2015 年 8 月 7 日

PostgreSQL 9.5 新機能検証結果

日本ヒューレット・パッカード株式会社
篠田典良

目次

1. 本文書について

1.1 本文書の概要

本文書は現在ベータ版が公開されているオープンソース RDBMS である PostgreSQL 9.5 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 9.4 と PostgreSQL 9.5 Alpha 2 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。内部動作の変更によるパフォーマンス向上等については調査の対象としていません。すべての新機能について検証しているわけではありません。

1.4 本文書の対応バージョン

本文書は原則として以下のバージョンを対象としています。

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 9.4.4 (比較対象) PostgreSQL 9.5 Alpha 2 (2015/8/3 8:41 p.m.)
オペレーティング・システム	Red Hat Enterprise Linux 7 Update 1 (x86-64)

1.5 本文書に対する質問・意見および責任

本文書の内容は日本ヒューレット・パッカード株式会社の公式見解ではありません。また内容の間違いにより生じた問題について作成者および所属企業は責任を負いません。ご意見等ありましたら本文書作成者 篠田典良（noriyoshi.shinoda@hp.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 プロンプト
postgres=>	PostgreSQL 一般ユーザーが利用する psql プロンプト
backend>	スタンドアロン・モードのプロンプト
下線部	特に注目すべき項目
<<以下省略>>	より多くの情報が出力されるが文書内では省略していることを示す
<<途中省略>>	より多くの情報が出力されるが文書内では省略していることを示す

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. 新機能概要

PostgreSQL 9.5 は多くの新機能や改善が行われました。

2.1 パフォーマンスの改善

以下の部分でパフォーマンスが改善されました。

- 集計関数に対する 128 ビット整数の使用
- GiST インデックスによる Index-Only Scan の実装
- BTree インデックスのロック削減
- CRC 計算アルゴリズムの改善
- text 型、numeric 型のソート処理の高速化
- KNN-GiST の高速化
- local xmin の積極的な更新
- Abbreviated Keys.
- その他

2.2 機能の追加

以下に主な追加機能を列挙します。() 内はより詳細が記載された本文書内の章番号です。

- Row Level Security（3.5）
- BRIN インデックス（3.6）
- Full Page Write 時の WAL 圧縮（3.1.2）
- プロセス名とデータベースクラスタ対応（3.7.1）
- pg_rewind コマンド（3.2.1）
- 各種ユーティリティの改善（3.2）
- contrib モジュールの追加（3.1.5）
- EXPLAIN 文によるソート情報の追加（3.7.2）
- レプリケーション関連ログ出力（3.7.3）
- スタンバイ・アクション（3.3.5）
- PL/pgSQL の ASSERT 文（3.4.14）
- パラメーターcheckpoint_segments の分割（3.3.4）
- その他

2.3 SQL 文の変更

以下の SQL 文がサポートされるようになりました。() 内はより詳細が記載された本文書内の章番号です。

- INSERT ON CONFLICT（3.4.1）
- ALTER TABLE SET UNLOGGED（3.4.2）
- IMPORT FOREIGN SCHEMA（3.4.3）
- SEELECT SKIP LOCKED（3.4.4）
- CREATE FOREIGN TABLE INHERITS（3.4.5）
- ALTER USER CURRENT_USER（3.4.6）
- UPDATE SET 拡張（3.4.7）
- TABLESAMPLE（3.4.8）
- REINDEX SCHEMA（3.4.9）
- REINDEX (VERBOSE)（3.4.10）
- GROUPING SETS, CUBE, ROLLUP（3.4.11）
- CREATE POLICY (3.5)
- CREATE FOREIGN TABLE (CHECK)（3.4.12）
- CREATE EVENT TRIGGER 拡張（3.4.13）
- jsonb 演算子の追加（3.4.15）
- CREATE | ALTER | COMMENT ON TRANSFORM
- CREATE | ALTER DATABASE ALLOW_CONNECTIONS
- CREATE | ALTER DATABASE IS_TEMPLATE
- IF NOT EXISTS 句が複数の CREATE 文で使用可能
- その他

その他の改善点は、PostgreSQL 9.5 Documentation Appendix E. Release Notes（http://www.postgresql.org/docs/9.5/static/release-9-5.html）に記載されています。

3. 新機能解説

3.1 アーキテクチャの変更

3.1.1 WAL フォーマットの変更

WAL ファイルの出力フォーマットが変更されました。新規に XLogRecordBlockHeader 構造体（src/include/access/xlogrecord.h で定義）が定義され、WAL ファイルの出力に使用されます。

3.1.2 WAL 圧縮

パラメーターwal_compression を on に指定すると、Full Page Write 時（CHECKPOINT 完了後の最初の更新時）の WAL が圧縮されて書き込まれます。圧縮はソースコード src/common/pg_lzcompress.c 内の pglz_compress 関数で実装されている PGLZ と呼ばれる方法で行われます。

パラメーターwal_compression のデフォルト値は off であるため、標準ではこの機能は動作しません。

3.1.3 カタログの追加

機能追加に伴い、以下のシステムカタログが追加されています。

表 4 追加されたシステムカタログ一覧

カタログ名	説明
pg_file_settings	パラメーター・ファイル設定情報
pg_policy	ポリシー情報
pg_policies	ポリシー適用テーブル情報
pg_replication_origin	詳細不明
pg_replication_origin_status	詳細不明
pg_stat_ssl	SSL 接続情報
pg_transform	トランスフォーム情報

□ pg_file_settings カタログ

パラメーター・ファイル（postgresql.conf と postgresql.auto.conf）に記述されたパラメーター情報が格納されます。このカタログの実体は pg_show_all_file_settings 関数です。

表 5 pg_file_settings カタログ

列名	データ型	説明
sourcefile	text	パラメーター設定ファイル名（フルパス）
sourceline	integer	ファイル内の行番号
seqno	integer	複数ファイル全体を通した通番
name	text	パラメーター名
setting	text	パラメーター値
applied	boolean	適用済みであるかを示す
error	text	適用エラーを示す文字列

このカタログには以下の特徴があります。

- pg_file_settings カタログに対する検索が行われるたびに、パラメーター・ファイルが解析されます。
- 検索には SUPERUSER 権限が必要です。
- postgresql.conf ファイルの include 構文にも対応しています。

例 1 リロードとカタログ

```
postgres=# ALTER SYSTEM SET max_connections = 1000 ;
ALTER SYSTEM
postgres=# SELECT sourcefile, name, setting FROM pg_file_settings  WHERE
                name = 'max_connections' ;
            sourcefile                    |      name          | setting
------------------------------------------+--------------------+---------
 /usr/local/pgsql/data/postgresql.auto.conf | max_connections | 1000
(1 row)
```

□ pg_policy カタログ

Row Level Security 機能で使用する POLICY オブジェクトの情報を提供します。このカタログはデータベース単位に作成されます。

表 6　pg_policy カタログ

列名	データ型	説明
polname	name	ポリシー名
polrelid	oid	ポリシーが適用されたテーブルの OID
polcmd	char	ポリシー制限された動作 ● SELECT = r ● INSERT = a ● UPDATE = w ● DELETE = d ● ALL = *
polroles	oid[]	ポリシーが適用されたロール OID の配列
polqual	pg_node_tree	USING 構文でチェックされる条件
polwithcheck	pg_node_tree	WITH CHECK 構文でチェックされる条件

□　pg_policies カタログ

ポリシーが適用されたテーブルの情報です。このカタログはデータベース単位に作成されます。

表 7　pg_policies カタログ

列名	データ型	説明
schemaname	name	ポリシー適用テーブルのスキーマ名
tablename	name	ポリシー適用テーブル名
policyname	name	ポリシー名
roles	name[]	ポリシー対象ロール
cmd	text	ポリシー対象 DML
qual	text	ポリシーをチェックする WHERE 句
with_check	text	WITH チェック句

□　pg_stat_ssl カタログ

インスタンスに接続しているセッションの SSL 情報を取得できます。

表 8　pg_stat_ssl カタログ

列名	データ型	説明	備考
pid	integer	バックエンド・プロセス ID	pg_stat_activity.pid と同じ
ssl	boolean	SSL 接続を行っているか	
version	text	バージョン情報	
cipher	text	サイファ情報	
bits	integer	ビット情報	
compression	boolean	圧縮を行っているか	
clientdn	text	DN 情報	

□　pg_transform カタログ

このカタログには CREATE TRANSFORM 文で作成したトランスフォーム情報が格納されています。CREATE TRANSFORM 文および本カタログの詳細は未検証です。

表 9　pg_transform カタログ

列名	データ型	説明
trftype	oid	TRANSFORM を適用するデータ型の OID
trflang	oid	TRANSFORM を適用する LANGUAGE の OID
trffromsql	regproc	データ変換を行う入力ファンクションの OID
trftosql	regproc	データ変換を行う出力ファンクションの OID

3.1.4 カタログの変更

以下のカタログが変更されました。

表 10　変更されたシステムカタログ一覧

カタログ名	説明	変更点
pg_replication_slots	レプリケーション・スロット情報	active_pid 列追加
pg_settings	パラメーター設定情報	pending_restart 列追加
pg_stat_statements	SQL 実行統計の収集（contrib）	統計情報列追加
pg_tables	テーブル情報	rowsecurity 列追加
pg_authid	ロールの権限情報	rolcatupdate 列削除 rolbypassrls 列追加

□ pg_replication_slots カタログ

wal sender プロセスのプロセス ID が追加されました。これにより pg_stat_replication カタログとの結合が可能になりました。

表 11 pg_replication_slots カタログの追加列

列名	データ型	説明
active_pid	integer	wal sender プロセスのＩＤ

下記の例では pg_stat_replication カタログと pg_replication_slots カタログを結合してスレーブ・インスタンスのホスト名とスロット名の対応を表示しています。スロットを使用しない場合を想定して LEFT OUTER JOIN にしています。

例 2 pg_stat_replication カタログと pg_replication_slots の結合

```
postgres=> SELECT r.client_hostname, r.pid, s.slot_name
        FROM pg_stat_replication r
        LEFT OUTER JOIN pg_replication_slots s ON r.pid = s.active_pid ;
 client_hostname |  pid  | slot_name
-----------------+-------+-----------
 dbslv1          | 12915 | slot_1
 dbslv2          | 12934 |
(2 rows)
```

□ pg_settings カタログ

pending_restart 列が追加されました。変更は行われたが、再起動待ちになっているパラメーターを確認できます。この列の値はパラメーター・ファイルを変更するか、ALTER SYSTEM 文を実行した後、ファイルのリロードを行った場合に true になります。ALTER SYSTEM 文を実行しただけでは値は変化しません。

pending_restart 列は一度 true になると、該当パラメーターを現在の値に戻しても元に戻りません。

表 12 pg_settings カタログに追加された列

列名	データ型	説明
pending_restart	boolean	変更されたが再起動待ちになっているか

□ pg_stat_statements カタログ

Contrib モジュール pg_stat_statements を登録すると作成される pg_stat_statements カタログに SQL 文実行時の最大、最小、平均、標準偏差の実行時間を示す列が追加されました。

表 13 pg_stat_statements カタログの追加列

列名	データ型	説明
min_time	double precision	SQL 文の最小実行時間
max_time	double precision	SQL 文の最大実行時間
mean_time	double precision	SQL 文の平均実行時間
stddev_time	double precision	SQL 文の標準偏差

□ pg_tables カタログ

テーブル情報を提供する pg_tables カタログに Row Level Security 機能を利用しているかを示す rowsecurity 列が追加されました。この列は ENABLE ROW LEVEL SECURITY の設定が行われたテーブルに対して true になります。ポリシー設定のみの場合は false です。

表 14 pg_tables カタログの追加列

列名	データ型	説明
rowsecurity	boolean	Row Level Security 機能を利用しているか

3.1.5 Contrib モジュールの変更

PostgreSQL 9.5 では、いくつかの Contrib モジュールが変更されました。従来から広く使われていたいくつかのプログラムが Contrib モジュールから PostgreSQL 本体に移動されています。

