



EDB Postgres Extended Server

リリース *16.1.0*

EnterpriseDB Corporation.

2023 年 11 月 27 日

目次

第 1 章	Release notes	3
1.1	EDB Postgres Extended Server 16.1 release notes	3
第 2 章	Deployment options	5
第 3 章	Installing EDB Postgres Extended Server on Linux	7
3.1	Linux x86-64 (amd64)	7
3.1.1	Red Hat Enterprise Linux RHEL とその派生	7
3.1.2	Debian とデリバティブ	7
第 4 章	Features	19
4.1	Transparent data encryption	19
4.2	Replication	19
4.2.1	非同期処理	20
第 5 章	Reference	21
5.1	Operations	21
5.1.1	トランザクションログのフラッシュを回避する	21
5.1.2	追加のトレースと診断オプション	21
5.1.3	選択的な物理ベースのバックアップとその後の選択的なリカバリ/リストア	21
5.1.4	追加の操作機能	22
5.2	SQL enhancements	22
5.2.1	ロールバックオプション	22
5.2.2	準備されたステートメントを使用したカーソル	22
5.2.3	PL/pgSQL の互換性	23
5.3	Configuration parameters (GUCs)	26
5.3.1	バックエンドパラメーター	26
5.3.2	server_2q_version_num および server_2q_version	28
5.3.3	テーブルレベルの圧縮制御オプション	28
5.3.4	pg2q.max_tuple_field_size	28
5.3.5	pg2q.max_tuple_size	30

EDB Postgres Extended Server は、オープンソースのコミュニティ PostgreSQL に基づいて構築された Postgres データベースサーバディストリビューションです。PostgreSQL と完全な互換性があります。PostgreSQL で動作するように作成およびテストされたアプリケーションがある場合、EDB Postgres Extended Server で同じ動作をします。コミュニティ PostgreSQL と EDB Postgres Extended Server の機能または動作の違いをサポートおよび修正します。

EDB Postgres Extended Server の主な目的は、他の点でパリティを維持しながら、

EDB Postgres Distributed および *Transparent data encryption* で使用される *enhanced replication optimization* など、拡張機能として実装できない限定的な数の機能で PostgreSQL を拡張することです。

追加の付加価値エンタープライズ機能には次のものが含まれます。

- *Transparent data encryption* によるセキュリティ
- コミュニティ PostgreSQL へのオプションの SQL superset
- *WAL pacing delays to avoid flooding transaction logs*
- 追加のトレースと診断オプション

第1章 Release notes

EDB Postgres Extended Server ドキュメントでは、マイナーリリースとパッチを含む EDB Postgres Extended Server 16 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。

Version	Release date
<i>EDB Postgres Extended Server 16.1 release notes</i>	09 Nov 2023

1.1 EDB Postgres Extended Server 16.1 release notes

リリース日:2023 年 11 月 09 日

EDB Postgres Extended Server 16.1 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Upstream merge	コミュニティ PostgreSQL 16.1 とマージされました。詳細については、 PostgreSQL 16.1 Release Notes を参照してください。
Enhancement	透過的データ暗号化 TDE は、256 ビット AES キー長をサポートするようになりました。
	NA

第2章 Deployment options

展開オプションには次のものが含まれます。

- ネイティブパッケージを使用する仮想マシンまたは物理サーバーでの *Installing EDB Postgres Extended Server on Linux*
- Trusted Postgres Architect を使用して EDB Postgres Distributed で展開する
- 超高可用性クラスタータイプを使用した BigAnimal への展開

第3章 Installing EDB Postgres Extended Server on Linux

リンクを選択して、該当するインストール手順にアクセスします。

3.1 Linux x86-64 (amd64)

3.1.1 Red Hat Enterprise Linux RHEL とその派生

- *Installing EDB Postgres Extended Server on RHEL 9 or OL 9 x86_64*、*Installing EDB Postgres Extended Server on RHEL 8 or OL 8 x86_64*、*Installing EDB Postgres Extended Server on RHEL 7 or OL 7 x86_64*
- *Oracle Linux (OL) 9*、*Oracle Linux (OL) 8*、*Oracle Linux (OL) 7*
- *Installing EDB Postgres Extended Server on AlmaLinux 9 or Rocky Linux 9 x86_64*、*Installing EDB Postgres Extended Server on AlmaLinux 8 or Rocky Linux 8 x86_64*
- *Installing EDB Postgres Extended Server on AlmaLinux 9 or Rocky Linux 9 x86_64*、*Installing EDB Postgres Extended Server on AlmaLinux 8 or Rocky Linux 8 x86_64*
- *Installing EDB Postgres Extended Server on CentOS 7 x86_64*

3.1.2 Debian とデリバティブ

- *Installing EDB Postgres Extended Server on Ubuntu 22.04 x86_64*、*Installing EDB Postgres Extended Server on Ubuntu 20.04 x86_64*、*Installing EDB Postgres Extended Server on Ubuntu 18.04 x86_64*
- *Installing EDB Postgres Extended Server on Debian 11 x86_64*、*Installing EDB Postgres Extended Server on Debian 10 x86_64*

