Hewlett Packard
Enterprise

2017 年 5 月 22 日

PostgreSQL 10 Beta1 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

Hewlett Packard
Enterprise

目次

Hewlett Packard
Enterprise

1. 本文書について

1.1 本文書の概要

本文書は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 10 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 9.6 と PostgreSQL 10 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（一部掲載）
- ドキュメントの改善、ソース内の Typo 修正

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 9.6.3 (比較対象) PostgreSQL 10 Beta 1 (2017/5/15 21:27:43)
オペレーティング・システム	Red Hat Enterprise Linux 7 Update 1 (x86-64)

Hewlett Packard
Enterprise

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。正式版までに本文書で検証した仕様が変更される場合があります。本文書に対するご意見等ありましたら作成者 篠田典良（noriyoshi.shinoda@hpe.com）までお知らせください。

1.6 表記

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

表 2 例の表記ルール

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

構文は以下のルールで記載しています。

表 3 構文の表記ルール

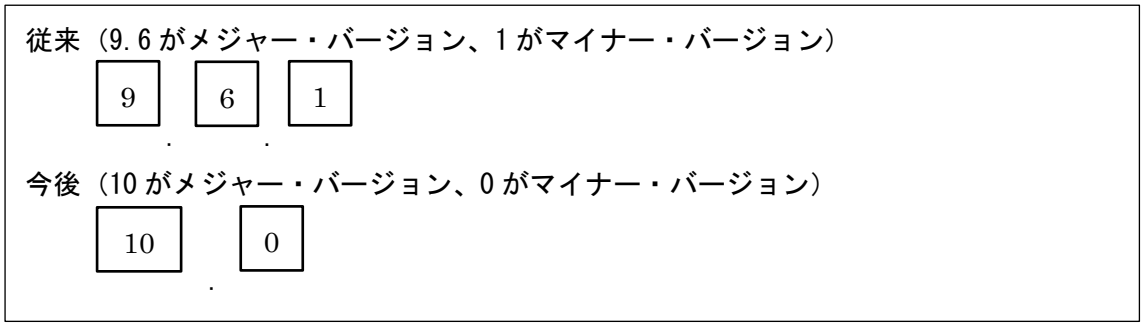
表記	説明
斜体	ユーザーが利用するオブジェクトの名前やその他の構文に置換
[ ]	省略できる構文であることを示す
{ A | B }	A または B を選択できることを示す
…	旧バージョンと同一である一般的な構文

Hewlett Packard
Enterprise

2. バージョン表記

PostgreSQL 10 からメジャー・バージョンとマイナー・バージョンの表記が変更されます。従来は最初の２つの数字がメジャー・バージョンを示していましたが、今後は最初の数のみがメジャー・バージョンを示します。

図 1 バージョン番号の表記



Hewlett Packard
Enterprise

3. 新機能解説

3.1 PostgreSQL 10 における変更点概要

PostgreSQL 10 には 100 以上の新機能が追加されました。代表的な新機能と利点について説明します。

3.1.1 大規模環境に対応する新機能

□ パーティション・テーブル

大規模なテーブルを物理的に分割する方法として、パーティション・テーブルが提供されます。従来の継承テーブルを利用したテーブル分割と異なり、データ格納時のパフォーマンスが大幅に向上しています。パーティション・テーブルの提供により、大規模なデータベースの構築がより容易になります。

□ Logical Replication

Logical Replication 機能により複数インスタンス間で一部のテーブルのみレプリケーションをおこなうことができます。従来のストリーミング・レプリケーションではスレーブ側インスタンスは読み取り専用でしたが、Logical Replication で同期されているテーブルには書き込みを行うこともできます。このためスレーブ側インスタンスに分析クエリー用のインデックスを作成することもできます。詳細は「3.3 Logical Replication」に記述されています。

□ パラレル・クエリーの拡張

PostgreSQL 9.6 では、大規模なテーブルの検索性能を向上させるためにパラレル・クエリー機能が提供されました。パラレル・クエリーは PostgreSQL 9.6 では Seq Scan だけで利用されていましたが、インデックス検索、マージ結合、ビットマップ結合等、多くの場面でパラレル・クエリーが利用できるようになりました。大規模テーブルに対する検索性能の向上が期待できます。詳細は「3.4 パラレル・クエリーの拡張」に記述されています。

3.1.2 信頼性向上に関する新機能

同期レプリケーションを行うインスタンスを任意に選択する Quorum-based 同期レプリケーションが利用できるようになりました（3.7 Quorum-base 同期レプリケーション）。従来のバージョンは WAL を出力しなかったハッシュ・インデックスが WAL を出力するようになりました。このためハッシュ・インデックスをレプリケーション環境でも利用できるよ

うになりました（3.5.7 ハッシュ・インデックスの WAL）。

3.1.3 運用性を向上させる新機能

pg_stat_activity カタログには出力される待機イベントが大幅に増えました。またすべてのバックエンド・プロセスの情報が格納されるようになりました（3.5.2 カタログの変更）。運用状況を確認する専用のロールが追加されています（3.5.8 ロールの追加）。

3.1.4 非互換

残念ながら PostgreSQL 10 には従来のバージョンとは互換性を持たない部分もあります。

□ 名称の変更

XLOG という名称はすべて WAL に統一されました。このため XLOG という名前を持つデータベース・クラスタ内のディレクトリ名、ユーティリティ・コマンド名、関数名、パラメーター名、エラー・メッセージが変更されています。例えばデータベース・クラスタ内の pg_xlog ディレクトリは pg_wal ディレクトリに変更されました。pg_receivexlog コマンドは pg_receivewal コマンドに変更されました。またログ・ファイルが出力されるディレクトリのデフォルト値が pg_log から log に変更されました。詳細は「3.5.4 XLOG から WAL へ変更」に記述されています。

□ pg_basebackup ユーティリティのデフォルト動作変更

標準で WAL のストリーミングを用いるようになりました。また-x パラメーターが廃止されました。詳細は「3.11.3 pg_basebackup」に記述されています。

□ pg_ctl のデフォルト動作変更

デフォルト状態で、すべての操作で、処理の完了を待機するようにふるまいが変更されました。従来のバージョンではインスタンスの起動処理等では処理の完了を待ちませんでした。詳細は「3.11.2 pg_ctl」に記述されています。

□ 平文パスワードの廃止

パスワードを暗号化せずに保存することができなくなりました。これによりセキュリティを向上させます。詳細は「3.9.12 CREATE USER 文」「3.10.2 変更されたパラメーター」に記述されています。

□ 廃止されたパラメーター

パラメーターmin_parallel_relation_size は min_parallel_table_scan_size に変更されました。パラメーターsql_inheritance は廃止されました。詳細は「3.10.4 廃止されたパラメーター」に記述されています。

□ 動作が変更された関数

to_date 関数、to_timestamp 関数は動作が変更されました。時刻を構成する各要素の数字のチェックが厳密に行われるようになった結果、従来のバージョンでは問題なかった値でエラーが発生します。また make_date 関数は紀元前の日付を指定できるようになりました。詳細は「3.9.13 関数」に記載されています。

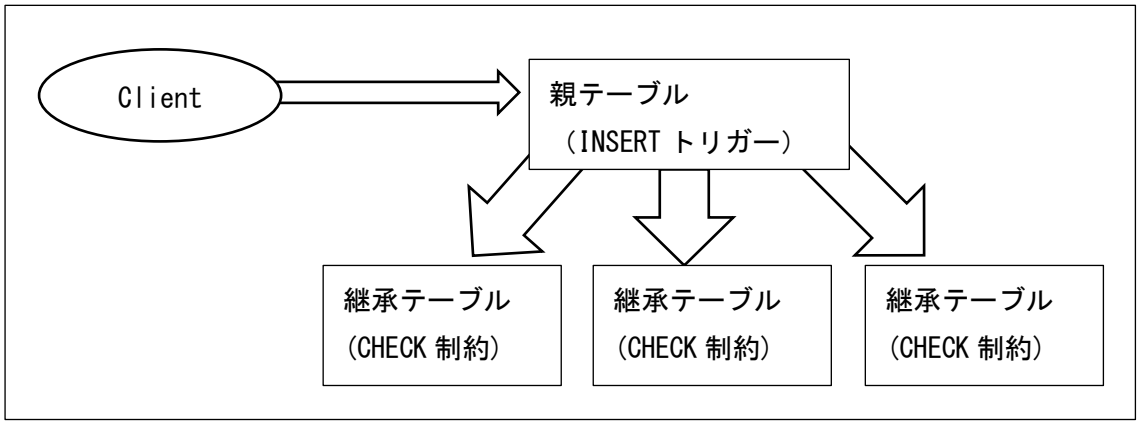
3.2 パーティション・テーブル

3.2.1 概要

従来の PostgreSQL では大規模なテーブルを物理的に分割する方法として、継承テーブル（INHERIT TABLE）の機能を利用していました。継承テーブルは親となるテーブルに対して複数の子テーブルを作成し、CHECK 制約とトリガーによりデータの整合性を維持する仕組みです。アプリケーションは親テーブルにアクセスを行い、透過的に子テーブルのデータを利用できます。しかしこの方法は以下の欠点がありました。

- データの整合性は子テーブルに個別に指定する CHECK 制約に依存する
- 親テーブルに対する INSERT 文の振り分けにはトリガー設定が必要で低速

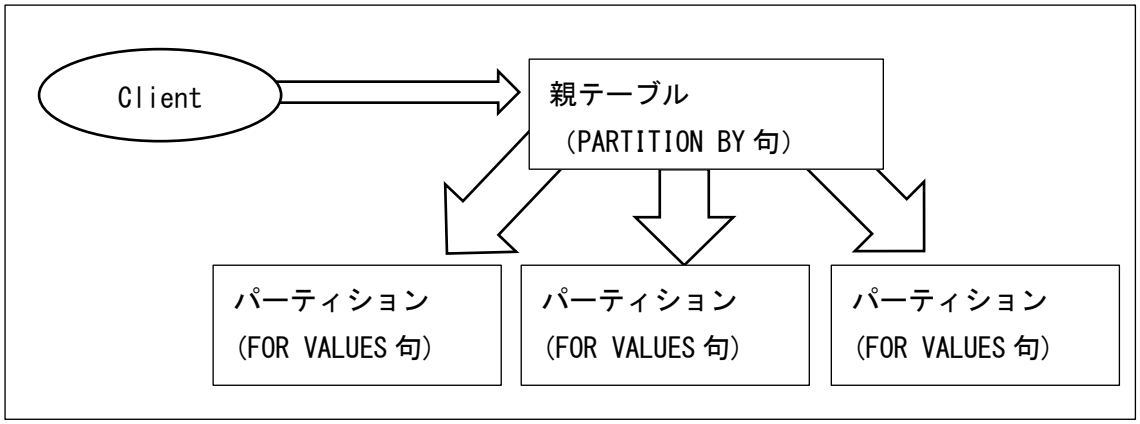
図 2 継承テーブルを使ったテーブル分割の仕組み



PostgreSQL 10 では、より洗練されたテーブルの分散方法としてパーティション・テーブルの機能が提供されました。パーティション・テーブルはアプリケーションがアクセスする親テーブルと同一構造を持つ子テーブルから構成されることは従来の継承テーブルと同じですが、INHERIT 指定、CHECK 制約、トリガーの設定が不要で、パーティションとなる子テーブルの追加や削除が簡単に行えるようになっています。

PostgreSQL のパーティション・テーブルにはパーティション化される列（または計算値）を指定します。格納される値の範囲を指定するレンジ・パーティションと、特定の値のみを指定するリスト・パーティションが利用できます。パーティションの種類は親テーブル作成時に決定されます。

図 3 パーティション・テーブルによる分割



3.2.2 リスト・パーティション

リスト・パーティション・テーブルは特定の値のみが格納できるパーティションを複数まとめる方法です。リスト・パーティション・テーブルを作成するには、まずアプリケーションからアクセスされる親テーブルを作成します。CREATE TABLE 文に PARTITION BY LIST 句を指定します。LIST 句にはパーティション分割対象の列名（または計算値）を指定します。列名は１つだけ指定できます。この時点ではテーブルに対する INSERT 文は失敗します。PARTITION BY 句を指定して作成されたテーブルは pg_class カタログの relkind 列の値が'p'になっています。

例 1 リスト・パーティション・テーブルの作成

```
postgres=> CREATE TABLE plist1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY LIST
(c1) ;
CREATE TABLE
```

次に実際にデータが格納される子テーブル（パーティション）を作成します。その際には PARTITION OF 句を使って親テーブルを指定し、FOR VALUES IN 句を使ってパーティション列に含む値を指定します。値はカンマ (,) で区切って複数指定することができます。

例 2 子テーブルの作成

```
postgres=> CREATE TABLE plist1_v100 PARTITION OF plist1 FOR VALUES IN (100) ;
CREATE TABLE
postgres=> CREATE TABLE plist1_v200 PARTITION OF plist1 FOR VALUES IN (200) ;
CREATE TABLE
```

作成が完了したパーティションテーブルの定義を参照します。

例 3 テーブル定義の参照

```
postgres=> ¥d+ plist1
                                        Table "public.plist1"
 Column |         Type          | Collation | Nullable | Default | …
--------+-----------------------+-----------+----------+---------+ …
 c1     | numeric               |           |          |         | …
 c2     | character varying(10) |           |          |         | …
Partition key: LIST (c1)
Partitions: plist1_v100 FOR VALUES IN ('100'),
            plist1_v200 FOR VALUES IN ('200')

postgres=> ¥d+ plist1_v100
                                      Table "public.plist1_v100"
 Column |         Type          | Collation | Nullable | Default | …
--------+-----------------------+-----------+----------+---------+ …
 c1     | numeric               |           |          |         | …
 c2     | character varying(10) |           |          |         | …
Partition of: plist1 FOR VALUES IN ('100')
Partition constraint: ((c1 IS NOT NULL) AND (c1 = ANY (ARRAY['100'::numeric])))
```

親テーブルに対する INSERT 文はパーティション化された子テーブルに自動的に振り分けられます。パーティションに含まれないデータの INSERT 文はエラーになります。

例 4 親テーブルに対する INSERT 文の実行

```
postgres=> INSERT INTO plist1 VALUES (100, 'data1') ;
INSERT 0 1
postgres=> INSERT INTO plist1 VALUES (200, 'data2') ;
INSERT 0 1
postgres=> INSERT INTO plist1 VALUES (300, 'data3') ;
ERROR:  no partition of relation "plist1" found for row
DETAIL:  Partition key of the failing row contains (c1) = (300).
```

パーティション化された子テーブルにも直接アクセス可能です。ただしパーティション対象列で指定された値以外は格納できません。

□ パーティション情報の取得

パーティション方法と列情報の取得には pg_get_partkeydef 関数を使用できます。各パーティションの制限は pg_get_partition_constraintdef 関数で取得できます。

例 5 パーティション情報の取得

```
postgres=> SELECT pg_get_partkeydef('plist1'::regclass) ;
 pg_get_partkeydef
-------------------
 LIST (c1)
(1 row)

postgres=> SELECT pg_get_partition_constraintdef('plist1_v100'::regclass) ;
              pg_get_partition_constraintdef
-----------------------------------------------------------
 ((c1 IS NOT NULL) AND (c1 = ANY (ARRAY['100'::numeric])))
(1 row)
```

3.2.3 レンジ・パーティション

レンジ・パーティション・テーブルは特定の値の範囲が格納できるパーティションを複数まとめる方法です。レンジ・パーティション・テーブルを作成するには、まずアプリケーションからアクセスされる親テーブルを作成します。CREATE TABLE 文に PARTITION BY RANGE 句を指定します。RANGE 句にはパーティション分割対象の列名（または計算値）を指定します。列名はカンマ（,）で区切ることで複数指定できます。パーティション化される列には自動的に NOT NULL 制約が指定されます（計算値の場合を除く）。この時点ではテーブルに対する INSERT 文は失敗します。

例 6 レンジ・パーティション・テーブルの作成

```
postgres=> CREATE TABLE prange1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY RANGE
(c1) ;
CREATE TABLE
```

次に実際にデータが格納される子テーブル（パーティション）を作成します。その際にはPARTITION OF 句を使って親テーブルを指定し、FOR VALUES FROM TO 句を使ってパーティションに含む値の範囲を指定します。パーティションには「FROM <= 値 <TO」の値のみ格納できます。

例 7 子テーブルの作成

```
postgres=> CREATE TABLE prange1_a1 PARTITION OF prange1 FOR VALUES FROM (100)
TO (200) ;
CREATE TABLE
postgres=> CREATE TABLE prange1_a2 PARTITION OF prange1 FOR VALUES FROM (200)
TO (300) ;
CREATE TABLE
```

