OSS-DB

OSS-DB Exam Silver
技術解説セミナー
2025/9/6 開催

主題	S3 開発/SQL
副題	S3.1 SQLコマンド - レプリケーション - JSON

本日の講師

NTT DATA
株式会社ＮＴＴデータ
清野　裕貴

LPI-JAPAN

自己紹介

■清野 裕貴（セイノ ユウキ）

- 株式会社ＮＴＴデータ / OSSソリューション統括部 所属
- OSS関連のサポートやソフトウェア開発に従事
  - テクニカルサポート
  - KVM(Linuxの仮想化技術)を管理するためのソフトウェアの開発
    - Prossione Virtulization
- OSS教科書 OSS-DB Silver Ver.3.0対応 の執筆
  - Ver3.0の試験範囲に完全対応、練習問題も豊富。

プロモーション（１／２）

- Prossione Virtualization（https://oss.nttdata.com/pv/）
  - KVM(Linuxの仮想化技術)を利用した仮想化基盤を管理・運用するためのサービス

プロモーション（２／２）

■最近のニュースリリース

- KVM仮想化基盤を管理する「Prossione Virtualization® 1.0」のサービス提供を開始（2025/7/31）
  - サービス概要や価格、提供機能についてのご案内
- NTTデータとサイバートラスト、「Prossione Virtualization®」の製品強化および長期サポート体制確立に向けた協業を開始（2025/7/31）
  - サイバートラスト社と連携し「Prossione Virtualizationサブスクリプション with AlmaLinux」を提供予定

■お問い合わせ

- prossione-support@hml.nttdata.co.jp

OSS-DBとは

オープンソースのデータベースソフトウェア「PostgreSQL」を扱うことができる技術力の認定です。様々な分野でPostgreSQLの利用拡大が進む中でOSS-DBの認定を持つことは、自分のキャリアのアピールにもつながります。

- ✓OSS-DB Goldは設計やコンサルティングができる技術力の証明

  PostgreSQLについての深い知識を持ち、データベースの設計や開発のほか、パフォーマンスチューニングやトラブルシューティングまで行えることが証明できます

- ✓OSS-DB Silverは導入や運用ができる技術力の証明

  PostgreSQLについての基本的な知識を持ち、データベースの運用管理が行えるエンジニアとしての証明ができます

- ✓最新はVer3.0（PostgreSQL 12～14 に対応）

セミナーのテーマについて

構築・運用の現場で使えるポイントを
この吹き出しで示します。

■本日のテーマ

- 試験範囲の中で重要度が一番高い「S3.1 SQLコマンド」の中でも、Ver3.0(2023/2/1～)から追加された試験範囲を例題中心に解説します。
  - **レプリケーション　※Ver3.0から追加**
  - **JSON ※Ver3.0から追加**

■過去の技術解説セミナーのテーマ

- 重要テーマのため要チェック。ただし、Ver2.0とVer3.0の試験範囲の差分に注意。
  - S2.4 バックアップ方法（2023/10/15）
    - https://oss-db.jp/event/20231015
  - S1.1 OSS-DBの一般的特徴、S1.2 リレーショナルデータベースに関する一般的な知識（2023/2/5）
    - https://oss-db.jp/event/20230205
  - S2.5 基本的な運用管理作業（2023/1/21）
    - https://oss-db.jp/event/20230121
  - S2.2 標準付属ツールの使い方（2022/12/4）
    - https://oss-db.jp/event/20221204
  - DBの環境設計・運用設計の基礎（2021/7/25）
    - https://oss-db.jp/event/20210725
  - DBの環境設計・運用設計の基礎（2021/7/25）
    - https://oss-db.jp/event/20210725
  - S2.3 設定ファイル（2021/6/29）
    - https://oss-db.jp/event/20210629
  - S2.2 標準付属ツールの使い方、S3.1 SQLコマンド（2021/4/10）
    - https://oss-db.jp/event/20210410
  - S3.3 トランザクションの概念、S3.1 SQLコマンド（2020/10/17）
    - https://oss-db.jp/event/20201017
  - S2.4 バックアップ方法（2020/7/19）
    - https://oss-db.jp/event/20200719

S3.1 SQLコマンド

■出題範囲

S3.1 SQL コマンド 【重要度： 13】

- **説明：**
  基本的なSQL文およびデータベースの構成要素に関する知識を問う
  レプリケーションの基本機能、種類、特徴などの理解を問う
- **主要な知識範囲：**
  SELECT 文
  INSERT 文
  UPDATE 文
  DELETE 文
  データ型
  テーブル定義
  インデックス
  ビュー
  マテリアライズドビュー
  トリガー
  シーケンス
  スキーマ
  テーブルスペース
  パーティション
  関数定義 / プロシージャ定義
  PL/pgSQL
  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

