SQL performance tuning 2017

PGConf.ASIA 2017
DAY2 12/06 15:00 – 15:40 @ Track B

TAKATSUKA Haruka　SRA OSS, Inc. Japan

- About this speaker
  - PostgreSQL help desk, consulting, setup and installation, training coach, and etc. for over 15 years.

- About this talk
  - SQL(Query) performance tuning
  - NOT server tuning

TOC

- Why query tuning
- Query tuning basic
- Tuning with PostgreSQL new features
- Using extensions

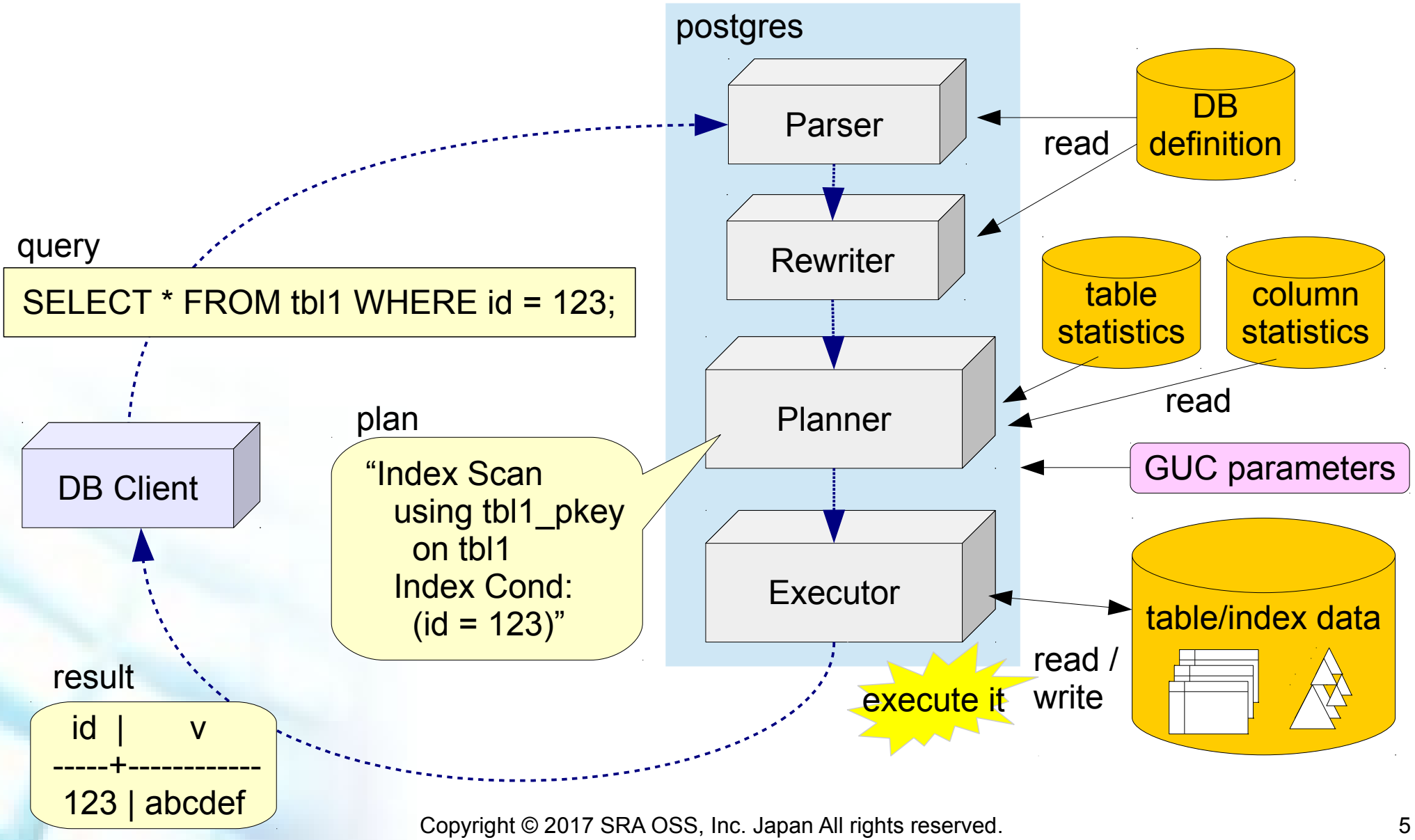
Why query tuning

- Why?
  - SQL is 4th generation language, so don't think about internal behavior
  - Do analyze enough and trust the great optimizer unless you are super hacker

- You need query tuning
  1. Fact: PostgreSQL optimizer is enough good, but too MODEST
     - Esp. query from other DBMS as the migration source.
  2. Huge effect: bad query/plan 100 to 10000 times slower.

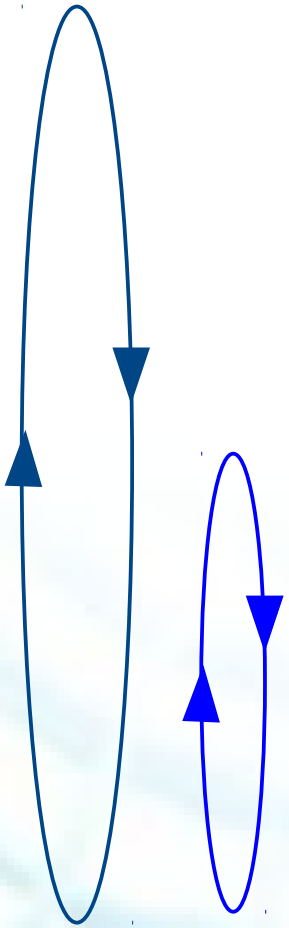
Query tuning basic (1)

- Query processing flow



Query tuning basic (2)

- Query tuning cycle



- Pick a slow query
  - user's complaint
  - monitoring tool
    (pg_stat_statements, pg_statsinfo,
    log_min_duration_statement, pg_badger)
- check its plan
  - Use EXPLAIN
  - Study why is it slow
- Try a query tuning
  - measure its effect, and apply it permanently

Query tuning basic (2)

- Use EXPLAIN

```
db1=# explain (analyze, buffers)
      SELECT * FROM t1 JOIN t3 ON (t1.id = t3.id1) WHERE t3.ts < '2017-11-24 9:00;

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.29..578.65 rows=341 width=89)
              (actual time=0.063..3.781 rows=340 loops=1)
   Buffers: shared hit=1028
   ->  Seq Scan on t3  (cost=0.00..19.50 rows=341 width=52)
                       (actual time=0.027..0.579 rows=340 loops=1)
         Filter: (ts < '2017-11-24 09:00:00'::timestamp without time zone)
         Rows Removed by Filter: 660
         Buffers: shared hit=7
   ->  Index Scan using t1_pkey on t1  (cost=0.29..1.64 rows=1 width=37)
                                       (actual time=0.007..0.007 rows=1 loops=340)
         Index Cond: (id = t3.id1)
         Buffers: shared hit=1021
 Planning time: 0.468 ms
 Execution time: 4.077 ms
(11 rows)
```

