Hewlett Packard
Enterprise

2021 年 5 月 21 日

PostgreSQL 14 新機能検証結果（Beta 1）

日本ヒューレット・パッカード株式会社
篠田典良

Hewlett Packard
Enterprise

目次

Hewlett Packard
Enterprise

1. 本文書について

1.1. 本文書の概要

本文書はオープンソース RDBMS である PostgreSQL 14 (14.0) Beta 1 の主な新機能について検証した文書です。

1.2. 本文書の対象読者

本文書は、既にある程度 PostgreSQL に関する知識を持っているエンジニア向けに記述しています。インストール、基本的な管理等は実施できることを前提としています。

1.3. 本文書の範囲

本文書は PostgreSQL 13 (13.3) と PostgreSQL 14 (14.0) Beta 1 の主な差分を記載しています。原則として利用者が見て変化がわかる機能について調査しています。すべての新機能について記載および検証しているわけではありません。特に以下の新機能は含みません。

- バグ解消
- 内部動作の変更によるパフォーマンス向上
- レグレッション・テストの改善
- psql コマンドのタブ入力による操作性改善
- pgbench コマンドの改善
- ドキュメントの改善、ソース内の Typo 修正
- 動作に変更がないリファクタリング

1.4. 本文書の対応バージョン

本文書は以下のバージョンとプラットフォームを対象として検証を行っています。

表 1　対象バージョン

種別	バージョン
データベース製品	PostgreSQL 13.3 (比較対象) PostgreSQL 14 (14.0) Beta 1 (2021/05/17 20:15:58)
オペレーティング・システム	Red Hat Enterprise Linux 7 Update 8 (x86-64)
Configure オプション	--with-llvm --with-ssl=openssl --with-perl --with-lz4

1.5. 本文書に対する質問・意見および責任

本文書の内容は日本ヒューレット・パッカード株式会社の公式見解ではありません。また内容の間違いにより生じた問題について作成者および所属企業は責任を負いません。本文書で検証した仕様が変更される場合があります。本文書に対するご意見等ありましたら作成者 篠田典良（Mail: noriyoshi.shinoda@hpe.com）までお知らせください。

1.6. 表記

本文書内にはコマンドや SQL 文の実行例および構文の説明が含まれます。実行例は以下のルールで記載しています。

表 2 例の表記ルール

表記	説明
#	Linux root ユーザーのプロンプト
$	Linux 一般ユーザーのプロンプト
太字	ユーザーが入力する文字列
postgres=#	PostgreSQL 管理者が利用する psql コマンド・プロンプト
postgres=>	PostgreSQL 一般ユーザーが利用する psql コマンド・プロンプト
下線部	特に注目すべき項目
<<以下省略>>	より多くの情報が出力されるが文書内では省略していることを示す
<<途中省略>>	より多くの情報が出力されるが文書内では省略していることを示す
<<パスワード>>	パスワードの入力を示す

構文は以下のルールで記載しています。

表 3 構文の表記ルール

表記	説明
斜体	ユーザーが利用するオブジェクトの名前やその他の構文に置換
[ ]	省略できる構文であることを示す
{ A | B }	A または B を選択できることを示す
…	旧バージョンと同一である一般的な構文

Hewlett Packard
Enterprise

2. PostgreSQL 14 における変更点概要

PostgreSQL 14 Beta 1 には 200 以上の新機能が追加されました。代表的な新機能と利点について説明します。

2.1. 大規模環境に対応する新機能

大規模環境に適用できる以下の機能が追加されました。

□ ロジカル・ストリーミング・レプリケーションの拡張

ロジカル・レプリケーションはトランザクションの完了を待たずにデコードされた更新情報をスタンバイ・インスタンスに転送できるようになりました。また従来はテキスト変換されていた転送データをバイナリ状態で送信できるようになりました。

□ パラレル・クエリーの拡張

REFRESH MATERIALIZED VIEW 文についてパラレル・クエリーが動作するようになりました。

□ LZ4 圧縮

TOAST 列に対して LZ4 圧縮を利用できるようになりました。

□ BRIN インデックス

Bloom フィルターを利用できるようになりました。

2.2. 信頼性向上に関する新機能

信頼性を向上させるために以下の拡張が実装されました。

□ 構造チェック機能の拡張

Contrib モジュール amcheck にオブジェクト構造をチェックする関数が追加されました。また amcheck モジュールによるチェックをコマンドラインから実行する pg_amcheck コマンドが追加されました。

□ パスワード長制限の撤廃

パスワード長に対する制限が撤廃され、非常に長いパスワードも利用できるようになりました。

2.3. 運用性向上に関する新機能

運用性を向上できる以下の機能が追加されました。

□ REINDEX 文の拡張

パーティション・テーブルに対して REINDEX 文を実行できるようになりました。このため各パーティションに対して REINDEX を実行する必要がなくなりました。また TABLESPACE 句を指定して再作成するインデックスを移動させることができるようになりました。

□ モニタリング機能の拡張

COPY コマンドの実行状況をリアルタイムに確認できるビューが追加されました。また WAL の使用状況、バックエンド・プロセスのメモリー状況を確認できるビューが追加されました。Contrib モジュール pg_stat_statements では REFRESH MATERIALIZED VIEW 文等ユーティリティ文の情報取得が可能になり、取得できる統計情報が大幅に増えました。

2.4. 将来の新機能に対する準備

将来のバージョンで提供される機能の準備が進みました。

□ 添え字によるデータアクセス

新しいデータ型に対して添え字によるデータアクセスを行う基盤が実装されました。hstore モジュールはこの機能を利用できるようになりました。

2.5. 非互換

PostgreSQL 14 は PostgreSQL 13 から以下の仕様が変更されました。

2.5.1. 関数

以下の関数が変更されました。

□ LEAD / LAG

関数戻り値のデータ型が一部 anyelement 型から anycompatible 型に変更されました。

□ EXTRACT

extract 関数は戻り値のデータ型が double precision 型から numeric 型に変更されました。これまでのバージョンでは extract 関数は内部で date_part 関数をコールしていましたが独立しました。date_part 関数の戻り値のデータ型は double precision のままです。

□ VAR_SAMP, STDDEV_SAMP

numeric 型の NaN 値を指定された戻り値が NaN から NULL に変更されました。

2.5.2. 正規表現

改行を区別するモードにおいて、エスケープ¥W および¥D が改行コードと一致するようになりました。

例 1 改行の一致 (PostgreSQL 13)

```
postgres=> SELECT regexp_match(E'A¥nC', '¥D¥D¥D', 'n') ;
 regexp_match
--------------
 null
(1 row)
```

例 2 改行の一致 (PostgreSQL 14)

```
postgres=> SELECT regexp_match(E'A¥nC', '¥D¥D¥D', 'n') ;
 regexp_match
--------------
 {"A         +
 C"}
(1 row)
```

2.5.3. 階乗の計算

階乗の計算結果と演算子が変更されました。

□ マイナス値の階乗

マイナス値の階乗計算はエラーになります。

例 3 階乗の計算 (PostgreSQL 13)

```
postgres=> SELECT factorial(-4) ;
factorial
-----------
         1
(1 row)
```

例 4 階乗の計算 (PostgreSQL 14)

```
postgres=> SELECT factorial(-4) ;
ERROR:  factorial of a negative number is undefined
```

□ 演算子の削除

階乗演算子（!および!!）は削除されました。

例 5 階乗の計算 (PostgreSQL 14)

```
postgres=> SELECT 4! ;
RROR:  syntax error at or near ";"
LINE 1: SELECT 4! ;
                  ^
```

□ 関数の削除

関数 numeric_fac は削除されました。

2.5.4. 演算子

Contrib モジュール cube、hstore、intarray、seg から非推奨の演算子@と~が削除されました。

2.5.5. マニュアル

マニュアルの表記とプログラム・ソース内のデフォルトロール（Default Roles）は事前定義ロール（Predefined Roles）に変更されました。

2.5.6. トランザクション周回の警告

トランザクションIDの周回に対する残トランザクションの閾値ポイントが以下のように変更されました。

表 4 トランザクション周回の警告

比較	PostgreSQL 13	PostgreSQL 14	備考
警告	1,100 万	4,000 万	
システム停止	100 万	300 万	

2.5.7. 起動オプション

postmaster 起動時に追加オプションを指定する-o オプションは削除されました。

2.5.8. 拡張統計情報

システムカタログに対する CREATE STATISTICS 文は実行できなくなりました。この制約は旧バージョンにもバックポートされます。

例 6 システムカタログに対する CREATE STATISTICS 文

```
postgres=# CREATE STATISTICS stat1_collation ON collname, collowner
        FROM pg_collation ;
ERROR:  permission denied: "pg_collation" is a system catalog
```

2.5.9. プロトコル

クライアントとバックエンド間通信プロトコルのバージョン 2 が削除されました。このプロトコルは PostgreSQL 7.4 から利用されていました。

2.5.10. ALTER OPERATOR

ALTER OPERATOR 文、DROP OPERATOR 文では右オペランドに NONE は指定できなくなりました。

2.5.11. ALTER TABLE

ALTER TABLE ONLY 文では DROP EXPRESSION 句は指定できなくなりました。

2.5.12. CREATE LANGUAGE

CREATE LANGUAGE 文と DROP LANGUAGE 文ではシングル・クオーテーションで名前を指定するとエラーになります。

例 7 CREATE LANGUAGE 文（PostgreSQL 14）

```
postgres=# CREATE LANGUAGE 'plperl' ;
ERROR:  syntax error at or near "'plperl'"
LINE 1: CREATE LANGUAGE 'plperl';
```

2.5.13. LIBPQ 接続文字列

接続文字列の authtype および tty は削除されました。SSL 圧縮を行うクライアント接続設定 sslcompression はバックエンドでは無視されます。

2.5.14. findoidjoins

findoidjoins コマンドは削除されました。

2.5.15. intarray

intarray モジュールの包含演算子（<@, @>）が GiST インデックスを使用しなくなりました。

2.5.16. pg_hba.conf

clientcert オプションの設定値に「0」、「1」、「no-verify」は使用できなくなりました。

2.5.17. pg_standby

pg_standby コマンドは削除されました。

2.5.18. tablefunc

normal_rand 関数の第一パラメーターに負の値が指定できなくなりました。

例 8 normal_rand 関数

```
postgres=# CREATE EXTENSION tablefunc ;
CREATE EXTENSION
postgres=# SELECT normal_rand(-1, 1.2, 1.3) ;
ERROR:  number of rows cannot be negative
```

Hewlett Packard
Enterprise

3. 新機能解説

3.1. アーキテクチャの変更

3.1.1. システムカタログの変更

以下のシステムカタログやビューが変更されました。

表 5 追加されたシステムカタログ／ビュー

カタログ／ビュー名	説明
pg_backend_memory_contexts	バックエンド・プロセスのメモリー情報を出力します。
pg_stat_progress_copy	COPY 文の実行状況を出力します。
pg_stat_replication_slots	レプリケーション・スロットの利用状況を出力します。
pg_stat_wal	WAL データ出力状況を出力します。
pg_stats_ext_exprs	式に関する拡張統計を出力します。

表 6 情報スキーマ（information_schema）内に追加されたビュー

ビュー名	説明
routine_column_usage	関数またはプロシージャによって使用される列を識別します。現状では追跡されていません。
routine_routine_usage	関数またはプロシージャによって使用される関数を識別します。現状では一部追跡されていません。
routine_sequence_usage	関数またはプロシージャによって使用されるシーケンスを識別します。現状では一部追跡されていません。
routine_table_usage	関数またはプロシージャによって使用されるテーブルを識別します。現状では追跡されていません。

表 7 列が追加されたシステムカタログ／ビュー

カタログ／ビュー名	追加列名	データ型	説明
pg_attribute	attcompression	char	列の圧縮メソッド
pg_inherits	inhdetachpending	boolean	デタッチ中のパーティションかどうか