■Ver2.0からVer3.0への変更点

•S3 開発/SQL

S3.1 SQL コマンド
追加
- レプリケーションの基本機能、種類、特徴などの理解を問う
- 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
- GENERATED (AS IDENTITY)
- JSON
- JSONB
- CREATE PUBLICATION/SUBSCRIPTION
- JSON PATH
- CALL

削除
- ルール
- RULE

S3.2 組み込み関数
なし

S3.3 トランザクションの概念
なし

S3.1 SQLコマンド - 本日説明する範囲

■出題範囲

S3.1 SQL コマンド 【重要度： 13】

- **説明：**
  基本的なSQL文およびデータベースの構成要素に関する知識を問う
  レプリケーションの基本機能、種類、特徴などの理解を問う
- **主要な知識範囲：**
  SELECT 文
  INSERT 文
  UPDATE 文
  DELETE 文
  データ型
  テーブル定義
  インデックス
  ビュー
  マテリアライズドビュー
  トリガー
  シーケンス
  スキーマ
  テーブルスペース
  パーティション
  関数定義 / プロシージャ定義
  PL/pgSQL
  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

■Ver2.0からVer3.0への変更点

•S3 開発/SQL

S3.1 SQL コマンド

追加
- レプリケーションの基本機能、種類、特徴などの理解を問う
- 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
- GENERATED (AS IDENTITY)
- JSON
- JSONB
- CREATE PUBLICATION/SUBSCRIPTION
- JSON PATH
- CALL

削除
- ルール
- RULE

S3.2 組み込み関数

なし

S3.3 トランザクションの概念

なし

動作環境

- 実際のPostgreSQL 14における動作結果を記載しています。可能な方は手元に環境をご用意の上、講演後に実際に動作確認をしてみてください。
- **PostgreSQLのコマンドを試すための環境を用意**
  - dockerコマンドでPostgreSQLコンテナを起動

```
$ docker run --name pg14  -e POSTGRES_PASSWORD=secret  -e POSTGRES_USER=myuser  -e POSTGRES_DB=mydb  -p 5432:5432  -d postgres:14
```

  - コンテナ内のpsql(PostgreSQLクライアントツール)にアクセス

```
$ docker exec -it pg14 psql -U myuser -d mydb
```

  - クリーンアップ

```
$ docker rm -fv pg14
```

- **Linux仮想マシンを用いて環境を用意する場合**
  - 過去のセミナー資料をご参照ください。
    - OSS-DB Silver 技術解説無料セミナー(2023/10/15)のp.8～12
      - https://ferret-one.akamaized.net/files/6522720d82a9482783e899fa/20231015-silver.pdf?utime=1696756237

テーマ1

レプリケーション

例題

ロジカルレプリケーションに関する問題

- **問題**
  - **ロジカルレプリケーションの説明として、正しい記述をすべて答えなさい**
- **選択肢**

A) Windows と Linux の PostgreSQL 間でのレプリケーションが可能である
B) 異なるメジャーバージョン間のレプリケーションが可能である
C) サブスクライバー側のDBサーバは参照のみ可能であり、更新はできない
D) 1つのパブリッシャーに対して、複数のサブスクライバーが設定できる
E) 既にストリーミングレプリケーションを実行しているプライマリに対して、パブリッシャーは設定できない

例題

ロジカルレプリケーションに関する問題

- **問題**
  - **ロジカルレプリケーションの説明として、正しい記述をすべて答えなさい**
- **選択肢**

A) Windows と Linux の PostgreSQL 間でのレプリケーションが可能である
B) 異なるメジャーバージョン間のレプリケーションが可能である
C) サブスクライバー側のDBサーバは参照のみ可能であり、更新はできない
D) 1つのパブリッシャーに対して、複数のサブスクライバーが設定できる
E) 既にストリーミングレプリケーションを実行しているプライマリに対して、パブリッシャーは設定できない

- **ポイント**
  - ロジカルレプリケーションの基礎知識や構築・運用手順、制約など総合的な理解が問われる
    - プラットフォームやバージョンの環境差によるレプリケーション可否（A, B）
    - レプリケーションの種別による参照・更新の可否（C）
    - レプリケーションの接続関係、1対1, 1対n, n対n …？（D）
    - レプリケーションを実行しているDBサーバに関する制約（E）