estimation and actual result

0.007ms x 340

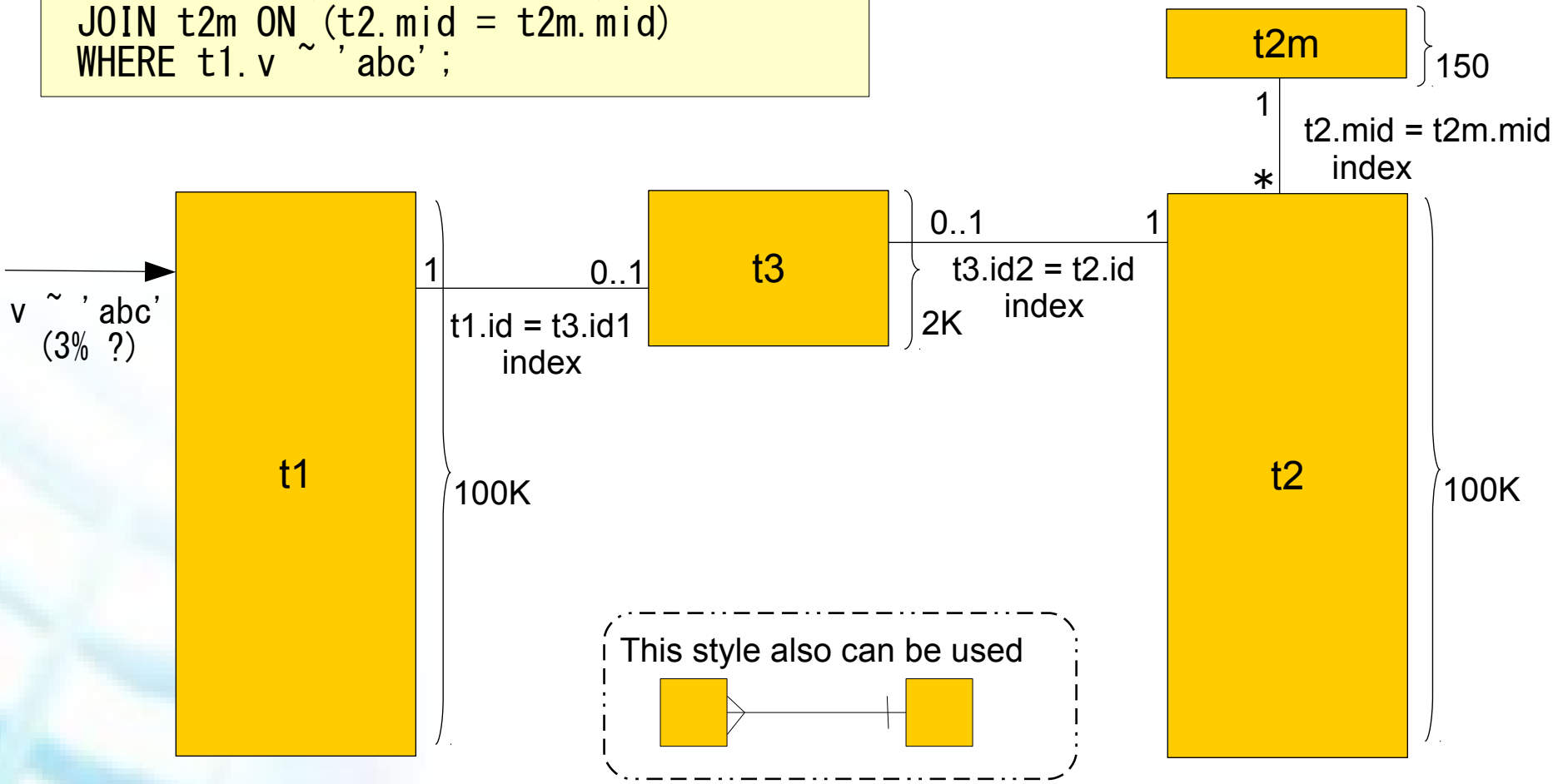
To see other candidate plans

```
$ CFLAGS='-D OPTIMIZER_DEBUG=1' ./configure
```

Query tuning basic (2)

- Query chart

```
SELECT t1.v, t2m.v, t2.v, t3.ts FROM t1
 JOIN t3 ON (t1.id = t3.id1)
 JOIN t2 ON (t2.id = t3.id2)
 JOIN t2m ON (t2.mid = t2m.mid)
 WHERE t1.v ~ 'abc';
```



Query tuning basic (3)

- Query tuning
  - Store more statistics
  - SET parameters
  - Add indices
  - Rewrite the query

