PGECons
PostgreSQL Enterprise Consortium

商用運用ができる実践的PostgreSQL技術者の育成

2021年11月12日
PostgreSQL Conference Japan 2021 (A3)
PostgreSQLエンタープライズ・コンソーシアム
富士通Japan株式会社　多田　明弘

自己紹介

- 多田　明弘（ただ あきひろ）
- a_tada@fujitsu.com
- 富士通Japan　民需ソリューション開発本部所属
- データベース歴はOracle7から始めて25年以上

  開発者、アーキティクト、開発PM、運用など一通り経験。現在は共通技術部門として本部内の標準化等施策を主務
- PostgreSQLは5年程度
- PostgreSQL エンタープライズコンソーシアム（PGECons）の活動には2017年度より参画

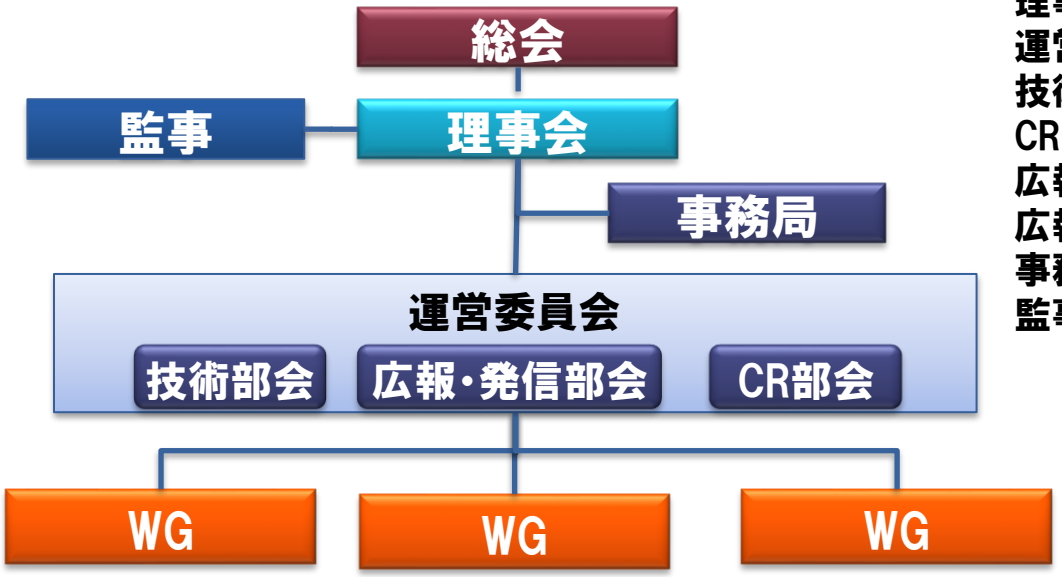
PGECons について

- PGEConsの発足と目的
  - 2012年4月11日発足
  - ミッションクリティカル性の高いエンタープライズ領域へのPostgreSQLの普及を推進するため、各種ツールやPostgreSQL本体に関する利用技術情報の収集と提供および、その整備などの活動を企業ベースで展開する

活動項目	概要
情報発信	会員の導入実績を基に、PostgreSQLおよび周辺ツールに関する情報を集約し、情報発信サイトやセミナ等を通じて提供する
共同検証	エンタープライズ領域への適用に向けて必要となる情報を、実証を通じて充実を図る
開発コミュニティへのフィードバック	よりミッションクリティカル性の高い領域への適用に向けた技術的な課題を集約し、開発コミュニティに要望を発信する
開発プロジェクト支援	会員間での機能拡張に関する連携開発や、必要な周辺ツールの開発プロジェクト支援を進める

2021年度 体制・会員構成

- 会員は法人およびそれに準ずる団体で構成



2021年度体制

理事長	:日本電信電話株式会社
運営委員長	:日本電気株式会社
技術部会長	:富士通株式会社
CR部会長	:SRA OSS, Inc. 日本支社
広報・発信部会長	:株式会社 日立製作所
広報・発信副部会長	:株式会社アシスト
事務局長	:SRA OSS, Inc. 日本支社
監事	:オープンソース活用研究所

種別		概要	総会議決権
正会員	理事	理事会に参加、理事長および運営委員長は理事のうちから就任する	あり
	運営委員	運営委員会に参加、部会長およびWG長は運営委員から就任する	
		ワーキンググループ(WG)に参加し、活動に貢献	
一般会員		メーリングリストやWebなどから、活動情報を取得することが可能	なし

