PostgreSQL のインデックスを使い倒す

2020-11-13　PostgreSQL カンファレンス T3

日本 PostgreSQL ユーザ会
高塚 遥

【当日版】

TOC ：

- インデックスとは
- 様々なインデックス
- 利用の技法
- 保守の技法

講演者：

- 高塚 遥
- 日本 PostgreSQL ユーザ会 理事
- 仕事ではヘルプデスク、コンサルティングなど、PostgreSQL 支援業務

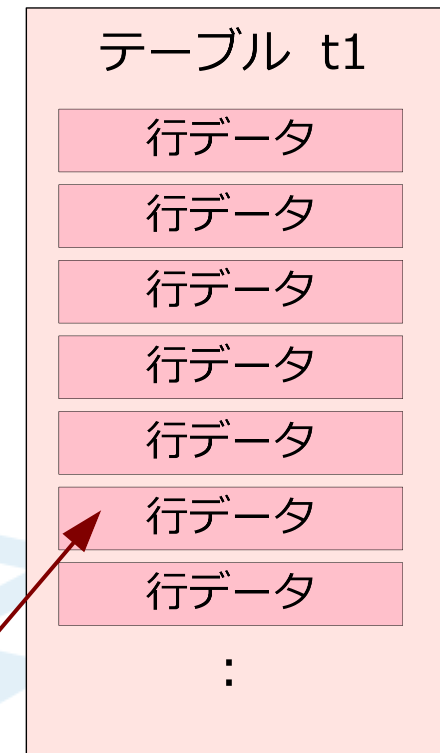
インデックスとは (1)

- 検索条件に当てはまる行の位置を素早く見つけるためのデータ構造

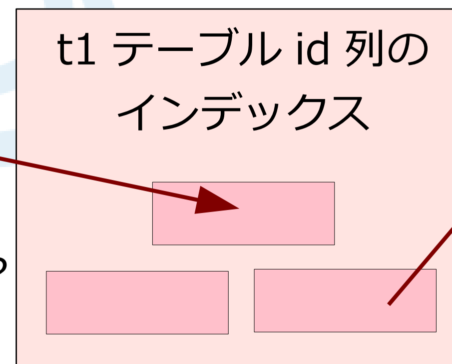
```
SELECT * FROM t1 WHERE id = 123;
```

```
CREATE INDEX idx_t1_id ON t1 (id);
```

順スキャン



インデックス検索



id=123
のデータは？

ここにあります

インデックスとは（２）

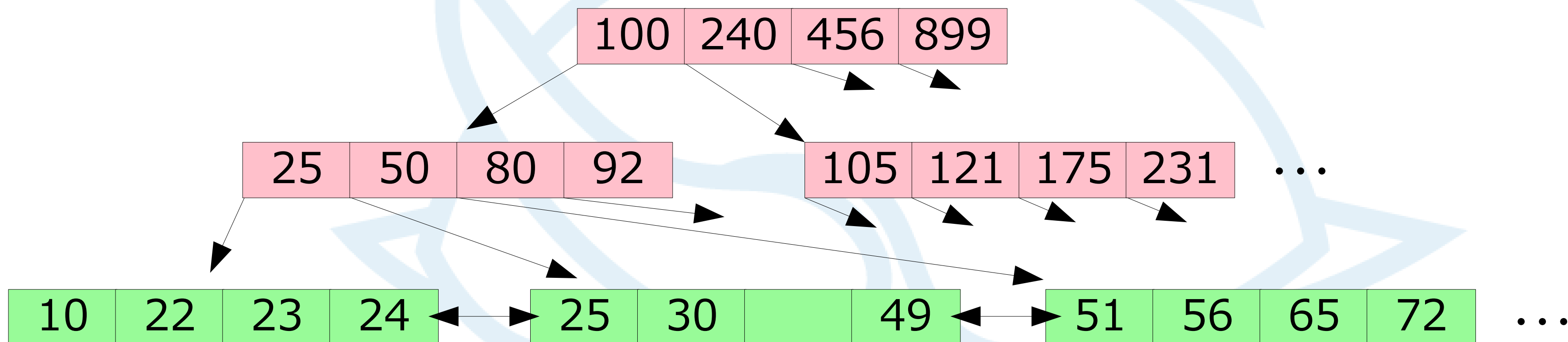
- データ型　×　種類 ×　方式
- 種類（ＡＭ）
  - btree 、 hash 、 gin 、 gist 、 spgist 、 bloom
- 方式（演算子クラス、～ ops ）
  - text 型に対する btree インデックス の text_ops （ロケールあり） と text_pattern_ops

```
CREATE INDEX idx_t1_val ON t1
  USING btree (val text_pattern_ops);
```

