Hewlett Packard Enterprise

2024 年 4 月 9 日

PostgreSQL 17 新機能検証結果（Dev / Feature Freezed）

日本ヒューレット・パッカード合同会社
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 17 Beta 1 (17.0) 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 16 (16.3) と PostgreSQL 17 Beta 1 (17.0) 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 16.3 (比較対象) PostgreSQL 17 (17.0) Beta 1 (2024/05/99 21:20:39)
オペレーティング・システム	Red Hat Enterprise Linux 8 Update 5 (x86-64)
Configure オプション	--with-ssl=openssl --with-python --with-lz4 --with-zstd --with-llvm --with-libxml --enable-injection-points

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・パッカード合同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は後日予告なく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
…	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。

Hewlett Packard
Enterprise

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

2. PostgreSQL 17 における変更点概要

PostgreSQL 17 には 200 以上の新機能が追加されました。本章では代表的な新機能と利点の概要について説明します。新機能の詳細は「3. 新機能解説」で説明します。

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。

□ 増分バックアップ

標準機能で増分バックアップがサポートされるようになりました。pg_basebackup コマンドに増分バックアップを行うオプションが追加されました。基準となるバックアップのマニフェストを利用して増加分のみを取得します。基準バックアップと増分バックアップをマージする pg_combinebackup コマンドが追加されました。

□ パーティション機能の新機能

既存パーティションの分割とマージ機能が追加されました。

□ 大規模メモリーへの対応

SLRU キャッシュが複数のバンクに分割され、ロック範囲の削減やキャッシュ検索速度の向上が見込まれます。また SLRU メモリーのサイズを決定する複数のパラメーターが追加されました。

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。

□ ロジカル・レプリケーション・スロットの同期

ロジカル・レプリケーションに使用されるレプリケーション・スロットの情報がストリーミング・レプリケーションのスタンバイサーバーに同期できるようになりました。

□ ストリーミング・レプリケーションの待機

ストリーミング・レプリケーションのスタンバイサーバーに変更情報が送られるまでロジカル・レプリケーションの更新を待つ機能が追加されました。

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。

□ チェックポイントの統計

チェックポイントの統計情報を取得する pg_stat_checkpointer ビューが追加されました。pg_stat_checkpointer ビューの一部の列は pg_stat_bgwriter ビューから移動されています。

□ パラメータ・ファイルの拡張

pg_hba.conf ファイル、pg_ident.conf ファイルに使えるトークン名の最大長が 256 バイトから無制限に変更されました。

□ MAINTAIN 権限

VACUUM 文、ANALYZE 文、REINDEX 文などのメンテナンス処理を実行するための権限 MAINTAIN が追加されました。全ユーザーのオブジェクトに対するメンテナンス操作を許可する pg_maintain 事前定義ロールが追加されました。

2.4. プログラミングに関する新機能

SQL 文に以下の機能が追加されました。

□ JSON 関連

複数の JSON コンストラクターと、多くの JSONPATH メソッドが追加されました。また JSON_EXISTS、JSON_QUERY、JSON_VALUE、JSON_TABLE 関数が追加されました。

□ COPY 文

データ型の変換エラー発生時にも処理を継続するオプションが追加されました。

□ MERGE 文

MERGE 文は更新可能ビューに対応しました。また RETURNING 句や BY SOURCE 句が指定できるようになりました。

2.5. 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。

□ 待機イベント・ビュー

待機イベント名を取得できる pg_wait_events ビューが追加されました。現状では待機イベントの名前と説明のみ出力されます。将来的には待機イベントの累計時間等を取得することが期待されます。

□ アクセスメソッドの拡張

テーブル・アクセスメソッドに多くの機能が追加されました。独自に高度なアクセスメソッドを持つテーブルを作成できます。

2.6. 非互換

PostgreSQL 17 は PostgreSQL 16 から以下の仕様が変更されました。

2.6.1. サポート終了

PostgreSQL 17 では以下のプラットフォームやツール向けのサポート・バージョンが変更されました。[1301c80, 8e278b6, 820b5af, 0b16bb8, cc09e65]

サポートが終了したプラットフォームとツールは以下の通りです。

- Microsoft Visual Studio
- LLVM 3.9～7
- IBM AIX
- adminpack 拡張モジュール

PostgreSQL 17 のビルドに必要なコンポーネントのサポート・バージョンの変化は以下の通りです。

- OpenSSL 1.0.2 以降
- LLVM 10 以降

2.6.2. configure コマンド

以下の configure コマンドのオプションが削除されました。[68a4b58, 1c1eec0]

表 4 削除オプション

オプション	説明
--disable-thread-safety	クライアント・ライブラリのスレッド安全を無効化
--with-CC	コンパイラの指定（2000 年 7 月から非推奨扱い）

2.6.3. pgrowlocks 関数

pgrowlocks 関数の実行結果のうち、modes 列に出力される文字列が変更されました。この変更は旧バージョンにもバックポートされます。[15d5d74]

表 5 modes 列の出力

変更前	変更後	備考
Share	For Share	
Key Share	For Key Share	

例 1 pg_wait_events ビューの検索

```
postgres=*# SELECT * FROM pgrowlocks('data1') ;
 locked_row | locker | multi | xids  |       modes       |  pids
------------+--------+-------+-------+-------------------+--------
 (0,1)      |    747 | f     | {747} | {"For Share"}     | {9555}
 (0,2)      |    747 | f     | {747} | {"For Key Share"} | {9555}
(2 rows)
```

2.6.4. MERGE 文

MERGE 文で DO NOTHING 句を指定する場合でも対象テーブルに対する SELECT 権限が必要になりました。この仕様は PostgreSQL 15 以降にバックポートされます。下記の MERGE 文は過去のバージョンでは成功していました。[4989ce7]

例 2 PostgreSQL 17 の動作

```
postgres=# CREATE TABLE merge1(c1 INT, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=# ¥connect postgres demo
You are now connected to database "postgres" as user "demo".
postgres=> MERGE INTO merge1 USING (SELECT 1) ON true
                WHEN MATCHED THEN DO NOTHING ;
ERROR:  permission denied for table merge1
```

2.6.5. EXPLAIN 文

EXPLAIN 文によるサブクエリーの出力方法が変更されました。[fd0398f]

例 3 PostgreSQL 16 の出力

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1 WHERE
        c1=(SELECT MAX(c1) FROM data1) ;
                                    QUERY PLAN
-------------------------------------------------------------------------------
Index Scan using data1_pkey on data1
   Index Cond: (c1 = $1)
   InitPlan 2 (returns $1)
     ->  Result
           InitPlan 1 (returns $0)
             ->  Limit
                   ->   Index  Only  Scan  Backward  using  data1_pkey  on  data1
data1_1
                         Index Cond: (c1 IS NOT NULL)
(8 rows)
```

例 4 PostgreSQL 17 の出力

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1 WHERE
                c1=(SELECT MAX(c1) FROM data1) ;
                                    QUERY PLAN
------------------------------------------------------------------------------
Index Scan using data1_pkey on data1
   Index Cond: (c1 = (InitPlan 2).col1)
   InitPlan 2
     ->  Result
           InitPlan 1
             ->  Limit
                   ->   Index  Only  Scan  Backward  using  data1_pkey  on  data1
data1_1
(7 rows)
```

2.6.6. その他の関数

pg_walfile_name / pg_walfile_name_offset 関数に非互換があります。従来のバージョンではこれらの関数は LSN がセグメント境界にあるときに前のセグメント番号を返していました。常に LSN の現在のセグメント番号を返すように変更されました。この修正は過去バージョンにも反映されます。[344afc7]

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。[1e68e43, 007693f, 46ebdfe, 78806a9, 13aeaf0, 3ee2f25, b0e96f3, e64c733 ,e83d1b0, 74604a3, 96f0526, bc3c8db, 12915a5, 4f62250, 46a0cd4, c393308, 776621a, ddd5f4f, 030e10f, f696c0c, 012460e, 6ae701b, a11f330, 6d49c8d, 7294396, 667e65a]

表 6 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_wait_events	待機イベントの名前と説明が出力されます
pg_stat_checkpointer	チェックポイントの実行状況が出力されます

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_constraint	conperiod	boolean	WITHOUT OVERLAPS 句が指定されているか
pg_database	dathasloginevt	boolean	login イベントトリガーが定義されているか
pg_replication_slots	failover	boolean	スタンバイサーバーと同期できるスロットか
	synced	boolean	プライマリーサーバーと同期しているか
	invalidation_reason	text	Invalid 状態になった理由
	inactive_since	timestamp with time zone	スロットが非アクティブになった時刻
pg_stat_progress_copy	tuples_skipped	bigint	スキップされたタプル数
pg_stat_progress_vacuum	indexes_total	bigint	VACUUM 対象インデックス数

カタログ／ビュー名	追加列名	データ型	説明
	indexes_processed	bigint	VACUUM 処理済インデックス数
	dead_tuple_bytes	bigint	デッドタプルのバイト数
	max_dead_tuple_bytes	bigint	デッドタプルの最大バイト数
pg_stat_subscription	worker_type	text	ワーカーのタイプ
pg_stats	range_length_histogram	anyarray	範囲型の長さのヒストグラム
	range_empty_frac	real	範囲型の空要素の割合
	range_bounds_histogram	anyarray	範囲型の上限・下限ヒストグラム
pg_subscription	subfailover	boolean	スタンバイと同期可能か

表 8 列が削除された pg_catalog スキーマ内のビュー

カタログ／ビュー名	削除列名	説明
pg_replication_slots	conflicting	conflict_reason 列に移行
pg_stat_bgwriter	buffers_backend	pg_stat_checkpoint に移行
	buffers_backend_fsync	pg_stat_checkpoint に移行
	checkpoints_timed	pg_stat_checkpoint に移行
	checkpoint_req	pg_stat_checkpoint に移行
	checkpoint_write_time	pg_stat_checkpoint に移行
	checkpoint_sync_time	pg_stat_checkpoint に移行
	buffers_checkpoint	pg_stat_checkpoint に移行
pg_stat_progress_vauum	num_dead_tuples	dead_tuple_bytes に変更
	max_dead_tuples	max_dead_tuple_bytes に変更

表 9 列名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	変更前	変更後
pg_database	daticulocale	datlocale
pg_collation	coliculocale	collocate

表 10 列が削除された information_schema スキーマ内のビュー

カタログ／ビュー名	削除列名	説明
element_types	domain_default	標準仕様の不具合のため

表 11 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_attribute	attstattarget 列の NOT NULL 制約が削除され、デフォルトの統計情報を示す値が-1 から NULL に変更されました。
pg_statistic_ext	stxstattarget 列のデータ型が integer から smallint に変更されました。stxstattarget 列の NOT NULL 制約が削除されました。
pg_constraint	NOT NULL 制約の情報が出力されます。
pg_stat_wal	bgwriter プロセスが出力する WAL の情報が追加されます。

□ pg_wait_events ビュー

追加された pg_wait_events ビューについて詳細を以下に記載します。pg_wait_events ビューは登録された待機イベントの名前と説明を提供します。[1e68e43]

表 12 pg_wait_events ビュー

列名	データ型	説明
type	text	待機イベントのタイプ
name	text	待機イベント名
description	text	待機イベントの説明

例 5 pg_wait_events ビューの検索

```
postgres=# SELECT name, description FROM pg_wait_events ;
      name             |      description
-----------------------+-------------------------------------------------
 ArchiverMain          | Waiting in main loop of archiver process
 AutoVacuumMain        | Waiting in main loop of autovacuum launcher process
 BgWriterHibernate     | Waiting in background writer process, hibernating
 BgWriterMain          | Waiting in main loop of background writer process
...
```

□ pg_stat_checkpointer ビュー

追加された pg_stat_checkpointer ビューについて詳細を以下に記載します。[96f0526, 12915a5]

表 13 pg_stat_checkpointer ビュー

列名	データ型	説明
num_timed	bigint	スケジュールされたチェックポイント数
num_requested	bigint	要求されたチェックポイント数
restart_points_timed	bigint	タイムアウトまたは再スケジュールされたリスタートポイント数
restart_points_req	bigint	要求されたリスタートポイント数
restart_points_done	bigint	実行されたリスタートポイント数
write_time	double precision	書き込み時間の合計
sync_time	double precision	同期時間の合計
buffers_written	bigint	書き込まれたバッファ数
stats_reset	timestamp with time zone	リセットされたタイムスタンプ

例 6 pg_stat_checkpointer ビューの検索

```
postgres=# SELECT * FROM pg_stat_checkpointer ;
-[ RECORD 1 ]---+------------------------------
num_timed       | 5
num_requested   | 3
write_time      | 269236
sync_time       | 4
buffers_written | 10895
stats_reset     | 2023-10-30 12:02:27.749023+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新機能が実装されました。[bf279dd]

□　ストリーミング・レプリケーションとの連携

ロジカル・レプリケーションの WAL センダー・プロセスは、ストリーミング・レプリケーションのレプリケーション・スロットが WAL を受信したことを確認してから更新データをプラグインに送信するように設定することができます。プライマリーサーバーのパラメーターstandby_slot_names に転送を確認するストリーミング・レプリケーション・スロット名を指定します。standby_slot_names に指定されたレプリケーション・スロットが非アクティブになるとロジカル・レプリケーションの転送も停止され、プライマリーサーバーのログに以下のメッセージが定期的に出力されます。

例 7 ストリーミング・レプリケーション停止時のログ

```
WARNING:  replication slot "slot1" specified in parameter standby_slot_names does not have active_pid
DETAIL:  Logical replication is waiting on the standby associated with "slot1".
HINT:  Consider starting standby associated with "slot1" or amend parameter standby_slot_names.
```

□　レプリケーション・スロットの属性

pg_create_logical_replication_slot 関数にパラメーターfailover が追加されました。このパラメーターのデフォルト値は false です。このパラメーターが true に設定されているレプリケーション・スロットはストリーミング・レプリケーションのスタンバイ・インスタンスに情報を同期することができます。追加された属性値を保存するために pg_replication_slots カタログに failover 列が追加されています。[c393308]

例 8 failover を指定したレプリケーション・スロットの作成

```
postgres=# ¥df 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, failover boolean DEFAULT false, OUT
slot_name name, OUT lsn pg_lsn
Type                | func
postgres=# SELECT pg_create_logical_replication_slot
               ('logislot1', 'test_decoding', false, false, true) ;
-[ RECORD 1 ]----------------------+---------------------
pg_create_logical_replication_slot | (logislot1,0/70004C0)
```

□ サブスクリプションの属性

サブスクリプションの属性 FAILOVER を指定できるようになりました。
この属性を保存するために pg_subscription カタログに subfailover 列が追加されました。
この属性を指定した場合、プライマリーサーバーで作成されるレプリケーション・スロットにも FAILOVER 属性が同時に付与されます。[776621a]

例 9 SUBSCRIPTION の作成

```
postgres=# CREATE SUBSCRIPTION sub1 CONNECTION 'host=dbsvr1 port=5432
        dbname=postgres' PUBLICATION pub1 WITH (FAILOVER = TRUE) ;
NOTICE:  created replication slot "sub1" on publisher
CREATE SUBSCRIPTION
postgres=# SELECT subname, subfailover FROM pg_subscription ;
 subname | subfailover
---------+-------------
 sub1    | t
(1 row)
```

□　ロジカル・レプリケーション・スロットの同期

ストリーミング・レプリケーションのスタンバイ・インスタンスでパラメーターsync_replication_slot を on に設定することで、プライマリーサーバーで作成されたロジカル・レプリケーションの情報を受け取りロジカル・レプリケーション・スロットの情報を定期的に同期することができます。[93db6cb]

この機能を利用するためにはスタンバイ・インスタンスで以下の条件が必要です。

- failover オプションが指定されたレプリケーション・スロット
- パラメーターprimary_slot_name の設定
- パラメーターhot_standby_feedback に on を指定
- パラメーターprimary_conninfo に dbname の設定を含める

上記の条件が満たされる場合「postgres: slotsync worker」プロセスが起動します。条件が満たされない場合、以下のログが出力されます。

例 10 レプリケーション・スロットの同期エラー

