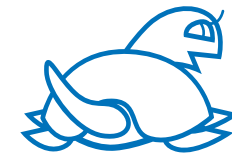




**Hewlett Packard**  
Enterprise



# POSTGRESQL 主要機能の進化と 最新バージョン情報

Noriyoshi Shinoda

Jun 22, 2024

# SPEAKER

- 篠田 典良(しのだ のりよし)
- 所属
  - 日本ヒューレット・パカード合同会社
- 現在の業務など
  - 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 Pro (2009～)
  - PostgreSQL 開発 (PostgreSQL 10～17 beta)
- 関連する URL
  - Redgate 100 in 2022 (Most influential in the database community 2022)
    - <https://www.red-gate.com/hub/redgate-100/>
  - 「PostgreSQL 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>



Featured in  
The Redgate 100



# AGENDA

---

- PostgreSQL 概要
- ロジカルレプリケーション
- パーティショニング
- パラレルクエリー
- その他の主要な新機能

# POSTGRESQL 概要

---

## 歴史とバージョン



# POSTGRESQL 概要

## PostgreSQL とは

- オープンソースで開発されている RDBMS
  - MySQL, MariaDB, SQLite, Firebird 等の仲間
- ライセンスは PostgreSQL License
  - ≒BSD License
- 活発な開発コミュニティ
  - The PostgreSQL Global Developer Group (<http://www.postgresql.org/>)
  - Commitfests (<https://commitfest.postgresql.org/>)
  - 日本 PostgreSQL ユーザ会 (<http://www.postgresql.jp/>)
  - PostgreSQL Enterprise Consortium (<http://www.pgecons.org/>)
- バージョン構成
  - 原則として 1 年毎に新バージョンが公開
  - 現時点の最新バージョンは PostgreSQL 16 (16.3)
  - PostgreSQL 17 Beta 1 開発中

# POSTGRESQL 概要

## 歴史

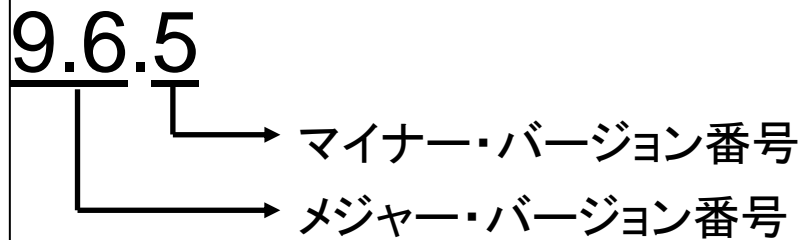
- 1974年 Ingres プロトタイプ
  - HPE NonStop SQL, SAP Sybase ASE, Microsoft SQL Serverの元になる
- 1989年 POSTGRES 1.0～
- 1997年 PostgreSQL 6.0～
  - GEQO, MVCC, マルチバイト
- 2000年 PostgreSQL 7.0～
  - WAL, TOAST
- 2005年 PostgreSQL 8.0～
  - 自動 VACUUM, HOT, PITR
- 2017年10月 PostgreSQL 10
  - ロジカル・レプリケーション、パーティショニング
- 2023年10月 PostgreSQL 16
- 2024年5月 PostgreSQL 17 Beta 1

本日の範囲

# POSTGRESQL 概要

## バージョン

- PostgreSQL 9.6 まで
  - 2つの数字がメジャー、最後の数字がマイナー



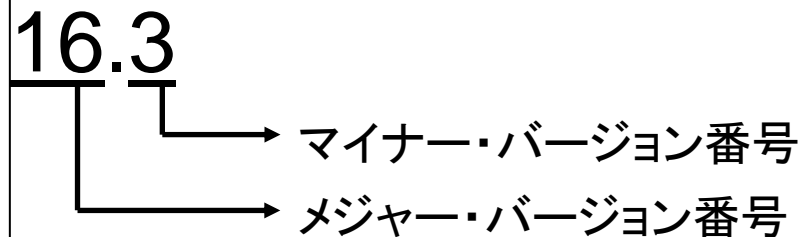
9.6.5

マイナー・バージョン番号

メジャー・バージョン番号

The diagram shows the version number 9.6.5. A horizontal line is drawn under the '9' and '6'. An arrow points from the '5' to the text 'マイナー・バージョン番号' (Minor version number). Another arrow points from the '9' and '6' to the text 'メジャー・バージョン番号' (Major version number).

