1) can be expressed using window functions

Discrete heat equation via window functions

- The expression

$$z(x-1,y)-2z(x,y)+z(x+1,y)$$

can be written as a window function:

```
sum(z) OVER (
    PARTITION BY y
    ORDER BY x
    ROWS BETWEEN 1 PRECEDING
             AND 1 FOLLOWING
) - 3 * z
```

- The same idea applies to the other half of Equation (1):

$$z(x,y-1)-2z(x,y)+z(x,y+1)$$

Heat flow scenarios

- We set initial data according to one of the following scenarios:
  - a hot point
  - a hot segment
  - a hot square
  - a cold square
- *hot* and *cold* are meant in comparison to the rest of the points
- then we run the simulation and view the video...

Benchmark (sort of) with $N = 16$

```
Subquery Scan on h0 (cost=153.32..210.49 rows=1089 width=36)
 -> WindowAgg (cost=153.32..175.10 rows=1089 width=20)
    InitPlan 2 (returns $1)
     -> Result (cost=0.05..0.06 rows=1 width=0)
        InitPlan 1 (returns $0)
         -> Limit (cost=0.00..0.05 rows=1 width=4)
            -> Index Scan Backward using h1_t_idx on h1
              (cost=0.00..57.03 rows=1089 width=4)
               Index Cond: ((t IS NOT NULL) AND (t < 500))
    -> Sort (cost=153.26..155.98 rows=1089 width=20)
       Sort Key: pg_temp_2.h1.x, pg_temp_2.h1.y
       -> WindowAgg (cost=76.55..98.33 rows=1089 width=20)
          -> Sort (cost=76.55..79.27 rows=1089 width=20)
             Sort Key: pg_temp_2.h1.y, pg_temp_2.h1.x
             -> Seq Scan on h1
               (cost=0.00..21.61 rows=1089 width=20)
                Filter: (t = $1)
```

Question time

- **Any questions?**

Thank you for your attention!

Feedback

`http://2011.pgconf.eu/feedback`

Licence