表 15　Contrib モジュールの変更点

モジュール	変更点	備考
hstore_plperl	モジュール追加	
hstore_plpython	モジュール追加	
ltree_plpython	モジュール追加	
tsm_system_rows	モジュール追加	
tsm_system_time	モジュール追加	
pg_archivecleanup	PostgreSQL 本体へ移動	
pg_test_fsync	PostgreSQL 本体へ移動	
pg_test_timing	PostgreSQL 本体へ移動	
pg_upgrade	PostgreSQL 本体へ移動	
pg_xlogdump	PostgreSQL 本体へ移動	
pgbench	PostgreSQL 本体へ移動	
test_parser	src/test/modules へ移動	ソースコードの保存ディレクトリ
test_shm_mq	src/test/modules へ移動	ソースコードの保存ディレクトリ
worker_spi	src/test/modules へ移動	ソースコードの保存ディレクトリ
dummy_seclabel	src/test/modules へ移動	ソースコードの保存ディレクトリ
pg_stat_statements	機能追加	統計情報列の追加
pageinspect	機能追加	関数の追加
pgcrypto	機能追加	関数の追加
pg_buffercache	機能追加	表示の追加

3.1.6 インターフェース／API／フック

PostgreSQL を拡張するための基盤が整備されました。

□　パラレル処理基盤

パラレル処理を行うための API が整理されました。動的共有メモリーやワーカープロセスの使用方法等の説明が「src/backend/access/transam/README.parallel」に記載されています。

□　Custom Scan/Join Interface

実行計画を置き換えるためのフック関数をコールする機能が提供されました。以下のフックが追加されました。

表 16　フックの追加

フック	説明	定義ソース
set_rel_pathlist_hook	カスタム･スキャンを行うフック	src/include/optimizer/paths.h
set_join_pathlist_hook	カスタム結合を行うフック	src/include/optimizer/paths.h

3.1.7 OOM Killer 対応

PostgreSQL 9.5 には Linux の OOM Killer に対する重み付けを指定する環境変数が追加されました。ただし Linux では OOM Killer 設定を行う/proc/self/oom_score_adj ファイルの書き込みには root ユーザー権限が必要になります。

表 17　OOM Killer に対応する環境変数

環境変数	説明
PG_OOM_ADJUST_FILE	重み付けを設定するファイル
PG_OOM_ADJUST_VALUE	重み付け値

3.1.8 書き込み途中の WAL ファイルのアーカイブ

スレーブ・インスタンスが途中まで書き込んだ WAL ファイルが、プロモーション実行後に拡張子 partial ファイルとして出力されます。

例 3 書き込み途中 WAL セグメントのアーカイブ

```
$ pg_ctl -D data.slave1 promote
server promoting
$
$ psql -c 'SELECT pg_switch_xlog()'
 pg_switch_xlog
----------------
 0/146D2FA8
(1 row)

$ ls -1 arch.slave1
000000010000000000000014.partial
000000020000000000000014
00000002.history
```

3.2 ユーティリティ

ユーティリティ・コマンドの主な機能強化点を説明します。

3.2.1 pg_rewind

pg_rewind コマンドは PostgreSQL 9.5 で追加されました。

□ 概要

pg_rewind コマンドはレプリケーション環境を構築するツールです。pg_basebackup コマンドと異なり、既存のデータベースクラスタに対して同期を行うことができます。プロモーションされたスレーブ・インスタンスと旧マスター・インスタンスの再同期を行う場面を想定しています。

□ パラメーター

pg_rewind コマンドには以下のパラメーターを指定できます。

表 18 パラメーター

パラメーター	説明
-D / --target-pgdata	更新を行うデータベースクラスタのディレクトリ
--source-pgdata	データ取得元のディレクトリ
--source-server	データ取得元の接続情報（リモート・インスタンス）
-P / --progress	実行状況の出力
-n / --dry-run	実行シミュレーションを行う
--debug	デバッグ情報の表示
-V / --version	バージョン情報表示
-? / --help	使用方法のメッセージ表示

□ 条件

pg_rewind コマンドを実行するためには、いくつかの条件があります。pg_rewind コマンドは、実行する条件をソースとターゲットの pg_control ファイルの内容からチェックしています。

まず PostgreSQL インスタンスのパラメーターwal_log_hints を on（デフォルト値 off）に指定するか、データ・チェックサムの機能を有効にする必要があります。またパラメーターfull_page_writes を on に設定する必要があります（デフォルト値 on）。

例 4 パラメーター設定上のエラー・メッセージ

```
$ pg_rewind --source-server='host=remhost1 port=5432 user=postgres'
        --target-pgdata=data -P
connected to remote server

target server need to use either data checksums or "wal_log_hints = on"
Failure, exiting
```

またターゲットとなるデータベースクラスタのインスタンスは正常に停止している必要があります。

例 5 ターゲット・インスタンス起動中のエラー・メッセージ

```
$ pg_rewind --source-server='host=remhost1 port=5432 user=postgres'
        --target-pgdata=data -P
target server must be shut down cleanly
Failure, exiting
```

データのコピー処理は pg_basebackup コマンドと同様に wal sender プロセスに対する接続を使用します。接続先（データ提供元）インスタンスの pg_hba.conf ファイル設定や、max_wal_senders パラメーターの設定が必要です。

□ 実行手順

pg_rewind は以下の手順で実行します。下記の例は、プロモーションを行ったスレーブ・インスタンスに接続し、旧マスター・インスタンスを新しいスレーブ・インスタンスに設定しています。

1. パラメーター設定

   現在のマスター・インスタンスのパラメーター、pg_hba.conf ファイルの設定を行います。必要に応じてファイルの情報をリロードします。

2. ターゲット・インスタンス停止

   同期を取る（旧マスター）インスタンスを停止します。

3. pg_rewind 実行

   旧マスター・インスタンス（データを更新する側）で pg_rewind コマンドを実行します。最初に-n パラメーターを指定して、テストを行った後、-n パラメーターを取って実

行します。

例 6 pg_rewind コマンドの実行

```
$ pg_rewind --source-server='host=remhost1 port=5432 user=postgres'
        --target-pgdata=data
connected to server
The servers diverged at WAL position 0/9000060 on timeline 1.
Rewinding from last common checkpoint at 0/8000060 on timeline 1
reading source file list
reading target file list
reading WAL in target
need to copy 53 MB (total source directory size is 76 MB)
54294/54294 kB (100%) copied
creating backup label and updating control file
Done!
```

4. recovery.conf ファイルの編集

pg_rewind コマンドでは更新先データベースクラスタに recovery.conf ファイルは作成されません。このため新スレーブ・インスタンス用に recovery.conf ファイルを作成します。

5. postgresql.conf ファイルの編集

pg_rewind コマンドの実行により、postgresql.conf ファイルはリモートホストからコピーされて上書きされています。必要に応じてパラメーターを編集します。

6. スレーブ・インスタンスの起動

新しいスレーブ・インスタンスを起動します。

□ 終了ステータス

pg_rewind コマンドは、処理が正常に終了すると 0 を、失敗すると 1 を返して終了します。

3.2.2 pg_ctl

インスタンス停止モード（-m オプション）のデフォルト値が smart から fast に変更されました。

3.2.3 vacuumdb

複数プロセッサ・コアを積極的に使用するために--jobs パラメーターが追加されました。--jobs パラメーターには並列処理させるジョブ数を指定します。ジョブ数は 1 以上、「マクロ FD_SETSIZE - 1」以下（Red Hat Enterprise Linux 7 では 1,023 以下）です。

例 7 --jobs パラメーターの上限と下限

```
$ vacuumdb --jobs=-1
vacuumdb: number of parallel "jobs" must be at least 1
$ vacuumdb --jobs=1025
vacuumdb: too many parallel jobs requested (maximum: 1023)
```

□ --jobs パラメーターとセッション数

--jobs パラメーターに数値を指定すると、パラメーターで指定された数と同数のセッションが作成されます。全データベースに対して VACUUM を行う場合（--all 指定）は、データベース単位で、単一のデータベースに対して VACUUM を行う場合は、テーブル単位で並列に処理を行います。このパラメーターのデフォルト値は 1 で、従来バージョンと同じ動作になります。

下記の例では--jobs=10 を指定したため、postgres プロセスが 10 個起動しています。

例 8 --jobs パラメーターの指定とセッション

```
$ vacuumdb --jobs=10 -d demodb &
vacuumdb: vacuuming database "demodb"
$ ps -ef|grep postgres
postgres 14539     1  0 10:59 pts/2  00:00:00 /usr/local/pgsql/bin/postgres -D data
postgres 14540 14539  0 10:59 ?      00:00:00 postgres: logger process
postgres 14542 14539  0 10:59 ?      00:00:00 postgres: checkpointer process
postgres 14543 14539  0 10:59 ?      00:00:00 postgres: writer process
postgres 14544 14539  0 10:59 ?      00:00:00 postgres: wal writer process
postgres 14545 14539  0 10:59 ?      00:00:00 postgres: stats collector process
postgres 14569 14539  6 11:00 ?      00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14570 14539  0 11:00 ?      00:00:00 postgres: postgres demodb [local] idle
postgres 14571 14539  5 11:00 ?      00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14572 14539  7 11:00 ?      00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14573 14539  0 11:00 ?      00:00:00 postgres: postgres demodb [local] idle
postgres 14574 14539  0 11:00 ?      00:00:00 postgres: postgres demodb [local] idle
postgres 14575 14539  9 11:00 ?      00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14576 14539  5 11:00 ?      00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14577 14539  0 11:00 ?      00:00:00 postgres: postgres demodb [local] idle
postgres 14578 14539  1 11:00 ?      00:00:00 postgres: postgres demodb [local] idle
```

--jobs パラメーターで指定した値が、--table パラメーター数以上だった場合、並列度の上限はテーブル数になります。またセッション数の計算には PostgreSQL パラメーター max_connections は考慮されないため、セッション数の超過が検知されると「FATAL: sorry, too many clients already」エラーが発生します。

3.2.4 pg_dump

以下の拡張が行われました。

□ --enable-row-security パラメーターの追加

デフォルト状態では pg_dump コマンドは row_security パラメーターを off に指定してダンプを取得します。このため Row Level Security をバイパスできないユーザー権限で接続するとデータの取得がエラーになります。--enable-row-security パラメーターを指定することで、権限があるデータのみダンプファイルに含まれることになります。Alpha 2 ではエラーが発生します。

□ --snapshot パラメーターの追加

pg_export_snapshot 関数で作成したスナップショット ID を指定します。指定されたスナップショットＩＤを使用したダンプを取得することができます。

□ --verbose パラメーターの出力

--verbose パラメーターを指定した場合に出力されるログにスキーマ名が含まれるようになります。

□ --ignore-version パラメーターの削除

--ignore-version パラメーターは削除されました。同じ修正が pg_dumpall コマンド、pg_restore コマンドにも適用されました。

3.2.5 psql

psql コマンドには以下の機能が追加されました。

□ pager_min_lines

pager_min_lines パラメーターが追加されました。このパラメーターを指定すると、指定した行数未満の出力にはページャが使用されません。デフォルト値は 0 で、端末の行数に合わせてページャが動作します。この設定は執筆者環境では動作しませんでした。

例 9 pager_min_lines の設定

```
postgres=> ¥pset pager_min_lines 20
Pager won't be used for less than 20 lines
```

□ ¥set ECHO errors

¥set ECHO errors を指定すると、エラー発生時に実行しようとした文が表示されるようになります。

例 10 ¥set ECHO errors

```
postgres=> SELECT * FROM notexists1 ;
ERROR:  relation "notexists1" does not exist
LINE 1: SELECT * FROM notexists1;
                      ^

postgres=> ¥set ECHO errors
postgres=> SELECT * FROM notexists1 ;
ERROR:  relation "notexists1" does not exist
LINE 1: SELECT * FROM notexists1;
                      ^

STATEMENT:  SELECT * FROM notexists1 ;
```

□ スキーマ名エラーの表示

存在しないスキーマ名を指定した場合に、エラー発生場所を示すメッセージが追加されました。

例 11 スキーマ名のエラー

```
postgres=> CREATE TABLE badschema.table1(c1 NUMERIC, c2 VARCHAR(10)) ;
ERROR:  schema "badschema" does not exist
LINE 1: CREATE TABLE badschema.table1(c1 NUMERIC, c2 VARCHAR(10)) ;
```

