PostgreSQLの力を引き出す！
拡張機能で実現する
高度なデータベース管理

2024年9月7日
Open Developers Conference 2024

株式会社SRA OSS
馬 雪テイ

自己紹介

名前: 馬 雪テイ（MA XUETING）

所属: システム・インフラ開発室

担当業務:

- PostgreSQLサポート
- PostgreSQL HAクラスタ構築・コンサルティング
- 社内システム開発

会社紹介

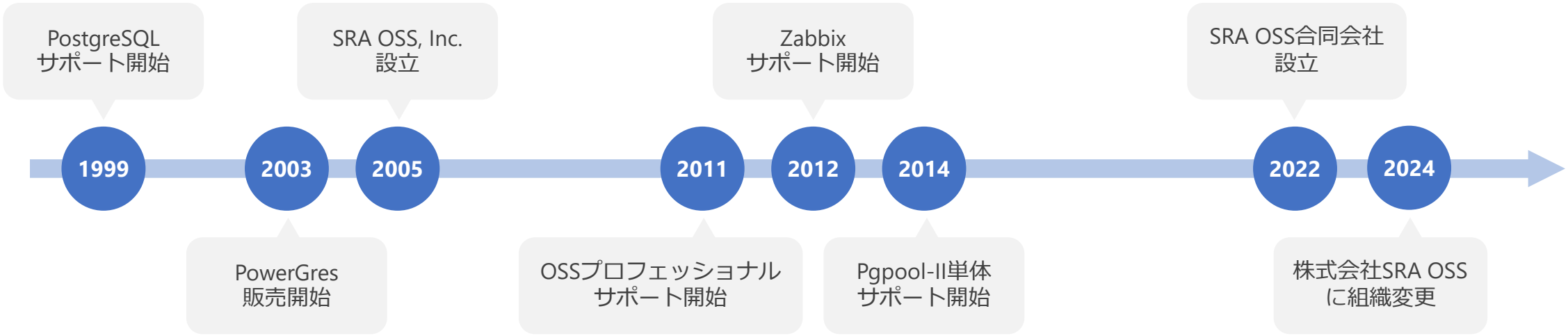
株式会社SRA OSS

所 在 地: 東京都豊島区南池袋2-32-8
設 立 日: 2022年6月17日
株　　主: 株式会社SRA
株式会社NTTデータグループ
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)



概要

- 拡張機能とは何か
- PostgreSQLの拡張機能の利点
- 応用例を紹介
  - pg_hint_plan: クエリの実行計画の制御
  - pg_stat_statements:クエリパフォーマンスのモニタリング
  - pg_rman: バックアップ管理
  - pgaudit: 監査ログ機能

拡張機能とは何か

拡張機能とは何か

- データベースの機能を追加・強化するモジュール
- 要件や業務に応じて機能を拡張・補足するオープンソースソフトウェアを組み合わせて利用する
- 周辺OSSは、PostgreSQLのコミュニティーとは別に、独自のコミュニティーによって活発に開発が行われる

