PostgreSQL インサイド

論理レプリケーション解説

仕組み、設定方法

技術を知る

FUJITSU

- [ ] 導入／環境設定
- [ ] 移行
- [ ] 性能
- [ ] チューニング
- [ ] バックアップ／リカバリー
- [x] 冗長化／負荷分散
- [ ] 監視
- [ ] データ連携
- [ ] 災害対策
- [ ] 豆知識

データベースでの「レプリケーション」とは、データベースのレプリカ（複製）を作成することです。レプリケーションの方式には、データベースクラスタ全体を複製する「ストリーミングレプリケーション(物理レプリケーション)」と、スキーマ単位やテーブル単位に複製する「ロジカルレプリケーション(論理レプリケーション)」があります。
この記事の前半では、論理レプリケーションの概要、用途、ストリーミングレプリケーションとの違い、構築可能な構成を説明します。後半では、設定方法や基本的な使い方についてサンプルを交えて解説します。
本記事の設定方法などの検証は、PostgreSQL 17 で実施しています。

なお、ストリーミングレプリケーションの概要を押さえておくことで、論理レプリケーションの理解がより深まります。以下の記事で解説していますので、まだ読まれていない方は、ぜひ先にお読みください。

- ストリーミングレプリケーション ～ 仕組み、構成のポイント ～
- ストリーミングレプリケーション ～ 運用のための機能 ～

1. 論理レプリケーションとは

論理レプリケーションを適切に使う上で、レプリカアイデンティティなどの専門用語を理解しつつ機能概要を押さえることは重要です。以降で、専門用語を解説しつつ論理レプリケーションの機能概要を以下の順で説明します。

- 機能概要
- レプリカアイデンティティとは
- 利用例
- 利用者起点での機能の十分性
- ストリーミングレプリケーションとの違い

機能概要

論理レプリケーションとは、レプリカアイデンティティ（主キーなど）に基づいて、データオブジェクトと、それに対する変更に関する情報（WAL）を、論理的な情報に変換して複製（転送）する機構です。
データの複製元をパブリッシャー、複製先をサブスクライバーと呼びます（PUB/SUB モデルをベースにしています）。また、レプリケーション対象のテーブルなどを指定したオブジェクトをパブリケーション（publication）、レプリケーション対象のパブリケーションとの接続先を定義したオブジェクトをサブスクリプション（subscription）と呼びます。

論理レプリケーションのイメージを以下に示します。

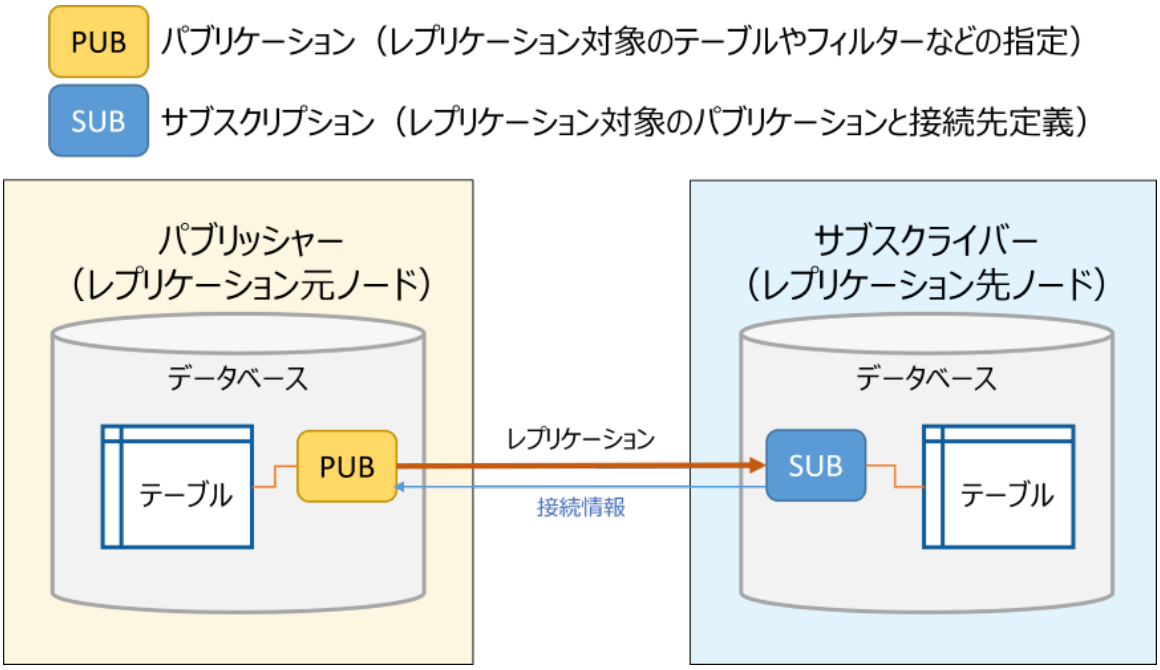

図 1：論理レプリケーションのイメージ

レプリカアイデンティティとは

前述したレプリカアイデンティティとは、パブリケーションとサブスクリプションとのレコードを対応づけるためのカラム名のことです。例えば、パブリケーションとサブスクリプションの両テーブルのカラムに主キーやユニークキーを付けることで、レプリカアイデンティティを作成できます。

