2025 年 5 月 26 日

PostgreSQL 18 新機能検証結果（Beta 1）

篠田典良

目次

目次 2
1. 本文書について 5
1.1. 本文書の概要 5
1.2. 本文書の対象読者 5
1.3. 本文書の範囲 5
1.4. 本文書の対応バージョン 5
1.5. 本文書に対する質問・意見および責任 6
1.6. 表記 6
2. PostgreSQL 18 における変更点概要 8
2.1. 大規模環境に対応する新機能 8
2.2. 信頼性向上に関する新機能 8
2.3. 運用性向上に関する新機能 8
2.4. プログラミングに関する新機能 9
2.5. 将来の新機能に対する準備 9
2.6. 非互換 10
2.6.1. サポート情報 10
2.6.2. MD5 パスワード 10
2.6.3. configure コマンド 11
2.6.4. initdb コマンド 11
2.6.5. COPY 文 12
2.6.6. CREATE OPERATOR CLASS 文 13
2.6.7. CREATE SUBSCRIPTION 文 14
2.6.8. CREATE TABLE 文 14
2.6.9. EXPLAIN 文 14
2.6.10. GRANT 文 15
2.6.11. psql 16
2.6.12. pg_backend_memory_contexts ビュー 16
3. 新機能解説 18
3.1. アーキテクチャの変更 18
3.1.1. システムカタログの変更 18
3.1.2. ロジカル・レプリケーションの拡張 23
3.1.3. オプティマイザー統計の管理 26
3.1.4. ロールと権限 28
3.1.5. I/O 処理 28

3.1.6. ロケール・プロバイダー 29
3.1.7. OAUTH SASL 30
3.1.8. ハッシュ結合の高速化 30
3.1.9. Libpq 31
3.1.10. アクセスメソッド 32
3.1.11. NUMA 32
3.1.12. 数値計算の高速化 33
3.1.13. 一意キーの利用範囲 33
3.1.14. フック 33
3.2. SQL 文の拡張 34
3.2.1. ALTER DEFAULT PRIVILEGE 文 34
3.2.2. ALTER SUBSCRIPTION 文 34
3.2.3. ALTER TABLE 文 35
3.2.4. ANALYZE 文 36
3.2.5. COPY 文 38
3.2.6. CREATE FOREIGN TABLE 文 39
3.2.7. CREATE INDEX 文 40
3.2.8. CREATE TABLE 文 40
3.2.9. EXPLAIN 文 44
3.2.10. INSERT / UPDATE / DELETE / MERGE 文 48
3.2.11. VACUUM 文 49
3.2.12. データ型 51
3.2.13. 関数 52
3.2.14. PL/pgSQL 62
3.2.15. オプティマイザー 63
3.3. パラメーターの変更 71
3.3.1. 追加されたパラメーター 71
3.3.2. 変更されたパラメーター 73
3.3.3. デフォルト値が変更されたパラメーター 74
3.4. ユーティリティの変更 76
3.4.1. configure 76
3.4.2. initdb 76
3.4.3. pg_combinebackup 77
3.4.4. pg_createsubscriber 77
3.4.5. pg_dump 78
3.4.6. pg_dumpall 79

3.4.7. pg_dump / pg_dumpall / pg_restore コマンド 80
3.4.8. pg_recvlogical 81
3.4.9. pg_resetwal 81
3.4.10. pg_restore 81
3.4.11. pg_rewind 81
3.4.12. pg_upgrade 82
3.4.13. pg_verifybackup 82
3.4.14. psql 82
3.4.15. vacuumdb 89
3.5. Contrib モジュール 90
3.5.1. amcheck 90
3.5.2. bloom 90
3.5.3. dblink 91
3.5.4. file_fdw 92
3.5.5. injection_points 94
3.5.6. isn 94
3.5.7. passwordcheck 94
3.5.8. pgcrypto 95
3.5.9. pg_buffercacahe 97
3.5.10. pg_logicalinspect 98
3.5.11. pg_overexplain 99
3.5.12. pg_stat_statements 100
3.5.13. pgstattuple / pageinspect 103
3.5.14. postgres_fdw 104
3.5.15. その他 106
参考にした URL 107
変更履歴 108

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 18 (18.0) Beta 1 の主な新機能について検証した文書です。

1.2. 本文書の対象読者

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

1.3. 本文書の範囲

本文書は PostgreSQL 17 (17.5) と PostgreSQL 18 Beta 1 (18.0) 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 17.5 (比較対象) PostgreSQL 18 Beta 1 (18.0) (2025/05/05 20:41)
オペレーティング・システム	Red Hat Enterprise Linux 9 Update 5 (x86-64)
Configure オプション	--with-ssl=openssl --with-lz4 --with-zstd --with-llvm --with-libxml --enable-injection-points --with-libnuma --with-liburing

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 PostgreSQL Global Development Group 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@gmail.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=#	SUPERUSER 属性を持つ PostgreSQL ユーザーが利用する psql コマンド・プロンプト
postgres=>	SUPERUSER 属性を持たない PostgreSQL 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。

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 18 における変更点概要

PostgreSQL 18 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。

□ 仮想列の作成

データの保存を伴わない純粋な仮想列を作成できるようになりました。

□ パーティション・テーブルに対する VACUUM / ANALYZE

VACUUM 文、ANALYZE 文にパーティションを再帰的に処理しない ONLY 句が指定できるようになりました。

2.2. 信頼性向上に関する新機能

信頼性を向上させるために以下の拡張が実装されました。

□ pg_dumpall コマンドの出力フォーマット

pg_dumpall コマンドにテキスト以外の出力フォーマットを指定できるようになりました。

□ pg_verifybackup コマンドの tar フォーマット対応

pg_verifybackup コマンドが tar フォーマットのバックアップにも実行できるようになりました。

2.3. 運用性向上に関する新機能

運用性を向上できる以下の機能が追加されました。

□ 統計情報の移行

pg_dump / pg_dumpall コマンドのダンプファイルにオプティマイザー統計情報を含めることができるようになりました。またテーブルや列のオプティマイザー統計を設定・削除する関数が追加されました。

□ OAUTH SASL

OAUTHBearer/SASL による認証をサポートします。

□ EXPLAIN 文

EXPLAIN 文には多くのオプションが追加されました。

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。

□ 更新前データに対するアクセス

更新 DML の RETURNING 句に、更新前データ（OLD）と更新後データ（NEW）を示す識別子を指定できるようになりました。

□ UUID v7 関数

先頭部分が固定された UUID v7 を返す関数が提供されます。

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。

□ 非同期 I/O インフラストラクチャー

非同期 I/O を利用するための基盤が整備されました。パラメーターio_method に使用する非同期 I/O メソッドを指定します。

□ NUMA 対応

基本的な NUMA 環境を利用する関数とビューが追加されました。

2.6. 非互換

PostgreSQL 18 は PostgreSQL 17 から以下の仕様が変更されました。

2.6.1. サポート情報

PostgreSQL 18 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。

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。[edadeb0, 972c2cd, 6c66b74, 45363fc]

- PA-RISC プロセッサ
- LLVM 13 以下
- OpenSSL 1.1.0 以下
- Python 3.2 以下

PostgreSQL 18 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。[a70e01d, 6c66b74]

- OpenSSL 1.1.1 以上が必要
- Python 3.6.8 以上

PostgreSQL 18 では以下のコンポーネントがサポートされるようになりました。[32a2aa7]

- Tcl 9 のサポート追加

2.6.2. MD5 パスワード

MD5 パスワードは非推奨となりました。MD5 パスワードを生成すると、デフォルトでは警告が出力されます。この警告はパラメーターmd5_password_warnings を off に設定することで抑制できます。[db6a4a98]

例 1 MD5 パスワードの警告

```
postgres=# SET password_encryption = md5 ;
SET
postgres=# SHOW md5_password_warnings ;
 md5_password_warnings
-----------------------
 on
(1 row)

postgres=# CREATE USER user1 PASSWORD 'user1' ;
WARNING:  setting an MD5-encrypted password
DETAIL:  MD5 password support is deprecated and will be removed in a future release of PostgreSQL.
HINT:  Refer to the PostgreSQL documentation for details about migrating to another password type.
CREATE ROLE
```

2.6.3. configure コマンド

configure コマンドは以下のように変更されました。

□ --disable-spin-locks オプション
本オプションは削除されました。[e256266]

□ --disable-atomics オプション
本オプションは削除されました。[8138526]

□ flex コマンド
バージョンチェックが削除されました。[0869ea4]

2.6.4. initdb コマンド

データのチェックサム機能がデフォルトで有効になります。チェックサムを無効化するためには--no-data-checksums オプションを指定します。[04bec89]

例 2 チェックサムのデフォルト有効化

```
$ initdb data
The files belonging to this database system will be owned by user "postgres".
...
The default text search configuration will be set to "simple".

Data page checksums are enabled.
...
```

2.6.5. COPY 文

COPY 文は以下の仕様が変更されました。

□ 外部テーブルに対する COPY FREEZE 文の禁止

外部テーブルに対する COPY FREEZE 文は明示的に禁止されるようになりました。[401a695]

例 3 外部テーブルに対する COPY FREEZE 文の禁止

```
postgres=# CREATE FOREIGN TABLE data1(col1 INT, col2 VARCHAR(10))
                SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=# COPY data1 FROM stdin FREEZE ;
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.
>> 1,data1
>> ¥.
ERROR:  cannot perform COPY FREEZE on a foreign table
```

□ CSV モードの STDIN 以外からの入力仕様

CSV モードで COPY 文実行しデータを STDIN 以外から読み込む場合、「¥.」はデータの完了とはみなされなくなりました。[7702337]

例 4 CSV ファイル data1.csv の内容

```
Data1
Data2
¥.
Data3
```

例 5 PostgreSQL 17 の動作

```
postgres=> COPY data11 FROM '/home/postgres/data1.csv' CSV ;
COPY 2
postgres=> SELECT * FROM copy1 ;
  col1
-------
 Data1
 Data2
(2 rows)
```

例 6 PostgreSQL 18 の動作

```
postgres=> COPY data1 FROM '/home/postgres/data1.csv' CSV ;
COPY 4
postgres=> SELECT * FROM data1 ;
  col1
-------
 Data1
 Data2
 ¥.
 Data3
(4 rows)
```

2.6.6. CREATE OPERATOR CLASS 文

CREATE OPERATOR CLASS 文と ALTER OPERATOR FAMILY 文の RECHECK 句は使用できなくなりました。これらの文の RECHECK 句は PostgreSQL 8.4 以前は有効であり、PostgreSQL 17 までは指定してもエラーにはなりませんでした。[7da1bdc]

2.6.7. CREATE SUBSCRIPTION 文

SUBSCRIPTION 作成時の streaming オプションのデフォルト値が off から parallel に変更されました。[1bf1140]

例 7 SUBSCRIPTION 作成時のデフォルト変更

```
postgres=# CREATE SUBSCRIPTION sub1 CONNECTION 'host=dbsvr1 port=5432
                dbname=postgres' PUBLICATION pub1 ;
NOTICE:  created replication slot "sub1" on publisher
CREATE SUBSCRIPTION
postgres=# SELECT subname, substream FROM pg_subscription ;
 subname | substream
---------+-----------
 sub1    | p
(1 row)
```

2.6.8. CREATE TABLE 文

パーティション・テーブルを UNLOGGED テーブルとして作成できなくなりました。パーティションは UNLOGGED テーブルとして作成できます。[e2bab2d]

例 8 UNLOGGED パーティション・テーブルの作成エラー

```
postgres=> CREATE UNLOGGED TABLE part1(col1 INT, col2 VARCHAR(10))
                PARTITION BY RANGE(col1) ;
ERROR:  partitioned tables cannot be unlogged
postgres=> CREATE TABLE part1(col1 INT, col2 VARCHAR(10)) PARTITION BY
                RANGE(col1) ;
CREATE TABLE
postgres=> CREATE UNLOGGED TABLE part1v1 PARTITION OF part1 FOR VALUES
                FROM (0) TO (10) PARTITION BY LIST(col2) ;
ERROR:  partitioned tables cannot be unlogged
```

2.6.9. EXPLAIN 文

BUFFERS 句がデフォルトで有効になりました。[c2a4078]

例 9 PostgreSQL 17 の動作

```
postgres=> EXPLAIN ANALYZE SELECT * FROM data1 WHERE col1=1000 ;
                            QUERY PLAN
-------------------------------------------------------------------------------…
 Index Scan using data1_pkey on data1  (cost=0.42..8.44 rows=1 width=10) (act…
   Index Cond: (col1 = 1000)
 Planning Time: 0.062 ms
 Execution Time: 0.018 ms
(4 rows)
```

例 10 PostgreSQL 18 の動作

```
postgres=> EXPLAIN ANALYZE SELECT * FROM data1 WHERE col1=1000 ;
                             QUERY PLAN
-------------------------------------------------------------------------------…
 Index Scan using data1_pkey on data1  (cost=0.42..8.44 rows=1 width=10) (act…
   Index Cond: (col1 = 1000)
   Buffers: shared hit=4
 Planning Time: 0.065 ms
 Execution Time: 0.023 ms
(5 rows)
```

2.6.10. GRANT 文

オブジェクトに対する RULE 権限は PostgreSQL 8.2 から削除されていますが、GRANT 文ではエラーになりませんでした。本バージョンから RULE 権限は完全に削除されました。[fefa76f]

例 11 RULE 権限の削除

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 VARCHAR(10)) ;
CREATE TABLE
postgres=> GRANT RULE ON data1 TO PUBLIC ;
ERROR:  unrecognized privilege type "rule"
```

2.6.11. psql

¥conninfo メタコマンドの出力フォーマットが変更されました。[bba2fbc]

例 12 PostgreSQL 17 の psql

```
postgres=> ¥conninfo
You are connected to database "postgres" as user "demo" on host "dbsvr1" (address
"192.168.1.100") at port "5442".
```

例 13 PostgreSQL 18 の psql

```
postgres=> ¥conninfo
        Connection Information
      Parameter       |     Value
----------------------+---------------
 Database             | postgres
 Client User          | demo
 Host                 | dbsvr1
 Host Address         | 192.168.1.100
 Server Port          | 5432
 Options              |
 Protocol Version     | 3
 Password Used        | true
 GSSAPI Authenticated | false
 Backend PID          | 160057
 TLS Connection       | false
 Superuser            | off
 Hot Standby          | off
(13 rows)
```

2.6.12. pg_backend_memory_contexts ビュー

pg_backend_memory_contexts ビューの level 列の開始番号が 0 から 1 に変更されました。pg_log_backend_memory_contexts 関数の出力結果も同様の変更が実装されています。[d9e0386]

例 14 PostgreSQL 17 の pg_backend_memory_contexts ビュー

```
postgres=# SELECT level,name FROM pg_backend_memory_contexts order by 1
               LIMIT 2 ;
 level |         name
-------+----------------------
     0 | TopMemoryContext
     1 | TopTransactionContext
(2 rows)
```

例 15 PostgreSQL 18 の pg_backend_memory_contexts ビュー

