PGECons
PostgreSQL Enterprise Consortium

設計運用WG 2015年度活動報告

ー企業でPostgreSQLを使いこなすために
～周辺ツールとセキュリティ～ー

PostgreSQL エンタープライズ・コンソーシアム
WG3 (設計運用WG)
(株)日立ソリューションズ　稲垣／(株)アシスト　喜田

アジェンダ

- WG3の紹介
  - WG3の活動テーマ、活動体制
- データベースツールテーマの成果紹介
  - 運用監視
  - バックアップ
  - パーティショニング
  - 実行計画制御
- セキュリティテーマの成果紹介
  - セキュリティに関連する新機能、技術要素の紹介
  - セキュリティ機能使用時の性能検証
- エンディング
  - 2015年活動を振り返って

WG3の紹介

WG3 2015年度活動テーマ

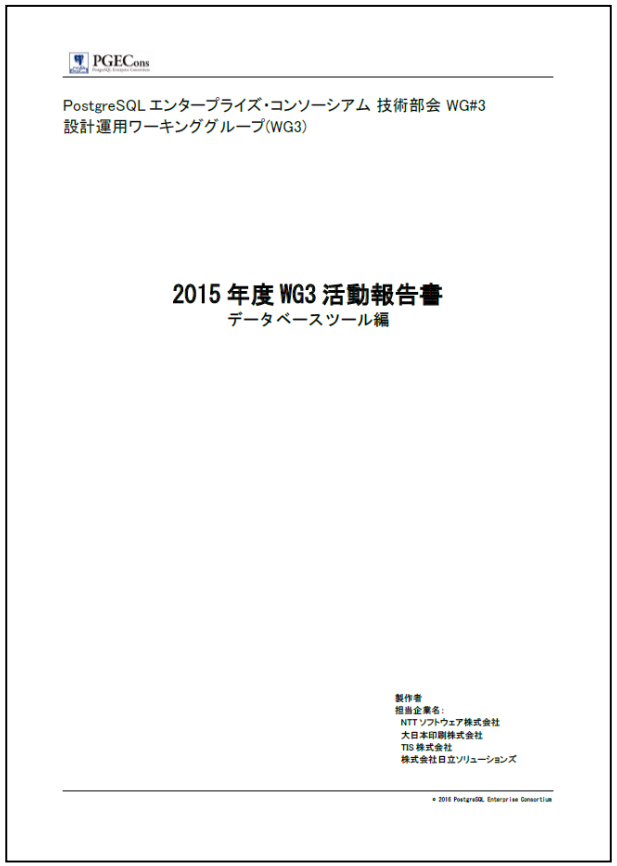
- **データベースツール（今年度からの新設テーマ）**
  - 基盤情報システムに要求される性能・拡張性・運用保守性をPostgreSQLで対応するために、有効な周辺ツールの整理および機能と実用レベルの調査

- **セキュリティ（昨年度からの継続テーマ）**
  - 新たに公開された暗号化機能や監査機能に関する調査および性能検証を行い、実用への適性や留意するポイントを整理
  - 活動のベースとして、他DBMS向けに記述されたDBSC※による文書を参考に、PCI DSS準拠のためのPostgreSQLにおける対処策をまとめた昨年度成果物を用いた

※DBSC: DataBase Security Consortium

WG3 2015年度の活動成果

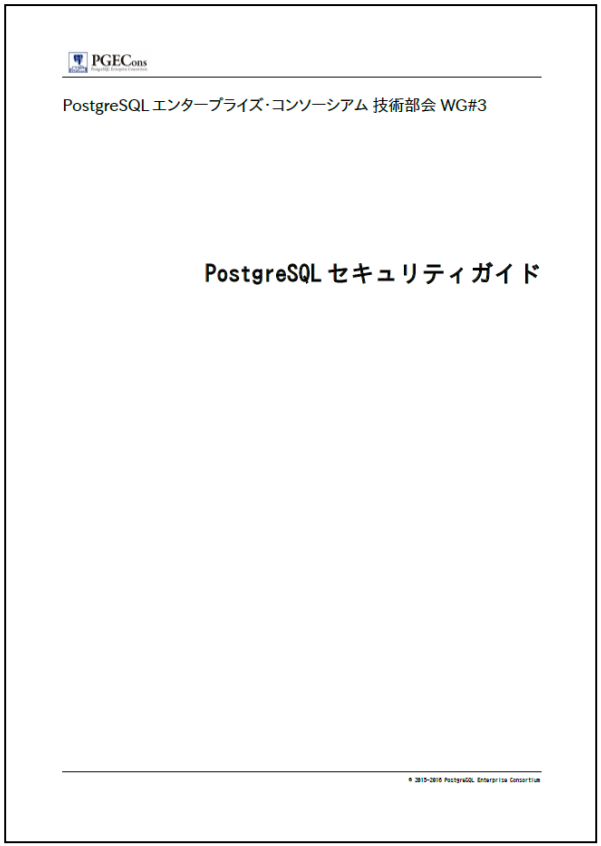
- 「2015年度WG3活動報告書」として、PGEConsのサイトにドキュメントを公開しています。
  - データベースツール編：58ページ、セキュリティ編：139ページ


PGECons

PostgreSQL エンタープライズ・コンソーシアム 技術部会 WG#3
設計運用ワーキンググループ(WG3)

2015 年度 WG3 活動報告書
データベースツール編

製作者
担当企業名：
NTT ソフトウェア株式会社
大日本印刷株式会社
TIS 株式会社
株式会社日立ソリューションズ


PGECons

PostgreSQL エンタープライズ・コンソーシアム 技術部会 WG#3

PostgreSQL セキュリティガイド

WG3 2015年度活動体制

- データベースツール
  - NTTソフトウェア株式会社
  - 大日本印刷株式会社
  - TIS株式会社
  - 株式会社日立ソリューションズ 【主査】
- セキュリティ
  - 株式会社アシスト【副主査】
  - サイオステクノロジー株式会社
  - 日本電気株式会社
  - 日本電信電話株式会社

（企業名50音順・敬略称）

データベースツールテーマ
成果紹介

データベースツール班の活動目的

- 背景
  - 商用データベース製品と比較して運用系の機能が弱い
- 問題点
  - PostgreSQLの標準機能だけでは対応できない場合がある
  - データベースツール（以降、周辺ツール）が見つかっても、情報やノウハウが少なく、どれを使ったらよいかわからない
- 目的
  - 各カテゴリの周辺ツールをピックアップし、機能や課題を調査
  - 調査した周辺ツールの使いどきをまとめる

検証した周辺ツール

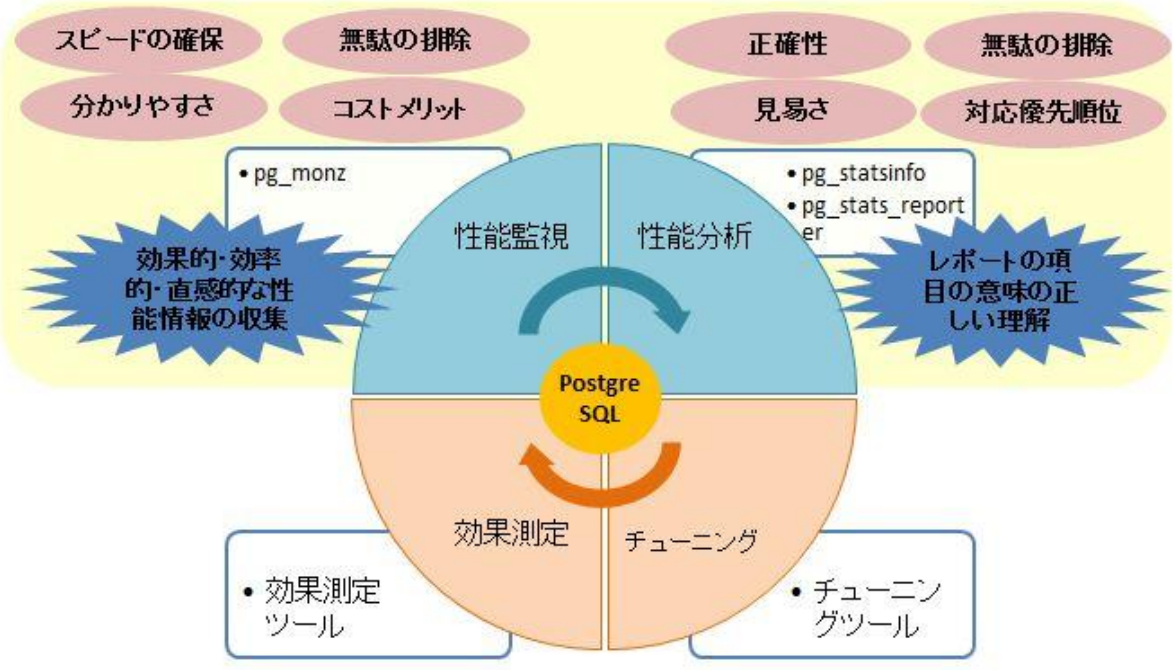
ツールの種類（カテゴリ）	理由	ツール
運用監視（性能監視、分析）	サービスの安定的な運用のため、システムを監視し、統計情報を取得、分析、通知する	pg_statsinfo pg_stats_reporter pg_monz
バックアップ	万一、システム障害が発生しても、企業活動を継続する上で必要なデータを保持する	pg_rman Barman OmniPITR
パーティショニング	テーブル肥大化による性能劣化や運用性低下が起こったとき、テーブル分割で改善する	pg_part pg_partman pg_shard
性能維持（実行計画制御）	PostgreSQLが選択する実行計画が非効率で要件が満たせないとき、意図的に制御する	pg_hint_plan pg_dbms_stats