Installing EDB Postgres Extended Server on Linux x86 (amd64)

オペレーティングシステム固有のインストール手順は、対応するドキュメントに記載されています。

Red Hat Enterprise Linux RHEL とその派生

- *Installing EDB Postgres Extended Server on RHEL 9 or OL 9 x86_64*
- *Installing EDB Postgres Extended Server on RHEL 8 or OL 8 x86_64*
- *Installing EDB Postgres Extended Server on RHEL 7 or OL 7 x86_64*
- *Oracle Linux (OL) 9*
- *Oracle Linux (OL) 8*
- *Oracle Linux (OL) 7*
- *Installing EDB Postgres Extended Server on AlmaLinux 9 or Rocky Linux 9 x86_64*
- *Installing EDB Postgres Extended Server on AlmaLinux 8 or Rocky Linux 8 x86_64*
- *Installing EDB Postgres Extended Server on AlmaLinux 9 or Rocky Linux 9 x86_64*
- *Installing EDB Postgres Extended Server on AlmaLinux 8 or Rocky Linux 8 x86_64*
- *Installing EDB Postgres Extended Server on CentOS 7 x86_64*

Debian とデリバティブ

- *Installing EDB Postgres Extended Server on Ubuntu 22.04 x86_64*
- *Installing EDB Postgres Extended Server on Ubuntu 20.04 x86_64*
- *Installing EDB Postgres Extended Server on Ubuntu 18.04 x86_64*
- *Installing EDB Postgres Extended Server on Debian 11 x86_64*
- *Installing EDB Postgres Extended Server on Debian 10 x86_64*

Installing EDB Postgres Extended Server on RHEL 9 or OL 9 x86_64

前提条件

インストールプロセスを開始する前に

- EDB 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。

リポジトリが存在するかどうかを判断するには、次のコマンドを入力します。

```
dnf repolist | grep enterprisedb
```

出力が生成されない場合、リポジトリはインストールされていません。

EDB リポジトリを設定するには

1. EDB repositories に移動します。
 2. EDB リポジトリへのアクセスを提供するボタンを選択します。
 3. ダウンロードするプラットフォームとソフトウェアを選択します。
 4. EDB リポジトリのセットアップの指示に従います。
- EPEL リポジトリをインストールします。

```
sudo dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.
→noarch.rpm
```

パッケージをインストールする

```
sudo dnf -y install edb-postgresextended<xx>-server edb-postgresextended<xx>-contrib
```

<xx> は、インストールする EDB Postgres Extended Server のバージョンです。たとえば、バージョン 16 をインストールする場合、パッケージ名は edb-postgresextended16-server edb-postgresextended16-contrib になります。

Installing EDB Postgres Extended Server on RHEL 8 or OL 8 x86_64

前提条件

インストールプロセスを開始する前に

- EDB 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。

リポジトリが存在するかどうかを判断するには、次のコマンドを入力します。

```
dnf repolist | grep enterprisedb
```

出力が生成されない場合、リポジトリはインストールされていません。

EDB リポジトリを設定するには

1. EDB repositories に移動します。
 2. EDB リポジトリへのアクセスを提供するボタンを選択します。
 3. ダウンロードするプラットフォームとソフトウェアを選択します。
 4. EDB リポジトリのセットアップの指示に従います。
- EPEL リポジトリをインストールします。

```
sudo dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.
→noarch.rpm
```

パッケージをインストールする

```
sudo dnf -y install edb-postgresextended<xx>-server edb-postgresextended<xx>-contrib
```

<xx> は、インストールする EDB Postgres Extended Server のバージョンです。たとえば、バージョン 16 をインストールする場合、パッケージ名は edb-postgresextended16-server edb-postgresextended16-contrib になります。

Installing EDB Postgres Extended Server on AlmaLinux 9 or Rocky Linux 9 x86_64

前提条件

インストールプロセスを開始する前に

- EDB 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。

リポジトリが存在するかどうかを判断するには、次のコマンドを入力します。

```
dnf repolist | grep enterprisedb
```

出力が生成されない場合、リポジトリはインストールされていません。

EDB リポジトリを設定するには

1. [EDB repositories](#) に移動します。
 2. EDB リポジトリへのアクセスを提供するボタンを選択します。
 3. ダウンロードするプラットフォームとソフトウェアを選択します。
 4. EDB リポジトリのセットアップの指示に従います。
- EPEL リポジトリをインストールします。

```
sudo dnf -y install epel-release
```

- 追加のリポジトリを有効にして依存関係を解決します。

```
sudo dnf config-manager --set-enabled powertools
```

パッケージをインストールする

```
sudo dnf -y install edb-postgresextended<xx>-server edb-postgresextended<xx>-contrib
```

<xx> は、インストールする EDB Postgres Extended Server のバージョンです。たとえば、バージョン 16 をインストールする場合、パッケージ名は edb-postgresextended16-server edb-postgresextended16-contrib になります。