```
postgres=# SELECT level,name FROM pg_backend_memory_contexts order by 1
               LIMIT 2 ;
 level |                      name
-------+-------------------------------------------------
     1 | TopMemoryContext
     2 | Map from relid to OID of cached composite type
(2 rows)
```

3. 新機能解説

3.1. アーキテクチャの変更

PostgreSQL 18 で拡張されたアーキテクチャや機能について解説しています。

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。[12227a1, 32d3ed8, f0d1127, 559efce, 17cc5f6, 7054186, e7a9496, 02a8d0c, a0f1fce, ca87c41, f92c854, 30a6ed0, 75eb976, a051e71, 83ea6c5, bb8dff9, ac0e331, 2421e9a, 99f8f3f, 60f566b, 89f908a, 73eba50, 8cc139b, 0d6c477]

表 4 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_aios	使用中の非同期 I/O ハンドル
pg_shmem_allocations_numa	共有メモリーの NUMA ノード間による分散状況

表 5 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_backend_memory_contexts	type	text	コンテキストのタイプ
	path	integer[]	コンテキストの階層を示す配列
pg_class	relallfrozen	integer	すべて凍結と指定されたページ数
pg_constraint	conenforced	boolean	制約が ENFORCED 指定されているか
	conperiod	boolean	WITHOUT OVERLAPS または PERIOD の指定
pg_publication	pubgencols	boolean	生成列の値をレプリケーションするか
pg_replication_slots	two_phase_at	pg_lsn	プリペアトランザクションのデコードを有効にする LSN
pg_stat_{all|user|sys}_tables	total_vacuum_time	double precision	VACUUM 実行の合計時間

カタログ／ビュー名	追加列名	データ型	説明
	total_autovacuum_time	double precision	自動 VACUUM 実行の合計時間
	total_analyze_time	double precision	ANALYZE 実行の合計時間
	total_autoanalyze_time	double precision	自動 ANALYZE 実行の合計時間
pg_stat_checkpointer	num_done	bigint	実行完了したチェックポイント回数
	slru_written	bigint	チェックポイント中に書き込まれた SLRU バッファの数
pg_stat_database	parallel_workers_to_launch	bigint	起動しようとしたパラレルワーカー数
	parallel_workers_launched	bigint	起動したパラレルワーカー数
pg_stat_io	read_bytes	numeric	読み込みバイト数
	write_bytes	numeric	書き込みバイト数
	extend_bytes	numeric	拡張バイト数
pg_stat_progress_analyze	delay_time	double precision	コストベース遅延のためスリープした時間
pg_stat_progress_vacuum	delay_time	double precision	コストベース遅延のためスリープした時間
pg_stat_subscription_stats	confl_update_origin_differs	bigint	UPDATE 文実行中に他のソースにより変更されたタプル数
	confl_update_exists	bigint	UPDATE 文実行中に一意制約に違反したタプル数
	confl_update_missing	bigint	UPDATE 文実行中に変更対象として見つからなかったタプル数
	confl_delete_origin_differs	bigint	DELETE 文実行中に他のソースにより変更されたタプル数
	confl_delete_missing	bigint	DELETE 文実行中に削除対象として見つからなかったタプル数
	confl_multiple_unique_conflicts	bigint	複数タプルの INSERT 文実行時に一意制約に違反したタプル数

カタログ／ビュー名	追加列名	データ型	説明
	confl_insert_exists	bigint	INSERT 文の実行時に一意制約に違反したタプル数

表 6 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_backend_memory_contexts	parent	path 列の追加により不要になったため
pg_attribute	attcacheoff	キャッシュ構造の変更により不要になったため
pg_stat_wal	wal_write	pg_stat_io ビューへ統合のため
	wal_sync	pg_stat_io ビューへ統合のため
	wal_write_time	pg_stat_io ビューへ統合のため
	wal_sync_time	pg_stat_io ビューへ統合のため
pg_stat_io	op_bytes	固定値だったため

表 7 内容が変更された information_schema スキーマ内のビュー

カタログ／ビュー名	列名	説明
table_constraints	enforced	列設定によって YES または NO が出力（従来は YES のみ）

表 8 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_aggregate	ユーザー定義型に対する max/min 関数を含む
pg_attribute	attgenerated 列に仮想列を示す v が出力
pg_class	pg_index カタログの TOAST テーブルの情報を出力
pg_constraint	列の NOT NULL 制約の情報を含む
pg_default_acl	defaclobjtype 列に L (Large Object) が出力
pg_proc	ユーザー定義型に対する max/min 関数を含む
pg_replication_slots	invalidation_reason 列に idle_timeout が出力
pg_stat_io	WAL 情報が追加。object='wal'条件で検索可能

□ pg_aios ビュー

pg_aios ビューは現在使用されている非同期 I/O ハンドルの状況を出力します。[60f566b]

表 9 pg_aios ビューの構造

列名	データ型	説明
pid	integer	I/O を行っているサーバー・プロセス ID
io_id	integer	I/O ハンドルの ID
io_generation	bigint	I/O ハンドルの世代
state	text	I/O ハンドルの状態
operation	text	I/O ハンドルの操作
off	bigint	I/O 操作のオフセット
length	bigint	I/O 操作の長さ
target	text	I/O 実行先
handle_data_len	smallint	I/O 操作のデータ長
raw_result	integer	低レベルの I/O 結果
result	text	高レベルの I/O 結果
target_desc	text	I/O ターゲットの説明
f_sync	boolean	I/O 実行フラグ
f_localmem	boolean	ローカルメモリーフラグ
f_buffered	boolean	バッファ I/O を示すフラグ

□ pg_shmem_allocations_numa ビュー

このビューは共有メモリーセグメントが NUMA ノード間でどのように分散されているかを示します。[8cc139b]

表 10 pg_shmem_allocations_numa ビューの構造

列名	データ型	説明
name	text	メモリー領域名
numa_node	integer	NUMA ノード番号
size	bigint	サイズ

例 16 pg_shmem_allocations_numa ビューの検索

```
postgres=# SELECT * FROM pg_shmem_allocations_numa ;
                      name                      | numa_node |   size
------------------------------------------------+-----------+----------
 subtransaction                                 |         0 |   270336
 notify                                         |         0 |   139264
 Shared Memory Stats                            |         0 |   319488
 serializable                                   |         0 |   270336
 PROCLOCK hash                                  |         0 |     4096
 FinishedSerializableTransactions               |         0 |     4096
...
```

□ NOT NULL 制約

テーブル列の NOT NULL 制約が pg_constraint カタログに含まれるようになりました。制約名はテーブル名と列名から自動生成されます。NOT NULL 制約の contype 列の値は n が指定されます。[14e87ff]

例 17 NOT NULL 制約のカタログ化

```
postgres=> CREATE TABLE data1(col1 INT NOT NULL, col2 VARCHAR(10)) ;
CREATE TABLE
postgres=> ¥d+ data1
                                              Table "public.data1"
 Column |         Type          | Collation | Nullable | Default | Storage …
--------+-----------------------+-----------+----------+---------+---------…
 col1   | integer               |           | not null |         | plain   …
 col2   | character varying(10) |           |          |         | extended…
Not-null constraints:
    "data1_col1_not_null" NOT NULL "col1"
Access method: heap
postgres=> SELECT conname FROM pg_constraint WHERE contype='n' AND
                conname LIKE 'data1%' ;
      conname
--------------------
 data1_col1_not_null
(1 row)
```

□ pg_stat_{all|user|sys}_tables ビュー

pg_stat_{all|user|sys}_tables ビューに VACUUM 実行時間、ANALYZE 実行時間が追加されました。手動で実行した場合と自動実行された場合がそれぞれ累積されます。[30a6ed0]

例 18 VACUUM / ANALYZE 実行時間の累積

```
postgres=> ANALYZE data1 ;
ANALYZE
postgres=> VACUUM data1 ;
VACUUM
postgres=> SELECT relname, total_vacuum_time, total_autovacuum_time,
         total_analyze_time, total_autoanalyze_time FROM pg_stat_user_tables ;
-[ RECORD 1 ]----------+------
relname                | data1
total_vacuum_time      | 1826
total_autovacuum_time  | 14607
total_analyze_time     | 125
total_autoanalyze_time | 1475
```

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。

□ コンフリクト発生時のログ出力

スタンバイ・データベースに対する更新時に発生したコンフリクトの情報がログに出力されるようになりました。コンフリクトの理由が「conflict=｛理由｝」のフォーマットで出力されます。コンフリクトの理由 update_differ と delete_differ を検知するにはサブスクライバ側のパラメーターtrack_commit_timestamp を on に設定する必要があります（デフォルト値 off）。[9758174, 73eba50]

表 11 出力されるコンフリクトの理由

出力文字列	説明（DETAIL ログ）	レプリケーション
insert_exists	INSERT 文実行したが既に同じキーのタプルが存在した（Key already exists in unique index）	停止
update_differ	UPDATE 文の対象タプルが他のオリジンにより更新された（Updating the row that was modified locally）	継続
update_exists	UPDATE 文によりキーを更新したが同一キーのタプルが存在した（Key already exists in unique index）	停止
update_missing	UPDATE 文を実行したが対象となるタプルが存在しない（Could not find the row to be updated）	継続
delete_differ	DELETE 文の対象タプルが他のオリジンにより更新された（Updating the row that was modified locally）	継続
delete_missing	DELETE 文を実行したが対象となるタプルが存在しない（Could not find the row to be deleted）	継続
multiple_unique_conflicts	複数タプルが一意制約に違反している（Key already exists in unique index）	停止

例 19 INSERT 文実行時のエラーログ

```
ERROR:  conflict detected on relation "public.data1": conflict=insert_exists
DETAIL:  Key already exists in unique index "data1_pkey", modified in transaction 776.
        Key (col1)=(100); existing local tuple (100, standby1); remote tuple (100, primary1).
CONTEXT:  processing remote data for replication origin "pg_24577" during message type "INSERT" for replication target relation "public.data1" in transaction 761, finished at 0/3059248
```

□　生成列の伝播

CREATE PUBLICATION 文のオプションに生成列（GENERATED ALWAYS AS STORED 指定列）の値をレプリケーションするかを決める publish_generated_columns が追加されました。このオプションのデフォルト値は none で、生成列の値は伝播されません。生成列を伝播する場合にはこのオプションに stored を指定します。指定されたオプションの値は pg_publication カタログの pubgencols_type 列に格納されます。[7054186, 745217a, 7054186, e65dbc9, 117f9f3]

例 20 生成列の伝播設定（PUBLICATION 側）

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY,
            col2 INT GENERATED ALWAYS AS (col1*2) STORED, col3 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE PUBLICATION pub1 FOR TABLE data1(col1, col2, col3)
               WITH (publish_generated_columns = stored) ;
CREATE PUBLICATION
postgres=> SELECT pubname, pubgencols FROM pg_publication ;
-[ RECORD 1 ]----
pubname    | pub1
pubgencols | s
postgres=> INSERT INTO data1(col1, col3) VALUES (100, 'data1') ;
INSERT 0 1
```

例 21 生成列の伝播確認（SUBSCRIPTION 側）

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 INT,
               col3 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE SUBSCRIPTION sub1 CONNECTION 'port=5432 dbname=postgres'
        PUBLICATION pub1 ;
NOTICE:  created replication slot "sub1" on publisher
CREATE SUBSCRIPTION
postgres=> SELECT * FROM data1 ;
 col1  | col2  |  col3
-------+-------+-------
   100 |   200 | data1
(1 row)
```

3.1.3. オプティマイザー統計の管理

テーブルと列のオプティマイザー統計を設定・クリアする関数が提供されました。これらの機能により pg_dump コマンド、pg_dumpall コマンド、pg_upgrade コマンドに統計情報の移行を行うオプションが追加されました。コマンドの追加オプションについて「3.4 ユーティリティ」を参照してください。

□ pg_restore_relation_stats 関数

pg_restore_relation_stats 関数はテーブルのオプティマイザイー統計情報を設定します。軽微な問題が発生した場合には false を返します。[d32d146]

構文

```
boolean pg_restore_relation_stats(VARIADIC kwargs "any")
```

例 22 pg_restore_relation_stats 関数の実行

```
postgres=> SELECT * FROM pg_catalog.pg_restore_relation_stats(
        'version', '180000'::integer,
        'schemaname', 'public',
        'relname', 'data1',
        'relpages', '5406'::integer,
        'reltuples', '1e+06'::real,
        'relallvisible', '0'::integer,
        'relallfrozen', '0'::integer) ;
pg_restore_relation_stats
--------------------------
 t
(1 row)
```

□ pg_clear_relation_stats 関数

pg_clear_relation_stats 関数は指定されたテーブルのオプティマイザー統計情報を削除します。この関数を実行すると、pg_class カタログの該当列をリセットします。 [e839c8e, 650ab8a]

構文

```
void pg_clear_relation_stats(schemaname text, relname text)
```

例 23 pg_clear_relation_stats 関数の実行

```
postgres=> SELECT pg_clear_relation_stats('public', 'data1') ;
 pg_clear_relation_stats
-------------------------


(1 row)

postgres=> SELECT relname, reltuples FROM pg_class WHERE relname='data1' ;
 relname | reltuples
---------+-----------
 data1   |        -1
(1 row)
```

□ pg_restore_attribute_stats 関数

pg_restore_attribute_stats 関数は列のオプティマイザー統計情報を設定します。軽微な問題が発生した場合には false を返します。[d32d146]

構文

```
boolean pg_restore_attribute_stats(VARIADIC kwargs "any")
```

例 24 pg_restore_attribute_stats 関数の実行

```
postgres=> SELECT * FROM pg_catalog.pg_restore_attribute_stats(
        'version', '180000'::integer,
        'schemaname', 'public',
        'relname', 'data1',
        'attname', 'col1',
        'inherited', 'f'::boolean,
        'null_frac', '0'::real,
        'avg_width', '4'::integer,
        'n_distinct', '-1'::real) ;
 pg_restore_attribute_stats
----------------------------
 t
(1 row)
```

□ pg_clear_attribute_stats 関数

pg_clear_attribute_stats 関数は指定された列のオプティマイザー統計情報を削除します。この関数を実行すると、pg_stats カタログの該当列をリセットします。[ce207d2, 650ab8a]

構文

```
void pg_clear_attribute_stats(schemaname text, relname text, attname text,
inherited boolean)
```

例 25 pg_clear_attribute_stats 関数の実行

```
postgres=> SELECT pg_clear_attribute_stats('public', 'data1', 'col1', true) ;
-[ RECORD 1 ]-------------+-
pg_clear_attribute_stats |
```

3.1.4. ロールと権限

事前定義ロールとして pg_signal_autovacuum_worker が追加されました。このロールは Autovacuum ワーカープロセスへのシグナル送信を許可します。[ccd3802]

3.1.5. I/O 処理

PostgreSQL 18 ではストレージ I/O に大きな拡張が加えられました。

□ ストリーム I/O の範囲拡大

PostgreSQL 17 で導入されたストリーム I/O の適用範囲が広がりました。以下の場合にストリーム I/O が利用されます。

- テーブルの VACUUM 文 [9256822, c3e775e, 9256822, c3e775e]
- CREATE DATABASE STRATEGY=WAL_LOG 文 [8720a15]
- B-Tree インデックスの VACUUM [c5c239e]
- GiST インデックスの VACUUM [69273b8]
- SP-GiST インデックスの VACUUM [e215166]
- amcheck モジュール [043799f]
- autoprewarm モジュール [d9c7911]

□ 非同期 I/O の提供

非同期 I/O を利用する基盤が整備されました。パラメーターio_method が追加され、非同期 I/O に使用するメソッドを指定できます。このパラメーターのデフォルト値は woker で、I/O ワーカープロセスによる非同期 I/O を実行します。I/O ワーカープロセスの起動数はパラメーターio_workers（デフォルト値 3）で決定されます。Linux の io_uring を使用する場合にはこのパラメーターに io_uring を指定します。[da72269, 55b454d, 247ce06, c325a76]

例 26 I/O ワーカープロセスの確認

```
$ ps -ef | grep 'io worker' | grep -v grep
postgres  148065  148063  0 10:08 ?        00:00:00 postgres: io worker 0
postgres  148066  148063  0 10:08 ?        00:00:00 postgres: io worker 2
postgres  148067  148063  0 10:08 ?        00:00:00 postgres: io worker 1
```

表 12 パラメーターio_method 設定値

設定値	説明	備考
worker	io worker プロセスによる非同期 I/O が行われます。	デフォルト
sync	同期 I/O が行われます。これは従来バージョンと同じ動作です。	
io_uring	非同期 I/O に io_uring を使用します。	

□ プリフェッチ対応

macOS 上で動作する PostgreSQL でプリフェッチが可能になりました。[6654bb9]

3.1.6. ロケール・プロバイダー

builtin ロケール・プロバイダーにロケール PG_UNICODE_FAST が追加されました。このロケールはコードポイントのソート順を使用し、Unicode の完全な大文字／小文字のマッピングに準拠しています。ただし「ß」の小文字化が「ss」になる等、変換によって文字数の変化が発生することがあります。[d3d0983, 286a365]

例 27 PG_UNICODE_FAST ロケールを指定したデータベースクラスタ作成

