



EDB

PG Squeeze

リリース 1.6.1

EnterpriseDB Corporation.

2024 年 02 月 21 日

目次

第 1 章	pg_squeeze release notes	3
1.1	Release notes for pg_squeeze version 1.6.1	3
第 2 章	Installing pg_squeeze	5
2.1	インストール	5
2.2	パッケージをインストールする	5
第 3 章	Configuring pg_squeeze	7
第 4 章	Using pg_squeeze	9
4.1	通常処理のためのテーブルの登録	9
4.2	テーブルのアドホック処理	11
4.2.1	サンプル実行	11
4.3	テーブル処理の有効化または無効化	11
4.4	他のバックエンドへの影響の制御	12
4.5	データベースごとに複数のワーカーを実行する	13
4.6	モニタリング	13
4.7	テーブルの登録解除	13
4.8	コンカレンシー	14
4.9	ディスク領域の要件	14

PG Squeeze `pg_squeeze` は、PostgreSQL ライセンスの下でオープンソースソフトウェアとしてリリースされた拡張機能です。

`pg_squeeze` は、テーブルから未使用領域を削除し、オプションで特定のインデックスに従ってタプルをソートします (`CLUSTER` コマンドが通常の読み取りおよび書き込みと同時に実行されたかのように)。実際、`pg_squeeze` は `pg_repack` 拡張機能を置き換えようとしています。 `pg_squeeze` は、非常に似た機能を提供しますが、次のことにより異なるアプローチを採用します。

- サーバーサイドにのみ機能を実装します。このアプローチでは、サーバーサイドとクライアントサイドのコードの両方を使用する `pg_repack` よりも構成と使用の両方が簡単になります。サーバーサイドの実装では、バックグラウンドワーカーを使用した無人処理をスムーズに実装することもできます。
- PostgreSQL データベースサーバーの最近の改善を使用します。最も注目に値するのは、バックグラウンドワーカーの使用に加えて、`pg_squeeze` はトリガーの代わりに論理デコードを使用して同時変更をキャプチャすることです。

`pg_squeeze` の詳細については、以下を参照してください。

- *[Installing pg_squeeze](#)*
- *[Configuring pg_squeeze](#)*
- *[Using pg_squeeze](#)*

第1章 pg_squeeze release notes

pg_squeeze ドキュメントでは、マイナーリリースとパッチを含む pg_squeeze の最新バージョンについて説明しています。これらのリリースノートでは、各リリースの新機能をカバーしています。マイナーまたはパッチリリースで導入された新機能の場合、機能を導入したリリースに関するコンテンツにインジケータもあります。

Version	Release Date
<i>Release notes for pg_squeeze version 1.6.1</i>	15 Feb 2024

1.1 Release notes for pg_squeeze version 1.6.1

pg_squeeze のこのリリースには以下が含まれます。

Type	Description
Feature	これは最初のリリースです。

第2章 Installing pg_squeeze

pg_squeeze は、使用している Postgres ディストリビューションと同じプラットフォームでサポートされています。pg_squeeze のサポートは Postgres 12 から始まります。詳細については、以下を参照してください。

- [EDB Postgres Advanced Server Product Compatibility](#)
- [PostgreSQL Product Compatibility](#)
- [EDB Postgres Extended Server Product Compatibility](#)

2.1 インストール

インストールプロセスを開始する前に

- Postgres をインストールします。参照
- [Installing EDB Postgres Advanced Server](#)
- [Installing PostgreSQL](#)
- [Installing EDB Postgres Extended Server](#)
- リポジトリの設定をします。

リポジトリのセットアップは1回限りのタスクです。すでにリポジトリを設定している場合、この手順を実行する必要はありません。

リポジトリを設定するには、

[EDB repositories](#) に移動し、そこに提供されている指示に従ってください。

2.2 パッケージをインストールする

Package install コマンドの構文は次のとおりです。

```
# For SLES, CentOS, RHEL and its derivatives

sudo <package-manager> -y install edb-<postgres><postgres_version>-pg-squeeze1

# For Debian and Ubuntu

sudo <package-manager> -y install edb-<postgres><postgres_version>-pg-squeeze-1
```

そこで

- `<package-manager>` は、オペレーティングシステムで使われるパッケージマネージャーです。

Package manager	Operating system
dnf	RHEL 8/9 and derivatives
yum	RHEL 7 and derivatives, CentOS 7
zypper	SLES
apt-get	Debian and derivatives

