Better, Faster TPS Reports

New!

Reach Outside the Current Row
Better, Faster TPS Reports

- **Windowing Function**
  - Operates on a window
  - Returns a value for each row
  - Calculates value from the rows in the window
Better, Faster TPS Reports

• You can use...
  – New window functions
  – Existing aggregate functions
  – User-defined window functions
  – User-defined aggregate functions
Better, Faster TPS Reports

[Aggregates]  SELECT key, SUM(val) FROM tbl GROUP BY key;
[Windowing Functions] 

SELECT key, SUM(val) OVER (PARTITION BY key) FROM tbl;
ROW_NUMBER (Before)

SELECT
    el.empno,
    el.depname,
    el.salary,
    count(*) AS row_number
FROM
    empsalary el
JOIN
    empsalary e2
    ON (el.empno < e2.empno)
GROUP BY el.empno, el.depname, el.salary
ORDER BY el.empno DESC;
# ROW_NUMBER (Before)

## Oops!

<table>
<thead>
<tr>
<th>empno</th>
<th>depname</th>
<th>salary</th>
<th>row_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>develop</td>
<td>6000</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>sales</td>
<td>5500</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>develop</td>
<td>5200</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>develop</td>
<td>5200</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>sales</td>
<td>5000</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>sales</td>
<td>4800</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>sales</td>
<td>4800</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>develop</td>
<td>4500</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>develop</td>
<td>4200</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>personnel</td>
<td>3900</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>personnel</td>
<td>3500</td>
<td>11</td>
</tr>
</tbody>
</table>

(11 rows)
SELECT
  empno,
  depname,
  salary,
  row_number() OVER (  
    ORDER BY salary DESC NULLS LAST
  )
FROM
  empsalary
ORDER BY salary DESC;
# ROW_NUMBER (After)

## Yippee!

| empno | depname | salary | row_number |
|-------+---------+--------+------------|
| 8     | develop | 6000   | 1          |
| 6     | sales   | 5500   | 2          |
| 10    | develop | 5200   | 3          |
| 11    | develop | 5200   | 4          |
| 1     | sales   | 5000   | 5          |
| 3     | sales   | 4800   | 6          |
| 4     | sales   | 4800   | 7          |
| 9     | develop | 4500   | 8          |
| 7     | develop | 4200   | 9          |
| 2     | personnel | 3900 | 10         |
| 5     | personnel | 3500 | 11         |

(11 rows)
Built-in Windowing Functions

- `row_number()`
- `rank()`
- `dense_rank()`
- `percent_rank()`
- `cume_dist()`
- `ntile()`
- `lag()`
- `lead()`
- `first_value()`
- `last_value()`
- `nth_value()`
Generate Points

WITH RECURSIVE x(i)
AS ( 
  VALUES (0)
UNION ALL 
  SELECT i + 1 

  FROM x
  WHERE i < 101 
),
Generate Points

\[ Z(I_x, I_y, C_x, C_y, X, Y, I) \]
\[
\text{AS (}
\]
\[
\begin{align*}
\text{SELECT } & I_x, I_y, \\
& X::\text{float}, Y::\text{float}, \\
& X::\text{float}, Y::\text{float}, \\
& 0 \\
\end{align*}
\]
\[
\text{FROM}
\]
Generate Points

\[(\text{SELECT } -2.2 + 0.031 \times i, i \text{ FROM } x) \text{ AS } xgen(x,ix)\]
\[\text{CROSS JOIN}\]
\[(\text{SELECT } -1.5 + 0.031 \times i, i \text{ FROM } x) \text{ AS } ygen(y,iy)\]
Generate Points

UNION ALL
Generate Points

SELECT
    Ix, Iy, Cx, Cy,
    X * X - Y * Y + Cx AS X,
    Y * X * 2 + Cy,
    I + 1
FROM Z
WHERE X * X + Y * Y < 16.0
AND I < 27
)
Choose Some

\[ Z_t \ (I_x, \ I_y, \ I) \ AS \ ( \ 
\quad \text{SELECT} \ I_x, \ I_y, \ \text{MAX}(I) \ \text{AS} \ I \\
\quad \text{FROM} \ Z \\
\quad \text{GROUP BY} \ I_y, \ I_x \\
\quad \text{ORDER BY} \ I_y, \ I_x \\
) \]
SELECT array_to_string(
    array_agg(
        SUBSTRING(
            '.....----------%%%%@@@###',
            GREATEST(I,1)
        ),'
    )
)
FROM Zt
GROUP BY Iy
ORDER BY Iy;
Travelling Salesman Problem

