

2023 年 5 月 28 日

PostgreSQL 16 新機能検証結果 (Beta 1)

日本ヒューレット・パカード合同会社
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 16 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.....	9
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.....	10
2.6. 非互換	10
2.6.1. サポート終了.....	10
2.6.2. プロモーション	11
2.6.3. 起動ユーザー属性.....	11
2.6.4. WAL アーカイブ	11
2.6.5. エクステンション.....	12
2.6.6. ASCII 専用文字列	12
2.6.7. 日付フォーマット.....	13
2.6.8. ALTER DEFAULT PRIVILEGES 文	13
2.6.9. ALTER TABLE 文	14
2.6.10. CREATE RULE 文.....	14
2.6.11. CREATE TABLE 文	14
2.6.12. CREATEROLE 属性	14
2.6.13. POWER 関数	15
2.6.14. RESET 文	16
2.6.15. pg_stat_get_backend_idset 関数.....	16
2.6.16. pg_walinspect エクステンション.....	17
2.6.17. postmaster.....	18
2.6.18. psql.....	18
2.6.19. libpq.....	19



2.6.20. PL/Python.....	19
2.6.21. ECPG.....	19
3. 新機能解説.....	20
3.1. アーキテクチャの変更.....	20
3.1.1. システムカタログの変更.....	20
3.1.2. ロジカル・レプリケーションの拡張.....	23
3.1.3. パラレル・クエリー.....	27
3.1.4. 設定ファイル.....	29
3.1.5. ICU ロケール.....	30
3.1.6. 事前定義ロール.....	33
3.1.7. 待機イベント.....	34
3.1.8. トリガー.....	34
3.1.9. ログ.....	35
3.1.10. Kerberos 資格情報の委任.....	36
3.1.11. libpq.....	36
3.1.12. PL/pgSQL.....	39
3.1.13. MATERIALIZED VIEW.....	39
3.1.14. WAL sender プロセス.....	39
3.1.15. GIN インデックスのコスト.....	40
3.1.16. Meson 対応.....	40
3.1.17. UNICODE.....	40
3.1.18. LLVM.....	40
3.1.19. エクステンション.....	40
3.1.20. WAL 出力量の削減.....	41
3.2. SQL 文の拡張.....	42
3.2.1. データ型.....	42
3.2.2. COPY.....	45
3.2.3. CREATE ROLE/USER.....	46
3.2.4. CREATE STATISTICS.....	47
3.2.5. CREATE TABLE.....	48
3.2.6. EXPLAIN.....	49
3.2.7. GRANT.....	49
3.2.8. JSON 関連.....	52
3.2.9. REINDEX.....	54
3.2.10. VACUUM.....	55
3.2.11. サブクエリー.....	56



3.2.12. 実行計画	57
3.2.13. 関数	61
3.3. パラメーターの変更	70
3.3.1. 追加されたパラメーター	70
3.3.2. 変更されたパラメーター	71
3.3.3. デフォルト値が変更されたパラメーター	72
3.3.4. 削除されたパラメーター	72
3.4. ユーティリティの変更	73
3.4.1. configure	73
3.4.2. createuser	73
3.4.3. initdb	73
3.4.4. pgindent	75
3.4.5. pg_basebackup	75
3.4.6. pg_bsd_indent	76
3.4.7. pg_dump	76
3.4.8. pg_receivewal / pg_recvlogical	78
3.4.9. pg_upgrade	78
3.4.10. pg_verifybackup	78
3.4.11. pg_waldump	79
3.4.12. psql	80
3.4.13. vacuumdb	85
3.5. Contrib モジュール	87
3.5.1. auto_explain	87
3.5.2. fuzzystmatch	87
3.5.3. ltree	88
3.5.4. pageinspect	88
3.5.5. pg_buffercache	88
3.5.6. pg_stat_statements	90
3.5.7. pg_walinspect	90
3.5.8. postgres_fdw	92
参考にした URL	94
変更履歴	95



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 16 (16.0) 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 15 (15.3) と PostgreSQL 16 (16.0) Beta 1 の主な差分を記載しています。原則として利用者が見て変化がわかる機能について調査しています。すべての新機能について記載および検証しているわけではありません。特に以下の新機能は含みません。

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

1.4. 本文書の対応バージョン

本文書は以下のバージョンとプラットフォームを対象として検証を行っています。

表 1 対象バージョン

種別	バージョン
データベース製品	PostgreSQL 15.3 (比較対象) PostgreSQL 16 (16.0) Beta 1 (2023/05/22 21:20:39)
オペレーティング・システム	Red Hat Enterprise Linux 8 Update 5 (x86-64)
Configure オプション	--with-ssl=openssl --with-python --with-lz4 --with-zstd --with-llvm --with-icu --with-libxml

1.5. 本文書に対する質問・意見および責任

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

1.6. 表記

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

表 2 例の表記ルール

表記	説明
#	Linux root ユーザーのプロンプト
\$	Linux 一般ユーザーのプロンプト
太字	ユーザーが入力する文字列
postgres=#	PostgreSQL 管理者が利用する psql コマンド・プロンプト
postgres=>	PostgreSQL 一般ユーザーが利用する psql コマンド・プロンプト
<u>下線部</u>	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 16 における変更点概要

PostgreSQL 16 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。

□ パラレル・クエリーの拡張

FULL OUTER JOIN 構文や RIGHT OUTER JOIN 構文、string_agg 関数、array_agg 関数の実行にパラレル・クエリーが利用できるようになりました。

□ 統計情報の拡張

詳細な I/O 統計を取得できる pg_stat_io ビューが追加されました。また pg_stat_all_tables ビュー、pg_stat_all_indexes ビューにはオブジェクトに対する最終アクセス時刻がわかる列が追加されています。

□ バッチ処理性能の向上

postgres_fdw を使った外部テーブルに対する COPY 文で複数レコードを一括挿入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。

□ ロジカル・レプリケーションの拡張

ロジカル・レプリケーションは本バージョンで大きく拡張された部分です。長時間実行されるトランザクションの適用パフォーマンスの改善、限定的な双方向レプリケーション、SUBSCRIPTION 作成を許可するロール、初期データ転送性能の改善、スタンバイ・インスタンスにおけるロジカル・デコーディングなど多くの機能が追加されました。

□ バックアップの圧縮

pg_dump コマンドの出力に LZ4 圧縮、Zstandard 圧縮が利用できるようになりました。

2.3. 運用性向上に関する新機能

運用性を向上できる以下の機能が追加されました。

☐ パラメータ・ファイルの拡張

pg_hba.conf ファイル、pg_ident.conf ファイルの一部設定項目に正規表現が使えるようになりました。また pg_hba.conf と pg_ident.conf ファイルには postgresql.conf と同じように外部のファイルをインクルードすることができるようになりました。

☐ セキュリティ

GSSAPI/Kerberos 資格情報の委任がサポートされるようになりました。事前定義ロール pg_maintain が追加されました。このロールを付与されたユーザーは所有者以外のテーブルに対しても ANALYZE 文、VACUUM 文、REINDEX 文等を実行できます。またテーブル毎に上記 SQL 文の実行を他のロールに対して権限を付与できるようになりました。一般ユーザー用の予備セッションを確保する pg_use_reserved_connections が追加されました。

2.4. プログラミングに関する新機能

SQL 文に以下の機能が追加されました。

☐ JSON 関連

SQL/JSON を生成する標準関数が追加されました。JSON 構文をチェックできる IS JSON 句が追加されました。

☐ サブクエリー

FROM 句にサブクエリーが記述されている場合、エイリアス名はオプションになりました。

☐ リテラル

数値リテラルに 2 進数、8 進数、16 進数も記述できるようになりました。数値文字列内のアンダースコアがサポートされます。

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。

□ ICU ロケールの拡張

標準のロケール・プロバイダが ICU になりました。また ICU ロケールに独自ルールを追加がサポートされ、UNICODE ロケールも追加されました。

□ スタンバイ・インスタンス

スタンバイ・インスタンスでロジカル・デコーディングが可能になりました。

2.6. 非互換

PostgreSQL 16 は PostgreSQL 15 から以下の仕様が変更されました。

2.6.1. サポート終了

PostgreSQL 16 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。[[9db300c](#), [495ed0e](#), [6203583](#), [8efefa7](#), [8b878bf](#), [b086a47](#), [4c15327](#), [8efefa7](#), [e692727](#)]

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。

- HP-UX
- HP/Intel Itanium プロセッサ
- M68K、M88K、M32R、SuperH プロセッサ
- Windows 10 より前の Microsoft Windows
- Microsoft Visual Studio 2013

PostgreSQL 16 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。

- Bison 2.3 以降
- Flex 2.5.35 以降
- Perl 5.14 以降
- GNU make 3.81 以降

2.6.2. プロモーション

トリガー・ファイルの作成によるスタンバイ・データベースのプロモーション方法はサポートされなくなりました。これに伴いパラメーター `promote_trigger_file` は削除されました。プロモーションを実行するには `pg_ctl promote` コマンドまたは `pg_promote` 関数を使用します。[[cd4329d](#)]

2.6.3. 起動ユーザー属性

ブートストラップ・ユーザーから `SUPERUSER` 属性を削除できなくなりました。[[e530be2](#)]

例 1 ブートストラップ・ユーザーの属性変更

```
postgres=# \du
List of roles
-[ RECORD 1 ]-----
Role name | postgres
Attributes | Superuser, Create role, Create DB, Replication, Bypass RLS
Member of  | {}

postgres=# ALTER USER postgres NOSUPERUSER ;
ERROR:  permission denied to alter role
DETAIL:  The bootstrap user must have the SUPERUSER attribute.
```

2.6.4. WAL アーカイブ

PostgreSQL 16 の WAL アーカイブには以下の非互換があります。[[d627ce3](#), [35739b8](#)]

□ パラメーター設定

パラメーター `archive_command` と `archive_library` は同時に値を設定することができなくなりました。PostgreSQL 15 では `archive_library` が優先されていました。同時に設定するとログに以下のエラーが出力され、WAL アーカイブは出力されなくなります。

例 2 WAL アーカイブ設定の重複エラー

```
$ tail -2 data/log/postgresql-2023-05-25_225124.log
FATAL:  both archive_command and archive_library set
DETAIL:  Only one of archive_command, archive_library may be set.
```

□ WAL アーカイブ・モジュールの仕様変更

パラメーター `archive_module` に指定する共有ライブラリのコールバック関数の仕様が変わりました。各コールバックのパラメーターに `ArchiveModuleState` 型のアドレスが渡されます。

表 4 WAL アーカイブ・モジュールの API

コールバック	変更点	PostgreSQL 16 の呼び出し定義
初期化	追加	(*ArchiveStartupCB) (ArchiveModuleState *state)
チェック	変更	(*ArchiveCheckConfiguredCB) (ArchiveModuleState *state)
WAL アーカイブ	変更	(*ArchiveFileCB) (ArchiveModuleState *state, const char *file, const char *path)
シャットダウン	変更	(*ArchiveShutdownCB) (ArchiveModuleState *state)

2.6.5. エクステンション

エクステンションに属していない既存オブジェクトの `CREATE OR REPLACE` 文の実行は禁止されます。従来はエクステンション・スクリプトが `CREATE OR REPLACE` 文を実行し、エクステンションに属さない既存のオブジェクトが存在する場合、オブジェクトを上書きしていました。 [\[b9b21ac\]](#)

2.6.6. ASCII 専用文字列

パラメーター `application_name` や `cluster_name` 等、ASCII 専用の文字列に対して非 ASCII 文字を指定した場合の変換ルールが変更されました。従来はバイト単位にクエスチオン記号 (?) で変換していましたが、PostgreSQL 16 では 16 進数の文字列に変換されます。 [\[45b1a67\]](#)

表 5 ASCII 変換ルール

変更前 (UTF-8)	PostgreSQL 15	PostgreSQL 16	備考
Abc 漢字	Abc?????	Abc¥xe6¥xbc¥xa2¥xe5¥xad¥x97	

2.6.7. 日付フォーマット

PostgreSQL 16 では「epoch」および「infinity」と他の日時フィールドを組み合わせた日付時刻文字列は禁止されます。また文字列フォーマットに ISO 8601 が厳密に適用されるようになります。[[bcc704b](#), [5b3c595](#)]

例 3 PostgreSQL 16 でエラーが発生する日時表現

```
postgres=> SELECT date '1995-08-06 epoch' ;
ERROR:  invalid input syntax for type date: "1995-08-06 epoch"
LINE 1: SELECT date '1995-08-06 epoch' ;
        ^

postgres=> SELECT timestamp '1995-08-06 infinity' ;
ERROR:  invalid input syntax for type timestamp: "1995-08-06 infinity"
LINE 1: SELECT timestamp '1995-08-06 infinity';

postgres=> SELECT date '1995-08-06 J J J' ;
ERROR:  invalid input syntax for type date: "1995-08-06 J J J"
LINE 1: SELECT date '1995-08-06 J J J' ;
```

2.6.8. ALTER DEFAULT PRIVILEGES 文

ALTER DEFAULT PRIVILEGES 文の実行にはロールの所属ではなく、明示的に権限が必要になります。以下の例に含まれる ALTER DEFAULT PRIVILEGES 文は PostgreSQL 15 では成功します。[[48a257d](#)]

例 4 PostgreSQL 16 の動作

```
postgres=# CREATE USER role1 ;
CREATE ROLE
postgres=# CREATE USER role2 IN ROLE role1 NOINHERIT ;
CREATE ROLE
postgres=# \connect postgres role2
You are now connected to database "postgres" as user "role2".
postgres=> ALTER DEFAULT PRIVILEGES FOR ROLE role1 IN SCHEMA public GRANT SELECT
ON TABLES TO public ;
ERROR:  permission denied to change default privileges
```

2.6.9. ALTER TABLE 文

NULLS NOT DISTINCT 句を指定された一意インデックスはテーブルの主キーに指定できなくなりました。[[d959523](#)]

例 5 ALTER TABLE 文の失敗

```
postgres=> CREATE TABLE pktest1 (c1 INTEGER, c2 INTEGER) ;
CREATE TABLE
postgres=> CREATE UNIQUE INDEX idx1_pktest1 ON pktest1 (c1) NULLS NOT DISTINCT ;
CREATE INDEX
postgres=> ALTER TABLE pktest1 ADD PRIMARY KEY USING INDEX idx1_pktest1 ;
ERROR:  primary keys cannot use NULLS NOT DISTINCT indexes
```

2.6.10. CREATE RULE 文

従来のバージョンではテーブルに対して ON SELECT ルールを作成してビューに変換することができましたが、PostgreSQL 16 では禁止されます。[[b23cd18](#)]