カタログ／ビュー名	追加列名	データ型	説明
pg_locks	waitstart	timestamp with time zone	サーバー・プロセスがロック待ちを開始した時刻
pg_prepared_statements	generic_plans	bigint	汎用プランの実行数
	custom_plans	bigint	カスタムプランの実行数
pg_proc	prosqlbody	pg_node_tree	パース済の SQL ファンクション
pg_range	rngmultitypid	oid	複数範囲型の OID
pg_replication_slots	two_phase	boolean	2 フェーズ・コミットのサポート
pg_stat_activity	query_id	bigint	直近に実行されたクエリーの ID
pg_stat_database	session_time	double precision	データベース上で費やされたセッション時間（ミリ秒）
	active_time	double precision	データベース上で実行された SQL 文の実行時間（ミリ秒）
	idle_in_transaction_time	double precision	データベース上で費やされたアイドル時間（ミリ秒）
	sessions	bigint	セッション数
	sessions_abandoned	bigint	クライアントが停止して破棄されたセッション数
	sessions_fatal	bigint	エラーにより終了したセッション数
	sessions_killed	bigint	強制終了したセッション数
pg_stats_ext	exprs	text[]	拡張統計を取得する式
pg_statistic_ext	stxexprs	pg_node_tree	拡張統計をカバーする式のリスト
pg_statistic_ext_data	stxdexpr	pg_statistic []	この統計オブジェクトでカバーされる式のリスト。
pg_subscription	subbinary	boolean	バイナリ転送を行うか
	substream	boolean	ストリーミング転送を行うか
pg_type	typsubscript	regproc	サブスクリプティングを行う関数の OID

表 8 列が削除されたシステムカタログ／ビュー

カタログ名	削除列名	説明
pg_stat_ssl	compression	圧縮が無効になったため削除されました。

表 9 出力内容が変更されたシステムカタログ／ビュー

カタログ／ビュー名	説明
pg_class	reltuples 列の初期値が-1 に変更されました。
pg_type	typtype 列に複数範囲型を示す'm'が出力される場合があります。
pg_subscription_rel	srsubstate 列に'f' = finished table copy が出力される場合があります。
pg_stat_activity	archiver プロセスと walsender プロセスの情報が出力されます。
pg_statistic_ext	stxkind 列に'e'=式による拡張統計が出力される場合があります。

追加されたシステムカタログやビューから、いくつか詳細を以下に記載します。

□ pg_backend_memory_contexts

pg_backend_memory_contexts ビューは現在のセッションに対応するサーバー・プロセスのメモリー・コンテキストの使用状況を確認できます。このビューは SUPERUSER 属性を持つユーザーのみ参照できます。

表 10 pg_backend_memory_contexts ビュー

列名	データ型	説明
name	text	メモリー・コンテキスト名
ident	text	メモリー・コンテキスト ID
parent	text	メモリー・コンテキストの親名
level	integer	TopMemoryContext からの距離
total_bytes	bigint	合計バイト数
total_nblocks	bigint	合計ブロック数
free_bytes	bigint	空きバイト数
free_chunks	bigint	空きチャンク数
used_bytes	bigint	使用バイト数

例 9 pg_backend_memory_contexts ビューの検索

```
postgres=# SELECT name, parent, level, total_bytes FROM
                pg_backend_memory_contexts ;
          name           |       parent       | level | total_bytes
-------------------------+--------------------+-------+-------------
 TopMemoryContext        | null               |     0 |       68704
 TopTransactionContext   | TopMemoryContext   |     1 |        8192
 CFuncHash               | TopMemoryContext   |     1 |        8192
 Operator lookup cache   | TopMemoryContext   |     1 |       24576
 TableSpace cache        | TopMemoryContext   |     1 |        8192
 Type information cache  | TopMemoryContext   |     1 |       24376
 Record information cache | TopMemoryContext  |     1 |        8192
 RowDescriptionContext   | TopMemoryContext   |     1 |        8192
 MessageContext          | TopMemoryContext   |     1 |       32768
<<<以下省略>>>
```

□ pg_stat_progress_copy

このビューには COPY 文の実行状態が出力されます。

表 11 pg_stat_progress_copy ビュー

列名	データ型	説明
pid	integer	バックエンド・プロセス ID
datid	oid	データベース OID
datname	name	データベース名
relid	oid	COPY 文対象オブジェクト OID
command	text	コマンド名
type	text	タイプ
bytes_processed	bigint	処理されたバイト数
bytes_total	bigint	合計バイト数
tuples_processed	bigint	処理されたタプル数
tuples_excluded	bigint	除外されたタプル数

例 10 pg_stat_progress_copy ビューの検索

```
postgres=> SELECT * FROM pg_stat_progress_copy ;
-[ RECORD 1 ]---+----------
pid             | 83409
datid           | 13887
datname         | postgres
relid           | 16385
command         | COPY FROM
type            | FILE
bytes_processed | 22806528
bytes_total     | 138888897
tuples_processed | 1703999
tuples_excluded | 0
```

□ pg_stat_replication_slots

ロジカル・レプリケーション・スロットの状態が出力されます。ビューの内容をリセットするには pg_stat_reset_replication_slot 関数を実行します。

表 12 pg_stat_replication_slots ビュー

列名	データ型	説明
slot_name	text	レプリケーション・スロット名
spill_txns	bigint	ディスクにあふれたトランザクション数
spill_count	bigint	ディスクにあふれた回数
spill_bytes	bigint	ディスクにあふれたバイト数
stream_txns	bigint	ストリーミングされたトランザクション数
stream_count	bigint	ストリーミングされた回数
stream_bytes	bigint	ストリーミングされたバイト数
total_txns	bigint	合計トランザクション数
total_bytes	bigint	合計バイト数
stats_reset	timestamp with time zone	統計情報がリセットされた日時

例 11 pg_stat_replication_slots ビューの検索

```
postgres=> SELECT * FROM pg_stat_replication_slots ;
-[ RECORD 1 ]+----------
slot_name    | sub1
spill_txns   | 3
spill_count  | 9
spill_bytes  | 416939406
stream_txns  | 0
stream_count | 0
stream_bytes | 0
total_txns   | 0
total_bytes  | 0
stats_reset  |
```

□ pg_stat_wal

pg_stat_wal ビューには WAL の出力状況が 1 タプルのみ出力されます。このビューの値は pg_stat_reset_shared 関数に'wal'を指定することでリセットできます。wal_write_time 列と wal_sync_time 列は、track_wal_io_timing パラメーターが on の場合にのみ値が出力されます。

表 13 pg_stat_wal ビュー

列名	データ型	説明
wal_records	bigint	生成された WAL レコード数
wal_fpi	bigint	生成された WAL フルページ数
wal_bytes	numeric	生成された WAL バイト数
wal_buffers_full	bigint	WAL buffer がいっぱいになったため WAL 情報がディスクに出力された回数
wal_write	bigint	WAL バッファから書き込みが行われた回数
wal_sync	bigint	WAL 情報がストレージに同期を行われた回数
wal_write_time	double precision	WAL バッファが書き込まれた時間
wal_sync_time	double precision	WAL 情報をストレージ同期に要した時間
stats_reset	timestamp with time zone	ステータスがリセットされた日時

例 12 pg_stat_wal ビューの検索

```
postgres=> SELECT * FROM pg_stat_wal ;
-[ RECORD 1 ]----+------------------------------
wal_records      | 52417581
wal_fpi          | 98
wal_bytes        | 4139801371
wal_buffers_full | 341903
wal_write        | 342723
wal_sync         | 876
wal_write_time   | 0
wal_sync_time    | 0
stats_reset      | 2021-05-20 23:21:41.173493+09
```

3.1.2. ロジカル・レプリケーションの拡張

ロジカル・レプリケーションには以下の機能が追加されました。

□ ロジカル・ストリーミング・レプリケーション

従来のロジカル・レプリケーションはトランザクションのコミット単位でサブスクリプション・インスタンスに更新情報を転送していました。PostgreSQL 14 ではトランザクションのコミットを待たずにデータを転送するロジカル・ストリーミング・レプリケーション機能を利用することができます。ロジカル・ストリーミング・レプリケーションを行う場合には SUBSCRIPTION の作成または変更時に streaming 属性を on に設定します。STREAMING 属性のデフォルト値は off です。

例 13 ストリーミング転送の設定

```
postgres=# CREATE SUBSCRIPTION sub1 CONNECTION 'host=remhost1 dbname=postgres
user=postgres password=<<PASSWORD>>' PUBLICATION pub1 WITH (streaming = on) ;
CREATE SUBSCRIPTION
postgres=# SELECT subname, substream  FROM pg_subscription WHERE
        subname = 'sub1' ;
subname | substream
--------+-----------
 sub1   | t
(1 row)
```

□ バイナリ転送

従来のロジカル・レプリケーションはデータをテキストに変換して転送していました。SUBSCRIPTION の属性 binary を on に設定することで、バイナリ転送が可能になりました。これによりネットワークの転送量削減を期待できます。属性を指定しない場合は従来通りテキスト形式で転送されます。

例 14 バイナリ転送の設定

```
postgres=# CREATE SUBSCRIPTION sub2 CONNECTION 'host=remhost1 dbname=postgres
user=postgres password=<<PASSWORD>>' PUBLICATION pub2 WITH (binary = on) ;
CREATE SUBSCRIPTION
postgres=# SELECT subname, subbinary  FROM pg_subscription WHERE
        subname = 'sub2' ;
subname | subbinary
---------+-----------
 sub2    | t
(1 row)
```

□ 初期同期と更新トランザクションの分離

データの初期同期と更新情報のトランザクションが分離されました。これによりエラー発生時にデータの初期同期からやり直す必要がなくなりました。初期データの移行時には追加の一時ロジカル・レプリケーション・スロットが必要になります。このレプリケーション・スロットの名前は pg_%u_sync_%u_%llu (SUBSCRIPTION の oid、テーブルの relid、System ID) です。

□ 待機イベント追加

以下の待機イベントが確認できるようになりました。

表 14 追加された待機イベント

待機イベント名	説明
LogicalChangesRead	論理変更ファイルからの読み込み待ち。
LogicalChangesWrite	論理変更ファイルへの書き込み待ち。
LogicalSubxactRead	論理サブ・トランザクション・ファイルからの読み込み待ち。
LogicalSubxactWrite	論理サブ・トランザクション・ファイルへの書き込み待ち。

□　メッセージの追加

プラグインはロジカル・デコードメッセージがレプリケーションストリームに書き込まれるかどうかを制御する新しいパラメーター「メッセージ」を受け入れます。この機能は将来の機能に向けた追加機能です。

3.1.3. パラレル・クエリーの拡張

REFRESH MATERIALIZED VIEW 文の SELECT 部分がパラレルに実行できるようになりました。下記の例は REFRESH MATERIALIZED VIEW 文を実行したときの auto_explain モジュールのログです。

例 15 実行計画のログ

```
LOG:  duration: 6649.485 ms  plan:
        Query Text: REFRESH MATERIALIZED VIEW mview1;
        Gather  (cost=0.00..95721.67 rows=10000000 width=12)
          Workers Planned: 2
          ->  Parallel Seq Scan on data1  (cost=0.00..95721.67 rows=4166667
width=12)
```

3.1.4. TOAST 列の LZ4 圧縮

TOAST 列データの圧縮方法に LZ4 を指定できるようになりました。LZ4 圧縮を行うためには configure コマンドのオプションに--with-lz4 を指定した環境が必要です。CREATE TABLE 文で TOAST 列の圧縮を行うには列属性 COMPRESSION 句に pglz または lz4 を指定します。省略時の値は default_toast_compression によって決定されます。ALTER TABLE 文（または ALTER MATERIALIZED VIEW 文）では列定義の変更構文に SET COMPRESSION 句が指定できます。ALTER TABLE 文で圧縮方法を変更した場合でも既存のタプルは変更されません。

例 16 LZ4 圧縮

```
postgres=> CREATE TABLE compress1(c1 INT, c2 TEXT COMPRESSION lz4) ;
CREATE TABLE
postgres=> ¥d+ compress1
                                        Table "public.compress1"
 Column |  Type   | Collation | Nullable | Default | Storage  | Compression | …
--------+---------+-----------+----------+---------+----------+-------------+ …
 c1     | integer |           |          |         | plain    |             | …
 c2     | text    |           |          |         | extended | lz4         | …
Access method: heap

postgres=> ALTER TABLE compress1 ALTER COLUMN c2 SET COMPRESSION pglz ;
ALTER TABLE
postgres=> ¥d+ compress1
                                        Table "public.compress1"
 Column |  Type   | Collation | Nullable | Default | Storage  | Compression | …
--------+---------+-----------+----------+---------+----------+-------------+ …
 c1     | integer |           |          |         | plain    |             | …
 c2     | text    |           |          |         | extended | pglz        | …
Access method: heap
```

CREATE TABLE (LIKE)文では圧縮定義までテーブル定義をコピーする場合には「LIKE テーブル名 INCLUDING COMPRESSION」句を指定します。各列の圧縮属性は pg_attribute カタログの attcompression 列で確認できます。'p'の場合は pglz 圧縮、'l'の場合は LZ4 圧縮です。

列値の圧縮方法を確認するには pg_column_compression 関数に列名を指定して実行します。

例 17 pg_column_compression 関数

```
postgres=> SELECT c1, pg_column_compression(c2) FROM compress1 ;
 c1  | pg_column_compression
-----+-----------------------
 100 | pglz
 200 | lz4
(2 rows)
```

