Hewlett Packard Enterprise

2016 年 5 月 30 日

PostgreSQL 9.6 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 9.6 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 9.5 と PostgreSQL 9.6 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。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 9.5.3 (比較対象) PostgreSQL 9.6 Beta 1 (2016/5/9 9:04:48 pm)
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@hpe.com）までお知らせください。

Hewlett Packard
Enterprise

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

Hewlett Packard
Enterprise

2. 新機能概要

PostgreSQL 9.6 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。

- 外部ソートに Quick Sort を使用
- GROUP BY 句の推定精度向上
- 結合の予測に外部キーを利用
- チェックポイントや bgwriter 等の書き込み改善
- 部分インデックスで Index Only Scan の実行
- CREATE INDEX CONCURRENTLY 文の高速化
- etc

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。

- Parallel Seq Scan（3.5）
- 待機状態のモニタリング（3.6）
- FOREIGN DATA WRAPPER の拡張（3.7）
- 複数同期レプリケーション（3.8）
- Full Table Vacuum の廃止（3.1.4）
- 時間指定による Snapshot Too Old の実装（3.3.1）
- 各種 Contrib モジュールの変更（3.1.3）
- 各種ユーティリティの改善（3.2）
- wal receiver の状態確認カタログ（3.1.1）
- Generic WAL Records
- etc

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。

- COPY 文の拡張（3.4.1）
- ALTER TABLE ADD COLUMN 文の拡張 （3.4.2）
- ALTER TABLESPACE SET 文の拡張（3.4.3）
- CREATE EXTENSION 文の拡張（3.4.4）
- ALTER OPERATOR 文の拡張（3.4.5）
- CREATE / ALTER FUNCTION 文の拡張（3.5）
- CREATE ACCESS METHOD 文の追加
- CREATE TABLE ... LIKE 文の拡張
- 関数の追加（3.4.6 / 3.4.7）
- etc

その他の改善点は、PostgreSQL 9.6 Documentation Appendix E. Release Notes（http://www.postgresql.org/docs/9.6/static/release-9-6.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 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_config	インストール情報
pg_stat_wal_receiver	wal receiver プロセスの活動状況
pg_stat_progress_vacuum	Vacuum 進捗状況の確認
pg_init_privs	オブジェクトの初期権限

□ pg_config カタログ

PostgreSQL バイナリのコンパイル時に指定された各種マクロ値を取得できます。従来は pg_config コマンドで確認していました。本カタログの実体は pg_config 関数です。このカタログは superuser 権限を持つユーザーのみが参照できます。

表 5 pg_config カタログ

列名	データ型	説明
name	text	マクロ名
setting	text	設定値

例 1 pg_config カタログの検索

```
postgres=# SELECT * FROM pg_config ;
        name         |  setting
---------------------+-----------------------------------------------------------
 BINDIR              | /usr/local/pgsql/bin
 DOCDIR              | /usr/local/pgsql/share/doc
 HTMLDIR             | /usr/local/pgsql/share/doc
 INCLUDEDIR          | /usr/local/pgsql/include
 PKGINCLUDEDIR       | /usr/local/pgsql/include
<<以下省略>>
```

□ pg_stat_wal_receiver カタログ

レプリケーション環境のスレーブ・インスタンスで参照できるカタログです。wal receiver プロセスの活動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。

表 6 pg_stat_wal_receiver カタログ

列名	データ型	説明
pid	integer	wal receiver プロセス ID
status	text	ステータス
receive_start_lsn	pg_lsn	受信開始 LSN
receive_start_tli	integer	受信開始 TimeLine ID
received_lsn	pg_lsn	受信 LSN
received_tli	integer	受信 TimeLine ID
last_msg_send_time	timestamp with time zone	最終メッセージ送信時刻
last_msg_receipt_time	timestamp with time zone	最終メッセージ受信時刻
latest_end_lsn	pg_lsn	最新の受信 LSN
latest_end_time	timestamp with time zone	最新の受信 LSN タイムスタンプ
slot_name	text	接続先スロット名

このカタログは pg_stat_get_wal_receiver 関数の実行結果から作成されています。

例 2 pg_stat_wal_receiver カタログの検索

```
postgres=# SELECT * FROM pg_stat_wal_receiver ;
-[ RECORD 1 ]---------+----------------------------
pid                   | 2782
status                | streaming
receive_start_lsn     | 0/36000000
receive_start_tli     | 1
received_lsn          | 0/36000000
received_tli          | 1
last_msg_send_time    | 2016-05-15 11:49:09.753784+09
last_msg_receipt_time | 2016-05-15 11:49:09.753879+09
latest_end_lsn        | 0/360000D0
latest_end_time       | 2016-05-15 11:49:09.753784+09
slot_name             | slot_1
```

□ pg_stat_progress_vacuum カタログ

pg_stat_progress_vacuum は Vacuum 処理の進捗状況を確認するためのカタログです。このカタログは一般ユーザーでも検索できます。

表 7 pg_stat_progress_vacuum カタログ

列名	データ型	説明
pid	integer	バックエンド・プロセス ID
datid	oid	接続データベースの OID
datname	name	接続データベース名
relid	oid	Vacuum 処理を行っているテーブルの OID
phase	text	Vacuum 実行フェーズ
heap_blks_total	bigint	テーブルの合計ブロック数（Vacuum 開始時）
heap_blks_scanned	bigint	スキャンされたブロック数
heap_blks_vacuumed	bigint	Vacuum 処理されたブロック数
index_vacuum_count	bigint	インデックスに対する Vacuum 完了回数
max_dead_tuples	bigint	maintenance_work_mem に含むことができる最大の不要タプル数
num_dead_tuples	bigint	直前の Vacuum で収集した不要タプル数

例 3 pg_stat_progress_vacuum カタログの検索

```
postgres=# SELECT * FROM pg_stat_progress_vacuum ;
-[ RECORD 1 ]------+--------------
pid                | 3184
datid              | 16385
datname            | demodb
relid              | 16398
phase              | scanning heap
heap_blks_total    | 10052
heap_blks_scanned  | 2670
heap_blks_vacuumed | 2669
index_vacuum_count | 0
max_dead_tuples    | 291
num_dead_tuples    | 185
```

□ pg_init_privs カタログ

pg_init_privs カタログはデータベースに格納されたオブジェクトのデフォルト以外の初期権限を格納します。このカタログは一般ユーザーでも検索できます。

表 8 pg_init_privs カタログ

列名	データ型	説明
objoid	oid	対象オブジェクト ID
classoid	oid	オブジェクトが格納されたカタログの OID
objsubid	integer	テーブルの列番号（テーブル以外の場合は 0）
privtype	char	権限の種類
initprivs	aclitem[]	初期権限リスト

3.1.2 カタログの変更

以下のカタログが変更されました。

表 9 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	confirmed_flush_lsn 列追加
pg_stat_activity	クライアントのセッション情報	waiting 列削除 wait_event_type 列、wait_event 列追加
pg_proc	ファンクション情報	proparallel 列追加
pg_aggregate	集約関数情報	aggcombinefn 列、 aggserialfn 列、aggdeserialfn 列、aggserialtype 列追加
pg_am	インデックス・アクセス・メソッド情報	全面的に変更