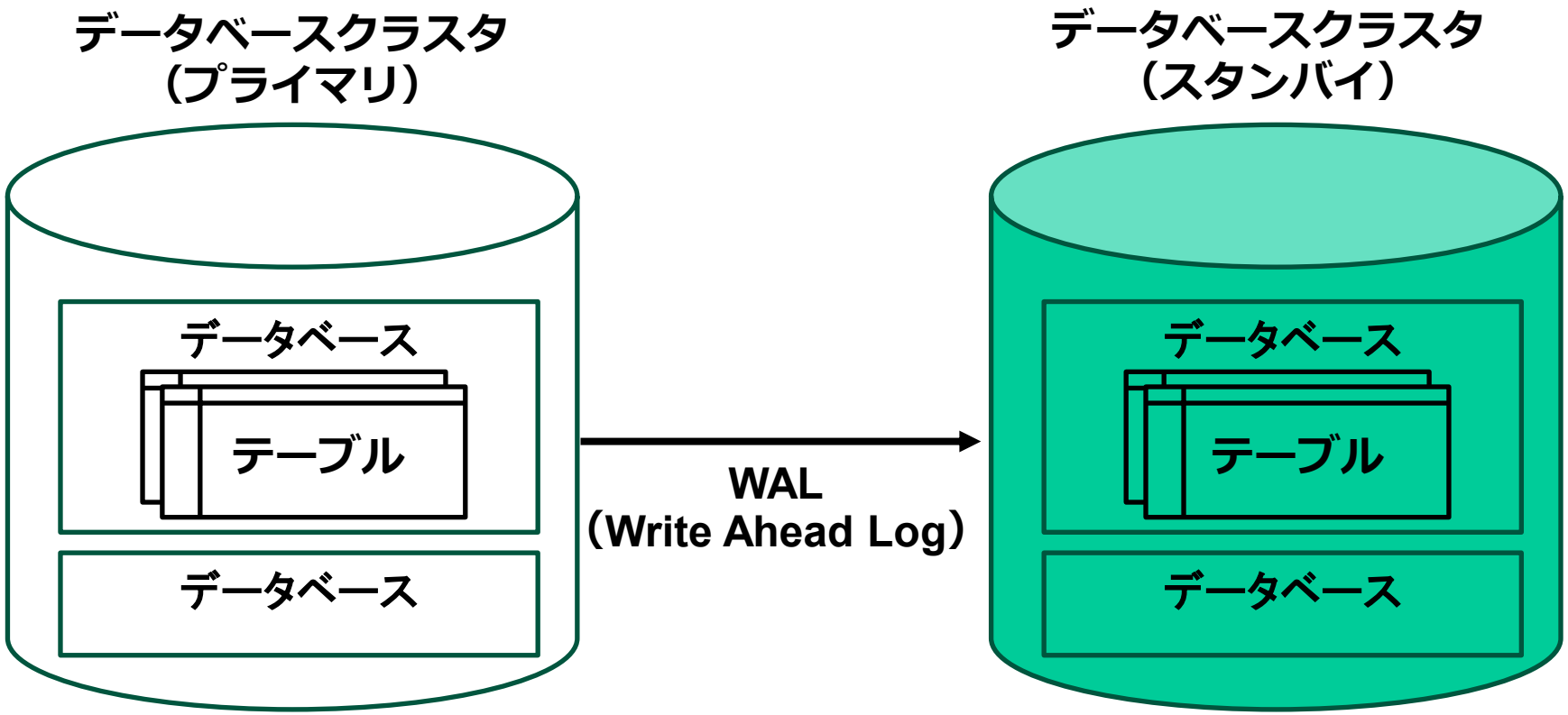
レプリケーション

レプリケーション

- データベースの内容を複製（replica）して、別のデータベースに転送・保管し、性能／可用性／信頼性を向上させる仕組み
- PostgreSQLに組み込まれているレプリケーション
  - 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）
  - **ロジカルレプリケーション（論理レプリケーション）**
    - 先ほどの例題はこちら。