Given a number of cities and the costs of travelling from any city to any other city, what is the least-cost round-trip route that visits each city exactly once and then returns to the starting city?
TSP Schema

CREATE TABLE pairs (  
    from_city TEXT NOT NULL,  
    to_city TEXT NOT NULL,  
    distance INTEGER NOT NULL,  
    PRIMARY KEY(from_city, to_city),  
    CHECK (from_city < to_city)  
);
TSP Data

INSERT INTO pairs
VALUES
('Bari','Bologna',672),
('Bari','Bolzano',939),
('Bari','Firenze',723),
('Bari','Genova',944),
('Bari','Milan',881),
('Bari','Napoli',257),
('Bari','Palermo',708),
('Bari','Reggio Calabria',464),
....
TSP Program:
Symmetric Setup

WITH RECURSIVE both_ways(
    from_city,
    to_city,
    distance
) AS (
    SELECT
        from_city,
        to_city,
        distance
    FROM
        pairs
    UNION ALL
    SELECT
        to_city AS "from_city",
        from_city AS "to_city",
        distance
    FROM
        pairs
),
TSP Program:
Symmetric Setup

WITH RECURSIVE both_ways(
    from_city,
    to_city,
    distance
)
AS /* Distances One Way */
(SELECT
    from_city,
    to_city,
    distance
FROM
    pairs
UNION ALL
SELECT
    to_city AS "from_city",
    from_city AS "to_city",
    distance
FROM
    pairs
),
TSP Program:
Symmetric Setup

WITH RECURSIVE both_ways(
    from_city,
    to_city,
    distance
)
AS (  
    SELECT  
        from_city,
        to_city,
        distance
    FROM
    pairs
    UNION ALL /* Distances Other Way */
    SELECT
        to_city AS "from_city",
        from_city AS "to_city",
        distance
    FROM
    pairs
)
TSP Program:

Path Initialization Step

paths (  
    from_city,  
    to_city,  
    distance,  
    path  
)  
AS (  
    SELECT  
        from_city,  
        to_city,  
        distance,  
        ARRAY[from_city] AS "path"  
    FROM  
    both_ways b1  
    WHERE  
        b1.from_city = 'Roma'  
    UNION ALL  
)
SELECT
  b2.from_city,
  b2.to_city,
  p.distance + b2.distance,
  p.path || b2.from_city
FROM
  both_ways b2
JOIN
  paths p
ON (
  p.to_city = b2.from_city
AND
  b2.from_city <> ALL (p.path[
    2:array_upper(p.path,1)
  ]) /* Prevent re-tracing */
AND
  array_upper(p.path,1) < 6
)
TSP Program:
Timely Termination Step

SELECT
    b2.from_city,
    b2.to_city,
    p.distance + b2.distance,
    p.path | | b2.from_city
FROM
    both_ways b2
JOIN
    paths p
ON (  
    p.to_city = b2.from_city
AND
    b2.from_city <> ALL (p.path[
        2:array_upper(p.path,1)
    ]) /* Prevent re-tracing */
AND
    array_upper(p.path,1) < 6 /* Timely Termination */
)

AND
TSP Program: Filter and Display

```
SELECT
    path || to_city AS "path",
distance
FROM
    paths
WHERE
    to_city = 'Roma'
AND
    ARRAY['Milan','Firenze','Napoli'] <@ path
ORDER BY distance, path
LIMIT 1;
```
TSP Program:
Filter and Display

davidfetter@tsp=# \i travelling_salesman.sql
path | distance
----------------------------------
{Roma,Firenze,Milan,Napoli,Roma} | 1553
(1 row)

Time: 11679.503 ms
Who Posts Most?
CREATE TABLE forum_users ( 
    user_name TEXT NOT NULL, 
    CHECK(user_name = trim(user_name)), 
    user_id SERIAL UNIQUE 
); 

CREATE UNIQUE INDEX forum_user_user_name_unique 
    ON forum_users(lower(user_name)); 

INSERT INTO forum_users (user_name) 
VALUES 
    ('Tom Lane'), ('Robert Haas'), ('Alvaro Herrera'), ('Dave Page'), ('Heikki Linnakangas'), ('Magnus Hagander'), ('Gregory Stark'), ('Josh Berkus'), ('David Fetter'), ('Benjamin Reed');
CREATE TABLE message (  
    message_id INTEGER PRIMARY KEY,  
    parent_id INTEGER  
    REFERENCES message(message_id),  
    message_text TEXT NOT NULL,  
    forum_user_id INTEGER  
    NOT NULL REFERENCES forum_users(user_id)  
);
Add some posts

INSERT INTO message
WITH RECURSIVE m(
    message_id,
    parent_id,
    message_text,
    forum_user_id)
AS (VALUES(1, NULL::integer, md5(random()::text),1)
Add some posts

UNION ALL
Add some posts

SELECT
    message_id+1,
    CASE
        WHEN random() >= .5 THEN NULL
        ELSE FLOOR(random() * message_id)+1
    END::integer,
    md5(random()::text),
    floor(random() * 10)::integer +1
FROM m
WHERE message_id < 1001
)
SELECT * FROM m;
WELL?!?

