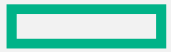


Hewlett Packard
Enterprise

# PostgreSQL 主要機能の進化

Noriyoshi Shinoda

November 13, 2020

# SPEAKER
## 篠田典良（しのだのりよし）

- 所属
  - 日本ヒューレット・パッカード株式会社
- 現在の業務
  - PostgreSQL をはじめ、Oracle Database, Microsoft SQL Server, Vertica 等 RDBMS 全般に関するシステムの設計、移行、チューニング、コンサルティング
  - Oracle ACE（2009 年 4 月～）
  - Oracle Database 関連書籍15冊の執筆
  - オープンソース製品に関する調査、検証
- 関連する URL
  - 「PostgreSQL 篠田の虎の巻」シリーズ
    - http://h30507.www3.hp.com/t5/user/viewprofilepage/user-id/838802
  - Oracle ACE ってどんな人？
    - http://www.oracle.com/technetwork/jp/database/articles/vivadeveloper/index-1838335-ja.html

# AGENDA

- パラレル・クエリー
- パーティショニング
- ロジカル・レプリケーション
- JIT

# PostgreSQL 概要

## PostgreSQL の歴史

- 1974年 Ingres プロトタイプ
  - HPE NonStop SQL, SAP Sybase ASE, Microsoft SQL Server の元になる
- 1989年 Postgres 1.0～
- 1997年 PostgreSQL 6.0～
- 2000年 PostgreSQL 7.0～
- 2005年 PostgreSQL 8.0～
- 2010年 PostgreSQL 9.0～
- 2017年10月 PostgreSQL 10
- 2018年10月 PostgreSQL 11
- 2019年10月 PostgreSQL 12
- 2020年9月 PostgreSQL 13（現状の最新）
- 2021年秋の予定 PostgreSQL 14

今日お話しする範囲

# パラレル・クエリー

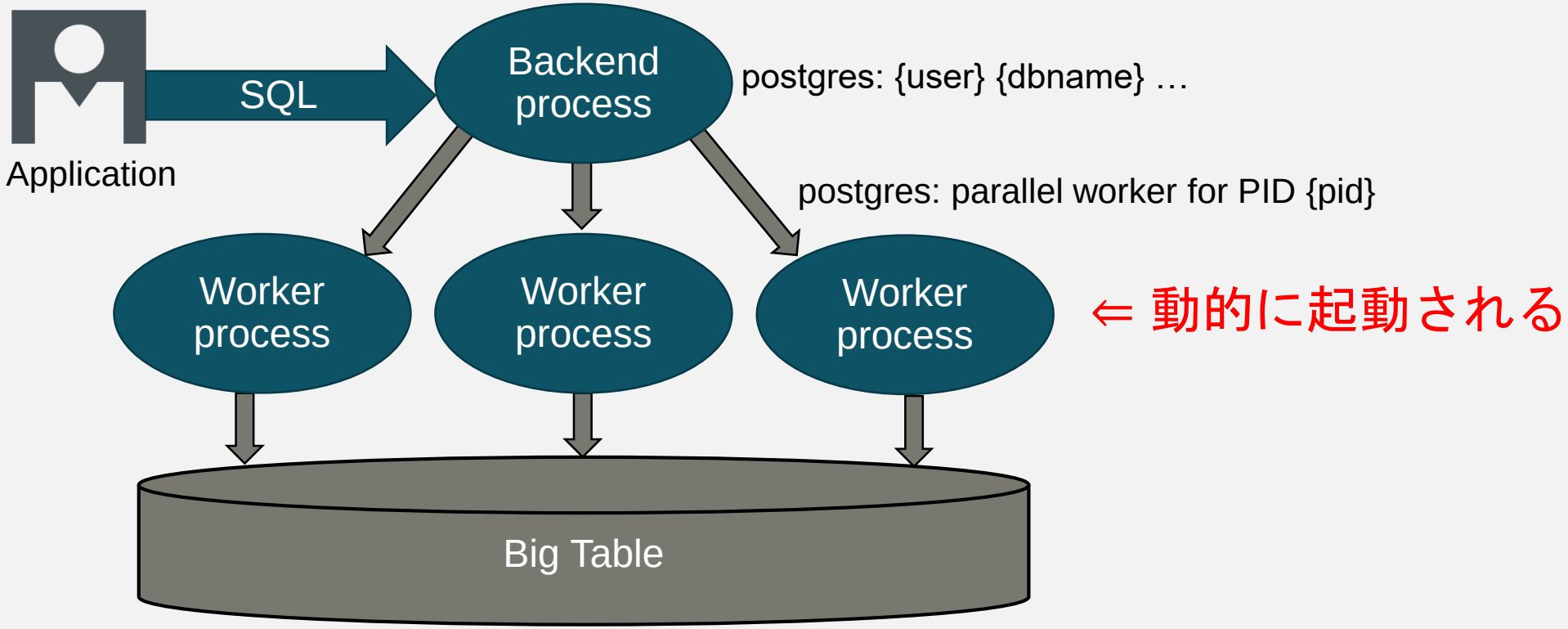
# パラレル・クエリー
## 概要

- 概要説明
  - 単一の SQL 文を複数のバックエンド・プロセスで並列に処理を行う機能
  - パラレル／シリアルの選択はコスト量により自動的に決定
  - PostgreSQL 9.6 ～