→上記に挙げたカテゴリのツールについて、調査・検証結果を示す

性能監視・分析ツール

検証概要

- **性能監視が必要な背景**
  - ICTサービスを安定的に運用するためには監視処理が必要となる



- **性能監視の要件**
  - 定期的に統計情報を取得できる
  - 過去の統計情報と比較できる（時系列で確認できる）
  - 性能分析を容易に実施できる

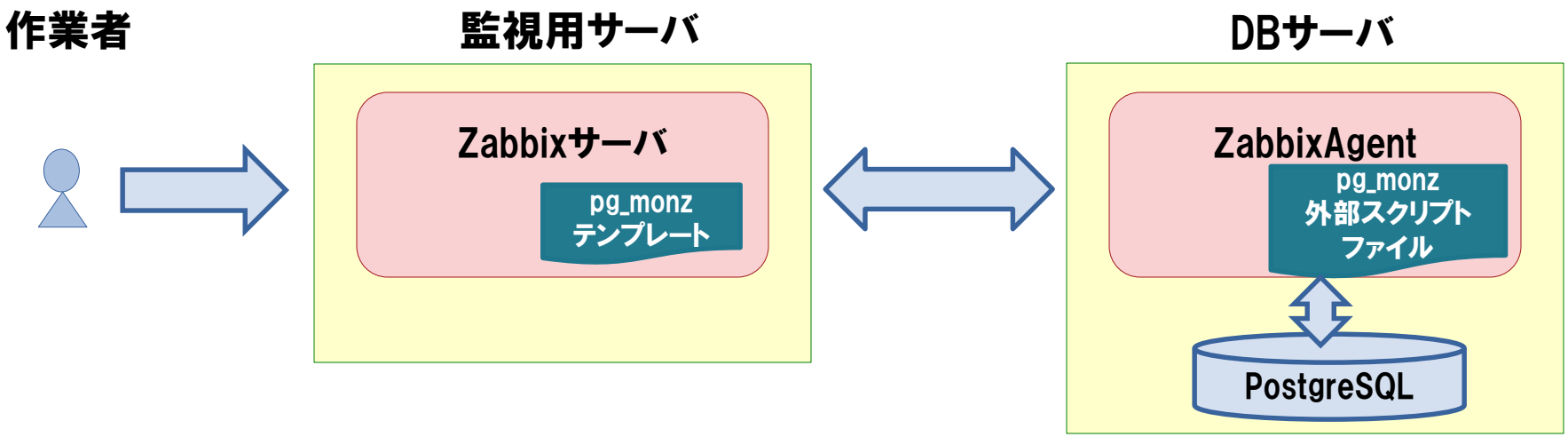
性能監視・分析ツール

- **検証したツール**

ツール名	用途	役割
pg_statsinfo	性能分析	PostgreSQLの統計情報を定期的に取得し、問題特定・性能分析作業を補助する。
pg_stats_reporter	性能分析	pg_statsinfoの情報をレポートする。 pg_statsinfoで閲覧できるレポートよりも可読性を良くする。
pg_monz	性能監視	ZabbixでPostgreSQLの監視を行うためのテンプレートで、死活監視、リソース監視、ストリーミングレプリケーションの冗長構成のステータス、pgpool-IIの状態監視を実施できる。

評価・検証結果

- **検証ツール**
  - pg_monz
- **検証概要**
  - ① 監視手順の検証
    - pg_monzを利用した場合に、どのような内容を監視することができるか手順等を含めて検証
  - ② WebAPIの利用検証
    - WebAPIを利用した監視制御の方法を検証

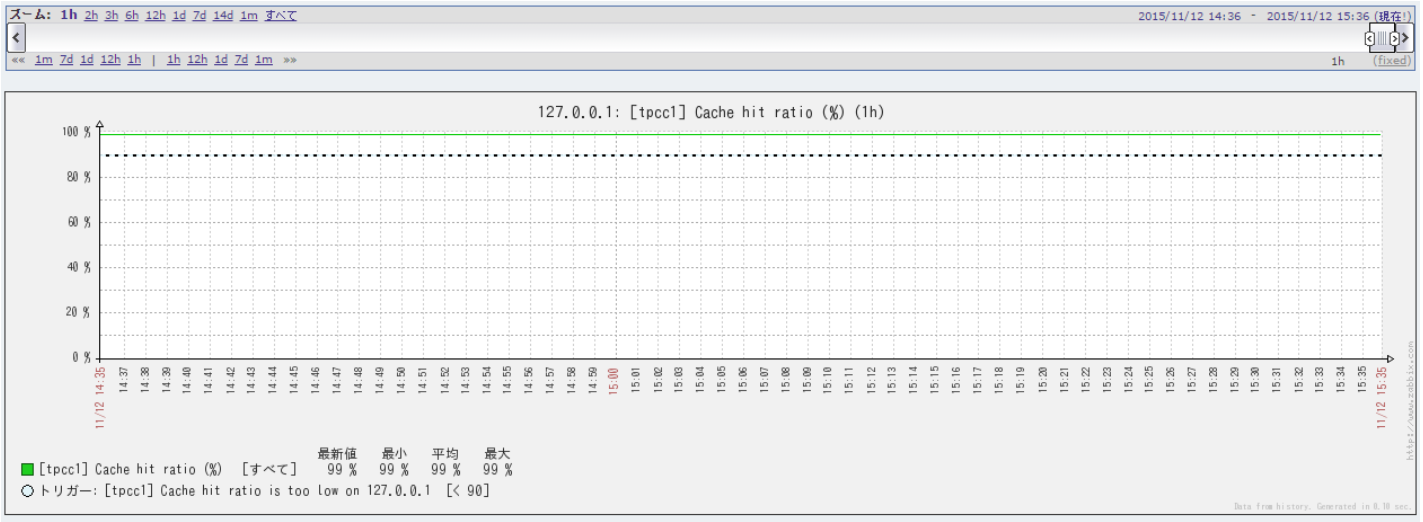


評価・検証結果　①監視方法や手順の検証

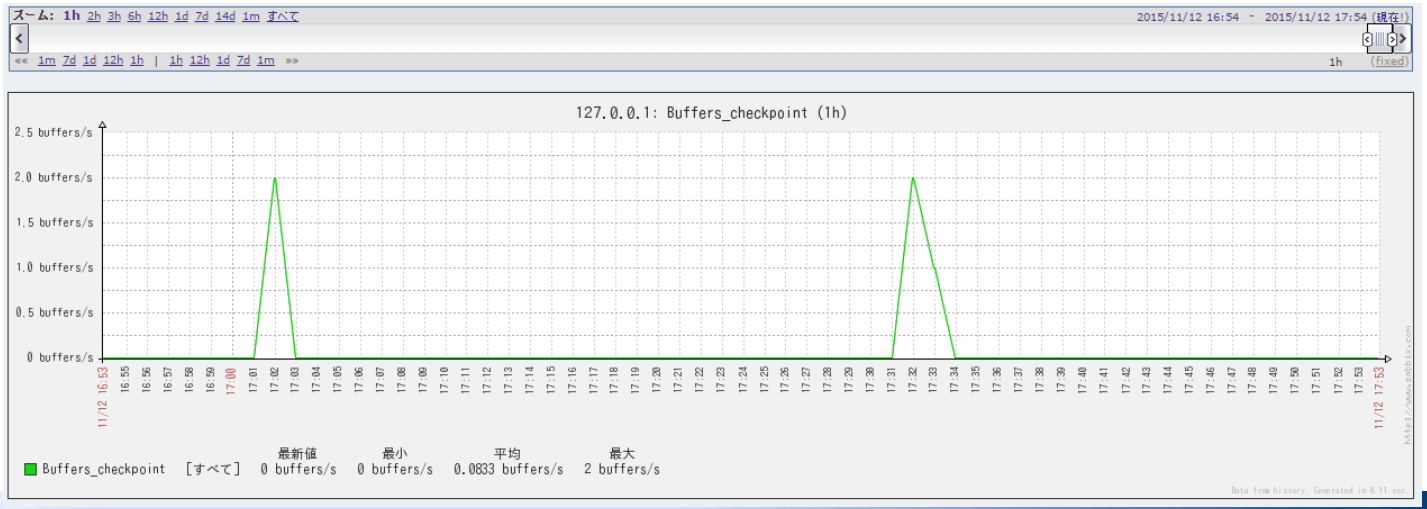
- **検証内容**
  - pg_monzによる監視方法や手順を検証
- **検証結果**
  - メモリ、チェックポイント、autovacuum等の監視をZabbixの中で容易に実施することができる
  - グラフ等監視を実施できるため監視作業の効率化につながると考えられる
- **メリット**
  - Zabbixで監視項目を一元管理することが可能。管理ツールを二重管理する必要がない
  - アプリケーションの追加/拡張を繰り返すようなシステムの場合などで、手軽に監視項目を追加できる

評価・検証結果　①監視方法や手順の検証

- キャッシュヒット率のグラフ



- チェックポイント書込み量のグラフ



評価・検証結果　②WebAPIの利用検証

- **検証内容**
  - pg_monzに対するWebAPIの利用方法を検証
- **検証結果**
  - WebAPIを利用することにより、pg_monzに標準で用意されていない監視結果を取得することができ、監視作業の効率化が可能

例）キャッシュヒット率ワースト3の取得結果イメージ

#	itemid	value	get_time	name
1	26300	53	2015-11-15 11:35:42 +0900	[tpcc1] (public.warehouse) heap cache hit ratio %
2	26311	63	2015-11-15 11:35:52 +0900	[tpcc_large] (public.customer) heap cache hit ratio %
3	26309	73	2015-11-15 11:35:50 +0900	[tpcc_large] (public.warehouse) heap cache hit ratio %

※JSON形式のデータを加工して表にしています

- **メリット**
  - 監視で取得したデータを用いて標準で用意されていない表示形式が可能
  - 監視作業を容易にすることにより運用作業の効率化が可能

性能監視・分析ツール　まとめ

- **検証したツール**

ツール名	用途	役割
pg_statsinfo	性能分析	PostgreSQLの統計情報を定期的に取得し、問題特定・性能分析作業を補助する。
pg_stats_reporter	性能分析	pg_statsinfoの情報をレポートする。 pg_statsinfoで閲覧できるレポートよりも可読性を良くする。
pg_monz	性能監視	ZabbixでPostgreSQLの監視を行うためのテンプレートで、死活監視、リソース監視、ストリーミングレプリケーションの冗長構成のステータス、pgpool-IIの状態監視を実施できる。

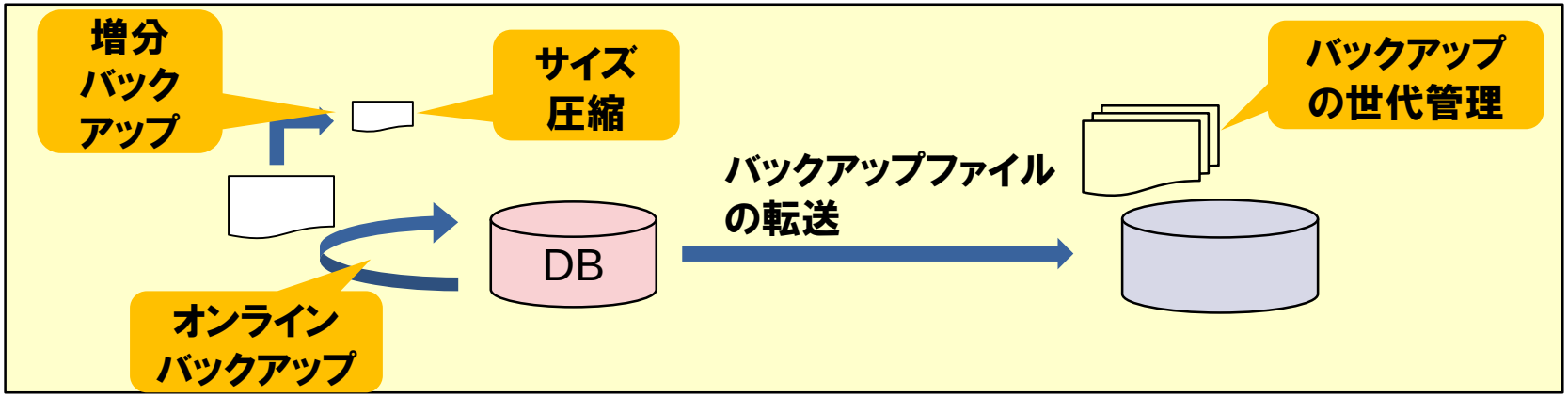
- **性能監視・分析ツールの使いどき**

	pg_statsinfo、 pg_stats_reporter	pg_monz
使いどき	PostgreSQLの性能の詳細を分析したい時	・性能監視内容を細かくコントロールしたいなど、拡張性が求められる時 ・システム監視ツールとしてZabbixを採用する予定であり、監視全般をZabbixで一括提供したい時
対象となるシステムの特徴	複雑なデータ構造等により、性能に問題が生じやすいシステム	OSやサービス、PostgreSQLの死活監視機能を一括で提供したいシステム