レプリカアイデンティティは、以下を実現するために必要です。

- 対象テーブルに対して UPDATE 文と DELETE 文を実行する
- パブリケーションで行フィルターを設定する
- パブリケーションで列フィルターを設定する

なお、論理レプリケーションで複製する対象が INSERT のみ、かつ、行フィルターや列フィルターを使わない場合、レプリカアイデンティティは必要ありません。行フィルターと列フィルターの機能概要および使い方に関しては、本記事の後半で説明します。

利用例

以下の要件に対して、論理レプリケーションを利用できます。

- 特定データ（テーブル）を他のデータベースに共有
- 複数のデータベースに散らばったデータの統合や整理
- OS や PostgreSQL のバージョンアップ

例えば、「複数のデータベースに散らばったデータの統合や整理」のユースケースでは、図 2 のように論理レプリケーションを利用することで、下記の要件を満たせます。

要件）

A 社と B 社の合併により新会社が設立された。業務統合をする上で、データ統合が必要であるが、A 社、B 社ともに、業務を停止することが許されず、かつ、従業員テーブルにも更新が入る。この状況で、新会社のデータベースに統合された従業員テーブルを動的に構築することを要件とする。

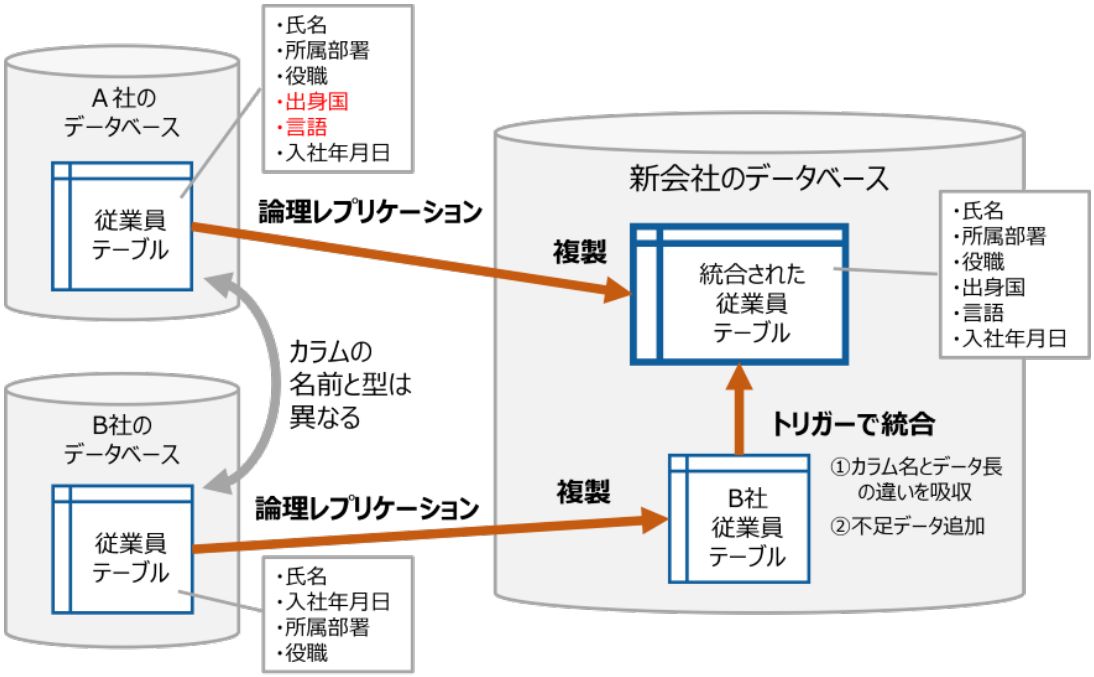

図 2：経営統合による従業員テーブルの動的な統合および整理

利用者起点での機能の十分性

PostgreSQL のバージョンアップごとに、論理レプリケーションに対する以下の機能追加や改善が実施されました。

- PostgreSQL 14 での論理レプリケーションの開発項目：実用レベルへの基盤作りを主とした開発
  システムビューpg_stat_replication_slots、二相コミット基盤、実行中の大規模トランザクションをストリーム、安定性向上
- PostgreSQL 15 での論理レプリケーションの開発項目：適応範囲を広げる機能を主に開発
  コンフリクト対処、行フィルター、列フィルター、スキーマフィルター、通信の改善、二相コミット
- PostgreSQL 16 での論理レプリケーションの開発項目：転送方式と性能に関する機能を主に開発
  ストリーミングレプリケーションのスタンバイからの論理レプリケーション（パブリケーション）、双方向論理レプリケーション、サブスクライバーの性能向上
- PostgreSQL 17 での論理レプリケーションの開発項目：運用改善に関する機能を主に開発
  フェイルオーバー制御、pg_createsubscriber による論理レプリカの作成、論理レプリケーションスロットが pg_upgrade に対応

ストリーミングレプリケーションとの違い

PostgreSQL のレプリケーションには、ストリーミングレプリケーションと論理レプリケーションがあります。PostgreSQL 15 で利用できるストリーミングレプリケーションと論理レプリケーションの違いを以下の表にまとめます。

