



PGECons
PostgreSQL Enterprise Consortium

2022年度WG1活動成果報告 新機能検証編 PostgreSQLロジカルレプリケーション検証

**PostgreSQL エンタープライズコンソーシアム
WG1 新機能検証チーム**

Contents

1. ロジカルレプリケーション新機能検証
2. マルチマスタ検証

責任範囲

- 本資料は、PGECconsが独自に検証した結果であり、結果はPGECconsの責任の元、公開しています。

ロジカルレプリケーション新機能検証

検証概要

■ 目的

- PostgreSQL 15で強化されたロジカルレプリケーション新機能の調査
- 新機能のうち、仕様が大きく変更されロジカルレプリケーションの使用
方法や利用シーンに影響する機能を確認

■ 検証内容

- PostgreSQL 15で新規に追加されたロジカルレプリケーション機能のうち、ユーザへの影響が大きい変更である以下の機能の動作を確認
 - 列指定機能
 - 行フィルタ機能
- PostgreSQL 15で新規に追加されたロジカルレプリケーション機能利用時に使用する以下システムビューの動作を確認
 - pg_stat_subscription_stats

PostgreSQL15

ロジカルレプリケーション追加機能一覧

- PostgreSQL15でロジカルレプリケーションの追加機能は以下となる。

機能名	機能概要
列指定	ロジカルレプリケーションする列をパブリッシャーの特定列に制限できる
行フィルタ	WHERE句を使用してパブリッシャーのコンテンツをフィルタリングできるようにする
スキーマ内の全テーブル指定	スキーマ内のすべてのテーブルの公開を許可
LSNスキップ	ALTER SUBSCRIPTION ... SKIPでサブスクライバーでのトランザクションのスキップを許可する
二相コミット対応	準備済み (2 フェーズ) トランザクションのサポートを論理レプリケーションに追加
空のトランザクションのレプリケーションを防止	空のトランザクションの論理レプリケーションを防止
論理レプリケーションスロットのディレクトリの内容を監視するSQL関数	論理レプリケーションスロットのディレクトリの内容を監視するSQL関数を追加 ・ pg_ls_logicalsnapdir() ・ pg_ls_logicalmapdir() ・ pg_ls_replslotdir()
エラー時のSUBSCRIPTION無効化	サブスクライバー側オプションでサブスクライバーエラー時にロジカルレプリケーションの変更適用を停止できる
サブスクライバー サーバー変数をパブリッシャーと一致するように調整 (datetime と float8 の値)	datetime と float8 の値が一貫して解釈されるように、サブスクライバー サーバー変数をパブリッシャーと一致するように調整
pg_stat_subscription_statsの追加	サブスクライバーのアクティビティをレポートするシステム ビューを追加 ・ pg_stat_subscription_stats
pg_stat_reset_subscription_stats()	上述の統計カウンターをリセット
pg_publication_tables	システム ビューでの重複エントリの抑制

- 当該資料では、上記表中の**青字**で記載した「列指定」「行フィルタ」および「pg_stat_subscription_statsの追加」について説明する。

PostgreSQL 15

ロジカルレプリケーションメイン機能

- PostgreSQL 15ではロジカルレプリケーション機能が大幅に強化されている。
- その中でも、レプリケーションの最小単位がテーブル単位という既存の仕様から大きく変更が加わり、ユーザーニーズに合ったより細かな単位のレプリケーションを可能とした以下機能にフォーカスをあて検証を実施
 - 列指定
 - 行フィルタ
- 以下のビューが追加されたことにより、ロジカルレプリケーション運用時の監視効率化を見込めるため、動作を検証
 - pg_stat_subscription_stats

検証環境

- 今回の検証では、Virtual Box上でLinuxマシン環境を構築し検証を実施した。
- 下記の条件で2台のVMを作成しロジカルレプリケーションを構築している。