PostgreSQLの拡張機能の利点

PostgreSQL拡張機能の利点

機能の追加

カスタマイズ性

簡単な導入と管理

性能向上

コミュニティの支援

応用例を紹介

環境：RHEL 9.4

PostgreSQLバージョン： PostgreSQL 16.0

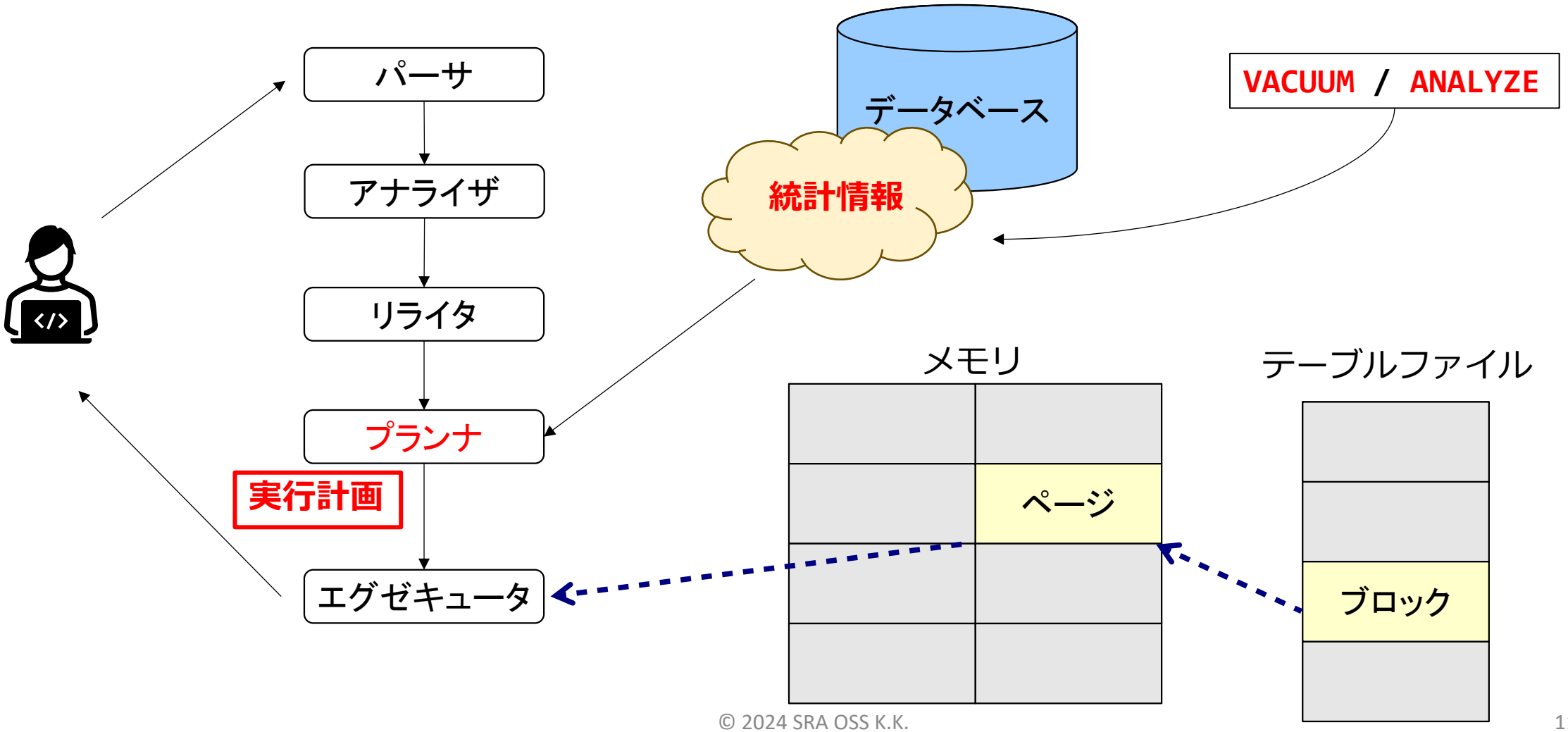
pg_hint_planの紹介

- pg_hint_plan とは
- ヒント句を使ってSQLの実行計画をクライアントが制御できる

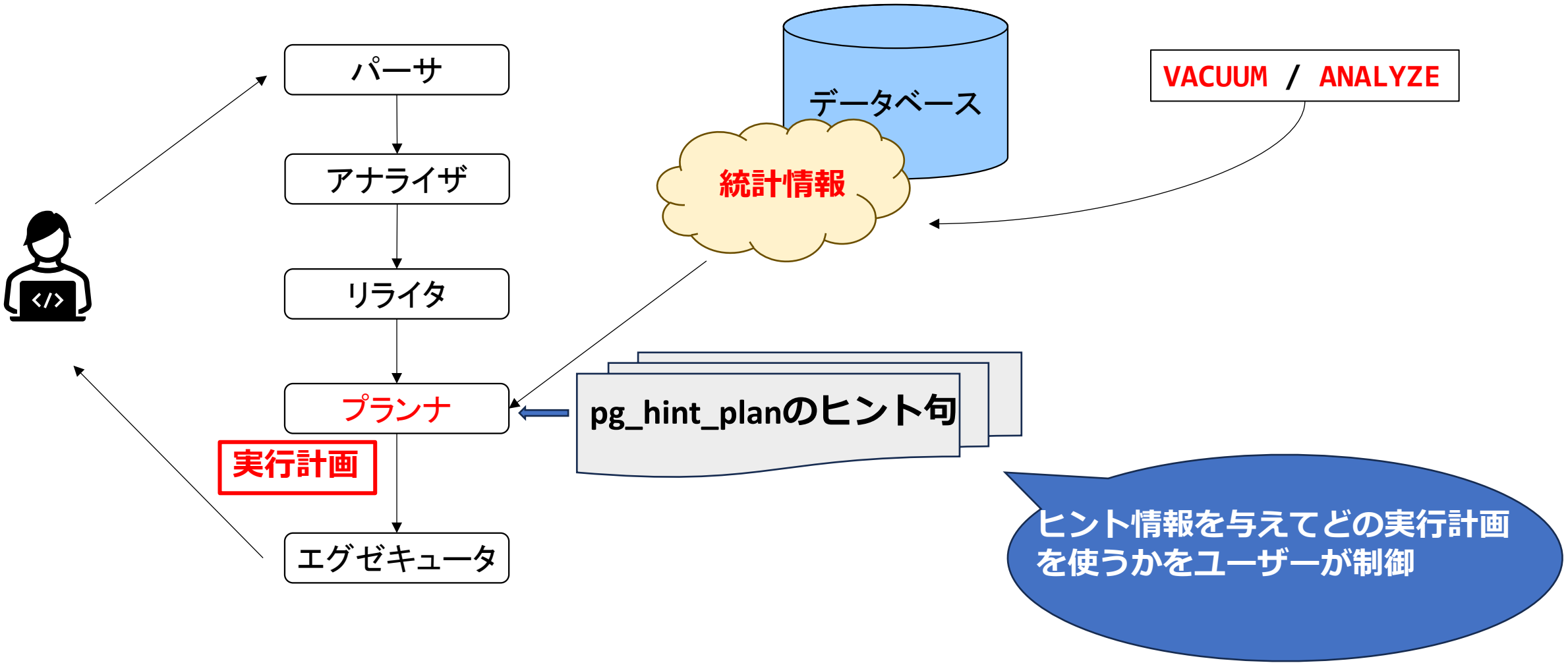
・用途

- ● チューニング
- ● 性能の速度より性能の安定を重視
- ● 実行計画の変化による突然の性能変動を避けたい

SQL 実行仕組み



SQL 実行仕組み（pg_hint_planを利用）



運用：pg_hint_plan

・インストール

- 現時点 PostgreSQL 16.x に対応した 1.6.0 がリリースされている
- PostgreSQL 9.1 以降のそれぞれのバージョンに対応する pg_hint_plan バージョンがリリースされている
- Windows むけのバイナリは公式サイトからリリースされていないが、Windows むけのビルドは可能