- `<postgres>` は、使用している Postgres のディストリビューションです。

Postgres distribution	Value
PostgreSQL	pg
EDB Postgres Advanced Server	as
EDB Postgres Extended Server	postgresextended

- `<postgres_version>` は、使用している Postgres のバージョンです。

たとえば、RHEL 8 プラットフォームに EDB Postgres Advanced Server 15 の `pg_squeeze` をインストールするには

```
sudo dnf -y install edb-as15-pg-squeeze1
```

第3章 Configuring pg_squeeze

1. `pg_squeeze` を `shared_preload_libraries` パラメーターに追加します。

```
shared_preload_libraries = pg_squeeze
```

注釈: `shared_preload_libraries` に他の拡張機能がある場合、`pg_squeeze` をリストに追加できます。順序は関係ありません。 !!!

1. 次の変更を `postgresql.conf` に追加します。

```
wal_level = logical  
max_replication_slots = 1 # ... or add 1 to the current value.
```

1. Postgres を再起動します。
2. データベースに `pg_squeeze` 拡張機能を作成します。

```
CREATE EXTENSION pg_squeeze;
```

注釈: `pg_squeeze` がインストールされているデータベースクラスターをアップグレードする場合 `pg_dumpall` `pg_restore` または `pg_upgrade` を使用して、アップグレードする前に、新しいクラスターの `shared_preload_libraries` に `pg_squeeze` があることを確認してください。それ以外の場合、アップグレードは失敗します。

第4章 Using pg_squeeze

4.1 通常処理のためのテーブルの登録

最初に、テーブルに主キーまたは一意性制約のいずれかが存在することを確認します。これは、`pg_squeeze` が作業を行っているときに他のトランザクションから発生した変更を処理するために必要です。

`pg_squeeze` 拡張機能にテーブルを認識させるには、`squeeze.tables` テーブルにレコードを挿入する必要があります。追加後、テーブル統計は定期的にチェックされます。テーブルがスキーズに必要な基準を満たしている場合、タスクがキューに追加されます。タスクは作成された順にシークエンシャルに処理されます。

次に、簡単な登録の例を示します。

```
INSERT INTO squeeze.tables (tabschema, tabname, schedule)
VALUES (public, foo, ({30}, {22}, NULL, NULL, {3, 5}));
```

オプションで追加の列を指定できます。例

```
INSERT INTO squeeze.tables (
    tabschema,
    tabname,
    schedule,
    free_space_extra,
    vacuum_max_age,
    max_retry
)
VALUES (
    public,
    bar,
    ({30}, {22}, NULL, NULL, {3, 5}),
    30,
    2 hours,
    2
);
```

次のリストでは、テーブルのメタデータを説明します。

- `tabschema` —スキーマ名。
- `tabname` —テーブル名。

- **schedule** —テーブルをチェックし、場合によってはスキーズする場合を指定します。スケジュールは、次の複合データ型の値で記述されます。これは、`crontab` エントリに似ています。

```
CREATE TYPE schedule AS (
    minutes      minute[],
    hours        hour[],
    days_of_month dom[],
    months       month[],
    days_of_week dow[]
);
```

このデータタイプでは、`minutes` 0-59 および `hours` 0-23 は、1 日以内にチェックが発生した時間を決定しますが、`days_of_month` 1-31、`months` 1-12、および `days_of_week` 0-7、0 と 7 は両方とも日曜日を表します) によって、小切手の日が決まります。

`minutes`、`hours`、および `months` がすべて現在のタイムスタンプと一致する場合に、チェックが実行されます。NULL は、それぞれ分、時、月を意味します。チェックが行われるには、`days_of_month` または `days_of_week` のいずれかが現在のタイムスタンプと一致するか、両方が NULL である必要があります。

たとえば、サンプル登録のエントリは、毎週水曜日と金曜日の 22:30 にテーブル `public.bar` をチェックするように指定しています。

- **free_space_extra** —テーブルの処理をトリガーするために必要な追加空き領域の最小割合。**extra** 形容詞は、**fillfactor** から派生する空き領域はテーブルを圧迫する理由ではないという事実を指します。