3.1.5. 実行計画

SQL 文の実行計画作成機能に以下の拡張が実装されました。

□ 非同期実行

ForeignScan 処理において非同期処理を実行できるようになりました。異なるリモート・サーバー上のデータにアクセスする際に ForeignScan を並列に実行し、パフォーマンスを向上させます。この機能を利用する場合は postgres_fdw モジュールでは async_capable オプションを on に設定します（デフォルト値は off)。

例 18 非同期設定

```
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw OPTIONS
(host 'remhost1', port '5432', dbname 'postgres', async_capable 'on') ;
CREATE SERVER
```

実行計画上は Async Foreign Scan として表示されます。下記の例は、パーティション・テーブル part1 に含まれるパーティション part1v1 と part1v2 がそれぞれ FOREIGN TABLE として実装されています。

例 19 非同期実行の実行計画

```
postgres=> EXPLAIN SELECT * FROM part1 ;
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 Append  (cost=100.00..280.97 rows=1742 width=70)
   ->  Async Foreign Scan on part1v1 part1_1  (cost=100.00..136.13 rows=871 width=70)
   ->  Async Foreign Scan on part1v2 part1_2  (cost=100.00..136.13 rows=871 width=70)
(3 rows)
```

□ Result Cache

Result Cache という名前の新しいエグゼキュータノードタイプが追加されました。プランナーは、このノードタイプをプランに含めて、パラメーター化されたネストされたループ結合の内側からの結果をエグゼキューターにキャッシュさせることができます。この実行計画の利用はパラメーターenable_resultcache で制御することができます。デフォルト値は on です。現状ではこの実行計画は通常の結合と LATERAL タイプの結合で検討されます。

例 20 Result Cache ノード

```
postgres=> EXPLAIN SELECT COUNT(*),AVG(t1.unique1) FROM tenk1 t1 INNER JOIN tenk1 t2
          ON t1.unique1 = t2.twenty WHERE t2.unique1 < 1000 ;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Aggregate  (cost=508.86..508.87 rows=1 width=40)
   ->  Nested Loop  (cost=0.30..503.86 rows=1000 width=4)
         ->  Seq Scan on tenk1 t2  (cost=0.00..470.00 rows=1000 width=4)
               Filter: (unique1 < 1000)
         ->  Result Cache  (cost=0.30..0.43 rows=1 width=4)
               Cache Key: t2.twenty
               ->  Index Only Scan using tenk1_unique1 on tenk1 t1  (cost=0.29..0.42 rows=1
width=4)
                     Index Cond: (unique1 = t2.twenty)
(8 rows)
```

3.1.6. 拡張統計

式に関する拡張統計を定義することができるようになりました。式を使った拡張統計を参照するためのシステム・ビューpg_stats_ext_exprs が追加されています。

例 21 式を使った拡張統計

```
postgres=> CREATE STATISTICS extstat1_data1 ON MOD(c1, 10), MOD(c2, 10)
        FROM data1 ;
CREATE STATISTICS
postgres=> ANALYZE data1 ;
ANALYZE
postgres=> SELECT * FROM pg_stats_ext_exprs ;
-[ RECORD 1 ]----------+-----------------------------------------------------
schemaname             | public
tablename              | data1
statistics_schemaname  | public
statistics_name        | extstat1_data1
statistics_owner       | demo
expr                   | mod(c1, 10)
null_frac              | 0
avg_width              | 4
<<以下省略>>
```

3.1.7. データ型

以下のデータ型が追加／拡張されています。

□ 数値データ型の最大／最小値

numeric 型、float 型には最大値と最小値を示す Infinity と-Infinity が指定できるようになりました。非ゼロの浮動小数点を Infinity 値で割った値や、exp 関数、power 関数に-Infinity を指定した場合の戻り値ゼロになります。

例 22 Infinity/-Infinity

```
postgres=> CREATE TABLE numeric1(c1 NUMERIC, c2 float) ;
CREATE TABLE
postgres=> INSERT INTO numeric1 VALUES ('Infinity', '-Infinity') ;
INSERT 0 1
postgres=> SELECT * FROM numeric1 WHERE c1='Infinity' ;
    c1    |    c2
----------+-----------
 Infinity | -Infinity
(1 row)

postgres=> SELECT 0.1 / '-Infinity', exp('-Infinity'), power(2, '-Infinity') ;
 ?column? | exp | power
----------+-----+-------
        0 |   0 |     0
(1 row)
```

□ multirange 型

重複しない複数の範囲を示すデータ型が提供されました。具体的なデータ型は以下の通りです。

表 15 追加されたデータ型

データ型	説明
datemultirange	date 型の範囲
int4multirange	int 型の範囲
int8multirange	bigint 型の範囲
nummultirange	numeric 型の範囲
tsmultirange	timestamp 型の範囲
tstzmultirange	timestamp with time zone 型の範囲
anymultirange	複数範囲型の疑似データ型
anycompatiblemutirange	複数範囲型の任意疑似データ型

以下は bigint 型による範囲指定の例です。

例 23 multirange 型

```
postgres=> SELECT '{[1, 5), (10, 20]}'::int8multirange ;
int8multirange
----------------
 {[1,5),[11,21)}
(1 row)
postgres=> SELECT nummultirange( numrange(1, 10, '()'),
        numrange(15, 20, '[]')) ;
  nummultirange
-----------------
 {(1,10),[15,20]}
(1 row)
```

□ jsonb 型

jsonb 型に対する添え字によるアクセス方法が拡張されました。

例 24 jsonb 型の拡張

```
postgres=> SELECT ('[1, "2", null]'::jsonb)[1] ;
 jsonb
-------
 "2"
(1 row)

postgres=> SELECT ('{"age": 25}'::jsonb)['age'] ;
 jsonb
-------
 25
(1 row)
postgres=> SELECT ('{"email":"pgsql-hackers@postgresql.org",
                "phone":"+3012345678"}'::jsonb) ['phone'] ;
      jsonb
----------------
 "+3012345678"
(1 row)
```

□ pg_lsn 型

pg_lsn 型の値に対して numeric 型による加減計算を実行できるようになりました。

例 25 pg_lsn 型の計算

```
postgres=# SELECT pg_current_wal_lsn(), pg_current_wal_lsn() + 256 ;
 pg_current_wal_lsn | ?column?
--------------------+-----------
 0/15D1F88          | 0/15D2088
(1 row)
```

□ point 型

厳密な下方／上方にあるかを取得する演算子（<<|、>>|）が point 型にも使えるようになりました。

3.1.8. BRIN インデックス

BRIN インデックスには以下の演算子クラスが利用できるようになりました。

□ Bloom Filter

Bloom フィルターを使用する演算子クラス「{データ型}_bloom_ops」が追加されました。この演算子クラスには以下の属性を指定できます。

表 16 追加された属性

属性名	説明
n_distinct_per_range	ブロック範囲内の非 NULL 値の推定数を指定します。デフォルト値は-0.1 です。
false_positive_rate	インデックスによって使用される望ましい偽陽性率を指定します。デフォルト値は 0.01 です。

例 26 BRIN インデックスの Bloom Filter 使用

```
postgres=> CREATE INDEX idx1_data1 ON data1 USING brin (c1 numeric_bloom_ops
(false_positive_rate = 0.05, n_distinct_per_range = 100)) ;
CREATE INDEX
```

□ Min/Max-Multi Index

複数の最小値と最大値を格納する演算子クラス「{データ型}_minmax_multi_ops」が追加されました。この演算子クラスには以下の属性を指定できます。

表 17 追加された属性

属性名	説明
values_per_range	インデックスによって格納される値の最大数を指定します。デフォルト値は 32 で、8～32 の値を指定できます。

3.1.9. 待機イベント

以下の待機イベントが追加されました。

表 18 追加された待機イベント

イベント名	タイプ	説明
AppendReady	IPC	サブプランノードの準備待ち
BaseBackupRead	IO	ベースバックアップのファイル読み込み待ち
BufferIO	IPC	バッファ I/O の完了待ち
BufFileTruncate	IO	バッファ・ファイルのトランケート待ち
LogicalChangesRead	IO	論理変更ファイルの読み込み待ち
LogicalChangesWrite	IO	論理変更ファイルの書き込み待ち
LogicalSubxactRead	IO	論理サブトランザクションの読み込み待ち
LogicalSubxactWrite	IO	論理サブトランザクションの書き込み待ち
WalReceiverExit	IPC	WAL receiver の終了待ち

以下の待機イベントはタイプが変更されました。

表 19 タイプが変更された待機イベント

イベント名	変更前	変更後	備考
WalReceiverWaitStart	Client	IPC	

3.1.10. ストアド・プロシージャの拡張

以下の関数が追加されました。

☐ SPI_execute_extended

非推奨となった SPI_execute_plan_with_paramlist に代わる API です。実行計画の固定に使用する再所有者を指定できます。

構文

```
int SPI_execute_extended(const char *src, const SPIExecuteOptions *options)
```

☐ SPI_cursor_parse_open

クエリー文字列とパラメーターを使用してカーソルをセットアップします。

構文

```
Portal SPI_cursor_parse_open(const char *name, const char *src,
        const SPIParseOpenOptions *options)
```

☐ SPI_scroll_cursor_fetch

カーソルからいくつかの行をフェッチします。

構文

```
void SPI_scroll_cursor_fetch(Portal portal, FetchDirection direction,
        long count)
```

☐ SPI_scroll_cursor_move

カーソルを移動します。

構文

```
void SPI_scroll_move(Portal portal, FetchDirection direction,
        long count)
```

3.1.11. pg_hba.conf ファイルの拡張

行末にバックスラッシュ（¥）を追記することで単一の設定を複数行にまたがって記述できるようになりました。

例 27 pg_hba.conf ファイルの改行

```
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
host    postgres        postgres        192.168.1.219/32        ¥
                                                                md5
```

3.1.12. LIBPQ

libpq ライブラリには以下の拡張が実装されました。

□ 接続文字列 target_session_attrs

target_session_attrs パラメーターの設定値に以下の値が追加されました。

表 20 追加された設定値

設定値	説明
read-only	接続先データベースは更新できません。read-write の逆です。
primary	ストリーミング・レプリケーションのプライマリ・インスタンスに接続します。
standby	ストリーミング・レプリケーションのスタンバイ・インスタンスに接続します。
prefer-standby	スタンバイ・インスタンスが存在する場合には接続します。スタンバイ・インスタンスが存在しない場合にはプライマリ・インスタンスに接続します。

□ 接続文字列 sslcrldir

SSL 接続で CRL 保存ディレクトリを指定する sslcrldir を設定できるようになりました。

□ 接続文字列 sslsni

SSL 接続で Server Name Indication (SNI)を設定します。

□ パイプライン・モード

libpq のパイプライン・モードを使用すると、アプリケーションは、各クエリーの後に古い libpqAPI に暗黙的に含まれる FE / BE プロトコルの同期メッセージを回避できます。以下の関数が追加されました。

表 21 追加された関数

関数名	説明
PQpipelineStatus	現在のパイプライン・モードのステータスを返す
PQenterPipelineMode	パイプライン・モードに入る
PQexitPipelineMode	パイプライン・モードを抜ける
PQpipelineSync	パイプラインの同期をリクエストする

□ デバッグ・ログ

PQtraceSetFlags 関数が追加されました。

3.1.13. ECPG

ECPG には DECLARE STATEMENT 文を指定できるようになりました。

例 28 DECLARE STATEMENT 文

```
EXEC SQL BEGIN DECLARE SECTION;
    char *selectStr = "SELECT c1 FROM data1";
    long f1;
EXEC SQL END DECLARE SECTION;

int main()
{
    EXEC SQL CONNECT TO postgres USER postgres;

    EXEC SQL DECLARE stmt1 STATEMENT;
    EXEC SQL PREPARE stmt1 FROM :selectStr;
    EXEC SQL DECLARE cur1 CURSOR FOR stmt1;
    EXEC SQL OPEN cur1;
    …
}
```

3.1.14. ロール

以下のロールが追加されました。

□ pg_database_owner

pg_database_owner ロールが追加されました。このロールのメンバーは暗黙的にデータベースの所有者で構成されます。通常はテンプレート・データベースで利用されることが想定されています。

□ pg_read_all_data / pg_write_all_data

データベース上のすべてのオブジェクトの読み込みが可能となるロール pg_read_all_data と、書き込みできる pg_write_all_data ロールが追加されました。

