
PGECons
PostgreSQL Enterprise Consortium

# 柔軟性がより向上した論理レプリケーションの活用について

## 2022年度PGECons 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/

NTTテクノクロス　金澤 竣平

# 責任範囲

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

# PostgreSQL15/PostgreSQL16
# ロジカルレプリケーション追加機能一覧

- PostgreSQL15/16のロジカルレプリケーション追加機能は以下となる。

| PostgreSQL 15 |
| --- |
| 列指定 |
| 行フィルタ |
| スキーマ内の全テーブル指定 |
| LSNスキップ |
| 二相コミット対応 |
| 空のトランザクションの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 |
| エラー時のSUBSCRIPTION無効化 |
| 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） |
| pg_stat_subscription_statsの追加 |
| pg_stat_reset_subscription_stats() |
| pg_publication_tables |

| PostgreSQL16 |
| --- |
| スタンバイでの論理デコードを許可 |
| 論理デコードパブリッシャが変更を転送する方法とサブスクライバが変更を適用する方法を制御するサーバー変数を追加 |
| 論理レプリケーションの初期テーブル同期で行をバイナリ形式でコピーできるようにしました |
| 論理レプリケーションの並列アプリケーションを許可 |
| 主キーを使用しない論理レプリケーション適用のパフォーマンスを向上 |
| 論理レプリケーション サブスクライバーがオリジンのない変更のみを処理できるようにする |
| テーブル所有者として論理レプリケーションSELECTとDMLアクションを実行 |
| wal_retrieve_retry_interval をサブスクリプションごとに動作させる |

- 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」について説明し、応用事例を示す。
- また、 マルチマスタ検証としてORIGINオプションを用いた検証を実施した結果について報告する。

# ロジカルレプリケーション新機能を使った応用事例

# PostgreSQL15
# ロジカルレプリケーションメイン機能

- PostgreSQL15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る
  - 列指定
  - 行フィルタ

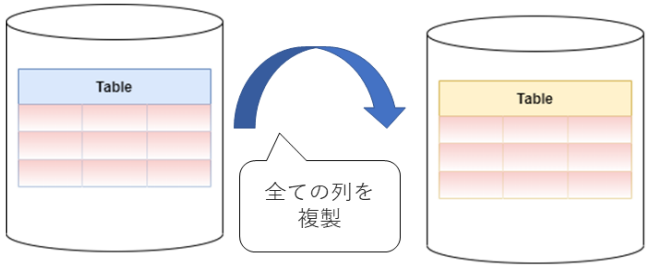
# 検証概要

- 目的
  - PostgreSQL15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
  - PostgreSQL15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、仕様が大きく変更された以下の機能について確認し、応用例を検討
    - 列指定機能
    - 行フィルタ機能

