マネージドデータベースサービスで利用できる PostgreSQL の拡張モジュール解説

2019/02/22

SRA OSS, Inc. 日本支社

内容

- クラウドサービスの BIG 3（AWS, Azure, GCP)
- マネージドサービスとしての PostgreSQL
- PostgreSQL の拡張モジュール
  - 拡張モジュールの紹介
  - 各クラウドサービスの対応状況

クラウドサービスの BIG 3

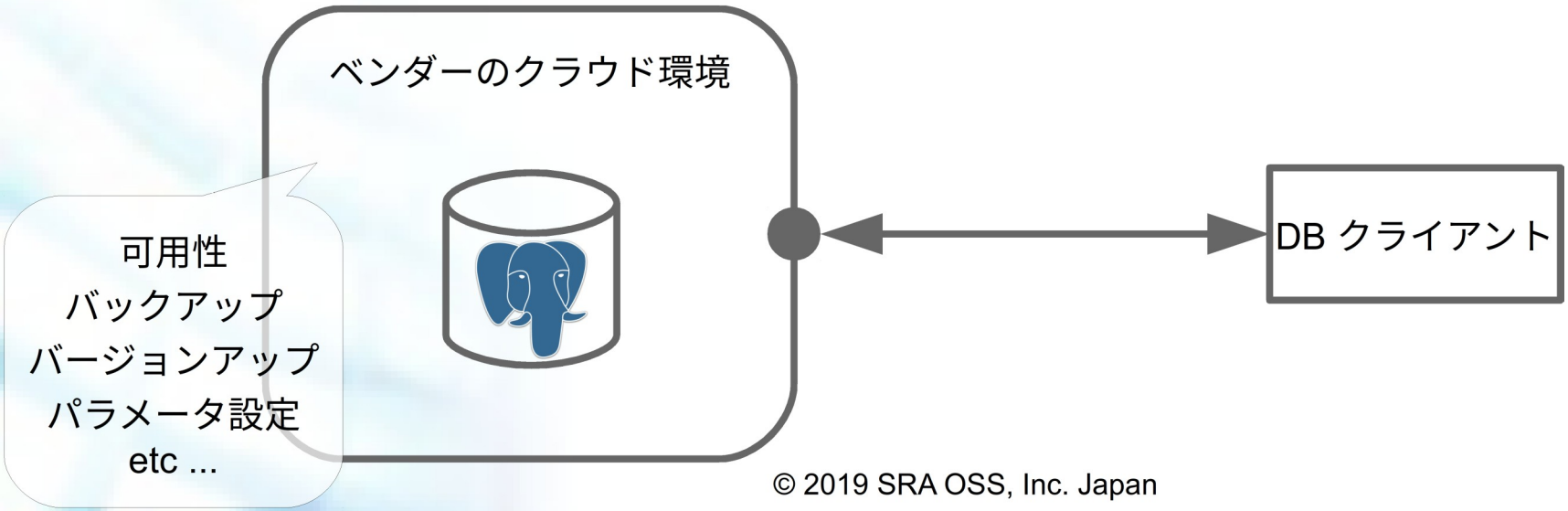
- AWS (Amazon Web Service)
- Azure (Microsoft Azure)
- GCP (Google Cloud Platform)

マネージドサービスとしての PostgreSQL

- AWS, Azure, GCP のそれぞれが PaaS として PostgreSQL を提供している　括弧内は対応メジャーバージョン
  - AWS:　Amazon RDS (PostgreSQL 10)
  - Azure:　Azure Database for PostgreSQL (PostgreSQL 10)
  - GCP:　Cloud SQL for PostgreSQL (PostgreSQL 9.6)

