2014 年 7 月 2 日

PostgreSQL 9.4 新機能検証結果

Version 1.0

日本ヒューレット・パッカード株式会社
テクノロジーコンサルティング事業統括
デリバリー統括本部 オープンソース部

目次

用語集

表 1　略語／用語

略語/用語	説明
PL/pgSQL	PostgreSQL のストアド・プロシージャ記述言語のひとつ PL/SQL とある程度互換性がある。
PostgreSQL	オープンソース・データベース製品
psql	PostgreSQL に付属する SQL 文を実行するためのユーティリティ
WAL	PostgreSQL のトランザクション・ログ（Write Ahead Logging）
システムカタログ	PostgreSQL データベース全体のメタ情報を格納している領域
タプル	テーブル内のレコードを示す
データベース・クラスタ	PostgreSQL データベース全体の管理情報が格納されているディレクトリ
リレーション	テーブルをリレーションと呼ぶ場合がある
${PGDATA}	データベース・クラスタを示す環境変数が展開される

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 9.4 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

本文書の適用範囲は PostgreSQL 9.3 と PostgreSQL 9.4 の差分を記載しています。

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 2 対象バージョン

種別	バージョン	備考
データベース製品	PostgreSQL 9.3.4 (比較対象) PostgreSQL 9.4 beta 1 (2014/5/11 9:23 pm)	
オペレーティング・システム	Red Hat Enterprise Linux 6 Update 4 (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は責任を負いません。

2. 新機能概要

PostgreSQL 9.4 は多くの新機能や改善が行われました。大規模な変更を伴う新機能は少なく、既存機能の改善が主な内容です。以下に主な変更点や改善点を記述します。

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。

- WAL バッファに対する同時書き込み
- ウィンドウ関数のパフォーマンス
- 数値型の計算（SUM 関数、AVG 関数、STDDEV 関数、VARIANCE 関数等）
- COPY 文で使用する NEXTVAL 関数
- 同一セッション内で複数シーケンスに対するアクセス
- 4 GB を超えるメモリーを使用する B-tree インデックスの作成

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章番号です。

- pg_recvlogical コマンドの追加（3.2.4）
- Huge Page 対応（3.1.1）
- カタログの追加（3.1.2）
- ワーカー・プロセスの上限指定（3.1.5）
- セッション起動時のライブラリ・ロード（3.1.6）
- 動的パラメータ・ファイル（3.5）
- 集計関数の追加（3.6.8）
- jsonb 型、line 型、LSN 型の追加／拡張（3.7）
- 自動 Vacuum のメモリー設定（3.9）
- pg_prewarm 関数（3.12）
- WAL log hint ビット
- WITH ORDINALITY 句の追加
- WITHIN GROUP 句の追加
- MSVC Installer の使用

2.3 機能の改善

以下の機能が改善されました（）内はより詳細が記載された章番号です。

- vacuumdb, pg_basebackup コマンドのパラメータ追加（3.2.3）
- ストリーミング・レプリケーションの拡張（3.4）
- COPY 文の FORCE_NULL FORCE_NOT_NULL 句（3.6.1）
- EXPLAIN 文に推定時間の表示（3.6.2）
- シーケンス・キャッシュの破棄（3.6.3）
- マテリアライズド・ビューの拡張（3.6.4）
- 集計関数の条件設定（3.5.5）
- オブジェクトの移動（3.6.7）
- WITH CHECK OPTION 付きビュー（3.6.1）
- 自動更新可能ビューの拡張（3.6.2）
- 外部テーブルに対する拡張（3.10）
- チェックサムの確認パラメータ（3.11）
- auto explain モジュールの拡張（3.13）
- PL/PGSQL のスタックトレース（3.14）
- recovey_target=immediate の追加
- 表領域作成と同時に属性設定
- GIN 索引のサイズ削減

2.4 動作の変更

- SET LOCAL / TRANSACTION 文の動作変更（3.6.6）

その他の改善点は、PostgreSQL 9.4 Documentation Appendix E. Release Notes（http://www.postgresql.org/docs/devel/static/release.html）に記載されています。

2.5 その他

INSERT 文でキー重複の場合は更新する INSERT ON DUPLICATE KEY LOCK FOR UPDATE 文（Oracle Database の MERGE 文と同等）が追加されるという情報もありましたが、このリリースでは含まれていません。

3. 新機能詳細

3.1 アーキテクチャの変更

3.1.1 Huge Page 対応

大規模メモリー環境向けに Linux Huge Page に対応しました。Huge Page への対応はパラメータ huge_pages により決定されます。Huge Page を使用する場合のページ・サイズは 2 MB です（2 * 1024 * 1024）。 Huge Page を使用する場合、確保される共有メモリーは 2 MB の倍数に拡大され、mmap システム・コールに MAP_HUGETLB が指定されます。

□ パラメータ設定

PostgreSQL が仕様する共有メモリーに Huge Page を使用するには、パラメータ huge_pages を設定します。

表 3 パラメータ huge_pages に指定できる値

パラメータ値	説明	備考
on	Huge Page を使用する	
off	Huge Page を使用しない	
try	Huge Page の使用を試し、使えれば使う	デフォルト値

デフォルト値の try を指定すると、プラットフォームが Huge Page をサポートしている場合は使用し、サポートしていない場合は使用しません。このパラメータを on に指定すると強制的に Huge Page を使用します。プラットフォームが Huge Page をサポートしていない場合（ビルド時に）、pg_ctl コマンドは以下のエラー・メッセージを出力してインスタンスは起動できません。

```
FATAL:  huge pages not supported on this platform
```