3.1.15. ログファイル

ログファイルに追加情報が出力されるようになりました。

□ log_autovacuum_min_duration

自動 VACUUM のログ（log_autovacuum_min_duration）にインデックス単位の詳細情報が出力されるようになりました。

例 29 自動 VACUUM のインデックス情報

```
LOG:  automatic vacuum of table "postgres.public.data1": index scans: 1
        pages: 0 removed, 5406 remain, 0 skipped due to pins, 0 skipped frozen
        tuples: 500000 removed, 500000 remain, 0 are dead but not yet removable,
oldest xmin: 566
        buffer usage: 21775 hits, 0 misses, 1 dirtied
        index scan needed: 5406 pages from table (100.00% of total) had 500000
dead item identifiers removed
        index "idx1_data1": pages: 2745 in total, 0 newly deleted, 0 currently
deleted, 0 reusable
        index "idx2_data1": pages: 2745 in total, 0 newly deleted, 0 currently
deleted, 0 reusable
        avg read rate: 20.330 MB/s, avg write rate: 20.239 MB/s
        system usage: CPU: user: 0.26 s, system: 0.00 s, elapsed: 0.56 s
        WAL usage: 21687 records, 0 full page images, 5144230 bytes
```

□ log_connection

接続時のログ（log_connection）に認証に関する追加情報が出力されます。

例 30 接続時の認証情報ログ

```
LOG:  connection received: host=192.168.1.219 port=51524
LOG:  connection authenticated: identity="demo" method=md5
(/usr/local/pgsql/data/pg_hba.conf:91)
LOG:  connection authorized: user=demo database=postgres application_name=psql
```

3.1.16. プロセス名

インスタンスを構成するプロセス名に情報が付加されています。

□ WAL Receiver

wal receiver プロセスのプロセス名にステータスが出力されるようになりました。

例 31 WAL receiver プロセス名

```
$ ps -ef|grep receiver | grep -v grep
postgres   93255   93249   0 14:34 ?          00:00:01 postgres: walreceiver
streaming 0/5D744F8
$
```

□ WAL sender

ロジカル・レプリケーション用の WAL sender プロセスのプロセス名にレプリケーションコマンド名が出力されるようになりました。

例 32 WAL sender プロセス名

```
$ ps -ef|grep sender | grep -v grep
postgres  92995  92944  0 14:28 ?        00:00:00 postgres: walsender postgres
[local] START_REPLICATION
$
```

3.1.17. 正規表現

補集合クラスのエスケープ¥D、¥S、¥W が角かっこ式内で許可されるようになりました。また文字クラスとして word が追加されました。これは¥w と同等の機能です。

例 33 word 文字クラス

```
postgres=> SELECT 'abc def' ~ '[[:word:]] [[:word:]]' ;
 ?column?
----------
 t
(1 row)
```

3.1.18. テキスト検索

テキスト検索の対応言語が増えました。PostgreSQL 14 では以下の言語が増えています。

表 22 追加された言語

名前	説明
armenian	アルメニア語
basque	バスク語
catalan	カタロニア語
hindi	ヒンディー語
serbian	セルビア語
yiddish	イディッシュ語

3.1.19. テスト・モジュール

プロシージャ言語のテンプレートとして使用できる PL/Sample モジュールが追加されました。

例 34 PL/Sample モジュール

```
$ cd src/test/modules/plsample
$ ls -l
total 24
drwxrwxr-x. 2 postgres postgres   26 May 18 05:14 expected
-rw-r--r--. 1 postgres postgres  440 May 18 05:11 Makefile
-rw-r--r--. 1 postgres postgres  469 May 18 05:11 plsample--1.0.sql
-rw-r--r--. 1 postgres postgres 5046 May 18 05:11 plsample.c
-rw-r--r--. 1 postgres postgres  177 May 18 05:11 plsample.control
-rw-r--r--. 1 postgres postgres  228 May 18 05:11 README
drwxrwxr-x. 2 postgres postgres   26 May 18 05:14 sql
```

3.1.20. パスワード長

クライアント認証に使うパスワード文字列の最大長制限が撤廃されました。

3.1.21. LLVM

LLVM 12 をサポートするようになりました。

3.2. SQL 文の拡張

ここでは SQL 文に関係する新機能を説明しています。

3.2.1. ALTER CURRENT_ROLE

ALTER AGGREGATE 文、ALTER CONVERSION 文、ALTER DATABASE 文など、CURRENT_USER 句を指定できる文には CURRENT_ROLE 句も指定できるようになりました。

例 35 ALTER DATABASE 文の実行

```
postgres=# ALTER DATABASE demodb OWNER TO CURRENT_ROLE ;
ALTER DATABASE
```

3.2.2. ALTER SUBSCRIPTION

既存のサブスクリプションに対してパブリケーションを追加／削除する構文が追加されました。ALTER SUBSCRIPTION 文に ADD PUBLICATION 句または DROP PUBLICATION 句を指定します。

構文

```
ALTER SUBSCRIPTION subscription_name [ ADD | DROP ] PUBLICATION
        publication_name
```

3.2.3. ALTER TABLE

ALTER TABLE / CREATE TABLE 文には以下の新機能が実装されました。

□ DETACH PARTITION

ALTER TABLE DETACH PARTITION 文に CONCURRENTLY 句が指定できるようになりました。CONCURRENTLY が指定されている場合、パーティション・テーブルにアクセスしている可能性のある他のセッションのブロックを回避するために、ロックレベルを下げて実行されます。このモードでは、2 つのトランザクションが内部で使用されるため、トランザクション・ブロック内では実行できません。またデフォルト・パーティションを含むパーティション・テーブルに対して実行できません。

FINALIZE が指定されている場合、キャンセルまたは中断された前の DETACH CONCURRENTLY 呼び出しが完了します。

構文

```
ALTER TABLE table_name DETACH PARTITION partition_name
 [ FINALIZE | CONCURRENTLY ]
```

□ パーティション・テーブルの自動 VACUUM

これまで自動 VACUUM はパーティション・テーブルを無視してきましたが、リーフ・パーティションの分析結果を上位階層に伝播することで自動 VACUUM にこれらのテーブルを認識させます。これに伴いテーブル・オプションの autovacuum_enabled、autovacuum_analyze_scale_factor、autovacuum_analyze_threshold をパーティション・テーブルに指定できるようになりました。

例 36 パーティション・テーブルに対する自動 VACUUM 設定

```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION
        BY RANGE(c1) ;
CREATE TABLE
postgres=> ALTER TABLE part1 SET (autovacuum_enabled = off) ;
ALTER TABLE
```

3.2.4. COPY FREEZE

COPY FREEZE 文の実行で Visibility Map が更新されるようになりました。

例 37 COPY FREEZE 文の実行後に Visibility Map の確認

```
postgres=# BEGIN ;
BEGIN
postgres=*# TRUNCATE TABLE visible1 ;
TRUNCATE TABLE
postgres=*# COPY visible1 FROM '/home/postgres/visible1.csv' CSV FREEZE ;
COPY 3
postgres=*# COMMIT ;
COMMIT
postgres=# SELECT * FROM pg_visibility_map('visible1', 0) ;
 all_visible | all_frozen
-------------+------------
 t           | t
(1 row)
```

3.2.5. CREATE INDEX

SP-GiST インデックス作成時に INCLUDE 句を使用できるようになりました。

例 38 SP-GiST インデックスの INCLUDE 句

```
postgres=> CREATE INDEX idx1_gist1 ON gist1 USING spgist (c1) INCLUDE (c2) ;
CREATE INDEX
postgres=> ¥d+ idx1_gist1
                  Index "public.idx1_gist1"
 Column | Type | Key? | Definition | Storage  | Stats target
--------+------+------+------------+----------+--------------
 c1     | text | yes  | c1         | extended |
 c2     | text | no   | c2         | extended |
spgist, for table "public.gist1"
```

3.2.6. CREATE PROCEDURE/FUNCTION

CREATE PROCEDURE 文および CREATE FUNCTION 文は以下の拡張が実装されました。

□ OUT 句

PROCEDURE のパラメーターに OUT 句を指定することができるようになりました。

例 39 CREATE PROCEDURE 文の実行

```
postgres=> CREATE PROCEDURE sum_n_product(x int, y int, OUT sum int,
    OUT prod int) AS $$
    BEGIN
         sum := x + y ;
         prod := x * y ;
     END $$ LANGUAGE plpgsql ;
CREATE PROCEDURE
postgres=> SELECT proargmodes FROM pg_proc WHERE proname='sum_n_product' ;
 proargmodes
-------------
 {i,i,o,o}
(1 row)
postgres=> CALL sum_n_product(2, 4, NULL, NULL) ;
 sum | prod
-----+------
   6 |    8
(1 row)
```

□ LANGUAGE SQL

CREATE FUNCTION 文、CREATE PROCEDURE 文に LANGUAGE SQL 句を指定した場合、文字リテラルで囲む必要が無く、SQL 標準に準拠した構文がサポートされます。この構文で作成された FUNCTION / PROCEDURE のソースは pg_proc カタログの prosqlbody 列に格納されます。

例 40 LANGUAGE SQL

```
postgres=> CREATE PROCEDURE insert_data1(id INTEGER, val TEXT)
   LANGUAGE SQL
   BEGIN ATOMIC
      INSERT INTO data1 VALUES (id, val) ;
      INSERT INTO data1 VALUES (id, val) ;
   END ;
CREATE PROCEDURE
```

標準形式で作成された FUNCTION や PROCEDURE は使用されているオブジェクトの依存関係を認識しているため、DROP TABLE CASCADE 文を実行すると依存関係を持つ FUNCTION が一緒に削除されます。

例 41 オブジェクトの削除

```
postgres=> CREATE PROCEDURE insert_data1(id NUMERIC, val TEXT) LANGUAGE SQL
         BEGIN ATOMIC
             INSERT INTO data1 VALUES (id, val);
         END ;
CREATE PROCEDURE
postgres=> DROP TABLE data1 ;
ERROR:  cannot drop table data1 because other objects depend on it
DETAIL:  function insert_data1(numeric,text) depends on table data1
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
postgres=> DROP TABLE data1 CASCADE ;
NOTICE:  drop cascades to function insert_data1(numeric,text)
DROP TABLE
```

3.2.7. CREATE TABLE

パーティション境界に COLLATION が異なる値でも指定できるようになりました。

例 42 パーティション境界に COLLATION を指定

```
postgres=>  CREATE TABLE part_coll (c1 TEXT) PARTITION BY RANGE
        (c1 COLLATE "POSIX") ;
CREATE TABLE
postgres=> CREATE TABLE part_coll_p1 PARTITION OF part_coll FOR VALUES FROM
        ('a' collate "C") to ('g') ;
CREATE TABLE
```

3.2.8. CREATE TRIGGER

CREATE TRIGGER 文に OR REPLACE 句を指定できるようになりました。

例 43 CREATE OR REPLACE TRIGGER 文の実行

```
postgres=> CREATE OR REPLACE TRIGGER data1_trig1 AFTER INSERT
        ON data1 FOR EACH ROW EXECUTE FUNCTION data1_ins_func1 () ;
CREATE TRIGGER
```

3.2.9. CREATE TYPE

CREATE TYPE 文には複数範囲型を指定する MULTIRANGE_TYPE_NAME 句と汎用的な添え字アクセス用関数を指定する SUBSCRIPT 句が追加されました。

構文

```
CREATE TYPE name AS RANGE (
  SUBTYPE = subtype
  [ , MULTIRANGE_TYPE_NAME = multirange_type_name ]
)
```

構文

```
CREATE TYPE name (
  INPUT = input_function,
  OUTPUT = output_function
  [ , SUBSCRIPT = subscript_function ]
)
```

3.2.10. GRANT / REVOKE

SQL 標準に準拠して GRANT 文と REVOKE 文によるロールの付与に GRANTED BY 句を指定できるようになりました。GRANTED BY 句には GRANT 文を実行する現在のロールのみが指定できるため、従来のバージョンと動作は変わりません。

構文

```
GRANT role_name [, …] TO role_specification
    [ WITH ADMIN OPTION ] [ GRANTED BY role_specification ]

REVOKE [ ADMIN OPTION FOR ] role_name [, …] FROM role_specification
    [ GRANTED BY role_specification ] [ CASCADE | RESTRICT ]
```

例 44 GRANTED BY 句の指定

```
postgres=> GRANT SELECT ON data1 TO user1 GRANTED BY CURRENT_USER ;
GRANT
postgres=> GRANT SELECT ON data1 TO user1 GRANTED BY postgres ;
ERROR:  grantor must be current user
```

3.2.11. INSERT

INSERT 文には以下の拡張が実装されました。

□ ON CONFLICT

ON CONFLICT 句で WHERE 句の列指定にテーブル名を修飾できるようになりました。

例 45 ON CONFLICT 句にテーブル指定

```
postgres=> INSERT INTO data1 AS d1 VALUES (10, 'data1') ON CONFLICT (c1)
        WHERE d1.c2 = 'data1' DO NOTHING ;
INSERT 0 1
```