□ ¥watch コマンド出力

¥watch コマンドの出力に¥timing コマンドの情報が付加されるようになりました。

例 12 ¥watch コマンドと¥timing コマンド

```
postgres=> ¥timing
Timing is on.
demodb=> ¥watch 1
Watch every 1s  Fri Aug  7 11:56:58 2015

              now
-------------------------------
 2015-08-07 11:56:59.931996+09
(1 row)

Time: 0.458 ms
```

□ ¥コマンド

以下の拡張が行われました。旧バージョンでは不要だった¥db+コマンドの実行には superuser 権限が必要になりました。

表 19 追加／変更された¥コマンド

コマンド	変更	説明
¥db+	表示追加	表スペースのサイズ（Size）が追加表示されます
¥dT+	表示追加	データタイプに所有者（Owner）が追加表示されます

3.2.6 pgbench

pgbench コマンドにはいくつかの新機能が提供されました。

□ --latency-limit パラメーター

pgbench コマンドのパラメーターに --latency-limit (-L) パラメーターが使用できるようになりました。このパラメーターにミリ秒単位の値を指定すると、指定された秒数未満のトランザクション割合を出力します。

例 13 --latency-limit パラメータ指定時の出力

```
$ pgbench -c 10 -U pgbench -L 10 pgbench
pghost:  pgport:  nclients: 10 nxacts: 10 dbName: pgbench
starting vacuum...end.
<<途中省略>>
number of transactions actually processed: 100/100
number of transactions above the 10.0 ms latency limit: 64 (64.000 %)
latency average: 17.065 ms
latency stddev: 12.816 ms
tps = 479.927051 (including connections establishing)
tps = 546.603406 (excluding connections establishing)
```

□ カスタム・スクリプト使用時の動作変更

カスタム・スクリプト（-f オプション）を使用し、-n オプションを指定しない場合の動作が変更されました。従来のバージョンでは pgbench が作成する各テーブルの VACUUM 処理を行い、テーブルが存在しなければコマンドを終了していました。PostgreSQL 9.5 では、VACUUM 処理が失敗してもカスタム・スクリプトの処理を継続するように変更されました。

例 14 カスタム・スクリプト使用時の動作

```
$ pgbench -f bench.sql -U user1 postgres
starting vacuum...ERROR:  relation "pgbench_branches" does not exist
(ignoring this error and continuing anyway)
ERROR:  relation "pgbench_tellers" does not exist
(ignoring this error and continuing anyway)
ERROR:  relation "pgbench_history" does not exist
(ignoring this error and continuing anyway)
end.
transaction type: Custom query
<<以下省略>>
```

pgbench コマンドには上記以外にもいくつかの新機能が実装されました。

3.2.7 pg_receivexlog

レプリケーション・スロットの制御を行うパラメーターが追加されました。制御を行うスロット名は、従来通り--slot パラメーターで指定します。

□ --create-slot

--slot パラメーターで指定された名前のスロットを作成し、そのまま WAL 情報を受信し続けます。PostgreSQL インスタンスのパラメーターmax_replication_slots に余裕が無い場合や既に同名のスロットが存在する場合にはエラーになります。

□ --drop-slot

指定されたスロットを削除して、コマンドを終了します。

□ --synchronous

受信した WAL 情報を同期的に書き込みます。

3.2.8 reindexdb

詳細情報を出力する「-v」「--verbose」オプションが追加されました。内部的には「REINDEX (VERBOSE) DATABASE データベース名」文が実行されています。

例 15 reindexdb –v オプション

```
$ reindexdb -v demodb
INFO:  index "pg_class_oid_index" was reindexed
DETAIL:  CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  index "pg_class_relname_nsp_index" was reindexed
DETAIL:  CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  index "pg_class_tblspc_relfilenode_index" was reindexed
DETAIL:  CPU 0.00s/0.00u sec elapsed 0.00 sec.
<<以下省略>>
```

3.3 パラメーターの変更

PostgreSQL 9.5 では以下のパラメーターが変更されました。

3.3.1 追加されたパラメーター

以下のパラメーターが追加されました。

表 20　追加されたパラメーター

パラメーター	説明	デフォルト値
cluster_name	データベースクラスタ名の指定	''
gin_pending_list_limit	GIN インデックスの待機リストの最大サイズ	4MB
row_security	Row Level Security 機能の有効化 使用できる値は on, off, force	on
track_commit_timestamp	トランザクションの最終コミット時間を WAL に出力する	off
wal_compression	WAL の full page write 圧縮	off
log_replication_commands	レプリケーション関連ログの出力	off
max_wal_size	チェックポイントを開始する WAL サイズ。値の範囲は 2～2147483647	1GB
min_wal_size	WAL ファイルのリサイクルを開始する WAL サイズ。値の範囲は 2～2147483647	80MB
operator_precedence_warning	優先順位の変更により結果が変わる可能性がある SQL に警告を出力	off
wal_retrieve_retry_interval	WAL データの再取得間隔をミリ秒単位で指定。	5s

□　wal_retrieve_retry_interval パラメーター

wal_retrieve_retry_interval は旧バージョンでは固定値だった設定をパラメーター化しています。デフォルトの 5 秒は旧バージョンと同じ動作です。

3.3.2 変更されたパラメーター

以下のパラメーターは設定範囲や選択肢が変更されました。

表 21　変更されたパラメーター

パラメーター	変更内容
log_autovacuum_min_duration	テーブル単位に指定可能
archive_mode	always が追加指定可能
trace_sort	追加情報の出力
debug_assertions	読み取り専用に変更
local_preload_libraries	ALTER ROLE SET 文で設定可能

□　log_autovacuum_min_duration

パラメーターlog_autovacuum_min_duration はテーブル単位に指定することができるようになりました。

例 16　テーブル単位のパラメーター設定

```
postgres=> ALTER TABLE vacuum1 SET (log_autovacuum_min_duration = 1000) ;
ALTER TABLE
postgres=> ¥d+ vacuum1
                                Table "public.vacuum1"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 c1     | numeric               |           | main     |              |
 c2     | character varying(10) |           | extended |              |
Options: log_autovacuum_min_duration=1000
```

□　archive_mode

パラメーターarchive_mode の選択肢に always が追加されました。レプリケーション環境のスレーブ・インスタンス以外では on と always には違いがありません。スレーブ・インスタンスでこのパラメーターを always に指定すると、archiver プロセスが起動し、アーカイブログの出力処理が行われます。

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。

表 22　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 9.4	PostgreSQL 9.5	備考
server_version	9.4.4	9.5alpha2	
server_version_num	90404	90500	
search_path	"$user",public	"$user", public	スペース含む

3.3.4 廃止されたパラメーター

以下のパラメーターが廃止されました。

□　checkpoint_segments の廃止

チェックポイントの発生時点を決定するパラメーターcheckpoint_segments が廃止され、パラメーターmax_wal_size と min_wal_size に分割されました。

表 23　checkpoint_segments の代替パラメーター

パラメーター	説明	デフォルト値
max_wal_size	チェックポイントを開始する	1GB
min_wal_size	WAL ファイルのリサイクルを開始する	80MB

従来のパラメーターcheckpoint_segments では、チェックポイントの開始と WAL ファイルのリサイクルを同一パラメーターで行っていたため、チェックポイント間隔を伸ばすと WAL ファイルが大量に作成されていました。PostgreSQL 9.5 では２つの機能を別々のパラメーターによる制御するように変更されました。

□　ssl_renegotiation_limit の廃止

パラメーターssl_renegotiation_limit は廃止されました。

3.3.5 recovery.conf ファイルの変更パラメーター

recovery.conf ファイルは以下のパラメーターが変更されました。

□　recovery_target_action パラメーターの追加

パラメーターrecovery_target_action が追加されました。このパラメーターは旧バージョ

ンの pause_at_recovery_target パラメーターをより汎用的に拡張しています。リカバリー・ターゲットに到達した場合の動作を指定します。

このパラメーターには以下の値を指定することができます。

- pause（デフォルト値）
  ターゲットに到達した状態で待機します。
- promote
  リカバリーを完了し、ユーザーの接続を受け付けます。
- shutdown
  リカバリーを完了し、インスタンスを停止します。recovery.conf ファイルの名前変更は行われません。

□ 削除されたパラメーター

パラメーターpause_at_recovery_target は削除されました。代替パラメーターとして、前述の recovery_target_action を使用します。recovery.conf ファイル内で、pause_at_recovery_target パラメーターを指定すると、ログファイルに以下のメッセージが記録されインスタンスが起動できません。

例 17　pause_at_recovery_target パラメーターの指定エラー

```
$ cat data/recovery.conf
restore_command = 'cp /usr/local/pgsql/arch/%f %p'
pause_at_recovery_target = on
$ pg_ctl -D data -w start
waiting for server to start....LOG:  redirecting log output to logging
collector process
HINT:  Future log output will appear in directory "pg_log".
.... stopped waiting
pg_ctl: could not start server
Examine the log output
$ cat data/pg_log/postgresql-2015-08-07_111751.log
LOG:  database system was shut down at 2015-08-07 11:17:20 JST
FATAL:  unrecognized recovery parameter "pause_at_recovery_target"
LOG:  startup process (PID 12233) exited with exit code 1
LOG:  aborting startup due to startup process failure
```

3.4 SQL 文の機能追加

ここでは SQL 文に関する新機能を説明しています。

3.4.1 INSERT ON CONFLICT 文

制約違反となる INSERT 文実行時に自動的に UPDATE 文に切り替えること（いわゆる UPSERT 文）ができるようになりました。INSERT 文に ON CONFLICT 句を指定します。

構文

```
INSERT INTO …
ON CONFLICT [ { (column_name, …) | ON CONSTRAINT constraint_name }]
{ DO NOTHING | DO UPDATE SET column_name = value }
[ WHERE … ]
```

ON CONFLICT 部分には制約違反が発生する場所を指定します。

- 列名のリストまたは、「ON CONSTRAINT 制約名」の構文で制約名を指定します。
- 複数列で構成される制約を指定する場合は、制約に含まれる全ての列名を指定する必要があります。
- ON CONFLICT 以降を省略すると全ての制約違反がチェックされます。省略できるのは DO NOTHING を使用する場合のみです。
- ON CONFLICT 句で指定された列または制約以外の制約違反が発生すると、INSERT 文はエラーになります。

ON CONFLICT 句以降には制約違反が発生した場合の動作を記述します。DO NOTHING 句を指定すると、制約違反が発生しても何もしません（制約違反も発生しません）。DO UPDATE 句を指定すると、特定の列を UPDATE します。以下に実行例を記載します。

例 18　テーブルの準備

```
postgres=> CREATE TABLE upsert1 (key NUMERIC, val VARCHAR(10)) ;
CREATE TABLE
postgres=> ALTER TABLE upsert1 ADD CONSTRAINT pk_upsert1 PRIMARY KEY (key) ;
ALTER TABLE
postgres=> INSERT INTO upsert1 VALUES (100, 'Val 1') ;
INSERT 0 1
postgres=> INSERT INTO upsert1 VALUES (200, 'Val 2') ;
INSERT 0 1
postgres=> INSERT INTO upsert1 VALUES (300, 'Val 3') ;
INSERT 0 1
```

以下は ON CONFLICT 句の記述例です。処理部分には DO NOTHING を指定しているので、制約違反が発生しても何もしません。

例 19　ON CONFLICT 句

```
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
        ON CONFLICT DO NOTHING ;          ←　制約名や列を省略
INSERT 0 0
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
        ON CONFLICT(key) DO NOTHING ;  ← 制約違反が発生する列を記述
INSERT 0 0
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
        ON CONFLICT(val) DO NOTHING ;   ← 制約が無い列を指定するとエラー発生
ERROR:  there is no unique or exclusion constraint matching the ON CONFLICT
specification
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
        ON CONFLICT ON CONSTRAINT pk_upsert1 DO NOTHING ;  ← 制約名を指定
INSERT 0 0
```

DO UPDATE 句には、更新処理を記述します。基本的には UPDATE 文の SET 句以降と同じです。EXCLUDED というエイリアスを使用すると、INSERT 文を実行しようとして格納できなかったレコードにアクセスできます。