ワーキンググループ活動について

- 現在3つのワーキンググループにて活動中

新技術検証WG（WG1）

- 新バージョンの性能や新技術の検証を通じて有用性を明確化
- スケールアップ検証、新機能における性能特性調査等

移行WG（WG2）

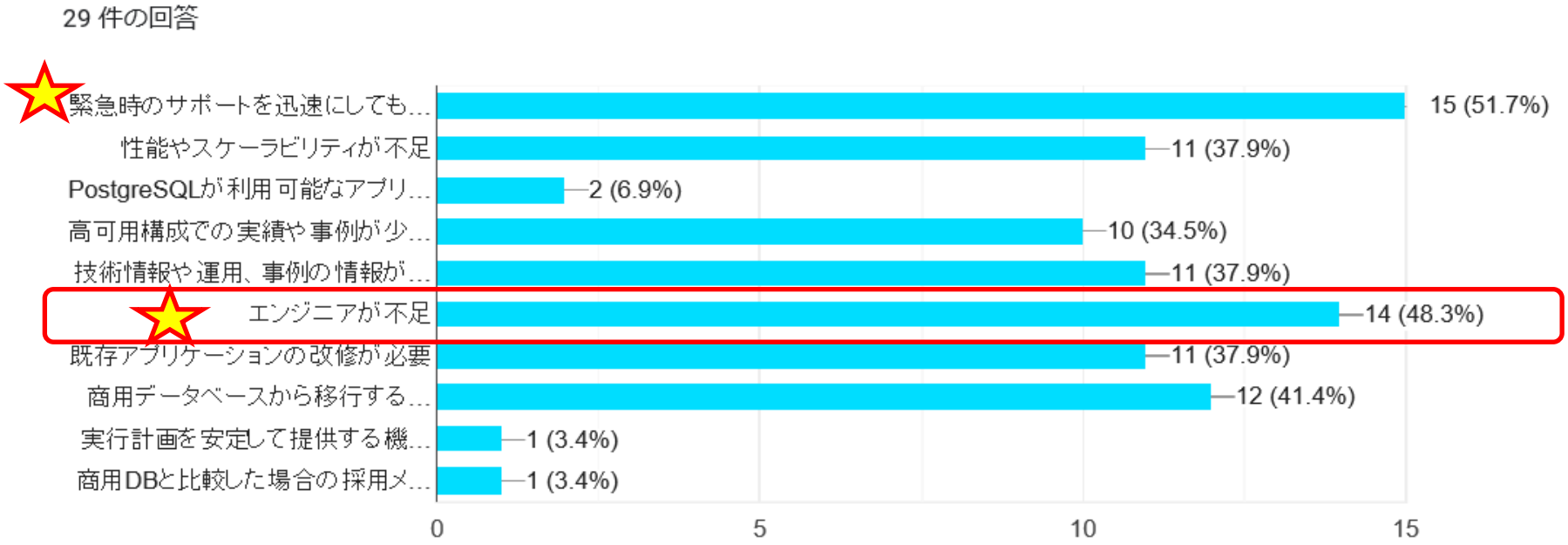
- 異種DBMSからの移行をテーマに活動
- 商用DBMSからの移行プロセスに伴う技術調査や検証を実施

課題検討WG（WG3）

- 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
- 可用性・運用性・保守性・セキュリティ・接続性が主な課題領域

PostgreSQL自習書のねらい

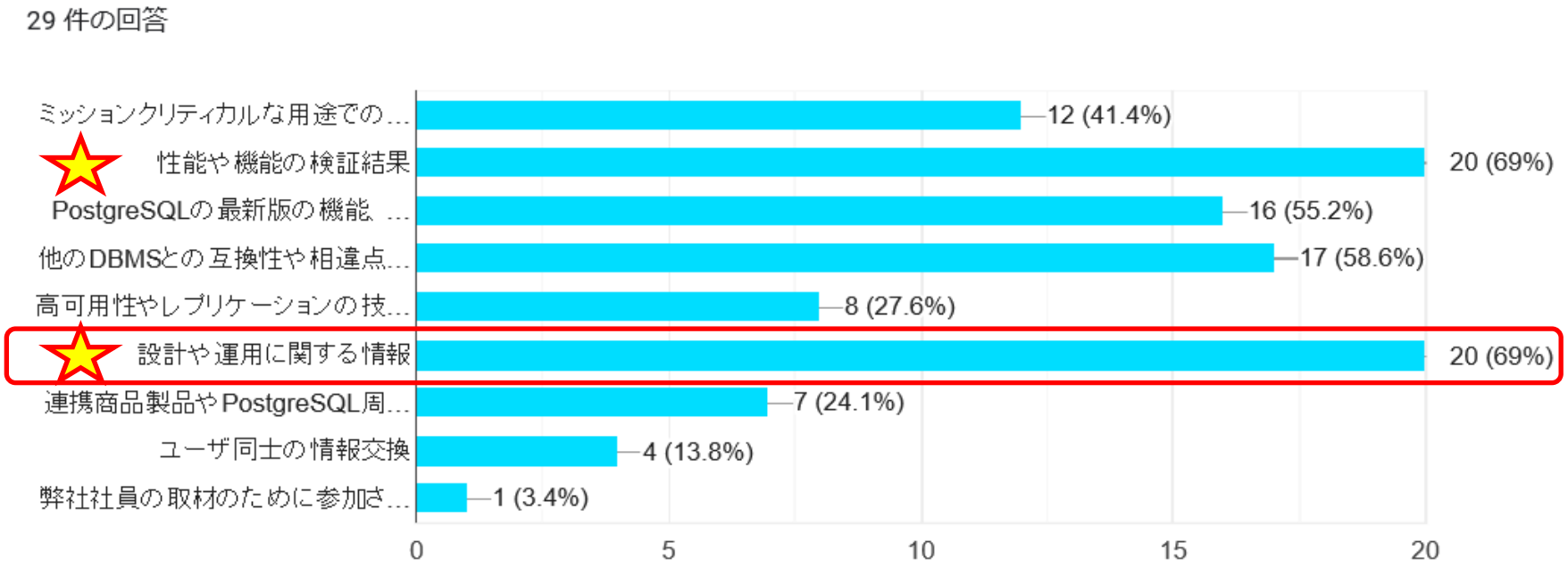
Q7:PostgreSQLをよりミッションクリティカル性の高いエンタープライズ領域で採用するための課題は何だとお考えですか？



