Data-Modifying CTEs:
A New Programming Paradigm

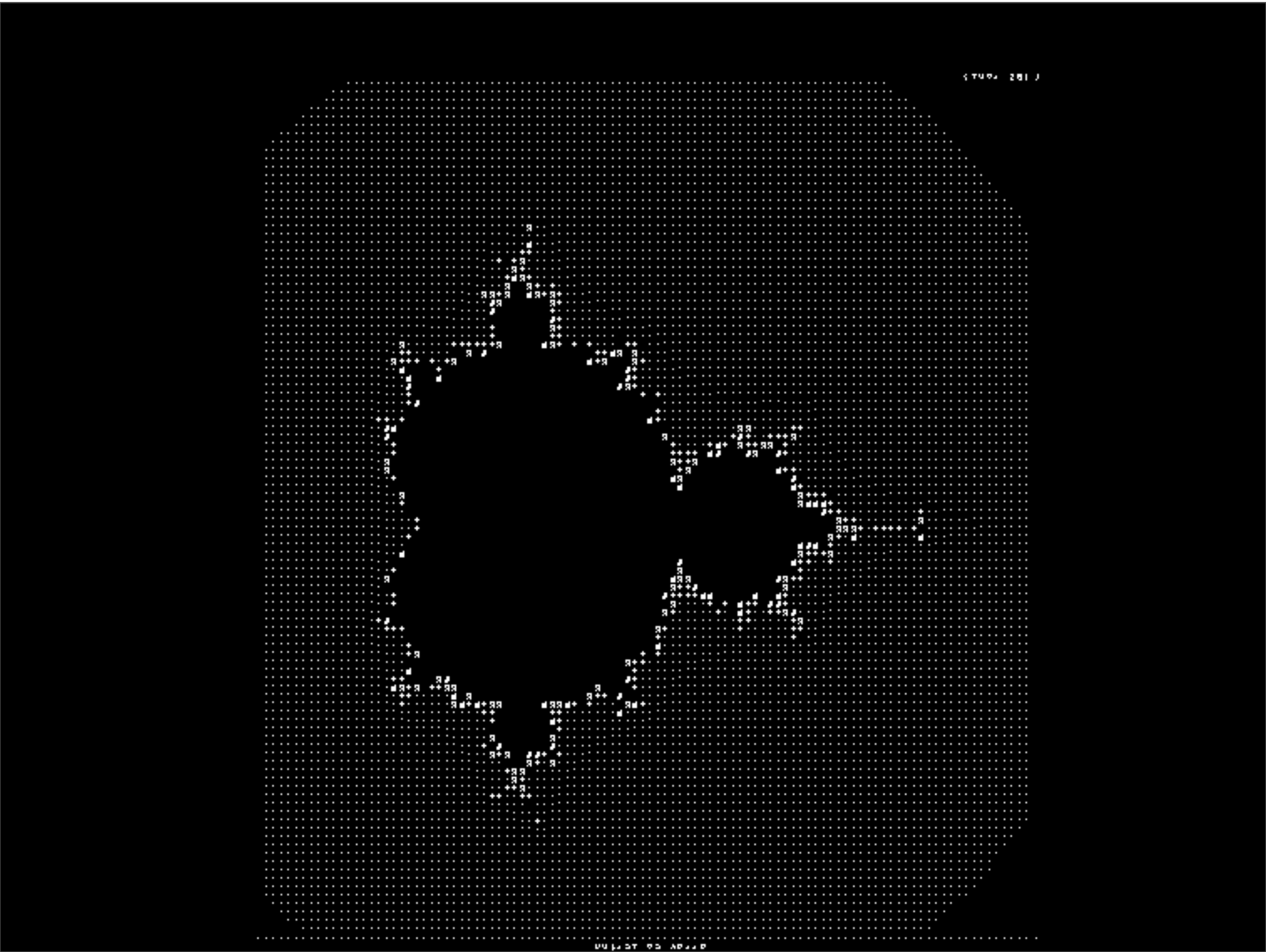
PostgreSQL China
July 16, 2011

David Fetter dfetter@vmware.com

Current CTEs

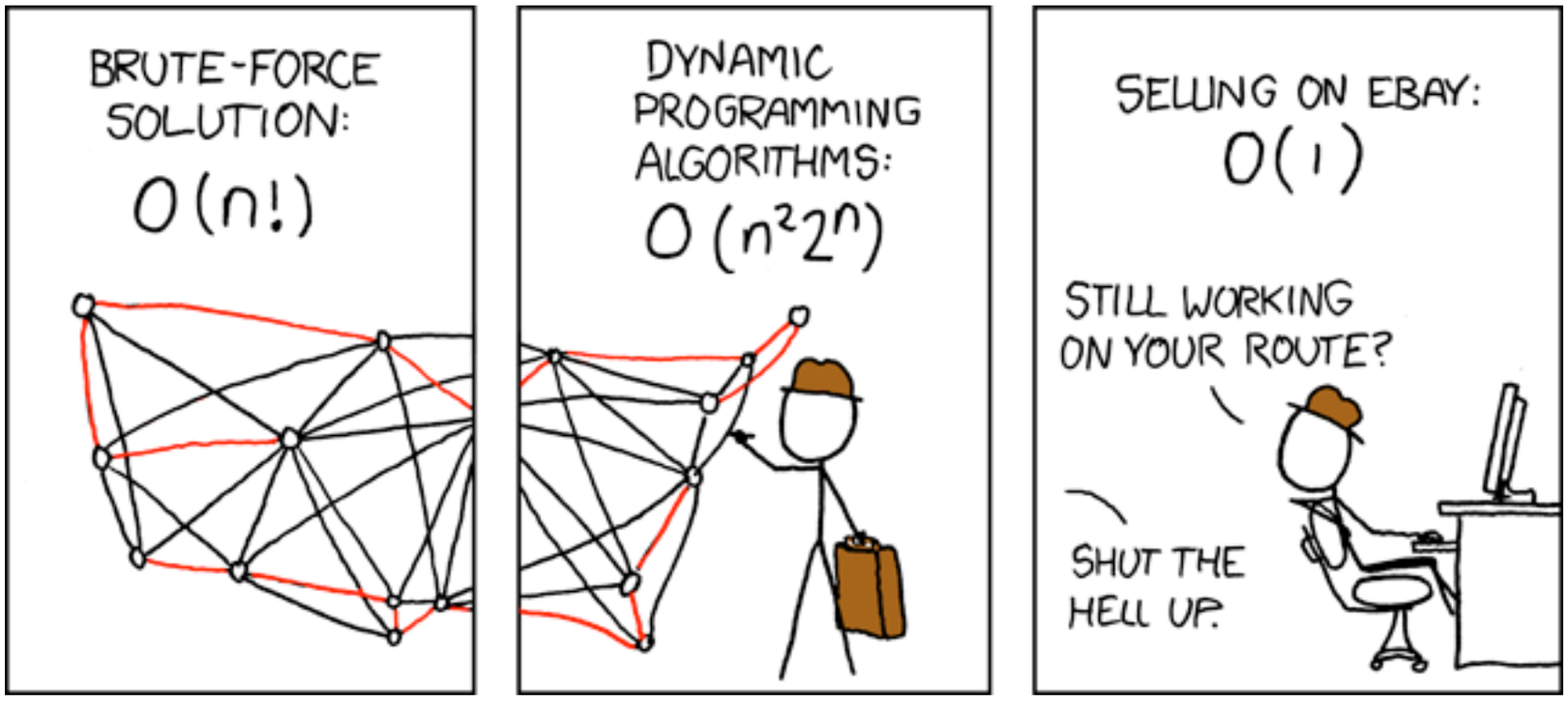
```
WITH [RECURSIVE] t1 [(column type,…)] AS
(
    [SELECT | VALUES]
[UNION [ALL]
    [SELECT]
),
t2 AS…tn AS…
SELECT…
```

11年7月30日星期六



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?