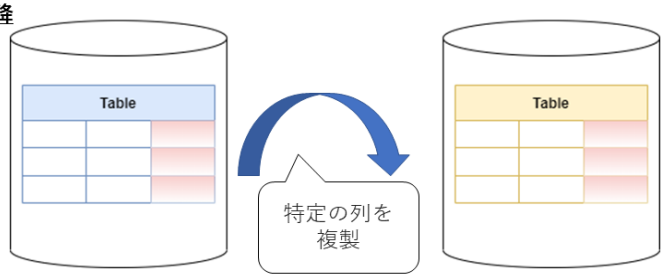
# 列指定 機能概要

- 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



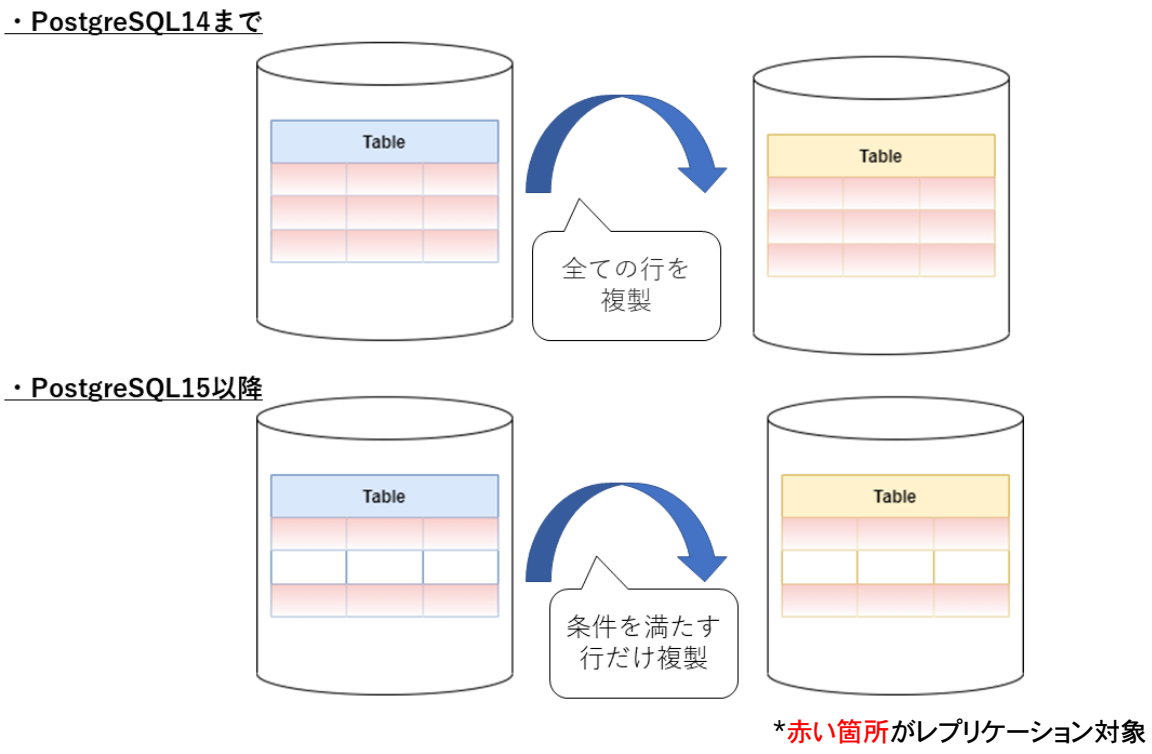

・PostgreSQL15以降




*赤い箇所がレプリケーション対象

# 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




*赤い箇所がレプリケーション対象

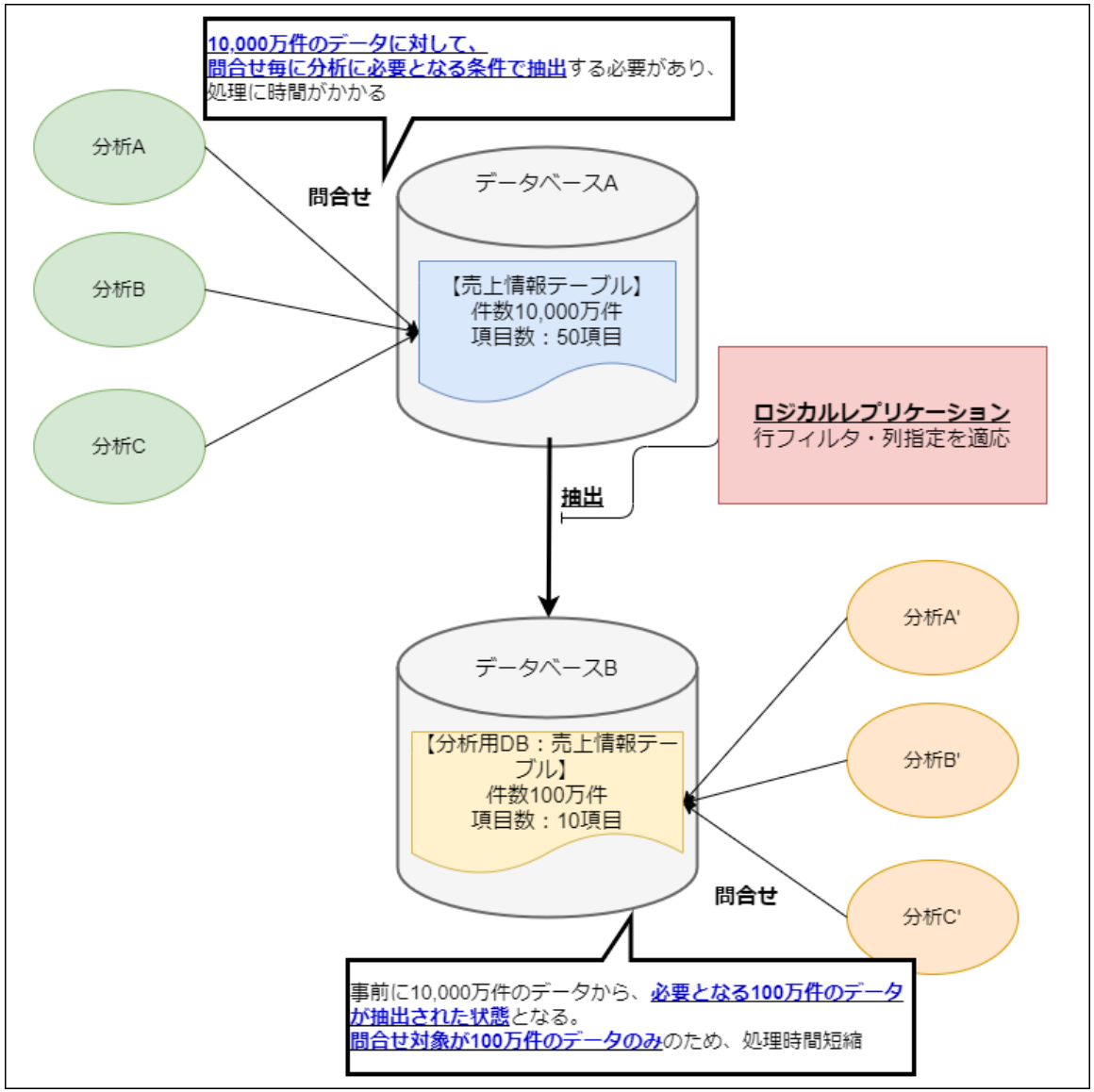
# 列指定/行フィルタの応用事例

- 冗長性や可用性のためにデータベースクラスタ全体のレプリカを作成するストリーミングレプリケーションと比較し、ロジカルレプリケーションは、必要なデータのみを他の場所で使用したりする場合に適している。

| ストリーミングレプリケーション | ロジカルレプリケーション |
| --- | --- |
| データベース全体に対する物理的なレプリケーション<br>冗長化や高可用な構成に使用 | 意味的に等価となるようにWALをデコードした論理的な情報を用いたレプリケーション<br>特定のデータを複製等で使用 |