■ VMスペック

CPU数	メモリ (GiB)	ストレージサイズ(GiB)
1	2	16GB

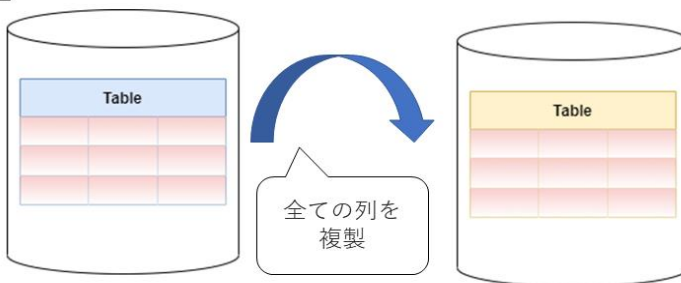
■ 各種バージョン情報

名称	バージョン
Rocky Linux	8.7 (Green Obsidian)
PostgreSQL	15.1

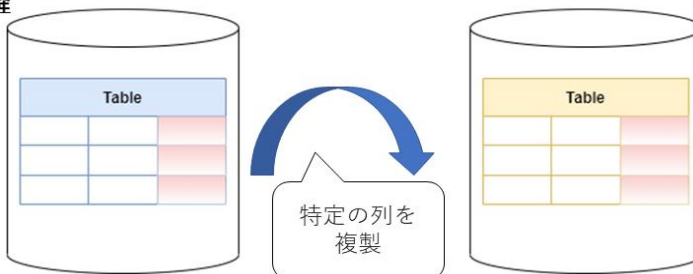
列指定 機能概要

- 列指定とは、レプリケーションをするテーブルのうち、特定の列だけをレプリケーションする機能。
- 以前はテーブル定義の全部の列のレプリケーションしかできなかったが、 当該機能の追加によりテーブル内の特定列のみを指定したレプリケーションが可能となった。

・ PostgreSQL14まで



・ PostgreSQL15以降



*赤い箇所がレプリケーション対象

列指定 機能概要

- 列指定のロジカルレプリケーションは、パブリケーションを定義するときに、レプリケーション対象列を指定することで機能する。

```
# CREATE PUBLICATION name  
FOR TABLE table_name(column_name_1 , column_name_2 );
```

テーブル名以下に列名を記載することで、
記載した列のみをレプリケーションする

指定する列には、「主キー」を含まないといけない。

列指定 実行例

- 下記のようなテーブルを作成し、主キーとなる”id”列と”a”列を指定し、レプリケーションする。

■ パブリッシャー側テーブル

id	a	b
1	test-a-1	test-b-1
2	test-a-2	test-b-2
3	test-a-3	test-b-3
4	test-a-4	test-b-4

■ サブスクライバー側テーブル

id	a	b
1	test-a-1	
2	test-a-2	
3	test-a-3	
4	test-a-4	



id列・a列を指定してレプリケーション
(b列はレプリケーションしない)

列指定 検証結果

- 列指定はパブリッシャー側で対象列を定義する
- パブリッシャー側で指定する列には、主キー列を含める必要がある
- サブスクライバー側で主キーが設定されていない場合、UPDATEが動作しない
- パブリッシャー側とサブスクライバー側の列名は一致する必要がある。(型変換されるため、型は異なっても動作する)

パブリッシャー側とサブスクライバー側それぞれで、指定列を主キーとした場合/していない場合の関係性は以下となる。

		サブスクライバー側	
		主キーを列指定する	主キーを列指定しない
パブリッシャー側	主キーを列指定する	INSERT :できる UPDATE:できる	INSERT :できる UPDATE:できない
	主キーを列指定しない	INSERT :できない UPDATE:できない	INSERT :できない UPDATE:できない

列指定 実行例

- p_test_1にt1テーブルを作成し、パブリケーション設定を実行。

```
p_test_1=# CREATE TABLE t1(id int PRIMARY KEY, a text, b text);
```

```
CREATE TABLE
```

```
p_test_1=# CREATE PUBLICATION p1 FOR TABLE t1(id ,a);
```

```
CREATE PUBLICATION
```

```
p_test_1=# SELECT * FROM t1;
```

```
id | a | b
```

```
----+-----+-----
```

```
1 | test-a-1 | test-b-1
```

```
2 | test-a-2 | test-b-2
```

```
3 | test-a-3 | test-b-3
```

```
4 | test-a-4 | test-b-4
```

```
(4 行)
```

列指定 実行例

- s_test_1にt1テーブルを作成し、サブスクリプション設定を実行。

```
s_test_1=# CREATE TABLE t1(id int PRIMARY KEY, a text, b text);  
CREATE TABLE
```

```
s_test_1=# CREATE SUBSCRIPTION s1  
            CONNECTION 'host=192.168.162.17 port=5432 user=logi_user dbname=p_test_1'  
            PUBLICATION p1;
```

NOTICE: 発行サーバーでレプリケーションスロット"s1"を作成しました

```
CREATE SUBSCRIPTION
```

```
s_test_1=# SELECT * FROM t1 ORDER BY id;
```

```
id | a | b
```

```
----+-----+----
```

```
1 | test-a-1 |
```

```
2 | test-a-2 |
```

```
3 | test-a-3 |
```

```
4 | test-a-4 |
```

