



EDB

Postgres® for the AI Generation

pglogical 2

リリース 2.4.3

EnterpriseDB UK

2024 年 07 月 30 日

目次

第 1 章	Release Notes	3
1.1	pglogical 2.4.3	3
1.1.1	変更	3
1.2	pglogical 2.4.2	3
1.2.1	変更	3
1.3	pglogical 2.4.1	3
1.3.1	変更	4
1.4	pglogical 2.4.0	4
1.4.1	変更	4
第 2 章	Requirements	5
第 3 章	Installation	7
3.1	パッケージ	7
3.1.1	YUM を使用した pglogical のインストール	7
3.1.2	前提条件	7
3.1.3	インストール	8
3.1.4	APT を使用した pglogical のインストール	9
3.1.5	前提条件	9
3.1.6	インストール	9
3.2	ソースコードから	10
第 4 章	Usage	11
4.1	クイックセットアップ	11
4.2	ベースバックアップを使用してサブスクライバノードの作成	12
4.3	ノード管理	13
4.4	サブスクリプション管理	13
4.5	レプリケーションセット	17
4.5.1	新しいテーブルのレプリケーションセットの自動割り当て	19
4.6	追加機能	20
4.7	行フィルタリング	20
4.7.1	プロバイダーの行フィルタリング	20
4.7.2	サブスクライバーの行フィルタリング	21
第 5 章	Synchronous Replication	23
第 6 章	Conflicts	25
第 7 章	Configuration options	27
第 8 章	Limitations and restrictions	29
8.1	スーパーユーザーが必要です	29

8.2	UNLOGGED および TEMPORARY はレプリケートされません	29
8.3	一度に 1 つのデータベース	29
8.4	PRIMARY KEY または REPLICATION IDENTITY が必要です	29
8.5	一意のインデックス/コンストレイン/PK は 1 つだけ	29
8.6	一意のコンストレインは遅延可能であってはなりません	30
8.7	DDL	30
8.8	レプリケーションキューのフラッシュなし	30
8.9	外部キー	31
8.10	TRUNCATE	31
8.11	シークエンス	31
8.12	トリガー	31
8.13	PostgreSQL バージョンの違い	32
8.14	データベースエンコーディングの違い	32
8.15	ラージオブジェクト	32
8.16	Postgres-XL	32

pglogical 2 拡張機能は、パブリッシュ/サブスクライバモデルを使用して、PostgreSQL の論理ストリーミングレプリケーションを提供します。これは、EDB Postgres Distributed の一部として開発されたテクノロジーに基づいています。

私たちは次の用語を使用してノード間のデータストリームを説明し、以前の Slony テクノロジーから意図的に再利用しました。

- ノード - PostgreSQL データベースインスタンス
- プロバイダーとサブスクライバー - ノードが実行するロール
- レプリケーションセット - テーブルのコレクション

pglogical は最新のインコア機能を利用しているため、次のバージョン制限があります。

- プロバイダーとサブスクライバーノードは PostgreSQL 9.4+を実行する必要があります
- レプリケーションオリジンフィルタリングと競合検出には、PostgreSQL 9.5 以降が必要です
- さらに、サブスクライバーは Postgres-XL 9.5+にすることができます

サポートされているユースケースは次のとおりです。

- メジャーバージョン間のアップグレード上記の制限を考慮
- データベースの完全なレプリケーション
- レプリケーションセットを使用したテーブルのセットの選択的レプリケーション
- パブリッシャーまたはサブスクライバー側のテーブル行の選択的レプリケーション `row_filter`
- パブリッシャー側でのテーブル列の選択的レプリケーション
- 複数の上流サーバーからのデータ収集/マージ

アーキテクチャの詳細

- pglogical は、物理的ストリーミングレプリケーションのようなサーバー全体レベルではなく、データベースごとのレベルで動作します
- 1つのプロバイダーが、追加のディスク書き込みオーバーヘッドを発生させることなく、マルチプルのサブスクライバーをフィードできます
- 1つのサブスクライバーは、複数のオリジンからの変更をマージし、自動で構成可能な競合解決マルチマスターに必要なすべてではなく一部の側面を使用して変更間の競合を検出できます。
- カスケードレプリケーションは、変更セット転送の形式で実装されます。

第1章 Release Notes

1.1 pglogical 2.4.3

バージョン 2.4.3 は、pglogical 2 のメンテナンスリリースです。

1.1.1 変更

- 初期同期中に正しいタプルにデータフィルタリングを適用します。
- 変更のデコード中に、正しいメモリコンテキストを復元します。
- PostgreSQL 15 では、データベースのドロップが完了しません。
- TRUNCATE をグローバルメッセージとして複製しないでください。

1.2 pglogical 2.4.2

バージョン 2.4.2 は、pglogical 2 のメンテナンスリリースです。

1.2.1 変更

- PostgreSQL 15 のサポートを追加します。

1.3 pglogical 2.4.1

バージョン 2.4.1 は、pglogical 2 のメンテナンスリリースです。

1.3.1 変更

- 「スナップショットはまだアクティブです」警告とドメインでのスナップショットの処理を修正します。バージョン 2.4.0 で発生した問題。
- リモートレプリケーションスロットのドロップに失敗した場合のフラッシュエラー状態

1.4 pglogical 2.4.0

バージョン 2.4.0 は、pglogical 2 のメンテナンスリリースです。

1.4.1 変更

- PostgreSQL 14 ベータの予備サポートを追加
- 行セットではなく行を返すように `pglogical_show_subscription_table` を修正
- 出力プラグインと適用ワーカーのスナップショット処理を修正
- Windows でコマンドの引用符を修正して、実際に動作するように以前に使用した API にはコマンドライン長に 1024 制限があるようです
- Windows でスキーマ同期プロセスを中断できることを確認します
- Windows での `pglogical_create_subscriber` のコンパイルの問題を修正
- 適用ワーカーで BEFORE ROW DELETE トリガーが NULL を返した場合のリレーシヨンのダブルクローズを修正
- SPI 適用ワーカーのマルチインサートクラッシュを修正
- 適用ワーカーで最後のタプルの挿入を複数回実行するマルチインサートを修正
- `debug_query_string` が常に設定されていることを確認します PostgreSQL の新しいバージョンでは、`debug_query_string` を常に設定する必要があります。