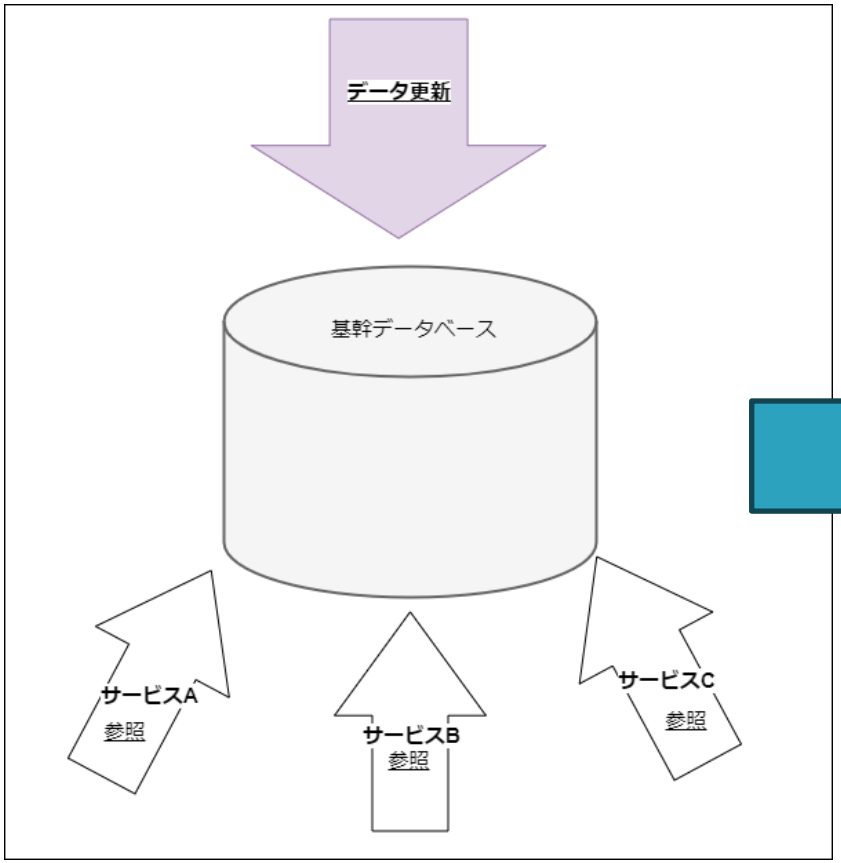
- ▼ロジカルレプリケーションの応用事例
  - 分析用DBのデータ抽出
  - データベースの統合/データベースからのデータ移行
  - テスト環境へのデータコピー
- 以降のスライドで、いくつかの応用事例を示す。

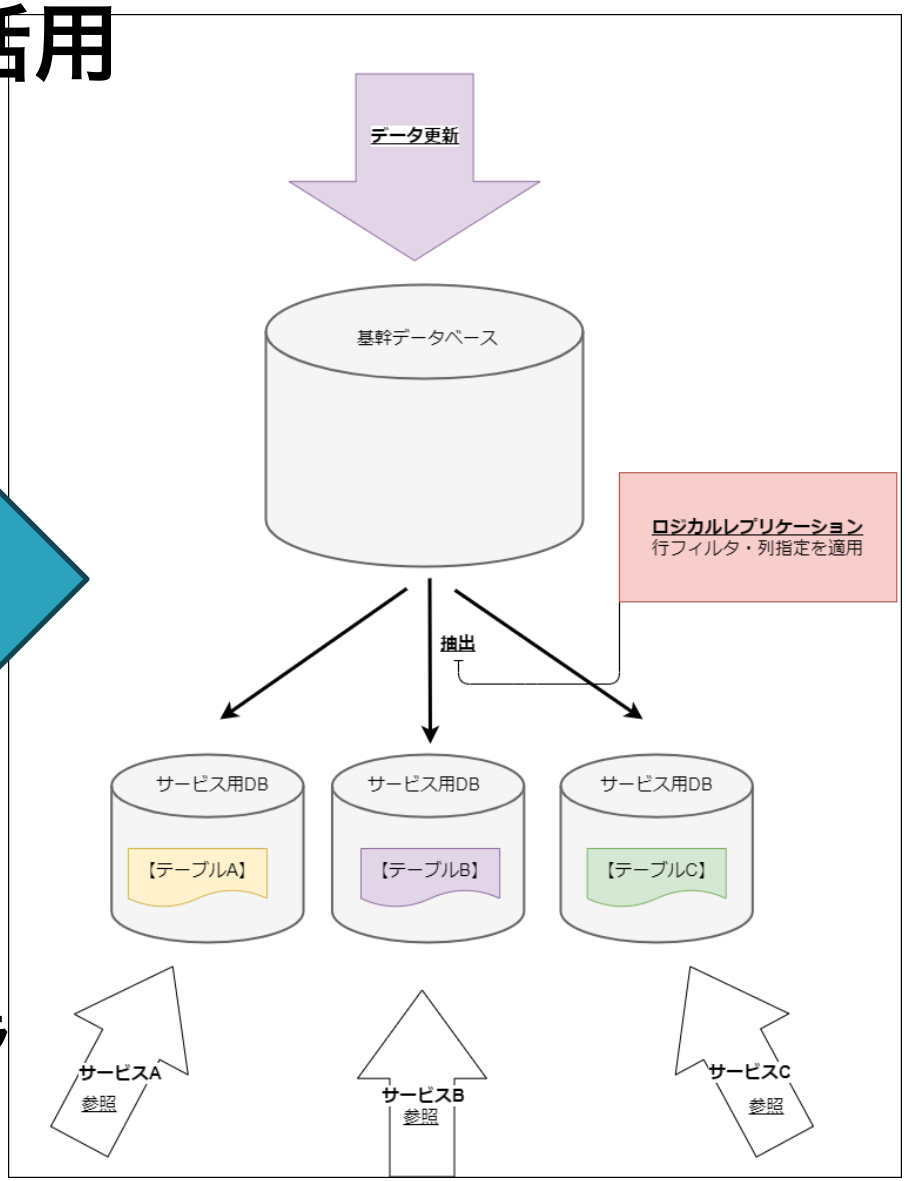
# 分析用DBでのデータ抽出の改善




- 問合せ毎に条件を指定した抽出処理が不要になるため、処理時間短縮。
- 対象範囲を限定することで、データ抽出のコストが小さくなり、リアルタイム更新の分析にも向いている。
- PostgreSQL16では、並列レプリケーション機能が追加され、リアルタイム性が強化（★想定・要検証）