- アーキテクチャ
  - パラレル処理は Dynamic Background Worker (9.3～) を使う
  - プロセス間通信には Dynamic Shared Memory (9.4～) を使う
  - パラレル処理 API (9.5～) を使う

# パラレル・クエリー
## 概要

– SQL 文を複数のプロセスで並列に処理を行う



# パラレル・クエリー
## 実行計画

- 実行計画例

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=11614.55..11614.56 rows=1 width=8) (actual time=1106.746..1106…)
   ->  Gather  (cost=11614.33..11614.54 rows=2 width=8) (actual time=1105.972..1106.766 …)
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial Aggregate  (cost=10614.33..10614.34 rows=1 width=8) (actual time=1087.…)
               ->  Parallel Seq Scan on data1  (cost=0.00..9572.67 rows=416667 width=0)
                     (actual time=0.018..591.216 rows=333333 loops=3)
 Planning Time: 0.030 ms
 Execution Time: 1106.803 ms
(8 rows)
```

# パラレル・クエリー
## 実行計画

- 実行計画演算子

| オペレータ | 説明 | 備考 |
|---|---|---|
| Parallel Seq Scan | テーブル全件検索 | |
| Gather / Gather Merge | 集約／マージ処理 | 並列処理の起点 |
| Parallel Index Only Scan / Parallel Index Scan | インデックス検索 | |
| Finalize Aggregate / Partial Aggregate | 最終集約、部分集約 | |
| Partial GroupAggregate | グループ集計 | |
| Parallel Hash Join / Parallel Hash | ハッシュ結合 | |
| Parallel Append | アペンド処理 | |

- ワーカー・プロセス数
  - Workers Planned = 計画されたワーカー数
  - Workers Launched = 実行されたワーカー数

# パラレル・クエリー
## 実行計画

- 並列度の計算
  - テーブルやインデックスのサイズに依存（3倍を超える度に並列度を上げる）
    - min_parallel_table_scan_size (8MB)
    - min_parallel_index_scan_size (512kB)
  - テーブルの属性（parallel_workers）
    - テーブルのサイズより優先される
- 並列度の最大値
  - パラメーター　max_parallel_workers_per_gather (2)
  - パラメーター　max_parallel_workers (8)
  - パラメーター　max_worker_processes (8) など
- 強制的にパラレル・クエリーを実行
  - force_parallel_mode (off)

# パラレル・クエリー
## 実行計画

- SQL 文に含まれる関数の PARALLEL 属性
  - UNSAFE: パラレルクエリーでは実行できない
  - RESTRICTED: パラレルクエリーで実行可能だが、リーダー・プロセス内に限られる
  - SAFE: パラレルクエリーで実行可能
- 代表的な Parallel Unsafe 関数
  - currval / nextval / setval
  - lo_open / lo_get / lo_...
- 代表的な Parallel Restricted 関数
  - random
  - setseed
- pg_proc カタログの proparallel 列（'s', 'r', 'u'）で確認

# パラレル・クエリー
## 実行計画

- Parallel Safe 関数の実行例

```
postgres=> EXPLAIN SELECT * FROM data1 WHERE c1=CAST('2' AS INTEGER);
                                QUERY PLAN
-------------------------------------------------------------------------
 Gather  (cost=1000.00..11614.43 rows=1 width=12)
   Workers Planned: 2
   ->  Parallel Seq Scan on data1  (cost=0.00..10614.33 rows=1 width=12)
         Filter: (c1 = '2'::numeric)
```

- Parallel Unsafe 関数の実行例

```
postgres=> EXPLAIN SELECT * FROM data1 WHERE c1=NEXTVAL('seq1');
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on data1  (cost=0.00..22906.00 rows=1 width=12)
   Filter: (c1 = (nextval('seq1'::regclass))::numeric)
```

# パラレル・クエリー
## 実行計画

- CREATE FUNCTION 文で作成した UDF のデフォルトは Parallel Unsafe
  - 強制的に Parallel Safe に設定することはできる
  - 正常に動作するかは自己責任(nextval 関数は自己チェックされている)