たとえば、**fillfactor** が 60 に等しい場合、通常のオペレーション中に各ページの少なくとも 40 パーセントが空きままになります。空き領域の 70 パーセントで `pg_squeeze` がテーブルに関心を持つようにしたい場合は、**free_space_extra** を 30 つまり、解放するために必要な 70 パーセントから **fillfactor** のための 40 パーセントを差し引いたものを設定します。

free_space_extra のデフォルト値は 50 です。

- **min_size** —処理の対象となるためにテーブルが占有する必要があるメガバイト単位の最小ディスク領域。デフォルト値は 8 です。
- **vacuum_max_age** —最後の **VACUUM** の完了から空き領域マップ FSM が新しいと見なされるまでの最大時間。この間隔が経過すると、デッドタブルの部分が重要になる可能性があります。この場合、FSM をチェックするだけでなく、`pg_squeeze` の潜在的な効果を評価するためにより多くの労力を費やす必要があります。デフォルト値は 1 時間です。
- **max_retry** —対応するタスクの最初の処理が失敗した場合に、テーブルをスキーズしようとする追加の試行の最大数。処理を再試行する一般的な理由は、テーブルが圧迫されているときにテーブル定義が変更されたことです。再試行回数に達すると、テーブルの処理が完了したとみなされます。次のタスクは、次のスケジュールされた時刻に作成されます。**max_retry** のデフォルト値は 0 つまり、再試行しません。

注釈: `squeeze.table` は、変更する唯一のテーブルです。他に変更したい場合は、自分が何をしているのかを理解してください。

4.2 テーブルのアドホック処理

登録せずに、対応するレコードを `squeeze.tables` に挿入せず、実際の肥大化を事前に確認せずに、テーブルを手動でスクイーズすることが可能です。

ファンクションシングネチャ

```
squeeze.squeeze_table(
    tabschema name,
    tabname name,
    clustering_index name,
    rel_tablespace name,
    ind_tablespaces name[]
)
```

次のリストでは、アドホック処理のテーブルメタデータを説明します。

- **clustering_index** —処理されたテーブルのインデックス。処理が完了すると、テーブルのタプルがこのインデックスのキーによって物理的にソートされます。
- **rel_tablespace** —テーブルの移動先の既存の表領域。 NULL は、テーブルを現在の場所に残すことを意味します。
- **ind_tablespaces** — 各行がインデックスのテーブルスペースマッピングを指定する 2 次元配列。最初と 2 番目の列は、それぞれインデックス名とテーブルスペース名を表します。マッピングが指定されていないすべてのインデックスは、元の表領域に残ります。 **.. note::**
 テーブルに表領域が指定されているが、インデックスは指定されていない場合、テーブルはその表領域に移動されますが、インデックスは元の表領域に残ります。言い換えれば、テーブルの表領域はインデックスのデフォルトではありません。

4.2.1 サンプル実行

```
SELECT squeeze.squeeze_table(public, pgbench_accounts);
```

4.3 テーブル処理の有効化または無効化

肥大化したテーブルの処理を有効にするには、スーパーユーザーとして次のステートメントを実行します。

```
SELECT squeeze.start_worker();
```

ファンクションは、バックグラウンドワーカー `scheduler worker` を起動します。特定のデータベースにタスクが存在するときは常に、別のワーカー `squeeze worker` が起動されます。

スケジューラーワーカーが現在のデータベースで既に実行されている場合、ファンクションはエラーを報告せず、新しいワーカーはすぐに終了します。

ワーカーが現在のデータベースで実行されている場合、次のステートメントを使用してワーカーを停止できます。

```
SELECT squeeze.stop_worker();
```

注釈: このドキュメントで記載されている機能のみがユーザーインターフェイスの一部とみなされます。他のファンクションを呼び出したい場合は、自分が何をしているのかを理解してください。

PostgreSQL クラスター全体の起動中にバックグラウンドワーカーを起動したい場合は、次のようなエントリを `postgresql.conf` ファイルに追加します。

```
squeeze.worker_autostart = my_database your_database  
squeeze.worker_role = postgres
```

次回クラスターを起動したときに、2 つ以上のワーカーつまり 1 つのスケジューラーワーカーと 1 つ以上のスクイーズワーカーが `my_database` で起動され、`your_database` でも同じです。このアプローチを採用した場合、次のいずれかの場合、ワーカーは作業を実行せずに開始または停止しません。

- `pg_squeeze` 拡張機能はデータベースに存在しません。
- `squeeze.worker_role` パラメーターは、スーパーユーザー権限を持たないロールを指定しています。

実際には 2 つのワーカーがありますが、ここで説明するファンクションと構成変数は、ワード `worker` の単数形を使用しています。これは、タスクのスケジューリングと実行の両方を保証していた `pg_squeeze` の以前のバージョンには `worker` が 1 つだけ存在していたためです。したがって、この実装の変更は、アップグレード中にすべてのユーザーに構成ファイルの調整を強制するものではありません。