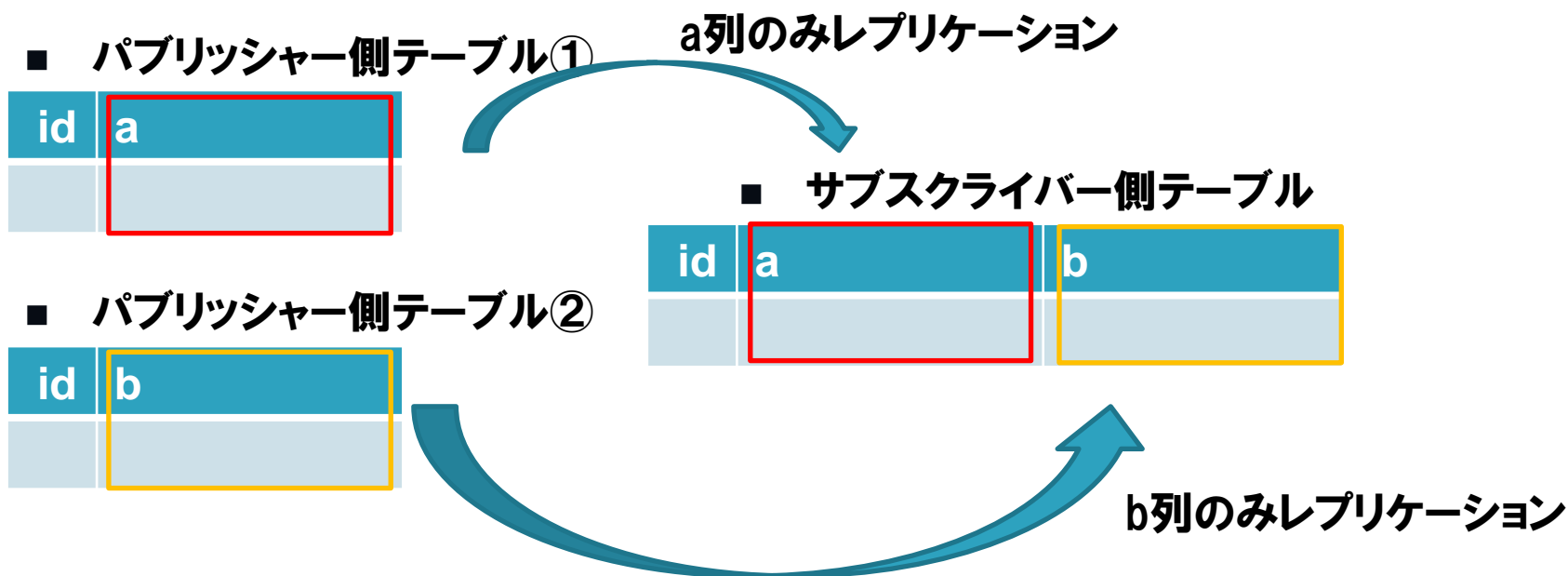
```
(4 行)
```

パブリッシャー側テーブルから、
id列・a列のみサブスクライバー側
テーブルへレプリケーション

列指定

複数パブリッシャー：単一サブスクライバーについて

- 列指定ロジカルレプリケーションが可能になったことで、
複数のパブリッシャー側の特定列を1つのサブスクライバー側にマージする
といった使い方が想定される。
- 複数パブリッシャー：単一サブスクライバー のイメージ



列指定

複数パブリッシャー: 単一サブスクライバーについて

- 現状では以下制約がある。
 - 列指定時にレプリケーション対象とする主キー列の値が被らない範囲でのみ正常に動作可能。
 - 列指定レプリケーションを正常に動作させるためには、レプリケーションする列に主キーを含める必要がある。
 - そのため、パブリッシャー側/サブスクライバー側で主キー列を指定する必要があるが、2つ目以降のパブリッシャー側では主キーの一意性によりサブスクライバーへの初回同期のINSERTでエラーとなる。