```
postgres=> CREATE OR REPLACE FUNCTION getseq() RETURNS INTEGER AS $$
postgres$>   BEGIN
postgres$>     RETURN nextval('seq01');
postgres$>   END;
postgres$> $$ LANGUAGE plpgsql PARALLEL SAFE;
CREATE FUNCTION
postgres=>
postgres=> SELECT * FROM data1 WHERE c1=getseq();
ERROR:  cannot execute nextval() during a parallel operation
CONTEXT:  PL/pgSQL function getseq() line 3 at RETURN
```

# パラレル・クエリー

## pg_hint_plan

- 実行計画のヒントを指定できるOSS（https://ja.osdn.net/projects/pghintplan/）
- パラレル・クエリの指定
  - 構文： Parallel(テーブル名 並列度 [優先度])
  - 優先度は soft (default) または hard
  - soft を指定すると max_parallel_workers_per_gather を更新
- AWS/RDS, Aurora でも使用可能
- 実行例

```
postgres=> /*+ Parallel(data1 4 hard) */ EXPLAIN ANALYZE SELECT COUNT(*) FROM data1;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------...
Finalize Aggregate  (cost=15706.10..15706.11 rows=1 width=8) (actual time=338.465..338.567…)
   ->  Gather  (cost=15705.68..15706.09 rows=4 width=8) (actual time=338.148..338.562 …)
         Workers Planned: 4
         Workers Launched: 4
```

# パラレル・クエリー
## 関連するパラメーター

– 関連するパラメーター（PostgreSQL 13）

| パラメーター名 | 説明 | デフォルト値　備考 |
|---|---|---|
| max_parallel_workers | パラレルワーカーの最大数 | 8 |
| max_parallel_maintenance_workers | ユーティリティの最大ワーカー数 | 2 |
| max_parallel_workers_per_gather | Gather ノード内のワーカー最大数 | 2 |
| min_parallel_table_scan_size | 最小テーブルサイズ | 8MB |
| min_parallel_index_scan_size | 最大インデックスサイズ | 512kB |
| enable_parallel_append | 並列 Append 実行可否 | on |
| enable_parallel_hash | 並列 Hash 実行可否 | on |
| force_parallel_mode | 強制パラレル・モード | off |
| parallel_setup_cost | 並列化初期コスト | 1000 |
| parallel_tuple_cost | 並列化タプルコスト | 0.1 |
| parallel_leader_participation | リーダー・プロセスがタプルを処理 | on |

# パラレル・クエリー

## max_parallel_maintenance_workers

- 有効な SQL 文
  - CREATE INDEX 文（B-Tree のみ）
  - VACUUM 文 （FULL 無 ／ PostgreSQL 13 以降）
  - 一時テーブルでは使用されない

- 並列度の計算（CREATE INDEX）
  - テーブル属性 parallel_workers と比較して小さいほうが最大値
  - 計算値「maintenance_work_mem / (ワーカー数 + 1)」が 32 MB 以上になるまで小さくなる

# パラレル・クエリー
## バージョン間の差異

| 構文／環境 | 9.6 | 10 | 11 | 12 | 備考 |
|---|---|---|---|---|---|
| 全件検索（Seq Scan）と集約（Aggregate） | ● | | | | |
| インデックス検索（Index Scan） | | ● | | | |
| 結合（Nest Loop / Merge Join） | | ● | | | |
| ビットマップ・スキャン（Bitmap Heap Scan） | | ● | | | |
| PREPARE / EXECUTE 文 | | ● | | | |
| サブクエリー（Sub Plan） | | ● | | | |
| COPY 文 | | ● | | | |
| 結合（Hash Join） | | | ● | | |
| UNION 文（Append） | | | ● | | |
| CREATE 文（TABLE AS SELECT / MATERIALIZED VIEW / INDEX） | | | ● | | |
| SELECT INTO 文 | | | ● | | |
| パラレル・メンテナンス・ワーカー | | | ● | | |

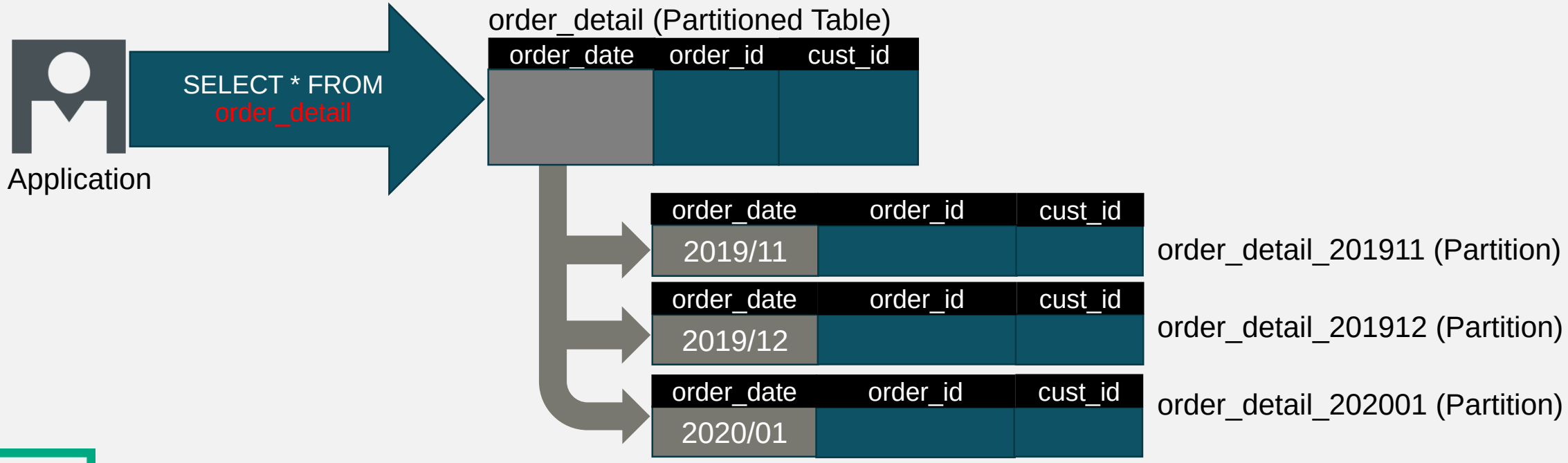
# パラレル・クエリー
## バージョン間の差異

| 構文／環境 | 9.6 | 10 | 11 | 12 | 13 | 14 | 備考 |
|---|---|---|---|---|---|---|---|
| SERIALIZABLE トランザクション分離レベル | | | | ● | | | |
| Parallel Vacuum | | | | | ● | | |
| COPY / VACUUM / INSERT 文に対する排他処理 | | | | | ● | | |
| COPY FROM 文 | | | | | | △ | Review |
| DISTINCT 句 | | | | | | △ | Review |

# パーティショニング

# パーティショニング
## 概要

- 概要説明
  - 大規模なテーブルを物理的に分割する機能
  - 通常は列値を使って自動的に分割先を決定
  - パーティションもテーブルとしてアクセス可能
  - PostgreSQL 10 ～



# パーティショニング
## 分割方法

- LIST Partition
  - 特定の値でパーティション化
  - 列値に一致するパーティションが選択される
- RANGE Partition
  - 値の範囲でパーティション化
  - 「下限値 <= 列値 < 上限値」によりパーティションが選択される
- HASH Partition
  - 値のハッシュ値でパーティション化
  - 分割数を指定する
  - PostgreSQL 11 ～

# パーティショニング
## パーティション・テーブル作成例

- LIST パーティションの作成例