```
$ initdb --locale-provider=builtin --builtin-locale=PG_UNICODE_FAST
    --encoding=utf8 data

…
The database cluster will be initialized with this locale configuration:
  locale provider:   builtin
  default collation: PG_UNICODE_FAST
…
```

例 28 PG_UNICODE_FAST ロケールを指定したデータベース作成

```
postgres=# CREATE DATABASE fastuni1 LOCALE_PROVIDER builtin
        LOCALE 'PG_UNICODE_FAST' lc_collate 'C' LC_CTYPE 'C' ENCODING 'UTF8'
        TEMPLATE template0 ;
CREATE DATABASE
```

3.1.7. OAUTH SASL

データベースのユーザー認証に OAUTHBearer/SASL を利用できるようになりました。OAUTHBearer/SASL 認証を行うには pg_hba.conf 認証メソッドに oauth を指定します。この認証を利用するためにはクライアントモジュールが libcurl ライブラリを利用するため configure コマンドで--with-libcurl を指定したバイナリが必要です。またパラメーター oauth_validator_libraries にバリデーター用ライブラリを指定します。この指定が無い場合、oauth 認証メソッドによる認証は失敗します。[b3f0be7]

3.1.8. ハッシュ結合の高速化

ハッシュ結合実行時のデフォーム処理が最適化されました。[adf97c1]

例 29 ハッシュ結合時のデフォーム

```
postgres=> EXPLAIN ANALYZE SELECT * FROM data1, data2
                WHERE data1.col1 = data2.col1 ;
                                    QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=32789.00..76628.00 rows=1000000 width=20) …
   Hash Cond: (data2.col1 = data1.col1)
   ->  Seq Scan on data2  (cost=0.00..15440.00 rows=1000000 width=10) …
   ->  Hash  (cost=15406.00..15406.00 rows=1000000 width=10) …
         Buckets: 262144  Batches: 8  Memory Usage: 7179kB
         ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=10) …
 Planning Time: 0.069 ms
 JIT:
   Functions: 10
   Options: Inlining false, Optimization false, Expressions true, …
   Timing: Generation 0.728 ms (Deform 0.395 ms), Inlining 0.000 ms,
Optimization 0.250 ms, Emission 3.996 ms, Total 4.974 ms
 Execution Time: 396.935 ms
(12 rows)
```

3.1.9. Libpq

以下の API が追加・拡張されました。

□ 接続文字列

接続文字列に最大／最小プロトコルバージョン及び SSL 接続ログファイル名が追加されました。対応する環境変数も追加されています。[285613c, 2da74d8]

表 13 接続文字列

接続文字列	環境変数	説明
max_protocol_version	PGMAXPROTOCOLVERSION	最大バージョン
min_protocol_version	PGMINPROTOCOLVERSION	最小バージョン
sslkeylogfile	-	SSL キーログファイル

□ その他の API

以下の API が追加されました。[4b99fed, cdb6b0f]

表 14 追加された関数名

関数	説明
PQservice	現在の接続からサービス名を返す
PQfullProtocolVersion	マイナーバージョンを含むプロトコルのバージョンを返す

3.1.10. アクセスメソッド

インデックスのアクセスメソッドに以下の拡張が追加されました。[56fead4, af4002b, ce62f2f, af4002b]

表 15 アクセスメソッド定義

要素名	説明
amgettreeheight	インデックス・ツリーの高さを計算
amconsistentequality	一貫した等価性セマンティクスを持つ。旧バージョンの amcancrosscompare を分割
amconsistentequality	一貫した順序セマンティクスを持つ。旧バージョンの amcancrosscompare を分割
amcanhash	ハッシュをサポートする

3.1.11. NUMA

基本的な NUMA 環境を認識する API、SQL 関数および pg_shmem_allocations_numa ビューが追加されました。この機能を有効にするためには configure コマンドに--with-libnuma オプションが必要です。[65c298f, 8cc139b]

表 16 追加された API

API	説明
pg_numa_init	NUMA 環境の初期化
pg_numa_get_max_node	最大ノード数の取得
pg_numa_get_pagesize	ページサイズの取得
pg_numa_query_pages	個々のメモリーページの NUMA ノードを識別

例 30 pg_numa_available SQL 関数の実行

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

3.1.12. 数値計算の高速化

以下の基本的な計算及び CRC32C 関数の高速化が実装されています。

- numeric 型の掛け算の高速化 [c4e4422, c4e4422, ca481d3]
- numeric 型の割り算の高速化 [9428c00]
- CRC32C 関数の Intel AVX-512 利用 [3c6e8c1]

3.1.13. 一意キーの利用範囲

従来は外部キー、マテリアライズドビュー、パーティション・キーに使用できるインデックスは B-Tree インデックスのみでしたが、一意性を持つインデックスであれば他のインデックス・アクセス・メソッドでも作成できるようになりました。[bfe21b7, 9d6db8b, f278e1f]

3.1.14. フック

EXPLAIN 文に対する以下のフックが追加されました[fd02bf7, 50ba65e]

- EXPLAIN 文に対するフック explain_per_plan_hook
- ノード毎に実行されるフック explain_per_node_hook
- オプション検証用フック explain_validate_options_hook

3.2. SQL 文の拡張

本セクションでは SQL 文に関係する新機能を説明しています。

3.2.1. ALTER DEFAULT PRIVILEGE 文

ラージオブジェクトに対してデフォルト権限を設定できるようになりました。pg_default_acl カタログの defaclobjtype 列にはラージオブジェクトを示す L が出力されます。[0d6c477]

例 31 LARGE OBJECT のデフォルト権限

```
postgres=> ALTER DEFAULT PRIVILEGES GRANT SELECT ON LARGE OBJECTS TO PUBLIC ;
ALTER DEFAULT PRIVILEGES
```

3.2.2. ALTER SUBSCRIPTION 文

ALTER SUBSCRIPTION 文で two_phase オプションを変更できるようになりました。two_phase オプションを変更するためには SUBSCRIPTION オブジェクトを一旦無効化する必要があります。[1462aad]

例 32 two_phase オプションの変更

```
postgres=# SELECT subname, subtwophasestate FROM pg_subscription ;
 subname | subtwophasestate
---------+------------------
 sub1    | e
(1 row)

postgres=# ALTER SUBSCRIPTION sub1 DISABLE ;
ALTER SUBSCRIPTION
postgres=# ALTER SUBSCRIPTION sub1 SET (two_phase=off) ;
ALTER SUBSCRIPTION
postgres=# SELECT subname, subtwophasestate FROM pg_subscription ;
 subname | subtwophasestate
---------+------------------
 sub1    | d
(1 row)
```

3.2.3. ALTER TABLE 文

ALTER TABLE 文には以下の構文が追加されました。

□ ALTER TABLE ALTER CONSTRAINT

ALTER CONSTRAINT 句に SET INHERIT 句（または SET NO INHERIT 句）が指定できるようになりました。[f4e53e1]

例 33 SET INHERIT 句の指定

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 VARCHAR(10) NOT NULL) ;
CREATE TABLE
postgres=> ALTER TABLE data1 ALTER CONSTRAINT data1_col2_not_null NO INHERIT ;
ALTER TABLE
```

□ ALTER TABLE NOT NULL NOT VALID

ALTER TABLE 文で NOT NULL 制約に NOT VALID 句を指定できるようになりました。[a379061]

例 34 NOT VALID 句の指定

```
postgres=> ALTER TABLE data1 ADD CONSTRAINT nn_data1_col2 NOT NULL
                col2 NOT VALID ;
ALTER TABLE
postgres=> ¥d+ data1
                                                  Table "public.data1"
 Column |         Type          | Collation | Nullable | Default | Storage  …
--------+-----------------------+-----------+----------+---------+--------- …
 col1   | integer               |           | not null |         | plain    …
 col2   | character varying(10) |           | not null |         | extende  …
Indexes:
    "data1_pkey" PRIMARY KEY, btree (col1)
Not-null constraints:
    "data1_col1_not_null" NOT NULL "col1"
    "nn_data1_col2" NOT NULL "col2" NOT VALID
Access method: heap
```

□ ALTER TABLE ONLY DROP CONSTRAINT

パーティション・テーブルに対する ONLY 付きの ALTER TABLE DROP CONSTRAINT 文が実行できるようになりました。以前のバージョンではエラーが発生していました。[4dea33c]

例 35 ONLY 句の指定

```
postgres=> CREATE TABLE part1(c1 INT PRIMARY KEY, c2 INT CHECK(c2 > 0))
                PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES
                FROM (0) TO (1000000) ;
CREATE TABLE
postgres=> ALTER TABLE ONLY part1 DROP CONSTRAINT part1_c2_check ;
ALTER TABLE
```

3.2.4. ANALYZE 文

ANALYZE 文には以下の新機能が実装されました。

□ ONLY 句

従来のバージョンではパーティション・テーブルに対する ANALYZE 文の実行は再帰的にパーティションの統計情報が収集されていました。ONLY 句を指定することで、パーティション・テーブルのみ統計情報を取得することができます。 [62ddf7e]

例 36 ONLY 句の指定

```
postgres=> ANALYZE VERBOSE ONLY part1 ;
INFO:  analyzing "public.part1" inheritance tree
INFO:  "part1v1": scanned 5406 of 5406 pages, containing 500000 live rows and
0 dead rows; 30000 rows in sample, 500000 estimated total rows
INFO:  finished analyzing table "postgres.public.part1"
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 5445 hits, 0 reads, 0 dirtied
WAL usage: 5 records, 0 full page images, 1195 bytes, 0 buffers full
system usage: CPU: user: 0.04 s, system: 0.00 s, elapsed: 0.04 s
ANALYZE
```

□ I/O 統計と WAL 統計の出力

ANALYZE VERBOSE 文の出力に I/O 統計と WAL 統計が含まれるようになりました。I/O 統計はパラメーターlog_autovacuum_min_duration 設定のログに近い情報が出力されます。[4c1b4cd, bb77752]

例 37 ANALYZE VERBOSE 文の出力

```
postgres=> ANALYZE VERBOSE data1 ;
INFO:  analyzing "public.data1"
INFO:  "data1": scanned 5406 of 5406 pages, containing 333333 live rows and
666667 dead rows; 30000 rows in sample, 333333 estimated total rows
INFO:  finished analyzing table "postgres.public.data1"
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 5421 hits, 0 reads, 0 dirtied
WAL usage: 4 records, 0 full page images, 928 bytes, 0 buffers full
system usage: CPU: user: 0.02 s, system: 0.00 s, elapsed: 0.02 s
ANALYZE
```

「INFO:」から始まる行が I/O 統計、「WAL usage:」から始まる行が WAL 統計です。

□ 遅延時間の出力

パラメーターtrack_cost_delay_timing に on が指定されている場合、ANALYZE VERBOSE 文に「delay time」が出力されるようになりました。自動 VACUUM のログにも同様情報が出力されます。[7720082]

例 38 ANALYZE VERBOSE 文の出力

```
postgres=> ANALYZE VERBOSE data1 ;
INFO:  analyzing "public.data1"
INFO:  "data1": scanned 5406 of 5406 pages, containing 500000 live rows and 0
dead rows; 30000 rows in sample, 500000 estimated total rows
INFO:  finished analyzing table "postgres.public.data1"
delay time: 0.000 ms
I/O timings: read: 25.770 ms, write: 0.000 ms
...
WAL usage: 4 records, 1 full page images, 8240 bytes, 0 buffers full
system usage: CPU: user: 0.01 s, system: 0.03 s, elapsed: 0.05 s
ANALYZE
```

□ WAL バッファ・フルの出力

ANALYZE VERBOSE 文実行時に WAL バッファ・フルの回数が出力されるようになりました。[6a8a7ce]

例 39 ANALYZE VERBOSE 文の出力

```
postgres=> ANALYZE VERBOSE data1 ;
INFO:  analyzing "public.data1"
INFO:  "data1": scanned 5406 of 5406 pages, containing 500000 live rows and 0
dead rows; 30000 rows in sample, 500000 estimated total rows
INFO:  finished analyzing table "postgres.public.data1"
avg read rate: 0.000 MB/s, avg write rate: 0.300 MB/s
buffer usage: 5425 hits, 0 reads, 1 dirtied
WAL usage: 3 records, 1 full page images, 3544 bytes, 0 buffers full
system usage: CPU: user: 0.02 s, system: 0.00 s, elapsed: 0.02 s
```

3.2.5. COPY 文

COPY 文には以下の機能が追加されました。

□ マテリアライズドビューからの COPY TO 文

COPY TO 文にマテリアライズドビューを指定できるようになりました。[534874f]

例 40 マテリアライズドビューに対する COPY TO 文

```
postgres=> CREATE MATERIALIZED VIEW mview1 AS SELECT col1, col2 FROM data1 ;
SELECT 1000000
postgres=> COPY mview1 TO '/tmp/data1.csv' CSV ;
COPY 1000000
```

□ reject_limit オプションの追加

COPY FROM 文に reject_limit オプションが追加されました。このオプションにはエラー発生を無視する設定（on_error=ignore）時にエラー発生数の最大値を指定できます。指定された値を超えるエラーが発生した場合、COPY 文は失敗します。[4ac2a9b, a39297e]

例 41 REJECT_LIMIT オプションの指定

```
postgres=> ¥! cat data1.csv
  1,data1
  2,data2
ABC,data3
  4,data4
DEF,data5
postgres=> COPY data1 FROM '/home/postgres/data1.csv' WITH (REJECT_LIMIT 1,
                ON_ERROR ignore , FORMAT csv, LOG_VERBOSITY verbose) ;
NOTICE:  skipping row due to data type incompatibility at line 3 for column
"col1": "ABC"
NOTICE:  skipping row due to data type incompatibility at line 5 for column
"col1": "DEF"
ERROR:  skipped more than REJECT_LIMIT (1) rows due to data type incompatibility
CONTEXT:  COPY data1, line 5, column col1: "DEF"
```

□ log_verbosity オプションの設定値追加

COPY 文で log_verbosity オプションに silent を指定できるようになりました。この値を指定するとログ出力を抑制します。[e7834a1

3.2.6. CREATE FOREIGN TABLE 文

LIKE 句により他のテーブル（または外部テーブル）の列定義をコピーできるようになりました。 [302cf157]

例 42 外部テーブルに対する LIKE 句の指定

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE FOREIGN TABLE data2(LIKE data1 EXCLUDING ALL) SERVER remsvr1 ;
CREATE FOREIGN TABLE
```

INCLUDING / EXCLUDING 句に指定できる値は CREATE TABLE 文と一部異なります。

表 17 INCLUDING / EXCLUDING 句に指定できる値

設定値	CREATE TABLE 文	CREATE FOREIGN TABLE 文
COMMENTS	○	○
COMPRESSION	○	×
CONSTRAINTS	○	○
DEFAULTS	○	○
GENERATED	○	○
IDENTITY	○	×
INDEXES	○	×
STATISTICS	○	○
STORAGE	○	×
ALL	○	○

3.2.7. CREATE INDEX 文

GIN インデックスの作成が B-Tree インデックスや BRIN インデックスと同じようにパラレル処理されるようになりました。[8492feb]

3.2.8. CREATE TABLE 文

CREATE TABLE 文には以下の拡張が実装されました。

□ WITHOUT OVERLAPS 句

主キー制約（PRIMARY KEY）、一意制約（UNIQUE）、外部キー制約（FOREITN KEY）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。主キー制約に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。ALTER TABLE 文で主キーを追加する場合に指定することもできます。WITHOUT OVERLAPSS 句を表すために pg_constraint カタログに conperiod 列が追加されました。ON {UPDATE, DELETE}句はまだサポートされません。[fc0438b, 89f908a]

構文

