PostgreSQL インサイド

pg_bigm で全文検索する
技術を知る

FUJITSU

☐ 導入／環境設定 ☐ 移行 ☑ 性能 ☐ チューニング ☐ バックアップ／リカバリー
☐ 冗長化／負荷分散 ☐ 監視 ☐ データ連携 ☐ 災害対策 ☐ 豆知識

PostgreSQL の周辺ツールの 1 つである pg_bigm を使って、全文検索する方法について解説します。

1. pg_bigm とは

pg_bigm（ピージーバイグラム）は、SQL 文で全文検索を使用できるツールです。このツールは、2-gram（バイグラム）と呼ばれる方法で、文字列から全文検索用のインデックスを作成します。このインデックスを使って、ユーザーは高速に文字列検索を実行できます。
PostgreSQL には、pg_trgm（ピージートライグラム）という全文検索モジュールが contrib に付属されていますが、pg_trgm は日本語のようなマルチバイト文字列の検索には適していません。そこで、マルチバイト文字列を扱う全文検索用の周辺オープンソースソフトウェアとして、pg_trgm をベースに開発されたのが pg_bigm です。pg_trgm と pg_bigm の違いを簡単にまとめます。

表 1. pg_trmg と pg_bigm の違い

機能や特長	pg_trgm	pg_bigm
全文検索用のインデックスの作成方法	3-gram	2-gram
利用できるインデックスの種類	GIN と GiST	GIN
利用できるテキスト検索演算子	LIKE（～～）、ILIKE（～～*）、～、～*	LIKE
日本語検索	未対応	対応済

全文検索用のインデックスとは

全文検索とは、複数の検索対象の全文書から特定の文字列を検索することです。PostgreSQL で全文検索を高速化するためには、GiST インデックスと GIN インデックスが使用できますが、pg_bigm と組み合わせる場合は GIN インデックスにのみ対応します。GIN インデックスは文書中の単語の位置を保持しているため、特定の単語の検索を効率的に行えるようになります。GIN インデックスの詳細については、「PostgreSQL 文書」を参照してください。

2-gram とは

全文検索技術の 1 つに「N-gram」という方法があります。これは任意の文字列や文章を連続した N 個の文字単位または単語単位で分割するテキスト分割方法です。N が 1 の場合を 1-gram（ユニグラム）、 N が 2 の場合を 2-gram（バイグラム）、 N が 3 の場合を 3-gram（トライグラム）と呼びます。
「これは本です。」という例文を N-gram に適用して文字単位に分割すると、以下になります。

種別	1	2	3	4	5	6	7
1-gram	こ	れ	は	本	で	す	。
2-gram	これ	れは	は本	本で	です	す。	
3-gram	これは	れは本	は本で	本です	です。		

pg_bigm は 2-gram を採用しているため、文字列中の連続する 2 文字ごとに全文検索用のインデックスを作成します。

参考

- pg_bigm の仕様については、オープンソースソフトウェアに同梱されているマニュアルを参照してください。
- pg_bigm は便利な機能ですが、注意点もあります。ご利用の前には必ず「4. pg_bigm 利用時の注意点」をお読みください。

2. 準備

pg_bigm を利用するには、以下の準備が必要です。なお、pg_bigm は Linux と Mac OS 上で動作します。今回は、Linux 上で pg_bigm 1.2 をベースに説明します。
PostgreSQL および pg_bigm のインストールは完了し、インスタンスのエンコーディングは UTF8 とします。

1. postgresql.conf ファイルの shared_preload_libraries に「pg_bigm」を追加します。

```
shared_preload_libraries = 'pg_bigm'
```

2. PostgreSQL を起動または再起動します。
3. 本機能を利用するデータベース postgres に対して、CREATE EXTENSION を実施します。

```
$ psql -d postgres -c "CREATE EXTENSION pg_bigm;"
```

3. pg_bigm の使い方

実際に pg_bigm を使って、文字列を検索する方法について説明します。
使用したデータは、PostgreSQL インサイドの「PostgreSQL 技術インデックス」に掲載されている各記事のタイトルと記事本文を抽出したものです。