PGECons 2019年度 成果発表会アンケートより

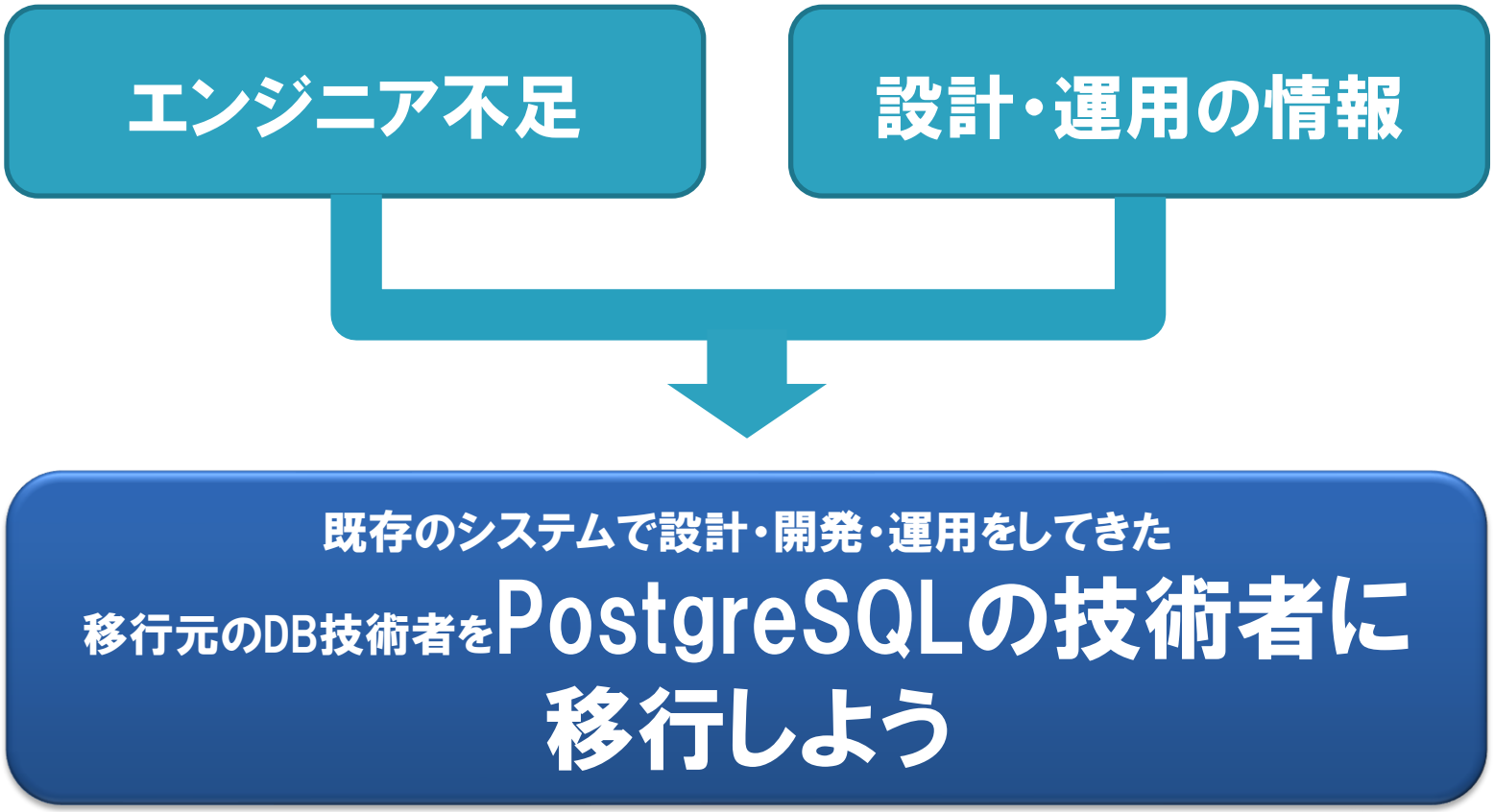
PostgreSQL自習書のねらい

Q21：PGEConsに何を期待していますか？またどんな情報を発信してほしいですか？（複数回答可）



PGECons 2019年度 成果発表会アンケートより

PostgreSQL自習書のねらい



PostgreSQL自習書のねらい

「PostgreSQL自習書」の内容とダウンロード先

- 「PostgreSQL自習書」の内容
  - DBAの立場として把握しておきたい物理的、論理的な構造と全体俯瞰
  - 商用運用上の基本的なポイント
  - RDBMSやSQLの基本は割愛されています
    → データベースを初めて学ぶ方には向きません

- ダウンロード先
  - https://pgecons-sec-tech.github.io/tech-report/pdf/wg2_PostgresSelfStudyBook.pdf

商用での安全な運用の勘所

- お話の前提
  - Oracleの経験はあるけどPostgreSQLは初めての人
- 商用運用でデータベースに望むこと
  - 「PostgreSQL自習書」の運用への応用として以下2点の勘所を説明します

SQLの応答時間が一定であること

**SQLが遅くなったとき
原因が特定できること**

商用での安全な運用の勘所

- SQLの応答時間が一定であること

絶対避けたいこと

運用中に遅くなる

一部運用の変化に伴う場合は許容します

運用の変化	SQLの応答劣化	特徴
データ量の増加	許容できる＝ 遅くなることを理解できる	予測できる＝対策できる
同時接続数の増加		
上記以外	許容できない	????