バックアップツール

検証概要

- **バックアップに求められる要件**



要件	目的	標準のPostgreSQLで実現可能か
オンラインバックアップ	サーバを停止させない	○
バックアップの自動実行	運用コストの削減	×
バックアップの世代管理	バックアップ管理	×
バックアップファイルの圧縮	リソース節約	○
増分バックアップの取得	リソース節約	×
バックアップファイルを遠隔地で保存	災害対策	○

バックアップツール

- **検証したツール**

ツール	概要
pg_basebackup	PostgreSQLの標準機能
pg_rman	物理バックアップ、リストア実行ツール
Barman	DR管理ツール
OmniPITR	WALの運用を補助するスクリプト（WALアーカイブ、他サーバへの転送など）

評価・検証結果

- **バックアップ機能の評価観点**

評価項目		ツール			
大項目	小項目	pg_base backup	pg_rman	Barman	OmniPITR
バックアップ取得方法	リモートサイトからのバックアップ取得	○	×	○	×
バックアップの管理	バックアップの世代管理	×	○	○	×
	リストア機能の有無	-	○	○	×
	不要なバックアップの削除	×	○	○	○
バックアップの対象範囲	サーバログの取得	×	○	×	×
	増分バックアップの取得	×	○	○	×

評価・検証結果

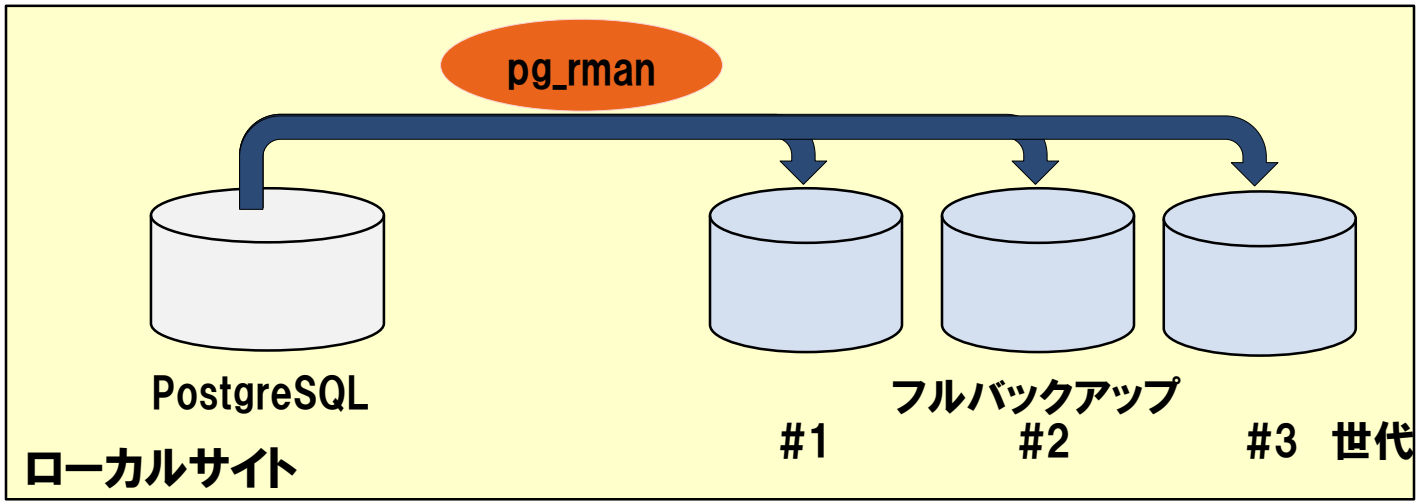
- **バックアップの世代管理(pg_rman)**

pg_rmanの管理コンソール

```
[postgres@postgres01 ~]$ pg_rman show detail
===============================================================================
StartTime Mode Duration Data ArcLog SrvLog Total Compressed CurTLI ParentTLI Status
===============================================================================
2016-02-25 11:14:18 FULL 0m 21MB 134MB ---- 4470kB true 1 0 OK
2016-02-24 17:40:42 INCR 0m 0B ---- ---- 0B true 1 0 ERROR
    0m 21MB 67MB ---- 4362kB true 1 0 OK
ULL 0m 21MB 33MB ---- 4307kB true 1 0 OK
```

バックアップファイルのサイズが確認可能

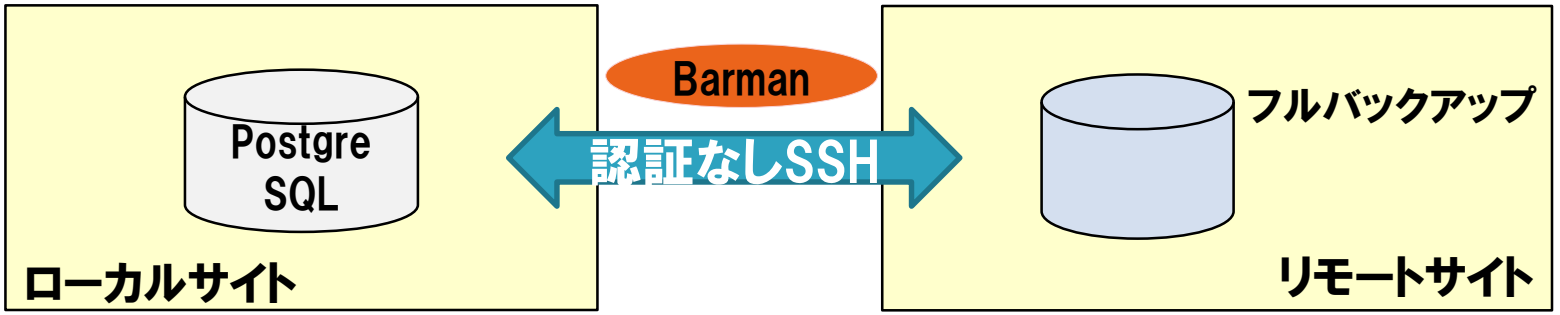
バックアップファイルの状態が確認可能



評価・検証結果

- セキュリティ要件の評価観点

評価項目	ツール			
	pg_basebackup	pg_rman	Barman	OmniPITR
特定ポート解放の必要性	不要	-	必要	不要
rootユーザである必要性	不要	不要	不要	不要
認証なしのSSH接続である必要性	不要	不要	必要	不要
DBのsuperuser権限の必要性	不要	-	必要	必要
pg_hba.confのtrust認証の必要性	不要	不要	不要	不要



バックアップツール　まとめ

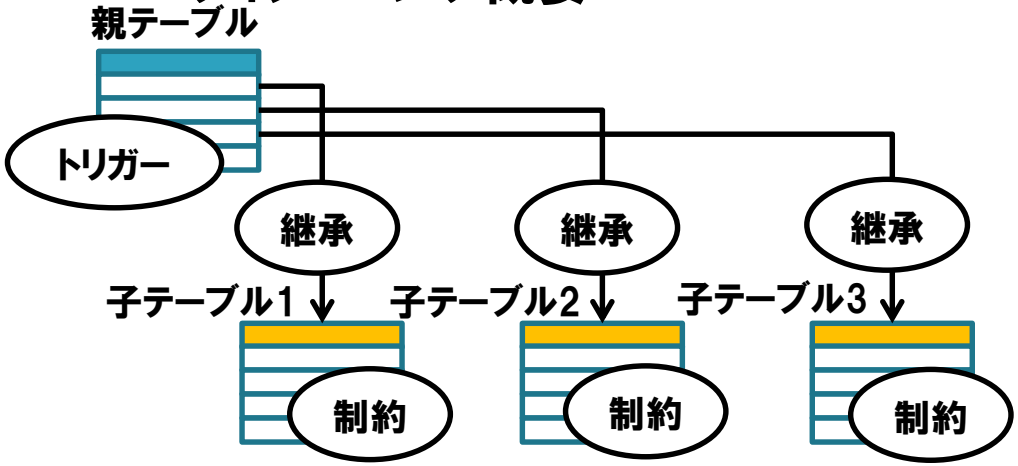
- **バックアップツールの使いどき**

ツール	概要	使いどき
pg_basebackup	・PostgreSQLの標準機能	・セキュリティ要件を緩めたくない ・フルバックアップのみで十分
pg_rman	・物理バックアップ、リストア実行ツール	・フルバックアップと増分バックアップを組み合わせた運用 ・複数世代のバックアップファイルを管理
Barman	・DR管理ツール	・バックアップ対象サーバが複数台ある ・バックアップファイルを別環境に保存 ・複数世代のバックアップファイルを管理
OmniPITR	・WALの運用を補助するスクリプト（WALアーカイブ、他サーバへの転送など）	・archive_commandやrestore_commandパラメータをOSコマンドより多機能に実装 – 圧縮が可能 – ファイルを複数サーバに転送可能