□ Huge Page の設定方法

Linux 環境で Huge Page を有効にするにはカーネル・パラメータ vm.nr_hugepages に 2MB 単位のページ数の最大値を指定します。このパラメータのデフォルト値は 0 です。使用中の Huge Page の情報は、/proc/meminfo ファイルを参照します。

例 1 Huge Page の設定

```
# sysctl -a | grep nr_hugepages
vm.nr_hugepages = 0
vm.nr_hugepages_mempolicy = 0
# sysctl -w vm.nr_hugepages = 1000
vm.nr_hugepages = 1000
# grep ^Huge /proc/meminfo
HugePages_Total:    1000
HugePages_Free:     1000
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
#
```

パラメータ huge_pages=on を指定した環境でインスタンス起動時に必要なページが確保できない場合、以下のエラーが発生してインスタンスは起動できません。

例 2 Huge Page ページ不足エラー

```
$ pg_ctl -D data start
server starting
FATAL:  could not map anonymous shared memory: Cannot allocate memory
HINT:  This error usually means that PostgreSQL's request for a shared memory segment exceeded available memory, swap space or huge pages. To reduce the request size (currently  148324352 bytes), reduce PostgreSQL's shared memory usage, perhaps by reducing shared_buffers or max_connections.
```

Red Hat Enterprise Linux 6.4 では、MAP_HUGETLB マクロが欠落しているため、ソースコードからビルドすると Huge Pages 非対応のバイナリが作成されます。バイナリ作成時に、/usr/include/bits/mman.h 内に以下の行があるか確認してください。

```
# define MAP_HUGETLB      0x40000          /* Create huge page mapping.   */
```

3.1.2 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　変更されたシステム・カタログ

カタログ名	説明	備考
pg_replication_slots	スロット情報	追加
pg_stat_archiver	アーカイブ処理情報	追加

□ pg_replication_slots ビュー

ストリーミング・レプリケーションで使用されるスロットの情報を参照できます。

表 5　pg_replication_slots ビュー

列名	データ型	説明	備考
slot_name	name	スロット名	
plugin	name	プラグイン名	
slot_type	text	タイプ（physical | logical）	
datoid	oid	論理レプリケーションに対応するデータベース ID	
database	name	論理レプリケーションで使用するデータベース名	
active	boolean	アクティブかどうか（t | f）	
xmin	xid	スロットが必要とする最も古いデータベース XID	
catalog_xmin	xid	スロットが必要とする最も古いシステムカタログ XID	
restart_lsn	pg_lsn	コンシューマーが必要とする最も古い LSN	

□ pg_stat_archiver ビュー

アーカイブログが作成された回数情報を参照できます。このビューには 1 レコードだけ格納されています。このビューの値は関数「pg_stat_reset_shared('archiver')」が実行されるとリセットされます。

表 6　pg_stat_archiver ビュー

列名	データ型	説明	備考
archived_count	bigint	アーカイブ出力成功回数	
last_archived_wal	text	最後に出力が成功した WAL ファイル	
last_archived_time	timestamp with timezone	最後に出力した時刻	
failed_count	bigint	アーカイブ出力失敗回数	
last_failed_wal	text	最後に出力が失敗した WAL ファイル	
last_failed_time	timestamp with timezone	最後に失敗した時刻	
stats_reset	timestamp with timezone	統計情報のリセット時刻	

3.1.3 動的共有メモリー

PostgreSQL 9.2 以前では、プロセス間のメモリー共有機能を System V 共有メモリー（shmget システムコール）を使って実装していました。PostgreSQL 9.3 では、System V 共有メモリー領域は縮小し、ほとんどの領域を mmap システム・コールで代替しました。PostgreSQL 9.4 では固定領域以外に必要に応じて動的に作成される動的共有メモリー機能が提供されています。この機能は「src/backend/storage/ipc/dsm_impl.c」に実装されています。どのような場合に動的共有メモリーが使用されるかは未検証です。

パラメータ dynamic_shared_memory_type に動的共有メモリーのタイプを指定します。動的共有メモリー作成の状況はパラメータ log_min_messages を debug2 に設定すると表示されます。

表 7　パラメータ dynamic_shared_memory_type の指定

値	説明	備考
posix	POSIX 共有メモリー（shm_open）を使用する	Linux のデフォルト値
sysv	System V 共有メモリー（shmget）を使用する	
windows	Win32 API CreateFileMapping システムコールを使用する	Windows のデフォルト値
mmap	mmap システムコールを使用する	
none	動的共有メモリーを使用しない	

このパラメータを指定していない場合、Windows プラットフォームでは windows が選択されます。その他のプラットフォームでは shm_open システム・コールの実行がキーを変

更しながら 10 回まで試され、1 回でも成功すれば posix が、すべて失敗すれば sysv が選択されます（src/bin/initdb/initdb.c）。

3.1.4 データベース・クラスタ内の変更

データベース・クラスタには以下のファイル／ディレクトリが追加されました。

表 8　データベース・クラスタの変更

名前	種類	説明	備考
pg_dynshmem	ディレクトリ	動的共有メモリーの管理情報	
pg_llog	ディレクトリ	論理レプリケーション用？	
pg_replslot	ディレクトリ	レプリケーション・スロット用	
postgresql.auto.conf	ファイル	動的パラメータ変更ファイル	

3.1.5 ワーカー・プロセスの最大数

ワーカー・プロセスの最大数を決定するパラメータ max_worker_processes が追加されました。このパラメータのデフォルト値は 8 です。指定できる値は、以下の式で示された値以下です。

