Lists and Recursion and Trees

Oh My!
FOSDEM, Brussels, February 7, 2010

Copyright © 2010
David Fetter david.fetter@pgexperts.com
All Rights Reserved

T.P.S. REPORT

COVER SHEET

Prepared By:____Date:____________

Device/Program Type:____

Product Code:_____________Customer:____

Vendor:____

Due Date:_____________________Data Loss:____________________________________

Test Date:_____________________Target Run Date:_____________________________

Program Run Time:______________Reference Guide:______________________________

Program Language:______________Number of Error Messages:_____________________

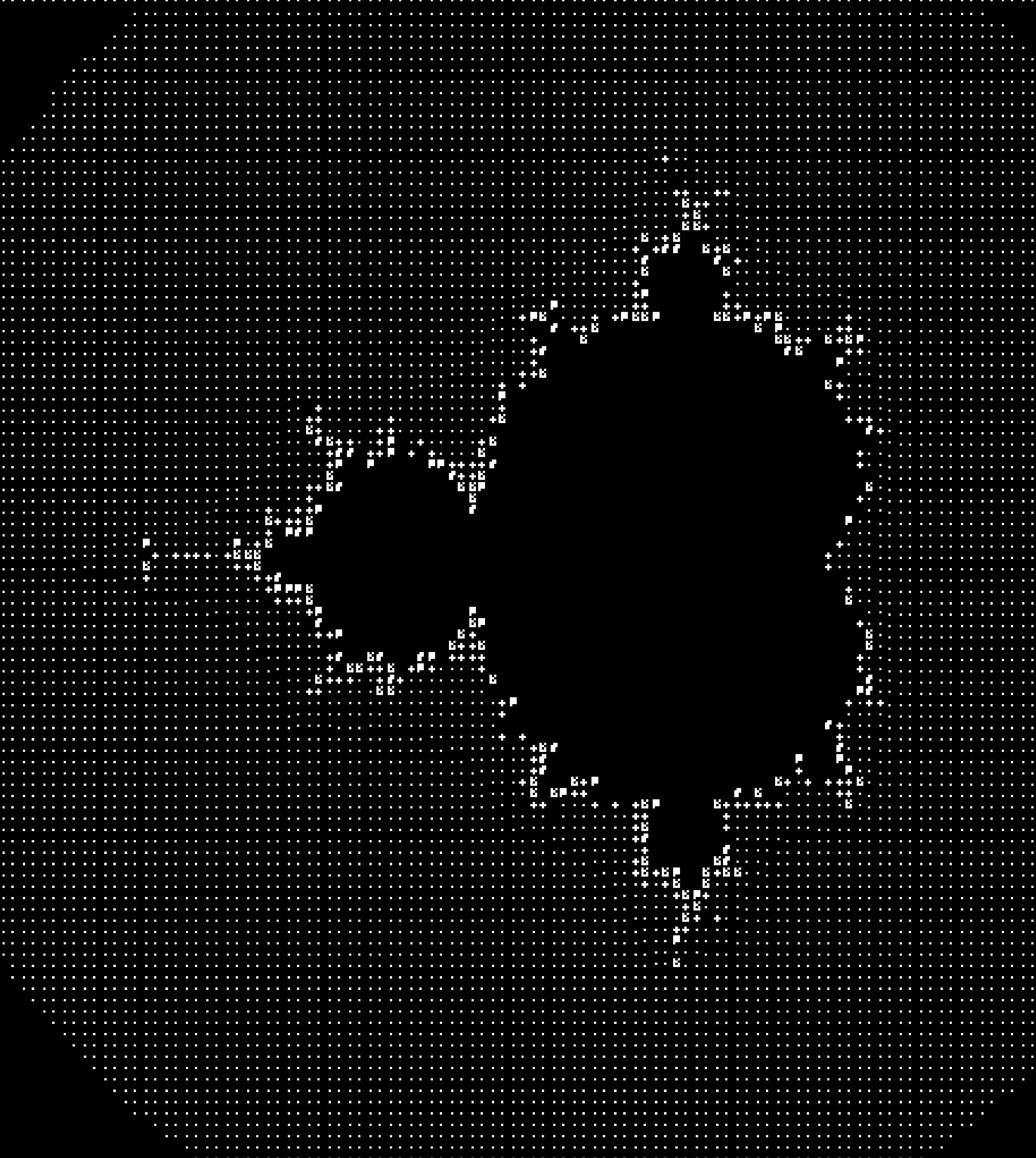
Comments:____

____

____

____

CONFIDENTIAL


array_to_string
(102 rows)

Better, Faster TPS Reports

New!