パーティショニングツール

検証概要

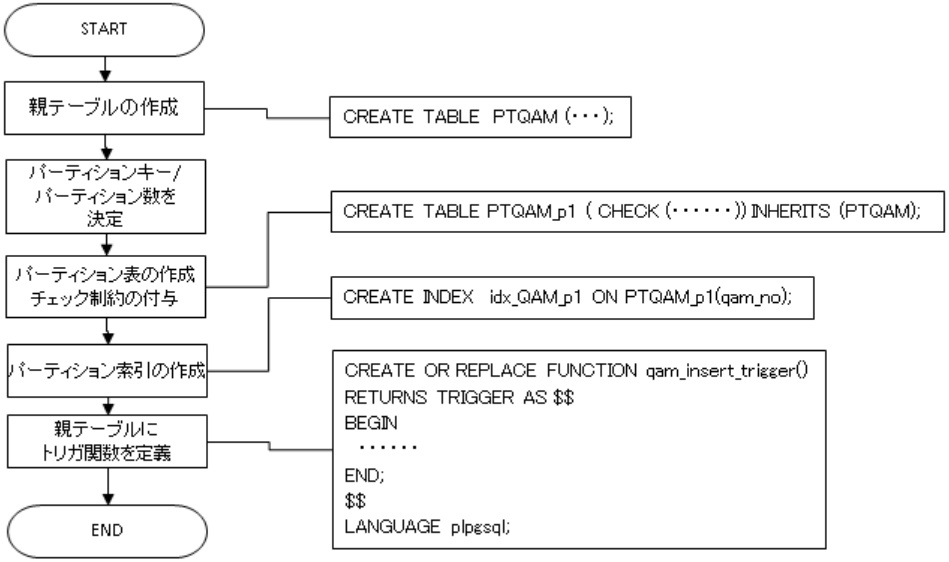
- パーティショニング概要



PostgreSQLにパーティショニング専用機能はなく、
- テーブル継承
- テーブル制約
- トリガー

を組み合わせて実現している

- パーティショニング構築フロー



構築時、構成変更時に煩雑な作業が要求される

要件：
管理コストの低減

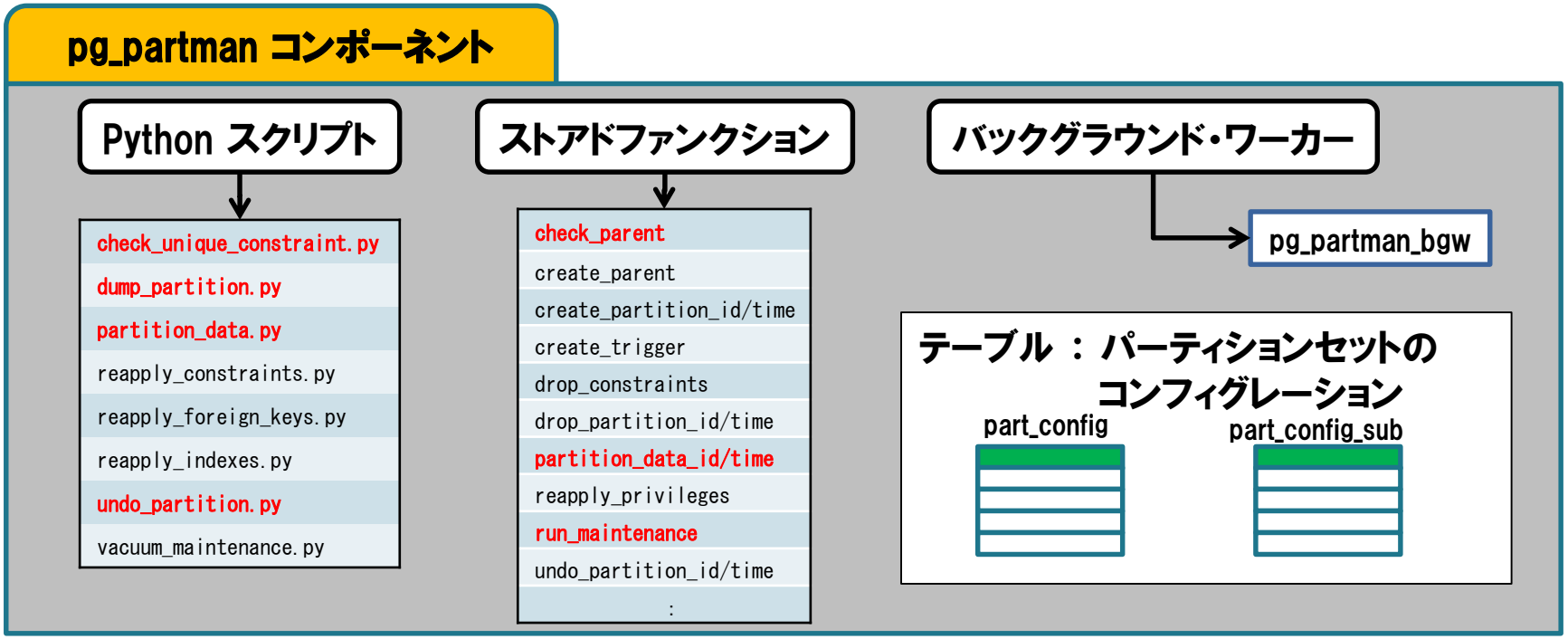
パーティショニングツール

- **検証したツール**

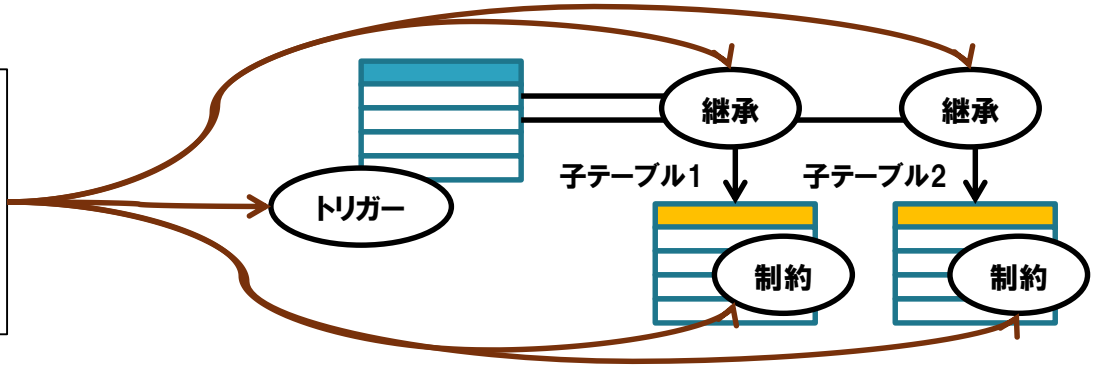
ツール名	概要
pg_part	パーティションの作成(＋データの移行)/解除(＋データの移行)/追加/切り離しを簡単に行う関数群を提供する
pg_partman	時系列と連番によるパーティショニングテーブルの作成、管理を行う拡張モジュール
pg_shard	テーブルのシャーディング(パーティショニング)とレプリケーションを同時に実現するツール

評価・検証結果

■ pg_partman を使用した 自動パーティション管理



partman.run_maintenance
の定期実行による
自動パーティション
メンテナンス処理



評価・検証結果

■ pg_partman を使用した 自動パーティション管理

パーティションの変換

pg_partman で非パーティション表をパーティション表のパーティションに変換する場合の手順

1. create parent () 関数を実行し、指定した表をパーティション表として管理する
2. partition_data_id () 関数またはpartition_data_time () 関数を実行し、親テーブルのデータを
   パーティション表へ移動する(該当するパーティション表がない場合は自動的にパーティションが作成)
3. check_parent () 関数を実行し、親テーブルにデータが残っていないことを確認する

パーティションの変換

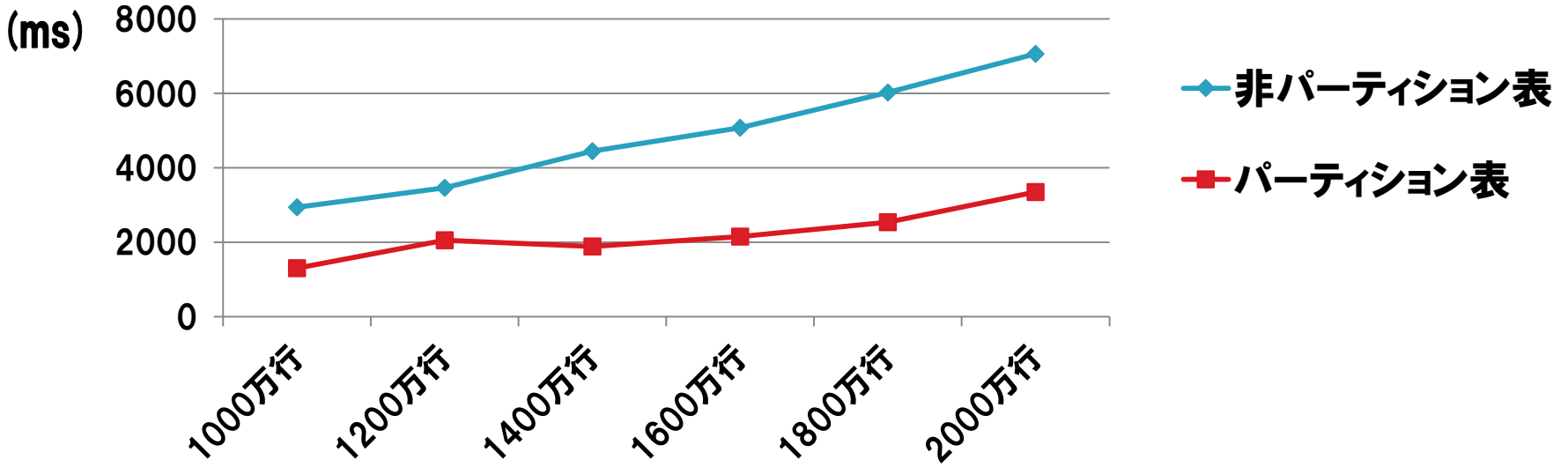
```
SELECT partman.create_parent('public.pttesttbl', 'startdate', 'time', 'monthly');
```

