



EDB

pglogical

リリース *2.4.1*

EnterpriseDB Corporation

2023 年 04 月 05 日

目次

第 1 章	Release Notes	3
1.1	pglogical 2.4.1	3
1.1.1	変更	3
1.2	pglogical 2.4.0	3
1.2.1	変更	3
第 2 章	Requirements	5
第 3 章	Installation	7
3.1	パッケージ	7
3.1.1	YUM で pglogical をインストールする	7
3.1.2	前提条件	7
3.1.3	インストール	8
3.1.4	APT で pglogical をインストールする	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	9
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たサブスクライバ 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	複製セット	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	プロバイダでの行フィルタリング	20
4.7.2	サブスクライバーで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 と TEMPORARY は複製されません	29
8.3	一度に 1 つのデータベース	29
8.4	PRIMARY KEY または REPLICA IDENTITY が必要です	29
8.5	1 つの一意のインデックス/制約/PK のみ	29

8.6	一意の制約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	切り捨て	31
8.11	シーケンス	31
8.12	トリガー	31
8.13	PostgreSQL のバージョンの違い	32
8.14	データベースのエンコーディングの違い	32
8.15	大きなオブジェクト	32
8.16	Postgres-XL	32

pglogical 2 拡張機能は、パブリッシュ/サブスクライブモデルを使用して、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。EDB Postgres Distributed の一部として開発された技術に基づいています。

以下の用語を使用して、ノード間のデータ ストリームを説明します。以前の Slony テクノロジーから意図的に再利用されています。

- 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 - ノードが担う役割
- 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。

- プロバイダーおよびサブスクライバー ノードは PostgreSQL 9.4+ を実行する必要があります
- レプリケーション元の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 (上記の制限がある場合)
- データベースの完全複製
- レプリケーション セットを使用したテーブルセットの選択的レプリケーション
- パブリッシャー側またはサブスクライバー側でのテーブル行の選択的複製 (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、物理ストリーミングレプリケーションのようにサーバー全体のレベルではなく、データベースごとのレベルで動作します
- 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、1 つのプロバイダーが複数のサブスクライバーにフィードできます
- 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、自動および構成可能な競合解決 (マルチマスターに必要なすべての側面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 2.4.1

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

1.1.1 変更

- 「スナップショットはまだアクティブです」という警告と、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 で導入された問題。
- リモート レプリケーションスロットの削除に失敗した場合のエラー状態のフラッシュ

1.2 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。

1.2.1 変更

- PostgreSQL 14 (ベータ) の予備サポートを追加
- 行のセットではなく行を返すように `pglogical_show_subscription_table` を修正
- 出力プラグインでのスナップショット処理を修正し、ワーカーを適用します
- Windows でのコマンドの引用を修正して、実際に機能するようにします以前に使用されていた API では、コマンド ラインの長さが 1024 に制限されているようです
- Windows でスキーマの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ョンの二重クローズを修正
- SPI 適用ワーカーで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を複数回挿入するマルチ挿入を修正
- `debug_query_string` が常に設定されていることを確認してください PostgreSQL の新しいバージョンでは、`debug_query_string` が常に設定されている必要があります。

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` を実行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、同じ名前を持ち、同じスキーマ内にある必要があります。今後の改訂で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、各列のデータ型が同じ同じ列が必要です。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、より弱い (より寛容である) 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 制約を追加することはお勧めしません (下記参照)。

いくつかの追加要件については、以下の「制限事項と制約事項」で説明されています。

第3章 Installation

3.1 パッケージ

pglogical は、Fedora、CentOS、および RHEL の yum を介して RPM として、Debian および Ubuntu の apt を介して DEB として、またはこちらの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 で pglogical をインストールする

以下の手順は、Red Hat ファミリ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/> からの 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 PostgreSQL リリースでは使用できません。PostgreSQL をまだ持っていない場合:

- <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。
- PostgreSQL をインストールする
 - 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`

次に、宛先 Linux サーバーでルートとして次の手順を実行して、PostgreSQL バージョンの「2ndQuadrant's 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`

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`

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
→key 0xD6BAF0C3: Userid : "Public repository signing key 2ndQuadrant <ci@2ndquadrant.
→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（上記と一致する場合）、Enter を押します。（確認したい場合は、2ndQuadrant マスター パッケージキーで署名されています。）