```
postgres=> CREATE TABLE plist1 (c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY LIST (c1);
CREATE TABLE
postgres=> CREATE TABLE plist1_p1 PARTITION OF plist1 FOR VALUES IN (100);
CREATE TABLE
postgres=> CREATE TABLE plist1_p2 PARTITION OF plist1 FOR VALUES IN (200, 300);
CREATE TABLE
postgres=> ¥d plist1
Table "public.plist1"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | numeric               |           |          |
 c2     | character varying(10) |           |          |
Partition key: LIST (c1)
Number of partitions: 2 (Use ¥d+ to list them.)
```

# パーティショニング
## パーティション・テーブル作成例

- RANGE パーティションの作成例

```
postgres=> CREATE TABLE prange1 (c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY RANGE (c1);
CREATE TABLE
postgres=> CREATE TABLE prange1_p1 PARTITION OF prange1 FOR VALUES FROM (100) TO (200);
CREATE TABLE
```

- HASH パーティションの作成例

```
postgres=> CREATE TABLE phash1 (c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY HASH (c1);
CREATE TABLE
postgres=> CREATE TABLE phash1_p1 PARTITION OF phash1 FOR VALUES WITH (MODULUS 4, REMAINDER 0);
CREATE TABLE
```

# パーティショニング
## パーティションの追加／削除

- パーティションのアタッチ／デタッチ例

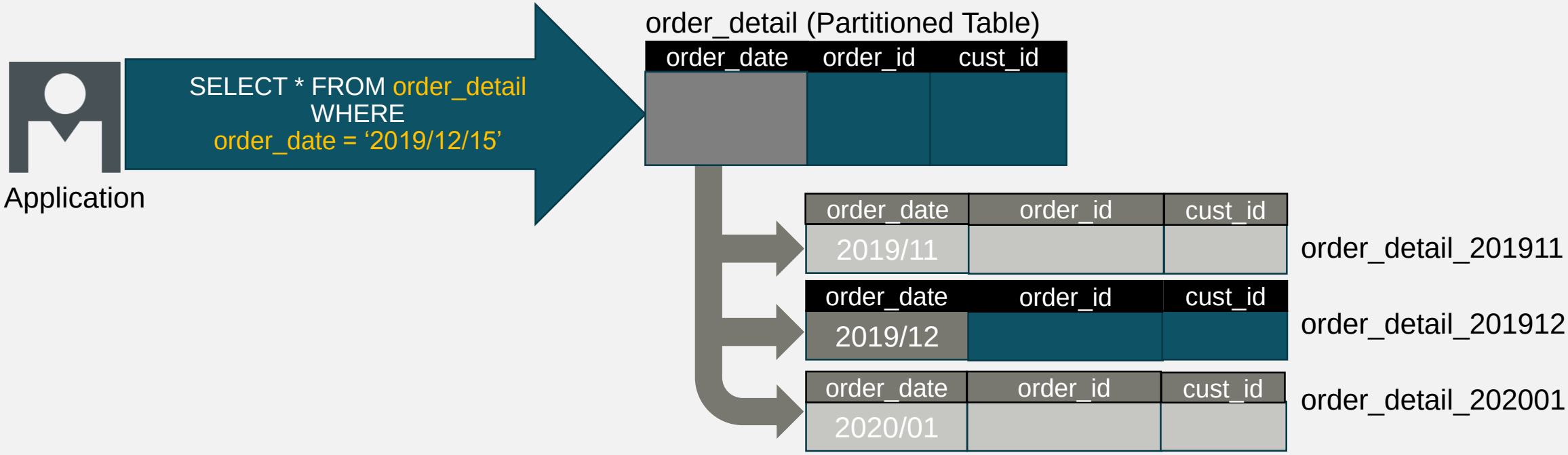
```
postgres=> ALTER TABLE plist1 ATTACH PARTITION plist1_p3 FOR VALUES IN (300);
ALTER TABLE
postgres=> ALTER TABLE plist1 DETACH PARTITION plist1_p3;
ALTER TABLE
```

- パーティションのアタッチ時の動作と制約
  - ATTACH PARTITION 句で指定するテーブルは、他のパーティションと同一構造（列名、データ型）が一致している必要がある。
  - ATTACH PARTITION 句実行時に格納済のデータは FOR VALUES 句に合致しているかチェックされる。

- パーティションの削除は DROP TABLE 文を実行する
  - 親となるパーティション・テーブルを削除すると、全パーティションが削除される

# パーティショニング
## パーティション・プルーニング

– 自動的に特定のパーティションのみにアクセスする機能
  – WHERE 句内にパーティションを特定できる情報がある場合など



# パーティショニング
## パーティション・プルーニング