表 1 ストリーミングレプリケーションと論理レプリケーションの違い

比較項目	ストリーミングレプリケーション	論理レプリケーション
利用例	高可用性、災害対策(ディザスタリカバリー)、参照負荷分散、バックアップ	特定データ（テーブル）の複製、複数のデータベースの情報を統合／整理、運用中の PostgreSQL のメジャーバージョンアップ、異なる PostgreSQL バージョン間のレプリケーション、異なる OS 間のレプリケーション(例：Windows から Linux)、バックアップ
対応バージョン	PostgreSQL 9.0 以降	PostgreSQL 10 以降
レプリケーション対象	すべてのオブジェクト	テーブルのみ
レプリケーション単位	データベースクラスタ	単一テーブル、複数テーブル、全テーブル、 スキーマ内のすべてのテーブル（PostgreSQL 15 以降）、複数スキーマ、行/カラム（PostgreSQL 15 以降）
レプリケーションされる操作	WAL が出力されるすべての操作	INSERT、UPDATE、DELETE、 TRUNCATE（PostgreSQL 11 以降）、COMMIT／ROLLBACK ※INSERT、UPDATE、DELETE、TRUNCATE に関してはレプリケーション対象の操作を選択可能
同期	同期／非同期（デフォルト）	同期／非同期（デフォルト）
転送データ	WAL レコード（トランザクションログ）	WAL をデコードした論理的な情報
双方向のデータレプリケーション	不可	可（PostgreSQL 16 以降）
レプリケーション元とレプリケーション先の環境	同一 OS、同一メジャーバージョン	同一／異なる OS、同一／異なるメジャーバージョン
レプリケーション先ノードのアクセス	参照のみ	参照、更新（更新時は制約違反によるコンフリクトに注意）
レプリケーション先トリガー	不可	可（レプリケーション先テーブルにトリガーの設定が可能）

備考.上記表内の情報は今後のバージョンアップで変わる可能性があります。

2. 論理レプリケーションの仕組み

論理レプリケーションの複製や内部動作の流れを把握することで、論理レプリケーションを安全に業務へ導入できます。また、構築可能な構成を押さえておくことで、無駄のないシステムを設計できます。
ここでは、論理レプリケーションの採用できる構成から内部の仕組みについて説明します。

基本構成

論理レプリケーションの仕組みを理解するために、何を転送するのか、どのように転送するのかを、詳しく説明します。

構成要素

論理レプリケーションは、1 つ以上のパブリケーション、1 つ以上のサブスクリプション、レプリケーションの状態を保持するオブジェクトであるレプリケーションスロット、複製元のテーブル、複製先のテーブルで構成されます。

以下の図に示した構成を基本として、カスケード構成など様々な構成を採用できます。採用可能な構成に関しては、次節で説明します。

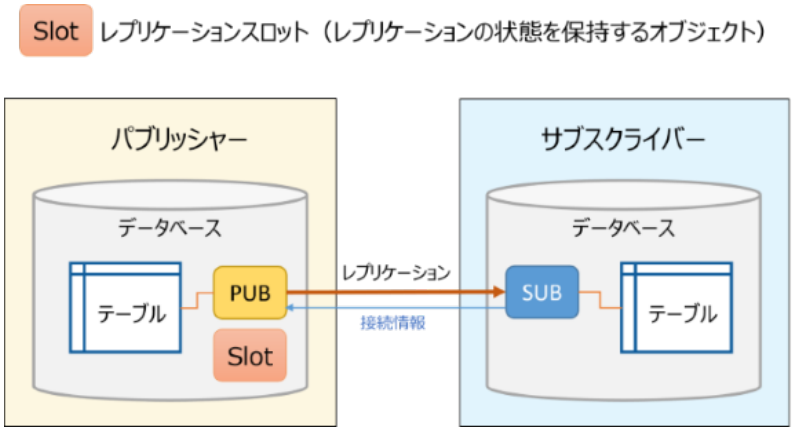

図 3：論理レプリケーションの基本構成

レプリケーションの流れ

PostgreSQL は、更新情報をトランザクションログ（以降、WAL と略します）として保存します。論理レプリケーションでは、この WAL をデコードし、パブリッシャー内にある walsender プロセスがサブスクライバーへ転送します。そして、サブスクライバー内にある適用ワーカーがその WAL を適用します。

論理レプリケーションの流れにおいて、初回時とそれ以降で送信対象が異なります。初回時は、パブリッシャーの対象データのスナップショットを採取し、データを送信します（初期データ同期）。初回時以降、デコードされた WAL を同期または非同期でサブスクライバーへ送信します。