3.1.4 APT で pglogical をインストールする

以下の手順は、Debian および Debian ベースのすべての Linux フレーバー (Ubuntu など) に有効です。

3.1.5 前提条件

「2ndQuadrant's General Public」リポジトリをインストールするには、宛先の Linux サーバーで root として次の手順を実行します: `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、お使いのバージョンの PostgreSQL に対して次のコマンド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`

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 を使用して構築された他の 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てください。 `pg_config` がない場合は、パッケージ マネージャーから PostgreSQL リリース用の `-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 `sudo` を使用する必要がある場合があります。

例えば通常の Fedora または RHEL 7 インストールの場合、PostgreSQL に

yum.postgresql.org パッケージを使用していると仮定します。

```
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 では機能しません) を使用していて、最後/最初の更新が優先される競合解決を処理したい場合 (

Conflicts を参照)、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、localhost 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、すべてのノードに pglogical 拡張機能をインストールする必要があります。

```
CREATE EXTENSION pglogical;
```

PostgreSQL 9.4 を使用している場合、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]);
```

必要に応じて、追加のレプリケーションセットを作成し、それらにテーブルを追加することもできます ([複製セット](#)を参照)。

通常は、サブスクリाइブする前にレプリケーションセットを作成することをお勧めします。これにより、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ます。ただし、大規模なデータベースのユーザーは、制御を強化するためにデータベースを段階的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をサブスクリाइブできます。まず、サブスクライバー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
);
```

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たサブスクライバ 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 でプロバイダーを複製し、それを pglogical サブスクライバーとして起動することで、サブスクライバーの作成もサポートします。これは pglogical_create_subscriber ツールで行われます。--help の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、特に高帯域幅のリンクでは、多くの場合、はるかに高速です。

4.3 ノード管理

ノードは、SQL インターフェイスを使用して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：

1 `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。

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でも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)`
ノードに追加のインターフェイスを追加します。

ノードが作成されると、ノードと同じ名前でも `create_node` で指定された `dsn` を使用して、そのためのインターフェイスも作成されます。このインターフェイスを使用すると、異なる接続文字列を持つ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ースの名前
- `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)` ノードから既存のインターフェイスを削除します。

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ースの名前

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。一意でなければなりません
- `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。これらは既に存在している必要があります。デフォルトは “{default,default_insert_only,ddl_sql}” です。
- **synchronize_structure** - プロバイダーからサブスクライバーに構造を同期するかどうかを指定します。デフォルトは 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。デフォルトは true
- **forward_origins** - 転送する元の名前の配列。現在サポートされている値のみが空の配列であり、プロバイダーノードで発生していない変更を転送しないことを意味します (これは、ノード間の双方向のレプリケーションに役立ちます)、または “{all}” は、変更元が何であれ、すべての変更を複製することを意味します。デフォルトは “{all}” です。
- **apply_delay** - レプリケーションをどれだけ遅らせるか、デフォルトは 0 秒
- **force_text_transfer** - テキスト表現を使用してすべての列をレプリケートするようプロバイダーに強制します (遅くなりますが、サブスクライバーでレプリケートされた列のタイプ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

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** 監視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** でも使用できます。

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的にレプリケーションを開始し、必要なスキーマや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でも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。それ以外の場合は、現在のトランザクションの最後にのみ停止されます。デフォルトは false です。
- **pglogical.alter_subscription_enable(subscription_name name, immediate bool)** 無効化された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。それ以外の場合は、現在のトランザクションの最後にのみ開始されます。デフォルトは false です。

- `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は、1回の操作で同期されます。テーブルは1つずつコピーされ、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ちます。

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捨てられます。デフォルトは `false`
- `pglogical.alter_subscription_resynchronize_table(subscription_name name, relation regclass)`

1つの既存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はない可能性があります。

ブロックせず、`pglogical.wait_for_table_sync_complete` を使用して完了を待ちます。

パラメーター：

- `subscription_name` - 既存のサブスクリプションの名前
- `relation` - オプションで修飾された既存のテーブルの名前

`pglogical.wait_for_subscription_sync_complete(subscription_name name)`