Reach Outside the Current Row

Better, Faster TPS Reports

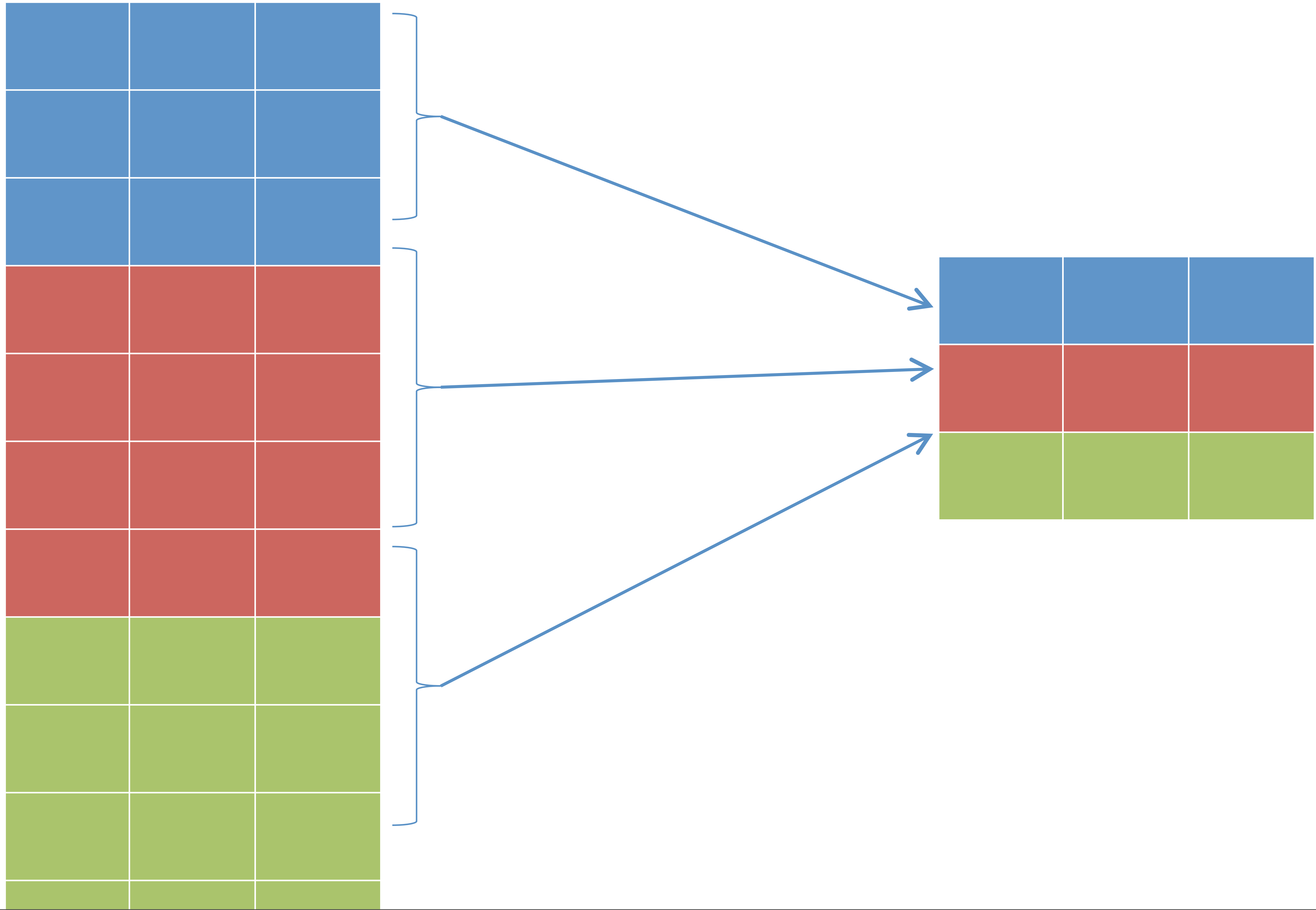
- Windowing Function
  - Operates on a window
  - Returns a value for each row
  - Calculates value from the rows in the window

Better, Faster TPS Reports

- You can use…
  - New window functions
  - Existing aggregate functions
  - User-defined window functions
  - User-defined aggregate functions

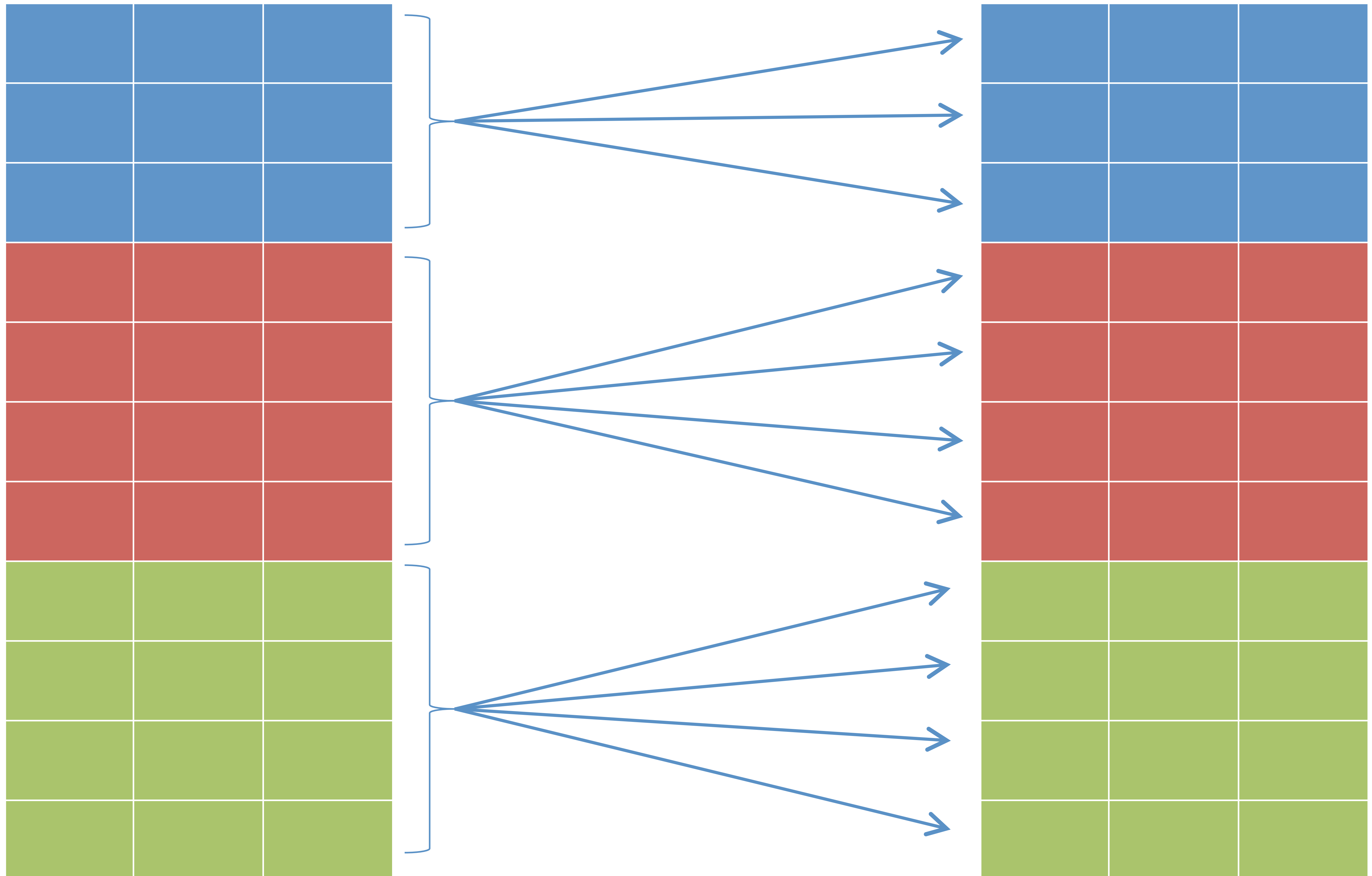
Better, Faster TPS Reports

[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)

OOPS!