例 20　DO UPDATE 句

```
postgres=> INSERT INTO upsert1 VALUES (400, 'Upd4')
        ON CONFLICT DO UPDATE SET val = EXCLUDED.val ; ← 制約を省略してエラー
ERROR:  ON CONFLICT DO UPDATE requires inference specification or constraint
name
LINE 2: ON CONFLICT DO UPDATE SET val = EXCLUDED.val;
        ^
HINT:  For example, ON CONFLICT ON CONFLICT (<column>).
postgres=> INSERT INTO upsert1 VALUES (300, 'Upd3')
   ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val ; ← EXCLUDED エイリアスを
使用
INSERT 0 1
postgres=> INSERT INTO upsert1 VALUES (300, 'Upd3')
 ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val WHERE upsert1.key = 100 ;
INSERT 0 0       ↑ WHERE 句を指定して UPDATE 条件を決定できる
```

□　ON CONFLICT 句とトリガー

INSERT ON CONFLICT 文の実行時にトリガーがどのように動作するかを検証しました。BEFORE INSERT トリガーは常に動作しました。DO UPDATE 文によりレコードが更新される場合は、BEFORE INSERT トリガー、BEFORE / AFTER UPDATE トリガーが動作しました。WHERE 句により UPDATE が行われなかった場合は BEFORE INSERT トリガーのみが実行されました。

表 24　トリガーの起動

トリガー	INSERT 成功	DO NOTHING	DO UPDATE （更新あり）	DO UPDATE （更新なし）
BEFORE INSERT	実行	実行	実行	実行
AFTER INSERT	実行	-	-	-
BEFORE UPDATE	-	-	実行	-
AFTER UPDATE	-	-	実行	-

□　ON CONFLICT 句と実行計画

ON CONFLICT 句の部分が実行されることで、実行計画が変化します。EXPLAIN 文を実行すると、実行計画内に Conflict Resolution, Conflict Arbiter Indexes, Conflict Filter 等が表示されます。具体的な出力は以下の例の通りです。

例 21　ON CONFLICT 句と実行計画

```
postgres=> EXPLAIN INSERT INTO upsert1 VALUES (200, 'Update 1')
        ON CONFLICT(key) DO NOTHING ;
                      QUERY PLAN
------------------------------------------------------
 Insert on upsert1  (cost=0.00..0.01 rows=1 width=0)
   Conflict Resolution: NOTHING
   Conflict Arbiter Indexes: pk_upsert1
   ->  Result  (cost=0.00..0.01 rows=1 width=0)
(4 rows)
postgres=> EXPLAIN INSERT INTO upsert1 VALUES (400, 'Upd4')
        ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val ;
                      QUERY PLAN
------------------------------------------------------
 Insert on upsert1  (cost=0.00..0.01 rows=1 width=0)
   Conflict Resolution: UPDATE
   Conflict Arbiter Indexes: pk_upsert1
   ->  Result  (cost=0.00..0.01 rows=1 width=0)
(4 rows)
postgres=> EXPLAIN INSERT INTO upsert1 VALUES (400, 'Upd4')
  ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val WHERE upsert1.key = 100 ;
                      QUERY PLAN
------------------------------------------------------
 Insert on upsert1  (cost=0.00..0.01 rows=1 width=0)
   Conflict Resolution: UPDATE
   Conflict Arbiter Indexes: pk_upsert1
   Conflict Filter: (upsert1.key = '100'::numeric)
   ->  Result  (cost=0.00..0.01 rows=1 width=0)
(5 rows)
```

現在のバージョンでは、postgres_fdw モジュールを使ったリモート・インスタンスに対しては ON CONFLICT DO UPDATE 文はサポートされていません。

□　ON CONFLICT 句とパーティション・テーブル

INSERT トリガーを使ったパーティション・テーブルに対する ON CONFLICT 句は無視されます。

例 22　パーティション・テーブルに対する INSERT ON CONLICT(1)

```
postgres=> CREATE TABLE main1 (key1 NUMERIC, val1 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE TABLE main1_part100 (CHECK(key1 < 100)) INHERITS (main1) ;
CREATE TABLE
postgres=> CREATE TABLE main1_part200 (CHECK(key1 >= 100 AND key1 < 200))
        INHERITS (main1) ;
CREATE TABLE
postgres=> ALTER TABLE main1_part100 ADD CONSTRAINT pk_main1_part100
        PRIMARY KEY (key1);
ALTER TABLE
postgres=> ALTER TABLE main1_part200 ADD CONSTRAINT pk_main1_part200
        PRIMARY KEY (key1);
ALTER TABLE
postgres=> CREATE OR REPLACE FUNCTION func_main1_insert()
        RETURNS TRIGGER AS $$
         BEGIN
           IF    (NEW.key1 < 100) THEN
                 INSERT INTO main1_part100 VALUES (NEW.*) ;
           ELSIF (NEW.key1 >= 100 AND NEW.key1 < 200) THEN
                 INSERT INTO main1_part200 VALUES (NEW.*) ;
           ELSE
                  RAISE EXCEPTION 'ERROR! key1 out of range.' ;
           END IF ;
           RETURN NULL ;
         END ;
        $$  LANGUAGE 'plpgsql';
CREATE FUNCTION
```

例 23　パーティション・テーブルに対する INSERT ON CONLICT(2)

```
postgres=> CREATE TRIGGER trg_main1_insert BEFORE INSERT ON main1
            FOR EACH ROW EXECUTE PROCEDURE func_main1_insert() ;
CREATE TRIGGER
postgres=> INSERT INTO main1 VALUES (100, 'DATA100') ;
INSERT 0 0
postgres=> INSERT INTO main1 VALUES (100, 'DATA100') ;
ERROR:  duplicate key value violates unique constraint "pk_main1_part200"
DETAIL:  Key (key1)=(100) already exists.
CONTEXT:  SQL statement "INSERT INTO main1_part200 VALUES (NEW.*)"
PL/pgSQL function func_main1_insert() line 6 at SQL statement
postgres=> INSERT INTO main1 VALUES (100, 'DATA100')
        ON CONFLICT DO NOTHING ;
ERROR:  duplicate key value violates unique constraint "pk_main1_part200"
DETAIL:  Key (key1)=(100) already exists.
CONTEXT:  SQL statement "INSERT INTO main1_part200 VALUES (NEW.*)"
PL/pgSQL function func_main1_insert() line 6 at SQL statement
```

3.4.2 ALTER TABLE SET UNLOGGED 文

テーブルに対して更新トランザクションが発生すると変更履歴が WAL に書き込まれます。標準設定ではユーザーが発行する COMMIT 文は WAL の書き込みが完了するまで待機します。旧バージョンの PostgreSQL では、信頼性が重要ではない場合、CREATE UNLOGGED TABLE 文を使って WAL の書き込みを行わないテーブルを作成することができました。PostgreSQL 9.5 では WAL の書き込みの制御をテーブル単位に変更することができるようになりました。

構文　テーブルを UNLOGGED TABLE に変更

```
ALTER TABLE table_name SET UNLOGGED
```

構文　UNLOGGED TA BLE を通常のテーブルに変更

```
ALTER TABLE table_name SET LOGGED
```

例 24　UNLOGGED TABLE への切り替え

```
postgres=> CREATE TABLE logtbl1 (c1 NUMERIC, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> ¥d+ logtbl1
                                Table "public.logtbl1"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 c1     | numeric               |           | main     |              |
 c2     | character varying(10) |           | extended |              |

postgres=> ALTER TABLE logtbl1 SET UNLOGGED ;
ALTER TABLE
postgres=> ¥d+ logtbl1
                            Unlogged table "public.logtbl1"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 c1     | numeric               |           | main     |              |
 c2     | character varying(10) |           | extended |              |
```

¥d+コマンドにより、通常のテーブルが UNLOGGED テーブルに変更されたことがわかります。

□ 実装

内部的には、同一構造を持つ新規の UNLOGGED TABLE（または TABLE）を作成し、データのコピーを行っています。pg_class カタログの relfilenode 列と relpersistence 列が変更されます。

例 25　pg_class カタログの変化

```
postgres=> SELECT relname, relfilenode, relpersistence FROM pg_class WHERE
        relname='logtbl1' ;
 relname | relfilenode | relpersistence
---------+-------------+----------------
 logtbl1 |       16483 | p
(1 row)

postgres=> ALTER TABLE logtbl1 SET UNLOGGED ;
ALTER TABLE
postgres=> SELECT relname, relfilenode, relpersistence FROM pg_class WHERE
        relname='logtbl1' ;
 relname | relfilenode | relpersistence
---------+-------------+----------------
 logtbl1 |       16489 | u
(1 row)
```

□ INHERIT テーブルに対する変更

継承テーブルの設定を変更した場合、各テーブルはそれぞれ独立した設定が維持されます。子テーブルのひとつを UNLOGGED に設定しても、他のテーブルには影響を与えません。

3.4.3 IMPORT FOREIGN SCHEMA 文

FOREIGN DATA WRAPPER を使って外部テーブルにアクセスするためには、SERVER と USER MAPPING を作成した後、CREATE FOREIGN TABLE 文を使って外部テーブルを作成します。CREATE FOREIGN TABLE 文はリモート参照するテーブル単

位に実行し、リモート・テーブルと同じ列の定義をもう一度記述する必要がありました。この処理をスキーマ単位で一括して行う文が IMPORT FOREIGN SCHEMA 文です。IMPORT FOREIGN SCHEMA 文はテーブルだけでなく、リモート・インスタンスのスキーマに格納されたビュー、マテリアライズド・ビューもインポートすることができます。

構文

```
IMPORT FOREIGN SCHEMA
[ { LIMIT TO | EXCEPT } (table_name, ...) ]
FROM SERVER server_name
INTO local_schema
[ OPTIONS (option 'value', ...) ]
```

表 25　IMPORT FOREIGN SCHEMA 構文

構文	説明
LIMIT TO EXCEPT	インポートするテーブルを限定するために使用します。省略時は全テーブルをインポートします。
FROM SERVER	インポートを行うリモート・インスタンスを示す SERVER オブジェクトを指定します。
INTO	インポートを行うローカル・インスタンスのスキーマ名を指定します。

例 26　IMPORT FOREIGN SCHEMA 文の実行

```
postgres=# CREATE EXTENSION postgres_fdw ;
CREATE EXTENSION
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw
        OPTIONS (host '192.168.1.200', port '5432', dbname 'demodb') ;
CREATE SERVER
postgres=# CREATE USER MAPPING FOR public SERVER remsvr1
        OPTIONS (user 'user1', password 'secret') ;
CREATE USER MAPPING
postgres=# CREATE SCHEMA schema2 ;
CREATE SCHEMA
postgres=# IMPORT FOREIGN SCHEMA schema1 FROM SERVER remsvr1 INTO schema2 ;
IMPORT FOREIGN SCHEMA
```

- インポートする FOREIGN TABLE を格納するスキーマはあらかじめ作成しておく必要があります。
- インポートのために指定したローカル・スキーマ内に、リモート・スキーマと同じ名前のテーブルが存在した場合、エラーが発生して FOREIGN TABLE はまったく作成されません。
- リモート・インスタンスがパスワードを要求しない（trust）場合、一般ユーザーによる IMPORT FOREIGN SCHEMA 文はエラーになります。

例 27　エラーが発生する動作

```
postgres=# IMPORT FOREIGN SCHEMA public FROM SERVER remsvr1 INTO schemax ;
ERROR:  schema "schemax" does not exist ↑ローカル・スキーマが存在しない

postgres=# IMPORT FOREIGN SCHEMA schema1 FROM SERVER remsvr1 INTO schema1 ;
ERROR:  relation "data1" already exists ↑ 同名のテーブルが存在する
CONTEXT:  importing foreign table "table2"

postgres=> IMPORT FOREIGN SCHEMA public FROM SERVER remsvr1 INTO schema1 ;
ERROR:  password is required
DETAIL:  Non-superuser cannot connect if the server does not request a password.
HINT:  Target server's authentication method must be changed.
```

□　オプション

IMPORT FOREIGN SCHEMA 文には以下のオプションを指定できます。

表 26　IMPORT FOREIGN SCHEMA 文オプション

オプション	デフォルト	説明
import_collate	true	COLLATE 句のインポートを行う
import_default	false	DEFAULT 句のインポートを行う
import_not_null	true	NOT NULL 制約のインポートを行う

import_default オプションを true に設定し、リモート・テーブルにシーケンス・オブジェクトを使った DEFAULT 句が使用されていた場合 IMPORT FOREIGN SCHEMA 文はエラーになります。

