

Trees and More in SQL

Common Table Expressions
FOSDEM 2009

Copyright © 2009
David Fetter david@fetter.org
All Rights Reserved

Some Approaches

- * External Processing
- * Functions and Stored Procedures
- * Materialized Path Enumeration
- * Nested Set

External Processing

- * Pros:

- * Lots of Language Choices

- * Cons:

- * Scaling

- * No Relational Operators

Functions and Stored Procedures

- * Pros:

- * Easy to visualize

- * Easy to write

- * Cons:

- * Scaling

- * Hard to do relational operators

Materialized Path Enumeration

- * Pros:

- * Easy to visualize

- * Cons:

- * DDL for each hierarchy

- * Full-table changes on write

- * Hard to do relational operators

Nested Sets

- * Pros:

- * Easy to manipulate on SELECT

- * Sounds cool

- * Endorsed by a famous bald guy



- * Cons:

- * DDL for each hierarchy

- * Full-table changes on write

- * Hard to do relational operators

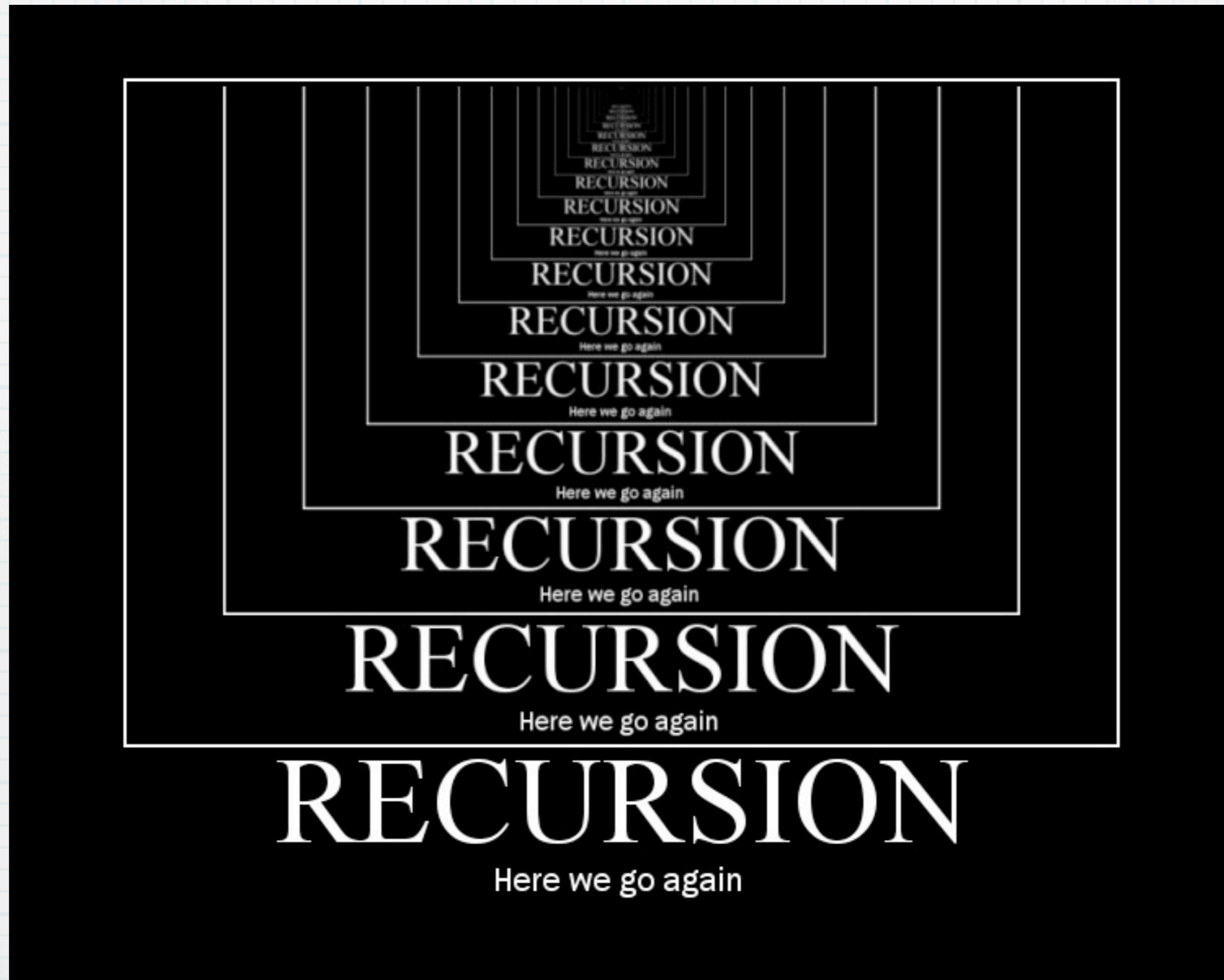
Thanks to:

- * The ISO SQL Standards Committee
- * Yoshiyuki Asaba
- * Ishii Tatsuo
- * Jeff Davis
- * Gregory Stark
- * Tom Lane
- * etc., etc., etc.

SQL Standard

*** Common Table Expressions (CTE)**

Recursion



Recursion in General

- *Initial Condition

- *Recursion step

- *Termination condition

El List Table

```
CREATE TABLE employee(  
    id INTEGER NOT NULL,  
    boss_id INTEGER,  
    UNIQUE(id, boss_id)/*, etc., etc. */  
);
```

```
INSERT INTO employee(id, boss_id)  
VALUES(1, NULL), /* El capo di tutti capi */  
(2, 1), (3, 1), (4, 1),  
(5, 2), (6, 2), (7, 2), (8, 3), (9, 3), (10, 4),  
(11, 5), (12, 5), (13, 6), (14, 7), (15, 8),  
(1, 9);
```

Tree Query Initiation

```
WITH RECURSIVE t(node, path) AS (  
    SELECT id, ARRAY[id] FROM employee WHERE boss_id IS NULL  
    /* Initiation Step */  
UNION ALL  
    SELECT e1.id, t.path || ARRAY[e1.id]  
    FROM employee e1 JOIN t ON (e1.boss_id = t.node)  
    WHERE id NOT IN (t.path)  
)  
SELECT  
    CASE WHEN array_upper(path,1)>1 THEN '+-' ELSE ' ' END ||  
    REPEAT('--', array_upper(path,1)-2) ||  
    node AS "Branch"  
FROM t  
ORDER BY path;
```

Tree Query Recursion

```
WITH RECURSIVE t(node, path) AS (  
    SELECT id, ARRAY[id] FROM employee WHERE boss_id IS NULL  
UNION ALL  
    SELECT e1.id, t.path || ARRAY[e1.id] /* Recursion */  
    FROM employee e1 JOIN t ON (e1.boss_id = t.node)  
    WHERE id NOT IN (t.path)  
)  
SELECT  
    CASE WHEN array_upper(path,1)>1 THEN '+-' ELSE '' END ||  
    REPEAT('--', array_upper(path,1)-2) ||  
    node AS "Branch"  
FROM t  
ORDER BY path;
```

Tree Query Termination

```
WITH RECURSIVE t(node, path) AS (  
    SELECT id, ARRAY[id] FROM employee WHERE boss_id IS NULL  
UNION ALL  
    SELECT e1.id, t.path || ARRAY[e1.id]  
    FROM employee e1 JOIN t ON (e1.boss_id = t.node)  
    WHERE id NOT IN (t.path) /* Termination Condition */  
)  
SELECT  
    CASE WHEN array_upper(path,1)>1 THEN '+-' ELSE '' END ||  
    REPEAT('--', array_upper(path,1)-2) ||  
    node AS "Branch"  
FROM t  
ORDER BY path;
```

Tree Query Display

```
WITH RECURSIVE t(node, path) AS (  
    SELECT id, ARRAY[id] FROM employee WHERE boss_id IS NULL  
UNION ALL  
    SELECT e1.id, t.path || ARRAY[e1.id]  
    FROM employee e1 JOIN t ON (e1.boss_id = t.node)  
    WHERE id NOT IN (t.path)  
)  
SELECT  
    CASE WHEN array_upper(path,1)>1 THEN '+-' ELSE '' END ||  
    REPEAT('--', array_upper(path,1)-2) ||  
    node AS "Branch" /* Display */  
FROM t  
ORDER BY path;
```

