



EDB Postgres Extended Server

リリース *15.3.0*

EnterpriseDB Corporation

2023 年 06 月 20 日

目次

第 1 章	Release notes	3
1.1	EDB Postgres Extended Server version 15.3	3
1.2	EDB Postgres Extended Server version 15.2	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) およびその派生物	7
3.1.2	Debian とその派生物	7
第 4 章	Transparent data encryption	15
第 5 章	Replication	17
5.1	非同期処理	17
第 6 章	Configuration parameters (GUCs)	19
6.1	バックエンドパラメーター	19
6.1.1	pg2q.probe_point	19
6.1.2	pg2q.probe_counter	20
6.1.3	pg2q.probe_sleep	20
6.1.4	pg2q.probe_action	20
6.1.5	pg2q.probe_backend_pid	20
6.2	server_2q_version_num および server_2q_version	21
6.3	テーブルレベルの圧縮制御オプション	21
6.4	pg2q.max_tuple_field_size	21
6.5	pg2q.max_tuple_size	23
第 7 章	SQL enhancements	25
7.1	ロールバックオプション	25
7.2	準備されたステートメントを含むカーソル	25
7.3	PL / pgSQL の互換性	26
7.3.1	transaction_rollback_scope parameter	26
7.3.2	JDBC properties for setting rollback scope	27
第 8 章	Operations	31
8.1	トランザクションログのフラッシュを避ける	31
8.2	追加のトレースと診断のオプション	31
8.3	選択的な物理ベースのバックアップとそれに続く選択的なリカバリ/リストア	31
8.4	追加操作機能	32

EDB Postgres Extended Server は、オープンソースのコミュニティ PostgreSQL 上に構築された Postgres データベースサーバーディストリビューションです。PostgreSQL と完全に互換性があります。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、それらは EDB Postgres Extended Server でも同じように動作します。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、 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られた数の機能で PostgreSQL を拡張し、他の点でパリティを維持することです。

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 *SQL superset*
- *WAL pacing delays to avoid flooding transaction log*
- 追加のトレースと診断のオプション

第1章 Release notes

EDB Postgres Extended Server のドキュメントには、マイナーリリースとパッチを含む EDB Postgres Extended Server 15 の最新バージョンが記載されています。これらのリリースノートには、各リリースの新機能が記載されています。

Version	Release date
<i>EDB Postgres Extended Server version 15.3</i>	2023 May 11
<i>EDB Postgres Extended Server version 15.2</i>	2023 Feb 14

1.1 EDB Postgres Extended Server version 15.3

EDB Postgres Extended Server 15.2 の新機能、機能強化、バグ修正、およびその他の変更には以下が含まれます。

Type	Description	Category
Upstream merge	コミュニティ PostgreSQL 15.3 と合併しました。詳細については、 Transparent data encryption を参照してください。	
Enhancement	SQL Profiler と Index Advisor が拡張機能になり、 Transparent data encryption からダウンロードできます。	

1.2 EDB Postgres Extended Server version 15.2

EDB Postgres Extended Server 15.2 の新機能、機能強化、バグ修正、およびその他の変更には以下が含まれます。

Type	Description
Upstream merge	コミュニティ PostgreSQL 15.2 と合併しました。詳細については、 Transparent data encryption を参照してください。

次のページに続く

表 3 – 前のページからの続き

Type	Description
Feature	<i>Transparent data encryption</i> のサポート。TDE は、データベースシステムに格納されているユーザーデータをユーザーに対して透過的に暗号化します。

第2章 Deployment options

展開オプションは次のとおりです。

- ネイティブパッケージを使用した仮想マシンまたは物理サーバー上の *Installing EDB Postgres Extended Server on Linux*
- Trusted Postgres Architect を使用して EDB Postgres Distributed で展開
- 極端な高可用性クラスタータイプで BigAnimal にデプロイ