2.6.11. CREATE TABLE 文

システム列は外部キー (FOREIGN KEY) として使用できなくなりました。この変更は旧バージョンへバックポートされます。[[f0d65c0](#)]

例 6 システム列の外部キー指定エラー

```
postgres=> CREATE TABLE pktable (c1 INT PRIMARY KEY, c2 OID) ;
CREATE TABLE
postgres=> CREATE TABLE fktable (c1 INT, CONSTRAINT fk1 FOREIGN KEY (tableoid)
REFERENCES pktable(c2)) ;
ERROR:  system columns cannot be used in foreign keys
```

2.6.12. CREATEROLE 属性

PostgreSQL 16 では CREATEROLE 属性を持っても、他のユーザーに対して WITH ADMIN オプションを持たないロールを付与することはできなくなりました。従来のバージョンでは CREATEROLE 属性を持つユーザーは他のユーザーに対してほとんどの変更を実行することができました。

下記の例にある最初の GRANT 文は PostgreSQL 15 では成功します。[\[cf5eb37\]](#)

例 7 GRANT 文の失敗

```
postgres=# CREATE USER useradm1 PASSWORD '<<PASSWORD>>' CREATEROLE ;
CREATE ROLE
postgres=# \connect postgres useradm1
You are now connected to database "postgres" as user "useradm1".
postgres=> CREATE USER monitor1 PASSWORD '<<PASSWORD>>' ;
CREATE ROLE
postgres=> GRANT pg_monitor TO monitor1 ;
ERROR: permission denied to grant role "pg_monitor"
DETAIL:  Only roles with the ADMIN option on role "pg_monitor" may grant this
role.
postgres=> \connect postgres postgres
You are now connected to database "postgres" as user "postgres".
postgres=# GRANT pg_monitor TO useradm1 WITH ADMIN OPTION ;
GRANT ROLE
postgres=# \connect postgres useradm1
You are now connected to database "postgres" as user "useradm1".
postgres=> GRANT pg_monitor TO monitor1 ;
GRANT ROLE
```

2.6.13. POWER 関数

PostgreSQL 16 では POWER 関数に整数を指定した場合の精度が向上し、非整数を指定した場合と有効精度が一致します。この修正は旧バージョンへバックポートされません。
[\[40c7fcb\]](#)

例 8 PostgreSQL 15 の動作

```
postgres=> SELECT power (0.084738, 16) ;
      power
-----
0.0000000000000000
(1 row)
```



例 9 PostgreSQL 16 の動作

```
postgres=> SELECT power(0.084738, 16) ;  
  
           power  
-----  
0.000000000000000000007067264604102046  
(1 row)
```

2.6.14. RESET 文

下記のパラメーターに対する RESET 文は失敗するようになりました。[[3853664](#)]

- transaction_read_only
- transaction_deferrable
- seed
- transaction_isolation

例 10 RESET 文の失敗

```
postgres=> RESET transaction_read_only ;  
ERROR:  parameter "transaction_read_only" cannot be reset  
postgres=> RESET transaction_deferrable ;  
ERROR:  parameter "transaction_deferrable" cannot be reset  
postgres=> RESET seed ;  
ERROR:  parameter "seed" cannot be reset  
postgres=> BEGIN ;  
BEGIN  
postgres=> RESET transaction_isolation ;  
ERROR:  parameter "transaction_isolation" cannot be reset  
postgres!=> ROLLBACK ;  
ROLLBACK
```

2.6.15. pg_stat_get_backend_idset 関数

pg_stat_get_backend_idset 関数はバックエンド・プロセスの ID を返すようになりました。旧バージョンでは 1 から始まる順番を返していました。[[d7e39d7](#)]

例 11 PostgreSQL 15 の動作

```
postgres=> SELECT pg_stat_get_backend_idset() ;
pg_stat_get_backend_idset
-----
1
2
3
4
5
6
(6 rows)
```

例 12 PostgreSQL 16 の動作

```
postgres=> SELECT pg_stat_get_backend_idset() ;
pg_stat_get_backend_idset
-----
1
2
4
124
126
127
(6 rows)
```

2.6.16. pg_walinspect エクステンション

pg_walinspect エクステンションの pg_get_wal_records_info 関数、pg_get_wal_stats 関数、pg_get_wal_block_info 関数に指定される終了 LSN は現在の LSN よりも高い値を指定してもエラーが発生しないようになりました。この変更により不要になった pg_get_wal_records_info_till_end_of_wal 関数と、pg_get_wal_stats_till_end_of_wal 関数が削除されました。[[5c1b662](#)]



2.6.17. postmaster

-T オプションは SIGSTOP シグナルではなく `send_abort_for_crash` パラメーターの設定により SIGABRT シグナルを送信する場合があります。また -n オプションは削除されました。postmaster を指すシンボリックリンクは作成されなくなりました。[[51b5834](#), [37e2673](#)]

2.6.18. psql

psql コマンドは以下の仕様に変更されました。[[3dfae91](#), [00beecf](#)]

□ `\df+` メタコマンド

`\df+` メタコマンドの出力列名の「Source code」は「Internal name」に変更されました。

例 13 PostgreSQL 16 の動作

```
postgres=> \df+ scale
List of functions
-[ RECORD 1 ]-----+
Schema          | pg_catalog
Name            | scale
Result data type | integer
Argument data types | numeric
Type            | func
Volatility       | immutable
Parallel        | safe
Owner           | postgres
Security        | invoker
Access privileges |
Language        | internal
Internal name    | numeric_scale
Description      | number of decimal digits in the fractional part
```

□ `\watch` メタコマンド

`\watch` メタコマンドに渡された負の数値や文字列はエラーになります。従来は 1 秒とみなされていました。また 0 を指定するとインターバル無しでコマンドが再実行されます。



例 14 PostgreSQL 16 の動作

```
postgres=> \watch -1
\watch: incorrect interval value '-1'
postgres=> \watch abc
\watch: incorrect interval value 'abc'
```

2.6.19. libpq

SCM 資格証明による認証コードが削除されました。バックエンドのコードは PostgreSQL 9.1 で削除されていますが、libpq のコードには残っていました。[[98ae2c8](#)]

2.6.20. PL/Python

{インストール先}/lib/pgxs/src/pl/plpython ディレクトリは作成されなくなりました。[[7d5852c](#)]

2.6.21. ECPG

EXEC SQL コマンドで使用する SQL キーワードを typedef として宣言できなくなりました。以下の例は構文エラーになります。[[83f1c7b](#)]

例 15 エラーになる ECPG プログラム

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef int start;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL START TRANSACTION;
```

3. 新機能解説

3.1. アーキテクチャの変更

3.1.1. システムカタログの変更

以下のシステムカタログやビューが変更されました。[[d540a02](#), [84ad713](#), [c037471](#), [3662839](#), [6566133](#), [e3ce2de](#), [c591300](#), [ae4fdde](#), [90189ee](#), [216a784](#), [e0b0142](#), [e056c55](#), [0c67946](#)]

表 6 追加されたシステムカタログ／ビュー

カタログ／ビュー名	説明
pg_stat_io	詳細な I/O 統計を取得します。

表 7 列が追加されたシステムカタログ／ビュー

カタログ／ビュー名	追加列名	データ型	説明
pg_auth_members	oid	oid	Object ID
	inherit_option	boolean	継承の許可
	set_option	boolean	SET ROLE 文の許可
pg_collation	collicurules	text	ICU カスタムルール
pg_database	daticurules	text	ICU カスタムルール
pg_hba_file_rules	rule_number	integer	ルール番号
	file_name	text	設定ファイル名
pg_ident_file_mappings	map_number	integer	マッピング番号
	file_name	text	設定ファイル名
pg_prepared_statements	result_types	regtype[]	ステートメントから返される列のタイプ
pg_replication_slots	conflicting	boolean	リカバリと競合しているか
pg_stat_database_conflicts	confl_active_logicalslot	bigint	キャンセルされた論理スロットの使用数
pg_stat_gssapi	credentials_delegated	boolean	GSSAPI 委任されたセッションか

カタログ／ビュー名	追加列名	データ型	説明
pg_stat_subscription	leader_pid	integer	パラレル適用リーダー・プロセスの ID
pg_stat_*_indexes	last_idx_scan	timestamp with time zone	インデックス・アクセス最終時刻
pg_stat_*_tables	last_seq_scan	timestamp with time zone	シーケンシャル・アクセス最終時刻
	last_idx_scan	timestamp with time zone	インデックス・アクセス最終時刻
	n_tup_newpage_upd	bigint	後継バージョンが新しいヒープ・ページに移動した行数
pg_subscription	suborigin	text	PUBLICATION へのデータ送信依頼種別
	subpasswordrequired	boolean	認証にパスワードが必要
	subrunasowner	boolean	SUBSCRIPTION 所有者による実行

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_attribute	attndims 列、attstattarget 列、attinhcount 列のデータ型が integer から smallint に変更されました。また列の順序が変更されています。
pg_constraint	coninhcount 列は integer 型から smallint 型に変更されました。
pg_locks	locktype 列に applytransaction が出力されるようになりました。
pg_stat_activity	backend_type 列に standalone backends が出力されるようになりました。
pg_stat_subscription	SUBSCRIPTION 作成直後にレコードが作成されます。従来は最初の統計情報が受信された時点で作成されていました。
pg_subscription	substream 列が bool 型から char 型に変更されました。パラレル適用時には 'p' が出力されます。

追加された pg_stat_io ビューについて詳細を以下に記載します。pg_stat_io ビューはデータベース・クラスタ上で発生した詳細な I/O 統計を提供します。[\[a9c70b4, 8aaa04b, ac8d53d, ac8d53d, 0ecb87e, 093e5c5\]](#)



表 9 pg_stat_io ビュー

列名	データ型	説明
backend_type	text	バックエンド・プロセスのタイプ。 <ul style="list-style-type: none">● background worker● client backend● walsender● standalone backend● autovacuum worker● autovacuum launcher● background writer● startup● checkpointer 等
object	text	対象リレーションの種類 (relation, temp relation)
context	text	I/O 処理の種類 (normal, vacuum, bulkread, bulkwrite)。
reads	bigint	io_bytes 列で示されたサイズの読み込み回数
read_time	double precision	読み込み時間の合計
writes	bigint	io_bytes 列で示されたサイズの書き込み回数
write_time	double precision	書き込み時間の合計
writebacks	bigint	ライトバック回数
writeback_time	double precision	ライトバック時間
extends	bigint	io_bytes 列で示されたサイズの拡張回数
extend_time	double precision	拡張時間の合計
op_bytes	bigint	1 回の I/O 処理バイト数
hits	bigint	キャッシュにヒットした回数
evictions	bigint	別の用途に利用できるように、共有バッファまたはローカルバッファから書き出された回数
reuses	bigint	共有バッファ外の既存リング・バッファが I/O 操作で再利用された回数
fsyncs	bigint	fsync システムコールの実行回数
fsync_time	double precision	fsync システムコールの実行時間
stats_reset	timestamp with time zone	ビューのリセット時刻

例 16 pg_stat_io ビューの検索

```
postgres=> SELECT backend_type, context, reads, writes FROM pg_stat_io WHERE
           backend_type='autovacuum worker' ;
```

backend_type	context	reads	writes
autovacuum worker	bulkread	0	0
autovacuum worker	normal	548	0
autovacuum worker	vacuum	145	0

(3 rows)

read_time 列、write_time 列、extend_time 列、fsync_time 列を確認するにはパラメータ track_io_timing を on に設定する必要があります。ビューの内容をリセットするには pg_stat_reset_shared 関数を実行します。

例 17 ビューのリセット

```
postgres=# SELECT pg_stat_reset_shared('io') ;
```

```
pg_stat_reset_shared
```

```
-----
```

(1 row)

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。[[216a784](#), [3662839](#), [4826759](#), [1e10d49](#), [ecb6965](#), [c3afe8c](#), [89e46da](#), [0fdab27](#), [5de94a0](#), [9f2213a](#), [2666975](#), [c9f7f92](#)]

□ パラレル適用

従来は大規模なトランザクションは一時ファイルに保存された後に適用されていました。PostgreSQL 16 では複数のワーカー・プロセスによってトランザクションの差分を直接適用できるようになりました。この機能を有効にするためには CREATE SUBSCRIPTION 文のオプション streaming に parallel を指定します。これに伴い pg_subscription カタログの substream 列は bool 型から char 型に変更され、CREATE SUBSCRIPTION 文の stream オプションに parallel に指定された場合は 'p' が出力されます。サブスクリプションが使用

するワーカー・プロセスの最大値はパラメーター `max_parallel_apply_workers_per_subscription` で決定されます。

例 18 パラレル適用の設定

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

□ SUBSCRIPTION の origin オプション

CREATE/ALTER SUBSCRIPTION 文のオプションに `origin` を指定できるようになりました。このパラメーターには PUBLICATION に対して送信要求を行う変更の種類を指定します。このオプション情報を格納するために `pg_subscription` カタログに `suborigin` 列が追加されました。

表 10 origin オプション設定値

設定値	説明	備考
none	SUBACRIPTION は PUBLICATION に対してオリジンに関連しない変更のみを要求する。	
any	PUBLICATION はオリジンに関係なくすべての変更内容を SUBSCRIPTION に送信する。	デフォルト値

`origin` オプションを `none` に設定することで、制限付きながら同一テーブルに対して双方向レプリケーションが可能になります。

例 19 origin オプションの設定

```
postgres=# CREATE SUBSCRIPTION sub1 CONNECTION 'port=5432 dbname=postgres
user=postgres' PUBLICATION pub1 WITH (origin=none) ;
NOTICE:  created replication slot "sub1" on publisher
CREATE SUBSCRIPTION
postgres=# SELECT subname, suborigin FROM pg_subscription ;
 subname | suborigin 
-----+-----
 sub1    | none
(1 row)
```

□ SUBSCRIPTION の run_as_owner オプション

従来のロジカル・レプリケーションでは SUBSCRIPTION は SUBSCRIPTION 所有者の権限でテーブルを更新していました。PostgreSQL 16 のデフォルトではテーブル所有者の権限で適用が行われます。run_as_owner オプションを true に設定することで旧バージョンと同じ動作にすることができます。このオプション情報を格納するために pg_subscription カタログに subrunasowner 列が追加されました。

□ 初期データ同期の BINARY 転送

PUBLICATION / SUBSCRIPTION 双方が PostgreSQL 16 以上で、SUBSCRIPTION の binary オプションが有効な場合、初期データ移行をバイナリ形式で転送できるようになりました。以下の例は PUBLICATION 側で初期データ同期のための実行される COPY 文のログです。

例 20 初期データ同期で実行される COPY 文のログ