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

ROW_NUMBER (After)

```
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)
```

More Ranking

```
SELECT
    empno,
    depname,
    salary,
    row_number() OVER (
        ORDER BY salary DESC NULLS LAST
    ),
    rank() OVER (
        ORDER BY salary DESC NULLS LAST
    ),
    dense_rank() OVER (
        ORDER BY salary DESC NULLS LAST
    )
FROM
    empsalary
ORDER BY salary DESC;
```

More Ranking

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

Built-in Windowing Functions

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

row_number()

- Returns number of the current row

```
SELECT val, row_number() OVER (ORDER BY val DESC) FROM tbl;
```

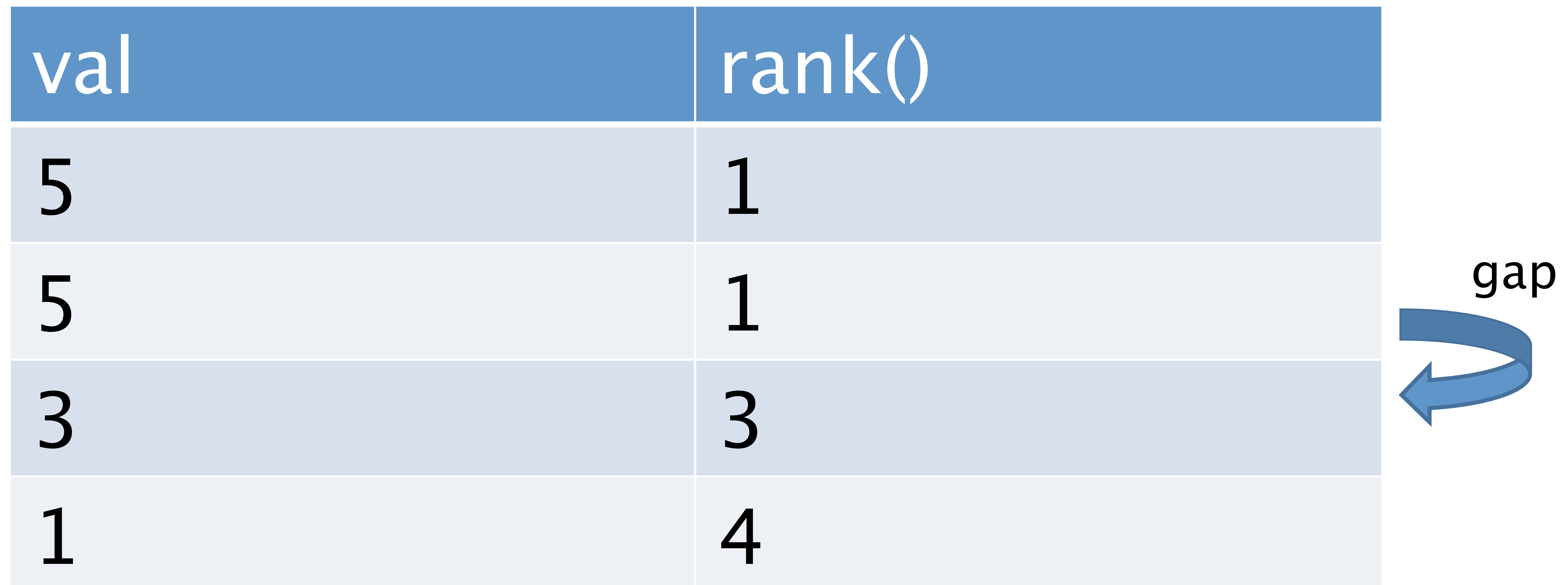
val	row_number()
5	1
5	2
3	3
1	4

Note: row_number() always incremented values independent of frame

rank()

- Returns rank of the current row with gap

SELECT val, rank() OVER (ORDER BY val DESC) FROM tbl;



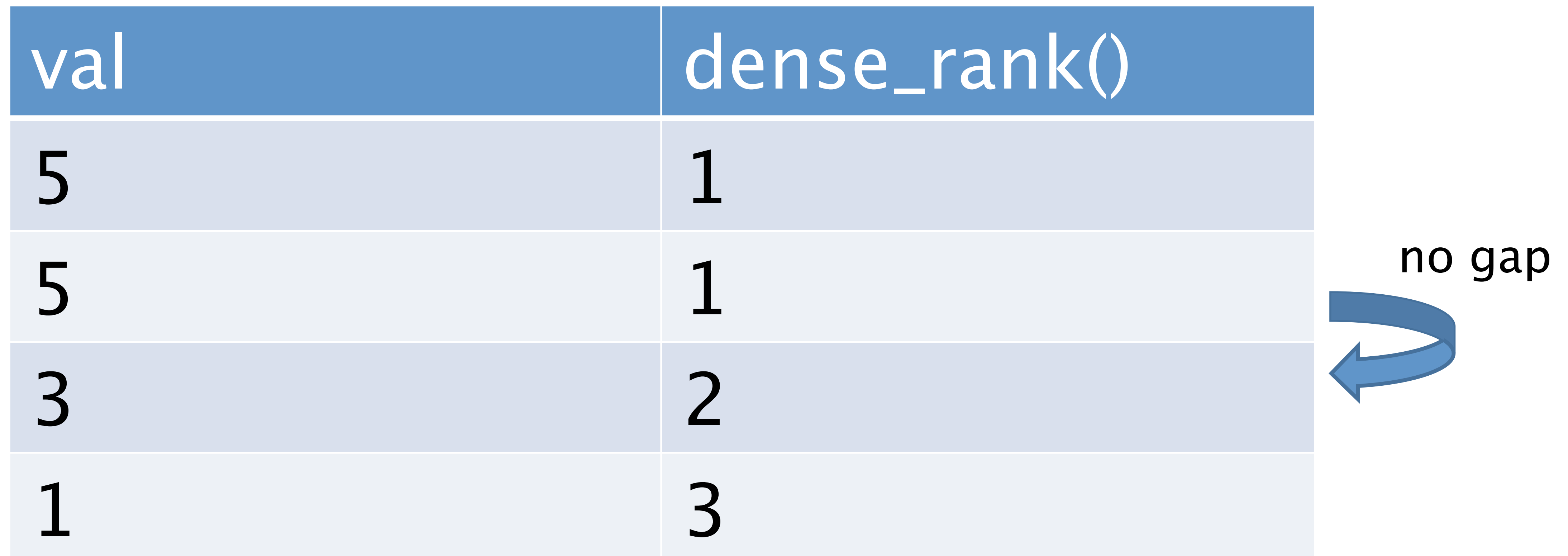
val	rank()
5	1
5	1
3	3
1	4

Note: rank() OVER(*empty*) returns 1 for all rows, since all rows are peers to each other

dense_rank()

- Returns rank of the current row without gap

```
SELECT val, dense_rank() OVER (ORDER BY val DESC) FROM tbl;
```



val	dense_rank()
5	1
5	1
3	2
1	3

Note: dense_rank() OVER(*empty*) returns 1 for all rows, since all rows are peers to each other

percent_rank()

- Returns relative rank; (rank() – 1) / (total row – 1)

```
SELECT val, percent_rank() OVER (ORDER BY val DESC) FROM tbl;
```

val	percent_rank()
5	0
5	0
3	0.666666666666667
1	1

values are always between 0 and 1 inclusive.