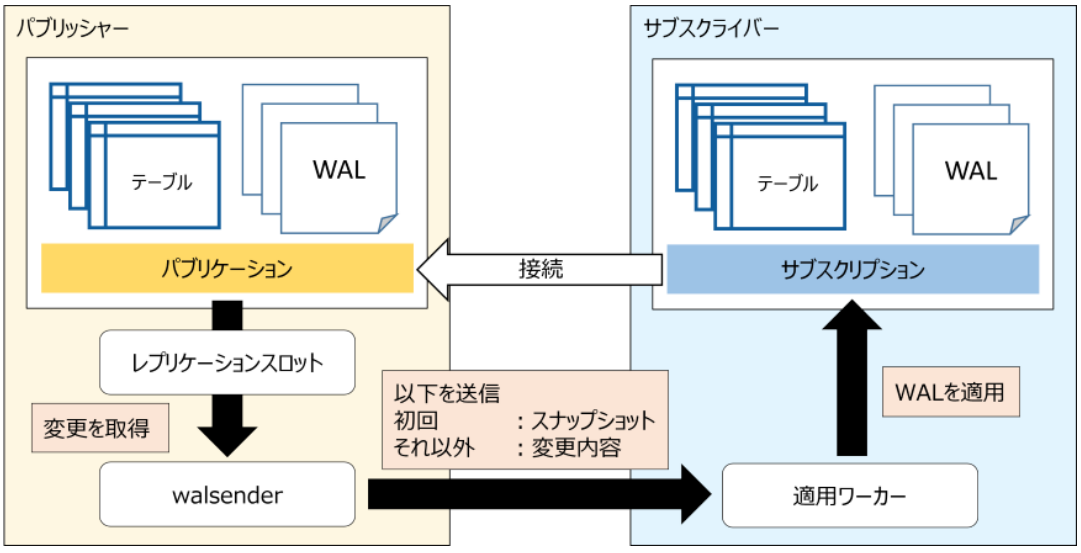

図 4：レプリケーションの流れ

内部の仕組み

論理レプリケーションの内部の仕組みに関しては、以下の記事で解説しています。walsender や適用ワーカーの内部動作に関して図を用いてわかりやすく説明していますので、内部の仕組みをさらに知りたい人は、ぜひお読みください。

- スキーマ内のテーブルを対象とした論理レプリケーション
- 論理レプリケーションのテーブル同期ワーカー

構築可能な構成

論理レプリケーションで構築可能な構成、および構築不可能な構成について説明します。

N 対 N とカスケードは可能

論理レプリケーションは、「テーブルに対して N 対 N」および「カスケード」の構成で構築できます。

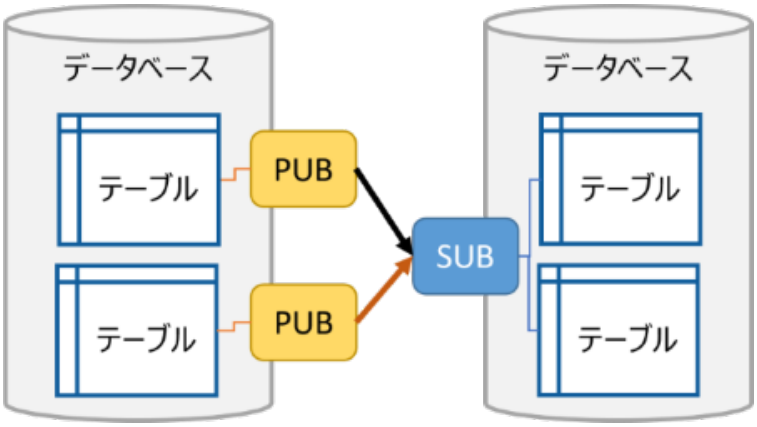

図 5：N 対 N の構成

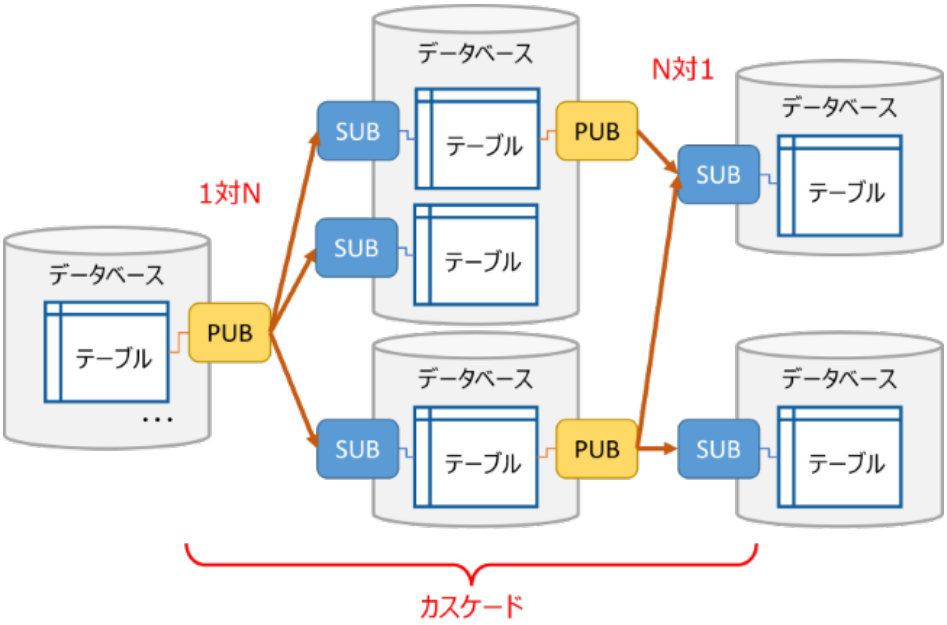

図 6：カスケード構成

パブリケーションは、「1 テーブルに対して複数のパブリケーション」、「複数テーブルに対して 1 パブリケーション」、「複数スキーマに対して 1 パブリケーション」のようにレプリケーション対象のテーブルを指定できます。