➡ **SRAOSSでは、Windowsバイナリをサポート契約者に無償で提供している**

(REHL9で pg_hint_plan 1.6.0 を導入する例を示す)

```
# dnf install pg_hint_plan16-1.6.0-1.el9.x86_64.rpm
もしくは
# rpm -ivh pg_hint_plan16-1.6.0-1.el9.x86_64.rpm
```

運用：pg_hint_plan

・**設定**

- 利用したいデータベースでエクステンション登録

```
db1=# CREATE EXTENSION pg_hint_plan;
```

- 全セッションで利用する場合

＊ 設定postgresql.confを修正（編集後PostgreSQLを再起動）

```
shared_preload_libraries = 'pg_hint_plan'
```

補足：DB接続セッションで pg_hint_plan をロードすることも可能

```
db1=# LOAD 'pg_hint_plan';
```

運用：pg_hint_plan

・指定できるヒントの種類

スキャン方式	テーブルのスキャン方式を指定
結合順序	複数テーブルをどの順で結合するか指定
結合方式	テーブル結合の方式を指定
見積件数補正	テーブル結合結果の件数見積もりを補正
パラレル実行	パラレル実行の強制or禁止
設定パラメータ	SET コマンドで指定できるパラメータを設定
結合時の挙動制御	指定したテーブル間の結合時において、 最上位の内部結合の計画の結果の キャッシング(Memoize)の有効/無効を 指定する

運用：pg_hint_plan

- **ヒント句の指定方法**
  - SQL にコメント指定：
    「**/*+ ヒント */ クエリ**」というフォーマット

  - ヒント用テーブルに登録：
    hint_plan.hints テーブルにデータを **INSERT** する

SRA OSS 運用：pg_hint_plan

- サンプル：SQLヒント句を指定

```
db1=# EXPLAIN ANALYZE
      SELECT * FROM pgbench_branches b
        JOIN pgbench_accounts a ON b.bid = a.bid ORDER BY a.aid;
```

```
db1=# EXPLAIN ANALYZE
        SELECT * FROM pgbench_branches b
          JOIN pgbench_accounts a ON b.bid = a.bid ORDER BY a.aid;
                                                                            QUERY PLAN

--------------------------------------------------------------------------------------------------------------------------
-------------------------------------------
 Nested Loop  (cost=0.57..66151.81 rows=1000000 width=461) (actual time=0.427..567.363 rows=1000000 loops=1)
   ->  Index Scan using pgbench_accounts_pkey on pgbench_accounts a  (cost=0.42..42377.43 rows=1000000 width=97) (actua
l time=0.386..206.943 rows=1000000 loops=1)
   ->  Memoize  (cost=0.15..0.16 rows=1 width=364) (actual time=0.000..0.000 rows=1 loops=1000000)
         Cache Key: a.bid
         Cache Mode: logical
         Hits: 999990  Misses: 10  Evictions: 0  Overflows: 0  Memory Usage: 2kB
         ->  Index Scan using pgbench_branches_pkey on pgbench_branches b  (cost=0.14..0.15 rows=1 width=364) (actual t
ime=0.008..0.008 rows=1 loops=10)
               Index Cond: (bid = a.bid)
 Planning Time: 0.617 ms
 Execution Time: 600.809 ms
(10 行)
```

- ヒントを指定しない場合は Index Scan, Nested Loop が計画されている

運用：pg_hint_plan

```
db1=# /*+ HashJoin(a b) SeqScan(a) */
      EXPLAIN ANALYZE
        SELECT * FROM pgbench_branches b
          JOIN pgbench_accounts a ON b.bid = a.bid ORDER BY a.aid;
```

```
db1=# /*+ HashJoin(a b) SeqScan(a) */
      EXPLAIN ANALYZE
        SELECT * FROM pgbench_branches b
          JOIN pgbench_accounts a ON b.bid = a.bid ORDER BY a.aid;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 Gather Merge  (cost=235759.02..332988.11 rows=833334 width=461) (actual time=789.455..1196.567 rows=1000000 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Sort  (cost=234759.00..235800.67 rows=416667 width=461) (actual time=622.244..665.686 rows=333333 loops=3)
         Sort Key: a.aid
         Sort Method: external merge  Disk: 49456kB
         Worker 0:  Sort Method: external merge  Disk: 31744kB
         Worker 1:  Sort Method: external merge  Disk: 31608kB
         ->  Hash Join  (cost=1.23..22119.19 rows=416667 width=461) (actual time=0.192..273.873 rows=333333 loops=3)
               Hash Cond: (a.bid = b.bid)
               ->  Parallel Seq Scan on pgbench_accounts a  (cost=0.00..20560.67 rows=416667 width=97) (actual time=0.04
1..66.432 rows=333333 loops=3)
               ->  Hash  (cost=1.10..1.10 rows=10 width=364) (actual time=0.045..0.047 rows=10 loops=3)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on pgbench_branches b  (cost=0.00..1.10 rows=10 width=364) (actual time=0.015..0.018 r
ows=10 loops=3)
 Planning Time: 41.768 ms
 Execution Time: 1236.031 ms
(16 行)
```