Installing EDB Postgres Extended Server on AlmaLinux 8 or Rocky Linux 8 x86_64

前提条件

インストールプロセスを開始する前に

- EDB 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。

リポジトリが存在するかどうかを判断するには、次のコマンドを入力します。

```
dnf repolist | grep enterprisedb
```

出力が生成されない場合、リポジトリはインストールされていません。

EDB リポジトリを設定するには

1. [EDB repositories](#) に移動します。
 2. EDB リポジトリへのアクセスを提供するボタンを選択します。
 3. ダウンロードするプラットフォームとソフトウェアを選択します。
 4. EDB リポジトリのセットアップの指示に従います。
- EPEL リポジトリをインストールします。

```
sudo dnf -y install epel-release
```

- 追加のリポジトリを有効にして依存関係を解決します。

```
sudo dnf config-manager --set-enabled powertools
```

パッケージをインストールする

```
sudo dnf -y install edb-postgresextended<xx>-server edb-postgresextended<xx>-contrib
```

<xx> は、インストールする EDB Postgres Extended Server のバージョンです。たとえば、バージョン 16 をインストールする場合、パッケージ名は edb-postgresextended16-server edb-postgresextended16-contrib になります。

Installing EDB Postgres Extended Server on RHEL 7 or OL 7 x86_64

前提条件

インストールプロセスを開始する前に

- EDB 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。

リポジトリが存在するかどうかを判断するには、次のコマンドを入力します。

```
dnf repolist | grep enterprisedb
```

出力が生成されない場合、リポジトリはインストールされていません。

EDB リポジトリを設定するには

1. EDB repositories に移動します。
 2. EDB リポジトリへのアクセスを提供するボタンを選択します。
 3. ダウンロードするプラットフォームとソフトウェアを選択します。
 4. EDB リポジトリのセットアップの指示に従います。
- EPEL リポジトリをインストールします。

```
sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.  
↪noarch.rpm
```

- 追加のリポジトリを有効にして依存関係を解決します。

```
subscription-manager repos --enable "rhel-*optional-rpms" --enable "rhel-*extras-  
↪rpms" --enable "rhel-ha-for-rhel-*server-rpms"
```

パッケージをインストールする

```
sudo yum -y install edb-postgresextended<xx>-server edb-postgresextended<xx>-contrib
```

<xx> は、インストールする EDB Postgres Extended Server のバージョンです。たとえば、バージョン 16 をインストールする場合、パッケージ名は edb-postgresextended16-server edb-postgresextended16-contrib になります。

Installing EDB Postgres Extended Server on CentOS 7 x86_64

前提条件

インストールプロセスを開始する前に

- EDB 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。

リポジトリが存在するかどうかを判断するには、次のコマンドを入力します。

```
dnf repolist | grep enterprisedb
```

出力が生成されない場合、リポジトリはインストールされていません。

EDB リポジトリを設定するには

1. [EDB repositories](#) に移動します。
 2. EDB リポジトリへのアクセスを提供するボタンを選択します。
 3. ダウンロードするプラットフォームとソフトウェアを選択します。
 4. EDB リポジトリのセットアップの指示に従います。
- EPEL リポジトリをインストールします。

```
sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.
↪noarch.rpm
```

パッケージをインストールする

```
sudo yum -y install edb-postgresextended<xx>-server edb-postgresextended<xx>-contrib
```

<xx> は、インストールする EDB Postgres Extended Server のバージョンです。たとえば、バージョン 16 をインストールする場合、パッケージ名は edb-postgresextended16-server edb-postgresextended16-contrib になります。

Installing EDB Postgres Extended Server on Ubuntu 22.04 x86_64

前提条件

インストールプロセスを開始する前に

- EDB 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。

リポジトリが存在するかどうかを判断するには、次のコマンドを入力します。

```
apt-cache search enterprisedb
```

出力が生成されない場合、リポジトリはインストールされていません。

EDB リポジトリを設定するには

1. [EDB repositories](#) に移動します。
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリのセットアップの指示に従います。

パッケージをインストールする

```
sudo apt-get -y install edb-postgresextended-<xx> edb-postgresextended-<xx>-contrib
```

<xx> は、インストールする EDB Postgres Extended Server のバージョンです。たとえば、バージョン 16 をインストールする場合、パッケージ名は edb-postgresextended-16 edb-postgresextended-16-contrib になります。

Installing EDB Postgres Extended Server on Ubuntu 20.04 x86_64

前提条件

インストールプロセスを開始する前に

- EDB 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。

リポジトリが存在するかどうかを判断するには、次のコマンドを入力します。

```
apt-cache search enterprisedb
```

出力が生成されない場合、リポジトリはインストールされていません。

EDB リポジトリを設定するには

1. EDB repositories に移動します。
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリのセットアップの指示に従います。

パッケージをインストールする

```
sudo apt-get -y install edb-postgresextended-<xx>
```

<xx> は、インストールする EDB Postgres Extended Server のバージョンです。たとえば、バージョン 16 をインストールする場合、パッケージ名は edb-postgresextended-16 になります。