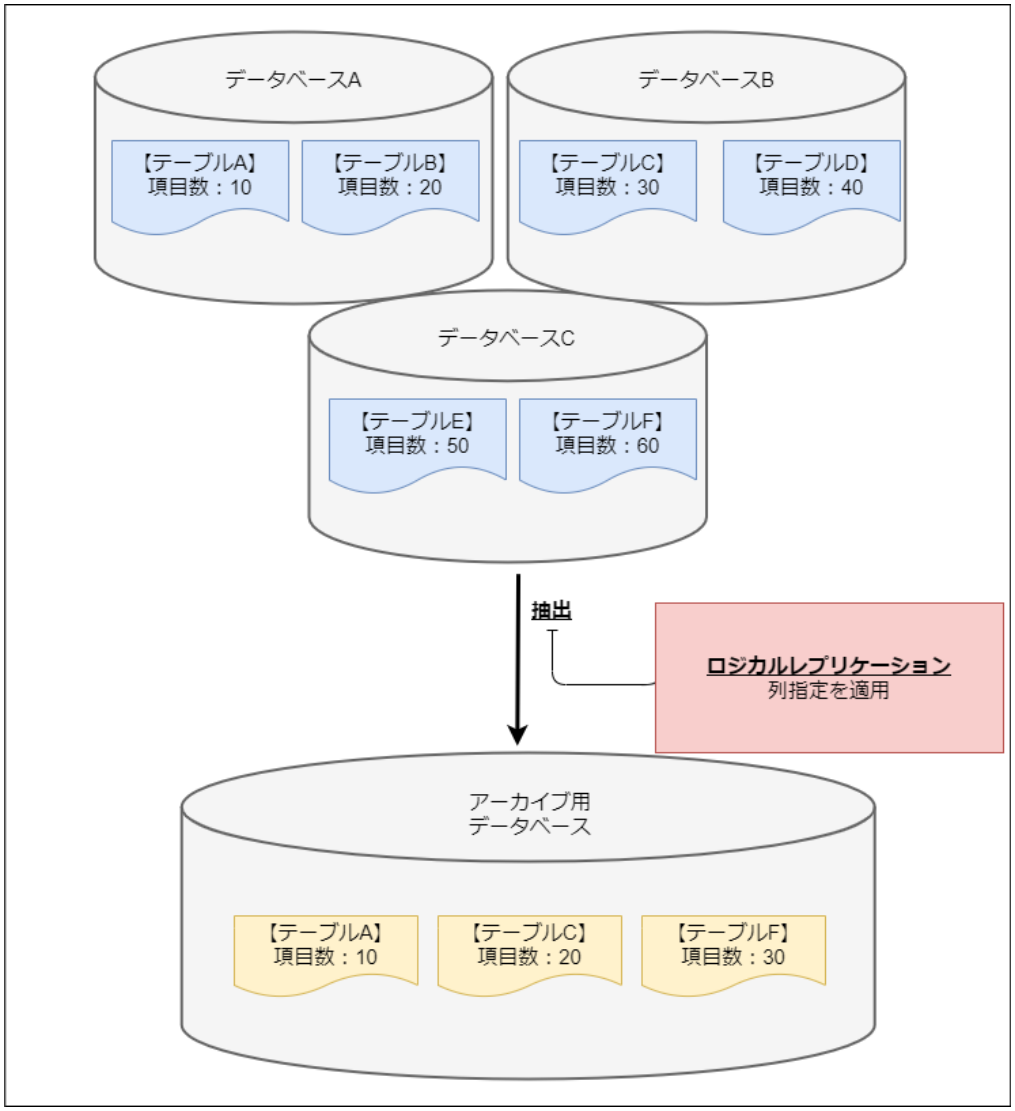
# データ移行時の抽出に活用




- 各サービスで使用するテーブル・カラムのみサービス用DBに抽出することで、サービスからのアクセスを分散

# データベース統合時の抽出に活用




- 複数のデータベースから、アーカイブとして残したいデータを絞り込んだ上で、別のDBに抽出する
- 必要なデータのみをレプリケーションできるため、データ容量を削減。

# 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。
- PostgreSQL16では、並列処理や主キーを使用しないレプリケーションのパフォーマンス向上されており、リアルタイムにデータを同期して利用するようなケースでの利用も期待できる。