- HashJoin(a b): aテーブルとbテーブルの結合にHashJoinを使う
- SeqScan(a): a テーブルをSeqScanする

運用：pg_hint_plan

- サンプル：ヒント用テーブルに登録

```
db1=# INSERT INTO hint_plan.hints
      (norm_query_string, application_name, hints)
      VALUES (
       'EXPLAIN ANALYZE
        SELECT * FROM pgbench_branches b
        JOIN pgbench_accounts a ON b.bid = a.bid
        ORDER BY a.aid;',
       '', 'HashJoin(a b) SeqScan(a)');
```

- norm_query_string: クエリ文

- application_name:ヒント適用対象のアプリケーション名（空文字の場合は全てのアプリケーションにヒントを適用する）

- hints: hint句

運用：pg_hint_plan

```
db1=# EXPLAIN ANALYZE
      SELECT * FROM pgbench_branches b
       JOIN pgbench_accounts a ON b.bid = a.bid ORDER BY a.aid;
```

```
db1=# EXPLAIN ANALYZE
          SELECT * FROM pgbench_branches b
          JOIN pgbench_accounts a ON b.bid = a.bid
          ORDER BY a.aid;
                                                              QUERY PLAN

----------------------------------------------------------------------------------------------------------------
------------------------------------
 Gather Merge  (cost=235759.02..332988.11 rows=833334 width=461) (actual time=645.403..1100.826 rows=1000000 loops=
1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Sort  (cost=234759.00..235800.67 rows=416667 width=461) (actual time=620.815..664.972 rows=333333 loops=3)
         Sort Key: a.aid
         Sort Method: external merge  Disk: 40088kB
         Worker 0:  Sort Method: external merge  Disk: 36360kB
         Worker 1:  Sort Method: external merge  Disk: 36376kB
         ->  Hash Join  (cost=1.23..22119.19 rows=416667 width=461) (actual time=0.171..336.102 rows=333333 loops=3
)
               Hash Cond: (a.bid = b.bid)
               ->  Parallel Seq Scan on pgbench_accounts a  (cost=0.00..20560.67 rows=416667 width=97) (actual time
=0.045..95.153 rows=333333 loops=3)
               ->  Hash  (cost=1.10..1.10 rows=10 width=364) (actual time=0.033..0.034 rows=10 loops=3)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on pgbench_branches b  (cost=0.00..1.10 rows=10 width=364) (actual time=0.016..0.
018 rows=10 loops=3)
 Planning Time: 0.593 ms
 Execution Time: 1143.464 ms
```

運用：pg_hint_plan

- 注意点

- ヒント句は基本的にSQL文の先頭に配置
- テーブル名に別名が付いている場合、ヒント句内では別名を使う必要がある
- 「コメントでの指定」と「テーブルでの指定」の両方で指定した場合、「テーブルでの指定」が優先される
- 指示対象クエリ中の定数は？で指定する必要がある

○ `SELECT * FROM table_1 t1 WHERE t1.id = ?;`

× `SELECT * FROM table_1 t1 WHERE t1.id = 500000;`

- ヒント用のテーブルに登録したSQL文と実行されるSQL文がスペース・改行も含めて完全合致でないとヒントは適用されない

pg_stat_statementsの活用

- pg_stat_statementsとは
  - 実行されたSQLの種類、実行回数、実行時間などを集計・記録してくれる

・用途
- ● チューニングの起点
- ● ボトルネックとなっている SQL を見つけたい
- ● PostgreSQL ログ を大量に出したくない

運用： pg_stat_statements

・インストール

- contribから入手

```
# dnf install postgresql16-contrib
```

- モジュールの確認

```
# SELECT * FROM pg_available_extensions WHERE name LIKE 'pg_stat_statements';
```

```
db1=# select * from pg_available_extensions where name like 'pg_stat_statements';
        name        | default_version | installed_version |                              comment
--------------------+-----------------+-------------------+-----------------------------------------------------------------
 pg_stat_statements | 1.10            | 1.10              | track planning and execution statistics of all SQL statements executed
(1 行)
```