□ pg_replication_slots カタログ

confirmed_flush_lsn 列が追加されました。

表 10 pg_replication_slots カタログの追加列

列名	データ型	説明
confirmed_flush_lsn	pg_lsn	ロジカルスロットの受信 LSN 情報

□ pg_stat_activity カタログ

待ち状態かどうかを示すだけの waiting 列が削除され、待機イベントを示す wait_event_type 列と wait_event 列が追加されました。

表 11 pg_stat_activity カタログの追加列

列名	データ型	説明
wait_event_type	text	待機イベントのタイプ
wait_event	text	待機イベント名

待機イベントの情報は以下の URL を参照してください。
https://www.postgresql.org/docs/9.6/static/monitoring-stats.html

□ pg_proc カタログ

プロシージャが PARALLEL SAFE か PARALLEL UNSAFE かを示す proparallel 列が追加されました。

表 12 pg_proc カタログの追加列

列名	データ型	説明
proparallel	char	PARALLEL SAFE な場合は's'、PARALLEL UNSAFE の場合は'u'となる。

□ pg_aggregate カタログ

以下の列が追加されました。

表 13 pg_aggregate カタログの追加列

列名	データ型	説明
aggcombinefn	regproc	コンバイン関数（存在しない場合は 0）
aggserialfn	regproc	シリアライズ関数（存在しない場合は 0）
aggdeserialfn	regproc	デシリアライズ関数（存在しない場合は 0）
aggserialtype	oid	シリアライズ関数の種類

□ pg_am カタログ

pg_am カタログは全面的に書き換えられ、シンプルになりました。

表 14　pg_am カタログの構成列

列名	データ型	説明
amname	name	アクセスメソッド名
amhandler	oid	ハンドラー関数の OID
amtype	char	アクセスメソッドの種類

2016 年 5 月 30 日現在、マニュアル（https://www.postgresql.org/docs/9.6/static/catalog-pg-am.html）上には amtype 列の記述がありません。

3.1.3 Contrib モジュールの変更

PostgreSQL 9.6 では、いくつかの Contrib モジュールが変更されました。

表 15　Contrib モジュールの変更点

モジュール	変更点	備考
auto_explain	パラメーター追加	sample_rate オプション追加
postgres_fdw	オプション追加	fetch_size オプション追加 extensions オプション追加
pg_visibility	モジュール追加	Visibility Map の情報検索
bloom	モジュール追加	Bloom filter を使ったインデックス
sslinfo	関数の追加	ssl_extension_info 関数追加
tsearch2	機能拡張	フレーズ検索のための演算子追加
pg_trgm	機能拡張	単語類似性のサポート。パラメーター pg_trgm.similarity_threshold の追加
pgcrypto	機能拡張	S2K 関連パラメーターの追加
pgpageinspect	機能拡張	heap_page_items 関数に出力データ追加
hstore	関数の追加	json 関連関数の追加

□　auto_explain モジュールの機能拡張

auto_explain モジュールに、パラメーターauto_explain.sample_ratio が追加されました。このパラメーターは実行計画を出力する SQL 文の割合を指定します。auto_explain.log_min_duration パラメーター等で実行計画を出力する予定の SQL 文をこのパラメーターで指定された割合に削減します。デフォルト値は 1（=100%）です。

□　postgres_fdw モジュールの機能拡張

SELECT 文実行時にタプルを取得するフェッチ・サイズを指定できるようになりました。

SERVER または FOREIGN TABLE 単位に fetch_size オプションを指定することができます。従来のバージョンではフェッチ・サイズは 100 で固定されていました。

例 4 fetch_size オプション

```
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS (host 'remhost1', port '5432', dbname 'demodb', fetch_size '200') ;
CREATE SERVER
postgres=# CREATE FOREIGN TABLE table1(c1 NUMERIC, c2 VARCHAR(10))
  SERVER remsvr1 OPTIONS(fetch_size '300') ;
CREATE FOREIGN TABLE
```

以下の例はリモート・インスタンスに対して「SELECT * FROM table1」文を投入した場合に実際に実行される SQL 文です。

例 5 実行される SQL 文（ログファイルから）

```
LOG:  duration: 0.072 ms  statement: START TRANSACTION ISOLATION LEVEL
REPEATABLE READ
LOG:  duration: 156.616 ms  parse <unnamed>: DECLARE c1 CURSOR FOR
        SELECT c1, c2 FROM public.table1
LOG:  duration: 0.102 ms  bind <unnamed>: DECLARE c1 CURSOR FOR
        SELECT c1, c2 FROM public.table1
LOG:  duration: 0.039 ms  execute <unnamed>: DECLARE c1 CURSOR FOR
        SELECT c1, c2 FROM public.table1
LOG:  duration: 0.272 ms  statement: FETCH 300 FROM c1
LOG:  duration: 0.202 ms  statement: FETCH 300 FROM c1
LOG:  duration: 0.028 ms  statement: CLOSE c1
LOG:  duration: 0.038 ms  statement: COMMIT TRANSACTION
```

リモート・インスタンスに対する EXTENSION を指定できるようになりました。

例 6 extensions 指定

```
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'remsvr1', port '5433', dbname 'postgres', extensions 'hstore') ;
CREATE SERVER
```

□ pg_visibility モジュールの追加

Visibility Map の情報を取得できる pg_visibility モジュールが追加されました。以下の関数が提供されます。実行には superuser 権限が必要です。

表 16 pg_visibility モジュールで定義された関数

関数	説明
pg_visibility	指定テーブルのブロック毎の状態を表示
pg_visibility_map	指定テーブルのブロック毎の状態を表示
pg_visibility_map_summary	指定テーブルの状態を表示

下記の例では pg_visibility_map_summary 関数を実行して、VISIBLE なブロック数と FREEZE されたブロック数を表示しています。

例 7 pg_visibility モジュール

```
postgres=# CREATE EXTENSION pg_visibility ;
CREATE EXTENSION
postgres=# SELECT pg_visibility_map_summary('data1') ;
 pg_visibility_map_summary
---------------------------
 (5406,5406)
(1 row)
```

□ bloom モジュールの追加

Contrib モジュールに bloom モジュールが追加されました。bloom モジュールをロードすることで Bloom Filter を使ったインデックスを作成できます。このモジュールを使うと、多数の列に対して同時にインデックスを作成することができ、ストレージ容量の削減にもなります。簡単な検証の結果、WHERE 句に記述された列に BTree インデックスが作成されている場合には BTree インデックスが使用されます。

bloom モジュールを使用するには CREATE INDEX 文に USING bloom 句を指定します。専用のオプションとして size, col1,col2,…がありますが、用途は確認していません。