「max_connections + autovacuum_max_workers + 1 + max_worker_processes > MAX_BACKENDS（0x7ffff=8,388,607）」

3.1.6 セッション起動時のライブラリ

パラメータ session_preload_libraries が追加されました。インスタンスに対する接続時にロードされるライブラリ名を指定することができます。この機能により、PostgreSQL を拡張する際にインスタンスの停止を行う必要が無くなります。

3.2 ユーティリティ

ユーティリティに対して追加された主な機能を説明します。

3.2.1 psql

対話的に SQL 文を実行する psql コマンドは以下の拡張が行われました。¥pset コマンドの実行により設定値の一覧が表示されるようになりました。

例 3 ¥pset コマンドによる一覧表示

```
postgres=> ¥pset
Border style (border) is 1.
Target width (columns) unset.
Expanded display (expanded) is off.
Field separator (fieldsep) is "|".
Default footer (footer) is on.
Output format (format) is aligned.
Line style (linestyle) is ascii.
Null display (null) is "".
Locale-adjusted numeric output (numericlocale) is off.
Pager (pager) is used for long output.
Record separator (recordsep) is <newline>.
Table attributes (tableattr) unset.
Title (title) unset.
Tuples only (tuples_only) is off.
```

3.2.2 pg_basebackup

pg_basebackup ユーティリティには３つのパラメータが追加されました。

□ --tablespace-mapping

PostgreSQL 9.3 までは、データベース・クラスタ以外に表領域を作成した場合、バックアップ先の表領域ディレクトリはバックアップ元と同じ位置に作成されました。PostgreSQL 9.4 ではこの制限が緩和され、--tablespace-mapping パラメータ（または-T）に「旧ディレクトリ・パス=新ディレクトリ・パス」の形式でディレクトリ名を変更することができるようになりました。

例 4 ディレクトリのマッピング

```
postgres=# CREATE TABLESPACE ts1 LOCATION '/home/postgres/ts1' ;
CREATE TABLESPACE
$ mkdir /home/postgres/backup/data
$ mkdir /home/postgres/backup/ts1
$ pg_basebackup -D backup/data -h localhost -U postgres -x
  --tablespace-mapping=/home/postgres/ts1=/home/postgres/backup/ts1
$
$ ls -l backup/data/pg_tblspc/
total 0
lrwxrwxrwx. 1 postgres postgres 22 May 13 17:05 41022 -> /home/postgres/backup/ts1
```

上記の例ではローカルホスト内で、データベース・クラスタを/home/postgres/data から /home/postgres/base/data へコピーしています。その際に表領域 ts1 のパスを変更しています。複数の表領域をマップする場合は、--tablespace-mapping パラメータを複数回記述します。

□ --max-rate

pg_basebackup コマンドは標準ではデータの転送量を制限しません。このため稼働中のシステムに対して過大な読み取り I/O を発生させる可能性があります。--max-rate パラメータ（または-r）はデータの転送レートを制限する場合に指定します。デフォルト値は 0 で制限は行われません。転送レートはキロバイト単位で指定します。指定できる値は 32（32KB）より大きく 1048576（1GB）よりも小さい値を指定する必要があります。検証の結果、-r にかなり大きい値を指定しても実行時間は長くなる傾向があります。

□ --xlogdir

WAL 書込みディレクトリを指定します。書き込み先の${PGDATA}/pg_xlog はシンボリック・リンクとして--xlogdir に指定されたディレクトリを示します。--xlogdir パラメータには絶対パスを指定する必要があります。

3.2.3 vacuumdb

vacuumdb コマンドに --analyze-in-stages パラメータが追加されました。このパラメータは--analyze-only パラメータに似ていますが、より高速に統計情報を収集することができます。

3.2.4 pg_recvlogical

pg_recvlogical コマンドが PostgreSQL 9.4 で追加されました。論理レプリケーションを制御します。pg_receivexlog コマンドの論理レプリケーション版です。このユーティリティは以下のパラメータを持ちます。

表 9　パラメーター一覧

パラメータ	説明	備考
--create	レプリケーション・スロットの作成	
--start	レプリケーションの開始	
--drop	レプリケーション・スロットの削除	
--username	接続ユーザー名	-U
--dbname	接続データベース名	-d
--host	接続ホスト名	-h
--port	接続ポート番号	-p
--no-password	パスワード無し	-w
--password	パスワード指定	-W
--file	書込みファイル名	-f
--no-loop	再接続を行わない	-n
--option	オプションの指定	-o
--fsync-interval	同期書込み間隔	-F
--plugin	スロット作成時に使用するプラグイン名	-p
--status-interval	ステータス・パケット送信間隔（秒）	-s
--slot	操作を行うスロット名	-S
--startpos	レプリケーション開始 LSN	-I
--verbose	詳細出力モード	-v
--version	バージョン情報の表示	-V
--help	コマンドライン・パラメータの表示	-?