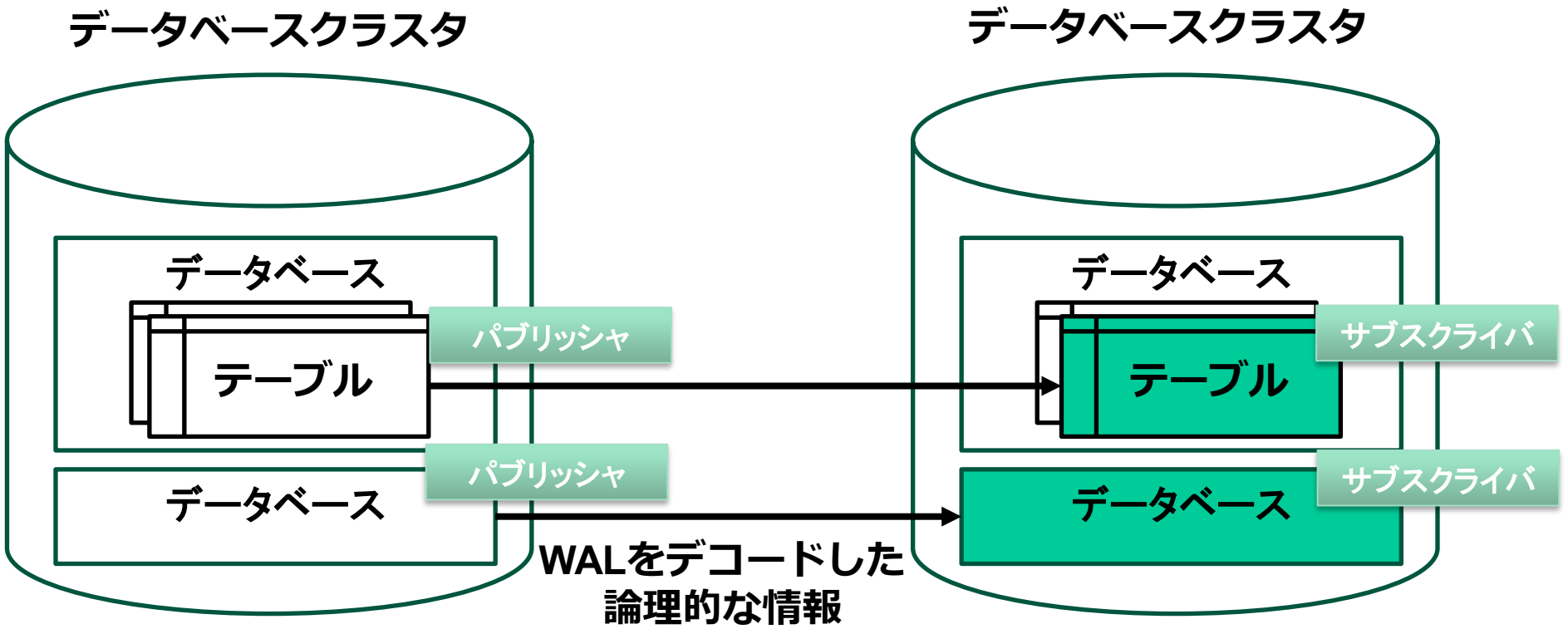
ストリーミングレプリケーション

- **データベースクラスタ全体を複製する**
- レプリケーション元の“物理的”な変更内容を複製する
- 一つのプライマリに対して、複数のスタンバイを接続することが可能



ロジカルレプリケーション

- PostgreSQL 10 から導入されたレプリケーション機能
- レプリケーション元の“論理的”な変更内容を複製する
- **柔軟なレプリケーションを実現することができる**
  - 特定のデータベース・テーブルのみ、一部の操作（INSERT/UPDATE/DELETE 等）のみ
- 一つのパブリッシャに対して、複数のサブスクライバを接続することが可能



レプリケーション方式の違い

これらの違いを把握しておくことは、レプリケーション方式選定において必要です

■2つのレプリケーション方式の違いを抑えておくことは重要

項目	ストリーミングレプリケーション （プライマリ⇔スタンバイ）	ロジカルレプリケーション （パブリケーション⇔サブスクリプション）
同期範囲	データベースクラスタ全体 （全データベース・全テーブル・全スキーマ）	データベース・テーブル単位
転送する内容	WALレコード単位のコピー	INSERT / UPDATE / DELETE などの論理的な変更（WALをデコードした論理的な情報）
互換性	同一メジャーバージョン・同一OS種別・同一CPUアーキテクチャでのみ可	・異バージョンで動作可（例: PG13 → PG16） ・異OSで動作可（例：Linux→Windows） ・異CPUアーキテクチャで動作可 (例:x86_64→aarch64)
用途	高可用性 (HA), バックアップの負荷分散	データ配信, 部分同期, バージョン移行, 双方向レプリケーション
データ参照	スタンバイは基本リードオンリー（書き込み不可）	サブスクライバ側は通常のDBとして書き込み可能（双方向構成も可能）
セットアップの方法	pg_basebackup(CLIコマンド) / primary_conninfo(GUCパラメータ)	PUBLICATION / SUBSCRIPTION の作成