第2章 Requirements

pglogical を使用するには、プロバイダーとサブスクライバーが PostgreSQL 9.4 以降を実行している必要があります。

pglogical 拡張機能は、プロバイダーとサブスクライバーの両方にインストールする必要があります。両方で `CREATE EXTENSION pglogical` する必要があります。

プロバイダーとサブスクライバーのテーブルは同じ名前で、同じスキーマ内にある必要があります。将来のバージョンでは、マッピング機能が追加される可能性があります。

プロバイダーとサブスクライバーのテーブルには同じ列があり、各列のデータ型が同じである必要があります。CHECK 制約、NOT NULL 制約などは、プロバイダーよりもサブスクライバーで同じか弱いより許容的である必要があります。

テーブルには同じ PRIMARY KEY s が必要です。PRIMARY KEY 以外の UNIQUE コンストレインを追加することはお勧めできません。以下を参照してください。

一部の追加要件は *Limitations and restrictions* でカバーされています。

第3章 Installation

3.1 パッケージ

pglogical は、Fedora、CentOS、および RHEL の場合は yum を介して RPM として、Debian および Ubuntu の apt を介して DEB として、またはここにソースコードとして利用できます。ソースからインストールする手順については、以下を参照してください。

3.1.1 YUM を使用した pglogical のインストール

以下の手順は、Red Hat ファミリーのオペレーティングシステム RHEL、CentOS、Fedora に有効です。前提条件

3.1.2 前提条件

これらの RPM はすべて、<http://yum.postgresql.org/> の PGDG PostgreSQL リリースを必要とします。Fedora および RHEL に含まれるストックの PostgreSQL リリースでは使用できません。PostgreSQL をまだお持ちでない場合

- <http://yum.postgresql.org/repopackages.php> から適切な PGDG リポジトリ rpm をインストールします。
- Install PostgreSQL
- PostgreSQL 9.4: `yum install postgresql94-server postgresql94-contrib`
- PostgreSQL 9.5: `yum install postgresql95-server postgresql95-contrib`
- PostgreSQL 9.6: `yum install postgresql96-server postgresql96-contrib`
- PostgreSQL 10: `yum install postgresql10-server postgresql10-contrib`
- PostgreSQL 11: `yum install postgresql11-server postgresql11-contrib`
- PostgreSQL 12: `yum install postgresql12-server postgresql12-contrib`
- PostgreSQL 13: `yum install postgresql13-server postgresql13-contrib`
- PostgreSQL 14: `yum install postgresql14-server postgresql14-contrib`
- PostgreSQL 15: `yum install postgresql15-server postgresql15-contrib`

次に、宛先の Linux サーバーで root として次の手順を実行して、PostgreSQL バージョンの「2ndQuadrant の General Public」リポジトリをインストールします。

- PostgreSQL 9.4 `curl https://techsupport.enterprisedb.com/api/repository/dl/default/release/9.4/rpm | bash`
- PostgreSQL 9.5 `curl https://techsupport.enterprisedb.com/api/repository/dl/default/release/9.5/rpm | bash`
- PostgreSQL 9.6 `curl https://techsupport.enterprisedb.com/api/repository/dl/default/release/9.6/rpm | bash`
- PostgreSQL 10 `curl https://techsupport.enterprisedb.com/api/repository/dl/default/release/10/rpm | bash`
- PostgreSQL 11 `curl https://techsupport.enterprisedb.com/api/repository/dl/default/release/11/rpm | bash`
- PostgreSQL 12 `curl https://techsupport.enterprisedb.com/api/repository/dl/default/release/12/rpm | bash`
- PostgreSQL 13 `curl https://techsupport.enterprisedb.com/api/repository/dl/default/release/13/rpm | bash`
- PostgreSQL 14 `curl https://techsupport.enterprisedb.com/api/repository/dl/default/release/14/rpm | bash`
- PostgreSQL 15 `curl https://techsupport.enterprisedb.com/api/repository/dl/default/release/15/rpm | bash`

3.1.3 インストール

リポジトリがインストールされたら、PostgreSQL バージョンの pglogical に進むことができます。

- PostgreSQL 9.4 `yum install postgresql94-pglogical`
- PostgreSQL 9.5 `yum install postgresql95-pglogical`
- PostgreSQL 9.6 `yum install postgresql96-pglogical`
- PostgreSQL 10 `yum install postgresql10-pglogical`
- PostgreSQL 11 `yum install postgresql11-pglogical`
- PostgreSQL 12 `yum install postgresql12-pglogical`
- PostgreSQL 13 `yum install postgresql13-pglogical`
- PostgreSQL 14 `yum install postgresql14-pglogical`
- PostgreSQL 15 `yum install postgresql15-pglogical`

パッケージ署名用のリポジトリ GPG キーを受け入れるように求められる場合があります。

```
Retrieving key from file:///etc/pki/rpm-gpg/RPM-GPG-KEY-2NDQ-DL-DEFAULT Importing GPG
→key 0xD6BAF0C3: Userid : "Public repository signing key 2ndQuadrant <ci@2ndquadrant.
```

(次のページに続く)

(前のページからの続き)

```
→com>" Fingerprint: 8565 305c ea7d 0b66 4933 d250 9904 cd4b d6ba f0c3 From : /etc/  
→pki/rpm-gpg/RPM-GPG-KEY-2NDQ-DL-DEFAULT Is this ok [y/N]:
```

その場合は、y を押してキーを受け入れてから Enter します。2ndQuadrant マスターパッケージキーで署名されていますそれを確認したい場合。

3.1.4 APT を使用した pglogical のインストール

以下の手順は、Debian および Debian をベースとしたすべての Linux フレーバー Ubuntu などに有効です。

3.1.5 前提条件

宛先の Linux サーバーで root として次の指示を実行することにより、“2ndQuadrant の General Public” リポジトリをインストールできます。curl <https://techsupport.enterprisedb.com/api/repository/dl/default/release/deb> | bash