3.1 GIN インデックスの作成

データベース postgres 上にテーブルを作成し、GIN インデックスを作成します。

1. 検索対象のテーブル pg_tbl を作成してデータを格納し、全文検索用のインデックス pg_idx を作成します。
   （1）インデックスメソッドには"gin"を指定します。
   （2）演算子クラスには"gin_bigm_ops"を指定します。


```
postgres=# CREATE TABLE pg_tbl (title text, article text);
CREATE TABLE
postgres=# ¥COPY pg_tbl FROM data.csv WITH CSV;
COPY 33
postgres=# CREATE INDEX pg_idx ON pg_tbl USING gin (article gin_bigm_ops);
CREATE INDEX
                                                  (1)          (2)
```

2. pg_tbl テーブルの構成を確認します。

```
postgres=# ¥d pg_tbl
                テーブル"public.pg_tbl"
   列     | タイプ | 照合順序 | Null 値を許容 | デフォルト
----------+--------+----------+---------------+------------
 title    | text   |          |               |            ← タイトルを格納
 article  | text   |          |               |            ← 記事本文を格納
インデックス:
    "pg_idx" gin (article gin_bigm_ops)   ← GINインデックス
```

3.2 全文検索の実行

pg_bigm では、LIKE 演算子の中間一致検索を使って全文検索ができます。

例 1）複数のキーワードを検索する

記事本文に「レプリケーション」および「トランザクションログ」という単語を含む、PostgreSQL 技術インデックスのタイトルを検索します。

```
postgres=# SELECT title FROM pg_tbl WHERE article LIKE  '%レプリケーション%'  and
article LIKE  '%トランザクションログ%';
                          title
-----------------------------------------------------------
 pg_statsinfoで統計情報を収集・蓄積する
 Pgpool-IIで高可用なシステム構成を考える
 PostgreSQLのアーキテクチャー概要
 ストリーミングレプリケーション ～ 仕組み、構成のポイント ～
 データベースシステムの監視 ～監視方法と監視例～
 パフォーマンスチューニング9つの技 ～「書き」について～
(6 行)
```

例 2）英字を含む文字列を検索する

記事本文に「Index Only Scan」という単語を含む、PostgreSQL 技術インデックスのタイトルを検索します。


```
postgres=# SELECT title FROM pg_tbl WHERE article LIKE '%Index Only Scan%';
                          title
-----------------------------------------------------------
 pg_hint_planで実行計画を制御する
 PostgreSQLのアーキテクチャー概要
 SQLチューニングの概要
 パフォーマンスチューニング9つの技 ～「基盤」について～
 パフォーマンスチューニング9つの技 ～「書き」について～
 パフォーマンスチューニング9つの技 ～「探し」について～
 パフォーマンスチューニング9つの技 ～はじめに～
(7 行)
```

ヒント

検索キーワードに英字を指定する場合、大文字と小文字は区別する必要があります。大文字と小文字を区別せずに検索する場合は、文字列関数の lower 関数や upper 関数を使うことで対応できます。

```
postgres=# SELECT title FROM pg_tbl WHERE lower(article) LIKE '%index only scan%';
                          title
-----------------------------------------------------------
 パフォーマンスチューニング9つの技 ～「基盤」について～
 パフォーマンスチューニング9つの技 ～「探し」について～
 パフォーマンスチューニング9つの技 ～「書き」について～
 パフォーマンスチューニング9つの技 ～はじめに～
 PostgreSQLのアーキテクチャー概要
 SQLチューニングの概要
 pg_hint_planで実行計画を制御する
(7 行)
```

pg_bigm を使いこなすためには、以下の関数が用意されています。本記事では、bigm_similarity 関数の使用例を紹介します。

関数名	機能の説明
bigm_similarity(引数 1,引数 2)	文字列（引数 1）と文字列（引数 2）の類似度（文字列がどの程度似ているかを示す数値）を返却する関数
likequery(引数 1)	全文検索できるように、検索文字列（引数 1）を LIKE 演算子のパターンに変換する関数
pg_gin_pending_stats(引数 1)	GIN インデックス（引数 1）の待機リストに含まれているデータのページ数とタプル数を返却する関数
show_bigm(引数 1)	文字列（引数 1）のすべての 2-gram 文字列を配列として表示する関数

3.3 類似度検索の実行

pg_bigm では、=%演算子を使って類似度検索もできます。類似度検索では、検索条件の文字列との類似度が閾値以上の行が検索結果となり、検索キーワードに指定した文字列と似ている文字列を検索できます。類似度は、postgresql.conf ファイルに pg_bigm.similarity_limit パラメーターを追加指定します。設定値は 0 以上 1 以下の小数点で、デフォルトは 0.3 です。1 に近いほど類似度が高いことを示します。