- 演算子族
  - 似た型に対する演算子クラスをグルーピング

Btree インデックス

- デフォルトで使われるツリー構造のインデックス
  - 一致比較、大小比較、ソート、
    前方一致パターンマッチ、主キー／ユニーク制約
  - 大きい列値には適さない（e.g. 長い文字列）

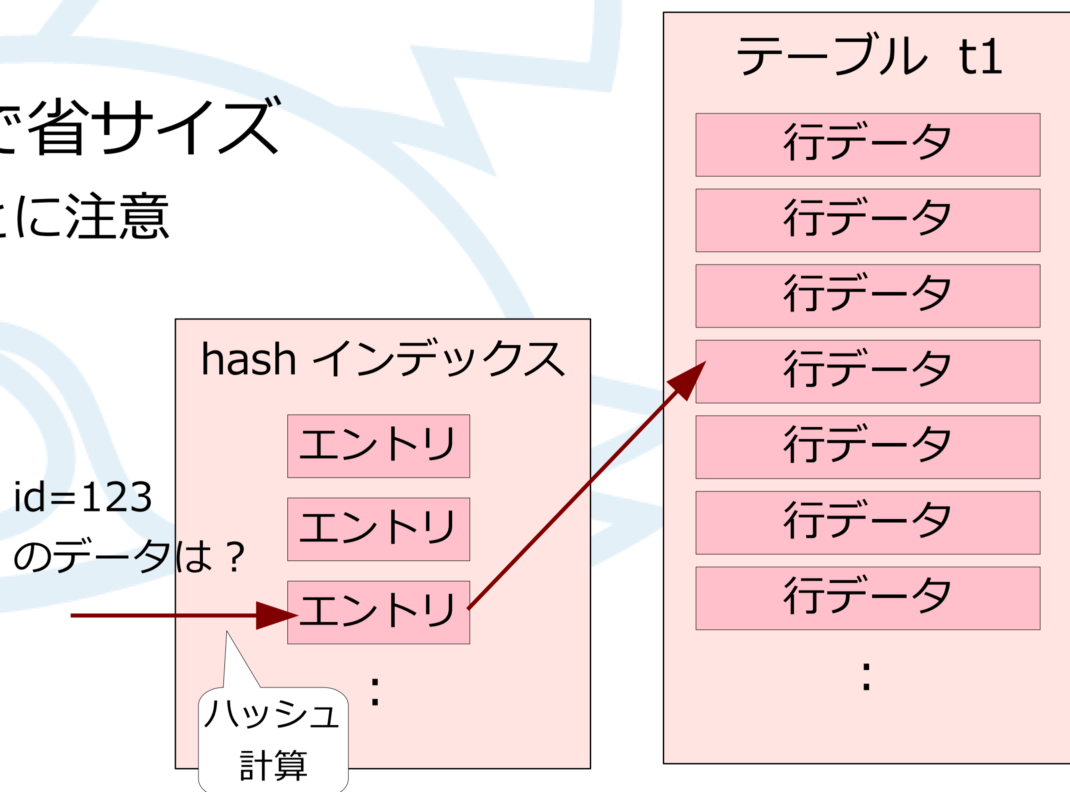


※ 実際はもっと横に長く、
1 ページ（デフォルト 8KB ）が 1 ノード。
4 バイトの int 型で 1 ノードに 367 件、
ツリー部分 2 階層で 3000 万件以上格納できる。

PostgreSQL
http://www.postgresql.jp

Hash インデックス

- ハッシュ表によるインデックス
  - 値のハッシュ値 → インデックスエントリ格納位置
    - 件数が増えても遅くならない
      - 例外：大量の重複値
    - ハッシュ値格納なので省サイズ
      - 空スペースも多いことに注意
    - データ増に対応した アルゴリズム
  - 一致比較のみ対応
  - v10 から WAL 対応