Note: percent_rank() OVER(*empty*) returns 0 for all rows, since all rows are peers to each other

cume_dist()

- Returns relative rank; (# of preced. or peers) / (total row)

```
SELECT val, cume_dist() OVER (ORDER BY val DESC) FROM tbl;
```

val	cume_dist()	
5	0.5	= 2 / 4
5	0.5	= 2 / 4
3	0.75	= 3 / 4
1	1	= 4 / 4

The result can be emulated by
“count(*) OVER (ORDER BY val DESC) / count(*) OVER ()”

Note: cume_dist() OVER(*empty*) returns 1 for all rows, since all rows are peers to each other

ntile()

- Returns dividing bucket number

```
SELECT val, ntile(3) OVER (ORDER BY val DESC) FROM tbl;
```

val	ntile(3)
5	1
5	1
3	2
1	3

4 % 3 = 1

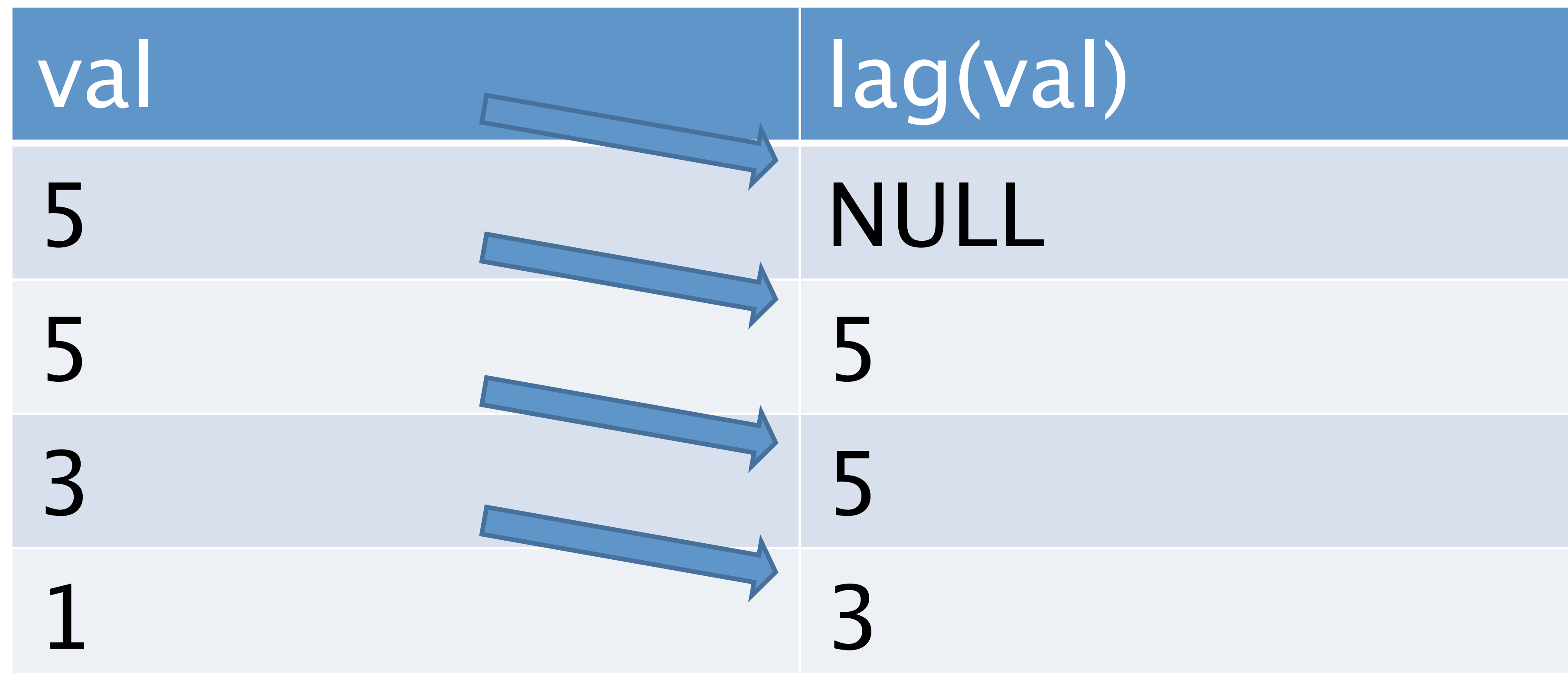
The results are the divided positions, but if there's remainder add row from the head

Note: ntile() OVER (*empty*) returns same values as above, since ntile() doesn't care the frame but works against the partition

lag()

- Returns value of row above

SELECT val, lag(val) OVER (ORDER BY val DESC) FROM tbl;