- STOP! Firstly, check vacuuming and analyzing

```
SELECT schemaname,  relname,  n_mod_since_analyze,
           last_vacuum,  last_autovacuum,  last_analyze,  last_autoanalyze
  FROM pg_stat_all_tables;
```

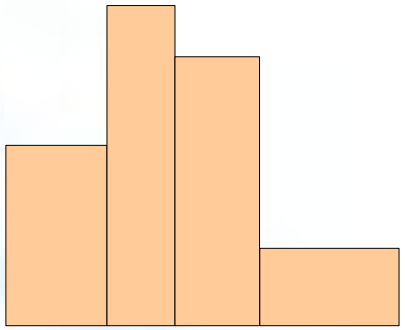
Query tuning basic (3)

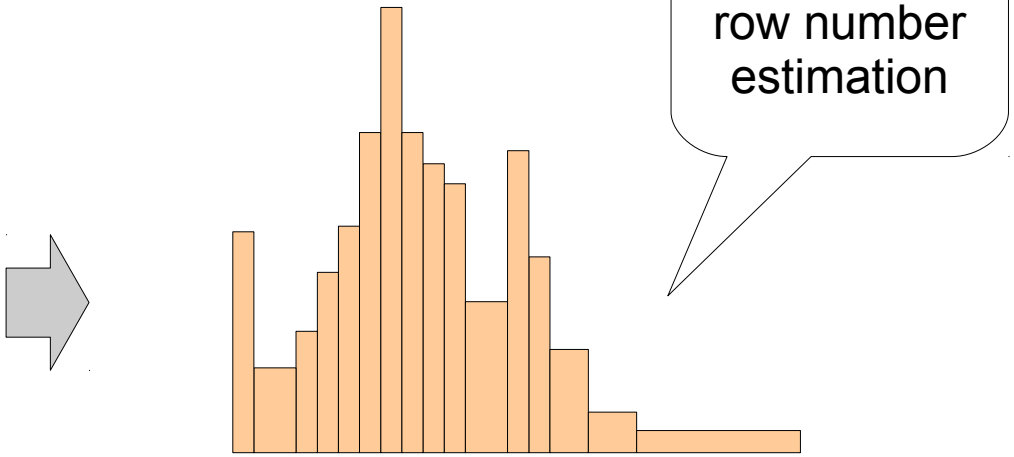
- Store more statistics

```
default_statistics_target = 200   #  default 100
```

```
ALTER TABLE t1 ALTER c1 SET STATISTICS 200;
```

  - Histogram resolution
  - Most Common Value entries





Value:	100	50	0	15
Retio:	0.10	0.05	0.03	0.03

Value:	100	50	0	15	20	70	65	90
Retio:	0.10	0.05	0.03	0.03	0.02	0.02	0.01	0.01

Query tuning basic (4)

- SET parameters

```
SET [LOCAL]  parameter TO 'value'
```

```
ALTER ROLE  role  SET  parameter  TO  'value';
ALTER DATABASE  db  SET  parameter  TO  'value';
ALTER FUNCTION  func  SET  parameter  TO  'value';
```

intervene in the planning

```
enable_bitmapscan      enable_indexonlyscan  enable_nestloop
enable_gathermerge     enable_indexscan      enable_seqscan
enable_hashagg         enable_material       enable_sort
enable_hashjoin        enable_mergejoin      enable_tidscan
cursor_tuple_fraction    replacement_sort_tuples
```

```
SET from_collapse_limit TO 1;   -- default 8
SET join_collapse_limit TO 1;   -- default 8
```

Don't optimize my query

```
random_page_cost = 1.5  # default 4.0
cpu_index_tuple_cost = 0.01
effective_cache_size = 4GB
constraint_exclusion = partition
```

at server tuning

Query tuning basic (5)

- Add indices
  - JOIN condition
  - WHERE condition

  - Btree Index
    - Index Scan、Bitmap Index Scan

  - Hash Index
    - OK in PostgreSQL 10+

  - GIN Index

  - pg_trgm / pg_bigm (GiST / GIN)

```
... WHERE  long_string = 'long long string'
```