ロジカルレプリケーション

ロジカルレプリケーションの用途

- **選択的なデータ複製**
  - 特定のテーブルだけを複製できることを利用し、必要最低限のテーブルだけ複製できるようにする
  - 利用例：監査ログテーブルや、マスタデータのみ別データベースに複製
- **PostgreSQLのバージョン間の複製（メジャーアップグレード）**
  - ストリーミングレプリケーションは同一メジャーバージョン間でしか使用できない
  - ロジカルレプリケーションなら異なるメジャーバージョン間で（例:PG13→PG16）も使用できる
  - 利用例：ダウンタイムを最小化したメジャーアップグレード
- **異種システム間の連携**
  - データをPostgreSQLから他のシステムにリアルタイムで転送する用途
  - 利用例：異なるプラットフォームのPostgreSQL、データウェアハウス（DWH）、分析基盤、検索エンジン（Elasticsearch）、ストリーミング処理（Kafka）への複製

ロジカルレプリケーションの注意点

ロジカルレプリケーションを使用する際の制約をしっかり理解しておきましょう

レプリケーションされないものがある

- データベーススキーマ
- DDL（TRUNCATEを除く）
- シーケンス

初期同期に負荷がかかる

- CREATE SUBSCRIPTION 時に**既存データのコピー**が走る
  - *CREATE SUBSCRIPTION sub1 CONNECTION 'conninfo' PUBLICATION pub1 WITH (copy_data = false)* のように *copy_data = false* にした場合は既存データのコピーをスキップする。
- 大規模テーブルだと時間がかかり、**ネットワーク負荷やロック競合**が発生する場合がある

レプリケーションスロットを使用する

- ロジカルレプリケーションのデータ送信の際に**レプリケーションスロットを使用**する
- サブスクリプション作成時に**パブリケーション側にレプリケーションスロット**が自動的に作成される
  - サブスクリプション作成時にパラメータ指定することで任意の名称のレプリケーションスロットを作成可
    - CREATE SUBSCRIPTION sub1 CONNECTION 'conninfo' PUBLICATION pub1 WITH (slot_name = 'sub1_slot');

ロジカルレプリケーションの構成方法

① 2つのPostgreSQLインスタンスを用意(今回は2つのPostgreSQLコンテナを用意)

```
# 1) 2つのコンテナが接続するためのネットワークを作成
$ docker network create pgnet

# 2) Publisher（送信元）
$ docker run --name pg14-pub --hostname pg14-pub --network pgnet ¥
  -e POSTGRES_PASSWORD=secret ¥
  -e POSTGRES_USER=myuser ¥
  -e POSTGRES_DB=mydb ¥
  -p 5433:5432 ¥
  -d postgres:14 ¥
  -c wal_level=logical

# 3) Subscriber（受信側）
$ docker run --name pg14-sub --hostname pg14-sub --network pgnet ¥
  -e POSTGRES_PASSWORD=secret ¥
  -e POSTGRES_USER=myuser ¥
  -e POSTGRES_DB=mydb ¥
  -p 5434:5432 ¥
  -d postgres:14
```

ロジカルレプリケーションの構成方法

② パブリケーションの設定（複製元）

```
# 1) pg14pubコンテナのpsql(PostgreSQLのクライアントツール)を起動
$ docker exec -it pg14-pub psql -U myuser -d mydb

# 2) レプリケーション用のテーブル作成とサンプルデータの挿入
mydb=# CREATE TABLE mytable(id bigint PRIMARY KEY, data text);
mydb=# INSERT INTO mytable VALUES (1,'hello'), (2,'world');

# 3) レプリケーション用ユーザー作成（SELECT権限付与）
mydb=# CREATE ROLE repl_user WITH REPLICATION LOGIN PASSWORD 'secret';
mydb=# GRANT SELECT ON mytable TO repl_user;

# 4) Publication作成（対象テーブルを指定）
mydb=# CREATE PUBLICATION mypub FOR TABLE mytable;
```

ロジカルレプリケーションの構成方法

③ サブスクリプションの設定（複製先）

```
# 1) pg14-subコンテナのpsql(PostgreSQLのクライアントツール)を起動
$ docker exec -it pg14-sub psql -U myuser -d mydb

# 2) Publisher と同じ定義のテーブルを事前作成(DDLはレプリケートされません)
mydb=# CREATE TABLE mytable(id bigint PRIMARY KEY, data text);

# 3) Subscription作成(host は Docker ネットワーク上のコンテナ名を指定)
mydb=# CREATE SUBSCRIPTION mysub
   CONNECTION 'host=pg14-pub port=5432 dbname=mydb user=repl_user password=secret'
   PUBLICATION mypub;
```