- <http://apt.postgresql.org/> リポジトリを追加します。手順についてはサイトを参照してください。

3.1.6 インストール

前提条件が完了したら、pglogical のインストールは、ご使用の PostgreSQL のバージョンで次を実行するだけです。

- PostgreSQL 9.4 `sudo apt-get install postgresql-9.4-pglogical`
- PostgreSQL 9.5 `sudo apt-get install postgresql-9.5-pglogical`
- PostgreSQL 9.6 `sudo apt-get install postgresql-9.6-pglogical`
- PostgreSQL 10 `sudo apt-get install postgresql-10-pglogical`
- PostgreSQL 11 `sudo apt-get install postgresql-11-pglogical`
- PostgreSQL 12 `sudo apt-get install postgresql-12-pglogical`
- PostgreSQL 13 `sudo apt-get install postgresql-13-pglogical`
- PostgreSQL 14 `sudo apt-get install postgresql-14-pglogical`
- PostgreSQL 15 `sudo apt-get install postgresql-15-pglogical`

3.2 ソースコードから

ソースコードのインストールは、PGXS を使用してビルドされた他の PostgreSQL 拡張機能の場合と同じです。

PostgreSQL リリースの `pg_config` を含むディレクトリが `PATH` 環境変数にリストされていることを確認します。 `pg_config` がない場合は、パッケージマネージャーから PostgreSQL リリースの `-dev` または `-devel` パッケージをインストールする必要がある場合があります。

次に、`make` を実行してコンパイルし、`make install` を実行してインストールします。インストール手順では `sudo` の使用が必要になる場合があります。

例 PostgreSQL の yum.postgresql.org パッケージを使用していると仮定すると、一般的な Fedora または RHEL 7 インストールの場合。

```
sudo dnf install postgresql95-devel
PATH=/usr/pgsql-9.5/bin:$PATH make clean all
sudo PATH=/usr/pgsql-9.5/bin:$PATH make install
```

第4章 Usage

このセクションでは、pglogical レプリケーション拡張機能の基本的な使用法について説明します。

4.1 クイックセットアップ

最初に、論理デコードをサポートするように PostgreSQL サーバーを適切に構成する必要があります。

```
wal_level = logical
max_worker_processes = 10  # one per database needed on provider node
                             # one per node needed on subscriber node
max_replication_slots = 10 # one per node needed on provider node
max_wal_senders = 10       # one per node needed on provider node
shared_preload_libraries = pglogical
```

PostgreSQL 9.5+これは 9.4 では動作しませんを使用しており、最後/最初の更新の勝利で競合解決を処理する場合 [Conflicts](#) を参照してください、この追加オプションを postgresql.conf に追加できます。

```
track_commit_timestamp = on # needed for last/first update wins conflict resolution
                             # property available in PostgreSQL 9.5+
```

pg_hba.conf は、ローカルホストからの論理レプリケーション接続を許可する必要があります。PostgreSQL 9.6 までは、論理レプリケーション接続は pg_hba.conf の replication キーワードを使用して管理されていました。PostgreSQL 10 以降では、論理レプリケーション接続は、プロバイダーデータベースへの通常の接続として pg_hba.conf によって処理されます。

次に、pglogical 拡張機能をすべてのノードにインストールする必要があります。

```
CREATE EXTENSION pglogical;
```

PostgreSQL 9.4 を使用する場合、pglogical_origin 拡張機能もそのノードにインストールする必要があります。

```
CREATE EXTENSION pglogical_origin;
```

次に、プロバイダーノードを作成します。

```
SELECT pglogical.create_node(
    node_name := provider1,
    dsn := host=providerhost port=5432 dbname=db
);
```

public スキーマのすべてのテーブルを default レプリケーションセットに追加します。

```
SELECT pglogical.replication_set_add_all_tables(default, ARRAY[public]);
```

オプションで、追加のレプリケーションセットを作成し、テーブルを追加することもできます。[レプリケーションセット](#)を参照してください。

通常、サブスクライブする前にレプリケーションセットを作成して、単一の初期トランザクションでの初期レプリケーションセットアップ中にすべてのテーブルが同期されるようにすることをお勧めします。ただし、大規模なデータベースのユーザーは、代わりに、より適切な制御のためにそれらをインクリメンタルに作成することを希望する場合があります。

プロバイダーノードがセットアップされると、サブスクライバーはそれにサブスクライブできます。最初にサブスクライバノードを作成する必要があります。

```
SELECT pglogical.create_node(
    node_name := subscriber1,
    dsn := host=thishost port=5432 dbname=db
);
```

そして最後に、サブスクライバーノードで、バックグラウンドで同期とレプリケーションプロセスを開始するサブスクリプションを作成できます。

```
SELECT pglogical.create_subscription(
    subscription_name := subscription1,
    provider_dsn := host=providerhost port=5432 dbname=db
);

SELECT pglogical.wait_for_subscription_sync_complete(subscription1);
```

4.2 ベースバックアップを使用してサブスクライバノードの作成

SQL レベルのノードとサブスクリプションの作成に加えて、pglogical は、pg_basebackup でプロバイダーのクローンを作成し、pglogical サブスクライバーとして起動することによるサブスクライバーの作成もサポートしています。これは、pglogical_create_subscriber ツールを使用して実行されます。--help 出力を参照してください。

pglogical.create_subscription のデータ同期オプションとは異なり、このクローンはレプリケーションセットを無視し、すべてのデータベースのすべてのテーブルをコピーします。ただし、多くの場合、特に高帯域幅リンクでは、はるかに高速です。

4.3 ノード管理

SQL インターフェイスを使用して、ノードを動的に追加および削除できます。

- `pglogical.create_node(node_name name, dsn text)` ノードを作成します。

パラメーター

- `node_name` -新しいノードの名前、データベースごとに1つのノードのみが許可されます
- `dsn` -ノードへの接続文字列、プロバイダーであるノードの場合、これは外部から到達可能である必要があります
- `pglogical.drop_node(node_name name, ifexists bool)` pg 論理ノードを削除します。

パラメーター

- `node_name` -既存のノードの名前
- `ifexists` -true の場合、サブスクリプションが存在しない場合にエラーはスローされません、デフォルトは false
- `pglogical.alter_node_add_interface(node_name name, interface_name name, dsn text)` ノードにインターフェイスを追加します。