OBTW

With CTE and Windowing, SQL is Turing Complete.

What Didn't the Old Syntax Do?

WRITE!

```
WITH [RECURSIVE] t1 [(column type,…)] AS
(
     [SELECT | VALUES |
    (INSERT | UPDATE | DELETE) [RETURNING]]
[UNION [ALL]
     [SELECT | VALUES |
    (INSERT | UPDATE | DELETE) [RETURNING]]
)
(SELECT | INSERT | UPDATE | DELETE)    …
```

For 9.1:
Simple Partition Management

```
CREATE TABLE log (
        ts TIMESTAMPTZ NOT NULL,
        msg TEXT NOT NULL
);
```

For 9.1:
Simple Partition Management

```
CREATE TABLE log_201106 ()
INHERITS(log);
```

```
ALTER TABLE log_201106 ADD
CONSTRAINT right_month CHECK(
    ts >= '2011-06-01' AND
    ts <  '2011-07-01');
```

For 9.1:
Simple Partition Management

```
happiness@pg_cn_2011:54321# WITH t1 AS (
    DELETE FROM ONLY log
    WHERE ts >='2011-06-01' AND ts < '2011-07-01'
    RETURNING *
)
INSERT INTO log_201101 SELECT * FROM t1;
INSERT 0 45270
```

Query Clustering: I/O Minimization

```
CREATE TABLE person (
    id SERIAL PRIMARY KEY,
    first_name TEXT,
    last_name TEXT,
    CHECK (CASE WHEN first_name IS NULL THEN 0 ELSE 1 END +
           CASE WHEN last_name IS NULL THEN 0 ELSE 1 END >= 1),
    birthdate DATE NOT NULL,
    gender TEXT
);
```

Query Clustering: I/O Minimization

```
CREATE TABLE im (
    id SERIAL PRIMARY KEY,
    provider TEXT NOT NULL, /* should be fk */
    handle TEXT NOT NULL
);
```

Query Clustering: I/O Minimization

```
CREATE TABLE phone (
    id SERIAL PRIMARY KEY,
    country_code TEXT NOT NULL,
    phone_number TEXT NOT NULL,
    extension TEXT
);
```

Query Clustering: I/O Minimization

```
CREATE TABLE street (
        id SERIAL PRIMARY KEY,
        street1 TEXT NOT NULL,
        street2 TEXT,
        street3 TEXT,
        city TEXT NOT NULL,
        state TEXT,
        country TEXT NOT NULL,
        post_code TEXT
    );
```

Query Clustering: I/O Minimization

```
CREATE TABLE person_im (
    person_id INTEGER NOT NULL REFERENCES person (id),
    im_id INTEGER NOT NULL REFERENCES im (id),
    UNIQUE (person_id, im_id)
);

CREATE TABLE person_phone (
    person_id INTEGER NOT NULL REFERENCES person (id),
    phone_id INTEGER NOT NULL REFERENCES phone (id),
    UNIQUE (person_id, phone_id)
);

CREATE TABLE person_street (
    person_id INTEGER NOT NULL REFERENCES person (id),
    street_id INTEGER NOT NULL REFERENCES street (id),
    UNIQUE (person_id, street_id)
);
```

Query Clustering: I/O Minimization

```
WITH t_person AS (
    INSERT INTO person (first_name, last_name)
    VALUES ('David', 'Fetter')
    RETURNING id
),
```

Query Clustering: I/O Minimization

```
t_im AS (
    INSERT INTO im (provider, handle)
    VALUES
        ('Yahoo!', 'dfetter'),
        ('AIM', 'dfetter666'),
        ('XMPP', 'davidfetter@postgresql.org')
    RETURNING id
),
t_person_im AS (
    INSERT INTO person_im
    SELECT * FROM t_person CROSS JOIN t_im
),
```