```
postgres=> CREATE TABLE measurement (city_id int not null, logdate date not null,
    unitsales int) PARTITION BY RANGE (logdate);
CREATE TABLE
postgres=> CREATE TABLE measurement_y2019m02 PARTITION OF measurement
    FOR VALUES FROM ('2019-02-01') TO ('2019-03-01');
CREATE TABLE
postgres=> CREATE TABLE measurement_y2020m12 PARTITION OF measurement
    FOR VALUES FROM ('2020-12-01') TO ('2021-01-01');
CREATE TABLE
...
postgres=> EXPLAIN SELECT * FROM measurement WHERE logdate = '2020-12-02';
                                QUERY PLAN
-------------------------------------------------------------------------
 Seq Scan on measurement_y2020m12  (cost=0.00..33.12 rows=9 width=16)
   Filter: (logdate = '2020-12-02'::date)
(2 rows)
```

# パーティショニング
## 関連するパラメーター

– 関連するパラメーター（PostgreSQL 13）

| パラメーター名 | 説明 | デフォルト値　備考 |
|---|---|---|
| constraint_exclusion | テーブル制約に対する制約チェック | partition |
| enable_partition_pruning | パーティション・プルーニング実施 | on |
| enable_partitionwise_aggregate | パーティションワイズ集約の実施 | off |
| enable_partitionwise_join | パーティションワイズ結合の実施 | off |

# パーティショニング
## バージョン間の差異

| 構文／環境 | 10 | 11 | 12 | 13 | 備考 |
|---|---|---|---|---|---|
| 範囲によるパーティション（RANGE PARTITION） | ● | | | | |
| 値によるパーティション（LIST PARTITION） | ● | | | | |
| ハッシュ値によるパーティション（HASH PARTITION） | | ● | | | |
| その他の値が格納されるパーティション（DEFAULT PARTITION） | | ● | | | |
| パーティションを移動する UPDATE 文の実行 | | ● | | | |
| 親パーティション・テーブルに対するインデックス作成と伝播 | | ● | | | |
