PostgreSQL におけるSQL チューニング入門

2024-05-25

SRA OSS LLC
千田 貴大

会社紹介

SRA OSS合同会社

所　在　地: 東京都豊島区南池袋2-32-8
設　立　日: 2022年6月17日
出　　資: 株式会社SRA
資　本　金: 7,000万円
社　　長: 稲葉 香理

事業内容

- オープンソースソフトウェア（OSS）関連のサポート、製品開発・販売、構築・コンサル
- OSSの教育、開発、コミュニティ運営支援
- ソフトウェアの研究開発

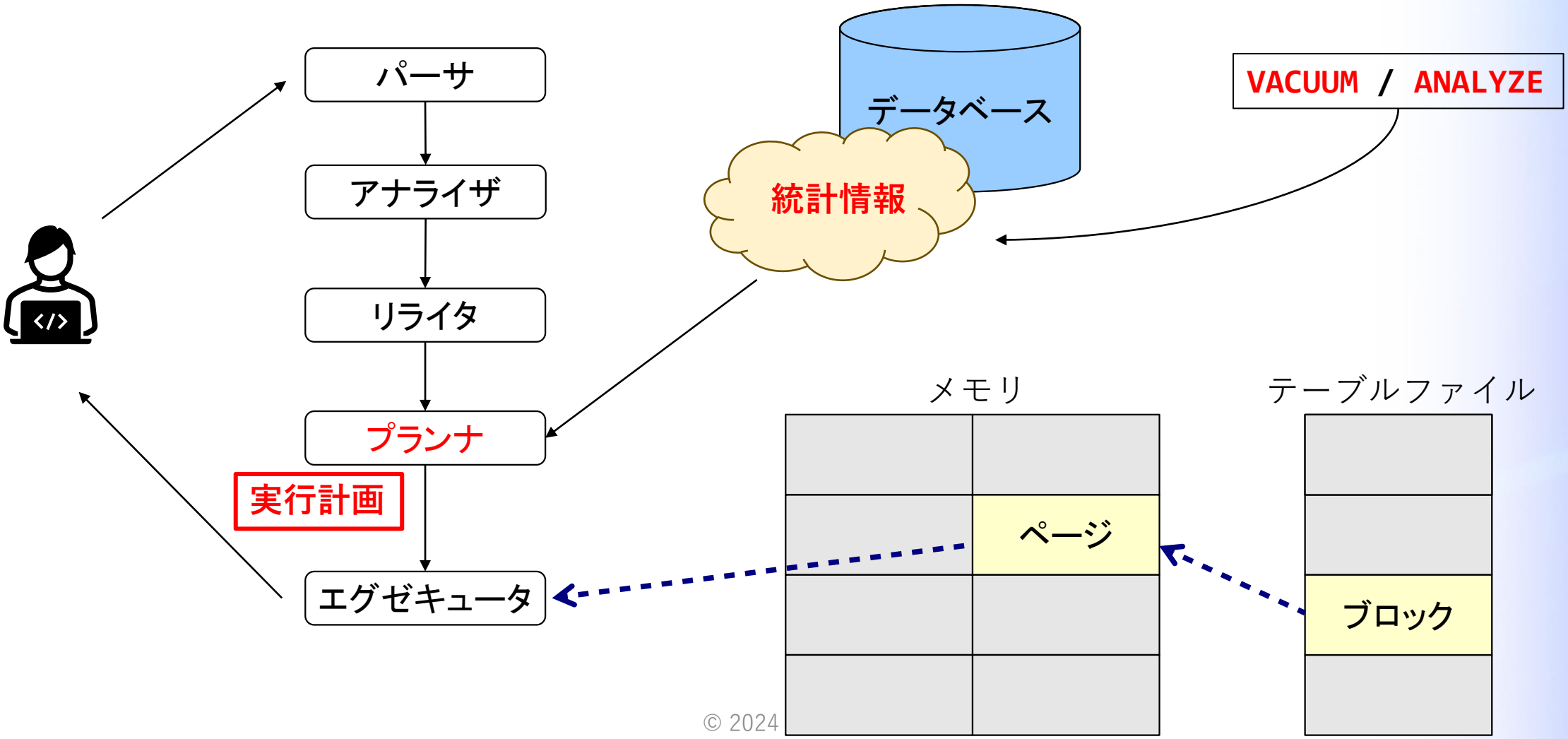
顧　　問: 石井 達夫
技術顧問: 増永 良文（お茶の水女子大学名誉教授）

今回のターゲット

```
WITH list AS (
           SELECT mt_matter_id,
                  max(mt_matter_update)
           FROM matters
           WHERE mt_matter_update < '2024-05-01'::date + '1 day'::interval
           GROUP BY mt_matter_id
)

SELECT mt_matter_id AS matter_id, mt_matter_update AS matter_update, mt_matter_name, mt_matter_status,
       mt_employee_no, pm_matter_product_no, COALESCE(mt_matter_kind, 0) AS mt_matter_kind,
       pm_amount, COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) AS order_date,
       pm_probability_id, pr_product_name,
       CASE WHEN COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) < current_date THEN true END order_date_over,
       pm_new_or_next
FROM matters
JOIN products_matters
  ON (pm_matter_id, pm_matter_update) = (mt_matter_id, mt_matter_update)
JOIN products
  ON (pm_product_code, pm_product_update) = (pr_product_code, pr_last_updated_when)
JOIN products_segments
  ON (pr_product_code, pr_last_updated_when) = (ps_product_code, ps_product_update)
WHERE (pm_matter_id, pm_matter_update) IN (SELECT * FROM list)
  AND '2024-04-01'::date <= COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule)
  AND COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) < '2024-04-01'::date + '1 month'::interval
  AND pm_probability_id = 7
  AND ps_segment_id = 3
  AND ps_segment_class_id = 7
ORDER BY order_date desc, matter_id, pm_matter_product_no;
```

補足① SQL 実行仕組み



チューニング手順① 実行計画取得

- 実行計画
  プランナが統計情報を元に最も効率良くデータ取得できるかを計算した結果
  選択されたデータの処理、アクセス方法

- 実行計画参照方法
  - EXPLAIN
    - SQL 実行なし
    - プランナーが生成した実行計画参照
  - EXPLAIN ANALYZE
    - SQL 実行あり
    - 実行結果に基づく情報表示

```
=$ EXPLAIN 《SQL》
```

```
=$ EXPLAIN ANALYZE 《SQL》
```

[PostgreSQL 16.0文書 - EXPLAIN]
https://www.sraoss.co.jp/PostgreSQL/Manual/document/16/sql-explain.html