PaaS として提供される PostgreSQL

- 資材の調達や運用の手間はベンダーが代行・便利な仕組みを提供
- ユーザはベンダーが提供する PostgreSQL のエンドポイントにアクセスすれば良い



PostgreSQL の拡張モジュール

- PostgreSQL のコア機能に組み込まれていないプラグイン機能
  - 移植用ツール、解析ユーティリティ、利用者が限定されている、コア機能に含めるには実験的すぎる
  - コア機能と比較して有用性が劣っているわけではない
- 最新の PostgreSQL 11 で 54 種類 + サードパーティー製
- Amazon RDS, Azure Database for PostgreSQL, Cloud SQL for PostgreSQL でもそれぞれ部分的に対応

本題

- マネージドデータベースサービスで利用できる PostgreSQL の拡張モジュールについて
  - 各サービスの対応状況は？
  - 拡張モジュールのうちオススメのものは？
  - 運用向け・開発向けに分類して紹介

運用: auto_explain

- 概要
  - 時間がかかった SQL 文の実行計画を自動的にログ出力してくれる
- 用途
  - チューニングの起点
  - 大規模アプリケーションで最適化されていない SQL を後から特定する
    - 調査にために一時的に使う

運用: auto_explain

- 使い方
  - postgresql.conf で設定
    - PostgreSQL 起動時に auto_explain をロード（再起動で設定反映）

```
shared_preload_libraries = 'auto_explain'
```

    - 3 sec 以上実行に時間のかかる SQL をログ出力

```
auto_explain.log_min_duration = '3s'
```

    - EXPLAIN ANALYZE を行なう（高負荷注意、デフォルトの off 推奨）

```
auto_explain.log_analyze = on
```

運用: auto_explain

- サンプル: ログ出力内容
  時間のかかった SQL のEXPLAIN の出力結果がログに出力される（正確には auto_explain.log_analyze=on なので EXPLAIN ANALYZE の結果）

```
LOG:  duration: 3.651 ms  plan:
  Query Text: SELECT count(*)
              FROM pg_class, pg_index
              WHERE oid = indrelid AND indisunique;
  Aggregate  (cost=16.79..16.80 rows=1 width=0) ↵
             (actual time=3.626..3.627 rows=1 loops=1)
    ->  Hash Join  (cost=4.17..16.55 rows=92 width=0) ↵
                   (actual time=3.349..3.594 rows=92 loops=1)
          Hash Cond: (pg_class.oid = pg_index.indrelid)
          ->  Seq Scan on pg_class  (cost=0.00..9.55 rows=255 width=4) ↵
                                    (actual time=0.016..0.140 rows=255 loops=1)
          ->  Hash  (cost=3.02..3.02 rows=92 width=4) ↵
                    (actual time=3.238..3.238 rows=92 loops=1)
                Buckets: 1024  Batches: 1  Memory Usage: 4kB
                ->  Seq Scan on pg_index  (cost=0.00..3.02 rows=92 width=4) ↵
                                          (actual time=0.008..3.187 rows=92 loops=1)
                      Filter: indisunique
```

運用: pg_hint_plan

- 概要
  - 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

運用: pg_hint_plan

- 使い方
  - 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;
```

  - DB 接続セッションで pg_hint_plan をロード

```
db1=# LOAD ‘pg_hint_plan’;
```

    - postgresql.conf で pg_hint_plan_enable_hint_table = on しておく必要がある
- ヒント句の指定方法
  - SQL にコメント指定
  - ヒント用テーブルに登録

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

運用: pg_hint_plan

- サンプル: SQL でヒント句を指定

```
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
-------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=563632.07..566132.07 rows=1000000 width=465)↵
       (actual time=942.519..1081.707 rows=1000000 loops=1)
   Sort Key: a.aid
   Sort Method: external sort  Disk: 119264kB
   ->  Hash Join  (cost=1.23..40145.22 rows=1000000 width=465)↵
                  (actual time=0.038..395.556 rows=1000000 loops=1)
         Hash Cond: (a.bid = b.bid)
         ->  Seq Scan on pgbench_accounts a  (cost=0.00..26394.00 rows=1000000 width=97)↵
                                             (actual time=0.020..103.247 rows=1000000 loops=1)
         ->  Hash  (cost=1.10..1.10 rows=10 width=364)↵
                   (actual time=0.010..0.010 rows=10 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on pgbench_branches b  (cost=0.00..1.10 rows=10 width=364)↵
                                                   (actual time=0.003..0.007 rows=10 loops=1)
 Planning time: 0.134 ms
 Execution time: 1201.389 ms
```

運用: pg_hint_plan

- サンプル: SQL でヒント句を指定