# マルチマスタ検証

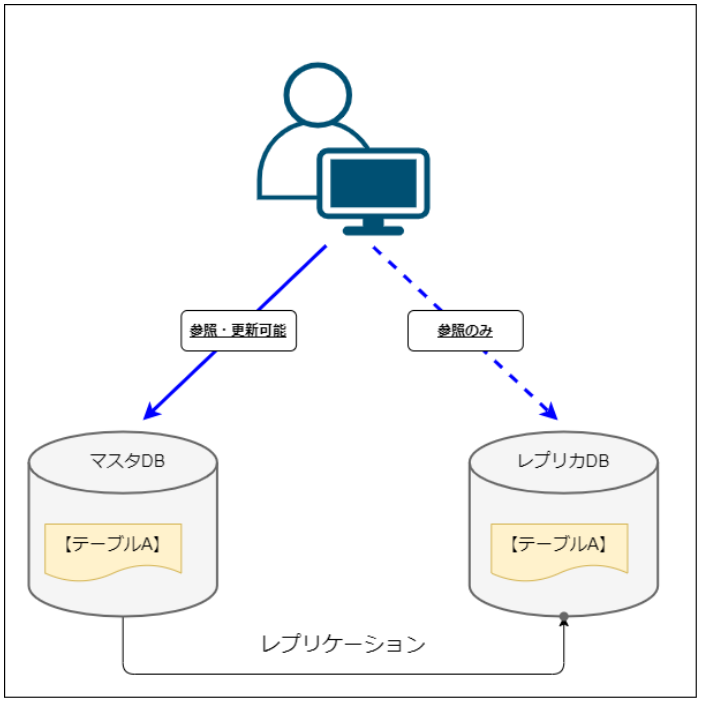
# 検証概要

- 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

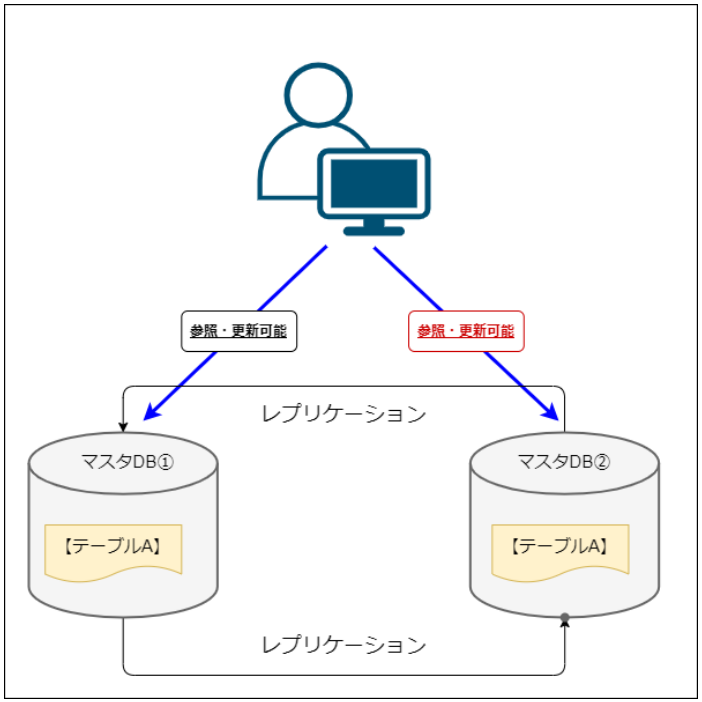
# マルチマスタとは

- マルチマスタとは、更新可能なデータベースが複数あるサーバー構成のこと。
- ⇔更新可能なデータベースは1つだけで、他は読み取り専用な構成はシングルマスタ

- シングルマスタ




- マルチマスタ

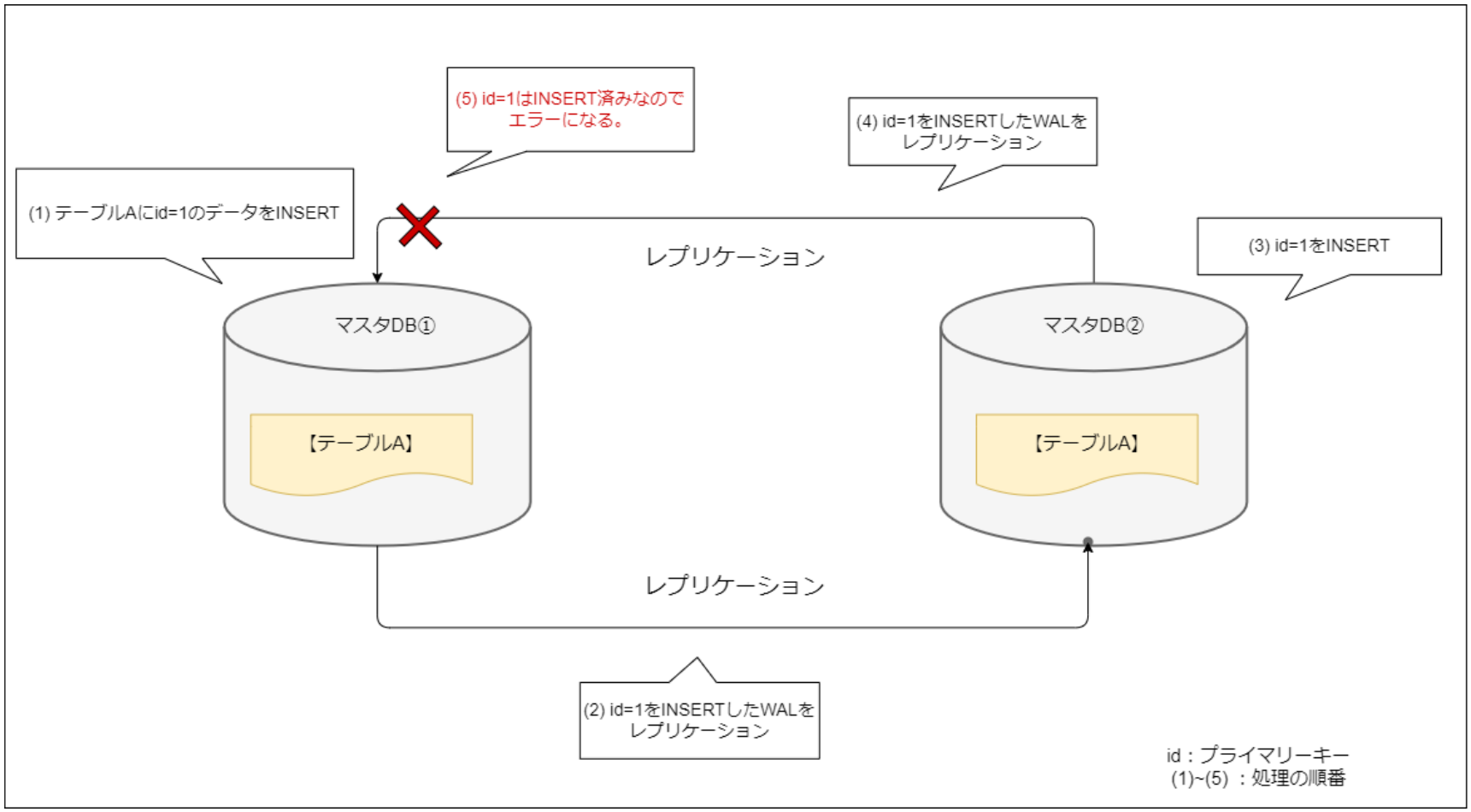


# PostgreSQL 15までの問題点

- 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
  - 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。