例 8 bloom モジュールの作成と実行計画

```
postgres=# CREATE EXTENSION bloom ;
CREATE EXTENSION
postgres=> CREATE TABLE bloom1(c1 INTEGER, c2 INTEGER, c3 INTEGER, c4 INTEGER,
                c5 INTEGER);
CREATE TABLE
postgres=> CREATE INDEX bl1_bloom1 ON bloom1 USING bloom (c1, c2, c3, c4, c5) ;
CREATE INDEX
postgres=> EXPLAIN ANALYZE SELECT * FROM bloom1 WHERE c1 = 10000 AND c5 =
10000 ;
                          QUERY PLAN
-------------------------------------------------------------------------------------
 Bitmap Heap Scan on bloom1  (cost=17848.00..17852.02 rows=1 width=20)
        (actual time=8.376..8.538 rows=1 loops=1)
   Recheck Cond: ((c1 = 10000) AND (c5 = 10000))
   Rows Removed by Index Recheck: 76
   Heap Blocks: exact=76
   ->  Bitmap Index Scan on bl1_bloom1  (cost=0.00..17848.00 rows=1 width=0)
         (actualtime=8.337..8.337 rows=77 loops=1)
          Index Cond: ((c1 = 10000) AND (c5 = 10000))
 Planning time: 0.123 ms
 Execution time: 8.579 ms
(8 rows)
```

□ pageinspect モジュールの拡張

heap_page_items 関数に実データの出力列（t_data）が追加されました。

例 9 heap_page_items 関数の実行

```
postgres=# CREATE EXTENSION pageinspect ;
CREATE EXTENSION
postgres=# SELECT lp, t_data FROM heap_page_items(get_raw_page('insp1', 0)) ;
 lp |         t_data
----+------------------------
  1 | ¥x0b008064000b696e6974
  2 | ¥x0b0080c8000b696e6974
(2 rows)
```

3.1.4 Full Table Vacuum の廃止

PostgreSQL はトランザクション（XID）の新旧を、符号なし 32 ビット整数で管理しています。実行されるトランザクション数が膨大になると 32 ビット整数を使い切る可能性がでてきます。このため PostgreSQL はトランザクション ID を使い切る前に、データベース内に格納された古い XID を特殊な XID（FrozenXID=2）に更新する処理を行います。この処理を FREEZE と呼びます。従来のバージョンの FREEZE 処理はテーブルの更新に関わらずパラメーターautovacuum_freeze_max_age で指定された年代（age）を超過すると、テーブルに対するフルスキャンが実行されていました。

PostgreSQL 9.6 では Visibility Map を拡張することで FREEZE 対象となるブロックを特定し、フルスキャンによる大規模な I/O を防止することができるようになりました。

3.1.5 CHECKPOINT の改善

これまでのチェックポイント処理は、共有バッファ内のダーティ・ページをランダムに検索して書き込みを行っていました。PostgreSQL 9.6 ではダーティ・ページを表領域単位に分割し、ファイルとブロック番号によりソートを行ってから書き込みを行います。これにより、よりシーケンシャルな書き込みが行われるようになります。

3.2 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。

3.2.1 psql

psql コマンドには以下の機能が追加されました。

□ プロンプト

psql コマンドのプロンプトに、バックエンド・プロセスの ID を指定できるようになりました。PROMPT1 / PROMPT2 / PROMPT3 変数に%p を指定します。

例 10 プロンプトの指定

```
postgres=> ¥set PROMPT1 '%/(%p)=> '
postgres(2619)=>
```

□ ¥コマンド

以下の拡張が行われました。

表 17　追加／変更された¥コマンド

コマンド	変更	説明
¥ev *view_name*	機能追加	外部エディタで View 定義を変更
¥sv *view_name*	機能追加	View 定義を表示
¥sv+ *view_name*	機能追加	View 定義を行番号付きで表示
¥gexec	機能追加	出力結果内の SQL 文を実行
¥errverbose	機能追加	直前に発生したエラー情報を表示
¥crosstabview *column_name*	機能追加	クロスタブの表示
¥x auto	動作変更	EXPLAIN ANALYZE では拡張表示しない
¥watch と ¥pset title	動作変更	¥watch 実行時に¥pset title の値を表示

□ VIEW 定義の編集と参照

psql に¥ev と¥sv コマンドが追加されました。¥ev コマンドはビュー名を指定することで、ビュー定義をエディタ内に読み込むことができます。¥sv コマンドはビュー定義を表示します。¥sv+コマンドを実行すると、ビュー定義に行番号が表示されます。

例 11 VIEW 定義の参照

```
postgres=> CREATE VIEW view1 AS SELECT COUNT(*) cnt FROM data1 ;
CREATE VIEW
postgres=> ¥sv+ view1
1        CREATE OR REPLACE VIEW public.view1 AS
2         SELECT count(*) AS cnt
3           FROM data1
postgres=>
```

□ バッファ内の SQL 文を実行

¥gexec コマンドを実行すると、直前に出力バッファに表示された内容を SQL 文として実行することができます。SELECT 文を使ってオブジェクトを生成する CREATE 文を作成した場合に便利です。

例 12 出力結果を実行

```
postgres=> SELECT 'CREATE TABLE data2(c1 NUMERIC)' ;
            ?column?
--------------------------------
 CREATE TABLE data4(c1 NUMERIC)
(1 row)
postgres=> ¥gexec
CREATE TABLE
```

□ ¥watch と¥pset title

¥watch コマンドの実行時に、¥pset title コマンドで指定した文字列が表示されるようになりました。

例 13 ¥watch と¥pset title

```
postgres=> ¥pset title 'Demo Title'
Title is "Demo Title".
postgres=> SELECT COUNT(*) FROM data1 ;
Demo Title
  count
---------
 3000000
(1 row)
postgres=> ¥watch 1
Demo Title      Wed May 13 12:07:41 2016 (every 1s)

 count
-------
 10000
(1 row)
```

□ 複数の-c オプションと-f オプション

接続時に実行する SQL 文を指定する-c オプションと-f オプションが複数記述できるようになりました。

3.2.2 pg_basebackup

バックアップに使用するスロット名を指定する--slot（または-S）オプションが追加されました。存在しないスロット名を指定すると警告が出力されますが、バックアップ処理自体は実行されます。--write-recovery-conf オプション（-R）と共に指定すると、バックアップ先の recovery.conf ファイルの primary_slot_name オプションが追加されます。

--slot オプションを使うためには、同時に--xlog-method=stream (-Xs)の設定が必要です。

例 14 --slot オプションの指定

```
$ pg_basebackup -D back -x -v -R --slot=slot_1 ← -Xs 指定せず
pg_basebackup: replication slots can only be used with WAL streaming
Try "pg_basebackup --help" for more information.
$
$ pg_basebackup -D back -v -R --slot=slot_X -Xs ← 存在しないスロット
transaction log start point: 0/4000028 on timeline 1
pg_basebackup: starting background WAL receiver
pg_basebackup: could not send replication command "START_REPLICATION": ERROR:
replication slot "slot_X" does not exist
transaction log end point: 0/4000130
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: child process exited with error 1
$
$ pg_basebackup -D back -v -R --slot=slot_1 -Xs ← 正常に実行
transaction log start point: 0/6000028 on timeline 1
pg_basebackup: starting background WAL receiver
transaction log end point: 0/60000F8
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: base backup completed
$
$ cat back/recovery.conf ← recovery.conf の確認
standby_mode = 'on'
primary_conninfo = 'user=postgres port=5432 sslmode=disable sslcompression=1'
primary_slot_name = 'slot_1'
$
$ ls -l back/pg_replslot/ ← バックアップ先のスロット確認
total 0
$
```

バックアップ先のデータベース・クラスタにスロットが作成されるわけではありません。

3.2.3 pg_rewind

pg_rewind コマンドがタイムラインの変更にも対応できるようになりました。この機能は検証していません。以下の URL を参照してください。

http://michael.otacoo.com/postgresql-2/postgres-9-6-feature-highlight-pg-rewind-timeline/