ノードが作成されると、そのインタフェースも `create_node` で指定された `dsn` とノードと同じ名前で作成されます。このインターフェイスを使用すると、さまざまな接続文字列を使用して代替インターフェイスを既存のノードに追加できます。

パラメーター

- `node_name` -既存のノードの名前
- `interface_name` -追加する新しいインターフェイスの名前
- `dsn` -新しいインターフェイスに使用されるノードへの接続文字列
- `pglogical.alter_node_drop_interface(node_name name, interface_name name)` 既存のインターフェイスをノードから削除します。

パラメーター

- `node_name` - 既存のノードの名前
- `interface_name` - 既存のインターフェイスの名前

4.4 サブスクリプション管理

- `pglogical.create_subscription(subscription_name name, provider_dsn text, replication_sets text[], synchronize_structure boolean, synchronize_data boolean, forward_origins text[], apply_delay interval)` 現在のノードからプロバイダーノードへのサブスクリプションを作成します。コマンドはブロックせず、アクションを開始するだけです。

パラメーター

- `subscription_name` -サブスクリプションの名前、一意である必要があります
- `provider_dsn` -プロバイダーへの接続文字列
- `replication_sets` -サブスクライブするレプリケーションセットの配列、これらは既に存在する必要があります、デフォルトは” {default,default_insert_only,ddl_sql}”
- `synchronize_structure` -指定しますプロバイダーからサブスクライバーに構造を同期する場合、デフォルト false
- `synchronize_data` -プロバイダーからサブスクライバーにデータを同期するかどうかを指定します、デフォルト true
- `forward_origins` -転送するオリジン名の配列、現在サポートされている値のみが空の配列であり、変更を転送しないプロバイダーノードで発生したものではないこれはノード間の双方向レプリケーションに役立ちます、または「{all}」つまり起点が何であるかに関係なくすべての変更をレプリケートします。デフォルトは「{all}」
- `apply_delay` -レプリケーションをどれだけ遅延するか、デフォルトは 0 秒です
- `force_text_transfer` - プロバイダーにテキストを使用してすべての列をレプリケートするように強制します表現速度は遅くなりますが、サブスクライバでレプリケートされた列の型を変更するために使用される場合があります、デフォルトは false です

`subscription_name` は、レプリケーション接続によって `application_name` として使用されます。これは、`pg_stat_replication` モニタリングビューに表示されることを意味します。pglogical が *Synchronous Replication* セットアップの一部として使用される場合、`synchronous_standby_names` で使用することもできます。

`pglogical.wait_for_subscription_sync_complete(sub_name)` を使用して、サブスクリプションが非同期でレプリケートを開始し、必要なスキーマおよび/またはデータの同期を完了するまで待機します。

- `pglogical.drop_subscription(subscription_name name, ifexists bool)` サブスクリプションを切断し、カタログから削除します。

パラメーター

- `subscription_name` -既存のサブスクリプションの名前
- `ifexists` -true の場合、サブスクリプションが存在しない場合にエラーはスローされません、デフォルトは false
- `pglogical.alter_subscription_disable(subscription_name name, immediate bool)` サブスクリプションを無効にし、プロバイダーから切断します。

パラメーター

- `subscription_name` -既存のサブスクリプションの名前
- `immediate` -true の場合、サブスクリプションはすぐに停止されます。それ以外の場合、現在のトランザクションの終了時にのみ停止されます。デフォルトは false
- `pglogical.alter_subscription_enable(subscription_name name, immediate bool)` 無効なサブスクリプションを有効にします。

パラメーター

- `subscription_name` -既存のサブスクリプションの名前
- `immediate` -true の場合、サブスクリプションはすぐに開始されます。それ以外の場合、現在のトランザクションの終了時にのみ開始されます。デフォルトは false
- `pglogical.alter_subscription_interface(subscription_name name, interface_name name)` 別のインターフェイスを使用してプロバイダーノードに接続するようにサブスクリプションを切り替えます。

パラメーター

- `subscription_name` -既存のサブスクリプションの名前
- `interface_name` -現在のプロバイダーノードの既存のインターフェイスの名前
- `pglogical.alter_subscription_synchronize(subscription_name name, truncate bool)` すべてのセットのすべての非同期テーブルが、単一のオペレーションで同期されます。テーブルは 1 つずつコピーおよび同期されます。コマンドはブロックせず、アクションを開始するだけです。`pglogical.wait_for_subscription_sync_complete` を使用して、完了を待機します。

パラメーター

- `subscription_name` -既存のサブスクリプションの名前
- `truncate` -true の場合、テーブルはコピー前に切り詰められます、デフォルト false
- `pglogical.alter_subscription_resynchronize_table(subscription_name name, relation regclass)` 既存の 1 つのテーブルを再同期します。テーブルは、外部キー制約のターゲットであるではありません。警告このファンクションはテーブルをすぐに切り捨て、その後にのみ同期を開始するため、同期中は空になります

ブロックしません。 `pglogical.wait_for_table_sync_complete` を使用して完了を待機します。

パラメーター

- `subscription_name` -既存のサブスクリプションの名前
- `relation` -既存のテーブルの名前、オプションで修飾
- `pglogical.wait_for_subscription_sync_complete(subscription_name name)`

`pglogical.create_subscription` または `pglogical.alter_subscription_synchronize` の後に、サブスクリプションまたは同期が終了するまで待機します。

このファンクションは、サブスクリプションの初期スキーマ/データ同期がある場合は、完了し、個々の再同期保留中のテーブルも同期が終了するまで待機します。

最良の結果を得るには、再同期を要求したレプリケーションセットの変更の後にプロバイダーで `SELECT pglogical.wait_slot_confirm_lsn(NULL, NULL)` を実行し、その後にのみサブスクライバーで `pglogical.wait_for_subscription_sync_complete` を呼び出します。

- `pglogical.wait_for_table_sync_complete(subscription_name name, relation regclass)`

`pglogical.wait_for_subscription_sync_complete` と同じですが、サブスクリプションの初期同期と名前付けテーブルのみを待機します。再同期保留中の他のテーブルは無視されます。

- `pglogical.wait_slot_confirm_lsn`

```
SELECT pglogical.wait_slot_confirm_lsn(NULL, NULL)
```

