PostgreSQL インサイド

pg_dbms_stats で統計情報を固定化して実行計画を制御する

技術を知る

- [ ] 導入／環境設定
- [ ] 移行
- [ ] 性能
- [x] 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

統計情報を固定化して実行計画を制御するチューニング方法について解説します。SQL が実行される仕組みとプランナ（オプティマイザ）については「SQL チューニングの概要」を参照してください。

1. 統計情報を固定化する

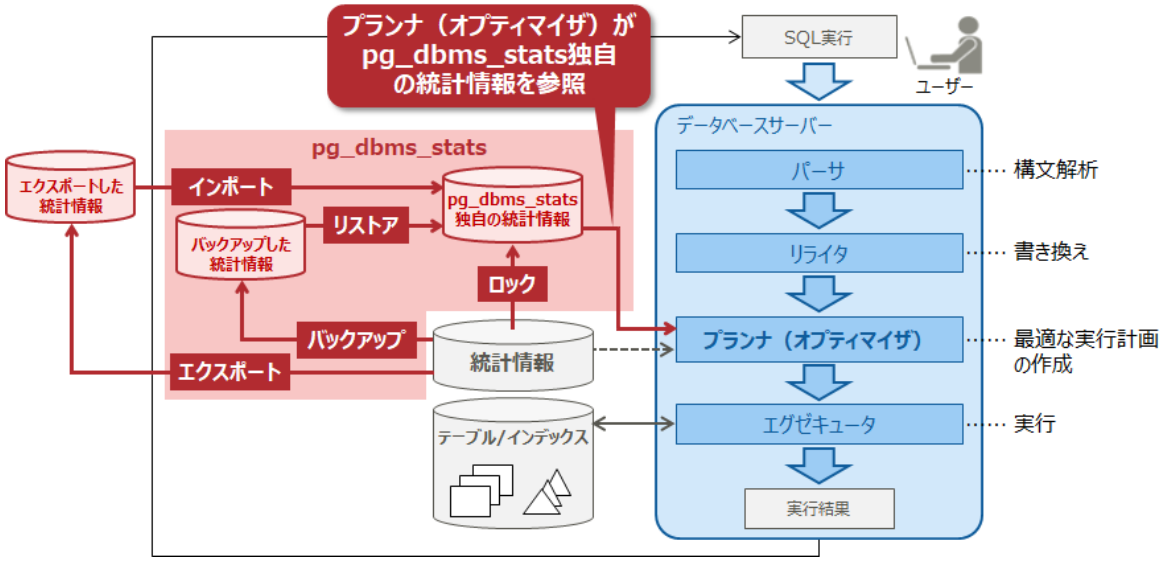
PostgreSQL では、プランナ（オプティマイザ）が入力された SQL 文のクエリをもとに統計情報を参照して、最も速くてコストが低いと予測される方法を選択し、実行計画を作成します。しかし、必ずしもプランナ（オプティマイザ）が最適な実行計画を作成するとは限りません。例えば、大量の更新クエリなどで統計情報の最新化が間にあわない場合や、刻々と統計情報が変化する場合に、プランナ（オプティマイザ）が作成する実行計画が不安定になることがあります。その結果、レスポンスが平準化されなかったり、一時的なスループットの遅延が発生したりします。
その対策の 1 つとして、pg_dbms_stats を利用して統計情報を固定化し、常にその統計情報が利用されるようにチューニングする方法があります。

pg_dbms_stats とは

pg_dbms_stats は、統計情報を管理し、間接的に実行計画を制御するツールで、Linux 上、Windows 上および Solaris 上で動作します。PostgreSQL のプランナ（オプティマイザ）が不適切な実行計画を選択した場合の対処として、ユーザーが pg_dbms_stats を利用して統計情報を固定化できます。pg_dbms_stats には以下の機能があります。