```
PRIMARY KEY (column_name1, …, column_nameN WITHOUT OVERLAPS)
```

例 43 主キーの WITHOUT OVERLAPS 句

```
postgres=> CREATE TABLE temporal_range1 (
    id int4range,
    valid_at daterange,
    CONSTRAINT temporal_rng_pk PRIMARY KEY (id, valid_at WITHOUT OVERLAPS)
) ;
CREATE TABLE
postgres=> ¥d temporal_range1
            Table "public.temporal_range1"
  Column  |   Type    | Collation | Nullable | Default
----------+-----------+-----------+----------+---------
 id       | int4range |           | not null |
 valid_at | daterange |           | not null |
Indexes:
    "temporal_rng_pk" PRIMARY KEY (id, valid_at WITHOUT OVERLAPS)
```

□ ENFORCED 句

CHECK 制約および外部キー制約に ENFORCED 句または NOT ENFORCED 句を指定できるようになりました。何も指定しないか ENFORCED 句を指定した場合、PostgreSQL は従来と同じ動作で CHECK 制約が適切なタイミングで動作します。NOT ENFORCED 句を指定すると、CHECK 制約は動作しません。この指定を示すために pg_constraint カタログに conenforced 列が追加されました。[ca87c41, eec0040]

例 44 動作しない CHECK 制約

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY,
                col2 INT CHECK (col2 < 10) NOT ENFORCED) ;
CREATE TABLE
postgres=> INSERT INTO data1 VALUES (100, 100) ;
INSERT 0 1
postgres=> SELECT * FROM data1 ;
 col1  | col2
-------+-------
   100 |   100
(1 row)

postgres=> SELECT conenforced FROM pg_constraint
                WHERE conname='data1_col2_check' ;
 conenforced
-------------
 f
(1 row)
```

□ NOT VALID 句

パーティション・テーブルの外部キーとして NOT VALID 句を指定できるようになりました。[b663b94]

例 45 パーティション・テーブルの外部キーに NOT VALID 句

```
postgres=> CREATE TABLE fkdata1(col1 INT PRIMARY KEY, col2 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE TABLE part1(col1 INT PRIMARY KEY, col2 VARCHAR(10))
                PARTITION BY RANGE(col1) ;
CREATE TABLE
postgres=> ALTER TABLE part1 ADD FOREIGN KEY (col1) REFERENCES fkdata1
                NOT VALID ;
ALTER TABLE
```

□ GENERATED ALWAYS VIRTUAL 句

データを保持しない自動計算列が作成できるようになりました。列定義に GENERATED ALWAYS VIRTUAL 句を指定します。従来は実データを保持する GENERATED ALWAYS STORED 句のみがサポートされていました。VIRTUAL 句の指定は pg_attribute 列の attgenerated 列に'v'が出力されることで確認できます。VIRTUAL 句を指定された生成列にはインデックスを作成できません。[83ea6c5]

例 46 GENERATED ALWAYS VIRTUAL 句

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 INT
               GENERATED ALWAYS AS (col1 * 2) VIRTUAL, col3 VARCHAR(10)) ;
CREATE TABLE
postgres=> ¥d data1
                                Table "public.data1"
 Column |         Type          | Collation | Nullable |           Default
--------+-----------------------+-----------+----------+------------------...
 col1   | integer               |           | not null |
 col2   | integer               |           |          | generated always as
(col1 * 2)
 col3   | character varying(10) |           |          |
Indexes:
    "data1_pkey" PRIMARY KEY, btree (col1)

postgres=> INSERT INTO data1(col1, col3) VALUES (100, 'data1') ;
INSERT 0 1
postgres=> SELECT * FROM data1 ;
 col1  | col2  |  col3
-------+-------+-------
   100 |   200 | data1
(1 row)
```

生成した値が列定義からオーバーフローする場合、INSERT 文や UPDATE 文ではチェックされません。検索時にエラーが発生します。CHECK 制約は INSERT 文や UPDATE 文実行時にチェックされます。

例 47 オーバーフロー時の挙動

```
postgres=> CREATE TABLE overflow1(col1 VARCHAR(3), col2 VARCHAR(3)
                GENERATED ALWAYS AS (col1 || 'X')) ;
CREATE TABLE
postgres=> INSERT INTO overflow1(col1) VALUES ('ABC') ;
INSERT 0 1
postgres=> SELECT * FROM overflow1 ;
ERROR:  value too long for type character varying(3)
```

3.2.9. EXPLAIN 文

EXPLAIN 文には以下の機能が追加されました。

□ 無効化ノードの出力

EXPLAIN 文の出力に無効化されたノード数（Disabled Nodes:）の情報が含まれるようになりました。例えばパラメーターenable_seqscan を false に設定して一部の実行計画を無効にすると、従来はコストを大きくすることで対応していました。PostgreSQL 18 では無効ノードとして明確に認識されるようになりました。 [e222534, c01743a, 161320b]

例 48 ノードの無効化（PostgreSQL 18）

```
postgres=> SET enable_seqscan = off ;
SET
postgres=> EXPLAIN SELECT * FROM data1 ;
                            QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=10)
   Disabled: true
(2 rows)
```

例 49 ノードの無効化（PostgreSQL 17）

```
postgres=> SET enable_seqscan = off ;
SET
postgres=> EXPLAIN SELECT * FROM data1 ;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on data1  (cost=10000000000.00..10000015406.00 rows=1000000 width=10)
(1 row)
```

例 50 src/backend/optimizer/path/costsize.c（PostgreSQL 17）

```
cost_seqscan(Path *path, PlannerInfo *root,
                         RelOptInfo *baserel, ParamPathInfo *param_info)
{
...
        if (!enable_seqscan)
                startup_cost += disable_cost;
...
```

□ Storage 句の出力

Window Aggregation と CTE Scan 実行時に使用される一時ストレージの情報が Storage 句として出力されるようになりました。[95d6e9a, 40708ac]

例 51 Windows Aggregation 一時ストレージ情報

```
postgres=> EXPLAIN (ANALYZE, COSTS OFF) SELECT COUNT(*) OVER() FROM data1 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 WindowAgg (actual time=130.172..189.102 rows=1000000 loops=1)
   Storage: Disk  Maximum Storage: 9766kB
   ->  Seq Scan on data1 (actual time=0.008..38.947 rows=1000000 loops=1)
 Planning Time: 0.037 ms
 Execution Time: 206.478 ms
(5 rows)
```

例 52 CTE Scan 一時ストレージ情報

```
postgres=> EXPLAIN (ANALYZE, COSTS OFF) WITH cte1 AS MATERIALIZED (SELECT col1,
col2 FROM data1) SELECT * FROM cte1 WHERE col1 < 10000 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 CTE Scan on cte1 (actual time=0.012..115.005 rows=9999 loops=1)
   Filter: (col1 < 10000)
   Rows Removed by Filter: 990001
   Storage: Disk  Maximum Storage: 19528kB
   CTE cte1
     ->  Seq Scan on data1 (actual time=0.009..34.816 rows=1000000 loops=1)
 Planning Time: 0.064 ms
 Execution Time: 116.189 ms
(8 rows)
```

□ Parallel Bitmap Heap Scan の統計

Parallel Bitmap Heap Scan 実行時の統計情報が各ワーカーのノードに出力されるようになりました。[5a1e6df]

例 53 Parallel Bitmap Heap Scan 詳細情報

```
postgres=> EXPLAIN (COSTS OFF, ANALYZE, VERBOSE) SELECT AVG(col1) FROM data1
                WHERE col1 BETWEEN 1000000 AND 2000000 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Finalize Aggregate (actual time=98.946..99.991 rows=1 loops=1)
   Output: avg(col1)
   Buffers: shared hit=8125 read=16
   ->  Gather (actual time=98.801..99.981 rows=3 loops=1)
…
               ->  Parallel Bitmap Heap Scan on public.data1 (actual time=46…
                     Output: col1, col2
                     Recheck Cond: ((data1.col1 >= 1000000) AND (data1.col1 …
                     Heap Blocks: exact=1598
                     Buffers: shared hit=8125 read=16
…
```

□ レコード数出力フォーマット

レコード数の表示（actual rows）が小数点２桁まで出力されるようになりました。[ddb17e3, 95dbd827]

例 54 ループ処理の処理行数出力

```
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT * FROM data1 WHERE col1 < 1000 ;
                                  QUERY PLAN
-------------------------------------------------------------------------------
Index Scan using data1_pkey on public.data1  (cost=0.42..39.77 rows=1048
width=10) (actual time=0.008..0.084 rows=999.00 loops=1)
   Output: col1, col2
   Index Cond: (data1.col1 < 1000)
   Buffers: shared hit=11
 Planning:
   Buffers: shared hit=4
 Planning Time: 0.096 ms
 Execution Time: 0.115 ms
(8 rows)
```

□ インデックス検索

インデックス検索回数を示す「Index Searches」項目が出力されるようになりました。[0fbceae]

例 55 インデックス検索回数

```
postgres=> EXPLAIN (ANALYZE, COSTS OFF) SELECT COUNT(*) FROM data1 WHERE
               col1 IN (100, 1000, 10000) ;
                                      QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate (actual time=0.021..0.021 rows=1.00 loops=1)
   Buffers: shared hit=10
   ->  Index Only Scan using data1_pkey on data1 (actual time=0.015..0.019 …
         Index Cond: (col1 = ANY ('{100,1000,10000}'::integer[]))
         Heap Fetches: 0
         Index Searches: 3
…
```

□ Window 関数の出力

Window 関数使用時に実行計画内に「Window:」句が出力されるようになりました。[8b1b342]

例 56 Window 関数の出力

```
postgres=> EXPLAIN SELECT AVG(col1) OVER (PARTITION BY col2) FROM data1 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 WindowAgg  (cost=130488.33..146238.34 rows=1000000 width=36)
   Window: w1 AS (PARTITION BY col2)
   ->  Sort  (cost=128738.34..131238.34 rows=1000000 width=8)
         Sort Key: col2
         ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=8)
(5 rows)
```

3.2.10. INSERT / UPDATE / DELETE / MERGE 文

更新 DML 文の RETURNING 句に更新前情報を示す OLD 句と更新後情報を示す NEW 句を指定できるようになりました。INSERT 文の更新前データ（OLD）、DELETE 文の更新後データ（NEW）は NULL になります。[80feb72]

構文

```
RETURNING OLD.column1, NEW.column2
RETURNING OLD.*, NEW.*
RETURNING WITH (OLD AS alias1, NEW AS alias2) alias1.column1, …
```

例 57 RETURNING 句の新旧データ出力

```
postgres=> UPDATE data1 SET col2='after' WHERE col1=100
                RETURNING WITH (OLD AS o, NEW AS n) o.col1, o.col2, n.col2 ;
 col1  |   col2   |  col2
-------+----------+--------
 100   | before   | after
(1 row)

postgres=> ¥pset null
Null display is "null".

postgres=> DELETE FROM data1 WHERE col1=200 RETURNING OLD.*, NEW.* ;
 col1  |   col2   |  col1  |  col2
-------+----------+--------+--------
   200 | before   | null   | null
(1 row)
```

3.2.11. VACUUM 文

VACUUM 文には以下の新機能が追加されました。

□ ONLY 句

パーティション・テーブルに対して ONLY 句を指定できるようになりました。VACUUM 処理は行われません。[62ddf7e]

例 58 VACUUM ONLY 文の実行

```
postgres=> VACUUM VERBOSE ONLY part1 ;
WARNING:  VACUUM ONLY of partitioned table "part1" has no effect
VACUUM
```

□ VERBOSE/ログ出力

パラメーターtrack_io_timing に on が指定されている場合、ANALYZE VERBOSE 文に「delay time」が出力されるようになりました。自動 VACUUM のログにも同様の情報が出力されます。[7720082]

例 59 VACUUM VERBOSE 文の実行

```
postgres=> VACUUM VERBOSE data1 ;
INFO:  vacuuming "postgres.public.data1"
INFO:  finished vacuuming "postgres.public.data1": index scans: 0
pages: 0 removed, 5406 remain, 1 scanned (0.02% of total), 0 eagerly scanned
tuples: 0 removed, 500000 remain, 0 are dead but not yet removable
removable cutoff: 786, which was 0 XIDs old when operation ended
frozen: 0 pages from table (0.00% of total) had 0 tuples frozen
visibility map: 0 pages set all-visible, 0 pages set all-frozen (0 were all-visible)
index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed
delay time: 0.000 ms
I/O timings: read: 0.000 ms, write: 0.000 ms
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 13 hits, 0 reads, 0 dirtied
WAL usage: 0 records, 0 full page images, 0 bytes
system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
VACUUM
```

□ WAL バッファ・フルの出力

VACUUM VERBOSE 文実行時に WAL バッファ・フルの回数が出力されるようになりました。[6a8a7ce]

例 60 VACUUM VERBOSE 文の出力

```
postgres=> VACUUM VERBOSE data1 ;
INFO:  vacuuming "postgres.public.data1"
INFO:  finished vacuuming "postgres.public.data1": index scans: 0
pages: 0 removed, 5406 remain, 1 scanned (0.02% of total), 0 eagerly scanned
tuples: 0 removed, 500000 remain, 0 are dead but not yet removable
removable cutoff: 763, which was 0 XIDs old when operation ended
frozen: 0 pages from table (0.00% of total) had 0 tuples frozen
visibility map: 0 pages set all-visible, 0 pages set all-frozen (0 were all-visible)
index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 13 hits, 0 reads, 0 dirtied
WAL usage: 0 records, 0 full page images, 0 bytes, 0 buffers full
system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
VACUUM
```

□ テーブル終端空きブロックの解放

パラメーターvacuum_truncate が追加されました。このパラメーターはテーブル属性 vacuum_truncate や VACUUM 文の TRUNCATE オプションが設定されていない場合の動作を決定します。[0164a0f]

例 61 vacuum_truncate パラメーターの設定

```
postgres=> SET vacuum_truncate = off ;
SET
postgres=> VACUUM data1 ;
VACUUM
```

3.2.12. データ型

PostgreSQL 18 のデータ型には以下の拡張が実装されました。

□ bytea 型と整数型のキャスト

bytea 型と整数型(int2, int4, int8)のデータを相互にキャストできるようになりました。[6da469b]

例 62 bytea 型と整数型のキャスト

```
postgres=> SELECT '¥x1122334455667788'::bytea::int8 ;
        int8
--------------------
 1234605616436508552
(1 row)

postgres=> SELECT 1234567890::int8::bytea ;
       bytea
--------------------
 ¥x00000000499602d2
(1 row)
```

3.2.13. 関数

以下の関数が追加／拡張されました。

□ array_reverse 関数

配列の最初の次元を反転した配列を返す array_reverse 関数が追加されました。[49d6c7d]

構文

```
anyarray array_reverse(anyarray)
```

例 63 array_reverse 関数の実行

```
postgres=> SELECT array_reverse(ARRAY['AB', 'CD', 'EF']) ;
-[ RECORD 1 ]-+----------
array_reverse | {EF,CD,AB}

postgres=> SELECT array_reverse(ARRAY[[1,2],[3,4],[5,6]]) ;
-[ RECORD 1 ]-+--------------------
array_reverse | {{5,6},{3,4},{1,2}}
```

□ casefold 関数

casefold 関数は lower 関数とほぼ同じですが、コレーションに従って小文字変換を行います。主な目的は大文字／小文字の区別をしない文字列比較を容易にすることです。データベースのエンコードが UTF-8 の場合のみ使用できます。

文字によっては文字長が変わる可能性があります。ロケール PG_UNICODE_FAST を使っている場合、ß (U+00DF) は ss に変換されます。[bfc5992]

例 64 データベースのコレーションが PG_C_UTF8 の場合