```
LOG:  statement: COPY public.data1 (c1, c2) TO STDOUT WITH (FORMAT binary)
LOG:  statement: COMMIT
```

□ REPLICA IDENTITY FULL 設定時のインデックス

パブリッシャー側で REPLICA IDENTITY や主キーが利用できない環境でもサブスクライバー側で主キーや REPLICA IDENTITY 以外のインデックスが使用できるようになります。使用できるインデックスは部分インデックスではなく、B-tree インデックスであり、少なくとも 1 つの列参照が必要です。

□ SUBSCRIPTION の作成ロール

pg_create_subscription ロールを許可された一般ユーザーは SUBSCRIPTION を作成できるようになります。ただし CONNECTION 句にパスワードが必要になります。

password_required 属性をオフにできるのは SUPERUSER 属性を持つユーザーのみです。このオプションが追加されたことに伴い pg_subscription カタログに subpasswordrequired 列が追加されました。

例 21 SUBSCRIPTION 作成ロール

```
postgres=# GRANT pg_create_subscription TO demo ;
GRANT ROLE
postgres=# \connect postgres demo
You are now connected to database "postgres" as user "demo".
postgres=> CREATE SUBSCRIPTION sub1 CONNECTION 'port=5432 dbname=postgres'
PUBLICATION pub1 ;
ERROR: password is required
DETAIL: Non-superusers must provide a password in the connection string.
postgres=> CREATE SUBSCRIPTION sub1 CONNECTION 'port=5432 dbname=postgres
user=postgres' PUBLICATION pub1 WITH (password_required=false) ;
ERROR: password_required=false is superuser-only
HINT: Subscriptions with the password_required option set to false may only be
created or modified by the superuser.
```

□ スタンバイ・インスタンスでロジカル・デコーディング

スタンバイ・インスタンス上でロジカル・デコーディングが実行可能になりました。スタンバイ・インスタンスでロジカル・レプリケーション・スロットの作成を高速化するために、プライマリ・インスタンス上で pg_log_standby_snapshot 関数を実行することができます。

例 22 pg_log_standby_snapshot 関数の実行（プライマリ・インスタンス）

```
postgres=# SELECT pg_log_standby_snapshot() ;
pg_log_standby_snapshot
-----
0/5000098
(1 row)
```

□ レプリケーション・モード

ロジカル・レプリケーションのモードを変更できるパラメーター

`logical_replication_mode` が追加されました。デフォルト値は `buffered` です。このパラメーターには以下の値が指定できます。

表 11 レプリケーション・モードの設定値（パブリッシャー側）

設定値	説明
<code>buffered</code>	変更データ量が <code>logical_decoding_work_mem</code> に達した場合にシリアル化されます。
<code>immediate</code>	<code>CREATE SUBSCRIPTION</code> 文の <code>streaming</code> オプションが有効な場合はストリーミングが行われます。

表 12 レプリケーション・モードの設定値（サブスクライバー側）

設定値	説明
<code>buffered</code>	<code>streaming</code> オプションが <code>parallel</code> に設定されている場合、共有メモリー・キューを通じてデータをパラレル・ワーカーに送信します。
<code>immediate</code>	すべてのデータをファイルにシリアル化し、トランザクション終了時に適用するようにパラレル・ワーカーに通知します。

□ 競合の検知

`pg_stat_database_conflicts` ビューに `confl_active_logicalslot` 列が追加されました。この列にはロジカル・レプリケーション・スロットでキャンセルされた使用数が出力されます。

□ ALTER TABLE REPLICA IDENTITY 文

インデックスが `INVALID` な状態でも `ALTER TABLE REPLICA IDENTITY USING` 文が成功するようになりました。これは `pg_dump` コマンドがテーブル定義を出力する順番によるエラーを回避するための修正です。

3.1.3. パラレル・クエリー

パラレル・クエリーが実行される構文や関数が増えました。[[11c2d6f](#), [16fd03e](#)]

□ Parallel Hash Full Join

`RIGHT OUTER JOIN` 構文と `FULL OUTER JOIN` 構文でパラレル・クエリーがサポートされるようになりました。パラメーター `enable_parallel_hash` で実行を制御できます。



例 23 Parallel Hash Full Join 実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT COUNT(*) FROM data1 d1 FULL OUTER JOIN
           data2 d2 USING (c1) ;
           QUERY PLAN

-----

Finalize Aggregate
  -> Gather
      Workers Planned: 2
      -> Partial Aggregate
          -> Parallel Hash Full Join
              Hash Cond: (d1.c1 = d2.c1)
                  -> Parallel Seq Scan on data1 d1
                  -> Parallel Hash
                      -> Parallel Seq Scan on data2 d2

(9 rows)
```

□ 関数

`string_agg` 関数および `array_agg` 関数の実行時に集約処理を並列に実行できるようになりました。パラレル・クエリーが実行される場合には実行計画に `Partial Aggregate` が出力されます。

例 24 `string_agg` 関数の実行

```
postgres=> EXPLAIN SELECT string_agg(c2, ':') FROM data1 ;
           QUERY PLAN

-----

Finalize Aggregate (cost=11614.56..11614.57 rows=1 width=32)
  -> Gather (cost=11614.34..11614.55 rows=2 width=32)
      Workers Planned: 2
      -> Partial Aggregate (cost=10614.34..10614.35 rows=1 width=32)
          -> Parallel Seq Scan on data1 (cost=0.00..9572.67 rows=416667
width=6)

(5 rows)
```

3.1.4. 設定ファイル

いくつかの設定ファイルについて記述方法が改善されました。[\[8fea868, efb6f4a, a54b658, efb6f4a\]](#)

□ ホスト名とユーザー名の正規表現

pg_hba.conf ファイルではデータベース名とユーザー名を正規表現で記述できるようになりました。スラッシュ (/) から始まるユーザー名とデータベース名は正規表現とみなされます。

例 25 正規表現を使った pg_hba.conf ファイルのユーザー名とデータベース名

#	TYPE	DATABASE	USER	ADDRESS	METHOD
host		/^demodb[1-3]	all	192.168.1.0/24	scram-sha-256
host		demodb4	/^user.*\$	192.168.1.0/24	scram-sha-256

正規表現は pg_ident.conf ファイルのデータベース・ユーザー項目にも使用できます。

□ 別ファイルのインクルード

pg_hba.conf ファイルと pg_ident.conf ファイルには postgresql.conf ファイルと同様に他のファイルをインクルードすることができるようになりました。

表 13 追加された構文

構文	説明
include ファイル名	ファイルをインクルードする
include_if_exists ファイル名	ファイルが存在すればインクルードする
include_dir ディレクトリ名	指定されたディレクトリ以下の拡張子.conf を持つ全ファイルをインクルードする

例 26 ファイルをインクルードする設定

<pre>\$ cat pg_hba.conf include pg_hba_1.conf include_if_exists pg_hba_2.conf include_dir hba_dir</pre>

pg_hba_file_rules カタログ、pg_ident_file_mappings カタログには設定ファイル名を示す file_name 列が追加されました。

□ pg_ident.conf ファイル

PostgreSQL ユーザー名部分に対して以下のように pg_hba.conf ファイルと同様の設定が許可されます。

- 全データベース・ユーザーを意味する all
- プラス (+) を使ったメンバーチェック
- データベース・ユーザー名の正規表現

3.1.5. ICU ロケール

ICU ロケールを利用する機能が拡張されました。[[fcb21b3](#), [27b6237](#), [c1f1c1f](#), [5cd1a5a](#), [c45dc7f](#), [30a53b7](#), [cd42785](#), [0d21d4b](#)]

□ ロケール・プロバイダのデフォルト

ICU が利用可能な環境ではデフォルトのロケール・プロバイダが libc から icu に変更されました。ソースコードからビルドする際も ICU が標準で組み込まれます。configure コマンドの --with-icu オプションは廃止され、ICU を使用しない場合に指定する --without-icu オプションが追加されました。ただし initdb コマンドに --no-locale のみ指定された環境では従来通りロケール・プロバイダに LIBC が使用されます。

例 27 デフォルトのロケール・プロバイダの変更 (initdb)

```
$ export LANG=ja_JP.utf8
$ initdb data
This user must also own the server process.

Using default ICU locale "ja".
Using language tag "ja" for ICU locale "ja".
The database cluster will be initialized with this locale configuration:
    provider:    icu
    ICU locale:  ja
    LC_COLLATE:  ja_JP.utf8
...
```

例 28 デフォルトのロケール・プロバイダの変更 (psql)

```
postgres=# \l
```

List of databases							
Name	Owner	Encoding	Locale Provider	Collate	Ctype	ICU Locale	...
postgres	postgres	UTF8	icu	ja_JP.utf8	ja_JP.utf8	ja	...
template0	postgres	UTF8	icu	ja_JP.utf8	ja_JP.utf8	ja	...
							...
template1	postgres	UTF8	icu	ja_JP.utf8	ja_JP.utf8	ja	...
							...

(3 rows)

□ デフォルト・エンコード

ICU ロケールを使用する際のデフォルト・エンコードは UTF8 が使用されます。

□ カスタムルールの指定

ICU ロケールにカスタムルールを指定できるようになりました。ルールを指定するために CREATE COLLATION 文に RULES 句、CREATE DATABASE 文に ICU_RULES 句が追加されました。ルールの詳細な構文は以下の URL を参照してください。

<https://unicode-org.github.io/icu/userguide/collation/customization/>

構文

```
CREATE COLLATION [ IF NOT EXISTS ] name (
    [ RULES = rules ]
)
```

```
CREATE DATABASE name
    [ WITH ] [ ICU_RULES [=] icu_rules ]
```

以下の例ではデータベース作成時のロケール「en_US」に独自のルール「&a < g」を追加しています。ルールが適用されたテーブルではソート時に apple, green, bird の順に出力されています。



例 29 カスタムルール の指定

```
postgres=# CREATE DATABASE en_db1 LOCALE_PROVIDER icu ICU_LOCALE 'en_US'
          ICU_RULES '&a < g' TEMPLATE template0 ;
CREATE DATABASE
postgres=# \connect en_db1 postgres
You are now connected to database "en_db1" as user "postgres".
en_db1=# CREATE TABLE data1(c1 text) ;
CREATE TABLE
en_db1=# INSERT INTO data1 values ('apple'),('bird'),('green') ;
INSERT 0 3
en_db1=# SELECT c1 FROM data1 ORDER BY c1 ;
 c1
-----
apple
green
bird
(3 rows)
```

カスタムルールの内容は pg_database カタログの daticurules 列、pg_collation カタログの collicurules 列、psql コマンドのオブジェクト定義等で確認できます。

例 30 カスタムルール の確認

```
postgres=# \l en_db1
List of databases
-[ RECORD 1 ]-----+
Name           | en_db1
Owner           | postgres
Encoding        | UTF8
Locale Provider | icu
Collate         | C
Ctype           | C
ICU Locale      | en_US
ICU Rules       | &a < g
Access privileges |
```


□ ロケールの追加

ICU ロケールに UNICODE が追加されました。

例 31 UNICODE ロケールの確認

```
postgres=> SELECT * FROM pg_collation WHERE collname = 'unicode' ;
-[ RECORD 1 ]-----+-----
oid              | 12344
collname         | unicode
collnamespace    | 11
collowner        | 10
collprovider     | i
collisdeterministic | t
collencoding     | -1
collcollate      |
collctype        |
colliculocale    | und
collicurules     |
collversion      | 153.80
```

3.1.6. 事前定義ロール

PostgreSQL 16 には以下の事前定義ロールが追加されました。 [[c3afe8c](#), [60684dd](#), [6e2775e](#)]

表 14 追加された事前定義ロール

ロール名	説明
pg_create_subscription	SUBSCRIPTION の作成を許可します。
pg_maintain	すべてのテーブルに対するメンテナンス文の実行を許可します。
pg_use_reserved_connections	パラメーターreserved_connections に割り当てられたコネクションを利用できます。

pg_maintain ロールを付与されたユーザーは他のユーザーが所有するテーブルやマテリアライズドビューに対して ANALYZE 文、VACUUM 文、REINDEX 文、CLUSTER 文、REFRESH MATERIALIZED VIEW 文および LOCK TABLE 文を実行できます。従来は許

可が無い他のユーザー所有のオブジェクト操作は SUPERUSER 属性を持つユーザーに限られていました。

3.1.7. 待機イベント

以下の待機イベントが追加・変更されました。[[7bae3bb](#), [fce003c](#), [216a784](#), [5a3a953](#), [d8cd0c6](#), [92daeca](#), [8fba928](#)]

表 15 追加された待機イベント

イベント名	タイプ	説明
DSMAllocate	IO	動的共有メモリの取得待ち
LogicalApplySendData	IPC	リーダー・プロセスが並列適用ワーカーにデータ送信完了待ち
LogicalParallelApplyMain	Activity	パラレル適用待ち
LogicalParallelApplyState Change	Activity	パラレル適用のステータス変更待ち
LogicalRepLauncherDSA	LWLock	ランチャー・プロセスの DSA メモリー・アロケータへのアクセス待ち
LogicalRepLauncherHash	LWLock	ランチャー・プロセスの共有ハッシュ・テーブルへのアクセス待ち
RelationMapReplace	IO	リレーションマップ・ファイルの置き換え待ち。 RelationMapSync から変更
SLRUFlushSync	IO	SLRU データのストレージ到達待ち
SpinDelay	Timeout	pg_usleep 実行待ち時間

表 16 名前が変更された待機イベント

変更前イベント名	変更後イベント名	備考
HashGrowBatchesAllocate	HashGrowBatchesReallocate	
HashGrowBucketsAllocate	HashGrowBucketsReallocate	

3.1.8. トリガー

トリガーには以下の拡張が実装されました。[[3b00a94](#), [93f2349](#)]



□ TRUNCATE トリガー

外部テーブルに対する TRUNCATE 文の実行時に TRUNCATE トリガーが実行されるようになりました。

□ イベント・トリガー

ALTER MATERIALIZED VIEW 文の実行でイベント・トリガーが実行されるようになりました。

3.1.9. ログ

いくつかの場面でログに出力される情報が追加されました。[[62c46ee](#), [d977ffd](#), [71cb84e](#)]

□ チェックポイント

パラメーターlog_checkpoints を on に設定した場合のログに LSN の情報が追加されます。

例 32 チェックポイントのログ

```
LOG:  checkpoint starting: time
LOG:  checkpoint complete: wrote 44 buffers (0.3%); 0 WAL file(s) added, 0
removed, 0 recycled; write=4.134 s, sync=0.005 s, total=4.143 s; sync files=11,
longest=0.002 s, average=0.001 s; distance=258 kB, estimate=258 kB;
lsn=0/153EC70, redo lsn=0/153EC38
```

□ 自動 VACUUM

自動 VACUUM のログにフリーズされたテーブルの情報が追加されるようになりました。

例 33 自動 VACUUM のログ

```
LOG:  automatic vacuum of table "postgres.public.data1": index scans: 1
      pages: 0 removed, 541 remain, 541 scanned (100.00% of total)
      tuples: 50000 removed, 50000 remain, 0 are dead but not yet removable
      removable cutoff: 750, which was 0 XIDs old when operation ended
      new relfrozenxid: 747, which is 1 XIDs ahead of previous value
      frozen: 0 pages from table (0.00% of total) had 0 tuples frozen
      avg read rate: 0.414 MB/s, avg write rate: 0.829 MB/s
```

...