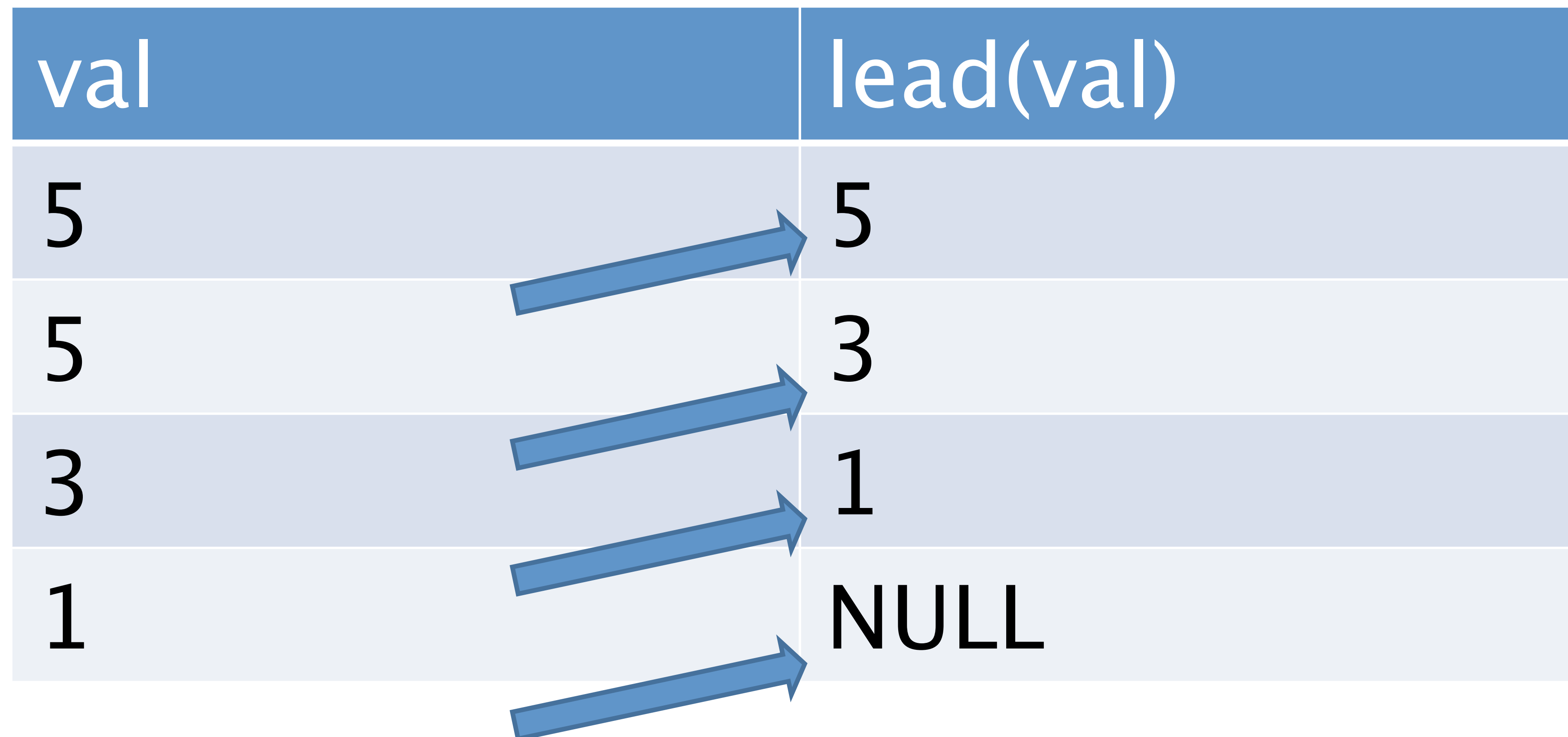
val	lag(val)
5	NULL
5	5
3	5
1	3

Note: lag() only acts on a partition.

lead()

- Returns value of the row below

```
SELECT val, lead(val) OVER (ORDER BY val DESC) FROM tbl;
```



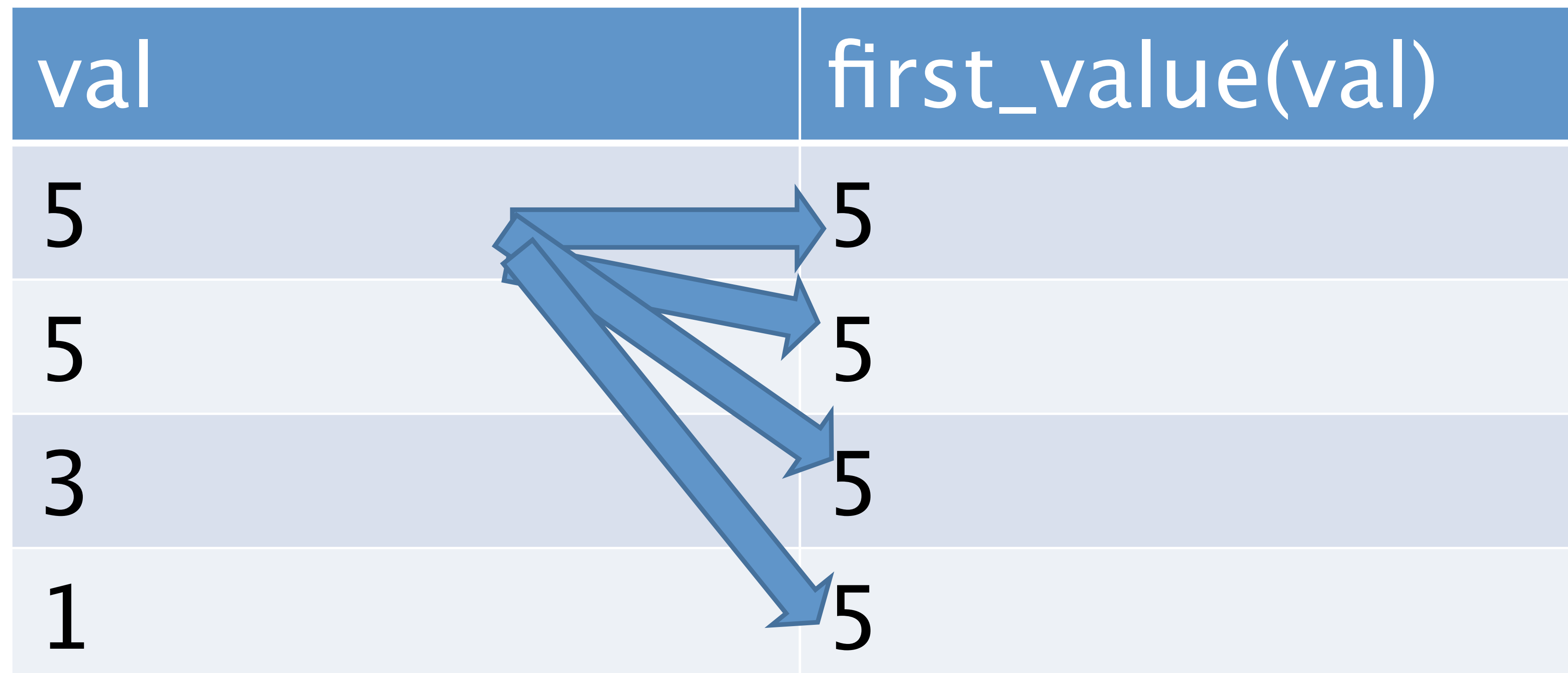
val	lead(val)
5	5
5	3
3	1
1	NULL

Note: lead() acts against a partition.