列指定

複数パブリッシャー：単一サブスクライバーについて

■ 成功例

- id列が主キー
- 主キー範囲が被っていないため、レプリケーションが動作する

■ パブリッシャー側テーブル①

id	a
1	aaaaa

id列・a列をレプリケーション

■ サブスクライバー側テーブル

id	a	b
1	aaaaa	
2		bbbbbb

■ パブリッシャー側テーブル②

id	b
2	bbbbbb

id列・b列をレプリケーション

列指定

複数パブリッシャー：単一サブスクライバーについて

■ 失敗例

- id列が主キー
- 主キー範囲が被ってるため、パブリッシャー②のレプリケーションは失敗

- パブリッシャー側テーブル① id列・a列をレプリケーション

id	a
1	aaaaa

- サブスクライバー側テーブル

id	a	b
1	aaaaa	

- パブリッシャー側テーブル②

id	b
1	bbbbbb

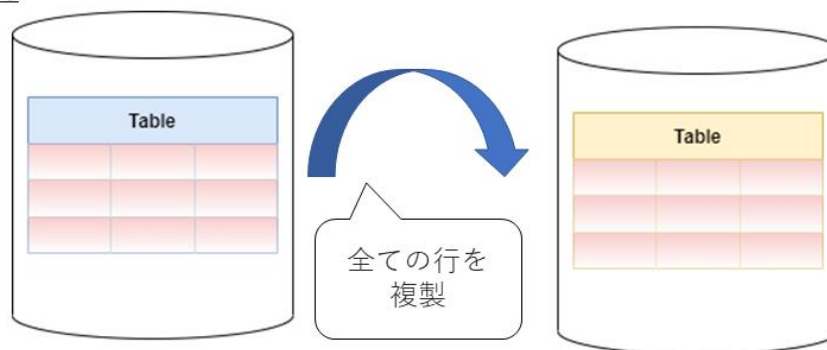
id列・b列をレプリケーション

★id列に”1”をINSERTしようと動作するが、一意性制約エラーにより更新ができない

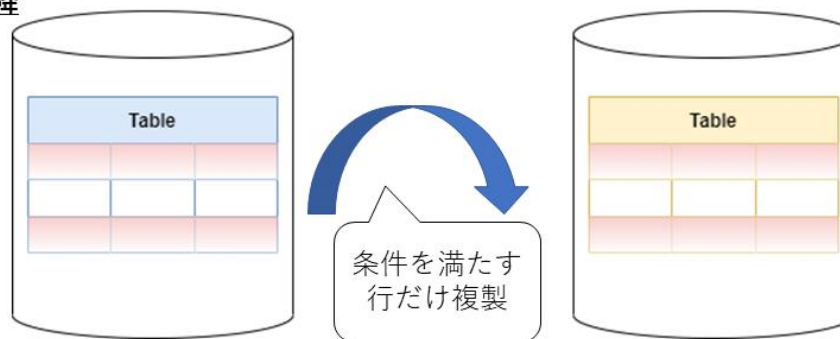
行フィルタ 機能概要

- 行フィルタ機能とは、WHERE句を用いて、レプリケーションするテーブルのうち、条件を満たしている行だけを複製する機能

・ PostgreSQL14まで



・ PostgreSQL15以降



*赤い箇所がレプリケーション対象

行フィルタ 機能概要

- 行フィルタのロジカルレプリケーションは、パブリケーションを定義するときに、レプリケーションの条件をWHERE句で指定することで機能する。

```
# CREATE PUBLICATION name  
FOR TABLE table_name WHERE (条件);
```

テーブル名以下にWHERE句で条件を記載することで、条件に合致した行のみをレプリケーションする

条件とする列は主キーとする必要がある

行フィルタ 実行例

- 下記のようなテーブルを作成し、WHERE句でa列が5以下の行のみ、レプリケーションする。

パブリッシャー側 / サブスクライバー側の両方とも、
id列とa列を主キーにして、ロジカルレプリケーション

※ (P) の列が主キー

■ パブリッシャー側テーブル

id (P)	a (P)	b
1	1	test-b-1
2	2	test-b-2
3	3	test-b-3
4	10	test-b-4
5	111	test-b-5
6	4	test-b-6

■ サブスクライバー側テーブル

id (P)	a (P)	b
1	1	test-b-1
2	2	test-b-2
3	3	test-b-3
6	4	test-b-6

a<5の行をレプリケーション

行フィルタ 実行例

- WHERE句のフィルタリング条件とするa列はサブスクライバー側で主キーでなくともINSERT/UPDATE可能。

パブリッシャー側はid列とa列を主キー

サブスクライバー側はid列のみ主キーにして、ロジカルレプリケーション

※ (P) の列が主キー

■ パブリッシャー側テーブル

id (P)	a (P)	b
1	1	test-b-1
2	2	test-b-2
3	3	test-b-3
4	10	test-b-4
5	111	test-b-5
6	4	test-b-6

■ サブスクライバー側テーブル

id (P)	a *	b
1	1	test-b-1
2	2	test-b-2
3	3	test-b-3
6	4	test-b-6

*サブスクライバーa列は主キーではない

a<5の行をレプリケーション

行フィルタ 検証結果

- 行フィルタは、パブリッシャー側でWHERE句を用いて条件を設定する
- WHEREでフィルタリングの条件とする識別列は、パブリッシャー側の主キーである必要がある
- サブスクライバー側のテーブルでは、識別列は主キーである必要はない

行フィルタ指定 実行例

- p_test_1にt1テーブルを作成し、パブリケーション設定を実行。

```
p_test_1=# CREATE TABLE t1(id int, a int, b text ,PRIMARY KEY (id,a));
```

```
CREATE TABLE
```

```
p_test_1=# CREATE PUBLICATION p1 FOR TABLE t1 WHERE (a < 5);
```

```
CREATE PUBLICATION
```

```
p_test_1=# table t1;
```

```
id | a | b ----+-----+-----
```

```
1 | 1 | test-b-1
```

```
2 | 2 | test-b-2
```

```
3 | 3 | test-b-3
```

```
4 | 10 | test-b-4
```

```
5 | 111 | test-b-5
```

```
6 | 4 | test-b-6
```

```
(6 行)
```