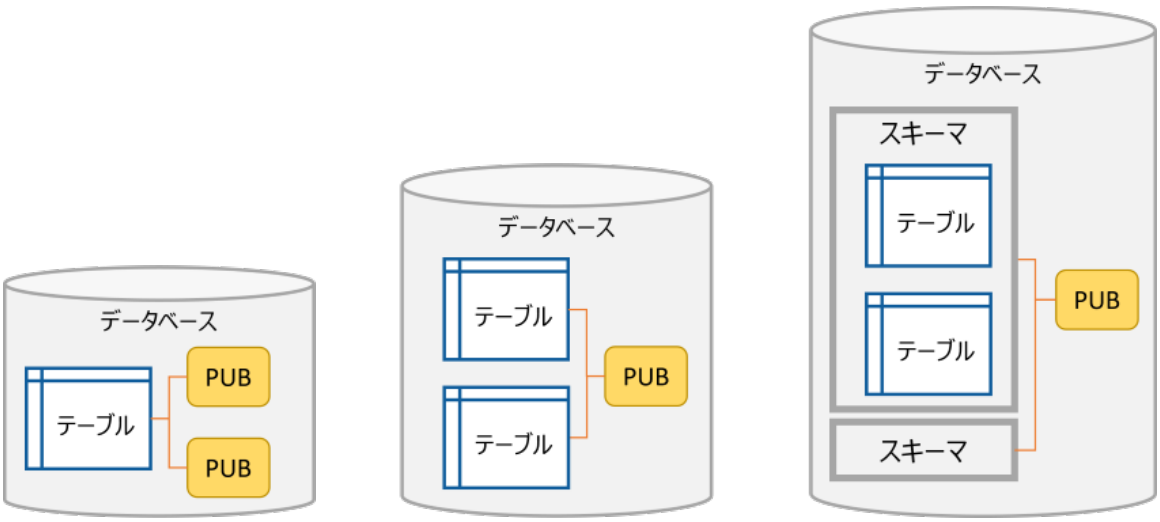

図 7：パブリケーションと対象テーブルの関係

サブスクリプションは、1 つあるいは複数のパブリケーションへの接続情報を指定するだけであるため、対象テーブルは接続するパブリケーションに依存します。

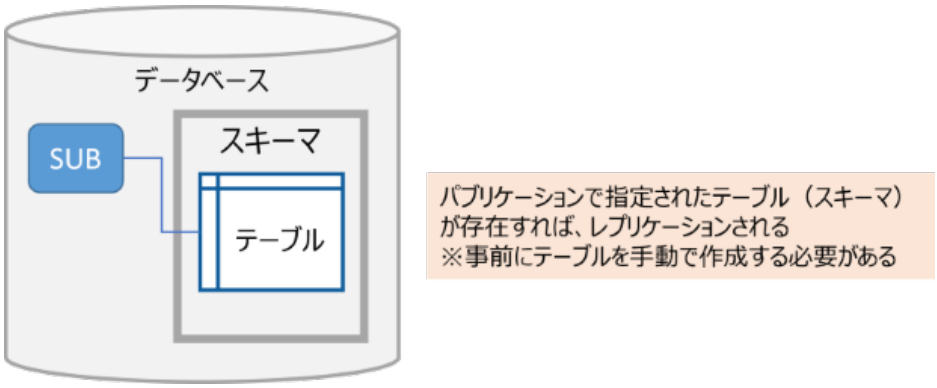

図 8：サブスクリプションと対象テーブルの関係

双方向は可能（PostgreSQL 16 以降）

論理レプリケーションで、「双方向（マルチマスタ）」および「ループ」の構成で構築できます。
双方向のレプリケーションの詳細や注意事項については、「origin パラメーターによる双方向の論理レプリケーション」を参照してください

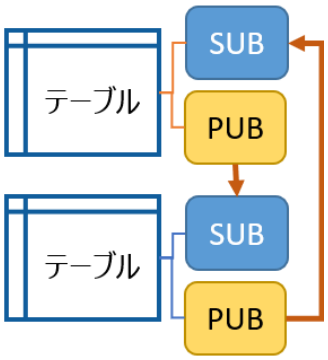

図 9：双方向／ループ構成

同一テーブルの複数サブスクリプション設定は不可

複数のサブスクリプションから、同一のテーブルへレプリケーションすることができません。

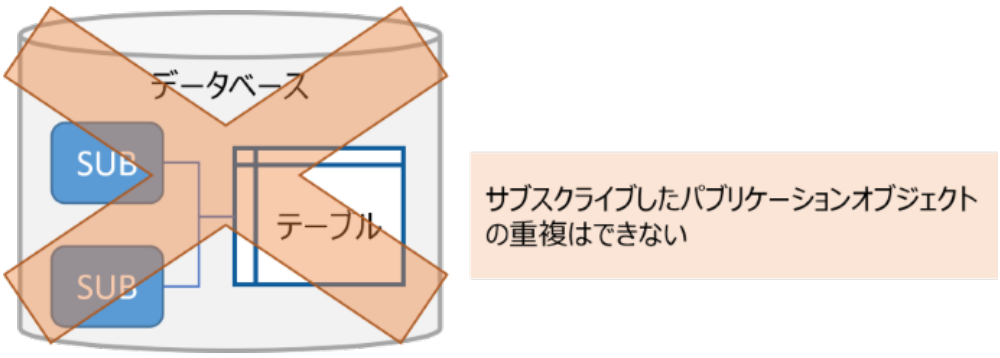

図 10：サブスクリプションを設定できないパターン

3. 設定方法と基本的な使い方

論理レプリケーションを使い、テーブル単位やスキーマ単位をレプリケーションする方法について順を追って、見ていきましょう。なお、今回は PostgreSQL 17 で設定方法を説明します。