作成が完了したパーティションテーブルの定義を参照します。

例 8 テーブル定義の参照

```
postgres=> ¥d+ prange1
                                        Table "public.prange1"
 Column |         Type          | Collation | Nullable | Default | …
--------+-----------------------+-----------+----------+---------+ …
 c1     | numeric               |           | not null |         | …
 c2     | character varying(10) |           |          |         | …
Partition key: RANGE (c1)
Partitions: prange1_a1 FOR VALUES FROM ('100') TO ('200'),
            prange1_a2 FOR VALUES FROM ('200') TO ('300')

postgres=> ¥d+ prange1_a1
                                       Table "public.prange1_a1"
 Column |         Type          | Collation | Nullable | Default | …
--------+-----------------------+-----------+----------+---------+ …
 c1     | numeric               |           | not null |         | …
 c2     | character varying(10) |           |          |         | …
Partition of: prange1 FOR VALUES FROM ('100') TO ('200')
Partition constraint: ((c1 >= '100'::numeric) AND (c1 < '200'::numeric))
```

親テーブルに対する INSERT 文はパーティション化された子テーブルに自動的に分割されます。パーティションに含まれないデータの INSERT 文はエラーになります。

例 9 親テーブルに対する INSERT 文の実行

```
postgres=> INSERT INTO prange1 VALUES (100, 'data1') ;
INSERT 0 1
postgres=> INSERT INTO prange1 VALUES (200, 'data2') ;
INSERT 0 1
postgres=> INSERT INTO prange1 VALUES (300, 'data3') ;
ERROR:  no partition of relation "prange1" found for row
DETAIL:  Partition key of the failing row contains (c1) = (300).
```

パーティション化された子テーブルにも直接アクセス可能です。ただしパーティション対象列で指定された値以外は格納できません。

例 10 子テーブルに対するアクセス

```
postgres=> SELECT * FROM prange1_a1 ;
 c1  |  c2
-----+-------
 100 | data1
(1 row)
postgres=> INSERT INTO prange1_a1 VALUES (200, 'data2') ;
ERROR:  new row for relation "prange1_a1" violates partition constraint
DETAIL:  Failing row contains (200, data2).
```

□ 範囲の UNBOUNDED 指定

RANGE パーティションの FROM 句または TO 句には具体的な値以外に UNBOUNDED を指定することができます。この指定は下限（FROM）または上限（TO）の範囲制限を行わないパーティションを作成できます。下記の例では prange1 テーブルのパーティションとして、100 未満の値と 100 以上の値で 2 つのテーブルを指定しています。

例 11 UNBOUNDED 指定

```
postgres=> CREATE TABLE prange1_1 PARTITION OF prange1 FOR VALUES FROM
(UNBOUNDED) TO (100) ;
CREATE TABLE
postgres=> CREATE TABLE prange1_2 PARTITION OF prange1 FOR VALUES FROM (100)
TO (UNBOUNDED) ;
CREATE TABLE
```

「既存の UNBOUNDED を含むパーティションを分割する値」を指定したパーティションは追加できません。

例 12 UNBOUNDED パーティションの分割

```
postgres=> CREATE TABLE prange1_3 PARTITION OF prange1 FOR VALUES FROM (200)
TO (300) ;
ERROR:  partition "prange1_3" would overlap partition "prange1_2"
```

3.2.4 既存テーブルとパーティション

既存のテーブルをパーティション・テーブルに登録、削除する方法について検証しました。

□ 子テーブルの ATTACH

既存テーブルをパーティション（子テーブル）として親テーブルに登録することができます。子テーブルは親テーブルと同様の列構成で作成する必要があります。またパーティションとして登録したテーブルを通常のテーブルに戻すこともできます。

例 13 親テーブルと同一構成のテーブル作成

```
postgres=> CREATE TABLE plist1_v100 (LIKE plist1) ;
CREATE TABLE
postgres=> CREATE TABLE plist1_v200 (LIKE plist1) ;
CREATE TABLE
```

作成したテーブルを親テーブルのパーティションとして登録します。ALTER TABLE ATTACH 文を実行します。同時にパーティション化列の値を指定します。下記の例では LIST パーティション c1 = 100 のデータが格納される plist1_v100 テーブルと c1 = 200 のデータが格納される plist1_v200 テーブルを登録しています。

例 14 パーティションの登録

```
postgres=> ALTER TABLE plist1 ATTACH PARTITION plist1_v100 FOR VALUES IN (100) ;
ALTER TABLE
postgres=> ALTER TABLE plist1 ATTACH PARTITION plist1_v200 FOR VALUES IN (200) ;
ALTER TABLE
```

パーティションとして登録できるオブジェクトは、テーブルまたは FOREIGN TABLE に限られます。

□ 子テーブルの DETACH

パーティション化された子テーブルを通常のテーブルに戻す場合は ALTER TABLE DETACH 文を実行します。

例 15 パーティションの解除

```
postgres=> ALTER TABLE plist1 DETACH PARTITION plist1_v100 ;
ALTER TABLE
```

3.2.5 パーティション・テーブルに対する操作

ここでは親テーブルや子テーブルに対する DDL や COPY 文を実行した場合の動作について検証しています。

□ 親テーブルに対する TRUNCATE

親テーブルに対する TRUNCATE 文の実行は全パーティションに伝播します。

例 16 親テーブルに対する TRUNCATE

```
postgres=> TRUNCATE TABLE part1 ;
TRUNCATE TABLE
postgres=> SELECT COUNT(*) FROM part1_v1 ;
 count
-------
     0
(1 row)
```

□ 親テーブルに対する COPY

親テーブルに対する COPY 文は子テーブルに伝播します。

例 17 親テーブルに対する COPY

```
postgres=# COPY part1 FROM '/home/postgres/part1.csv' WITH (FORMAT text) ;
COPY 10000
postgres=# SELECT COUNT(*) FROM part1_v1 ;
 count
-------
 10000
(1 row)
```

□ 親テーブルの削除

親テーブルを削除すると、子テーブルもすべて削除されます。子テーブルに対する DROP TABLE 文は子テーブルのみ削除します。

□ 親テーブルに対する列の追加／削除

親テーブルに列に対して列を追加／削除すると子テーブルも同じように変更されます。ただしパーティション・キーになっている列は削除できません。またパーティションが FOREIGN TABLE の場合、列の追加は自動的には行われません。

例 18 親テーブルに対する列の追加・削除

```
postgres=> ¥d part1
                    Table "public.part1"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | numeric               |           |          |
 c2     | character varying(10) |           |          |
Partition key: LIST (c1)
Number of partitions: 2 (Use ¥d+ to list them.)

postgres=> ALTER TABLE part1 ADD c3 NUMERIC ;
ALTER TABLE
postgres=> ¥d part1_v1
                    Table "public.part1_v1"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | numeric               |           |          |
 c2     | character varying(10) |           |          |
 c3     | numeric               |           |          |
Partition of: part1 FOR VALUES IN ('100')

postgres=> ALTER TABLE part1 DROP c1 ;
ERROR:  cannot drop column named in partition key
```

□ 一時テーブル

親テーブルまたはパーティション・テーブル共に一時テーブル(TEMPORATY TABLE)を使用することができます。ただし親テーブルが一時テーブルの場合、パーティション・テーブルも一時テーブルにする必要があります。

□ UNLOGGED テーブル

親テーブルまたはパーティション・テーブルに UNLOGGED テーブルを使用することができます。

□ 階層構造

異なる列をパーティション化することで、階層構造のパーティション・テーブルも作成で

きます。下記の例では c1 列でパーティション化したテーブル配下に c2 列でパーティション化したテーブルを追加しています。

例 19 階層パーティション

```
postgres=> CREATE TABLE part2 (c1 NUMERIC, c2 NUMERIC, c3 VARCHAR(10)) PARTITION
BY LIST (c1) ;
CREATE TABLE
postgres=> CREATE TABLE part2_v1 PARTITION OF part2 FOR VALUES IN (100)
PARTITION BY LIST (c2) ;
CREATE TABLE
postgres=> CREATE TABLE part2_v1_v2 PARTITION OF part2_v1 FOR VALUES IN (200) ;
CREATE TABLE
```

3.2.6 実行計画

WHERE 句にパーティションを特定できる情報が存在する場合、特定のパーティションのみにアクセスする実行計画が作成されます。

例 20 パーティションを特定できる SQL と実行計画

```
postgres=> EXPLAIN SELECT * FROM plist1 WHERE c1 = 100 ;
                            QUERY PLAN
------------------------------------------------------------------
 Append  (cost=0.00..20.38 rows=4 width=70)
   ->  Seq Scan on plist1_v100  (cost=0.00..20.38 rows=4 width=70)
         Filter: (c1 = '100'::numeric)
(3 rows)
```

しかし、WHERE 句の左辺が計算式になっていた場合等、パーティションが特定できない場合は全パーティションにアクセスする実行計画が作成されます。

例 21 パーティションを特定できない SQL と実行計画

```
postgres=> EXPLAIN SELECT * FROM plist1 WHERE c1 + 1 = 101 ;
                            QUERY PLAN
------------------------------------------------------------------------
 Append  (cost=0.00..44.90 rows=8 width=70)
   ->  Seq Scan on plist1_v100  (cost=0.00..22.45 rows=4 width=70)
         Filter: ((c1 + '1'::numeric) = '101'::numeric)
   ->  Seq Scan on plist1_v200  (cost=0.00..22.45 rows=4 width=70)
         Filter: ((c1 + '1'::numeric) = '101'::numeric)
(5 rows)
```

3.2.7 カタログ

パーティション化された親テーブルの情報は pg_partitioned_table カタログで確認できます。下記はテーブル名 part1、リスト・パーティション（partstrat='l'）、アタッチした子テーブル数が 2（partnatts=2）のテーブルの情報です。

例 22 親テーブルの情報

```
postgres=> SELECT partrelid::regclass, * FROM pg_partitioned_table ;
-[ RECORD 1 ]-+------
partrelid     | part1
partrelid     | 16444
partstrat     | l
partnatts     | 2
partattrs     | 1
partclass     | 3125
partcollation | 0
partexprs     |
```

pg_class カタログの relispartition 列が true になっているテーブルが子テーブルになっているテーブルです。また子テーブルには pg_class カタログの relpartbound 列にパーティション境界の情報が格納されます。この列の情報は pg_get_expr 関数で見やすく変換できます。

例 23 子テーブルの情報

```
postgres=> SELECT relname, relispartition, relpartbound FROM pg_class WHERE
relname = 'prange1v1' ;
-[ RECORD 1 ]--+-----------------------------------------------------------------
relname        | prange1v1
relispartition | t
relpartbound   | {PARTITIONBOUND :strategy r :listdatums <> :lowerdatums
({PARTRANGEDATUM :infinite false :value {CONST :consttype 1700 :consttypmod -
1 :constcollid 0 :constlen -1 :constbyval false :constisnull false :location -
1 :constvalue 8 [ 32 0 0 0 0 -128 100 0 ]}}) :upperdatums
({PARTRANGEDATUM :infinite false :value {CONST :consttype 1700 :consttypmod -
1 :constcollid 0 :constlen -1 :constbyval false :constisnull false :location -
1 :constvalue 8 [ 32 0 0 0 0 -128 -56 0 ]}})}

postgres=> SELECT relname, relispartition, pg_get_expr(relpartbound, oid) FROM
pg_class WHERE relname = 'prange1v1' ;
-[ RECORD 1 ]--+---------------------------------
relname        | prange1v1
relispartition | t
pg_get_expr    | FOR VALUES FROM ('100') TO ('200')
```

3.2.8 制約

パーティション・テーブルには以下の制約があります。

□ パーティション化列の個数

CREATE TABLE 文の PARTITION BY LIST 句に指定できる列はひとつだけです。列名部分に関数や括弧で囲んだ計算式を指定することはできます。

例 24 関数を使ったパーティション

```
postgres=> CREATE TABLE plist2(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY LIST
(upper(c2)) ;
CREATE TABLE
```

□ パーティション化列に対する NULL

レンジ・パーティションのパーティション化列には NULL 値を格納することができません。リスト・パーティションの場合は NULL 値を含むパーティションを作成することで格納することができるようになります。

例 25 レンジ・パーティションと NULL 値

```
postgres=> CREATE TABLE partnl(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY RANGE (c1) ;
CREATE TABLE
postgres=> CREATE TABLE partnlv PARTITION OF partnl FOR VALUES FROM (UNBOUNDED) TO (UNBOUNDED) ;
CREATE TABLE
postgres=> INSERT INTO partnl VALUES (NULL, 'null value') ;
ERROR:  range partition key of row contains null
```

□ 子テーブルの制約

子テーブルは親テーブルと同一構造を持つ必要があります。列の追加、不足、データ型の変更はできません。

例 26 異なる構造の子テーブルを使ったパーティション

```
postgres=> CREATE TABLE plist3(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY LIST (c1) ;
CREATE TABLE
postgres=> CREATE TABLE plist3_v100 (c1 NUMERIC, c2 VARCHAR(10), c3 NUMERIC) ;
CREATE TABLE
postgres=> ALTER TABLE plist3 ATTACH PARTITION plist3_v100 FOR VALUES IN (100) ;
ERROR:  table "plist3_v100" contains column "c3" not found in parent "plist3"
DETAIL:  New partition should contain only the columns present in parent.
postgres=> CREATE TABLE plist3_v200 (c1 NUMERIC);
CREATE TABLE
postgres=> ALTER TABLE plist3 ATTACH PARTITION plist3_v200 FOR VALUES IN (200) ;
ERROR:  child table is missing column "c2"
```

□ 主キー制約／一意制約／チェック制約

親テーブルに主キー制約（または一意制約）を指定できません。パーティション・テーブル全体の一意性は子テーブルの主キー設定に依存します。親テーブルに対する CHECK 制約は指定できます。子テーブルを作成すると、CHECK 制約は子テーブルにも自動的に追加されます。

例 27 親テーブルに主キーを追加

```
postgres=> ALTER TABLE plist1 ADD CONSTRAINT pl_plist1 PRIMARY KEY (c1) ;
ERROR:  primary key constraints are not supported on partitioned tables
LINE 1: ALTER TABLE plist1 ADD CONSTRAINT pl_plist1 PRIMARY KEY (c1)...
                                          ^
```

□ INSERT ON CONFLICT 文

親テーブルに対する INSERT ON CONFLICT 文は実行できません。

□ パーティション化列の UPDATE

パーティション化された列の値を更新する場合は、子テーブルの FOR VALUES 句に含まれる値にのみ更新できます。子テーブルに含むことができない値には更新できません。

例 28 パーティション化列の更新

```
postgres=> UPDATE plist1 SET c1 = 200 WHERE c1 = 100;
ERROR:  new row for relation "plist1_v100" violates partition constraint
DETAIL:  Failing row contains (200, data1).
```

上記エラーが発生するため、子テーブル間のデータ移動は UPDATE 文では実現できません（DELETE RETURNING INSERT 文を使用）。

□ データ格納済みテーブルの ATTACH

既にデータが格納されている子テーブルを親テーブルのパーティションとして ATTACH できます。しかしその場合はパーティションに含めることができるか全タプルがチェックされます。

例 29 タプルを含むパーティションの ATTACH

```
postgres=> CREATE TABLE plist2 (c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY LIST
(c1) ;
CREATE TABLE
postgres=> CREATE TABLE plist2_v100 (LIKE plist2) ;
CREATE TABLE
postgres=> INSERT INTO plist2_v100 VALUES (100, 'data1') ;
INSERT 0 1
postgres=> INSERT INTO plist2_v100 VALUES (200, 'data2') ;
INSERT 0 1
postgres=> ALTER TABLE plist2 ATTACH PARTITION plist2_v100 FOR VALUES IN (100) ;
ERROR:  partition constraint is violated by some row
```

□ 列値が重なるパーティション

範囲が重なるレンジ・パーティションや、同一の値が指定されたリスト・パーティションは作成できません。下記の例では列値が 100 から 200 のパーティションと 150 から 300 のパーティションをアタッチしようとしていますがエラーになっています。

例 30 列値が重なるパーティション

```
postgres=> ALTER TABLE prange2 ATTACH PARTITION prange2_v1 FOR VALUES FROM
(100) TO (200) ;
ALTER TABLE
postgres=> ALTER TABLE prange2 ATTACH PARTITION prange2_v2 FOR VALUES FROM
(150) TO (300) ;
ERROR:  partition "prange2_v2" would overlap partition "prange2_v1"
```