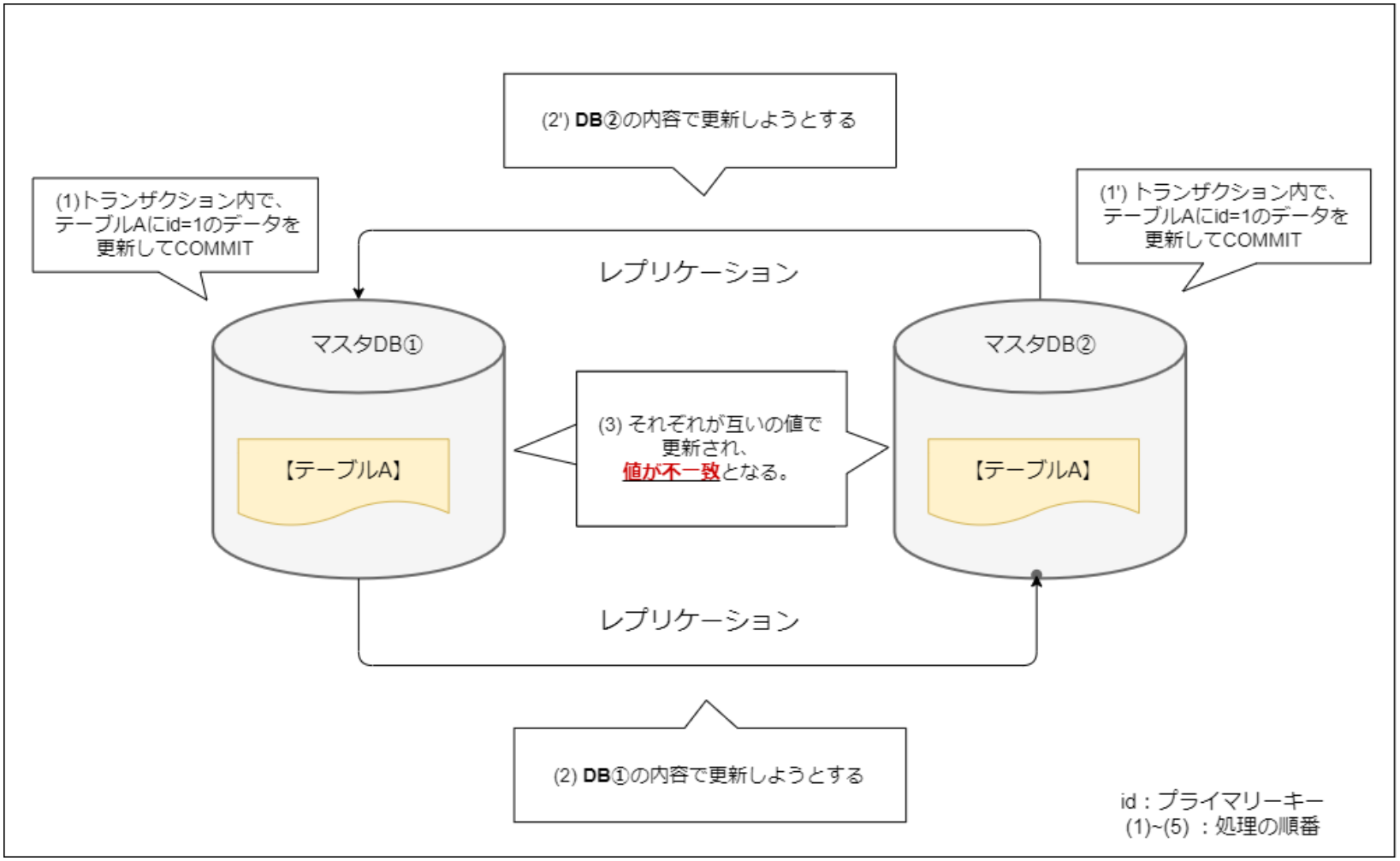
# WALの循環

- INSERT WALが永遠に2ノード間でループ



# 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。

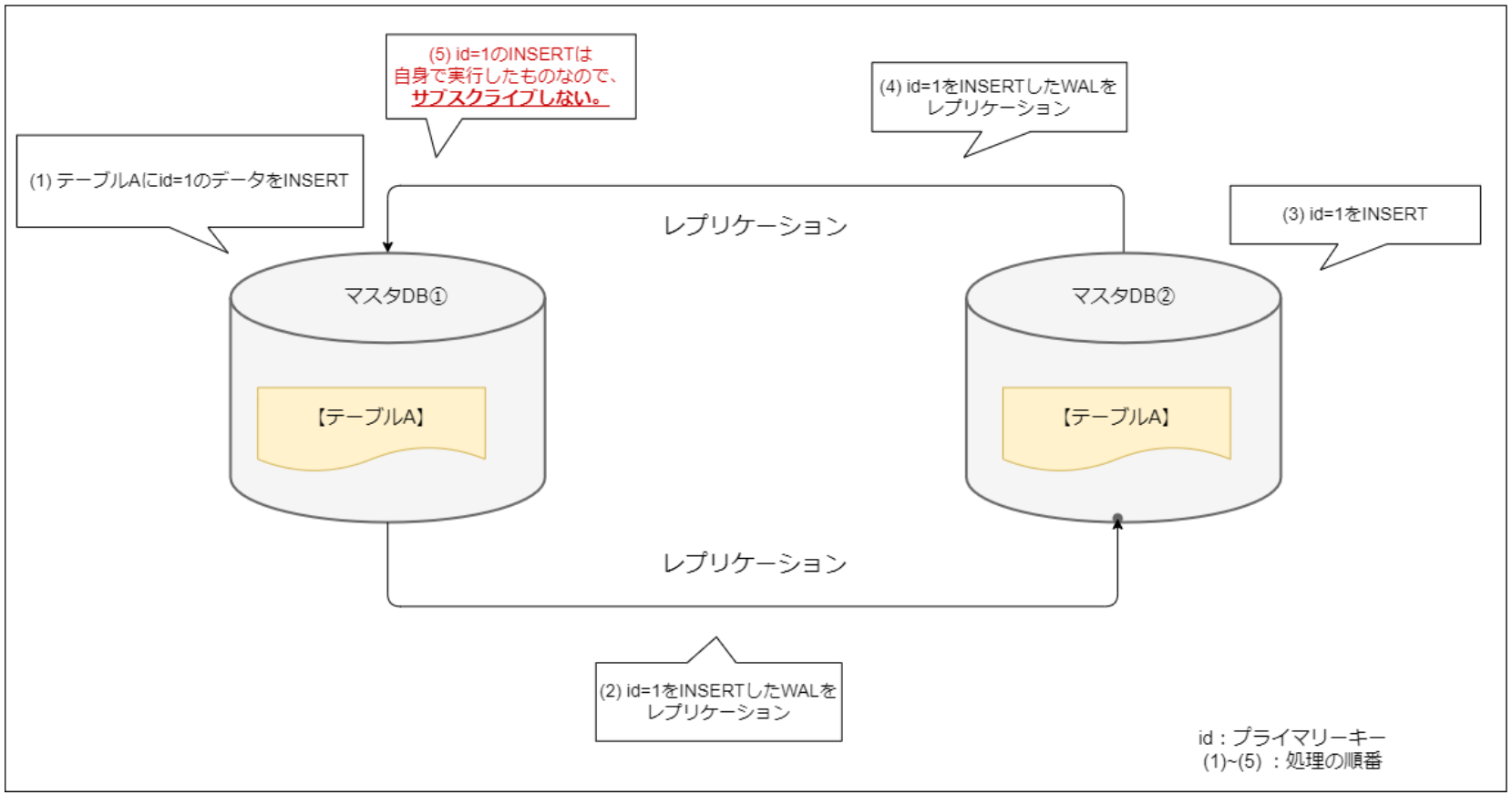


# PostgreSQL16での改善点

- CREATE SUBSCRIPTIONコマンドに、origin オプションが指定可能になり、自身がパブリッシュしたレコードをサブスクライブしない指定が可能となった
  - 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 の指定により、レプリケーションの循環が抑止される。