```
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
-----------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=503632.01..506132.01 rows=1000000 width=465)↵
       (actual time=942.519..1081.707 rows=1000000 loops=1)
   Sort Key: a.aid
   Sort Method: external sort  Disk: 119264kB
   ->  Hash Join  (cost=1.23..40145.22 rows=1000000 width=465)↵
                  (actual time=0.038..395.556 rows=1000000 loops=1)
         Hash Cond: (a.bid = b.bid)
         ->  Seq Scan on pgbench_accounts a  (cost=0.00..26394.00 rows=1000000 width=97)↵
                                             (actual time=0.020..103.247 rows=1000000 loops=1)
         ->  Hash  (cost=1.10..1.10 rows=10 width=364)↵
                   (actual time=0.010..0.010 rows=10 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on pgbench_branches b  (cost=0.00..1.10 rows=10 width=364)↵
                                                   (actual time=0.003..0.005 rows=10 loops=1)
 Planning time: 0.134 ms
 Execution time: 1201.389 ms
```

/*+{{ヒント}}{{ヒント}}…*/というコメント形式でヒント句を指定する

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_prewarm


- 概要

  任意のテーブルかインデックスを共有バッファまたは OS キャッシュに載せる

- 用途
  - 性能向上
  - 使いどころは DB の再起動直後
  - 早期にデータをメモリヒットさせたい
  - バッチの前処理で時間短縮

運用: pg_prewarm

- 使い方
  - 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_prewarm;
```

  - ついでに: キャッシュヒットを確認するために pg_buffercache も登録

```
db1=# CREATE EXTENSION pg_buffercache;
```

    - pg_buffercache 
      - 共有バッファの状況が確認できる

運用: pg_prewarm

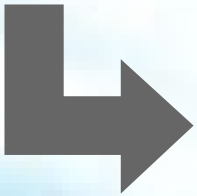

- サンプル
  - pg_prewarm 前

```
db1=# SELECT count(*)
      FROM
        pg_buffercache
      WHERE
        Relfilenode = (
          SELECT
            relfilenode
          FROM pg_class
          WHERE relname = 'pgbench_accounts');
 count
-------
     0
(1 row)
```

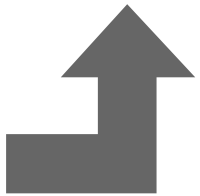
  - pg_prewarm 後

```
db1=# SELECT count(*)
      FROM
        pg_buffercache
      WHERE
        Relfilenode = (
          SELECT
            relfilenode
          FROM pg_class
          WHERE relname = 'pgbench_accounts');
 count
-------
 16295
(1 row)
```

  - pg_prewarm



```
db1=# SELECT * FROM pg_prewarm('pgbench_accounts');
 pg_prewarm
------------
     163935
(1 row)
```



運用: pg_repack

- 概要
  - オンラインでテーブルやインデックスの再編成ができる
- 用途
  - メンテナンス
  - サービス中にどうしても VACUUM FULL したい
  - サービス中にどうしても REINDEX したい

運用: pg_repack

- 使い方
  - 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_repack;
```

  - クライアント端末に pg_repack をインストール

```
$ wget https://github.com/reorg/pg_repack/archive/ver_1.4.4.tar.gz
$ tar xf ver_1.4.4.tar.gz
$ cd pg_repack_1.4.4
$ make
$ sudo make install;
```

運用: pg_repack

- サンプル: エンドポイントに向けて pg_repack
  - オンライン VACUUM FULL と REINDEX