ロジカルレプリケーションの構成方法

④ レプリケーションの確認（初期同期）

```
# 1) pg14-subコンテナのpsql(PostgreSQLのクライアントツール)を起動
$ docker exec -it pg14-sub psql -U myuser -d mydb

# 2) テーブルに複製されていることを確認
mydb=# SELECT * FROM mytable;
 id | data
----+------
  1 | hello
  2 | world
(2 rows)
```

⑤ レプリケーションの確認（更新同期）

```
# 1) pg14-pubコンテナのpsql(PostgreSQLのクライアントツール)を起動
$ docker exec -it pg14-pub psql -U myuser -d mydb

# 2) 1行挿入
mydb=# INSERT INTO mytable VALUES (3,'replication');

# 3) pg14-subコンテナのpsql(PostgreSQLのクライアントツール)を起動
$ docker exec -it pg14-sub psql -U myuser -d mydb

# 4) テーブルに複製されていることを確認
mydb=# SELECT * FROM mytable;
 id | data
----+------
  1 | hello
  2 | world
  3 | replication
(3 rows)
```

ロジカルレプリケーションの構成方法

⑥ クリーンアップ

```
# 1) pg14-pub, pg14-sub コンテナの削除
$ docker rm -fv pg14-pub pg14-sub

# 2) pgnet ネットワークの削除
$ docker network rm pgnet
```

例題（再掲）

ロジカルレプリケーションに関する問題

- **次ページで答え合わせ**
- **問題**
  - **ロジカルレプリケーションの説明として、正しい記述をすべて答えなさい**
- **選択肢**

A) Windows と Linux の PostgreSQL 間でのレプリケーションが可能である
B) 異なるメジャーバージョン間のレプリケーションが可能である
C) サブスクライバー側のDBサーバは参照のみ可能であり、更新はできない
D) 1つのパブリッシャーに対して、複数のサブスクライバーが設定できる
E) 既にストリーミングレプリケーションを実行しているプライマリに対して、パブリッシャーは設定できない

例題（再掲）

■ロジカルレプリケーションに関する問題

- **正解は A, B, D**
- **問題**
  - **ロジカルレプリケーションの説明として、正しい記述をすべて答えなさい**
- **選択肢**

◯ A) Windows と Linux の PostgreSQL 間でのレプリケーションが可能である
  - プラットフォームが異なっていてもレプリケーション可能

◯ B) 異なるメジャーバージョン間のレプリケーションが可能である
  - メジャーバージョンが異なっていてもレプリケーション可能

C) サブスクライバー側のDBサーバは参照のみ可能であり、更新はできない
  - サブスクライバは更新処理が可能

◯ D) 1つのパブリッシャーに対して、複数のサブスクライバーが設定できる
  - 1つのパブリッシャに対して複数のサブスクライバが設定可能

E) 既にストリーミングレプリケーションを実行しているプライマリに対して、パブリッシャーは設定できない
  - ストリーミングレプリケーションとロジカルレプリケーションは競合しない（同時に設定できる）
  - プライマリには更新制約がないので、パブリッシャを作ることは問題ない（スタンバイには更新制約があるので、パブリッシャを作成することはできない）

例題

ロジカルレプリケーションに関する問題

- **次ページで答え合わせ**
- **問題**
  - **このサブスクリプションを 有効化するとき に行われる挙動として正しいものはどれか。**
    - CREATE SUBSCRIPTION sub1 CONNECTION 'conninfo' PUBLICATION pub1 WITH (copy_data = false);
- **選択肢**

A) プライマリからスナップショットを取得して全データを初期コピーする
B) 初期コピーは行わず、以後の変更だけがレプリケーションされる
C) pg_basebackup が自動で実行され、クラスタ全体をコピーする
D) ロジカルレプリケーションスロットは作成されない
E) slot_name を必ず指定しなければならないためエラーになる

例題

ロジカルレプリケーションに関する問題

- **正解は B**
- **問題**
  - **このサブスクリプションを 有効化するとき に行われる挙動として正しいものはどれか。**
    - CREATE SUBSCRIPTION sub1 CONNECTION 'conninfo' PUBLICATION pub1 WITH (copy_data = false);
- **選択肢**

A) プライマリからスナップショットを取得して全データを初期コピーする
◯B) 初期コピーは行わず、以後の変更だけがレプリケーションされる
C) pg_basebackup が自動で実行され、クラスタ全体をコピーする
D) ロジカルレプリケーションスロットは作成されない
E) slot_name を必ず指定しなければならないためエラーになる