| 親パーティションに対する一意制約の作成 | | ● | | | |
| パーティション・ワイズ結合 | | ● | | ● | |
| パーティション・ワイズ集計 | | ● | | | |
| INSERT ON CONFLICT 文の対応 | | ● | | | |
| 計算値によるパーティション | | | ● | | |
| 外部キーとしてパーティション・テーブルの参照 | | | ● | | |

# パーティショニング
## バージョン間の差異

| 構文／環境 | 10 | 11 | 12 | 13 | 14 | 備考 |
|---|---|---|---|---|---|---|
| BEFORE INSERT トリガー対応 | | | | ● | | |
| TEXT ARRAY 列によるハッシュ・パーティション | | | | | ● | |
| 自動　LIST / HASH パーティション作成 | | | | | △ | Ready |
| ALTER TABLE DETACH PARTITION CONCURRENTRY | | | | | △ | Review |

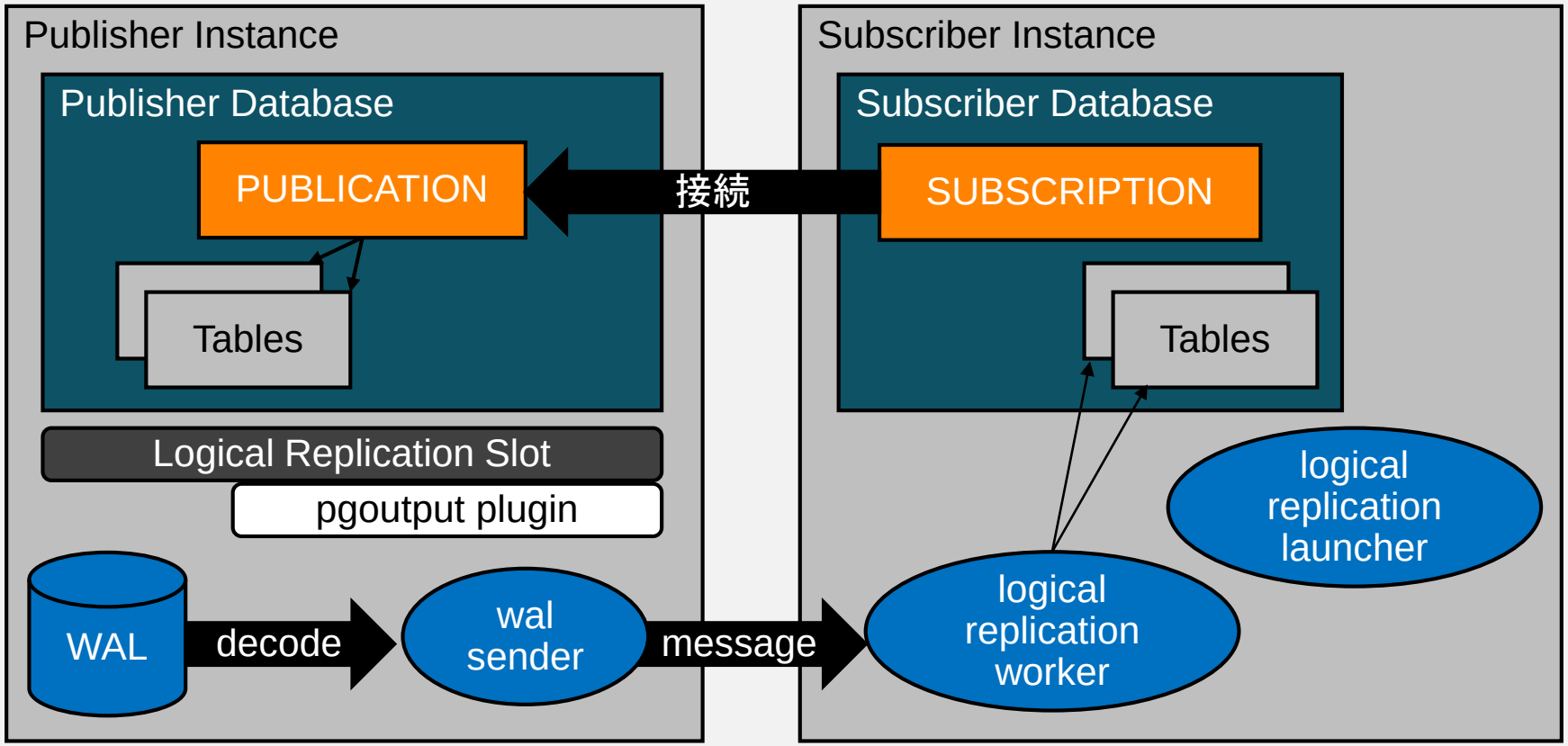
# ロジカル・レプリケーション

# ロジカル・レプリケーション
## 概要

- 概要説明
  - テーブル単位のレプリカ作成機能
  - レプリケーション先のテーブルも 更新可能
  - SQL 文の結果が同一であることを保証（=Logical）
  - PostgreSQL 10 ～
- ストリーミング・レプリケーションとの比較

| 比較 | Streaming Replication | Logical Replication | 備考 |
|---|---|---|---|
| レプリカ対象 | データベースクラスタ | テーブル群 | |
| レプリカの更新 | Read Only | Read Write | |
| ブロックの差分 | なし | あり（文字コード、列定義） | |
| 利用可能バージョン | PostgreSQL 9.0 ～ | PostgreSQL 10 ～ | |
| 類似の技術 | Oracle Data Guard<br>SQL Server AlwaysOn | Slony-I<br>MySQL RBR | |

# ロジカル・レプリケーション
## アーキテクチャ



# ロジカル・レプリケーション
## オブジェクト

- PUBLICATION オブジェクト
  - データ提供側データベースに作成
  - 一般ユーザー権限で作成可能（データベースに対する CREATE 権限が必要）
  - レプリケーション対象テーブルを決定
  - CREATE PUBLICATION 文で作成

- SUBSCRIPTION オブジェクト
  - データ受信側データベースに作成
  - SUPERUSER 権限が必要
  - 作成時に接続先インスタンスの接続情報と PUBLICATION 名を指定
  - CREATE SUBSCRIPTION 文で作成

# ロジカル・レプリケーション
## 作成例(データ提供元)

– レプリケーション対象テーブルの作成