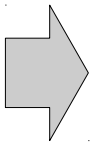
```
... WHERE  tags_array && ARRAY['Tuning']
```

```
... WHERE  string_data LIKE '%pattern%'
```

Query tuning basic (6)

- Rewrite the query: Tweaking index use
  - Index cannot be used

```
... WHERE price * 1.08 > 10000
```



```
... WHERE price > 10000.0 / 1.08
```

  - Do it intentionally

```
CREATE INDEX ix1 ON t1 (v);
CREATE INDEX ix2 ON t2 (v);
```

```
SELECT * FROM t1
 JOIN t3 ON (t1.id = t3.id1)
 JOIN t2 ON (t2.id = t3.id2)
 JOIN t2m ON (t2.mid = t2m.mid)
 WHERE t1.v LIKE 'aaa%' AND t2.v LIKE 'bbb%';
```

Which index does we use?

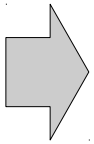
“Don’t use ix1 index”

```
SELECT * FROM t1
 JOIN t3 ON (t1.id = t3.id1)
 JOIN t2 ON (t2.id = t3.id2)
 JOIN t2m ON (t2.mid = t2m.mid)
 WHERE t1.v || '' LIKE 'aaa%' AND t2.v LIKE 'bbb%';
```

Query tuning basic (7)

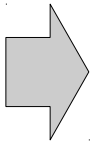
- Rewrite the query: Push down

```
WITH v1 AS (SELECT * FROM t1
) SELECT * FROM v1
   JOIN t3 ON v1.id = t3.id1
   JOIN t2 ON t3.id2 = t2.id
   WHERE v1.v ~ 'abc';
```



```
WITH v1 AS (SELECT * FROM t1 WHERE v ~ 'abc'
) SELECT * FROM v1
   JOIN t3 ON v1.id = t3.id1
   JOIN t2 ON t3.id2 = t2.id
   WHERE v1.v ~ 'abc';
```

```
WITH v1 AS (SELECT * FROM t1)
 SELECT * FROM v1
  JOIN t3 ON v1.id = t3.id1
  JOIN t2 ON t3.id2 = t2.id;
```



```
WITH v1 AS (SELECT * FROM t1 WHERE id
 IN (SELECT id2 FROM t3)
) SELECT * FROM v1
   JOIN t3 ON v1.id = t3.id1
   JOIN t2 ON t3.id2 = t2.id;
```

t1: huge table
t3: small table

Query tuning basic (8)

- Rewrite the query: CTE vs Subquery
  - CTE (WITH clause) tend not to optimize

```
SELECT * FROM
  (SELECT * FROM t1) AS v1
   JOIN t3 ON v1.id = t3.id1
   JOIN t2 ON t3.id2 = t2.id
   WHERE v1.v ~ 'abc';
```

This does not become slow stupidly.

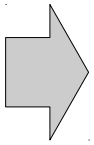
- want to process it separately as the query
  → rewrite subquery to CTE

- want to optimize it freely
  → rewrite CTE to subquery

Query tuning basic (7)

- Rewrite the query: misc.
  - count(*) > 0 TO EXISTS

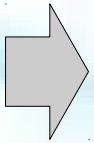
```
SELECT t2m.mid, t2m.v, count(*) > 0 AS exist
  FROM t2m LEFT JOIN t2 USING (mid) WHERE mid < 50
  GROUP BY t2m.mid, t2m.v ORDER BY mid;
```



```
SELECT t2m.mid, t2m.v,
  EXISTS (SELECT 1 FROM t2 WHERE t2.mid = t2m.mid) AS exist
 FROM t2m WHERE mid < 50 ORDER BY mid;
```

  - Avoid correlated subquery

```
SELECT t2m.*, (SELECT count(*) FROM t2 WHERE t2.mid = t2m.mid) AS count
  FROM t2m;
```



```
SELECT t2m.* FROM t2m JOIN
 (SELECT mid, count(*) FROM t2 GROUP BY mid) AS v USING (mid);
```

Tuning with PostgreSQL new features (1)

- Parallelism
  - PG 9.6+
  - GUC parameters
    - prefer parallel or not
    - data size
    - number of workers
  - Storage parameter
  - parallel safe
    - CTE, Foreign Table
    - FUNCTION attribute
  - OLAP v.s. OLTP
  - Cannot control which part is processed in parallel within the query.