3.2.4 pg_dump / pg_restore

pg_dump コマンド、pg_restore コマンドに--strict-names オプションが追加されました。また pg_restore コマンドの-t オプションに通常テーブル以外のリレーションもマッチします。この機能は検証していません。

3.2.5 pgbench

pgbench コマンドにはいくつかの新機能が提供されましたが、検証していません。

3.3 パラメーターの変更

PostgreSQL 9.6 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 18　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
backend_flush_after	バックエンドが指定サイズ以上の書き込み発生時にフラッシュを強制（user）	128kB
bgwriter_flush_after	bgwriter が指定サイズ以上の書き込み発生時にフラッシュを強制（sighup）	512kB
checkpoint_flush_after	checkpointer が指定サイズ以上の書き込み発生時にフラッシュを強制（sighup）	256kB
enable_fkey_estimates	結合コストの推定に外部キーを利用（user）	on
force_parallel_mode	並列処理を強制（user）	2
idle_in_transaction_session_timeout	アイドル時間のタイムアウト（user）	0
max_parallel_degree	並列度の最大値（user）	2
old_snapshot_threshold	スナップショットが古いとみなす時間（postmaster）	-1
parallel_setup_cost	並列処理の開始コスト（user）	1000
parallel_tuple_cost	並列処理のタプル処理コスト（user）	0.1
replacement_sort_tuples	ソート時に Replacement Selection を使用する最大タプル数（user）	150000
syslog_sequence_numbers	SYSLOG 出力メッセージにシーケンス番号を追加（sighup）	on
syslog_split_messages	長いSYSLOGメッセージの分割(sighup)	on
wal_writer_flush_after	wal writer が指定サイズ以上の書き込み発生時にフラッシュを強制（sighup）	1MB

□　idle_in_transaction_session_timeout パラメーター

トランザクションが一定時間アイドル状態になると強制的にセッションを切断する時間をミリ秒単位で指定します。デフォルト値は 0 でタイムアウトは発生しません。以下の例

では psql からセッションのパラメーターを指定し、BEGIN 文実行後にしばらく待ってから COMMIT 文を実行しています。

例 15　アイドル・トランザクションの自動終了

```
postgres=> SET idle_in_transaction_session_timeout = 1000 ;
SET
postgres=> BEGIN ;
BEGIN
postgres=> -- Wait 2 seconds
postgres=> COMMIT ;
FATAL:  terminating connection due to idle-in-transaction timeout
server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
The connection to the server was lost. Attempting reset: Succeeded.
```

□　syslog_sequence_numbers パラメーター、syslog_split_messages パラメーター

SYSLOG に出力されるメッセージに情報を追加します。syslog_sequence_numbers パラメーターを on に設定すると、SYSLOG に出力されるメッセージにプロセスごとのシーケンス番号が追加されます。デフォルト値は on です。syslog_split_messages パラメーターにはメッセージが PG_SYSLOG_LIMIT（900）バイトを超えた場合にメッセージを分割します。

下記はインスタンス起動時のログです。プロセス ID 3155 のプロセスが、メッセージを出力しています、[1-1], [1-2], [2-1], [2-2]の部分が syslog_sequence_numbers パラメーターで追加されています。

例 16　SYSLOG 出力の変更

```
May 15 13:35:55 rel71-2 postgres[3155]: [1-1] LOG:  redirecting log output to
logging collector process
May 15 13:35:55 rel71-2 postgres[3155]: [1-2] HINT:  Future log output will
appear in directory "pg_log".
May 15 13:35:55 rel71-2 postgres[3155]: [2-1] LOG:  ending log output to stderr
May 15 13:35:55 rel71-2 postgres[3155]: [2-2] HINT:  Future log output will go
to log destination "syslog".
```

□ old_snapshot_threshold パラメーター

このパラメーターにはスナップショットの生存期間を秒単位で指定します。閾値を越えた不要タプルは Vacuum により開放することができます。このパラメーターに指定できる秒数は-1 から 86,400 までですが、「1d」のように日数を指定する場合には 60d まで指定できます。デフォルト値の-1 は旧バージョンと同じ動作で、この機能を無効にします。

削除されたスナップショットを参照するとエラー「ERROR:　snapshot too old」が発生します。

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 19　変更されたパラメーター

パラメーター	変更内容
log_line_prefix	数値型のタイムスタンプを指定する設定が追加されました。
effective_io_concurrency	ALTER TABLESPACE SET 文で指定可能になりました。
wal_level	設定値 archive と hot_standby は replica に統一されました。
synchronous_commit	remote_apply を指定可能になりました。
autovacuum_max_workers	最大値が 8,388,607 から 262,143 に変更されました。
max_connections	最大値が 8,388,607 から 262,143 に変更されました。
max_replication_slots	最大値が 8,388,607 から 262,143 に変更されました。
max_wal_senders	最大値が 8,388,607 から 262,143 に変更されました。
max_worker_processes	最大値が 8,388,607 から 262,143 に変更されました。
superuser_reservrd_connections	最大値が 8,388,607 から 262,143 に変更されました。
wal_writer_delay	pg_settings カタログの short_desc 列が変更されました。

□ パラメーターlog_line_prefix

タイムスタンプの数値表現である%n を指定できるようになりました。

例 17　パラメーターlog_line_prefix

```
$ grep log_line_prefix data/postgresql.conf
log_line_prefix = '%n '
$ tail -1 data/pg_log/postgresql-2016-05-15_171832.log
1460362712.163 LOG:  autovacuum launcher started
```

□　パラメーターwal_level

パラメーターwal_level の値 archive と hot_standby は replica に統一されました。ただし archive や hot_standby を指定してもエラーにはならず、replica とみなされます。

例 18　パラメーターwal_level

```
$ grep wal_level data/postgresql.conf
wal_level = hot_standby
$ psql
postgres=> show wal_level ;
wal_level
-----------
 replica
(1 row)
```

□　パラメーターsynchronous_commit

パラメーター設定値として remote_apply を指定できるようになりました。この値を設定すると、同期レプリケーション環境で、マスター・インスタンスの更新トランザクションがスレーブ・インスタンス上で WAL が適用されるまで待機します。これにより更新トランザクションの完了と同時にスレーブ・インスタンスで更新済みデータが参照できることになります。

スレーブ・インスタンスの recovery.conf ファイルにパラメーター recovery_min_apply_delay の指定があると、指定時間だけマスター・インスタンスの COMMIT 文の完了が待機します。

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 20　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 9.5	PostgreSQL 9.6
server_version	9.5.3	9.6beta1
server_version_num	90503	90600

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 COPY 文の拡張

従来バージョンでも COPY 文を使って SELECT 文の結果をファイルや標準出力に出力することができました。PostgreSQL 9.6 では、COPY 文に UPDATE / DELETE / INSERT 文を指定して、影響があったレコードを出力することができるようになりました。

例 19　COPY 文に DELETE 文を指定

```
postgres=# COPY (DELETE FROM data1 WHERE col1 < 100 RETURNING *) TO
        '/home/postgres/data1.csv' ;
COPY 5
```