```
pubdb=> CREATE TABLE data1 (c1 INT PRIMARY KEY, c2 VARCHAR(5));
CREATE TABLE
```

– レプリケーション対象テーブルの参照を接続ユーザーに許可

```
pubdb=> GRANT SELECT ON data1 TO repusr1;
GRANT
```

– PUBLICATION オブジェクトの作成

```
pubdb=> CREATE PUBLICATION pub1 FOR TABLE data1;
CREATE PUBLICATION
```

# ロジカル・レプリケーション
## 作成例（データ受信先）

– レプリケーション対象テーブルの作成

```
subdb=> CREATE TABLE data1 (c1 INT PRIMARY KEY, c2 VARCHAR(5));
CREATE TABLE
```

– SUBSCRIPTION オブジェクトの作成（SUPERUSER）

```
subdb=# CREATE SUBSCRIPTION sub1 CONNECTION
  'host=pubhost1 dbname=pubdb user=repusr1 password=******' PUBLICATION pub1;
CREATE SUBSCRIPTION
```

- デフォルトでは SUBSCRIPTION と同じ名前のロジカル・レプリケーション・スロットが自動的に作成される
- 接続ユーザの認証には pg_hba.conf ファイルの DATABASE = replication 項目は参照しない
- 初期データの移行が実行される

# ロジカル・レプリケーション
## 制約

- 同じである必要があること
  - スキーマ名
  - テーブル名
  - 列名
  - 列データ型（暗黙の型変換ができれば違っていても可）
  - タプルを一意に決定する列情報（Replica Identity = 通常は主キー）
- 違っていて良いこと
  - データベース名
  - 文字エンコーディング（UTF-8, 日本語 EUC 等）
  - 列の定義順序
  - インデックスの追加
  - 制約の追加
  - 列の追加（レプリケーション先）

# ロジカル・レプリケーション
## 制約

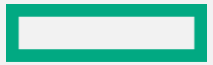
- テーブルの相互更新不可
  - 同じテーブルに対して双方向レプリケーション不可
  - WAL がループするため
- インスタンス内レプリケーション
  - レプリケーション・スロットと SUBSCRIPTION を別々に作成する必要がある
- 伝播不可な操作やオブジェクト
  - TRUNCATE 文以外の DDL
  - UNLOGGED TABLE / TEMPORARY TABLE
  - SEQUENCE / MATERIALIZED VIEW / INDEX
- シーケンスを列値に指定した場合（GENERATED ALWAYS 列等）
  - シーケンス操作ではなく、値が伝播する
  - SERIAL 列も同様の動作

# ロジカル・レプリケーション
## 関連するパラメーター

- 関連するパラメーター（PostgreSQL 13）

| パラメーター名 | 説明 | デフォルト値　備考 |
|---|---|---|
| wal_level | WAL出力レベル（要 logical 設定） | replica |
| max_logical_replication_workers | レプリケーション・ワーカーの最大数 | 4 |
| max_sync_workers_per_subscription | サブスクリプション単位の同期ワーカー数 | 2 |
| max_worker_processes | ワーカー・プロセスの最大値 | 8 |
| logical_decoding_work_mem | レプリケーション用作業メモリー量 | 64MB |
| max_replication_slots | レプリケーション・スロットの最大数 | 10 |
| max_wal_senders | wal senderプロセスの最大数 | 10 |

- AWS/RDS では、「rds.logical_replication = on」を指定することで、ロジカル・レプリケーションに必要なパラメータを変更



# ロジカル・レプリケーション
## 関連するビュー

- 関連するビュー（PostgreSQL 13）

| パラメーター名 | 説明 | 備考 |
|---|---|---|
| pg_publication | PUBLICATION オブジェクト情報 | |
| pg_replication_tables | PUBLICATION に含まれるテーブル情報 | |
| pg_subscription | SUBSCRIPTION オブジェクト情報 | |
| pg_subscription_rel | SUBSCRIPTION に含まれるテーブル情報 | |
| pg_replication_slots | レプリケーション・スロット情報 | |
| pg_replication_origin | レプリケーション起点情報 | |
| pg_stat_replication | レプリケーションの状態確認 | |
| pg_stat_replication_slots | レプリケーション・スロットの状態確認 | PostgreSQL 14 ～ |

# ロジカル・レプリケーション
## バージョン間の差異

| 構文／環境 | 10 | 11 | 12 | 13 | 14 | 備考 |
|---|---|---|---|---|---|---|
| PUBLISHER / SUBSCRIBER によるレプリケーション | ● | | | | | |
| テーブル単位の伝播 | ● | | | | | |
| 全テーブルの伝播 | ● | | | | | |
| 文字コード変換 | ● | | | | | |
| TRUNCATE 文の伝播 | | ● | | | | |
| ロジカル・レプリケーション・スロットのコピー | | | ● | | | |
| パーティション・テーブルの伝播 | | | | ● | | |
| 一時レプリケーション・スロット | | | | ● | | |
| デコード用メモリー設定パラメーター | | | | ● | | |
| バイナリ転送 | | | | | ● | Committed |
| ストリーミング転送 | | | | | ● | Committed |

# ロジカル・レプリケーション
## コンフリクトの発生

– コンフリクトが発生する条件とスタンバイ・インスタンスの状態

| スタンバイ・インスタンスの状態 | レプリケーション状態 | 備考 |
|---|---|---|
| 主キー違反 | レプリケーション停止 | logical replication worker 停止 |
| 列が無い | レプリケーション停止 | logical replication worker 停止 |
| 列のデータ型変換エラー | レプリケーション停止 | logical replication worker 停止 |
| 更新タプルが無い | レプリケーション継続 | |
| 削除タプルが無い | レプリケーション継続 | |
| テーブルがロックされている | レプリケーション一時停止 | |
| タプルがロックされている | レプリケーション一時停止 | |
| Replica Identity が無い | UPDATE / DELETE 実行不可 | |

– テキスト変換でエラー
  – フォーマットは bytea_output (hex) パラメータで決定される（デフォルトでは 2 倍強のメモリーが必要）
  – メモリー取得エラー発生の可能性がある

# ロジカル・レプリケーション
## コンフリクトの解決

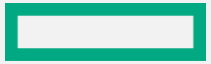
- コンフリクトの解決方法
  - スタンバイ・インスタンス側でコンフリクト対象タプルを解消
  - 問題が解決すると、自動的にレプリケーション再開
  - コンフリクト発生トランザクションをスキップ