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) およびその派生物

- *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) 8*、*Oracle Linux (OL) 7*
- *Installing EDB Postgres Extended Server on AlmaLinux 8 or Rocky Linux 8 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 とその派生物

- *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)

リポジトリへのアクセスを含む、オペレーティングシステム固有のインストール手順については、以下を参照してください。

Red Hat Enterprise Linux (RHEL) およびその派生物

- *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) 8*
- *Oracle Linux (OL) 7*
- *Installing EDB Postgres Extended Server on AlmaLinux 8 or Rocky Linux 8 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 とその派生物

- *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 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：

- リポジトリをセットアップする

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

リポジトリをセットアップするには、

EDB repositories に移動し、そこに記載されている指示に従います。

- 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 15 をインストールする場合、パッケージ名は edb-postgresextended15-server edb-postgresextended15-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：

- リポジトリをセットアップする

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

リポジトリをセットアップするには、

[EDB repositories](#) に移動し、そこに記載されている指示に従います。

- 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 15 をインストールする場合、パッケージ名は edb-postgresextended15-server edb-postgresextended15-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：

- リポジトリをセットアップする

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

リポジトリをセットアップするには、

EDB repositories に移動し、そこに記載されている指示に従います。

- 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 15 をインストールする場合、パッケージ名は edb-postgresextended15-server edb-postgresextended15-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：

- リポジトリをセットアップする

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

リポジトリをセットアップするには、

EDB repositories に移動し、そこに記載されている指示に従います。

- 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 15 をインストールする場合、パッケージ名は edb-postgresextended15-server edb-postgresextended15-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：

- リポジトリをセットアップする

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

リポジトリをセットアップするには、

[EDB repositories](#) に移動し、そこに記載されている指示に従います。

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 15 をインストールする場合、パッケージ名は edb-postgresextended-15 edb-postgresextended-15-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：

- リポジトリをセットアップする

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

リポジトリをセットアップするには、

[EDB repositories](#) に移動し、そこに記載されている指示に従います。

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 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：

- リポジトリをセットアップする

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

リポジトリをセットアップするには、

[EDB repositories](#) に移動し、そこに記載されている指示に従います。

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 11 x86_64

前提条件

インストールプロセスを開始する前に：

- リポジトリをセットアップする

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

リポジトリをセットアップするには、

[EDB repositories](#) に移動し、そこに記載されている指示に従います。

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：

- リポジトリをセットアップする

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

リポジトリをセットアップするには、

[EDB repositories](#) に移動し、そこに記載されている指示に従います。

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 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。

第5章 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 Once (CAMO)
- グループコミット
- 熱心な複製
- 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 を使用したテーブルで UPDATE トリガーは必要ありません
- FREEZE を自動でホールド

5.1 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 `async` は、利用可能なスタンバイサーバーが十分でない場合に非同期処理を有効にします (`synchronous_standby_names` による値と比較した場合)。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6章 Configuration parameters (GUCs)

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

6.1 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、スリープまたはエラーを PostgreSQL と拡張機能に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信号を送信するか、`LOG /ERROR /FATAL /PANIC` を `elog(...)` するためにアクティブ化される場合があります。また、または代わりに、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されない限り、プローブポイントは効果がなく、オプティマイザーヒント付きの単一のブランチのみを追加するため、ホットパスで安全です。

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た場合、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。

6.1.1 `pg2q.probe_point`

2ndQPostgres のコード内、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た場合、ヒットすることはありません。

アクティブにできるプローブポイントは1つだけです。これはリストではありません。リストを提供しようとしても、一致するものではありません。

プローブポイントには通常、一意の名前が付けられます。これは、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。いずれかのパスが取られると、プローブが起動します。

6.1.2 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、プローブを操作する必要がある場合があります。このような場合、この GUC をプローブポイントが発砲する反復回数に設定し、カウンターをリセットします。