Patience :)

WITH RECURSIVE t1 AS ( 
  SELECT
    /* First message in the thread is the thread ID */
    message_id AS thread_id,
    message_id,
    message_id,
    parent_id,
    forum_user_id,
    ARRAY[message_id] AS path
  FROM message
  WHERE parent_id IS NULL
Find the Next Ones

UNION ALL
Find the Next Ones

```sql
SELECT
t1.thread_id,
m.message_id,
m.parent_id,
m.forum_user_id,
t1.path || m.message_id
FROM message m
JOIN t1 ON
  (t1.message_id = m.parent_id)
),
```
Count Posters in Each Thread

t2 AS ( 
    SELECT
        thread_id,
        forum_user_id,
        count(*) AS reply_count
    FROM tl
    GROUP BY thread_id, forum_user_id
    ORDER BY thread_id, count(*)
),
Find the Top Posters

t3 AS (  
    SELECT thread_id,  
    max(reply_count) AS reply_count  
    FROM t2  
    GROUP BY thread_id  
)
SELECT t2.thread_id, f.user_name, t3.reply_count
FROM t2
JOIN t3 USING (thread_id, reply_count)
JOIN forum_users f ON (f.user_id = t2.forum_user_id)
WHERE reply_count > 3
ORDER BY reply_count DESC;
Top Posters :)  

<table>
<thead>
<tr>
<th>thread_id</th>
<th>user_name</th>
<th>reply_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tom Lane</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>Gregory Stark</td>
<td>9</td>
</tr>
<tr>
<td>82</td>
<td>Magnus Hagander</td>
<td>5</td>
</tr>
<tr>
<td>108</td>
<td>Dave Page</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>Josh Berkus</td>
<td>4</td>
</tr>
</tbody>
</table>

(5 rows)
With CTE and Windowing, SQL is Turing Complete.
Cyclic Tag System

The productions are encoded in the table "p" as follows:

- "iter" is the production number;
- "rnum" is the index of the bit;
- "tag" is the bit value.

This example uses the productions:
110 01 0000

The initial state is encoded in the non-recursive union arm, in this case just '1'.

The \((r.iter \% n)\) subexpression encodes the number of productions, which can be greater than the size of table "p", because empty productions are not included in the table.
Cyclic Tag System

Parameters:
   the content of "p"
   the content of the non-recursive branch
   the 3 in (r.iter % 3)

"p" encodes the production rules; the non-recursive branch is the initial state, and the 3 is the number of rules

The result at each level is a bitstring encoded as 1 bit per row, with rnum as the index of the bit number.

At each iteration, bit 0 is removed, the remaining bits shifted up one, and if and only if bit 0 was a 1, the content of the current production rule is appended at the end of the string.
Proof:

Construct a Cyclic Tag System with CTEs and Windowing.
WITH RECURSIVE
p(iter,rnum,tag) AS ( 
    VALUES (0,0,1),(0,1,1),(0,2,0),
            (1,0,0),(1,1,1),
            (2,0,0),(2,1,0),(2,2,0),(2,3,0)
),
Proof:

\[
\begin{align*}
\text{Proof:} \\
\text{r(iter, rnum, tag) AS (} \\
\text{ VALUES (0,0,1)} \\
\text{ UNION ALL } \\
\text{ SELECT r.iter+1,} \\
\text{ CASE} \\
\text{ WHEN r.rnum=0 THEN p.rnum + max(r.rnum) OVER ()} \\
\text{ ELSE r.rnum-1} \\
\text{ END,} \\
\text{ CASE} \\
\text{ WHEN r.rnum=0 THEN p.tag} \\
\text{ ELSE r.tag} \\
\text{ END } \\
\text{ FROM r} \\
\text{ LEFT JOIN p} \\
\text{ ON (r.rnum=0 and r.tag=1 and p.iter=(r.iter % 3))} \\
\text{ WHERE} \\
\text{ r.rnum>0} \\
\text{ OR p.iter IS NOT NULL}
\end{align*}
\]
Proof:

```
SELECT iter, rnum, tag
FROM r
ORDER BY iter, rnum;
```
Thanks
Andrew (RhodiumToad) Gierth
Questions?
Comments?
Straitjackets?
Thank You!