現在のノードのすべてのレプリケーションスロットが、すべてのプロバイダーの呼び出し時に xlog 挿入位置まで再生されるまで待機します。呼び出し時にすべてのスロットの `confirmed_flush_lsn` が `pg_current_wal_insert_lsn()` を通過したときに返されます。

オプションで、1つのレプリケーションスロット最初の引数のみを待機できます。オプションで、insert lsn 2番目の引数の代わりに渡される任意の LSN を待機することができます。通常、どちらも null のままです。

この機能は、すべての加入者がプロバイダーの特定のポイントまでの変更を受信したことを保証するために非常に役立ちます。

- `pglogical.show_subscription_status(subscription_name name)` サブスクリプションに関するステータスと基本情報を表示します。

パラメーター

- `subscription_name` - 既存のサブスクリプションのオプションの名前。名前が指定されない場合、関数はローカルノード上のすべてのサブスクリプションのステータスを表示します
- `pglogical.show_subscription_table(subscription_name name, relation regclass)` テーブルの同期状態を示します。

パラメーター

- `subscription_name` - 既存のサブスクリプションの名前
- `relation` - 既存のテーブルの名前、オプションで修飾
- `pglogical.alter_subscription_add_replication_set(subscription_name name, replication_set name)` 1つのレプリケーションセットをサブスクライバーに追加します。同期はせず、イベントの使用をアクティブ化するだけです。

パラメーター

- `subscription_name` - 既存のサブスクリプションの名前
- `replication_set` - 追加するレプリケーションセットの名前
- `pglogical.alter_subscription_remove_replication_set(subscription_name name, replication_set name)` サブスクライバーから1つのレプリケーションセットを削除します。

パラメーター

- `subscription_name` - 既存のサブスクリプションの名前
- `replication_set` - 削除するレプリケーションセットの名前

`postgresql.conf` パラメーター `pglogical.extra_connection_options` もあり、`pglogical` によって作成されるすべての接続に適用される接続オプションを割り当てるように設定できます。これは、カスタムキープアライブオプションなどを設定するのに役立ちます。

`pglogical` は、TCP キープアライブを有効にして、上流サーバーが予期せず消えたときに通知できるようにします。それらを無効にするには、`keepalives = 0` を `pglogical.extra_connection_options` に追加します。

4.5 レプリケーションセット

レプリケーションセットは、データベース内のどのテーブルをレプリケートするか、およびこれらのテーブルのどのアクションをレプリケートするかを制御するメカニズムを提供します。

レプリケートされた各セットは、セット上の INSERTs、UPDATEs、DELETEs、および TRUNCATEs をレプリケートするかどうかを個別に指定できます。すべてのテーブルは複数のレプリケーションセットにすることができ、すべてのサブスクライバーは同様にマルチプルのレプリケーションセットをサブスクライブできます。レプリケートされたテーブルとアクションの結果のセットは、テーブルが含まれるセットの和集合です。テーブルは、レプリケーションセットに追加されるまでレプリケートされません。

「default」、「default_insert_only」、および「ddl_sql」という名前の3つの既存のレプリケーションセットがあります。「デフォルト」レプリケーションセットは、その内のテーブルへのすべての変更をレプリケートするように定義されます。“default_insert_only”はINSERTのみをレプリケートし、主キーを持たないテーブルを対象としています。詳細については、[Limitations and restrictions](#) セクションを参照してください。“ddl_sql”レプリケーションセットは、pglogical.replicate_ddl_command で指定されたスキーマ変更をレプリケートするように定義されます

レプリケーションセットを管理するために次の機能が提供されています。

- `pglogical.create_replication_set(set_name name, replicate_insert bool, replicate_update bool, replicate_delete bool, replicate_truncate bool)` この関数は、新しいレプリケーションセットを作成します。

Parameters:

- `set_name` - name of the set, must be unique
- `replicate_insert` - specifies if INSERT is replicated, default true
- `replicate_update` - specifies if UPDATE is replicated, default true
- `replicate_delete` - specifies if DELETE is replicated, default true
- `replicate_truncate` - specifies if TRUNCATE is replicated、デフォルト true
- `pglogical.alter_replication_set(set_name name, replicate_inserts bool, replicate_updates bool, replicate_deletes bool, replicate_truncate bool)` この関数は、既存のレプリケーションセットのパラメーターを変更します。

Parameters:

- `set_name` - name of the existing replication set
- `replicate_insert` - specifies if INSERT is replicated, default true
- `replicate_update` - specifies if UPDATE is replicated, default true
- `replicate_delete` - specifies if DELETE is replicated, default true
- `replicate_truncate` - specifies if TRUNCATE is replicated, default true
- `pglogical.drop_replication_set(set_name text)` レプリケーションセットを削除します。

パラメーター

- `set_name` - 既存のレプリケーションセットの名前
- `pglogical.replication_set_add_table(set_name name, relation regclass, synchronize_data boolean, columns text[], row_filter text)` レプリケーションセットにテーブルを追加します。

パラメーター

- `set_name` - 既存のレプリケーションの名前 `set`
- `relation` - `set`
- に追加するテーブルの名前または OID `synchronize_data` - `true` の場合、テーブルデータは指定されたレプリケーションセット、デフォルト `false`
- `columns` - レプリケートする列のリスト。通常、すべての列をレプリケートする必要がある場合、これはデフォルトである `NULL` に設定されます `b_tran_13` `row_filter` - 行フィルタリング式、デフォルト `NULL` フィルタリングなし、詳細については、[行フィルタリング](#) を参照してください。**警告** 有効な行フィルタとデータを同期するときは注意してください。有効な `row_filter` とともに `synchronize_data=true` を使用することは、テーブルの 1 回限りのオペレーションに似ています。`row_filter` を変更して再度実行しても、データはサブスクライバーに同期されません。サブスクリプションは、`pglogical.alter_subscription_resynchronize_table()` を呼び出して修正する必要がある場合があります。
- `pglogical.replication_set_add_all_tables(set_name name, schema_names text[], synchronize_data boolean)` 指定されたスキーマのすべてのテーブルを追加します。既存のテーブルのみが追加されます。今後作成されるテーブルは自動的に追加されません。今後作成されるテーブルが正しいレプリケーションセットに追加されることを確認する方法については、[新しいテーブルのレプリケーションセットの自動割り当て](#) を参照してください。