# PostgreSQL16での改善点

- 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。

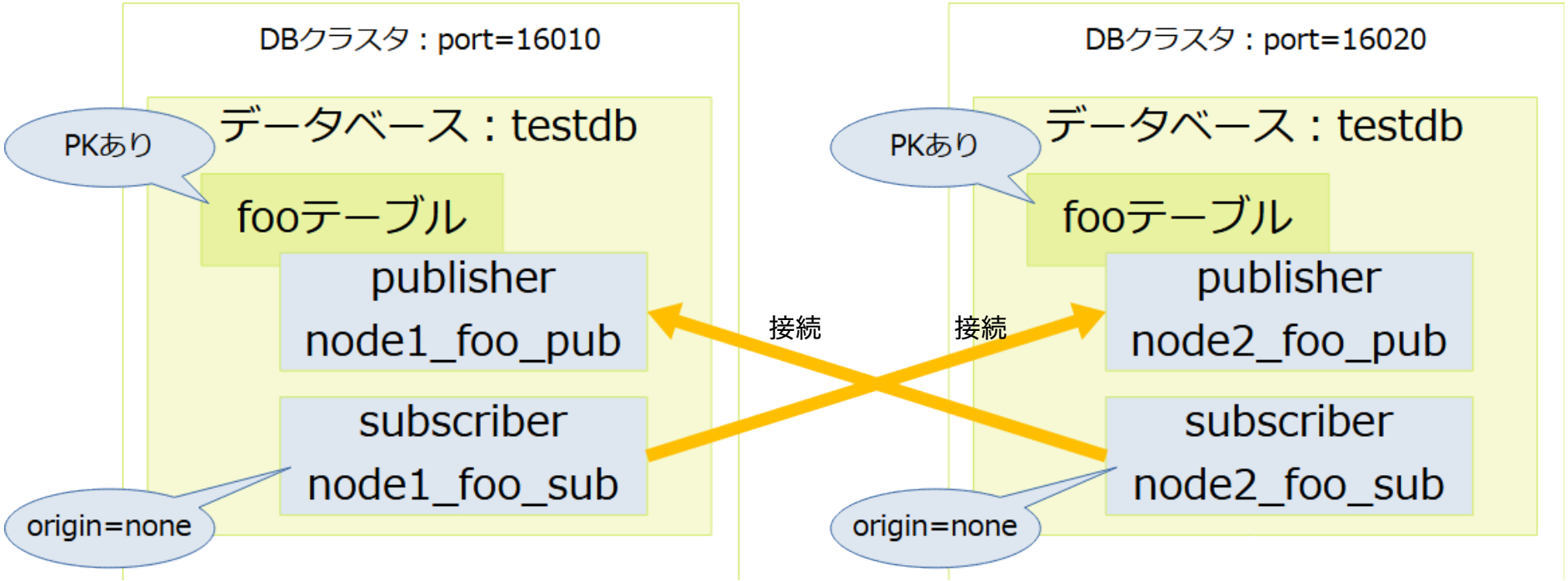


# PostgreSQL 16の機能詳細

- 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

- 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);
```

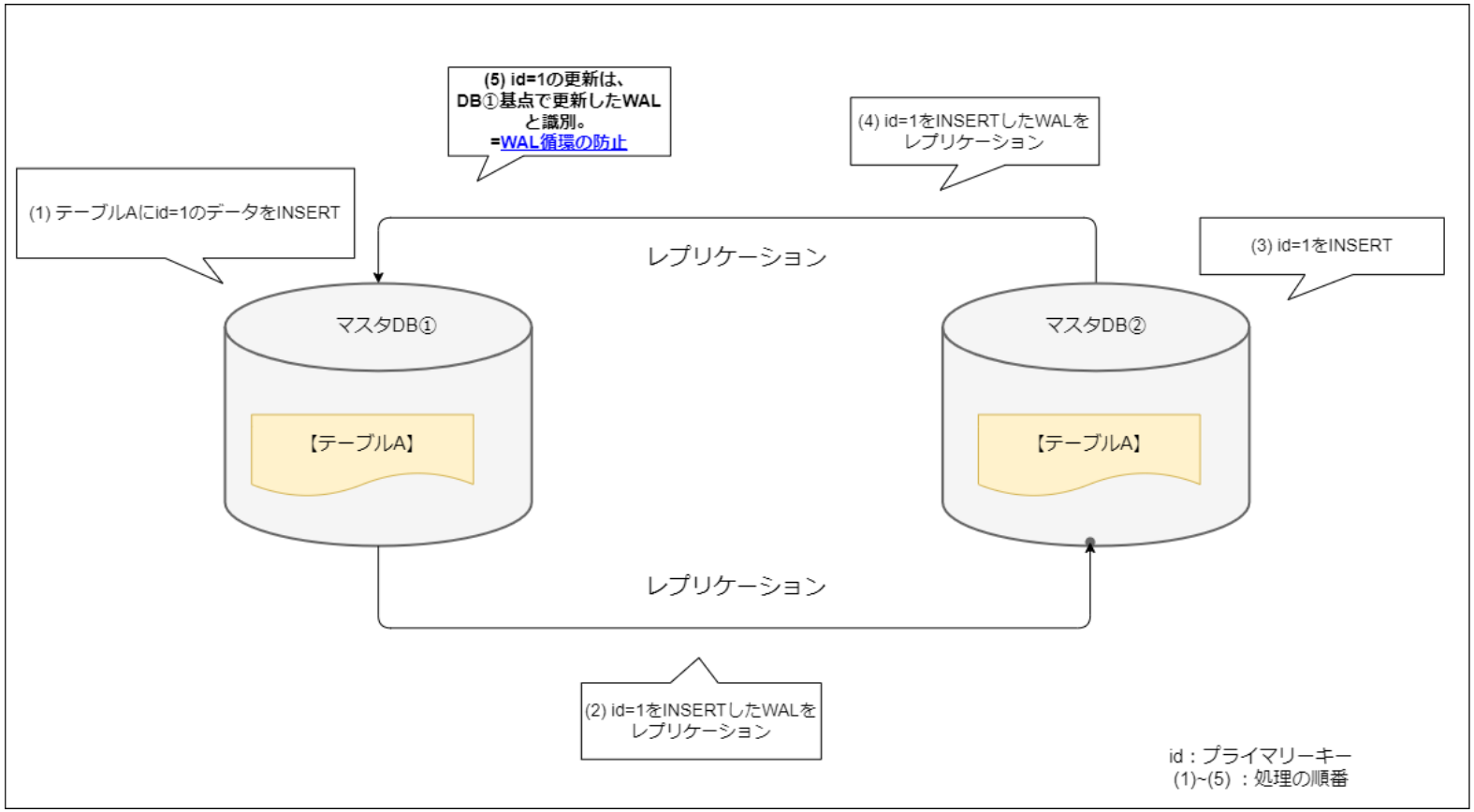
- **NODE2**

```
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);
```

# “WALの循環防止”を確認

- 各ノードをシーケンシャルに更新(同一PKで競合が発生しない状態)では、相互に更新が伝播することを確認した。
  - 以下DMLで確認
    - INSERT/UPDATE/DELETE/TRUNCATE
- WALの循環防止の効果があった。

# “WALの循環防止”を確認



# “更新の衝突”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、かつ運用介在が必要になる。

| 検証パターン | ノードへの操作 | | 実行結果(実際のテーブルの値) | | |
|---|---|---|---|---|---|
| | Node1 | Node2 | Node1 | Node2 | 系全体 |
| 同一PKへINSERT | INSERT (‘ABC’) | INSERT (‘abc’) | Node2への挿入内容 (‘abc’) | Node1への挿入内容 (‘ABC’) | 値は不一致<br>レプリケーション停止<br>(運用介在が必要) |
| 同一PKへUPDATE | UPDATE (‘ABC’) | UPDATE (‘abc’) | Node2への更新内容 (‘abc’) | Node1への更新内容 (‘ABC’) | 値は不一致<br>(対向側の値で更新) |
| 同一PKへUPDATE/DELETE | UPDATE (‘ABC’) | DELETE | Node2へのDELETE結果 (UPDATEしたレコードも削除) | Node2へのDELETE結果 | 値は一致 |