Installing EDB Postgres Extended Server on Ubuntu 18.04 x86_64

前提条件

インストールプロセスを開始する前に

- EDB 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。

リポジトリが存在するかどうかを判断するには、次のコマンドを入力します。

```
apt-cache search enterprisedb
```

出力が生成されない場合、リポジトリはインストールされていません。

EDB リポジトリを設定するには

1. EDB repositories に移動します。
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリのセットアップの指示に従います。

パッケージをインストールする

```
sudo apt-get -y install edb-postgresextended-<xx>
```

<xx> は、インストールする EDB Postgres Extended Server のバージョンです。たとえば、バージョン 16 をインストールする場合、パッケージ名は edb-postgresextended-16 になります。

Installing EDB Postgres Extended Server on Debian 11 x86_64

前提条件

インストールプロセスを開始する前に

- EDB 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。

リポジトリが存在するかどうかを判断するには、次のコマンドを入力します。

```
apt-cache search enterprisedb
```

出力が生成されない場合、リポジトリはインストールされていません。

EDB リポジトリを設定するには

1. EDB repositories に移動します。
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリのセットアップの指示に従います。

パッケージをインストールする

```
sudo apt-get -y install edb-postgresextended-<xx>
```

<xx> は、インストールする EDB Postgres Extended Server のバージョンです。たとえば、バージョン 15 をインストールする場合、パッケージ名は edb-postgresextended-15 になります。

Installing EDB Postgres Extended Server on Debian 10 x86_64

前提条件

インストールプロセスを開始する前に

- EDB 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。

リポジトリが存在するかどうかを判断するには、次のコマンドを入力します。

```
apt-cache search enterprisedb
```

出力が生成されない場合、リポジトリはインストールされていません。

EDB リポジトリを設定するには

1. EDB repositories に移動します。
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリのセットアップの指示に従います。

パッケージをインストールする

```
sudo apt-get -y install edb-postgresextended-<xx>
```

<xx> は、インストールする EDB Postgres Extended Server のバージョンです。たとえば、バージョン 15 をインストールする場合、パッケージ名は edb-postgresextended-15 になります。

第4章 Features

4.1 Transparent data encryption

透過的データ暗号化 TDE は、データベースシステムに保存されているユーザーデータを暗号化します。この暗号化はユーザーに対して透過的です。ユーザーデータには、テーブルやその他のオブジェクトに格納されている実際のデータ、およびオブジェクトの名前などのシステムカタログデータが含まれます。

詳細については、[Transparent data encryption](#) を参照してください。

4.2 Replication

EDB Postgres Extended Server は、

EDB Postgres Distributed の次のレプリケーションおよび高可用性機能をサポートするコア機能を提供します。

- Commit At Most OnceCAMO
- グループコミット
- イージャーレプリケーション
- デコードワーカー
- 評価ツール
- ラグトラッカー
- ラグコントロール
- タイムスタンプのスナップショット
- トランザクションストリーミング
- 不足しているパーティションの競合
- TOAST を使用するテーブルで UPDATE トリガーは必要ありません
- 自動的にフリーズを抑制

4.2.1 非同期処理

EDB Postgres Extended Server には、`synchronous_replication_availability` パラメーターが含まれています。このパラメーターの値が `async` の場合、十分なスタンバイサーバーが使用できない場合に、非同期処理が有効になります `synchronous_standby_names` ごとの値と比較した場合。必要な数の同期スタンバイサーバーが再び表示されると、動作は同期レプリケーションに戻ります。

第5章 Reference

5.1 Operations

EDB Postgres Extended Server には、操作に関連する多数の機能があります。

5.1.1 トランザクションログのフラッシングを回避する

EDB Postgres Extended Server は、WAL ペーシング遅延を提供して、トランザクションログのフラッシングを回避します。WAL ペーシング構成パラメーターは次のとおりです。-wal_insert_delay_enabled -wal_insert_delay -wal_insert_delay_size wal_insert_delay_enabled が有効になっている場合、少なくとも wal_insert_delay_size の値の WAL データが生成された後、wal_insert_delay の値に基づいてセッションはスリープします。デフォルトはオフです。

5.1.2 追加のトレースと診断オプション

EDB Postgres Extended Server では、特定のコードパスのトレースメッセージのログGINGに基づいてタイムアウトを有効にできます。tracelog_timeout 構成パラメーターを使用して、指定された時間のタイムアウトが発生した後、トレースメッセージのログGINGを許可します。

5.1.3 選択的な物理ベースのバックアップとその後の選択的なリカバリ/リストア

デフォルトでは、バックアップは常にデータベースクラスター全体から取得されます。複数のデータベースに pg_basebackup ユーティリティで-L オプションを複数回指定することにより、個々のデータベースまたはデータベースオブジェクトをバックアップすることもできます。

テンプレートデータベースはデフォルトでバックアップされます。除外されたデータベースの WAL データは、引き続き WAL アーカイブの一部です。

バックアップアクティビティは、このオプションを使用して指定されたデータベースオブジェクトのリストをバックアップラベルファイルに保存します。バックアップラベルファイルにこれらのオブジェクトが存在すると、これらのデータベースの選択的なリカバリが発生します。テンプレートデータベースの復旧と、ユーザー、言語などに関連するグローバルメタデータの復旧も通常通り実行されます。除外されたデータベースに属する WAL データは、復旧プロセス中に無視されます。復旧後に通常の操作が開始された後、除外されたデータベースに接続しようとするエラーが発生します。