運用： pg_stat_statements

・**設定**

- 利用したいデータベースでエクステンション登録

```
db1=# CREATE EXTENSION pg_stat_statements;
```

- postgresql.conf で設定

```
shared_preload_libraries = 'pg_stat_statements'
```

＊ **PostgreSQL 起動時に pg_stat_statements をロード**

運用：pg_stat_statements

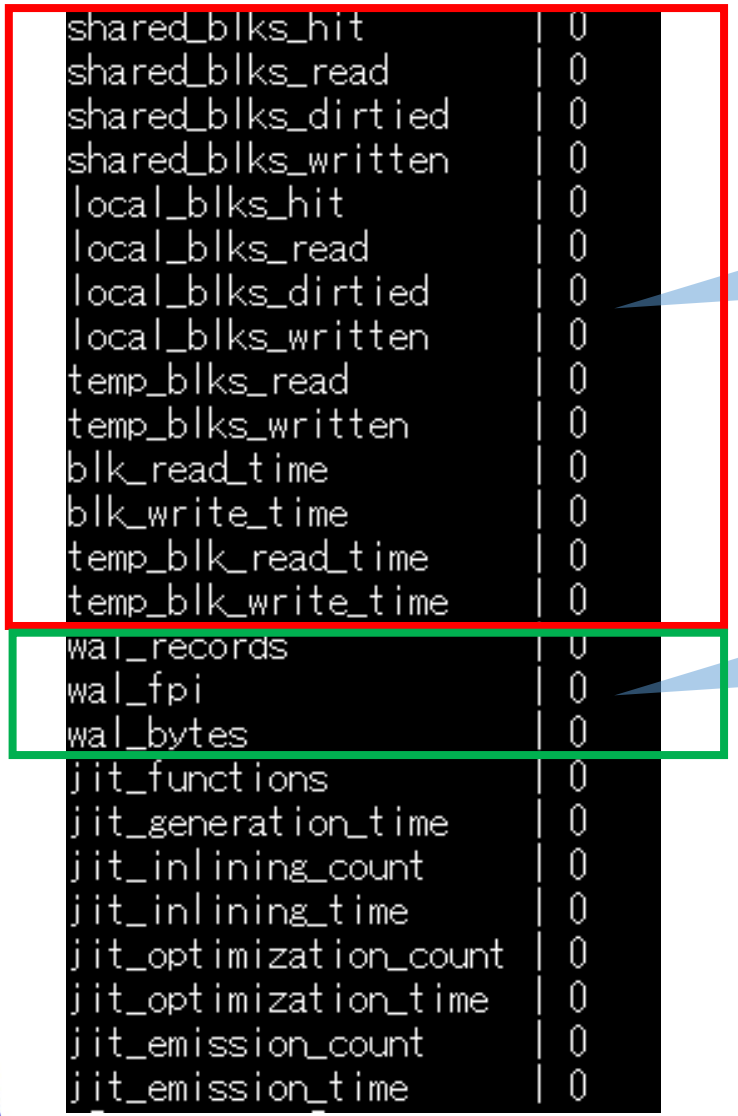
```
db1=# SELECT * FROM pg_stat_statements;
-[ RECORD 1 ]---------+----------------------------------
---
userid                | 10
dbid                  | 16384
toplevel              | t
queryid               | 2169608607147504250
query                 | SELECt * FROM pg_stat_statements
plans                 | 0
total_plan_time       | 0
min_plan_time         | 0
max_plan_time         | 0
mean_plan_time        | 0
stddev_plan_time      | 0
calls                 | 2
total_exec_time       | 0.639119
min_exec_time         | 0.227256
max_exec_time         | 0.411863
mean_exec_time        | 0.3195595
stddev_exec_time      | 0.09230349999999998
rows                  | 7
```

SQLのプラン生成の情報
プラン生成回数
プラン生成にかかる時間（総時間、最小時間、最大時間、平均時間、母標準偏差）

→ **PostgreSQL13以降**

SQLの実行情報
SQLの実行回数
SQLの実行時間（総時間、最小時間、最大時間、平均時間、母標準偏差）、影響を受けた行の総数

運用：pg_stat_statements



shared_blks_hit	0
shared_blks_read	0
shared_blks_dirtied	0
shared_blks_written	0
local_blks_hit	0
local_blks_read	0
local_blks_dirtied	0
local_blks_written	0
temp_blks_read	0
temp_blks_written	0
blk_read_time	0
blk_write_time	0
temp_blk_read_time	0
temp_blk_write_time	0
wal_records	0
wal_fpi	0
wal_bytes	0
jit_functions	0
jit_generation_time	0
jit_inlining_count	0
jit_inlining_time	0
jit_optimization_count	0
jit_optimization_time	0
jit_emission_count	0
jit_emission_time	0

SQLのブロックアクセス情報

ブロックアクセスでディスクリードした回数、キャッシュヒットした回数など

SQLのWAL出力情報

出力されたWALレコードの総数、WALフルページイメージの総数、バイト単位のWAL総量

➡ **PostgreSQL13以降**

運用：pg_stat_statements

- サンプル

```
db1=# SELECT query, calls, total_exec_time, rows, 100.0 * shared_blks_hit /
             nullif(shared_blks_hit + shared_blks_read, 0) AS hit_percent
        FROM pg_stat_statements ORDER BY total_exec_time DESC LIMIT 5;
                                                  query                                                  | calls |  total_exec_time   |  rows  |     hit_percent
---------------------------------------------------------------------------------------------------------+-------+--------------------+--------+---------------------
 UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE bid = $2                                     |  3000 | 6608.4296170000025 |   3000 |  99.9983984881728352
 UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE tid = $2                                      |  3000 |  3890.484274000001 |   3000 |  99.9953776462974947
 copy pgbench_accounts from stdin with (freeze on)                                                       |     1 |            161.574737 | 100000 | 100.0000000000000000
 alter table pgbench_branches add primary key (bid)                                                      |     1 |            113.068876 |      0 |  91.8604651162790698
 UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE aid = $2                                     |  3000 |  81.25760399999996 |   3000 |  98.7514755289203668
(5 行)
```