```
LOG:  slot synchronization requires hot_standby_feedback to be enabled
LOG:  slot synchronization requires primary_slot_name to be defined
ERROR:  slot synchronization requires dbname to be specified in primary_conninfo
```

レプリケーション・スロットの同期状態はスタンバイサーバーのログに出力されます。以下はスロット同期ワーカーの起動と、新しいレプリケーション・スロットが自動作成されたログです。

例 11 レプリケーション・スロットの同期成功

```
LOG:  slot sync worker started
LOG:  newly created slot "sub1" is sync-ready now
```

□　pg_sync_replication_slot 関数

ストリーミング・レプリケーションのスタンバイサーバーで、ロジカル・レプリケーション・スロットを同期する関数 pg_sync_replication_slot が追加されました。この関数はパラメーターsync_replication_slots が off に設定している環境でもレプリケーション・スロットの情報を強制的に同期します。[ddd5f4f]

例 12 レプリケーション・スロットの強制同期

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


(1 row)
```

□ サブスクライバーでハッシュ・インデックス利用

サブスクライバーで更新処理に BTREE インデックスだけでなく HASH インデックスが利用できるようになりました。[edca342]

□ pg_logical_emit_message 関数

WAL フラッシュを制御するためのパラメーターflush が追加されました。デフォルト値は false でメッセージをフラッシュしません。[173b56f]

構文

```
pg_lsn pg_logical_emit_message(transactional boolean, prefix text, message
        bytea|text, flush boolean DEFAULT false)
```

3.1.3. ストリーミング・レプリケーションの拡張

スタンバイ・インスタンスへ転送するデータを可能であれば WAL バッファから取得するようになりました。従来は書き込みが完了した WAL ファイルから再読み込みを行っていました。[91f2cae]

3.1.4. 差分バックアップ

標準機能で差分バックアップが利用できるようになりました。[174c480,dc21234, ee1bfd1, d9ef650, f8ce4ed]

□ WAL サマライズ

差分バックアップを利用するためには WAL サマライズ機能を有効化する必要があります。WAL サマライズ機能はパラメーターwal_level を replica または logical に設定し、かつパラメーターsummarize_wal（デフォルト値 off）を on に変更することで有効化できま

す。この機能を有効にすると、インスタンス内に walsummarizer プロセスが起動し、${PGDATA}/pg_wal/summaries ディレクトリに WAL ファイルのサマリー・データが格納されます。

例 13 WAL サマリー

```
$ ls data/pg_wal/summaries/
000000010000000001000028000000000010B2D50.summary
00000001000000000010B2D5000000000014E8508.summary
00000001000000000014E850800000000014E8608.summary
00000001000000000014E860800000000014EEF98.summary
00000001000000000014EEF9800000000014EF098.summary
00000001000000000014EF09800000000014EF3D0.summary
00000001000000000014EF3D00000000002000028.summary
000000010000000002000028000000000008000028.summary
…
```

WAL サマリー・ファイルはパラメーターwal_summary_keep_time(デフォルト値 10d)を超えると自動的に削除されます。

WAL サマリーの情報を取得するために以下の関数が提供されています。

表 14 WAL サマリー取得関数

関数名	説明
pg_available_wal_summaries	WAL サマリー・ファイル単位で開始 LSN、終了 LSN を返す
pg_wal_summary_contents	指定された LSN 間の更新情報を返す
pg_get_wal_summarizer_state	WAL サマリー作成の状態を返す

構文

```
record pg_available_wal_summaries()
record pg_wal_summary_contents(tli bigint, start_lsn pg_lsn, end_lsn pg_lsn)
record pg_get_wal_summarizer_state()
```

例 14 WAL サマリー情報を取得する関数

```
postgres=# SELECT * FROM pg_available_wal_summaries() ;
-[ RECORD 1 ]--------
tli       | 1
start_lsn | 0/5A77BB0
end_lsn   | 0/BF29980
…
postgres=# SELECT * FROM pg_wal_summary_contents(1, '0/5A77BB0','0/BF29980') ;
-[ RECORD 1 ]--+-----
relfilenode    | 1259
reltablespace  | 1663
reldatabase    | 1
relforknumber  | 0
relblocknumber | 3
is_limit_block | f
…
postgres=# SELECT * FROM pg_get_wal_summarizer_state() ;
-[ RECORD 1 ]--+----------
summarized_tli | 1
summarized_lsn | 0/BF11AE8
pending_lsn    | 0/BF11BF0
summarizer_pid | 9876
```

□ 増分バックアップ

増分バックアップの取得にはベースバックアップ時に出力されるマニフェストファイルを利用します。pg_basebackup コマンドの--incremental オプション（短縮形 -i）に基準となるベースバックアップのマニフェストファイルを指定して増分バックアップを行います。

例 15 増分バックアップの取得

```
$ pg_basebackup -D back.1
$ psql postgres demo
psql (17devel)
Type "help" for help.
postgres=> INSERT INTO data1 VALUES (generate_series(1, 1000000), 'data1') ;
INSERT 0 1000000
postgres=> ¥q
$ pg_basebackup -D back.inc1 --incremental=back.1/backup_manifest
$
```

ベースバックアップと増分バックアップをマージするために pg_combinebackup コマンドが追加されました。増加分をマージするためには pg_combinebackup コマンドにベースバックアップと差分バックアップが格納された全ディレクトリを指定し、--output オプション（短縮形 -o）にマージ先のディレクトリを指定します。マージ先に指定するディレクトリは存在しないか、空ディレクトリを指定する必要があります。

例 16 差分バックアップのマージ

```
$ pg_combinebackup back.1 back.inc1 --output=data.2
$ ls data.2
backup_label      pg_dynshmem    pg_serial     PG_VERSION
backup_manifest   pg_hba.conf    pg_snapshots  pg_wal
base              pg_ident.conf  pg_stat       pg_xact
current_logfiles  pg_logical     pg_stat_tmp   postgresql.auto.conf
global            pg_multixact   pg_subtrans   postgresql.conf
log               pg_notify      pg_tblspc
pg_commit_ts      pg_replslot    pg_twophase
```

□ マニフェストファイルの変更

マニフェストファイルに System Identifier が出力されるようになりました。これに伴いマニフェストファイルのバージョンが 2 に変更されました。[2041bc4]

例 17 マニフェストファイルの変更

```
$ pg_basebackup -D back.1
$ grep System-Identifier back.1/backup_manifest
"System-Identifier": 7345983777657046843,
$ grep Manifest-Version back.1/backup_manifest
{ "PostgreSQL-Backup-Manifest-Version": 2,
```

pg_combinebackup コマンドは同一のシステム ID（System Identifier）を持つバックアップを指定する必要があります。以下の例では異なるデータベースで取得した差分を指定してエラーが発生しています。

例 18 システム ID のチェック

```
$ pg_combinebackup back.a1 inc.b2 --output=data.1
pg_combinebackup: error: back.a1/global/pg_control: expected system identifier
7347556981648665444, but found 7347557495025884081
```

3.1.5. パーティション

パーティション・テーブルには以下の新機能が実装されました。

□ パーティションのマージ

ALTER TABLE 文に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する MERGE PARTITIONS 句が追加されました。MERGE PARTITIONS 句に既存の複数パーティションを指定し、INTO 句にマージ先として新規作成されるパーティション名を指定します。[1adf16b]

構文

```
ALTER TABLE table_name MERGE PARTITIONS (partition, partition, …)
        INTO new_partition
```

下記の例ではパーティション part1v1 と part1v2 を part1v12 パーティションにマージしています。

例 19 パーティションのマージ

```
postgres=> CREATE TABLE part1(c1 INT PRIMARY KEY, c2 VARCHAR(10))
                PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES
                FROM (0) TO (1000000) ;
CREATE TABLE
postgres=> CREATE TABLE part1v2 PARTITION OF part1 FOR VALUES
                FROM (1000000) TO (2000000) ;
CREATE TABLE
postgres=> INSERT INTO part1 VALUES (generate_series(1,1999999), 'data1') ;
INSERT 0 1999999
postgres=>
postgres=> ALTER TABLE part1 MERGE PARTITIONS (part1v1, part1v2) INTO part1v12 ;
ALTER TABLE
```

データの格納範囲が隣接しない RANGE パーティションはマージできません。

例 20 パーティションのマージ

```
postgres=> CREATE TABLE part2(c1 INT PRIMARY KEY, c2 VARCHAR(10))
                PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part2v1 PARTITION OF part2 FOR VALUES
                FROM (0) TO (1000000) ;
CREATE TABLE
postgres=> CREATE TABLE part2v2 PARTITION OF part2 FOR VALUES
                FROM (1000000) TO (2000000) ;
CREATE TABLE
postgres=> CREATE TABLE part2v3 PARTITION OF part2
                FOR VALUES FROM (2000000) TO (3000000) ;
CREATE TABLE
postgres=> ALTER TABLE part2 MERGE PARTITIONS (part2v1, part2v3) INTO part2v13 ;
ERROR:  lower bound of partition "part2v3" conflicts with upper bound of
previous partition "part2v1"
```

□　一意制約の制限解除

従来はパーティションに対して Btree を使った一意制約のみ許可されていました。PostgreSQL 17 では同じ制限を持つ排他制約（EXCLUDE CONSTRAINT）も許可されるようになりました。ただし列の比較は等しいかどうかを比較する必要があります。下記の３つ目の例ではパーティション・キーに制約対象列が含まれないためエラーになっています。[8c852ba]

例 21 除外制約の許可

```
postgres=> CREATE TABLE partc1 (col1 INT4RANGE,
        EXCLUDE USING GIST (col1 WITH = )) PARTITION BY RANGE (col1) ;
CREATE TABLE
postgres=> CREATE TABLE partc2 (col1 INT4RANGE, col2 INT4RANGE,
  EXCLUDE USING GIST (col1 WITH =, col2 WITH &&)) PARTITION BY RANGE (col1) ;
CREATE TABLE
postgres=> CREATE TABLE partc3 (col1 INT4RANGE, col2 INT4RANGE, col3 INT4RANGE,
EXCLUDE USING GIST (col2 WITH =, col3 WITH &&)) PARTITION BY RANGE (col1) ;
ERROR:  unique constraint on partitioned table must include all partitioning
columns
DETAIL:  EXCLUDE constraint on table "partc3" lacks column "col1" which is part
of the partition key.
```

□　パーティションの分割

既存のパーティションを複数のパーティションに分割できるようになりました。ALTER TABLE 文の SPLIT PARTITION 句に分割元のパーティションを指定し、INTO 句に分割先のパーティションとパーティション分割の値を指定します。[87c21bb]

構文

```
ALTER TABLE table_name SPLIT PARTITION partition INTO
        (PARTITION partition1 VALUES …,
         PARTITION parttiotn2 VALUES …)
```

例 22 パーティションの分割

```
postgres=> CREATE TABLE part3(c1 INT PRIMARY KEY, c2 VARCHAR(10))
                PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part3v1 PARTITION OF part3 FOR VALUES
                FROM (0) TO (2000000) ;
CREATE TABLE
postgres=> INSERT INTO part3 VALUES (generate_series(1,1999999), 'data1') ;
INSERT 0 1999999
postgres=> ALTER TABLE part3 SPLIT PARTITION part3v1 INTO
             (PARTITION part3v2 FOR VALUES FROM (0) TO (1000000),
              PARTITION part3v3 FOR VALUES FROM (1000000) TO (2000000)) ;
ALTER TABLE
```

□ IDENTITY 列のサポート

IDENTITY 列を持つパーティション・テーブルのパーティションに対する INSERT 文がサポートされるようになりました。下記例の２つ目の INSERT 文は PostgreSQL 16 ではエラーになります。[6995863]

例 23 IDENTITY 列のサポート

```
postgres=> CREATE TABLE part1(c1 INT, c2 INT GENERATED ALWAYS AS IDENTITY)
               PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1
               FOR VALUES FROM (0) TO (1000000) ;
CREATE TABLE
postgres=> INSERT INTO part1(c1) VALUES (100) ;
INSERT 0 1
postgres=> INSERT INTO part1v1(c1) VALUES (200) ; ← PostgreSQL 16 でエラー
INSERT 0 1
postgres=> SELECT * FROM part1 ;
 c1  | c2
-----+----
 100 |  1
 200 |  2
(2 rows)
```

例 24 PostgreSQL 16 の動作

```
postgres=> INSERT INTO part1v1(c1) VALUES (200) ;
ERROR:  null value in column "c2" of relation "part1v1" violates not-null
constraint
DETAIL:  Failing row contains (200, null).
```

□ パーティション・プルーニングの拡張

boolean 型列に対する IS UNKNOWN, IS NOT UNKNOWN 句によりパーティション・プルーニングが行われるようになりました。[07c36c1]

3.1.6. ロールと権限

事前定義ロールと付与できる権限が増えました。

□ MAINTAIN 権限

テーブルやマテリアライズド・ビューに対して MAINTAIN 権限を付与できるようになりました。この権限は対象のリレーションに対して VACUUM 文、ANALYZE 文、REINDEX 文、REFRESH MATERIALIZED VIEW 文、CLUSTER 文および LOCK TABLE 文を実行できます。psql コマンドの¥dp メタコマンドの結果には m が出力されます。[ecb0fd3]

例 25 MAINTAIN 権限の付与

```
postgres=# CREATE TABLE data1(c1 INT, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=# GRANT MAINTAIN ON data1 TO demo ;
GRANT
postgres=# ¥dp data1
                                Access privileges
 Schema | Name  | Type  |     Access privileges      | Column privileges | …
--------+-------+-------+----------------------------+-------------------+-…
 public | data1 | table | postgres=arwdDxtm/postgres+|                   |
        |       |       | demo=m/postgres            |                   |
(1 row)

postgres=# ¥connect postgres demo
You are now connected to database "postgres" as user "demo".
postgres=> VACUUM data1 ;
VACUUM
```

□ pg_maintain ロール

pg_maintain 事前定義ロールが追加されました。このロールは全リレーションに対して MAINTAIN 権限を付与できます。

□ pg_monitor ロールの変更

事前定義ロール pg_monitor に pg_current_logfile 関数の実行権限が付与されました。[8d8afd4]

例 26 pg_current_logfile 関数の実行権限

```
postgres=# CREATE USER demo PASSWORD '<<password>>' ;
CREATE ROLE
postgres=# GRANT pg_monitor TO demo ;
GRANT ROLE
postgres=# ¥connect postgres demo
You are now connected to database "postgres" as user "demo".
postgres=> SELECT pg_current_logfile() ;
          pg_current_logfile
---------------------------------------
 log/postgresql-2024-02-15_085557.log
(1 row)

postgres=> SELECT pg_current_logfile('csvlog') ;
 pg_current_logfile
--------------------


(1 row)
```

3.1.7. Built-in ロケール・プロバイダー

Libc や icu のような外部環境に依存しないロケール・プロバイダーbuiltin が追加されました。現状ではこのロケール・プロバイダーが提供するロケールは C および C.UTF8 のみです。ロケール・プロバイダーを指定するコマンドや SQL 文に BUILTIN が指定できるようになっています。[2d819a0, f69319f]

例 27 builtin ロケール・プロバイダーの指定

```
$ initdb --locale-provider=builtin --locale=C.UTF8 data
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with this locale configuration:
  default collation provider:  builtin
  default collation: C.UTF-8
  LC_COLLATE:  C.UTF8
...
```

ロケール C.UTF8 は libc ロケールと比較して、ソートの高速化、大文字／小文字変換の高速化、プラットフォーム非依存などの利点があります。

3.1.8. イベントトリガー

以下のイベントトリガーが追加されました。

□ login イベントトリガー

クライアントからの認証成功時に実行される login イベントトリガーが追加されました。以下は接続時間をテーブルに追記するイベントトリガーの例です。[e83d1b0]