サブスクリプションを待つか、`pglogical.create_subscription` または `pglogical.alter_subscription_synchronize` の後に同期を終了します。

この関数は、サブスクリプションの最初のスキーマ/データ同期があればそれが完了するまで待機し、個々の再同期を保留しているテーブルも同期を完了するまで待機します。

最良の結果を得るには、再同期を要求するレプリケーションセットの変更後にプロバイダーで `SELECT pglogical.wait_slot_confirm_lsn(NULL, NULL)` を実行し、その後で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最初の同期と名前付きテーブルのみを待ちます。再同期を保留している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ちます。呼び出し時にすべてのスロットの `confirmed_flush_lsn` が `pg_current_wal_insert_lsn()` を通過すると戻ります。

必要に応じて、レプリケーション スロットを 1 つだけ待機することもできます (最初の引数)。オプションで、insert lsn (2 番目の引数) の代わりに任意の LSN が渡されるのを待つことができます。どちらも通常は null のままです。

この関数は、すべてのサブスクライバーがプロバイダーの特定のポイントまで変更を受信したことを確認するの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。名前が指定されていない場合、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)`

テーブルの同期ステータスを表示します。

パラメーター：

- **subscription_name** - 既存のサブスクリプションの名前
- **relation** - オプションで修飾された既存のテーブルの名前
- `pglogical.alter_subscription_add_replication_set(subscription_name name, replication_set name)`

1 つの複製セットをサブスクライバーに追加します。同期せず、イベントの消費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

pglogical によって作成された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ために設定できる `postgres.conf` パラメータ、`pglogical.extra_connection_options` もあります。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便利な場所です。

pglogical はデフォルトで TCP キープアライブを有効にして、アップストリームサーバーが予期せず消えたときに確実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 複製セット

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、およびそれらのテーブルで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、セットの INSERTs、UPDATEs、DELETEs、および TRUNCATEs がレプリケートされるかどうかを個別に指定できます。すべてのテーブルを複数のレプリケーションセットに含めることができ、すべてのサブスクライバーも複数のレプリケーションセットに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」、「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」複製セットは、その中のテーブルに対するすべての変更を複製するように定義されています。「default_insert_only」は INSERT のみを複製し、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ています。

複製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。

パラメーター：

- `set_name` - セットの名前、一意でなければなりません
- `replicate_insert` - INSERT がレプリケートされるかどうかを指定します。デフォルトは true
- `replicate_update` - UPDATE がレプリケートされるかどうかを指定します。デフォルトは true
- `replicate_delete` - DELETE がレプリケートされるかどうかを指定します。デフォルトは true
- `replicate_truncate` - TRUNCATE がレプリケートされるかどうかを指定します。デフォルトは 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。

パラメーター：

- `set_name` - 既存の複製セットの名前
- `replicate_insert` - INSERT がレプリケートされるかどうかを指定します。デフォルトは true
- `replicate_update` - UPDATE がレプリケートされるかどうかを指定します。デフォルトは true
- `replicate_delete` - DELETE がレプリケートされるかどうかを指定します。デフォルトは true
- `replicate_truncate` - TRUNCATE がレプリケートされるかどうかを指定します。デフォルトは 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` - 既存の複製セットの名前
- `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 セットの名前
- `relation` - セットに追加され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にサブスクライブされ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
- `columns` - 複製する列のリスト。通常、すべての列を複製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。**警告: 有効な行フィルター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` で `synchronize_data=true` を使用することは、テーブルに対する 1 回限りの操作のようなものです。変更された `row_filter` で再度実行すると、データはサブスクライバーに同期されません。サブスクライバーは、`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に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、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ように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` - テーブルを追加する必要がある既存のスキーマの名前の配列
- `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にサブスクライブされ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)` 複製セットにシーケンスを追加します。

パラメーター：

- `set_name` - 既存のレプリケーション セットの名前
- `relation` - セットに追加され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。デフォルトは 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、将来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
- `schema_names` - テーブルを追加する既存のスキーマの名前の配列
- `synchronize_data` - true の場合、シーケンス値はすぐに同期されます。デフォルトは false
- `pglogical.replication_set_remove_sequence(set_name name, relation regclass)` 複製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` - セットから削除されるシーケンスの名前または OID

`pglogical.tables` ビューをクエリすることで、どのテーブルがどのセットに含まれてい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、新しく作成された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;

CREATE EVENT TRIGGER pglogical_assign_repset_trg
ON ddl_command_end
```