1. pg_bigm.similarity_limit の値を確認します。タイトルに「pg_statinfo」と類似の単語を持つ、PostgreSQL 技術インデックスのタイトルを検索します。

```
postgres=# SHOW pg_bigm.similarity_limit ;  ← 設定値を確認
 pg_bigm.similarity_limit
--------------------------
 0.3      ← 値は0.3（デフォルト値）
(1 行)

postgres=# SELECT title FROM pg_tbl WHERE title =% 'pg_statinfo' ;
                title
--------------------------------------
 pg_statsinfoで統計情報を収集・蓄積する
 pg_statsinfoのサイジング
(2 行)
```

2. pg_bigm.similarity_limit の類似度を「0.15」に変更して、同様に検索します。
類似度を 0.3 から 0.15 に下げることで、「pg_statinfo」に近いタイトルが 2 件追加されました。

```
postgres=# SET pg_bigm.similarity_limit TO 0.15;  ← 0.15に変更
SET
postgres=# SELECT title FROM pg_tbl WHERE title =% 'pg_statinfo' ;
                        title
------------------------------------------------------
 pg_dbms_statsで統計情報を固定化して、実行計画を制御する   ← 追加
 pg_hint_planで実行計画を制御する                       ← 追加
 pg_statsinfoで統計情報を収集・蓄積する
 pg_statsinfoのサイジング
(4 行)
```

3. 「pg_statinfo」と「pg_statsinfo」の類似度、および「pg_statinfo」と「pg_dbms_stats」の類似度を bigm_similarity 関数で計算してみましょう。

```
postgres=# SELECT bigm_similarity('pg_statinfo', 'pg_statsinfo');
 bigm_similarity
-----------------
      0.84615386     ← 類似度が高い
(1 行)

postgres=# SELECT bigm_similarity('pg_statinfo', 'pg_dbms_stats');
 bigm_similarity
-----------------
             0.5     ← 先ほどよりも類似度は下がる
(1 行)
```

検索キーワードに類似した文字列に絞りたい場合は類似度を高くし、類似した文字列をより多く検出したい場合は類似度を低くするなど調整してください。類似度の計算方法などについては、オープンソースソフトウェアに同梱されているマニュアルの bigm_similarity 関数を参照してください。

3.4 大量データによる全文検索の性能検証

約 17 万件の日本語版 Wikipedia の要約データを使って、全文検索の性能を検証します。

1. 検索対象のテーブル wiki_tbl を作成してデータを格納します。全文検索用のインデックスを作成せずに、要約文中に「平安時代後期から鎌倉時代前期にかけての公卿」という文言を含む、Wikipedia のタイトルを検索します。
   （1）検索処理の実行時間は、約 1.1 秒です。

```
postgres=# CREATE TABLE wiki_tbl (id serial PRIMARY KEY, title text, abstract text);
CREATE TABLE
postgres=# ¥COPY wiki_tbl(title, abstract) FROM jawiki-latest-abstract5.csv WITH DELIMITER ',' ;
COPY 174778
postgres=# ¥timing     ← SQL文の実行時間の表示をonに設定
タイミングは on です。
postgres=# SELECT id,title FROM wiki_tbl WHERE abstract LIKE '%平安時代後期から鎌倉時代前期にかけての公卿%';
  id   |        title
-------+---------------------
  4021 | Wikipedia: 平親国
  5769 | Wikipedia: 源有雅
  7736 | Wikipedia: 藤原公清
 18436 | Wikipedia: 平親範
 28602 | Wikipedia: 藤原資頼
(5 行)

時間: 1142.971 ミリ秒(00:01.143) (1)
```

2. SELECT 文の実行計画を確認します。
   （2）全文検索用のインデックスが無いため、テーブル wiki_tbl に対して検索方式「Parallel Seq Scan」が選択されています。

```
postgres=# EXPLAIN SELECT id,title FROM wiki_tbl WHERE abstract LIKE  '%平安時代後期から鎌倉時代前期にかけての公卿%';
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 Gather  (cost=1000.00..5790.81 rows=15 width=35)
   Workers Planned: 2
   ->  Parallel Seq Scan on wiki_tbl  (cost=0.00..4789.31 rows=6 width=35)
         Filter: (abstract ~~ '%平安時代後期から鎌倉時代前期にかけての公卿%'::text)
(4 行))
                    (2)
```