商用での安全な運用の勘所

- SQLの応答時間が一定であること

運用で起こる現象	原因	対策
だんだん遅くなる	表のブロック密度劣化による物理的なアクセス量の増加	• 適切なFILLFACTORでブロック密度を一定に保つ
	索引の物理的な状態が悪化	• 適切なFILLFACTORでHOTを有効にする • 索引の再構築
急に遅くなる	実行計画が変化した	• 物理的な状態を一定に保つ • 複雑なSQLを避ける • ヒント句 • 統計情報を固定
開発中の性能テストはクリアしたが運用では遅い	テスト時、データの格納状態が物理的に最良の状態だっただけ	• 運用状態を見越したテスト（一定期間運用テストした後に性能テストを実施する）

商用での安全な運用の勘所

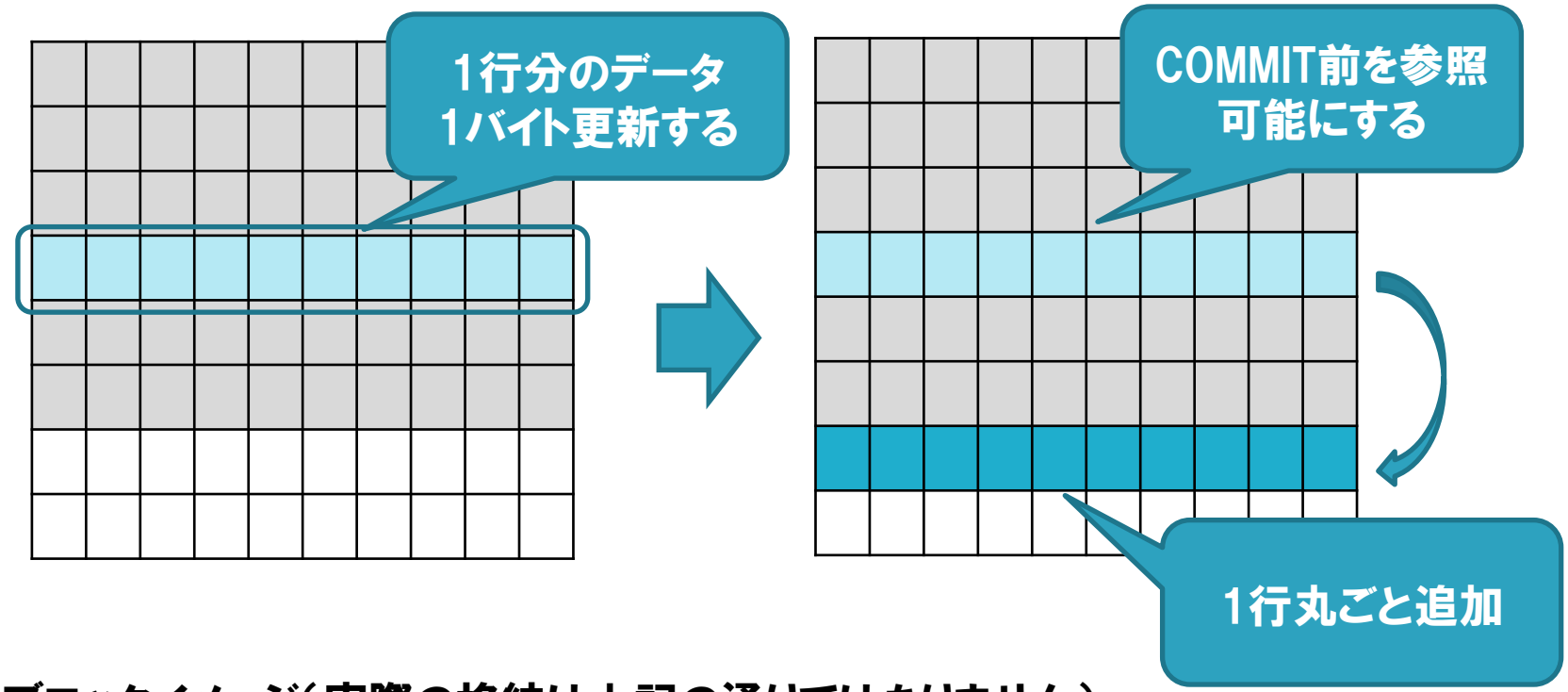
- SQLの応答時間が一定であること

運用で起こる現象	原因	対策
だんだん遅くなる	表のブロック密度劣化による物理的なアクセス量の増加	• 適切なFILLFACTORでブロック密度を一定に保つ
	索引の物理的な状態が悪化	• 適切なFILLFACTORでHOTを有効にする • 索引の再構築
急に遅くなる		• 物理的な状態を一定に保つ • 複雑なSQLを避ける • ヒント句 • 統計情報を固定
開発中の性能テストはクリアしたが運用では遅い	テスト時、データの格納状態が物理的に最良の状態だっただけ	• 運用状態を見越したテスト（一定期間運用テストした後に性能テストを実施する）