パラメーター

- `set_name` - 既存のレプリケーションセットの名前
- `schema_names` - 名前の配列 テーブルを追加する既存のスキーマの名前
- `synchronize_data` - `true` の場合、テーブルデータは指定されたレプリケーションセットをサブスクライブしているすべてのサブスクライバーで同期されます。デフォルトは `false`
- `pglogical.replication_set_remove_table(set_name name, relation regclass)` レプリケーションセットからテーブルを削除します。

パラメーター

- `set_name` - 既存のレプリケーションセットの名前
- `relation` - セットから削除するテーブルの名前または OID
- `pglogical.replication_set_add_sequence(set_name name, relation regclass, synchronize_data boolean)` シーケンスをレプリケーションセットに追加します。

パラメーター

- `set_name` - 既存のレプリケーションの名前 `set`
- `relation` - `set`

- に追加するシーケンスの名前または OID `synchronize_data -true` の場合、シーケンス値はすぐに同期されます、デフォルト `false`
- `pglogical.replication_set_add_all_sequences(set_name name, schema_names text[], synchronize_data boolean)` 指定されたスキーマからすべてのシーケンスを追加します。既存のシーケンスのみが追加されます。今後作成されるシーケンスは自動的に追加されません。

パラメーター

- `set_name` - 既存のレプリケーションセットの名前 `b_tran_7` `schema_names` - 名前の配列テーブルを追加する既存のスキーマの名前
- `synchronize_data -true` の場合、シーケンス値はすぐに同期されます、デフォルト `false`
- `pglogical.replication_set_remove_sequence(set_name name, relation regclass)` レプリケーションセットからシーケンスを削除します。

パラメーター

- `set_name` - 既存のレプリケーションセットの名前
- `relation` - セットから削除するシーケンスの名前または OID

`pglogical.tables` ビューを照会することにより、どのテーブルがどのセットにあるかに関する情報を表示できます。

4.5.1 新しいテーブルのレプリケーションセットの自動割り当て

イベントトリガー機能は、新しく作成されるテーブルのレプリケーションセットを定義するルールを記述するために使用できます。

例

```
CREATE OR REPLACE FUNCTION pglogical_assign_repset()
RETURNS event_trigger AS $$
DECLARE obj record;
BEGIN
  FOR obj IN SELECT * FROM pg_event_trigger_ddl_commands()
  LOOP
    IF obj.object_type = table THEN
      IF obj.schema_name = config THEN
        PERFORM pglogical.replication_set_add_table(configuration, obj.objid);
      ELSIF NOT obj.in_extension THEN
        PERFORM pglogical.replication_set_add_table(default, obj.objid);
      END IF;
    END IF;
  END LOOP;
END;
$$ LANGUAGE plpgsql;
```

(次のページに続く)

```
CREATE EVENT TRIGGER pglogical_assign_repset_trg
  ON ddl_command_end
  WHEN TAG IN (CREATE TABLE, CREATE TABLE AS)
  EXECUTE PROCEDURE pglogical_assign_repset();
```

上記の例では、スキーマ `config` で作成されたすべての新しいテーブルをレプリケーションセット `configuration` に配置し、拡張機能によって作成されない他のすべての新しいテーブルは `default` レプリケーションセットに配置されます。

4.6 追加機能

- `pglogical.replicate_ddl_command(command text, replication_sets text[])` ローカルで実行し、指定されたコマンドをレプリケーションキューに送信して、指定された `replication_sets` のいずれかにサブスクライブしているサブスクライバーで実行します。

パラメーター

- `command` - 実行する DDL クエリー
- `replication_sets` - このコマンドを関連付けるレプリケーションセットの配列、デフォルト "{ddl_sql}"
- `pglogical.synchronize_sequence(relation regclass)` シーケンス状態をすべてのサブスクライバーにプッシュします。サブスクリプションやテーブル同期ファンクションとは異なり、このファンクションはプロバイダーで実行する必要があります。このファンクションが実行されたトランザクションをレプリケートすると、すべてのサブスクライバが消費する追跡シーケンスの状態の更新を強制しますレプリケーションセットフィルタリングは引き続き適用されます。

パラメーター

- `relation` - 既存のシーケンスの名前、オプションで修飾

4.7 行フィルタリング

PGLogical では、プロバイダー側とサブスクライバー側の両方で行ベースのフィルタリングができます。

4.7.1 プロバイダーの行フィルタリング

プロバイダーでは、`pglogical.replication_set_add_table` ファンクションの `row_filter` パラメーターを指定することにより、行フィルタリングを実行できます。`row_filter` は、`CHECK` 制約と許可されるものと同じ制限がある通常の PostgreSQL 式です。

単純な `row_filter` は `row_filter := 'id > 0'` のようになり、`id` 列の値がゼロより大きい行のみがレプリケートされます。

`row_filter` 内で `volatile` ファンクションを使用することはできますが、書き込みを行う式はエラーをスローし、レプリケーションを停止するため、書き込みに関して注意を払う必要があります。

`row_filter` はレプリケーションセッション内で実行されているため、`CURRENT_USER` などのセッション固有の式は、書き込みを行ったセッションではなく、レプリケーションセッションの値を持つことにも注意してください。

4.7.2 サブスライバーの行フィルタリング

サブスライバーでは、標準の `BEFORE TRIGGER` メカニズムを使用して行ベースのフィルタリングを実装できます。

このようなトリガーを `ENABLE REPLICA` または `ENABLE ALWAYS` としてマークする必要があります。それ以外の場合、レプリケーションプロセスによって実行されません。

第5章 Synchronous Replication

同期レプリケーションは、物理レプリケーション用に PostgreSQL が提供するのと同じ標準のメカニズムを使用してサポートされています。

`synchronous_commit` および `synchronous_standby_names` 設定は、`synchronous_standby_names` で pg 論理サブスクリプション名が使用されている場合、`COMMIT` コマンドがクライアントに成功を報告するタイミングに影響を与えます。これら 2 つの変数を構成する方法の詳細については、PostgreSQL のドキュメントを参照してください。