例 28　DEFAULT とシーケンス

```
(remote) postgres=> CREATE SEQUENCE seq1 ;
CREATE SEQUENCE
(remote)postgres=> CREATE TABLE data1(c1 NUMERIC DEFAULT nextval('seq1')) ;
CREATE TABLE

(local) postgres=# IMPORT FOREIGN SCHEMA public FROM SERVER remsvr2 INTO schema1
        OPTIONS (import_default 'true') ;
ERROR:  relation "public.seq1" does not exist
CONTEXT:  importing foreign table "data1"
```

3.4.4 SELECT SKIP LOCKED 文

競合するロックを保持しているタプルに対してアクセスを行う場合、通常はロックが解除されるまで待機します。待機をせずにエラーを発生する方法として NOWAIT 句を使用することができます。PostgreSQL 9.5 では、ロックされているタプルをスキップして検索を行うことができるようになりました。SELECT 文に SKIP LOCKED 句を指定します。

例 29　準備とロック

```
postgres=> CREATE TABLE lock1 (key NUMERIC, val TEXT) ;
CREATE TABLE
postgres=> INSERT INTO lock1 VALUES (generate_series(1, 2000), 'initial') ;
INSERT 0 2000
postgres=> BEGIN ;
BEGIN
postgres=> SELECT * FROM lock1 WHERE key = 1000 FOR SHARE ;
 key  |   val
------+---------
 1000 | initial
(1 row)
```

上記例では、テーブルを作成し「key=1000」のタプルを FOR SHARE 句でロックしています。下記例では別セッションで SKIP LOCKED 句を使った検索を行っています。「key=1000」のタプルは FOR UPDATE 句を指定した SELECT 文と競合するため、標準では待機状態になるか、NOWAIT 句を指定した場合エラーになります。SKIP LOCKED 句を指定したことで、「key=1000」のタプル以外が正常に検索されています。

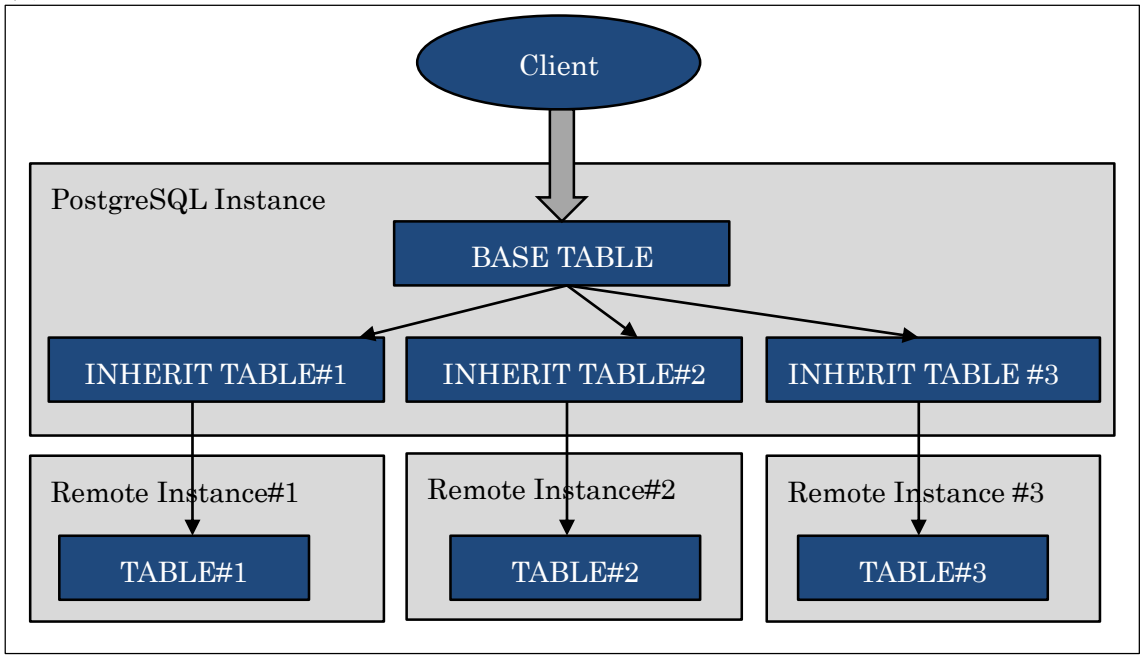
例 30　別セッションによる SELECT

```
postgres=> SELECT * FROM lock1 WHERE key IN (999, 1000, 1001) FOR UPDATE SKIP
LOCKED ;
 key  |   val
------+---------
  999 | initial
 1001 | initial
```

3.4.5 CREATE FOREIGN TABLE INHERITS 文

既存テーブルの継承テーブルとして、外部テーブル（FOREIGN TABLE）を作成できるようになりました。この機能により、パーティショニングと外部テーブルを組み合わせて処理を複数ホストに分散することが可能になります。

図 1 FOREIGN TABLE INHERIT



構文

```
CREATE FOREIGN TABLE table_name (check_constraints ...)
INHERITS (parent_table)
SERVER server_name
OPTIONS (option = 'value' ...)
```

従来のバージョンと異なる部分は INHERITS 句です。ここで親テーブルを指定します。以下に実装例と、実行計画を検証します。

例 31　親テーブルの作成

```
postgres=> CREATE TABLE parent1(key NUMERIC, val TEXT) ;
CREATE TABLE
```

リモート・インスタンス上でテーブル（inherit1、inherit2、inherit3）を作成します。

例 32　リモート・インスタンスで子テーブルの作成

```
postgres=> CREATE TABLE inherit1(key NUMERIC, val TEXT) ;
CREATE TABLE
```

CREATE FOREIGN TABLE 文を実行して、各リモート・インスタンス上のテーブル（inherit1, inherit2, inherit3）に対する外部テーブルを作成します。

例 33　外部テーブルの作成

```
postgres=# CREATE EXTENSION postgres_fdw ;
CREATE EXTENSION
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw
        OPTIONS (host 'remsvr1', dbname 'demodb', port '5432') ;
CREATE SERVER
postgres=# CREATE USER MAPPING FOR public SERVER remsvr1
        OPTIONS (user 'demo', password 'secret') ;
CREATE USER MAPPING
postgres=# GRANT ALL ON FOREIGN SERVER remsvr1 TO public ;
GRANT
postgres=> CREATE FOREIGN TABLE inherit1(CHECK(key < 1000))
        INHERITS (parent1) SERVER remsvr1 ;
CREATE FOREIGN TABLE
```

従来バージョンと異なる部分は CREATE FOREIGN TABLE 文に列定義ではなく、CHECK 制約を指定している部分と INHERITS 句で継承元テーブルを指定している点です。上記は 1 サーバのみの例ですが、複数インスタンスに対して CREATE SERVER 文、CREATE USER MAPPING 文、CREATE FOREIGN TABLE 文を作成します。

例 34　INHERITS 句を指定した FOREIGN TABLE 定義

```
postgres=> ¥d+ inherit2
                          Foreign table "public.inherit2"
 Column |  Type   | Modifiers | FDW Options | Storage  | Stats target | Description
--------+---------+-----------+-------------+----------+--------------+-------------
 key    | numeric |           |             | main     |              |
 val    | text    |           |             | extended |              |
Check constraints:
    "inherit2_key_check" CHECK (key >= 1000::numeric AND key < 2000::numeric)
Server: remsvr2
Inherits: parent1
```

実行計画を確認すると、CHECK 制約により特定のインスタンスにのみアクセスしていることがわかります。

例 35　実行計画の確認

```
postgres=> EXPLAIN SELECT * FROM parent1 WHERE key = 1500 ;
                               QUERY PLAN
------------------------------------------------------------------------
 Append  (cost=0.00..121.72 rows=6 width=64)
   ->  Seq Scan on parent1  (cost=0.00..0.00 rows=1 width=64)
         Filter: (key = '1500'::numeric)
   ->  Foreign Scan on inherit2  (cost=100.00..121.72 rows=5 width=64)
(4 rows)
```

3.4.6 ALTER USER 文

ALTER USER 文に指定するユーザー名に CURRENT_USER および CURRENT_ROLE を指定することができるようになりました。接続中ユーザーのパスワード変更等に ALTER USER 文を実行できます。

例 36　CURRENT_USER 句の使用

```
postgres=> ALTER USER CURRENT_USER PASSWORD 'secret' ;
ALTER ROLE
```

3.4.7 UPDATE SET 文

SELECT 結果を用いる UPDATE 文で FROM 句の記述が不要になりました。

例 37　テーブルの準備

```
postgres=> CREATE TABLE upd1(c1 NUMERIC, c2 NUMERIC, c3 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE TABLE upd2(c1 NUMERIC, c2 NUMERIC, c3 VARCHAR(10)) ;
CREATE TABLE
```

テーブル upd2 の c2, c3 列の値をテーブル upd1 の c2, c3 列で更新します。その際に 2 つのテーブルの c1 列の値が同一であるレコードを使用します。

例 38　PostgreSQL 9.4 までの構文

```
postgres=> UPDATE upd2 SET c2 = upd1.c2, c3 = upd1.c3
        FROM (SELECT * FROM upd1) AS upd1
        WHERE upd1.c1 = upd2.c1 ;
UPDATE 2
```

例 39　PostgreSQL 9.5 の構文

```
postgres=> UPDATE upd2 SET (c2, c3) =
        (SELECT c2, c3 FROM upd1 WHERE upd1.c1 = upd2.c1) ;
UPDATE 2
```

3.4.8 SELECT TABLESAMPLE 文

テーブルから一定割合のレコードをサンプリングする TABLESAMPLE 句が利用できるようになりました。

構文

```
SELECT ... FROM table_name ...
TABLESAMPLE {SYSTEM | BERNOULLI} (percent)
[ REPEATABLE (seed) ]
```

サンプリング方法として SYSTEM と BERNOULLI を指定できます。SYSTEM はサンプリングしたブロック全体のタプルを使用します。BERNOULLI はサンプリングしたブロックから更に一定割合のタプルを選択します。

percent にはサンプリング割合（1～100）を指定します。1～100 以外の値を指定すると

SELECT 文はエラーになります。REPEATABLE 句はオプションです。サンプリングのシードを指定します。

□ 実行計画

サンプリングを行った場合の実行計画は以下の通りとなります。BERNOULLI を指定するとコストが大きくなることがわかります。

例 40 サンプリング時の実行計画（TABLESAMPLE SYSTEM）

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 TABLESAMPLE SYSTEM (10) ;
                                QUERY PLAN
-------------------------------------------------------------------------------
Aggregate  (cost=341.00..341.01 rows=1 width=0) (actual time=4.914..4.915 rows=1
loops=1)
   ->  Sample Scan (system) on data1  (cost=0.00..316.00 rows=10000 width=0)
   (actualtime=0.019..3.205 rows=10090 loops=1)
 Planning time: 0.106 ms
 Execution time: 4.977 ms
(4 rows)
```

例 41 サンプリング時の実行計画（TABLESAMPLE BERNOULLI）

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 TABLESAMPLE
                BERNOULLI (10) ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=666.00..666.01 rows=1 width=0) (actual time=13.654..13.655
rows=1 loops=1)
   -> Sample Scan (bernoulli) on data1  (cost=0.00..641.00 rows=10000 width=0)
   (actual time=0.013..12.121 rows=10003 loops=1)
 Planning time: 0.195 ms
 Execution time: 13.730 ms
```

3.4.9 REINDEX SCHEMA 文

REINDEX 文に SCHEMA 句を指定できるようになりました。スキーマ単位にインデックスの再作成を行うことができます。

スキーマ内に他のユーザーが所有するインデックスがあった場合には、REINDEX 文実

行ユーザーがアクセス可能なインデックスのみ再構成を行います。

例 42　REINDEX SCHEMA 文の実行

```
postgres=> REINDEX SCHEMA schema1 ;
REINDEX
postgres=> REINDEX SCHEMA schema2 ;
ERROR:  must be owner of schema schema2
postgres=> BEGIN ;
BEGIN
postgres=> REINDEX SCHEMA schema1 ;
ERROR:  REINDEX SCHEMA cannot run inside a transaction block
postgres=>
```