(PG10 GUC)

```
parallel_tuple_cost = 0.1
parallel_setup_cost = 1000.0
min_parallel_table_scan_size = 8MB
min_parallel_index_scan_size = 512kB
force_parallel_mode = off
max_parallel_workers_per_gather = 2
max_parallel_workers = 8
```

```
parallel_workers
```

Tuning with PostgreSQL new features (2)

- Extended statistics (PG10+)

```
db1=# EXPLAIN SELECT * FROM t2m
       WHERE subcatcode = 'C3-1' AND catcode = 'C3';
                      QUERY PLAN
-------------------------------------------------------
 Seq Scan on t2m  (cost=0.00..5.25 rows=2 width=45)
   Filter: ((subcatcode = 'C3-1'::text)
           AND (catcode = 'C3'::text))
(2 rows)
```

selectivity × selectivity

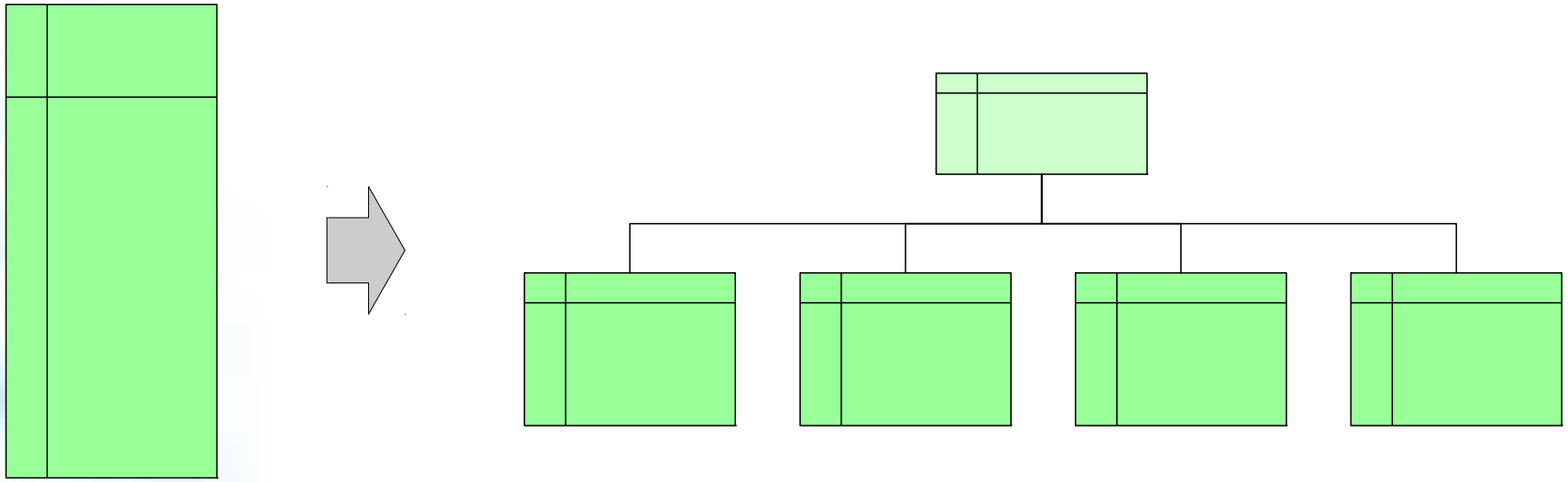
```
db1=# CREATE STATISTICS s_t2m
        ON catcode, subcatcode FROM t2m ;

db1=# ANALYZE t2m ;

db1=# EXPLAIN SELECT * FROM t2m
       WHERE subcatcode = 'C3-1' AND catcode = 'C3';
                      QUERY PLAN
-------------------------------------------------------
 Seq Scan on t2m  (cost=0.00..5.25 rows=10 width=45)
   Filter: ((subcatcode = 'C3-1'::text)
           AND (catcode = 'C3'::text))
(2 rows)
```

```
SELECT mid, catcode,
  subcatcode FROM t2m;
 mid | catcode | subcatcode
-----+---------+------------
  28 | C3      | C3-1
  29 | C1      | C1-1
  30 | C2      | C2-3
  31 | C4      | C4-1
  32 | C3      | C3-3
  33 | C4      | C4-2
  34 | C4      | C4-4
  35 | C4      | C4-4
  36 | C5      | C5-2
  37 | C1      | C1-3
  38 | C2      | C2-1
  39 | C2      | C2-5
  40 | C4      | C4-1
  41 | C5      | C5-5
  42 | C3      | C3-3
  43 | C1      | C1-4
  44 | C1      | C1-2
  45 | C2      | C2-2
```