- PostgreSQL 10 以降
  - 最初の数がメジャー、最後の数がマイナー



16.3

マイナー・バージョン番号

メジャー・バージョン番号

The diagram shows the version number 16.3. A horizontal line is drawn under the '16'. An arrow points from the '3' to the text 'マイナー・バージョン番号' (Minor version number). Another arrow points from the '16' to the text 'メジャー・バージョン番号' (Major version number).

# POSTGRESQL 概要

## 派生した製品

---

- EDB Postgres Enterprise Edition (EnterpriseDB)
  - Oracle Database 互換機能
- Amazon Aurora PostgreSQL (AWS)
  - OLTP
- Azure Database for PostgreSQL – Hyperscale (Citus)
  - OLTP / DWH
- AlloyDB (Google Cloud)
  - OLTP / DWH
- Vertica (OpenText)
  - DWH
- その他
  - Amazon Redshift, YugabyteDB, CockroachDB, etc





# LOGICAL REPLICATION

---

部分レプリケーション



# LOGICAL REPLICATION

## 概要

- ロジカル・レプリケーションとは？
  - PostgreSQL 10 以降で利用可能
  - 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 Read / Write可能
  - SQL 文の結果が同一であることを保証 (=Logical)
  - ≒ MySQL の Row-based Replication (RBR)
- ストリーミング・レプリケーション (Physical Streaming Replication) とは？
  - PostgreSQL 9.0 以降で利用可能
  - データベース・クラスタ全体のレプリカ作成機能
  - レプリケーション先インスタンスは更新不可 (INSERT / UPDATE / DELETE 実行不可)
  - 物理的に同一ブロックを作成 (=Physical)