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

6.1.3 pg2q.probe_sleep

プローブポイントにヒットした後、pg2q.probe_sleep ミリ秒スリープします。次に、pg2q.probe_action でアクションを起動します。

6.1.4 pg2q.probe_action

名前付き 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** — プローブ名を指定して DEBUG メッセージを発行します。
- **log** — プローブ名を指定して LOG メッセージを発行します。
- **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（信号 6）を配信して、バックエンドをコアダンプします。プローブポイントは、ハード `ulimit` が許可する場合、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** : 名前付き信号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。

6.1.5 pg2q.probe_backend_pid

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

6.2 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、それぞれ 2ndQuadrant 固有のバージョン番号とバージョン部分文字列に外部モジュールからアクセスできます。

6.3 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ていました。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、トーストしきい値を超えるには十分大きくない場合に有害でした。

6.4 pg2q.max_tuple_field_size

テーブルに書き込むことができる 1 つの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、1GiB 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たが、論理デコード出力プラグインで処理してダウンストリームサーバーに送信できないケースを防ぐのに役立ちます。

`pg2q.max_tuple_field_size` を 1GB または 11073741823 に設定して、機能を無効にします。

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、はるかに小さな値に設定することを検討してください。とりわけ、大規模なフィールドは次のことができます。

- 驚くべきアプリケーションの動作を引き起こす
- クエリ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を増やす
- 論理レプリケーションを遅くする

このパラメーターが有効になっている間、特大のフィールドにより、特大のフィールドを INSERT または UPDATE するクエリが次のような 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
```

`pg2q.max_tuple_field_size` を設定できるのはスーパーユーザーだけです。非スーパーユーザーによる設定を許可する場合は、SECURITY DEFINER 関数ラッパーを使用できます。

`pg2q.max_tuple_field_size` を変更した場合、すで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。いつものように SELECT できま

す。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、特大のフィールドが変更されなくても、特大のフィールドを持つタプルに影響を与える UPDATE は失敗します。

DELETE 操作は、フィールドサイズの制限をチェックしません。

制限は、フィールドの I/O のテキスト表現サイズに強制されません。そうすると、PostgreSQL が制限よりも大きな一時的なインメモリ 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、セットを返す関数、CTE、ビューなど、タプルストア内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** に対しても意図的に強制されません。

警告: `pg2q.max_tuple_field_size` は `pg_restore` に適用されます。データベースに特大のタプルが含まれている場合、通常どおり `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。これ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 進または 8 進のエスケープ形式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'` を想定すると、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、測定サイズは、文字数ではなく、現在のデータベースエンコーディング（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、言語やスクリプトによっては 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、測定サイズは、PostgreSQL 内部の **jsonb** エンコードデータ型表現、**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** 表現がテキスト表現よりも小さくなります。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持つ **json** ドキュメントを挿入できることを意味します。
- 拡張定義データ型の動作は、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()` 関数によって報告されるサイズから、ターゲットデータ型のリテラルで使われる場合、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。

6.5 `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、潜在的に圧縮可能または外部ストレージ対応のすべてのフィールド値の非圧縮幅を含む合計行幅です。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、行全体で機能します。この設定を使用して、アプリケーションにとって妥当と考える最大行サイズに制限を課し、操作上の問題を引き起こす可能性のある特大の行を誤って作成するのを防ぐことができます。

既存のタプルの `UPDATE` に適用すると、`pg2q.max_tuple_size` は `pg2q.max_tuple_field_size` ほど厳密には強制されません。PLAIN 以外のストレージを持つ列の変更されていない値のフルサイズはカウントしません。

警告: `pg2q.max_tuple_size` は `pg_restore` に適用されます。 `pg2q.max_tuple_field_size` の警告を参照してください。

第7章 SQL enhancements

EDB Postgres Extended Server には、多数の SQL 拡張機能が含まれています。

7.1 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、Oracle や SQL Server などの他の DBMS とは異なります。エラーにより、最後のステートメントのみがロールバックされます。トランザクション処理のセマンティクスのこの違いは、すべての場合に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 にビジネスロジックを実装することは困難です。

1つの回避策は、内部的にサブトランザクションと呼ば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、テストするのが困難です。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待機時間が長くなります。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えると、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 を使用すると、現在のステートメントだけ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。手動での記録は必要ありません。オーバーヘッドがいくらか追加されますが、セーブポイントよりも低くなります。

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、JDBC バッチジョブで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。

7.2 準備されたステートメントを含むカーソル

EDB Postgres Extended Server を使用すると、以前に作成した準備済みステートメントにカーソルを宣言できます。

例：

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

7.3 PL / pgSQL の互換性

EDB Postgres Extended Server は、多数の PL / pgSQL 互換性機能を備えた他の移行ツールと統合されています。

一般的に簡単にするために、EDB Postgres Extended Server では、PERFORM キーワードなしで `plpgsql` を使用して関数を呼び出すことができます。たとえば、

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

`somefunc` はキーワードではありません。

7.3.1 transaction_rollback_scope parameter

データベース内で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、次の2つの値があります。

- **transaction** — 各エラーがトランザクション全体を中止する標準の Postgres の動作。
- **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。

パラメータの設定

パラメーターは、ユーザーレベルのプロパティ、接続オプション、または特定の関数や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する