上記の例では DELETE 文により削除された 5 件のレコードをファイルに出力しています。DELETE | UPDATE | INSERT 文を指定する場合には RETURNING 句が必要です。

例 20　RETURNING 句が無い場合

```
postgres=# COPY (DELETE FROM data1 WHERE col1 < 100) TO
       '/home/postgres/data1.csv' ;
ERROR:  COPY query must have a RETURNING clause
```

3.4.2 ALTER TABLE ADD COLUMN 文の拡張

列を追加する ALTER TABLE ADD COLUMN 文に、列の存在をチェックする IF NOT EXISTS 句が指定できるようになりました。指定された列が存在しない場合に限り、列追加操作が行われます。

構文

```
ALTER TABLE table_name ADD COLUMN IF NOT EXISTS column_name type
```

例 21　ALTER TABLE ADD COLUMN IF NOT EXISTS 文

```
postgres=> ALTER TABLE data1 ADD COLUMN c3 CHAR(1) ;
ALTER TABLE
postgres=> ALTER TABLE data1 ADD COLUMN IF NOT EXISTS c3 CHAR(1) ;
NOTICE:  column "c3" of relation "data1" already exists, skipping
ALTER TABLE
postgres=> ALTER TABLE data1 ADD COLUMN IF NOT EXISTS c4 CHAR(1) ;
ALTER TABLE
```

3.4.3 ALTER TABLESPACE SET 文の拡張

表スペース単位にパラメーターeffective_io_concurrency を設定できるようになりました。旧バージョンでは表スペース単位に指定できるパラメーターは random_page_cost と seq_page_cost のみでした。

例 22 ALTER TABLESPACE 文の拡張

```
postgres=# ALTER TABLESPACE ts1 SET (effective_io_concurrency = 2) ;
ALTER TABLESPACE
postgres=# ¥db+ ts1
                                                List of tablespaces
 Name | Owner |       Location      | Access privileges |        Options
    |  Size  | Description
------+-------+---------------------+-------------------+--------------------
----+--------+------------
 ts1  | demo  | /home/postgres/ts1 |                   |
{effective_io_concurrency=2} | 472 kB |
(1 row)
```

3.4.4 CREATE EXTENSION 文の拡張

CREATE EXTENSION 文に、関連するモジュールを自動的にロードする CASCADE 句が指定できるようになりました。

構文

```
CREATE EXTENSION module_name [CASCADE]
```

例 23 CREATE EXTENSION 文の拡張

```
postgres=# CREATE EXTENSION earthdistance ;
ERROR:  required extension "cube" is not installed
HINT:  Use CREATE EXTENSION CASCADE to install required extensions too.
postgres=#
postgres=# CREATE EXTENSION earthdistance CASCADE ;
NOTICE:  installing required extension "cube"
CREATE EXTENSION
postgres=#
```

3.4.5 ALTER OPERATOR 文の拡張

オペレーターの変更を行う ALTER OPERATOR 文が大幅に拡張されました。CREATEOPERATOR 文と同様に以下の構文を追加利用できます。

構文

```
ALTER OPERATOR name ({type}, {type}) SET RESTRICT res_proc
ALTER OPERATOR name ({type}, {type}) SET JOIN join_proc

ALTER OPERATOR name ({type}, {type}) SET RESTRICT NONE
ALTER OPERATOR name ({type}, {type}) SET JOIN NONE
```

3.4.6 jsonb 型に対する関数

jsonb 型に対する関数が追加されました。

□ jsonb_insert 関数

要素の追加を行う jsonb_insert 関数が提供されました。

例 24　jsonb_insert 関数

```
postgres=> SELECT jsonb_insert('{"a": [0,1,2]}', '{a, 1}', '"new_value"') ;
         jsonb_insert
-------------------------------
 {"a": [0, "new_value", 1, 2]}
(1 row)

postgres=> SELECT jsonb_insert('{"a": [0,1,2]}', '{a, 1}', '"new_value"',
true) ;
         jsonb_insert
-------------------------------
 {"a": [0, 1, "new_value", 2]}
(1 row)
```

3.4.7 関数

以下の関数が追加／拡張されました。

□　scale 関数

引数に numeric 型の数値を受けて小数点以下の桁数を返します。float 型には使用できません。NULL が渡された場合には NULL を返します。

□　num_nulls / num_nonnulls 関数

任意数の引数の中から NULL 値の数（num_nulls）、非 NULL 値の数（num_nonnulls）を返します。

例 25　num_nulls 関数

```
postgres=> SELECT num_nulls(1, 'A', NULL), num_nonnulls(1, 'A', NULL) ;
num_nulls | num_nonnulls
-----------+--------------
         1 |            2
(1 row)
```

□ current_setting 関数

第 2 パラメーターが追加されたバージョンが作成されました。第 2 パラメーターに true を指定すると、存在しないパラメーターを指定してもエラーが発生しません。

例 26 current_setting 関数

```
postgres=> ¥pset null null
Null display is "null".
postgres=> SELECT current_setting('module1.param1') ;
ERROR:  unrecognized configuration parameter "module1.param1"
postgres=> SELECT current_setting('module1.param1', true) ;
 current_setting
-----------------
 null
(1 row)
```

□ pg_control_*関数

従来は pg_controldata コマンドで取得していた情報を SQL 関数で取得できるようになりました。以下の関数が使用できます。これらの関数は一般ユーザーも実行できます。

表 21 追加された関数名

関数名	取得情報
pg_control_init	データベース・クラスタ情報
pg_control_checkpoint	チェックポイント情報
pg_control_recovery	リカバリー情報

例 27　pg_control_init 関数の実行

```
postgres=> ¥x
Expanded display is on.
postgres=> SELECT * FROM pg_control_init() ;
-[ RECORD 1 ]--------------+---------
max_data_alignment         | 8
database_block_size        | 8192
blocks_per_segment         | 131072
wal_block_size             | 8192
bytes_per_wal_segment      | 16777216
max_identifier_length      | 64
max_index_columns          | 32
max_toast_chunk_size       | 1996
large_object_chunk_size    | 2048
bigint_timestamps          | t
float4_pass_by_value       | t
float8_pass_by_value       | t
data_page_checksum_version | 0
```

例 28　pg_control_checkpoint 関数の実行

```
postgres=> ¥x
Expanded display is on.
postgres=> SELECT * FROM pg_control_checkpoint() ;
-[ RECORD 1 ]-------+----------------------
checkpoint_location  | 0/E52BD28
prior_location       | 0/E42B3C8
redo_location        | 0/E52BD28
redo_wal_file        | 00000001000000000000000E
timeline_id          | 1
prev_timeline_id     | 1
full_page_writes     | t
next_xid             | 0:1767
next_oid             | 24576
next_multixact_id    | 1
next_multi_offset    | 0
oldest_xid           | 1748
oldest_xid_dbid      | 1
oldest_active_xid    | 0
oldest_multi_xid     | 1
oldest_multi_dbid    | 1
oldest_commit_ts_xid | 0
newest_commit_ts_xid | 0
checkpoint_time      | 2016-05-18 15:24:31+09
```

例 29　pg_control_recovery 関数の実行

```
postgres=> ¥x
Expanded display is on.
postgres=> SELECT * FROM pg_control_recovery() ;
-[ RECORD 1 ]-----------------+---
min_recovery_end_location     | 0/0
min_recovery_end_timeline     | 0
backup_start_location         | 0/0
backup_end_location           | 0/0
end_of_backup_record_required | f
```