first_value()

- Returns the first value of the frame

```
SELECT val, first_value(val) OVER (ORDER BY val DESC) FROM tbl;
```

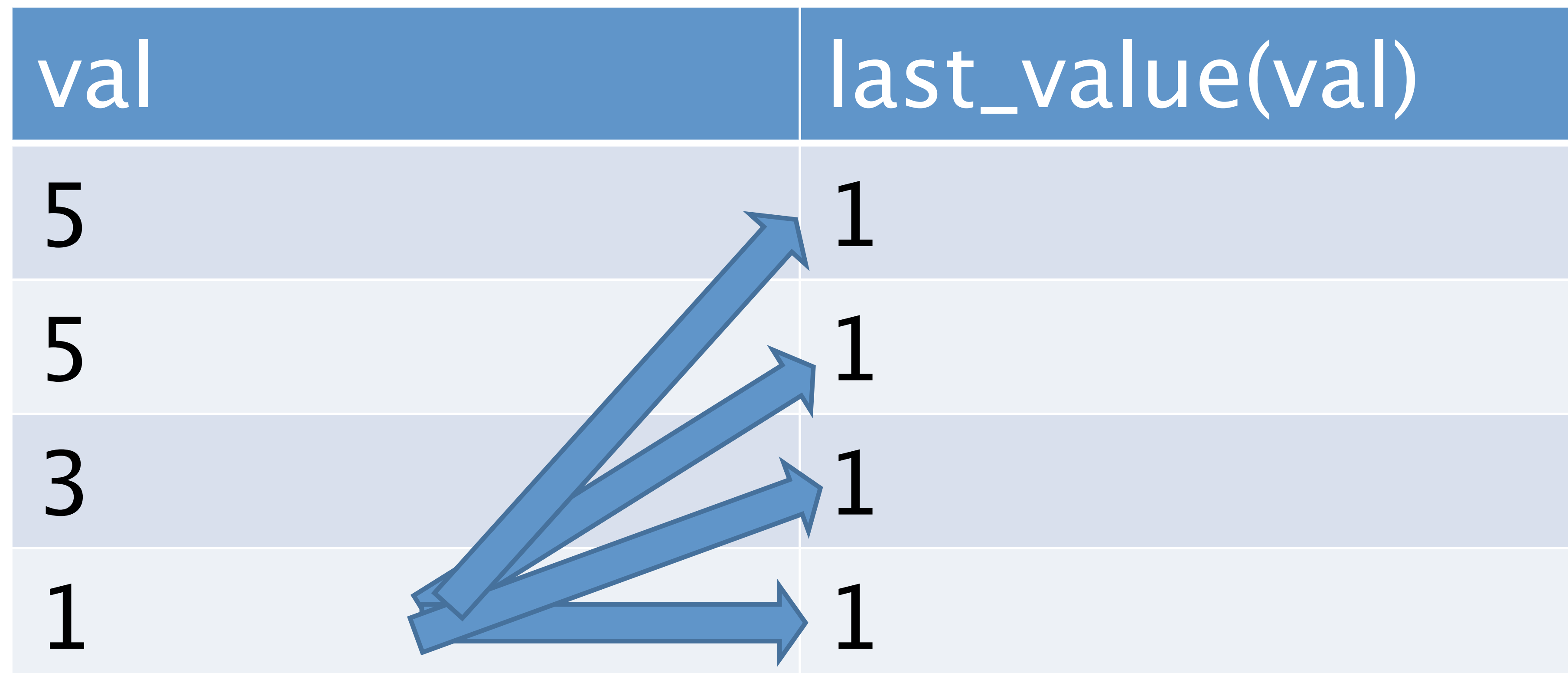


val	first_value(val)
5	5
5	5
3	5
1	5

last_value()

- Returns the last value of the frame

```
SELECT val, last_value(val) OVER
 (ORDER BY val DESC ROWS BETWEEN UNBOUNDED PRECEEDING
AND UNBOUNDED FOLLOWING) FROM tbl;
```



val	last_value(val)
5	1
5	1
3	1
1	1

Note: frame clause is necessary since you have a frame between the first row and the current row by only the order clause

nth_value()

- Returns the n-th value of the frame

```
SELECT val, nth_value(val, val) OVER
 (ORDER BY val DESC ROWS BETWEEN UNBOUNDED PRECEEDING
AND UNBOUNDED FOLLOWING) FROM tbl;
```

val	nth_value(val, val)
5	NULL
5	NULL
3	3
1	5

Note: frame clause is necessary since you have a frame between the first row and the current row by only the order clause

aggregates(all peers)

- Returns the same values along the frame

```
SELECT val, sum(val) OVER () FROM tbl;
```

val	sum(val)
5	14
5	14
3	14
1	14

Note: all rows are the peers to each other

cumulative aggregates

- Returns different values along the frame

```
SELECT val, sum(val) OVER (ORDER BY val DESC) FROM tbl;
```

val	sum(val)
5	10
5	10
3	13
1	14

Note: row#1 and row#2 return the same value since they are the peers.
the result of row#3 is sum(val of row#1…#3)

recursion – Google Search

http://www.google.com.au/search?q=recursion&ie=utf-8&oe=utf-8&aq=t&rls=org.

/. local pg Pg Current PgFoundry Planet Pg Pg PG TPM C&L Kos TSG xkcd.com

recursion – Google Search

Web Images Videos Maps News Books Gmail more ▼ Search settings | Si

recursion Search Advanced Search

Search: the web pages from Australia

Web Show options... Results **1 - 10** of about **715,000** for **recursion** [definition]. (**0.08** seconds)

Did you mean: ***recursion***

Recursion - Wikipedia, the free encyclopedia
A visual form of **recursion** known as the Droste effect. The woman in this image is holding an object which contains a smaller image of her holding the same **...**
en.wikipedia.org/wiki/**Recursion** - Cached - Similar

Recursion (computer science) - Wikipedia, the free encyclopedia
Recursion in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem. **...**
en.wikipedia.org/wiki/**Recursion**_(computer_science) - Cached - Similar