5.1.4 追加の操作機能

- ALTER TABLE ... REPLICA IDENTITY のロックの削減

5.2 SQL enhancements

EDB Postgres Extended Server には、多数の SQL の機能強化が含まれています。

5.2.1 ロールバックオプション

PostgreSQL では、トランザクションでエラーが発生すると、そのトランザクションによるすべてのアクションがロールバックされます。この動作は、エラーによって最後のステートメントのみがロールバックされる Oracle や SQL Server などの他の DBMS とは異なります。トランザクション処理セマンティクスのこの違いは、すべての場合で問題を引き起こすわけではありませんが、Oracle Database および Microsoft SQL Server の開発者にとって、PostgreSQL でのビジネスロジックの実装が困難になります。

1 つの回避策は、内部でサブトランザクションとして知られるセーブポイントをアプリケーションコードに手動で導入することです。これは時間がかかり、テストが困難です。セーブポイントは追加のステートメントであるため、トランザクションのレイテンシーが増加します。追加の開発作業のオーバーヘッドとパフォーマンスの低下を考慮すると、このアプローチはほとんどの場合実行可能ではありません。

EDB Postgres Extended Server では、現在のステートメントのみをロールバックできます。ステートメントレベルのロールバック機能は、トランザクション全体または現在のステートメントのみのロールバックを許可するかどうかを選択するオプションのモードを提供します。手動でのコーディングは必要ありません。若干のオーバーヘッドが追加されますが、セーブポイントよりも低いです。

データベース内のトランザクションロールバックスコープの設定については [transaction_rollback_scope](#) を参照し、JDBC バッチジョブのエラーを過ぎて続行する方法については [JDBC properties for rollback scope](#) を参照してください。

5.2.2 準備されたステートメントを使用したカーソル

EDB Postgres Extended Server では、以前に作成したプリペアドステートメントの上でカーソルを宣言できます。

例

```
PREPARE foo AS ...; DECLARE c1 CURSOR FOR foo;
```

5.2.3 PL/pgSQL の互換性

EDB Postgres Extended Server は、多くの PL/pgSQL 互換性機能を備えた他の移行ツールと統合しています。

全体的な簡素化のために、EDB Postgres Extended Server では、PERFORM キーワードなしで `plpgsql` を使用してファンクションを呼び出すことができます。

例

```
BEGIN somefunc(); END
```

`somefunc` はキーワードではありません。

transaction_rollback_scope parameter

データベース内のトランザクションロールバックスコープを設定するには、`transaction_rollback_scope` パラメーターを使用します。`transaction_rollback_scope` パラメーターには、2つの値があります。

- **transaction** — 標準の Postgres 動作。各エラーはトランザクション全体を中止します。
- **statement** — 1つのステートメントの実行中のエラーは、そのステートメントにのみ影響を及ぼし、トランザクション全体のステータスには影響しません。

パラメーターの設定

パラメーターをユーザーレベルのプロパティ、接続オプション、または特定のファンクションまたはプロシージャのモードとして設定できます。

パラメーターをユーザーレベルのプロパティとして設定します

```
ALTER USER somebody SET transaction_rollback_scope TO statement;
```

パラメーターを接続オプションとして設定する

```
PGOPTIONS="-c transaction_rollback_scope=statement" psql <other options>
```

特定のファンクションまたはプロシージャのモードを設定する

PL/pgSQL を使用する場合、特定のファンクションまたはプロシージャのモードを設定できます。

```
ALTER FUNCTION myfunc SET transaction_rollback_scope TO statement;
```

サブトランザクションの処理方法

statement 値を選択した場合、各 SQL コマンドの直前にサブトランザクションが開きます。コマンドが成功した場合、サブトランザクションは**コミット**されます。コマンドによってエラーが発生した場合、サブトランザクションはロールバックされ、親トランザクションは通常に続行できます。その結果、1つのステートメントの実行中のエラーは、そのステートメントにのみ影響を与え、トランザクション全体のステータスには影響しません。

サブトランザクションをコミットすると、それが保持するリソースがその親トランザクションにのみ割り当てられます。これがトップレベルのトランザクションである場合があります。または、ユーザー定義のセーブポイントが関係する場合、それは他のサブトランザクションである可能性があります。したがって、これは「自律トランザクション」ではありません。サブトランザクションをロールバックすると、取得したロックなど、保持しているすべてのリソースが解放されます。

JDBC properties for setting rollback scope

JDBC コネクタを使用してクライアントアプリケーションに接続している場合、**autosave** および **transaction_rollback_scope** プロパティを一緒に使用してトランザクションのロールバックスコープを指定します。

これらのプロパティは、接続 URL で、または `DriverManager.getConnection` への追加プロパティオブジェクトパラメーターとして指定できます。

autosave

autosave パラメーターは、複数のステートメントを含むクエリーが失敗した場合にドライバーが実行することを指定する文字列です。使用可能な値は **server**、**always**、**never**、および **conservative** です。