□ pg_current_xlog_flush_location 関数

WAL ファイルの書き込み場所を示す LSN を返す関数 pg_current_xlog_flush_location が追加されました。この関数は一般ユーザーの権限で実行できますが、レプリケーション環境のスレーブ・インスタンスでは実行できません。

例 30 pg_current_xlog_flush_location 関数

```
postgres=> SELECT pg_current_xlog_flush_location() ;
 pg_current_xlog_flush_location
--------------------------------
 0/3000060
(1 row)
```

□ parse_ident 関数

スキーマ名を含むオブジェクト名を示す文字列をスキーマ名とオブジェクト名から構成される配列に分解します。指定されたオブジェクトの存在はチェックされません。また search_path パラメーターは考慮されません。

例 31 parse_ident 関数

```
postgres=> SELECT parse_ident('public.data1') ;
  parse_ident
----------------
 {public,data1}
(1 row)
```

□ pg_size_bytes 関数

pg_size_bytes 関数は kB, MB, GB, TB 等の単位で指定された文字列からバイト数を返します。pg_size_pretty 関数と逆の動作です。数字と単位の間にはスペースを入れることができます。

例 32　pg_size_bytes 関数

```
postgres=> SELECT pg_size_bytes ('1.2 TB') ;
pg_size_bytes
---------------
 1319413953331
(1 row)
```

□　pg_blocking_pids 関数

関数に指定されたプロセス ID のプロセスをブロックしているプロセスの一覧を取得できます。

例 33 pg_blocking_pids 関数

```
postgres=> SELECT pg_blocking_pids(2953) ;
 pg_blocking_pids
------------------
 {2950}
(1 row)
```

□　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。

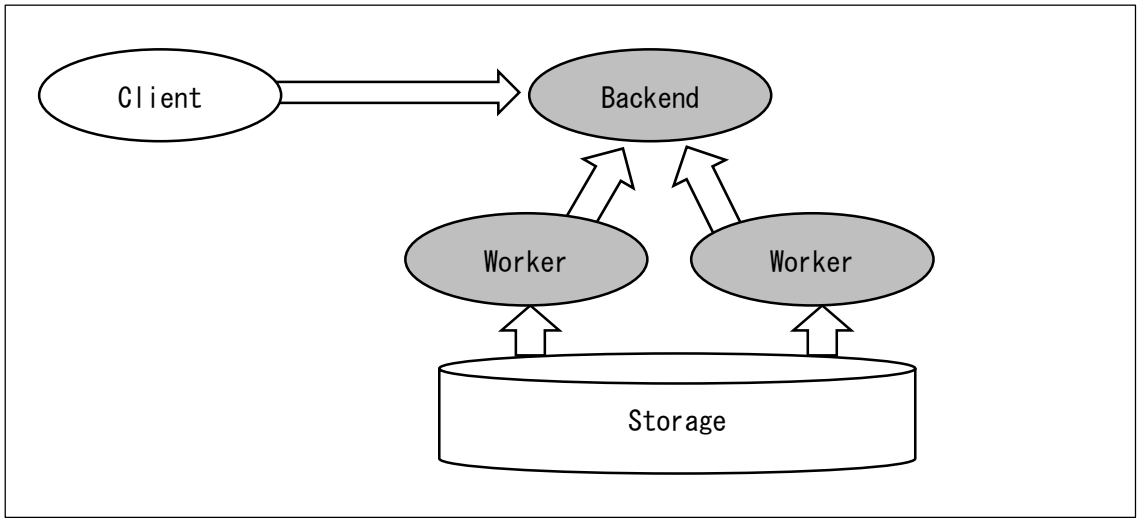
- pg_notification_queue_usage
- acosd / asind / atan2d / atand / cosd / cotd / sind / tand

3.5 Parallel Seq Scan

3.5.1 概要

従来の PostgreSQL では、SQL 文の実行は接続を受け付けたバックエンド・プロセスがすべて実行していました。PostgreSQL では複数のワーカー・プロセスによる並列処理を行うことができるようになりました。

図 1 Parallel Seq Scan / Parallel Aggregate



並列処理が行われるのは Seq Scan および Aggregate のみで、並列数はテーブルのサイズに依存します。並列処理を行うプロセスは Background Worker の仕組みを利用します。並列数の最大値はパラメーターmax_parallel_degree または max_worker_processes のいずれか小さい方で決定されます。パラメーターmax_parallel_degree は一般ユーザーがセッション単位に変更できます。

表 22 関連するパラメーター

パラメーター名	説明（context）	デフォルト値
max_parallel_degree	並列度の最大値（user）	0
parallel_setup_cost	並列処理の開始コスト（user）	1000
parallel_tuple_cost	並列処理のタプル処理コスト（user）	0.1
max_worker_processes	ワーカー・プロセスの最大値（postmaster）	8
force_parallel_mode	並列処理を強制（user）	off

□ パラメーターforce_parallel_mode

並列処理は通常のシリアル処理よりもコストが低いとみなされる場合にのみ実行されます。パラメーターforce_parallel_mode を on に指定すると、並列処理を強制することができます（同様の動作で再帰テスト用に設定値 regress を指定することができます)。ただし並列処理が行われるのはパラメーターmax_parallel_degree が 1 以上の場合に限ります。

□ テーブル・オプション

テーブル・オプション parallel_degree は並列度の上限をテーブル単位に決定することができます。設定値を 0 に指定すると並列処理が禁止されます。設定されない場合はセッションのパラメーターmax_parallel_degree が上限になります。

parallel_degree の設定値を大きくしても実際の並列度の上限は max_parallel_degree を超えることはできません。

例 34 並列処理の実行計画

```
postgres=> ALTER TABLE data1 SET (parallel_degree = 2) ;
ALTER TABLE
postgres=> ¥d+ data1
                                Table "public.data1"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 c1     | numeric               |           | main     |              |
 c2     | character varying(10) |           | extended |              |
Options: parallel_degree=2
```

3.5.2 実行計画

下記は並列処理を行う SELECT 文の実行計画出力例です。大規模なテーブルの COUNT 処理を 3 並列で処理しています。

例 35 並列処理の実行計画

```
postgres=> SET max_parallel_degree = 10 ;
SET
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT COUNT(*) FROM data1 ;
                                  QUERY PLAN
------------------------------------------------------------------------------
Finalize Aggregate  (cost=29314.09..29314.10 rows=1 width=8)
        (actual time=662.055..662.055 rows=1 loops=1)
   Output: pg_catalog.count(*)
   ->  Gather  (cost=29313.77..29314.08 rows=3 width=8)
         (actual time=654.818..662.043 rows=4 loops=1)
          Output: (count(*))
          Workers Planned: 3
          Workers Launched: 3
          ->  Partial Aggregate  (cost=28313.77..28313.78 rows=1 width=8)
                 (actual time=640.330..640.331 rows=1 loops=4)
                Output: count(*)
                Worker 0: actual time=643.386..643.386 rows=1 loops=1
                Worker 1: actual time=645.587..645.588 rows=1 loops=1
                Worker 2: actual time=618.493..618.494 rows=1 loops=1
                ->  Parallel Seq Scan on public.data1  (cost=0.00..25894.42
        rows=967742 width=0) (actual time=0.033..337.848 rows=750000 loops=4)
                      Output: c1, c2
                      Worker 0: actual time=0.037..295.732 rows=652865 loops=1
                      Worker 1: actual time=0.026..415.235 rows=772230 loops=1
                      Worker 2: actual time=0.042..359.622 rows=620305 loops=1
 Planning time: 0.130 ms
 Execution time: 706.955 ms
(18 rows)
```