```
$ pg_repack --table=t1 -dbname=db2 --username=postgres↵
--host=... --port=...
```

  - オンライン CLUSTER と REINDEX

```
$ pg_repack --order-by=col_id -table=t1↵
--dbname=db2 --username=postgres --host=... --port=...
```

  - Db2 の可能なテーブルにオンライン CLUSTER、全テーブルにオンライン VADUUM FULL と REINDEX

```
$ pg_repack --dbname=db2 --username=postgres --host=... --port=...
```

運用: pg_repack

- 注意点 1/2:
  - 対象テーブルとインデックスの 2 倍以上のディスク空き領域が必要。
  - pg_repack の実行中には、VACUUM と ANALYZE 以外の DDL を実行してはいけない。
  - 実行対象テーブルには、主キー、または NOT NULL 属性とユニークインデックスを持つカラムが少なくとも一つ必須。

運用: pg_repack

- 注意点 2/2:
  - 一時テーブル、または GiST インデックスがクラスタインデックスとなっているテーブルは操作の対象外。
  - -T (--wait-timeout) で指定するタイムアウト時間（デフォルト60秒）を超えて pg_repack がロック取得ができない場合には他プロセスを終了させる動作が生じる。
    - ロック取得できなかった場合に pg_repack の方を中断させるには -D (--no-kill-backend) を指定する。

運用: pg_stat_statements

- 概要
  - 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 ログに SQL を大量に出したくない

運用: pg_stat_statements


- 使い方
  - postgresql.conf で設定
    - PostgreSQL 起動時に pg_stat_statements をロード

```
shared_preload_libraries = 'pg_stat_statements'
```

  - 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;
```

運用: pg_stat_statements


- サンプル

```
db1=# SELECT query, calls, total_time, ROWS,
        100.0 * shared_blks_hit / nullif (
          shared_blks_hit + shared_blks_read, 0) AS hit_raito
      FROM pg_stat_statements ORDER BY total_time DESC LIMIT 5;
                  query                    | calls | total_time | rows  | hit_raito
-------------------------------------------+-------+------------+-------+-----------
 UPDATE pgbench_accounts SET abalance = a↵| 72294 | 19389.5610 | 72294 |  86.72913
 balance + ? WHERE aid = ?;                |       |            |       |
 UPDATE pgbench_branches SET bbalance = b↵| 72294 | 4499.92100 | 72294 | 100.00000
 balance + ? WHERE bid = ?;                |       |            |       |
 UPDATE pgbench_tellers SET tbalance = tb↵| 72294 | 2607.26099 | 72294 | 100.00000
 alance + ? WHERE tid = ?;                 |       |            |       |
 SELECT abalance FROM pgbench_accounts WH↵| 72294 | 1979.69899 | 72294 | 100.00000
 ERE aid = ?;                              |       |            |       |
 INSERT INTO pgbench_history (tid, bid, a↵| 72294 | 1381.01899 | 72294 |  99.36885
 id, delta, mtime) VALUES (?, ?, ?, ?, CU↵|       |            |       |
 RRENT_TIMESTAMP);                         |       |            |       |
(5 rows)
```

運用: pg_stat_statements

- サンプル

```
db1=# SELECT query, calls, total_time, ROWS,
        100.0 * shared_blks_hit / nullif (
          shared_blks_hit + shared_blks_read, 0) AS hit_ratio
      FROM pg_stat_statements ORDER BY total_time DESC LIMIT 5;
                 query                       | calls | total_time | rows  | hit_raito
---------------------------------------------+-------+------------+-------+-----------
 UPDATE pgbench_accounts SET abalance = a↵| 72294 | 19389.5610 | 72294 | 86.72913
 balance + ? WHERE aid = ?;                  |       |            |       |
 UPDATE pgbench_branches SET bbalance = b↵| 72294 | 4499.92100 | 72294 | 100.00000
 balance + ? WHERE bid = ?;                  |       |            |       |
 UPDATE pgbench_tellers SET tbalance = tb↵| 72294 |            | 72294 | 100.00000
 alance + ? WHERE tid = ?;                   |       |            |       |
 SELECT abalance FROM pgbench_accounts WH↵| 72294 | 1979.69899 | 72294 | 100.00000
 ERE aid = ?;                                |       |            |       |
 INSERT INTO pgbench_history (tid, bid, a↵| 72294 | 1381.01899 | 72294 | 99.36885
 id, delta, mtime) VALUES (?, ?, ?, ?, CU↵|       |            |       |
 RRENT_TIMESTAMP);                           |       |            |       |
(5 rows)
```