3.3 パラメータの変更

PostgreSQL 9.4 では以下のパラメータが変更されました。

3.3.1 追加されたパラメータ

以下のパラメータが追加されました。

表 10　追加されたパラメータ

パラメータ	説明	デフォルト値
autovacuum_work_mem	自動 Vacuum 時の一時メモリー量	-1
data_checksums[1]	データ領域のチェックサム有効	off
dynamic_shared_memory_type	動的共有メモリーの使用	posix
huge_pages	Huge Page を使用するか	try
max_replication_slots	最大レプリケーション・スロット数	0
max_worker_processes	ワーカー・プロセスの最大数	8
session_preload_libraries	セッション追加時のライブラリ	''
ssl_ecdh_curve	ECDH キーの CURVE 名	none
ssl_prefer_server_ciphers	SSL Cpher Preference を使うか	on
wal_log_hints	WAL に追加情報を付加	off
xloginsert_locks	詳細不明	8

3.3.2 選択肢が変更されたパラメータ

以下のパラメータは指定できる値が変更されました。

表 11　選択肢が変更されたパラメータ

パラメータ	説明	変更内容
wal_level	WAL 出力レベル	logical が指定可能になった

3.3.3 デフォルト値が変更されたパラメータ

以下のパラメータはデフォルト値が変更されました。メモリー関連パラメータの初期値が 4 倍になりました。

[1] PostgreSQL 9.3.4 で追加されました。本パラメータは Read Only です。

表 12　デフォルト値が変更されたパラメータ

パラメータ	PostgreSQL 9.3	PostgreSQL 9.4
effective_cache_size	128MB	4GB[2]
maintenance_work_mem	16MB	64MB
server_version	9.3	9.4beta1
server_version_num	90300	90400
ssl_ciphers	DEFAULT:!LOW:!EXP:!MD5:@STRENGTH	none
work_mem	1MB	4MB

3.3.4 廃止されたパラメータ

以下のパラメータは廃止され、分割または名前が変更されています。

表 13　廃止されたパラメータ

パラメータ	代替
krb_srvname	なし

[2] 初期開発時点では 512 MB でしたが、Beta 1 で大きく変更されました。

3.4 レプリケーションの機能追加

PostgreSQL 9.4 ではストリーミング・レプリケーションの構成が変更されました。

3.4.1 スロットとは

PostgreSQL 9.4 のストリーミング・レプリケーションは、マスター側にスロットと呼ばれるオブジェクトを作成し、スレーブ・データベースはスロット名を参照することで実現されます。PostgreSQL 9.3 までのレプリケーションは、スタンバイ・インスタンスが停止している場合でも WAL ファイルはパラメータ wal_keep_segments で指定された個数までしかファイルを保持しませんでした。スロットを使うことでスレーブに必要な WAL ファイルが自動的に管理され、スレーブが受け取らない限りマスターの WAL が削除されないように変更されました。基本的なレプリケーションの構造は変更が無いため、WAL sender 用パラメータや pg_hba.conf ファイルの設定等は引き続き必要です。

□ スロットの管理

スロットは以下の関数で管理を行います。使用中のスロットは削除できません。従来から利用できるストリーミング・レプリケーションは Physical Replication と呼ばれます。

スロット作成関数

```
pg_create_physical_replication_slot(スロット名)
pg_create_logical_replication_slot(スロット名, プラグイン名)
```

スロット削除関数

```
pg_drop_replication_slot(スロット名)
```

クラスタに作成できるスロット数の最大値はパラメータ max_replication_slots で指定します。このパラメータのデフォルト値は 0 であるため、レプリケーションを行う場合には変更する必要があります。インスタンス起動時には、パラメータ max_replication_slots で指定された値を元に共有メモリー上にレプリケーション関連の情報が展開されます。

作成したスロットの情報は pg_replication_slots ビューから確認することができます。

例 5 スロットの作成と確認

```
postgres=# SELECT pg_create_physical_replication_slot('slot_1')  ;
 pg_create_physical_replication_slot
-------------------------------------
 (slot_1,)
(1 row)
postgres=# SELECT * FROM pg_replication_slots ;
 slot_name | plugin | slot_type | datoid | database | active | xmin | catalog_xmin | restart_lsn
-----------+--------+-----------+--------+----------+--------+------+--------------+-------------
 slot_1    |        | physical  |        |          | f      |      |              |
(1 row)
```

マスター側で作成したスロットは、スレーブ側の recovery.conf から参照します。

例 6 recovery.conf ファイル内のスロット参照

```
primary_slotname = 'slot_1'
primary_conninfo = 'host=hostp port=5433 application_name=prim5433'
standby_mode = on
```

現状の実装では、スレーブ側に primary_slotname の指定が存在しない場合でもレプリケーションは成功します。primary_slotname が記述されているにも関わらずプライマリ側でスロットが作成されていないと以下のエラーが発生します。

```
FATAL:  could not start WAL streaming: ERROR:  replication slot "slot_1" does not exist
```

スロットが見つからない場合、レプリケーションはできませんが、スレーブ側のインスタンスは起動します。

レプリケーションが成功するとマスター側の pg_stat_replication ビュー、pg_replication_slots ビューは以下の表示になります。

例 7 レプリケーション状況の確認

```
postgres=# SELECT * FROM pg_stat_replication ;
  pid  | usesysid | usename | application_name | client_addr | client_hostname | client_port |
 backend_start         | backend_xmin |   state   | sent_location | write_location | flush_location |
replay_location | sync_priority | sync_state
-------+----------+---------+------------------+-------------+-----------------+-------------+--------
----------------------+-------------+-----------+---------------+----------------+----------------+-
---------------+---------------+------------
 18481 |       10 | postgres | stdby1           | 127.0.0.1   |                 |       45345 | 2014-05
-16 13:04:27.344758+09 |              | streaming | 0/30C6468     | 0/30C6468      | 0/30C6468      |
0/30C6468       |             0 | async
(1 row)
postgres=# SELECT * FROM pg_replication_slots ;
slot_name | plugin | slot_type | datoid | database | active | xmin | catalog_xmin | restart_lsn
-----------+--------+-----------+--------+----------+--------+------+--------------+-------------
 slot_1    |        | physical  |        |          | t      |      |              | 0/30C6468
(1 row)
```

pg_stat_replication ビューには backend_xmin 列が追加されています。この列は pg_stat_get_activity 関数の戻り値追加によるものです。