行フィルタ指定 実行例

- s_test_1にt1テーブルを作成し、サブスクリプション設定を実行。

```
s_test_1=# CREATE TABLE t1(id int ,a int,b text,PRIMARY KEY(id,a));
CREATE TABLE
s_test_1=# CREATE SUBSCRIPTION s1
      CONNECTION 'host=192.168.162.17 port=5432 user=logi_user dbname=p_test_1'
      PUBLICATION p1;
NOTICE: 発行サーバーでレプリケーションスロット"s1"を作成しました
CREATE SUBSCRIPTION
s_test_1=# table t1;
```

id	a	b
1	1	test-b-1
2	2	test-b-2
3	3	test-b-3
6	4	test-b-6

(4 行)

パブリッシャー側テーブルから、
a列の値が4未満の行のみサブスク
ライバー側テーブルへレプリケーショ
ン

システムビューpg_stat_subscription_stats

機能概要

- pg_stat_subscription_statsでは、ロジカルレプリケーションを組んでいる構成で、サブスクライバーを定義したDB側の情報が格納される。
- 本システムビューをサブスクライバー側で確認することで、下記の情報を取得できる。

Column	Type	Description
subid	oid	サブスクリプションのOIDです。
subname	name	サブスクリプションの名前です。
apply_error_count	bigint	変更の適用中にエラーが発生した回数です。
sync_error_count	bigint	初期テーブル同期中にエラーが発生した回数です。
stats_reset	timestamp with time zone	統計情報がリセットされた最終時刻です。

<https://pgsql-jp.github.io/current/html/monitoring-stats.html#MONITORING-PG-STAT-SUBSCRIPTION-STATS>
から引用

システムビューpg_stat_subscription_stats 実行例

- ロジカルレプリケーションが正常に動作している際に、サブスクライバー側でシステムビューを確認

```
s_test_1=# SELECT * FROM pg_stat_subscription_stats;
subid | subname | apply_error_count | sync_error_count | stats_reset
-----+-----+-----+-----+-----
49402 | s1      | 0                 | 0                 |
(1 行)
```

- サブスクライバー側でエラーを発生させると、実行結果が変化する。

```
s_test_1=# SELECT * FROM pg_stat_subscription_stats;
subid | subname | apply_error_count | sync_error_count | stats_reset
-----+-----+-----+-----+-----
49402 | s1      | 5                 | 0                 |
(1 行)
```

apply_error_countが加算されていることから、
パブリッシャー側で更新が発生したが、
サブスクライバー側では変更適用中にエラーが発生していることが確認できる。

システムビューpg_stat_subscription_stats 検証結果

- pg_stat_subscription_statsを使用することで、ロジカルレプリケーション時のエラー発生を、サーバログを見ることなく検知が可能になる。(※)
 - (※) サーバログを確認することで判別できるが、サーバログから自動検知は難しい。
 - PostgreSQL単体機能としてはもっていない。
 - 他の監視ツールでのサーバログのメッセージトラップが必要。
 - ログのトラップは重いため避けたほうが無難。
- PostgreSQL 15では、ロジカルレプリケーション時にサブスクライバー側でエラーが発生した際に、エラーを回避する手段が追加されている。
- ユーザーはpg_stat_subscription_statsのapply_error_countの数値変化を定期的に監視することで、エラーを検知し適切な回避を実施することができる。
- pg_stat_reset_subscription_statsで、pg_stat_subscription_statsの統計情報をリセットすることができる。
 - 定期的に apply_error_count をリセットすることで、一定期間内のロジカルレプリケーションエラーの統計情報を取得できる。

システムビュー pg_stat_subscription_stats

PostgreSQL 15で追加されたエラー回避手段

- LSNスキップ
 - サブスクライバーでのトランザクションのスキップを許可する
 - エラーが発生しているトランザクションをスキップしロジカルレプリケーションを継続する
 - `ALTER SUBSCRIPTION ... SKIP (LSN = 'lsn_value');`
- エラー時のSUBSCRIPTION無効化
 - サブスクライバー側エラー時にロジカルレプリケーションの変更の適用を停止する
 - サブスクライバーオプションに”`disable_on_error`”が追加され、サブスクライバー側エラー発生時には自動でロジカルレプリケーションを無効化し停止することが可能となった
 - PostgreSQL 14以前は、サブスクライバー側エラー発生時には、ユーザー側で”`ALTER SUBSCRIPTION ... DISABLE`”等を手動で実行し、ロジカルレプリケーションを無効化し停止させる必要があった
 - `CREATE SUBSCRIPTION ... WITH (disable_on_error = true);`
 - `ALTER SUBSCRIPTION ... SET (disable_on_error = true);`

まとめ

- PostgreSQL 15で追加されたロジカルレプリケーションの列指定・行フィルタを適切に用いることで、データベースサーバ間で必要な列・行だけを同期することができる。
- ネットワーク帯域幅の削減やパフォーマンス向上、より効率的で柔軟なデータ同期を実現することができる。
- システムビューpg_stat_subscription_statsを用いることで、ロジカルレプリケーション時のエラー発生を、サーバログを見ることなく検知可能
 - PostgreSQL 15のロジカルレプリケーション追加機能である以下を用いることでエラー回避を実現
 - LSNスキップ
 - エラー時のSUBSCRIPTION無効化