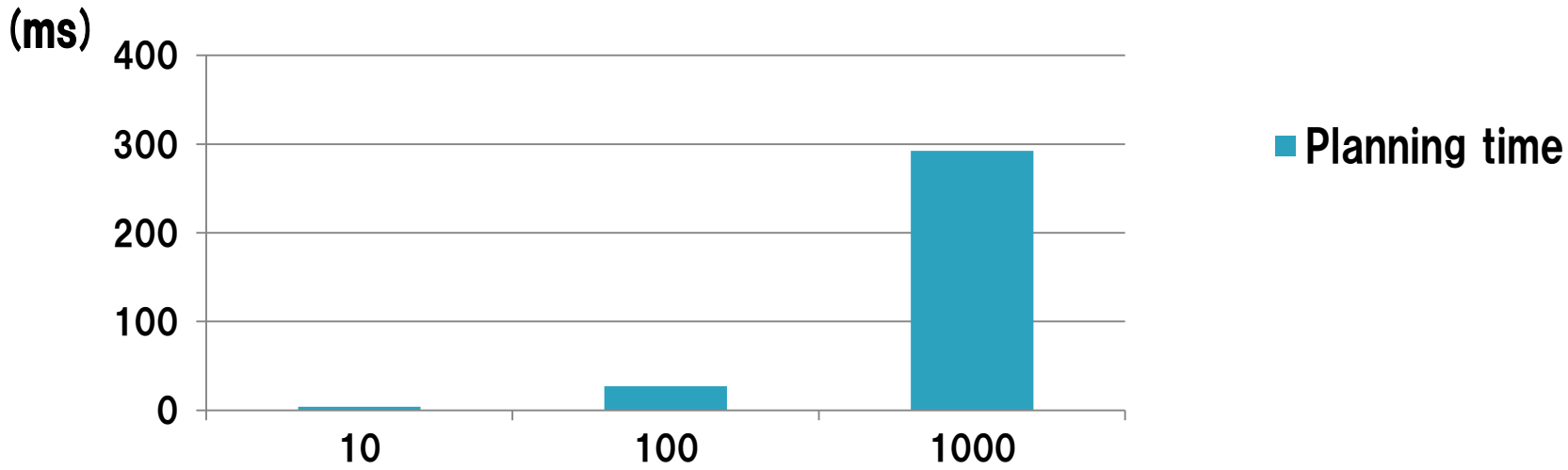
確認結果

```
 Name                 |  parent_t |                        con_detail                        |          tr
----------------------+-----------+----------------------------------------------------------+----------------------
 pttesttbl            |           | PRIMARY KEY (id, number, startdate                       | ptcall10tbl_part_trig
 pttesttbl_p2015_05 | pttesttbl | PRIMARY KEY (id, number, startdate                       | false
 pttesttbl_p2015_05 | pttesttbl | CHECK (startdate >= '2015-05-01 00:00:00'::              | false
 pttesttbl_p2015_06 | pttesttbl | PRIMARY KEY (id, number, startdate                       | false
 pttesttbl_p2015_06 | pttesttbl | CHECK (startdate >= '2015-06-01 00:00:00'::              | false
                                                        :
                                                        :
```

評価・検証結果

パーティション表のメリット・デメリット





パーティショニングツール

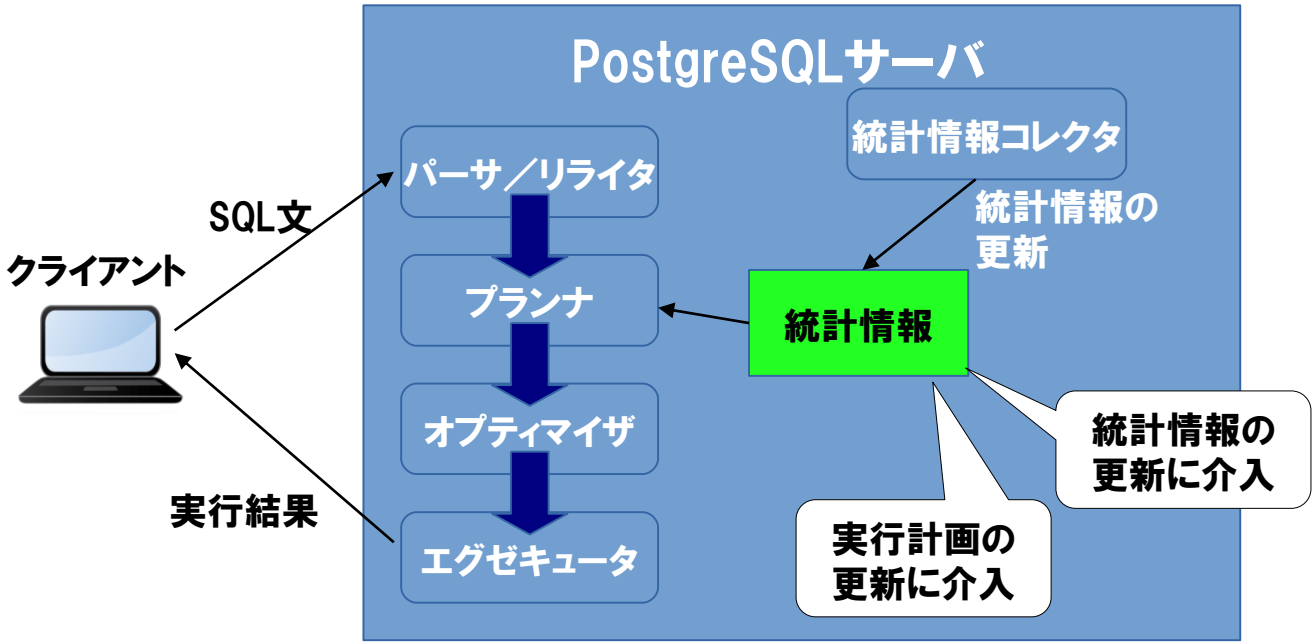
- **パーティションツールの概要と使いどき**

ツール名	概要	使いどき
pg_part	パーティションの作成(＋データの移行)/解除(＋データの移行)/追加/切り離しを簡単に行う関数群を提供する	パーティション管理を手動で行う場合の管理コストを低減したい場合
pg_partman	時系列と連番によるパーティショニングテーブルの作成、管理を行う拡張モジュール	パーティション管理を自動で行う形でシステム運用を行う場合
pg_shard	テーブルのシャーディング(パーティショニング)とレプリケーションを同時に実現するツール	シャーディングによる水平分散で大規模DBを容易に実現したい場合

実行計画制御ツール

検証概要

- PostgreSQLではクエリ内容と、アクセスするテーブルの統計情報を基に、自動的にクエリの実行計画を作成し、その実行計画にしたがって処理を行う。
- PostgreSQL自身にも、実行計画を制御する機能はあるが、細かい制御はできない
- 常に最新の統計情報から実行計画を制御するため、データの更新により、実行計画が変動する可能性がある。
- このような場合、実行計画制御ツールを導入して、システムにとって都合のよい実行計画を生成するように制御する。



実行計画制御ツール

- **検証したツール**

ツール名	概要
pg_hint_plan	クエリにヒントを与えて実行計画を制御する
pg_dbms_stats	統計情報を固定化する

評価・検証結果

- 検証ツール
  - pg_hint_plan
- 検証内容
  - pg_hint_planによって、クエリ本体を修正せず、最適な実行時間が得られる実行計画に変更したときの計画の差分を検証
- 検証環境
  - CentOS 7
  - PostgreSQL 9.4
  - pg_hint_plan94-1.1.3

評価・検証結果

- 2つのテーブルを結合させるクエリを発行

```
SELECT test_a.id, test_a.data FROM test_a INNER JOIN test_b ON test_a.data
= test_b.data  WHERE test_b.id < 500000 AND test_a.id < 500000;
```

- 結合方式をpg_hint_planで実行計画を制御しなかった場合

```
test=# EXPLAIN ANALYZE
SELECT test_a.id, test_a.data FROM test_a INNER JOIN test_b ON test_a.data = test_b.data  WHERE test_b.id <
500000 AND test_a.id < 500000;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Merge Join  (cost=141076.40..152367.51 rows=503085 width=8) (actual time=741.709..1195.927 rows=249706 loops=1)
   Merge Cond: (test_b.data = test_a.data)
   ->  Sort  (cost=70524.39..71772.71 rows=499326 width=4) (actual time=397.159..533.640 rows=499997 loops=1)
         Sort Key: test_b.data
         Sort Method: external merge  Disk: 6824kB
         ->  Index Scan using test_b_pkey on test_b  (cost=0.42..16435.63 rows=499326 width=4) (actual
time=0.058..100.504 rows=499999 loops=1)
               Index Cond: (id < 500000)
   ->  Materialize  (cost=70551.75..73049.17 rows=499484 width=8) (actual time=344.540..540.191 rows=553139 loops=1)
         ->  Sort  (cost=70551.75..71800.46 rows=499484 width=8) (actual time=344.535..487.402 rows=499999 loops=1)
               Sort Key: test_a.data
               Sort Method: external merge  Disk: 8776kB
               ->  Index Scan using test_a_pkey on test_a  (cost=0.42..16443.40 rows=499484 width=8) (actual
time=0.059..70.917 r
ows=499999 loops=1)
                     Index Cond: (id < 500000)
 Planning time: 0.162 ms
 Execution time: 1212.808 ms
```

評価・検証結果

- pg_hint_planを使ってHash Joinに変更した場合

```
test=# /*+
 HashJoin (test_a test_b)
*/
EXPLAIN ANALYZE SELECT test_a.id, test_a.data  FROM test_a INNER JOIN test_b ON test_a.data = test_b.data
WHERE test_b.id < 500000 AND test_a.id < 500000;
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=24628.63..170584.90 rows=503085 width=8) (actual time=209.579..571.218 rows=249706 loops=1)
   Hash Cond: (test_a.data = test_b.data)
   ->  Index Scan using test_a_pkey on test_a  (cost=0.42..16443.40 rows=499484 width=8) (actual time=0.016..73.706
rows=499999 loops=1)
         Index Cond: (id < 500000)
   ->  Hash  (cost=16435.63..16435.63 rows=499326 width=4) (actual time=209.398..209.398 rows=499999 loops=1)
         Buckets: 16384  Batches: 8  Memory Usage: 2207kB
         ->  Index Scan using test_b_pkey on test_b  (cost=0.42..16435.63 rows=499326 width=4) (actual
time=0.011..107.076 rows=499999 loops=1)
               Index Cond: (id < 500000)
 Planning time: 0.199 ms
 Execution time: 580.218 ms
```

Hash Join

- pg_hint_plan使用有無によるcostと実行時間の比較

pg_hint_plan	結合方式	推定コスト	実行時間(ms)
使用しない	Merge Join	**152367.51**	1212.808
使用する	Hash Join	170584.90	**580.218**

実行計画制御ツール

- 実行計画制御ツールの使いどき

ツール名	概要	使いどき
pg_hint_plan	クエリにヒントを与えて実行計画を制御する	PostgreSQLが生成する実行計画が不適切で性能上の問題が発生するとき
pg_dbms_stats	統計情報を固定化する	最速でなくても良いので、安定した性能を常に担保したいとき

データベースツールテーマ　まとめ

- 周辺ツールを活用することで、PostgreSQL単体で満たせない非機能要求を満たすことができる
- 周辺ツールの評価・検証を実施し、情報やノウハウ、使いどきを成果物ドキュメントにまとめたので参考にしてほしい
- 今回の発表内容には含めなかったが、開発元や活動状況(リリース状況)、ライセンス、サポートなどの基本情報はチェックしておく必要がある

セキュリティテーマ
成果紹介

セキュリティ班の活動目的

PostgreSQLのセキュリティ機能を調査中の利用者にとり、有用な資料を作成・公開

2015年度は、新機能をはじめ記載対象を拡大し昨年度成果物を改訂

- セキュリティ事故、マイナンバーといったキーワードが依然として話題
  →エンタープライズ・データベースにおいては、RDBMS機能で実現できるセキュリティ機能が検討される
- PostgreSQL開発においては、セキュリティ機能でも改善・拡張が活発