□ WAL 関連

WAL ヘッダの検証中に発生したエラーのメッセージに LSN 情報が出力されるようになりました。

3.1.10. Kerberos 資格情報の委任

クライアントによりサーバーに委任される GSSAPI/Kerberos 認証をサポートします。これにより Kerberos 資格情報を使用して PostgreSQL に対して認証を行うユーザーは、自分の資格情報を PostgreSQL に委任することができます。サーバーは委任された資格情報を使って他のサービスに接続できるようになります。GSSAPI 委任の受け入れはパラメーター `gss_accept_delegation` を `on` に変更することで有効になります。GSSAPI 資格を委任されたセッションは `pg_stat_gssapi` ビューの `credentials_delegated` 列で確認できます。

Libpq 接続文字列に `gssdelegation` が追加されました。設定値のデフォルト値は `0` です。GSS 認証委任を有効にするには `1` に指定します。接続文字列の代わりに環境変数 `PGGSSDELEGATION` も利用できます。この設定は `postgres_fdw` モジュール、`dblink` モジュール等でも使用できます。API として、認証方法を確認する `PQconnectionUsedGSSAPI` が追加されました。[[6633cfb](#), [1f9f6aa](#)]

構文

```
int PQconnectionUsedGSSAPI(const PGconn *conn);
```

3.1.11. libpq

PostgreSQL を利用する API である `libpq` には以下の拡張が実装されました。[[3a465cc](#), [36f40ce](#), [7f5b198](#), [8eda731](#), [9fcdf2c](#), [419a8dd](#), [19d8e23](#)]

□ 接続文字列 `require_auth`

必要な認証メソッドの一覧を指定する接続文字列 `require_auth` が追加されました。このパラメーターには以下の接続メソッドをカンマ (,) 区切りで指定します。サーバーが提供できる認証メソッドが無い場合、接続は失敗します。この接続文字列は環境変数 `PGREQUIREAUTH` でも指定できます。

表 17 設定値

認証メソッド文字列	説明
password	パスワード認証
md5	MD5 パスワードによる認証
gss	GSSAPI 認証
sspi	SSPI 認証
scram-sha-256	SCRAM-SHA-256 認証
none	必須認証メソッド無し

例 34 必須認証メソッドの設定

```
$ grep demo data/pg_hba.conf
host    all             demo          127.0.0.1/32          scram-sha-256
$ export PGREQUIREAUTH=scram-sha-256
$ psql -h 127.0.0.1 -d postgres -U demo
Password for user demo:
psql (16beta1)
Type "help" for help.

postgres=> ¥q
$ psql -h 127.0.0.1 -d postgres -U postgres
psql: error: connection to server on socket "/tmp/.s.PGSQL.5432" failed: auth
method "scram-sha-256" requirement failed: server did not complete
authentication
```

□ 接続文字列 sslcertmode

接続文字列 `sslcertmode` はサーバーがクライアントから証明書を送信するかを決定します。この接続文字列は環境変数 `PGSSLCERTMODE` に指定することもできます。

表 18 設定値

設定値	説明
allow	クライアント証明書はサーバーが要求した場合に送信します。これは従来と同様であり、デフォルトの動作です。
disable	クライアント証明書が利用可能な場合でも送信を拒否します。
require	クライアント証明書は送信されず、接続は失敗する。トラブルシュー트에役立つ可能性があります。

□ 接続文字列 `load_balance_hosts`

接続文字列 `load_balance_hosts` は接続先リストから特定のインスタンスを自動的に選択します。この接続文字列は環境変数 `PGLOADBALANCEHOSTS` に指定することもできます。接続先が存在しない場合、別の接続先が自動的に選択されます。

表 19 設定値

設定値	説明
random	複数指定された hosts と ports から接続先をランダムに選択します。
disable	接続先のランダム選択を無効にします（デフォルト値）。

例 35 接続先のランダム選択

```
$ psql "load_balance_hosts=random host=localhost,localhost port=5432,5433
dbname=postgres user=postgres" -c "%conninfo"
You are connected to database "postgres" as user "postgres" on host "localhost"
(address "::1") at port "5432".
$ psql "load_balance_hosts=random host=localhost,localhost port=5432,5433
dbname=postgres user=postgres" -c "%conninfo"
You are connected to database "postgres" as user "postgres" on host "localhost"
(address "::1") at port "5433".
```

□ 接続文字列 `sslrootcert`

接続文字列 `sslrootcert` の設定値に `system` を指定できるようになりました。この設定値は証明書の検証のためにシステムの信頼できる CA ルート証明書をロードします。

□ COPY 文のコールバック

`CopySendEndOfRow` API にコールバック関数を指定できるようになりました。この機能はエクステンションが `COPY TO` 文を実行するのに役立ちます。`COPY FROM` 文については既に同様のコールバック機能が提供されています。

□ フック

LDAP 認証のために `pg_hba.conf` ファイルの `ldapbindpasswd` で指定されたパスワードを変更するためのフックが追加されました。`src/backend/libpq/auth.c` ファイルに定義された `ldap_password_hook` をアップデートします。

□ インデックス・アクセス・メソッド

IndexAmRoutine 構造体に「bool amsummarizing;」フラグが追加されました。このフラグはアクセス・メソッドがタプルを要約するかを示します。

3.1.12. PL/pgSQL

自己プロシージャの OID を取得する PG_ROUTINE_OID が追加されました。[\[d3d53f9\]](#)

例 36 PG_ROUTINE_OID の取得

```
postgres=> CREATE OR REPLACE FUNCTION current_function()
           RETURNS regprocedure AS $$
           DECLARE
               fnc_oid regprocedure;
           BEGIN
               GET DIAGNOSTICS fnc_oid = PG_ROUTINE_OID;
               RETURN fnc_oid;
           END; $$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=> SELECT current_function() ;
           current_function
-----
current_function()
(1 row)
```

3.1.13. MATERIALIZED VIEW

REFRESH MATERIALIZED VIEW CONCURRENTLY 文を実行した際に直列化異常 (Serialization Anomaly) の発生を防ぐため、述語ロック (Predicate Lock) に対応します。[\[4335155\]](#)

3.1.14. WAL sender プロセス

論理レプリケーション環境の WAL sender プロセスのプロセス名にデータ提供元 (PUBLICATION 側) のデータベース名が出力されるようになりました。[\[af20515\]](#)

例 37 WAL sender プロセス名

```
$ ps -ef|grep walsender | grep -v grep
postgres    23488      23127   0 18:10 ?                00:00:00 postgres: walsender
postgres    demodb [local] START_REPLICATION
```

3.1.15. GIN インデックスのコスト

GIN インデックスのコスト計算の一部に CPU ベースのコストが考慮されるようになりました。このため GIN インデックスを利用するコスト見積もりが大きくなります。
[\[cd9479a\]](#)

3.1.16. Meson 対応

ビルド・システムとして Meson (<https://mesonbuild.com/>) が利用できるようになりました。ソースコード内の各ディレクトリに meson.build ファイルが配置されています。
[\[e692727\]](#)

3.1.17. UNICODE

対応する Unicode のバージョンが 14.0.0 から 15.0.0 に変更されました。[\[1091b48\]](#)

3.1.18. LLVM

LLVM 15 をサポートします。この変更は旧バージョンにもバックポートされます。
[\[c2ae01f\]](#)

3.1.19. エクステンション

エクステンションに @extschema:{name}@、no_relocate オプションが追加されました。@extschema:{name}@ は既存の @extschema@ 機能を拡張し、拡張スキーマ名を挿入できます。no_relocate オプションはこの拡張機能が依存する拡張機能名のリストを指定します。
[\[72a5b1f\]](#)



例 38 test_ext_req_schema2--1.0.sql の一部

```
CREATE FUNCTION dep_req2() RETURNS text
BEGIN ATOMIC
    SELECT @extschema:test_ext_req_schema1@.dep_req1() || ' req2';
END;
```

例 39 test_ext_req_schema3.control の一部

```
relocatable = true
requires = 'test_ext_req_schema1, test_ext_req_schema2'
no_relocate = 'test_ext_req_schema1'
```

3.1.20. WAL 出力量の削減

FREEZE 処理の WAL 出力量が削減されました。[[9e54059](#)]

3.2. SQL 文の拡張

ここでは SQL 文に関する新機能を説明しています。

3.2.1. データ型

データ型には以下の拡張が実装されました。[\[2ceea5a, 6fcda9a, 102a5c1, 6dfacbf, faff8f8, 096dd80\]](#)

□ +infinity 表記

date 型、timestamp 型、timestamp with time zone 型に+infinity 値を指定できるようになりました。+infinity は infinity と同じとみなされます。

例 40 +Infinity 指定

```
postgres=> SELECT '+infinity'::timestamp ;
timestamp
-----
infinity
(1 row)

postgres=> SELECT '+infinity'::timestampz ;
timestampz
-----
infinity
(1 row)
```

□ 整数リテラル

整数リテラルに 16 進数、8 進数、2 進数の表現ができるようになりました。それぞれ以下のように表現します。先頭の'0'の次に底を示す記号を指定します。大文字と小文字は区別されません。SQL:202x draft に対応する修正です。

表 20 整数リテラルの記述

指定方式	例	備考
16 進数	0x42F	
8 進数	0o273	
2 進数	0b100101	



例 41 整数リテラルの表記方法

```
postgres=> SELECT * FROM data1 WHERE c1 IN ( 100, 0b11001000, 0o454, 0x190 )  
ORDER BY 1 ;
```

c1	c2
100	data1
200	data2
300	data3
400	data4

(4 rows)

□ JSONPATH リテラル

16 進数、8 進数、2 進数の表現は JSONPATH にも指定できます。

例 42 JSONPATH の整数リテラル

```
postgres=> SELECT '0b100101'::jsonpath ;  
jsonpath  
-----  
37  
(1 row)
```

```
postgres=> SELECT '0o273'::jsonpath ;  
jsonpath  
-----  
187  
(1 row)
```

```
postgres=> SELECT '0x42F'::jsonpath ;  
jsonpath  
-----  
1071  
(1 row)
```

□ numeric 型

numeric 型は 16 進数、8 進数、2 進数の表記をサポートします。ただし小数点以下の記述はできません。

例 43 numeric 型の 16 進数表記

```
postgres=> SELECT pg_input_is_valid(' 0x1234', 'numeric') ;
pg_input_is_valid
-----
t
(1 row)

postgres=> SELECT message FROM
           pg_input_error_info(' 0x1234.56', 'numeric(10, 3)') ;
           message
-----
invalid input syntax for type numeric: "0x1234.56"
```

□ 数値リテラルとアンダースコア

数値リテラルにアンダースコア (_) を指定することができます。SQL:202x draft に対応する修正です。

例 44 数値リテラルにアンダースコアを指定

```
postgres=> SELECT 1_000 dec, 2_000.234 dec2, 0x100_200 hex,
                0o2_467 oct, 0b01_101 bin1, 0b_1001_0001 bin2 ;
-[ RECORD 1 ]--
dec  | 1000
dec2 | 2000.234
hex  | 1049088
oct  | 1335
bin1 | 13
bin2 | 145
```

3.2.2. COPY

COPY 文には以下の機能が実装されました。[9f8377f, 97da482]

□ デフォルト値の挿入

COPY FROM 文にオプション DEFAULT が追加されました。このオプションに指定した値が入力値に一致した場合、列の DEFAULT 値が入力されます。列に DEFAULT 指定が無く、COPY 文の DEFAULT オプションに一致するデータが存在した場合はエラーになります。

例 45 DEFAULT オプションの指定

```
postgres=> CREATE TABLE data1(c1 INT PRIMARY KEY, c2 TEXT DEFAULT 'DefVal1') ;
CREATE TABLE
postgres=> COPY data1 FROM stdin WITH (format csv, default '¥D') ;
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
>> 2,¥D
>> 3,data3
>> COPY 3
postgres=> SELECT * FROM data1 ;
 c1 |  c2
----+-----
  1 | data1
  2 | DefVal1
  3 | data3
(3 rows)
```

このオプションはエクステンション file_fdw でも使用できます。FOREIGN TABLE を参照するとファイル内の値が DEFAULT オプションに一致すると、列の DEFAULT 値が参照されます。

例 46 DEFAULT オプションの指定 (file_fdw)

```
postgres=# ¥! cat /tmp/data1.csv
1,data1
2,¥D
3,data3
postgres=# CREATE FOREIGN TABLE file1(c1 INT, c2 TEXT DEFAULT 'DefVal1')
  SERVER fsvr1 OPTIONS (filename '/tmp/data1.csv', format 'csv', default '¥D') ;
CREATE FOREIGN TABLE
postgres=# SELECT * FROM file1 ;
 c1 |  c2
----+-----
  1 | data1
  2 | DefVal1
  3 | data3
(3 rows)
```

□ batch_size オプションの有効範囲

外部テーブルに対する COPY 文の実行時に、外部サーバー (FOREIGN SERVER) や外部テーブル (FOREIGN TABLE) の batch_size オプションを利用できるようになりました。従来は INSERT 文実行時にのみ有効でした。

3.2.3. CREATE ROLE/USER

CREATEROLE 属性を持つロール/ユーザーは、自身が持つ属性、ADMIN OPTION 付で付与されたロールのみ他のユーザーに付与できるようになります。CRETEDB 属性は旧バージョンで同じ動作になっています。[f1358ca]



例 47 CREATEROLE 属性ユーザーによるユーザー作成

```
postgres=# CREATE USER usradm1 CREATEROLE REPLICATION CREATEDB BYPASSRLS ;  
CREATE ROLE  
postgres=# \connect postgres usradm1  
You are now connected to database "postgres" as user "usradm1".  
postgres=> CREATE USER user_rep11 REPLICATION ;  
CREATE ROLE  
postgres=> CREATE USER user_rls1 BYPASSRLS ;  
CREATE ROLE  
postgres=> CREATE USER user_db1 CREATEDB ;  
CREATE ROLE
```

ロールを作成したユーザーは自動的に作成したロールのメンバーとなります。しかし権限を自動的に継承（または SET ROLE）できません。この動作を変更したい場合はパラメーター `createrole_self_grant` に SET と INHERIT をカンマ (,) 区切りで指定します。

例 48 作成ロールに対する SET ROLE 文

```
postgres=> CREATE ROLE role1 ;  
CREATE ROLE  
postgres=> SET ROLE role1 ;  
ERROR: permission denied to set role "role1"  
postgres=> SET createrole_self_grant = 'SET' ;  
SET  
postgres=> CREATE ROLE role2 ;  
CREATE ROLE  
postgres=> SET ROLE role2 ;  
SET
```

3.2.4. CREATE STATISTICS

統計情報の名前は必須ではなくなりました。名前を省略した場合の統計情報名は元のテーブル名と列名から自動生成されます。[624aa2a]

例 49 CREATE STATISTICS 文の名前省略

```
postgres=> CREATE STATISTICS ON c1, c2 FROM data1 ;
```

```
CREATE STATISTICS
```

```
postgres=> \d data1
```

Table "public.data1"

Column	Type	Collation	Nullable	Default
c1	integer		not null	
c2	character varying(10)			

Indexes:

"data1_pkey" PRIMARY KEY, btree (c1)

Statistics objects:

"public.data1_c1_c2_stat" ON c1, c2 FROM data1

3.2.5. CREATE TABLE

CREATE TABLE 文には以下の新機能が追加されました。[[784cedd](#), [b9424d0](#)]

□ STORAGE 句の指定

CREATE TABLE 文に STORAGE 句が指定できるようになりました。従来は ALTER TABLE 文で設定する必要がありました。

例 50 CREATE TABLE 文の STORAGE 句

```
postgres=> CREATE TABLE data1(c1 INT PRIMARY KEY,
```

```
      c2 VARCHAR(10) STORAGE PLAIN, c3 TEXT STORAGE EXTENDED) ;
```

```
CREATE TABLE
```

```
postgres=> \d+ data1
```

Table "public.data1"

Column	Type	Collation	Nullable	Default	Storage	...
c1	integer		not null		plain	...
c2	character varying(10)				<u>plain</u>	...
c3	text				<u>extended</u>	...