第6章 Conflicts

ノードが複数のプロバイダーにサブスクライブされている場合、またはサブスクライバーでローカル書き込みが発生すると、着信変更で競合が発生する可能性があります。これらは自動的に検出され、構成に応じて対応できます。

競合リゾルバーの構成は、`pglogical.conflict_resolution` 設定を介して行われます。

解決された競合は、`pglogical.conflict_log_level` を使用して設定されたログレベルを使用してログに記録されます。このパラメーターのデフォルトは LOG です。`log_min_messages` より低いレベルに設定されている場合、解決された競合はサーバーログに表示されません。

第7章 Configuration options

PGLogical の一部の機能は、`postgresql.conf` または `ALTER SYSTEM SET` を介して設定できる構成オプションを使用して構成できます。

- `pglogical.conflict_resolution` ローカルデータと着信変更の間で検出された競合の解決方法を設定します。

可能な値

- `error` -競合が検出され、解決に手動アクションが必要な場合、レプリケーションはエラーで停止します
- `apply_remote` -常にローカルデータと競合する変更を適用します
- `keep_local` -データのローカルバージョンを維持し、競合する変更を無視しますリモートノードから来ています
- `last_update_wins` -最新のコミットタイムスタンプを持つデータのバージョンが保持されますローカルまたはリモートバージョンのいずれかです
- `first_update_wins` -最も古いタイムスタンプを持つデータのバージョンが保持されますこれはローカルまたはリモートのいずれかですバージョン)

使用可能な設定とデフォルトは、PostgreSQL のバージョンおよびその他の設定によって異なります。

PostgreSQL のデフォルト値は `apply_remote` です。

`keep_local`、`last_update_wins`、および `first_update_wins` 設定では、`track_commit_timestamp` PostgreSQL 設定を有効にする必要があります。`track_commit_timestamp` は PostgreSQL 9.4 では利用できないため、`pglogical.conflict_resolution` は `apply_remote` または `error` のみです。

Postgres-XL では、サポートされている唯一の値とデフォルトは `error` です。

- `pglogical.conflict_log_level` `pglogical.conflict_resolution` が `error` 以外に設定されている場合、検出された競合をレポートするためのログレベルを設定します。

この設定の主な使用は、競合のログを抑制することです。

指定できる値は `log_min_messages` PostgreSQL 設定と同じです。

デフォルトは LOG です。

- `pglogical.batch_inserts` 可能であればバッチ挿入メカニズムを使用するように PGLogical に指示します。バッチメカニズムは、COPY コマンドでも使用される PostgreSQL 内部バッチ挿入モードを使用します。

バッチ挿入は、1つのテーブルに多数の挿入を実行したトランザクションのレプリケーションパフォーマンスを向上させます。PGLogical は、トランザクションが5つを超える INSERT を行った場合、バッチモードに切り替えます。

テーブルに `INSTEAD OF INSERT` および `BEFORE INSERT` トリガーがなく、テーブルの列の揮発性式を使用したデフォルトがない場合にのみバッチモードに切り替えることができます。また、バッチモードは、`pglogical.conflict_resolution` が `error` に設定されている場合にのみ機能します。

デフォルトは `true` です。

- `pglogical.use_spi` 内部低レベルインターフェイスを使用するのではなく、SPI インターフェイスを使用して実際の SQL INSERT、UPDATE、DELETE ステートメントを形成し、入った変更を適用するように PGLogical に指示します。

これは、主に Postgres-XL およびデバッグの目的に役立ちます。

PostgreSQL のデフォルトは `false` です。

`pglogical.conflict_resolution` が `error` に設定されている場合にのみ `true` に設定できます。この状態では、競合は検出されません。

Postgres-XL では、デフォルトで唯一許可される設定は `true` です。

- `pglogical.temp_directory` スキーマの同期に必要な一時ファイルを配置するシステムパスを定義します。このパスは存在し、Postgres を実行しているユーザーが書き込み可能である必要があります。

デフォルトは空で、環境とオペレーティングシステムの設定に基づいて、デフォルトの一時ディレクトリを使用するように PGLogical に指示します。

第8章 Limitations and restrictions

8.1 スーパーユーザーが必要です

現在、pg 論理レプリケーションと管理にはスーパーユーザー権限が必要です。これは、後でより詳細な特権に拡張される場合があります。

8.2 UNLOGGED および TEMPORARY はレプリケートされません

UNLOGGED および TEMPORARY テーブルは、物理的ストリーミングレプリケーションの場合と同様に、レプリケートされず、レプリケートできません。

8.3 一度に1つのデータベース

複数のデータベースをレプリケートするには、それぞれに個々のプロバイダー/サブスクライバー関係を設定する必要があります。PostgreSQL インストール内のすべてのデータベースのレプリケーションを一度に構成する方法はありません。

8.4 PRIMARY KEY または REPLICA IDENTITY が必要です

UPDATE s および DELETE s は、PRIMARY KEY またはインデックスを使用するなどの他の有効なレプリカ ID が欠如しているテーブルにはレプリケートできません。インデックスは一意で、部分的ではなく、遅延不可で、NOT NULL としてマークされた列のみを含む必要があります。一意の識別子がないため、レプリケーションは更新/削除するタプルを見つける方法がありません。REPLICA IDENTITY FULL はまだサポートされていません。

8.5 一意のインデックス/コンストレイン/PK は1つだけ

複数のアップストリームが構成されている場合、またはダウンストリームがローカル書き込みを受け入れる場合、ダウンストリームのレプリケートテーブルには1つの UNIQUE インデックスのみが存在する必要があります。競合解決では一度に1つのインデックスのみを使用できるため、行が PRIMARY KEY を満たすが、ダウンストリーム側の UNIQUE 制約に違反する場合、競合する行は ERROR される場合があります。これにより、ダウンストリームテーブルが変更されて違反が削除されるまで、レプリケーションが停止されます。

ダウストリームがそのアップストリームからのみ書き込みを取得し、他の場所ではない場合、アップストリームに追加の一意制約があっても問題ありません。ルールは、下流のコンストレインが上流のコンストレインよりも制限的でないことです。