- **autosave=server** モードでは、JDBC はサーバーサイドパラメーター **transaction_rollback_scope** を利用して、次のステートメントを実行する前に内部サーバーセーブポイントを介して各ステートメントを保存します。クエリ内のステートメントが失敗した場合、サーバーは前のステートメントにロールバックします。このパラメーターがサーバーサイドでサポートされていない場合、JDBC は接続を拒否します。
- **autosave=always** モードでは、JDBC ドライバーは最初にサーバーサイドの **transaction_rollback_scope** プロパティを使用しようとします。サポートされていない場合、JDBC ドライバーは各クエリステートメントの前にセーブポイントを設定し、失敗した場合にそのセーブポイントにロールバックします。

- **autosave=never** モードデフォルトでは、セーブポイントアクティビティは実行されません。
autosave=conservative モードでは、クエリーごとにセーブポイントが設定されます。ただし、ロールバックは、「キャッシュされたステートメントは戻り値の型を変更できない」または「ステートメント XXX は有効ではない」などのまれな場合にのみ実行されるため、JDBC ドライバーはロールバックして再試行します。

このプロパティのデフォルト値は **never** です。

この **autosave=server** プロパティは、**transaction_rollback_scope** 機能を提供する PostgreSQL サーバーでのみ役立ちます。

transaction_rollback_scope

autosave パラメーターは、SQL ステートメントが失敗した場合にロールバックする操作の範囲を決定する文字列です。

デフォルト値は **TRANSACTION** で、トランザクション全体または現在のサブトランザクションがロールバックされます。これは、**SET TRANSACTION** コマンドで選択できる唯一のモードです。接続の確立中にのみ、他の可能なモード **STATEMENT**、**ALTER USER**、または **ALTER DATABASE** を指定できます。そのモードでは、失敗した SQL ステートメントのみがロールバックされ、トランザクションは通常モードに戻されます。

autosave テストケース

autosave プロパティの値を試すためのテストケースは、**BatchAutoSaveTest.java** ファイルで利用できます。次の SQL コードは、サーバーが **transaction_rollback_scope** 機能を提供し、JDBC 側で **autosave=server** が使用される場合に予想される動作を示しています。

autosave=server を使用する場合、次のクエリーは値 (1)、(3)、および (4) を挿入し、**duplicate key violation** エラーを無視します。

```
CREATE TABLE test (id INT PRIMARY KEY);
INSERT INTO test VALUES (2);
BEGIN;
INSERT INTO test VALUES (1);
INSERT INTO test VALUES (2);
INSERT INTO test VALUES (3);
INSERT INTO test VALUES (4);
COMMIT;
```

artifacts ディレクトリには、**pgjdbc.jar** ファイル **postgresql-REL2Q.42.2.3.180601.jar** が含まれています。このファイルは、通常のように **CLASSPATH** に追加する必要があります。また、最新の **autosave** 機能をテストするために使用できる **postgresql-REL2Q.42.2.3.180601-tests.jar** jar も含まれています。

次のとおりに、**artifacts** で提供されている **BatchAutoSaveTest.java** ファイルをテストできます。

1. **CLASSPATH** をエクスポートしてテストケースをビルドおよび実行します。

```
cd artifacts
export CLASSPATH=$PWD/postgresql-REL2Q.42.2.3.180601-tests.jar:$PWD/postgresql-
→REL2Q.42.2.3.180601.jar:$PWD/junit-4.12.jar:$PWD/hamcrest-core-1.3.jar
```

1. 提供されたテストファイルをコンパイルします。

```
javac -d . BatchAutoSaveTest.java
```

1. テストを実行しますユーザーを `test` と仮定し、ローカルホストで実行しています。

```
java -Dusername=test -Dport=5432 -Dhost=localhost -Ddatabase=postgres org.junit.
→runner.JUnitCore org.postgresql.test.jdbc2.BatchAutoSaveTest
__OUTPUT__
JUnit version 4.12
.Configuration file /Users/altaf/pg/artifacts/./build.properties does not exist.
→Consider adding it to specify test db host and login
Configuration file /Users/altaf/pg/artifacts/./build.local.properties does not exist.
→ Consider adding it to specify test db host and login
Configuration file /Users/altaf/pg/artifacts/./build.properties does not exist.
→Consider adding it to specify test db host and login
Configuration file /Users/altaf/pg/artifacts/./build.local.properties does not exist.
→ Consider adding it to specify test db host and login
.....
Time: 0.556

OK (10 tests)
```

テストケースを変更するには、`artifacts` ディレクトリの `BatchAutoSaveTest.java` ファイルを変更できます。次に、テストケースをコンパイルして実行します。

5.3 Configuration parameters (GUCs)

これらの Grand Unified Configuration GUC 構成パラメーターは、EDB Postgres Extended Server で使用できます。