例 28 login イベントトリガーの発行

```
postgres=# CREATE OR REPLACE FUNCTION save_login()
             RETURNS event_trigger SECURITY DEFINER
             LANGUAGE plpgsql AS $$
           BEGIN
             INSERT INTO public.login_history VALUES(current_timestamp);
           END; $$ ;
CREATE FUNCTION
postgres=# CREATE EVENT TRIGGER login_trigger ON login
             EXECUTE FUNCTION save_login() ;
CREATE EVENT TRIGGER
postgres=# ALTER EVENT TRIGGER login_trigger ENABLE ALWAYS ;
ALTER EVENT TRIGGER
postgres=# ¥connect postgres demo
You are now connected to database "postgres" as user "demo".
postgres=> SELECT * FROM login_history ;
           login
----------------------------
 2023-10-16 15:44:21.889142
(1 row)
```

□ REINDEX トリガー

REINDEX 文の実行開始と終了時に DDL イベントトリガーが発行できるようになりました。[f21848d]

例 29 REINDEX イベントトリガーの発行（REINDEX 開始）

```
postgres=# CREATE OR REPLACE FUNCTION reindex_start_func()
           RETURNS event_trigger AS $$
            BEGIN
              RAISE NOTICE 'START REINDEX: % %', tg_event, tg_tag;
            END; $$ LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=# CREATE EVENT TRIGGER trg_reindex_start ON ddl_command_start
             WHEN TAG IN ('REINDEX')
             EXECUTE PROCEDURE reindex_start_func() ;
CREATE EVENT TRIGGER
postgres=# REINDEX INDEX idx1_data1 ;
NOTICE:  START REINDEX: ddl_command_start REINDEX
REINDEX
```

例 30 REINDEX イベントトリガーの発行（REINDEX 終了）

```
postgres=# CREATE FUNCTION reindex_end_func()
           RETURNS event_trigger AS $$
             DECLARE
               obj record ;
             BEGIN
               FOR obj IN SELECT * FROM pg_event_trigger_ddl_commands()
               LOOP
                 RAISE NOTICE 'REINDEX END: command_tag=% type=% identity=%',
                   obj.command_tag, obj.object_type, obj.object_identity;
               END LOOP ;
            END; $$ LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=# CREATE EVENT TRIGGER trg_reindex_end ON ddl_command_end
             WHEN TAG IN ('REINDEX')
             EXECUTE PROCEDURE reindex_end_func() ;
CREATE EVENT TRIGGER
postgres=# REINDEX TABLE data1 ;
NOTICE:  START REINDEX: ddl_command_start REINDEX
NOTICE:  REINDEX END: command_tag=REINDEX type=index identity=public.idx1_data1
NOTICE:  REINDEX END: command_tag=REINDEX type=index identity=public.idx2_data1
REINDEX
```

3.1.9. 設定ファイル

pg_hba.conf ファイル、pg_ident.conf ファイルに記述されるトークンの最大長が 256 バイトから無制限に拡張されました。LDAP で構成された複雑な環境では従来の最大長を超える問題が発生していました。[38df84c]

3.1.10. ログ

稼働ログの出力に以下の変更が加えられました。

□ trust 接続のログ

パラメーターlog_connections を on に設定した場合、trust 接続時にもログが出力されるようになりました。[e48b19c]

例 31 trust 接続時のログ

```
LOG:  connection received: host=[local]
LOG:  connection authenticated: user="postgres" method=trust
        (/postgres/data/pg_hba.conf:117)
LOG:  connection authorized: user=postgres database=postgres
        application_name=psql
```

□ レプリケーション・スロット関連ログ

パラメーターlog_replication_commands が on に設定されている場合に LOG レベル（通常は DEBUG1 レベル）でレプリケーション・スロットの存続に関するログが出力されます。以下の例はレプリケーションの開始と終了時のログです。[7c3fb50]

例 32 レプリケーション・スロット関連ログ

```
LOG:  received replication command: START_REPLICATION SLOT "slot1" 0/3000000
TIMELINE 1
STATEMENT:  START_REPLICATION SLOT "slot1" 0/3000000 TIMELINE 1
LOG:  acquired physical replication slot "slot1"
STATEMENT:  START_REPLICATION SLOT "slot1" 0/3000000 TIMELINE 1
LOG:  released physical replication slot "slot1"
```

□ リカバリ開始終了時

リカバリの開始時、終了時のログが追加されました。以下の例はストリーミング・レプリケーション用インスタンス起動時のログです。[1d35f70]

例 33 リカバリ関連ログ

```
LOG:  starting backup recovery with redo LSN 0/2000028, checkpoint LSN
0/2000080, on timeline ID 1
LOG:  entering standby mode
LOG:  redo starts at 0/2000028
LOG:  completed backup recovery with redo LSN 0/2000028 and end LSN 0/2000120
```

3.1.11. フック

ALTER TABLE 文に以下の Object Access Type (OAT)フックが追加されました。[352ea3a]

- { ENABLE | DISABLE | [NO] FORCE } ROW LEVEL SECURITY
- { ENABLE | DISABLE } TRIGGER
- { ENABLE | DISABLE } RULE

3.1.12. Libpq

以下の関数が追加されました。[28b5726]

□ ポータルとステートメントのクローズ

ポータルとステートメントをクローズする関数が追加されました。名前に「send」を含む関数は非ブロックバージョンです。

表 15 追加された関数名

関数	説明	備考
PQclosePrepared	プリペアド文のクローズを行う	
PQclosePortal	ポータルのクローズを行う	
PQsendClosePrepared	プリペアド文のクローズを行う。完了を待たない	
PQsendClosePortal	ポータルのクローズを行う。完了を待たない	

追加された API

```
PGresult *PQclosePrepared(PGconn *conn, const char *stmt)
PGresult *PQclosePortal(PGconn *conn, const char *portal)
int PQsendClosePrepared(PGconn *conn, const char *stmt)
int PQsendClosePortal(PGconn *conn, const char *portal)
```

□ パスワード変更

接続済ユーザーのパスワードを変更する PQchangePassword 関数が追加されました。[a7be2a6]

追加された API

```
PGresult * PQchangePassword(PGconn *conn, const char *user, const char *passwd)
```

□ パイプラインの同期

PQsendPipelineSync 関数が追加されました。この関数は PQPQpipelineSync 関数とほぼ同じですが、出力バッファがしきい値に達した場合に限りサーバに同期メッセージをフラッシュしません。 [4794c2]

追加された API

```
int PQsendPipelineSync(PGconn *conn)
```

□ シーケンスの操作

relation_open 関数のように、シーケンスの操作を行う関数が追加されました。[449e798]

表 16 追加された関数名

関数	説明	備考
sequence_open	シーケンスのオープン	
sequence_close	シーケンスのクローズ	

追加された API

```
Relation sequence_open(Oid relationId, LOCKMODE lockmode)
void sequence_close(Relation relation, LOCKMODE lockmode)
```

□ インジェクション・ポイント

インジェクション・ポイントをサポートするために以下の関数が追加されました。[d86d20f]

表 17 追加された関数名

関数	説明
INJECTION_POINT	インジェクション・ポイントの宣言
InjectionPointAttach	コールバックのアタッチ
InjectionPointDetach	コールバックのデタッチ

□ 非ブロッキング・キャンセル関数

従来の PQcancel API はブロッキング・リクエストを使っていました。ブロッキングを行わないキャンセル・リクエストを送信できるようになりました。[61461a3]

表 18 追加された関数名

関数	説明
PQcancelCreate	キャンセル・リクエストの送信準備
PQcancelBlocking	現在のコマンドをブロッキング状態で破棄を要求する
PQcancelPoll	現在のコマンドを非ブロッキング状態で破棄を要求する
PQcancelStart	現在のコマンドを非ブロッキング状態で破棄を要求する
PQcancelStatus	キャンセルされた接続のステータスを取得する
PQcancelSocket	キャンセル接続ソケットの識別子を取得する
PQcancelErrorMessage	キャンセル操作により最後に生成されたエラーを取得する
PQcancelFinish	接続をクローズする
PQcancelReset	RGcancelConn をリセットして新しい接続用に再利用する

追加された API

```
PGcancelConn *PQcancelCreate(PGconn *conn)
int PQcancelBlocking(PGcancelConn *cancelConn)
int PQcancelStart(PGcancelConn *cancelConn)
PostgresPollingStatusType PQcancelPoll(PGcancelConn *cancelConn)
ConnStatusType PQcancelStatus(const PGcancelConn *cancelConn)
int PQcancelSocket(const PGcancelConn *cancelConn)
char *PQcancelErrorMessage(const PGcancelConn *cancelconn)
void PQcancelFinish(PGcancelConn *cancelConn)
void PQcancelReset(PGcancelConn *cancelConn)
```

□ Background Worker

BackgroundWorkerInitializeConnection、BackgroundWorkerInitializeConnectionBy-Oid 関数にフラグ BGWORKER_BYPASS_ROLELOGINCHECK が指定できるようになりました。このフラグはロールのログインチェックをバイパスすることができます。[e768919]

□ DSM レジストリ

共有ライブラリが共有メモリーを簡単に利用するための関数が追加されました。従来はパラメーターshared_preload_libraries に指定された共有ライブラリが shmem_request_hook フックを経由して共有メモリーをリクエストする必要がありました。[8b2bcf3]

追加された API

```
void GetNamedDSMSegment(const char *name, size_t size,
        void (*init_callback) (void *ptr), bool *found)
```

□ DSM の初期サイズ

従来は 1 MB で開始され最大で DSA_MAX_SEGMENT_SIZE まだ拡張されました。初期サイズと最大サイズを指定できる dsa_create_ext 関数、dsa_create_in_place_ext 関数が追加されました。[bb952c8]

追加された API

```
dsa_area *dsa_create_ext(int tranche_id, size_t init_segment_size,
                size_t max_segment_size)
dsa_area *dsa_create_in_place_ext(void *place, size_t size,
                int tranche_id, dsm_segment *segment,
                size_t init_segment_size, size_t max_segment_size)
```

□ ストリーミング I/O

リレーションに対するアクセスを抽象化し、リレーションデータをバッファのストリームとしてアクセスできるようにします。この修正により将来非同期 I/O 等の実装が追加された場合にも変更が容易になります。ANALYZE 文の実行、テーブルに対する Sequential Scan の実行で使われています。[b7b0f3f, 041b968, b7b0f3f]

追加された API

```
ReadStream *read_stream_begin_relation(int flags,
    BufferAccessStrategy strategy, Relation rel,
    ForkNumber  forknum,ReadStreamBlockNumberCB callback,
    void *callback_private_data, size_t per_buffer_data_size)
Buffer read_stream_next_buffer(ReadStream *stream, void **per_buffer_private)
void read_stream_reset(ReadStream *stream)
void read_stream_end(ReadStream *stream)
```

□ マルチブロック読み込み

ReadBuffer API のマルチブロック版が提供されました。ブロック数の最大値は新しいパ

ラメーターio_combine_limit で決定されます。将来的には内部的に非同期 I/O を利用する可能性があります。[210622c]

追加された API

```
bool StartReadBuffer(ReadBuffersOperation *operation, Buffer *buffer,
          BlockNumber blocknum, int flags)
bool StartReadBuffers(ReadBuffersOperation *operation, Buffer *buffers,
          BlockNumber blocknum, int *nblocks, int flags)
void WaitReadBuffers(ReadBuffersOperation *operation)
```

□ 短期メモリーの取得

有効期間が短いメモリー取得に適したメモリー・アロケーターが追加されました。このタイプのメモリー取得はタプルのソートや VACUUM 処理の一時メモリーとして使用されます。[29f6a95, 6ed83d5, 8a1b31e]

追加された API

```
MemoryContext BumpContextCreate(MemoryContext parent,
  const char *name, Size minContextSize, Size initBlockSize, Size maxBlockSize)
void *BumpAlloc(MemoryContext context, Size size, int flags)
void BumpFree(void *pointer)
void *BumpRealloc(void *pointer, Size size, int flags)
void BumpReset(MemoryContext context)
void BumpDelete(MemoryContext context)
MemoryContext BumpGetChunkContext(void *pointer)
Size BumpGetChunkSpace(void *pointer)
bool BumpIsEmpty(MemoryContext context)
BumpStats(MemoryContext context, MemoryStatsPrintFunc printfunc,
  void *passthru, MemoryContextCounters *totals, bool print_to_stderr);
void BumpCheck(MemoryContext context)
```

□ ファイル記述子の監視

ファイル記述子の監視を行う PQsocketPoll 関数が提供されました。[f5e4ded]

追加された API

```
int PQsocketPoll(int sock, int forRead, int forWrite, time_t end_time)
```

3.1.13. アクセスメソッド

アクセスメソッドには以下の拡張が追加されました。

□ BRIN インデックスの並列構築

BRIN インデックスの構築に複数のワーカー・プロセスを使用できるようになりました。インデックス・アクセスメソッドの構造体 IndexAmRoutine に amcanbuildparallel フラグが追加されました。[b437571]

□ tuple_insert()

tuple_insert 関数が異なるスロットを返せるようになりました。[c35a3fb]

変更されたコールバック

```
TupleTableSlot *(*tuple_insert) (Relation rel, TupleTableSlot *slot,
    CommandId cid, int options, struct BulkInsertStateData *bistate)
```

□ free_rd_amcache()

複雑なデータ構造を rd_amcache に保存できるようになりました。[02eb07e]

追加されたコールバック

```
void (*free_rd_amcache) (Relation rel);
```

□ tuple_update() / tuple_delete()

tuple_update 関数、tuple_delete 関数内でロックを取得できるようになりました。これにより tuple_lock 関数の呼び出しが不要になります。bool wait パラメーターが、int options に変更されました。[87985cc]

変更された API

```
TM_Result heap_delete(Relation relation, ItemPointer tid,
        CommandId cid, Snapshot crosscheck, int options,
        struct TM_FailureData *tmfd, bool changingPart,
        TupleTableSlot *oldSlot);

TM_Result heap_update(Relation relation, ItemPointer otid,
        HeapTuple newtup,
        CommandId cid, Snapshot crosscheck, int options,
        struct TM_FailureData *tmfd, LockTupleMode *lockmode,
        TU_UpdateIndexes *update_indexes,
        TupleTableSlot *oldSlot);
```

□ tuple_insert() / multi_insert()

以前のエグゼキューターはタプルのインサート後に無条件にインデックスの挿入処理を実行していました。タプルの挿入 API に新しいパラメーターinsert_indexes が追加され、インデックスに挿入処理を行うかを制御できるようになりました。[b1484a3]

変更されたコールバック

```
TupleTableSlot *(*tuple_insert) (Relation rel, TupleTableSlot *slot,
  CommandId cid, int options, struct BulkInsertStateData *bistate,
  bool *insert_indexes)

void        (*multi_insert) (Relation rel, TupleTableSlot **slots, int nslots,
  CommandId cid, int options, struct BulkInsertStateData *bistate,
  bool *insert_indexes)
```

□ heap_page_prune()

heap_page_prune 関数のオプションは現状 HEAP_PAGE_PRUNE_MARK_UNUSED_NOW のみですが、今後増える予定があるので複数オプションのビットマップを使用できるように変更されました。[3d0f730]

変更された API

```
void heap_page_prune(Relation relation, Buffer buffer,
  struct GlobalVisState *vistest, int options,
  PruneResult *presult, PruneReason reason, OffsetNumber *off_loc)
```

□ カスタム・オプション

CREATE TABLE 文の WITH 句で指定できるアクセスメソッド独自のオプションを定義できるようになりました。[9bd99f4]

変更された API

```
bytea *extractRelOptions(HeapTuple tuple, TupleDesc tupdesc,
    const TableAmRoutine *tableam, amoptions_function amoptions,
    CommonRdOptions *common)
bytea *heap_reloptions(char relkind, Datum reloptions, CommonRdOptions *common,
    bool validate)
const TableAmRoutine *GetTableAmRoutineByAmOid(Oid amoid)
```

3.1.14. 待機イベント

以下の待機イベント機能が追加・変更されました。[fa88928, c9af054, c8e318b, 0013ba2, 5c430f9, d86d20f]