部分的なセカンダリー意インデックスは許可されますが、競合解決の目的で無視されます。

8.6 一意のコンストレインは遅延可能であってはなりません

ダウストリームエンドでは、pglogical は、DEFERRABLE として定義されたインデックスベースの制約をサポートしていません。エラーが発生します

```
ERROR: pglogical doesnt support index rechecks needed for deferrable indexes
DETAIL: relation "public"."test_relation" has deferrable indexes: "index1", "index2"
```

テーブルに変更を適用しようとしたときにそのようなインデックスが存在する場合。

8.7 DDL

自動 DDL レプリケーションはサポートされていません。プロバイダーとサブスクライバーデータベースの互換性を保つように DDL を管理することはユーザーの責任です。

pglogical は、pglogical.replicate_ddl_command ファンクションを提供して、一貫したポイントでプロバイダーとサブスクライバーで DDL を実行できます。

8.8 レプリケーションキューのフラッシュなし

マスターでのトランザクションのフリーズと、保留中のキューに入れられているすべての xact がスロットから再生されるまで待機することはサポートされていません。これのアップストリームを読み取り専用にするサポートは、将来のリリースで追加される予定です。

これは、テーブル構造の変更を適用するときに注意する必要があることを意味します。まだレプリケートされていないコミットされたトランザクションがあり、サブスクライバテーブルとキュー内のトランザクションとの互換性をなくす方法でプロバイダーとサブスクライバーのテーブル構造が同時に変更された場合、レプリケーションは停止します。

管理者は、スキーマを変更する前にマスターへの書き込みが停止していることを確認するか、pglogical.replicate_ddl_command ファンクションを使用してスキーマ変更をキューに入れて、レプリカの一貫したポイントで再生されるようにする必要があります。

マルチマスターレプリケーションのサポートが追加されると、パブリッシャーでスキーマ変更がコミットされた後、サブスクライバーが古い構造で新しい xact を生成する可能性があるため、pglogical.replicate_ddl_command を使用するだけでは十分ではありません。ユーザーは、スキーマを変更する前に、すべてのノードで書き込みが停止され、すべてのスロットがキャッチされていることを確認する必要があります。

8.9 外部キー

外部キー制約はレプリケーションプロセスに適用されません。プロバイダー側で成功したものは、FOREIGN KEY に違反する場合でもサブスクライバーに適用されます。

8.10 TRUNCATE

TRUNCATE ... CASCADE を使用すると、プロバイダー側で CASCADE オプションのみが適用されます。

これを適切に処理するには、おそらく PostgreSQL の外部キーの ON TRUNCATE CASCADE サポートを追加する必要があります。

TRUNCATE ... RESTART IDENTITY はサポートされていません。ID 再起動ステップはレプリカに複製されません。

8.11 シーケンス

レプリケーションセットに追加されたシーケンスの状態は、リアルタイムではなく定期的にレプリケートされます。動的バッファは、複製される値に使用されるため、サブスクライバは実際にシーケンスの将来の状態を受信します。これにより、サブスクライバのシーケンスの `last_value` の概念が遅れる可能性が最小限に抑えられますが、可能性を完全に排除するものではありません。

`synchronize_sequence` を呼び出して、データのロードなどのデータベース内の「大きなイベント」の後、またはオンラインアップグレード中、すべてのサブスクライバーが特定のシーケンスに関する最新の情報を持っていることを確認することが望ましい場合があります。

小さなシーケンスはすぐにシーケンススペースの最後に到達する可能性があるため、マルチノードシステム上のシーケンスには `bigserial` および `bigint` タイプを使用することをお勧めします。

プロバイダーとサブスクライバーに独立したシーケンスが必要なユーザーは、レプリケーションセットへのシーケンスの追加を回避し、ノード数以上のステップ間隔でシーケンスを作成できます。次に、各ノードに異なるオフセットを設定します。CREATE SEQUENCE または ALTER SEQUENCE は INCREMENT BY オプションを使用し、`setval(...)` を使用してスタートポイントを設定します。

8.12 トリガー

適用プロセスと初期 COPY プロセスは、両方とも `session_replication_role` を `replica` に設定して実行されます。これは、ENABLE REPLICA および ENABLE ALWAYS トリガーが起動されることを意味します。

8.13 PostgreSQL バージョンの違い

PGLogical は、PostgreSQL メジャーバージョン間でレプリケートできます。それにもかかわらず、長期的なクロスバージョンのレプリケーションは、多くの場合機能する場合がありますが、デザインターゲットとは考えられません。プロバイダーでは変更が有効だが、サブスクライバーでは有効でない問題は、バージョン間でレプリケートするときに発生する可能性が高くなります。

PostgreSQL は強固な下位互換性を維持しますが、限定的な前方互換性のみを維持するため、古いバージョンから新しいバージョンに複製する方が安全です。初期スキーマの同期は、PostgreSQL の同じバージョン間または下位バージョンから上位バージョンに複製する場合にのみサポートされています。

異なるマイナーバージョン間でレプリケートしても、まったく違いはありません。

8.14 データベースエンコーディングの違い

PGLogical は、エンコードが異なるデータベース間のレプリケーションをサポートしていません。すべてのレプリケートデータベースで UTF-8 エンコードを使用することをお勧めします。

8.15 ラージオブジェクト

PostgreSQL の論理デコード機能は、ラージオブジェクトのデコード変更をサポートしていないため、pglogical はラージオブジェクトをレプリケートできません。

8.16 Postgres-XL

Postgres-XL の最小サポートバージョンは 9.5r1.5 です。

Postgres-XL は、サブスクライバーとしてのみサポートされていますプロバイダーになることはできません。多くの小規模なトランザクションを含むワークロードの場合、書き込みレイテンシーの増加により、レプリケーションのパフォーマンスが低下する場合があります。一方、大規模な挿入または大量コピートランザクションは、Postgres-XL で非常に高速に動作するように高度に最適化されています。

また、DDL 制限が適用されるため、`replicate_ddl_command()` を使用するときは特に注意する必要があります。

Postgres-XL は、`pglogical.conflict_resolution` および `pglogical.use_spi` 構成オプションのデフォルトと使用可能な設定を変更します。