□ GENERATED ALWAYS

GENERATED ALWAYS 句が指定された列を持つテーブルに対する INSERT 文を実行する場合、複数タプルの DEFAULT 句を指定できるようになりました。

例 46 複数行の DEFAULT 句

```
postgres=> CREATE TABLE gen1(c1 INT PRIMARY KEY, c2 INT GENERATED ALWAYS
                AS (c1+1) STORED) ;
CREATE TABLE
postgres=> INSERT INTO gen1 VALUES (100, DEFAULT),(200, DEFAULT) ;
INSERT 0 2
postgres=> SELECT * FROM gen1 ;
 c1  | c2
-----+-----
 100 | 101
 200 | 201
(2 rows)
```

3.2.12. REINDEX

REINDEX 文には以下の拡張が実装されました。

□ パーティション・テーブルの指定

REINDEX 文に対してパーティション・テーブルを指定できるようになりました。

例 47 パーティション・テーブルに対する REINDEX 文の実行

```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY
        RANGE (c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES FROM (0)
        TO (100000) ;
CREATE TABLE
postgres=> CREATE TABLE part1v2 PARTITION OF part1 FOR VALUES FROM (100000)
        TO (200000) ;
CREATE TABLE
postgres=> CREATE INDEX idx1_part1 ON part1(c1) ;
CREATE INDEX
postgres=> REINDEX TABLE part1 ;
REINDEX
```

□ テーブル空間の指定

オプションの TABLESPACE 句にインデックス作成先のテーブル空間を指定できるようになりました。

例 48 テーブル空間の指定

```
postgres=> REINDEX (TABLESPACE ts1, VERBOSE) INDEX idx1_data1 ;
INFO:  index "idx1_data1" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
REINDEX
postgres=> REINDEX (TABLESPACE ts1, VERBOSE) TABLE data1 ;
INFO:  index "data1_pkey" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
INFO:  index "idx1_data1" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
INFO:  index "pg_toast_16385_index" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
REINDEX
```

3.2.13. VACUUM

VACUUM には以下の拡張が実装されました。

□ TOAST テーブル

VACUUM 文には PROCESS_TOAST オプションを指定できるようになりました。このオプションは TOAST テーブルの VACUUM 処理を行います。このパラメーターはデフォルトで有効です。TOAST テーブルの処理を行わない場合は PROCESS_TOAST FALSE オプションを指定します。

例 49 VACUUM (PROCESS_TOAST)文の実行

```
postgres=> VACUUM (PROCESS_TOAST FALSE, VERBOSE) data1 ;
INFO:  vacuuming "public.data1"
INFO:  "data1": found 0 removable, 37 nonremovable row versions in 1 out of
5406 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 518
There were 38 unused item identifiers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
VACUUM
```

□ フェイルセーフ・メカニズム

テーブルの relfrozenxid や relminmxid が危険なほど過去にあると判断すると、VACUUM は動作を変更し、フリーズ以外の処理をバイパスします。これらの閾値としてパラメーターvacuum_failsafe_age と vacuum_multixact_failsafe_age が追加されました。

□ 処理のバイパス

デッド・タプルが存在するページがテーブルの 2%未満の場合、インデックスの VACUUM 処理をスキップします。

例 50 インデックス VACUUM をバイパスする自動 VACUUM のログ

```
LOG:  automatic vacuum of table "postgres.public.data1": index scans: 0
        pages: 0 removed, 5406 remain, 0 skipped due to pins, 0 skipped frozen
        tuples: 10000 removed, 990000 remain, 0 are dead but not yet removable,
oldest xmin: 610
        buffer usage: 10803 hits, 2 misses, 4 dirtied
        index scan not needed: 55 pages from table (1.02% of total) had 10000
dead item identifiers removed
        avg read rate: 0.095 MB/s, avg write rate: 0.191 MB/s
```

3.2.14. SELECT

SELECT 文には以下の拡張が実装されました。

□　予約語の対応

PostgreSQL 14 では SELECT 文に指定する列の別名に予約語を指定する場合でも AS は不要になりました。

例 51 AS 句不要

```
postgres=> SELECT loc analyze FROM dept ;
 analyze
---------
(0 rows)
```

□　GROUP BY DISTINCT

GROUP BY 句に重複を排除する DISTINCT を指定できるようになりました。すべて出力する ALL を指定することもできます（デフォルトの動作は ALL）。

例 52 GROUP BY DISTINCT

```
postgres=> SELECT a, b, c FROM (VALUES (1, 2, 3), (4, NULL, 6), (7, 8, 9))
AS t (a, b, c) GROUP BY ALL ROLLUP(a, b), ROLLUP(a, c) ORDER BY a, b, c ;
a | b | c
---+---+---
 1 | 2 | 3
 1 | 2 |
 1 | 2 |
<<途中省略>>
   |   |
(25 rows)

postgres=> SELECT a, b, c FROM (VALUES (1, 2, 3), (4, NULL, 6), (7, 8, 9))
AS t (a, b, c) GROUP BY DISTINCT ROLLUP(a, b), ROLLUP(a, c) ORDER BY a, b, c ;
 a | b | c
---+---+---
 1 | 2 | 3
 1 | 2 |
<<途中省略>>
   |   |
(13 rows)
```

□ SEARCH 句 / CYCLE 句

再帰 SQL 文の WITH 句に SQL 標準に含まれる SEARCH 句と CYCLE 句を指定できるようになりました。CYCLE 句は閉路検知に使用します。SEARCH 句は深さ優先探索（DEPTH FIRST）、幅優先探索（BREADTH FIRST）を指定できます。

例 53 SEARCH 句（DEPTH FIRST）

```
postgres=> CREATE TABLE graph0(f INT, t INT, label TEXT) ;
CREATE TABLE
postgres=> INSERT INTO graph0 VALUES (1, 2, 'arc 1 -> 2'),
       (1, 3, 'arc 1 -> 3'), (2, 3, 'arc 2 -> 3'),
       (1, 4, 'arc 1 -> 4'), (4, 5, 'arc 4 -> 5') ;
INSERT 0 5
postgres=> WITH RECURSIVE search_graph(f, t, label) AS (
                SELECT * FROM graph0 g
                UNION ALL
                SELECT g.* FROM graph0 g, search_graph sg WHERE g.f = sg.t
           ) SEARCH DEPTH FIRST BY f, t SET seq
           SELECT * FROM search_graph ORDER BY seq ;
 f | t |   label    |        seq
---+---+------------+-------------------
 1 | 2 | arc 1 -> 2 | {"(1,2)"}
 2 | 3 | arc 2 -> 3 | {"(1,2)","(2,3)"}
 1 | 3 | arc 1 -> 3 | {"(1,3)"}
 1 | 4 | arc 1 -> 4 | {"(1,4)"}
 4 | 5 | arc 4 -> 5 | {"(1,4)","(4,5)"}
 2 | 3 | arc 2 -> 3 | {"(2,3)"}
 4 | 5 | arc 4 -> 5 | {"(4,5)"}
(7 rows)
```

例 54 SEARCH 句 (BREADTH FIRST)

```
postgres=> WITH RECURSIVE search_graph(f, t, label) AS (
                SELECT * FROM graph0 g
                UNION ALL
                SELECT g.* FROM graph0 g, search_graph sg WHERE g.f = sg.t
            ) SEARCH BREADTH FIRST BY f, t SET seq
            SELECT * FROM search_graph ORDER BY seq ;
 f | t |   label    |   seq
---+---+------------+---------
 1 | 2 | arc 1 -> 2 | (0,1,2)
 1 | 3 | arc 1 -> 3 | (0,1,3)
 1 | 4 | arc 1 -> 4 | (0,1,4)
 2 | 3 | arc 2 -> 3 | (0,2,3)
 4 | 5 | arc 4 -> 5 | (0,4,5)
 2 | 3 | arc 2 -> 3 | (1,2,3)
 4 | 5 | arc 4 -> 5 | (1,4,5)
(7 rows)
```

例 55 CYCLE 句

```
postgres=> CREATE TABLE graph1(f INT, t INT, label TEXT) ;
CREATE TABLE
postgres=> INSERT INTO graph1 VALUES
        (1, 2, 'arc 1 -> 2'), (1, 3, 'arc 1 -> 3'), (2, 3, 'arc 2 -> 3'),
        (1, 4, 'arc 1 -> 4'), (4, 5, 'arc 4 -> 5'), (5, 1, 'arc 5 -> 1') ;
INSERT 0 6
postgres=> WITH RECURSIVE search_graph(f, t, label) AS (
              SELECT * FROM graph1 g
              UNION ALL
              SELECT g.* FROM graph1 g, search_graph sg WHERE g.f = sg.t
          ) CYCLE f, t SET is_cycle TO TRUE DEFAULT FALSE USING PATH
          SELECT * FROM search_graph ;
 f | t |   label    | is_cycle |                  path
---+---+------------+----------+---------------------------------------
 1 | 2 | arc 1 -> 2 | f        | {"(1,2)"}
 1 | 3 | arc 1 -> 3 | f        | {"(1,3)"}
 2 | 3 | arc 2 -> 3 | f        | {"(2,3)"}
 1 | 4 | arc 1 -> 4 | f        | {"(1,4)"}
<<途中省略>>
 5 | 1 | arc 5 -> 1 | t        | {"(5,1)","(1,4)","(4,5)","(5,1)"}
 2 | 3 | arc 2 -> 3 | f        | {"(1,4)","(4,5)","(5,1)","(1,2)","(2,3)"}
(25 rows)
```

□ USING AS

JOIN USING 句で指定する列に対してエイリアスを指定できるようになりました。エイリアスは SELECT 句内でも JOIN に指定した列に対して利用できます。これは SQL:2016 feature F404 "Range variable for common column names"で定義された機能です。

例 56 JOIN USING AS 句

```
postgres=> SELECT d1.c1, x.c1, x.c2 FROM data1 d1 JOIN data2 d2
        USING (c1, c2) AS x ;
```

3.2.15. TRUNCATE

FOREIGN TABLE に対して FOREIGN DATA WRAPPER がサポートしていれば TRUNCATE 文が実行できるようになりました。postgres_fdw モジュールでは TRUNCATE 文がサポートされます。

例 57 FOREIGN TABLE に対する TRUNCATE 文

```
postgres=> CREATE FOREIGN TABLE remote1(c1 NUMERIC, c2 VARCHAR(10))
        SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=> TRUNCATE TABLE remote1 ;
TRUNCATE TABLE
```

3.2.16. 関数

以下の関数が追加／拡張されました。

□ bit_count

指定されたビット列または bytea 型の中でオンになっているビット数を出力します。

構文

```
bigint BIT_COUNT(bit | bytea)
```

例 58 bit_count 関数の実行

```
postgres=> SELECT bit_count(B'0101011001'::bit(10)) ;
 bit_count
-----------
         5
(1 row)
```

□ bit_xor

ビット列の XOR を計算する集計関数 bit_xor が追加されました。bit_or 関数や bit_and 関数は既に提供されています。

構文

```
integer | bit BIT_XOR(bigint | bit | integer | smallint)
```

例 59 bit_xor 関数の実行

```
postgres=> SELECT SUM(c1), BIT_XOR(c1) FROM data1 ;
    sum     | bit_xor
------------+---------
 5000050000 |  100000
(1 row)
```

□ date_bin

入力されたタイムスタンプを指定された間隔に切り捨てます。date_trunc 関数に似ていますが、任意の間隔に切り捨てることができます。

構文

```
timestamp with time zone DATE_BIN(stride interval, source timestamp with time zone, origin timestamp with time zone)
```

例 60 date_bin 関数の実行

```
postgres=> SELECT date_bin('15 minutes', TIMESTAMP'2020-02-11 15:44:17', TIMESTAMP'2001-01-01 00:02:30') ;
      date_bin
---------------------
 2020-02-11 15:32:30
(1 row)
```

□ make_timestamp / make_timestamptz

年の指定に紀元前を示す負の値を指定できるようになりました。

例 61 紀元前の指定

```
postgres=> SELECT make_timestamp(-2, 12, 25, 10, 20, 30) ;
     make_timestamp
------------------------
 0002-12-25 10:20:30 BC
(1 row)
```

□ split_part

split_part 関数の第 3 パラメーターにマイナス値を指定できるようになりました。マイナス値を指定すると、第 1 パラメーターに指定された文字列の右側から部分を切り取ることができます。

例 62 split_part 関数の実行

```
postgres=> SELECT split_part('www.postgresql.org', '.', -1) ;
split_part
------------
 org
(1 row)
```