□ FOREIGN TABLE を子テーブルに指定

FOREIGN TABLE を子テーブルに指定することができます。ただしこの場合、集計処理のプッシュダウンは有効に動作しないようです。

例 31 FOREIGN TABLE をアタッチ

```
postgres=# CREATE FOREIGN TABLE datar2(c1 NUMERIC, c2 VARCHAR(10)) SERVER
remote1 ;
CREATE FOREIGN TABLE
postgres=# ALTER TABLE pfor1 ATTACH PARTITION datar2 FOR VALUES IN ('data2') ;
ALTER TABLE
postgres=# SELECT COUNT(*) FROM pfor1 WHERE c2='data2' ;
```

例 32 リモート・インスタンスで実行される SQL

```
statement: START TRANSACTION ISOLATION LEVEL REPEATABLE READ
execute <unnamed>: DECLARE c1 CURSOR FOR
        SELECT NULL FROM public.datar2
statement: FETCH 100 FROM c1
statement: CLOSE c1
statement: COMMIT TRANSACTION
```

子テーブルが FOREIGN TABLE の場合 INSERT 文が失敗します。

例 33 INSERT 文の失敗

```
postgres=# INSERT INTO pfor1 VALUES (100, 'data1') ;
ERROR:  cannot route inserted tuples to a foreign table
```

□ インデックス

インデックスの作成は子テーブル単位に行う必要があります。親テーブルにはインデックスを作成できません。

例 34 インデックス作成の失敗

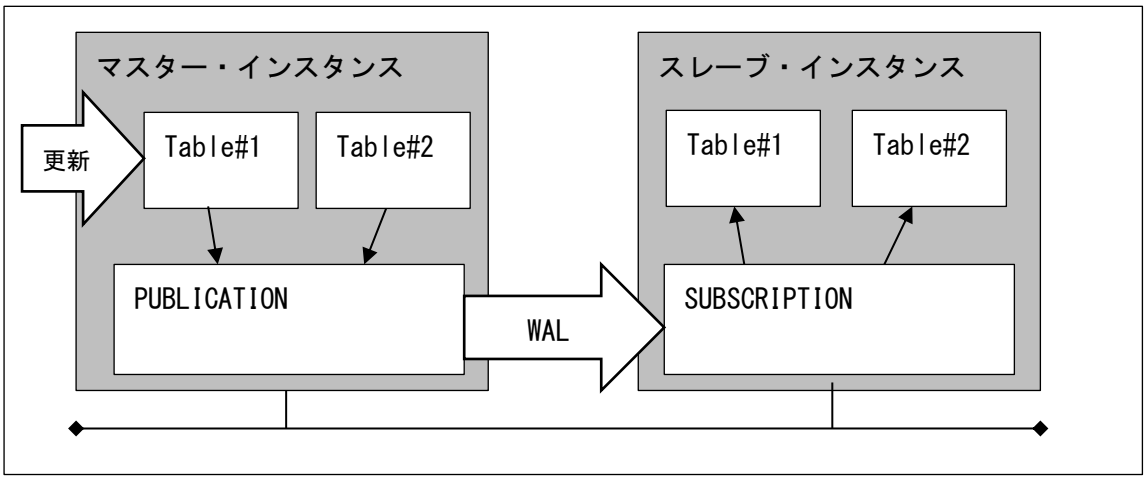
```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY LIST
(c1) ;
CREATE TABLE
postgres=> CREATE INDEX idx1_part1 ON part1(c1) ;
ERROR:  cannot create index on partitioned table "part1"
```

3.3 Logical Replication

3.3.1 概要

Logical Replication はテーブル単位にインスタンス間でレプリケーションを行う機能です。PostgreSQL 10 では Logical Replication 機能を実現するためにマスターとなるテーブルを管理する PUBLICATION と、スレーブ側インスタンスで作成される SUBSCRIPTION と呼ばれるオブジェクトを構成します。マスター側とスレーブ側でスキーマ名も含めて同一名称のテーブル間でレプリケーションを行います。同等の機能を持つ既存ソフトウェアには Slony-I がありますが、Logical Replication はトリガーを使用せず、スレーブ側のテーブルも更新可能である点が異なります。

図 4 オブジェクト構成



Logical Replication は PostgreSQL 9.4 で実装された Logical Decoding の基盤を元に、レプリケーション用標準プラグイン pgoutput により実装されています。

□ PUBLICATION オブジェクト

PUBLICATION はマスター・インスタンスで作成するオブジェクトです。PUBLICATION オブジェクトにはレプリケーションを行うテーブルを登録します。単一の PUBLICATION オブジェクトで複数のテーブルをレプリケーション対象とすることができます。PUBLICATION 単位でレプリケーションを行う操作(INSERT / DELETE / UPDATE)を選択することができます。デフォルトでは全操作 (DML) がスレーブ側で適用されます。PUBLICATION オブジェクトはデータベースに対する CREATE 権限を持つユーザーが作成できます。psql コマンドからは¥dRp コマンドで一覧を表示できます。

構文 1 PUBLICATION の作成

```
CREATE PUBLICATION name
[ FOR TABLE [ ONLY ] table_name [*] [, … ] | FOR ALL TABLES ]
[ WITH ( options [ = value] [, …] ) ]
```

FOR TABLE 句はレプリケーション対象のテーブルを指定します。カンマ（,）で区切って複数のテーブルを指定することもできます。WITH 句には対象となる DML 文を指定します。省略時はすべての DML が対象になります。publish オプションで DML 名をカンマ（,）で区切って指定することで対象となる DML を指定することができます。ONLY 句を省略すると継承関係がある子テーブルも含めてレプリケーション対象にします。

FOR ALL TABLES を指定するとデータベース内の全てのテーブルがレプリケーション対象になります。PUBLICATION 側でテーブルが追加されると自動的にレプリケーション対象として登録されます。

PUBLICATION を変更するには ALTER PUBLICATION 文を実行します。ADD TABLE 句を指定することでレプリケーション対象テーブルを PUBLICATION オブジェクトに追加できます。DROP TABLE 句はレプリケーション対象を削除します。SET TABLE 句は PUBLICATION に含まれるテーブルを指定されたテーブルのみに限定します。レプリケーション対象の DML を変更する場合は ALTER PUBLICATION SET 文を実行します。

構文 2 PUBLICATION の変更

```
ALTER PUBLICATION name ADD TABLE [ ONLY ] table_name [, table_name … ]
ALTER PUBLICATION name SET TABLE [ ONLY ] table_name [, table_name … ]
ALTER PUBLICATION name DROP TABLE [ ONLY ] table_name [, table_name … ]
ALTER PUBLICATION name SET ( option [ = value ] [ , … ] )
ALTER PUBLICATION name OWNER TO { owner | CURRENT_USER | SESSION_USER }
ALTER PUBLICATION name RENAME TO new_name
```

PUBLICATION オブジェクトを削除するには DROP PUBLICATION 文を実行します。

構文 3 PUBLICATION の削除

```
DROP PUBLICATION [IF EXISTS] name [ , …] [ { CASCADE | RESTRICT } ]
```

PUBLICATION オブジェクトは複数の SUBSCRIPTION からレプリケーションの依頼を受け取ることができます。また、テーブルは同時に複数の PUBLICATION に所属するこ

とができます。

□ SUBSCRIPTION オブジェクト

SUBSCRIPTION は PUBLICATION オブジェクトに接続し、wal sender プロセス経由で受信した WAL 情報を元にテーブルを更新するオブジェクトです。更新対象のテーブルは、接続先の PUBLICATION オブジェクトが管理するテーブルと同一名称（スキーマ名を含む）のテーブルです。

SUBSCRIPTION オブジェクトを作成するには CREATE SUBSCRIPTION 文を実行します。CONNECTION 句には PUBLICATION を作成したインスタンスに対する接続文字列を指定します。PUBLICATION オブジェクトを作成したデータベース名を dbname パラメーターに指定します。ストリーミング・レプリケーションと同様に、REPLICATION 属性を持つユーザーの接続が必要です。PUBLICATION 側で pg_hba.conf ファイルの編集が必要になる場合があります。PUBLICATION 句にはレプリケーション対象テーブルを管理する PUBLICATION オブジェクト名を指定します。PUBLICATION オブジェクトは複数指定することができます。SUBSCRIPTION オブジェクトの作成には SUPERUSER 権限が必要です。psql コマンドからは¥dRs コマンドで一覧を表示できます。

構文 4 SUBSCRIPTION の作成

```
CREATE SUBSCRIPTION name CONNECTION 'conn_info' PUBLICATION publication_name
        [, publication_name …] [ WITH ( option [ = value ] , … ) ]
```

表 4　option 指定

構文	説明	備考
enabled	SUBSCRIPTION は有効	デフォルト
create_slot	レプリケーション・スロット作成	デフォルト
slot_name = *name* | NONE	スロット名	デフォルトは SUBSCRIPTION 名
copy_data	初期データのコピーを行う	デフォルト
connect	PUBLICATION に接続する	デフォルト
synchronous_commit	同期コミット設定	デフォルト OFF

デフォルト設定では PUBLICATION インスタンスに SUBSCRIPTION と同じ名前の Logical Replication Slot が作成されます。また CREATE SUBSCRIPTION 文で指定した PUBLICATION オブジェクトが実際に存在するかはチェックされていません。

構文 5 SUBSCRIPTION の変更

```
ALTER SUBSCRIPTION name CONNECTION 'connection'
ALTER SUBSCRIPTION SET PUBLICATION publication_name [, publication_name …]
     { REFRESH [ WITH ( option [ = value ] ) ] | SKIP REFRESH }
ALTER SUBSCRIPTION name REFRESH PUBLICATION WITH ( option [, option … ] )
ALTER SUBSCRIPTION name { ENABLE | DISABLE }
ALTER SUBSCRIPTION SET ( option [ = value ] [ , … ] )
ALTER SUBSCRIPTION name OWNER TO owner | CURRENT_USER | SESSION_USER
ALTER SUBSCRIPTION name RENAME TO new_name
```

SUBSCRIPTION オブジェクトの変更には ALTER SUBSCRIPTION 文を実行します。option 句には CREATE SUBSCRIPTION 文と同じ値が指定できます。PUBLICATION オブジェクトに対してテーブルを追加した場合は ALTER SUBSCRIPTION REFRESH PUBLICATION 文を実行する必要があります。

SUBSCRIPTION オブジェクトの削除には DROP SUBSCRIPTION 文を使用します。デフォルトでは PUBLICATION 側に作成されたレプリケーション・スロットも削除します。PUBLICATION 側インスタンスが停止している場合等は ALTER SUBSCRIPTION DISABLE 文および ALTER SUBSCRIPTION SET (slot_name=NONE)文でレプリケーション・スロットを解除してから DROP SUBSCRIPTION 文を実行します。

構文 6 SUBSCRIPTION の削除

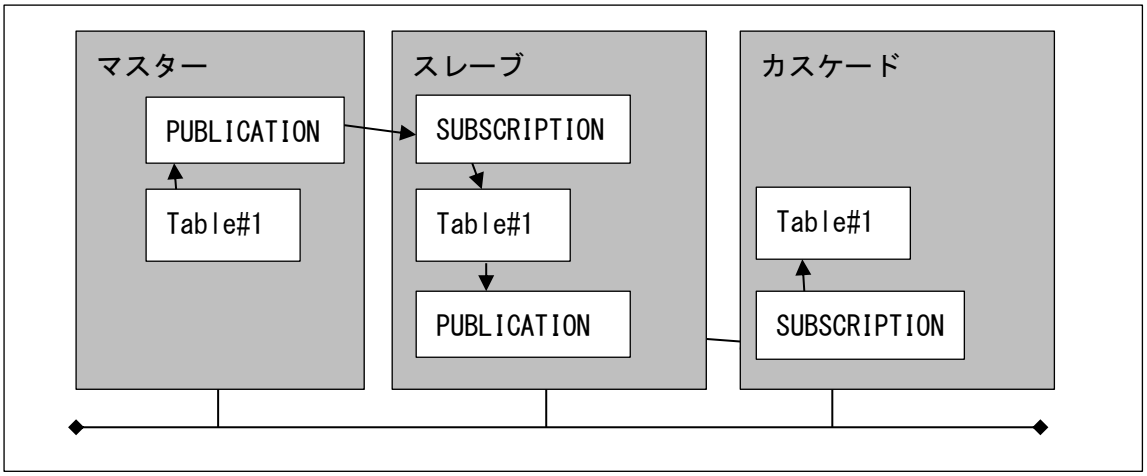
```
DROP SUBSCRIPTION [IF EXISTS] name [ { CASCADE | RESTRICT } ]
```

PUBLICATION オブジェクトと SUBSCRIPTION オブジェクトが追加されたため、COMMENT ON 文と SECURITY LABEL 文に PUBLICATION および SUBSCRIPTION が指定できるようになりました。

□ カスケード化

レプリケーション環境のカスケード化を行うことができることを確認しました。

図 5 カスケード構成



3.3.2 関連するリソース

Logical Replication を構成するオブジェクトや関連するパラメーターについて説明します。

□ プロセス

標準状態でプロセス「bgworker: logical replication launcher」が起動します。また SUBSCRIPTION オブジェクトを作成したインスタンスには SUBSCRIPTION 単位にワーカー・プロセス「bgworker: logical replication worker for subscription」が起動します。

SUBSCRIPTION ワーカープロセスはマスター・インスタンスに接続します。SUBSCRIPTION 側へ WAL を転送するために、マスター側インスタンスで wal sender プロセスが起動します。

□ カタログ

以下のカタログが新規に追加されました。

表 5 追加されたカタログ

カタログ	内容	インスタンス
pg_publication	PUBLICATION 情報	マスター
pg_publication_rel	WAL 転送対象のテーブル情報	マスター
pg_publication_tables	WAL 転送対象のテーブル情報	マスター
pg_stat_subscription	SUBSCRIPTION に受信した WAL の情報	スレーブ
pg_subscription	SUBSCRIPTION 情報	スレーブ
pg_subscription_rel	レプリケーション・テーブル情報	スレーブ

また pg_stat_replication カタログ、pg_replication_slots カタログにもレコードが追加されます。pg_stat_subscription カタログは SUBSCRIPTION オブジェクトを作成したインスタンスで Logical Replication の状況を確認できます。このカタログは一般ユーザーでも検索できます。

例 35 pg_stat_subscription カタログの検索

```
postgres=> SELECT * FROM pg_stat_subscription ;
-[ RECORD 1 ]---------+-----------------------------
subid                 | 16396
subname               | sub1
pid                   | 23275
relid                 |
received_lsn          | 0/1650C68
last_msg_send_time    | 2017-05-18 23:22:56.654912+09
last_msg_receipt_time | 2017-05-18 23:22:56.654939+09
latest_end_lsn        | 0/1650C68
latest_end_time       | 2017-05-18 23:22:56.654912+09
```

□ パラメーター

Logical Replication 設定には以下のパラメーターが関連します。

表 6　関連するパラメーター

パラメーター	インスタンス	内容
max_replication_slots	マスター	レプリケーション・スロットの最大数
max_wal_senders	マスター	wal senders プロセスの最大数
max_logical_replication_workers	スレーブ（新規）	logical replication worker プロセスの最大数
wal_level	マスター	logical に指定が必要
max_worker_processes	マスター／スレーブ	ワーカー・プロセスの最大数
max_sync_workers_per_subscription	スレーブ（新規）	初期データのコピーを行う際の並列度設定

□ レプリケーション・スロット

CREATE SUBSCRIPTION 文を実行すると、PUBLICATION 側インスタンスに SUBSCRIPTION と同じ名前のレプリケーション・スロットが作成されます（デフォルト設定の場合）。既に同じ名前のレプリケーション・スロットが存在する場合、CREATE SUBSCRIPTION 文はエラーになります。

例 36 レプリケーション・スロットの状態

```
postgres=> SELECT * FROM pg_replication_slots ;
-[ RECORD 1 ]-------+-----------
slot_name           | sub1
plugin              | pgoutput
slot_type           | logical
datoid              | 16385
database            | postgres
temporary           | f
active              | t
active_pid          | 12140
xmin                |
catalog_xmin        | 606
restart_lsn         | 0/535A1AF0
confirmed_flush_lsn | 0/535A1B28
```

3.3.3 実行例

下記の例ではレプリケーションを行うテーブル schema1.data1 を作成しています。次に PUBLICATION オブジェクトを作成し、schema1.data1 テーブルを PUBLICATION オブジェクトに登録しています。