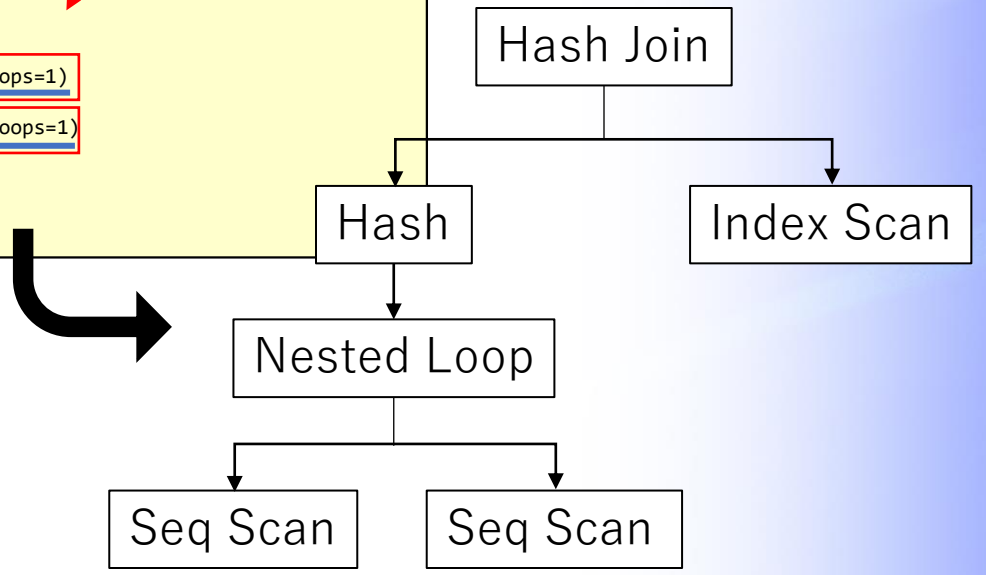
```
 Sort  (cost=16130.86..16130.87 rows=1 width=233) (actual time=30570.523..30570.533 rows=89 loops=1)
   Sort Key: (COALESCE(products_matters.pm_ordering_or_lost_date, products_matters.pm_ordering_schedule)) DESC, matters.mt_matter_id, products_matters.pm_matter_product_no
   Sort Method: quicksort  Memory: 47kB
   ->  Nested Loop  (cost=1495.42..16130.85 rows=1 width=233) (actual time=1395.616..30569.236 rows=89 loops=1)
         ->  Nested Loop  (cost=1495.14..16130.36 rows=1 width=206) (actual time=1395.584..30567.722 rows=89 loops=1)
               Join Filter: ((products_matters.pm_matter_id = matters_1.mt_matter_id) AND (products_matters.pm_matter_update = (max(matters_1.mt_matter_update))))
               Rows Removed by Join Filter: 5699383
               ->  Nested Loop  (cost=1494.72..8671.16 rows=1 width=218) (actual time=16.843..45.442 rows=1164 loops=1)
                     ->  Hash Join  (cost=1494.30..8665.13 rows=1 width=72) (actual time=16.823..27.479 rows=1164 loops=1)
                           Hash Cond: ((products_matters.pm_product_code = products_segments.ps_product_code) AND (products_matters.pm_product_update = products_segments.ps_product_update))
                           ->  Bitmap Heap Scan on products_matters  (cost=1476.89..8635.26 rows=2373 width=54) (actual time=16.024..19.915 rows=1999 loops=1)
                                 Recheck Cond: (('2024-04-01'::date <= COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule)) AND (COALESCE(pm_ordering_or_lost_date,
pm_ordering_schedule) < '2024-05-01 00:00:00'::timestamp without time zone) AND (pm_probability_id = 7))
                                 Heap Blocks: exact=465
                                 ->  BitmapAnd  (cost=1476.89..1476.89 rows=2373 width=0) (actual time=15.929..15.930 rows=0 loops=1)
                                       ->  Bitmap Index Scan on products_matters_order_date_index  (cost=0.00..194.48 rows=11006 width=0) (actual time=5.228..5.229 rows=11102 loops=1)
                                             Index Cond: ((COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) >= '2024-04-01'::date) AND
 (COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) < '2024-05-01 00:00:00'::timestamp without time zone))
                                       ->  Bitmap Index Scan on products_matters_probability_id_index  (cost=0.00..1280.97 rows=92340 width=0) (actual time=10.549..10.549 rows=92569
loops=1)
                                             Index Cond: (pm_probability_id = 7)
                           ->  Hash  (cost=17.07..17.07 rows=23 width=18) (actual time=0.657..0.658 rows=46 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 11kB
                                 ->  Index Only Scan using products_segments_pkey on products_segments  (cost=0.29..17.07 rows=23 width=18) (actual time=0.568..0.604 rows=46 loops=1)
                                       Index Cond: ((ps_segment_id = 3) AND (ps_segment_class_id = 7))
                                       Heap Fetches: 0
                     ->  Index Scan using matters_pkey on matters  (cost=0.42..6.03 rows=1 width=146) (actual time=0.010..0.010 rows=1 loops=1164)
                           Index Cond: ((mt_matter_id = products_matters.pm_matter_id) AND (mt_matter_update = products_matters.pm_matter_update))
               ->  GroupAggregate  (cost=0.42..7383.75 rows=5030 width=12) (actual time=0.062..25.890 rows=4896 loops=1164)
                     Group Key: matters_1.mt_matter_id
                     ->  Index Only Scan using matters_pkey on matters matters_1  (cost=0.42..6649.38 rows=136814 width=12) (actual time=0.005..14.681 rows=134350 loops=1164)
                           Index Cond: (mt_matter_update < '2024-05-02 00:00:00'::timestamp without time zone)
                           Heap Fetches: 0
         ->  Index Scan using products_pkey on products  (cost=0.28..0.48 rows=1 width=82) (actual time=0.009..0.009 rows=1 loops=89)
               Index Cond: ((pr_product_code = products_matters.pm_product_code) AND (pr_last_updated_when = products_matters.pm_product_update))
 Planning Time: 16.443 ms
 Execution Time: 30571.924 ms
```

補足② 実行計画の見方

- EXPLAIN 実行結果

```
=$ EXPLAIN ANALYZE
SELECT * FROM pgbench_accounts JOIN pgbench_branches USING (bid) JOIN pgbench_tellers USING (bid) WHERE aid < 1000;
                                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=2.65..183.81 rows=10880 width=805) (actual time=0.238..4.403 rows=9990 loops=1)
   Hash Cond: (pgbench_accounts.bid = pgbench_branches.bid)
   ->  Index Scan using pgbench_accounts_pkey on pgbench_accounts  (cost=0.29..56.33 rows=1088 width=97) (actual time=0.108..0.739 rows=999 loops=1)
         Index Cond: (aid < 1000)
   ->  Hash  (cost=2.24..2.24 rows=10 width=716) (actual time=0.082..0.084 rows=10 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Nested Loop  (cost=0.00..2.24 rows=10 width=716) (actual time=0.055..0.064 rows=10 loops=1)
               Join Filter: (pgbench_branches.bid = pgbench_tellers.bid)
               ->  Seq Scan on pgbench_branches  (cost=0.00..1.01 rows=1 width=364) (actual time=0.016..0.017 rows=1 loops=1)
               ->  Seq Scan on pgbench_tellers  (cost=0.00..1.10 rows=10 width=352) (actual time=0.007..0.010 rows=10 loops=1)
 Planning Time: 5.460 ms
 Execution Time: 5.415 ms
(12 rows)
```