- クエリの実行時間が最も長い上位5つのSQLクエリを取得

- 各クエリの実行回数、総実行時間、影響を受けた行数、およびキャッシュヒット率が表示される

運用： pg_stat_statements

- 関数

- pg_stat_statements_reset(userid oid, dbid oid, queryid bigint) returns void

＊ 指定されたuserid、dbid、queryid に対応するpg_stat_statementsによってこれまでに収集したすべての統計情報を削除する

```
db1=# SELECT pg_stat_statements_reset(0,0,s.queryid) FROM pg_stat_statements AS s
           WHERE s.query = 'UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE bid = $2';
 pg_stat_statements_reset
--------------------------

(1 行)

db1=# SELECT query, calls, total_exec_time, rows, 100.0 * shared_blks_hit /
              nullif(shared_blks_hit + shared_blks_read, 0) AS hit_percent
         FROM pg_stat_statements ORDER BY total_exec_time DESC LIMIT 5;
                                        query                                         | calls |  total_exec_time  |  rows  |     hit_percent
--------------------------------------------------------------------------------------+-------+-------------------+--------+---------------------
 UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE tid = $2                   |  3000 | 3890.484274000001 |   3000 |  99.9953776462974947
 copy pgbench_accounts from stdin with (freeze on)                                    |     1 |         161.574737 | 100000 | 100.0000000000000000
 alter table pgbench_branches add primary key (bid)                                   |     1 |         113.068876 |      0 |  91.8604651162790698
 UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE aid = $2                  |  3000 | 81.25760399999996 |   3000 |  98.7514755289203668
 drop table if exists pgbench_accounts, pgbench_branches, pgbench_history, pgbench_tellers |     1 |         55.714187 |      0 |  97.5483870967741935
(5 行)
```

運用： pg_stat_statements

*すべての指定されたパラメータが0（無効）ならば、すべての統計情報を削除する

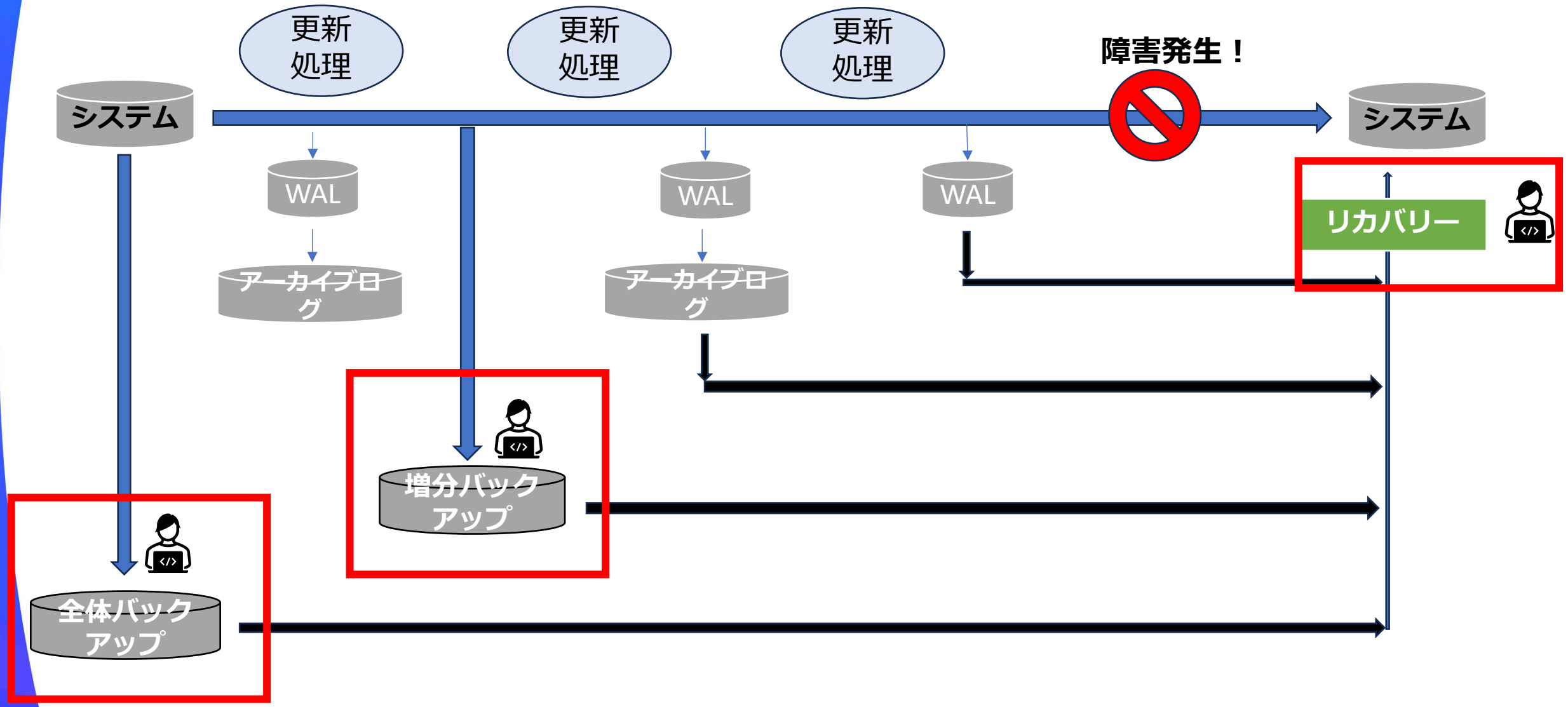
```
db1=# SELECT pg_stat_statements_reset(0,0,0);
 pg_stat_statements_reset
--------------------------

(1 行)

db1=# SELECT query, calls, total_exec_time, rows, 100.0 * shared_blks_hit /
               nullif(shared_blks_hit + shared_blks_read, 0) AS hit_percent
          FROM pg_stat_statements ORDER BY total_exec_time DESC LIMIT 5;
                  query                    | calls | total_exec_time | rows | hit_percent
-------------------------------------------+-------+-----------------+------+-------------
 SELECT pg_stat_statements_reset(0,0,0) |     1 |        0.603828 |    1 |
(1 行)
```