| 全行DELETE/INSERT | INSERT (‘ABC’) | 全DELETE | 全DELETE後、Node1へのINSERT(‘ABC’)で更新 | 全DELETE後、Node1へのINSERT(‘ABC’)で更新 | 値は一致 |
| TRUNCATE/INSERT | INSERT (‘ABC’) | TRUNCATE | TRUNCATE結果 | TRUNCATE後、Node1のINSERT(‘ABC’)で更新 | 値は不一致 |

# 衝突の例：同一PKのINSERT

- 実行内容
  - NODE1, NODE2に対して同一PK/別の列値で挿入する。
- 理想とする結果
  - どちらかのNODEに対しては挿入を抑止
    - 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等
- 実際の動作
  - 両NODEへ挿入はできるが、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、運用介在して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 SUBSCTIPTION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 SUBSCTIPTION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

- 実行内容
  - NODE1, NODE2から同一PKを持つレコードに対してUPDATEを実行。
    - 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。
- 理想とする結果
  - 同一PKに対するUPDATEは抑止等
- 実際の結果
  - どちらのUPDATEも動作するが、NODE１とNODE2の値は同一にならない

# 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)
=#
```

# 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。

# 結論

- 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、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

- PostgreSQL 16のSQL本体機能のみでは、実用的なマルチマスタ構成は困難
  - 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