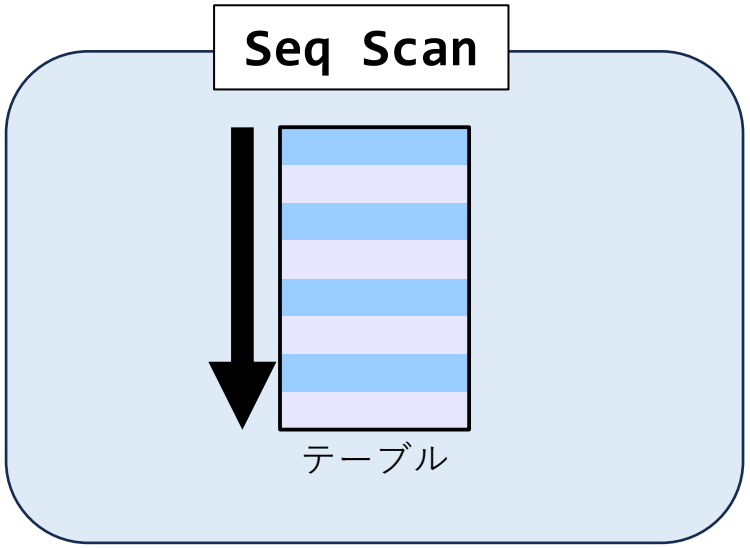
計画ノード

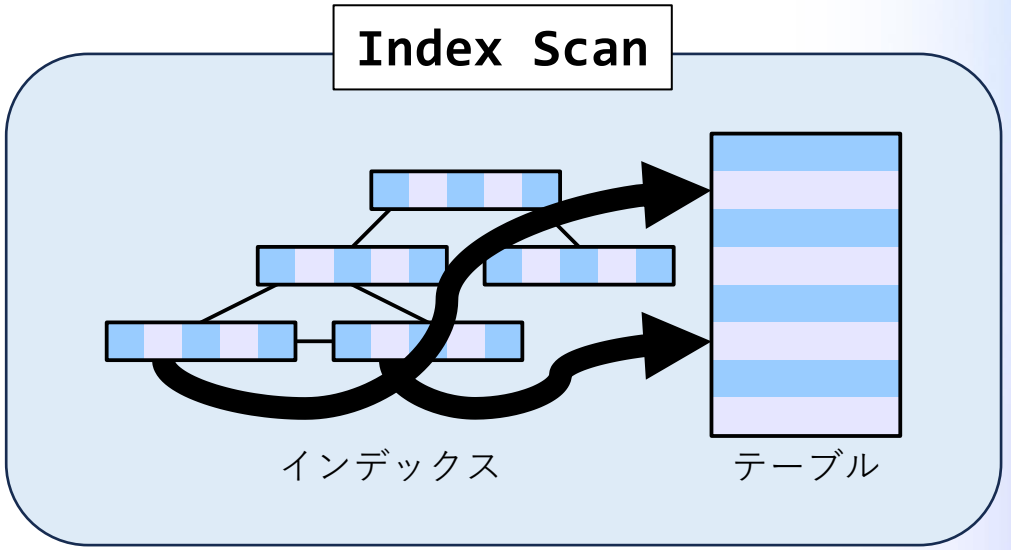


- 計画タイプ － 内部的な処理方式
  （例：Hash Join、Seq Scan など）

補足③ 計画タイプ（1）

- スキャン方式

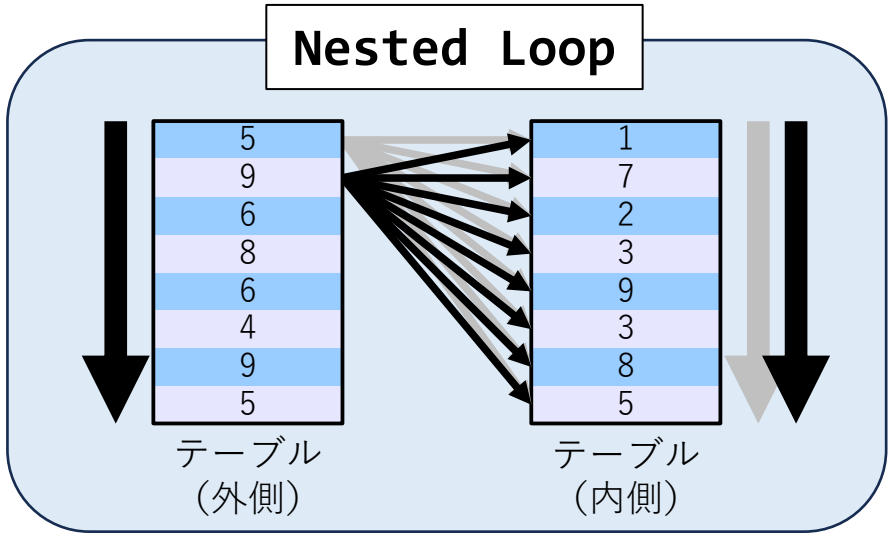


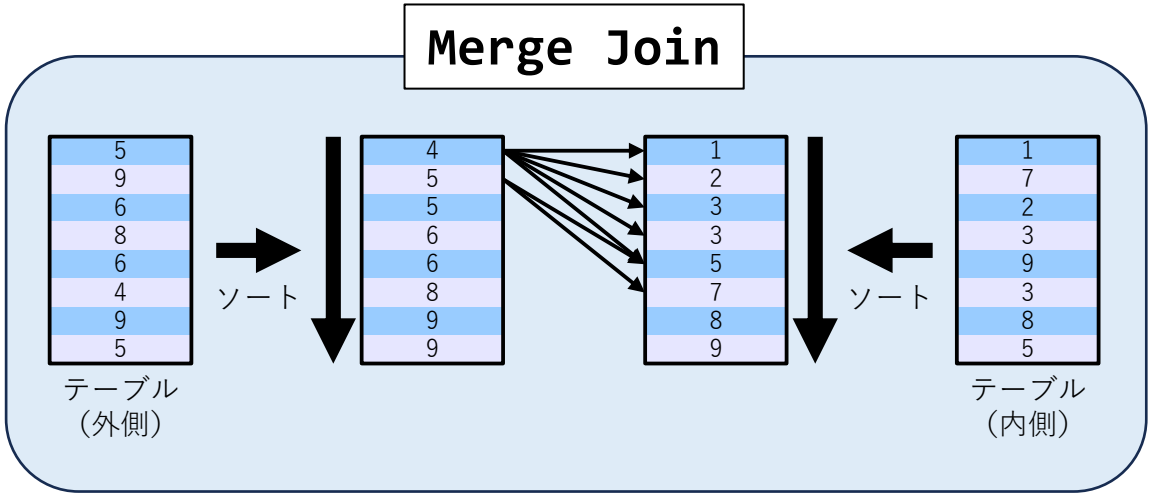


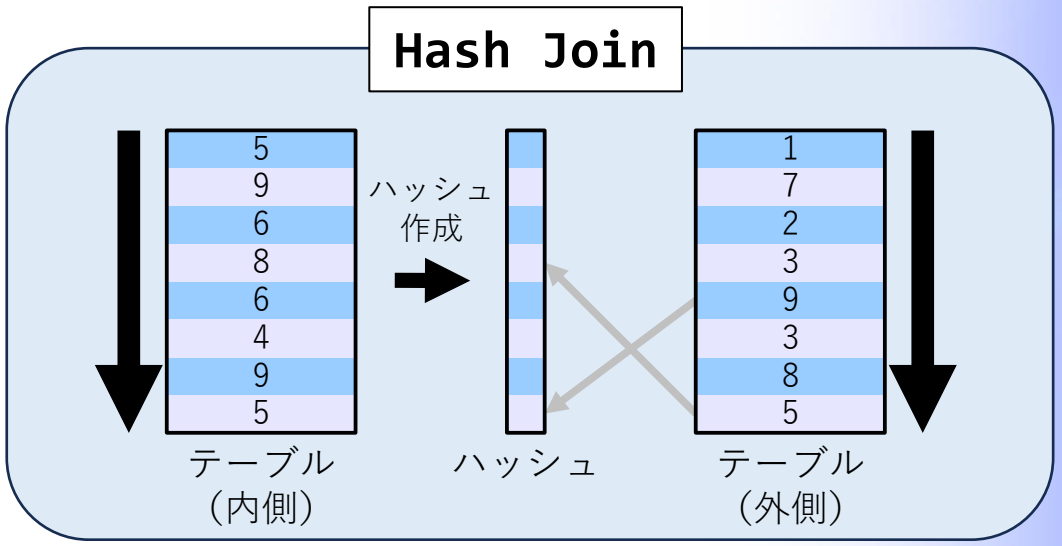
  - その他
    - IndexOnlyScan
    - Bitmap Index Scan
    - Bitmap Heap Scan　　など…