REINDEX SCHEMA 文の実行には以下の条件が必要です。

- スキーマのオーナーであること。
- トランザクション内では実行できません。

3.4.10 REINDEX (VERBOSE)文

REINDEX 文に、詳細メッセージを出力する VERBOSE 句が指定できるようになりました。reindexdb コマンドにも対応する-v オプションが追加されています。

構文

```
REINDEX (VERBOSE) {TABLE | DATABASE | SCHEMA} ...
```

例 43　REINDEX (VERBOSE)

```
postgres=> REINDEX (VERBOSE) TABLE data1 ;
INFO:  index "idx1_data1" was reindexed
DETAIL:  CPU 0.00s/0.00u sec elapsed 0.02 sec.
INFO:  index "pg_toast_16424_index" was reindexed
DETAIL:  CPU 0.00s/0.00u sec elapsed 0.00 sec.
REINDEX
```

3.4.11 ROLLUP, CUBE, GROUPING SETS

SELECT 文内で、小計値を計算する ROLLEUP, CUBE, GROUPING SETS 句が使用できるようになりました。これらの指定は GROUP BY 句と同時に使用します。詳細な構文はマニュアルを参照してください。

例 44　ROLLUP（c1 列ごとの小計値を出力する）

```
postgres=> SELECT c1, SUM(c2) FROM role1 GROUP BY ROLLUP (c1) ;
  c1   |  sum
-------+-------
 val1  |  5050
 val3  |  5050
 val4  |  5050
       | 15150
(4 rows)
```

3.4.12 CREATE FOREIGN TABLE (CHECK)

CREATE FOREIGN TABLE 文で CHECK 制約を記述することができるようになりました。ただし CREATE FOREIGN TABLE 文で指定した制約は INSERT 文や UPDATE 文に対しては無効です。またパラメーターconstraint_exclusion を on に設定したセッションの SELECT 文に限り有効になります。

例 45　CREATE FOREIGN TABLE の CHECK 制約

```
postgres=> CREATE FOREIGN TABLE data1 (c1 NUMERIC, c2 VARCHAR(10),
        CHECK(c1 > 0)) SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=> INSERT INTO data1 VALUES (-2, 'add') ;
INSERT 0 1
postgres=> SELECT * FROM data1 WHERE c1 = -2 ;
c1  | c2
----+-----
 -2 | add
(1 row)
postgres=> SET constraint_exclusion = on ;
SET
postgres=> SELECT * FROM data1 WHERE c1 = -20 ;
 c1 | c2
----+----
(0 rows)
```

パラメーターconstraint_execution を on に設定すると、実行計画が変化します。CHECK 制約に合致しない条件が記述された場合にはリモート・インスタンスに SQL 文が発行されません。

例 46　CHECK 制約と実行計画

```
postgres=> SHOW constraint_exclusion ;
 constraint_exclusion
----------------------
 partition
(1 row)
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 WHERE c1 < 0 ;
                                    QUERY PLAN
-------------------------------------------------------------------------------------------
 Aggregate  (cost=178.27..178.28 rows=1 width=0) (actual time=1.411..1.411 rows=1 loops=1)
   ->  Foreign Scan on data1  (cost=100.00..175.42 rows=1138 width=0) (actual
time=1.400..1.401 rows=6 loops=1)
 Planning time: 0.267 ms
 Execution time: 2.585 ms
(4 rows)
postgres=> SET constraint_exclusion = on ;
SET
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 WHERE c1 < 0 ;
                                    QUERY PLAN
-------------------------------------------------------------------------------------------
 Aggregate  (cost=0.01..0.02 rows=1 width=0) (actual time=0.005..0.006 rows=1 loops=1)
   ->  Result  (cost=0.00..0.01 rows=1 width=0) (actual time=0.001..0.001 rows=0 loops=1)
         One-Time Filter: false
 Planning time: 0.351 ms
 Execution time: 0.038 ms
(5 rows)
```

3.4.13 CREATE EVENT TRIGGER 文の拡張

CREATE EVENT TRIGGER 文の ON 句に table_rewrite が指定できるようになりました。table_rewrite トリガーは ALTER TABLE 文および ALTER TYPE 文の発生に対してトリガーが発行されます。

例 47　CREATE EVENT TRIGGER 文の拡張

```
postgres=# CREATE FUNCTION rewrite1() RETURNS event_trigger AS $$
BEGIN
  RAISE NOTICE 'Rewriting Table % for reason %',
               pg_event_trigger_table_rewrite_oid()::regclass,
               pg_event_trigger_table_rewrite_reason() ;
END;
$$ LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=# CREATE EVENT TRIGGER rewrite_trg1 ON table_rewrite
        EXECUTE PROCEDURE rewrite1() ;
CREATE EVENT TRIGGER
postgres=> ALTER TABLE data1 ALTER COLUMN c1 TYPE VARCHAR(10) ;
NOTICE:  Rewriting Table data1 for reason 4
ALTER TABLE
```

トリガー関数内では、以下の関数を使用することで ALTER TABLE 文または ALTER TYPE 文の発行元テーブルの情報を取得できます。

表 27 情報取得関数

関数名	説明	戻り値
pg_event_trigger_table_rewrite_oid	トリガー発生テーブル	oid
pg_event_trigger_table_rewrite_reason	トリガー発生理由	理由を示す整数

pg_event_trigger_table_rewrite_reason 関数は以下の値の OR を返します。これらの値はソースコード（src/include/commands/event_trigger.h）に記載されています。

表 28 pg_event_trigger_table_rewrite_reason 関数の戻り値

値	変更点	発行 SQL
1	永続化設定	ALTER TABLE SET UNLOGGED 等
2	DEFAULT	ALTER TABLE ADD COLUMN DEFAULT 等
4	列の変更	ALTER TABLE ADD COLUMN 等
8	OID 設定の変更	ALTER TABLE SET WITH OIDS 等

3.4.14 PL/pgSQL ASSERT 文

PL/pgSQL に ASSERT 文が追加されました。ASSERT 文はストアドプロシージャが想定する値のチェックを行うことができ、パラメーターによって ASSERT 文の動作を停止することができます。

構文

```
ASSERT condition [, 'message' ]
```

condition には値をチェックする条件文を記述します。条件が FALSE または NULL になると、ASSERT_EXCEPTION 例外が発生します。

message はオプションです。例外発生時に出力される文字列を指定することができます。省略すると「assertion failed」が出力されます。

例 48　ASSERT 文

```
postgres=> CREATE OR REPLACE FUNCTION assert1(NUMERIC)
        RETURNS NUMERIC
        AS $$
        BEGIN
            ASSERT $1 < 20 ;
            ASSERT $1 > 10, 'Assert Message' ;
            RETURN $1 * 2 ;
        END;
        $$ LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=> SELECT assert1(30) ;
ERROR:  assertion failed
CONTEXT:  PL/pgSQL function assert2(numeric) line 3 at ASSERT
postgres=> SELECT assert1(5) ;
ERROR:  Assert Message
CONTEXT:  PL/pgSQL function assert2(numeric) line 4 at ASSERT
```

□　パラメーター

ASSERT 文の動作を制御するためのパラメーターが plpgsql.check_asserts です。デフォルト値は on で、ASSERT 文が動作します。このパラメーターを off (false) に設定すると、

ASSERT 文は動作しなくなります。

例 49　パラメーターplpgsql.check_asserts

```
postgres=> SELECT assert1(5) ;
ERROR:  Assert Message
CONTEXT:  PL/pgSQL function assert1(numeric) line 3 at ASSERT
postgres=> SET plpgsql.check_asserts = false ;
SET
postgres=> SELECT assert1(5) ;
 assert1
---------
      10
(1 row)
```

3.4.15 jsonb 型に対する演算子と関数

jsonb 型に対する演算子と関数が追加されました。

□　「||」演算子

「||」演算子を使って、要素の追加と更新を行うことができます。同一キーを追加した場合は値が置換されます。「||」演算子と同じ動作は jsonb_concat 関数でも実現することができます。

例 50　追加と置換

```
postgres=> SELECT '{"key":"key1", "val1":"1000"}'::jsonb || '{"val2":"2000"}' ;
                    ?column?
-------------------------------------------------
 {"key": "key1", "val1": "1000", "val2": "2000"}
(1 row)

postgres=> SELECT '{"key":"key1", "val1":"1000"}'::jsonb || '{"val1":"3000"}' ;
             ?column?
---------------------------------
 {"key": "key1", "val1": "3000"}
(1 row)
```

□　「-」演算子、「#-」演算子

「-」演算子と「#-」演算子は要素の削除を行うことができます。キーが存在しない場合は元のデータは変化がありません。配列から番号を指定して要素を削除することもできます。番号は 0 から始まります。これらの演算子と同じ動作は jsonb_delete 関数でも実現できます。

例 51　削除

```
postgres=> SELECT '{"key":"key1", "val1":"1000"}'::jsonb - 'key' ;
     ?column?
------------------
 {"val1": "1000"}
(1 row)

postgres=> SELECT '["red", "green", "blue"]'::jsonb - 1 ;
    ?column?
----------------
 ["red", "blue"]
(1 row)
postgres=> SELECT '{"key":"key1", "val1":"1000"}'::jsonb - 'test' ;
            ?column?
---------------------------------
 {"key": "key1", "val1": "1000"}
(1 row)
```

例 52　入れ子構造の要素の削除

```
postgres=> SELECT
        '{"ename":{"first":"Mike", "last":"Stern"}, "gender":"M"}'::jsonb
        #-
        '{"ename","last"}'::text[] ;
                  ?column?
-------------------------------------------
 {"ename": {"first": "Mike"}, "gender": "M"}
(1 row)
```

□ jsonb_set 関数

jsonb_set 関数は要素の置換／追加を行います。4 つ目のパラメーターは要素が存在しない場合に、replacement パラメーターの要素を追加する場合に true を指定します。この関数は当初 jsonb_replace という名前でした。

構文

```
jsonb jsonb_set(target jsonb, path text[], new_value jsonb [,
        create_missing boolean DEFAULT true])
```

例 53　置換

```
postgres=> SELECT
 jsonb_set('{"key":"key1", "val1":"1000"}'::jsonb,'{"val1"}','2000') ;
         jsonb_set
-------------------------------
 {"key": "key1", "val1": 2000}
(1 row)
```

例 54　追加（1）

```
postgres=> SELECT
  jsonb_set('{"key":"key1", "val1":"1000"}'::jsonb,'{"val2"}','2000') ;
                    jsonb_set
-----------------------------------------------
 {"key": "key1", "val1": "1000", "val2": 2000}
(1 row)
```

例 55　追加（2）

```
postgres=> SELECT
  jsonb_set('{"key":"key1", "val1":"1000"}'::jsonb,'{"val2"}','2000', false) ;
                    jsonb_set
-----------------------------------------------
 {"key": "key1", "val1": "1000", "val1": 2000}
(1 row)
```

□ jsonb_pretty 関数

jsonb_pretty 関数は jsonb データの整形を行うことができます。

構文

```
jsonb jsonb_pretty(from_json jsonb)
```

例 56 整形

```
postgres=> SELECT jsonb_pretty('{"key":"key1", "val1":"1000",
        "arr1":["itm1","itm2"]}'::jsonb) ;
    jsonb_pretty
--------------------
 {                 +
     "key": "key1",+
     "arr1": [     +
         "itm1",   +
         "itm2"    +
     ],            +
     "val1": "1000"+
 }
(1 row)
```

□ jsonb_strip_nulls 関数

jsonb_strip_nulls 関数は jsonb データから null を削除します。

構文

```
jsonb jsonb_strip_nulls(from_json jsonb)
json json_strip_nulls(from_json json)
```

例 57 NULL 削除

```
postgres=> SELECT json_strip_nulls('[{"f1":1,"f2":null},2,null,3]') ;
  json_strip_nulls
--------------------
 [{"f1":1},2,null,3]
```

3.4.16 関数

以下の関数が拡張されました。

□ width_bucket 関数

width_bucket 関数に anyelement 型、anyarray 型をパラメーターに持つバージョンが追加されました。

□ generate_series 関数

generate_series 関数のパラメーターはすべて bigint 型、integer 型または timestamp 型でしたが、numeric 型もサポートされるようになりました。このため値の増分に小数点を含めることができるようになりました。