表 19 追加された待機イベント

イベント名	タイプ	説明
CheckpointDelayComplete	IPC	チェックポイントを完了するバックエンド待ち
CheckpointDelayStart	IPC	チェックポイントを開始するバックエンド待ち
Extension	Extension	拡張モジュール待ち
ReplicationSlotsyncMain	Activity	スロット同期待ち
ReplicationSlotsyncShutdown	Activity	スロット停止待ち
WaitForStandbyConfirmation	Client	スタンバイ確認待ち
WalSummarizerError	Timeout	WAL サマライズ・エラー
WALSummarizerWal	Activity	WAL サマライズ I/O 待ち

イベント名	タイプ	説明
WalSummaryRead	IO	WAL サマリー読み込み待ち
WalSummaryReady	IPC	WAL サマリー生成待ち
WalSummaryWrite	IO	WAL サマリー書き込み待ち
InjectionPoint	LWLock	インジェクション・ポイント読み込み待ち

表 20 名前が変更された待機イベント

変更前イベント名	変更後イベント名	備考
AutoVacuumMain	AutovacuumMain	
BaseBackupRead	BasebackupRead	
BaseBackupSync	BasebackupSync	
BaseBackupWrite	BasebackupWrite	
BgWorkerShutdown	BgworkerShutdown	
BgWorkerStartup	BgworkerStartup	
BgWriterHibernate	BgwriterHibernate	
BgWriterMain	BgwriterMain	
BufferIO	BufferIo	
BufFileRead	BuffileRead	
BufFileTruncate	BuffileTruncate	
BufFileWrite	BuffileWrite	
DSMAllocate	DsmAllocate	
DSMFillZeroWrite	DsmFillZeroWrite	
GSSOpenServer	GssOpenServer	
LibPQWalReceiverConnect	LibpqwalreceiverConnect	
LibPQWalReceiverReceive	LibpqwalreceiverReceive	
LockFileAddToDataDirRead	LockFileAddtodatadirRead	
LockFileAddToDataDirSync	LockFileAddtodatadirSync	
LockFileAddToDataDirWrite	LockFileAddtodatadirWrite	
LockFileReCheckDataDirRead	LockFileRecheckdatadirRead	
ProcArrayGroupUpdate	ProcarrayGroupUpdate	
SLRUFlushSync	SlruFlushSync	
SLRURead	SlruRead	
SLRUSync	SlruSync	
SLRUWrite	SlruWrite	

変更前イベント名	変更後イベント名	備考
SSLOpenServer	SslOpenServer	
SysLoggerMain	SysloggerMain	
WALBootstrapSync	WalBootstrapSync	
WALBootstrapWrite	WalBootstrapWrite	
WALCopyRead	WalCopyRead	
WALCopySync	WalCopySync	
WALCopyWrite	WalCopyWrite	
WALInitSync	WalInitSync	
WALInitWrite	WalInitWrite	
WALRead	WalRead	
WALSenderTimelineHistoryRead	WalsenderTimelineHistoryRead	
WalSenderWaitForWAL	WalSenderWaitForWal	
WALSync	WalSync	
WALSyncMethodAssign	WALSyncMethodAssign	
WALWrite	WALWrite	

□　新規の待機イベント

待機イベントの一覧は src/backend/utils/activity/wait_event_names.txt ファイルに記述され、コードとドキュメントが自動生成されるようになりました。[59cbf60]

例 34 wait_event_names.txt の一部

```
…
Section: ClassName - WaitEventClient

CLIENT_READ     "Waiting to read data from the client."
CLIENT_WRITE    "Waiting to write data to the client."
…
```

□　カスタム・タイプ

カスタム待機イベントの登録方法が変更されました。新しい API 「uint32 WaitEventExtensionNew(const char *wait_event_name)」が提供されています。これに伴い共有メモリーのフックが不要になりました。[af720b4]

3.1.15. VACUUM

VACUUM 対象のタプル ID を保存するために単純な配列ではなく TidStore を使うようになりました。これにより従来大きめに取得していたメモリー使用量を削減し、1 GB という制限も解消されました。この変更により、pg_stat_progress_vacuum ビューの num_dead_tuples 列の名前は dead_tuple_bytes に変更されました。[667e65a]

3.1.16. FILLFACTOR

テーブルの統計情報が取得されていない場合に推定されるタプル数に FILLFACTOR が考慮されるようになりました。これまでは FILLFACTOR が考慮されていなかったため、実際よりもタプル数が過大評価される可能性がありました。[29cf61a]

3.1.17. LLVM

LLVM 17, 18 をサポートします。サポートされる旧バージョンにバックポートされます。[76200e5, d282e88]

3.1.18. 高速化

LoongArch をサポートする環境では CRC の計算関数 COMP_CRC32C をネイティブ実行できるようになりました。[4d14ccd]

Intel AVX-512 命令セットを使用できる環境では、visibilitymap_count 関数と pg_popcount 関数を高速に実行できるようになりました。[41c51f0, 792752a]

3.1.19. UNICODE

対応する Unicode のバージョンが 15.0.0 から 15.1.0 に変更されました。また UNICODE テーブルを更新する update-unicode ビルドターゲットが追加されました。[9d17e5f, ad49994]

3.1.20. GiST

GiST サポート関数 stratnum がサポート番号 12 として追加されました。これにより一時的な PRIMARY KEY/UNIQUE/FOREIGN KEY/FOR PARTITION OF 機能をサポートします。[6db4598]

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 OPERATOR 文

以下の属性を変更できるようになりました。従来は CREATE OPERATOR 文でのみ指定できました。[2b5154b]

- COMMUTATOR
- NEGATOR
- HASHES
- MERGES

例 35 ALTER OPERATOR 文による属性変更

```
postgres=# CREATE OR REPLACE FUNCTION alter_op_test_fn_bool_real(boolean, real)
        RETURNS boolean AS $$ SELECT NULL::BOOLEAN; $$ LANGUAGE sql IMMUTABLE ;
CREATE FUNCTION
postgres=# CREATE OPERATOR === (
             LEFTARG = boolean,
             RIGHTARG = real,
             PROCEDURE = alter_op_test_fn_bool_real) ;
CREATE OPERATOR
postgres=# ALTER OPERATOR === (boolean, real) SET (MERGES = false) ;
ALTER OPERATOR
postgres=# ALTER OPERATOR === (boolean, real) SET (HASHES = false) ;
ALTER OPERATOR
```

3.2.2. ALTER SYSTEM 文

エクステンション等で使用するモジュール名を含むパラメーター（foo.bar など）を指定できるようになりました。[2d870b4]

例 36 ALTER SYSTEM 文によるエクステンション用パラメーターの変更

```
postgres=# ALTER SYSTEM SET foo.bar = 100 ;
ALTER SYSTEM
postgres=# SELECT pg_reload_conf() ;
 pg_reload_conf
----------------
 t
(1 row)

postgres=# SHOW foo.bar ;
 foo.bar
---------
 100
```

3.2.3. ALTER TABLE 文

ALTER TABLE 文には以下の拡張が実装されました。

□ デフォルトのアクセスメソッドへ変更

パラメーターdefault_table_access_method に指定されたアクセスメソッドに変更するため、アクセスメソッド名に DEFAULT を指定できるようになりました。[d61a6ca]

構文

```
ALTER TABLE table_name SET ACCESS METHOD DEFAULT
```

例 37 アクセスメソッドの変更

```
postgres=> ALTER TABLE data1 SET ACCESS METHOD DEFAULT ;
ALTER TABLE
```

□ 生成列の変更

生成列（GENERATE 句）の計算式を変更できるようになりました。列の計算式を変更した場合、テーブル内の既存データは再計算が行われます。 [5d06e99]

構文

```
ALTER TABLE table_name ALTER COLUMN column_name SET EXPRESSION AS (expression)
```

例 38 SET EXPRESSION 句の指定

```
postgres=> CREATE TABLE data1(c1 INT, c2 INT GENERATED ALWAYS AS (c1 * 2)
               STORED) ;
CREATE TABLE
postgres=> INSERT INTO data1(c1) VALUES (100) ;
INSERT 0 1
postgres=> SELECT * FROM data1 ;
 c1  | c2
-----+-----
 100 | 200
(1 row)

postgres=> ALTER TABLE data1 ALTER COLUMN c2 SET EXPRESSION AS (c1 * 200) ;
ALTER TABLE
postgres=> INSERT INTO data1(c1) VALUES (200) ;
INSERT 0 1
postgres=> SELECT * FROM data1 ;
 c1  |  c2
-----+-------
 100 | 20000
 200 | 40000
(2 rows)
```

□ 統計情報の設定

単一列の統計情報の格納量をパラメーターdefault_statistics_target に戻す場合に、DEFAULT を使用できるようになりました。従来のバージョンでは-1 を指定していました。これに伴い pg_attribute カタログの attstattarget 列は NOT NULL 制約が削除され、デフォルト値を使用する場合には NULL 値が指定されます。[4f62250, 5567996]

例 39 STATISTICS の変更

```
postgres=> ALTER TABLE data1 ALTER COLUMN c1 SET STATISTICS 110 ;
ALTER TABLE
postgres=> SELECT attname,attstattarget FROM pg_attribute WHERE attrelid=
  (SELECT oid FROM pg_class WHERE relname='data1') AND attname='c1' ;
 attname | attstattarget
---------+---------------
 c1      |           110
(1 row)

postgres=> ALTER TABLE data1 ALTER COLUMN c1 SET STATISTICS DEFAULT ;
ALTER TABLE
postgres=> SELECT attname,attstattarget FROM pg_attribute WHERE attrelid=
   (SELECT oid FROM pg_class WHERE relname='data1') AND attname='c1' ;
 attname | attstattarget
---------+---------------
 c1      |          null
(1 row)
```

3.2.4. CLUSTER 文

テーブル名を省略した場合に、オプションを括弧で囲む構文が許可されるようになりました。この修正により REINDEX 文や VACUUM 文と同一構文がサポートされるようになりました。[cdaedfc]

例 40 VERBOSE オプションの指定

```
postgres=> CLUSTER (VERBOSE) ;
INFO:  clustering "public.data1" using index scan on "data1_pkey1"
INFO:  "public.data1": found 0 removable, 1000000 nonremovable row versions in
5406 pages
DETAIL:  0 dead row versions cannot be removed yet.
CPU: user: 0.22 s, system: 0.00 s, elapsed: 0.25 s.
CLUSTER
```

3.2.5. COPY 文

COPY 文には以下の新機能が実装されました。

□ FORCE_NULL オプション

COPY 文のオプション FORCE_NULL、FORCE_NOT_NULL 句にアスタリスク（*）を指定できるようになりました。[8c852ba]

例 41 FORCE_NULL * 句の指定

```
postgres=# COPY data1 FROM stdin WITH (FORMAT csv, FORCE_NULL *) ;
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
>> 200,""
>> 300,data3
>> ¥.
COPY 3
postgres=# SELECT * FROM data1 ;
 c1  |  c2
-----+-------
 100 | data1
 200 | null
 300 | data3
(3 rows)
```

□ ON_ERROR オプション

COPY FROM 文に ON_ERROR オプションが追加されました。このオプションはエラー発生時の動作を変更します。このオプションには以下の値を指定できます。[9e2d870, b725b7e]

表 21 オプション設定値

設定値	説明	備考
stop	エラー発生時に処理を停止する	デフォルト
ignore	データ変換エラーの発生を無視し、処理を継続する	

このオプションを ignore に設定した場合でも、制約違反（主キー制約や CHECK 制約）が発生した場合 COPY 文は異常終了します。

例 42 SAVE_ERROR_TO オプションの指定

```
postgres=> CREATE TABLE data1(c1 INT, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> COPY data1 FROM STDIN WITH (FORMAT csv, ON_ERROR ignore) ;
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
>> 200,data2
>> ABC,data3
>> 400,data4
>> ¥.
NOTICE:  1 row were skipped due to data type incompatibility
COPY 3
postgres=> SELECT * FROM data1 ;
 c1  |   c2
-----+--------
 100 |  data1
 200 |  data2
 400 |  data4
(3 rows)
```

□ LOG_VERBOSITY オプション

COPY 文にログの出力レベルを変更する LOG_VERBOSITY オプションが追加されました。LOG_VERBOSITY オプションは ON_ERROR オプションに ignore を設定した COPY FROM 文で発生したエラーの出力を制御します。設定できる値は以下の通りです。[f5a2278]

表 22 オプション設定値

設定値	説明	備考
default	標準のログレベル	デフォルト
verbose	詳細なログレベル	

例 43 LOG_VERBOSE オプションの指定

```
postgres=> COPY data1 FROM STDIN WITH (FORMAT csv, ON_ERROR ignore,
                LOG_VERBOSITY default) ;
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
>> 200,data2
>> ABC,data3
>> 400,data4
>> ¥.
NOTICE:  1 row was skipped due to data type incompatibility
COPY 3
postgres=> TRUNCATE TABLE data1 ;
TRUNCATE TABLE
postgres=> COPY data1 FROM STDIN WITH (FORMAT csv, 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
>> 200,data2
>> ABC,data3
>> 400,data4
>> ¥.
NOTICE:  skipping row due to data type incompatibility at line 3 for column c1:
"ABC"
NOTICE:  1 row was skipped due to data type incompatibility
COPY 3
```

3.2.6. CREATE TABLE 文

CREATE TABLE 文には以下の新機能が実装されました。

□ 名前付き NOT NULL 制約

NOT NULL 制約に名前を付与し、pg_constraints カタログに格納されるようになりました。NOT NULL 制約は contype 列には'n'が指定されます。テーブル作成者が名前を指定しなかった場合はテーブル名や列名から制約名は自動的に作成されます。ALTER TABLE 文

で名前付き NOT NULL 制約を追加することもできます。[b0e96f3]

構文

```
CREATE TABLE table (col, …, CONSTRAINT cons_name NOT NULL column_name)
ALTER TABLE table ADD CONSTRAINT cons_name NOT NULL column_name
```

例 44 NOT NULL 制約の名前

```
postgres=> CREATE TABLE data1(c1 INT NOT NULL, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT conname, contype FROM pg_constraint WHERE contype='n' ;
      conname      | contype
-------------------+---------
 data1_c1_not_null | n
(1 row)

postgres=> ¥d+ data1
                                                  Table "public.data1"
 Column |         Type          | Collation | Nullable | Default | Storage …
--------+-----------------------+-----------+----------+---------+-------- …
c1      | integer               |           | not null |         | plain    …
c2      | character varying(10) |           |          |         | extended …
Not-null constraints:
    "data1_c1_not_null" NOT NULL "c1"
Access method: heap

postgres=> CREATE TABLE data2(c1 INT, CONSTRAINT c1_not_null NOT NULL c1) ;
CREATE TABLE
```

□ WITHOUT OVERLAPS 句

主キー制約に範囲の重複を許さない WITHOUT OVERLAPS 句を指定できるようになりました。主キー制約この句を指定する場合には少なくとも２つの列が必要になります。また ALTER TABLE 文で主キーを追加することもできます。[46a0cd4]

例 45 WITHOUT OVERLAPS 句を指定した主キー

```
postgres=> CREATE EXTENSION btree_gist ;
CREATE EXTENSION
postgres=> CREATE TABLE range1(
  id1 INT,
  rg1 INT4RANGE,
  val VARCHAR(10),
  CONSTRAINT range1_pkey PRIMARY KEY (id1, rg1 WITHOUT OVERLAPS)
) ;
CREATE TABLE
postgres=> INSERT INTO range1 VALUES (100, '[0, 2)', 'value1') ;
INSERT 0 1
postgres=> INSERT INTO range1 VALUES (100, '[2, 4)', 'value2') ;
INSERT 0 1
postgres=> INSERT INTO range1 VALUES (100, '[1, 3)', 'pkey error') ;
ERROR:  conflicting key value violates exclusion constraint "range1_pkey"
DETAIL:  Key (id1, rg1)=(100, [1,3)) conflicts with existing key (id1,
rg1)=(100, [0,2)).
postgres=> SELECT conname, conwithoutoverlaps FROM pg_constraint
  WHERE conname='range1_pkey' ;
  Conname    | conwithoutoverlaps
-------------+--------------------
 range1_pkey | t
(1 row)
```