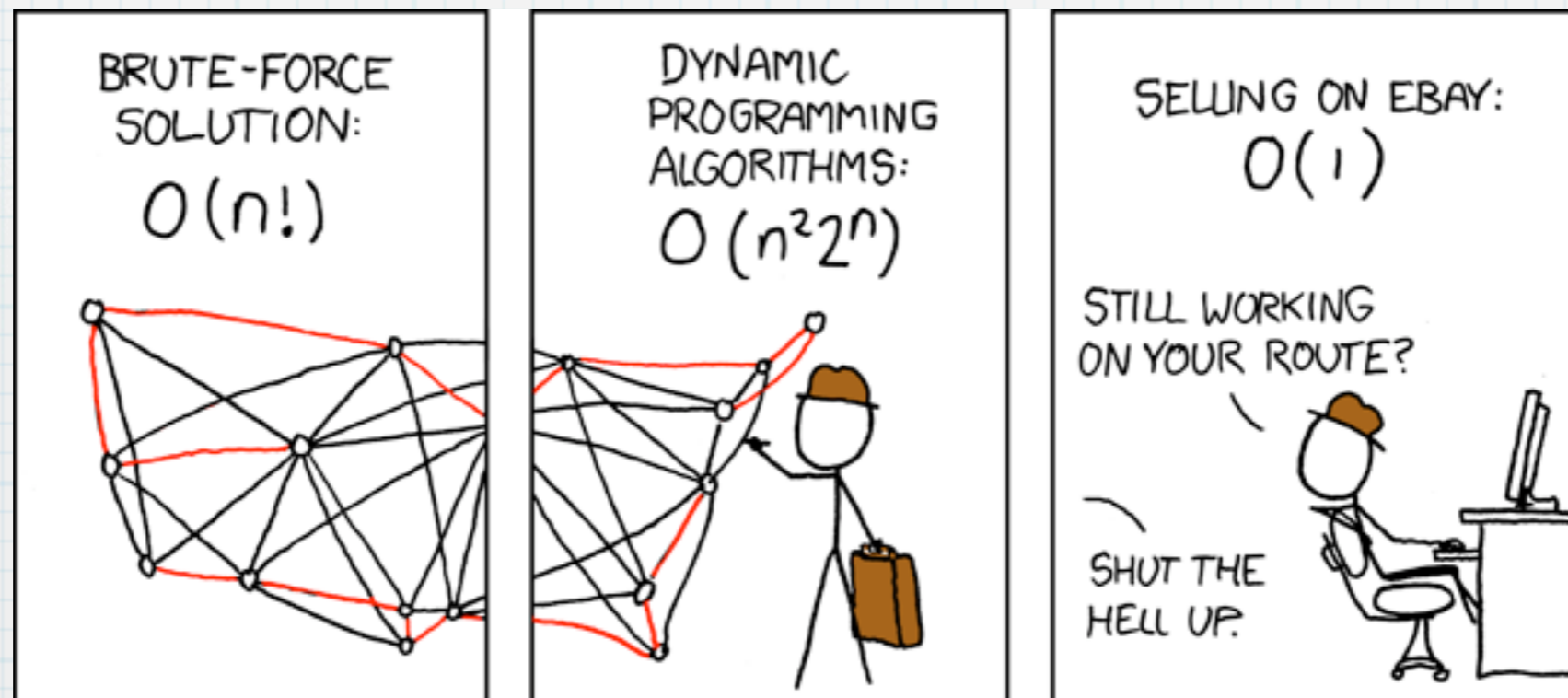
Tree Query Initiation

Branch

1
+-2
+---5
+-----11
+-----12
+---6
+-----13
+---7
+-----14
+-3
+---8
+-----15
+---9
+-4
+---10
+-9
(16 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
```

Hausdorff-Besicovich- WTF Dimension