例 37 レプリケーション対象テーブル作成（マスター／スレーブ）

```
postgres=> CREATE TABLE schema1.data1(c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
```

PUBLICATION オブジェクトを作成し、schema1.table1 テーブルを追加します。

例 38 PUBLICATION オブジェクト作成（マスター）

```
postgres=> CREATE PUBLICATION pub1 ;
CREATE PUBLICATION
postgres=> ALTER PUBLICATION pub1 ADD TABLE schema1.data1 ;
ALTER PUBLICATION
```

SUBSCRIPTION オブジェクトを作成します。SUBSCRIPTION オブジェクトを作成すると PUBLICATION 側インスタンスに同じ名前のレプリケーション・スロットが作成されます。SUBSCRIPTION オブジェクトの作成には SUPERUSER 権限が必要です。

例 39 SUBSCRIPTION オブジェクト作成（スレーブ）

```
postgres=# CREATE SUBSCRIPTION sub1 CONNECTION 'host=master1 port=5432
user=postgres dbname=postgres' PUBLICATION pub1 ;
NOTICE:  synchronized table states
NOTICE:  created replication slot "sub1" on publisher
CREATE SUBSCRIPTION
```

3.3.4 衝突と不整合

PUBLICATION 側、SUBSCRIPTION 側共、レプリケーション対象テーブルに対する更新処理を行うことができます。このため PUBLICATION から送信された WAL を SUBSCRIPTION 側で適用できない場合が発生する可能性があります。データの衝突等の問題が発生した場合、subscription worker プロセスは停止し、5 秒間隔で再起動が行われます。以下の例は主キー違反が発生した場合に出力されるログです（SUBSCRIPTION 側）。PUBLICATION 側から転送されたデータに対して、SUBSCRIPTION 側テーブルに設定された制約（PRIMARY KEY, UNIQUE, CHECK）はチェックが行われます。

例 40 SUBSCRIPTION 側で主キー違反を検知したログ

```
ERROR:  duplicate key value violates unique constraint "data1_pkey"
DETAIL:  Key (c1)=(14) already exists.
LOG:  worker process: logical replication worker for subscription 16399 (PID
3626) exited with exit code 1
```

例 41 ワーカーの再起動ログ

```
LOG:  starting logical replication worker for subscription "sub1"
LOG:  logical replication sync for subscription sub1, table data1 started
LOG:  logical replication synchronization worker finished processing
```

PUBLICATION 側では、wal sender プロセスがセッションの切断を検知して以下のログが出力されます。

例 42 セッション切断のログ

```
LOG:  unexpected EOF on client connection with an open transaction
```

□ 複数テーブルを単一トランザクションで更新した場合

トランザクション内で複数のテーブルを更新した場合、SUBSCRIPTION 側でもトランザクション単位で更新が行われます。このため SUBSCRIPTION 側でもトランザクションの一貫性は維持されます。衝突を解決するためには SUBSCRIPTION 側で問題となるタプルを更新します。pg_replication_origin_status カタログの参照と、pg_replication_origin_advance 関数を使用した衝突解決方法もありますが、テストしていません。また SUBSCRIPTION 側でテーブルがロックされている場合（LOCK TABLE 文）はレプリケーションの動作も停止します。

□ DELETE / UPDATE 対象が存在しない場合

PUBLICATION 側で UPDATE 文または DELETE 文が実行され、SUBSCRIPTION 側に対象レコードが存在しなかった場合、エラーは発生しません。

表 7 不整合発生時の動作

マスター操作	不整合	動作
INSERT	スレーブで制約違反	レプリケーション停止
	列定義が異なる（互換性あり）	処理継続／ログ出力無し
	列定義が異なる（互換性なし）	レプリケーション停止
UPDATE	スレーブに対象レコードなし	処理継続／ログ出力無し
	スレーブで制約違反	レプリケーション停止
DELETE	スレーブに対象レコードなし	処理継続／ログ出力無し
TRUNCATE	伝播しない	処理継続／ログ出力無し
ALTER TABLE	伝播しない	処理継続／ログ出力無し

3.3.5 制約

Logical Replication には以下の制約があります。

□ CREATE PUBLICATION 文実行権限

CREATE PUBLICATION FOR ALL TABLES 文の実行には SUPERUSER 権限が必要です。個別のテーブルに対応する PUBLICATION オブジェクトの作成は一般ユーザーでも実行できます。

□ 初期データ

既にデータが格納されているテーブルをレプリケーション対象とした場合、デフォルトでは既存データは SUBSCRIPTION 側に転送されます。その際に SUBSCRIPTION 側の既存データは削除されません。初期データの転送は、一時レプリケーション・スロットを使って非同期に行われます。CREATE SUBSCRIPTION 文は初期データの転送完了を待たずに終了します。

□ 主キーまたは一意キー

レプリケーション対象のテーブルで UPDATE 文や DELETE 文を伝播するには主キー（PRIMARY KEY）、または一意キー（UNIQUE）と NOT NULL 制約が必要です。また一意キーを設定したテーブルに対して UPDATE 文や DELETE 文をレプリケーションするには PUBLICATION 側のテーブルで ALTER TABLE REPLICA IDENTITY FULL 文または ALTER TABLE REPLICA IDENTITY USING INDEX 文を、SUBSCRIPTION 側のテーブルで ALTER TABLE REPLICA IDENTITY USING INDEX 文を実行する必要があります。

□ DDL 文

ALTER TABLE 文や TRUNCATE 文は SUBSCRIPTION 側に伝播しません。DDL 文はどちら側のテーブルに対しても実行できます。

□ 文字コード

文字コードが異なるデータベース間でもレプリケーションは行うことができます。文字コードは自動的に変換されます。

□ 監査

SUBSCRIPTION によるデータ更新処理は、パラメーターlog_statement を all に設定しても記録されません。

□ パーティション・テーブルとの組み合わせ

パーティション親テーブルは PUBLICATION に追加できません。子テーブルを PUBLICATION に追加することでレプリケーションが行われます。

例 43 パーティション・テーブルとレプリケーション

```
postgres=> ALTER PUBLICATION pub1 ADD TABLE range1 ;
ERROR:  "range1" is a partitioned table
DETAIL:  Adding partitioned tables to publications is not supported.
HINT:  You can add the table partitions individually.
```

□ インスタンス内レプリケーション

CREATE SUBSCRIPTION 文実行時、CONNECTION 句に自インスタンス（別データベースでも）を指定すると CREATE SUBSCRIPTION 文がハングします。あらかじめレプリケーション・スロットを作成し、CREATE SUBSCRIPTION 文に WITH（create_slot = false）句を指定することでインスタンス内レプリケーション環境を構築できます。

□ 相互レプリケーション

PUBLICATION と SUBSCRIPTION により相互に更新するテーブル構成（マルチマスター・レプリケーション）は作成できません。CREATE PUBLICATION 文、CREATE SUBSCRIPTION 文の実行は成功しますが、スレーブ側に適用された WAL がマスター側に戻ってしまうため WAL の適用エラーが発生します。

□ トリガーの実行

Logical Replication による更新処理では SUBSCRIPTION 側のテーブルのトリガーは実行されません。

3.4 パラレル・クエリーの拡張

PostgreSQL 10 ではパラレル・クエリーを利用できる範囲が拡大しました。

3.4.1 PREPARE / EXECUTE

PREPARE 文と EXECUTE 文による検索処理でもパラレル・クエリーが実行できるようになりました。PostgreSQL 9.6 ではパラレル処理は行われませんでした。

例 44 PREPARE 文と EXECUTE 文によるパラレル・クエリー

```
postgres=> EXPLAIN SELECT COUNT(*) FROM large1 ;
                                     QUERY PLAN
------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=11614.55..11614.56 rows=1 width=8)
   ->  Gather  (cost=11614.33..11614.54 rows=2 width=8)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=10614.33..10614.34 rows=1 width=8)
               ->  Parallel Seq Scan on large1  (cost=0.00..9572.67 rows=416667
width=0)
(5 rows)

postgres=> PREPARE p1 AS SELECT COUNT(*) FROM large1 ;
PREPARE
postgres=> EXPLAIN EXECUTE p1 ;
                                     QUERY PLAN
------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=11614.55..11614.56 rows=1 width=8)
   ->  Gather  (cost=11614.33..11614.54 rows=2 width=8)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=10614.33..10614.34 rows=1 width=8)
               ->  Parallel Seq Scan on large1  (cost=0.00..9572.67 rows=416667
width=0)
(5 rows)
```

3.4.2 Parallel Index Scan

パラレル・クエリーが Index Scan および Index Only Scan にも適用されるようになりました。

例 45 Parallel Index Scan

```
postgres=> EXPLAIN SELECT * FROM large1 WHERE c1 BETWEEN 10000 AND 20000000 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Gather  (cost=0.43..369912.83 rows=7917410 width=12)
   Workers Planned: 2
   ->  Parallel Index Scan using idx1_large1 on large1
               (cost=0.43..369912.83 rows=3298921 width=12)
         Index Cond: ((c1 >= '10000'::numeric) AND (c1 <= '20000000'::numeric))
(4 rows)
```

例 46 Parallel Index Only Scan

```
postgres=> EXPLAIN SELECT COUNT(c1) FROM large1 WHERE c1 BETWEEN 1000 AND
10000000 ;
                            QUERY PLAN
-------------------------------------------------------------------------------
Finalize Aggregate  (cost=316802.38..316802.39 rows=1 width=8)
   ->  Gather  (cost=316802.17..316802.38 rows=2 width=8)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=315802.17..315802.18 rows=1 width=8)
               ->  Parallel Index Only Scan using idx1_large1 on
                     large1  (cost=0.43..305386.50 rows=4166267 width=6)
                   Index  Cond:  ((c1  >=  '1000'::numeric)  AND  (c1  <=
'10000000'::numeric))
(6 rows)
```

3.4.3 SubPlan

SubPlan が記述された SELECT 文でもパラレル・クエリーが利用できるようになりました。

例 47 SubPlan とパラレル・クエリー

```
postgres=> EXPLAIN SELECT * FROM large1 l1 WHERE l1.c1 NOT IN (SELECT l2.c1
        FROM large2 l2 WHERE l2.c1 in (1000,2000,3000)) ;
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 Seq Scan on large1 l1  (cost=23269.95..59080.95 rows=1000000 width=11)
   Filter: (NOT (hashed SubPlan 1))
   SubPlan 1
     ->  Gather  (cost=1000.00..23269.93 rows=6 width=6)
           Workers Planned: 2
           ->  Parallel Seq Scan on large2 l2  (cost=0.00..22269.33 rows=2 width=6)
                 Filter: (c1 = ANY ('{1000,2000,3000}'::numeric[]))
(7 rows)
```

3.4.4 Parallel Merge Join / Gather Merge

Merge Join を選択した場合でもパラレル・クエリーが利用できるようになりました。またパラレル処理で Merge しながら結果を収集する Gather Merge が利用できるようになりました。

例 48 Parallel Merge Join

```
postgres=> EXPLAIN SELECT COUNT(*) FROM large1 INNER JOIN large2 ON large1.c1 =
large2.c1 ;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=447792.07..447792.08 rows=1 width=8)
   ->  Gather  (cost=447791.86..447792.07 rows=2 width=8)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=446791.86..446791.87 rows=1 width=8)
               ->  Merge Join  (cost=407305.94..442727.96 rows=1625561 width=0)
                     Merge Cond: (large2.c1 = large1.c1)
                     ->  Sort  (cost=112492.52..114575.86 rows=833333 width=6)
                           Sort Key: large2.c1
                           ->  Parallel Seq Scan on large2  (cost=0.00..19144.33
rows=833333 width=6)
<< 以下省略 >>
```

3.4.5 Parallel bitmap heap scan

Bitmap Heap Scan がパラレル・クエリーに対応しました。

例 49 Parallel Bitmap Heap Scan

```
postgres=> EXPLAIN SELECT COUNT(c1) FROM large1 WHERE c1 BETWEEN 100000 AND 200000 ;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=18500.74..18500.75 rows=1 width=8)
   ->  Gather  (cost=18500.52..18500.73 rows=2 width=8)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=17500.52..17500.53 rows=1 width=8)
               ->  Parallel Bitmap Heap Scan on large1  …
                     Recheck Cond: ((c1 >= '100000'::numeric) AND (c1 <= '200000'::numeric))
                     ->  Bitmap Index Scan on idx1_large1 …
                           Index Cond: ((c1 >= '100000'::numeric) AND …
(8 rows)
```

3.5 アーキテクチャの変更

3.5.1 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 8　追加されたシステムカタログ一覧

カタログ名	説明
pg_hba_file_rules	pg_hba.conf ファイルの情報
pg_partitioned_table	パーティション化テーブルの情報
pg_publication	Logical Replication の PUBLICATION オブジェクト
pg_publication_rel	Logical Replication 対象テーブル情報
pg_publication_tables	Logical Replication 対象テーブル情報
pg_sequence	シーケンス情報
pg_sequences	シーケンス情報
pg_stat_subscription	Logical Replication のステータス情報
pg_statistic_ext	拡張統計情報
pg_subscription	Logical Replication の SUBSCRIPTION オブジェクト
pg_subscription_rel	Logical Replication の WAL 受信対象テーブル情報

□　pg_hba_file_rules カタログ

pg_hba_file_rules カタログは pg_hba.conf ファイルの内容を参照できます。ファイルを変更するとビューの内容もすぐに反映されます。コメントのみの行は含まれません。

表 9　pg_hba_file_rules カタログ

列名	データ型	説明
line_number	integer	ファイル内の行番号
type	text	local, host 等の接続タイプ
database	text[]	対象データベースまたは all, replication
user_name	text[]	ユーザー名
address	text	TCP/IP アドレス
netmask	text	ネットマスク
auth_method	text	認証方法
options	text[]	オプション
error	text	構文エラー等エラー・メッセージ

□　pg_partitioned_table カタログ

pg_partitioned_table カタログにはパーティション化テーブル（親テーブル）の情報を格納しています。

表 10　pg_partitioned_table カタログ

列名	データ型	説明
partrelid	oid	テーブルの OID
partstrat	char	パーティション化方法（リスト='l'，レンジ='r'）
partnatts	smallint	アタッチされたパーティション数
partattrs	int2vector	パーティション列値の配列
partclass	oidvector	パーティション・キーのデータ型。pg_opclass の oid 列に対応
partcollation	oidvector	パーティション・キー列の Collation 情報
partexprs	pg_node_tree	パーティション化列の情報

□　pg_publication カタログ

pg_publication カタログには Logical Replication で使用する PUBLICATION オブジェクトの情報が格納されます。

表 11　pg_publication カタログ

列名	データ型	説明
pubname	name	PUBLICATION 名
pubowner	oid	PUBLICATION の所有者
puballtables	boolean	全テーブルを対象にするか
pubinsert	boolean	INSERT 文を伝播するか
pubupdate	boolean	UPDATE 文を伝播するか
pubdelete	boolean	DELETE 文を伝播するか

□　pg_publication_rel カタログ

pg_publication_rel カタログには PUBLICATION オブジェクトに含まれるレプリケーション対象テーブルの情報が格納されます。

表 12　pg_publication_rel カタログ

列名	データ型	説明
prpubid	oid	PUBLICATION オブジェクトの oid
prrelid	oid	テーブルの oid

□　pg_publication_tables カタログ

pg_publication_tables カタログには PUBLICATION オブジェクトに含まれるレプリケーション対象テーブルの情報が格納されます。

表 13　pg_publication_tables カタログ

列名	データ型	説明
pubname	name	PUBLICATION オブジェクト名
schemaname	name	テーブル・スキーマ名
tablename	name	レプリケーション対象テーブル名

□　pg_sequence カタログ

SEQUENCE オブジェクトの一覧を出力する pg_sequence カタログが提供されました。このカタログは一般ユーザーでも検索できます。

表 14　pg_sequence カタログ

列名	データ型	説明
seqrelid	oid	オブジェクトの OID
seqtypid	oid	シーケンスのデータタイプ
seqstart	bigint	開始シーケンス値
seqincrement	bigint	増分
seqmax	bigint	最大シーケンス値
seqmin	bigint	最小シーケンス値
seqcache	bigint	キャッシュ個数
seqcycle	boolean	循環するかを示す

□　pg_sequences カタログ

SEQUENCE オブジェクトの一覧を出力する pg_sequences カタログが提供されました。このカタログは一般ユーザーでも検索できますが、last_value 列は、nextval 関数が実行されていない場合や、検索ユーザーが対象 SEQUENCE に対して USAGE または SELECT 権限が無い場合には NULL になります。