# LOGICAL REPLICATION

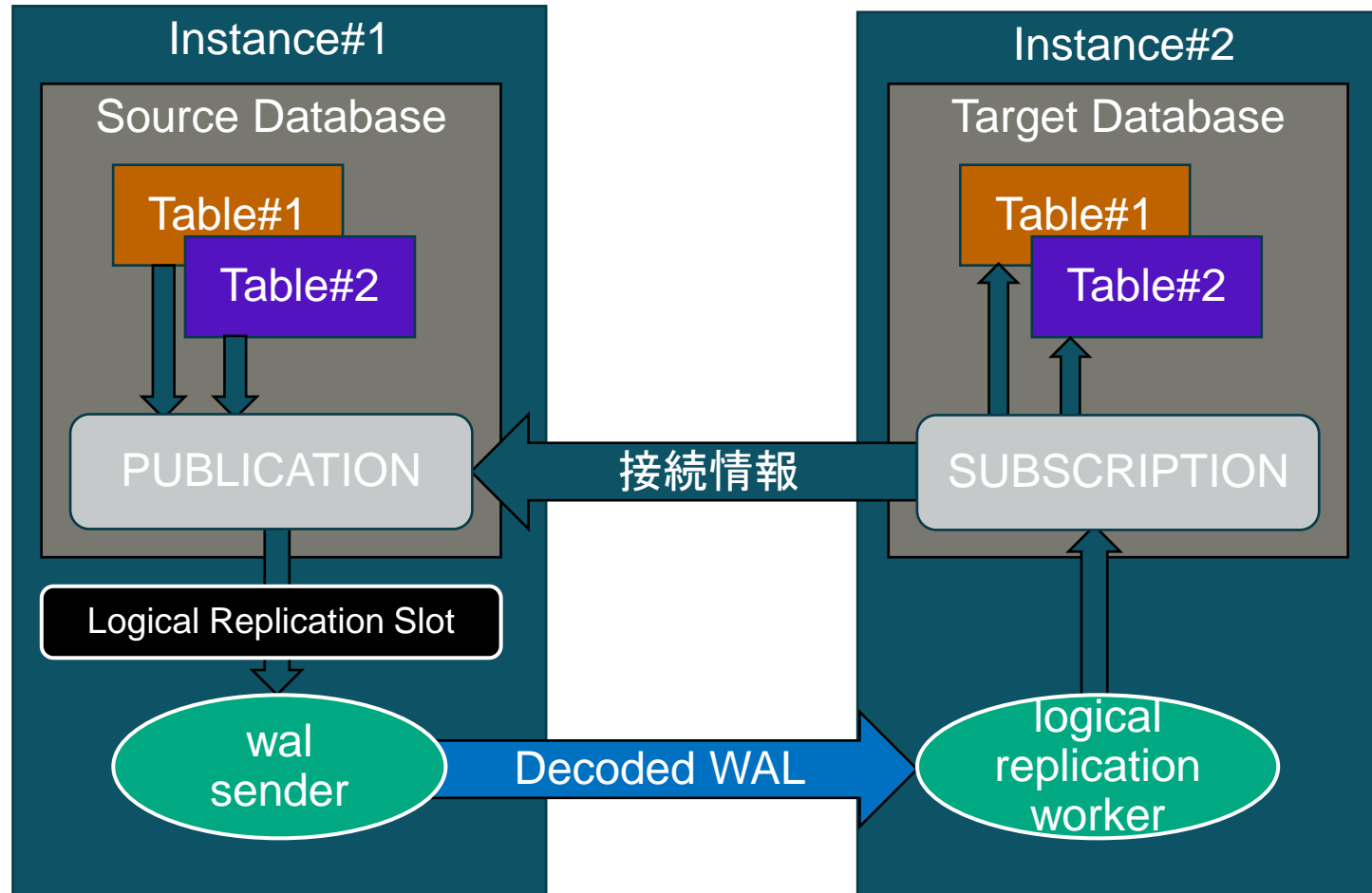
## オブジェクト

---

- 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
  - 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
  - CREATE SUBSCRIPTION 文で作成
  - 作成時に接続先インスタンスの接続情報と PUBLICATION 名を指定

# LOGICAL REPLICATION

## 概要



# LOGICAL REPLICATION

## バージョン毎の進化

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		●				
デコード用メモリー設定パラメーター				●		
ストリーミング化					●	
バイナリ転送					●	
初期データ転送と更新の分離					●	
待機イベント追加					●	



# LOGICAL REPLICATION

## バージョン毎の進化

構文／環境	15	16	備考
列指定レプリケーション	●		
行指定レプリケーション	●		
スキーマ内の全テーブル・レプリケーション	●		
LSNの更新スキップ	●		
エラー発生時の SUBSCRIPTION 無効化	●		
2 Phase Commit オプション	●		
パラレル適用		●	
origin オプション		●	
run_as_owner オプション		●	
初期データのバイナリ転送		●	
pg_create_subscription 事前定義ロール		●	



# LOGICAL REPLICATION

## レプリケーション・スロットの同期 (PostgreSQL 17 Beta)

- ストリーミング・レプリケーションのスタンバイとレプリケーション状況を同期
  - ストリーミング・レプリケーションのスタンバイが昇格した場合でも、ロジカル・レプリケーションの情報が維持される。
    - レプリケーション・スロットの failover オプション
    - SUBSCRIPTION の failover オプション
  - ストリーミング・レプリケーションに WAL を送信してから ロジカル・レプリケーションのデータ送信
- 関連する追加パラメーター

パラメーター名	説明	デフォルト値
standby_slot_names	ストリーミング・レプリケーション用スロット名	"
sync_replication_slots	レプリケーション・スロットの同期を行うか	Off



# LOGICAL REPLICATION

## pg\_createsubscriber コマンド (PostgreSQL 17 Beta)

- ストリーミング・レプリケーションのスタンバイ環境をロジカル・レプリケーションに変換するコマンド
  - 停止したスタンバイ・インスタンスを変換する
  - ロジカル・レプリケーションの初期データ移行を簡易化する目的
- 実行例

```
$ pg_createsubscriber -D data.stby --publisher-server='host=dbsvr1 port=5432 dbname=postgres'  
LOG:  redirecting log output to logging collector process  
HINT:  Future log output will appear in directory "log".  
LOG:  redirecting log output to logging collector process  
HINT:  Future log output will appear in directory "log".  
...
```