テーマ2

JSON

例題

JSONに関する問題

- **問題**
  - **JSONを扱うデータ型の説明として、誤っているものを1つ選びなさい。**
- **選択肢**

A) JSONデータを格納するためのデータ型として、json型とjsonb型が用意されている。

B) json型は単に入力値のコピーを格納しているので、JSONオブジェクト内に同じキーと値が複数含まれることがある。

C) jsonb型は処理をするたびに再解析が必要とされないので処理が大幅に高速化される。

D) jsonb型はインデックスをサポートしていない。

E) JSONを格納するデータに効率的に問い合わせするために、jsonpath型が用意されている。

例題

JSONに関する問題

- **問題**
  - **JSONを扱うデータ型の説明として、誤っているものを1つ選びなさい。**
- **選択肢**

A) JSONデータを格納するためのデータ型として、json型とjsonb型が用意されている。

B) json型は単に入力値のコピーを格納しているので、JSONオブジェクト内に同じキーと値が複数含まれることがある。

C) jsonb型は処理をするたびに再解析が必要とされないので処理が大幅に高速化される。

D) jsonb型はインデックスをサポートしていない。

E) JSONを格納するデータに効率的に問い合わせするために、jsonpath型が用意されている。

- **ポイント**
  - JSONデータ型(json型, jsonb型)、jsonpath型への理解が求められる

JSON

JSONについて

- JSON (JavaScript Object Notation) は、**データを表現するためのテキストフォーマット**
- WebアプリケーションやAPI間でデータをやり取りする際に広く利用されている
- JavaScriptのオブジェクト表記を基にしているが、様々な言語で利用可能
- PostgreSQLでも年々機能が強化されている。(OSS-DB Ver3.0から試験範囲に追加)

```
{
  "user": {
    "id": 1,
    "name": "Taro Yamada",
    "email": "taro@example.com"
  },
  "article": {
    "id": 101,
    "title": "PostgreSQL OSS-DB",
    "published": true
  }
}
```

PostgreSQLにおけるJSON

json, jsonbデータ型の特性を覚えておきましょう。

■JSONデータ型

- **json型**
  - 入力テキストを**正確なコピー**で格納する。取得する度に**再解析が必要**となる。
- **jsonb型**
  - 入力テキストを**バイナリ形式**で格納する。取得する度に**再解析が不要**となる。
  - 格納時に**バイナリ形式に変換するためのオーバヘッド**がかかる。

■JSONデータを問い合わせするためのパス言語

- **jsonpathデータ型**
  - JSONPath式を保持するためのデータ型　例：$.user.name
    - JSONデータの特定の値を取得するためのクエリ言語
  - JSONBデータ型に対して柔軟な検索を行うために使う
  - 実際には jsonpath 型の値を jsonb_* 系関数等に渡して使う

```
{
  "user": {
    "id": 1,
    "name": "Taro Yamada",
    "email": "taro@example.com"
  },
  "article": {
    "id": 101,
    "title": "PostgreSQL OSS-DB",
    "published": true
  }
}
```

JSONデータの挿入

- json型もjsonb型もデータ挿入方法は同じ
- json型
  - テーブル作成（JSON型）

```
CREATE TABLE tbl_json (id INT, payload JSON);
```

  - JSONの挿入

```
INSERT INTO tbl_json (payload)
VALUES (
  '{
    "user": {
      "id": 1,
      "name": "Taro Yamada",
      "email": "taro@example.com"
    },
    "article": {
      "id": 101,
      "title": "PostgreSQL OSS-DB",
      "published": true
    }
  }'
);
```

=

- jsonb型
  - テーブル作成（JSONB型）

```
CREATE TABLE tbl_jsonb (id INT, payload JSONB);
```

  - JSONの挿入

```
INSERT INTO tbl_jsonb (payload)
VALUES (
  '{
    "user": {
      "id": 1,
      "name": "Taro Yamada",
      "email": "taro@example.com"
    },
    "article": {
      "id": 101,
      "title": "PostgreSQL OSS-DB",
      "published": true
    }
  }'
);
```

JSONデータの取得

JSONデータの取得

```
# JSON型
mydb=# SELECT payload->'user'->>'name' AS user_name FROM tbl_json;
  user_name
------------
 Taro Yamada
(1 row)

# JSONB型
mydb=# SELECT payload->'user'->>'name' AS user_name FROM tbl_jsonb;
  user_name
------------
 Taro Yamada
(1 row)
```

```
{
  "user": {
    "id": 1,
    "name": "Taro Yamada",
    "email": "taro@example.com"
  },
  "article": {
    "id": 101,
    "title": "PostgreSQL OSS-DB",
    "published": true
  }
}
```

jsonpathデータ型(JSONPath式)での取得

```
mydb=# SELECT jsonb_path_query(payload, '$.user.name') FROM tbl_jsonb;
 jsonb_path_query
------------------
 "Taro Yamada"
(1 row)
```