5.3.1 バックエンドパラメーター

バックエンドパラメーターは、PostgreSQL および拡張機能にスリープまたはエラーを注入するためのテストプローブポイントインフラストラクチャを導入します。

Postgres コード全体で定義された `PROBE_POINT` は、重要なコードパスをマークします。これらのプローブポイントは、現在のバックエンドに通知するか、`elog(...)` `LOG` / `ERROR` / `FATAL` / `PANIC` に信号を送信するためにアクティブになる場合があります。また、またはその代わりに、コードのその時点で遅延を追加する場合があります。

明示的にアクティブ化しない限り、プローブポイントは効果がなく、単一のオプティマイザーヒント付きブランチを追加するだけであるため、ホットパスでは安全です。

アクティブなプローブポイントがヒットし、カウンターが満たされると、指定されたスリープ間隔の後、ログメッセージは常に `DEBUG1` 以上で出力されます。

pg2q.probe_point

2ndQPostgres のコードまたは `PROBE_POINT` を定義する拡張機能内の `PROBE_POINT` の名前。このパラメーターは検証されていません。存在しないプローブポイントに名前が付けられている場合、ヒットすることはありません。

1つのプローブポイントのみをアクティブにできます。これはリストではなく、リストを提供しようとしても、一致するものがないことを意味します。

プローブポイントには通常、一意の名前があり、それが定義されているコード内の `PROBE_POINT` マクロへの引数として指定されます。複数のコードパスが目的の同じアクションをトリガーする場合、同じ `PROBE_POINT` 名前を使用することもできます。プローブは、いずれかのパスが選択されたときに起動します。

pg2q.probe_counter

このパラメーターで指定された回数ループが実行された後にのみ、プローブへの対応が必要になる場合があります。このような場合、この GUC をプローブポイントが点火する反復数に設定し、カウンターをリセットします。

デフォルト値は 1 で、名前が一致する場合にプローブポイントが常に起動することを意味します。

pg2q.probe_sleep

プローブポイントに到達した後、`pg2q.probe_sleep` ミリ秒間スリープします。次に、`pg2q.probe_action` でアクションを起動します。

pg2q.probe_action

名前付けた `pg2q.probe_point` がヒットしたときに実行するアクション。使用可能なアクションは次のとおりです。

- `sleep` — プローブ名を使用して `DEBUG` メッセージを送信します。
- `log` — プローブ名を使用して `LOG` メッセージを送信します。
- `error` — `elog(ERROR, ...)` `ERROR` 条件を発生させます。
- `fatal` — `elog(FATAL, ...)` .

- **panic** — `elog(PANIC, ...)`、通常は `abort()` を呼び出し、`SIGABRT` シグナル 6 を配信して、バックエンドにコアダンプを発生させます。プローブポイントは、ハード `ulimit` が許可する場合にコアファイル制限を設定してコアダンプを有効にしようとします。
- **sigint**、**sigterm**、**sigquit**、**sigkill** — 名前付けたシグナルを、プローブポイントにヒットしたバックエンドに配信します。

pg2q.probe_backend_pid

ゼロ以外の場合、プローブのスリープとアクションは、この ID のバックエンド以外のバックエンドでスキップされます。

5.3.2 server_2q_version_num および server_2q_version

`server_2q_version_num` および `server_2q_version` 構成パラメーターを使用すると、外部モジュールからそれぞれ 2ndQuadrant 固有のバージョン番号とバージョンサブ文字列にアクセスできます。

5.3.3 テーブルレベルの圧縮制御オプション

テーブルレベルオプション `compress_tuple_target` を設定して、タプルの圧縮をトリガーするタイミングを決定できます。以前は、`toast_tuple_target` またはコンパイル時のデフォルトを使用して、タプルを圧縮するかどうかを決定しました。ただし、これは、タプルが十分に大きく、優れた圧縮率を持っているが、トーストのしきい値を超えるほど十分に大きくない場合、有害でした。

5.3.4 pg2q.max_tuple_field_size

テーブルに書き込むことができるフィールドの内部表現の最大非圧縮サイズをバイト単位で制限します。

デフォルトの `pg2q.max_tuple_field_size` は 1073740799 バイトで、これは 1 GiB 未満の 1024 バイトです。この値は、通常 PostgreSQL によって課される 1 GiB の最大フィールドサイズよりもわずかに小さいです。このマージンは、タプルがディスクにコミットされているが、論理デコード出力プラグインで処理できず、ダウンストリームサーバーに送信できない場合を防ぐのに役立ちます。

機能を無効にするには、`pg2q.max_tuple_field_size` を 1GB または 11073741823 に設定します。

アプリケーションが大きなフィールドの挿入に依存していない場合は、`pg2q.max_tuple_field_size` を 100MB 以下などのはるかに小さい値に設定することを検討します。他の問題の中でも、大規模なフィールドでは次のことが可能です。

- 予期しないアプリケーション動作を引き起こす
- クエリーとレプリケーション中のデータベースエンジンのメモリ消費量の増加
- 論理レプリケーションの速度の低下

このパラメーターが有効になっている間、過大なフィールドにより、特大フィールドを INSERT または UPDATE するクエリーは次のような ERROR で失敗します。

```
ERROR: field big_binary_field_name in row is larger than pg2q.max_tuple_field_size
DETAIL: New or updated row in relation some_table has field big_binary_field_name
        (attno=2) with size 8161 bytes which exceeds limit 1073740799B configured
        in pg2q.max_tuple_field_size
SQLSTATE: 53400 configuration_limit_exceeded
```