# PARALLEL QUERY

---

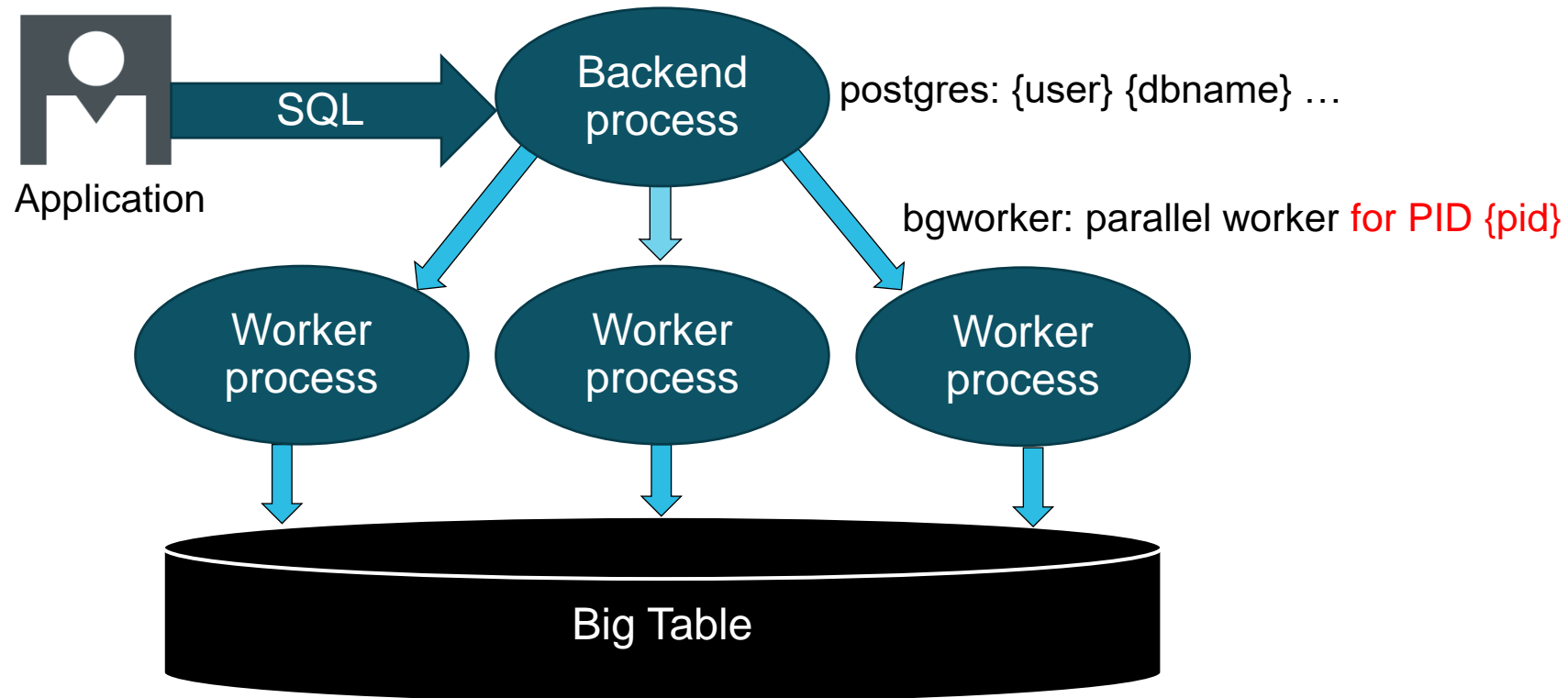
SQL 文の並列処理



# PARALLEL QUERY

## 概要

- 単一の SQL 文を**複数のプロセスで並列**に処理を行う



# PARALLEL QUERY

## 実行計画

- 大規模テーブル検索時に自動的に並列処理が行われる

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1;  
              QUERY PLAN
```

```
-----  
Finalize Aggregate  (cost=11614.55..11614.56 rows=1 width=8) (actual time=1106.746..1106.747 rows=1 ...)  
  -> Gather  (cost=11614.33..11614.54 rows=2 width=8) (actual time=1105.972..1106.766 rows=3 loops=1)  
      Workers Planned: 2  
      Workers Launched: 2  
        -> Partial Aggregate  (cost=10614.33..10614.34 rows=1 width=8) (actual time=1087.334.. ...)  
            -> Parallel Seq Scan on data1  (cost=0.00..9572.67 rows=416667 width=0) (actual time= ...)
```