マルチマスタ検証

検証概要

■ 目的

- PostgreSQLの論理レプリケーション機能を用いてマルチマスタ・レプリケーションが可能かを検証
- PostgreSQL 16に追加予定のロジカルレプリケーション関連の新機能を用いて検証

■ 検証内容

- 基本更新パターン
- 衝突時の挙動

PostgreSQL 15までの問題点

- PostgreSQL 15までの機能を使うことで、同一のテーブルに対するパブリッシャーを定義し、相互に対向側のノードからサブスクライバーで接続するマルチマスタ構成は構築できた。
- しかし、以下の問題があるため、実際にはマルチマスタ構成として機能はしない。
 - WALの循環
 - INSERT WALが永遠に2ノード間でループする。
 - 更新の衝突
 - 2ノード間で同一PKに対する操作等を抑止する機能がない。

PostgreSQL 16の機能追加

- CREATE SUBSCRIPTIONコマンドに、origin オプションが指定可能になった
 - any : 起点以外からのパブリッシャーからも購読する (デフォルト、PostgreSQL 15までの挙動)
 - none : 起点のパブリッシャーのみ購読する
- origin = none でない場合(およびPostgreSQL 15まで)は同一テーブルに相互にパブリッシャー/サブスクライバーを設定すると、レプリケーションが循環するという致命的な問題があった。
- origin = none の指定により、レプリケーションの循環が抑止される。

PostgreSQL 16の機能追加

- replication originの機能自体は、ロジカルデコーディング基盤を使ってレプリケーションを構築するための手段として PostgreSQL 15以前でも実装されていた。
 - SQL関数 `pg_replication_origin_create()` で生成する。
 - `pg_replication_origin` システムビューで状態を確認可能
 - 他のreplication origin関数を使うことで、レプリケーションの循環を防止する実装はできた。
- PostgreSQL 16ではロジカルレプリケーションの作成コマンドとして、replication originが指定可能になった。

検証環境

■ インスタンススペック

名称	インスタンスタイプ	vCPU	メモリ (GiB)	ルーストレージサイズ (GiB) / IOPS	追加ストレージサイズ (GiB) / IOPS
DB server	t2.medium	2	4	32GB / 100	32GB / 100

- ☐ AWS EC2インスタンスを1台使用
- ☐ 性能検証ではないため低スペックのインスタンスを使用

名称	OS	バージョン	備考
DB server	Amazon Linux 2	PostgreSQL-devel	PostgreSQL 16相当

- ☐ PostgreSQLはmasterブランチから検証前のバージョンをcloneしビルドして実行
- ☐ configureオプションは--prefix=<dir name> のみ

検証環境

- 同一EC2インスタンス上でportを分けて2つのデータベースクラスタを配置
 - 2つのインスタンス間でロジカルレプリケーション

■ postgresql.conf (ノード1)

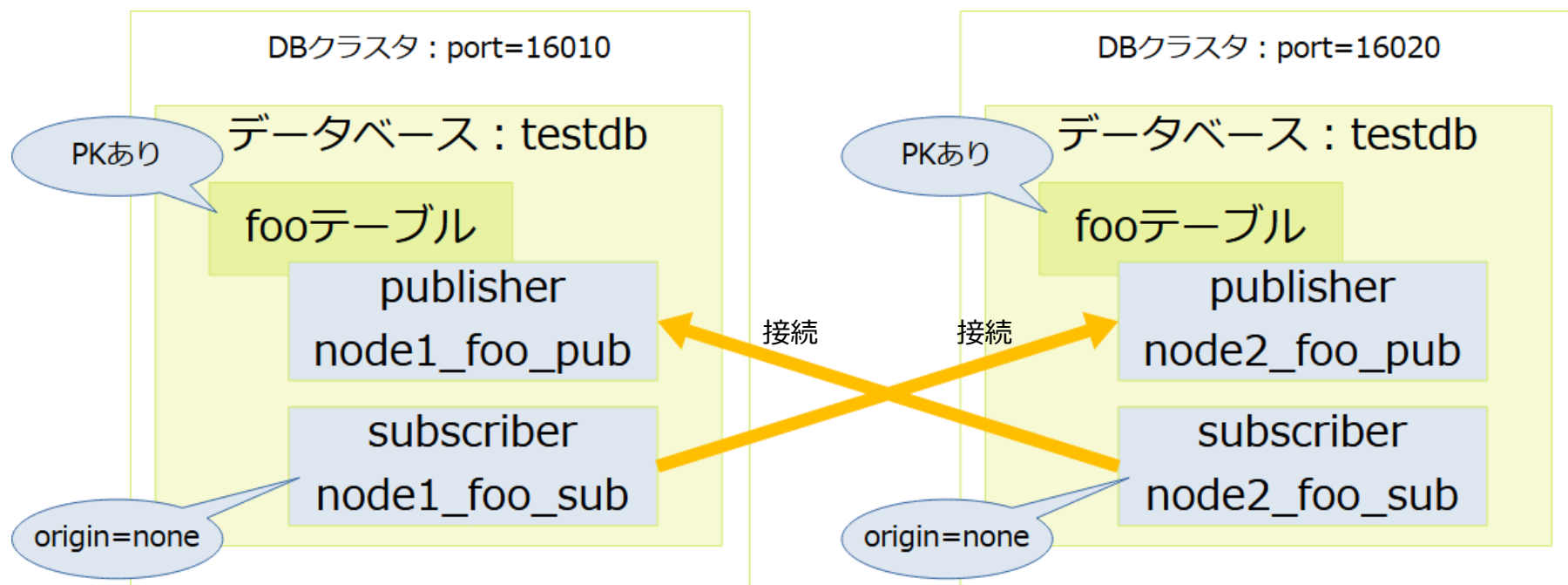
```
port = 16010  
wal_level = logical
```