補足③ 計画タイプ（2）

- 結合方式

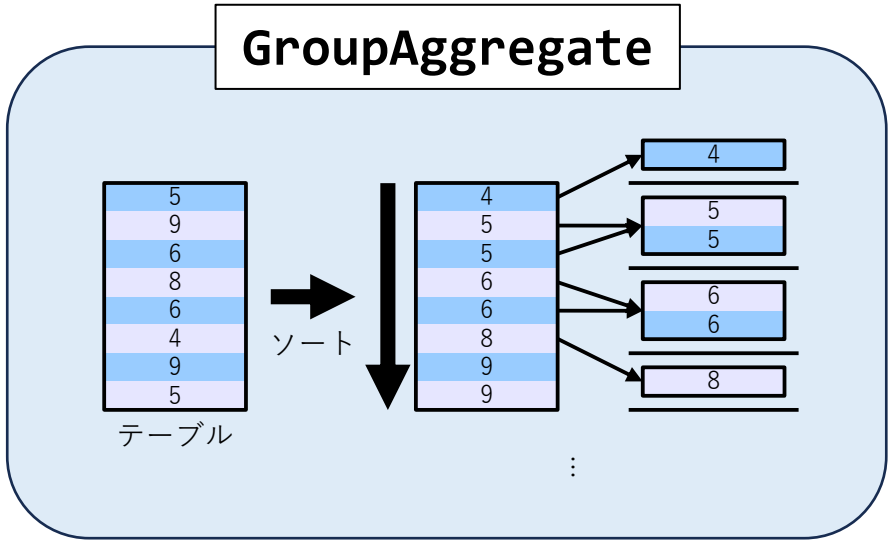


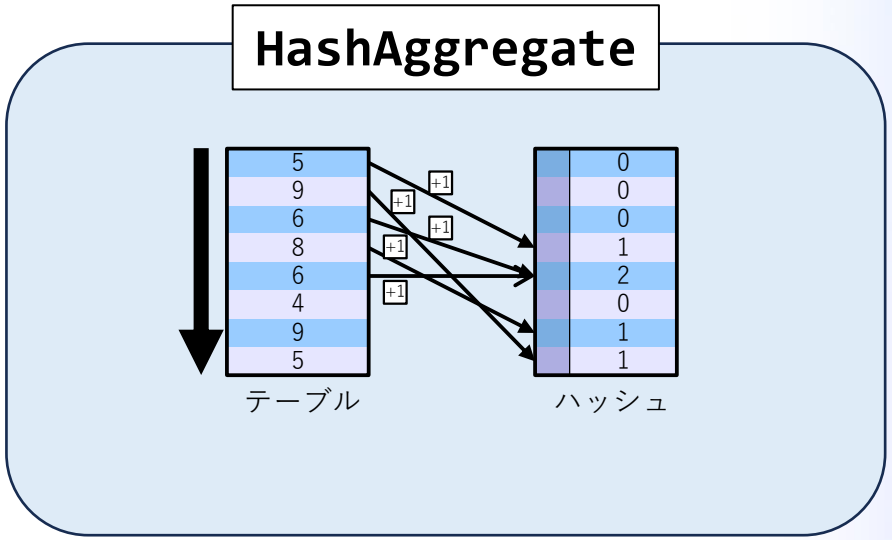




補足③ 計画タイプ（3）

- 集約
  - GROUP BY





※ 他にもさまざまな計画タイプが存在します

チューニング手順② 要因考察（1）

```
->  Nested Loop  (cost=1495.14..16130.36 rows=1 width=206) (actual time=1395.584..30567.722 rows=89 loops=1)
            Join Filter: ((products_matters.pm_matter_id = matters_1.mt_matter_id) AND
(products_matters.pm_matter_update = (max(matters_1.mt_matter_update))))
            Rows Removed by Join Filter: 5699383
            ->  Nested Loop  (cost=1494.72..8671.16 rows=1 width=218) (actual time=16.843..45.442 rows=1164 loops=1)
```

```
->  GroupAggregate  (cost=0.42..7383.75 rows=5030 width=12) (actual time=0.062..25.890 rows=4396 loops=1164)
                    Group Key: matters_1.mt_matter_id
                    ->  Index Only Scan using matters_pkey on matters matters_1
(cost=0.42..6649.38 rows=136814 width=12) (actual time=0.005..14.681 rows=134350 loops=1164)
                          Index Cond: (mt_matter_update < '2024-05-02 00:00:00'::timestamp without time zone)
                          Heap Fetches: 0
```

チューニング手順② 要因考察（2）

```
WITH list AS (
          SELECT mt_matter_id,
                 max(mt_matter_update)
          FROM matters
          WHERE mt_matter_update < '2024-05-01'::date + '1 day'::interval
          GROUP BY mt_matter_id
)

SELECT mt_matter_id AS matter_id, mt_matter_update AS matter_update, mt_matter_name, mt_matter_status,
       mt_employee_no, pm_matter_product_no, COALESCE(mt_matter_kind, 0) AS mt_matter_kind,
       pm_amount, COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) AS order_date,
       pm_probability_id, pr_product_name,
       CASE WHEN COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) < current_date THEN true END order_date_over,
       pm_new_or_next
FROM matters
JOIN products_matters
  ON (pm_matter_id, pm_matter_update) = (mt_matter_id, mt_matter_update)
JOIN products
  ON (pm_product_code, pm_product_update) = (pr_product_code, pr_last_updated_when)
JOIN products_segments
  ON (pr_product_code, pr_last_updated_when) = (ps_product_code, ps_product_update)
WHERE (pm_matter_id, pm_matter_update) IN (SELECT * FROM list)
  AND '2024-04-01'::date <= COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule)
  AND COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) < '2024-04-01'::date + '1 month'::interval
  AND pm_probability_id = 7
  AND ps_segment_id = 3
  AND ps_segment_class_id = 7
ORDER BY order_date desc, matter_id, pm_matter_product_no;
```

チューニング手順③ 対応案検討

- 統計情報更新
- 設定変更
- 一時テーブル
- （SQL 書き換え）
- （外部ツール）

チューニング手順④ 検証 その1

- 統計情報更新
  - ANALYZE 実行

```
=# ANALYZE;
ANALYZE
=# EXPLAIN ANALYZE 《SQL省略》;
《略》
Planning Time: 19.428 ms
Execution Time: 30233.434 ms
(34 rows)
```

  - default_statistics_target 変更
    - 収集される統計情報のエントリー数

```
=# SET default_statistics_target TO 10000;
SET
=# ANALYZE;
ANALYZE
=# EXPLAIN ANALYZE 《SQL省略》;
《略》
Planning Time: 111.763 ms
Execution Time: 29066.923 ms
(34 rows)
```

結果：変化なし

チューニング手順④ 検証 その2(1)