表 15　pg_sequences カタログ

列名	データ型	説明
schemaname	name	スキーマ名
sequencename	name	シーケンス・オブジェクト名
sequenceowner	name	シーケンス・オブジェクト所有者
data_type	regtype	シーケンスのデータタイプ
start_value	bigint	開始シーケンス値
min_value	bigint	最小シーケンス値
max_value	bigint	最大シーケンス値
increment_by	bigint	増分
cycle	boolean	サイクリックに使用するか
cache_size	bigint	キャッシュ数
last_value	bigint	最終シーケンス値

例 50 pg_sequences カタログの検索

```
postgres=> ¥x
Expanded display is on.
postgres=> SELECT * FROM pg_sequences ;
-[ RECORD 1 ]-+--------------------
schemaname    | public
sequencename  | seq1
sequenceowner | postgres
data_type     | bigint
start_value   | 1
min_value     | 1
max_value     | 9223372036854775807
increment_by  | 1
cycle         | f
cache_size    | 1
last_value    |
```

pg_sequences カタログが追加されたことで、シーケンスに対する SELECT 文の結果が変更されました。

例 51 シーケンスに対する検索（PostgreSQL 9.6）

```
postgres=> CREATE SEQUENCE seq1 ;
CREATE SEQUENCE
postgres=> SELECT * FROM seq1 ;
-[ RECORD 1 ]-+--------------------
sequence_name | seq1
last_value    | 1
start_value   | 1
increment_by  | 1
max_value     | 9223372036854775807
min_value     | 1
cache_value   | 1
log_cnt       | 0
is_cycled     | f
is_called     | f
```

例 52 シーケンスに対する検索（PostgreSQL 10）

```
postgres=> CREATE SEQUENCE seq1 ;
CREATE SEQUENCE
postgres=> SELECT * FROM seq1 ;
-[ RECORD 1 ]-----
last_value | 1
log_cnt    | 0
is_called  | f
```

□ pg_stat_subscription カタログ

pg_stat_subscription カタログには SUBSCRIPTION オブジェクトが受信した WAL の情報が格納されます。このカタログはレプリケーション処理が動作している間だけデータが参照できます。

表 16 pg_stat_subscription カタログ

列名	データ型	説明
subid	oid	SUBSCRIPTION の OID
subname	name	SUBSCRIPTION 名
pid	integer	logical replication worker のプロセス ID
relid	oid	同期中のテーブルの OID
received_lsn	pg_lsn	受信した LSN
last_msg_send_time	timestamp with time zone	メッセージ送信時刻
last_msg_receipt_time	timestamp with time zone	メッセージ受信時刻
latest_end_lsn	pg_lsn	最終 LSN
latest_end_time	timestamp with time zone	最終タイムスタンプ

□ pg_statistic_ext カタログ

pg_statistic_ext カタログには CREATE STATISTICS 文で作成した拡張統計に関する情報が格納されます。

表 17　pg_statistic_ext カタログ

列名	データ型	説明
stxrelid	oid	統計取得テーブルの OID
stxname	name	拡張統計の名前
stxnamespace	oid	namespace の OID
stxowner	oid	拡張統計の所有者
stxkind	int2vector	拡張統計を取得した列番号の配列
stxenabled	"char"[]	有効になった統計の種類
stxndistinct	pg_ndistinct	シリアライズ化された N-distinct 値
stxdependencies	pg_dependencies	列の依存関係

□　pg_subscription カタログ

pg_subscription カタログには Logical Replication で使用する SUBSCRIPTION オブジェクトの情報が格納されます。このカタログは SUPERUSER 権限を持つユーザーのみ参照できます。

表 18　pg_subscription カタログ

列名	データ型	説明
subdbid	oid	SUBSCRIPTION を構成するデータベースの OID
subname	name	SUBSCRIPTION オブジェクト名
subowner	oid	所有者の OID
subenabled	boolean	オブジェクトが有効か
subconninfo	text	PUBLICATION インスタンスへの接続情報
subslotname	name	レプリケーション・スロット名
subsynccommit	text	同期コミット設定
subpublications	text[]	PUBLICATION 名の配列

□　pg_subscription_rel カタログ

pg_subscription_rel カタログには Logical Replication で使用する SUBSCRIPTION オブジェクトが対象とするテーブルの情報が格納されます。

表 19　pg_subscription_rel カタログ

列名	データ型	説明
srsubid	oid	SUBSCRIPTION オブジェクトの OID
srrelid	oid	対象テーブルの OID
srsubstate	"char"	ステータス i=初期化中, d=データ転送中, s=同期中, r=通常
srsublsn	pg_lsn	srsubstate 列が s または r 状態の最終 LSN

3.5.2 カタログの変更

以下のカタログが変更されました。

表 20　列が追加されたシステムカタログ一覧

カタログ名	追加列名	データ型	説明
pg_class	relispartition	boolean	パーティション親テーブル
	relpartbound	pg_node_tree	パーティション分割情報
pg_replication_slots	temporary	boolean	一時スロットを示す
pg_policy	polpermissive	boolean	PERMISSIVE モードか
pg_policies	permissive	text	PERMISSIVE モードか
pg_stat_replication	write_lag	interval	書き込みラグ
	flush_lag	interval	フラッシュ・ラグ
	replay_lag	interval	リプレイ・ラグ
pg_collation	collprovider	char	プロバイダー情報
	collversion	text	バージョン情報
pg_stat_activity	backend_type	text	プロセスのタイプ
pg_attribute	attidentity	char	GENERATED 列を示す

□　pg_stat_activity カタログ

pg_stat_activity カタログには postmaster プロセスを除くすべてのバックエンド・プロセスの状態が出力されるようになりました。バックエンド･プロセスの種類が backend_type 列で確認できます。

例 53 pg_stat_activity カタログの検索

```
postgres=# SELECT pid,wait_event, backend_type FROM pg_stat_activity ;
  pid  |     wait_event      |    backend_type
-------+---------------------+---------------------
 12251 | AutoVacuumMain      | autovacuum launcher
 12253 | LogicalLauncherMain | background worker
 12269 |                     | client backend
 12249 | BgWriterHibernate   | background writer
 12248 | CheckpointerMain    | checkpointer
 12250 | WalWriterMain       | walwriter
(6 rows)
```

3.5.3 libpq ライブラリの拡張

PostgreSQL Client ライブラリである libpq に以下の拡張が追加されました。

□ マルチ・インスタンス指定

JDBC Driver ではすでに対応されている接続インスタンスを複数記述する方法が libpq ライブラリにも実装されました。下記のようにホスト名、ポート番号をカンマ区切りで複数記述することができます。

構文 7 マルチ・インスタンス接続

```
host=host1,host2
host=host1,host2 port=port1,port2
postgresql://host1,host2/
postgresql://host1:port2,host2:port2/
```

環境変数 PGHOST や PGPORT にカンマ（,）区切りで複数の値を指定できます。これに伴い psql や pg_basebackup コマンドの--host パラメーターや--port パラメーターに複数の値を指定できるようになりました。

□ target_session_attrs 属性の追加

新しい接続属性として target_session_attrs が追加されました。このパラメーターには接続先のインスタンスがホット・スタンバイでも良い場合「any」と書き込みもできるインスタンスに限る場合「read-write」を指定できます。同様の指定が環境変数 PGTARGETSESSIONATTRS に指定できます。内部では SHOW transaction_read_only

文を使って接続先を判断しているようです。

□ passfile 属性の追加

新しい接続属性として passfile が追加されました。従来は環境変数 PGPASSFILE 等で指定していました。

3.5.4 XLOG から WAL へ変更

関数、ディレクトリ名、ユーティリティで使用された XLOG という名称は WAL に統一されました。また pg_clog ディレクトリは pg_xact ディレクトリに変更されました。パラメーターlog_directory のデフォルト値が変更された影響により、ログ・ファイルの出力ディレクトリ名のデフォルトが変更されました。WAL の位置を示す location という名称は lsn に変更されました。

表 21　変更された名前

カテゴリー	変更前	変更後
ディレクトリ	pg_xlog	pg_wal
	pg_clog	pg_xact
	pg_log	log
ユーティリティ	pg_receivexlog	pg_receivewal
	pg_resetxlog	pg_resetwal
	pg_xlogdump	pg_waldump
	pg_basebackup --xlog-method	pg_basebackup --wal-method
	pg_basebackup --xlogdir	pg_basebackup --waldir
	initdb --xlogdir	initdb --waldir
関数	pg_xlog_location_diff	pg_wal_lsn_diff
	pg_switch_xlog	pg_switch_wal
	pg_current_xlog_*	pg_current_wal_*
	pg_xlogfile*	pg_walfile*
	pg_is_xlog_replay_replay_paused	pg_is_wal_replay_replay_paused
	pg_last_xlog_*	pg_last_wal_*
	pg_*location*	pg_*lsn*
カタログ	pg_stat_replication カタログ - sent_location 列 - write_location 列 - flush_location 列 - replay_location 列	pg_stat_replication カタログ - sent_lsn 列 - write_lsn 列 - flush_lsn 列 - replay_lsn 列

同時にエラー・メッセージに含まれる文字列 XLOG も WAL に変更されました。以下のようなメッセージに変更されています。

- Failed while allocating a WAL reading processor.
- could not read two-phase state from WAL at …
- expected two-phase state data is not present in WAL at …
- Failed while allocating a WAL reading processor.
- WAL redo at %X/%X for %s
- Forces a switch to the next WAL file if a new file has not been started within N seconds.

パラメーターarchive_timeout の説明文が以下のように変更されました。

- Forces a switch to the next WAL file if a new file has not been started within N seconds.

3.5.5 一時レプリケーション・スロット

ストリーミング・レプリケーション環境の構築や pg_basebackup コマンドにはレプリケーション・スロットを利用することができます。PostgreSQL 10 では一時レプリケーション・スロット（TEMPORARY REPLICATION SLOT）を作成できるようになりました。一時レプリケーション・スロットはセッション終了により自動的に削除されることを除けば通常のレプリケーション・スロットと同じです。作成には pg_create_physical_replication_slot 関数または pg_create_logical_replication_slot 関数の第 3 パラメーターに true を指定します。これに伴い、pg_replication_slots カタログには temporary 列が追加されました。

例 54 一時レプリケーション・スロットの作成

```
postgres=# SELECT pg_create_physical_replication_slot('temp1', true, true) ;
 pg_create_physical_replication_slot
-------------------------------------
 (temp1,0/30000370)
(1 row)

postgres=# SELECT slot_name, temporary FROM pg_replication_slots ;
 slot_name | temporary
-----------+-----------
 temp1     | t
(1 row)
```

3.5.6 インスタンス起動ログ

インスタンス起動ログにリッスン・アドレスとポート番号が出力されるようになりました。

例 55 インスタンス起動ログ（一部省略）

```
$ pg_ctl -D data start
waiting for server to start....
LOG:  listening on IPv4 address "0.0.0.0", port 5432
LOG:  listening on IPv6 address "::", port 5432
LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
LOG:  redirecting log output to logging collector process
HINT:  Future log output will appear in directory "log".
 done
server started
```

3.5.7 ハッシュ・インデックスの WAL

従来のバージョンのハッシュ・インデックスは、更新時に WAL の出力を行いませんでした。PostgreSQL 10 では WAL を出力するようになったため、ストリーミング・レプリケーション環境でも使用できるようになりました。CREATE INDEX USING HASH 文で出力されていた警告は出力されなくなりました。

例 56 ハッシュ・インデックスの作成（PostgreSQL 10）

```
postgres=> CREATE INDEX idx1_hash1 ON hash1 USING hash (c1) ;
CREATE INDEX
```

例 57 ハッシュ・インデックスの作成（PostgreSQL 9.6）

```
postgres=> CREATE INDEX idx1_hash1 ON hash1 USING hash (c1) ;
WARNING:  hash indexes are not WAL-logged and their use is discouraged
CREATE INDEX
```

3.5.8 ロールの追加

以下のロールが追加されました。いずれも login 権限はありません。

表 22　追加ロール

ロール	用途
pg_read_all_settings	全設定パラメーターを参照できます
pg_read_all_stats	すべての pg_stat_*ビューを参照できます
pg_stat_scan_tables	AccessShareLock ロックを取るモニタリング関数を実行できます
pg_monitor	上記 3 ロールすべての権限を持ちます

以下の Contrib モジュールを登録すると、上記のロールに自動的に関数の実行権限が追加されます。

- pg_buffercache
- pg_freespacemap
- pg_stat_statements
- pg_visibility
- pgstattuple

3.5.9 Custom Scan Callback

パラレル・クエリーの終了時に呼ばれる新たなコールバックが追加されました。マニュアル「58.3. Executing Custom Scans」に以下のように説明されています。

例 58 Custom Scan Callback

```
Initialize a parallel worker's custom state based on the shared state set up in
the leader by InitializeDSMCustomScan. This callback is optional, and needs only
be supplied if this custom path supports parallel execution.

void (*ShutdownCustomScan) (CustomScanState *node);
```

3.5.10 WAL ファイルのサイズ

configure コマンドの--with-wal-segsize オプションで決定する WAL ファイル・サイズの選択肢が増えました。従来の 1～64 に加えて、128, 256, 512, 1024 が使用できるようになりました。

3.5.11 ICU

ロケール機能に ICU を利用できるようになりました。configure コマンド実行時に--with-icu を指定します。Linux 環境でビルドを行う場合は libicu パッケージと libicu-devel パッケージのインストールが必要です。

3.5.12 EUI-64 データ型

EUI-64 アドレスを示すデータ型 macaddr8 が利用できるようになりました。

3.5.13 Unique Join

テーブルの結合時に一意インデックスを使った結合を行う実行計画を作成することができます。EXPLAIN VERBOSE 文で確認する実行計画上は「Inner Unique: true」と表示されます。

例 59 Inner Unique Join

```
postgres=> CREATE TABLE unique1(c1 INTEGER PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE TABLE unique2(c1 INTEGER PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
...
postgres=> EXPLAIN VERBOSE SELECT * FROM unique1 u1 INNER JOIN unique2 u2 ON u1.c1 = u2.c1 ;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=280.00..561.24 rows=10000 width=18)
   Output: u1.c1, u1.c2, u2.c1, u2.c2
   Inner Unique: true
   Hash Cond: (u1.c1 = u2.c1)
   ->  Seq Scan on public.unique1 u1  (cost=0.00..155.00 rows=10000 width=9)
         Output: u1.c1, u1.c2
   ->  Hash  (cost=155.00..155.00 rows=10000 width=9)
         Output: u2.c1, u2.c2
         ->  Seq Scan on public.unique2 u2  (cost=0.00..155.00 rows=10000 width=9)
               Output: u2.c1, u2.c2
(10 rows)
```

3.5.14 共有メモリーのアドレス

EXEC_BACKEND マクロを定義されてインストールされた場合、環境変数 PG_SHMEM_ADDR が利用できます。キャッシュの一部として使用される System V 共有メモリーの先頭アドレスを指定します。内部的には strtoul 関数で数値化されて、shmat システム・コールの第 2 パラメーターとして使用されます。

3.6 モニタリング

3.6.1 待機イベントのモニタリング

pg_stat_activity カタログの wait_event_type 列と wait_event 列に出力される待機イベントが追加されました。PostgreSQL 9.6 で wait_event_type 列に出力されていた LWLockNamed と LWLockTranche は LWLock に変更されました。

表 23　wait_event_type 列に出力される値

wait_event_type 列	内容	変更点
LWLock	軽量ロック待ち	名前変更
Lock	ロック待ち	
BufferPin	バッファ待ち	
Activity	バックグラウンド・プロセスの処理受付待ち状態	追加
Client	クライアントが処理を待っている状態	追加
Extension	バックグラウンド・ワーカーの待機	追加
IPC	他のプロセスからの処理を待っている状態	追加
Timeout	タイムアウト待ち	追加
IO	I/O 待ち	追加

3.6.2 EXPLAIN SUMMARY 文

EXPLAIN 文に、実行計画生成時間のみを出力する SUMMARY 句が追加されました。

例 60 EXPLAIN SUMMARY

```
postgres=> EXPLAIN (SUMMARY) SELECT * FROM data1 ;
                           QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=11)
 Planning time: 0.072 ms
(2 rows)
```

3.6.3 VACUUM VERBOSE 文

VACUUM VERBOSE 文の出力に oldest xmin と frozen pages が出力されるようになりました。

例 61 VACUUM VERBOSE 文の実行

```
postgres=> VACUUM VERBOSE data1 ;
NFO:  vacuuming "public.data1"
<< 途中省略 >>
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 587
There were 0 unused item pointers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
<< 以下省略 >>
```