□ 外部キーの PERIOD 句

FOREIGN KEY 列に指定された範囲型とマルチ範囲型の列に PERIOD 句を指定できるようになりました。この制約は等価性ではなく、範囲の含有をチェックします。ON {UPDATE | DELETE} {CASCADE, SET NULL, SET DEFAULT}句とは併用できません。[34768ee]

構文

```
CONSTRAINT name FOREIGN KEY (column, PERIOD column)
```

例 46 PERIOD 句の指定

```
postgres=> CREATE TABLE data1rng(
    id int4range,
    valid_at daterange,
    CONSTRAINT temporal_rng_pk PRIMARY KEY (id, valid_at WITHOUT OVERLAPS)) ;
CREATE TABLE
postgres=> CREATE TABLE data1rng_fk (
    id int4range,
    valid_at daterange,
    parent_id int4range,
    CONSTRAINT fk_data1rng_pk PRIMARY KEY (id, valid_at WITHOUT OVERLAPS),
    CONSTRAINT fk_data1rng_fk FOREIGN KEY (parent_id, PERIOD valid_at)
        REFERENCES data1rng (id, PERIOD valid_at)) ;
CREATE TABLE
```

□ パーティション・テーブルのアクセスメソッド

パーティション・テーブルに対してアクセスメソッドを指定できるようになりました。CREATE TABLE USING 文または ALTER TABLE SET ACCESS METHOD 文で指定できます。指定した値はパーティション・テーブル配下に作成されるパーティションのアクセスメソッドのデフォルト値として使用されます。既存のパーティションは変更されません。[374c7a2]

例 47 パーティション・テーブルとアクセスメソッド

```
postgres=> CREATE TABLE part1 (c1 INT PRIMARY KEY, c2 VARCHAR(10))
                PARTITION BY RANGE(c1) USING heap ;
CREATE TABLE
postgres=> ALTER TABLE part1 SET ACCESS METHOD DEFAULT ;
ALTER TABLE
```

3.2.7. EXPLAIN 文

EXPLAIN 文には以下の機能が追加されました。

□ JSON フォーマットの拡張

JSON フォーマットの出力に”Local I/O Read Time”と”Local I/O Write Time”が出力されるようになりました。この出力を有効化するにはパラメーターtrack_io_timing を on に設定する必要があります。[295c36c]

例 48 JSON フォーマットの出力

```
postgres=> EXPLAIN (ANALYZE, BUFFERS, FORMAT json) SELECT * FROM data1 ;
              QUERY PLAN
--------------------------------------
 [                                   +
   {                                 +
     "Plan": {                       +
       "Node Type": "Seq Scan",      +
       "Parallel Aware": false,      +
...
       "Shared I/O Write Time": 0.000,+
       "Local I/O Read Time": 0.000,  +
       "Local I/O Write Time": 0.000, +
       "Temp I/O Read Time": 0.000,   +
...
```

□ MEMORY 句の追加

EXPLAIN 文に実行計画作成時のメモリー設定を出力する MEMORY オプションが追加されました。MEMORY 句を指定された EXPLAIN 文には確保されたメモリー量(allocated)と実際に使用されたメモリー量（used）の情報が出力されます。[5de890e]

例 49 MEMORY 句の指定

```
postgres=> EXPLAIN (MEMORY) SELECT COUNT(*) FROM data1 ;
                            QUERY PLAN
-------------------------------------------------------------------
 Aggregate  (cost=17906.00..17906.01 rows=1 width=8)
   ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=0)
 Planning:
   Memory: used=9104 bytes  allocated=16384 bytes
(4 rows)
```

□ SERIALIZE 句の追加

クエリーの実行により出力されたデータ量と時間の情報を出力する SERIALIZE 句が追加されました。このオプションは ANALYZE 句と同時に使用します。オプションには以下の値が追加できます。[0628670]

表 23 オプション設定値

設定値	説明	備考
NONE	シリアライズ情報を出力しない	デフォルト
TEXT	テキスト・フォーマットの場合	SERIALIZE のみのデフォルト
BINARY	バイナリ・フォーマットの場合	

例 50 シリアライズ情報の出力

```
postgres=> EXPLAIN (ANALYZE, SERIALIZE BINARY) SELECT * FROM data1 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=10) (actual
 time=0.005..36.797 rows=1000000 loops=1)
 Planning Time: 0.056 ms
 Serialization: time=84.192 ms  output=18555kB  format=binary
 Execution Time: 154.041 ms
(4 rows)
```

3.2.8. MERGE 文

MERGE 文には以下の新機能が追加されました。

□ BY SOURCE 句

ソーステーブルと一致しない場合の動作（WHEN NOT MATCHED BY SOURCE）が追加されました。このアクションはソーステーブルに存在しないが、ターゲットテーブルに存在する列に対するタプルを操作します。UPDATE、DELETE、DO NOTHING を指定できます。[0294df2]

例 51 BY SOURCE 句の追加

```
postgres=> SELECT * FROM src1 ;
 c1  |  c2
-----+---------
 100 | source1
 200 | source2
(2 rows)

postgres=> SELECT * FROM tgt1 ;
 c1  |  c2
-----+---------
 100 | target1
 300 | target3
(2 rows)

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' ;
MERGE 2
postgres=> SELECT * FROM tgt1 ;
 c1  |    c2
-----+-------------
 100 | target1
 300 | not matched
 200 | source2
(3 rows)
```

従来の WHEN NOT MATCHED 句は BY TARGET 句の省略形とみなされます。

□ RETURNING 句のサポート

MERGE 文に RETURNING 句を指定できるようになりました。RETURNING 句により返されたタプルの種別（INSERT/UPDATE/DELETE）を判定するために特別な関数 merge_action が追加されました。RETURNING 句の出力は他の DML と同様にソース・リレーションとして利用できます。[c649fa2]

例 52 RETURNING 句を指定した MERGE 文

```
postgres=> CREATE TABLE src1(c1 INT PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> INSERT INTO src1 VALUES (100, 'data1'),(300, 'data3') ;
INSERT 0 2
postgres=> CREATE TABLE dst1(c1 INT PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> INSERT INTO dst1 VALUES (100, 'data1'),(200, 'data2') ;
INSERT 0 2
postgres=> MERGE INTO dst1 d
  USING src1 s ON s.c1 = d.c1
  WHEN MATCHED THEN
     UPDATE SET c2 = 'updated'
  WHEN NOT MATCHED THEN
     INSERT (c1, c2) VALUES (s.c1, s.c2)
  RETURNING merge_action(), d.* ;
 merge_action | c1  |   c2
--------------+-----+---------
 UPDATE       | 100 | updated
 INSERT       | 300 | data3
(2 rows)

MERGE 2
```

□ 更新可能ビューに対する MERGE 文

MERGE 文に指定するターゲットとして更新可能ビューを利用できるようになりました。[5f2e179]

例 53 更新可能ビューに対する MERGE 文

```
postgres=> CREATE TABLE src1(c1 INT PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> INSERT INTO src1 VALUES (generate_series(1, 10), 'data1') ;
INSERT 0 10
postgres=> CREATE TABLE dst1(c1 INT PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE VIEW view1 AS SELECT * FROM dst1 ;
CREATE VIEW
postgres=> MERGE INTO view1 AS v1 USING src1 AS s1
  ON v1.c1 = s1.c1
  WHEN NOT MATCHED THEN
    INSERT (c1, c2) VALUES (s1.c1, s1.c2) ;
MERGE 10
```

3.2.9. PL/pgSQL

PL/pgSQL には以下の拡張が実装されました。

□ データ型属性の追加

列のデータ型を示す%TYPE 属性、タプル全体のデータ型を示す%ROWTYPE 属性を使用できるようになりました。これにより PL/pgSQL プログラム内で列のデータ型を直接記述することが不要になりました。 [5e8674d]

例 54 %TYPE 属性と%ROWTYPE 属性

```
postgres=> CREATE OR REPLACE FUNCTION func1() RETURNS TEXT AS $$
  DECLARE
    curs1 CURSOR FOR SELECT * FROM data1;
    data1var public.data1%ROWTYPE;
    col2var public.data1.c2%TYPE;
  BEGIN
    OPEN curs1;
    LOOP
      FETCH curs1 INTO data1var;
      IF NOT FOUND THEN
        EXIT;
      END IF;
      col2var := data1var.c2;
      RAISE NOTICE 'Column2: %', col2var;
    END LOOP;
    CLOSE curs1;
    RETURN null;
  END; $$ LANGUAGE plpgsql;
CREATE FUNCTION
```

%TYPE、%ROWTYPE は Oracle Database の PL/SQL 互換構文です。ただし Oracle PL/SQL では%ROWTYPE はカーソルに対しても指定することができます。

例 55 Oracle Database の%ROWTYPE 記述

```
CREATE OR REPLACE FUNCTION func1 RETURN NUMBER
IS
  CURSOR curs1 IS SELECT * FROM data1;
  data1var curs1%ROWTYPE;
  col2var data1.c2%TYPE;
BEGIN
```

□ INTERNAL サブトランザクション

BeginInternalSubTransaction API をパラレル・モードで実行できるようになりました。以下の例は PostgreSQL 16 で PARALLEL SAFE 指定の PL/pgSQL 関数をパラレル・モードで実行し、エラーが発生しています。[0075d78]

例 56 PostgreSQL 16 では実行エラーが発生する

```
postgres=> CREATE FUNCTION zero_divide() RETURNS INT AS $$
             DECLARE v INT := 0;
           BEGIN
             RETURN 10 / v;
           END;
           $$ LANGUAGE plpgsql PARALLEL SAFE ;
CREATE FUNCTION
postgres=> CREATE FUNCTION error_trap_test() RETURNS TEXT AS $$
           BEGIN
             PERFORM zero_divide();
             RETURN 'no error detected!';
           EXCEPTION WHEN division_by_zero THEN
             RETURN 'division_by_zero detected';
           END;
           $$ LANGUAGE plpgsql PARALLEL SAFE ;
CREATE FUNCTION
postgres=> SET debug_parallel_query = on ;
SET
postgres=> SELECT error_trap_test() ;
ERROR:  cannot start subtransactions during a parallel operation
CONTEXT:  PL/pgSQL function error_trap_test() line 2 during statement block
entry
parallel worker
```

3.2.10. データ型

データ型には以下の拡張が実装されました。[]

□ INTERVAL データ型

INTERVAL 型で+/-Infinity がサポートされました。[519fc1b]

例 57 INTERVAL 型と Infinity

```
postgres=> SELECT timestamp 'infinity' - timestamp '-infinity' ;
 ?column?
----------
 infinity
(1 row)

postgres=> SELECT age(timestamp '-infinity', timestamp 'infinity') ;
    age
-----------
 -infinity
(1 row)
```

□ 範囲型

範囲演算子<@と@>に対して定数範囲値を含む式を直接比較できるようになりました。[075df6b]

例 58 PostgreSQL 17 の動作

```
postgres=> EXPLAIN (VERBOSE, COSTS OFF)
 SELECT CURRENT_DATE <@ DATERANGE 'empty' ;
   QUERY PLAN
----------------
 Result
   Output: false
(2 rows)
```

例 59 PostgreSQL 16 の動作

```
postgres=> EXPLAIN (VERBOSE, COSTS OFF)
  SELECT CURRENT_DATE <@ DATERANGE 'empty' ;
                    QUERY PLAN
-----------------------------------------------
 Result
   Output: (CURRENT_DATE <@ 'empty'::daterange)
 Query Identifier: 4806765806694262066
(3 rows)
```

3.2.11. JSON 関連

JSON 関連構文が強化されました。

□ JSON コンストラクター

以下の JSON コンストラクター関数が追加されました。[03734a7]

構文

```
JSON( expression [ FORMAT JSON [ ENCODING UTF8 ] ] [ { WITH | WITHOUT } UNIQUE
[ KEYS ]])
text JSON_SCALAR( expression )
text | bytea JSON_SERIALIZE( expression [ FORMAT JSON [ ENCODING UTF8 ] ]
[ RETURNING data_type [ FORMAT JSON [ ENCODING UTF8 ] ] ])
```

例 60 追加された JSON コンストラクター

```
postgres=> SELECT JSON('{ "val": 1}'::bytea FORMAT JSON ENCODING UTF8) ;
    json
-------------
 { "val": 1}
(1 row)

postgres=> SELECT JSON_SCALAR(123.45::numeric) ;
 json_scalar
-------------
 123.45
(1 row)

postgres=> SELECT JSON_SERIALIZE(JSON('{ "val" : 1 }')) ;
 json_serialize
----------------
 { "val" : 1 }
(1 row)
```

□ JSONPATH メソッドの追加

以下の JSONPATH メソッドが追加されました。[66ea94e, 4697454]

表 24 オプション設定値

メソッド	説明	備考
.bigint()	大規模整数値を返す	
.boolean()	真（True）か偽（False）かを返す	
.date()	日付を返す	
.decimal([p [,s]])	指定した精度の数値を返す	
.integer()	整数を返す	
.number()	数値を返す	
.string()	文字列を返す	
.time()	時刻を返す	
.time_tz()	タイムゾーン付の時刻を返す	注
.timestamp()	タイムスタンプを返す	
.timestamp_tz()	タイムゾーン付のタイムスタンプを返す	注

例 61 JSONPATH メソッドの実行

```
postgres=> SELECT
    jsonb_path_query_array('[1, "yes", false]', '$[*].boolean()'),
    jsonb_path_query('"2023-08-15"', '$.datetime().string()'),
    jsonb_path_query('{"len": "9876543219"}', '$.len.bigint()'),
    jsonb_path_query('1234.5678', '$.decimal(6, 2)'),
    jsonb_path_query('{"len": "12345"}', '$.len.integer()'),
    jsonb_path_query('{"len": "123.45"}', '$.len.number()'),
    jsonb_path_query('"2023-08-15"', '$.date()'),
    jsonb_path_query('"12:34:56"', '$.time()'),
    jsonb_path_query('"12:34:56 +05:30"', '$.time_tz()'),
    jsonb_path_query('"2023-08-15 12:34:56"', '$.timestamp()') ;
-[ RECORD 1 ]----------+---------------------
jsonb_path_query_array | [true, true, false]
jsonb_path_query       | "2023-08-15"
jsonb_path_query       | 9876543219
jsonb_path_query       | 1234.57
jsonb_path_query       | 12345
jsonb_path_query       | 123.45
jsonb_path_query       | "2023-08-15"
jsonb_path_query       | "12:34:56"
jsonb_path_query       | "12:34:56+05:30"
jsonb_path_query       | "2023-08-15T12:34:56"
```

タイムゾーンは IMMUTABLE ではないため、jsonb_path_query 関数上では.timestamp_tz()メソッドは実行できません。jsonb_path_query_tz 関数を使います。

例 62 タイムゾーン付タイムスタンプの扱い

```
postgres=> SELECT
        jsonb_path_query('"2023-08-15 12:34:56"', '$.timestamp_tz()') ;
ERROR:  cannot convert value from timestamp to timestamptz without time zone
usage
HINT:  Use *_tz() function for time zone support.
postgres=> SELECT
        jsonb_path_query_tz('"2023-08-15 12:34:56"', '$.timestamp_tz()') ;
     jsonb_path_query_tz
---------------------------
 "2023-08-15T03:34:56+00:00"
(1 row)
```

□ JSON 検索

JCON データ内を検索する関数が追加されました。JSON_EXISTS 関数は PASSING 句を使用して項目が生成された場合に true を返します。JSON_QUERY 関数と JSON_VALUE 関数は指定された JSON データから PASSING 句を使って適用した結果を返します。[6185c97]

構文

```
JSON_EXISTS ( context_item, path_expression [ PASSING { value AS varname }
[, ...]] [ { TRUE | FALSE | UNKNOWN | ERROR } ON ERROR ])

JSON_QUERY ( context_item, path_expression [ PASSING { value AS varname }
[, ...]] [ RETURNING data_type [ FORMAT JSON [ ENCODING UTF8 ] ] ] [ { WITHOUT
| WITH { CONDITIONAL | [UNCONDITIONAL] } } [ ARRAY ] WRAPPER ] [ { KEEP | OMIT }
QUOTES [ ON SCALAR STRING ] ] [ { ERROR | NULL | EMPTY { [ ARRAY ] | OBJECT } |
DEFAULT expression } ON EMPTY ] [ { ERROR | NULL | EMPTY { [ ARRAY ] | OBJECT }
| DEFAULT expression } ON ERROR ])

JSON_VALUE ( context_item, path_expression [ PASSING { value AS varname }
[, ...]] [ RETURNING data_type ] [ { ERROR | NULL | DEFAULT expression } ON
EMPTY ] [ { ERROR | NULL | DEFAULT expression } ON ERROR ])
```