Show more results from en.wikipedia.org

Recursion -- from Wolfram MathWorld
A **recursive** process is one in which objects are defined in terms of other objects of the same type. Using some sort of recurrence relation, the entire class **...**
mathworld.wolfram.com › ... › Algorithms › Recursion - Cached - Similar

recursion
Definition of **recursion**, possibly with links to more information and implementations.
www.itl.nist.gov/div897/sqg/dads/HTML/**recursion**.html - Cached - Similar

Did you mean **recursion**?
23 Jul 2009 **...** Bloodwine, on 07/23/2009, -8/+206Yo Dawg I herd you like **recursion** so we put **recursion** in yo **recursion** so you can repeat while you repeat **...**

Find: Next Previous Highlight all Match case

Done

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)
AS (
    SELECT Ix, Iy,
           X::float, Y::float,
           X::float, Y::float,
           0
    FROM
```

Generate Points

```
    (SELECT -2.2 + 0.031 * i, i
     FROM x) AS xgen(x,ix)
CROSS JOIN
    (SELECT -1.5 + 0.031 * i, i
     FROM x) 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
),
```

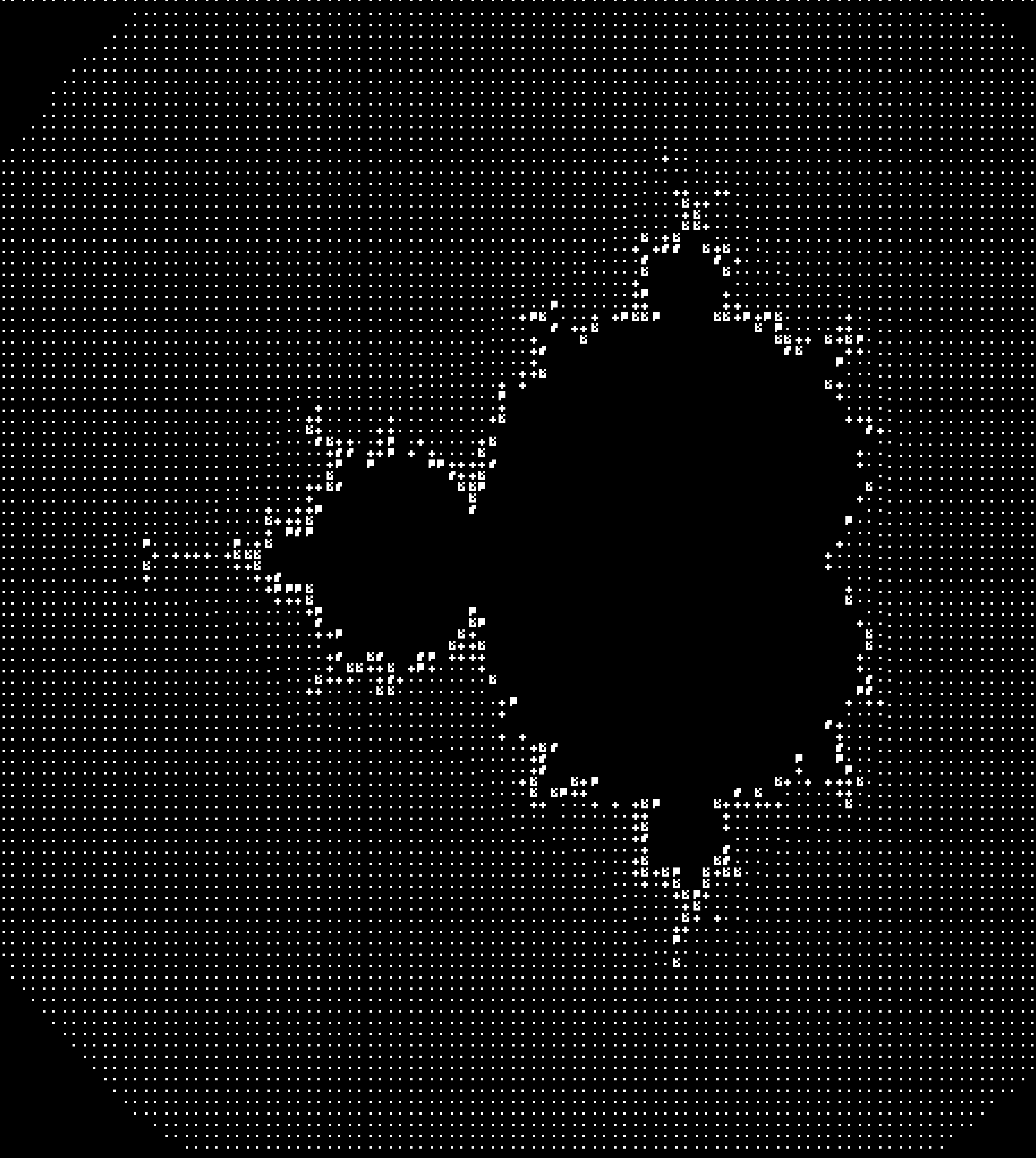
Choose Some

```
Zt (Ix, Iy, I) AS (
    SELECT Ix, Iy, MAX(I) AS I
    FROM Z
    GROUP BY Iy, Ix
    ORDER BY Iy, Ix
)
```

Display Them