■ postgresql.conf (ノード2)

```
port = 16020  
wal_level = logical
```

検証環境

- データベース名/テーブル名は同一
- fooテーブルにパブリッシャーを定義
- そのパブリッシャーに接続するサブスクライバーを定義
 - サブスクライバーオプションに `origin=none` を指定



検証環境(DDL例)

■ NODE1, NODE2共通

```
CREATE TABLE foo (id int primary key, data text);
```

■ NODE1

```
CREATE PUBLICATION node1_foo_pub FOR TABLE foo;
```

```
CREATE SUBSCRIPTION node1_foo_sub CONNECTION 'port=16020  
dbname=testdb user=postgres' PUBLICATION node2_foo_pub WITH  
(origin=none);
```

■ NODE2

```
CREATE PUBLICATION node2_foo_pub FOR TABLE foo;
```

```
CREATE SUBSCRIPTION node2_foo_sub CONNECTION 'port=16010  
dbname=testdb user=postgres' PUBLICATION node1_foo_pub WITH  
(origin=none);
```

検証内容

- 衝突が発生しないケースの確認
- 衝突が発生するケースの確認

衝突が発生しないケースの確認

- 同一IDのレコードの競合が発生しない状態で、相互に更新が伝播することを確認した。
 - INSERT
 - UPDATE
 - DELETE
 - TRUNCATE
- WALの循環防止の効果があつた。

衝突が発生するケースの確認

- 同一IDのレコードに対する更新が衝突するケースを確認した。
- 結論: 衝突が発生した場合には重大な問題が発生するケースがある。
 - 特に同一PKのINSERTはレプリケーション停止を引き起こす、かつ運用介入が必要になる。

パターン	系全体	node1	node2
同一PKのINSERT	値は不一致 レプリケーション停止 (運用介入が必要になる)	node1への挿入結果	node2への挿入結果
同一PKのUPDATE	値は不一致 (対向側の値が更新される)	node2への更新結果	node1への更新結果
同一PKのUPDATE/DELETE	値は一致	node2へのdelete結果	node2へのdelete結果
全行DELETEとINSERT	値は一致	全DELETE→node2へのinsert結果	全DELETE→node2へのinsert結果
TRUNCATEとINSERT	値は不一致	node2への挿入結果	TRUNCATE結果

衝突の例：同一PKのINSERT

- NODE1, NODE2に対して同一PK/別の列値で挿入する。
- 挿入は成功するが、NODE1とNODE2で異なる値が挿入される。
- また、この状態になると以降の更新処理がレプリケーションされなくなる。

衝突の例: 同一PKのINSERT

```
=# BEGIN;  
BEGIN  
=*# INSERT INTO foo VALUES (1, 'ABC');  
INSERT 0 1  
=*# TABLE foo;  
id | data  
----+-----  
 1 | ABC  
(1 row)  
=*# COMMIT;  
COMMIT  
=# TABLE foo;  
id | data  
----+-----  
 1 | ABC  
(1 row)  
=#
```

```
=# BEGIN;  
BEGIN  
=*# INSERT INTO foo VALUES (1, 'abc');  
INSERT 0 1  
=*# TABLE foo;  
id | data  
----+-----  
 1 | abc  
(1 row)  
=*# COMMIT;  
COMMIT  
=# TABLE foo;  
id | data  
----+-----  
 1 | abc  
(1 row)  
=#
```

衝突の例：同一PKのINSERT

- 前スライドの状態になると、レプリケーションされなくなるため、運用介入してWALをスキップする必要がある。
 - ALTER SUBSCRIPTION ...
SKIP (LSN = スキップするLSN)
 - これをNODE1, NODE2に対して実施する必要がある。
- なお、WALスキップの運用介入をしても、NODE1とNODE2でINSERTされた値が異なる問題は解決しない。
- 本来はどうあるべきか？
 - NODE1と同じPKを持つ行をNODE2から挿入しようとした場合には、NODE2からの挿入をロールバックすべきか？
 - 現状のPostgreSQL機能だけでは、このような防止はできない。