3.7 Quorum-based 同期レプリケーション

PostgreSQL 9.5 以前では同期レプリケーションができるインスタンスは一つにかぎられていました。PostgreSQL 9.6 では複数インスタンスに対して同期レプリケーションを行うことができるようになりました。

PostgreSQL 10 では同期レプリケーションを行うインスタンスを任意に選択する Quorum-based 同期レプリケーションが実装されました。同期レプリケーションの設定は従来通り、パラメーターsynchronous_standby_names で行います。

構文 8 PostgreSQL 9.5 まで

```
synchronous_standby_names = application_name, application_name, …
```

構文 9 PostgreSQL 9.6

```
synchronous_standby_names = num_sync (application_name, application_name, …)
```

構文 10 PostgreSQL 10

```
synchronous_standby_names = FIRST | ANY num_sync (application_name,
application_name, …)
```

FIRST を指定するか省略すると PostgreSQL 9.6 と同じ動作になります。パラメーターapplication_name に記述された順番で優先順位を決め、num_sync で指定された個数のインスタンスに対して同期レプリケーションを行います。

ANY を指定すると application_name に指定されたインスタンスの順番には依存せず、num_sync で指定されたスレーブ・インスタンスに WAL が転送された場合に同期レプリケーションの完了を決定します。パラメーターsynchronous_standby_names に ANY が指定された場合、pg_stat_replication カタログの sync_state 列は quorum が出力されます。

例 62 Quorum-based 同期レプリケーション

```
postgres=> SHOW synchronous_standby_names ;
         synchronous_standby_names
-----------------------------------------
 any 2 (standby1, standby2, standby3)
(1 row)

postgres=> SELECT application_name,sync_state, sync_priority
                FROM pg_stat_replication ;
 application_name | sync_state | sync_priority
------------------+------------+---------------
 standby1         | quorum     |             1
 standby2         | quorum     |             1
 standby3         | quorum     |             1
(3 rows)
```

3.8 Row Level Security の拡張

3.8.1 概要

テーブルに対して複数のポリシーが設定された場合、PostgreSQL 9.6 以前では OR 条件でポリシーが判断されました。PostgreSQL 10 ではポリシーを AND 条件で指定することができます。ポリシーを作成する CREATE POLICY 文に AS PERMISSIVE 句と AS RESTRICTIVE 句が指定できるようになりました。AS PERMISSIVE 句を指定すると制限がゆるくなり（OR）、AS RESTRICTIVE を指定すると制限が厳しくなります（AND）。指定を省略した場合は旧バージョンと同様に OR 条件になります。これに伴い、pg_policy カタログと pg_policies カタログに条件指定を示す列が追加されました。

表 24　追加された列（pg_policy）

列名	データ型	説明
polpermissive	boolean	POLICY のモード（true の場合は PERMISSIVE）

表 25　追加された列（pg_policies）

列名	データ型	説明
permissive	text	POLICY のモード（PERMISSIVE または RESTRICTIVE）

構文 11 CREATE POLICY 文

```
CREATE POLICY policy_name ON table_name
 [ AS { PERMISSIVE | RESTRICTIVE } ]
 [ FOR { ALL | SELECT | INSERT | UPDATE | DELETE } ]
 [ TO { role_name | PUBLIC | CURRENT_USER | SESSION_USER } [, ...] ]
 [ USING ( using_expression ) ]
```

3.8.2 複数 POLICY 設定の検証

テーブルに対して複数の POLICY を設定し、効果を検証しました。テーブル poltbl1 に対して、PERMISSIVE モードの POLICY pol1、RESTRICTIVE モードの POLICY pol2, pol3 を用意し、組み合わせて検証しました。

例 63 テーブルと POLICY の作成（PERMISSIVE + RESTRICTIVE）

```
postgres=> CREATE TABLE poltbl1 (c1 NUMERIC, c2 VARCHAR(10), uname VARCHAR(10)) ;
CREATE TABLE
postgres=> ALTER TABLE poltbl1 ENABLE ROW LEVEL SECURITY ;
ALTER TABLE
postgres=> CREATE POLICY pol1 ON poltbl1 FOR ALL USING (uname = current_user) ;
CREATE POLICY
postgres=> CREATE POLICY pol2 ON poltbl1 AS RESTRICTIVE FOR ALL USING (c2 = 'data') ;
CREATE POLICY
postgres=> SELECT polname, polpermissive FROM pg_policy ;
 polname | polpermissive
---------+---------------
 pol1    | t
 pol2    | f
(2 rows)
postgres=> SELECT tablename, policyname, permissive FROM pg_policies ;
 tablename | policyname | permissive
-----------+------------+-------------
 poltbl1   | pol1       | PERMISSIVE
 poltbl1   | pol2       | RESTRICTIVE
(2 rows)
postgres=> ¥d poltbl1
                    Table "public.poltbl1"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | numeric               |           |          |
 c2     | character varying(10) |           |          |
 uname  | character varying(10) |           |          |
Policies:
    POLICY "pol1"
      USING (((uname)::name = CURRENT_USER))
    POLICY "pol2" AS RESTRICTIVE
      USING (((c2)::text = 'data'::text))
```

上記の例ではテーブル poltbl1 の設定を見ると POLICY pol2 が RESTRICTIVE であることが表示されています。

例 64 実行計画の確認（PERMISSIVE + RESTRICTIVE）

```
postgres=> EXPLAIN SELECT * FROM poltbl1 ;
                                QUERY PLAN
---------------------------------------------------------------------------
 Seq Scan on poltbl1  (cost=0.00..20.50 rows=1 width=108)
   Filter: (((c2)::text = 'data'::text) AND ((uname)::name = CURRENT_USER))
(2 rows)
```

２つの条件が AND により結合されていることがわかります。次に POLICY pol1 を削除し、RESTRICTIVE モードの POLICY pol3 を適用したテーブルを作成します。

例 65 RESTRITIVE モードの POLICY 作成

```
postgres=> CREATE POLICY pol3 ON poltbl1 AS RESTRICTIVE FOR ALL USING (c1 > 1000) ;
CREATE POLICY
postgres=> ¥d poltbl1
                    Table "public.poltbl1"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | numeric               |           |          |
 c2     | character varying(10) |           |          |
 uname  | character varying(10) |           |          |
Policies:
    POLICY "pol2" AS RESTRICTIVE
      USING (((c2)::text = 'data'::text))
    POLICY "pol3" AS RESTRICTIVE
      USING ((c1 > (1000)::numeric))
```

例 66 実行計画の確認（RESTRICTIVE＋RESTRICTIVE）

```
postgres=> EXPLAIN SELECT * FROM poltbl1 ;
                    QUERY PLAN
-----------------------------------------------
 Result  (cost=0.00..0.00 rows=0 width=108)
   One-Time Filter: false
(2 rows)
```

すべてのポリシーが RESTRICTIVE モードの場合、実行計画は表示されないようです。

3.9 SQL 文の拡張

ここでは SQL 文に関係する新機能を説明しています。

3.9.1 UPDATE 文と ROW 句

UPDATE 文に ROW 句を使用できるようになりました。

例 67 UPDATE 文と ROW 句

```
postgres=> UPDATE pgbench_tellers SET (bid, tbalance) = ROW (2, 1) WHERE tid = 10;
UPDATE 1
```

3.9.2 CREATE STATISTICS 文

CREATE STATISTICS 文により、複数列の相関に関する統計情報を収集できるようになりました。実際に統計値が収集されるタイミングは ANALYZE 文実行時です。

構文 12 CREATE STATISTICS 文

```
CREATE STATISTICS [ IF NOT EXISTS ] stat_name [ ( stat_type [ , … ] ) ]
ON col1, col2 [, ... ] FROM table_name
```

stat_name には拡張統計の名前を指定します。スキーマ名で修飾することもできます。少なくとも 2 つの列の指定が必要です。stat_type には dependencies と ndistinct を指定することができます。省略時には両方指定されたとみなされます。拡張統計の変更を行う場合は ALTER STATISTICS 文を実行します。

構文 13 ALTER STATISTICS 文

```
ALTER STATISTICS stat_name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
ALTER STATISTICS stat_name RENAME TO new_name
ALTER STATISTICS stat_name SET SCHEMA new_schema
```

拡張統計の削除を行う場合は DROP STATISTICS 文を実行します。

構文 14 DROP STATISTICS 文

```
DROP STATISTICS [ IF EXISTS ] name [, ...]
```

例 68 CREATE STATISTICS 文による拡張統計の作成

```
postgres=> CREATE TABLE stat1(c1 NUMERIC, c2 NUMERIC, c3 VARCHAR(10)) ;
CREATE TABLE
postgres=> INSERT INTO stat1 VALUES(generate_series(1, 100000) / 5,
                generate_series(1, 100000) / 10, 'init') ;
INSERT 0 100000
postgres=> CREATE STATISTICS stat1_stat1 ON c1, c2 FROM stat1 ;
CREATE STATISTICS
```

例 69 拡張統計を作成したテーブルの情報確認

```
postgres=> ¥d stat1
                    Table "public.stat1"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | numeric               |           |          |
 c2     | numeric               |           |          |
 c3     | character varying(10) |           |          |
Statistics objects:
    "public"."stat1_stat1" (ndistinct, dependencies) ON c1, c2 FROM stat1
```

作成された拡張統計の情報は pg_statistic_ext カタログで確認することができます。

例 70 拡張統計の確認

```
postgres=> SELECT * FROM pg_statistic_ext ;
-[ RECORD 1 ]---+----------------------------------------
stxrelid        | 16575
stxname         | stat1_stat1
stxnamespace    | 2200
stxowner        | 16454
stxkeys         | 1 2
stxkind         | {d,f}
stxndistinct    | {"1, 2": 19982}
stxdependencies | {"1 => 2": 1.000000, "2 => 1": 0.170467}
```

3.9.3 GENERATED AS IDENTITY 列

CREATE TABLE 文に、列に一意な値を自動的に割り当てる GENERATED AS IDENTITY 制約が追加されました。従来のバージョンで使用できる serial 型とほぼ同様の機能ですが、一部仕様が異なります。serial 型も GENERATED AS IDENTITY 制約も内部的には SEQUENCE オブジェクトを使用しています。GENERATED AS IDENTITY 制約は複数の列に対して追加することができます。

構文 15 CREATE TABLE 文（列定義）

```
column_name type GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY
[ ( sequence_option ) ]
```

データ型（type）には SMALLINT, INT, BIGINT を使用できます。 LIKE 句を使ってテーブルの複製を行った場合、GENERATED 属性は引き継がれません。NOT NULL 制約のみ引き継がれます。

既存の列に GENERATED AS IDENTITY 制約を追加する場合は ALTER TABLE 文を実行します。指定された列には NOT NULL 制約が必要です。

構文 16 ALTER TABLE 文（制約の追加）

```
ALTER TABLE table_name ALTER COLUMN column_name ADD GENERATED { ALWAYS | BY
DEFAUT } AS IDENTITY { ( sequence_option ) }
```

構文 17 ALTER TABLE 文（制約の削除）

```
ALTER TABLE table_name ALTER COLUMN column_name DROP IDENTITY [ IF EXISTS ]
```

構文 18 ALTER TABLE 文（制約の更新）

```
ALTER TABLE table_name ALTER COLUMN column_name { SET GENERATED { ALWAYS | BY
DEFAULT } | SET sequence_option | RESTART [ [ WITH ] restart ] }
```

上記構文で作成した列の情報は、information_schema スキーマの columns テーブルに情報が格納されます。従来 is_identity 列は NO になっており、その他の情報は NULL 値でした。

□ GENERATED ALWAYS

GENERATED ALWAYS を指定した列は INSERT 文実行時に列値をアプリケーションから指定や、UPDATE 文による DEFAULT 値以外の値への更新が禁止されます。

例 71 GENERATED ALWAYS

```
postgres=> CREATE TABLE ident1 (c1 bigint GENERATED ALWAYS AS IDENTITY, c2 VARCHAR(10)) ;
CREATE TABLE
demodb=> ¥d ident1
                                Table "public.ident1"
 Column |         Type          | Collation | Nullable |           Default
--------+-----------------------+-----------+----------+------------------------------
 c1     | bigint                |           | not null | generated always as identity
 c2     | character varying(10) |           |          |

postgres=> INSERT INTO ident1(c1, c2) VALUES (1, 'data1') ;
ERROR:  cannot insert into column "c1"
DETAIL:  Column "c1" is an identity column defined as GENERATED ALWAYS.
HINT:  Use OVERRIDING SYSTEM VALUE to override.
postgres=> INSERT INTO ident1(c2) VALUES ('data1') ;
INSERT 0 1
postgres=> UPDATE ident1 SET c1=2 WHERE c1=1 ;
ERROR:  column "c1" can only be updated to DEFAULT
DETAIL:  Column "c1" is an identity column defined as GENERATED ALWAYS.
postgres=> UPDATE ident1 SET c1=DEFAULT WHERE c1=1 ;
UPDATE 1
```

INSERT 文に OVERRIDING SYSTEM VALUE 句を指定すると、GENERATED 列に任意の値を格納することができます。

例 72 OVERRIDING SYSTEM VALUE 句

```
pgbench=> INSERT INTO ident1 OVERRIDING SYSTEM VALUE VALUES (100, 'data1') ;
INSERT 0 1
```

□ GENERATED BY DEFAULT

GENERATED BY DEFAULT 句を指定すると、自動採番列も更新可能になります。serial 型の列と同じ動作になります。

例 73 GENERATED BY DEFAULT

```
postgres=> CREATE TABLE ident2 (c1 bigint GENERATED BY DEFAULT AS IDENTITY, c2
VARCHAR(10)) ;
CREATE TABLE
postgres=> INSERT INTO ident2 VALUES (1, 'data1') ;
INSERT 0 1
postgres=> INSERT INTO ident2(c2) VALUES ('data2') ;
INSERT 0 1
postgres=> UPDATE ident2 SET c1=2 WHERE c2='data2' ;
UPDATE 1
```

3.9.4 ALTER TYPE 文

ALTER TYPE 文を使って ENUM 型の名前を変更できるようになりました。

構文 19 ALTER TYPE RENAME VALUE 文

```
ALTER TYPE type_name RENAME VALUE existing_val TO replace_val
```

例 74 ALTER TYPE 文による ENUM 型の変更

```
postgres=> CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy') ;
CREATE TYPE
postgres=> ALTER TYPE mood RENAME VALUE 'ok' TO 'good' ;
ALTER TYPE
```

3.9.5 CREATE SEQUENCE 文

CREATE SEQUENCE 文にデータ型を指定できるようになりました。指定できるデータ型は SMALLINT、INTEGER、BIGINT（デフォルト）です。シーケンス値の範囲はデータ型の範囲に限定されます。

構文 20 CREATE SEQUENCE 文

```
CREATE SEQUENCE sequence_name [ AS type ] [ INCREMENT … ]
```

例 75 CREATE SEQUENCE 文に SMALLINT 型を指定

```
postgres=> CREATE SEQUENCE seq1 AS SMALLINT ;
CREATE SEQUENCE
```

ALTER SEQUENCE AS 文でデータ型を変更することもできます。データ型が変更された場合は SEQUENCE の最大値も更新されます。ただし現在のシーケンス値を小さくする変更は認められません。

3.9.6 COPY 文

INSTEAD OF INSERT トリガーが設定されたシンプルなビューに対して COPY 文が実行できるようになりました。

例 76 VIEW に対する COPY 文

```
postgres=> CREATE TABLE instead1(c1 NUMERIC, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE VIEW insteadv1 AS SELECT c1, c2 FROM instead1 ;
CREATE VIEW
postgres=> CREATE OR REPLACE FUNCTION view_insert_row1() RETURNS trigger AS
         $$
         BEGIN
            INSERT INTO instead1 VALUES (new.c1, new.c2);
            RETURN new;
         END;
         $$
         LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=> CREATE TRIGGER insteadv1_insert
         INSTEAD OF INSERT ON insteadv1 FOR EACH ROW
         EXECUTE PROCEDURE view_insert_row1() ;
CREATE TRIGGER
postgres=# COPY insteadv1 FROM '/home/postgres/instead.csv' ;
COPY 2
```

3.9.7 CREATE INDEX 文

BRIN インデックスを作成する CREATE INDEX 文の WITH 句に autosummarize が指定できるようになりました。この指定を行うとページにデータが挿入された場合に前のページに対して要約が行われることを指定します。

