High Performance SQL with PostgreSQL 8.4

Lists and Recursion and Trees, Oh My!

OSCON 2009
T.P.S. REPORT

COVER SHEET

Prepared By: ___________________________ Date: ___________________________

Device/Program Type: ____________________________________________________

Product Code: ___________________________ Customer: ______________________

Vendor: ___________________________ Data Loss: __________________________

Due Date: ___________________________ Target Run Date: ______________________

Test Date: ___________________________ Reference Guide: ______________________

Program Run Time: ___________________________ Number of Error Messages: ____________

Program Language: ___________________________ Comments: ___________________________

CONFIDENTIAL
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;
Better, Faster TPS Reports

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

SELECT
    e1.empno,
    e1.depname,
    e1.salary,
    count(*) AS row_number
FROM
    empsalary e1
JOIN
    empsalary e2
    ON (e1.empno < e2.empno)
GROUP BY e1.empno, e1.depname, e1.salary
ORDER BY e1.empno DESC;
## ROW_NUMBER (Before)

<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!

<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>10</td>
<td>develop</td>
<td>5200</td>
<td>3</td>
</tr>
<tr>
<td>11</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>6</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)
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

\[ Z(Ix, Iy, Cx, Cy, X, Y, I) \]

\[ \text{AS (} \]

\[ \text{SELECT Ix, Iy,} \]
\[ \text{X::float, Y::float,} \]
\[ \text{X::float, Y::float,} \]
\[ 0 \]

\[ \text{FROM} \]
Generate Points

\[
\begin{align*}
&(\text{SELECT } -2.2 + 0.031 \times i, i \text{ FROM } x) \text{ AS } xgen(x,ix) \\
&\text{CROSS JOIN} \\
&(\text{SELECT } -1.5 + 0.031 \times i, i \text{ FROM } x) \text{ AS } ygen(y,iy)
\end{align*}
\]
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

\[ Z_t (I_x, I_y, I) \text{ AS (} \]
\[ \begin{array}{l}
\text{SELECT } I_x, I_y, \text{MAX}(I) \text{ AS } I \\
\text{FROM } Z \\
\text{GROUP BY } I_y, I_x \\
\text{ORDER BY } I_y, I_x \\
\end{array} \]
\)
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
) /* Working Table */
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
TSP Program:
Path Recursion 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)

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 */
)
```
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

<table>
<thead>
<tr>
<th>path</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Roma, Firenze, Milan, Napoli, Roma}</td>
<td>1553</td>
</tr>
</tbody>
</table>

(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
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,  
        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)
)
,
```
t2 AS ( 
    SELECT 
        thread_id, 
        forum_user_id, 
        count(*) AS reply_count 
    FROM t1 
    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
)
Show Them :)

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 \((r.iter \mod n)\) 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.
Proof:

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:

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

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