pg_rmanによるバックアップ管理

- pg_rmanとは
  - PostgreSQL のバックアップ/リストア管理ツール

- 機能
  - 全体バックアップに加え、増分バックアップが可能
  - バックアップの圧縮が可能
  - バックアップの世代管理やバックアップ一覧を表示できる
  - バックアップの対象はデータベースクラスタの他にアーカイブログ、サーバログが含めることができる
  - タイムライン指定、リカバリしたい日付時刻指定が可能
  - データベースクラスタ外のテーブルスペースを含めたバックアップが可能

バックアップ・リカバリーの流れ



運用：pg_rman

・インストール

Windows には対応してない

- rpmパッケージをインストール

```
#  rpm -ivh pg_rman-1.3.16-1.pg16.rhel9.x86_64.rpm
```

- postgresql.confを設定

```
$ vi postgresql.conf
archive_mode = on
archive_command = 'cp %p /mnt/arch/%f'
```

- 環境変数を設定

```
$ export BACKUP_PATH=<バックアップカタログパス>
```

運用：pg_rman

- 初期化

```
$ pg_rman init
```

- 全体バックアップを取得

```
$ pg_rman backup --backup-mode=full
```

```
INFO: copying database files
INFO: copying archived WAL files
INFO: backup complete
INFO: Please execute 'pg_rman validate' to verify the files are correctly copied.
```

- バックアップ検証

```
$ pg_rman validate
```

運用：pg_rman

- 増分バックアップ

```
$ pg_rman backup --backup-mode=incremental --with-serverlog
```

```
INFO: copying database files
INFO: copying archived WAL files
INFO: copying server log files
INFO: backup complete
INFO: Please execute 'pg_rman validate' to verify the files are correctly copied.
```

```
$ pg_rman validate
```

- バックアップ一覧

```
$ pg_rman show
```

```
==========================================================================
 StartTime           EndTime             Mode   Size    TLI   Status
==========================================================================
2024-08-03 19:33:09  2024-08-03 19:33:12 INCR   33MB    1     OK
2024-08-03 19:31:54  2024-08-03 19:32:11 FULL   173MB   1     OK
```

未検証の場合はDONEが表示される

運用：pg_rman

・リストア

```
$ pg_rman restore
```

```
$ pg_rman restore --recovery-target-time '2024-08-03 19:33:00'
```

・バックアップの情報の詳細表示

```
$ pg_rman show detail
```

```
========================================================================================
StartTime           EndTime             Mode    Data ArcLog  SrvLog   Total  Compressed  CurTLI  ParentTLI  Status
========================================================================================
2024-08-03 19:33:09  2024-08-03 19:33:12  INCR    16kB    33MB    21kB    33MB      false      1        0  OK
2024-08-03 19:31:54  2024-08-03 19:32:11  FULL   155MB    33MB    ----   173MB      false      1        0  OK
```

```
$ pg_rman show '2024-08-03 19:31:54'
```

運用：pg_rman

・バックアップの削除

```
$ pg_rman delete <リストアが必要な日付時刻>
```

- delete コマンドは指定した日付時刻のバックアップを削除するのではなく、
  指定した日付時刻へのリカバリに不必要なバックアップを削除する

・世代管理

- 保持期限を過ぎると削除する

```
$ pg_rman backup -b full -Z --keep-data-generations=3 --keep-data-days=10
```

```
$ pg_rman backup -b incremental -Z --keep-data-generations=3 --keep-data-days=10
```

pg_basebackup との比較

- pg_rman が優れている点
  - バックアップの世代管理が可能
  - アーカイブログ、サーバログもバックアップ対象にできる
  - 増分バックアップが可能
  - PostgreSQL 8.2 から対応
- pg_basebackup が優れている点
  - リモート操作が可能
  - PostgreSQLの標準コマンドである

pgAuditでセキュリティを強化

- pgAudit とは
  - PostgreSQL で監査ログを取得するためのオープンソースソフトウェア
  - 監査する対象として設定した内容は、PostgreSQL ログに混在して出力される

pgAudit to Log Fileで監査と通常ログを分離できる

・用途

- ●定期的に不正なアクセスが発生していないか、その予兆がないかをチェック
- ●問題の検出および対策、もしくは予防措置

運用： pgAudit

・インストール

- rpmパッケージ：PostgreSQL 公式リポジトリに含まれる

```
# dnf install -y pgaudit_16
```

pgAudit バージョン	サポート対象 PostgreSQL バージョン
pgAudit v16.X	PostgreSQL 16
pgAudit v15.X	PostgreSQL 15
pgAudit v14.X	PostgreSQL 14
pgAudit v1.5.X	PostgreSQL 13
pgAudit v1.4.X	PostgreSQL 12
pgAudit v1.3.X	PostgreSQL 11
pgAudit v1.2.X	PostgreSQL 10
pgAudit v1.1.X	PostgreSQL 9.6
pgAudit v1.0.X	PostgreSQL 9.5