バックアップ	現在の統計情報をバックアップする。
リストア	過去にバックアップした統計情報を復元して固定する。
パージ	不要になったバックアップを一括削除する。
ロック	現在選択されている実行計画が選択され続けるように、統計情報を固定する。
ロック解除	統計情報の固定を解除する（オブジェクト単位）。
クリーンアップ	統計情報の固定を解除する（使用しない統計情報の一括削除）。
エクスポート	現在の統計情報を外部ファイルに出力する（バイナリー形式）。
インポート	エクスポート機能で作成した外部ファイルから統計情報を読み込み、統計情報を固定する。

以下に pg_dbms_stats の仕組みを図解します。



参考

- pg_dbms_stats の仕様については、オープンソース・ソフトウェアの Web ページを参照してください。
- pg_dbms_stats は便利な機能ですが、注意点もあります。お使いの前には必ず“3. pg_dbms_stats の注意点”をお読みください。

2. pg_dbms_stats を使う

pg_dbms_stats を使い、統計情報を固定化する方法について順を追って、見ていきましょう。検証には以下のようなシステムと業務を想定しています。

- 急なレスポンス低下が許されないシステムのため、VACUUM と ANALYZE の実行は自動バキュームではなく管理者が制御する。
- 対象の業務テーブルは頻繁にアクセスされ、データ量の変動が大きく、ANALYZE による統計情報の更新が間にあわないことが多い。そこで、統計情報を固定化してレスポンスを安定化させる。

なお、今回は PostgreSQL 11.1 と pg_dbms_stats 1.3.11 を組み合わせた環境での検証例とします。

2.1 準備する

pg_dbms_stats を利用するには、追加モジュールを公開サイトなどから取得してインストールしたあと、以下の準備が必要です。

1. PostgreSQL を起動して本機能を利用するデータベースに対して、CREATE EXTENSION を実施します。なお、対象のデータベースは「mydb」とします。

```
$ psql -d mydb -c "CREATE EXTENSION pg_dbms_stats;"
```

2. postgresql.conf ファイルの shared_preload_libraries パラメーターに「pg_dbms_stats」を追加します。

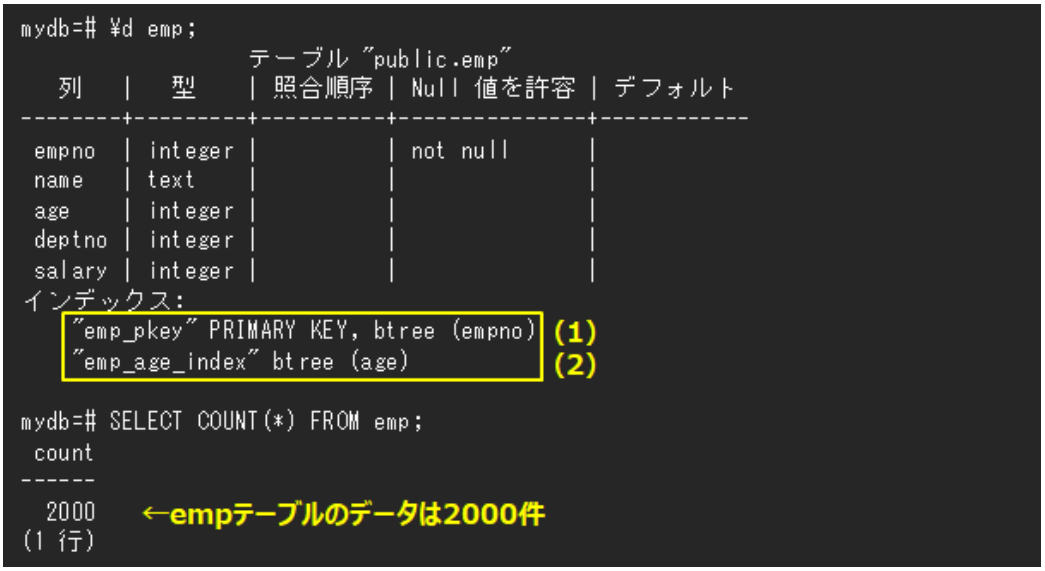
```
shared_preload_libraries = 'pg_dbms_stats'
```