```
subdb=# SELECT pg_replication_origin_advance('pg_16425', '0/7200B4F0');

 pg_replication_origin_advance
-------------------------------

(1 row)
```

# JIT



# JIT
## 概要

- SQL 文の実行(Executer)をネイティブ・コンパイルしたコードで実行
- LLVM を統合(https://llvm.org/)
- 一定コスト(jit_above_cost)以上の SQL 文に対して実行される

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1;
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=17906.00..17906.01 rows=1 width=8) (actual time=90.410..90.… loops=1)
   ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=0) (actual… loops=1)
 Planning Time: 0.026 ms
 JIT:
   Functions: 2
   Options: Inlining false, Optimization false, Expressions true, Deforming true
   Timing: Generation 1.633 ms, Inlining 0.000 ms, Optimization 1.669 ms, Emission 22.193
ms, Total 25.495 ms
 Execution Time: 187.440 ms
(8 rows)
```

# JIT
## 設定

– 関連するパラメーター

| パラメーター名 | 説明 | デフォルト値　備考 |
|---|---|---|
| jit | JIT 機能が有効か | on |
| jit_above_cost | JIT コンパイルを起動するかを決めるコスト | 100000 |
| jit_debugging_support | 生成した関数をデバッガに登録 | off |
| jit_dump_bitcode | 生成した LLVM IR を出力するか | off |
| jit_expressions | 式を JIT コンパイルするか | on |
| jit_inline_above_cost | 関数と演算子のインラインかを行うかを決めるコスト | 500000 |
| jit_optimize_above_cost | より高度な最適化を行うかを決めるコスト | 500000 |
| jit_profiling_support | perf によるプロファイリング・データの出力 | off |
| jit_provider | JIT プロバイダ名の参照 | llvmjit |
| jit_tuple_deforming | デフォーミングを行う | on |

# JIT
## パブリッククラウド

– パブリック・クラウドにおける対応

| パラメーター名 | RDS | RDS Aurora | Azure | Azure Hyperscale | GCP | 備考 |
|---|---|---|---|---|---|---|
| jit | off | off | 参照不可 | off | on | |
| jit_provider | llvmjit | llvmjit | 参照不可 | llvmjit | llvmjit | |

– 現状では実行計画から動作が確認できるのは RDS（≠Aurora）のみ

# THANK YOU

Mail: noriyoshi.shinoda@hpe.com
Twitter: @nori_shinoda