設定手順

シンプルな構成、かつ、簡単なサンプルを使用して論理レプリケーションを使ってみましょう。サンプルで使用するデータベースは「postgres」とします。以下の構成で論理レプリケーションを設定します。

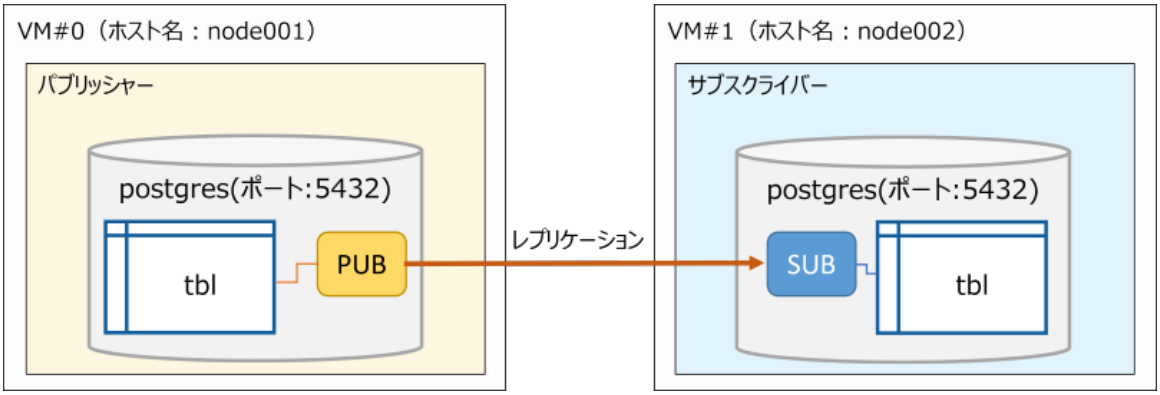

図 11：設定する論理レプリケーションの環境

上記の環境を想定として、以降で以下の手順を説明します。

- 1. パブリッシャー側の設定
- 2. サブスクライバー側の設定
- 3. 動作確認

1. パブリッシャー側の設定

1. まずは、postgresql.conf ファイルの wal_level を「logical」に変更します。

```
wal_level = 'logical'
```

2. 次に、サブスクライバーがパブリッシャーに接続できるように pg_hba.conf を設定します。
   今回は、検証のため以下のように「全許可」かつ「パスワードなし」を設定します。

```
host    all             all             0.0.0.0/0            trust
```

3. 続いて、レプリケーション用のテーブルとデータを作成しましょう。
   本検証では、以下のコマンドを実行しテーブルとデータを作成します。

```
postgres=# CREATE TABLE tbl(id INTEGER PRIMARY KEY, data TEXT);
postgres=# INSERT INTO tbl VALUES(1, 'TOKYO'),(2, 'OSAKA'),(3, 'NAGOYA'),(4, 'SHIZUOKA'),(5, 'KANAGAWA');
```

4. 次に、パブリケーションを作成するために以下のコマンドを実行してください。

```
postgres=# CREATE PUBLICATION pub FOR TABLE tbl;
```

作成したパブリケーションは、以下のいずれかのコマンドで確認できます。

```
postgres=# SELECT * FROM pg_publication;
postgres=# ¥dRp
postgres=# SELECT * FROM pg_publication_tables;
```

5. 最後に、レプリケーション用のロール作成とレプリケーション対象のテーブルに対して SELECT 権限を付与してください。
   本検証では、「logi_repl_user」というロールを以下のコマンドで作成し権限を付与します。

```
postgres=# CREATE ROLE logi_repl_user LOGIN REPLICATION PASSWORD 'logi_repl_user';
postgres=# GRANT SELECT ON tbl TO logi_repl_user;
```

参考

対象テーブルへの SELECT 権限を付与しなかった場合、サーバーログに以下のエラーが出力され、初期データ同期が行われません。

```
ERROR:  could not start initial contents copy for table "public.tbl": ERROR:  permission denied for table tbl
```

2. サブスクライバー側の設定

1. まずは、以下のようにサブスクリプション用のテーブルを作成しましょう。

```
postgres=# CREATE TABLE tbl(id INTEGER PRIMARY KEY, data TEXT);
```

2. 次に、サブスクリプションを作成するために以下のコマンドを実行してください。

```
postgres=# CREATE SUBSCRIPTION sub CONNECTION 'host=node001 port=5432 user=logi_repl_user dbname=postgres' PUBLICATION
pub;
```

参考

サブスクリプションを作成する前に対象テーブルを作成していない場合、以下のエラーが発生します。

```
ERROR:  relation "public.tbl" does not exist
```

サブスクリプションを作成すると、最初に初期データ同期が行われます。作成したサブスクリプションは、以下のいずれかのコマンドで確認できます。pg_subscription および pg_stat_subscription を使った確認方法に関しては、「状態を確認」を参照してください。

```
postgres=# SELECT * FROM pg_subscription;
postgres=# ¥dRs
postgres=# SELECT * FROM pg_stat_subscription;
postgres=# SELECT * FROM pg_subscription_rel;
```