functional dependency

Tuning with PostgreSQL new features (3)

- Partitioning
  - Efficient partial access
  - PG10 speeds up insertion
  - No global index
  - Join parted table v.s. parted table



Using extensions

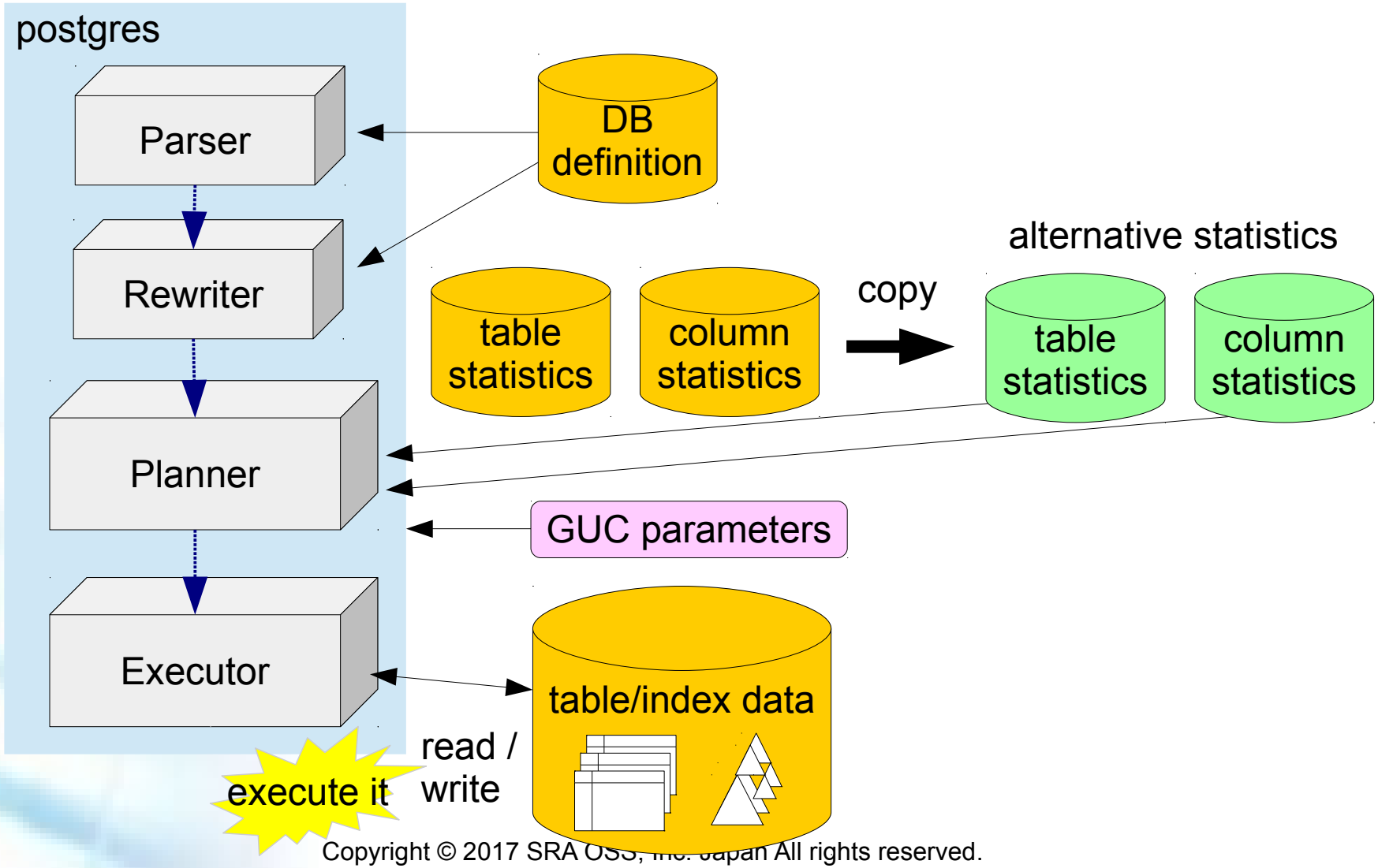
- pg_hint_plan

```
db1=# LOAD 'pg_hint_plan';
db1=# SET pg_hint_plan.debug_print TO on;
db1=# SET client_min_messages TO log;
db1=# SELECT
 /*+ IndexScan(t1 t1_v_idx) Leading(t3 t1) */
 t1.id id1, t2m.mid mid  FROM t1
 JOIN t3 ON (t1.id = t3.id1)
 JOIN t2 ON (t2.id = t3.id2)
 JOIN t2m ON (t2.mid = t2m.mid)
 WHERE t1.v LIKE 'aa%' AND t2.v LIKE 'bb%';

LOG:  available indexes for IndexScan(t1): t1_v_idx
LOG:  pg_hint_plan:
used hint:
IndexScan(t1 t1_v_idx)
Leading(t3 t1)
not used hint:
duplication hint:
error hint:

  id1   | mid2
--------+------
 65117 |   50
 17108 |  117
```