例 63 JSON 検索

```
postgres=> SELECT JSON_EXISTS(jsonb '{"a": 1, "b": 2}', '$.* ? (@ > $x)'
                PASSING 1 AS x) ;
 json_exists
-------------
 t
(1 row)

postgres=> SELECT JSON_QUERY(jsonb '"aaa"', '$' RETURNING text KEEP QUOTES) ;
 json_query
------------
 "aaa"
(1 row)

postgres=> SELECT JSON_VALUE(jsonb '[1,2]', '$[*]' DEFAULT '0' ON ERROR) ;
 json_value
------------
 0
(1 row)
```

□ テーブル変換

PostgreSQL 17 に JSON データをリレーショナルビューに変換する JSON_TABLE 関数が追加されました。[de36004, bb766cd]

構文

```
JSON_TABLE (
    context_item, path_expression [ AS json_path_name ] [ PASSING { value AS
varname } [, ...] ] COLUMNS ( json_table_column [, ...] )
    [ { ERROR | EMPTY } ON ERROR ]
)
```

例 64 JSON_TABLE 関数の実行

```
postgres=> SELECT * FROM JSON_TABLE(jsonb '123', '$'
        COLUMNS (item int PATH '$', foo int)) bar ;
 item | foo
------+-----
  123 |
(1 row)

postgres=>  SELECT * FROM JSON_TABLE(jsonb '"a"', '$'
  COLUMNS (a int PATH 'strict $.a' DEFAULT 1 ON EMPTY DEFAULT 2 ON ERROR)) jt ;
 a
---
 2
(1 row)
```

□ jsonb_populate_record_valid 関数

jsonb_populate_record_valid 関数は指定された JSON オブジェクトに対して jsonb_populate_record 関数がエラーを発生させない場合は true を、エラーになる場合は false を返します。[1edb3b4]

構文

```
boolean jsonb_populate_record_valid(anyelement, jsonb)
```

例 65 jsonb_populate_record_valid 関数の実行

```
postgres=> CREATE TYPE jsb_char2 AS (a char(2)) ;
CREATE TYPE
postgres=> SELECT jsonb_populate_record_valid(NULL::jsb_char2,
                '{"a": "aaa"}') ;
jsonb_populate_record_valid
-----------------------------
 f
(1 row)
```

3.2.12. 関数

以下の関数が追加／拡張されました。

□ pg_promote 関数

postmaster プロセスへのシグナル送信処理が失敗した場合にエラーメッセージを出力するようになりました。[f593c55]

出力される可能性があるエラーメッセージ

```
ERROR: failed to send signal to postmaster: {PID}
ERROR: terminating connection due to unexpected postmaster exit
```

□ pg_basetype 関数

pg_basetype 関数は指定されたドメインの基底タイプを返します。[b154d8a]

構文

```
regtype pg_basetype(regtype)
```

例 66 pg_basetype 関数の実行

```
postgres=> CREATE DOMAIN mytext AS text ;
CREATE DOMAIN
postgres=> CREATE DOMAIN mytext_child AS mytext ;
CREATE DOMAIN
postgres=> SELECT pg_basetype('mytext'::regtype) ;
 pg_basetype
-------------
 text
(1 row)

postgres=> SELECT pg_basetype('mytext_child'::regtype) ;
 pg_basetype
-------------
 text
(1 row)
```

□ pg_wal_replay_wait プロシージャ

スタンバイ側でトランザクション開始前に特定の WAL Location までリプレイされることを待機するプロシージャ pg_wal_replay_wait が追加されました。[06c418e]

構文

```
pg_wal_replay_wait(target_lsn pg_lsn, timeout bigint DEFAULT 0)
```

□ random 関数

最小値と最大値を指定して「最小値≦ランダム≦最大値」となる乱数を生成する関数が追加されました。[e634132]

構文

```
integer random(min integer, max integer)
bigint random(min bigint, max bigint)
numeric random(min numeric, max numeric)
```

例 67 最小値と最大値を指定した乱数の発生

```
postgres=> SELECT random(100, 200), random(10.25, 20.34) ;
 random | random
--------+--------
    153 |  19.21
(1 row)
```

□ to_bin / to_oct 関数

数値を２進数文字列または８進数文字列に変換する関数が追加されました。[260a1f1]

構文

```
text to_bin(integer | bigint)
text to_oct(integer | bigint)
```

例 68 to_bin, to_oct 関数の実行

```
postgres=> SELECT TO_BIN(100) bin, TO_OCT(100) oct ;
   bin    | oct
---------+-----
 1100100 | 144
(1 row)
```

□ to_regtypemod 関数

データ型文字列からデータ型の typemod 情報を取得します。この情報は format_type 関数等で使えます。[1218ca9]

構文

```
integer to_regtypemod(text)
```

例 69 to_regtypemod 関数の実行

```
postgres=> SELECT to_regtypemod('varchar(10)'),
        format_type(to_regtype('varchar(10)'), to_regtypemod('varchar(10)')) ;
 to_regtypemod |      format_type
---------------+-----------------------
            14 | character varying(10)
(1 row)
```

□ to_timestamp 関数

フォーマット文字列に TZ と OF を指定できるようになりました。従来のバージョンでは to_char 関数でのみ利用可能でした。[8ba6fdf]

例 70 to_timestamp 関数の実行

```
postgres=> SELECT
        TO_TIMESTAMP('2024-01-26 09:12 -05', 'YYYY-MM-DD HH12:MI OF') ;
      to_timestamp
------------------------
 2024-01-26 23:12:00+09
(1 row)

postgres=> SELECT
        TO_TIMESTAMP('2024-01-26 09:12 EST', 'YYYY-MM-DD HH12:MI TZ') ;
      to_timestamp
------------------------
 2024-01-26 23:12:00+09
(1 row)
```

□ UNICODE 関連

UNICODE のバージョンに関連する基本的な関数が追加されました。unicode_version 関数はデータベース・クラスタで使用する UNICODE のバージョンを、icu_unicode_version 関数は ICU ライブラリが使う UNICODE のバージョンを出力します。unicode_assigned 関数は指定された文字列が UNICODE にアサインされているかを検証します。[a02b37f]

構文

```
text unicode_version()
text icu_unicode_version()
boolean unicode_assigned(text)
```

例 71 UNICODE 関連関数の実行

```
postgres=> SELECT unicode_version() ;
-[ RECORD 1 ]---+-----
unicode_version | 15.1

postgres=> SELECT icu_unicode_version() ;
-[ RECORD 1 ]-------+-----
icu_unicode_version | 10.0

postgres=> SELECT unicode_assigned('ABC') ;
-[ RECORD 1 ]----+--
unicode_assigned | t
```

unicode_assigned 関数はデータベースのエンコードが UTF-8 の場合に限り実行できます。エンコードを EUC_JP に設定したデータベースで実行すると以下のエラーが発生します。

例 72 EUC_JP エンコードのデータベースで実行

```
postgres=> SELECT unicode_assigned('ABC') ;
ERROR:  Unicode categorization can only be performed if server encoding is UTF8
```

□ XMLText 関数

XML 標準（）の XMLText 関数が追加されました。この関数はテキスト情報を XML テキスト・ノードに変換します。この関数を実行するには PostgreSQL ビルド時に--with-libxml オプションが必要です。[526fe0d]

構文

```
xml xmltext(text)
```

例 73 XMLTEXT 関数の実行

```
postgres=> SELECT xmltext('foo & <"bar">') ;
              xmltext
-----------------------------------
 foo &amp; &lt;&quot;bar&quot;&gt;
(1 row)
```

□ uuid_extract_version / uuid_extract_timestamp 関数

UUID からバージョン番号、タイムスタンプを抽出する関数が追加されました。[794f10f]

構文

```
smallint uuid_extract_version(uuid)
timestamp with time zone uuid_extract_timestamp(uuid)
```

例 74 UUID から情報を抽出

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

postgres=> SELECT
        uuid_extract_timestamp('C232AB00-9414-11EC-B3C8-9F6BDECED846') ;
 uuid_extract_timestamp
------------------------
 2022-02-23 04:22:22+09
(1 row)
```

□ pg_column_toast_chunk_id 関数

pg_column_toast_chunk_id 関数は TOAST データのチャンク ID を返します。[d1162cf]

構文

```
oid pg_column_toast_chunk_id(any)
```

例 75 pg_column_toast_chunk_id 関数の実行

```
postgres=> CREATE TABLE data1 (col1 TEXT, col2 TEXT STORAGE EXTERNAL) ;
CREATE TABLE
postgres=> INSERT INTO data1 VALUES ('A', repeat('B', 8192)) ;
INSERT 0 1
postgres=> SELECT pg_column_toast_chunk_id(col2) FROM data1 ;
-[ RECORD 1 ]------------+------
pg_column_toast_chunk_id | 16412
```

□ pg_stat_reset_shared 関数

pg_stat_reset_shared 関数のパラメーターを省略するか NULL を指定すると、すべての統計情報カウンターをリセットします。また pg_stat_reset_shared 関数のパラメーターに pg_stat_slru ビューをリセットするための文字列'slru'を指定できるようになりました。[23c8c0c, 2e8a0ed]

構文

```
void pg_stat_reset_shared(target text DEFAULT NULL::text)
```

□ pg_stat_reset_slru 関数

pg_stat_reset_slru 関数のパラメーターを省略すると、すべての SLRU 統計情報カウンターをリセットします。[e5cca62]

構文

```
void pg_stat_reset_slru(target text DEFAULT NULL)
```

パラメーターに指定する要素名が変更されました。変更後の名前以外の文字列を入力すると、「other」項目をリセットします。[bcdfa5f]

表 25 変更された要素名

変更前	変更後	備考
CommitTs	commit_timestamp	
MultiXactMember	multixact_member	
MultiXactOffset	multixact_offset	
Notify	notify	
Serial	serializable	
Subtrans	subtransaction	
Xact	transaction	

3.2.13. 実行計画

より高速な実行計画が選択されるようになりました。

□ GROUP BY 句

複数列を使った GROUP BY 句がソート順に関係が無い場合、列の入れ替えを行うことでソート処理を削減できるようになりました。この動作はパラメーター enable_group_by_reordering を on に設定することで有効になります（デフォルト on）。[0452b46]

例 76 PostgreSQL 17 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT COUNT(*) FROM data1 GROUP BY col2, col1 ;
                    QUERY PLAN
-----------------------------------------------
 GroupAggregate
   Group Key: col1, col2
   ->  Index Only Scan using idx1_data1 on data1
(3 rows)
```

例 77 PostgreSQL 16 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT COUNT(*) FROM data1 GROUP BY col2, col1 ;
          QUERY PLAN
------------------------------
 GroupAggregate
   Group Key: col2, col1
   ->  Sort
         Sort Key: col2, col1
         ->  Seq Scan on data1
(5 rows)
```

□ UNION 句の最適化

サブクエリー内に UNION 句を含む検索で Merge Append が使用できるようになりました。従来は Sort が必要でした。[66c0185]

例 78 PostgreSQL 17 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT COUNT(*) FROM
  (SELECT c1 FROM merge1 UNION SELECT c1 FROM merge2) ;
                          QUERY PLAN
------------------------------------------------------------
 Aggregate
   ->  Unique
         ->  Merge Append
               Sort Key: merge1.c1
               ->  Index Only Scan using merge1_pkey on merge1
               ->  Index Only Scan using merge2_pkey on merge2
(6 rows)
```

例 79 PostgreSQL 16 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT COUNT(*) FROM
                (SELECT c1 FROM merge1 UNION SELECT c1 FROM merge2) ;
                 QUERY PLAN
--------------------------------------------
 Aggregate
   ->  Unique
         ->  Sort
               Sort Key: merge1.c1
               ->  Append
                     ->  Seq Scan on merge1
                     ->  Seq Scan on merge2
(7 rows)
```

□ より良い IS [NOT] NULL ハンドリング

IS NULL 句、IS NOT NULL 句に対して不要な評価を削減する最適化が実装されました。[b262ad4]

例 80 PostgreSQL 17 の実行計画

```
postgres=> ¥d data1
                    Table "public.data1"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | integer               |           | not null |
 c2     | character varying(10) |           |          |
Indexes:
    "data1_pkey" PRIMARY KEY, btree (c1)

postgres=> EXPLAIN SELECT COUNT(*) FROM data1 WHERE c1 IS NOT NULL ;
                            QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=17906.00..17906.01 rows=1 width=8)
   ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=0)
(2 rows)
```

例 81 PostgreSQL 16 の実行計画

```
postgres=> ¥d data1
                    Table "public.data1"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | integer               |           | not null |
 c2     | character varying(10) |           |          |
Indexes:
    "data1_pkey" PRIMARY KEY, btree (c1)

postgres=> EXPLAIN SELECT COUNT(*) FROM data1 WHERE c1 IS NOT NULL ;
                            QUERY PLAN
-------------------------------------------------------------------
 Aggregate  (cost=17906.00..17906.01 rows=1 width=8)
   ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=0)
         Filter: (c1 IS NOT NULL)
(3 rows)
```

□ GiST インデックス

GiST インデックス、SP-GiST インデックスでも Incremental Sort が利用できるようになりました。[625d5b3]

例 82 PostgreSQL 17 の実行計画

```
postgres=> CREATE INDEX idx1_data1 ON data1 USING GIST(c1) ;
CREATE INDEX
postgres=> EXPLAIN (COSTS OFF) SELECT c1, c2, c1<->POINT(5,5) dist FROM data1
               ORDER BY dist, c2 LIMIT 1 ;
                   QUERY PLAN
-----------------------------------------------
 Limit
   ->  Incremental Sort
         Sort Key: ((c1 <-> '(5,5)'::point)), c2
         Presorted Key: ((c1 <-> '(5,5)'::point))
         ->  Index Scan using idx1_data1 on data1
               Order By: (c1 <-> '(5,5)'::point)
(6 rows)
```

例 83 PostgreSQL 16 までの実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT c1, c2, c1<->POINT(5,5) dist FROM data1
                ORDER BY dist, c2 LIMIT 1 ;
                   QUERY PLAN
-------------------------------------------------
 Limit
   ->  Sort
         Sort Key: ((c1 <-> '(5,5)'::point)), c2
         ->  Seq Scan on data1
(4 rows)
```

□ 自己結合の最適化

自己結合時にアクセス対象を削減する最適化機能が実装され、デフォルトで有効になっています。PostgreSQL 16 以前の実行計画に戻すためにはパラメーター enable_self_join_removal を off に設定します。[d3d55ce]

例 84 PostgreSQL 17 の実行計画

```
postgres=> EXPLAIN  SELECT * FROM data1 d1, data1 d2 WHERE d1.c1=d2.c1 and
                d1.c1=100 ;
                                   QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using data1_pkey on data1 d2  (cost=0.42..8.44 rows=1 width=20)
   Index Cond: (c1 = 100)
(2 rows)
```

例 85 PostgreSQL 16 までの実行計画

```
postgres=> SET enable_self_join_removal = off ;
SET
postgres=> EXPLAIN SELECT * FROM data1 d1, data1 d2 WHERE d1.c1=d2.c1 and
                d1.c1=100 ;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Nested Loop  (cost=0.85..16.90 rows=1 width=20)
   ->  Index Scan using data1_pkey on data1 d1  (cost=0.42..8.44 …
         Index Cond: (c1 = 100)
   ->  Index Scan using data1_pkey on data1 d2  (cost=0.42..8.44 …
         Index Cond: (c1 = 100)
(5 rows)
```

□ OR 句を ANY 句に変換

OR 句が一定個数以上指定された場合に ANY 句に変換を行います。閾値はパラメーター or_to_any_transform_limit（デフォルト 5）で決定されます。[72bd38c]