3. 動作確認

1. 論理レプリケーションの動作を確認するために、パブリッシャー側で以下のようにデータを追加しましょう。

postgres=# INSERT INTO tbl VALUES (6, 'HYOGO');	

2. 実行後、サブスクライバー側でデータが追加されているかを確認するために、以下のコマンドを実行してください。

```
postgres=# SELECT * FROM tbl;
id  |   data
----+----------
  1 | TOKYO
  2 | OSAKA
  3 | NAGOYA
  4 | SHIZUOKA
  5 | KANAGAWA
  6 | HYOGO
```

パブリッシャー側で追加したデータがサブスクライバー側に存在する場合、論理レプリケーションを設定できたと判断できます。

状態を確認

サブスクライバー側で以下のコマンドを実行し、システムカタログ pg_subscription の subenabled が f である場合、サブスクリプションが無効な状態であることを把握できます。

```
postgres=# SELECT subname, subenabled FROM pg_subscription;
-[ RECORD 1 ]----+-----------------------------------------------------------
subname          | sub
subenabled       | f
```

また、サブスクライバー側で以下のコマンドを実行し、システムカタログ pg_stat_subscription の pid が空、received_lsn が空である場合、サブスクリプションが無効な状態であることを把握できます。

```
postgres=# SELECT subname, pid, received_lsn FROM pg_stat_subscription;
-[ RECORD 1 ]---------+------
subname              | sub
pid                  |
received_lsn         |
```

レプリケーションの停止と再開

サブスクライバー側でサブスクリプションを無効化することで、論理レプリケーションを停止できます。以下のコマンドをサブスクライバー側で実行することで、サブスクリプションを無効化できます。

ただし、運用中に一時停止すると、その間はパブリッシャー側で WAL が溜まります。再開時には停止時点からレプリケーションが再開されるため、データ同期までに時間がかかります。

```
postgres=# ALTER SUBSCRIPTION sub DISABLE;
```


論理レプリケーションを再開したい場合、サブスクライバー側でサブスクリプションを有効化しましょう。以下のコマンドをサブスクライバー側で実行することで、サブスクリプションを有効化できます。確認方法に関しては、「状態を確認」を参照してください。

```
postgres=# ALTER SUBSCRIPTION sub ENABLE;
```

不足しているテーブル情報をパブリッシャーから取得

論理レプリケーションをすべてのテーブルを対象に設定した後（パブリッシャー作成時に FOR ALL TABLES を指定した場合など）に、パブリケーションにテーブルを追加、または、パブリケーションからテーブルを削除した場合、自動でサブスクライバー側に伝わりません。そのため、サブスクライバー側で以下の REFRESH PUBLICATION の実行が必要です。実行することで、パブリッシャーからテーブルの情報を読み直し、追加されたテーブルのレプリケーションなどが行われるようになります。ちなみに、パブリケーションにテーブルを追加した場合、サブスクライバー側に追加したテーブルを手動で作成する必要があります。

```
postgres=# ALTER SUBSCRIPTION <サブスクリプション名> REFRESH PUBLICATION;
```

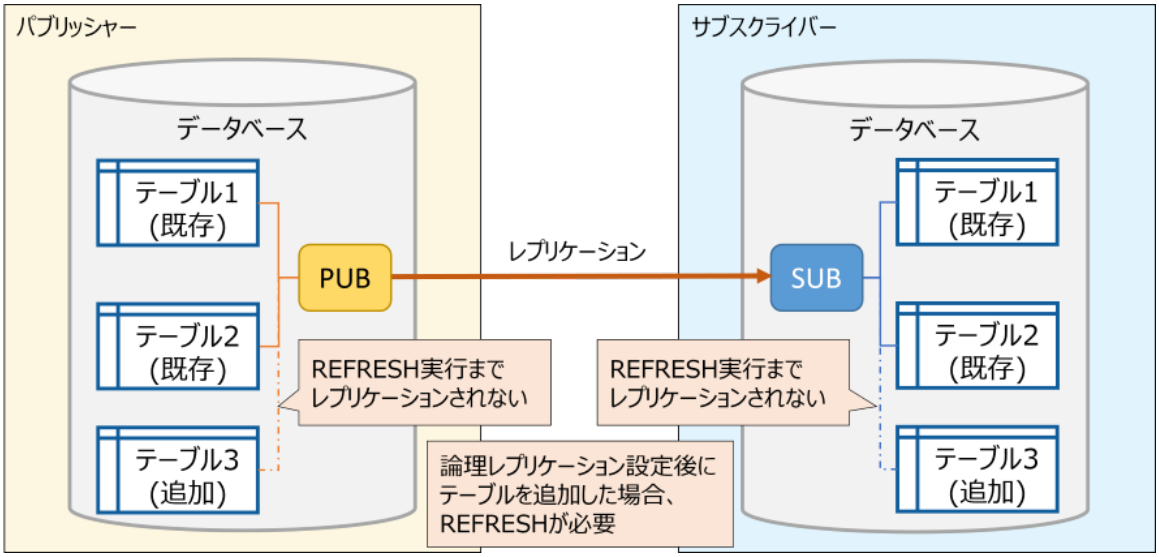

図 12：REFRESH 実行の必要性

フィルター機能

論理レプリケーションにはフィルター機能があります。フィルター機能を使うことで、指定した条件を満たすデータのみレプリケーションされるよう制限できます。
フィルター機能の種類として、列フィルター、行フィルター、スキーマフィルターがあります。以降で、各フィルターの機能概要と設定方法に関して説明します。