- 設定変更(1)
  - 計画タイプの無効化
  - enable_nestloop

```
=# SET enable_nestloop TO off;
SET
=# EXPLAIN ANALYZE 《SQL省略》;
                                                            QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Gather Merge  (cost=43747.79..43747.90 rows=1 width=233) (actual time=714.718..717.245 rows=89 loops=1)
   Workers Planned: 1
   Workers Launched: 1
   ->  Sort  (cost=42747.78..42747.78 rows=1 width=233) (actual time=673.670..673.682 rows=44 loops=2)
         Sort Key: (COALESCE(products_matters.pm_ordering_or_lost_date, products_matters.pm_ordering_schedule)) DESC, matters.mt_matter_id, products_matters.pm_matter_product_no
         Sort Method: quicksort  Memory: 34kB
         Worker 0:  Sort Method: quicksort  Memory: 37kB
         ->  Hash Join  (cost=35531.01..42747.77 rows=1 width=233) (actual time=672.676..673.542 rows=44 loops=2)
               Hash Cond: ((products_matters.pm_product_code = products.pr_product_code) AND (products_matters.pm_product_update = products.pr_last_updated_when))
               ->  Parallel Hash Join  (cost=35402.38..42619.11 rows=1 width=188) (actual time=660.414..661.321 rows=82 loops=2)
                     Hash Cond: ((products_matters.pm_matter_id = matters.mt_matter_id) AND (products_matters.pm_matter_update = matters.mt_matter_update))
                     ->  Parallel Bitmap Heap Scan on products_matters  (cost=1483.58..8689.72 rows=1412 width=54) (actual time=5.483..6.431 rows=1000 loops=2)
                           Recheck Cond: (('2024-04-01'::date <= COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule)) AND (COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) < '2024-05-01 00:00:00'::timestamp without time
                           Heap Blocks: exact=296
                           ->  BitmapAnd  (cost=1483.58..1483.58 rows=2400 width=0) (actual time=5.193..5.195 rows=0 loops=1)
                                 ->  Bitmap Index Scan on products_matters_order_date_index  (cost=0.00..199.44 rows=11102 width=0) (actual time=0.841..0.841 rows=11102 loops=1)
                                       Index Cond: ((COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) >= '2024-04-01'::date) AND (COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) < '2024-05-01 00:00:00'::timestamp
                                 ->  Bitmap Index Scan on products_matters_probability_id_index  (cost=0.00..1282.69 rows=92569 width=0) (actual time=4.164..4.164 rows=92569 loops=1)
                                       Index Cond: (pm_probability_id = 7)
                     ->  Parallel Hash  (cost=33918.78..33918.78 rows=1 width=158) (actual time=653.994..653.997 rows=2514 loops=2)
                           Buckets: 8192 (originally 1024)  Batches: 1 (originally 1)  Memory Usage: 1112kB
                           ->  Hash Join  (cost=7460.04..33918.78 rows=1 width=158) (actual time=73.994..642.817 rows=2514 loops=2)
                                 Hash Cond: ((matters.mt_matter_id = matters_1.mt_matter_id) AND (matters.mt_matter_update = (max(matters_1.mt_matter_update))))
                                 ->  Parallel Seq Scan on matters  (cost=0.00..26159.32 rows=57032 width=146) (actual time=0.016..521.909 rows=68438 loops=2)
                                 ->  Hash  (cost=7384.59..7384.59 rows=5030 width=12) (actual time=73.891..73.893 rows=5029 loops=2)
                                       Buckets: 8192  Batches: 1  Memory Usage: 300kB
                                       ->  GroupAggregate  (cost=0.42..7384.59 rows=5030 width=12) (actual time=0.243..71.553 rows=5029 loops=2)
                                             Group Key: matters_1.mt_matter_id
                                             ->  Index Only Scan using matters_pkey on matters matters_1  (cost=0.42..6649.94 rows=136870 width=12) (actual time=0.069..42.128 rows=136873 loops=2)
                                                   Index Cond: (mt_matter_update < '2024-05-02 00:00:00'::timestamp without time zone)
                                                   Heap Fetches: 0
               ->  Hash  (cost=128.28..128.28 rows=23 width=100) (actual time=12.041..12.043 rows=46 loops=2)
                     Buckets: 1024  Batches: 1  Memory Usage: 15kB
                     ->  Hash Join  (cost=17.41..128.28 rows=23 width=100) (actual time=4.562..11.956 rows=46 loops=2)
                           Hash Cond: ((products.pr_product_code = products_segments.ps_product_code) AND (products.pr_last_updated_when = products_segments.ps_product_update))
                           ->  Seq Scan on products  (cost=0.00..107.13 rows=713 width=82) (actual time=2.241..11.038 rows=713 loops=2)
                           ->  Hash  (cost=17.07..17.07 rows=23 width=18) (actual time=0.269..0.270 rows=46 loops=2)
                                 Buckets: 1024  Batches: 1  Memory Usage: 11kB
                                 ->  Index Only Scan using products_segments_pkey on products_segments  (cost=0.29..17.07 rows=23 width=18) (actual time=0.110..0.219 rows=46 loops=2)
                                       Index Cond: ((ps_segment_id = 3) AND (ps_segment_class_id = 7))
                                       Heap Fetches: 0
Planning Time: 76.578 ms
Execution Time: 717.499 ms
(43 rows)
```

結果：効果あり

チューニング手順④ 検証 その2(2)