Query Clustering: I/O Minimization

```
t_phone (phone_id) AS (
    INSERT INTO phone (country_code, phone_number)
    VALUES
        ('+1','415 235 3778'),
        ('+1','650 427 3751')
    RETURNING id
),
t_person_phone AS (
    INSERT INTO person_phone
    SELECT * FROM t_person CROSS JOIN t_phone
),
```

Query Clustering: I/O Minimization

```
t_street AS (
    INSERT INTO street (street1, city, state, country, post_code)
    VALUES
        ('904 N Humboldt St','San Mateo','California','USA','94401-1130'),
        ('3421 Hillview Avenue', 'Palo Alto','California','USA','94304')
    RETURNING id
),
t_person_street AS (
    INSERT INTO person_street
    SELECT * FROM t_person CROSS JOIN t_street
)
```

Query Clustering: I/O Minimization

```
VALUES(true);
```

Query Clustering: Transaction Management

```
CREATE TABLE foo (
    id SERIAL PRIMARY KEY,
    bar_id INTEGER NOT NULL
);

CREATE TABLE bar (
    id SERIAL PRIMARY KEY,
    foo_id INTEGER NOT NULL REFERENCES foo(id)
                            ON DELETE CASCADE
                            INITIALLY DEFERRED
);

ALTER TABLE foo ADD FOREIGN KEY (bar_id) REFERENCES bar(id)
                                    ON DELETE CASCADE
                                    INITIALLY DEFERRED;
```

Query Clustering: Transaction Management

```
WITH t AS
(
   INSERT INTO foo(id, bar_id)
   VALUES(
      DEFAULT,
      nextval(pg_get_serial_sequence('bar', 'id'))
   )
   RETURNING id AS foo_id, bar_id
)
INSERT INTO bar(id,foo_id)
SELECT bar_id, foo_id FROM t RETURNING *;
```

Materialized Views

```
WITH t AS (
    DELETE FROM foo
    RETURNING a, b
)
INSERT INTO mv_foo
SELECT a, SUM(b)
FROM t GROUP BY a
```

Just Cool Stuff™

```
WITH t AS (
    INSERT INTO foo
    VALUES (...),...,(...)
    RETURNING a
)
SELECT bar.a, bar.b
FROM bar JOIN t USING(a)
```

Head-Scratcher: SNAPSHOTS

```
WITH t AS (DELETE FROM foo)
SELECT * FROM foo;
```

(0 rows)?
(100123 rows)?

How'd He Do That?!?

First try: David digs into the grammar and gets cut a few times.

How'd He Do That?!?

First try: Marko reworks the planner. It needs to know when it creates a ModifyTable node. These used to have another name.

How'd He Do That?!?

First try: Marko reworks the executor. It needs new nodes. Mmmm...nodes.

How'd He Do That?!?

Marko reworks the executor, Part II:
Copy & Paste. Now it's getting ugly...

How'd He Do That?!?

Jaime Casanova, Tom Lane, and Robert Haas look at the reworked executor.

D'oh!

How'd He Do That?!?

FAIL!

Way too much code copying from top level to the new nodes.

How'd He Do That?!?

Planner changes for ModifyTable node (a few)

How'd He Do That?!?

Executor changes: ONE new node called ModifyTable

How'd He Do That?!?

Marko Tiikkaja restructures the whole code base for the ModifyTable node. "The usual stuff," (he said casually) for new nodes.

How'd He Do That?!?

WIN!

Next Steps

~~INSERT, UPDATE and DELETE on the top level.~~

~~Finish the patch~~

- ~~EXPLAIN ANALYZE~~
- ~~SPI~~
- ~~SQL functions~~

~~Bug fixes, cleanup, testing~~

RECURSIVE

Optimization

Future?

DCL (GRANT, REVOKE)

DDL (CREATE, ALTER, DROP)

Where Is It?!?

http://www.postgresql.org/developer/beta

```
WITH audience AS (
    VALUES
        ('Questions'),
        ('Comments'),
        ('Critiques')
)
INSERT INTO davidfetter
SELECT * FROM audience
RETURNING...
```

谢谢您

多谢