```
postgres=> SELECT casefold('Σ') ;
 casefold
----------
 σ
(1 row)
```

例 65 データベースのコレーションが PG_UNICODE_FAST の場合

```
postgres=> SELECT casefold('ß') ;
 casefold
----------
 ss
(1 row)
```

□ crc32 関数

CRC-32 を計算する関数 crc32 と、CRC-32C を計算する関数 crc32c が追加されました。[760162f]

構文

```
bigint crc32(bytea)
bigint crc32c(bytes)
```

例 66 crc32 / crc32c 関数の実行

```
postgres=> SELECT crc32('ABC'::bytea) ;
   crc32
------------
 2743272264
(1 row)

postgres=> SELECT crc32c('ABC'::bytea) ;
   crc32c
------------
 2285480319
(1 row)
```

□ extract 関数

INTERVAL 型に対しても WEEK 句を指定できるようになりました。[6be39d7]

例 67 INTERVAL 型に対する extract 関数の実行

```
postgres=> SELECT EXTRACT(WEEK FROM INTERVAL '13 days 24 hours') ;
 extract
---------
       1
(1 row)
```

□ gamma / lgamma 関数

ガンマ値、対数ガンマ値を求める関数 gamma、lgamma が追加されました。[a3b6dfd]

構文

```
double precision gamma(double precision)
double precision lgamma(double precision)
```

例 68 gamma / lgamma 関数の実行

```
postgres=> SELECT gamma(2.5), lgamma(3.5) ;
       gamma        |       lgamma
--------------------+--------------------
 1.329340388179137 | 1.2009736023470743
(1 row)
```

□ has_largeobject_privilege 関数

ラージオブジェクトに対する権限が存在するかをチェックする has_largeobject_privilege 関数が追加されました。[4eada20]

構文

```
boolean has_largeobject_privilege(name, oid, text)
boolean has_largeobject_privilege(oid, oid, text)
boolean has_largeobject_privilege(oid, text)
```

例 69 has_largeobject_privilege 関数の実行

```
postgres=> SELECT lo_create(1001) ;
-[ RECORD 1 ]---
lo_create | 1001

postgres=> SELECT has_largeobject_privilege(1001, 'SELECT') ;
-[ RECORD 1 ]-------------+--
has_largeobject_privilege | t
```

□ json[b]_strip_nulls 関数

json[b]_strip_nulls 関数に配列の NULL 値を削除するパラメーターstrip_in_arrays が追加されました。デフォルト値は FALSE です。[4603903]

構文

```
json json_strip_nulls(target json, strip_in_arrays boolean DEFAULT false)
jsonb jsonb_strip_nulls(target jsonb, strip_in_arrays boolean DEFAULT false)
```

例 70 json_strip_nulls 関数の実行

```
postgres=> SELECT json_strip_nulls('[1,2,null,4]', true) ;
-[ RECORD 1 ]----+--------
json_strip_nulls | [1,2,4]
```

□ max / min 関数

bytea 型の列に対して max 関数、min 関数が利用できるようになりました。[2d24fd9]

構文

```
bytea max(bytea)
bytea min(bytea)
```

例 71 bytea 型に対する max/min 関数の実行

```
postgres=> CREATE TABLE bytea1(col1 bytea) ;
CREATE TABLE
postgres=> INSERT INTO bytea1 VALUES
                (decode('abcdef', 'hex')), (decode('123456', 'hex')) ;
INSERT 0 2
postgres=> SELECT max(col1), min(col1) FROM bytea1 ;
   max    |   min
----------+----------
 ¥xabcdef | ¥x123456
(1 row)
```

□ reverse 関数

bytea 型に対応するバージョンが追加されました。[0697b23]

構文

```
bytea reverse(bytea)
```

例 72 reverse 関数の実行

```
postgres=> SELECT reverse('¥xabcdef12'::bytea) ;
  reverse
------------
 ¥x12efcdab
(1 row)
```

□ pg_get_acl 関数

データベース・オブジェクトの ACL 設定を取得できる pg_get_acl 関数が追加されました。[4564f1c, d898665]

構文

```
aclitem[] pg_get_acl(classid oid, objid oid, objsubid integer)
```

例 73 pg_get_acl 関数の実行

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 VARCHAR(10)) ;
CREATE TABLE
postgres=> GRANT SELECT, MAINTAIN ON data1 TO user1 ;
GRANT
postgres=> SELECT UNNEST(
  pg_get_acl('pg_class'::regclass, 'data1'::regclass::oid, 0)) ;
       unnest
--------------------
 demo=arwdDxtm/demo
 user1=rm/demo
(2 rows)
```

□ pg_get_sequence_data 関数

シーケンスのタプルを返す pg_get_sequence_data 関数が追加されました。この関数は SELECT 権限が無いシーケンスを指定すると NULL 値が返ります。この関数は主に pg_dump コマンドの最適化で使われることを想定しています。[c8b06bb, bd15b7d, a83a944]

構文

```
record pg_get_sequence_data(sequence_oid regclass)
```

例 74 pg_get_sequence_data 関数の実行

```
postgres=> CREATE SEQUENCE seq1 START WITH 100 ;
CREATE SEQUENCE
postgres=> SELECT NEXTVAL('seq1') ;
 nextval
---------
     100
(1 row)

postgres=> SELECT * FROM pg_get_sequence_data('seq1') ;
 last_value | is_called
------------+-----------
        100 | t
(1 row)
```

□ pg_get_process_memory_contexts 関数

指定したプロセスのメモリー・コンテキスト情報を取得する pg_get_process_memory_contexts 関数が追加されました。[042a662]

構文

```
record pg_get_process_memory_contexts(pid integer, summary boolean, timeout double precision)
```

例 75 pg_get_process_memory_contexts 関数の実行

```
postgres=# SELECT name, total_bytes FROM
 pg_get_process_memory_contexts(pg_backend_pid(), false, 0) ORDER BY path ;
                        name                         | total_bytes
-----------------------------------------------------+-------------
 TopMemoryContext                                    |      140496
 Map from relid to OID of cached composite type      |        8192
 Type information cache                              |       24624
 Record information cache                            |        8192
...
```

□ pg_ls_summariesdir 関数

PostgreSQL 18 に pg_ls_summariesdir 関数が追加されました。この関数は WAL サマリーファイルの一覧を取得できます。この関数は SUPERUSER 属性を持つか pg_maintain ロールを保持しているユーザーのみ実行できます。WAL サマライズ機能が無効の場合はレコードを返しません。[4e1fad3]

構文

```
SETOF record pg_ls_summariesdir()
```

表 18 pg_ls_summariesdir 関数の戻り値の列名と説明

列名	データ型	説明
name	text	WAL サマリーのファイル名
size	bigint	ファイルのバイト数
modification	timestamp with time zone	最終更新日時

例 76 pg_ls_summariesdir 関数の実行

```
postgres=# SELECT * FROM pg_ls_summariesdir() ;
                    name                     | size  |      modification
---------…-----------------------------------+-------+------------------------
 00000001…10000280000000000010B2FC8.summary |  4808 | 2025-05-09 15:42:38+09
 00000001…10B2FC8000000000015416F8.summary |  4994 | 2025-05-09 15:42:38+09
 00000001…15416F8000000000015417F8.summary |    56 | 2025-05-09 15:42:38+09
 00000001…15417F8000000000015481A0.summary |   230 | 2025-05-09 15:42:38+09
 00000001…15481A0000000000015482A0.summary |    56 | 2025-05-09 15:42:38+09
(5 rows)
```

□ pg_stat_get_backend_wal 関数

指定したバックエンド・プロセスの WAL 出力情報を取得する pg_stat_get_backend_wal 関数が追加されました。この関数実行結果は pg_stat_wal ビューとほぼ同じ内容をバックエンド・プロセス単位に取得できます。[76def4c]

構文

```
record pg_stat_get_backend_wal(backend_pid integer)
```

表 19 pg_stat_get_backend_wal 関数の戻り値の列名と説明

列名	データ型	説明
wal_records	bigint	WAL レコード数
wal_fpi	bigint	WAL フルページイメージ数
wal_bytes	numeric	WAL バイト数
wal_buffers_full	bigint	WAL バッファ・フルによるストレージ出力数
stats_reset	timestamp with time zone	データ・リセット時刻

例 77 pg_stat_get_backend_wal 関数の実行

```
postgres=> SELECT * FROM pg_stat_get_backend_wal(pg_backend_pid()) ;
 wal_records | wal_fpi | wal_bytes | wal_buffers_full | stats_reset
-------------+---------+-----------+------------------+-------------
     4005580 |      28 | 262315709 |            28954 |
(1 row)
```

□　正規表現関数

正規表現関数（regexp*）に引数名が追加されました。[580f872]

例 78 regexp_count 関数の引数名

```
postgres=> ¥df regexp_count
                                        List of functions
   Schema   |     Name     | Result data type |                 Argument data
------------+--------------+------------------+-------------------------------
 pg_catalog | regexp_count | integer          | string text, pattern text   …
 pg_catalog | regexp_count | integer          | string text, pattern text,  …
 pg_catalog | regexp_count | integer          | string text, pattern text,  …
(3 rows)
```

□　to_number 関数と to_char 関数

to_number 関数と to_char 関数のフォーマット指定文字にローマ数字を示す RN が指定できるようになりました。入力文字列やフォーマット指定文字の大文字／小文字は無視されます。[172e6b3]

例 79 to_number 関数にローマ数字を指定

```
postgres=> SELECT to_number('MMXXV', 'rn'), to_char(1560, 'RN') ;
 to_number |     to_char
-----------+-----------------
      2025 |            MDLX
(1 row)
```

□　uuidv7／uuidv4 関数

UUID v7 をサポートする uuidv7 関数と UUID v4 をサポートする uuidv4 関数が追加されました。UUID v7 は先頭部分に関数実行時のタイムスタンプを指定することでインデックス格納ページのランダム化を抑える効果が期待できます。[78c5e14]

例 80 uuidv7 関数と uuidv4 関数の実行

```
postgres=> SELECT uuidv7() FROM generate_series(1, 5) ;
                uuidv7
--------------------------------------
 0193b90c-0818-772e-8fe2-ea74b2446cb2
 0193b90c-0818-775d-b20b-4551f631cd81
 0193b90c-0818-7761-a496-972341cb066c
 0193b90c-0818-7763-b66a-511858d491d7
 0193b90c-0818-7766-b1b9-7a5c2d05ee74
(5 rows)

postgres=> SELECT uuidv4() ;
                uuidv4
--------------------------------------
 9aaf5cc9-893e-46da-9690-dda84405ffe8
(1 row)
```

3.2.14. PL/pgSQL

名前付きカーソルのパラメーターの指定にオペレーター「=>」を指定できるようになりました。以前はオペレーター「:=」を使っていました。[246dedc]

例 81 名前付きカーソルのパラメーター指定

```
CREATE FUNCTION func1(INTEGER) RETURNS TEXT AS $$
DECLARE
  cur1 CURSOR (v INTEGER) FOR SELECT col2 FROM data1 WHERE col1=v;
  val1 TEXT;
BEGIN
  OPEN cur1 (v => $1);
  FETCH cur1 INTO val1;
  RETURN val1;
END;
$$ LANGUAGE plpgsql;
```

3.2.15. オプティマイザー

より高速な実行計画が選択されるようになりました。

□ OR 条件の ANY 変換

インデックスが作成された列に対する OR 条件は ANY 句に自動変換されるようになりました。[d4378c0, ae45691]

例 82 PostgreSQL 17 の実行計画

```
postgres=> EXPLAIN SELECT * FROM data1 WHERE col1=1000 OR col1=2000 ;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Bitmap Heap Scan on data1  (cost=8.87..16.78 rows=2 width=10)
   Recheck Cond: ((col1 = 1000) OR (col1 = 2000))
   ->  BitmapOr  (cost=8.87..8.87 rows=2 width=0)
         ->  Bitmap Index Scan on data1_pkey  (cost=0.00..4.43 rows=1 width=0)
               Index Cond: (col1 = 1000)
         ->  Bitmap Index Scan on data1_pkey  (cost=0.00..4.43 rows=1 width=0)
               Index Cond: (col1 = 2000)
(7 rows)
```

例 83 PostgreSQL 18 の実行計画

```
postgres=> EXPLAIN SELECT * FROM data1 WHERE col1=1000 OR col1=2000 ;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using data1_pkey on data1  (cost=0.42..12.88 rows=2 width=10)
   Index Cond: (col1 = ANY ('{1000,2000}'::integer[]))
(2 rows)
```

□ 結合 OR 条件の ANY 変換

結合条件に OR が使用されている場合に ANY 句に変換できるようになりました。[627d634]

例 84 PostgreSQL 17 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT COUNT(*) FROM data1 AS d1, data2 AS d2
                WHERE d1.col1 = d2.col1 OR d1.col1 = d2.col2 ;
                              QUERY PLAN
-----------------------------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Workers Planned: 2
         ->  Partial Aggregate
               ->  Nested Loop
                     ->  Parallel Seq Scan on data1 d1
                     ->  Bitmap Heap Scan on data2 d2
                           Recheck Cond: ((d1.col1 = col1) OR (d1.con1 = col2))
                           ->  BitmapOr
                                 ->  Bitmap Index Scan on data2_pkey
                                       Index Cond: (col1 = d1.col1)
                                 ->  Bitmap Index Scan on idx1_data2
                                       Index Cond: (col2 = d1.col1)
(13 rows)
```

例 85 PostgreSQL 18 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT COUNT(*) FROM data1 AS d1, data2 AS d2
                WHERE d1.col1 = d2.col1 OR d1.col1 = d2.col2 ;
                              QUERY PLAN
-----------------------------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Workers Planned: 2
         ->  Partial Aggregate
               ->  Nested Loop
                     ->  Parallel Seq Scan on data2 d2
                     ->  Index Only Scan using data1_pkey on data1 d1
                           Index Cond: (col1 = ANY (ARRAY[d2.col1, d2.col2]))
(8 rows)
```

□ IN (VALUES)句の ANY 変換

IN (VALUES) を使った WHERE 句は ANY 句に変換されます。[c0962a1]

例 86 PostgreSQL 17 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1
                WHERE col1 IN (VALUES(100), (200)) ;
                  QUERY PLAN
------------------------------------------
 Nested Loop
   ->  Unique
         ->  Sort
               Sort Key: "*VALUES*".column1
               ->  Values Scan on "*VALUES*"
   ->  Index Scan using data1_pkey on data1
         Index Cond: (col1 = "*VALUES*".column1)
(7 rows)
```

例 87 PostgreSQL 18 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1
                WHERE col1 IN (VALUES(100), (200)) ;
                    QUERY PLAN
-------------------------------------------------
 Index Scan using data1_pkey on data1
   Index Cond: (col1 = ANY ('{100,200}'::integer[]))
(2 rows)
```

□ Right Semi Join

実行計画として Right Semi Join が利用できるようになりました。ハッシュ結合またはマージ結合で採用される可能性があります。[aa86129e]

例 88 Right Semi Join の採用

```
postgres=> EXPLAIN (COSTS OFF, VERBOSE)
  SELECT COUNT(*) FROM tenk1 a WHERE (unique1, two) IN
    (SELECT unique1, ROW_NUMBER() OVER() FROM tenk1 b) ;
                                    QUERY PLAN
------------------------------------------------------------------------------
 Aggregate
   Output: count(*)
   ->  Hash Right Semi Join
         Hash Cond: ((b.unique1 = a.unique1) AND ((row_number() OVER (?))…
         ->  WindowAgg
               Output: b.unique1, row_number() OVER (?)
               ->  Seq Scan on public.tenk1 b
                     Output: b.unique1
         ->  Hash
               Output: a.unique1, a.two
               ->  Seq Scan on public.tenk1 a
                     Output: a.unique1, a.two
(12 rows)
```