```
WITH RECURSIVE
Z(Ix, Iy, Cx, Cy, X, Y, I)
AS (
    SELECT Ix, Iy, X::float, Y::float, X::float, Y::float, 0
    FROM
        (SELECT -2.2 + 0.031 * i, i FROM generate_series(0,101) AS i) AS xgen(x,ix)
    CROSS JOIN
        (SELECT -1.5 + 0.031 * i, i FROM generate_series(0,101) AS i) AS ygen(y,iy)
    UNION ALL
    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::float
    AND I < 100
),
```

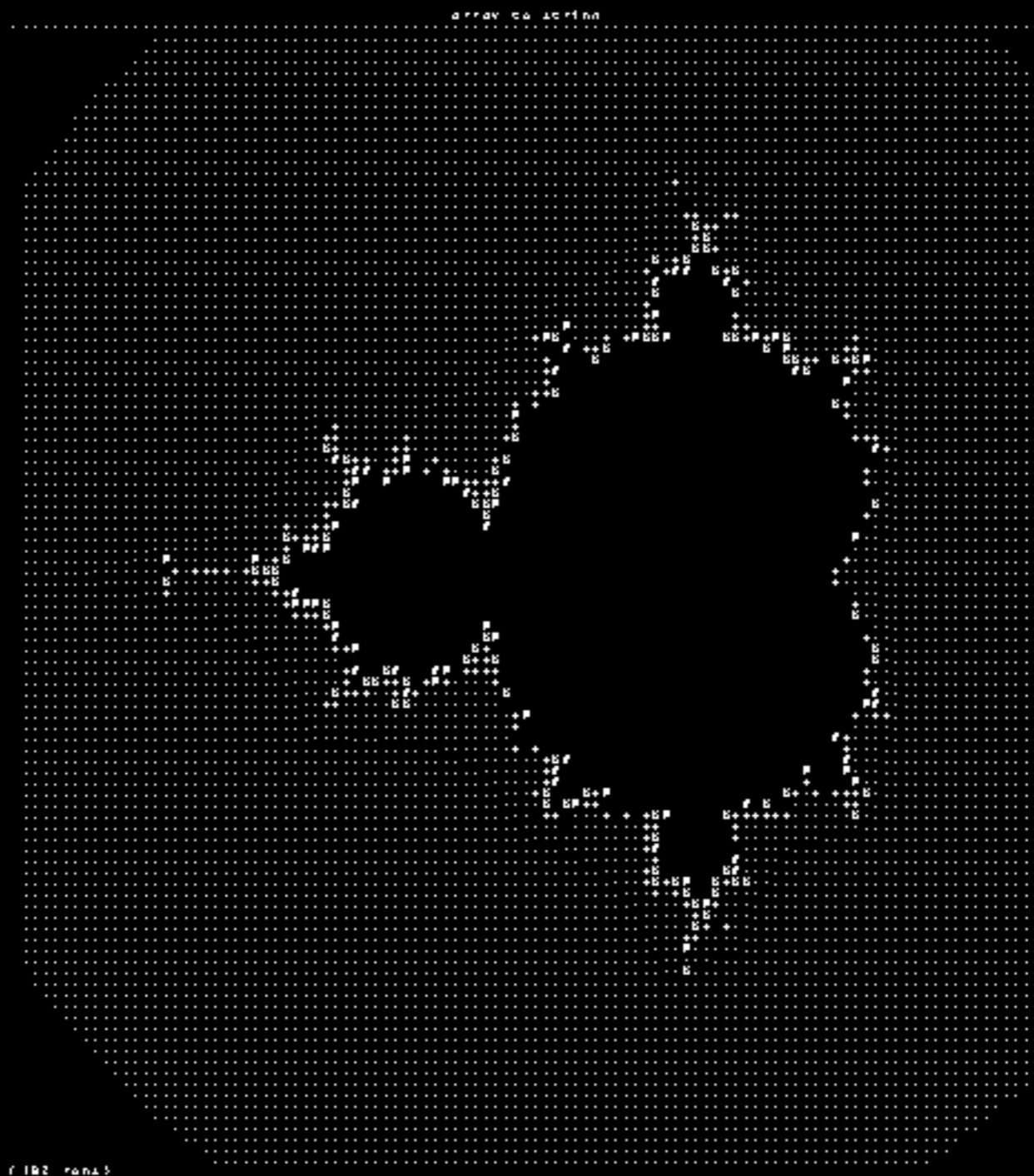
Filter

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

Display

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

Return Map



Questions?
Comments?
Straitjackets?



Merci!

Copyright © 2009
David Fetter david@fetter.org
All Rights Reserved