実行回数

SQLの影響を受けた行数

SQLの内容

総実行時間

キャッシュヒット率

定数値は？に置換されて一つのSQLにまとめられる

運用: pgstattuple

- 概要
  - テーブルの行単位の統計情報やインデックスの統計情報が分かる（統計情報には不要領域情報も含まれる）
- 用途
  - メンテナンス
  - VACUUM やインデックス再構築のタイミングを主体的に判断したい

運用: pgstattuple


- 使い方
  - 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 pgstattuple;
```

  - 関数が提供される

- 指定テーブルの行レベル情報
  - pgstattuple(regclass) returns record
    - 全件走査が発生
  - pgstattuple_approx(regclass) returns record
    - 全件走査を避けて近似値を返す
- 指定テーブルのインデックス情報
  - pgstatindex(regclass) returns record
    - B-tree 向け
  - pgstatginindex(regclass) returns record
    - GIN 向け

運用: pgstattuple

- サンプル: テーブル断片化の確認 SQL

```
db1=# SELECT
        objectname
        , pg_size_pretty(pg_relation_size(objectname)) AS total_size
        , app.approx_tuple_count, app.approx_tuple_percent
        , app.dead_tuple_count, app.dead_tuple_percent
        , app.approx_free_space, app.approx_free_percent
      FROM (
        SELECT
            *
            , quote_ident(schemaname) || '.' || quote_ident(relname) AS objectname
        FROM
            pg_stat_user_tables
      ) AS t, pgstattuple_approx(objectname::regclass) AS app
      ORDER BY app.approx_free_percent DESC;
```

- pg_stat_user_tables からテーブル情報を取得してテーブル毎に行レベルの統計情報を表示

運用: pgstattuple


- サンプル: テーブル断片化の確認 SQL 実行結果

```
-[ RECORD 1 ]--------+-------------------------
objectname           | public.pgbench_branches // テーブル名
total_size           | 8192 bytes              // テーブルサイズ
approx_tuple_count   | 100                     // 推定有効行数
approx_tuple_percent | 53.90625                // 有効行数の割合
dead_tuple_count     | 0                       // 無効行数
dead_tuple_percent   | 0                       // 無効行数の割合
approx_free_space    | 3776                    // 推定空き容量サイズ
approx_free_percent  | 46.09375                // 空き容量の割合
（後略）
```

- テーブルサイズ（大）& 空き容量の割合（大）
  - テーブルの肥大化 → VACUUM FULL の必要あり
- テーブルサイズ（大）& 無効行数の割合（大）
  - VACUUM が実行されていない → VACUUM 実行か、阻害要因を取り除く

運用: pgstattuple

- サンプル: インデックス断片化の確認 SQL

```
db1=# SELECT
  objectname
  , pg_size_pretty(pg_relation_size(objectname)) AS total_size
  , app.tree_level, app.index_size, app.root_block_no
  , app.internal_pages, app.leaf_pages, app.empty_pages
  , app.deleted_pages, app.avg_leaf_density, app.leaf_fragmentation
FROM (
  SELECT
    *, quote_ident(schemaname) || '.' || quote_ident(indexrelname) AS objectname
  FROM
    pg_stat_user_indexes) AS t, pgstatindex (objectname::regclass) AS app
ORDER BY
  app.leaf_fragmentation DESC;
```

  - pg_stat_user_indexes からインデックス情報を取得してインデックス毎に行レベルの統計情報を表示

運用: pgstattuple


- サンプル: インデックス断片化の確認 SQL 実行結果