3.4.2 スロットの実体

スロットの実体は${PGDATA}/pg_replslot ディレクトリ内に作成されたスロット名と同じ名前のディレクトリとファイルです。

例 8 スロットの実体

```
$ ls -l data/pg_replslot/
total 4
drwx------. 2 postgres postgres 4096 May  16 15:42 slot_1
$ ls -l data/pg_replslot/slot_1/
total 4
-rw-------. 1 postgres postgres 176 May  16 15:42 state
```

3.4.3 遅延レプリケーション

スレーブ・インスタンスにおいて WAL の適用を遅らせることができるようになりました。これにより、論理破壊からの復旧を早めることができるようになります。スレーブ側の recovery.conf ファイルで min_recovery_apply_delay を設定すると、指定された時間 WAL の適用時間を遅らせることができます。このパラメータのデフォルト値は 0 で、遅延は発生しません。パラメータには「ms, s, min, h, d」の単位を使うことができます。

3.4.4 論理レプリケーションのためのプラグイン

論理レプリケーションのためのプラグインを作成するフレームワークが提供されました。pg_create_logical_replication_slot 関数を使ってプラグインとスロットを対応させます。

またパラメータ wal_level には、論理レプリケーション情報を付加する値 logical が指定できるようになりました。本機能については未検証です。

3.5 パラメータファイルの変更

ALTER SYSTEM 文により、パラメータ設定が動的に永続化されるようになりました。ALTER SYSTEM 文は superuser 権限を持つユーザーのみ実行できます。

構文 1 ALTER SYSTEM 文

```
ALTER SYSTEM SET パラメータ名 = 値 | DEFAULT
```

ALTER SYSTEM 文で変更したパラメータの値は「${PGDATA}/postgresql.auto.conf」ファイルに記載されます。このファイルは手動で変更しないようにしてください。

例 9　ALTER SYSTEM 文によるパラメータ変更

```
postgres=# SHOW work_mem ;
 work_mem
----------
 4MB
(1 row)
postgres=# ALTER SYSTEM SET work_mem = '8MB' ;
ALTER SYSTEM
postgres=# SHOW work_mem ;
 work_mem
----------
 4MB
(1 row)
postgres=# ¥q
$ cat data/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by ALTER SYSTEM command.
work_mem = '8MB'
$
```

上記の例でもわかるように、ALTER SYSTEM 文はメモリー上のパラメータは変更せず、postgresql.auto.conf ファイルのみ書き換えます。このファイルはインスタンス起動時または pg_reload_conf 関数実行時に postgresql.conf ファイルが読み込まれた後解析され、値が適用されます。

ALTER SYSTEM 文のパラメータ値として DEFAULT を指定すると、postgresql.auto.conf ファイルからパラメータが削除されます。

例 10　ALTER SYSTEM 文によるパラメータ・リセット

```
postgres=# ALTER SYSTEM SET work_mem = DEFAULT ;
ALTER SYSTEM
postgres=# ¥q
$ cat data/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by ALTER SYSTEM command.
$
```

3.6 SQL 文の機能追加

3.6.1 COPY 文の拡張

COPY 文は CSV ファイル内の空文字列に対する動作を指定する FORCE_NULL 句および FORCE_NOT_NULL 句を指定できるようになりました。標準では CSV 内で値が無い列は NULL に、囲み文字のみの列は長さ 0 の文字列になりますが、この動作を変更することができます。

□ FORCE_NULL 句
　囲み文字で指定された文字を NULL 値として格納します。

□ FORCE_NOT_NULL 句
　CSV 内で値が存在しない列を長さ 0 の文字列として格納します。

表 14　FORCE_NULL、FORCE_NOT_NULL の動作

CSV 内の列値	デフォルト	FORCE_NULL	FORCE_NOT_NULL
囲み文字のみ	長さ 0 文字列	NULL	長さ 0 文字列
値なし	NULL	NULL	長さ 0 文字列

これらの機能は COPY 文の WITH 句に FORCE_NULL(列名)、FORCE_NOT_NULL(列名)として使用します。複数の列を指定する場合には、FORCE_NULL 句または FORCE_NOT_NULL 句を列の個数文記述します。

例 11　COPY 文内の FORCE_NULL、FORCE_NOT_NULL

```
postgres=# COPY copy1 FROM '/home/postgres/load.csv' WITH
       (FORMAT csv, FORCE_NULL(c2), FORCE_NOT_NULL(c3)) ;
COPY 4
postgres=#
```

3.6.2 EXPLAIN 文の拡張

実行計画を確認する EXPLAIN 文には以下の拡張が行われました。

□ 予測時間の出力

EXPLAIN 文に予測時間が出力されるようになりました。EXPLAIN 文を実行すると、実行計画の最後に「Planning time: 」行が出力されます。

例 12　PostgreSQL 9.3 の EXPLAIN 文

```
postgres=> EXPLAIN SELECT COUNT(*) FROM data1 ;
                            QUERY PLAN
-------------------------------------------------------------------
 Aggregate  (cost=208333.36..208333.37 rows=1 width=0)
   ->  Seq Scan on data1  (cost=0.00..183333.49 rows=9999949 width=0)
(2 rows)
```

例 13　PostgreSQL 9.4 の EXPLAIN 文

```
postgres=> EXPLAIN SELECT COUNT(*) FROM data1 ;
                            QUERY PLAN
-------------------------------------------------------------------
 Aggregate  (cost=17906.00..17906.01 rows=1 width=0)
   ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=0)
 Planning time: 0.115 ms
(3 rows)
```