例 86 OR 句から ANY 句へ変換

```
postgres=> SHOW or_to_any_transform_limit ;
 or_to_any_transform_limit
---------------------------
 5
(1 row)

postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1 WHERE c1=10000 OR c1=20000 ;
                    QUERY PLAN
-----------------------------------------------
 Bitmap Heap Scan on data1
   Recheck Cond: ((c1 = 10000) OR (c1 = 20000))
   ->  BitmapOr
         ->  Bitmap Index Scan on data1_pkey
               Index Cond: (c1 = 10000)
         ->  Bitmap Index Scan on data1_pkey
               Index Cond: (c1 = 20000)
(7 rows)

postgres=> SET or_to_any_transform_limit = 0 ;
SET
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1 WHERE c1=10000 OR c1=20000 ;
                      QUERY PLAN
-------------------------------------------------------
 Index Scan using data1_pkey on data1
   Index Cond: (c1 = ANY ('{10000,20000}'::integer[]))
(2 rows)
```

3.3. パラメーターの変更

PostgreSQL 17 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。[a740b21, d3d55ce, 7750fef, a14354c ,2cdf131, 174c480, 0452b46, 51efe38, 53c2a97, bf279dd, d3ae2a2, 210622c, 705843d, 72bd38c]

表 26 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
allow_alter_system	ALTER SYSTEM 文の実行を許可する（sighup）	on
backtrace_on_internal_error	内部エラー発生時のバックトレース出力を制御（superuser）	off
commit_timestamp_buffers	コミット・タイムスタンプ・キャッシュの容量（postmaster）	32
enable_group_by_reordering	GROUP BY の入れ替え最適化（user）	on
enable_self_join_removal	事故結合排除機能を利用する（user）	on
event_triggers	イベントトリガーの有効化（superuser）	on
huge_pages_status	Huge Pages を使っているかを示す（postmaster）	-
io_combine_limit	最大 Raad/Write ブロック数（user）	16
max_notify_queue_pages	NOTIFY/LISTEN キューに割り当てられる最大ページ数（postmaster）	1048576
multixact_member_buffers	マルチ・トランザクションのメンバー・キャッシュ容量（postmaster）	32
multixact_offset_buffers	マルチ・トランザクションのオフセット・キャッシュ容量（postmaster）	16
notify_buffers	LISTEN/NOTIFY メッセージ・キャッシュの容量（postmaster）	16

パラメーター	説明（context）	デフォルト値
or_to_any_transform_limit	OR 句が指定数以上の場合 ANY 句に変換する（user）	5
serializable_buffers	シリアライザブル・トランザクション・キャッシュの容量（postmaster）	32
standby_slot_names	スタンバイ・データベースのレプリケーション・スロット名（signup）	''
subtransaction_buffers	サブ・トランザクション・キャッシュの容量（postmaster）	32
summarize_wal	WAL サマリーを出力（sighup）	off
trace_connection_negotiation	SSL ネゴシエーションのトレースを取得（postmaster）	off
transaction_buffers	トランザクション・ステータス・キャッシュの容量（postmaster）	32
transaction_timeout	トランザクション実行時間のタイムアウト（user）	0
wal_summary_keep_time	WAL サマリーの保存期間（sighup）	10d

□ allow_alter_system

ALTER SYSTEM 文の実行を許可するかを決定します。デフォルト値は on で ALTER SYSTEM 文を実行できます。このパラメーター自身を ALTER SYSTEM 文で変更することはできません。

例 87 トランザクション・タイムアウト

```
postgres=# SHOW allow_alter_system ;
-[ RECORD 1 ]------+----
allow_alter_system | off

postgres=# ALTER SYSTEM SET work_mem='16MB' ;
ERROR:  ALTER SYSTEM is not allowed in this environment
postgres=# SELECT pg_reload_conf() ;
-[ RECORD 1 ]--+--
pg_reload_conf | t

postgres=# SHOW allow_alter_system ;
-[ RECORD 1 ]------+---
allow_alter_system | on

postgres=# ALTER SYSTEM SET allow_alter_system = off ;
ERROR:  parameter "allow_alter_system" cannot be changed
```

□ backtrace_on_internal_error

内部エラー（エラー・コード XX000）が発生した時にバックトレースをログに出力する設定です。デフォルト値は off でバックトレースは出力されません。

例 88 内部エラーのバックトレース

```
postgres=# SET backtrace_on_internal_error = on ;
SET
postgres=# DO $$ BEGIN RAISE EXCEPTION SQLSTATE 'XX000'; END $$ ;
ERROR:  XX000
CONTEXT:  PL/pgSQL function inline_code_block line 1 at RAISE
postgres=# ¥! tail data/log/postgresql.log
2024-02-13 17:07:51.562 JST [19370] ERROR:  XX000
2024-02-13  17:07:51.562  JST  [19370]  CONTEXT:  PL/pgSQL  function
inline_code_block line 1 at RAISE
2024-02-13 17:07:51.562 JST [19370] BACKTRACE:
        /usr/local/pgsql/lib/plpgsql.so(+0x114f3) [0x7f08d6c1e4f3]
        …
        /usr/local/pgsql/lib/plpgsql.so(plpgsql_inline_handler+0x143)
[0x7f08d6c2bbc3]
        postgres:  postgres  postgres  [local]  DO(FunctionCall1Coll+0x40)
[0x931690]
        …
        postgres: postgres postgres [local] DO() [0x78b61c]
        postgres: postgres postgres [local] DO(PostmasterMain+0xc6a) [0x78c51a]
        postgres: postgres postgres [local] DO(main+0x1e4) [0x4f8094]
        /lib64/libc.so.6(__libc_start_main+0xf3) [0x7f08e2531493]
        postgres: postgres postgres [local] DO(_start+0x2e) [0x4f836e]
2024-02-13 17:07:51.562 JST [19370] STATEMENT:  DO $$ BEGIN RAISE EXCEPTION
SQLSTATE 'XX000'; END $$;
```

□ huge_pages_status

パラメーターhuge_pages がデフォルト値（try）の場合、Huge Pages の取得に失敗した場合でもログ出力が行われません。パラメーターhuge_pages_status を確認することでクラスターが Huge Pages を利用しているかを確認できます。

表 27 取得可能な値

設定値	説明	備考
on	Huge Pages を使っている	
off	Huge Pages を使っていない	
unknown	不明（特殊な場合のみ表示される）	

□ transaction_timeout

トランザクションの最大実行時間をミリ秒単位で指定します。タイムアウトが発生するとセッションは強制的に切断されます。デフォルト値は 0 で、タイムアウトは発生しません。

例 89 トランザクション・タイムアウト

```
postgres=> SET transaction_timeout = '10s' ;
SET
postgres=> BEGIN;
BEGIN
postgres=*> SELECT pg_sleep(10) ;
FATAL:  terminating connection due to transaction timeout
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.
```

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。[d0c2860]

表 28 変更されたパラメーター

パラメーター	変更内容
wal_sync_method	Windows 環境では設定値 fsync_writethrough が使用できなくなりました。

3.3.3. 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。[98f320e]

表 29 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 16	PostgreSQL 17	備考
server_version	16.3	17.0	
server_version_num	160003	170000	
vacuum_buffer_usage_limit	256kB	2MB	

3.3.4. 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。[884eee5, f691f5b, c7a3e6b]

表 30 削除されたパラメーター

パラメーター	理由
db_user_namespace	利用者がほとんどいないと判断されて削除されました。
old_snapshot_threshold	正確性とパフォーマンスの問題があったため削除されました。望ましい機能ではあるため将来改善された実装が出てくる可能性はあります。
trace_recovery_messages	pg_waldump コマンド等で代替できるため削除されました。

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. clusterdb

--all と他のオプションを同時に使えるようになりました。[1b49d56]

例 90 --all オプションと--table オプション

```
$ clusterdb --all --table=data
clusterdb: clustering database "demodb"
clusterdb: clustering database "postgres"
clusterdb: clustering database "template1"
```

3.4.2. configure

configure コマンドには以下の機能が実装されました。

□ バージョン出力

コマンド実行時に LLVM と OpenSSL のバージョンが出力されるようになりました。[5e4dacb, 55a428a]

例 91 configure コマンドの出力

```
$ ./configure --with-llvm --with-ssl=openssl
…
checking for llvm-config... /usr/bin/llvm-config
configure: using llvm 12.0.1
…
checking for openssl... /usr/bin/openssl
configure: using openssl: OpenSSL 1.1.1k  FIPS 25 Mar 2021
…
```

□ インジェクション・ポイントの有効化

インジェクション・ポイントは開発者がカスタムコードを実行できるようにし、複雑な競合のテスト等を支援する機能です。この機能を利用するためには configure コマンド実行時に--enable-injection-points を指定する必要があります。[d86d20f]

3.4.3. pg_archivecleanup

pg_archivecleanup コマンドには以下の機能が追加されました。[dd7c60f, 3f8c98d]

□ 長い名前のオプション

単一文字のオプションに長い名前が追加されました。

表 31 追加された長いオプション

短縮形	長い名前のオプション	説明
-d	--debug	標準エラーにデバッグログを出力します。
-n	--dry-run	削除されるファイル名を標準出力に出力します。
-x	--strip-extension=EXT	ファイル名から除外する拡張子を指定します。

□ ヒストリー・ファイルの削除

ヒストリー・ファイルを削除する--clean-backup-history オプション（短縮形-b）が追加されました。

3.4.4. pg_createsubscriber

pg_createsubscriber コマンドが追加されました。このコマンドはストリーミング・レプリケーションのスタンバイサーバーを論理レプリケーションのスタンバイサーバーに変換します。[d44032d]

表 32 主なオプション

オプション	説明
--database=DB	SUBSCRIPTION を作成するデータベース
--pgdata=DATADIR	変換するストリーミング・レプリケーションのデータ
--dry-run	変換のテスト
--subscriber-port=PORT	サブスクライバーのポート番号
--publisher-server=CONN	パブリッシャーへの接続文字列
--subscriber-username=USER	SUBSCRIPTION 所有者
--publication=NAME	PUBLICATION の名前
--replication-slot=NAME	レプリケーション・スロット名
--subscription=NAME	SUBSCRIPTION 名

例 92 pg_createsubscriber コマンドの実行

```
$ pg_createsubscriber -D data.s --publisher-server='host=dbsvr1 port=5732 dbname=postgres'
2024-03-25 22:44:18.926 JST [44033] LOG:  redirecting log output to logging collector process
2024-03-25 22:44:18.926 JST [44033] HINT:  Future log output will appear in directory "log".
2024-03-25 22:44:19.153 JST [44047] LOG:  redirecting log output to logging collector process
2024-03-25 22:44:19.153 JST [44047] HINT:  Future log output will appear in directory "log".
```

pg_createsubscriber コマンドを実行すると、プライマリーサーバーでは指定されたデータベースの全テーブル（FOR ALL TABLES）を指定して PUBLICATION と論理レプリケーション・スロットが作成されます。スタンバーサーバーはストリーミング・レプリケーションのスタンバイではなくなり、SUBSCRIPTION が作成されます。

プライマリーサーバーに作成される PUBLICATION とレプリケーション・スロット名、スタンバイサーバーに作成される SUBSCRIPTION 名のデフォルトは「pg_createsubscriber_{DB OID}_{ランダム整数}」です。--pgdata オプション（-D）で指定したスタンバイサーバーは停止している必要があります。

3.4.5. pg_basebackup

pg_basebackup コマンドには以下の新機能が実装されました。

□ --d オプション

--write-recovery-conf オプション（-R）と同時に-d オプションにデータベース名を指定された場合、postgresql.auto.conf に指定される primary_conninfo パラメーターの設定にデータベース名の指定が出力されます。[a145f42]

例 93 primary_conninfo パラメーターに dbname 追加

```
$ pg_basebackup -D back -d "dbname=demodb" -R
$ cat back/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
primary_conninfo    =    'user=postgres    passfile=''/home/postgres/.pgpass''
channel_binding=disable     port=5432     sslmode=disable     sslcompression=0
sslcertmode=disable          sslsni=1          ssl_min_protocol_version=TLSv1.2
gssencmode=disable krbsrvname=postgres gssdelegation=0 target_session_attrs=any
load_balance_hosts=disable dbname=demodb'
```

□ --help オプション

ヘルプ・メッセージに--checkpoint オプションのデフォルト値が出力されるようになりました。 [cd02b35]

例 94 --help オプション

```
$ pg_basebackup --help
pg_basebackup takes a base backup of a running PostgreSQL server.
...
General options:
  -c, --checkpoint=fast|spread
                         set fast or spread (default) checkpointing
...
```

□ --incremental オプション

差分バックアップを行う--incremental オプション（短縮形 -b）が追加されました。このオプションにはバックアップの基準となるマニフェストファイルのパスを指定します。

3.4.6. pg_dump

pg_dump コマンドには以下のオプションが追加されました。

□ --filter オプション

pg_dump / pg_dumpall / pg_restore コマンドに--filter オプションが追加されました。このオプションにはダンプファイルに含まれる（または除外される）オブジェクトの一覧を記

述します。ファイルのフォーマットは以下の通りです。[a5cf808]

ファイルのフォーマット

```
include または exclude オブジェクトの種類　オブジェクトのパターン
```

表 33 オブジェクトの種類

種類	説明	同等のコマンド・オプション
extension	エクステンション	--extension
foreign_data	外部テーブル	--include-foreign-data
table	テーブル	--table
table_and_children	テーブルと子テーブル	--table-and-children
table_data	テーブルのデータ	--exclude-table-data
table_data_and_children	テーブルと子テーブルのデータ	--exclude-table-data-and-children
schema	スキーマ	--schema

□ --exclude-extension オプション

指定されたパターン名の拡張モジュールを除外する--exclude-extension オプションが追加されました。[522ed12]

3.4.7. pg_restore

指定されたオブジェクト数を処理した段階でコミットを実行する--transaction-size オプションが追加されました。デフォルトの動作は SQL 単位でコミットされます。このオプションは SQL 文単位のコミットと、--single-transaction オプションによる単一トランザクション処理の中間の動作を提供します。 [][959b38d]

例 95 --transaction-size オプションの指定

```
$ pg_restore --help | grep transaction
  -1, --single-transaction     restore as a single transaction
  --transaction-size=N         commit after every N objects
```

3.4.8. pg_resetwal

--help オプションで出力されるオプションの表示順序が変更されました。[b5da1b3]

例 96 --help オプション

```
$ pg_resetwal --help
pg_resetwal resets the PostgreSQL write-ahead log.

Usage:
  pg_resetwal [OPTION]... DATADIR

Options:
 [-D, --pgdata=]DATADIR  data directory
  -f, --force             force update to be done even after unclean shutdown
or
                          if pg_control values had to be guessed
  -n, --dry-run           no update, just show what would be done
  -V, --version           output version information, then exit
  -?, --help              show this help, then exit

Options to override control file values:
  -c, --commit-timestamp-ids=XID,XID
                                  set oldest and newest transactions bearing
                                  commit timestamp (zero means no change)
  -e, --epoch=XIDEPOCH            set next transaction ID epoch
  -l, --next-wal-file=WALFILE     set minimum starting location for new WAL
  -m, --multixact-ids=MXID,MXID   set next and oldest multitransaction ID
  -o, --next-oid=OID              set next OID
  -O, --multixact-offset=OFFSET   set next multitransaction offset
  -u, --oldest-transaction-id=XID  set oldest transaction ID
  -x, --next-transaction-id=XID   set next transaction ID
      --wal-segsize=SIZE          size of WAL segments, in megabytes

Report bugs to <pgsql-bugs@lists.postgresql.org>.
PostgreSQL home page: <https://www.postgresql.org/>
```

3.4.9. pg_upgrade

pg_upgrade コマンドには以下の機能が追加されました。

□ サブスクライバーの状態保持

サブスクライバーの状態を維持できるようになりました。以前のバージョンではメタデータのみ保存されていました。[9a17be1]

□ --copy-file-range オプション

Linux や FreeBSD 環境で copy_file_range システムコールを使ってファイルのコピーを行います。[d93627b]