□ substring

SQL 標準である SQL:2003 で策定された構文に準拠するようになりました。

構文

```
text SUBSTRING(search_text text SIMILAR pattern text ESCAPE escape text)
```

実行結果は SUBSTRING(text FROM pattern FOR escape)と同じです。

例 63 substring 関数の実行

```
postgres=> SELECT substring('Thomas' SIMILAR '%#"o_a#"_' ESCAPE '#') ;
substring
-----------
 oma
(1 row)
```

□ trim / ltrim / rtrim

ltrim 関数と rtrim 関数は bytea 型にも使えるようになりました。trim 関数は bytea 型に対して LEADING 句及び TRAILING 句を指定できるようになりました。

例 64 rtrim 関数の実行

```
postgres=> SELECT rtrim('abc'::bytea, 'c'::bytea) ;
 rtrim
--------
 ¥x6162
(1 row)

postgres=> ¥df rtrim
                          List of functions
   Schema   | Name  | Result data type | Argument data types | Type
------------+-------+------------------+---------------------+------
 pg_catalog | rtrim | bytea            | bytea, bytea        | func
 pg_catalog | rtrim | text             | text                | func
 pg_catalog | rtrim | text             | text, text          | func
(3 rows)
```

□ trim_array

配列の末尾を削除する関数 trim_array が追加されました。

構文

```
anyarray TRIM_ARRAY(array anyarray, n integer)
```

例 65 trim_array 関数の実行

```
postgres=> SELECT trim_array(ARRAY[1, 2, 3, 4, 5], 2) ;
trim_array
------------
 {1,2,3}
(1 row)
```

□ unistr

unistr 関数は指定された Unicode エスケープされた文字列を評価します。

構文

```
text UNISTR(text)
```

例 66 unistr 関数の実行

```
postgres=> SELECT unistr('¥0441¥+00043B¥u043E¥043D') ;
 unistr
--------
 с л о н
(1 row)
```

□ string_to_table

文字列を指定されたセパレータで区切り、テーブルに変換します。

構文

```
setoff STRING_TO_TABLE(string text, delimiter text [, null_string text])
```

例 67 string_to_table 関数の実行

```
postgres=> SELECT string_to_table('ABC-+-DEF-+-GHI', '-+-', 'abc') ;
 string_to_table
-----------------
 ABC
 DEF
 GHI
(3 rows)
```

□ jsonb 型の ISO 8601 形式日付

ISO 8601 形式の日付フォーマットをサポートします。この機能は PostgreSQL 13.1 以降にバックポートされました。

例 68 ISO 8601 形式の日付フォーマット

```
postgres => SELECT
         jsonb_path_query('"2017-03-10T12:34:56+3:10"', '$.datetime()') ;
       jsonb_path_query
-----------------------------
 "2017-03-10T12:34:56+03:10"
(1 row)
```

□ pg_get_wal_replay_pause_state

リカバリの一時停止状態を返します。一時停止が要求されていない場合は「not paused」、一時停止が要求されたがリカバリが停止されていない場合は「pause requested」、リカバリが停止されている場合は「paused」を返します。PostgreSQL 14 では必要なパラメーター値が不足している場合、リカバリーが一時停止するようになりました。

構文

```
text PG_GET_WAL_REPLAY_PAUSE_STATE()
```

例 69 pg_get_wal_replay_pause_state 関数の実行

```
postgres=# SELECT pg_wal_replay_pause() ;
 pg_wal_replay_pause
---------------------


(1 row)

postgres=# SELECT pg_get_wal_replay_pause_state() ;
 pg_get_wal_replay_pause_state
-------------------------------
 paused
(1 row)
```

□ pg_log_backend_memory_contexts

指定されたIDのプロセスのメモリー状態をLOGレベルのメッセージとして出力します。

構文

```
bool PG_LOG_BACKEND_MEMORY_CONTEXTS(pid integer)
```

例 70 pg_log_backend_memory_contexts 関数の実行

```
postgres=# SELECT pg_log_backend_memory_contexts(pg_backend_pid()) ;
[67300] LOG:  logging memory contexts of PID 67300
[67300] STATEMENT:  SELECT pg_log_backend_memory_contexts(pg_backend_pid());
[67300] LOG:  level: 0; TopMemoryContext: 68704 total in 5 blocks; 14416 free
(10 chunks); 54288 used
<<以下省略>>
```

□ pg_terminate_backend / pg_wait_for_backend_termination

pg_terminate_backend 関数にはタイムアウトを指定するパラメーターが追加されました。pg_wait_for_backend_termination 関数はバックエンド・プロセスの停止をリクエストせずにセッションの終了をタイムアウト（デフォルト 5 秒）まで待機します。タイムアウト以内で終了した場合は true を返します。

構文

```
boolean PG_TERMINATE_BACKEND(pid integer, timeout bigint DEFAULT 5000)
boolean PG_WAIT_FOR_BACKEND_TERMINATION(pid integer, timeout bigint DEFAULT 5000)
```

□ pg_xact_commit_timestamp_origin

この関数はトランザクション ID に対するタイムスタンプとレプリケーション・オリジンを出力します。この関数の実行には track_commit_timestamp に on を指定する必要があります。

構文

```
record PG_XACT_COMMITTED_TIMESTAMP_ORIGIN(xid xid, OUT timestamp timestamp with time zone, OUT roident oid)
```

例 71 pg_xact_commit_timestamp_origin 関数の実行

```
postgres=# SHOW track_commit_timestamp ;
 track_commit_timestamp
------------------------
 on
(1 row)

postgres=# SELECT * FROM
         pg_xact_commit_timestamp_origin(txid_current()::text::xid) ;
 timestamp | roident
-----------+---------
           |
(1 row)
```

□ pg_last_committed_xact

実行結果にレプリケーション・オリジンを示す roident 列が出力されるようになりました。

構文

```
record PG_LAST_COMMITTED_XACT()
```

例 72 pg_last_committed_xact 関数の実行

```
postgres=# SELECT * FROM pg_last_committed_xact() ;
 xid |           timestamp           | roident
-----+-------------------------------+--------
 515 | 2021-05-21 11:08:58.549866+09 |       0
(1 row)
```

□ pg_get_catalog_foreign_keys

システムカタログ内で定義された外部キーの情報を取得する pg_get_catalog_foreign_keys 関数が追加されました。この関数は一般ユーザーでも実行できます。

□ pg_create_logical_replication_slot

２フェーズ・コミットをサポートするパラメーターtwophase が追加されました。

例 73 pg_create_logical_replication_slot 関数の定義

```
postgres=# ¥dfS pg_create_logical_replication_slot
List of functions
-[ RECORD 1 ]-------+-------------------------------------------------------------
Schema              | pg_catalog
Name                | pg_create_logical_replication_slot
Result data type    | record
Argument data types | slot_name name, plugin name, temporary boolean DEFAULT
false, twophase boolean DEFAULT false, OUT slot_name name, OUT lsn pg_lsn
Type                | func
```

3.3. パラメーターの変更

PostgreSQL 14 では以下のパラメーターが変更されました。

3.3.1. 追加されたパラメーター

以下のパラメーターが追加されました。

表 23 追加されたパラメーター

パラメーター	説明（context）	デフォルト値
client_connection_check_interval	クエリー実行中のクライアント接続の定期的な検証間隔（user）	0
compute_query_id	クエリーID を計算する（superuser）	auto
debug_invalidate_system_caches_always	キャッシュクローバーの動作を実行時に制御する開発者用オプション（superuser）	0
default_toast_compression	TOAST 列の圧縮方法を指定（user）	pglz
enable_async_append	非同期 Append 処理を有効にする（user）	on
enable_resultcache	実行計画の検討に Result Cache を有効化（user）	on
huge_page_size	Huge Pages のページサイズを表示（postmaster）	0
idle_session_timeout	アイドル状態のセッションを切断するためのタイムアウト時間（user）	0
in_hot_standby	ホット・スタンバイ状態かを示す参照専用パラメーター（internal）	off
log_recovery_conflict_waits	リカバリ中に発生した衝突の情報をログに出力するか（sighup）	off
min_dynamic_shared_memory	動的共有メモリーの初期確保量を設定する（postmaster）	0
recovery_init_sync_method	クラッシュ・リカバリの開始時にストレージの同期を行う設定（postmaster）	fsync
remove_temp_files_after_crash	バックエンド・プロセスが異常終了した際に一時ファイルを削除（sighup）	on
ssl_crl_dir	SSL 証明書失効リストディレクトリのディレクトリ（sighup）	''

パラメーター	説明（context）	デフォルト値
track_wal_io_timing	WAL 書き込み時間をトラッキングする（superuser）	off
vacuum_failsafe_age	VACUUM がトランザクション ID 周回による失敗を回避する最大経過時間（user）	1600000000
vacuum_multixact_failsafe_age	VACUUM が MultiXact ID 周回による失敗を回避する最大経過時間（user）	1600000000

□ compute_query_id

パラメーター値に on を指定すると、実行されるクエリーに対して一意な ID を計算します。クエリーの ID はデータベース内で同じ意味の SQL に対しては同じ値になります。クエリーID は EXPLAIN VERBOSE 文、pg_stat_activity ビューの queryid 列、log_line_prefix パラメーターの%Q 指定等で表示できます。従来は pg_stat_statements モジュールで利用されていた機能を取り込んだものです。デフォルト値は auto で、pg_stat_statements 等のモジュールがこのパラメーターを必要とする場合には自動的に有効になります。

例 74 クエリーID の表示

```
postgres=> EXPLAIN VERBOSE SELECT * FROM data1 ;
                              QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on public.data1  (cost=0.00..15406.00 rows=1000000 width=12)
   Output: c1, c2
 Query Identifier: 3365166609774651210
(3 rows)
```

□ idle_session_timeout

アイドル状態のセッションを強制的に終了させるタイムアウト値をミリ秒単位で指定します。デフォルト値は 0 で、タイムアウトは発生しません。トランザクション実行中は idle_in_transaction_session_timeout が使用されるため、idle_session_timeout は無視されます。

例 75 アイドル状態のタイムアウト

```
postgres=> ¥set VERBOSITY verbose
postgres=> SET idle_session_timeout = 1000 ;
SET
postgres=> -- １秒待つ
postgres=> SELECT 1 ;
FATAL:  57P05: terminating connection due to idle-session timeout
LOCATION:  ProcessInterrupts, postgres.c:3356
server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
The connection to the server was lost. Attempting reset: Succeeded.
postgres=>
```

□ log_recovery_conflict_waits

スタンバイ・インスタンスで log_recovery_conflict_waits パラメーターを有効にした環境で、競合の発生が deadlock_timeout を超えるとログに競合情報が出力されます。また競合が解消した場合もログに出力されます。以下は出力された競合情報です。

例 76 競合状態のログ

```
-- 競合発生時
LOG:  recovery still waiting after 1031.776 ms: recovery conflict on snapshot
DETAIL:  Conflicting process: 53647.
 [49948] CONTEXT:  WAL redo at 1/8ECE0138 for Heap2/PRUNE: latestRemovedXid 895
nredirected 0 ndead 167; blkref #0: rel 1663/14892/17264, blk 108

-- 競合解消時
LOG:  recovery finished waiting after 26073.257 ms: recovery conflict on snapshot
CONTEXT:  WAL redo at 1/8ECE0138 for Heap2/PRUNE: latestRemovedXid 895 nredirected
0 ndead 167; blkref #0: rel 1663/14892/17264, blk 108
```

□ remove_temp_files_after_crash

バックエンド・プロセスが異常終了した場合に、プロセスが作成していた一時ファイルを削除するかを決定します。従来、これらのファイルはインスタンス再起動まで削除されませんでした。デフォルト値は on で、バックエンド・プロセスが異常終了した場合には一時ファイルは削除されます。

3.3.2. 変更されたパラメーター

以下のパラメーターは設定範囲や選択肢が変更されました。

表 24 変更されたパラメーター

パラメーター	変更内容
password_encryption	設定値 on / true / 1 は削除されました。
log_line_prefix	パラレル・グループ・リーダーのプロセス ID を示す%P が追加されました。
	クエリーID を示す%Q が追加されました。
unix_socket_directories	@から始まる抽象名前空間を指定できるようになりました。
restore_command	パラメーター・ファイルのリロードによる動的な変更が可能になりました。

3.3.3. 削除されたパラメーター

以下のパラメーターは削除されました。vacuum_cleanup_index_scale_factor は PostgreSQL 13.3 以降は無効になっています。

表 25 削除されたパラメーター

パラメーター	削除理由
operator_precedence_warning	PostgreSQL 9.4 以前がサポートされなくなったため不要と判断されました。
vacuum_cleanup_index_scale_factor	互換性維持のためインデックスに指定できますが効果はありません。