```
Planning Time: 0.030 ms
```

```
Execution Time: 1106.803 ms
```

```
(8 rows)
```

# PARALLEL QUERY

## 機能の進化

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			●		
SERIALIZABLE トランザクション分離レベル				●	



# PARALLEL QUERY

## 機能の進化

構文／環境	13	14	15	16	備考
インデックスVACUUMの並列化	●				
REFRESH MATERIALIZED VIEWの並列化		●			
SELECT DISTINCT文の並列化			●		
RIGHT   OUTER JOIN文の暗号化				●	
STRING_AGG / ARRAY_AGG 関数の並列化				●	



# PARTITIONING

---

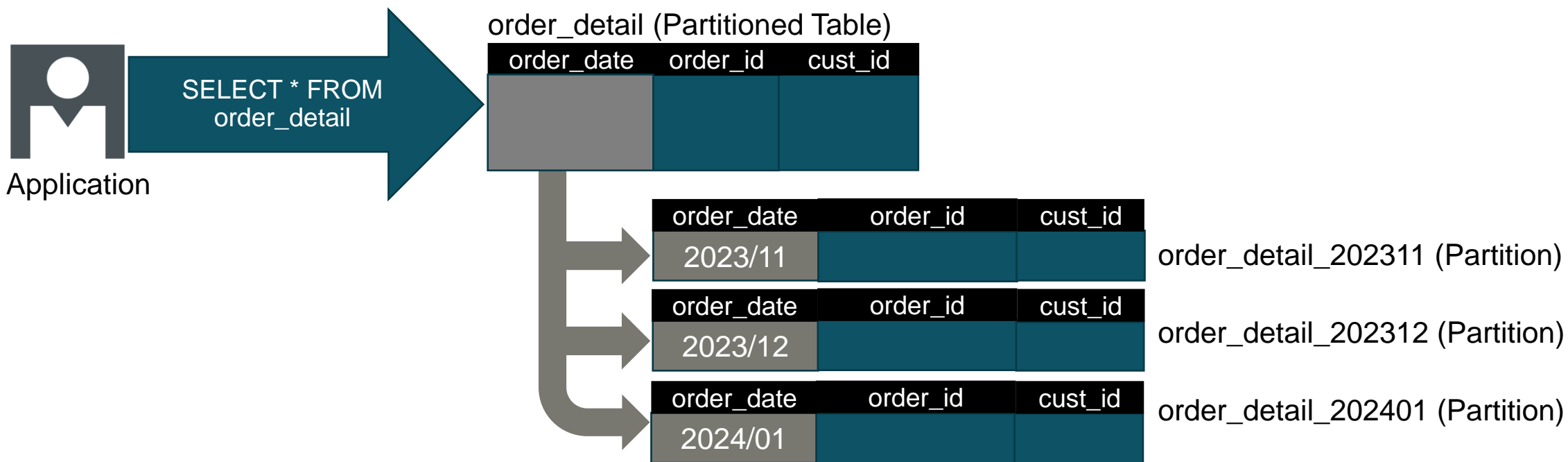
## 大規模テーブルの物理分割



# PARTITIONING

## 概要

- 大規模な**テーブルを物理的に分割**する機能
- 一般的に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



# PARTITIONING

## 概要

---

- 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  $\leq$  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 から利用可能



# PARTITIONING

## 実行計画

```
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_y2006m02 PARTITION OF measurement
           FOR VALUES FROM ('2006-02-01') TO ('2006-03-01');
```

```
CREATE TABLE
```

```
postgres=> CREATE TABLE measurement_y2007m11 PARTITION OF measurement
           FOR VALUES FROM ('2007-11-01') TO ('2007-12-01');
```

```
CREATE TABLE
```

```
...
```

```
postgres=> EXPLAIN SELECT * FROM measurement WHERE logdate = '2007-12-01';
           QUERY PLAN
```

---

```
Seq Scan on measurement_y2007m12 (cost=0.00..33.12 rows=9 width=16)
  Filter: (logdate = '2007-12-01'::date)
(2 rows)
```

# PARTITIONING

## バージョン毎の進化

構文／環境	10	11	12	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		●		
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			●	



# PARTITIONING

## 機能の進化

構文／環境	13	14	15	16	備考
ロジカル・レプリケーション対応	●				
BEFORE INSERT トリガー対応	●				
積極的なパーティション・ワイズ・ジョイン	●				



# PARTITIONING

## パーティションの分割 (PostgreSQL 17 Beta)

- 単一のパーティションを複数のパーティションに分割
  - LIST パーティションと RANGE パーティションのみ利用可能
  - 内部的には一時テーブルにデータを移動して入れ替えている
- 構文