この項目は EXPLAIN ANALYZE 文実行時も出力されます。

例 14　PostgreSQL 9.4 の EXPLAIN ANALYZE 文

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 ;
                                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=17906.00..17906.01 rows=1 width=0) (actual time=159.777..159.778 rows=1 loops=1)
   ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=0) (actual time=0.014..87.982 rows=1000000 loops=1)
 Planning time: 0.031 ms
 Execution time: 159.820 ms
```

□ GROUP BY 列の出力

EXPLAIN 文に指定された SQL に GROUP BY 句が含まれる場合、実行計画上にも対象となる列情報が出力されるようになりました。

例 15　PostgreSQL 9.3 の EXPLAIN 文

```
postgres=> EXPLAIN SELECT c2, COUNT(c1) FROM data1 GROUP BY c2 ;
                              QUERY PLAN
-----------------------------------------------------------------------
 HashAggregate  (cost=233333.23..233333.24 rows=1 width=36)
   ->  Seq Scan on data1  (cost=0.00..183333.49 rows=9999949 width=36)
(2 rows)
```

例 16　PostgreSQL 9.4 の EXPLAIN 文

```
postgres=> EXPLAIN SELECT c2, COUNT(c1) FROM data1 GROUP BY c2 ;
                              QUERY PLAN
---------------------------------------------------------------------
 HashAggregate  (cost=20406.00..20406.01 rows=1 width=14)
   Group Key: c2
   ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=14)
 Planning time: 0.194 ms
(4 rows)
```

3.6.3 SEQUENCE のキャッシュ廃棄

SEQUENCE オブジェクトは CREATE SEQUENCE 文の CACHE 句によってセッション単位にキャッシュを保持する数を指定することができます。DISCARD SEQUENCES 文を実行すると、セッション上でキャッシュされたキャッシュ値を破棄することができます。この文は異なるセッション上のシーケンス値に影響を与えません。また、個別のシーケンスに対して実行することはできません。

例 17 DISCARD SEQUENCES 文によるキャッシュのクリア

```
postgres=> CREATE SEQUENCE seq01 CACHE 10 ;
CREATE SEQUENCE
postgres=> CREATE SEQUENCE seq02 CACHE 10 ;
CREATE SEQUENCE
postgres=> SELECT NEXTVAL('seq01') ;
 nextval
---------
       1
(1 row)
postgres=> SELECT NEXTVAL('seq02') ;
 nextval
---------
       1
(1 row)
postgres=> DISCARD SEQUENCES ;
DISCARD SEQUENCES
postgres=> SELECT NEXTVAL('seq01') ;
 nextval
---------
      11
(1 row)
postgres=> SELECT NEXTVAL('seq02') ;
 nextval
---------
      11
(1 row)
```

3.6.4 マテリアライズド・ビューの拡張

マテリアライズド・ビューは PostgreSQL 9.3 から使用できるようになった検索高速化のためのオブジェクトです。マテリアライズド・ビューのリフレッシュを行うためには EXCLUSIVE ロックの取得が必要でした。強すぎるロックを縮小するために、リフレッシュ時に CONCURRENTLY を指定することができます。

ただし、CONCURRENTLY を指定するためには、マテリアライズド・ビューに対して一意インデックスの作成が必要です。

例 18 DISCARD SEQUENCES 文によるキャッシュのクリア

```
postgres=> CREATE MATERIALIZED VIEW mat1 AS SELECT COUNT(*) CT FROM data1 ;
SELECT 1
postgres=> REFRESH MATERIALIZED VIEW CONCURRENTLY mat1 WITH DATA ;
ERROR:  cannot refresh materialized view "public.mat1" concurrently
HINT:  Create a UNIQUE index with no WHERE clause on one or more columns of the
materialized view.
postgres=> CREATE UNIQUE INDEX idx1_mat1 on mat1(ct) ;
CREATE INDEX
postgres=> REFRESH MATERIALIZED VIEW CONCURRENTLY mat1 WITH DATA ;
REFRESH MATERIALIZED VIEW
postgres=>
```

3.6.5 集計関数の条件設定

SUM / AVG 等の集計関数に対して入力レコードの条件を記述することができるようになりました。以前は CASE 文との組み合わせで実現していました。集計関数の後に FILTER 句を指定します。

構文

```
集計関数 FILTER (WHERE 条件文)
```

例 19 SET 文の動作変更

```
postgres=> SELECT COUNT(*) C1, COUNT(*) FILTER (WHERE C1 % 2 = 0) FROM data1 ;
  count   |  count
----------+---------
 10000000 | 5000000
(1 row)
postgres=> SELECT SUM(C1) FILTER (WHERE C2 != 'ABC') FROM data1 ;
       sum
----------------
 50000005000000
(1 row)
```

3.6.6 SET 文の動作変更

SET LOCAL 文、SET TRANSACTION 文等、トランザクション内でのみ実行可能な SQL 文がトランザクション外で実行された場合に警告が出力されるようになりました。

例 20 SET 文の動作変更

```
postgres=> SET LOCAL search_path=public ;
WARNING:  SET LOCAL can only be used in transaction blocks
SET
postgres=> SET TRANSACTION READ ONLY ;
WARNING:  SET TRANSACTION can only be used in transaction blocks
SET
```

3.6.7 オブジェクトの移動

ALTER TABLESPACE 文にオブジェクトの移動構文が追加されました。

構文