PostgreSQL
http://www.postgresql.jp

GIN インデックス

- 汎用転置インデックス（Generic Inverced iNdex）
  - 複数要素を持つデータ型への「を含むもの」検索むけ
    - 配列、 JSONB 、全文検索に使われる
    - 大きくなりがち、更新は遅い
    - Bitmat Index Scan になる

fastupdate 格納パラメータ（デフォルト有効）

配列型の列を持つテーブル

```
1 | { 和食 , 個室 }
2 | { フレンチ , 夜景 , 個室 }
3 | { イタリアン }
4 | { 和食 , 夜景 }
```

転置 →

転置インデックス内容

```
和食       | {1, 4}
個室       | {1, 2}
夜景       | {2, 4}
フレンチ   | {2}
イタリアン | {3}
```

```
SELECT * FROM t_restaurant WHERE tag @> ARRAY[' 個室 '];
```

PostgreSQL
http://www.postgresql.jp

BRIN インデックス

- Block Range Index
  - ページ範囲（デフォルト 128 ページ）毎に最大最小値を保持
  - 挿入のみ大量データの単調増加列に適する
    - ログのタイムスタンプや連番キー列
    - 値と物理格納位置が連動していることが前提
    - 極めて小さいサイズ、 Btree と遜色ない性能
  - 大小比較、一致比較に利用可能
  - Bitmap Index Scan になる

※ 主キー制約には利用できない

タイムスタンプに対する
BRIN インデックス内容例

ページ範囲	最小値	最大値
(0,127)	2020-10-26 15:38:00	2020-11-13 10:00:00
(128,255)	2020-11-13 10:00:13	2020-12-25 23:22:11

PostgreSQL
http://www.postgresql.jp

GiST インデックス

- 汎用検索ツリー (Generalized Search Tree)
  - ツリー型インデックスの汎用フレームワーク
    - 個々の実装（ = 演算子クラス）毎に性質は異なる
    - 幾何データ（ R 木）や、範囲型と排他制約で使用

<table>
<tr><th>名前</th><th>インデックスされるデータ型</th><th>インデックス可能な演算子</th><th>順序付け演算子</th></tr>
<tr><td>box_ops</td><td>box</td><td>&& &> &< &<| >> << <<| <@ @> @ |&> |>> ~ ~=</td><td></td></tr>
<tr><td>circle_ops</td><td>circle</td><td>&& &> &< &<| >> << <<| <@ @> @ |&> |>> ~ ~=</td><td><-></td></tr>
<tr><td>inet_ops</td><td>inet, cidr</td><td>&& >> >>= > >= <> << <<= < <= =</td><td></td></tr>
<tr><td>point_ops</td><td>point</td><td>>> >^ << <@ <@ <@ <^ ~=</td><td><-></td></tr>
<tr><td>poly_ops</td><td>polygon</td><td>&& &> &< &<| >> << <<| <@ @> @ |&> |>> ~ ~=</td><td><-></td></tr>
<tr><td>range_ops</td><td>任意の範囲型</td><td>&& &> &< >> << <@ -|- = @> @></td><td></td></tr>
<tr><td>tsquery_ops</td><td>tsquery</td><td><@ @></td><td></td></tr>
<tr><td>tsvector_ops</td><td>tsvector</td><td>@@</td><td></td></tr>
</table>

PostgreSQL12 文書 より転載
［表 64.1 組込 GiST 演算子クラス］

SP-GiST インデックス

- 空間分割対応の汎用検索ツリー
  - 汎用フレームワーク、性質は個々の実装で異なる
    - 幾何データ（ KD 木、四分木）や、範囲型と排他制約で使われる、 text 型の基数木も

PostgreSQL12 文書 より転載［表 65.1　組込 SP-GiST 演算子クラス］

名前	インデックスされるデータ型	インデックス可能な演算子	順序付け演算子
kd_point_ops	point	<< <@ <^ >> >^ ~=	<->
quad_point_ops	point	<< <@ <^ >> >^ ~=	<->
range_ops	任意の範囲型	&& &< &> -|- << <@ = >> @>	
box_ops	box	<< &< && &> >> ~= @> <@ &<| <<| |>> |&>	
poly_ops	polygon	<< &< && &> >> ~= @> <@ &<| <<| |>> |&>	<->
text_ops	text	< <= = > >= ~<=~ ~<~ ~>=~ ~>~ ^@	
inet_ops	inet, cidr	&& >> >>= > >= <> << <<= < <= =	