```
SELECT array_to_string(
  array_agg(
    SUBSTRING(
      ' .,,,-----++++%%%%@@@@#### ',
      GREATEST(I,1)
  ),''
)
FROM Zt
GROUP BY Iy
ORDER BY Iy;
```

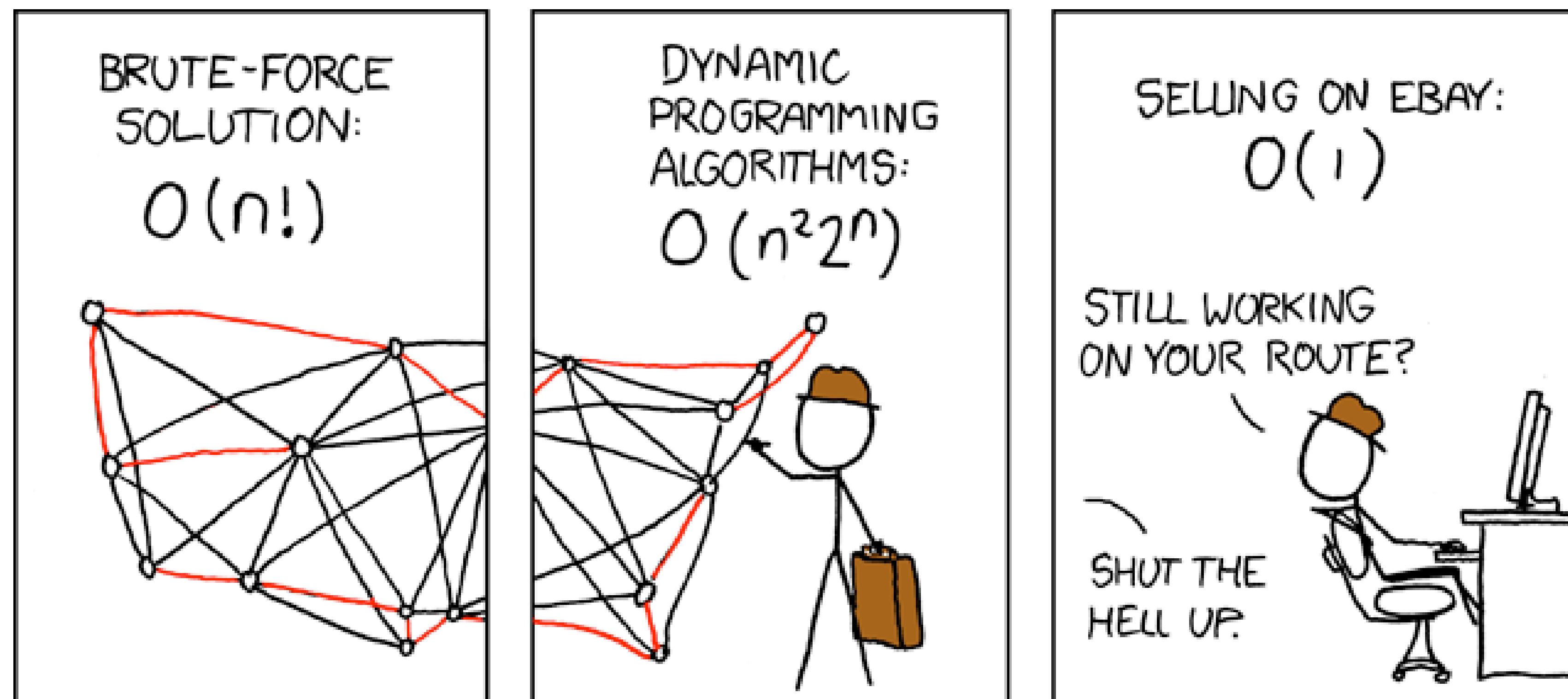
array to string



(102 rows)

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 */
    )
)
```

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?

Who

```
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');
```

Posts

```
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 :)

Find the fr1st ps0t

```
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

```
  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 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 :)

```
 thread_id |     user_name     | reply_count
-----------+-------------------+-------------
         1 | Tom Lane          |           9
         1 | Gregory Stark     |           9
        82 | Magnus Hagander   |           5
       108 | Dave Page         |           4
         9 | Josh Berkus       |           4
(5 rows)
```

OBTW

`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.

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:

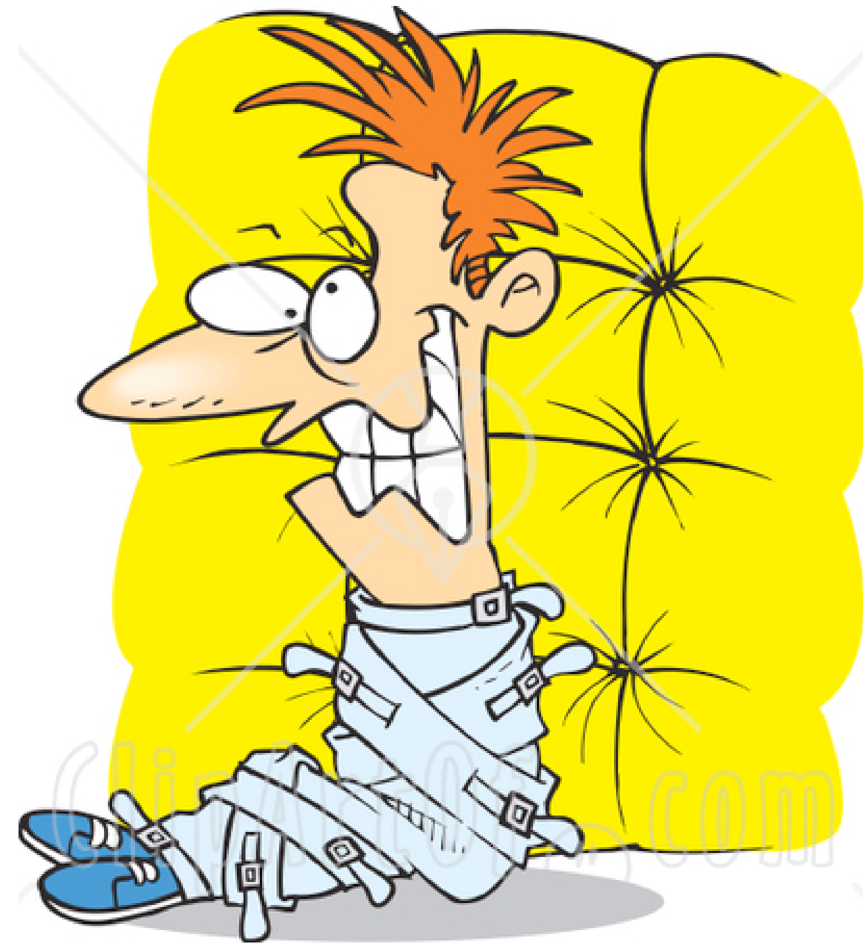
```
r(iter,rnum,tag) AS (
    VALUES (0,0,1)
UNION ALL
    SELECT r.iter+1,
           CASE
               WHEN r.rnum=0 THEN p.rnum + max(r.rnum) OVER ()
               ELSE r.rnum-1
           END,
           CASE
               WHEN r.rnum=0 THEN p.tag
               ELSE r.tag
           END
    FROM
        r
    LEFT JOIN p
        ON (r.rnum=0 and r.tag=1 and p.iter=(r.iter % 3))
    WHERE
        r.rnum>0
    OR p.iter IS NOT NULL
)
```

Proof:

```
SELECT iter, rnum, tag
FROM r
ORDER BY iter, rnum;
```

Thanks

Andrew (RhodiumToad) Gierth



Questions?
Comments?
Straitjackets?

Thank You!

Copyright © 2010
David Fetter david.fetter@pgexperts.com
All Rights Reserved

PGX
POSTGRESQL
EXPERTS, INC.