3.4.10. pg_walsummary

コマンド pg_walsummary が追加されました。このコマンドは WAL サマリー・ファイルの内容を解析します。

表 34 使用できるオプション

オプション	短縮形	説明
--indivisual	-i	ブロック情報の詳細を出力
--quiet	-q	ファイルのパースのみ実行
--help	-?	使用方法の出力

例 97 pg_walsummary コマンドの実行

```
$ pg_walsummary
        pg_wal/summaries/000000010000000005000030000000000005A829E8.summary
TS 1663, DB 1, REL 1259, FORK main: block 0
TS 1663, DB 1, REL 1259, FORK main: block 3
TS 1663, DB 1, REL 1259, FORK main: blocks 7..8
…
```

3.4.11. pgindent

オプション名が変更されました。[387aecc]

表 35 変更されたオプション

変更前のオプション	変更後のオプション	備考
--show-diff	--diff	
--silent-diff	--check	

3.4.12. psql

psql コマンドには以下の新機能が実装されました。

□ 集約関数の表示

複数引数を持つ集約関数の表示に引数名が追加されました。[b575a26]

例 98 関数情報の表示

```
postgres=> ¥df json_object_agg
List of functions
-[ RECORD 1 ]-------+----------------------
Schema              | pg_catalog
Name                | json_object_agg
Result data type    | json
Argument data types | key "any", value "any"
Type                | agg
```

□ ¥watch メタコマンド

¥watch メタコマンドに最小出力タプル数を示す min_rows（省略形 m）が追加されました。指定されたタプル数を満たさなくなった場合に繰り返し処理を終了します。下記の例では、data1 テーブルから全タプルが削除されたため、¥watch コマンドが終了しています。[f347ec7]

例 99 最小タプル数の指定

```
postgres=> ¥watch 10 min_rows=1
Tue 29 Aug 2023 09:28:17 PM JST (every 10s)

 c1  |  c2
-----+-------
 100 | data1
 200 | data2
(2 rows)

Tue 29 Aug 2023 09:28:27 PM JST (every 10s)

 c1 | c2
----+----
(0 rows)

postgres=>
```

□ ¥sf, ¥ef, ¥sv, ¥ev メタコマンド

行末のセミコロンを無視するようになりました。旧バージョンではエラーが発生していました。[390298f]

例 100 行末セミコロンの無視

```
postgres=> ¥sf func1() ;
CREATE OR REPLACE FUNCTION public.func1()
 RETURNS integer
 LANGUAGE plpgsql
AS $function$
  BEGIN
    RETURN 0;
  END;
$function$
```

3.4.13. reindexdb

reindexdb コマンドには以下の機能が追加されました。

□ オプションの併用

--all と他のオプションを同時に使えるようになりました。[24c928a]

例 101 --all オプションと--schema オプションの指定

```
$ reindexdb --all --schema=public
reindexdb: reindexing database "demodb"
reindexdb: reindexing database "postgres"
reindexdb: reindexing database "template1"
```

□ 並列化

--jobs オプションと--index オプションが同時に使用できるようになりました。異なるテーブルの複数インデックスを同時に処理することができるようになります。[47f99a4]

例 102 --index オプションと--jobs オプションの指定

```
$ reindexdb --jobs=2 --index=idx_data1 --index=idx_data2
```

3.4.14. vacuumdb

--all と他のオプションを同時に使えるようになりました。[648928c]

例 103 --all オプションと--exclude-schema オプション

```
$ vacuumdb --all --exclude-schema=public
vacuumdb: vacuuming database "demodb"
vacuumdb: vacuuming database "postgres"
vacuumdb: vacuuming database "template1"
```

3.4.15. 複数のコマンド

以下のコマンドでファイルのストレージ同期方法を決定するオプション（--sync-method）が追加されました。このオプションは--no-sync オプションが指定された場合には無効になります。--sync-method オプションが追加されたユーティリティは以下の通りです。[8c16ad3, cccc6cd]

- initdb
- pg_basebackup
- pg_checksums
- pg_dump
- pg_rewind
- pg_upgrade

--sync-method オプションに指定できる値は以下の通りです。

表 36 オプションに指定できる値

設定値	説明
fsync	fsync システムコールを使ってファイル単位に再帰的に同期する（デフォルト値）
syncfs	syncfs システムコールを使ってファイルシステム全体を同期する

例 104 オプション--sync-method の指定

```
$ pg_basebackup -D back.1 --sync-method=fsync
$ pg_dump -d postgres -f dump.dat --sync-method=fsync
```

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

bt_index_check 関数に一意制約の整合性をチェックする checkunique パラメーターが追加されました。以下の例は主キー用インデックス data1_pkey に破損が見つかった場合のエラーを示しています。[5ae2087]

例 105 checkunique パラメーターの指定

```
postgres=# SELECT bt_index_check('data1_pkey', true, true) ;
ERROR:  index uniqueness is violated for index "data1_pkey"
DETAIL:    Index tid=(1,84) and tid=(1,85) (point to heap tid=(0,84) and
tid=(0,85)) page lsn=0/1573AB8.
```

pg_amcheck コマンドにも同様の処理を行う--checkunique オプションが追加されました。

例 106 --checkunique オプションの指定

```
$ pg_amcheck --checkunique -d postgres
btree index "postgres.public.data1_pkey":
    ERROR:  index uniqueness is violated for index "data1_pkey"
    DETAIL:   Index tid=(1,84) and tid=(1,85) (point to heap tid=(0,84) and
tid=(0,85)) page lsn=0/1573AB8.
```

3.5.2. pg_buffercache

pg_buffercache モジュールに以下の関数が追加されました。[13453ee]

表 37 追加された関数

関数名	説明
pg_buffercache_evict	バッファプールから任意のブロックを削除する

3.5.3. pg_stat_statements

pg_stat_statements には以下の機能が追加されました。[31de7e6, 638d42a, bb45156 ,5a3423a, 11c34b3]

□ pg_stat_statements ビューの変更

pg_stat_statements ビューには I/O 関連の列が変更され、JIT 関連およびタイムスタンプ関連の列が追加されました。[13d0072, 5147ab1, dc9f8a7]

表 38 追加された列

追加列名	データ型	説明
local_blk_read_time	double precision	ローカル・ブロックの読み込み時間
local_blk_write_time	double precision	ローカル・ブロックの書き込み時間
shared_blk_read_time	double precision	共有ブロックの読み込み時間
shared_blk_write_time	double precision	共有ブロックの書き込み時間
jit_deform_count	bigint	deform 処理回数
jit_deform_time	double precision	deform 処理時間
stats_since	timestamp with time zone	ステートメントの統計取集時刻
minmax_stats_since	timestamp with time zone	ステートメントの最小／最大統計取集時刻

表 39 削除された列

追加列名	説明
blk_read_time	shared_blk_read_time と local_blk_read_time に分割
blk_write_time	shared_blk_write_time と local_blk_write_time に分割

□ 実行文の定数化

SAVEPOINT 文、ROLLBACK TO 文、RELEASE 文、COMMIT PREPARED 文、PREPARE TRANSACTION 文、ROLLBACKT PREPARED 文、DEALLOCATE 文、オーバーロードされた CALL 文はパラメーター記号付きの定数として pg_stat_statements テーブルに保存されます。従来は指定された名前が異なる文は別々に保存されていました。

例 107 SAVEPOINT 文の変換

```
postgres=> BEGIN ;
BEGIN
postgres=*> SAVEPOINT sp1 ;
SAVEPOINT
postgres=*> SAVEPOINT sp2 ;
SAVEPOINT
postgres=*> SAVEPOINT sp3 ;
SAVEPOINT
postgres=*> COMMIT ;
COMMIT
postgres=> SELECT calls, rows, query FROM pg_stat_statements WHERE
                query LIKE 'SAVEPOINT%' ;
 calls | rows |    query
-------+------+-------------
     3 |    0 | SAVEPOINT $1
(1 row)
```

例 108 CALL 文の変換

```
postgres=> CALL overload(1) ;
CALL
postgres=> CALL overload('A') ;
CALL
postgres=> CALL in_out(1, NULL, 1) ;
CALL
postgres=> CALL in_out(2, 1, 2) ;
CALL
postgres=> SELECT calls, rows, query FROM pg_stat_statements ;
 calls | rows |          query
-------+------+--------------------------
     1 |    1 | <insufficient privilege>
     1 |    0 | CALL overload($1)
     1 |    0 | CALL overload($1)
     2 |    0 | CALL in_out($1, $2, $3)
(4 rows)
```

□ pg_stat_statements_reset 関数

pg_stat_statements_reset 関数にパラメーターminmax_only が追加されました。このパラメーターを true に設定すると、最大値／最小値（min_plan_time 列、min_exec_time 列等）の情報をリセットできます。[43cbeda]

例 109 最小／最大値のリセット

```
postgres=# SELECT query, calls, min_exec_time FROM pg_stat_statements
                WHERE queryid=228406429875636958 ;
           query            | calls | min_exec_time
----------------------------+-------+---------------
 SELECT COUNT(*) FROM data1 |     2 |        20.938
(1 row)

postgres=# SELECT pg_stat_statements_reset(0, 0, 228406429875636958, true) ;
   pg_stat_statements_reset
-------------------------------
 2024-03-18 15:35:14.552288+09
(1 row)

postgres=# SELECT query, calls, min_exec_time FROM pg_stat_statements
                WHERE queryid=228406429875636958 ;
           query            | calls | min_exec_time
----------------------------+-------+---------------
 SELECT COUNT(*) FROM data1 |     2 |             0
```

3.5.4. postgres_fdw

リモート・テーブルにアクセスするデフォルトの 1 タプルに対するコストが 0.01 から 0.2 に変更されました。[cac169d]

3.5.5. ltree

一致検索でハッシュ・インデックスを利用できるようになりました。[485f0aa]

例 110 ltree 型に対するハッシュ・インデックスの利用

```
postgres=> ¥d ltree1
              Table "public.ltree1"
 Column | Type  | Collation | Nullable | Default
--------+-------+-----------+----------+---------
 path   | ltree |           |          |
Indexes:
    "path_hash_idx" hash (path)

postgres=> EXPLAIN (COSTS OFF)
        SELECT * FROM ltree1 WHERE path='Top.Hobbies.Amateurs_Astronomy' ;
                            QUERY PLAN
-----------------------------------------------------------------
 Index Scan using path_hash_idx on ltree1
   Index Cond: (path = 'Top.Hobbies.Amateurs_Astronomy'::ltree)
(2 rows)
```

3.5.6. injection_points

拡張モジュール injection_points が src/test/modules/injection_points ディレクトリに追加されました。このモジュールは基本的なインジェクション・ポイントを利用する基盤が提供されています。injection_points モジュールは以下の関数を提供します。[49cd2b9, f587338]

表 40 追加された関数

関数名	説明
injection_points_attach	インジェクション・ポイントの作成
injection_points_detach	インジェクション・ポイントの削除
injection_points_run	インジェクション・ポイントの実行
injection_points_set_local	インジェクション・ポイントの実行を現在のプロセスに限定

3.5.7. test_radixtree

拡張モジュール test_radixtree が src/test/modules/test_radixtree ディレクトリに追加されました。この拡張モジュールは 2023 年に Viktor Leis、Alfons Kemper および Thomas

Neumann により発表された論文「The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases」のテスト実装です。[ee1b30f]

表 41 追加された関数

関数名	戻り値	説明
test_radixtree	void	Radix Tree のテストを実行する

例 111 test_radixtree 拡張の利用

```
postgres=# CREATE EXTENSION test_radixtree ;
CREATE EXTENSION
postgres=# SELECT test_radixtree() ;
NOTICE:  testing node node-4 with shift 0 and ascending keys
NOTICE:  testing node node-4 with shift 0 and descending keys
NOTICE:  testing node node-4 with shift 8 and ascending keys
NOTICE:  testing node node-4 with shift 8 and descending keys
NOTICE:  testing node node-4 with shift 56 and ascending keys
NOTICE:  testing node node-4 with shift 56 and descending keys
NOTICE:  testing node node-16-lo with shift 0 and ascending keys
...
NOTICE:  testing node node-256 with shift 56 and ascending keys
NOTICE:  testing node node-256 with shift 56 and descending keys
 test_radixtree
----------------


(1 row)
```

3.5.8. test_tidstore

拡張モジュール test_tidstore が src/test/modules/test_tidstore ディレクトリに追加されました。test_tidstore モジュールは大規模な TID のセットを効率的に保存するためのテスト・モジュールです。[30e1442]

表 42 追加された関数

関数名	戻り値	説明
test_create	void	テストの作成
check_set_block_offsets	void	ストア内の TID をアレイに対して検証
test_is_full	boolean	メモリーあふれをチェック
test_destroy	void	テストの破棄

3.5.9. xid_wraparound

拡張モジュール xid_wraparound が src/test/modules/xid_wraparound ディレクトリに追加されました。このモジュールはトランザクション ID を強制的に進める関数が提供されています。 [e255b64]

表 43 追加された関数

関数名	戻り値	説明
consume_xids	xid8	トランザクション ID を指定した値進める
consume_xids_until	xid8	トランザクション ID を指定した値まで進める

例 112 トランザクション ID の更新

```
postgres=# CREATE EXTENSION xid_wraparound ;
CREATE EXTENSION
postgres=# SELECT txid_current() ;
 txid_current
--------------
          753
(1 row)

postgres=# SELECT consume_xids('100000000') ;
NOTICE:  consumed 10000055 / 100000000 XIDs, latest 0:10000809
NOTICE:  consumed 20000864 / 100000000 XIDs, latest 0:20001618
NOTICE:  consumed 30001673 / 100000000 XIDs, latest 0:30002427
NOTICE:  consumed 40002482 / 100000000 XIDs, latest 0:40003236
NOTICE:  consumed 50003214 / 100000000 XIDs, latest 0:50003968
NOTICE:  consumed 60003281 / 100000000 XIDs, latest 0:60004035
NOTICE:  consumed 70003982 / 100000000 XIDs, latest 0:70004736
NOTICE:  consumed 80004080 / 100000000 XIDs, latest 0:80004834
NOTICE:  consumed 90004750 / 100000000 XIDs, latest 0:90005504
 consume_xids
--------------
    100000754
(1 row)
```

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/16/release-16.html
- Commitfests
  https://commitfest.postgresql.org/
- PostgreSQL 17 Manual
  https://www.postgresql.org/docs/16/index.html
- PostgreSQL 17 Open Items
  https://wiki.postgresql.org/wiki/PostgreSQL_16_Open_Items
- Git
  git://git.postgresql.org/git/postgresql.git
- GitHub
  https://github.com/postgres/postgres
- PostgreSQL 17 Beta 1 のアナウンス
  https://www.postgresql.org/about/news/postgresql-16-beta-1-released-2643/
- Michael Paquier - PostgreSQL committer
  https://paquier.xyz/
- 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-17.html
- SQL Notes
  https://sql-info.de/postgresql/postgresql-16/articles-about-new-features-in-postgresql-16.html
- Slack - postgresql-jp (Japanese)
  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	2024/04/99	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・パッカード合同会社）
1.0	2024/05/99	篠田典良	PostgreSQL 17 Beta 1 公開版に合わせて修正完了

以上

記述未確定

https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=6dbb490261a6170a3fc3e326c6983ad63e795047
https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=44086b097537a8157ee1dd98d6635b3503f2f534
https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=71e4cc6b8ec6a08f81973bd387fe575134cd0bdf
https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f1889729dd3ab0352dc0ccc2ffcc1b1901f8e39f
https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=562bee0fc13dc95710b8db6a48edad2f3d052f2e
https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=957845789bb97dde2cb1ee11c1a769984131adf6
https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=b588cad688823b1e996ce05af4d88a954c005a3a
https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=7e0ade0ffe0a76b1926a4af39ecdf799c96ef1ba
https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=a65724dfa73db8b451d0c874a9161935a34a914e
https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=cafe1056558fe07cdc52b95205588fcd80870362
https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=ae6d06f09684d8f8a7084514c9b35a274babca61
https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=824dbea3e41efa3b35094163c834988dea7eb139