4.4 他のバックエンドへの影響の制御

スクイーズされるテーブルは、ほとんどの場合、他のトランザクションによる読み取りおよび書き込み操作の両方に利用できますが、処理を終了するには排他ロックが必要です。`pg_squeeze` がテーブルへのアクセスをブロックする必要がある場合は、`squeeze.max_xlock_time` GUC パラメーターの設定を検討してください。例

```
SET squeeze.max_xlock_time TO 100;
```

この例では、0.1 秒 100 ミリ秒を超えて排他ロックを保持しないように指定します。最終ステージにもっと時間が必要な場合、`pg_squeeze` は排他ロックを解放し、間の他のトランザクションによってコミットされた変更を処理し、最終ステージを再試行します。ロック期間を数回超過すると、エラーが報告されます。その場合は、設定を増やすか、書き込みアクティビティが低い別の時間に問題のあるテーブルの処理をスケジューリングします。

4.5 データベースごとに複数のワーカーを実行する

単一のスクイーズワーカーが負荷に対応しないと考えられる場合は、`squeeze.workers_per_database` 構成変数を 1 より大きい値に設定することを検討します。これにより、`pg_squeeze` 拡張機能はマルチプルのテーブルを同時に処理できます。スクイーズワーカーごとに 1 つのテーブル。

ただし、この設定は、`pg_squeeze` 拡張機能をアクティブに使用するすべてのデータベースに影響を与えることに注意してください。クラスター内のすべてのスクイーズワーカースケジューラーワーカーを含むの合計数は、コア内構成変数 `max_worker_processes` を超えることはできません。

4.6 モニタリング

`squeeze.log` テーブルには、正常にスクイーズされたテーブルごとに 1 つのエントリが含まれています。

列 `tabschema` および `tablename` は、処理されたテーブルを識別します。列 `started` および `finished` は、処理が開始および終了した時期をレポートします。`ins_initial` は、初期ロードステージ中に新しいテーブルストレージに挿入されたタプルの数、つまり、処理が開始される前にテーブルに存在していたタプルの数です。一方、`ins`、`upd`、および `del` は、テーブル処理中にアプリケーションによって挿入、更新、および削除されたタプルの数です。(これらの同時データの変更も、スクイーズされたテーブルに組み込む必要があります。そうしないと、失われます。)

`squeeze.errors` テーブルには、スクイーズ中に発生したエラーが含まれています。よくある問題は、処理が進行中のテーブルの定義を変更する人がいるということです。つまり、列を追加または削除します。

`squeeze.get_active_workers()` ファンクションは、現在のデータベースでテーブルを処理しているスクイーズワーカーのテーブルを結果ます。

`pid` 列には、ワーカープロセスのシステム PID が含まれています。他の列は、`squeeze.log` テーブルの対応する列と同じ意味を持ちます。`squeeze.log` テーブルは、完了したスクイーズ操作に関する情報のみを表示しますが、`squeeze.get_active_workers()` ファンクションを使用すると、処理中に進行状況を確認できます。

4.7 テーブルの登録解除

特定のテーブルが定期的なスクイーズの対象でなくなった場合、`squeeze.tables` テーブルから対応する行を削除します。

バックグラウンドワーカーは存在しないテーブルの登録を定期的に解除しますが、ドロップするテーブルの登録を解除することをお勧めします。

4.8 コンカレンシー

`pg_squeeze` 拡張機能は、他のトランザクションが処理の特定の段階でテーブルを変更するのを妨げません。スキーズが完了する前に、破壊的なコマンドつまり `ALTER TABLE`、`VACUUM FULL`、`CLUSTER`、または `TRUNCATE` がコミットできた場合、`squeeze_table()` ファンクションは中止され、テーブルに加えられたすべての変更はロールバックされます。`squeeze.tables` テーブルの `max_retry` 列は、スキーズワーカーが再試行する回数を決定します。スケジュールを変更すると、この種の中断を回避できる場合があります。

`pg_repack` と同様に、`pg_squeeze` も行の可視性を変更するため、MVCC の [Caveats](#) の最初の段落で説明されている MVCC 非安全な動作が許可されます。

4.9 ディスク領域の要件

フルテーブルスキーズを実行するには、ターゲットテーブルとそのインデックスの 2 倍の空きディスク領域が必要です。たとえば、圧縮するテーブルとインデックスの合計サイズが 1GB の場合、追加で 2GB のディスク領域が必要です。