並列処理を実行するにあたり、EXPLAIN 文を実行すると以下のような実行計画が表示される可能性があります。

表 23　EXPLAIN 文の出力

実行計画	説明	出力される構文
Parallel Seq Scan	並列検索処理	全て
Partial Aggregate	ワーカー・プロセスが行う集計処理	全て
Gather	ワーカー・プロセスを集約する処理	全て
Finalize Aggregate	最終的な集約処理	全て
Workers Planned:	計画されたワーカー数	全て
Workers Launched:	実際に実行されたワーカー数	ANALYZE
Worker N (N=0,1,…)	各ワーカーの処理時間等	ANALYZE, VERBOSE
Single Copy	単一プロセスで実行する処理	すべて

3.5.3 並列処理と関数

並列処理実行時に使用できる関数と使用できない関数があります。pg_proc カタログの proparallel 列が'u'になっている関数（PARALLEL UNSAFE）が SQL 文内に使用されていると、並列処理は行われません。以下の表は主な PARALLEL UNSAFE 標準関数です。

表 24　主な PARALLEL UNSAFE 標準関数

分類	関数例
JSON 関連	json_populate_record, json_populate_recordset, jsonb_insert, jsonb_set
シーケンス関連	nextval, currval, setval, lastval
Large Object 関連	lo_*, loread, lowrite
文字コード変換関連	ascii_to_*, big5_to_*, euc_*_to_*, gb18030_to_utf8, gbk_to_utf8, iso8859_*, iso_to_*, johab_to_utf8, koi8r_to_*, koi8u_to_utf8, latin*_to_*, mic_to_*, shift_jis_*_to_*, sjis_to_*, uhc_to_utf8, utf8_to_*, win1250_to_latin2
レプリケーション関連	pg_create_*_slot, pg_drop_*_slot, pg_logical_*, pg_replication_*
その他	make_interval, parse_ident, pg_extension_config_dump, pg_*_backup, set_config, ts_debug, txid_current, query_to_xml*

下記の例では、WHERE 句の条件部分のみ異なる 2 つの SQL 文を実行しています。WHERE 句にリテラルを指定した SELECT 文は並列処理が行われますが、シーケンス操作関数の currval を指定した SELECT 文はシリアルに実行されます。

例 36 PARALLEL UNSAFE 関数の使用による実行計画の違い

```
postgres=> EXPLAIN SELECT COUNT(*) FROM data1 WHERE c1=10 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=29314.08..29314.09 rows=1 width=8)
   ->  Gather  (cost=1000.00..29314.07 rows=3 width=0)
         Workers Planned: 3
         ->  Parallel Seq Scan on data1  (cost=0.00..28313.78 rows=1 width=0)
               Filter: (c1 = '10'::numeric)
(5 rows)

postgres=> EXPLAIN SELECT COUNT(*) FROM data1 WHERE c1=currval('seq1') ;
                        QUERY PLAN
-----------------------------------------------------------
 Aggregate  (cost=68717.01..68717.02 rows=1 width=8)
   ->  Seq Scan on data1  (cost=0.00..68717.00 rows=3 width=0)
         Filter: (c1 = (currval('seq1'::regclass))::numeric)
(3 rows)
```

□ ユーザー定義関数と PARALLEL SAFE

ユーザー定義関数に対して並列処理を行うことができるかどうかを示すため、CREATE FUNCTION 文または ALTER FUNCTION 文に PARALLEL SAFE 句または PARALLEL UNSAFE 句を使用することができます。デフォルトは PARALLEL UNSAFE です。

例 37 ユーザー定義関数と PARALLEL SAFE

```
postgres=> CREATE FUNCTION add(integer, integer) RETURNS integer
postgres->    AS 'select $1 + $2;'
postgres->    LANGUAGE SQL IMMUTABLE RETURNS NULL ON NULL INPUT ;
CREATE FUNCTION
postgres=> SELECT proname, proparallel FROM pg_proc WHERE proname = 'add' ;
 proname | proparallel
---------+-------------
 add     | u
(1 row)
postgres=> ALTER FUNCTION add(integer, integer) PARALLEL SAFE ;
ALTER FUNCTION
postgres=> SELECT proname, proparallel FROM pg_proc WHERE proname='add' ;
 proname | proparallel
---------+-------------
 add     | s
(1 row)
```

3.5.4 並列度の計算

並列度の計算は検索対象のテーブルのサイズが 3,000 ブロック、9,000 ブロック、27,000 ブロックを超えると増加します。以下上限を 3 倍しながら、パラメーターmax_parallel_degree またはパラメーターmax_worker_processes を超えない範囲で決定されます。実際の計算処理はソースコード src/backend/optimizer/path/allpaths.c 内の create_parallel_paths 関数内で計算されています。

例 38 create_parallel_paths 関数の一部

```
        int parallel_threshold = 1000;

        /*
         * If this relation is too small to be worth a parallel scan, just
         * In that case, we want to generate a parallel path here anyway.  It
         * might not be worthwhile just for this relation, but when combined
         * with all of its inheritance siblings it may well pay off.
         */
        if (rel->pages < parallel_threshold &&
                rel->reloptkind == RELOPT_BASEREL)
           return;

        /*
         * Limit the degree of parallelism logarithmically based on the size
         * of the relation.  This probably needs to be a good deal more
         * sophisticated, but we need something here for now.
         */
         while (rel->pages > parallel_threshold * 3 &&
                 parallel_degree < max_parallel_degree)
         {
               parallel_degree++;
               parallel_threshold *= 3;
               if (parallel_threshold >= PG_INT32_MAX / 3)
                    break;
         }
```

3.6 待機状態のモニタリング

PostgreSQL インスタンス内で発生している待機イベントの情報を入手できるようになりました。pg_stat_activity カタログから waiting 列が削除され、wait_event_type 列と wait_event 列が追加されました。wait_event_type 列には以下の値が格納されます。

表 25　wait_event_type 列の値

列値	説明
LWLockNamed	特定の軽量ロックによる待機
LWLockTranche	グループに対する軽量ロックによる待機
Lock	重量ロックによる待機（LOCK TABLE 文等）
BufferPin	バッファに対する PIN 待ち

詳しくは以下の URL を参照してください。

http://www.postgresql.org/docs/9.6/static/monitoring-stats.html

例 39 LOCK TABLE IN EXCLUSIVE 文同士の待機

```
postgres=> SELECT pid, wait_event_type, wait_event FROM pg_stat_activity
        WHERE pid=4070 ;
 pid  | wait_event_type | wait_event
------+-----------------+------------
 4070 | Lock            | relation
(1 row)
```

例 40 SELECT FOR UPDATE 文と UPDATE 文による待機

```
postgres=> SELECT pid, wait_event_type, wait_event FROM pg_stat_activity
        WHERE pid=4070 ;
pid  | wait_event_type |  wait_event
------+-----------------+---------------
 4070 | Lock            | transactionid
 (1 row)
```