(次のページに続く)

```
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てい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 セットに移動します。

4.6 追加機能

- `pglogical.replicate_ddl_command(command text, replication_sets text[])` ローカルで実行し、指定したコマンドをレプリケーションキューに送信して、指定した `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
- `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、デフォルトは「`{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およびテーブル同期関数とは異なり、この関数はプロバイダーで実行する必要があります。この関数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ー (レプリケーション セットのフィルタリングは引き続き適用されます) によって消費される、追跡されたシーケンス状態の更新が強制されます。

パラメーター：

- `relation` - オプションで修飾された既存のシーケンスの名前

4.7 行フィルタリング

PGLogical では、プロバイダ側とサブスクライバ側の両方で行ベースのフィルタリングが可能です。

4.7.1 プロバイダでの行フィルタリング

プロバイダーでは、`pglogical.replication_set_add_table` 関数に `row_filter` パラメーターを指定することで、行のフィルタリングを行うことができます。`row_filter` は通常の PostgreSQL 式であり、CHECK 制約と同じ制限があります。

単純な `row_filter` は `row_filter := 'id > 0'` のようになります。これにより、`id` 列の値がゼロより大きい行のみが複製されます。

`row_filter` 内で `volatile` 関数を使用することは許可されていますが、書き込みを行う式はエラーをスローしてレプリケーションを停止するため、書き込みに関しては注意が必要です。

また、`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には、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が含まれることにも注意してください。

4.7.2 サブスクリャーでの行フィルタリング

サブスクリャーでは、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。

そ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。そうしないと、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、PostgreSQL が提供する物理レプリケーションと同じ標準メカニズムを使用してサポートされています。

`synchronous_commit` と `synchronous_standby_names` の設定は、`synchronous_standby_names` で `pg-logical` 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ときに影響します。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した場合、着信変更に対して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、構成に応じて対処できます。

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` よりも低いレベルに設定すると、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 の一部の側面は、`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、解決のために手動の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** - ローカル データと競合する変更を常に適用します - **keep_local** - データのローカルバージョンを保持し、競合する変更を無視しますリモートノードから - **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 (これは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 のバージョンと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 では使用できないため、`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であり、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に、検出された競合を報告するためのログレベルを設定します。

この設定の主な用途は、競合のロギングを抑制することです。

可能な値は、`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。

バッチ挿入により、1 つのテーブルに多くの挿入を行っ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が向上します。トランザクションが 5 回以上の INSERT を実行すると、PGLogical はバッチ モードに切り替わります。

テーブルに `INSTEAD OF INSERT` および `BEFORE INSERT` トリガーがなく、テーブルの列に揮発性式の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` 内部の低レベルインターフェイスを使用する代わりに、SPI インターフェイスを使用して実際の SQL (`INSERT`、`UPDATE`、`DELETE`) ステートメントを作成し、着信変更を適用するように PGLogical に指示します。

これは主に Postgres-XL とデバッグの目的で役立ちます。

PostgreSQL のデフォルトは `false` です。

これは、`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ている設定は `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。このパスが存在し、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。

デフォルトは空です。これにより、PGLogical は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なります。

第8章 Limitations and restrictions

8.1 スーパーユーザーが必要です

現在、pglogical の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。後で、より詳細な権限に拡張される可能性があります。

8.2 UNLOGGED と TEMPORARY は複製されません

UNLOGGED および TEMPORARY テーブルは、物理ストリーミングレプリケーションと同様に、レプリケートされません。

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、それぞれに対して個別のプロバイダー/サブスクライバーの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 がないテーブルに対して複製することはできません。インデックスは一意であり、部分的ではなく、遅延可能ではなく、NOT NULL と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/削除する必要があ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 1 つの一意のインデックス/制約/PK のみ