```
ALTER TABLESPACE 表領域名 MOVE オブジェクト
 [OWNED BY ロール] TO 新規表領域 [NOWAIT]
```

オブジェクトには ALL / TABLES / INDEXES / MATERIALIZED VIEWS を指定できます。ロールにはオブジェクトの所有者を指定します。NOWAIT を指定すると、テーブル

がロックできない場合に直ちにエラーが返ります。NOWAIT を指定しない場合、テーブルに対するロックが解消するまで待機します。

アクセスできないテーブルに対して移動を行おうとすると、以下のエラーが発生します。

```
ERROR:  must be owner of relation {オブジェクト名}
```

下記例では、テーブル data2 を表領域 ts1 に作成後、表領域 pg_default に移動しています。

例 21 ALTER TABLESPACE 文の拡張

```
postgres=> CREATE TABLE data2(c1 NUMERIC, c2 VARCHAR(10)) TABLESPACE ts1 ;
CREATE TABLE
postgres=> ALTER TABLESPACE ts1 MOVE TABLES OWNED BY scott TO pg_default ;
ALTER TABLESPACE
```

3.6.8 集計関数の追加

いくつかの集計関数が追加されました。また集計関数に WITHIN GROUP 句を使うことができるようになりました。

表 15　集計関数の追加

関数名	説明	備考
PERCENTILE_CONT	連続型分散に基づく百分位数を返す	
PERCENTILE_DISC	並べ替えられた値の特定の百分位数を計算	
MODE	最も出現頻度の高い値を返す	

例 22　MODE 関数の実行例

```
postgres=> SELECT mode() WITHIN GROUP (ORDER BY c1) AS most_frequent FROM rank1 ;
 most_frequent
---------------
             2
(1 row)
```

3.7 データ型の拡張

3.7.1 jsonb 型

json 型の派生型で jsonb 型が使用できるようになりました。このデータ型は json 型と比較すると非圧縮状態でロードされるため、データ量は拡大しますが、インデックスを有効に使用できるため高速に検索を行うことができます。使用方法は json 型と同じです。

例 23 jsonb 型

```
postgres=> CREATE TABLE jsonb1 (col1 jsonb) ;
CREATE TABLE
postgres=> ¥d+ jsonb1
                    Table "public.jsonb1"
 Column | Type  | Modifiers | Storage  | Stats target | Description
--------+-------+-----------+----------+--------------+-------------
 col1   | jsonb |           | extended |              |
```

3.7.2 line 型

line 型が利用できるようになりました。PostgreSQL 9.3 でも line 型列を持つテーブルは作成できますが未実装でした。Ax + By + C = 0 になる値または、(x1, y1)(x2, y2)の形式で座標を指定して入力することができます。

例 24 line 型

```
postgres=> CREATE TABLE line1 (col1 line) ;
CREATE TABLE
postgres=> INSERT INTO line1 VALUES ('{1, -1, 1}') ;
INSERT 0 1
postgres=> INSERT INTO line1 VALUES ('(0,0),(6,6)') ;
INSERT 0 1
postgres=> SELECT * FROM line1 ;
    c1
----------
 {1,-1,1}
 {1,-1,0}
(2 rows)
```

3.7.3 LSN 型

LSN 型のデータに対して基本的な計算が行えるようになりました。

例 25 LSN 型

```
postgres=> SELECT pg_current_xlog_location() - '0/3000000' as diff ;
    diff
------------
 4362156680
(1 row)
```

3.8 ビューに対する拡張

3.8.1 WITH CHECK OPTION 付きビュー

CREATE VIEW 文に WITH CHECK OPTION 句を指定することができるようになりました。この句を指定したビューに対する INSERT 文は CREATE VIEW で指定した WHERE 句条件に合致するかがチェックされるようになります。下記の例では view2 に対する INSERT 文が WITH CHECK OPTION によるチェックにより失敗しています。

例 26 WITH CHECK OPTION

```
postgres=> CREATE TABLE data1 (c1 NUMERIC, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE VIEW view1 AS SELECT * FROM data1 WHERE c1 %2 = 0 ;
CREATE VIEW
postgres=> INSERT INTO view1 VALUES (1, 'view1') ;
INSERT 0 1
postgres=> CREATE VIEW view2 AS SELECT * FROM data1 WHERE c1 %2 = 0 WITH CHECK OPTION ;
CREATE VIEW
postgres=> INSERT INTO view2 VALUES (1, 'view2') ;
ERROR:  new row violates WITH CHECK OPTION for view "view2"
DETAIL:  Failing row contains (1, view2).
```

ビューに対するビューを作成した場合、WITH CHECK OPTION を実行すると、元となるビューに対してもチェックが実行されます。以下の例は WITH CHECK OPTION 句の指定が無い VIEW1 を検索する VIEW2 を作成し、VIEW2 に INSERT 文を実行した場合の動作です。

表 16　ネストされたビューに対する WITH CHECK OPTION

オブジェクト	WITH CHECK OPTION なし	WITH CHECK OPTION あり
VIEW2	チェックされない	チェックされる
VIEW1	チェックされない	チェックされる

下位のビューに対するチェックを実行しないようにするには、WITH LOCAL CHECK OPTION 句を指定します。

3.8.2 自動更新ビューの拡張

PostgreSQL 9.3 ではビューに対する更新が自動的に可能になりました。PostgreSQL 9.4 では自動更新可能な SQL のパターンが増加しました。列リスト内にサブ・クエリーがある場合でもその他の列は自動更新が可能になりました。

例 27 自動更新可能ビューの拡張