PostgreSQL自習書で解説される部分

商用での安全な運用の勘所

- SQLの応答時間が一定であること
  - 表のブロック密度が劣化
    - PostgreSQLの読み取り一貫性は「追記型」を採用

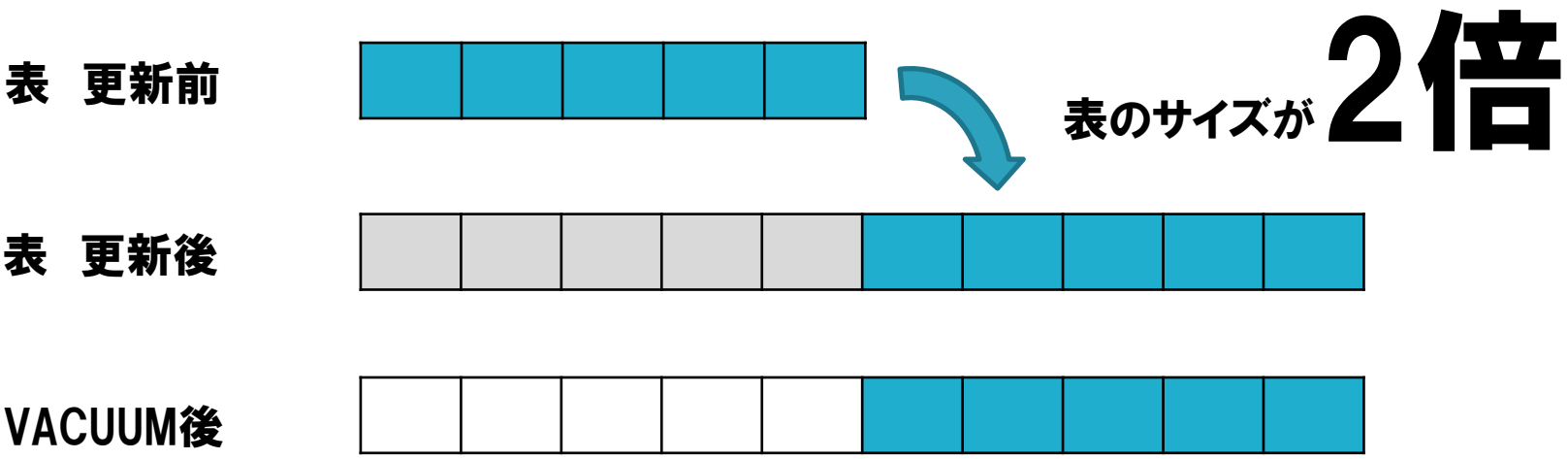


ブロックイメージ（実際の格納は上記の通りではありません）

商用での安全な運用の勘所

- SQLの応答時間が一定であること
  - 表のブロック密度が劣化
    - 全件更新したら表のサイズはどうなるか

      運用では処理フラグ(１バイト)を全件更新など



極端な例ですが、ブロックの利用率を定義するFILLFACTOR＝100%(デフォルト値)の場合、最良のサイズの2倍になります。全件読み込み時のアクセス量、キャッシュ使用量も2倍になります。

商用での安全な運用の勘所

- SQLの応答時間が一定であること
  - 索引のブロック密度が劣化
    - FILLFACTOR=100%(デフォルト)で表を更新すると更新カラムが索引カラムでなくても**索引も更新される**

表　更新前

行の格納ブロックの位置が変わる

表　更新後

更新前の索引

```
# select tree_level,index_size,avg_leaf_density from pgstatindex('pgbench_accounts_pkey')
 tree_level | index_size | avg_leaf_density
------------+------------+------------------
          2 |   11255808 |            90.03
```

更新後の索引

```
# select tree_level,index_size,avg_leaf_density from pgstatindex('pgbench_accounts_pkey');
 tree_level | index_size | avg_leaf_density
------------+------------+------------------
          2 |   22487040 |            45.18
```

索引まで更新される

商用での安全な運用の勘所

- SQLの応答時間が一定であること
  - FILLFACTORを適切に指定する。
    - 全件更新があると仮定し、FILLFACTOR=50%で指定

表　更新前

表のサイズは変わらない（100%のときの2倍ではあるが）

表　更新後

更新行は同一ブロック内に書き込まれる

更新前の索引

```
# select tree_level,index_size,avg_leaf_density from pgstatindex('pgbench_accounts_pkey');
 tree_level | index_size | avg_leaf_density
------------+------------+------------------
          2 |   11255808 |            90.03
```

更新後の索引

```
# select tree_level,index_size,avg_leaf_density from pgstatindex('pgbench_accounts_pkey');
 tree_level | index_size | avg_leaf_density
------------+------------+------------------
          2 |   11255808 |            90.03
```