Indexes:

...

□ DEFAULT ストレージ

CREATE TABLE 文、ALTER TABLE 文の STORAGE 句にはデフォルトのストレージ形式を示す DEFAULT を指定できます。

例 51 ALTER TABLE 文の STORAGE DEFAULT 句

```
postgres=> ALTER TABLE data1 ALTER COLUMN c2 SET STORAGE DEFAULT ;
ALTER TABLE
```

3.2.6. EXPLAIN

パラメーター化されたクエリ文字列の実行計画を表示するための GENERIC_PLAN オプションが追加されました。[[3c05284](#)]

例 52 GENERIC_PLAN オプションの設定

```
postgres=> EXPLAIN (GENERIC_PLAN) SELECT * FROM data1 WHERE c1=$1 ;
               QUERY PLAN
-----
Index Scan using data1_pkey on data1  (cost=0.43..8.45 rows=1 width=10)
   Index Cond: (c1 = $1)
(2 rows)

postgres=> EXPLAIN (GENERIC_PLAN FALSE) SELECT * FROM data1 WHERE c1=$1 ;
ERROR:  there is no parameter $1
LINE 1: ...XPLAIN (GENERIC_PLAN FALSE) SELECT * FROM data1 WHERE c1=$1 ;
                                     ^
```

3.2.7. GRANT

GRANT 文には以下の機能が追加されました。[[b5d6382](#), [e3ce2de](#), [3d14e17](#), [ce6b672](#), [c44f633](#)]

□ MAINTAIN 権限

テーブルおよびマテリアライズドビューのメンテナンス文（VACUUM、ANALYZE、REINDEX、REFRESH MATERIALIZED VIEW、CLUSTER、LOCK TABLE）の実行をテーブルに対して許可する MAINTAIN 権限が追加されました。

psql コマンドでアクセス権限を確認すると'm'と出力されます。

例 53 メンテナンス文の実行権限

```
postgres=> CREATE TABLE data1(c1 INT, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> GRANT MAINTAIN ON data1 TO demo2 ;
GRANT
postgres=> \dp+ data1
```

Access privileges					
Schema	Name	Type	Access privileges	Column privileges	Policies
public	data1	table	demo=arwdDxtm/demo+		
			demo2=m/demo		
(1 row)					

全テーブルに対して同様の操作を許可する事前定義ロールとして **pg_maintain** が追加されています。

□ LOCK TABLE 文の実行権限

LOCK TABLE 文の実行権限がシンプルになりました。テーブル所有者、スーパーユーザーおよび **pg_maintain** ロールの保持者は任意のモードで LOCK TABLE 文を実行できます。テーブルに対するアクセス権限と許可されるロック・モードの関係は以下の表のとおりです。

表 21 アクセス権限と実行可能なロック・モード

テーブル・アクセス権限	許可されるロック・モード
MAINTAIN, UPDATE, DELETE, TRUNCATE	任意のロック・モード
INSERT	ROW EXCLUSIVE MODE
SELECT	ACCESS SHARE MODE

□ WITH INHERIT 句

権限を継承する WITH INHERIT OPTION 句（または WITH INHERIT TRUE）と継承しない WITH INHERIT FALSE 句が指定できます。デフォルトは WITH INHERIT TRUE です。指定した値は **pg_auth_members** カタログの **inherit_option** 列で確認できます。



例 54 WITH INHERIT 句の指定

```
postgres=# GRANT pg_read_all_stats TO demo WITH INHERIT FALSE ;  
GRANT ROLE
```

□ WITH SET 句

SET ROLE 文の使用を許可する WITH SET OPTION 句(または WITH SET OPTION)と、許可しない WITH SET FALSE 句を指定できます。デフォルトは WITH SET TRUE です。指定した値は pg_auth_members カタログの set_option 列で確認できます。

例 55 WITH SET 句の指定

```
postgres=# GRANT role1 TO demo WITH SET FALSE ;  
GRANT ROLE  
postgres=# SET SESSION AUTHORIZATION demo ;  
SET  
postgres=> SET ROLE role1 ;  
ERROR: permission denied to set role "role1"
```



3.2.8. JSON 関連

JSON 関連構文が強化されました。[[7081ac4](#), [6ee3020](#)]

□ JSON コンストラクター

SQL/JSON 標準に準拠した JSON 型のコンストラクターが追加されました。以下の関数
が利用できます。

構文

```
json_array ( [ { value_expression [ FORMAT JSON ] } [, ...] ] [ { NULL | ABSENT }  
ON NULL ] [ RETURNING data_type [ FORMAT JSON [ ENCODING UTF8 ] ] ] )
```

```
json_arrayagg ( [ value_expression ] [ ORDER BY sort_expression ] [ { NULL |  
ABSENT } ON NULL ] [ RETURNING data_type [ FORMAT JSON [ ENCODING UTF8 ] ] ] )
```

```
json_object ( [ { key_expression { VALUE | ':' } value_expression [ FORMAT JSON  
[ ENCODING UTF8 ] ] } [, ...] ] [ { NULL | ABSENT } ON NULL ] [ { WITH | WITHOUT }  
UNIQUE [ KEYS ] ] [ RETURNING data_type [ FORMAT JSON [ ENCODING UTF8 ] ] ] )
```

```
json_objectagg ( [ { key_expression { VALUE | ':' } value_expression } ] [ { NULL  
| ABSENT } ON NULL ] [ { WITH | WITHOUT } UNIQUE [ KEYS ] ] [ RETURNING data_type  
[ FORMAT JSON [ ENCODING UTF8 ] ] ] )
```



例 56 JSON コンストラクター

```
postgres=> SELECT JSON_ARRAY('a', NULL, 'b' NULL ON NULL RETURNING jsonb) ;
           json_array
-----
["a", null, "b"]
(1 row)

postgres=> SELECT JSON_ARRAYAGG(i ORDER BY i DESC) FROM generate_series(1,5)
i ;
           json_arrayagg
-----
[5, 4, 3, 2, 1]
(1 row)

postgres=> SELECT JSON_OBJECT('a': 2 + 3) ;
           json_object
-----
{"a" : 5}
(1 row)

postgres=> SELECT JSON_OBJECTAGG(k: v ABSENT ON NULL WITH UNIQUE KEYS) FROM
(VALUE (1, 1), (0, NULL), (3, NULL), (2, 2), (4, NULL)) foo(k, v) ;
           json_objectagg
-----
{ "1" : 1, "2" : 2 }
(1 row)
```

□ IS JSON 構文

JSON 構文に適合するかをチェックする IS JSON 構文が追加されました。JSON 形式の種類によって複数の構文が用意されています。

構文

```
item IS [NOT] JSON
item IS JSON VALUE
item IS JSON OBJECT
item JS JSON ARRAY
item IS JSON SCALAR
item IS JSON WITHOUT UNIQUE KEYS
item IS JSON WITH UNIQUE KEYS
```

例 57 IS JSON 構文

```
postgres=> SELECT '[1, 2, 3]'::jsonb IS JSON ARRAY ;
?column?
-----
t
(1 row)

postgres=> SELECT '{"a": 1, "b": null}'::jsonb IS JSON OBJECT ;
?column?
-----
t
(1 row)
```

3.2.9. REINDEX

REINDEX SYSTEM 文と REINDEX DATABASE 文ではデータベース名を省略できるようになりました。データベース名を省略するとカレントのデータベースに対してコマンドが実行されます。また REINDEX DATABASE 文ではシステムカタログに対する REINDEX 処理は実行されなくなります。 [\[2cbc3c1\]](#)

例 58 REINDEX 文でデータベース名の省略

```
postgres=# REINDEX DATABASE ;
REINDEX
postgres=# REINDEX SYSTEM ;
REINDEX
```

3.2.10. VACUUM

VACUUM 文には以下の新機能が追加されました。[[a46a701](#), [d977ffd](#), [4211fbd](#), [1cbbec0](#), [1de58df](#)]

□ データベース統計

VACUUM 文にデータベース統計情報の更新を抑止する SKIP_DATABASE_STATS オプションが追加されました。このオプションを TRUE に設定すると pg_class カタログへのスキャンが抑止され、同時実行性が向上します。データベース統計情報のみを取得する ONLY_DATABASE_STATS オプションも利用できるようになりました。

例 59 VACUUM 文の実行

```
postgres=# VACUUM (SKIP_DATABASE_STATS TRUE) ;
VACUUM
postgres=# VACUUM (ONLY_DATABASE_STATS TRUE) ;
VACUUM
```

□ TOAST のみ VACUUM

PROCESS_MAIN FALSE を指定することで、TOAST データのみ VACUUM 処理を行うことができます。PROCESS_MAIN オプションはデフォルトで TRUE です。

例 60 TOAST データのみ VACUUM

```
postgres=> VACUUM (PROCESS_MAIN FALSE, VERBOSE) data1 ;
INFO:  vacuuming "postgres.pg_toast.pg_toast_16392"
INFO:  finished vacuuming "postgres.pg_toast.pg_toast_16392": index scans: 0
pages: 0 removed, 0 remain, 0 scanned (100.00% of total)
tuples: 0 removed, 0 remain, 0 are dead but not yet removable
removable cutoff: 739, which was 0 XIDs old when operation ended
new relfrozenxid: 739, which is 1 XIDs ahead of previous value
frozen: 0 pages from table (100.00% of total) had 0 tuples frozen
index scan not needed: 0 pages from table (100.00% of total) had 0 dead item
identifiers removed
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
...
system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
VACUUM
```

□ 共有バッファの制限

VACUUM 文および ANALYZE 文を実行する際に使用される共有バッファの使用量を指定する BUFFER_USAGE_LIMIT オプションが追加されました。このオプションを指定しない場合のデフォルト値はパラメーター vacuum_buffer_usage_limit (デフォルト値 256kB) で決定されます。

例 61 BUFFER_USAGE_LIMIT オプションの指定

```
postgres=> SHOW vacuum_buffer_usage_limit ;
-[ RECORD 1 ]-----+-----
vacuum_buffer_usage_limit | 256kB

postgres=> VACUUM (BUFFER_USAGE_LIMIT '1MB') data1 ;
VACUUM
```

□ ページレベル FREEZE

ページ単位で FREEZE 処理が実行できるようになりました。これにより WAL の重複排除が実行され書き込み量を削減できます。

3.2.11. サブクエリー

FROM 句のサブクエリーにエイリアス名を指定する必要がなくなりました。[[bcedd8f](#)]

例 62 サブクエリーの FROM 句

```
postgres=> INSERT INTO data1 SELECT * FROM (SELECT * FROM data2 WHERE c1<100) ;
INSERT 0 99
postgres=> SELECT COUNT(*) FROM (SELECT c2 FROM data1 WHERE c1 = 90) ;
count
-----
      1
(1 row)
```


3.2.12. 実行計画

以下の場合により高速な実行計画が選択されるようになりました。[[3c6fc58](#), [ed1a88d](#), [16dc270](#), [9bfd282](#)]

□ SELECT DISTINCT 文

ORDER BY 句または DISTINCT 句を持つ集計関数をより効率的に実行できるようになりました。パラメーター `enable_presorted_aggregate` パラメーターでこの機能を制御できます。このパラメーターの設定値を `off` に設定することで PostgreSQL 15 と同じ動作になります。

例 63 PostgreSQL 16 の実行計画

```
postgres=> EXPLAIN SELECT COUNT(DISTINCT c1) FROM data1 ;
               QUERY PLAN
-----
Aggregate  (cost=28480.42..28480.43 rows=1 width=8)
  ->  Index Only Scan using data1_pkey on data1  (cost=0.42..25980.42 rows=1000
000 width=4)
(2 rows)
```

例 64 PostgreSQL 15 までの実行計画

```
postgres=> SET enable_presorted_aggregate = off ;
SET
postgres=> EXPLAIN SELECT COUNT(DISTINCT c1) FROM data1 ;
               QUERY PLAN
-----
Aggregate  (cost=17906.00..17906.01 rows=1 width=8)
  ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=4)
(2 rows)
```

□ ウィンドウ関数

同一の OVER 句に対して実行計画を共有できるようになりました。以下の SELECT 文を実行した際に実行計画が最適化されます。



例 65 テスト用 SELECT 文

```
SELECT
  empno,
  depname,
  ROW_NUMBER() OVER (PARTITION BY depname ORDER BY enroll_date) rn,
  RANK() OVER (PARTITION BY depname ORDER BY enroll_date ROWS BETWEEN
    UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) rnk,
  DENSE_RANK() OVER (PARTITION BY depname ORDER BY enroll_date RANGE BETWEEN
    CURRENT ROW AND CURRENT ROW) drnk
FROM
  empsalary ;
```