□ UNIQUE 制約と GROUP BY 句

NOT NULL 制約と UNIQUE インデックスに指定された列が GROUP BY 句に含まれる場合、対象以外の列を GROUP BY から除外するようになりました。以下の例は col1 列が UNIQUE 制約と NOT NULL 制約が指定されています。[bd10ec5]

例 89 PostgreSQL 17 の実行計画

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

例 90 PostgreSQL 18 の実行計画

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

□ DISTINCT 句の最適化

DISTINCT 句に指定される列の順序は意味が無いため必要に応じて並べ替える最適化が実装されました。この動作を変更するために新しいパラメーター enable_distinct_reordering が追加されました。このパラメーターのデフォルト値は on で最適化が行われます。PostgreSQL 17 の動作に戻す場合にはこのパラメーターを off に設定します。下記の例では col1 列は主キーのため実行計画が変更され、col2 列の処理が無効化されています。[a8ccf4e]

例 91 PostgreSQL 17 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT DISTINCT col2, col1 FROM data1 ;
       QUERY PLAN
--------------------------
 HashAggregate
   Group Key: col2, col1
   ->  Seq Scan on data1
(3 rows)
```

例 92 PostgreSQL 18 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT DISTINCT col2, col1 FROM data1 ;
                    QUERY PLAN
-----------------------------------------------
 Unique
   ->  Incremental Sort
         Sort Key: col1, col2
         Presorted Key: col1
         ->  Index Scan using data1_pkey on data1
(5 rows)
```

□ 自己結合の最適化

結果に影響が無い場合、自己結合処理を結合しないスキャンに置き換える最適化が実装されました。この動作はパラメーターenable_self_join_elimination で制御します。デフォルト値は on で、最適化が行われます。以下の例では data1 テーブルの col1 列が主キーになっています。[fc069a3]

例 93 PostgreSQL 17 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1 d1, data1 d2 WHERE
                d1.col1 = d2.col1 ;
            QUERY PLAN
--------------------------------
 Hash Join
   Hash Cond: (d1.col1 = d2.col1)
   ->  Seq Scan on data1 d1
   ->  Hash
         ->  Seq Scan on data1 d2
(5 rows)
```

例 94 PostgreSQL 18 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1 d1, data1 d2 WHERE
                d1.col1 = d2.col1 ;
      QUERY PLAN
----------------------
 Seq Scan on data1 d2
(1 row)
```

□ Index Skip Scan

複数列から構成されるインデックスに対して、先頭列以外の一致条件が指定された場合に該当インデックスを利用する実行計画が作成されるようになりました。以下の例では col3、col2 列の順で複合インデックスが作成されています。[92fe23d]

例 95 PostgreSQL 17 の実行計画

```
postgres=> EXPLAIN SELECT * FROM data2 WHERE c2=100 ;
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on data2  (cost=0.00..17906.00 rows=1 width=14)
   Filter: (c2 = 100)
(2 rows)
```

例 96 PostgreSQL 18 の実行計画

```
postgres=> EXPLAIN SELECT * FROM data2 WHERE c2=100 ;
                                QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using idx2_data2 on data2  (cost=0.42..12.88 rows=1 width=14)
   Index Cond: (c2 = 100)
(2 rows)
```

□ HAVING 句の WHERE 句変換

GROUPING SETS 句で使用される一部の HAVING 句を WHERE 句に変換する最適化が実装されました。[67a54b9]

例 97 PostgreSQL 17 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT c1, c2, COUNT(c1) FROM group1
        GROUP BY GROUPING SETS ((c1, c2), (c1)) HAVING c1 > 1000 AND c2 > 1000 ;
               QUERY PLAN
----------------------------------------
 GroupAggregate
   Group Key: c1, c2
   Group Key: c1
   Filter: ((c1 > 1000) AND (c2 > 1000))
   ->  Sort
         Sort Key: c1, c2
         ->  Seq Scan on group1
(7 rows)
```

例 98 PostgreSQL 18 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT c1, c2, COUNT(c1) FROM group1
        GROUP BY GROUPING SETS ((c1, c2), (c1)) HAVING c1 > 1000 AND c2 > 1000 ;
            QUERY PLAN
---------------------------------
 GroupAggregate
   Group Key: c1, c2
   Group Key: c1
   Filter: (c2 > 1000)
   ->  Sort
         Sort Key: c1, c2
         ->  Seq Scan on group1
               Filter: (c1 > 1000)
(8 rows)
```

3.3. パラメーターの変更

PostgreSQL 18 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。[0dcaea5, db6a4a9, c758119, a8ccf4e, 306dc52, 052026c, bb8dff9, 45188c2, 3d1ef3a, fc069a3, ac0e331, b3f0be7, 6c349d8, 6d376c3, 0284401, 4f7f7b0, 10f6646, 0164a0f, 247ce06, 55b454d, 62d712e, 9fbd53d, f78ca6f]

表 20 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
autovacuum_worker_slots	autovacuum_max_workers 設定の上限値（postmaster）	16
autovacuum_vacuum_max_threshold	テーブルで VACUUM をトリガーするために必要なタプルの最大数（sighup）	100000000
enable_distinct_reordering	DISTINCT 句に指定された列順序変更最適化の使用有無（user）	on
enable_self_join_elimination	自己結合の最適化機能を有効化する（user）	on
extension_control_path	拡張コントロールファイルのパス（superuser）	$system
file_copy_method	ファイルのコピー方法を指定（user）	copy
idle_replication_slot_timeout	レプリケーションスロットを無効にするアイドル分数（sighup）	0
io_max_combine_limit	io_combine_limit の最大値（postmaster）	16
io_max_concurrency	単一プロセスが実行できる最大同時 I/O 数（postmaster）	64
io_method	非同期 I/O 実行メソッド（postmaster）	worker
io_workers	非同期 I/O ワーカープロセス数（sighup）	3
log_lock_failure	ロック失敗時のログ出力（superuser）	off

パラメーター	説明（context）	デフォルト値
max_active_replication_origins	レプリケーション・オリジンの最大数（postmaster）	10
md5_password_warnings	MD5 パスワードの生成時に警告を出力する（user）	on
num_os_semaphores	サーバーの実行に必要なセマフォ数（internal）	計算値
oauth_validator_libraries	OAuth 接続トークンの検証に使用するライブラリ（sighup）	''
vaccum_truncate	テーブル終端の空ページを解放するか（user）	on
ssl_groups	EDCH 鍵交換で使用する曲線名（sighup）	X25519:prime256v1
ssl_tls13_ciphers	TLS version 1.3 による接続で使用できる暗号スイートのリスト（sighup）	''
track_cost_delay_timing	コストベースの VACUUM 遅延のトラッキング（superuser）	off
vacuum_max_eager_freeze_failure_rate	VACUUM が EAGER スキャンを無効にするページ割合（user）	0.03
vacuum_truncate	テーブル終端の空き領域を解放するか（user）	on

□　自動 VACUUM ワーカープロセス数の上限

autovacuum_max_workers パラメーターの設定が pg_reload_conf 関数の実行により動的に変更できるようになりました。動的に変更する際の上限を示すパラメーター autovacuum_worker_slots（デフォルト 16）が追加されました。autovacuum_max_workers パラメーターの設定は autovacuum_worker_slots よりも大きくすることはできますが、autovacuum_worker_slots よりも多くのワーカープロセスは起動しません。[c758119, 6d01541]

例 99 自動 VACUUM ワーカーの上限

```
postgres=# SELECT name, setting, context FROM pg_settings WHERE name
                LIKE 'autovacuum%worker%' ;
          name           | setting |  context
-------------------------+---------+------------
 autovacuum_max_workers  | 3       | sighup
 autovacuum_worker_slots | 16      | postmaster
(2 rows)
```

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, e48b19c, 7c3fb50, c758119, d4c3a6b, 9219093, 3516ea7, adb5f85]

表 21 変更されたパラメーター

パラメーター	変更内容
autovacuum_max_workers	動的に更新可能に変更（context=sighup）
log_rotation_size	上限値が 2,147,483,647 に変更
log_connections	boolean 型から string 型に変更
log_line_prefix	ローカル IP アドレスを示すエスケープ%L が使用可能に
max_files_per_process	バックエンドに継承されたファイル数を除外
ssl_groups	ssl_ecdh_curve から名称変更

□ パラメーターlog_connections の変更

パラメーターlog_connections は boolean 型から text 型に変更され、ログを出力する内容をカンマ区切りで複数設定できるようになりました。[9219093, 18cd15e]

表 22 パラメーターlog_connections の設定値

設定値	出力内容
''	ログを出力しない（デフォルト値）
receipt	接続を受信した時点でログ出力
authentication	ユーザーを識別する ID をログ出力
authorization	認証が完了したことをログ出力
setup_durations	接続を完了し、最初のクエリーを実行する準備ができた時点までの時間をログ出力
all	receipt,authentication,authorization と同じ

例 100 log_connections パラメーター設定によるログ出力

```
# receipt ログ
LOG:  connection received: host=192.168.1.123 port=37374

# authentication ログ（成功）
LOG:  connection authenticated: identity="demo" method=scram-sha-256
(/data/pg_hba.conf:119)
# authentication ログ（失敗）
FATAL:  password authentication failed for user "demo"
DETAIL:  Connection matched file "/data/pg_hba.conf" line 119: "host     all
all             192.168.1.0/24          scram-sha-256"

# authorization ログ（成功）
LOG:  connection authorized: user=demo database=postgres application_name=psql
# authorization ログ（失敗）
FATAL:  password authentication failed for user "demo"
DETAIL:  Connection matched file "/data/pg_hba.conf" line 119: "host     all
all             192.168.1.0/24          scram-sha-256"

# setup_durations ログ（成功）
LOG:  connection ready: setup total=9.915 ms, fork=0.248 ms,
authentication=6.329 ms
# setup_durations ログ（失敗）
FATAL:  password authentication failed for user "demo"
DETAIL:  Connection matched file " /data/pg_hba.conf" line 119: "host     all
all             192.168.1.0/24          scram-sha-256"
```

従来の設定値である true/false、on/off、0/1 も互換性維持の目的で設定できます。true,0,on の設定は「receipt,authentication,authorization」と同じです。

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, 9219093, ff79b5b, cc6be07, daa02c6]

表 23 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 17	PostgreSQL 18	備考
server_version	17.5	18beta1	
server_version_num	170005	180000	
data_checksums	off	on	
log_connections	off	''	
effective_io_concurrency	1	16	
maintenance_io_concurrency	10	16	
ssl_groups	prime256v1	X25519:prime256v1	名称変更

3.4. ユーティリティの変更

ユーティリティ・コマンドの主な機能拡張点を説明します。

3.4.1. configure

configure コマンドには以下の新機能が追加されました。

□ --with-liburing オプションの追加

Linux で非同期 I/O に io_uring を使用するためのオプション--with-liburing が追加されました。このオプションを有効化するとパラメーターio_method に io_uring を指定できるようになります。[c325a76]

□ --with-libcurl オプションの追加

OAUTHBeare/SASL 機能を利用するためのオプション--with-libcurl オプションが追加されました。[b3f0be7]

□ --with-libnuma オプションの追加

NUMA 環境を検知するためのオプション--with-libnuma が追加されました。[65c298f]

3.4.2. initdb

initdb コマンドには以下のオプションが追加されました。

□ --no-data-checksums オプション

ページチェックサムの生成がデフォルトで有効になりました。この仕様に伴いページチェックサムを生成しないオプション--no-data-checksums が追加されました。チェックサムを生成する--data-checksums オプションと同時に使用した場合は後から指定したオプションが有効になります。[983a588]

例 101 --no-data-checksums オプションの指定

```
$ initdb --no-data-checksums 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 locale "en_US.UTF-8".
The default database encoding has accordingly been set to "UTF8".
initdb: could not find suitable text search configuration for locale "en_US.UTF-8"
The default text search configuration will be set to "simple".

Data page checksums are disabled.

creating directory data ... ok
...
```

□ --no-sync-data-files オプション

新規オプション--no-sync-data-files が追加されました。このオプションを指定するとデータベースクラスタ内のディレクトリやファイルの同期をスキップします。--no-sync オプションを指定しない限り、pg_wal や pg_xact 等のディレクトリは従来通り同期的に書き込まれます。[cf131fa]

3.4.3. pg_combinebackup

オプション--link（短縮形-k）が追加されました。このオプションはファイルのコピーの代わりにハードリンクを用いてバックアップデータを合成します。[99aeb84]

3.4.4. pg_createsubscriber

pg_createsubscriber コマンドには以下のオプションが追加されました。

□ --all オプション

--all オプション（短縮形-a）は接続するパブリッシャー側の全データベースに対してサブスクリプションを作成することを指示します。[fb2ea12]

□ --enable-two-phase オプション

作成するサブスクリプションに対して two_phase オプションを有効にするためのオプション--enable-two-phase が追加されました（短縮形-T）。[e117cfb]

□ --remove オプション

--remove オプション（短縮形-R）にはサブスクライバから削除するオブジェクトの種類を指定します。現状では publications のみがサポートされています。[e5aeed4]

例 102 オプションの確認

```
$ pg_createsubscriber --help
pg_createsubscriber creates a new logical replica from a standby server.

Usage:
  pg_createsubscriber [OPTION]...

Options:
  -a, --all                       create subscriptions for all databases
except template databases or databases that don't allow connections
  -d, --database=DBNAME           database in which to create a subscription
  -D, --pgdata=DATADIR            location for the subscriber data directory
  -n, --dry-run                   dry run, just show what would be done
  -p, --subscriber-port=PORT      subscriber port number (default 50432)
  -P, --publisher-server=CONNSTR  publisher connection string
  -R, --remove=OBJECTTYPE         remove all objects of the specified type from
specified databases on the subscriber; accepts: publications
      --replication-slot=NAME     replication slot name
<<以下省略>>
```

3.4.5. 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。[9c49f0e, bde2fb7]

表 24 追加されたオプション

追加オプション	説明	備考
--sequence-data	シーケンス情報を出力する	
--with-data	データのダンプ	
--with-schema	スキーマ情報のダンプ	
--with-statistics	統計情報のダンプ	

例 103 --sequence-data オプションの指定

```
$ pg_dump --sequence-data | grep -i sequence
-- Name: seq01; Type: SEQUENCE; Schema: public; Owner: demo
CREATE SEQUENCE public.seq01
ALTER SEQUENCE public.seq01 OWNER TO demo;
-- Name: seq01; Type: SEQUENCE SET; Schema: public; Owner: demo
$
```

3.4.6. pg_dumpall

テキスト以外の出力フォーマットが指定できるようになりました。pg_dump コマンドと同様に--format オプションに出力フォーマットを指定します。[1495eff]

表 25 --format オプションの指定

指定値	指定値（短縮）	説明	備考
custom	c	pg_restore 用カスタム形式	
directory	d	pg_restore 用ディレクトリ形式	
tar	t	tar フォーマット	
plain	p	テキスト形式	デフォルト

例 104 pg_dumpall コマンドの出力フォーマット指定

```
$ pg_dumpall --format=c --file=dump_dir
$ ls dump_dir/
databases  global.dat  map.dat
```

3.4.7. pg_dump / pg_dumpall / pg_restore コマンド

pg_dump コマンド、pg_restore コマンド、pg_dumpall コマンドには統計情報の移行を行うオプションが追加されました。以下のオプションが追加されました。[1fd1bd8, cd3c451, acea3fc]

表 26 追加されたオプション

追加オプション	説明	pg_dump / pg_dumpall	pg_restore
--no-data	データを含まない	○	○
--no-policies	Row Level Security 設定を含まない	○	○
--no-schema	スキーマを含まない	○	○
--no-statistics	統計情報を含まない	○	○
--statistics-only	統計情報のみを含む	○	-
--sequence-data	シーケンスを含む	○	-