```
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>
```

特定の機能または手順のモードを設定する

PL/pgSQL を使用している場合、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` 値を選択すると、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。コマンドで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正常に続行できます。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のみに影響し、トランザクション全体のステータスには影響しません。

サブトランザクションをコミットすると、保持しているリソースがその親トランザクション（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）にのみ割り当てられます。または、ユーザー定義のセーブポイントが関係する場合、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。

7.3.2 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` への追加の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。

自動保存

`autosave` パラメーターは、複数のステートメントを含むクエリが失敗した場合のドライバーの動作を指定する文字列です。可能な値は次のとおりです。 `server`、`always`、`never`、および `conservative`。

- `autosave=server` モードでは、JDBC はサーバー側パラメーター `transaction_rollback_scope` に依存して、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側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` プロパティの使用を試みます。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、失敗した場合は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ません」または「ステートメント XXX が無効です」などのまれな場合にのみ行われるため、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とき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。接続の確立時のみ、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ります。

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、`autosave=server` が JDBC 側で 사용되는場合に予想される動作を示しています。

`autosave=server` を使用して、次のクエリは値 (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、通常どおり `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。

`artifacts` で提供される `BatchAutoSaveTest.java` ファイルは、次のようにテストできます。

1. `CLASSPATH` をエクスポートして、テストケースをビルドして実行します。

```
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` とし、ローカルホストで実行していると仮定します）。

```
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` ディレクトリにある `BatchAutoSaveTest.java` ファイルを変更します。次に、テストケースをコンパイルして実行します。

第8章 Operations

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

8.1 トランザクションログのフラッシュバックを避ける

EDB Postgres Extended Server は、トランザクションログのフラッシュバックを回避するために WAL ページング遅延を提供します。WAL ページ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、セッションは少なくとも `wal_insert_delay_size` の値の WAL データが生成された後、`wal_insert_delay` の値に基づいてスリープします。デフォルトはオフです。

8.2 追加のトレースと診断のオプション

EDB Postgres Extended Server を使用すると、特定のコードパスのトレースメッセージのログギング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、指定した時間のタイムアウトが発生した後にトレースメッセージをログに記録できるようにします。

8.3 選択的な物理ベースのバックアップとそれに続く選択的なリカバリ/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対して `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、依然として 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回復も通常どおりに実行されます。除外されたデータ

ベースに属する WAL データは、回復プロセス中に無視されます。リカバリー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。

8.4 追加操作機能

- ALTER TABLE ... REPLICA IDENTITY のロックを削減