例 58 generate_series 関数の拡張

```
postgres=> SELECT generate_series (1.2, 2.1, 0.3) ;
 generate_series
-----------------
             1.2
             1.5
             1.8
             2.1
(4 rows)
```

□ 未検証の関数

以下の関数は追加されたことを確認しましたが、動作は未検証です。

- array_agg（anynonarray 型に対応）
- array_agg_array*
- array_position
- array_positions
- bernoulli
- binary_upgrade_*
- bound_box
- box（polygon 型に対応）
- brin*
- bttextsortsupport
- dist_cpoint
- dist_polyp

- dist_ppoly
- gist_bbox_distance
- gist_box_fetch
- gist_point_fetch
- gistcanreturn
- inet_gist_fetch
- inet_merge
- inet_same_family
- int8_avg_accum_inv
- json_strip_nulls
- jsonb_agg
- jsonb_agg_finalfn
- jsonb_agg_transfn
- jsonb_build_*
- jsonb_object*
- max（timezone without time zone 型に対応）
- min（timezone without time zone 型に対応）
- mxid_age
- network_larger
- network_smaller
- numeric_poly_*
- numeric_sortsupport
- pg_ddl_command_*
- pg_event_trigger_ddl_commands
- pg_event_trigger_table_rewite_*
- pg_get_object_address
- pg_identify_object_as_address
- pg_last_committed_xact
- pg_ls_dir（missing_ok, include_dot_dirs パラメータ追加）
- pg_read_binary_file（missing_ok パラメータ追加）
- pg_read_file（missing_ok パラメータ追加）
- pg_replication_origin_*
- pg_show_all_file_settings
- pg_show_replication_origin_status
- pg_stat_file（missing_ok パラメータ追加）
- pg_stat_get_snapshot_timestamp

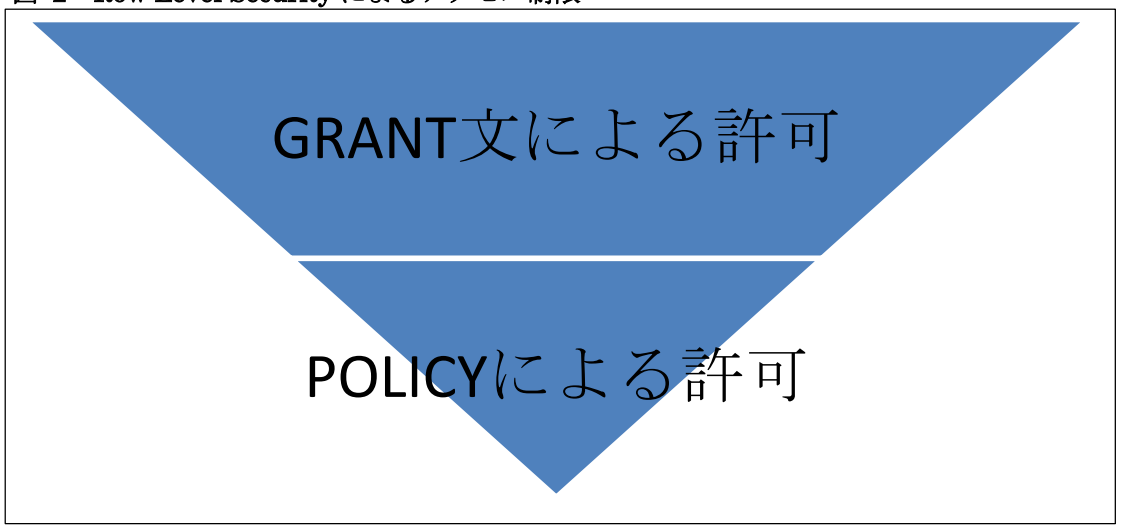
- pg_xact_commit_timestamp
- range_gist_fetch
- range_merge
- regnamespace*
- regrole*
- row_security_active
- system
- timestamp_izone_transform
- timestamp_zone_transform
- to_jsonb
- to_regnamespace
- to_regrole
- tsm_*

3.5 Row Level Security

3.5.1 Row Level Security とは

これまでの PostgreSQL では、テーブルや列に対してアクセスする権限を GRANT 文で指定しました。PostgreSQL 9.5 でもこの方法は有効です。Row Level Security は、GRANT 文で許可したレコードを更にレコード（タプル）レベルで制限することができる機能です。Row Level Security によるアクセス制限には POLICY と呼ばれるオブジェクトを作成します。

図 2　Row Level Security によるアクセス制限



3.5.2 準備

Row Level Security を利用するためには、ポリシーによる制限を行うテーブルに対して ALTER TABLE ENABLE ROW LEVEL SECURITY 文を実行します。標準ではテーブルに対する Row Level Security 設定は有効になっていません。テーブルに対する設定を無効にするには ALTER TABLE DISABLE ROW LEVEL SECURITY 文を実行します。

例 59　テーブルに対する機能の有効化

```
postgres=> ALTER TABLE poltbl1 ENABLE ROW LEVEL SECURITY ;
ALTER TABLE
postgres=> ¥d+ poltbl1
                                Table "public.poltbl1"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 c1     | numeric               |           | main     |              |
 c2     | character varying(10) |           | extended |              |
 uname  | character varying(10) |           | extended |              |
Policies (Row Security Enabled): (None)
```

3.5.3 ポリシーの作成

テーブルに対してアクセス権限を指定するにはポリシーを作成します。ポリシーは CREATE POLICY 文で作成します。POLICY の作成は一般ユーザーでも行うことができます。

構文

```
CREATE POLICY policy_name ON table_name
[ FOR { ALL | SELECT | INSERT | UPDATE | DELETE } ]
[ TO { roles | PUBLIC, ...} ]
USING (condition)
[ WITH CHECK (check_condition) ]
```

表 29 CREATE POLICY 文の構文

構文	説明
policy_name	ポリシーの名前を指定
ON	ポリシーを適用するテーブル名を指定
FOR	ポリシーを適用する操作または ALL
TO	ポリシーを許可する対象ロール名または PUBLIC
USING	タプルに対する許可を行う条件文（WHERE 句と同一構文）を記述します。USING 句で指定された条件が TRUE になるタプルのみが利用者に返されます。
WITH CHECK	UPDATE 文により更新できる条件を記述します。SELECT 文に対するポリシーでは CHECK 句は指定できません。

下記の例では、テーブル poltbl1 に対するポリシーを作成しています。TO 句を省略しているため、対象は全ユーザー（PUBLIC）、操作はすべての SQL（FOR ALL）、許可を行うタプルは uname 列が現在のユーザー名（current_user 関数）と同じタプルのみになります。

例 60　CREATE POLICY 設定

```
postgres=> CREATE POLICY pol1 ON poltbl1 FOR ALL USING (uname = current_user) ;
CREATE POLICY
postgres=> ¥d+ poltbl1
                                Table "public.poltbl1"
Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+------------
 c1     | numeric               |           | main     |              |
 c2     | character varying(10) |           | extended |              |
 uname  | character varying(10) |           | extended |              |
Policies:
    POLICY "pol1" FOR ALL
      USING (((uname)::name = "current_user"()))
```

作成したポリシーは pg_policy カタログから確認することができます。またポリシーを設定されたテーブルの情報は pg_policies カタログから確認できます。

以下の例では、ポリシーの効果を検証しています。

- テーブル poltbl1 のオーナーである user1 ユーザーが 3 レコードを格納しています（2～12 行）。
- ユーザーuser2 の権限でテーブル tblpol1 を検索していますが、uname 列の値が user2 であるレコード 1 件のみしか参照できません（15～19 行）。
- uname 列の値を変更しようとしていますが、CREATE POLICY 文の USING 句で指定した条件から逸脱するため、UPDATE 文が失敗しています（20～21 行）。

例 61　ポリシーの効果

```
1  $ psql -U user1
2  postgres=> INSERT INTO poltbl1 VALUES (100, 'Val100', 'user1') ;
3  INSERT 0 1
4  postgres=> INSERT INTO poltbl1 VALUES (200, 'Val200', 'user2') ;
5  INSERT 0 1
6  postgres=> INSERT INTO poltbl1 VALUES (300, 'Val300', 'user3') ;
7  INSERT 0 1
8  postgres=> SELECT COUNT(*) FROM poltbl1 ;
9   count
10 -------
11       3
12 (1 row)
13
14 $ psql -U user2
15 postgres=> SELECT * FROM poltbl1 ;
16  c1  |   c2    | uname
17 -----+---------+-------
18  200 | val200 | user2
19 (1 row)
20 postgres=> UPDATE poltbl1 SET uname='user3' ;
21 ERROR:  new row violates row level security policy for "poltbl1"
```

ポリシーを作成する CREATE POLICY 文は、ENABLE ROW LEVEL SECURITY 句を指定されていないテーブルに対して実行してもエラーにはなりません。この場合、該当テーブルに対しては ROW LEVEL SECURITY 機能は有効にならず、GRANT 文による許可のみが有効になります。

ポリシー設定の変更や削除はそれぞれ ALTER POLICY 文、DROP POLICY 文で行います。

3.5.4 パラメーターの設定

Row Level Security の機能はパラメーターrow_security で制御されます。以下の設定値をとることができます。このパラメーターはセッション単位で変更できます。

表 30　パラメーターrow_security

パラメーター値	説明
on	Row Level Security の機能を有効にします。この値はデフォルト値です。
off	Row Level Security の機能を無効にします。
force	Row Level Security の機能を強制します。ポリシーを設定されたテーブルに対してはポリシーの許可が強制されます。このためテーブル所有者でもポリシー違反のデータにはアクセスできなくなります。

□　ユーザー権限

SUPERUSER 権限を持つユーザーはポリシー設定をバイパスすることができます。BYPASSRLS 権限を持つユーザーは、パラメーターrow_security を off に設定することで、ポリシーをバイパスできます（セッション単位で設定可能）。BYPASSRLS 権限を持たないユーザーは、パラメーターrow_security を off にした環境ではテーブルにアクセスできません。CREATE USER 文のデフォルトは NOBYPASSRLS が指定されます。

3.6 BRIN インデックス

3.6.1 BRIN インデックスとは

BRIN インデックス（Block Range Index）は新しく作成されたインデックスの物理フォーマットです。従来の BTREE インデックスは、列値に対して正確なタプルのロケーションを保存していました。この方法はインデックスによる検索は高速ですが、インデックス自体の容量が大きくなるという欠点がありました。BRIN インデックスは、ブロック内の列値の最大値、最小値をまとめて保存することでストレージ容量の削減と高速化の両方を実現しています。特に大量のタプルを格納するテーブルに対するインデックスとして有効性が期待できます。

図 3 BRIN インデックスの構造イメージ

ブロック範囲	NULL 値あり	最小値	最大値
1-128	TRUE	100	1000
129-256	FALSE	200	400
257-384	TRUE	500	2000
385-512	FALSE	100	900

↑ pages_per_range

BRIN インデックスには追加オプションとして pages_per_range を指定できます。このパラメーターにはインデックスのエントリーが使用するページ数を指定します。デフォルト値は 128 です。オプションの最小値は 1、最大値は 131,072 です。

□ 構文

BRIN インデックスの作成には CREATE INDEX 文に USING BRIN 句を指定します。

構文

```
CREATE INDEX index_name ON table_name USING BRIN (column_name, ...)
[ WITH (pages_per_range = value) ]
```

3.6.2 作成例

以下は BRIN インデックスの作成例です。まずテーブルの作成とデータを格納します。

例 62　テーブルの作成

```
postgres=> CREATE TABLE brin1 (c1 NUMERIC, c2 NUMERIC, c3 NUMERIC, c4 NUMERIC) ;
CREATE TABLE
postgres=> INSERT INTO brin1 VALUES (
postgres(>   generate_series(1, 10000000),
postgres(>   generate_series(1, 10000000),
postgres(>   generate_series(1, 10000000),
postgres(>   generate_series(1, 10000000)
postgres(> ) ;
INSERT 0 10000000
```

次に、各列に対してインデックスを作成します。比較対象として c1 列には BTREE インデックスを作成します。c2 列から c4 列には、パラメーターpages_per_range が異なる 3 つの BRIN インデックスを作成します。