例 66 PostgreSQL 15 の実行計画

```
QUERY PLAN
-----
WindowAgg
  -> WindowAgg
    -> WindowAgg
      -> Sort
          Sort Key: depname, enroll_date
        -> Seq Scan on empsalary
(6 rows)
```

例 67 PostgreSQL 16 の実行計画

```
QUERY PLAN
-----
WindowAgg
  -> Sort
      Sort Key: depname, enroll_date
    -> Seq Scan on empsalary
(4 rows)
```



□ ハッシュ結合／マージ結合

非 NULL 値の入力における RIGHT ANTI JOIN にハッシュ結合とマージ結合がサポートされました。

例 68 HASH RIGHT ANTI JOIN 実行計画 (PostgreSQL 16)

```
postgres=> EXPLAIN (COSTS OFF) SELECT t1.* FROM prt1_adv t1 WHERE NOT EXISTS  
(SELECT 1 FROM prt2_adv t2 WHERE t1.a = t2.b) AND t1.b = 0 ORDER BY t1.a ;
```

QUERY PLAN

Sort

Sort Key: t1.a

-> Hash Right Anti Join

Hash Cond: (t2.b = t1.a)

-> Append

-> Seq Scan on prt2_adv_p1 t2_1

-> Seq Scan on prt2_adv_p2 t2_2

-> Seq Scan on prt2_adv_p3 t2_3

-> Hash

-> Append

-> Seq Scan on prt1_adv_p1 t1_1

Filter: (b = 0)

-> Seq Scan on prt1_adv_p2 t1_2

Filter: (b = 0)

-> Seq Scan on prt1_adv_p3 t1_3

Filter: (b = 0)

(16 rows)



例 69 PostgreSQL 15 の実行計画

QUERY PLAN

Sort
Sort Key: t1.a
-> Hash Anti Join
Hash Cond: (t1.a = t2.b)
-> Append
-> Seq Scan on prt1_adv_p1 t1_1
Filter: (b = 0)
-> Seq Scan on prt1_adv_p2 t1_2
Filter: (b = 0)
-> Seq Scan on prt1_adv_p3 t1_3
Filter: (b = 0)
-> Hash
-> Append
-> Seq Scan on prt2_adv_p1 t2_1
-> Seq Scan on prt2_adv_p2 t2_2
-> Seq Scan on prt2_adv_p3 t2_3
(16 rows)

□ UNION ALL による Memoize

UNION ALL 句による Append 操作で Memoize を利用できるようになりました。



例 70 PostgreSQL 16 の実行計画

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM t1, (SELECT * FROM t2 UNION ALL  
SELECT * FROM t2) t3 WHERE t1.c1 = t3.c1 ;
```

QUERY PLAN

Nested Loop

-> Seq Scan on t1

-> Memoize

Cache Key: t1.c1

Cache Mode: logical

-> Append

-> Index Scan using t2_c1_idx on t2

Index Cond: (c1 = t1.c1)

-> Index Scan using t2_c1_idx on t2 t2_1

Index Cond: (c1 = t1.c1)

(10 rows)

□ パーティション検索の高速化

LIST/RANGE パーティション・テーブルで、同一パーティションが 16 回検索された場合、パーティション情報をキャッシュするようになりました。多数のパーティションを持つテーブルに対する検索が高速化されました。[[a61b1f7](#), [a61b1f7](#)]

3.2.13. 関数

以下の関数が追加／拡張されました。[[888f2ea](#), [2ddab01](#), [38d8176](#), [d5d5741](#), [75bd846](#), [75bd846](#), [283129e](#), [0823d06](#), [cca1863](#), [13e0d7a](#), [10ea0f9](#), [1939d26](#), [1939d26](#), [bf03cfd](#), [483bdb2](#)]

□ array_sample / array_shuffle

配列からランダムに指定された個数の要素を返す array_sample と、配列の要素をシャッフルして返す array_shuffle が追加されました。

構文

```
anyarray array_sample(array anyarray, n integer)  
anyarray array_shuffle(array anyarray)
```

例 71 array_sample / array_shuffle 関数の実行

```
postgres=> SELECT array_sample(ARRAY[1, 2, 3, 4, 5, 6], 3) ;
      array_sample
-----
      {1, 3, 6}
(1 row)
```

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

☐ any_value

集計関数 `any_value` が追加されました。この関数は集約の入力値から任意の値を非決定的に返します。この関数は **SQL:2023** 標準に含まれます。

構文

```
anyelement any_value(anyelement)
```

例 72 any_value 関数の実行

```
postgres=> SELECT * FROM data1 ;
```

c1	c2
100	data1
200	data2
300	data2

(3 rows)


```
postgres=> SELECT ANY_VALUE(c1), MAX(c1), AVG(c1) FROM data1 ;
```

any_value	max	avg
100	300	200.000000000000000000000000

(1 row)



□ date_add / date_subtract

日付時刻の加算・減算を行う関数が追加されました。date_add 関数は時刻を加算し、date_subtract 関数は時刻を減算させます。タイムゾーンはオプションです。

構文

```
timestamp with time zone date_add(timestamp with time zone, interval [, text])
timestamp with time zone date_subtract(timestamp with time zone, interval [,
text])
```

例 73 date_add / date_subtract 関数の実行

```
postgres=> SELECT date_add(' 2021-10-31 00:00:00+02'::timestampz,
                        ' 1 day'::interval, 'Europe/Warsaw') ;
      date_add
-----
2021-11-01 08:00:00+09
(1 row)

postgres=> SELECT date_subtract(' 2021-11-01 00:00:00+01'::timestampz,
                        ' 1 day'::interval, 'Europe/Warsaw') ;
      date_subtract
-----
2021-10-31 07:00:00+09
(1 row)
```

□ date_trunc

date_trunc(unit, timestampz, time_zone)形式の関数の不変性が stable から immutable に変更されました。

□ erf / erfc

標準誤差と相補誤差を計算する erf と erfc 関数が追加されました。

構文

```
double precision erf(double precision)
double precision erfc(double precision)
```



例 74 erf / erfc 関数の実行

```
postgres=> SELECT erf(1.0), erfc(1.0) ;
           erf           |           erfc
-----+-----
0.8427007929497149 | 0.15729920705028513
(1 row)
```

□ generate_series

インターバルを指定できるバージョンが追加されました。

構文

```
timestamp without time zone generate_series(timestamp without time zone,
                                              timestamp without time zone, interval)
```

例 75 インターバルを指定した generate_series 関数の実行

```
postgres=> SELECT * FROM
           generate_series('2001-10-22 00:00 -04:00'::timestampz,
                           '2001-10-24 00:00 -05:00'::timestampz,
                           '1 day'::interval, 'America/New_York') ;
           generate_series
-----
2001-10-22 13:00:00+09
2001-10-23 13:00:00+09
2001-10-24 13:00:00+09
(3 rows)
```

□ random_normal

正規分布の乱数を発生させる関数 random_normal が追加されました。平均と標準偏差を指定します。エクステンション tablefunc には既に normal_rand 関数が提供されていますが、修正されてシステム関数として提供されました。



構文

```
double precision random_normal(mean double precision DEFAULT 0, stddev double  
precision DEFAULT 1)
```

例 76 random_normal 関数の実行

```
postgres=> SELECT random_normal(0.0, 1.0) ;  
random_normal  
-----  
-0.1803089817060607  
(1 row)
```

☐ pg_read_file / pg_read_binary_file

ファイル名とファイルが存在しない場合の挙動を示す `missing_ok` パラメーターを持つバージョンが追加されました。

構文

```
text pg_read_file(filename text, missing_ok boolean)  
bytea pg_read_binary_file(filename text, missing_ok boolean)
```

例 77 pg_read_file 関数の実行

```
postgres=# SELECT pg_read_file('PG_VERSION', true) ;  
pg_read_file  
-----  
16          +  
(1 row)
```

☐ pg_split_walfile_name

WAL ファイル名からシーケンス番号とタイムライン ID を返します。

構文

```
record pg_split_walfile_name(file_name text)
```



例 78 `pg_split_walfile_name` 関数の実行

```
postgres=# SELECT * FROM pg_split_walfile_name ('000000010000000000000001') ;
 segment_number | timeline_id 
-----+-----
              1 |           1
(1 row)
```

□ `pg_stat_get_backend_subxact`

特定のバックエンドのキャッシュ内のサブトランザクションの情報を取得します。

構文

```
record pg_stat_get_backend_subxact(bid integer)
```

例 79 `pg_stat_get_backend_subxact` 関数の実行

```
postgres=# SELECT * FROM pg_stat_get_backend_subxact(5) ;
-[ RECORD 1 ]-----+--
subxact_count      | 0
subxact_overflowed | f
```

□ `pg_input_is_valid`

`pg_input_is_valid` 関数は指定した値がデータ型に合致するかをチェックできます。桁溢れや不適切な日付文字列を事前に確認できます。入力データとデータ型を文字列で指定します。

構文

```
boolean pg_input_is_valid(string text, type text)
```



例 80 pg_input_is_valid 関数の実行

```
postgres=> SELECT pg_input_is_valid(' 1234. 48', 'numeric(5, 2)') ;
pg_input_is_valid
-----
f
(1 row)

postgres=> SELECT pg_input_is_valid(' 2022/12/32 09:30:60', 'timestamp') ;
pg_input_is_valid
-----
f
(1 row)
```

□ pg_input_error_info

pg_input_error_info 関数は入力データが指定したデータ型として適切でない場合のエラー・メッセージ、ヒント文字列、エラーコードを取得できます。

構文

```
record pg_input_error_info(value text, type_name text)
```

例 81 pg_input_is_valid / pg_input_error_info 関数の実行

```
postgres=> SELECT message FROM
      pg_input_error_info(' 1234. 48', 'numeric(5, 2)') ;
message
-----
numeric field overflow
(1 row)

postgres=> SELECT hint FROM
      pg_input_error_info(' 2022/12/32 09:30:60', 'timestamp') ;
hint
-----
Perhaps you need a different "datestyle" setting.
(1 row)
```



□ pg_size_bytes

単位として B（バイト）を指定できるようになりました。

例 82 pg_size_bytes 関数の実行

```
postgres=> SELECT pg_size_bytes('123456B') "B", pg_size_bytes('123456bytes')
"bytes" ;
-[ RECORD 1 ]-
B          | 123456
bytes      | 123456
```

□ pg_import_system_collations

Microsoft Windows 環境でも利用できるようになりました。

□ system_user

SYSTEM_USER は SQL 標準の予約語であり、認証方法と認証 ID を返す関数です。
TRUST 認証の場合には NULL を返します。

構文

```
text system_user()
```

例 83 system_user 関数の実行

```
$ psql -h dbsvr1 -d postgres -U demo
Password for user demo: <<PASSWORD>>
psql (16beta1)
Type "help" for help.

postgres=> SELECT system_user ;
      system_user
-----
scram-sha-256:demo
(1 row)
```



□ xmlserialize

XML ドキュメントを整形する（または整形しない）オプション INDENT（および NO INDENT）が追加されました。

例 84 INDENT の指定

```
postgres=> SELECT xmlserialize(DOCUMENT
        '<foo><bar><val x="y">42</val></bar></foo>' AS text INDENT) ;
xmlserialize
```

```
-----
<foo>                +
  <bar>                +
    <val x="y">42</val>+
  </bar>               +
</foo>               +
```

(1 row)

```
postgres=> SELECT xmlserialize(DOCUMENT
        '<foo><bar><val x="y">42</val></bar></foo>' AS text NO INDENT) ;
xmlserialize
```

```
-----
<foo><bar><val x="y">42</val></bar></foo>
```

(1 row)

3.3. パラメーターの変更

PostgreSQL 16 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。 [[6e2775e](#), [e5b8a4c](#), [1671f99](#), [3226f47](#), [5de94a0](#), [216a784](#), [b577743](#), [51b5834](#), [1e8b617](#), [3d4fa22](#), [6de31ce](#), [9c0a0e2](#)]

表 22 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
debug_io_direct	direct I/O の使用方法を依頼 (postmaster)	"
createrole_self_grant	GRANT 文に自動指定するオプション (user)	"
enable_presorted_aggregate	ソート処理の最適化機能を使用する (user)	on
gss_accept_delegation	クライアントからの GSSAPI 認証の委任を受け入れるか (sighup)	off
icu_validation_level	ICU ロケール・チェックのログ出力レベル (user)	warning
logical_replication_mode	論理レプリケーションの転送方法を設定する (user)	buffered
max_parallel_apply_workers_per_subscription	サブスクリプション単位のワーカー最大数 (sighup)	2
reserved_connections	一般ユーザー用の予約された接続数 (postmaster)	0
scram_iterations	SCRAM 認証の反復回数 (user)	4096
send_abort_for_crash	バックエンドがクラッシュした際に SIGABRT シグナルを送信する (sighup)	off
send_abort_for_kill	子プロセスがスタックした際に SIGABRT シグナルを送信する (sighup)	off
vacuum_buffer_usage_limit	VACUUM で使用される共有バッファサイズのデフォルト (user)	256kB

□ debug_io_direct

OS に対して Direct I/O の使用を依頼します。ファイルのオープン時に Linux/UNIX では O_DIRECT、macOS では F_NOCACHE、Windows では FILE_FLAG_NO_BUFFERING を指定します。パラメーターには対象となる領域をカンマ (,) 区切りで複数指定します。[[319bae9](#), [d4e71df](#)]

表 23 指定可能な値

設定値	説明
data	メイン・データ・ファイルへの I/O
wal	WAL ファイルへの I/O
wal_init	WAL ファイル初期化のための I/O

本パラメーターは現状では開発者向け (Developer Only) となっています。

□ send_abort_for_crash / send_abort_for_kill

postmaster プロセスは子プロセスのクラッシュ時にまず SIGQUIT シグナルを送信し、その後 SIGKILL シグナルを送信します。send_abort_for_crash と send_abort_for_kill はそれぞれ on に設定することでシグナルを SIGABRT に変更します。[[3226f47](#)]

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。[[9430fb4](#), [6c31ac0](#), [5352ca2](#), [0981846](#)]

表 24 変更されたパラメーター

パラメーター	変更内容
archive_command	archive_library と同時に設定できなくなりました。
archive_library	archive_command と同時に設定できなくなりました。
wal_sync_method	Windows 環境の NTFS 上で fdatasync を指定できるようになりました。
shared_preload_libraries	シングル・ユーザー・モードでも処理されるようになりしました (PostgreSQL 15 へバックポート)。
debug_parallel_query	force_parallel_mode から名前が変更されました。

3.3.3. デフォルト値が変更されたパラメーター

以下のパラメーターはデフォルト値が変更されました。

表 25 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 15	PostgreSQL 16	備考
server_version	15.3	16beta1	
server_version_num	150003	160000	

3.3.4. 削除されたパラメーター

以下のパラメーターは削除されました。 [[cd4329d](#), [1118cd3](#)]

表 26 削除されたパラメーター

パラメーター	理由
vacuum_defer_cleanup_age	妥当な設定値を決定することが難しいことと、代替案があることから削除されました。
promote_trigger_file	5 秒毎のプロセス起動が無駄と判断され、トリガー・ファイルがサポートされなくなったため削除されました。

3.4. ユーティリティの変更

ユーティリティ・コマンドの主な機能拡張点を説明します。

3.4.1. configure

セグメントサイズをブロック単位で指定する `--with-segsize=blocks` オプションが追加されました。 `--with-icu` オプションは廃止され、ICU を使用しない場合に指定する `--without-icu` オプションが追加されました。 [[d3b111e](#), [fcb21b3](#)]

3.4.2. createuser

`createuser` コマンドには複数のオプションが追加されました。 `--role` オプションも引き続き使用できますが、`deprecated` 扱いとなります。 [[08951a7](#), [2dcd157](#), [381d19b](#)]

表 27 追加・変更されたオプション

オプション	短縮形	説明
<code>--with-admin=ROLE</code>	<code>-a</code>	追加ユーザーが所属する ADMIN オプション付きロール
<code>--with-member=ROLE</code>	<code>-m</code>	追加ロールに所属するロール
<code>--valid-until=TIMESTAMP</code>	<code>-v</code>	パスワード有効期限
<code>--bypassrls</code>	<code>-</code>	Bypass RLS 属性を付与
<code>--no-bypassrls</code>	<code>-</code>	Bypass RLS 属性を付与しない
<code>--member-of=ROLE</code>	<code>-g</code>	追加ロールが所属するロール (<code>--role</code> から変更)

例 85 `createuser` コマンドの実行