3. 検索対象のテーブルに、全文検索用のインデックス wiki_idx を作成し、手順 1.と同じ SQL 文で検索します。
   （3）全文検索用のインデックスの作成時間です。約 7 秒かかっています。
   （4）検索処理の実行時間は、約 1.3 ミリ秒です。全文検索用のインデックスがない（1）と比較して、約 856 倍速くなっています。


```
postgres=# CREATE INDEX wiki_idx ON wiki_tbl USING gin (abstract gin_bigm_ops);
CREATE INDEX
時間: 7591.573 ミリ秒(00:07.592) (3)

postgres=# SELECT id,title FROM wiki_tbl WHERE abstract LIKE '%平安時代後期から鎌倉時代前期にかけての公卿%';
  id   |        title
-------+---------------------
  4021 | Wikipedia: 平親国
  5769 | Wikipedia: 源有雅
  7736 | Wikipedia: 藤原公清
 18436 | Wikipedia: 平親範
 28602 | Wikipedia: 藤原資頼
(5 行)

時間: 1.336 ミリ秒 (4)
```

4. SELECT 文の実行計画を確認します。
   （5）全文検索用のインデックス wiki_idx を利用した検索方式「Bitmap Index Scan」が選択され、高速に処理できていることが確認できます。

```
postgres=# EXPLAIN SELECT id,title FROM wiki_tbl WHERE abstract LIKE '%平安時代後期から鎌倉時代前期にかけての公卿%';
                                    QUERY PLAN
------------------------------------------------------------------------------------------
 Bitmap Heap Scan on wiki_tbl  (cost=232.11..289.50 rows=15 width=35)
   Recheck Cond: (abstract ~~ '%平安時代後期から鎌倉時代前期にかけての公卿%'::text)
   ->  Bitmap Index Scan on wiki_idx  (cost=0.00..232.11 rows=15 width=0)
         Index Cond: (abstract ~~ '%平安時代後期から鎌倉時代前期にかけての公卿%'::text)
(4 行)          (5)
```

4. pg_bigm 利用時の注意点

実際に pg_bigm を利用する際の注意点について紹介します。

大文字と小文字を同一視しない

「3.2 全文検索の実行」の「ヒント」に示したように pg_bigm は英字の大文字と小文字を同一視せずに、区別することに注意してください。
「INDEX」と「index」の類似度を bigm_similarity 関数で確認すると、以下になります。

```
postgres=# SELECT bigm_similarity('INDEX', 'index');
 bigm_similarity
-----------------
               0    ← 類似度が0
(1 行)
```

列の最大サイズ

pg_bigm のインデックス対象文字列にはサイズ上限があります。107,374,180 Bytes（約 102MB）を超える文字列には適用できません。

チューニングパラメーターの設定

pg_bigm の動作をチューニングできるパラメーターのうち、正常な動作をさせるために「pg_bigm.enable_recheck」は、デフォルト値（on）のまま利用してください。
pg_bigm を使った全文検索では、内部的には以下 2 つの処理で検索結果が取得されます。

- 全文検索用のインデックスからの検索結果候補の取得
- 検索結果候補からの正しい検索結果の選択

この後者の処理が Recheck と呼ばれます。全文検索用のインデックスからの検索結果の取得では、必ずしも正しい結果だけが得られるとは限りません。誤った結果が含まれる可能性があります。この誤った結果を取り除くのが Recheck になります。このパラメーターは、正しい検索結果を得る必要がある運用時には必ず on に設定されていなければなりません。ただし、Recheck のオーバーヘッドを評価するなど、デバッグ時に off に設定しても構いません。

ここまで説明してきましたが、pg_bigm は比較的簡単な手順で利用できることがおわかりいただけたと思います。注意事項に気を付けて、文字列検索処理の業務要件に合わせてご利用をご検討ください。

参考

FUJITSU Software Enterprise Postgres では、バージョン 10 から pg_bigm を同梱しています。製品のインストール時に pg_bigm をプリインストールしているので、追加モジュールを取得する必要はありません。

2021 年 9 月 24 日