例 105 pg_dump コマンドによる統計情報の出力

```
$ pg_dump -t data1 --statistics-only
…
SELECT * FROM pg_catalog.pg_restore_relation_stats(
        'relation', 'public.data1'::regclass,
        'version', '180000'::integer,
        'relpages', '6'::integer,
        'reltuples', '1000'::real,
        'relallvisible', '0'::integer
);
SELECT * FROM pg_catalog.pg_restore_attribute_stats(
        'relation', 'public.data1'::regclass,
        'attname', 'col1'::name,
        'inherited', 'f'::boolean,
        'version', '180000'::integer,
        'null_frac', '0'::float4,
        'avg_width', '4'::integer,
        'n_distinct', '-1'::float4,
        'histogram_bounds',
'{1,10,20,30,40,50,60,70,80,90,100,110,120,130,140,150,160,170,180,190,200,210
…
```

3.4.8. pg_recvlogical

pg_recvlogical コマンドには以下の機能が実装されました。

□ --failover オプション

オプション--failover が追加されました。このオプションを指定すると、レプリケーションスロットに failover オプションが追加されます。このオプションは--create-slot オプションと同時に指定します。[cf2655a]

□ --drop-slot オプション

従来のバージョンでは--drop-slot オプションを使用する場合には--dbname オプションが必須でした。PostgreSQL 18 では--dbname オプションは不要になりました。[c68100a]

3.4.9. pg_resetwal

pg_resetwal コマンドにオプション--char-signedness が追加されました。このオプションは制御ファイル内の char 型の符号設定を変更します。このオプションは pg_upgrade コマンド内で指定されることが想定されており、手動で指定することは考えられていません。[30666d1]

3.4.10. pg_restore

pg_restore コマンドには、pg_dumpall コマンドの変更に対応するため以下のオプションが追加されました。追加された--global-only オプションを指定すると、ロールと表スペースの情報のみがリストアされます。[1495eff]

表 27 追加オプション

オプション	オプション（短縮）	説明
--globals-only	-g	グローバル情報のみリストア
--exclude-database=PATTERN	-	リストア対象外データベース

3.4.11. pg_rewind

pg_rewind の--write-recovery-conf オプションを拡張し、--source-server オプションで指定された場合に生成される primary_conninfo 値にデータベース名（dbname）を含めるようになりました。この変更により sync_replication_slots オプションが有効な場合、巻き戻されたサーバーは設定ファイルを手動で変更することなくプライマリサーバーに接続で

きるようになります。[4ecdd41]

3.4.12. pg_upgrade

pg_upgrade コマンドには以下の新機能が追加されました。

□ --set-char-signedness オプション

pg_upgrade コマンドにオプション--char-signedness が追加されました。このオプションは制御ファイル内の char 型の符号設定を変更します。[1aab680]

□ 統計情報の移行

デフォルト状態で統計情報が移行されるようになりました。統計情報を移行しない場合には--no-statistics オプションを指定します。[1fd1bd8]

□ 内部で実行されるコマンド処理

内部的に実行される pg_dump コマンドと pg_dumpall コマンドにオプション--no-sync が指定されて実行されます。[6e16b1e]

内部的に実行される CREATE DATABASE 文が STORATEGY=FILE_COPY 句を指定して実行されます。[64f34eb]

3.4.13. pg_verifybackup

pg_verifybackup コマンドでは tar フォーマットのバックアップも検証できるようになりました。バックアップ先のフォーマットを--format オプション(短縮形-F)で指定します。このオプションを使用する場合は--no-parse-wal オプションも同時に指定する必要があります。[8dfd312]

例 106 tar フォーマットのバックアップに pg_verifybackup コマンド実行

```
$ pg_basebackup -D back.1 --format=tar
$ pg_verifybackup --format=tar --no-parse-wal back.1
backup successfully verified
```

3.4.14. psql

psql コマンドには以下の新機能が実装されました。

□ --help=variables オプション

コマンド・オプション--help=variables の出力先に xheader_width の説明が追加されました。[768dfd8]

例 107 --help オプションの出力

```
$ psql --help=variables | grep -e header -e full
  unicode_header_linestyle
  xheader_width
    set the maximum width of the header for expanded output
    [full, column, page, integer value]
```

□ プリペア文の定義

プリペア文（Prepared statement）の定義を操作する以下のメタコマンドが追加されました。[d55322b]

表 28 追加されたメタコマンド

メタコマンド	説明	備考
¥parse	プリペア文を作成する	
¥bind_named	パラメーターに値をバインドする	
¥close	プリペア文をクローズする	

例 108 プリペア文の操作

```
postgres=> INSERT INTO data1 VALUES ($1, $2) ¥parse stmt1

postgres=> SELECT name, statement FROM pg_prepared_statements ;
 name  |             statement
-------+-----------------------------------
 stmt1 | INSERT INTO data1 VALUES ($1, $2)
(1 row)

postgres=> ¥bind_named stmt1 100 'data1' ¥g
INSERT 0 1
postgres=> SELECT * FROM data1 ;
 col1  |  col2
-------+-------
 100   | data1
(1 row)

postgres=> ¥close stmt1
```

□ ¥dP+メタコマンド

パーティション・テーブルのアクセスメソッド名（Access method）が出力されるようになりました。[978f38c]

例 109 ¥dP+メタコマンドの出力

```
postgres=> CREATE TABLE part1(col1 INT PRIMARY KEY, col2 VARCHAR(10))
               PARTITION BY RANGE(col1) USING heap ;
CREATE TABLE
postgres=> ¥dP+
                                 List of partitioned relations
 Schema |    Name     | Owner |       Type        | Table | Access method |…
--------+-------------+-------+-------------------+-------+---------------+…
 public | part1       | demo  | partitioned table |       | heap          |…
 public | part1_pkey | demo  | partitioned index | part1 | btree         |…
(2 rows)
```

□ ¥df+, ¥do+, ¥dAo+, ¥dC+メタコマンド

各コマンドの出力にリークプルーフかどうかを示す「Leakproof?」列が出力されるようになりました。[2355e51]

例 110 ¥dAo+メタコマンドの出力

```
postgres=# ¥dAo+
                          List of operators of operator families
   AM   |      Operator family      | … | Sort opfamily | Leakproof?
--------+---------------------------+ … +---------------+------------
 brin   | bit_minmax_ops            | … |               | yes
 brin   | bit_minmax_ops            | … |               | yes
 brin   | bit_minmax_ops            | … |               | yes
…
```

□ パイプライン

パイプラインをサポートする以下のメタコマンドが追加されました。[41625ab, 3ce3575, 17caf66]

表 29 追加されたメタコマンド

メタコマンド	説明	備考
¥startpipeline	パイプラインの開始	
¥sendpipeline	拡張クエリーの送信	
¥syncpipeline	パイプラインの同期	
¥endpipeling	パイプラインの終了	
¥flushrequest	Flush をリクエスト	
¥flush	未送信データの送信	
¥getresults	実行結果の取得	取得数の指定可能

上記コマンドの実行結果を示す以下の変数が利用できます。

表 30 追加された変数

変数	説明	備考
PIPELINE_SYNC_COUNT	同期されたパイプ数	
PIPELINE_COMMAND_COUNT	パイプ化されたコマンド数	
PIPELINE_RESULT_COUNT	実行結果数	

例 111 パイプラインの実行

```
postgres=> ¥startpipeline
postgres=> SELECT * FROM data1 WHERE col1=$1 ¥bind 10 ¥sendpipeline
postgres=*> SELECT * FROM data1 WHERE col1=$1 ¥bind 20 ¥sendpipeline
postgres=*> ¥echo :PIPELINE_COMMAND_COUNT
2
postgres=*> ¥syncpipeline
postgres=*> ¥echo :PIPELINE_SYNC_COUNT
1
postgres=*> ¥flushrequest
postgres=*> ¥echo :PIPELINE_RESULT_COUNT
2
postgres=*> ¥getresults 2
 col1 |  col2
------+--------
   10 | data1
(1 row)

 col1 |  col2
------+--------
   20 | data2
(1 row)

postgres=> ¥endpipeline
```

□ 拡張テーブル形式モードの出力

いくつかのメタコマンドに x を指定することで実行結果を拡張テーブル形式モードで出力します。¥d*メタコマンド以外に¥l, ¥z, ¥lo でも指定できます。[00f4c29]

例 112 拡張テーブル形式モード出力

```
postgres=> ¥df bitand
                          List of functions
   Schema   |  Name  | Result data type | Argument data types | Type
------------+--------+------------------+---------------------+------
 pg_catalog | bitand | bit              | bit, bit            | func
(1 row)

postgres=> ¥dfx bitand
List of functions
-[ RECORD 1 ]-------+-----------
Schema              | pg_catalog
Name                | bitand
Result data type    | bit
Argument data types | bit, bit
Type                | func
```

□ 変数 SERVICE

接続に使用したサービス名を示す変数 SERVICE が利用できるようになりました。プロンプト文字列にサービス名を含める場合は%s を指定します。[477728b]

例 113 サービス名の出力

```
postgres=> ¥set PROMPT1 '%/[%n][%s]> '
postgres[demo][]> ¥connect "service=pg18"
You are now connected to database "postgres" as user "demo" on host "localhost"
(address "::1") at port "5432".
postgres[demo][pg18]> ¥echo :SERVICE
pg18
postgres[demo][pg18]>
```

□ オブジェクト種別のタイトル

¥dt や¥di 等のオブジェクト一覧メタコマンドのタイトルがオブジェクト種別に応じて変化するようになりました。旧バージョンではすべて「List of relations」でした。[a14707d]

例 114 オブジェクト種別に応じたタイトル

```
postgres=> ¥dt
         List of tables
 Schema | Name  | Type  | Owner
--------+-------+-------+-------
 public | data1 | table | demo
(1 row)

postgres=> ¥di
               List of indexes
 Schema |    Name     | Type  | Owner | Table
--------+-------------+-------+-------+-------
 public | data1_pkey | index | demo  | data1
(1 row)

postgres=> ¥dm
Did not find any materialized views.
postgres=> ¥ds
Did not find any sequences.
```

□ ¥dx メタコマンド

¥dx メタコマンドの出力にエクステンションのデフォルト・バージョン (Default Version) 項目が出勅されるようになりました。[d696406]

例 115 ¥dx メタコマンドの出力

```
postgres=> ¥dx
                              List of installed extensions
   Name   | Version | Default version |   Schema   |Description
----------+---------+-----------------+------------+-------------------------…
 pgcrypto | 1.4     | 1.4             | public     | cryptographic functions
 plpgsql  | 1.0     | 1.0             | pg_catalog | PL/pgSQL  procedural …
(2 rows)
```

3.4.15. vacuumdb

vacuumdb コマンドにはオブジェクト統計情報が欠落している列やインデックスのみ分析を行う--missing-stats-only オプションが追加されました。このオプションは--analyze-only オプションまたは--analyze-in-stages オプションと同時に指定する必要があります。[edba754]

例 116 --missing-stats-only オプションの指定

```
$ vacuumdb --verbose --missing-stats-only --analyze-only
vacuumdb: vacuuming database "postgres"
```

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

GIN インデックスをチェックする gin_index_check 関数が追加されました。[14ffaec]

例 117 GIN インデックスのチェック

```
postgres=# CREATE INDEX idx1_gin1 ON gin1 USING GIN
                (to_tsvector('english', title1)) ;
CREATE INDEX
postgres=# SELECT gin_index_check('idx1_gin1') ;
 gin_index_check
-----------------


(1 row)
```

3.5.2. bloom

利用状況が pg_stat_*_index ビューに格納されるようになりました。[93063e2]

例 118 Bloom インデックスの利用状況

```
postgres=> CREATE INDEX idx1_bloom ON data1 USING bloom (col1) ;
CREATE INDEX
postgres=> EXPLAIN ANALYZE SELECT * FROM data1 WHERE col1=10000 ;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Bitmap Heap Scan on data1  (cost=15348.00..15352.01 rows=1 width=10) …
   Recheck Cond: (col1 = 10000)
   Rows Removed by Index Recheck: 350
   Heap Blocks: exact=341
   ->  Bitmap Index Scan on idx1_bloom  (cost=0.00..15348.00 rows=1 width=0)…
         Index Cond: (col1 = 10000)
 Planning Time: 0.101 ms
 Execution Time: 3.602 ms
(8 rows)

postgres=> SELECT indexrelname, idx_scan, idx_tup_read FROM
                 pg_stat_user_indexes ;
 indexrelname | idx_scan | idx_tup_read
--------------+----------+--------------
 idx1_bloom   |        1 |          351
(1 row)
```

3.5.3. dblink

外部データラッパーdblink_fdw に SCRAM 認証をパススルーするオプション use_scram_passthrough が追加されました。[3642df2]

例 119 SCRAM 認証情報のパススルー

```
postgres=# CREATE SERVER link1 FOREIGN DATA WRAPPER dblink_fdw OPTIONS
(host 'dbsvr1', port '5432', dbname 'demodb', use_scram_passthrough 'true') ;
CREATE SERVER
```

3.5.4. file_fdw

以下のオプションが追加されました。これらのオプションには COPY 文のオプションと同じ値を指定できます。[a1c4c8a, 6c8f670]

表 31 追加されたオプション

オプション	説明
on_error	軽微なエラー発生時の動作を決定します。指定できる値は以下の通りです。 - default: 処理を停止します（デフォルト値） - ignore: 処理を継続します。
log_verbosity	ログ出力レベルを決定します。指定できる値は以下の通りです。 - default: エラー件数を出力します（デフォルト値） - verbose: より詳しいログを出力します。 - silent: ログを出力しません。
reject_limit	許容するデータ型変換エラー・レコード数を指定します。on_error=ignore オプションと同時に指定します。

例 120 file_fdw モジュールの追加オプション（オプション on_error = 'ignore'）

```
postgres=# ¥! cat /tmp/fs1.csv
100,data1
ABC,data2
300,data3
postgres=# CREATE EXTENSION file_fdw ;
CREATE EXTENSION
postgres=# CREATE SERVER fs FOREIGN DATA WRAPPER file_fdw ;
CREATE SERVER
postgres=# CREATE FOREIGN TABLE fs1 (id INT, val TEXT) SERVER fs
        OPTIONS (filename '/tmp/fs1.csv', FORMAT 'csv',
                ON_ERROR 'ignore', REJECT_LIMIT '1') ;
CREATE FOREIGN TABLE
postgres=# SELECT * FROM fs1 ;
NOTICE:  1 row was skipped due to data type incompatibility
 id  |  val
-----+-------
 100 | data1
 300 | data3
(2 rows)
```

例 121 file_fdw モジュールの追加オプション（オプション log_verbosity = 'verbose'）

```
postgres=# CREATE FOREIGN TABLE fs2 (id INT, val TEXT) SERVER fs
  OPTIONS (filename '/tmp/fs1.csv', FORMAT 'csv', ON_ERROR 'ignore',
        REJECT_LIMIT '1', LOG_VERBOSITY 'verbose') ;
CREATE FOREIGN TABLE
postgres=# SELECT * FROM fs2 ;
NOTICE:  skipping row due to data type incompatibility at line 2 for column
"id": "ABC"
NOTICE:  1 row was skipped due to data type incompatibility
 id  |  val
-----+-------
 100 | data1
 300 | data3
(2 rows)
```

3.5.5. injection_points

injection_points モジュールには以下の新機能が追加されました。

□ カスタム・パラメーターの追加

パラメーターinjection_points.stats が追加されました。このパラメーターはモジュール内で累積統計を有効にするかを決定します。デフォルトでは統計情報は無効 (off) になっています。[2e35c67]

□ コールバックの追加

shared_preload_libraries パラメーターに指定されたモジュールの共有メモリーを初期化するコールバックが追加されました。[b2b023a]