2014年～2015年に公開されたPostgreSQLのセキュリティ関連機能
PostgreSQL本体に行単位アクセス制御機能（RLS）を追加
PostgreSQL本体の監査機能拡張（pgaudit）が検討された （残念ながら後に却下、現在も別プロジェクトでの開発は続いている）
透過的データ暗号化モジュール（ tdeforpg [*]）が公開された
SQLインジェクション対策モジュール（sql_firewall）が公開された

- 性能とのトレードオフとされる機能では実用に足る情報が得づらい状況

[*] 正式名称：Transparent Data Encryption for PostgreSQL

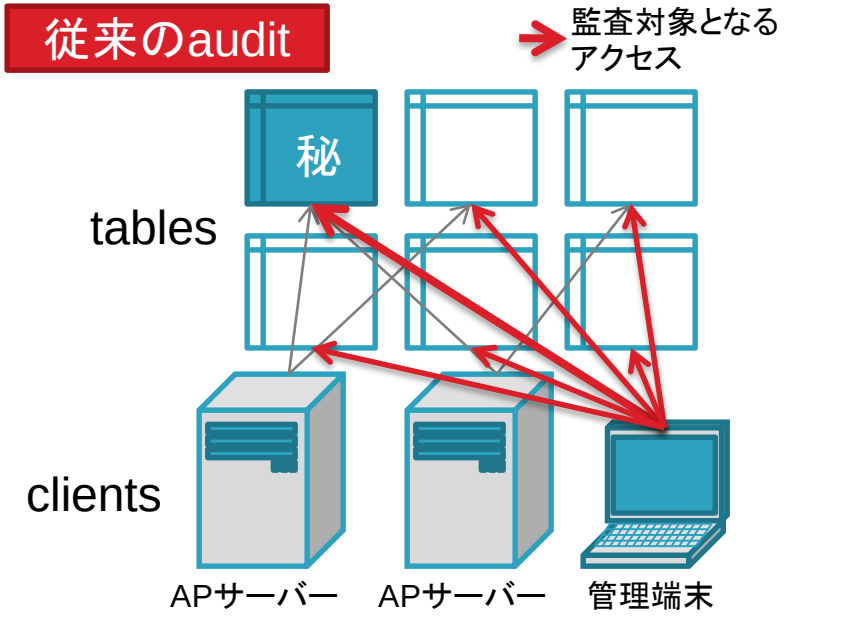
成果物に追加した技術要素

- 新機能、新規ツール、既存機能に対する追加調査
  - 監査
    - pgaudit
    - 現行の監査機能を総じた考察
  - 認証
    - 既存機能による認証の強化
  - アクセス制御
    - 行単位のアクセス制御
    - SQLインジェクション対策
  - 暗号化
    - 格納データの透過的暗号化

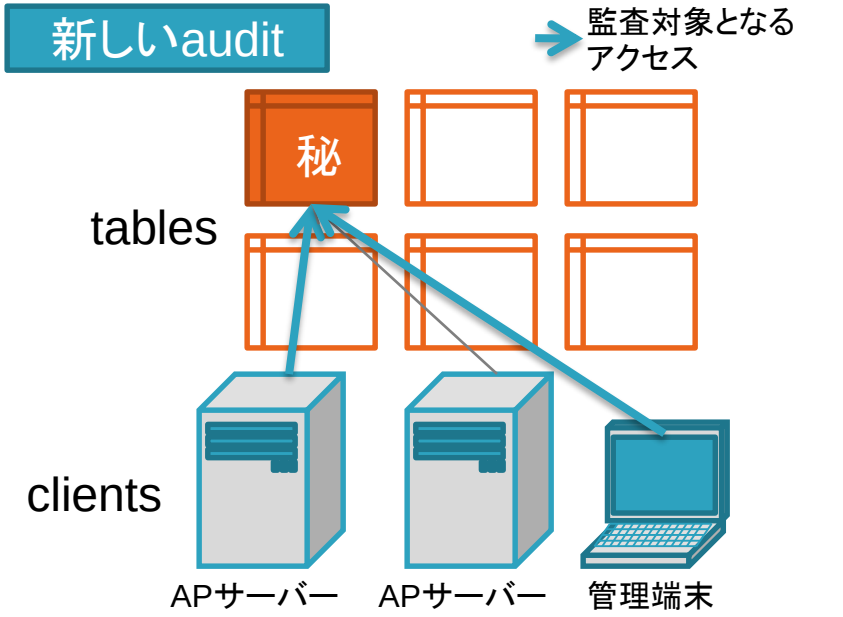
pgauditで実用的な監査を実現

対象テーブルを指定して監査ログを記録

- ログが肥大化しやすい監査の問題を解消
- 個人情報を含む等の特定のテーブルに対するアクセスに絞ったロギング
- PostgreSQL 9.5では追加モジュールとしての採用が検討された[*]



・接続元(実際にはロール)による制御　：○
・アクセスしたテーブルによる制御　：×



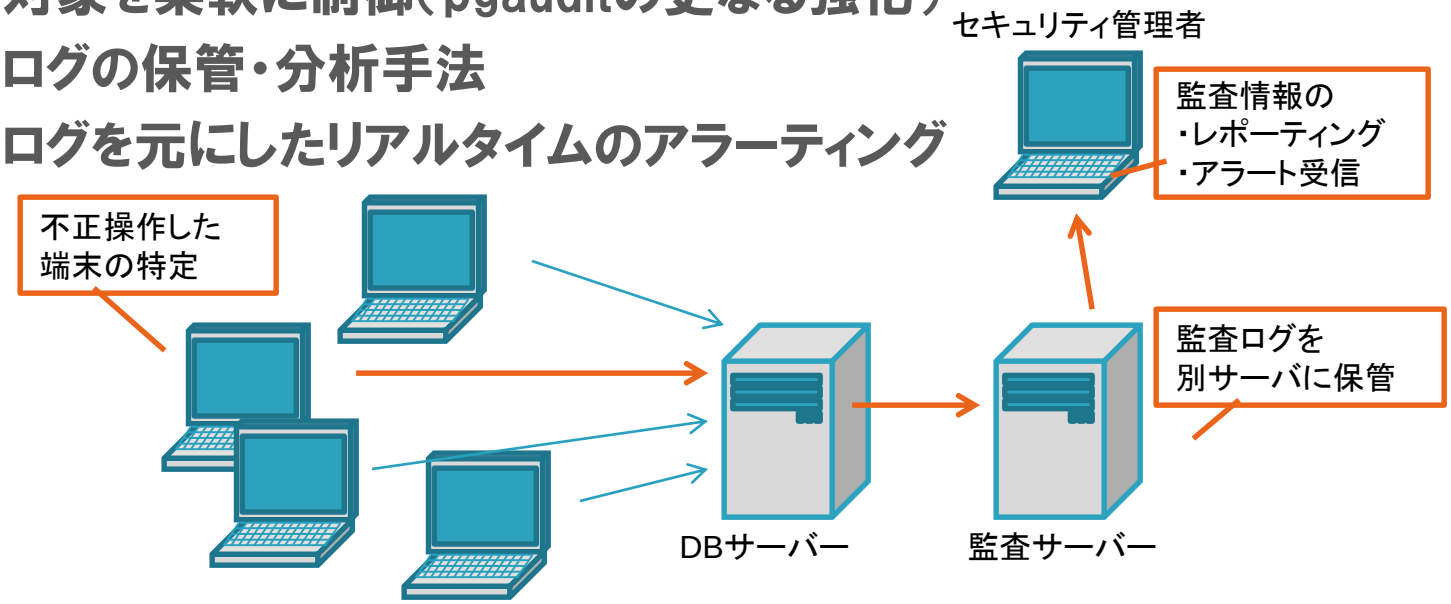
・接続元(実際にはロール)による制御　：○
・アクセスしたテーブルによる制御　：○

[*] 現在は別プロジェクト(https://github.com/pgaudit/pgaudit)として開発継続中

（参考）監査に求められる更なる強化

PostgreSQLベースの商用製品や、他RDBMS対応セキュリティ製品機能を調査し、監査機能に対する期待を整理

- PostgreSQLでロギングできない項目の監査
  - APユーザ名、アクセス元ポート番号、SELECT結果の件数など
- 監査出力データの柔軟な取り扱い
  - 記録対象を柔軟に制御（pgauditの更なる強化）
  - 監査ログの保管・分析手法
  - 監査ログを元にしたリアルタイムのアラーティング



既存機能による認証の強化

passwordcheckモジュールによるパスワードポリシーの強制

- 標準機能ではパスワードに対する制限は設定できない
- 本機能を使用して以下の制限を強制
  - 8文字以上であること
  - アルファベットとそれ以外の文字をそれぞれ1文字以上使用すること
  - ユーザー名を含まないこと