3.3.4. デフォルト値が変更されたパラメーター

以下のパラメーターはデフォルト値が変更されました。

表 26 デフォルト値が変更されたパラメーター

パラメーター	PostgreSQL 13	PostgreSQL 14	備考
checkpoint_completion_target	0.5	0.9	
password_encryption	md5	scram-sha-256	
server_version	13.3	14.0beta1	
server_version_num	130003	140000	
vacuum_cost_page_miss	10	2	

3.4. ユーティリティの変更

ユーティリティ・コマンドの主な機能拡張点を説明します。

3.4.1. configure

configure コマンドは以下のオプションが追加されました。

□ SSL ライブラリの指定

SSL ライブラリを指定する「--with-ssl={ライブラリ名}」オプションが追加されました。従来の「--with-openssl」オプションも互換性維持のため残されています。

例 77 configure コマンド

```
$ ./configure --help | grep ssl
  --with-ssl=LIB          use LIB for SSL/TLS support (openssl)
  --with-openssl          obsolete spelling of --with-ssl=openssl
$
```

□ 列圧縮

LZ4 列圧縮機能を利用するために--with-lz4 オプションが追加されました。環境変数にコンパイラ・フラグ LZ4_CFLAGS とリンカー・フラグ LZ4_LIBS が追加されました。

例 78 configure コマンド

```
$ ./configure --help | grep -i lz4
  --with-lz4              build with LZ4 support
  LZ4_CFLAGS  C compiler flags for LZ4, overriding pkg-config
  LZ4_LIBS    linker flags for LZ4, overriding pkg-config
$
```

3.4.2. initdb

initdb コマンドには以下のオプションが追加されました。

□ --no-instructions オプション

このオプションを指定すると、インスタンス起動方法を示すメッセージが出力されなくなります。

例 79 --no-instructions オプション

```
$ initdb -D data --no-instructions
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

<<途中省略>>

initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.
$
```

□ --data-checksums オプション

--help オプション指定時に出力されるメッセージで、チェックサムを指定する--data-checksums オプションは「Less commonly used options:」から通常使われるオプション「Options:」に昇格しました。

例 80 チェックサム指定オプション

```
$ initdb --help
initdb initializes a PostgreSQL database cluster.

Usage:
  initdb [OPTION]... [DATADIR]

Options:
  -A, --auth=METHOD         default authentication method for local connections
      --auth-host=METHOD      default authentication method for local TCP/IP
connections
      --auth-local=METHOD     default authentication method for local-socket
connections
 [-D, --pgdata=]DATADIR     location for this database cluster
  -E, --encoding=ENCODING   set default encoding for new databases
  -g, --allow-group-access  allow group read/execute on data directory
  -k, --data-checksums      use data page checksums
      --locale=LOCALE       set default locale for new databases
<<以下省略>>
```

3.4.3. pg_amcheck

pg_amcheck コマンドが追加されました。このコマンドは Contrib モジュール amcheck がインストールされたデータベースに対してテーブルやインデックスの構造を簡単にチェックすることができます。下記の例では data2 テーブルが一部破損していることがわかります。

例 81 pg_amcheck コマンド

```
$ pg_amcheck --database=postgres --table=data2 --verbose
pg_amcheck: including database: "postgres"
pg_amcheck: in database "postgres": using amcheck version "1.3" in schema "public"
pg_amcheck: checking heap table "postgres"."public"."data2"
heap table "postgres"."public"."data2", block 0, offset 17:
    xmin 3236212 equals or exceeds next valid transaction ID 0:546
heap table "postgres"."public"."data2", block 0, offset 48:
    line pointer to page offset 6282 is not maximally aligned
pg_amcheck: checking btree index "postgres"."pg_toast"."pg_toast_16399_index"
pg_amcheck: checking heap table "postgres"."pg_toast"."pg_toast_16399"
```

pg_amcheck コマンドには上記の例以外に多くのオプションを指定することができます。

表 27 主なオプション

オプション	説明
--all	すべてのデータベースをチェックする。
--index=PATTERN	指定されたインデックスをチェックする。
--schema=PATTERN	指定されたスキーマをチェックする。
--on-error-stop	破損が発見されたらコマンドを終了する。
--parent-check	インデックスの親子関係をチェックする。
--startblock=BLOCK	指定されたブロック番号からチェックを開始する。
--endblock=BLOCK	指定されたブロック番号までチェックする。
--jobs=NUM	並列ジョブ数を指定する。
--progress	実行状況を出力する。

3.4.4. pg_dump

pg_dump コマンドには以下の拡張機能が実装されました。

□ パーティションの復元

パーティション・テーブルから単一のパーティションをテーブルとして復元できるようになりました。

□ エクステンション設定のダンプ

エクステンションのみをダンプする--extension オプションが追加されました。エクステンション名のパターンを指定します。

例 82 --extension オプション

```
$ pg_dump --extension=cube
--
-- PostgreSQL database dump
<< 途中省略>>
--
-- Name: cube; Type: EXTENSION; Schema: -; Owner: -
--

CREATE EXTENSION IF NOT EXISTS cube WITH SCHEMA public;
<<以下省略>>
```

□ 複数の--verbose オプション（-v）

pg_dump コマンド、pg_dumpall コマンド、pg_restore コマンド、pg_rewind コマンドでは、-v オプションを複数回指定するとログの出力レベルが上昇します。

3.4.5. pg_rewind

接続先にストリーミング・レプリケーションのスタンバイ・インスタンスを指定できるようになりました。

3.4.6. psql

psql コマンドには以下の機能が追加されました。

□ アクセス・メソッドの表示

¥di+、¥dm+、¥dt+、¥dtS+コマンドの出力にアクセスメソッド（Access Method）が出力されるようになりました。

例 83 アクセス・メソッドの表示追加

```
postgres=> ¥di+
                                    List of relations
 Schema |    Name    | Type  | Owner | Table | Persistence | Access Method | …
--------+------------+-------+-------+-------+-------------+---------------+ …
 public | idx1_data1 | index | demo  | data1 | permanent   | btree         | …
 public | idx1_data2 | index | demo  | data2 | permanent   | btree         | …
(2 rows)
postgres=> ¥dt+
                                       List of relations
 Schema |  Name   |       Type        | Owner | Persistence | Access Method | …
--------+---------+-------------------+-------+-------------+---------------+ …-
 public | data1   | table             | demo  | permanent   | heap          | …
 public | data2   | table             | demo  | permanent   | heap          | …
(2 rows)
```

□ ¥dt, ¥di コマンド

TOAST テーブル、TOAST インデックスに対して指定できるようになりました。

例 84 TOAST テーブルの指定

```
postgres=> ¥dt pg_toast.pg_toast_16384
                List of relations
  Schema   |      Name      |    Type     |  Owner
-----------+----------------+-------------+----------
 pg_toast | pg_toast_16384 | TOAST table | postgres
(1 row)

postgres=> ¥di pg_toast.pg_toast_1213_index
                        List of relations
  Schema   |        Name          | Type  |  Owner   |     Table
-----------+----------------------+-------+----------+----------------
 pg_toast | pg_toast_1213_index | index | postgres | pg_toast_1213
(1 row)
```

□ ¥dX コマンド

拡張統計情報の一覧を出力します。

例 85 拡張統計の一覧

```
postgres=> CREATE STATISTICS stat1_data1 ON c1, c2 FROM data1 ;
CREATE STATISTICS
postgres=> ANALYZE data1 ;
ANALYZE
postgres=> ¥dX
                          List of extended statistics
 Schema |    Name     |    Definition     | Ndistinct | Dependencies |   MCV
--------+-------------+-------------------+-----------+--------------+---------
 public | stat1_data1 | c1, c2 FROM data1 | defined   | defined      | defined
(1 row)
```

□ ¥d+コマンド

¥d+コマンドにテーブル名を指定すると Storage 列と Stats target 列の間に圧縮設定列（Compression）が出力されます。この列は HIDE_TOAST_COMPRESSION 変数を on に設定することで抑制されます。

例 86 ¥d+コマンド

```
postgres=> ¥d+ compress1
                                        Table "public.compress1"
 Column |  Type   | Collation | Nullable | Default | Storage  | Compression | …
--------+---------+-----------+----------+---------+----------+-------------+ …
 c1     | integer |           |          |         | plain    |             | …
 c2     | text    |           |          |         | extended | lz4         | …
Access method: heap

postgres=> ¥set HIDE_TOAST_COMPRESSION on
postgres=> ¥d+ compress1
                                      Table "public.compress1"
 Column |  Type   | Collation | Nullable | Default | Storage  | Stats target | …
--------+---------+-----------+----------+---------+----------+--------------+ …
 c1     | integer |           |          |         | plain    |              | …
 c2     | text    |           |          |         | extended |              | …
Access method: heap
```

□ ¥df, ¥do コマンド

これらのコマンドにはデータ型を追加指定できるようになりました。

例 87 ¥df, ¥do コマンド

```
postgres=> ¥do - pg_catalog.int4
                                List of operators
   Schema   | Name | Left arg type | Right arg type | Result type | Description
------------+------+---------------+----------------+-------------+-----------
 pg_catalog | -    |               | integer        | integer     | negate
(1 row)

postgres=> ¥df sum (numeric)
                           List of functions
   Schema   | Name | Result data type | Argument data types | Type
------------+------+------------------+---------------------+------
 pg_catalog | sum  | numeric          | numeric             | agg
(1 row)
```

□ エディタ終了時の動作

¥e コマンド（¥ef、¥ev コマンドも）でエディタを起動し、ファイルを変更しなかった場合はファイル内の SQL 文を再実行しなくなりました。

3.4.7. reindexdb

reindexdb コマンドには--tablespace オプションが追加されました。再作成するインデックスを保存するテーブル空間（TABLESPACE）を指定できるようになりました。

例 88 REINDEXDB --tablespace

```
$ reindexdb --table=data1 --tablespace=ts1 --verbose postgres
INFO:  index "idx1_data1" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
INFO:  index "pg_toast_16385_index" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
```

3.4.8. vacuumdb

vacuumdb コマンドには以下のオプションが追加されました。

□ --no-index-cleanup オプション

インデックスの不要タプルを削除しません。

□ --no-truncate オプション

テーブル終端の空きページを切り詰めません。

□ --no-process-toast オプション

TOAST テーブルの VACUUM 処理を実行しません。

例 89 vacuumdb コマンドのオプション

```
$ vacuumdb --help
vacuumdb cleans and analyzes a PostgreSQL database.

Usage:
  vacuumdb [OPTION]... [DBNAME]

Options:
  -a, --all                       vacuum all databases
  -d, --dbname=DBNAME             database to vacuum
      --disable-page-skipping     disable all page-skipping behavior
  -e, --echo                      show the commands being sent to the server
  -f, --full                      do full vacuuming
  -F, --freeze                    freeze row transaction information
  -j, --jobs=NUM                  use this many concurrent connections to vacuum
      --min-mxid-age=MXID_AGE     minimum multixact ID age of tables to vacuum
      --min-xid-age=XID_AGE       minimum transaction ID age of tables to vacuum
      --no-index-cleanup          don't remove index entries that point to dead tuples
      --no-process-toast          skip the TOAST table associated with the table to vacuum
      --no-truncate               don't truncate empty pages at the end of the table
  -P, --parallel=PARALLEL_DEGREE  use this many background workers for vacuum, if available
<<以下省略>>
```

3.5. Contrib モジュール

Contrib モジュールに関する新機能を説明しています。

3.5.1. amcheck

テーブルの構造をチェックする verify_heapam 関数が追加されました。

例 90 verify_heapam 関数の実行

```
postgres=# CREATE EXTENSION amcheck ;
CREATE EXTENSION
postgres=# SELECT verify_heapam('data1') ;
 verify_heapam
---------------
(0 rows)
```

3.5.2. bool_plperl

新規の Contrib モジュール bool_plperl と bool_plperlu が追加されました。これらは PL/Perl モジュールで文字列'f'を false とみなすための TRANSFORM モジュールです。bool_plperl は Trusted モジュールのため、一般ユーザーでも CREATE EXTENSION 文で実行できます。

例 91 bool_plperlu モジュールの使用

```
postgres=> CREATE EXTENSION bool_plperl ;
CREATE EXTENSION
postgres=> CREATE FUNCTION perl2text(text) RETURNS bool
                LANGUAGE plperl
                TRANSFORM FOR TYPE bool
                AS $$
                   return shift ;
                $$ ;
CREATE FUNCTION
postgres=> SELECT perl2text('f') ;
 perl2text
-----------
 t
(1 row)
```