3.5.6. isn

isn モジュールに isn.weak パラメーターが追加されました。従来のバージョンでは isn_weak 関数を使用していました。isn_weak 関数は互換性を維持するために残されていますが VOLATILE 設定になっています。[4489044]

3.5.7. passwordcheck

passwordcheck モジュールにパスワード長の最小値を示すパラメーター passwordcheck.min_password_length が追加されました。デフォルト値は 8 です。[f7e1b38]

例 122 パスワード最小長の設定

```
postgres=# SET passwordcheck.min_password_length = 12 ;
SET
postgres=# ALTER ROLE demo PASSWORD 'ABCDEFGHIJK' ;
ERROR:  password is too short
DETAIL:  password must be at least "passwordcheck.min_password_length" (12)
bytes long
```

3.5.8. pgcrypto

pgcrypto モジュールには以下の拡張が追加されました。

□ fips_mode 関数の追加

OpenSSL が FIPS を有効にしているかを示す fips_mode 関数が追加されました。[924d89a]

構文

```
boolean fips_mode()
```

例 123 FIPS_MODE 関数の実行

```
postgres=> SELECT fips_mode() ;
 fips_mode
-----------
 f
(1 row)
```

□ ビルトイン暗号関数の制御

OpenSSL を FIPS モードで利用している場合、FIPS 認定を受けていない暗号実装は使用すべきではありません。このためにビルトイン関数を無効化するパラメーター pgcrypto.builtin_crypto_enabled が追加されました。このパラメーターは SUPERUSER のみ変更できます。[035f99c]

例 124 ビルトイン関数の無効化

```
postgres=# SET pgcrypto.builtin_crypto_enabled = off ;
SET
postgres=# SELECT gen_salt('des') ;
ERROR:  use of built-in crypto functions is disabled
postgres=# SELECT crypt('data1', 'salt1') ;
ERROR:  use of built-in crypto functions is disabled
```

□ AES 暗号化関数に CFB モードの追加

AES 暗号化モードに CFB (Cipher-FeedBack mode)が追加されました。[9ad1b3d]

例 125 CFB モードの追加

```
postgres=> SELECT encrypt('foo', '0123456789', 'aes-cfb') ;
 encrypt
----------
 ¥x6f8ced
(1 row)

postgres=> SELECT encode(decrypt(encrypt('foo', '0123456', 'aes-cfb'),
                '0123456', 'aes-cfb'), 'escape') ;
 encode
--------
 foo
(1 row)
```

□ SHA-2 ベースのハッシュと暗号化

SHA-2 ベースのハッシュアルゴリズム sha256crypt と sha512crypt を利用できるようになりました。[749a9e2]

例 126 SHA-256 / SHA-512 ハッシュと暗号化

```
postgres=> SELECT gen_salt('sha256crypt') ;
            gen_salt
------------------------------
 $5$rounds=5000$yTxNtev7NxPXhErT
(1 row)

postgres=> SELECT gen_salt('sha512crypt') ;
            gen_salt
------------------------------
 $6$rounds=5000$NoKe0.dTtV9RHhBN
(1 row)

postgres=> SELECT crypt('test', gen_salt('sha512crypt')) ;
                                                              crypt
------------------------------------------------------------------------------...
$6$rounds=5000$B6qwrNDTAV6Zq5ZX$D8mfxoa3IO2BUjZWG4fMCs52wcb9wLigkDwfA4ZZ6CE3yW
Mwy5sIZOxLCZI/dctvxQP9Grx7sk.ooQu5GI7hf1
(1 row)
```

3.5.9. pg_buffercacahe

pg_buffercache モジュールには以下の機能が追加されました。

□ pg_buffercache_numa ビューの追加

以下の列を持つ pg_buffercache_numa ビューが追加されました。[ba2a3c2]

表 32 pg_buffercache_numa ビューの構成

列名	データ型	説明
bufferid	integer	バッファ ID
os_page_num	bigint	OS ページ番号
numa_node	integer	NUMA ノード番号

□ 関数の追加

以下の関数が追加されました。[dcf7e16]

表 33 追加された関数

関数名	説明
pg_buffercache_evict_relation	バッファキャッシュから特定のテーブル情報を削除
pg_buffercache_evict_all	バッファキャッシュから全情報を削除

例 127 pg_buffercache_evict_relation 関数の実行

```
postgres=# SELECT * FROM pg_buffercache_evict_relation('data1') ;
 buffers_evicted | buffers_flushed | buffers_skipped
-----------------+-----------------+-----------------
            5410 |            5406 |               0
(1 row)
```

3.5.10. pg_logicalinspect

PostgreSQL 18 に pg_logicalinspect モジュールが追加されました。この Contrib モジュールは論理デコーディング・コンポーネントの検査を行う関数を提供しています。追加された関数には論理デコーディングのスナップショットファイルを指定します。[7cdfeee]

構文

```
record pg_get_logical_snapshot_meta(filename text)
record pg_get_logical_snapshot_info(filename text)
```

例 128 pg_get_logical_snapshot_meta 関数の実行

```
postgres=# SELECT * FROM pg_get_logical_snapshot_meta('0-57004740.snap') ;
-[ RECORD 1 ]--------
magic    | 1369563137
checksum | 1831531145
version  | 6
```

例 129 pg_get_logical_snapshot_info 関数の実行

```
 postgres=# SELECT * FROM pg_get_logical_snapshot_info('0-57004740.snap') ;
-[ RECORD 1 ]------------+-----------
state                    | consistent
xmin                     | 767
xmax                     | 766
start_decoding_at        | 0/570043B8
two_phase_at             | 0/0
initial_xmin_horizon     | 0
building_full_snapshot   | f
in_slot_creation         | f
last_serialized_snapshot | 0/57004708
next_phase_at            | 0
committed_count          | 0
committed_xip            |
catchange_count          | 0
catchange_xip
```

3.5.11. pg_overexplain

PostgreSQL 18 に pg_overexplain モジュールが追加されました。このモジュールをロードすると、EXPLAIN 文に DEBUG オプションを追加できます。EXPLAIN 文の出力をデバッグする際に利用できます。[8d5ceb1, 9f0c36a]

例 130 EXPLAIN (DEBUG)文の実行

```
postgres=# LOAD 'pg_overexplain' ;
LOAD
postgres=# EXPLAIN (DEBUG) SELECT * FROM data1 WHERE c1=1000 ;
                                QUERY PLAN
---------------------------------------------------------------------------
 Index Scan using data1_pkey on data1  (cost=0.42..8.44 rows=1 width=10)
   Index Cond: (c1 = 1000)
   Disabled Nodes: 0
   Parallel Safe: true
   Plan Node ID: 0
 PlannedStmt:
   Command Type: select
   Flags: canSetTag
   Subplans Needing Rewind: none
   Relation OIDs: 16389
   Executor Parameter Types: none
   Parse Location: 16 for 33 bytes
(12 rows)
```

3.5.12. pg_stat_statements

pg_stat_statements モジュールには以下の機能が追加されました。

□ 内部 SQL の取得

CREATE TABLE AS 文と DECLARE CURSOR 文に使われる内部 SQL 文にもクエリー ID を計算できるようになりました。このため pg_stat_statements.track オプションに all を指定した環境では内部 SQL の実行時間を取得できるようになりました。[6b652e6]

例 131 内部 SQL 文のキャプチャ

```
postgres=> SHOW pg_stat_statements.track ;
 pg_stat_statements.track
--------------------------
 all
(1 row)

postgres=> CREATE TABLE data2 AS SELECT * FROM data1 WHERE col1=1000 ;
SELECT 1
postgres=> SELECT queryid, query FROM pg_stat_statements ;
       queryid       |                          query
---------------------+-------------------------------------------------------------
 4147066388115963993 | SHOW pg_stat_statements.track
 3197110067974111441 | SELECT * FROM data1 WHERE col1=$1
 6773306165436199945 | CREATE TABLE data2 AS SELECT * FROM data1 WHERE col1=$1
(3 rows)
```

□ SET 文の定数化

SET 文の設定値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。従来は指定された値が異なる文は別々に保存されていました。定数 DEFAULT や FROM CURRENT を使用した場合は定数化の対象外になります。[dc68515]

例 132 SET 文の定数化

```
postgres=> SET work_mem = '8MB' ;
SET
postgres=> SET work_mem = '16MB' ;
SET
postgres=> SET work_mem = DEFAULT ;
SET
postgres=> SELECT query, calls FROM pg_stat_statements WHERE query LIKE 'SET%' ;
         query          | calls
------------------------+-------
 SET work_mem = DEFAULT |     1
 SET work_mem = $1      |     2
(2 rows)
```

□ 列の追加

pg_stat_statements ビューに以下の列が追加されました。パラレルクエリーの実行情報と WAL バッファがフルになった回数です。 [cf54a2c, ce5bcc4]

表 34 追加された列

列名	データ型	説明
parallel_workers_to_launch	bigint	実行を予定されたパラレルワーカー数
parallel_workers_launched	bigint	実行されたパラレルワーカー数
wal_buffers_full	bigint	WAL バッファがフルになった回数

例 133 追加列の確認（パラレルクエリー情報）

```
postgres=> SELECT COUNT(*) FROM data1 ;
-[ RECORD 1 ]--
count | 1000000

postgres=> SELECT query, parallel_workers_to_launch, parallel_workers_launched
FROM pg_stat_statements WHERE userid=16384 ;
-[ RECORD 1 ]---------------+---------------------------
query                       | SELECT COUNT(*) FROM data1
parallel_workers_to_launch | 4
parallel_workers_launched  | 4
```

□ 定数リストの圧縮

WHERE 句の IN リストの個数が可変の場合、従来のバージョンでは個別の Query ID が割り当てられていました。これは pg_stat_statements ビューに格納されるデータ量が大きくなることになります。PostgreSQL 18 では定数リストを圧縮して単一の Query ID を割り当てます。[62d712e, 9fbd53d]

例として、以下の SQL 文を実行し、pg_stat_statements ビューで確認します。

例 134 実行する SQL 文

```
SELECT * FROM data1 WHERE col1 IN (10, 20) ;
SELECT * FROM data1 WHERE col1 IN (10, 20, 30, 40, 50) ;
```

例 135 PostgreSQL 17 の動作

```
postgres=> SELECT calls, query FROM pg_stat_statements WHERE query LIKE
'%data1%' ;
 calls |                          query
-------+------------------------------------------------------------
     1 | SELECT * FROM data1 WHERE col1 IN ($1, $2)
     1 | SELECT * FROM data1 WHERE col1 IN ($1, $2, $3)
(2 rows)
```

例 136 PostgreSQL 18 の動作

```
postgres=> SELECT calls, query FROM pg_stat_statements WHERE query LIKE
'%data1%' ;
 calls |                       query
-------+-------------------------------------------------
     2 | SELECT * FROM data1 WHERE col1 IN ($1 /*, ... */)
(1 row)
```

3.5.13. pgstattuple / pageinspect

Contrib モジュールの各関数がシーケンスにも対応しました。[05036a3]

例 137 pageinspect モジュール

```
postgres=# CREATE EXTENSION pageinspect ;
CREATE EXTENSION
postgres=# SELECT tuple_data_split('seq1'::regclass, t_data, t_infomask,
t_infomask2, t_bits) FROM heap_page_items(get_raw_page('seq1', 0)) ;
-[ RECORD 1 ]----+-----------------------------------------------------
tuple_data_split | {"¥¥x0100000000000000","¥¥x0000000000000000","¥¥x00"}
```

例 138 pgstattuple モジュール

```
postgres=# CREATE SEQUENCE seq1 ;
CREATE SEQUENCE
postgres=# CREATE EXTENSION pgstattuple ;
CREATE EXTENSION
postgres=# SELECT * FROM pgstattuple('seq1') ;
-[ RECORD 1 ]------+------
table_len          | 8192
tuple_count        | 1
tuple_len          | 41
tuple_percent      | 0.5
dead_tuple_count   | 0
dead_tuple_len     | 0
dead_tuple_percent | 0
free_space         | 8108
free_percent       | 98.97
```

3.5.14. postgres_fdw

postgres_fdw モジュールには以下の機能が追加されました。

□ SCRAM 認証のパススルー

SCRAM 認証情報を自動的に送信することで認証を行うオプション use_scram_passthrough が追加されました。SERVER または USER MAPPING の属性として指定できます。[761c795]

例 139 SCRAM 認証のパススルー設定

```
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw
                OPTIONS (host 'dbsvr1', port '5432', dbname 'postgres') ;
CREATE SERVER
postgres=# CREATE USER MAPPING FOR demo SERVER remsvr1
        OPTIONS (user 'demo', use_scram_passthrough 'true') ;
CREATE USER MAPPING
```

□ postgres_fdw_get_connections 関数の拡張

postgres_fdw_get_connections 関数の戻り値に used_in_xact 列と closed 列が追加されました。また、この関数のパラメーターに接続をチェックする check_conn パラメーター（デフォルト false）が追加されました。[c297a47, 857df3c, 4f08ab5]

表 35 追加された出力列

列名	データ型	説明
user_name	text	接続マッピング・ユーザー名（または public）
used_in_xact	boolean	トランザクション中か
closed	boolean	セッションがクローズされているか

例 140 postgres_fdw_get_connections 関数の実行

```
postgres=> BEGIN ;
BEGIN
postgres=*> INSERT INTO data1 VALUES (100, 'data1') ;
INSERT 0 1
postgres=*> SELECT * FROM postgres_fdw_get_connections(true) ;
 server_name | user_name | valid | used_in_xact | closed | remote_backend_pid
-------------+-----------+-------+--------------+--------+--------------------
 remsvr1     | public    | t     | t            | f      |             309160
(1 row)

postgres=*> COMMIT ;
COMMIT
postgres=> SELECT * FROM postgres_fdw_get_connections(true) ;
 server_name | user_name | valid | used_in_xact | closed | remote_backend_pid
-------------+-----------+-------+--------------+--------+--------------------
 remsvr1     | public    | t     | f            | f      |             309160
(1 row)
```

□ postgres_fdw_get_connections 関数

postgres_fdw_get_connections 関数の戻り値にリモートサーバーのバックエンド・プロセス ID を示す remote_backend_pid 列（int4 型）が追加されました。[fe186bd]

例 141 postgres_fdw_get_connections 関数の実行

```
postgres=> CREATE FOREIGN TABLE data1(col1 INT, col2 VARCHAR(10))
                SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=> SELECT * FROM data1 ;
 col1 | col2
------+------
(0 rows)

postgres=> SELECT * FROM postgres_fdw_get_connections() ;
 server_name | user_name | valid | used_in_xact | closed | remote_backend_pid
-------------+-----------+-------+--------------+--------+--------------------
 remsvr1     | demo      | t     | f            |        |             370760
(1 row)
```

3.5.15. その他

上記で説明した以外に、以下のテスト用拡張モジュールが src/test/modules ディレクトリ以下に追加されました。[93bc3d7, b3f0be7, b85a9d0]

表 36 追加された拡張モジュール

モジュール名	説明	備考
oauth_validator	OAUTHBeare/SASL Validator	
test_aio	非同期 I/O テスト	
typecache	タイプキャッシュのテスト	

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/18/release.html
- Commitfests
  https://commitfest.postgresql.org/
- PostgreSQL 18 Manual
  https://www.postgresql.org/docs/18/index.html
- PostgreSQL 18 Open Items
  https://wiki.postgresql.org/wiki/PostgreSQL_18_Open_Items
- Git
  git://git.postgresql.org/git/postgresql.git
- GitHub
  https://github.com/postgres/postgres
- PostgreSQL 18 Beta 1 のアナウンス
  https://www.postgresql.org/about/news/postgresql-18-beta-1-released-3070/
- 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-18.html
- Slack - postgresql-jp (Japanese)
  https://postgresql-jp.slack.com/

変更履歴

変更履歴

版	日付	作成者	説明
1.0	2025/05/27	篠田典良	PostgreSQL 18 Beta 1 公開版に合わせて修正完了

以上