索引はそのまま

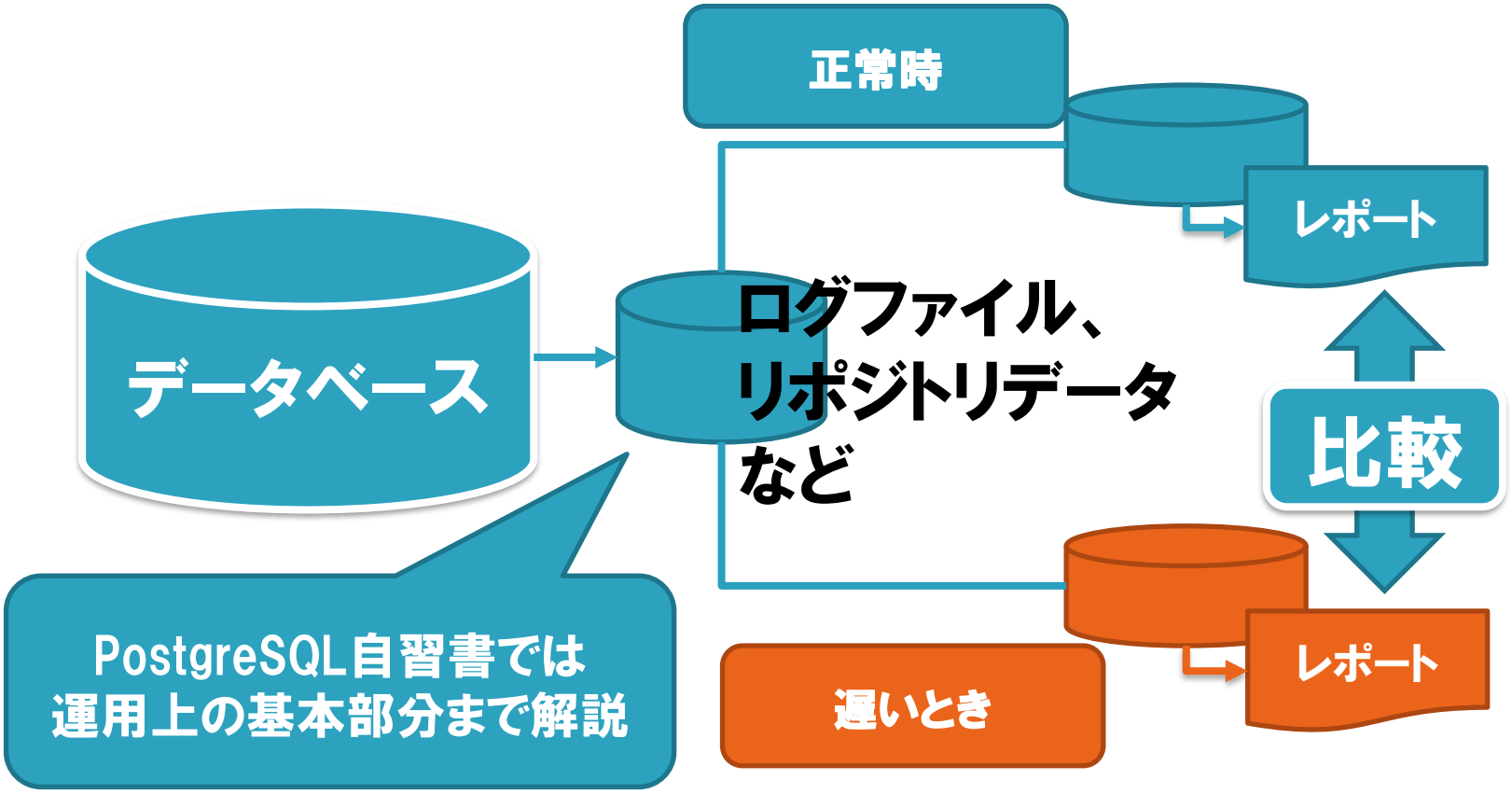
商用での安全な運用の勘所

- SQLの応答時間が一定であること

運用の勘所	効果
表毎にFILLFACTORを適切に指定しましょう	• SQL応答のバラツキが減少する • 索引の更新が減少する • 実行計画の変化が抑えられる

商用での安全な運用の勘所

- SQLが遅くなったときに原因が特定できること
  - 正常時と何が違うか



商用での安全な運用の勘所

- SQLが遅くなったときに原因が特定できること
  - PostgreSQLの標準機能から取得できる情報

情報	手段
現時点の情報 または 現時点までの累積情報	• システムカタログ • 統計情報コレクタ • pg_stat_statementsなどの追加モジュール
過去の時系列的な情報	サーバログ （何を出力するかはパラメータファイルで調整）

過去の情報はサーバログのみ

商用での安全な運用の勘所

- SQLが遅くなったときに原因が特定できること
  - PostgreSQLの標準機能の情報からできる対応
    - バッチ処理で遅いSQLを知りたい

      サーバログで確認する。

      log_min_duration_statementで出力閾値を指定

      →遅いSQLはわかるが、なぜ遅いかは不明

    - バッチ処理で遅いSQLの実行計画を知りたい

      サーバログで確認する。

      auto_explainで出力閾値を指定

      →実行計画から“遅そうな部分”を想定

商用での安全な運用の勘所

- SQLが遅くなったときに原因が特定できること
  - PostgreSQLの標準機能の情報からできる対応
    - オンライン処理で遅いSQLを知りたい

      調査時点でも遅い場合

      ・explain analyzeでSQLのどの実行部分が遅いのか確認

      ・DMLの場合は本番データに影響を与えないように工夫
      - 調査用ダミーデータを事前に用意しておく
      - explain analyzeをトランザクションで実行し確認後にrollback

商用での安全な運用の勘所

- SQLが遅くなったときに原因が特定できること
  - PostgreSQLの標準機能の情報からできる対応
    - バッチ、オンライン共通
    - pgstattupleで表のデータ格納状態が劣化していないか
    - pgstatindexで索引の状態が劣化していないか
    - pg_stat_all_tablesで想定外にseq処理が多くないか
    - pg_stat_all_indexで想定した索引が使われているか