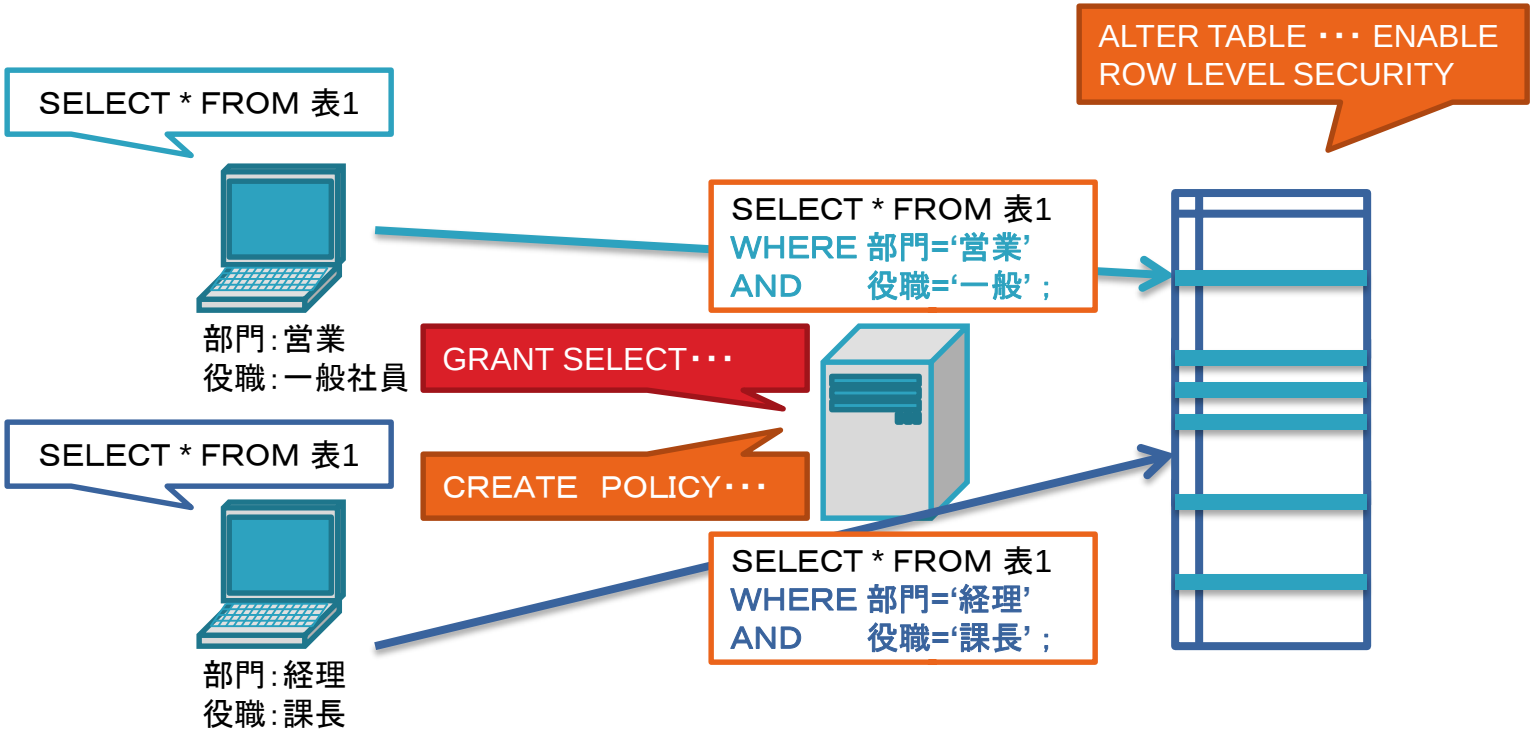
auth_delayモジュールによるパスワードの総当たり攻撃対策

- デフォルトではパスワード認証時のエラーは即座に返される
- 本機能でパスワード認証時のエラー報告前に指定時間だけ待機

行単位アクセス制御(Row Level Security)

表単位、列単位でのアクセス制御に加え、行単位での制御を実現

- 従来の表単位、列単位アクセス制御はGRANT文で実現
- 行単位アクセス制御では事前定義したポリシー(WHERE条件)に合致する行のみへのアクセスを可能とする



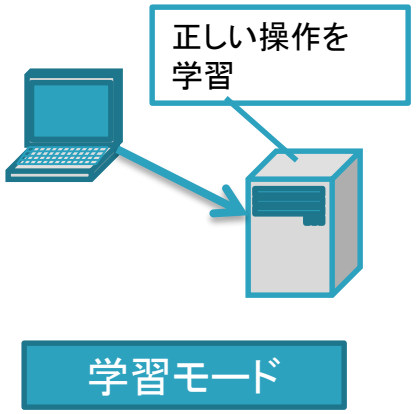
SQLレベルのファイアウォール制御（sql_firewall）

学習済みSQLのみを実行可能とし、不正なSQLの実行を防止

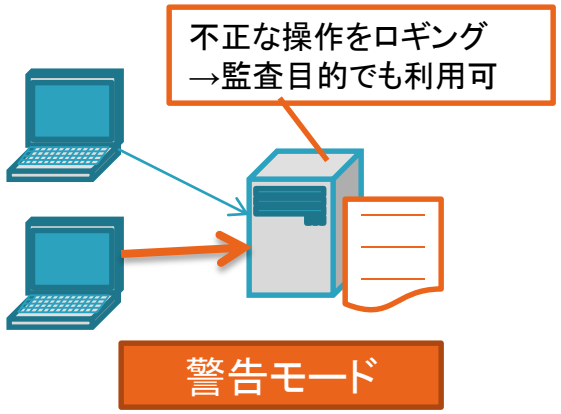
- 作成はアップタイムテクノロジーズ合同会社 永安氏
- ソースコードはGitHubで公開中（ https://github.com/uptimejp/sql_firewall ）
- 学習～防御までモードを切り替えて運用

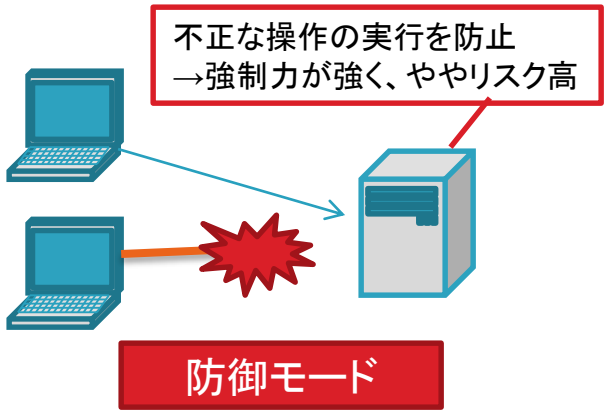
選択可能なモード	動作
学習モード（learning）	実行されたSQLを正常なものとして登録
警告モード（permissive）	学習済みSQL以外を検知し警告を発生
防御モード（enforcing）	学習済みSQL以外は実行時エラーとする

試験期間



本番稼働中





格納データの透過的暗号化

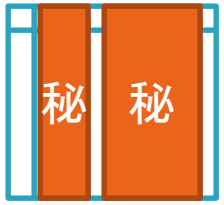
（Transparent Data Encryption for PostgreSQL）

アプリケーションプログラムの修正を抑えたデータ暗号化を実現

- 作成は日本電気株式会社
- ソースコードはGitHubで公開中（ https://github.com/nec-postgres/tdeforpg ）
- 文字列型およびバイナリ型を格納する暗号化データ型を提供
- 対象列の暗号化/復号を行うTDEセッションの開始を宣言して使用

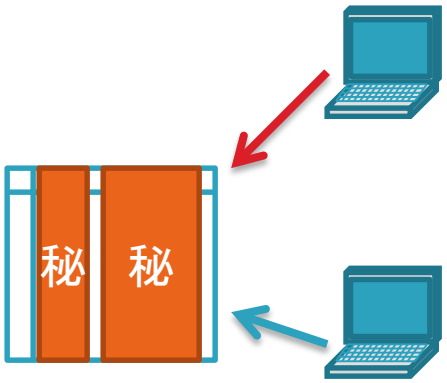
テーブル定義

tdeforpgにより追加された
暗号化データ型を使用



列	型
cust_no	integer
cname	encrypt_text
address	encrypt_text

使用例




```
pgtde_begin_session(key)
    ---TDEセッションの開始
INSERT INTO ・・・;
SELECT * FROM ・・・;
    ---SQLコードの書き換え不要
```

```
---非TDEセッション
SELECT * FROM ・・・;
ERROR, TDE-E0017 ・・・
could not decrypt data ・・・
```

セキュリティ機能使用時の性能検証

■ 検証の目的

- □ セキュリティ機能は性能とのトレードオフであることが多い
- □ 想定しうる性能影響を考察するため、モデルケースでの性能検証を実施

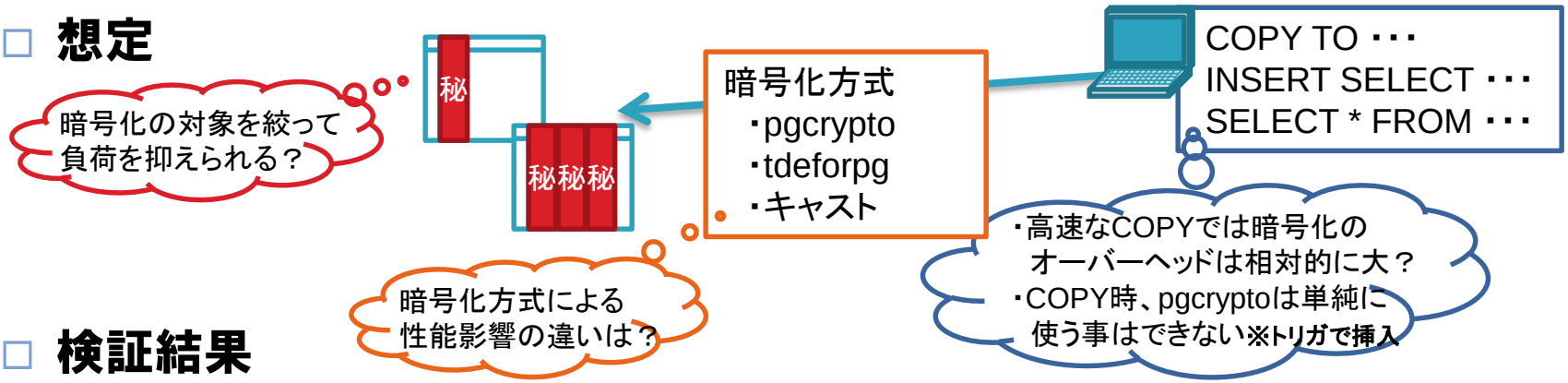
■ 検証内容

- □ 格納データの透過的暗号化
  暗号化方式別に、格納および取り出し時の性能を計測
- □ 行単位アクセス制御
  暗黙的にWHERE句が追加されることによる実行計画の変動を確認
- □ 監査データ出力
  監査ログ出力のon/off、処理傾向の差による性能影響を計測

透過的暗号化の性能影響

暗号化方式別に、暗号化対象列数による性能への影響度合いを確認

□ 想定



□ 検証結果

概ね想定通りの結果、COPY時の性能・管理性でtdeforpgが有利

暗号化対象列	暗号化方式	格納（暗号化）（COPY） ※トリガ経由	格納（暗号化）（INSERT）	取り出し（復号）
15列/15列中	pgcrypto	511※	223	217
	tdeforpg	153	206	143
	キャスト	218※	36	36
1列/15列中	pgcrypto	106※	46	28
	tdeforpg	41	36	26
	キャスト	85※	23	15
平文		29	20	12

単位（秒）

透過的暗号化の考察

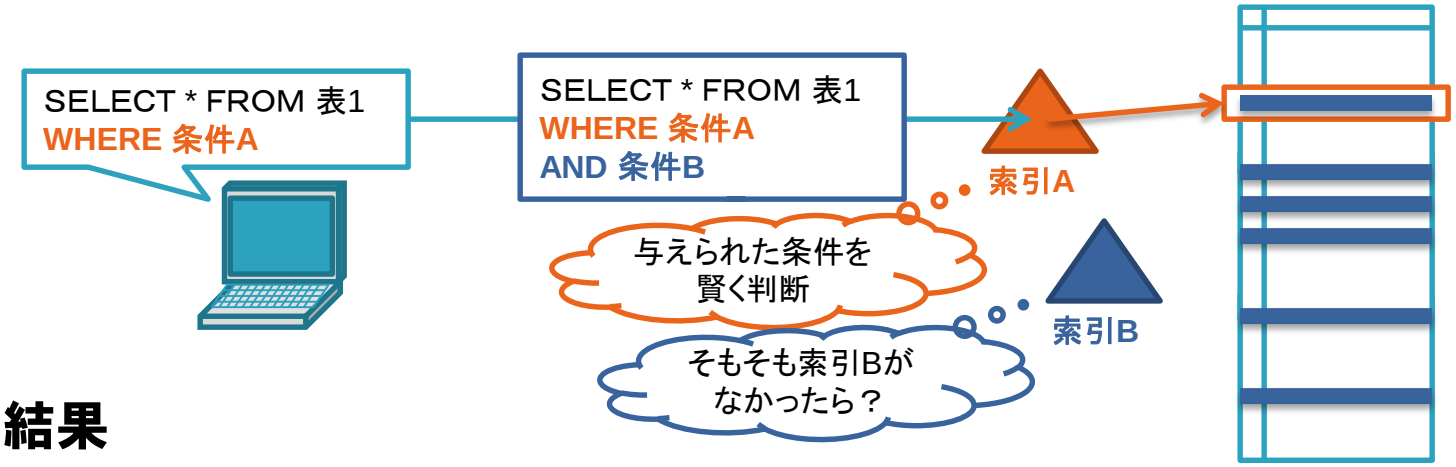
- 性能影響は大きいものの、要件次第で検討に値する
  - 暗号化によるアプリケーション修正を激減
  - これまで暗号化の唯一の手段であったpgcryptoとの比較
    - すべての検証においてtdeforpgの結果が良好
    - COPYによるデータ投入では大きな効果を発揮