複数のアップストリームが構成されているか、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複製されたテーブルには UNIQUE インデックスが 1 つだけ存在する必要があります。競合解決では一度に 1 つのインデックスしか使用できないため、行が PRIMARY KEY を満たしているが下流側で UNIQUE 制約に違反している場合、競合する行は ERROR になる可能性があります。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解消されるまで、レプリケーションが停止します。

ダウン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、他のどこにも書き込みを取得しない場合は、アップストリームに追加の一意の制約を設定しても問題ありません。ルールは、下流の制約が上流の制約よりも制限的であってはならないということです。

部分的なセカンダリ ユニークインデックスは許可されますが、競合解決のために無視されます。

8.6 一意の制約は遅延可能であってはなりません

下流側では、pglogical は DEFERRABLE として定義されたインデックススペ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。プロバイダーデータベースとサブスクライバーデータベースの互換性を維持するための DDL の管理は、ユーザーの責任です。

pglogical は `pglogical.replicate_ddl_command` 関数を提供して、プロバイダーとサブスクライバーで一貫した時点で DDL を実行できるようにします。

8.8 レプリケーション キューのフラッシュなし

マスターでトランザクションをフリーズし、保留中のキューに入れられたすべての xacts がスロットからリプレイされるまで待機することはサポートされていません。このためにアップストリームを読み取り専用にするためのサポートは、将来のリリースで追加される予定です。

つまり、テーブル構造の変更を適用するときは注意が必要です。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、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、サブスクライバーテーブルがキューに入れられたトランザクションと互換性がなくなる場合、レプリケーションは停止します。

管理者は、スキーマの変更を行う前にマスターへの書き込みが停止されていることを確認するか、`pglogical.replicate_ddl_command` 関数を使用してスキーマの変更をキューに入れ、レプリカの一貫した時点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 xacts 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、スキーマの変更を行う前に、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追いつくようにする必要があります。

8.9 外部キー

レプリケーションプロセスには外部キー制約は適用されません。FOREIGN KEY に違反したとしても、プロバイダー側で成功したものがサブスクライバーに適用されます。

8.10 切り捨て

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、PostgreSQL の外部キーに対する 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 再開ステップはレプリカに複製されません。

8.11 シーケンス

複製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複製されます。サブスクライバーがシーケンスの将来の状態を実際に受信できるように、複製される値に動的バッファが使用されます。これにより、シーケンスの last_value が遅れるというサブスクライバーの概念の可能性が最小限に抑えられますが、可能性が完全に排除されるわけではありません。

synchronize_sequence を呼び出して、データの読み込みやオンラインアップグレードなどのデータベースでの「大きなイベント」の後、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確実に入手できるようにすることが望ましい場合があります。

通常、マルチノードシステムのシーケンスには bigserial および bigint タイプを使用することをお勧めします。小さいシーケンスは、シーケンススペースの最後にすばやく到達する可能性があるためです。

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を持たせたいユーザーは、レプリケーションセットにシーケンスを追加することを避け、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 には INCREMENT BY オプションを使用し、setval(...) を使用して開始点を設定します。

8.12 トリガー

適用プロセスと初期 COPY プロセスの両方が 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 のバージョンの違い

PGLogical は、PostgreSQL のメジャーバージョン間で複製できます。それにもかかわらず、長期的なクロスバージョンレプリケーションは設計目標とは見なされていませんが、多くの場合は機能する可能性があります。変更がプロバイダーでは有効であるが、サブスクライバーでは有効でないという問題は、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 は完全な後方互換性を維持していますが、前方互換性は限られているため、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。初期スキーマ同期は、同じバージョンの PostgreSQL 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ます。

異なるマイナーバージョン間で複製しても、まったく違いはありません。

8.14 データベースのエンコーディングの違い

PGLogical は、エンコーディング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。レプリケートされたすべてのデータベースで UTF-8 エンコーディングを使用することをお勧めします。

8.15 大きなオブジェクト

PostgreSQL の論理デコード機能は、大きなオブジェクトへの変更のデコードをサポートしていないため、pglogical は大きなオブジェクトを複製できません。

8.16 Postgres-XL

サポートされている Postgres-XL の最小バージョンは 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ます (プロバイダーになることはできません)。小さなトランザクションが多数含まれるワークロードでは、書き込みレイテンシが増加するため、レプリケーションのパフォーマンスが低下する可能性があります。一方、大規模な挿入 (または一括コピー) 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大幅に最適化されています。

また、すべての DDL 制限が適用されるため、`replicate_ddl_command()` を使用する場合は特別な注意が必要です。

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。