Bloom インデックス

- ブルームフィルタによるインデックス
  - contrib 追加モジュール
  - 複数列インデックスの一部を使った問い合わせに有効
  - 一致比較のみ対応
  - Bitmap Index Scan　として使われる
    - 擬陽性が含まれ recheck が行われる
  - サイズ小さめ（パラメータ依存： サイズ 大 ⇔ 擬陽性 小）

対応データ型は
int 型と text 型のみ

```
CREATE INDEX ON t_survey USING bloom
  (q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12);

SELECT * FROM t_survey
  WHERE q1 = 5 AND q5 = 2 AND q10 = 8 AND q12 = 1;
```

インデックスが使われない

- インデックス使用はプランナが選択する
  - 全件を順スキャンする方が速い（と判定）
  - 無理やりインデックスを使わせてコスト / 時間を確認

```
db=> SET enable_seqscan TO off;
db=> EXPLAIN (ANALYZE, VERBOSE) SELECT ...
```

- 検索条件の形状　e.g. 左辺が式 `... WHERE col + 1 = 100;`
- データ型不一致　e.g. 暗黙キャストを追加定義している場合
- ロケール照合順序　一つのインデックスは一つの COLLATE むけ

PostgreSQL
http://www.postgresql.jp

複数列に対するインデックス

```
SELECT * FROM t_survey WHERE q2 = 1 AND q3 = 2 AND q4 = 3;
```

- 複数列 Btree インデックス
  - 一部の列を検索条件にする場合でも利用可
    - – 後方列の条件では非効率

```
CREATE INDEX ON
 t_survey (q1, q2, q3, q4);
```

- 各列毎に Btree インデックス
  - Bitmat Index Scan で AND/OR 処理が可能

```
CREATE INDEX ON t_survey (q1);
CREATE INDEX ON t_survey (q2);
   :
```

- 複数列 Bloom インデックス
  - 優れた特性 ／ 対応データ型が限定 ／ 実用実績不足

ソートにおけるインデックス

- Btree インデックスはソートにも使われる
  - インデックス作成時の細かなソート指定

```
SELECT * FROM t_sort ORDER BY c1, c2 DESC, c3 NULLS FIRST;

CREATE INDEX ON t_sort (c1, c2, c3);
CREATE INDEX ON t_sort (c1, c2 DESC, c3 NULLS FIRST);
```

– インクリメンタルソート対応 (v13) 以降は、一部適合でもそれなりの性能

PostgreSQL
http://www.postgresql.jp

式インデックス

- 検索条件の左辺の式をインデックスにする

```
SELECT * FROM t_email WHERE lower(addr) = 'foo@example.com';
CREATE INDEX ON t_email (lower(addr));
```

全文検索の基本的な使用法

```
SELECT * FROM t_docs WHERE
 to_tsvector('english', body) @@ to_tsquery('english', 'JPUG');
CREATE INDEX ON t_docs
  USING GIN (to_tsvector('english', body));
```

加工された値に対するユニーク制約 - 出勤簿テーブルに同じ人の出勤は１日１回

```
CREATE TABLE t_attend (uid int, ts timestamp);
CREATE UNIQUE INDEX idx_t_attend ON t_attend (uid, (ts::date));
```

式インデックスの統計情報

- 式インデックスは式に対するプランナ統計情報を作る

```
db1=# CREATE INDEX idx_t_ab ON t_ab ((a + b));
db1=# ANALYZE t_ab;
db1=# SELECT * FROM pg_stats WHERE tablename = 'idx_t_ab';
-[ RECORD 1 ]----------+-----------------------------------
schemaname             | public
tablename              | idx_t_ab
attname                | expr
inherited              | f
null_frac              | 0
avg_width              | 4
n_distinct             | 21
most_common_vals       | {10,9,8,11,12,13,5,6,7,15,14,4,16,3,17,18,
most_common_freqs      | {0.106,0.096,0.088,0.088,0.085,0.071,0.064
histogram_bounds       |
correlation            | 0.052369732
most_common_elems      |
most_common_elem_freqs |
elem_count_histogram   |
```