- 設定変更(2)
  - テーブルの結合順序
  - join_collapse_limit

```
=# SET join_collapse_limit TO 1;
SET
=# EXPLAIN ANALYZE 《SQL省略》;
                                                                                                                                      QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------
 Gather Merge  (cost=25893.50..25893.62 rows=1 width=233) (actual time=75.295..76.891 rows=89 loops=1)
   Workers Planned: 1
   Workers Launched: 1
   ->  Sort  (cost=24893.49..24893.50 rows=1 width=233) (actual time=65.527..65.537 rows=44 loops=2)
         Sort Key: (COALESCE(products_matters.pm_ordering_or_lost_date, products_matters.pm_ordering_schedule)) DESC, matters.mt_matter_id, products_matters.pm_matter_product_no
         Sort Method: quicksort  Memory: 36kB
         Worker 0:  Sort Method: quicksort  Memory: 35kB
         ->  Hash Join  (cost=24853.31..24893.48 rows=1 width=233) (actual time=64.793..65.438 rows=44 loops=2)
               Hash Cond: ((products_matters.pm_matter_id = matters_1.mt_matter_id) AND (products_matters.pm_matter_update = (max(matters_1.mt_matter_update))))
               ->  Merge Join  (cost=17393.27..17433.20 rows=45 width=246) (actual time=18.828..19.366 rows=582 loops=2)
                     Merge Cond: ((products_matters.pm_product_code = products_segments.ps_product_code) AND (products_matters.pm_product_update = products_segments.ps_product_update))
                     ->  Merge Join  (cost=17392.98..17408.95 rows=1412 width=282) (actual time=18.550..19.101 rows=774 loops=2)
                           Merge Cond: ((products_matters.pm_product_code = products.pr_product_code) AND (products_matters.pm_product_update = products.pr_last_updated_when))
                           ->  Sort  (cost=17252.06..17255.59 rows=1412 width=200) (actual time=17.481..17.543 rows=774 loops=2)
                                 Sort Key: products_matters.pm_product_code, products_matters.pm_product_update
                                 Sort Method: quicksort  Memory: 307kB
                                 Worker 0:  Sort Method: quicksort  Memory: 246kB
                                 ->  Nested Loop  (cost=1484.00..17178.19 rows=1412 width=200) (actual time=3.591..15.278 rows=1000 loops=2)
                                       ->  Parallel Bitmap Heap Scan on products_matters  (cost=1483.58..8689.72 rows=1412 width=54) (actual time=3.479..4.505 rows=1000 loops=2)
                                             Recheck Cond: (('2024-04-01'::date <= COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule)) AND (COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) < '2024-05-01 00:00:00'::t
(pm_probability_id = 7))
                                             Heap Blocks: exact=268
                                             ->  BitmapAnd  (cost=1483.58..1483.58 rows=2400 width=0) (actual time=6.798..6.800 rows=0 loops=1)
                                                   ->  Bitmap Index Scan on products_matters_order_date_index  (cost=0.00..199.44 rows=11102 width=0) (actual time=1.273..1.274 rows=11102 loops=1)
                                                         Index Cond: ((COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) >= '2024-04-01'::date) AND (COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) < '2024-05-01 00
                                                   ->  Bitmap Index Scan on products_matters_probability_id_index  (cost=0.00..1282.69 rows=92569 width=0) (actual time=5.331..5.331 rows=92569 loops=1)
                                                         Index Cond: (pm_probability_id = 7)
                                       ->  Index Scan using matters_pkey on matters  (cost=0.42..6.01 rows=1 width=146) (actual time=0.010..0.010 rows=1 loops=1999)
                                             Index Cond: ((mt_matter_id = products_matters.pm_matter_id) AND (mt_matter_update = products_matters.pm_matter_update))
                           ->  Sort  (cost=140.92..142.70 rows=713 width=82) (actual time=1.060..1.092 rows=466 loops=2)
                                 Sort Key: products.pr_product_code, products.pr_last_updated_when
                                 Sort Method: quicksort  Memory: 102kB
                                 Worker 0:  Sort Method: quicksort  Memory: 102kB
                                 ->  Seq Scan on products  (cost=0.00..107.13 rows=713 width=82) (actual time=0.027..0.356 rows=713 loops=2)
                     ->  Index Only Scan using products_segments_pkey on products_segments  (cost=0.29..17.07 rows=23 width=18) (actual time=0.040..0.078 rows=46 loops=2)
                           Index Cond: ((ps_segment_id = 3) AND (ps_segment_class_id = 7))
                           Heap Fetches: 0
               ->  Hash  (cost=7384.59..7384.59 rows=5030 width=12) (actual time=45.806..45.807 rows=5029 loops=2)
                     Buckets: 8192  Batches: 1  Memory Usage: 300kB
                     ->  GroupAggregate  (cost=0.42..7384.59 rows=5030 width=12)
(actual time=0.233..44.353 rows=5029 loops=2)
                           Group Key: matters_1.mt_matter_id
                           ->  Index Only Scan using matters_pkey on matters matters_1  (cost=0.42..6649.94 rows=136870 width=12) (actual time=0.069..27.234 rows=136873 loops=2)
                                 Index Cond: (mt_matter_update < '2024-05-02 00:00:00'::timestamp without time zone)
                                 Heap Fetches: 0
 Planning Time: 68.663 ms
 Execution Time: 77.046 ms
(45 rows)
```

結果：効果あり

チューニング手順④ 検証 その3

- 一時テーブル
  - 同セッション/トランザクション内のみにて参照可
  - セッション/トランザクション終了時に自動削除
- SQL 書き換え要（WITH 句を削除し、一時テーブルを利用）

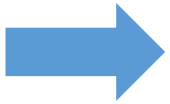
```
=# CREATE TEMP TABLE tmp_20240508 (tmp_id INT, tmp_update TIMESTAMP);
CREATE TABLE
=# INSERT INTO tmp_20240508
   SELECT mt_matter_id, max(mt_matter_update) FROM matters
   WHERE mt_matter_update < '2024-05-08'::date + '1 day'::interval GROUP BY mt_matter_id;
INSERT 0 5030
=# EXPLAIN ANALYZE 《SQL 省略》
                                                            QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=4819.90..4819.91 rows=2 width=233) (actual time=28.042..28.052 rows=89 loops=1)
   Sort Key: (COALESCE(products_matters.pm_ordering_or_lost_date, products_matters.pm_ordering_schedule)) DESC, matters.mt_matter_id, products_matters.pm_matter_product_no
   Sort Method: quicksort  Memory: 47kB
   ->  Nested Loop  (cost=115.08..4819.89 rows=2 width=233) (actual time=26.387..27.931 rows=89 loops=1)
         ->  Nested Loop  (cost=114.81..4819.40 rows=1 width=206) (actual time=26.370..27.742 rows=89 loops=1)
               ->  Nested Loop  (cost=114.39..4813.39 rows=1 width=84) (actual time=26.355..27.482 rows=89 loops=1)
                     Join Filter: ((products_segments.ps_product_code = products_matters.pm_product_code) AND (products_segments.ps_product_update = products_matters.pm_product_update))
                     Rows Removed by Join Filter: 7455
                     ->  Index Only Scan using products_segments_pkey on products_segments  (cost=0.29..17.07 rows=23 width=18) (actual time=0.032..0.076 rows=46 loops=1)
                           Index Cond: ((ps_segment_id = 3) AND (ps_segment_class_id = 7))
                           Heap Fetches: 0
                     ->  Materialize  (cost=114.10..4774.32 rows=55 width=66) (actual time=0.037..0.571 rows=164 loops=46)
                           ->  Nested Loop  (cost=114.10..4774.05 rows=55 width=66) (actual time=1.711..25.807 rows=164 loops=1)
                                 ->  HashAggregate  (cost=113.68..119.39 rows=571 width=12) (actual time=1.643..2.668 rows=5030 loops=1)
                                       Group Key: tmp_20240508.tmp_id, tmp_20240508.tmp_update
                                       Batches: 1  Memory Usage: 465kB
                                       ->  Seq Scan on tmp_20240508  (cost=0.00..85.12 rows=5712 width=12) (actual time=0.018..0.483 rows=5030 loops=1)
                                 ->  Index Scan using products_matters_pkey on products_matters  (cost=0.42..8.14 rows=1 width=54) (actual time=0.004..0.004 rows=0 loops=5030)
                                       Index Cond: ((pm_matter_id = tmp_20240508.tmp_id) AND (pm_matter_update = tmp_20240508.tmp_update))
                                       Filter: (('2024-04-01'::date <= COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule)) AND (COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) < '2024-05-01 00:00:00'::timestamp without time zone) AND (pm_probability_id = 7))
                                       Rows Removed by Filter: 3
               ->  Index Scan using matters_pkey on matters  (cost=0.42..6.01 rows=1 width=146) (actual time=0.002..0.002 rows=1 loops=89)
                     Index Cond: ((mt_matter_id = products_matters.pm_matter_id) AND (mt_matter_update = products_matters.pm_matter_update))
         ->  Index Scan using products_pkey on products  (cost=0.28..0.48 rows=1 width=82) (actual time=0.001..0.001 rows=1 loops=89)
               Index Cond: ((pr_product_code = products_matters.pm_product_code) AND (pr_last_updated_when = products_matters.pm_product_update))
 Planning Time: 41.459 ms
 Execution Time: 28.164 ms
(27 rows)
```

結果：効果あり

チューニング手順④ 検証 その他

- SQL 書き換え
  - 設定変更や別テーブルを利用せずに
    実行計画に変化をもたらす SQL を検討
  - 今回の場合
    - GroupAggregate の Loop 数を減らしたい