例 77 BRIN インデックスの拡張

```
postgres=> CREATE INDEX idx1_brin1 ON brin1 USING brin (c1) WITH
(autosummarize) ;
CREATE INDEX
postgres=> ¥d brin1
                    Table "public.brin1"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | numeric               |           |          |
 c2     | character varying(10) |           |          |
Indexes:
    "idx1_brin1" brin (c1) WITH (autosummarize='true')
```

3.9.8 CREATE TRIGGER 文

CREATE TRIGGER 文に、REFERENCING 句が使用できるようになりました。更新差分を格納するテーブル名を指定できます。この設定は AFTER トリガーにのみ設定できます。

構文 21 CREATE TRIGGER 文

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } …
[ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
[ REFERENCING { { OLD | NEW } TABLE [ AS ] transition_relation_name } [ ... ] ]
[ FOR [ EACH ] { ROW | STATEMENT } ]
…
```

3.9.9 DROP FUNCTION 文

DROP FUNCTION 文に複数の FUNCTION を指定できるようになりました。複数の FUNCTION を指定する場合はカンマ（,）で区切ります。

3.9.10 ALTER DEFAULT PRIVILEGE 文

ALTER DEFAULT PRIVILEGE 文の GRANT 句、REVOKE 句に ON SCHEMAS 句が指定できるようになりました。従来は ON FUNCTIONS、ON SEQUENCES、ON TABLES、ON TYPES のみでした。

3.9.11 CREATE SERVER 文

CREATE SERVER 文、CREATE USER MAPPING 文に IF NOT EXISTS 句が利用できるようになりました。

3.9.12 CREATE USER 文

CREATE USER 文、CREATE ROLE 文、ALTER USER 文に UNENCRYPTED 句が使用できなくなりました。pg_shadow カタログにパスワードが変換されずに格納されることがなくなりました。

例 78 UNENCRYPTED 句

```
postgres=# CREATE USER user1 UNENCRYPTED PASSWORD 'user1' ;
ERROR:  UNENCRYPTED PASSWORD is no longer supported
LINE 1: CREATE USER user1 UNENCRYPTED PASSWORD 'user1' ;
                          ^
HINT:  Remove UNENCRYPTED to store the password in encrypted form instead.
```

3.9.13 関数

以下の関数が追加／拡張されました。

□ JSONB 配列から要素削除

JSONB 配列から要素を削除できるようになりました。

例 79 JSONB 配列からの要素削除

```
postgres=> SELECT '{"a":1 , "b":2, "c":3}'::jsonb - '{a,c}'::text[] ;
 ?column?
----------
 {"b": 2}
(1 row)
```

□ pg_current_logfile

pg_current_logfile 関数は出力中のログ・ファイルのパスを返します。パラメーターlog_directory を含めたパスを取得できます。パラメーターlog_destination が syslog に設定されている場合や、パラメーターlogging_collector が off に設定されている場合は NULL が返ります。この関数の実行には SUPERUSER 権限が必要です。

例 80 pg_current_logfile 関数

```
postgres=# SELECT pg_current_logfile() ;
           pg_current_logfile
----------------------------------------
 log/postgresql-2017-05-20_092939.log
(1 row)
```

□ xmltable

XML データから表形式の出力を得る xmltable 関数が提供されました。この関数を利用するためにはインストール時の configure コマンドのパラメーターに--with-libxml の指定が必要です。また--with-libxml パラメーターを指定してバイナリーをビルドするためには以下のパッケージのインストールが必要です（Red Hat Enterprise Linux 7 の場合）。

- libxml2 (version >= 2.6.23)
- libxml2-devel
- xz-devel

例 81 xmltable 関数

```
postgres=> SELECT xmltable.*
postgres->   FROM xmldata,
postgres->        XMLTABLE('//ROWS/ROW'
postgres(>                 PASSING data
postgres(>                 COLUMNS id int PATH '@id',
postgres(>                         ordinality FOR ORDINALITY,
postgres(>                         "COUNTRY_NAME" text,
postgres(>                         country_id text PATH 'COUNTRY_ID',
postgres(>                         size_sq_km float PATH 'SIZE[@unit = "sq_km"]',
postgres(>                         size_other text PATH
postgres(>                  'concat(SIZE[@unit!="sq_km"], " ", SIZE[@unit!="sq_km"]/@unit)',
postgres(>                   premier_name text PATH 'PREMIER_NAME' DEFAULT 'not specified') ;
 id | ordinality | COUNTRY_NAME | country_id | size_sq_km |  size_other  | premier_name
----+------------+--------------+------------+------------+--------------+--------------
  1 |          1 | Australia    | AU         |            |              | not specified
  5 |          2 | Thailand     | TH         |            |              | Prayuth Chan
  6 |          3 | Singapore    | SG         |        697 |              | not specified
(3 rows)
```

□ regexp_match

パターンマッチを行う regexp_match 関数が追加されました。従来からあった regexp_matches とは異なり text 型の配列を返します。 citext Contrib モジュールにも citext 型に対応した regexp_match 関数が追加されました。

例 82 regexp_match 関数

```
postgres=> ¥dfS regexp_match
                                List of functions
   Schema   |     Name     | Result data type | Argument data types |  Type
------------+--------------+------------------+---------------------+--------
 pg_catalog | regexp_match | text[]           | text, text          | normal
 pg_catalog | regexp_match | text[]           | text, text, text    | normal
(2 rows)
```

□ pg_ls_logdir / pg_ls_waldir

これらの関数はログ・ファイルおよび WAL ファイル一覧の名前、サイズ、更新日時を返します。これらの関数の実行には SUPERUSER 権限が必要です。

例 83 pg_ls_logdir / pg_ls_waldir 関数

```
postgres=# SELECT * FROM pg_ls_logdir() ;
              name                | size |      modification
----------------------------------+------+------------------------
 postgresql-2017-05-20_092939.log |  5220 | 2017-05-20 21:44:21+09
(1 row)

postgres=# SELECT * FROM pg_ls_waldir() ;
           name           |   size   |      modification
--------------------------+----------+------------------------
 000000010000000000000002E | 16777216 | 2017-05-19 22:55:33+09
 (1 row)
```

□ txid_status

トランザクションの状態を確認する txid_status 関数が追加されました。トランザクション ID を指定すると、該当トランザクションの状態を返します。

例 84 txid_status 関数

```
postgres=> BEGIN ;
BEGIN
postgres=> SELECT txid_current() ;
 txid_current
--------------
          578
(1 row)

postgres=> SELECT txid_status(578) ;
 txid_status
-------------
 in progress
(1 row)

postgres=> COMMIT ;
COMMIT
postgres=> SELECT txid_status(578) ;
 txid_status
-------------
 committed
(1 row)

postgres=> SELECT txid_status(1000) ;
ERROR:  transaction ID 1000 is in the future
```

□ JSON / JSONB 型

以下の関数が JSON 型、JSONB 型に対応しました。

- to_tsvector
- ts_headline

□ pg_stop_backup

pg_stop_backup 関数に WAL アーカイブの待機を指定するパラメーター wait_for_archive が追加されました。デフォルト（true）では従来通り、WAL のアーカイブを待機します。

□ pg_import_system_collations

pg_import_system_collations 関数は OS に新しい Collation がインストールされた場合に、PostgreSQL インスタンスに情報をインポートします。この関数の実行には SUPERUSER 権限が必要です。

構文 22 pg_import_system_collations

```
pg_import_system_collations(if_not_exists boolean, schema regnamespace)
```

□ to_date / to_timestamp

to_date 関数と to_timestamp 関数は各フィールドの入力値が厳密にチェックされるようになりました。従来のバージョンでは自動計算されていた値がエラーになります。

例 85 to_date 関数（PostgreSQL 9.6）

```
postgres=> SELECT to_date('2017-04-40', 'YYYY-MM-DD') ;
  to_date
------------
 2017-05-10
(1 row)
```

例 86 to_date 関数（PostgreSQL 10）

```
postgres=> SELECT to_date('2017-04-40', 'YYYY-MM-DD') ;
ERROR:  date/time field value out of range: "2017-04-40"
```

□ make_date 関数

年を指定するパラメーターに負の値（紀元前）が指定できるようになりました。

例 87 make_date 関数（PostgreSQL 9.6）

```
postgres=> SELECT make_date(-2000, 4, 30) ;
ERROR:  date field value out of range: -2000-04-30
```

例 88 make_date 関数（PostgreSQL 10）

```
postgres=> SELECT make_date(-2000, 4, 30) ;
   make_date
---------------
 2000-04-30 BC
(1 row)
```

3.9.14 手続き言語

手続き言語の拡張について説明しています。

□ PL/Python

plan.execute 文と plan.cursor 文が追加されました。

例 89 execute 構文 / cursor 構文

```
# plan.execute
plan = plpy.prepare("SELECT val FROM data1 WHERE key=$1", [ "NUMERIC" ])
result = plan.execute(key)

# plan.cursor
plan = plpy.prepare("SELECT val FROM data1 WHERE key=$1", [ "NUMERIC" ])
rows = plan.cursor([2])
```

□ PL/Tcl

subtransaction 構文によるトランザクションを実行できるようになりました。

例 90 subtransaction 構文

```
CREATE FUNCTION transfer_funds2() RETURNS void AS $$
if [catch {
    subtransaction {
        spi_exec "UPDATE accounts SET balance = balance - 100 WHERE account_name = 'joe'"
        spi_exec "UPDATE accounts SET balance = balance + 100 WHERE account_name = 'mary'"
   }
} errormsg] {
      set result [format "error transferring funds: %s" $errormsg]
} else {
      set result "funds transferred correctly"
}
    set plan  [spi_prepare "INSERT INTO operations (result) VALUES ($1)"]
    spi_execp $plan, [list $result]
$$ LANGUAGE pltclu;
```

初期化プロシージャ名を指定する GUC、pltcl.start_proc と pltclu.start_proc が追加されました。

3.10 パラメーターの変更

PostgreSQL 10 では以下のパラメーターが変更されました。

3.10.1 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 26　追加されたパラメーター

パラメーター	説明（context）	デフォルト値
enable_gathermerge	実行計画 Gather Merge を有効にする（user）	on
max_parallel_workers	パラレル・ワーカー・プロセスの最大値（user）	8
max_sync_workers_per_subscription	SUBSCRIPTION に対する最大同期ワーカー数（sighup）	2
wal_consistency_checking	スタンバイ・インスタンスで WAL の整合性をチェック（superuser）	なし
max_logical_replication_workers	Logical Replication Worker プロセスの最大値（postmaster）	4
max_pred_locks_per_relation	リレーション全体をロックする前に述語ロックできる最大ページ（sighup）	-2
max_pred_locks_per_page	ページ全体をロックする前に述語ロックできる最大レコード（sighup）	2
min_parallel_table_scan_size	Parallel Table Scan が考慮される最小テーブル・サイズ（user）	8MB
min_parallel_index_scan_size	Parallel Index Scan が考慮される最小インデックス・サイズ（user）	512kB

□　max_parallel_workers パラメーター

インスタンス全体で同時実行できる、パラレル・クエリー・ワーカー・プロセスの最大数を指定します。デフォルト値は 8 です。旧バージョンでは max_worker_processes パラメーターが上限でした。この値を 0 に指定すると、パラレル・クエリーが無効化されます。

□ max_logical_replication_workers パラメーター

SUBSCRIPTION 単位で起動する Logical Replication Worker プロセスの最大値を指定します。このパラメーターが不足した場合でも CREATE SUBSCRIPTION 文の実行はエラーになりません。レプリケーション開始時に以下のログが定期的に出力されます。

例 91　max_logical_replication_workers パラメーター不足

```
WARNING:  out of logical replication worker slots
HINT:  You might need to increase max_logical_replication_workers.
```

□ wal_consistency_checking パラメーター

このパラメーターはレプリケーション環境で WAL 再実行プログラムのバグチェックに使用します。パラメーターにはチェック対象のオブジェクト種別をカンマ（,）区切りで指定します。次の値が使用できます all、hash, heap, heap2, btree, gin, gist, sequence, spgist, brin, generic。

□ max_pred_locks_per_page パラメーター

ページロックに遷移するための最大タプル・ロック数を指定します。

□ max_pred_locks_per_relation パラメーター

リレーション・ロックに遷移するための最大ページ・ロック数を指定します。

3.10.2 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 27　変更されたパラメーター（pg_settings カタログ）

パラメーター	変更内容
ssl	context 列の値が sighup に変更されました
ssl_ca_file	context 列の値が sighup に変更されました
ssl_cert_file	context 列の値が sighup に変更されました
ssl_ciphers	context 列の値が sighup に変更されました
ssl_crl_file	context 列の値が sighup に変更されました
ssl_ecdh_curve	context 列の値が sighup に変更されました
ssl_key_file	context 列の値が sighup に変更されました
ssl_prefer_server_ciphers	context 列の値が sighup に変更されました
bgwriter_lru_maxpages	max_val 列の値が INT_MAX / 2 に変更されました
archive_timeout	short_desc 列の値が変更されました
server_version_num	max_val/min_val 列の値が 100000 に変更されました
password_encryption	vertype が enum に変更されました。指定できる値は md5, scram-sha-256 です。"on"は"md5"の別名です。
max_wal_size	unit 列の値が 1MB に変更されました
min_wal_size	unit 列の値が 1MB に変更されました

3.10.3 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 28　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 9.6	PostgreSQL 10
hot_standby	off	on
log_line_prefix	''	%m [%p]
max_parallel_workers_per_gather	0	2
max_replication_slots	0	10
max_wal_senders	0	10
password_encryption	on	md5
server_version	9.6.3	10beta1
server_version_num	90603	100000
wal_level	minimal	replica
log_directory	pg_log	log

□ log_line_prefix パラメーター

パラメーターのデフォルト値が変更されました。

例 92 パラメーターlog_line_prefix

```
postgres=# SHOW log_line_prefix ;
 log_line_prefix
-----------------
 %m [%p]
(1 row)

$ tail -1 data/log/postgresql-2017-05-20_093448.log
2017-05-20 09:34:48.617 JST [12187] LOG:  autovacuum launcher started
```

3.10.4 廃止されたパラメーター

以下のパラメーターは廃止されました。

表 29 廃止されたパラメーター

パラメーター	代替値
min_parallel_relation_size	min_parallel_table_scan_size に変更
sql_inheritance	なし（on と同じ動作）

3.10.5 認証メソッドの新機能

pg_hba.conf ファイルには以下の変更が加えられました。

□ RADIUS サーバーの指定

RADIUS 認証を行う際に必要な RADIUS サーバーの指定が radiusserver から radiusservers に変更されました。コンマ（,）で区切った複数のサーバーを指定できるようになりました。

□ SCRAM 認証の追加

pg_hba.conf の認証メソッドに scram-sha-256 を指定することができるようになりました。これは RFC 5802 および 7677 で規定された SCRAM-SHA-256 の実装です。パラメーターpassword_encryption にも scram-sha-256 が指定できるようになりました。

3.10.6 認証設定のデフォルト値

pg_hba.conf ファイルに含まれるレプリケーション関連のデフォルト値が変更されました。デフォルトでローカル接続が trust 設定になります。

例 93　pg_hba.conf ファイルのデフォルト設定

```
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust
```

3.10.7 その他パラメーター変更

recovery.conf ファイルに Point In Time Recovery 関連のパラメータ recovery_target_lsn が追加されました。このパラメーターにはリカバリー完了 LSN を指定します。

3.11 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強化点を説明します。

3.11.1 psql

psql コマンドには以下の機能が追加されました。

□ ¥d コマンド

¥d コマンドでテーブル情報を出力する際のフォーマットが変更されました。従来は Modifier として一括表示されていた列の情報が Collation, Nullable, Default に分割されるようになりました。

例 94 テーブル情報（PostgreSQL 9.6）

```
postgres=> ¥d data1
            Table "public.data1"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 c1     | numeric               | default 1
 c2     | character varying(10) | not null
```

例 95 テーブル情報（PostgreSQL 10）

```
postgres=> ¥d data1
                      Table "public.data1"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | numeric               |           |          | 1
 c2     | character varying(10) |           | not null |
```

□ ¥timing コマンドの出力追加

¥timing コマンドは SQL 文の実行時間の出力を制御します。新バージョンでは実行時間出力時によりわかりやすい時間表示が追加されます。実行時間が 1 秒未満の場合は旧バージョンと同じフォーマットで出力されます。

例 96 ¥timing コマンドの出力追加