```
ALTER TABLE table_name SPLIT PARTITION partition_name INTO (partition_name1 FOR VALUES value1, ...)
```

- 例
  - RANGE PARTITION 列値 1,000,000～2,000,000 が格納されたパーティション part1v1 を半分ずつに分割

```
postgres=> ALTER TABLE part1 SPLIT PARTITION part1v1 INTO (  
    PARTITION part1v2 FOR VALUES FROM (1000000) TO (1500000),  
    PARTITION part1v3 FOR VALUES FROM (1500000) TO (2000000));  
ALTER TABLE
```

# PARTITIONING

## パーティションのマージ (PostgreSQL 17 Beta)

- 複数のパーティションを単一のパーティションにマージ
  - LIST パーティションと RANGE パーティションのみ利用可能
  - 内部的には一時テーブルにデータを移動して入れ替えている
- 構文

```
ALTER TABLE table_name MERGE PARTITIONS (partition_name1, partition_name2, ...) INTO partition_name
```

- 例
  - RANGE PARTITION 列値 100, 200 が格納されたパーティション part1v1, part1v2 をマージ

```
postgres=> ALTER TABLE part1 MERGE PARTITIONS (part1v1, part1v2) INTO part1v3;
```

```
ALTER TABLE
```

```
postgres=> ¥d
```

List of relations

Schema	Name	Type	Owner
public	part1	partitioned table	demo
public	part1v3	table	demo

# その他の主要な新機能

---

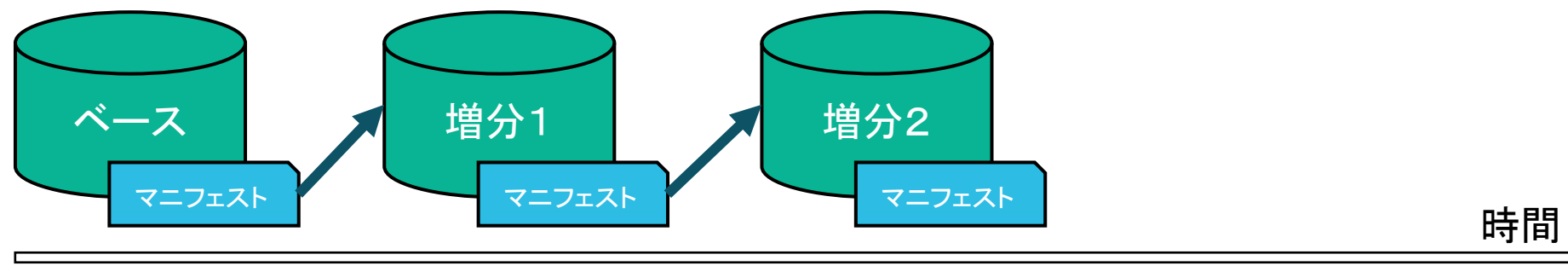
## PostgreSQL 17 新機能



# INCREMENTAL BACKUP

## 増分バックアップ

- 更新分のみのバックアップを取得可能
  - 従来は Barman 等の追加プロダクトが必要だった
- リカバリ時は基準となるベースバックアップと、増分バックアップから最新のデータを作成



- `pg_basebackup --incremental` オプションに、前回バックアップのマニフェストを指定

```
$ pg_basebackup -D back. 1
```

```
$ pg_basebackup -D back. inc1 --incremental=back. 1/backup_manifest
```

```
$ pg_basebackup -D back. inc2 --incremental=back. inc1/backup_manifest
```

初回フルバックアップ

増分バックアップ 1 回目

増分バックアップ 2 回目

# INCREMENTAL BACKUP

## 増分バックアップ

- 条件
  - WAL Summarize 機能の有効化が必要 (summarize\_wal = on)
    - \${PGDATA}/pg\_wal/summaries に WAL サマリーファイルが出力される
    - サマリーファイルは一定時間経過で消える (wal\_summary\_keep\_time = 10d)
  - バックアップ時にマニフェストファイルが必要 (デフォルトで有効)
- ベースバックアップと増分バックアップのマージ
  - pg\_combinebackup コマンドに全バックアップを指定