```
$ createuser --member=role1 user1 --echo
SELECT pg_catalog.set_config('search_path', '', false);
CREATE ROLE user1 NOSUPERUSER NOCREATEDB NOCREATEROLE INHERIT LOGIN ROLE role1;
```

3.4.3. initdb

`initdb` コマンドには以下の拡張が実装されました。 [[27b6237](#), [30a53b7](#), [3e51b27](#), [af3abca](#)]

□ `--locale-provider` オプション

デフォルトのプロバイダーが `libc` から `icu` に変更されました。

□ `--icu-rules` オプション

ICU ロケールに独自のルールを追加する `--icu-rules` が追加されました。

例 86 `initdb` コマンドの実行

```
$ initdb --icu-local=en_US --locale-provider=icu --icu-rules='&a < g' data
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with this locale configuration:
  provider:      icu
  ICU locale:    en_US
  LC_COLLATE:    en_US.UTF-8
  ...
```

□ `--set` オプション

`--set` オプション（短縮形 `-c`）には `postgresql.conf` ファイルに記述されるパラメーターの初期値を指定します。このオプションは複数回指定できます。

例 87 `initdb` コマンドの実行

```
$ initdb --set port=5433 --set listen_addresses='*' --locale-provider=libc --
no-locale --encoding=utf8 data
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "C".
The default text search configuration will be set to "english".

Data page checksums are disabled.
...
```

□ locale コマンドのチェック

locale コマンドが存在しない場合でも initdb コマンドは正常に終了するようになりました。従来は locale -a コマンドの実行が失敗すると initdb コマンドはエラーになっていました。

3.4.4. pgindent

pgindent コマンドには以下のオプションが追加・拡張されました。[[b90f0b5](#), [a1c4cd6](#), [068a243](#), [b16259b](#)]

表 28 追加・変更されたオプション

オプション	変更	説明
--build	削除	pg_bsd_indent のビルド
--code-base	削除	ソースコードの基準ディレクトリ指定
--commit	追加	指定されたコミットで変更されたファイルを対象
--excludes	変更	複数回指定可能
--help	追加	使用方法の出力
--show-diff	追加	変更点を表示する
--silent-diff	追加	変更点があった場合には戻り値 2 で終了する

3.4.5. pg_basebackup

Zstandard 圧縮メソッドのレベルに long を指定できるようになりました。このオプションは大規模データの圧縮率を向上させますが、リソース使用量が大きくなります。
[[2820adf](#)]

例 88 圧縮レベル long の指定

```
$ pg_basebackup -D data.zstd --compress=zstd:long --format=tar
$ file data.zstd/*
data.zstd/backup_manifest: ASCII text
data.zstd/base.tar.zst:      Zstandard compressed data (v0.8+), Dictionary ID:
None
data.zstd/pg_wal.tar:       POSIX tar archive
```

3.4.6. pg_bsd_indent

別プロジェクトで開発されていた BSD スタイルのインデントツール `pg_bsd_indent` が
コア・リポジトリに取り込まれました。[[4e831f4](#)]

例 89 `pg_bsd_indent` コマンドの実行

```
$ pg_bsd_indent -v indent.c
There were 1273 output lines and 223 comments
(Lines with comments)/(Lines with code): 0.548
```

3.4.7. pg_dump

`pg_dump` コマンドには以下の拡張が実装されました。[[5e73a60](#), [0da243f](#), [84adc8e](#),
[2820adf](#), [35ce24c](#), [2f80c95](#), [a563c24](#), [5f53b42](#)]

□ `--compress` オプション

`--compress` オプションには圧縮メソッドと圧縮レベルをコロン (:) で区切って指定できる
ようになりました。圧縮メソッドに指定できるのは `none`、`gzip`、`zstd` または `lz4` です。
圧縮メソッドによって圧縮レベルに指定できる値は異なります。

表 29 圧縮レベルの範囲

圧縮メソッド	デフォルト	圧縮レベルの範囲	備考
none	-	-	
gzip	-1	1～9	
lz4	0	1～12	
zstd	3	-131072～22, long	



例 90 pg_dump コマンドの実行

```
$ pg_dump --compress=zip:9 --file=postgres.gz postgres
$ file postgres.gz
postgres.gz: gzip compressed data, max compression, from Unix, original size
11963
$ pg_dump --compress=lz4:12 --file=postgres.lz4 postgres
$ file postgres.lz4
postgres.lz4: LZ4 compressed data (v1.4+)
$ pg_dump --compress=zstd:22 --file=postgres.zstd postgres
$ file postgres.zstd
postgres.zstd: Zstandard compressed data (v0.8+), Dictionary ID: None
```

□ --large-objects オプション

前バージョンの--blob と--no-blob オプションは--large-objects と--no-large-objects に変更されました。従来のオプションも利用できますが、deprecated 扱いとなります。

例 91 pg_dump ヘルプ・メッセージ

```
$ pg_dump --help
pg_dump dumps a database as a text file or to other formats.
...
Options controlling the output content:
-a, --data-only          dump only the data, not the schema
-b, --large-objects      include large objects in dump
--blobs                  (same as --large-objects, deprecated)
-B, --no-large-objects   exclude large objects in dump
--no-blobs               (same as --no-large-objects, deprecated)
-c, --clean              clean (drop) database objects before recreating
...
```

□ --table-and-children オプション

基本的には--table オプションと同じです。パターンに合致するテーブルとその子テーブルやパーティションを含みます。



□ `--exclude-table-and-children` オプション

基本的には`--exclude-table` オプションと同じです。パターンに合致するテーブルとその子テーブルやパーティションを除外します。

□ `--exclude-table-data-and-children` オプション

基本的には`--exclude-table-data` オプションと同じです。パターンに合致するテーブルとその子テーブルやパーティションのデータを除外します。

□ ロック取得

`pg_dump` コマンドは起動時に対象テーブルに対して `LOCK TABLE` 文を実行します。PostgreSQL 16 では1回の `LOCK TABLE` に対して複数テーブル名を記述して、クライアントとサーバー間の通信量を削減します。

3.4.8. `pg_receivewal` / `pg_recvlogical`

プログラムの終了条件として従来の `SIGINT` シグナル以外に `SIGTERM` シグナルが追加されました。これは `systemd` の終了シグナル (`KillSignal`) のデフォルト値に対応する変更です。[[8b60db7](#)]

3.4.9. `pg_upgrade`

オプション`--copy` が追加されました。これはデフォルトの動作です。またロケールとエンコードの情報を移行元から移行先にコピーするようになりました。[[746915c](#), [9637bad](#)]

3.4.10. `pg_verifybackup`

オプション`--progress` (短縮形 `-P`) が追加されました。進行状況をレポートします。[[d07c294](#)]

例 92 pg_verifybackup コマンドの実行

```
$ pg_basebackup -D back.1 -v
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/5E000028 on timeline 1
pg_basebackup: starting background WAL receiver
pg_basebackup: created temporary replication slot "pg_basebackup_120380"
pg_basebackup: write-ahead log end point: 0/5E0001B0
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: syncing data to disk ...
pg_basebackup: renaming backup_manifest.tmp to backup_manifest
pg_basebackup: base backup completed
$ pg_verifybackup back.1 --progress
887802/887802 kB (100%) verified
backup successfully verified
```

3.4.11. pg_waldump

pg_waldump コマンドには以下の新機能が追加されました。[[d497093](#), [4c8044c](#)]

□ --save-fullpage オプション

全ページのイメージを保存するディレクトリを指定する--save-fullpage オプションが追加されました。指定したディレクトリは空である必要があります。フルページのイメージは以下のファイル名で作成されます。

```
<lsn>.<tablespace>.<database>.<relation>.<block>_<forkname>
```

表 30 ファイル名の説明

表記	説明	例
<lsn>	LSN の 16 進数フォーマット	00000000-05AD6E48
<tablespace>	テーブルスペースの OID	1663
<database>	データベースの OID	5
<relation>	リレーション filenode	16388
<block>	ブロック番号	20
<forkname>	フォーク名	main



例 93 --save-fullpage オプションの指定

```
$ pg_walddump --save-fullpage=$HOME/fullpage
data/pg_wal/00000001000000000000000004
rmgr: Heap          len (rec/tot):    65/    65, tx:          740, lsn: ...
rmgr: Heap          len (rec/tot):    65/    65, tx:          740, lsn: ...
...
$ ls -l $HOME/fullpage
00000000-05A26608.1663.5.16388.0_vm
00000000-05A7D2E8.1663.5.2619.18_main
00000000-05A7E920.1663.5.2696.1_main
00000000-05A80B20.1663.5.16388.0_main
00000000-05A853B0.1663.5.16388.1_main
```

□ --timeline オプション

--timeline オプションには 16 進数でも指定できるようになりました。

例 94 --timeline オプションの指定

```
$ pg_walddump 00000001000000000000000001 --timeline=0xA
rmgr: XLOG          len (rec/tot):    114/    114, tx:          0, lsn:
0/01000028, prev 0/00000000, desc: CHECKPOINT_SHUTDOWN redo 0/1000028; tli 1;
pr
ev tli 1; fpw true; xid 0:3; oid 10000; multi 1; offset 0; oldest xid 3 in DB
1; oldest multi 1 in DB 1; oldest/newest commit timestamp xid: 0/0; old
...
```

3.4.12. psql

psql コマンドには以下の拡張が実装されました。[[b0d8f2d](#), [31ae2aa](#), [bd95816](#), [5b66de3](#), [a45388d](#), [d913928](#), [6f9ee74](#), [31ae2aa](#)]

□ コマンド実行ステータス

¥!メタコマンドおよび¥g、¥o、¥w、¥copy メタコマンドの実行ステータスが以下の変数で取得できるようになりました。

表 31 コマンド実行ステータス

変数名	説明
SHELL_ERROR	コマンド実行結果が 0 の場合 true、0 以外の場合 false
SHELL_EXIT_CODE	コマンドの終了コード

例 95 コマンド実行ステータスの確認

```
postgres=> ¥! ls postgresql.conf
postgresql.conf
postgres=> ¥echo :SHELL_ERROR
false
postgres=> ¥echo :SHELL_EXIT_CODE
0
postgres=> ¥! ls postgresql.conf.bad
ls: cannot access 'postgresql.conf.bad': No such file or directory
postgres=> ¥echo :SHELL_ERROR
true
postgres=> ¥echo :SHELL_EXIT_CODE
2
```

□ ¥d+メタコマンド

パーティション・テーブルに対して¥d+メタコマンドを実行すると、パーティションが外部テーブルである場合に「FOREIGN」が出力されます。

例 96 外部テーブルのパーティション情報

```
postgres=> CREATE FOREIGN TABLE remote1(c1 INT, c2 VARCHAR(10)) SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=> CREATE TABLE part1(c1 INT, c2 VARCHAR(10)) PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> ALTER TABLE part1 ATTACH PARTITION remote1 FOR VALUES FROM (0) TO
(100) ;
ALTER TABLE
postgres=> \d+ part1
```

Partitioned table "public.part1"

Column	Type	Collation	Nullable	Default	Storage	...
c1	integer				plain	...
c2	character varying(10)				extended	...

Partition key: RANGE (c1)
Partitions: remote1 FOR VALUES FROM (0) TO (100), FOREIGN

□ \bind メタコマンド

SQL 文内のバインド変数に値をセットする\bind メタコマンドが追加されました。セミコロンの (;) は行終端とみなされないため、\g 等を指定します。

例 97 バインド変数の設定

```
postgres=> SELECT $1, $2
postgres=> \bind 'value1' 'value2' \g
?column? | ?column?
-----+-----
value1   | value2
(1 row)

postgres=> SELECT * FROM data1 WHERE c1=$1 \bind 100 \g
c1 | c2
---+---
100 | data1
(1 row)
```

□ **¥pset** メタコマンド

¥pset メタコマンドに指定できるオプション **xheader_width** が追加されました。このオプションは拡張テーブル形式で出力する場合ヘッダの幅を制限できます。指定できる値は以下の通りです。

表 32 指定できる値

設定値	説明	備考
full	最も幅の広い行の長さ（デフォルト）	
column	最初の列の幅に切り捨て	
page	ターミナル・ページの幅に切り捨て	
数字	指定された数値に切り捨て	

例 98 xheader_width オプションの指定