```
postgres=> CREATE TABLE datav1 (c1 NUMERIC, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE TABLE datav2 (c1 NUMERIC, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE VIEW updvw1 AS SELECT datav1.*, (SELECT AVG(c1) avgc1 FROM datav2)
FROM datav1 ;
CREATE VIEW
postgres=> SELECT column_name, is_updatable FROM information_schema.columns WHERE
table_name = 'updvw1' ;
 column_name | is_updatable
-------------+--------------
 c1          | YES
 c2          | YES
 avgc1       | NO
(3 rows)
```

上記の例では、ビューupdvw1 の列 c1, c2 が自動更新可能であることを示しています。PostgreSQL 9.3 では同一構成のビューを作成してもすべての列が自動更新不可でした。

3.9 自動 Vacuum 用ワーク・メモリー

自動 Vacuum で使用されるワーク・メモリー領域を示す専用のパラメータが追加されました。パラメータ名は autovacuum_work_mem です。単位はキロバイトで、デフォルト値は-1 です。

デフォルト値の-1 を指定した場合、自動 Vacuum 処理に使うメモリー量は従来バージョンの通り、パラメータ maintenance_work_mem で決定されます。また 1,024 未満の数値を指定した場合、この値は 1024（1 MB）に変更されます。

3.10 外部テーブルに対する拡張

FOREIGN DATA WRAPPER に対する更新処理でトリガーが実行できるようになりました。ただし INSTEAD OF トリガーと TRUNCATE トリガー、CONSTRAINT トリガーには対応していません。

postgres_fdw はリモートの PostgreSQL インスタンスのテーブルにアクセスできますが、旧バージョンと変化がありません。このため集計関数（COUNT / AVG / SUM 等）や GROUP BY 句はリモート・インスタンスに送信されず、全データをローカルに転送してから処理が行われます。

3.11 チェックサムの確認

PostgreSQL 9.3 では、ブロック破損に対応するためにブロック内にチェックサムを使用することができるようになりました。この設定は initdb コマンド実行時に行われ、データベース・クラスタがチェックサムに対応しているかは pg_controldata コマンドを実行する必要がありました。

PostgreSQL 9.4 では、読み取り専用のパラメータ data_checksums を確認することでチェックサムの有無を把握することができます。このパラメータは PostgreSQL 9.3.4 から利用することができます。

例 28 パラメータ data_checksums（チェックサム有効）

```
postgres=# SHOW data_checksums ;
data_checksums
----------------
 on
(1 row)
$ pg_controldata data | grep checksum
Data page checksum version:           1
$
```

3.12 pg_prewarm Contrib パッケージ

Contrib パッケージに pg_prewarm が追加されました。この EXTENSION を追加すると pg_prewarm 関数が使用できるようになります。この関数は実行されると該当テーブルのタプルを共有バッファに展開します。

表 17　pg_prewarm 関数のパラメータ

パラメータ	説明	指定できる値	備考
regclass	テーブル名	テーブル名	
mode	モード	buffer（共有バッファへ読込）	デフォルト値
		prefetch（ファイルの読込）	
		read（ファイルの非同期読込）	
fork	対象	main（データ・ファイル）	デフォルト値
		fsm（FSM ファイル）	
		vm（VM ファイル）	
first_block	最初のブロック	ブロック番号	
last_block	最終ブロック	ブロック番号	

例 29 パラメータ data_checksums（チェックサム有効）

```
postgres=# CREATE EXTENSION pg_prewarm ;
CREATE EXTENSION
postgres=> SELECT pg_prewarm('data1') ;
pg_prewarm
------------
       5406
(1 row)
```

3.13 auto_explain モジュールの拡張

実行計画等を自動的にログに出力する auto_explain モジュールに、log_trigger 設定が追加されました。on に設定するとトリガーの実行時間、回数が出力されます。

例 30 auto_explain の拡張

```
postgres=# LOAD 'auto_explain' ;
LOAD
postgres=# SET auto_explain.log_trigger = on ;
SET
```

3.14 PL/pgSQL の拡張

PG_CONTEXT 句を使用することで、コール・スタックを取得することができるようになりました。PG_CONTEXT 句は、PL/pgSQL ブロック内で、GET DIAGNOSTICS 文と共に使用します。

例 31 PG_CONTEXT 文

```
postgres=> CREATE OR REPLACE FUNCTION public.outer_func() RETURNS integer AS $$
BEGIN
  RETURN inner_func() ;
END;
$$ LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=> CREATE OR REPLACE FUNCTION public.inner_func() RETURNS integer AS  $$
DECLARE
  stack text ;
BEGIN
  GET DIAGNOSTICS stack = PG_CONTEXT ;
  RAISE NOTICE E'--- Call Stack ---¥n%', stack ;
  RETURN 1 ;
END;
$$ LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=> SELECT outer_func() ;

NOTICE:  --- Call Stack ---
PL/pgSQL function inner_func() line 4 at GET DIAGNOSTICS
PL/pgSQL function outer_func() line 3 at RETURN
 outer_func
 ------------
           1
(1 row)
```

4. 参考資料

以下の URL を参照しました。

- PostgreSQL 9.4 Beta Manual
  http://www.postgresql.org/docs/devel/static/index.html
- PostgreSQL 9.4 新機能の情報
  http://michael.otacoo.com/
- FOSDEM 2014 の発表資料
  http://www.hagander.net/talks/postgresql94.pdf
- PostgreSQL 9.4 Beta 1 のアナウンス
  http://www.postgresql.org/about/news/1522/

変更履歴

変更履歴

版	日付	作成者	説明
1.0	2-Jul-2014	篠田典良	公開版を作成

以上