```
$ pg_combinebackup --output data.new back.1 back.inc1 back.inc2
```

- --output には空ディレクトリまたは存在しない名前が必要 = バックアップの上書きはできない



# MERGE STATEMENT

## 構文の追加

- BY SOURCE 句のサポート

- ソース・テーブルに存在しないが、ターゲット・テーブルには存在するレコードに対する操作

```
postgres=> MERGE INTO tgt1 AS t USING src1 AS s ON s.c1 = t.c1
          WHEN NOT MATCHED THEN INSERT VALUES (s.c1, s.c2)
          WHEN NOT MATCHED BY SOURCE THEN UPDATE SET c2='not matched';
```

- RETURNING 句のサポート

- 更新されたレコードを返す (INSERT / UPDATE / DELETE 文は既に対応済)
- 変更理由を取得する merge\_action 関数を利用可能

```
postgres=> MERGE INTO tgt1 t USING src1 s ON s.c1 = t.c1
          WHEN MATCHED THEN
            UPDATE SET c2 = 'updated'
          WHEN NOT MATCHED THEN
            INSERT (c1, c2) VALUES (s.c1, s.c2)
          RETURNING merge_action(), t.*;
```

# COPY STATEMENT

## オプションの追加

- ON\_ERROR オプション
  - データ型変換エラー発生時の動作を変更します (IGNORE 指定時にはエラーを無視)
- LOG\_VERBOSITY オプション
  - データ型変換エラーのログ出力レベルを変更できる
- FORCE\_NULL オプション
  - 列名にアスタリスク(\*)を指定可能に

```
postgres=> COPY data1 FROM stdin WITH
  (FORMAT csv, FORCE_NULL *, ON_ERROR ignore, LOG_VERBOSITY verbose);
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself, or an EOF signal.
>> 100,data1
>> ABC,data2    ← 数字以外のデータ=エラーになる
>> 300,data3
>> ¥.
NOTICE:  skipping row due to data type incompatibility at line 2 for column c1: "ABC"
NOTICE:  1 row was skipped due to data type incompatibility
COPY 2
```

# EXPLAIN STATEMENT

## オプションの追加

- MEMORY オプション
  - SQL 文解析に使用したメモリー量を出力できる
- SERIALIZE オプション
  - SQL 文実行結果のデータ量を出力できる
  - 設定値として text か binary を選択
  - ANALYZE 句と同時に指定

```
postgres=> EXPLAIN (ANALYZE, MEMORY, SERIALIZE text) SELECT * FROM data1;  
QUERY PLAN
```

---

```
Seq Scan on data1 (cost=0.00..15406.00 rows=1000000 width=10) (actual time=0.010...)
```

```
Planning:
```

```
  Memory: used=7kB allocated=8kB
```

```
Planning Time: 0.024 ms
```

```
Serialization: time=77.995 ms output=20400kB format=text
```

```
Execution Time: 150.980 ms
```

```
(6 rows)
```

# MAINTAIN PRIVILEGE

## 管理系 SQL 文の実行権限

- 管理系 SQL 文を実行できる権限
  - VACUUM, ANALYZE, REINDEX, REFRESH MATERIALIZED VIEW, CLUSTER, LOCK TABLE 文の権限
- pg\_maintain 定義済ロール
  - 全テーブルに対する MAINTAIN 権限の付与
- 実行例

```
postgres=# GRANT MAINTAIN ON data1 TO demouser1; ← テーブル data1 の管理権限を付与
GRANT
postgres=# GRANT pg_maintain TO demouser2;      ← ロール pg_maintain を付与
GRANT ROLE
```

# まとめ

---

## PostgreSQL の開発



# まとめ

## 開発は続く

- PostgreSQL の活発な開発は続く
  - ベンダーに依存しない真の OSS ソフトウェアです。
- 魅力的な新機能が追加されつつあります
  - Commitfests (<https://commitfest.postgresql.org/>) を覗いてみましょう。
- 誰でも開発に参加できます
  - Mailing-list: pgsql-hackers に登録(<https://lists.postgresql.org/>)しましょう。

# THANK YOU

---

Mail : [noriyoshi.shinoda@hpe.com](mailto:noriyoshi.shinoda@hpe.com)

Twitter : @nori\_shinoda

Qiita : @ plusultra