スーパーユーザーのみが `pg2q.max_tuple_field_size` を設定できます。非スーパーユーザーが設定できるようにする場合は、`SECURITY DEFINER` ファンクションラッパーを使用できます。

`pg2q.max_tuple_field_size` を変更しても、既にディスク上にある現在の `pg2q.max_tuple_field_size` より大きいフィールドは変更されません。通常のように `SELECT` できます。特大のフィールドを含むタプルに影響を与える `UPDATE` は、更新操作によって作成された新しいタプルが現在アクティブなサイズ制限を満たさない限り、特大フィールドが変更されない場合でも失敗します。

`DELETE` オペレーションは、フィールドサイズ制限をチェックしません。

この制限は、フィールドの I/O のテキスト表現サイズに適用されません。そうすることにより、PostgreSQL が制限を超える一時的なメモリ内 json オブジェクトを作成および処理できないためです。

この制限は、セットを返すファンクション、CTE、ビューなど、タプルストアの一時的なタプルには適用されません。サイズチェックは `MATERIALIZED VIEW` にも意図的に適用されません。

警告: `pg_restore` には `pg2q.max_tuple_field_size` が適用されます。データベースに大きすぎるタプルが含まれている場合、通常のように `pg_dump` を実行します。ただし、後続の `pg_restore` は、前述のエラーで失敗します。この問題をワークアラウンドするには、`PGOPTIONS` または `options` 接続パラメーター文字列を使用して、接続オプションで `pg2q.max_tuple_field_size` をオーバーライドしてダンプを復元します。例

```
PGOPTIONS=-c pg2q.max_tuple_field_size=11073741823 pg_restore ...
```

データ型の詳細

- `bytea` フィールドの場合、使用されるサイズはデコードされたバイナリサイズです。これは、16 進数または 8 進数のエスケープ形式でのテキスト表現サイズ、つまりフィールドの `octet_length()` ではありません。

`bytea_output = 'hex'` と仮定すると、I/O 表現の最大サイズは $2 * \text{pg2q.max_tuple_field_size} + 2$ バイトです。

- `text`、`json`、または `xml` フィールドの場合、測定されたサイズは、文字数ではなく、現在のデータベースエンコードフィールドの `octet_length()` のテキストのバイト数です。UTF-8 エンコーディングでは、通常 1 文字は 1 バイトを消費しますが、一部の言語とスクリプトでは 6 バイト以上を消費する場合があります。
- `jsonb` フィールドの場合、測定されたサイズは、PostgreSQL の内部 `jsonb` エンコードされたデータ型表現、`json` ドキュメントのテキスト表現のサイズです。場合によっては、大きな `json` ドキュメント

の jsonb 表現はテキスト表現よりも小さい場合があります。これは、一般的ではありませんが、特定の `pg2q.max_tuple_field_size` より大きいテキスト表現を含む json ドキュメントを挿入できることを意味します。

- 拡張機能で定義されたデータ型の動作は、データ型の実装に依存します。

この制限に使用されるフィールドサイズは、`pg_column_size()` ファンクションによって報告されたサイズから、ターゲットデータ型のリテラルで使用される場合、PostgreSQL が可変長データ型に追加するヘッダーの 4 バイトを差し引いたサイズです。例

```
demo=> SELECT pg_column_size(BYTEA \x00010203040506070809) - 4;
14
```

たとえば、jsonb フィールドの計算されたサイズを表示するには、次を使用します。

```
SELECT pg_column_size(JSONB {"my_json_document": "yes"}) - 4;
```

TOAST 圧縮のため、`pg_column_size()` は、既存のディスク上のフィールドで呼び出されたときに、より小さい値を報告することがよくあります。また、ディスク上の短い値のヘッダーは 4 バイトではなく 1 バイトになる場合があります。

5.3.5 pg2q.max_tuple_size

テーブルに書き込むことができる単一のタプルの最大サイズを制限します。この値は、合計行幅であり、潜在的な圧縮可能または外部ストレージ対応フィールド値のすべての非圧縮幅を含みます。フィールドヘッダーはサイズに対してカウントされますが、固定行ヘッダーはカウントされません。

論理レプリケーションなどの多くの PostgreSQL 操作は、多くのアプリケーションと同様に、行全体で動作します。この設定を使用して、アプリケーションに合理的と考えられる最大行サイズに制限を課して、操作上の問題を引き起こす可能性のある大きすぎる行の誤って作成を防ぐことができます。

既存のタプルの UPDATE に適用される場合、`pg2q.max_tuple_size` は `pg2q.max_tuple_field_size` ほど厳密には適用されません。PLAIN 以外のストレージを使用する列の未変更の値のフルサイズはカウントしません。

警告: `pg_restore` には ``pg2q.max_tuple_size`` が適用されます。 `pg2q.max_tuple_field_size` の注意事項を参照してください。