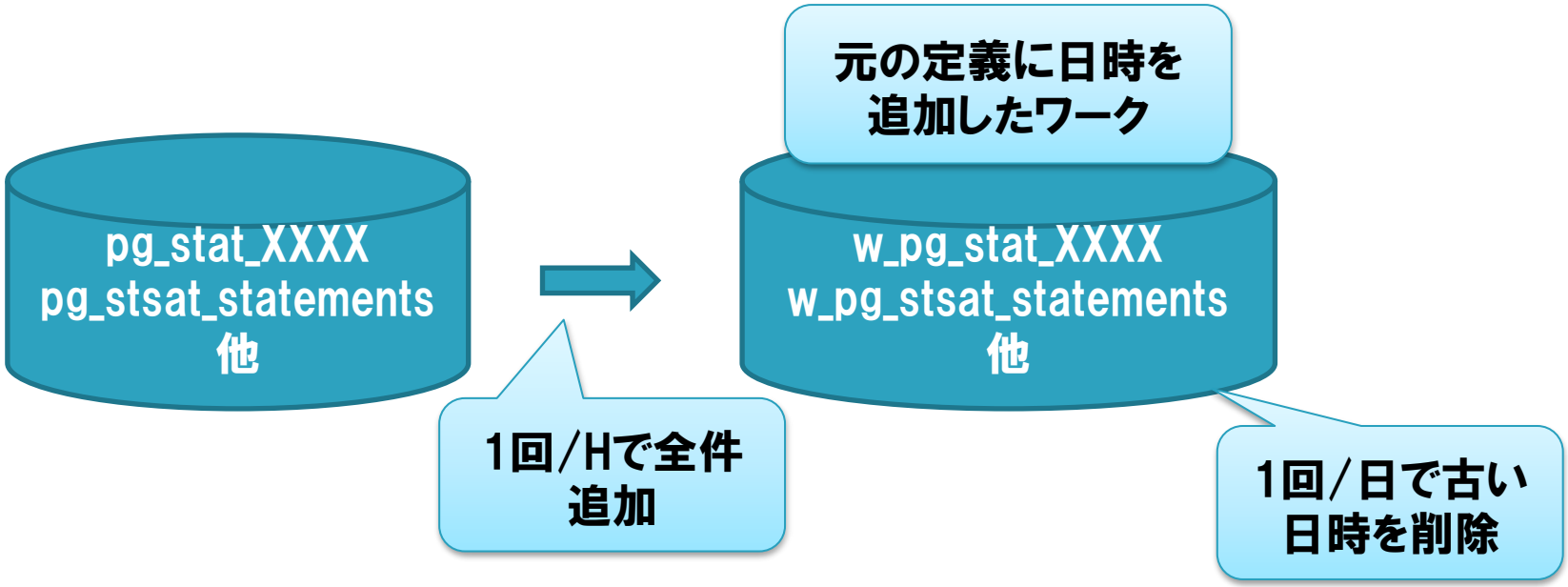
    など、全般的な状況から原因を絞り込む

欠点：統計情報コレクタはリセットしてからの累積のため、
過去の問題のあった“時間帯”の発生量はわからない

商用での安全な運用の勘所

- SQLが遅くなったときに原因が特定できること
  - PostgreSQLの標準機能の情報からできる対応
    - バッチ、オンライン共通（前スライド欠点の解決案）

      統計情報コレクタを累積して調査したい時間帯を終了時間ー開始時間の発生の差分から求める



商用での安全な運用の勘所

- SQLが遅くなったときに原因が特定できること
  - PostgreSQLの標準機能の情報からできる対応
    - バッチ、オンライン共通（解決案の実施例）

pg_stat_statementsを累積し任意時間の差分から分析

start_time	end_time	ユーザ名	クエリID	クエリ	実行回数	実行時間(ms)	平均実行時間(ms)	影響受けた行数
2019-09-10 18:00:00+09	2019-09-10 19:00:00+09	postgres	831879657	UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE tid = $2	5,000	145,593.28	29.12	5,000
2019-09-10 18:00:00+09	2019-09-10 19:00:00+09	fsepuser	831879657	UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE tid = $2	5,000	141,064.75	28.21	5,000
2019-09-10 18:00:00+09	2019-09-10 19:00:00+09	postgres	3872086279	UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE bid = $2	5,000	138,750.83	27.75	5,000
2019-09-10 18:00:00+09	2019-09-10 19:00:00+09	fsepuser	3872086279	UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE bid = $2	5,000	115,700.57	23.14	5,000
2019-09-10 18:00:00+09	2019-09-10 19:00:00+09	postgres	3371683456	UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE aid = $2	5,000	1,243.99	0.25	5,000
2019-09-10 18:00:00+09	2019-09-10 19:00:00+09	fsepuser	3371683456	UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE aid = $2	5,000	879.24	0.18	5,000

開始-終了時刻間での発生量

商用での安全な運用の勘所

- SQLが遅くなったときに原因が特定できること
  - PostgreSQLの周辺ソフトウェアから取得できる情報

情報	手段(例)
現時点までの累積情報	pg_store_plans (SQL,実行計画毎に実行統計が累積される)
過去の時系列的な情報	pg_statsinfo(+pg_store_plans)

上記はLinux版のみ利用可能。
Windows版,クラウドのデータベースサービス上では使用できません。
→ クラウドサービスが提供するツールかP24のようなスライドの対策で対応

商用での安全な運用の勘所

- SQLが遅くなったときに原因が特定できること
  - PostgreSQLの周辺ソフトウェアから情報からできる対応
    pg_statsinfo(＋pg_store_plans)で運用している場合

    - バッチ処理、オンライン処理共通
    - 正常な時間帯と遅い時間帯でレポートを出力し比較

    pg_statsinfoのレポートには実行計画は出力されないのでレポート内の遅いSQLのQuery ID、Plan IDでpg_stat_statementsとpg_store_plansビューからSQL文と実行計画も別途取得(→　次スライドに調査イメージ)
    するかpg_stats_reporterを追加で導入しコマンドライン機能でレポート出力するとよい。