列フィルター

列フィルターの機能概要と設定方法については、以下の記事で解説しています。図やコマンド例を用いてわかりやすく説明していますので、ぜひお読みください。

- 論理レプリケーションにおける列リスト

行フィルター

行フィルターの機能概要と設定方法については、以下の記事で解説しています。図やコマンド例を用いてわかりやすく説明していますので、ぜひお読みください。

- 論理レプリケーションにおけるパブリケーションの行フィルター

スキーマフィルター

スキーマフィルターの機能概要と設定方法については、以下の記事で解説しています。図やコマンド例を用いてわかりやすく説明していますので、ぜひお読みください。

- スキーマ内のテーブルを対象とした論理レプリケーション

4. 利用時に注意するべきポイント

論理レプリケーション機能を利用する場合に注意するべき 5 つのポイントに関して説明します。

パブリケーション

パブリケーションを定義する場合、以下の注意が必要です。

- パブリケーションは 1 つのデータベース内の 1 つ以上のテーブルを対象として定義すること
- パブリケーションで行フィルターや列フィルターの指定および UPDATE 文と DELETE 文を実行する場合、レプリカアイデンティティが必須

サブスクリプション

サブスクリプションを定義する場合、以下の注意が必要です。

- サブスクリプションは他のデータベースへの接続と、サブスクリプション対象の 1 つ以上のパブリケーションの集合を定義すること
- サブスクリプションを作成する際に、あらかじめ複製先テーブルの作成が必要
- レプリケーション用のユーザーには、REPLICATION 属性、LOGIN 属性設定、および、対象テーブルへの SELECT 権限が必要。さらに、スキーマを指定する場合は、対象スキーマへの USAGE 権限が必要

レプリケーション対象のテーブル

論理レプリケーションを利用する場合、パブリッシャーとサブスクライバーの間ではテーブル照合が実施されます。テーブル照合は、完全修飾されたテーブル名（例：スキーマ名.テーブル名）に基づいて行われます。このため、パブリッシャーとサブスクライバー間で異なる名前になっているテーブルに対してレプリケーションはできません。
その他に押さえておくべきテーブル観点のポイントを以下にまとめます。

- サブスクライバー側でレコードを格納する際、パブリッシャーから転送されないカラムの値は、カラム定義に従ってデフォルト値（指定が無い場合は NULL）が挿入される
- パブリッシャーとサブスクライバー間でテーブルのカラム名が一致していること
- パブリッシャーとサブスクライバー間でテーブルのカラムの順序は一致している必要はない
- サブスクライバー側で定義するテーブルは、パブリッシャー側のテーブルに無いカラムが存在してもよく、また、パブリッシャー側のテーブル内のすべてのカラムを指定する必要もない
- カラムのデータ型は、PostgreSQL 内部で変換可能である限り一致している必要はない

- レプリケーション中でもカラム追加や削除は可能。例えば、パブリッシャー側で 1 カラム削除した場合、サブスクライバー側へ正しくレプリケーションができる（削除したカラムには NULL 値が格納）

レプリケーションスロット

論理レプリケーションでのデータ送信の記録には、レプリケーションスロットを使用します。障害からの復旧をする上でデータ送信の記録は重要です。サブスクリプションを作成すると自動的にレプリケーションスロットがパブリケーション側に作成されます。任意の名称でレプリケーションスロットを作成したい場合、サブスクリプション作成時のパラメーターで指定してください。

UPDATE 文と DELETE 文のレプリケーション

UPDATE 文と DELETE 文を論理レプリケーションでレプリケーションするためには、利用するカラムにレプリカアイデンティティ（主キー、ユニークキー）の指定が必要です。

参考

利用するカラムにレプリカアイデンティティを指定せずに、UPDETE 文や DELETE 文を実行すると以下のエラーが発生します。

```
postgres=# UPDATE tbl SET area = 4000.0 WHERE id = 1;
ERROR:  cannot update table "tbl" because it does not have a replica identity and publishes updates
HINT:  To enable updating the table, set REPLICA IDENTITY using ALTER TABLE.
```

```
postgres=# DELETE FROM tbl;
ERROR:  cannot delete from table "tbl" because it does not have a replica identity and publishes deletes
HINT:  To enable deleting from the table, set REPLICA IDENTITY using ALTER TABLE.
```

ALTER TABLE で該当カラムにレプリカアイデンティティを指定することで上記エラーを解消できます。

ここまで説明してきましたが、論理レプリケーションは比較的簡単な手順で利用できることがおわかりいただけたと思います。注意事項に気を付けて、業務要件に合わせてご利用をご検討ください。

なお、論理レプリケーションには、冗長化の範囲がテーブル内のデータのみで、可用性全般を担保する機構がありません。そのため、実際の業務においては、ストリーミングレプリケーションと論理レプリケーションを併用するケースが多いと考えられます。併用理由やフェイルオーバー時の復旧方法については、「論理レプリケーションの復旧方法 ～ ストリーミングレプリケーションのフェイルオーバー併用時 ～」を参照してください。

2025 年 3 月 11 日