3. PostgreSQL を再起動します。

2.2 統計情報をバックアップする

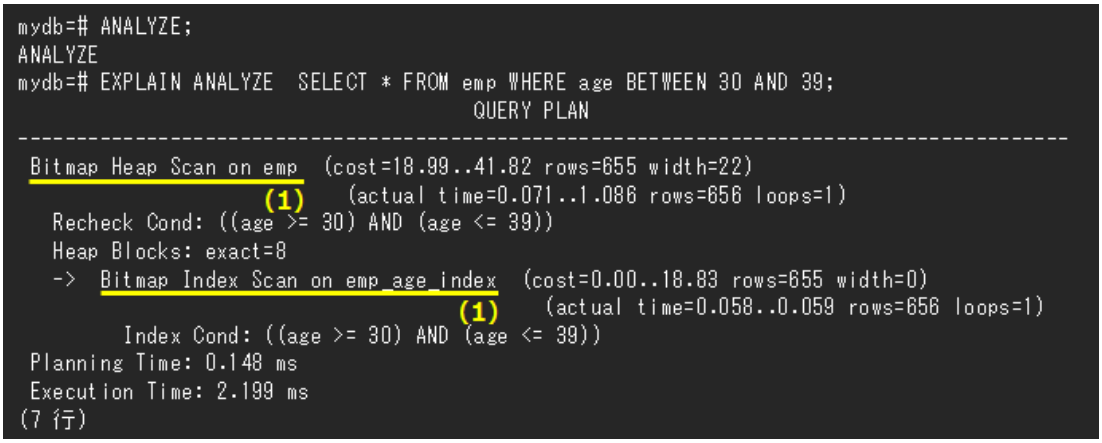
インデックスのあるテーブルの検索処理に対して、速く安定したレスポンスが得られていたときの統計情報をバックアップします。なお、emp テーブルは作成されているものとします。

1. emp テーブルの構成とデータ件数を確認します。



   - （1）emp テーブルの列「empno」に「emp_pkey」というインデックス（主キー）が設定されています。
   - （2）emp テーブルの列「age」に「emp_age_index」というインデックスが設定されています。

2. ANALYZE コマンド（SQL コマンド）で統計情報を更新し、対象 SQL の実行計画を確認します。



   - （1）emp テーブルに対して「Bitmap Scan」が選択されたことがわかります。
     - 注意　バックアップの前に、必ず一度は ANALYZE コマンド（SQL コマンド）で統計情報を取得してください。統計情報が存在しない状態でバックアップやロックを実行した場合、実行計画は制御できません。

3. 検証した結果、実行計画に「Bitmap Scan」が選択されたときのレスポンスが速く安定していたので、この統計情報をバックアップします。ここではデータベース単位にバックアップしたいので、dbms_stats.backup_database_stats()関数を指定し、パラメーターにはコメントとして「Bitmap Scan for emp」を指定します。


```
mydb=# SELECT dbms_stats.backup_database_stats('Bitmap Scan for emp');
 backup_database_stats
-----------------------
                     1
(1 行)
```

4. 現在保存されているバックアップ情報は、dbms_stats.backup_history テーブルで参照できます。dbms_stats.backup_history テーブルは pg_dbms_stats の導入時に生成されるテーブルで、バックアップ ID やバックアップ時のタイムスタンプなどの履歴を管理しています。統計情報のバックアップ一覧を表示し、バックアップ ID（列名は id）を確認します。

```
mydb=# SELECT * FROM dbms_stats.backup_history;
 id |              time               | unit |          comment
----+---------------------------------+------+---------------------------
(1) 1 | 2019-07-08 19:24:11.48549+09  | d    | Bitmap Scan for emp
  (1 行)                                (2)
```

   - （1）バックアップ ID は「1」です。バックアップの取得時間順に 1 から採番されます。
   - （2）「d」はバックアップ単位がデータベースであることを意味します。