- 外部ツール
  - 統計情報や実行計画に介入するためのツール利用



今回の場合既に効果ありの対応策がいくつか確認できたため検証無し

チューニング手順⑤ 結果

- 一時テーブル採用

BEFORE

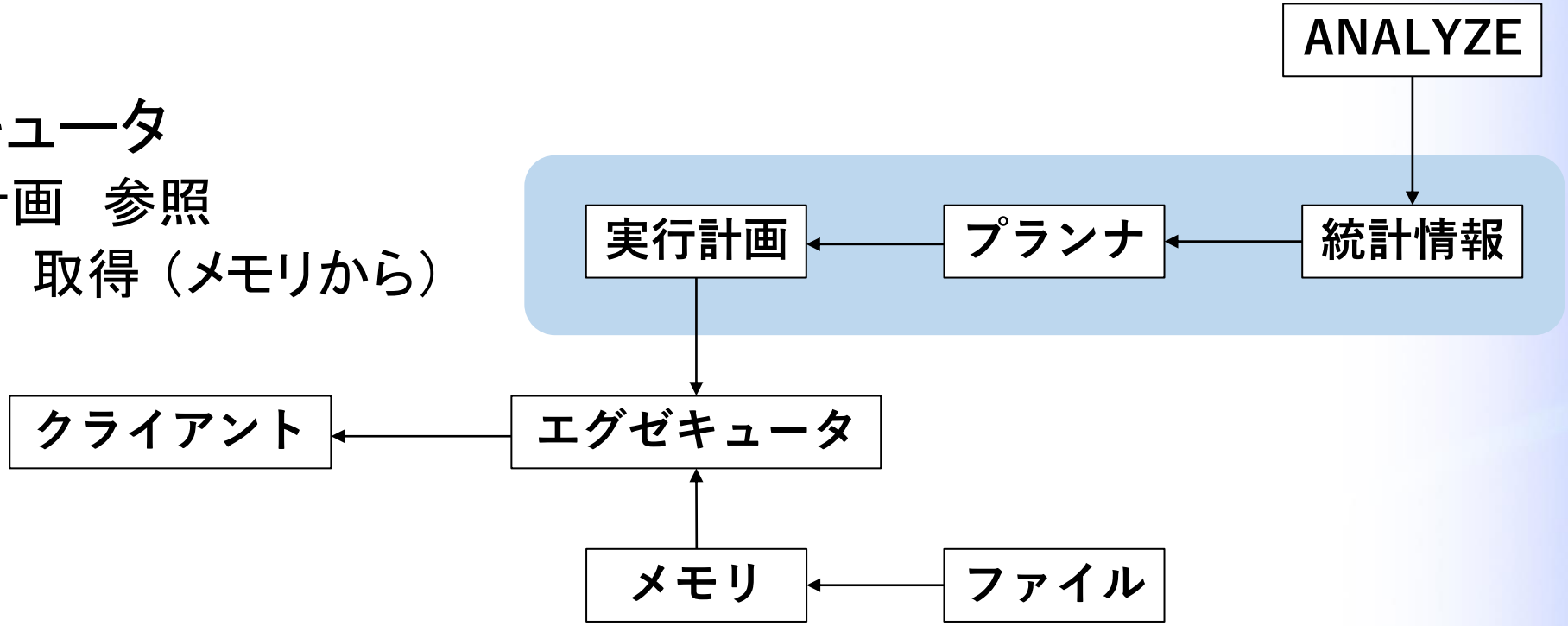
```
Sort  (cost=16130.86..16130.87 rows=1 width=233) (actual time=30570.523..30570.533 rows=89 loops=1)
  Sort Key
  Sort Met
  ->  Nest                                                                                                    l time=1395.616..30569.236 rows=89 loops=1)
        ->  Nested Loop  (cost=1405.14..16139.26 rows=1 width=206) (actual time=1395.584..30567.722 rows=89 loops=1)
              Join
              Rows
              ->  N
00:00:00'::timestamp
(COALESCE(pm_orderi
        ->  GroupAggregate  (cost=0.42..7383.75 rows=5030 width=12) (actual time=0.062..25.890 rows=4896 loops=1164)
              Group Key: matters_1.mt_matter_id
              ->  Index Only Scan using matters_pkey on matters matters_1  (cost=0.42..6649.38 rows=136814 width=12) (actual time=0.005..14.681 rows=134350 loops=1164)
                    Index Cond: (mt_matter_update < '2024-05-02 00:00:00'::timestamp without time zone)
                    Heap Fetches: 0
        ->  Index Scan using products_pkey on products  (cost=0.28..0.48 rows=1 width=82) (actual time=0.009..0.009 rows=1 loops=89)
              Index Cond: ((pr_product_code = products_matters.pm_product_code) AND (pr_last_updated_when = products_matters.pm_product_update))
Planning Time: 16.443 ms
Execution Time: 30571.924 ms
```