3.7 FOREIGN DATA WRAPPER の拡張

外部オブジェクトに対するアクセス手段を提供する FOREIGN DATA WRAPPER が拡張されました。

3.7.1 Sort Push-down

従来のバージョンでは FOREIGN TABLE に対するソート処理は外部システムからのデータ転送後にローカル・インスタンスで実行されていました。PostgreSQL 9.6 では外部オブジェクトに ORDER BY 句を送信することができるようになりました。以下の例では 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 ORDER BY 句付きの検索を実行しています。

例 41 Sort Push-down

```
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT * FROM table1 ORDER BY 1 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
Foreign Scan on public.table1  (cost=100.00..139.87 rows=871 width=70)
        (actual time=0.986..7109.985 rows=1000000 loops=1)
   Output: c1, c2
   Remote SQL: SELECT c1, c2 FROM public.table1 ORDER BY c1 ASC NULLS LAST
 Planning time: 0.130 ms
 Execution time: 7201.854 ms
(5 rows)
```

FOREIGN TABLE である table1 テーブルを検索する SQL 文がリモート・インスタンス上で ORDER BY 句付きで実行されていることがわかります。

3.7.2 Direct Modify

従来のバージョンでは FOREIGN TABLE に対する DELETE 文や UPDATE 文は、SELECT FOR UPDATE 文によるカーソルを作成し、カーソル内のレコード単位で更新処理を行っていました。PostgreSQL 9.6 では更新 DML を直接実行できるようになりました。

下記の例では PostgreSQL 9.5 と PostgreSQL 9.6 で FOREIGN TABLE に対して DELETE 文を実行しています。PostgreSQL 9.5 では SELECT FOR UPDATE 文が実行されていますが、PostgreSQL 9.6 では DELETE 文になっています。

例 42 PostgreSQL 9.5 の DELETE 文

```
postgres=> EXPLAIN (ANALYZE, VERBOSE) DELETE FROM data1 WHERE c1=100 ;
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 Delete on public.data1  (cost=100.00..144.40 rows=14 width=6)
        (actual time=582.328..582.328 rows=0 loops=1)
   Remote SQL: DELETE FROM public.data1 WHERE ctid = $1
   ->  Foreign Scan on public.data1  (cost=100.00..144.40 rows=14 width=6)
                (actual time=527.345..527.347 rows=1 loops=1)
         Output: ctid
         Remote SQL: SELECT ctid FROM public.data1 WHERE ((c1 = 100::numeric))
                FOR UPDATE
 Planning time: 0.746 ms
 Execution time: 583.628 ms
(7 rows)
```

例 43 PostgreSQL 9.6 の DELETE 文

```
postgres=> EXPLAIN (ANALYZE, VERBOSE) DELETE FROM data1 WHERE c1=100 ;
                              QUERY PLAN
-------------------------------------------------------------------------------------------
 Delete on public.data1  (cost=100.00..144.40 rows=14 width=6)
        (actual time=1.019..1.019 rows=0 loops=1)
   ->  Foreign Delete on public.data1  (cost=100.00..144.40 rows=14 width=6)
                (actual time=1.016..1.016 rows=1 loops=1)
         Remote SQL: DELETE FROM public.data1 WHERE ((c1 = 100::numeric))
 Planning time: 0.222 ms
 Execution time: 1.414 ms
(5 rows)
```

3.7.3 Join Push-down

同一 FOREIGN SERVER 上のテーブル同士を結合する処理をリモート・インスタンス上で行うことができるようになりました。

例 44 Join Push-down

```
postgres=> CREATE FOREIGN TABLE foreign1(c1 numeric, c2 varchar(10))
        SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=> CREATE FOREIGN TABLE foreign2(c1 numeric, c2 varchar(10))
        SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=>
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT COUNT(*) FROM
        foreign1 f1, foreign2 f2 WHERE f1.c1 = f2.c1 AND f1.c1 =  100 ;
                                QUERY PLAN
------------------------------------------------------------------------------
 Aggregate  (cost=35912.03..35912.04 rows=1 width=8)
        (actual time=2.558..2.558 rows=1 loops=1)
   Output: count(*)
   ->  Foreign Scan  (cost=100.00..35912.03 rows=1 width=0)
        (actual time=2.549..2.550 rows=1 loops=1)
         Relations: (public.foreign1 f1) INNER JOIN (public.foreign2 f2)
         Remote SQL: SELECT NULL FROM (public.foreign1 r1 INNER
                JOIN public.foreign2 r2 ON (((r2.c1 = 100::numeric)) AND
                ((r1.c1 = 100::numeric))))
 Planning time: 0.284 ms
 Execution time: 3.822 ms
(7 rows)
```

3.8 複数同期レプリケーション

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。プライマリ・インスタンスのパラメーターsynchronous_standby_names に同期レプリケーションを行うインスタンス数を指定します。

構文

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

num_sync には同期レプリケーションを行うインスタンス数を 1 以上の整数で指定します。0 以下の値を指定した場合や省略した場合はインスタンスを起動できません。指定されたインスタンス数が確保できない場合、プライマリ・インスタンスの更新トランザクションは停止します。下記の例では同期レプリケーションが可能なインスタンスが 3 個あり、2 インスタンスに同期レプリケーションを行います。

例 45 複数同期レプリケーション

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

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

パラメーターsynchronous_standby_names には旧バージョンと同じ形式でも記述できます。その場合は同期レプリケーションされるインスタンスは 1 個になります。

3.9 セキュリティ

3.9.1 デフォルト・ロール

pg_signal_backend ロールが標準で作成されます。このロールはバックエンド・プロセスに対するシグナルの送信を許可します。

3.9.2 名前空間

pg_から始まるロール名（ユーザー名）は予約済みとみなされて作成できなくなりました。

例 46 pg_から始まるロール名

```
postgres=# CREATE ROLE pg_test1 ;
ERROR:  role name "pg_test1" is reserved
DETAIL:  Role names starting with "pg_" are reserved.

$ initdb --username=pg_admin data
initdb: superuser name "pg_admin" is disallowed; role names cannot begin with
"pg_"
```

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
  http://www.postgresql.org/docs/9.6/static/release-9-6.html
- What's new in PostgreSQL 9.6
  https://wiki.postgresql.org/wiki/What%27s_new_in_PostgreSQL_9.6
- Commitfests
  https://commitfest.postgresql.org/
- PostgreSQL 9.6 Beta Manual
  http://www.postgresql.org/docs/9.6/static/index.html
- GitHub
  https://github.com/postgres/postgres
- PostgreSQL 9.6 Beta 1 のアナウンス
  http://www.postgresql.org/about/news/1668/
- NewIn96
  https://wiki.postgresql.org/wiki/NewIn96
- PostgreSQL 9.6 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/

Hewlett Packard
Enterprise

変更履歴

変更履歴

版	日付	作成者	説明
0.1	2016/05/20	篠田典良	社内レビュー版作成 レビュー担当（敬称略）： 高橋智雄 北山貴広 竹島彰子 池田拓磨
1.0	2015/05/30	篠田典良	公開版を作成

以上

Hewlett Packard
Enterprise