ポイント

バックアップは、以下のオブジェクト単位で指定することができます。

- dbms_stats.backup_database_stats()
- dbms_stats.backup_schema_stats()
- dbms_stats.backup_table_stats()
- dbms_stats.backup_column_stats()

dbms_stats_backup_history テーブルで情報を一覧表示したとき、「unit」には各オブジェクトの頭文字が表示されます。

```
mydb=#  SELECT * FROM dbms_stats.backup_history;
 id |              time               | unit |     comment
----+---------------------------------+------+-----------------------------
  1 | 2019-07-08 19:24:11.48549+09    | d    | Bitmap Scan for emp
  2 | 2019-07-09 13:23:34.274064+09   | s    | Index Scan_sc
  3 | 2019-07-10 15:25:32.942122+09   | t    | Index Scan_tbl
  4 | 2019-07-11 17:27:47.453239+09   | c    | Index Scan_aid
(4 行)
```

2.3 リストア機能を使って統計情報を固定化する

バックアップした統計情報を、リストア機能を使って固定化します。

1. 初めに、実行計画が不安定になる状況を作ってみます。
   emp テーブルのデータから 500 件削除します。ANALYZE コマンド（SQL コマンド）で統計情報を更新すると、emp テーブルに対して「Seq Scan」が選択されます。続けて emp テーブルのデータに 2000 件追加します。統計情報を更新せずに対象 SQL の実行計画を確認します。ここではデータを大量追加した直後にデータ検索要求があり、ANALYZE コマンド（SQL コマンド）による統計情報の更新が間にあわなかったことを想定しています。


```
mydb=# EXPLAIN ANALYZE  SELECT * FROM emp WHERE age BETWEEN 30 AND 39;
                                 QUERY PLAN
-------------------------------------(2)a --------------------------------
 Seq Scan on emp  (cost=0.00..62.81 rows=864 width=22)
             (1)    (actual time=0.020..4.042 rows=1148 loops=1)
   Filter: ((age >= 30) AND (age <= 39))          (2)b
   Rows Removed by Filter: 2352
 Planning Time: 0.334 ms
 Execution Time: 6.962 ms
(5 行)
```

- （1）emp テーブルに対して「Seq Scan」が選択されたことがわかります。
- （2）統計情報が更新されていないため、emp テーブルに対する推測値の a と実測値の b は大きく異なります。

2. 「2.2 統計情報をバックアップする」でバックアップした「Bitmap Scan」の統計情報が選択されるように、リストア機能で復元して固定します。
   バックアップした統計情報（Bitmap Scan）を、バックアップ ID を指定してリストアするには、dbms_stats.restore_stats() 関数を指定します。パラメーターには backup_history テーブルの id「1」を指定します。

```
mydb=# SELECT dbms_stats.restore_stats(1);
     restore_stats
-----------------------
 emp
 emp_age_index
 emp_pkey
(3行)
```

ポイント

リストア機能には、以下の 2 種類の方法があります。用途に応じて使い分けてください。

- **バックアップ ID を指定する方法**
  - dbms_stats.restore_stats()
- **タイムスタンプを指定する方法**
  - dbms_stats.restore_database_stats()
  - dbms_stats.restore_schema_stats()
  - dbms_stats.restore_table_stats()
  - dbms_stats.restore_column_stats()

3. 対象 SQL の実行計画を確認します。

```
mydb=# EXPLAIN ANALYZE  SELECT * FROM emp WHERE age BETWEEN 30 AND 39;
                                 QUERY PLAN
-------------------------------------------------(2)a ----------------------
 Bitmap Heap Scan on emp  (cost=18.99..41.82 rows=655 width=22)
                    (1)   (actual time=0.115..1.970 rows=1148 loops=1)
   Recheck Cond: ((age >= 30) AND (age <= 39))
   Heap Blocks: exact=17                                  (2)b
   ->  Bitmap Index Scan on emp_age_index  (cost=0.00..18.83 rows=655 width=0)
      (1)                                 (actual time=0.094..0.096 rows=1148 loops=1)
         Index Cond: ((age >= 30) AND (age <= 39))                    (2)a
 Planning Time: 0.188 ms                                              (2)b
 Execution Time: 3.776 ms
                 (3)
(7 行)
```