AFTER

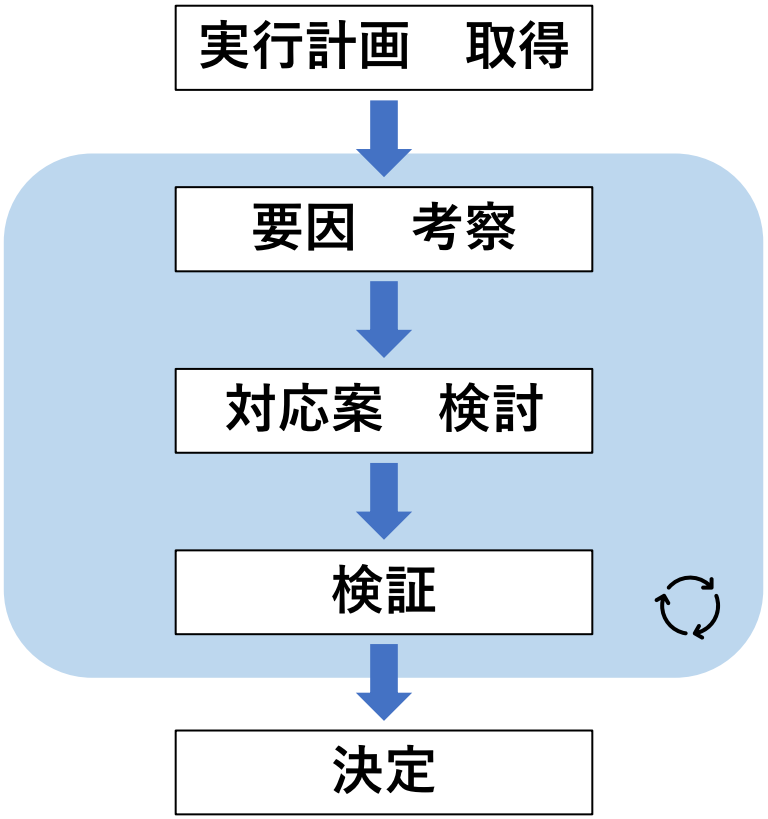
```
Sort  (cost=4819.90..4819.91 rows=2 width=233) (actual time=28.042..28.052 rows=89 loops=1)
  Sort Key: (COALESCE(products_matters.pm_ordering_or_lost_date, products_matters.pm_ordering_schedule)) DESC, matters.mt_matter_id, products_matters.pm_matter_product_no
  Sort Method: quicksort  Memory: 47kB
  ->  Nested Loop  (cost=115.08..4819.89 rows=2 width=233) (actual time=26.387..27.931 rows=89 loops=1)
        ->  Nested Loop  (cost=114.81..4819.40 rows=1 width=206) (actual time=26.370..27.742 rows=89 loops=1)
              ->  Nested Loop  (cost=114.39..4813.39 rows=1 width=84) (actual time=26.355..27.482 rows=89 loops=1)
                    Join Filter: ((products_segments.ps_product_code = products_matters.pm_product_code) AND (products_segments.ps_product_update = products_matters.pm_product_update))
                    Rows Removed by Join Filter: 7455
                    ->  Index Only Scan using products_segments_pkey on products_segments  (cost=0.29..17.07 rows=23 width=18) (actual time=0.032..0.076 rows=46 loops=1)
                          Index Cond: ((ps_segment_id = 3) AND (ps_segment_class_id = 7))
                          Heap Fetches: 0
                    ->  Materialize  (cost=114.10..4774.32 rows=55 width=66) (actual time=0.037..0.571 rows=164 loops=46)
                          ->  Nested Loop  (cost=114.10..4774.05 rows=55 width=66) (actual time=1.711..25.807 rows=164 loops=1)
                                ->  HashAggregate  (cost=113.68..119.39 rows=571 width=12) (actual time=1.643..2.668 rows=5030 loops=1)
                                      Group Key: tmp_20240508.tmp_id, tmp_20240508.tmp_update
                                      Batches: 1  Memory Usage: 465kB
                                      ->  Seq Scan on tmp_20240508  (cost=0.00..85.12 rows=5712 width=12) (actual time=0.018..0.483 rows=5030 loops=1)
                                ->  Index Scan using products_matters_pkey on products_matters  (cost=0.42..8.14 rows=1 width=54) (actual time=0.004..0.004 rows=0 loops=5030)
                                      Index Cond: ((pm_matter_id = tmp_20240508.tmp_id) AND (pm_matter_update = tmp_20240508.tmp_update))
                                      Filter: (('2024-04-01'::date <= COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule)) AND
                                              (COALESCE(pm_ordering_or_lost_date, pm_ordering_schedule) < '2024-05-01 00:00:00'::timestamp without time zone) AND (pm_probability_id = 7))
                                      Rows Removed by Filter: 3
              ->  Index Scan using matters_pkey on matters  (cost=0.42..6.01 rows=1 width=146) (actual time=0.002..0.002 rows=1 loops=89)
                    Index Cond: ((mt_matter_id = products_matters.pm_matter_id) AND (mt_matter_update = products_matters.pm_matter_update))
        ->  Index Scan using products_pkey on products  (cost=0.28..0.48 rows=1 width=82) (actual time=0.001..0.001 rows=1 loops=89)
              Index Cond: ((pr_product_code = products_matters.pm_product_code) AND (pr_last_updated_when = products_matters.pm_product_update))
Planning Time: 41.459 ms
Execution Time: 28.164 ms
```

まとめ① SQL 実行仕組み

- プランナ
  - 統計情報　参照
  - 実行計画　生成
- エグゼキュータ
  - 実行計画　参照
  - データ　取得（メモリから）



まとめ② チューニング手順



まとめ③ 実行計画（1）

- SQL の内部処理方式/順序
  - プランナが統計情報と元に作成したもの
  - EXPLAIN を使って確認可

```
=$ EXPLAIN 《SQL》
```

- ツリー構造
  - 下の階層から実行
- 計画ノード
  - 計画タイプ　Hash Join、Index Scan など
  - コスト（cost）　≠ 実行時間
  - 行数（rows）　計画ノード完了時点で取得される推定行数
  - サイズ（width）　取得される行の推定平均サイズ（バイト）

コスト、実行時間には
下位階層の値が含まれる

- ANALYZE で以下（actual）追加
  - 時間（time）　実行時間（ミリ秒）
  - 行数（rows）　実際に取得された行数
  - 回数（loops）　対象の処理を繰り返し実行した回数
  - 実行計画作成時間（Planning time）
  - 実行時間（Execution time）

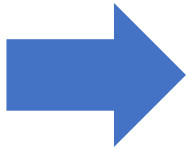
実行時間と行数には
回数を乗じる

まとめ③ 実行計画（2）

- 注目ポイント
  - rows
    - EXPLAIN ANALYZE を使っている場合
    - 推測と実際とに大きな差が無いか
    - 早い段階で絞込できているか
  - cost ／ actual time
    - どの処理に大きなコスト/時間を要しているか
    - loops にも注意 （EXPLAIN ANALYZE を使っている場合）
  - 計画タイプ
    - インデックスを使えているか
    - テーブル、データ分布に適した処理内容になっているか

まとめ④ 対応策（1）

- 統計情報
  - 統計情報が古い
  - 予測行数と実際とに大きな差
  - 計画タイプに改善余地



  - VACUUM / ANALYZE
  - default_statistics_target
    - 統計情報の精度
    - カラム毎に設定可能

- 補足
  - パーティションテーブル
    - 自動バキュームでは ANALYZE が実行されない
  - 拡張統計情報
    - 複数行にまたがる統計情報作成可

まとめ④ 対応策（2）

- インデックス
  - スキャン性能向上
    - 結合条件、検索条件、ソート条件等に利用されている列に付与
    - 慎重に検討、検証が必要
  - 多種多様
    - B-Tree　－　もっとも一般的
    - BRIN / GIN / Hash / GiST　など、、

    - 複数列インデックス
    - 式インデックス
    - 部分インデックス

注意：
・挿入、更新処理へのオーバーヘッドあり
・インデックス自体のサイズ肥大化が問題になるケースあり

[PostgreSQL 16.0文書 － 第11章 インデックス]

https://www.sraoss.co.jp/PostgreSQL/Manual/document/16/indexes.html

まとめ④ 対応策（3）

- パラメータ調整
  - 計画タイプ
    - enable_indexscan
    - enable_seqscan
    - enable_hashjoin
    - enable_mergejoin

    など…
  - 結合順序
    - join_collapse_limit
    - from_collapse_limit
  - プランナコスト定数
    - seq_page_cost
    - random_page_cost
    - effective_cache_size

    など…
  - メモリ
    - shared_buffers
    - work_mem

まとめ④ 対応策（4）

- SQL 書き換え
  - 同じ結果を得るためにも多様な書き方があり
    コスト削減できる可能性
  - 考え方例
    - インデックス使用
    - 早い段階で行数を絞り込み
    - 相関サブクエリ回避
    - IN ⇔ EXISTS ⇔ JOIN

まとめ④ 対応策（5）

- 外部ツール
  - pg_hint_plan
    - 実行計画制御ツール
    - 「ヒント」を指定し、プランナを誘導
  - pg_dbms_stats
    - 統計情報管理ツール
    - 統計情報固定可能
    - 実行計画の安定化

SRA OSS の TechBlog に
紹介ページがあるので
是非ご参照ください。

まとめ⑤ 最後に

- SQL チューニング
  - SQL 単体の性能を要件を満たす状態まで向上させること
  - 実行計画から要因を探り検証を繰り返す
- PostgreSQL では基本的にプランナ任せ
  - 外部ツール無しでは実行計画に介入する方法は少ない
  - プランナの方が優秀
- サービスとしての性能チューニングであれば他要因も多種多様
  - ロック待ち
  - 内部処理
  - テーブル設計
  - マシンスペック　など…

実施するコスト、期待できる効果、期待どおりの効果となる可能性の考慮が必要

ご清聴ありがとうございました。