＊ Windows 向けには公式のバイナリ提供はないが、ビルドは可能

➡ **SRAOSSでは、Windowsバイナリをサポート契約者に無償で提供している**

運用： pgAudit

・**設定**

- 利用したいデータベースでエクステンション登録

```
db1=# CREATE EXTENSION pgaudit;
```

- postgresql.conf で設定

```
shared_preload_libraries = 'pgaudit'
```

＊ **PostgreSQL 起動時に pgauditをロード**

運用： pgAudit

・セッション監査とオブジェクト監査の違い

監査方法	監査の主体	監査の粒度	監査対象のDBユーザ
オブジェクト監査	テーブル	テーブル、列単位	全DBユーザ
セッション監査	DBユーザ	DBユーザ単位	監査設定を行ったDBユーザのみ

運用： pgAudit

・pgAudit を使ってみる（オブジェクト監査）

- auditmrole マスターロールの作成

```
postgres=# CREATE ROLE auditmrole;
```

- pgaudit.role の設定

```
$ cd $PGDATA
$ vi postgresql.conf
pgaudit.role = 'auditmrole'
$ pg_ctl reload
```

- マスターロールへの権限の付与 ⟹

```
postgres=# CREATE TABLE table1 (id int, t1 text);
CREATE TABLE
postgres=# INSERT INTO table1 VALUES (1, 'aaa'),(2, 'bbb');
INSERT 0 2
postgres=# SELECT * FROM table1;
 id | t1
----+-----
  1 | aaa
  2 | bbb
(2 rows)

postgres=# GRANT SELECT (t1) on table1 to auditmrole;
GRANT
```

運用： pgAudit

・マスターロールでの種々の操作と監査ログ

- t1 カラムを含む SELECT と t1 カラムを含まない SELECT を実施する

```
postgres=# SELECT * FROM table1;
 id | t1
----+-----
  1 | aaa
  2 | bbb

postgres=> SELECT id FROM table1;
 id
----
  1
  2

postgres=# SELECT id,t1 FROM table1;
 id | t1
----+-----
  1 | aaa
  2 | bbb
```

PostgreSQLログ:

```
2024-08-04 15:32:37.487 JST [171092] LOG:  AUDIT:
OBJECT,1,1,READ,SELECT,TABLE,public.table1,SELECT *
FROM table1;,<not logged>

2024-08-04 15:32:43.754 JST [171092] LOG:  AUDIT:
OBJECT,2,1,READ,SELECT,TABLE,public.table1,"SELECT
id,t1 FROM table1;",<not logged>
```

運用： pgAudit

・pgAudit のログ出力形式

```
2024-08-04 15:32:43.754 JST [171092] LOG:  AUDIT: OBJECT,2,1,READ,SELECT,TABLE,public.table1,"SELECT id,t1 FROM table1;",<not logged>
```

・「AUDIT: OBJECT」：オブジェクト監査ログであること
・「2,1」：セッション内で一意なSQLのID
・「READ,SELECT」：監査ログを出力する要因となった処理の種別
・「TABLE,public.table1」：監査されたオブジェクトの種別と、オブジェクト名
・「SELECT…」：監査の契機となったSQL文
・「<not logged>」：pgaudit.log_parameter = 'off' （渡されたパラメータを監査ログの最後のエントリに追記するかどうかを指定）

運用：pgAudit

- オブジェクト監査の設定方針

⇒ **何を監査したいのかを明確にしておくこと**

- 管理者はロールの管理やメンテナンス処理の運用目的 :全ての操作を記録するようにする

- マスターテーブルは定期的な更新を行う場合：更新処理（INSERT, UPDATE, DELETE）を記録しておく

- 個人情報が含まれる特定列の参照を追跡する場合：参照処理（SELECT）を記録しておく

まとめ

- 拡張機能とは何か
- PostgreSQLの拡張機能の利点
- 応用例を紹介
  - pg_hint_plan: クエリの実行計画の制御
  - pg_stat_statements:クエリパフォーマンスのモニタリング
  - pg_rman: バックアップ管理
  - pgaudit: 監査ログ機能

参考資料

- pg_hint_plan:
https://github.com/ossc-db/pg_hint_plan

- pg_stat_statments
https://www.sraoss.co.jp/PostgreSQL/Manual/document/16/pgstatstatements.html

- pg_rman
https://github.com/ossc-db/pg_rman

- pgaudit
https://github.com/pgaudit/pgaudit

ご清聴ありがとうございました

■講演資料

■技術情報配信中！

YouTube

https://www.youtube.com/c/sraoss-official

Tech Blog

https://www.sraoss.co.jp/tech-blog

>> 技術者 募集中！

- ホームページをご確認ください
https://www.sraoss.co.jp/personnel/