- （1）emp テーブルに対して「Bitmap Scan」が選択されたことがわかります。
- （2）emp テーブルに対する推測値の a と実測値の b は大きく異なります。a は「2.2 統計情報をバックアップする」の手順 2.で「Bitmap Scan」を採用したときに参照した統計情報を元に推定された値のためです。なお、ここで ANALYZE コマンド（SQL コマンド）を実行しても a は更新されません。
- （3）手順 1.と比べて、実行時間が半分近く短縮されたことがわかります。

このように、統計情報を固定化することで、実行計画の変化を避けて安定したレスポンスを実現できます。

2.4 統計情報の固定化を解除する

pg_dbms_stats 独自の統計情報を、PostgreSQL 本来の統計情報に戻す場合は、ロック解除機能を使います。

1. 固定を解除するには、dbms_stats.unlock_database_stats()関数を指定します。

```
mydb=# SELECT dbms_stats.unlock_database_stats();
 unlock_database_stats
-----------------------
 emp
 emp_pkey
 emp_age_index
(3 行)
```

2. VACUUM ANALYZE コマンド（SQL コマンド）を実施したあと、対象 SQL の実行計画を確認します。

```
mydb=# EXPLAIN ANALYZE  SELECT * FROM emp WHERE age BETWEEN 30 AND 39;
                              QUERY PLAN
-----------------------------------------------(2)a ----------------------
Bitmap Heap Scan on emp  (cost=32.05..75.27 rows=1148 width=23)
                   (1)      (actual time=0.080..1.877 rows=1148 loops=1)
   Recheck Cond: ((age >= 30) AND (age <= 39))
   Heap Blocks: exact=14                                  (2)b
   ->  Bitmap Index Scan on emp_age_index  (cost=0.00..31.76 rows=1148 width=0)
      (1)                                 (actual time=0.067..0.069 rows=1148 loops=1)
         Index Cond: ((age >= 30) AND (age <= 39))                    (2)a
 Planning Time: 0.317 ms                                              (2)b
 Execution Time: 3.659 ms
(7 行)
```


- （1）「2.3 リストア機能を使って統計情報を固定化する」の手順 1.では「Seq Scan」でしたが、クエリ実行前に統計情報が更新されていると、「Bitmasp Scan」が選択されます。
- （2）emp テーブルに対する推測値の a と実測値の b が一致しており、統計情報が更新され、統計情報の固定化が解除されていることがわかります。

参考

FUJITSU Software Enterprise Postgres では、バージョン 9.5 から、pg_dbms_stats を同梱しています。製品のインストール時に pg_dbms_stats 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。

3. pg_dbms_stats の注意点

pg_dbms_stats を使うことで、間接的に実行計画を制御できますが、以下のような注意点があります。

- **テーブルのデータサイズの変更による影響**
  統計情報を固定化すると、データのサイズや偏りなどデータの特性が変わった場合でも、統計情報が固定化されたままです。必要に応じて、統計情報の固定化を見直す必要があります。
- **オブジェクト削除時の注意**
  テーブルや列などのオブジェクトが不要になった場合は、pg_dbms_stats のロック解除機能やパージ機能を使って pg_dbms_stats 独自の統計情報を先に削除してください。オブジェクトを先に削除した場合は、pg_dbms_stats のクリーンアップ機能を使って pg_dbms_stats 独自の統計情報を削除してください。

性能チューニングの 1 つの手段として、pg_dbms_stats の機能・用途を理解して、ご利用のシステムや業務要件に適した方法を選択してください。

2019 年 8 月 23 日