3.5.3. btree_gist

btree_gist モジュールが提供するすべての関数はパラレル・セーフになりました。

3.5.4. cube

cube 型はバイナリ I/O をサポートするようになりました。

例 92 cube 型のバイナリ・アクセス

```
postgres=> CREATE EXTENSION cube ;
CREATE EXTENSION
postgres=> SELECT typname, typreceive, typsend FROM pg_type
        WHERE typname='cube' ;
 typname | typreceive |  typsend
---------+------------+-----------
 cube    | cube_recv  | cube_send
(1 row)
```

3.5.5. hstore

hstore 型の列に対して添え字を使ってアクセスできるようになりました。

例 93 添え字を使ったアクセス

```
postgres=> CREATE EXTENSION hstore ;
CREATE EXTENSION
postgres=> CREATE TABLE hstore1 (id INT, attr hstore) ;
CREATE TABLE
postgres=> INSERT INTO hstore1 VALUES (1, '"attr1"=>"val1", "attr2"=>"val2",
"attr3"=>"val3"') ;
INSERT 0 1
postgres=> SELECT id, attr['attr1'], attr->'attr2' FROM hstore1 ;
 id | attr | ?column?
----+------+----------
  1 | val1 | val2
(1 row)
```

3.5.6. old_snapshot

新規追加された Contrib モジュールです。このモジュールには pg_old_snapshot_time_mapping 関数が定義されています。パラメーター old_snapshot_threshold を-1 以外に設定した場合に XID とタイムスタンプのマッピングを表示することができます。

例 94 pg_old_snapshot_time_mapping 関数の実行

```
postgres=# SHOW old_snapshot_threshold ;
 old_snapshot_threshold
------------------------
 90min
(1 row)

postgres=# CREATE EXTENSION old_snapshot ;
CREATE EXTENSION
postgres=# SELECT * FROM pg_old_snapshot_time_mapping() ;
 array_offset |      end_timestamp       | newest_xmin
--------------+--------------------------+-------------
            0 | 2021-05-21 10:13:00+09 |         533
            1 | 2021-05-21 10:14:00+09 |         534
            2 | 2021-05-21 10:15:00+09 |         535
            3 | 2021-05-21 10:16:00+09 |         535
(4 rows)
```

3.5.7. pageinspect

GiST インデックスに関する以下の関数が追加されました。

表 28 追加された関数

関数名	説明
gist_page_opaque_info	opaque エリア情報を返します。
gist_page_items	ページ内のデータを record 型で返します。
gist_page_items_bytea	ページ内のデータを bytea 型で返します。

例 95 GiST インデックスの情報

```
postgres=# SELECT * FROM gist_page_opaque_info(get_raw_page('idx1_gist1', 1)) ;
    lsn     |    nsn     | rightlink | flags
------------+------------+-----------+--------
 0/2216510 | 0/2216510 |       473 | {leaf}
(1 row)

postgres=# SELECT * FROM gist_page_items(get_raw_page('idx1_gist1', 1),
         'idx1_gist1') ;
 itemoffset |    ctid     | itemlen |     keys
------------+-------------+---------+--------------
          1 | (457,65535) |     144 | (c2)=('' '')
          2 | (249,65535) |     144 | (c2)=('' '')
          3 | (487,65535) |     144 | (c2)=('' '')
<<以下省略>>
```

□ bt_metap 関数の出力

追加情報「last cleanup num delpages」が出力されるようになりました。

例 96 bt_metap 関数の出力

```
postgres=# SELECT * FROM bt_metap('idx1_data1') ;
-[ RECORD 1 ]-------------+-------
magic                     | 340322
version                   | 4
root                      | 290
level                     | 2
fastroot                  | 290
fastlevel                 | 2
last_cleanup_num_delpages | 0
last_cleanup_num_tuples   | -1
allequalimage             | t
```

3.5.8. passwordcheck

Cracklib ライブラリをを使ったチェックの出力に理由が出力されるようになりました。

例 97 パスワードのチェック

```
postgres=# CREATE USER demo PASSWORD 'password123' ;
ERROR:  password is easily cracked
postgres=# ¥! tail -3 data/log/postgresql-2021-05-21_151319.log
ERROR:  password is easily cracked
DETAIL:  cracklib diagnostic: it is based on a dictionary word
STATEMENT:  CREATE USER demo PASSWORD 'password123';
```

3.5.9. pgstattuple

pgstattuple_approx 関数は TOAST テーブルに対しても実行できるようになりました。

例 98 pgstattuple_approx 関数の実行

```
postgres=# SELECT pgstattuple_approx((SELECT reltoastrelid FROM pg_class WHERE
relname='data1')) ;
  pgstattuple_approx
----------------------
 (0,0,0,0,0,0,0,0,0,0)
(1 row)
```

3.5.10. pg_stat_statements

pg_stat_statements モジュールには以下の機能が拡張されました。pg_stat_statements を利用する際には compute_query_id を on または auto に指定する必要があります。

□ pg_stat_statements ビュー

以下の列が追加されました。

表 29 追加された列

列名	データ型	説明
toplevel	boolean	取得された SQL がトップレベルかどうか

□ ユーティリティ・コマンドのトラック

REFRESH MATERIALIZED VIEW 文、CREATE TABLE AS 文、SELECT INTO 文、CREATE MATERIALIZED VIEW 文等についてトラッキングができるようになりました。

例 99 REFRESH MATERIALIZED VIEW のトラッキング

```
postgres=# REFRESH MATERIALIZED VIEW mview1 ;
REFRESH MATERIALIZED VIEW
postgres=#
postgres=# SELECT query, rows FROM pg_stat_statements WHERE query
        LIKE 'REFRESH%';
              query               |  rows
----------------------------------+---------
 REFRESH MATERIALIZED VIEW mview1 | 1000000
(1 row)
```

□ pg_stat_statements_info ビューの追加

pg_stat_statements_info ビューが追加されました。このビューは pg_stat_statements モジュールの稼働状況を確認することができます。現状では pg_stat_statements ビューから削除された SQL 文の個数を示す dealloc 列と統計情報がリセットされた日時を示す stats_reset 列のみが提供されています。dealloc 列は pg_stat_statements.max パラメーターの妥当性を検証するのに役立ちます。

例 100 pg_stat_statements_info ビュー

```
postgres=# ¥d pg_stat_statements_info
       View "public.pg_stat_statements_info"
 Column      |  Type                     | Collation | Nullable | Default
-------------+---------------------------+-----------+----------+---------
 dealloc     | bigint                    |           |          |
 stats_reset | timestamp with time zone  |           |          |
```

3.5.11. pg_trgm

GiST/GIN インデックスで等式演算子がサポートされるようになりました。

例 101 GiST インデックスの等式演算子

```
postgres=> CREATE INDEX trgm_idx ON test_trgm USING gist (t gist_trgm_ops) ;
CREATE INDEX
postgres=> EXPLAIN ANALYZE SELECT * FROM test_trgm WHERE t='100000' ;
                                                     QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Index Scan using trgm_idx on test_trgm  (cost=0.28..8.30 rows=1 width=5)
(actual time=0.424..0.732 rows=1 loops=1)
   Index Cond: (t = '100000'::text)
   Rows Removed by Index Recheck: 2
 Planning Time: 0.038 ms
 Execution Time: 0.744 ms
(5 rows)
```

3.5.12. pg_surgery

新しい Contrib モジュールとして pg_surgery が追加されました。このモジュールには、タプルを強制的にフリーズさせる関数 heap_force_freeze と、強制的に削除する heap_force_kill 関数が提供されています。

例 102 pg_surgery Contrib モジュール

```
postgres=# SELECT xmin, ctid, c1 FROM data1 ;
 xmin | ctid  | c1
------+-------+-----
  517 | (0,1) | 100
(1 row)
postgres=# SELECT heap_force_freeze('data1'::regclass,
                ARRAY['(0, 1)']::tid[]) ;
 heap_force_freeze
-------------------


(1 row)
postgres=# SELECT xmin, ctid, c1 FROM data1 ;
 xmin | ctid  | c1
------+-------+-----
    2 | (0,1) | 100
(1 row)
postgres=# SELECT heap_force_kill('data1'::regclass, ARRAY['(0, 1)']::tid[]) ;
 heap_force_kill
-----------------


(1 row)
postgres=# SELECT xmin, ctid, c1 FROM data1 ;
 xmin | ctid | c1
------+------+----
(0 rows)
```

3.5.13. postgres_fdw

postgres_fdw モジュールには以下の拡張が実装されました。

□ 関数

アクティブなセッションを制御する以下の関数が追加されました。

表 30 追加された関数

関数名	説明
postgres_fdw_disconnect	アクティブな特定のセッションを切断
postgres_fdw_disconnect_all	アクティブな全セッションを切断
postgres_fdw_get_connections	アクティブなセッションを取得

例 103 postgres_fdw 関連関数の実行

```
postgres=> CREATE FOREIGN TABLE remote1(c1 NUMERIC, c2 VARCHAR(10))
        SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=> SELECT COUNT(*) FROM remote1 ;
  count
---------
 1000000
(1 row)

postgres=> SELECT * FROM postgres_fdw_get_connections() ;
 server_name | valid
-------------+-------
 remsvr1     | t
(1 row)

postgres=> SELECT postgres_fdw_disconnect_all() ;
 postgres_fdw_disconnect_all
-----------------------------
 t
(1 row)
```

□ Bulk Insert 対応

Foreign Data Wrapper に Bulk Insert 用 API が追加されたことに対応し、postgres_fdw を利用する FOREIGN SERVER および FOREIGN TABLE のオプションとして batch_size を指定できるようになりました。

例 104 batch_size オプションの指定

```
postgres=> CREATE FOREIGN TABLE remote1(c1 NUMERIC, c2 VARCHAR(10))
        SERVER remsvr1 OPTIONS (batch_size '1000') ;
CREATE FOREIGN TABLE
postgres=> ¥d remote1
                        Foreign table "public.remote1"
 Column |         Type          | Collation | Nullable | Default | FDW options
--------+-----------------------+-----------+----------+---------+-------------
 c1     | numeric               |           |          |         |
 c2     | character varying(10) |           |          |         |
Server: remsvr1
FDW options: (batch_size '1000')
```

Bulk Insert 用に Foreign Data Wrapper に追加されたインターフェースは以下の通りです。

表 31 追加されたインターフェース

関数名	説明
ExecForeignBatchInsert	一括送信処理の実行
GetForeignModifyBatchSize	バッチサイズの取得

□ オプション keep_connections

FOREIGN SERVER のオプション keep_connections はトランザクション完了後のリモート・コネクションを制御します。デフォルト値は on でトランザクション完了後もコネクションを維持します。このオプションを off に設定すると、トランザクション完了時にコネクションをクローズします。

□ IMPORT FOREIGN SCHEMA 文

LIMIT TO 句、EXCEPT 句にパーティションを指定できるようになりました。

□ 再接続

リモート・インスタンスとのセッションが切れていることを確認した場合、再接続されるようになりました。

Hewlett Packard
Enterprise

参考にした URL

本資料の作成には、以下の URL を参考にしました。

- Release Notes
  https://www.postgresql.org/docs/14/release-14.html
- Commitfests
  https://commitfest.postgresql.org/
- PostgreSQL 14 Manual
  https://www.postgresql.org/docs/14/index.html
- Git
  git://git.postgresql.org/git/postgresql.git
- GitHub
  https://github.com/postgres/postgres
- PostgreSQL 14 のアナウンス
  https://www.postgresql.org/about/news/postgresql-14-beta-1-released-2213/
- Postgres Professional
  https://habr.com/ru/company/postgrespro/blog/541252/
- PostgreSQL 14 Open Items
  https://wiki.postgresql.org/wiki/PostgreSQL_14_Open_Items
- Qiita（ぬこ@横浜さん）
  http://qiita.com/nuko_yokohama
- pgsql-hackers Mailing list
  https://www.postgresql.org/list/pgsql-hackers/
- PostgreSQL Developer Information
  https://wiki.postgresql.org/wiki/Development_information
- pgPedia
  https://pgpedia.info/postgresql-versions/postgresql-14.html
- SQL Notes
  https://sql-info.de/postgresql/postgresql-14/articles-about-new-features-in-postgresql-14.html
- Slack - postgresql-jp
  https://postgresql-jp.slack.com/

Hewlett Packard
Enterprise

変更履歴

変更履歴

版	日付	作成者	説明
0.1	2021/04/09	篠田典良	内部レビュー版作成 レビュー担当（敬称略）: 高橋智雄 竹島彰子 （日本ヒューレット・パッカード株式会社）
1.0	2021/05/21	篠田典良	PostgreSQL 14 Beta 1 公開版に合わせて修正完了

以上