```
-[ RECORD 1 ]------+-----------------------------
objectname         | public.pgbench_tellers_pkey  // インデックス名
total_size         | 56 kB                        // インデックスサイズ
tree_level         | 1                            // ルートページのツリーレベル
index_size         | 57344                        // バイト単位のインデックスサイズ
root_block_no      | 3                            // ルートページの場所
internal_pages     | 1                            // 上位レベルページ数
leaf_pages         | 5                            // リーフページ数
empty_pages        | 0                            // 空ページ数
deleted_pages      | 0                            // 削除ページ数
avg_leaf_density   | 49.31                        // リーフページの平均密度
leaf_fragmentation | 40                           // リーフページの断片化
（後略）
```

  - avg_leaf_density（大）、leaf_fragmentation（小）、tree_level（小）→ 望ましい
  - リーフページの断片化（大）→ REINDEX または CREATE INDEX CONCURRENTLY の必要あり

開発: pg_trgm


- 概要
  - 3-gram 方式の全文検索機能
- 用途
  - 全文検索
  - 2 文字の全文検索は行わない
  - 検索ワードが何度も登場しない短めのドキュメントが対象

開発: pgcrypto


- 概要
  - データを暗号化、複合するための関数を提供
- 用途
  - セキュリティ
  - データの情報漏洩に対応したい
  - PostgreSQL 独自の SQL を発行してもいい

開発: pl/pgsql

- 概要
  - PostgreSQL 独自言語でユーザ定義関数を作成できる
- 用途
  - DB サーバ側で SQL よりも複雑な処理や計算をしたい
  - DB サーバ側の処理をアプリケーションから隠蔽したい

補足: その他の pl/○○系の対応状況

- pl/coffee 

- pl/ls 

- pl/perl 

- pl/tcl 

- pl/v8 

開発: PostGIS

- 概要
  - 緯度経度による位置を持った地点や経路線、領域などの要素を操作できる
- 用途
  - 地理情報システムを実現したい

対応状況一覧

auto_explain	AWS			https://www.postgresql.jp/document/10/html/auto-explain.html
pg_hint_plan	AWS			http://pghintplan.osdn.jp/pg_hint_plan-ja.html
pg_prewarm	AWS			https://www.postgresql.jp/document/10/html/pgprewarm.html
pg_repack	AWS			https://github.com/reorg/pg_repack
pg_stat_statements	AWS			https://www.postgresql.jp/document/10/html/pgstatstatements.html
pgstattuple	AWS			https://www.postgresql.jp/document/10/html/pgstattuple.html
pg_trgm	AWS			https://www.postgresql.jp/document/10/html/pgtrgm.html
pgcrypto	AWS			https://www.postgresql.jp/document/10/html/pgcrypto.html
pl/pgsql	AWS			https://www.postgresql.jp/document/10/html/plpgsql.html
PostGIS	AWS			http://postgis.net/
postgres_fdw	AWS			https://www.postgresql.jp/document/10/html/postgres-fdw.html

補足

- マネージドデータベースサービスで**利用できない** PostgreSQL の拡張モジュールについて
  - 拡張モジュールのうちオススメのものは？

なぜ利用できない？

- いずれもサードパーティー製の拡張モジュール
- DB サーバ側にインストール作業が発生してしまう
- マネージドデータベースサービスなので SSH 接続は不可能

運用: pg_dbms_stats

- 概要
  - 実行計画をユーザが固定できる
- 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

運用: pg_bulkload

- 概要
  - 高速なデータローダツール
- 用途
  - とにかく早くデータをロードさせたい
  - サーバ移行
  - 定期的に溜まったデータをテーブルに流し込む
  - テスト用の大量のデータを少しでも早く読み込ませたい

開発: pg_bigm

- 概要
  - 2-gram 方式の全文検索機能
- 用途
  - 全文検索