- 参考：実用にあたっての考慮
  - 検証モデルケースは1SQLで大量のデータ（680万件）を扱った
    - 各行に対して暗号化を行うため、オーバーヘッドが最大化するケース
    - 処理で使われるコアは1つのみ、フル稼働し常時100%
  - OLTP系処理では暗号化による性能影響が小さいことを期待
    - 1行あたりの暗号化オーバーヘッドは小さい
    - 暗号化も多数のコアで行うため、CPUネックが解消される可能性
    - クロック数の高いCPUを選択することを検討

行単位アクセス制御(RLS)の性能影響

暗黙的にWHERE句を付与するRLSでは、実行計画の変動影響を確認

□ 想定



□ 検証結果

想定に反し、RLSポリシー条件Bを優先、明示的条件Aはその後評価

	条件A (索引あり)	条件B	索引B	Scan 方式	結果
1	無	無	--	Seq	両条件が無いため索引は使用しない
2	無	有	有	Index	RLSポリシーに適合する索引が使用された
3	無	有	無	Seq	RLSポリシーに適合する索引が無い
4	有	有	無	Seq	明示的条件の索引があるが索引を使用しない
5	有	無	--	Index	明示的条件のみ指定し、索引が使用された

行単位アクセス制御を利用する上での注意点

- RLS制御への期待は開発負担の軽減
  - 別途定められたポリシーに従った条件が自動的に付与される
  - 開発者は、欲しいデータへのクエリさえ書けば良い
    - アクセス制御の観点では期待通り使えることが確認できた
    - 性能を維持するために、RLSポリシーを含む実行計画の確認が必要

- 実行計画を見るときの注意点

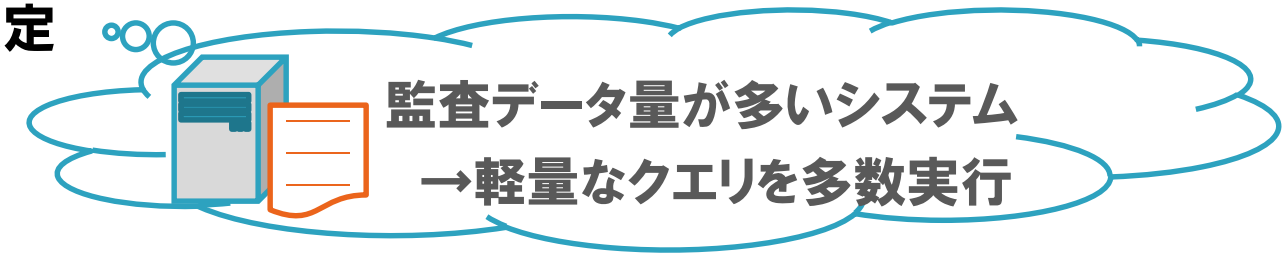
	条件A (索引あり)	条件B	索引B	Scan 方式	結果
4	有	有	無	Seq	明示的条件の索引があるが索引を使用しない

```
=> EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM zip WHERE addr_code LIKE '13%';
------------------------------------------------------------------------------
Subquery Scan on zip  (actual time=17.067..13725.720 rows=1675520 loops=1)
    Filter: ( zip.addr_code  ~~  '13% % '::text)
    Buffers: shared hit=1122500
        -> Seq Scan on zip zip_1(actual time=17.059..13328.265 rows=1675520 loops=1)
            Filter: ( ken_name = ' 東京都 '::text)
            Rows Removed by Filter: 52824640
```

監査データ出力による性能影響

pgbenchで軽量なSELECTを実行、監査ログ出力の有無で性能を比較

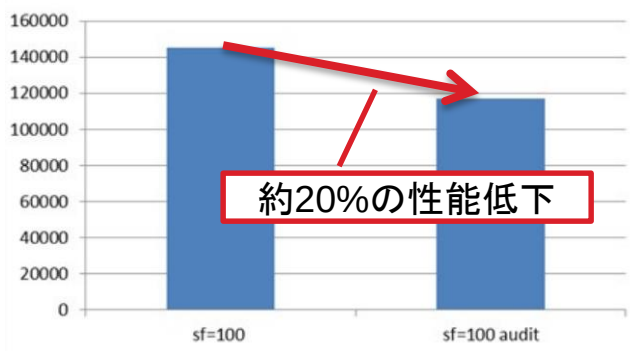
- 想定



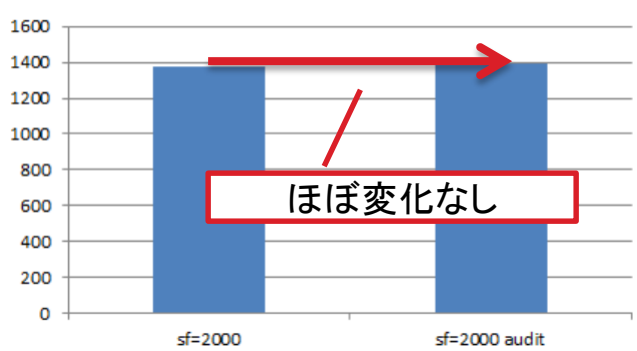
- ベンチマークツールpgbench　select only　48 clients
  - オンメモリ　他にボトルネックが無く、監査ログ出力の**影響が最大化**
  - I/O発生　監査ログ出力無しでもI/Oネックとなり、監査の**影響小**

検証結果

オンメモリ

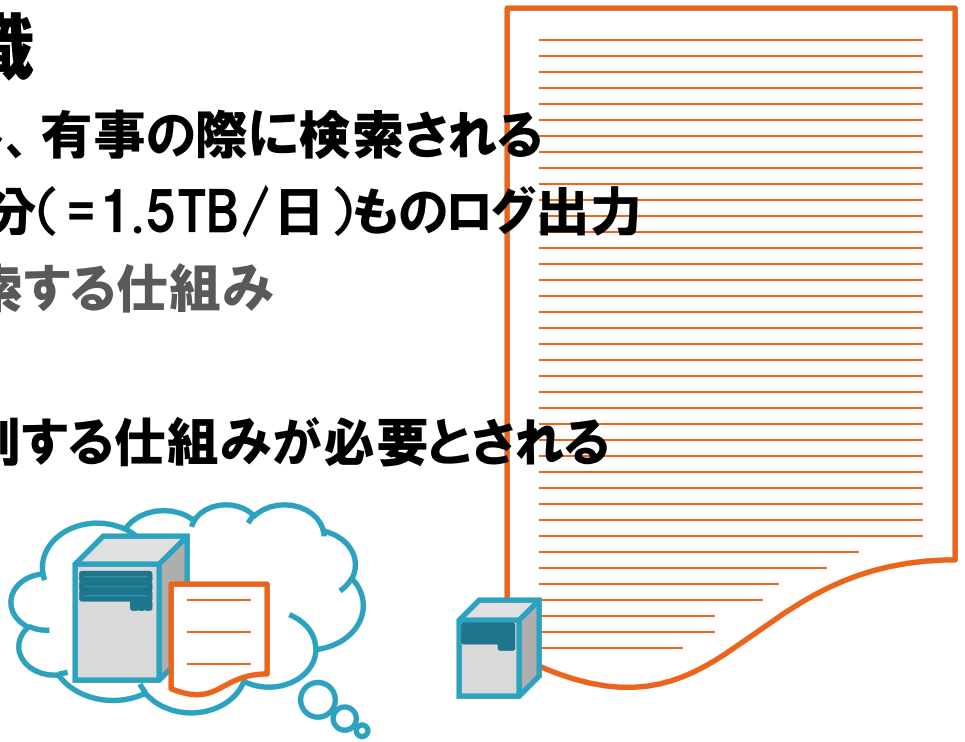


I/O発生



監査データ出力に対する考察

- 性能影響は限定的、要件次第で採用の余地は十分
  - 想定しうる監査ログ出力量が最大化するケースで検証
    - 20%程度の性能影響で済むことが確認できた
    - ボトルネック(I/O負荷が高い)がある状態では監査の影響は表面化しない
- 運用面での課題を認識
  - 監査データは数年間保管し、有事の際に検索される
  - 本検証では、最大で1GB/分(=1.5TB/日)ものログ出力
    - 大量のログを保管、検索する仕組み
    - ログ領域の容量監視
  - pgaudit等のログ出力を抑制する仕組みが必要とされる



セキュリティテーマのまとめ

- PostgreSQLセキュリティの「今」を整理
  - 標準機能または周辺ツールやOS機能との組合せで、セキュリティ関連の課題に概ね対応できる

  - 本WG成果物「PostgreSQLセキュリティガイド」では、PostgreSQLセキュリティ機能の実装、課題、実稼働まで考慮して記載

    - PCI DSS要件への対応可否
    - PostgreSQLでの実装方法
    - 更に進んだ対策が求められる場合の課題
    - モデルケースでの性能検証結果

クロージング

2015年度の活動を振り返って

- 参加企業の皆様からいただいたコメント
  - 同業、異業種の方々と交流をすることで、普段の仕事とは別の気づきが得られました
  - PGEConsを通じて、解決しづらい課題に対して効率的かつ効果的にアプローチする事ができました
  - 公開情報が少なく、とっつきづらいツールを整理した成果がPostgreSQLの普及につながることを期待しています
  - セキュリティは2年目でネタが懸念されましたが、新機能や新製品のおかげで何とか形になりました
  - 性能検証では、各社様にアドバイスいただき勉強になりました

2016年度のWG3活動

- 課題検討WG(名称が改称)
  - より実務に近い形として、データベース管理者やアプリケーション開発者が抱える、現場の課題や困り事に対するテーマを設定し検討

一緒にPGECons WG活動しませんか？

PGECons

PostgreSQL Enterprise Consortium