商用での安全な運用の勘所

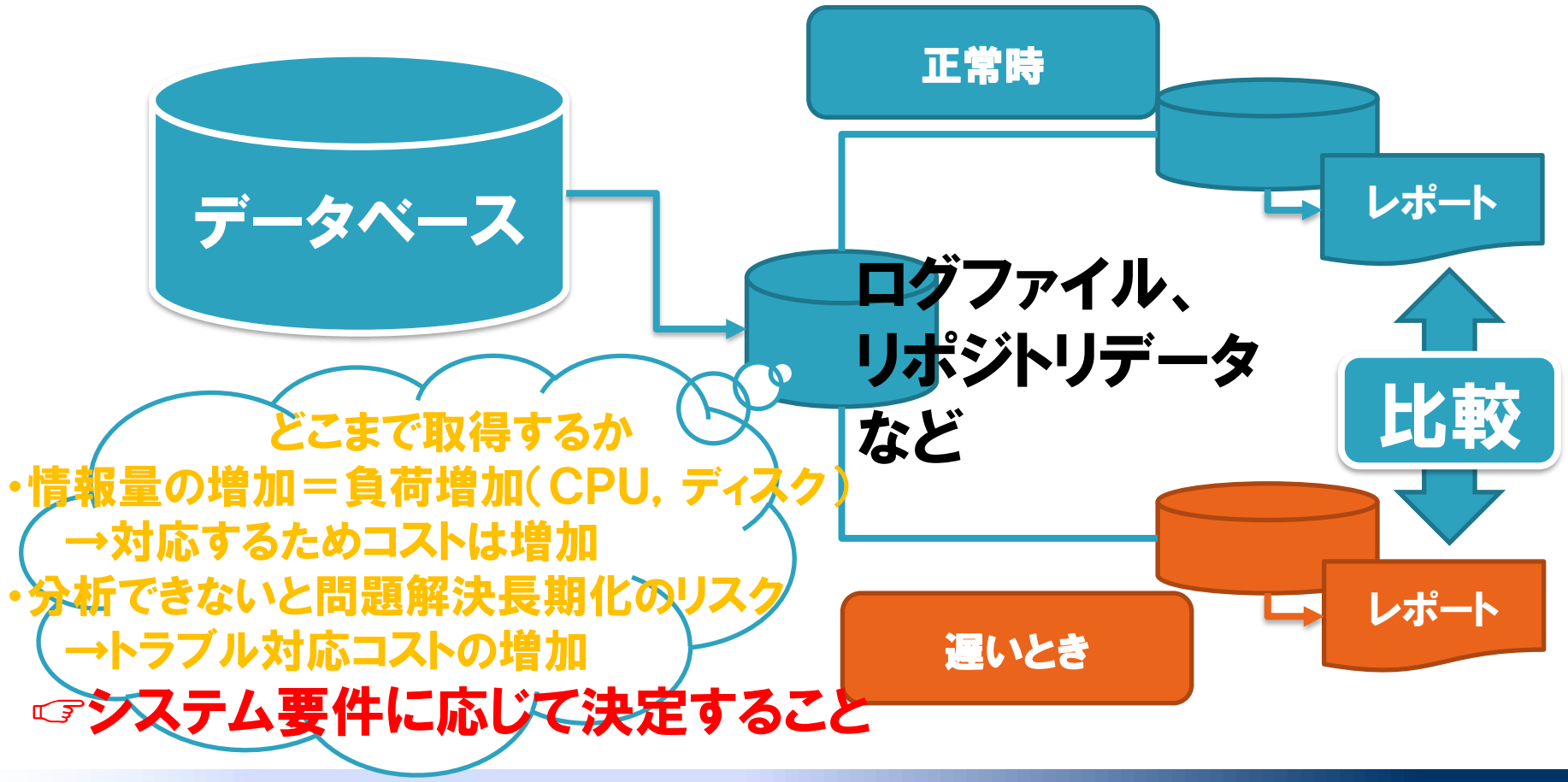
- SQLが遅くなったときに原因が特定できること
  - PostgreSQLの周辺ソフトウェアから情報からできる対応

```
test01=# select pss.query,psp.* from pg_store_plans as psp,pg_stat_statements as pss
where psp.queryid_stat_statements = pss.queryid
and psp.queryid_stat_statements = -1228162415474808213;
-[ RECORD 1 ]----------+-----------------------------------------------------------------------------------------------------------
query                  | UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE tid = $2
userid                 | 10
dbid                   | 26809
queryid                | 1445217560
planid                 | 43793328
queryid_stat_statements | -1228162415474808213
plan                   | Update on pgbench_tellers  (cost=0.14..8.16 rows=1 width=358)                                          +
                       |   ->  Index Scan using pgbench_tellers_pkey on pgbench_tellers  (cost=0.14..8.16 rows=1 width=358)+
                       |         Index Cond: (tid = 2)
calls                  | 810800
total_time             | 32290370.74642882
min_time               | 0.015441
max_time               | 1444.273831
mean_time              | 39.82532159155185
stddev_time            | 62.29389686886388
rows                   | 810800
shared_blks_hit        | 5926214
shared_blks_read       | 1
shared_blks_dirtied    | 205
shared_blks_written    | 17
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
first_call             | 2021-09-29 10:44:04.066434+09
last_call              | 2021-09-29 13:57:27.097104+09
```

調査イメージ

商用での安全な運用の勘所

- SQLが遅くなったときに原因が特定できること
  - 運用の勘所



PGEConsウェブサイト

- https://www.pgecons.org/

会員募集

- 正会員・一般会員を広く募集いたします
  - WG・CR部会で一緒に活動を行っていただける団体様 ⇒ 正会員
  - PostgreSQLのエンタープライズ領域に興味を持っている団体様 ⇒ 一般会員

お問い合わせ先：
PostgreSQLエンタープライズ・コンソーシアム事務局
メール : jimukyoku@pgecons.org
Web : http://www.pgecons.org/

PGECons

PostgreSQL Enterprise Consortium