JSONの注意点

■json型は重複キーをそのまま保持する

重複キー（user）

```
INSERT INTO tbl_json (payload)
VALUES (
  '{
     "user": {
       "id": 1,
       "name": "Taro Yamada",
       "email": "taro@example.com"
     },
     "user": {
       "id": 3,
       "name": "Taro Yamada",
       "email": "taro@example.com"
     },
     "article": {
       "id": 101,
       "title": "PostgreSQL OSS-DB",
       "published": true
     }
   }'
);
```

userを[リスト]としているため、重複していない

```
INSERT INTO tbl_json (payload)
VALUES (
  '{
     "user": [
       {
         "id": 1,
         "name": "Taro Yamada",
         "email": taro@example.com
       },
       {
         "id": 3,
         "name": "Taro Yamada",
         "email": taro@example.com
       }
     ],
     "article": {
       "id": 101,
       "title": "PostgreSQL OSS-DB",
       "published": true
     }
   }'
);
```

JSONの注意点

■json型は重複キーをそのまま保持する

```
mydb=# SELECT payload FROM tbl_json;
               payload
--------------------------------------
 {                                   +
      "user": {                      +
        "id": 1,                     +
        "name": "Taro Yamada",       +
        "email": "taro@example.com"  +
      },                             +
      "user": {                      +
        "id": 3,                     +
        "name": "Taro Yamada",       +
        "email": "taro@example.com"  +
      },                             +
      "article": {                   +
        "id": 101,                   +
        "title": "PostgreSQL OSS-DB",+
        "published": true            +
      }                              +
    }
(1 row)

mydb=# SELECT payload FROM tbl_jsonb;
                                                                    payload
------------------------------------------------------------------------------------------------------------------------------------------------
 {"user": {"id": 3, "name": "Taro Yamada", "email": "taro@example.com"}, "article": {"id": 101, "title": "PostgreSQL OSS-DB", "published": true}}
(1 row)
```

JSONの注意点

- **jsonb型は GIN, btree, hash インデックスをサポート**
  - json型でインデックスを使う場合は型キャストが必要
    - *CREATE INDEX payload_idx ON tbl_json ((payload::text));*

例題（再掲）

JSONに関する問題

- **次ページで答え合わせ**
- **問題**
  - **JSONを扱うデータ型の説明として、誤っているものを1つ選びなさい。**
- **選択肢**

A) JSONデータを格納するためのデータ型として、json型とjsonb型が用意されている。

B) json型は単に入力値のコピーを格納しているので、JSONオブジェクト内に同じキーと値が複数含まれることがある。

C) jsonb型は処理をするたびに再解析が必要とされないので処理が大幅に高速化される。

D) jsonb型はインデックスをサポートしていない。

E) JSONを格納するデータに効率的に問い合わせするために、jsonpath型が用意されている。

例題（再掲）

JSONに関する問題

- **正解は D (Dが誤り)**
- **問題**
  - **JSONを扱うデータ型の説明として、誤っているものを1つ選びなさい。**
- **選択肢**

A) JSONデータを格納するためのデータ型として、json型とjsonb型が用意されている。

B) json型は単に入力値のコピーを格納しているので、JSONオブジェクト内に同じキーと値が複数含まれることがある。

C) jsonb型は処理をするたびに再解析が必要とされないので処理が大幅に高速化される。

◯ D) jsonb型はインデックスをサポートしていない。

  - jsonb型はGIN, btree, hashインデックスをサポートしている

E) JSONを格納するデータに効率的に問い合わせするために、jsonpath型が用意されている。

まとめ - 重点

レプリケーション

- **レプリケーション方式（ストリーミング、ロジカル）の違い**
  - 同期範囲（データベースクラスタ、データベース・テーブル）
  - それぞれの方式毎の用途（HA、バックアップ、バージョンアップ、移行）
- **ロジカルレプリケーションの注意点**
  - レプリケーションされないもの（データベーススキーマ、DDL、シーケンス）
  - 初期同期による負荷

JSON

- **PostgreSQLにおけるJSONの扱い方**
  - jsonデータ型(json型, jsonb型)の格納形式、特性
  - jsonpath型によるjsonデータの検索

おわりに

■例題解説をぜひご活用ください

- **https://oss-db.jp/measures/sample**