部分インデックス

- 一部の行だけを対象にしたインデックス
  - 頻出データの除外
  - 使われ方に対応したサイズ節約

Btree 自体が重複排除できる
ようになって必要性低下

```
CREATE INDEX idx_access_log_client_ip
  ON t_access_log (client_ip)
  WHERE NOT (client_ip = '127.0.0.1'::inet);

SELECT * FROM t_access_log WHERE client_ip = '127.0.0.1';
```

- 条件付きユニーク制約

1 件だけ NULL を許す制約

```
CREATE UNIQUE INDEX tests_target_one_null
  ON tests ((target IS NULL)) WHERE target IS NULL;
```

カバリング

- Index Only Scan

btree と gist で対応

  - インデックスだけを見て答えを出せる場合

```
CREATE TABLE t_only (id int PRIMARY KEY, name text);
SELECT id FROM t_only WHERE id < 10;
```

    – VACUUM 後に未更新のページに限られる

- INCLUDE 指定
  - リーフノードにだけ列データを含める

```
CREATE INDEX ON t_only (id) INCLUDE (name);
SELECT id, name FROM t_only WHERE id < 10;
```

全文検索インデックス

- 組込で備わっている全文検索手段
  - テキスト検索
    - tsvector ／ tsquery 型、 GIN/GiST
  - pg_trgm 拡張モジュール
    - 3-gram 、マルチバイト非対応、 LIKE や ~ で、 GIN/GiST
- サードパーティモジュール
  - Pgroonga　　詳しくは PGroonga のセッションで！
  - pg_bigm　　2-gram 、 LIKE で、 GIN
  - textsearch_ja　　tsvector/tsquery を日本語対応

メンテナンス状況は悪いが、最新版 PostSQL でも概ね動作

全文検索における GIN と GiST

- GIN

サイズ：大きい
検索：速い
更新：遅い

  - テキスト検索：

キー、位置、共に木構造で格納

単語	文書列を持つ行
query	1, 5, 10, 25, 31, 51, 53
index	1, 2, 9, 23, 31, 81, 93
text	2, 5, 10, 75

  - 2-gram 、 3-gram:

文字列片	文書列を持つ行
Po	1, 2, 4, 7, 10, 11, 15, 16,
os	1, 2, 3, 9, 12, 15, 19, 21,
st	1, 5, 6, 9 , 10, 11, 17, 18,
tg	1, 3, 4, 6, 8, 13, 14, 17,
gr	1, 2, 3, 5, 9, 10, 11, 16,

- GiST

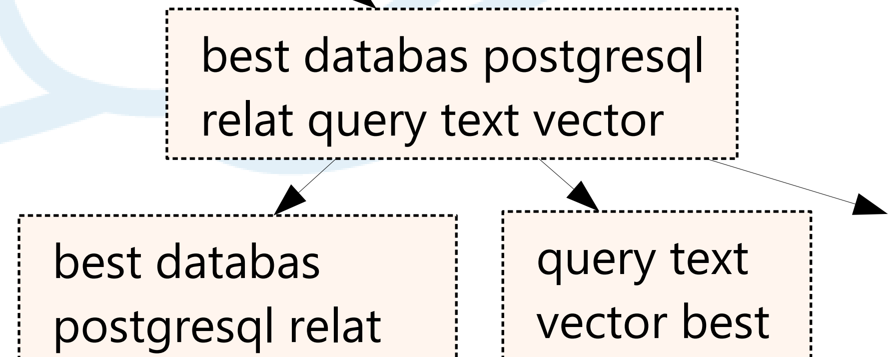
サイズ：小さい
検索：遅い
更新：速い

  - ts_vector を検索

PostgreSQL is the best relational database
↓ to_tsvector()
'best':4 'databas':6 'postgresql':1 'relat':5

  - RD-tree
    - 集合の木構造

実際は符号化して格納



PostgreSQL
http://www.postgresql.jp

プランナがインデックスを使う

- プランナがインデックスを参照することがある
  - 最大値、最小値を把握するため
  - インデックスはプラン作成自体にも役立っている
  - プランナ遅延の原因がインデックスにあるかも