例 63　インデックスの作成と確認

```
postgres=> CREATE INDEX btree1 ON brin1 (c1) ;
CREATE INDEX
postgres=> CREATE INDEX brin1_def ON brin1 USING BRIN (c2) ;
CREATE INDEX
postgres=> CREATE INDEX brin1_64 ON brin1 USING BRIN (c3)
        WITH (pages_per_range = 64) ;
CREATE INDEX
postgres=> CREATE INDEX brin1_512 ON brin1 USING BRIN (c4)
        WITH (pages_per_range = 512) ;
CREATE INDEX
postgres=> ANALYZE VERBOSE brin1 ;
INFO:  analyzing "public.brin1"
INFO:  "brin1": scanned 30000 of 73520 pages, containing 4080483 live rows and
0 dead rows; 30000 rows in sample, 9999961 estimated total rows
ANALYZE
postgres=> ¥d+ brin1
                        Table "public.brin1"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 c1     | numeric |           | main    |              |
 c2     | numeric |           | main    |              |
 c3     | numeric |           | main    |              |
 c4     | numeric |           | main    |              |
Indexes:
    "brin1_512" brin (c4) WITH (pages_per_range=512)
    "brin1_64" brin (c3) WITH (pages_per_range=64)
    "brin1_def" brin (c2)
    "btree1" btree (c1)
```

格納領域を確認します。BRIN インデックスは非常に小さい領域で構成されていることがわかります。

例 64　ストレージの確認

```
postgres=> SELECT relname, pg_size_pretty(pg_relation_size(oid)) FROM
        pg_class WHERE relname LIKE 'brin1%' OR relname = 'btree1' ;
  relname  | pg_size_pretty
-----------+----------------
 brin1     | 574 MB
 btree1    | 214 MB
 brin1_def | 32 kB
 brin1_64  | 48 kB
 brin1_512 | 24 kB
(5 rows)
```

パフォーマンスを検証します。列名のみ異なる同一 SQL 文を複数回実行して、作成したインデックスを使用した結果を検証します。

例 65　BTREE インデックスの使用

```
postgres=> EXPLAIN ANALYZE SELECT * FROM brin1 WHERE c1 = 5000000 ;
                            QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using btree1 on brin1  (cost=0.43..8.45 rows=1 width=24)
        (actual time=0.085..0.086 rows=1 loops=1)
   Index Cond: (c1 = '5000000'::numeric)
 Planning time: 0.389 ms
 Execution time: 0.240 ms
(4 rows)
```

次は BRIN インデックスを使った例です。

例 66　BRIN インデックスの使用（パラメータはデフォルト）

```
postgres=> EXPLAIN ANALYZE SELECT * FROM brin1 WHERE c2 = 5000000 ;
                           QUERY PLAN
-----------------------------------------------------------------------------------------
 Bitmap Heap Scan on brin1  (cost=16.01..20.02 rows=1 width=24)
        (actual time=1.955..9.026 rows=1 loops=1)
   Recheck Cond: (c2 = '5000000'::numeric)
   Rows Removed by Index Recheck: 17407
   Heap Blocks: lossy=128
   ->  Bitmap Index Scan on brin1_def  (cost=0.00..16.01 rows=1 width=0)
         (actual time=0.747..0.747 rows=1280 loops=1)
          Index Cond: (c2 = '5000000'::numeric)
 Planning time: 0.144 ms
 Execution time: 9.222 ms
(8 rows)
```

例 67　BRIN インデックスの使用（pages_per_range=64）

```
postgres=> EXPLAIN ANALYZE SELECT * FROM brin1 WHERE c3 = 5000000 ;
                           QUERY PLAN
-----------------------------------------------------------------------------------------
Bitmap Heap Scan on brin1  (cost=24.01..28.02 rows=1 width=24)
        (actual time=2.542..5.310 rows=1 loops=1)
   Recheck Cond: (c3 = '5000000'::numeric)
   Rows Removed by Index Recheck: 8703
   Heap Blocks: lossy=64
   ->  Bitmap Index Scan on brin1_64  (cost=0.00..24.01 rows=1 width=0)
         (actual time=1.420..1.420 rows=640 loops=1)
          Index Cond: (c3 = '5000000'::numeric)
 Planning time: 0.140 ms
 Execution time: 5.360 ms
(8 rows)
```

例 68　BRIN インデックスの使用（pages_per_range=512）

```
postgres=> EXPLAIN ANALYZE SELECT * FROM brin1 WHERE c4 = 5000000 ;
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Bitmap Heap Scan on brin1  (cost=12.01..16.02 rows=1 width=24)
        (actual time=29.661..36.501 rows=1 loops=1)
   Recheck Cond: (c4 = '5000000'::numeric)
   Rows Removed by Index Recheck: 69631
   Heap Blocks: lossy=512
   ->  Bitmap Index Scan on brin1_512  (cost=0.00..12.01 rows=1 width=0)
         (actual time=0.258..0.258 rows=5120 loops=1)
          Index Cond: (c4 = '5000000'::numeric)
 Planning time: 0.146 ms
 Execution time: 36.561 ms
(8 rows)
```

3.6.3 情報確認

Contrib モジュール pageinspect には、BRIN インデックスの情報を取得する以下の関数が追加されました。

表 31　pageinspect モジュールに追加された関数

関数名	説明
brin_page_type	インデックス構成ページのページ種別を取得する
brin_metapage_info	メタデータ内の各種情報を取得する
brin_revmap_data	マップ・ページからタプルのリストを取得する
brin_page_items	データ・ページから格納されたデータを取得する

例 69　brin_page_items 関数の実行例

```
postgres=# CREATE EXTENSION pageinspect ;
CREATE EXTENSION
postgres=# SELECT itemoffset, blknum, value
   FROM brin_page_items(get_raw_page('brin1_def', 2), 'brin1_def') LIMIT 10 ;
 itemoffset | blknum |       value
------------+--------+--------------------
          1 |      0 | {1 .. 18745}
          2 |    128 | {18746 .. 36153}
          3 |    256 | {36154 .. 53561}
          4 |    384 | {53562 .. 70969}
          5 |    512 | {70970 .. 88377}
          6 |    640 | {88378 .. 105785}
          7 |    768 | {105786 .. 123193}
          8 |    896 | {123194 .. 140601}
          9 |   1024 | {140602 .. 158009}
         10 |   1152 | {158010 .. 175417}
(10 rows)
```

BRIN インデックスのオプション情報は、pg_class カタログの reloptions 列からも確認できます。

例 70 オプションの確認

```
postgres=> SELECT relname,reloptions FROM pg_class where relname like 'brin1%' ;
  relname  |      reloptions
-----------+----------------------
 brin1     |
 brin1_512 | {pages_per_range=512}
 brin1_64  | {pages_per_range=64}
 brin1_def |
(4 rows)
```

3.7 その他の新機能

3.7.1 プロセス名

新規追加されたパラメーターcluster_name に文字列を指定すると、プロセス名に指定された文字列が付与されます。パラメーターcluster_name に指定できる文字は ASCII 文字列（0x20～0x7E）です。これ以外のコードはクエスチョン・マーク（?）に変換されて出力されます。以下の例は cluster_name パラメーターに cluster1 を指定した場合のプロセス名です。postmaster プロセスはこのパラメーターの影響を受けません。

例 71 cluster_name パラメーター設定例

```
$ ps -ef | grep postgres
postgres 12364     1  0 06:14 pts/0  00:00:00 /usr/local/pgsql/bin/postgres -D data
postgres 12365 12364  0 06:14 ?    00:00:00 postgres: cluster1: logger process
postgres 12367 12364  0 06:14 ?    00:00:00 postgres: cluster1: checkpointer process
postgres 12368 12364  0 06:14 ?    00:00:00 postgres: cluster1: writer process
postgres 12369 12364  0 06:14 ?    00:00:00 postgres: cluster1: wal writer process
postgres 12370 12364  0 06:14 ?    00:00:00 postgres: cluster1: autovacuum launcher
process
```

データベースクラスタ間で cluster_name パラメーターのチェックを行っているわけではありません。単一ホスト内で同じ値の cluster_name を指定した複数インスタンスを起動してもエラーにはなりません。

3.7.2 EXPLAIN 文の出力

EXPLAIN VERBOSE 文の出力に、ソートに関する追加情報が出力されるようになりました。下記例の下線部が PostgreSQL 9.5 で追加された出力です。

例 72 ソート追加情報

```
postgres=> EXPLAIN VERBOSE SELECT * FROM sort1
                        ORDER BY c1 DESC, c2 COLLATE "C" ;
                                QUERY PLAN
-------------------------------------------------------------------------
 Sort  (cost=65.83..68.33 rows=1000 width=12)
   Output: c1, c2, ((c2)::character varying(10))
   Sort Key: sort1.c1 DESC, sort1.c2 COLLATE "C"
   ->  Seq Scan on public.sort1  (cost=0.00..16.00 rows=1000 width=12)
         Output: c1, c2, c2
```

3.7.3 レプリケーション関連ログ

レプリケーション環境のマスター・インスタンスでパラメーター log_replication_commands を on に設定すると、wal sender プロセスが実行するレプリケーション操作のログが出力されるようになります。このパラメーターを on にすると、ログの出力レベル LOG のメッセージが出力されます。デフォルト値(off)の場合には、DEBUG1 レベルでログが出力されます。ログの先頭には「received replication command:」の文字列と、レプリケーション関連コマンドが続きます。例えばスレーブ・インスタンスからの接続時に以下のログが出力されます。

例 73 スレーブ・インスタンス接続時のログ

```
LOG:  received replication command: IDENTIFY_SYSTEM
LOG:  received replication command: START_REPLICATION SLOT "slot_1" 0/7000000
TIMELINE 1
```

スレーブ・インスタンス停止時にはログ出力は行われません。

pg_basebackup コマンドもレプリケーションの機能を使用するため、以下のログが出力されます。

例 74 pg_basebackup コマンド実行時のログ

```
LOG:  received replication command: IDENTIFY_SYSTEM
LOG:  received replication command: BASE_BACKUP LABEL 'pg_basebackup base backup' WAL NOWAIT
```

3.7.4 型キャスト

oid 型からオブジェクト名に変換するための型キャストが追加されました。以下の型キャストが使用できます。

表 32　型キャスト

型キャスト	説明
regnamespace	スキーマ名を取得
regrole	ロール名を取得

下記の例では pg_class カタログから、public スキーマに含まれるテーブル名、オーナー名を取得しています。relnamespace 列、relowner 列は oid 型であるため型変換を行っています。

例 75 pg_class 検索

```
postgres=> SELECT relnamespace::regnamespace, relname, relowner::regrole
        FROM pg_class WHERE relnamespace = 'public'::regnamespace ;
 relnamespace | relname | relowner
--------------+---------+----------
 public       | data1   | user1
 public       | data2   | user1
(2 rows)
```

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。

- リリースノート
  http://www.postgresql.org/docs/9.5/static/release.html
- What's new in PostgreSQL 9.5
  https://wiki.postgresql.org/wiki/What%27s_new_in_PostgreSQL_9.5
- Commitfests
  https://commitfest.postgresql.org/
- PostgreSQL 9.5 Alpha Manual
  http://www.postgresql.org/docs/9.5/static/index.html
- GitHub
  https://github.com/postgres/postgres
- PostgreSQL 9.5 Alpha 2 のアナウンス
  http://www.postgresql.org/about/news/1604/
- PostgreSQL 9.5 WAL format
  https://wiki.postgresql.org/images/a/af/FOSDEM-2015-New-WAL-format.pdf
- PostgreSQL 9.5 新機能の情報（Michael Paquier さん）
  http://michael.otacoo.com/
- 日々の記録 別館（ぬこ@横浜さん）
  http://d.hatena.ne.jp/nuko_yokohama/
- v9.5 の新機能 Custom Scan/Join Interface
  http://www.slideshare.net/kaigai/postgresql-unconference-30may-tokyo
- Tablesample In PostgreSQL 9.5
  http://blog.2ndquadrant.com/tablesample-in-postgresql-9-5/

変更履歴

変更履歴

版	日付	作成者	説明
0.1	2015/07/06	篠田典良	社内レビュー版作成（Alpha 1） レビュー担当（敬称略）: 高橋智雄 竹島彰子 北山貴広
1.0	2015/08/07	篠田典良	公開版を作成（Alpha 2）

以上