```
postgres=> ¥x
Expanded display is on.
postgres=> CREATE TABLE data1(c1 char(10), c2 char(20)) ;
CREATE TABLE
postgres=> INSERT INTO data1 VALUES ('1234567890', '12345678901234567890') ;
INSERT 0 1
postgres=> ¥pset xheader_width
Expanded header width is 'full'.
postgres=> SELECT * FROM data1 ;
-[ RECORD 1 ]-----
c1 | 1234567890
c2 | 12345678901234567890

postgres=> ¥pset xheader_width column
Expanded header width is 'column'.
postgres=> SELECT * FROM data1 ;
-[ RECORD 1 ]
c1 | 1234567890
c2 | 12345678901234567890
```



□ `¥dpS` メタコマンド

`¥dp` メタコマンドと`¥z` メタコマンドに'`S`'が指定できるようになりました。これに伴い旧バージョンから`¥dp` または`¥z` コマンドの動作が変更されました。一時オブジェクトが一覧に含まれ、`information_schema` スキーマのオブジェクトが対象外になりました。

□ `¥watch` メタコマンド

入力された値のチェックが追加されました。数値以外の値や負の値はエラーになります。ゼロ (0) を指定するとインターバル無しで再実行されます。また再実行回数を指定できるようになりました。再実行回数を制限する場合はインターバルと実行回数を「`i=インターバル c=実行回数`」のフォーマットで指定します。

例 99 `¥watch` メタコマンドの拡張

```
postgres=> SELECT current_time ;
      current_time
-----
16:58:33.012005+09
(1 row)

postgres=> ¥watch i=1 c=2
Fri 26 May 2023 04:58:34 PM JST (every 1s)

      current_time
-----
16:58:34.850061+09
(1 row)

Fri 26 May 2023 04:58:35 PM JST (every 1s)

      current_time
-----
16:58:35.850556+09
(1 row)
```



3.4.13. vacuumdb

vacuumdb コマンドには以下のオプションが追加されました。[[7781f4e](#), [4211fbd](#), [ae78cae](#)]

□ --schema オプション

VACUUM を実行するテーブルが所属するスキーマ名を指定します。短縮形は-n です。このオプションは複数回指定できます。--exclude-schema オプションと同時に使用できません。

□ --exclude-schema オプション

VACUUM を実行しないテーブルが所属するスキーマ名を指定します。短縮形は-N です。--schema オプションとは同時に使用できません。

例 100 スキーマ指定の VACUUM

```
$ vacuumdb -d postgres --schema=public
vacuumdb: vacuuming database "postgres"
$ vacuumdb -d postgres --exclude-schema=public
vacuumdb: vacuuming database "postgres"
$ vacuumdb -d postgres --schema=public --exclude-schema=public
vacuumdb: error: cannot vacuum all tables in schema(s) and exclude schema(s) at
the same time
```

□ --no-process-main オプション

VACUUM (PROCESS_MAIN FALSE) 文に対応するオプション--no-process-main が追加されました。



例 101 `--no-process-main` オプション

```
$ vacuumdb --no-process-main --echo postgres
SELECT pg_catalog.set_config('search_path', '', false);
vacuumdb: vacuuming database "postgres"
RESET search_path;
SELECT c.relname, ns.nspname FROM pg_catalog.pg_class c
JOIN pg_catalog.pg_namespace ns ON c.relnamespace OPERATOR(pg_catalog.=)
ns.oid
LEFT JOIN pg_catalog.pg_class t ON c.reltoastrelid OPERATOR(pg_catalog.=) t.oid
WHERE c.relkind OPERATOR(pg_catalog.=) ANY (array['r', 'm'])
ORDER BY c.relpages DESC;
SELECT pg_catalog.set_config('search_path', '', false);
VACUUM (PROCESS_MAIN FALSE, SKIP_DATABASE_STATS) pg_catalog.pg_proc;
...
```

□ `--buffer-usage-limit` オプション

VACUUM (BUFFER_USAGE_LIMIT) 文に対応するオプション `--buffer-usage-limit` が追加されました。

例 102 `--buffer-usage-limit` オプション

```
$ vacuumdb --buffer-usage-limit=1MB --echo postgres
SELECT pg_catalog.set_config('search_path', '', false);
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. auto_explain

auto_explain には以下の機能が追加されました。[[d4bfe41](#), [9d2d972](#)]

□ パラメーターlog_parameter_max_length

パラメーターlog_parameter_max_length が追加されました。このパラメーターはログ出力の最大値を制限します。デフォルト値は-1 で制限を設けません。0 に指定するとロギングを無効にします。

□ Query ID の出力

パラメーターauto_explain.log_verbose を on に指定し、かつ compute_query_id を on に指定した場合 Query ID が出力されます。

例 103 Query ID の出力

```
$ tail -5 data/log/postgresql-2023-05-26_135719.log
LOG:  duration: 0.005 ms  plan:
      Query Text: SELECT 1;
      Result  (cost=0.00..0.01 rows=1 width=4)
      Output: 1
      Query Identifier: -1801652217649936326
```

3.5.2. fuzzystmatch

daitch_mokotoff 関数が追加されました。この関数は英語以外の言語において SOUNDEX 関数よりも優れています。[[a290378](#)]

例 104daitch_mokotoff 関数の実行

```
postgres=> SELECT daitch_mokotoff('Müller') ;
daitch_mokotoff
-----
{479000}
(1 row)
```



3.5.3. ltree

ラベルにハイフン (-) を利用できるようになりました。以前はアルファベット、数字、アンダーライン (_) のみ許可されていました。ラベルの最大サイズが 255 文字から 1,000 文字に拡大されました。[[b1665bf](#)]

3.5.4. pageinspect

pageinspect には以下の機能が追加されました。[[1fd3dd2](#), [3581cbd](#), [428c0ca](#)]

□ bt_multi_page_stats 関数

B-Tree インデックス・ページの要約を出力する関数 `bt_multi_page_stats` が追加されました。`bt_page_stats` 関数の複数ページ版です。先頭ブロック番号とページ数を指定します。

例 105 bt_multi_page_stats 関数の実行

```
postgres=# SELECT * FROM bt_multi_page_stats('idx1_data1', 2, 1) ;
-[ RECORD 1 ]-+-----
 blkno          | 2
 type           | I
 live_items     | 367
 dead_items     | 0
 avg_item_size  | 16
 page_size      | 8192
 free_size      | 808
 btpo_prev      | 1
 btpo_next      | 4
 btpo_level     | 0
 btpo_flags     | 1
```

□ brin_page_items 関数

`brin_page_items` 関数の出力に空きブロックを示す「empty」列が追加されました。

3.5.5. pg_buffercache

`pg_buffercache` モジュールには以下の関数が追加されました。[[f3fa313](#), [f3fa313](#)]

□ pg_buffercache_summary 関数

バッファキャッシュのサマリーを SQL 文で確認できる pg_buffercache_summary 関数が追加されました。

例 106 pg_buffercache_summary 関数の実行

```
postgres=# SELECT * FROM pg_buffercache_summary();
-[ RECORD 1 ]--+-----
buffers_used   | 251
buffers_unused | 16133
buffers_dirty  | 62
buffers_pinned | 0
usagecount_avg | 3.3346613545816735
```

表 33 pg_buffercache_summary 実行結果

列名	説明	備考
buffers_used	使用されているバッファ総数	
buffers_unused	使用されていないバッファ数	
buffers_dirty	ダーティバッファ数	
buffers_pinned	ピン留めされているバッファ数	
usagecount_avg	使用済バッファの平均使用量	

□ pg_buffercache_usage_counts 関数

すべての共有バッファの状態の要約を返します。pg_buffercache ビューを集計するよりも簡易に情報を確認できます。

例 107 pg_buffercache_usage_counts 関数の実行

```
postgres=# SELECT * FROM pg_buffercache_usage_counts();
usage_count | buffers | dirty | pinned
-----+-----+-----+-----
0 | 16035 | 0 | 0
1 | 76 | 6 | 0
2 | 31 | 1 | 0
3 | 13 | 6 | 0
4 | 8 | 3 | 0
5 | 221 | 48 | 0
(6 rows)
```

3.5.6. pg_stat_statements

SQL 文の一般化方法が更新されました。SET 文、CHECKPOINT 文、CREATE 文等が大文字／小文字の区別なく同一のクエリーID として収集されます。[[9ba37b2](#), [daa8365](#)]

例 108 pg_stat_statements ビューの検索

```
postgres=> SET enable_seqscan = off ;
SET
postgres=> set enable_seqscan = off ;
SET
postgres=> CHECKPOINT;
CHECKPOINT
postgres=> checkpoint;
CHECKPOINT
postgres=> SELECT queryid, query FROM pg_stat_statements WHERE userid=16384 ;
      queryid      |      query
-----+-----
-4537744274534392533 | SET enable_seqscan = off
-7184411855217750948 | CHECKPOINT
(2 rows)
```

3.5.7. pg_walinspect

pg_get_wal_block_info 関数が追加されました。この関数は指定された LSN 間のフルページイメージを抽出します。[[c31cf1c](#), [9ecb134](#), [122376f](#)]

構文

```
record pg_get_wal_block_info(start_lsn pg_lsn, end_lsn pg_lsn, show_data
boolean)
```

例 109 pg_get_wal_block_info 関数の実行

```
postgres=# SELECT lsn, relfilenode, relblocknumber, forkname FROM
           pg_get_wal_block_info('0/150A0A0', '0/23413F0', false) ;
   lsn      | relfilenode | relblocknumber | forkname
-----+-----+-----+-----
0/150A0B0 |      2619 |           0 | main
0/150C0B0 |      2619 |          16 | main
...
```

この関数を実行すると以下の情報が返ります。

表 34 関数の戻り値

列名	データ型	説明	備考
start_lsn	pg_lsn	開始 LSN	
end_lsn	pg_lsn	終了 LSN	
prev_lsn	pg_lsn	前 LSN	
block_id	smallint	ブロック ID	
reftablespace	oid	テーブル空間 OID	
reldatabase	oid	データベース OID	
relfilenode	oid	ファイル OID	
relforknumber	smallint	フォーク番号	
relblocknumber	bigint	テーブルのブロック番号	
xid	xid	トランザクション ID	
resource_manager	text	リソース・マネージャ名	
record_type	text	WAL レコードのタイプ	
record_length	integer	WAL レコード長	
main_data_length	integer	メインデータ長	
block_data_length	integer	ブロックデータ長	
block_fpi_length	integer	ブロック FPI データ長	
block_fpi_info	text[]	ブロック FPI 情報	
description	text	説明	
block_data	bytea	ブロックデータ	
block_fpi_data	bytea	FPI データ	

3.5.8. postgres_fdw

postgres_fdw には以下の拡張が実装されました。[[97da482](#), [8ad51b5](#), [983ec23](#)]

□ COPY FROM 文

COPY FROM 文で外部テーブルの batch_size オプションが利用されるようになりました。下記は実テーブルを保有するインスタンスの、log_statement パラメーターを利用したログです。複数レコードが一括で挿入されていることがわかります。

例 110 外部テーブル側の実行ログ

```
LOG:  statement: START TRANSACTION ISOLATION LEVEL REPEATABLE READ
LOG:  statement: TRUNCATE public.data1 CONTINUE IDENTITY RESTRICT
LOG:  statement: COMMIT TRANSACTION
LOG:  statement: START TRANSACTION ISOLATION LEVEL REPEATABLE READ
LOG:  execute pgsql_fdw_prep_7: INSERT INTO public.data1(c1, c2) VALUES ($1,
$2), ($3, $4), ($5, $6), ($7, $8)
DETAIL:  parameters: $1 = '1', $2 = 'data1', $3 = '2', $4 = 'data2', $5 = '3',
$6 = 'data3', $7 = '4', $8 = 'data4'
LOG:  statement: DEALLOCATE pgsql_fdw_prep_7
LOG:  statement: COMMIT TRANSACTION
```

□ ANALYZE 文

FOREIGN SERVER または FOREIGN TABLE のオプションに analyze_sampling が追加されました。従来のバージョンでは統計情報の取得のために外部テーブルのレコード全体を取得していました。外部テーブルの情報取得時に TABLESAMPLE 句を使用することでリモートデータベースの負荷を削減することができます。デフォルト値は auto で、PostgreSQL 9.5 未満では random、PostgreSQL 9.5 以上では bernoulli とみなされます。設定できる値は以下の通りです。

表 35 analyze_sampling オプションの設定値

設定値	説明
off	サンプリングを行いません。
auto	バージョンによってサンプリング方法を決定します（デフォルト値）
random	random 関数を使ってサンプリングを行います。
system	TABLESAMPLE SYSTEM 句を使ってサンプリングを行います。
bernoulli	TABLESAMPLE BERNOULLI 句を使ってサンプリングを行います。



例 111 ANALYZE 文の実行

```
postgres=> CREATE FOREIGN TABLE foreign1(c1 INT, c2 VARCHAR(10)) SERVER remsvr1
           OPTIONS (analyze_sampling 'bernoulli', table_name 'data1') ;
CREATE FOREIGN TABLE
postgres=> ANALYZE VERBOSE foreign1 ;
INFO:  analyzing "public.foreign1"
INFO:  "foreign1": table contains 1000000 rows, 30000 rows in sample
ANALYZE
```

外部テーブルの元テーブルで ANALYZE 文が実行されていない場合にはサンプリングは行われません。

□ parallel_abort オプション

postgres_fdw はローカル・トランザクションを中止した場合、リモート・トランザクションを1つずつ中止します。オプション parallel_abort を true に設定することでリモート・トランザクションを平行に中止し、パフォーマンスを向上します。このオプションのデフォルト値は false です。



参考にした URL

本資料の作成には、以下の URL を参考にしました。

- Release Notes
<https://www.postgresql.org/docs/16/release-16.html>
- Commitfests
<https://commitfest.postgresql.org/>
- PostgreSQL 16 Manual
<https://www.postgresql.org/docs/16/index.html>
- PostgreSQL 16 Open Items
https://wiki.postgresql.org/wiki/PostgreSQL_16_Open_Items
- Git
[git://git.postgresql.org/git/postgresql.git](https://git.postgresql.org/git/postgresql.git)
- GitHub
<https://github.com/postgres/postgres>
- PostgreSQL 16 Beta 1 のアナウンス
<https://www.postgresql.org/about/news/postgresql-16-beta-1-released-2643/>
- Michael Paquier - PostgreSQL committer
<https://paquier.xyz/>
- Qiita (ぬこ@横浜さん)
http://qiita.com/nuko_yokohama
- pgsql-hackers Mailing list
<https://www.postgresql.org/list/pgsql-hackers/>
- PostgreSQL Developer Information
https://wiki.postgresql.org/wiki/Development_information
- pgPedia
<https://pgpedia.info/postgresql-versions/postgresql-16.html>
- SQL Notes
<https://sql-info.de/postgresql/postgresql-16/articles-about-new-features-in-postgresql-16.html>
- Slack - postgresql-jp (Japanese)
<https://postgresql-jp.slack.com/>

変更履歴

変更履歴

版	日付	作成者	説明
0.1	2023/04/25	篠田典良	内部レビュー版作成 レビュー担当（敬称略）： 高橋智雄 竹島彰子 （日本ヒューレット・パカード合同会社）
1.0	2023/05/28	篠田典良	PostgreSQL 16 Beta 1 公開版に合わせて修正完了

以上