```
postgres=> ¥timing
Timing is on.
postgres=> INSERT INTO data1 values (generate_series(1, 10000000)) ;
INSERT 0 10000000
Time: 61086.012 ms (01:01.086)
```

□ ¥gx コマンド

¥gx コマンドは直前に実行された SQL 文を、拡張フォーマットで再実行します。

例 97 ¥gx コマンド

```
postgres=> SELECT * FROM data1 ;
 c1 |  c2
----+------
  1 | data
(1 row)

postgres=> ¥gx
-[ RECORD 1 ]
c1 | 1
c2 | data
```

□ ¥set コマンド

¥set コマンドで表示されるパラメーターが増えました。

例 98 ¥set コマンド

```
postgres=> ¥set
AUTOCOMMIT = 'on'
COMP_KEYWORD_CASE = 'preserve-upper'
DBNAME = 'demodb'
ECHO = 'none'
<<以下省略>>
```

□ ¥if, ¥elif, ¥else, ¥endif コマンド

psql コマンド内で条件分岐を行うことができるようになりました。¥if、¥else、¥endif の間で条件分岐を行い、その間のコマンドをブロック化できます。¥if コマンドおよび¥elif コマンドは True または False を判断できるパラメーターを指定します。条件文のネストを行うこともできます。

例 99 ¥if コマンド

```
SELECT
    EXISTS(SELECT 1 FROM customer WHERE customer_id=123) AS is_customer,
    EXISTS(SELECT 1 FROM employee WHERE employee_id=456) AS is_employee ;
¥gset
¥if :is_customer
     SELECT * FROM customer WHERE customer_id = 123 ;
¥elif :is_employee
     SELECT * FROM employee WHERE employee_id = 456 ;
¥endif
```

3.11.2 pg_ctl

pg_ctl コマンドには下記の機能が追加されました。

□ プロモート時の待機

pg_ctl コマンドはスタンバイ・インスタンスをプロモート時に待機する（-w）オプションを指定できるようになりました。従来はプロモートの完了を確認するためにはトリガー・ファイルを参照する必要がありました。

□ オプションの別名追加

操作の待機を行うオプション「-w」と、待機を行わないオプション「-W」の別名としてそれぞれ「--wait」と「--no-wait」を利用できるようになりました。

またオプションを指定する「-o」にも「--options」が利用できるようになりました。

□ 起動時も待機（-w）がデフォルト

すべての操作について、デフォルトで操作の完了を待つ（--wait）ようになりました。従来はインスタンス起動やプロモート処理のデフォルトの動作は操作の完了を待たない動作がデフォルトでした。

3.11.3 pg_basebackup

pg_basebackup コマンドは以下の変更が加わりました。

□ デフォルト・モードの変更

デフォルトの WAL 転送モードが Stream になりました。このためデフォルト状態で複数の wal sender プロセスへのコネクションを使用します。

□ -x オプションの廃止

-x オプション（--xlog オプション）は廃止されました。

□ -X オプションの変更

-X オプションに、トランザクション・ログを含めない none を指定可能になりました。また長いオプション名が「--xlog-method」から「--wal-method」に変更されました。

□ --xlogdir オプションの変更

--xlogdir オプションは--waldir オプションに変更されました。

□ -Ft オプションと-Xstream の組み合わせ

バックアップ・データを tar ファイルに出力する-Ft オプションと-Xstream オプションが同時に使用できるようになりました。この場合-D オプションで指定したディレクトリにトランザクション・ログが格納された pg_wal.tar ファイルが出力されます。

例 100　-Ft オプションと-Xstream オプション

```
$ pg_basebackup -D back1 -v -Ft -Xstream
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
<<途中省略>>
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: base backup completed
$ ls back1/
base.tar  pg_wal.tar
$ tar tvf back1/pg_wal.tar
-rw------- postgres/postgres 16777216 2017-05-20 16:36 000000100000000000000002F
-rw------- postgres/postgres        0 2017-05-20 16:36
archive_status/000000010000000000000002F.done
$
```

□ 一時レプリケーション・スロットの利用

スロット名（-S）を指定しない場合（かつ--no-slot を指定しない場合）には一時レプリケーション・スロットを利用します。下記は log_replication_commands パラメーターを on に指定した場合のログです。pg_basebackup_で始まる名前の一時スロットが作成されていることがわかります。

例 101 一時レプリケーション・スロットの作成ログ

```
LOG:  received replication command: IDENTIFY_SYSTEM
LOG:  received replication command: BASE_BACKUP LABEL 'pg_basebackup base
backup'     NOWAIT
LOG:  received replication command: IDENTIFY_SYSTEM
LOG:  received replication command: CREATE_REPLICATION_SLOT
"pg_basebackup_12889" TEMPORARY PHYSICAL RESERVE_WAL
LOG:  received replication command: START_REPLICATION SLOT
"pg_basebackup_12889" 0/49000000 TIMELINE 1
```

レプリケーション・スロットに空きがない場合、レプリケーション・スロットの作成が失敗するため pg_basebackup コマンドは失敗します。パラメーターmax_replication_slots に空きがあることを確認してください。

例 102 レプリケーション・スロット個数に余裕が無い場合のエラー

```
$ pg_basebackup -D back
pg_basebackup: could not connect to server: FATAL:  number of requested
standby connections exceeds max_wal_senders (currently 0)
pg_basebackup: removing contents of data directory "back"
$ echo $?
1
```

□ エラー発生クリーンアップ

pg_basebackup コマンド実行中にエラーが発生時や、シグナルを受けた場合に-D パラメーターで指定したディレクトリのファイルを削除するようになりました。この動作を行わない場合はパラメーター--no-clean（または-n）を指定します。

□ --verbose モードの出力

パラメーター--verbose（または-v）を指定した場合、より詳しい情報が表示されるように

なりました。

例 103 --verbose モードの出力

```
$ pg_basebackup  -D back --verbose
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/35000028 on timeline 1
pg_basebackup: starting background WAL receiver
pg_basebackup: write-ahead log end point: 0/35000130
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: base backup completed
$
```

3.11.4 pg_dump

以下のオプションが追加されました。

- □ Large Object を除外する-B オプション（--no-blobs オプション）
- □ Logical Replication で使用される SUBSCRIPTION オブジェクトを含めない --no-subscriptions オプション
- □ Logical Replication で使用される PUBLICATION オブジェクトを含めない --no-publications オプション
- □ ファイル書き込み後に sync システムコールを実行しない--no-sync オプション
  デフォルトでは sync システム・コールを呼び出し、確実に書き込みが行われることを保証します。

3.11.5 pg_dumpall

以下のオプションが追加されました。

- □ ファイル書き込み後に sync システムコールを実行しない--no-sync オプション
  デフォルトでは sync システム・コールを呼び出し、確実に書き込みが行われることを保証します。
- □ ロールのパスワードをダンプしない--no-role-passwords オプション
- □ Logical Replication で使用される SUBSCRIPTION オブジェクトを含めない --no-subscriptions オプション

□ Logical Replication で使用される PUBLICATION オブジェクトを含めない
--no-publications オプション

3.11.6 pg_recvlogical

指定された LSN を受信後にプログラムを終了する-E オプション(--endpos オプション)が追加されました。

3.11.7 pgbench

ログ・ファイルの先頭文字列を変更する--log-prefix パラメーターが追加されました。省略時の値は従来のバージョンと同様 pgbench_log です。pgbench コマンドには上記以外にもいくつかの新機能が提供されましたが、検証は行っていません。

3.11.8 initdb

「--noclean」「--nosync」オプションは「--no-clean」「--no-sync」オプションに変更されました。

3.11.9 pg_receivexlog

コマンドの名称が pg_receivewal に変更されました。出力される WAL ファイルを圧縮する--compress パラメーターが指定できるようになりました。圧縮率を 0 から 9 まで指定できます。この機能を利用するには libz ライブラリがインストールされた環境でビルドする必要があります。

3.11.10 pg_restore

以下のオプションが追加されました。

□ リストア対象外のスキーマを指定する-N オプション（--exclude-schema）
□ Logical Replication で使用される SUBSCRIPTION オブジェクトを含めない
--no-subscriptions オプション
□ Logical Replication で使用される PUBLICATION オブジェクトを含めない
--no-publications オプション

3.11.11 pg_upgrade

内部的にテーブルとシーケンスを別オブジェクトとして扱うようになりました。

3.11.12 createuser

createuser コマンドの--unencrypted オプション（-N オプション）は廃止されました。

3.11.13 createlang / droplang

createlang コマンド、droplang コマンドは廃止されました。

3.12 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。

3.12.1 postgres_fdw

postgres_fdw モジュールには以下の拡張機能が加えられました。

□ 集計処理のプッシュダウン

リモート・インスタンスに対する集計処理のプッシュダウンが可能になりました。

例 104 ローカル実行の SQL

```
postgres=# SELECT COUNT(*), AVG(c1), SUM(c1) FROM datar1 ;
count |         avg          |  sum
-------+----------------------+--------
  1000 | 500.5000000000000000 | 500500
(1 row)
```

上記 SQL 文が FOREIGN SERVER では下記の SQL 文に変換されます。

例 105 リモート実行の SQL（log_statement='all'のログから）

```
statement: START TRANSACTION ISOLATION LEVEL REPEATABLE READ
execute <unnamed>: DECLARE c1 CURSOR FOR
        SELECT count(*), avg(c1), sum(c1) FROM public.datar1
statement: FETCH 100 FROM c1
statement: CLOSE c1
statement: COMMIT TRANSACTION
```

□ FULL JOIN のプッシュダウン

リモート・テーブル同士の FULL JOIN を行う場合にプッシュダウンが行われるようになりました。

例 106 リモート・テーブル同士の FULL JOIN

```
postgres=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM (SELECT * FROM remote1
WHERE c1 < 10000) r1 FULL JOIN (SELECT * FROM remote2 WHERE c1 < 10000) r2 ON
(TRUE) LIMIT 10 ;

                              QUERY PLAN
-----------------------------------------------------------------------------
 Limit
   Output: remote1.c1, remote1.c2, remote2.c1, remote2.c2
   ->  Foreign Scan
         Output: remote1.c1, remote1.c2, remote2.c1, remote2.c2
         Relations: (public.remote1) FULL JOIN (public.remote2)
         Remote SQL: SELECT s4.c1, s4.c2, s5.c1, s5.c2 FROM ((SELECT c1, c2
FROM public.remote1 WHERE ((c1 < 10000::numeric))) s4(c1, c2) FULL JOIN (SELECT
c1, c2 FROM public.remote2 WHERE ((c1 < 10000::numeric))) s5(c1, c2) ON (TRUE))
(6 rows)
```

3.12.2 file_fdw

プログラムを実行する program オプションが指定できるようになりました。program オプションはファイル名を示す filename オプションの代わりに指定します。FOREIGN TABLE に対する SELECT 文が実行されると指定されたプログラムが自動的に実行されます。実行されたプログラムが標準出力に出力する内容がタプルとしてアプリケーションに返されます。

例 107 プログラム指定の file_fdw モジュール

```
postgres=# CREATE EXTENSION file_fdw ;
CREATE EXTENSION
postgres=# CREATE SERVER fs FOREIGN DATA WRAPPER file_fdw ;
CREATE SERVER
postgres=# CREATE FOREIGN TABLE tfile1 (id NUMERIC, val VARCHAR(10)) SERVER fs
    OPTIONS (program '/home/postgres/bin/file_fdw.py', delimiter ',') ;
CREATE FOREIGN TABLE
postgres=# SELECT * FROM tfile1 ;
```

上記では FOREIGN TABLE である tfile1 を検索すると、プログラム file_fdw.py が実行されます。以下のように標準出力を利用します。

例 108 実行されるプログラム例

```
#!/bin/python
for x in range(1000):
    print x,",test"
```

プログラムの終了が SELECT 文の終了とみなされます。内部的には popen(3)関数にプログラムを渡して実行しています（src/backend/storage/file/fd.c ファイル内の OpenPipeStream 関数）。

3.12.3 amcheck

BTree インデックスの整合性をチェックする amcheck モジュールが Contrib モジュールに追加されました。このモジュールには以下の関数が追加されています。

- bt_index_check(index regclass)
  指定された BTree インデックスの整合性をチェック
- bt_index_parent_check(index regclass)
  親子関係を持つインデックスの整合性もチェック

下記の例は一部のデータが破壊されているインデックス（idx1_check1）に対して bt_index_check 関数を実行しています。

例 109 bt_index_check 関数の実行

```
postgres=# CREATE EXTENSION amcheck ;
CREATE EXTENSION
postgres=# SELECT bt_index_check('idx1_check1') ;
ERROR:  item order invariant violated for index "idx1_check1"
DETAIL:  Lower index tid=(1,2) (points to heap tid=(0,2)) higher index tid=(1,3)
(points to heap tid=(0,3)) page lsn=0/7EFE4638.
```

3.12.4 pageinspect

Hash Index に対応する以下の関数が追加されました。

- hash_page_type
- hash_page_stats
- hash_page_items
- hash_metapage_info
- page_checksum
- bt_page_items(IN page bytea)

3.12.5 pgstattuple

以下の関数が追加されました。

- pgstathashindex

Hash Index の情報を提供します。

例 110 pgstathashindex

```
postgres=# SELECT * FROM pgstathashindex('idx1_hash1') ;
-[ RECORD 1 ]--+-----------------
version        | 3
bucket_pages   | 33
overflow_pages | 15
bitmap_pages   | 1
unused_pages   | 32
live_items     | 13588
dead_items     | 0
free_percent   | 58.329244357213
```

3.12.6 btree_gist / btree_gin

GiST インデックスを UUID 型の列と ENUM 型の列に作成できるようになりました。GIN インデックスを ENUM 型の列に作成できるようになりました。

例 111 UUID 型と ENUM 型の列に GiST インデックス作成

```
postgres=# CREATE EXTENSION btree_gist ;
CREATE EXTENSION

postgres=> CREATE TYPE type1 AS ENUM ('typ1', 'typ2', 'typ3') ;
CREATE TYPE
postgres=> CREATE TABLE gist1(c1 UUID, c2 type1) ;
CREATE TABLE
postgres=> CREATE INDEX idx1_gist1 ON gist1 USING gist (c1) ;
CREATE INDEX
postgres=> CREATE INDEX idx2_gist1 ON gist1 USING gist (c2) ;
CREATE INDEX
```

3.12.7 pg_stat_statements

pg_stat_statements ビューの query 列に格納される SQL 文のフォーマットが変更されました。WHERE 句のリテラル値が従来はクエスチョン・マーク（?）で出力されていましが、${N} (N=1,2,...) に変更されました。

例 112 pg_stat_statements ビュー

```
postgres=> SELECT query FROM pg_stat_statements WHERE query LIKE '%part1%' ;
                      query
-------------------------------------------------
SELECT COUNT(*) FROM part1 WHERE c1=$1
SELECT COUNT(*) FROM part1 WHERE c1=$1 AND c2=$2
(2 rows)
```

3.12.8 tsearch2

tsearch2 モジュールは削除されました。

Hewlett Packard
Enterprise

参考にした URL

本資料の作成には、以下の URL を参考にしました。

- Release Notes
  https://www.postgresql.org/docs/devel/static/release-10.html
- Commitfests
  https://commitfest.postgresql.org/
- PostgreSQL 10 Beta Manual
  https://www.postgresql.org/docs/devel/static/index.html
- GitHub
  https://github.com/postgres/postgres
- Open source developer based in Japan（Michael Paquier さん）
  http://paquier.xyz/
- 日々の記録 別館（ぬこ@横浜さん）
  http://d.hatena.ne.jp/nuko_yokohama/
- Qiita（ぬこ@横浜さん）
  http://qiita.com/nuko_yokohama
- pgsql-hackers Mailing list
  https://www.postgresql.org/list/pgsql-hackers/
- PostgreSQL 10 Beta 1 のアナウンス
  https://www.postgresql.org/about/news/1749/
- PostgreSQL 10 Roadmap
  https://blog.2ndquadrant.com/postgresql-10-roadmap/
- PostgreSQL10 Roadmap
  https://wiki.postgresql.org/wiki/PostgreSQL10_Roadmap
- Slack - postgresql-jp
  https://postgresql-jp.slack.com/

Hewlett Packard
Enterprise

変更履歴

変更履歴

版	日付	作成者	説明
0.1	2017/04/04	篠田典良	内部レビュー版作成 レビュー担当（敬称略）： 永安悟史（アップタイム・テクノロジーズ合同会社） 高橋智雄
0.9	2017/05/21	篠田典良	PostgreSQL 10 Beta 1 公開版に合わせて修正完了
1.0	2017/05/22	篠田典良	公開版を作成

以上