Hints
Leading(t1 t2 t3)
Leading((t1 t2))
SeqScan(t1) NoSeqScan(t1)
IndexScan(t1 idx_t1) NoIndexScan(t1)
BitmapScan(t1) NoBitmapScan(t1)
NestLoop(t1 t2) NoNestLoop(t1 t2)
MergeJoin(t1 t2) NoMergeJoin(t1 t2)
HashJoin(t1 t2) NoHashJoin(t1 t2)
Set(from_collapse_limit 1)
Rows(t1 t2 1000)
Parallel(4 soft)

Using extensions

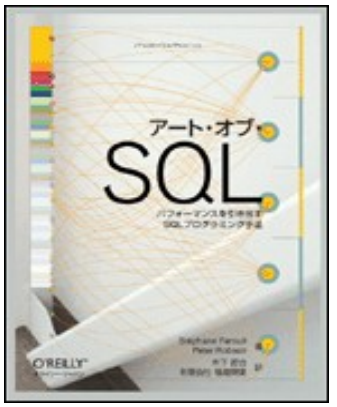
- pg_dbms_stats
  - Lock or Backup the statistics



FIN

- Books

SQL実践入門 -
高速でわかりやすいクエリの書き方
ミック（技術評論社）



アート・オブ・SQL
パフォーマンスを引き出す
SQLプログラミング手法
Stephane Faroult / Peter Robson
（O'REILLY）

- Question ?

オープンソースとともに

SRA OSS,INC.

URL：http://www.sraoss.co.jp/