衝突の例：同一PKのINSERT

■ ALTER SUBSCRIPTIONによるWALスキップの例

node2 : サーバログ

```
2023-03-12 15:22:34.111 JST [3040] CONTEXT: processing remote data for replication origin "pg_16401"  
during message type "INSERT" for replication target relation "public.foo" in transaction 752, finished at  
0/1949D88
```

サーバログに出力されたLSN情報を元に
ALTER SUBSCRIPTIONコマンドを実行する

node1 : ALTER SUBSCRIPTIONコマンドの実行

```
=# ALTER SUBSCRIPTION node1_foo_sub SKIP ( LSN = '0/1949CC8'),  
ALTER SUBSCRIPTION  
=#
```

衝突の例：同一PKに対するUPDATE

- NODE1, NODE2から同一PKを持つレコードに対してUPDATEを行った場合、どちらのUPDATEも成功する。
- NODE1
 - NODE1に対するUPDATEの結果に対し、NODE2に対するUPDATEの結果が上書きされる。
- NODE2
 - NODE2に対するUPDATEの結果に対し、NODE1に対するUPDATEの結果が上書きされる。
- NODE1とNODE2の値は同一にならない。
- 本来はどうあるべきか？
 - 同一PKに対するUPDATEは抑止すべきか？

衝突の例: 同一PKに対するUPDATE

```
=# TABLE foo;  
id | data  
----+-----  
1  | ABC  
2  | DEF  
(2 rows)
```

```
=# BEGIN;  
BEGIN  
=*# UPDATE foo SET data = 'XYZ' WHERE id = 1;  
UPDATE 1  
=*# TABLE foo;  
id | data  
----+-----  
2  | DEF  
1  | XYZ  
(2 rows)  
=*# COMMIT;  
COMMIT  
=# TABLE foo;  
id | data  
----+-----  
2  | DEF  
1  | xyz  
(2 rows)
```

```
=# BEGIN;  
BEGIN  
=*# UPDATE foo SET data = 'xyz' WHERE id = 1;  
UPDATE 1  
=*# TABLE foo;  
id | data  
----+-----  
2  | DEF  
1  | xyz  
(2 rows)  
=*# COMMIT;  
COMMIT  
=# TABLE foo;  
id | data  
----+-----  
2  | DEF  
1  | XYZ  
(2 rows)  
=#
```


衝突の例: TRUNCATEとINSERT

- NODE1に対してTRUNCATE, NODE2に対してINSERTを行った場合、どちらの操作も成功する。
- NODE1
 - TUNCATEされたあと、NODE2に対して行ったINSERTの結果が反映される
- NODE2
 - NODE2に対して行ったINSERT後の結果に対してTRUNCATEが反映される
- NODE1とNODE2の値は同一にならない。
- 本来はどうあるべきか？
 - どちらかの操作を抑止すべきか？
 - TRUNCATE/INSERTのどちらを優先すべきか？

衝突の例: TRUNCATEとINSERT

```
=# TABLE foo;  
id | data  
----+-----  
 1 | ABC  
(1 row)
```

```
=# BEGIN;  
BEGIN  
=*# TRUNCATE foo;  
TRUNCATE TABLE  
=*# TABLE foo;  
id | data  
----+-----  
(0 rows)  
=*# COMMIT;  
COMMIT  
=#  
=# TABLE foo;  
id | data  
----+-----  
 2 | def  
(1 row)  
=#
```

TRUNCATE後、
defを挿入した
WALの反映

```
=# BEGIN;  
BEGIN  
=*# INSERT INTO foo VALUES (2, 'def');  
INSERT 0 1  
=*# TABLE foo;  
id | data  
----+-----  
 1 | ABC  
 2 | def  
(2 rows)  
=*# COMMIT;  
COMMIT  
=# TABLE foo;  
id | data  
----+-----  
(0 rows)  
=#
```

defを挿入後、
TRUNCATEの
WALの反映

まとめ

- PostgreSQL 16機能によるマルチマスタ構成の問題
 - WALの循環
 - originオプションにより、WALの循環は解決する。
 - 更新の総突
 - 更新の衝突は解決できていない。
 - 同一IDのINSERTケースはレプリケーションが停止し、運用者介入が必要になる。
 - 同一レコードのUPDATE/DELETE/TRUNCATE競合はレプリケーション停止にはならないが、想定外の結果となる。

まとめ

■ 結論

- PostgreSQL 16のSQL本体機能のみでは、実用的なマルチマスター構成は困難
- 衝突発生時のルールを定義して制御する機能が必要
 - 例：同一PK挿入時に片側の挿入をエラーにする等



PGECons

PostgreSQL Enterprise Consortium