インデックスの状態確認

- pgstattuple 拡張モジュール
  - btree, gin, hash に対応

```
db1=# SELECT * FROM pgstatindex('idx_t1_btree');
-[ RECORD 1 ]------+---------
version            | 4
tree_level         | 2
index_size         | 11665408
root_block_no      | 161
internal_pages     | 5
leaf_pages         | 468
empty_pages        | 0
deleted_pages      | 950
avg_leaf_density   | 68.84
leaf_fragmentation | 50.21
```

tree_level は リーフノードを含まない階層数

リーフの密度。
ページごと格納できる値の範囲があるので、ある程度空きが生じる

論理的な並び順と物理的な並び順の不一致度合いを 0 ～ 100 で表現

gin インデックスの状態情報

```
db=# SELECT * FROM pgstatginindex('idx_t1_gin');
-[ RECORD 1 ]--+--
version        | 2
pending_pages  | 1
pending_tuples | 2
```

ストレージオプション fastupdate = on　(デフォルト)のとき、挿入直後はデータを pending list に入れて、後からインデックス内に格納する。

hash インデックスの状態情報

```
db=# SELECT * FROM pgstathashindex('idx_t1_hash');
-[ RECORD 1 ]--+------------------
version        | 4
bucket_pages   | 326
overflow_pages | 0
bitmap_pages   | 1
unused_pages   | 256
live_items     | 50002
dead_items     | 0
free_percent   | 78.9219604153881
```

overflow_pages が多いのは望ましくない状態。同値が多い場合などで発生。

Btree と比べて構造上、空きが多く出る

VACUUM とインデックス

- テーブル VACUUM でインデックスも VACUUM
  - 空ページ再利用は行われるが、
    ファイルサイズ縮小は行われない
  - hash は性質上ページ再利用が報告されない
  - 「テーブルを VACUUM 」が最小操作単位
    - v13　で並列実行に対応
  - GIN で、ペンディングリストを本体に適用する
  - BRIN で、要約を作り直す

PostgreSQL
http://www.postgresql.jp

インデックス再構築

- データ削減後のディスクスペース解放
- 乱れたデータ構造を整える　（本当に必要か？）
- 破損インデックスを修復　（ e.g. WAL 非対応の場合）
- REINDEX コマンド
  - DROP INDEX / CREATE INDEX　コマンドは…
- CONCURRENTLY　オプションで同時作成
  - Access Exclusive ロックなし
  - 長時間、失敗するかもしれない
  - 失敗したら手動で残骸インデックスを DROP

pg_repack を使う方法も有力、テーブル空間移動や振る舞いの多様な指定が可能。

インデックス保守操作関数

- BRIN
  - 要約する　brin_summarize_new_values(idx)、
    brin_summarize_range(idx,blk)、
  - 要約を戻す brin_desummarize_range(idx,blk)

ストレージオプション autosummarize = on （デフォルト off ）でないと
新たなブロックの行に対して、新たな要約は自動では作られない。

- GIN
  - ペンディングリストを（本体適用して）一掃する
    gin_clean_pending_list(idx)

VACUUM 時やリスト上限に達したときにも
行われるので、明示実行は必須ではない。

PostgreSQL
http://www.postgresql.jp

インデックス破損と検査

- インデックス破損
  - バグ、基盤側の障害、二重起動による破壊
- インデックスが破損すると
  - 問い合わせ結果が時々狂う（実行プラン依存）
  - 主キーが重複した列が発生
- 破損の検査
  - ページチェックサム　　データベースクラスタに指定
  - amcheck　　　btree 専用の検査ツール
  - pageinspect　インデックス内容を参照するツール

PostgreSQL
http://www.postgresql.jp

ご清聴ありがとうございました

- 参考文献 ／ 更なる情報：
  - PostgreSQL ドキュメント
    - 例が足されていたり、最新版を読み直すと発見がある
  - wiki.postgresql.org
  - Lets.postgresql.jp
    - 古い記事が多いが現在でも役立つ内容も
  - habr.com/en/company/postgrespro/blog/
    - インデックスに貢献している　PostgresPro 社の記事