PostgreSQL 9.6 新機能紹介

Noriyoshi Shinoda

June 18, 2016

自己紹介
篠田典良（しのだのりよし）

- 所属
  - 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
- 現在の業務
  - PostgreSQLをはじめOracle Database, Microsoft SQL Server, Vertica, Sybase ASE等 RDBMS全般に関するシステムの設計、チューニング、コンサルティング
  - オープンソース製品に関する調査、検証
  - Oracle Database関連書籍の執筆
  - 弊社講習「Oracle DatabaseエンジニアのためのPostgreSQL入門」講師
- 関連する URL
  - 「PostgreSQL 虎の巻」シリーズ
    - http://h30507.www3.hp.com/t5/user/viewprofilepage/user-id/838802
  - Oracle ACEってどんな人？
    - http://www.oracle.com/technetwork/jp/database/articles/vivadeveloper/index-1838335-ja.html

Agenda
PostgreSQL 9.6 新機能紹介

1. PostgreSQL 9.6概要
2. Parallel Query
3. FOREIGN DATA WRAPPERの拡張
4. レプリケーション関連の新機能
5. モニタリング
6. パフォーマンスと内部構造
7. Contribモジュール

まとめ

1. PostgreSQL 9.6概要

1. PostgreSQL 9.6概要
スケジュール

- PostgreSQL 9.6 Beta 1
  - 2016年5月12日発表　　← 今日のお話
- PostgreSQL 9.6 Beta 2?
  - 2016年9月？
- PostgreSQL 9.6 Final?
  - 2016年12月？

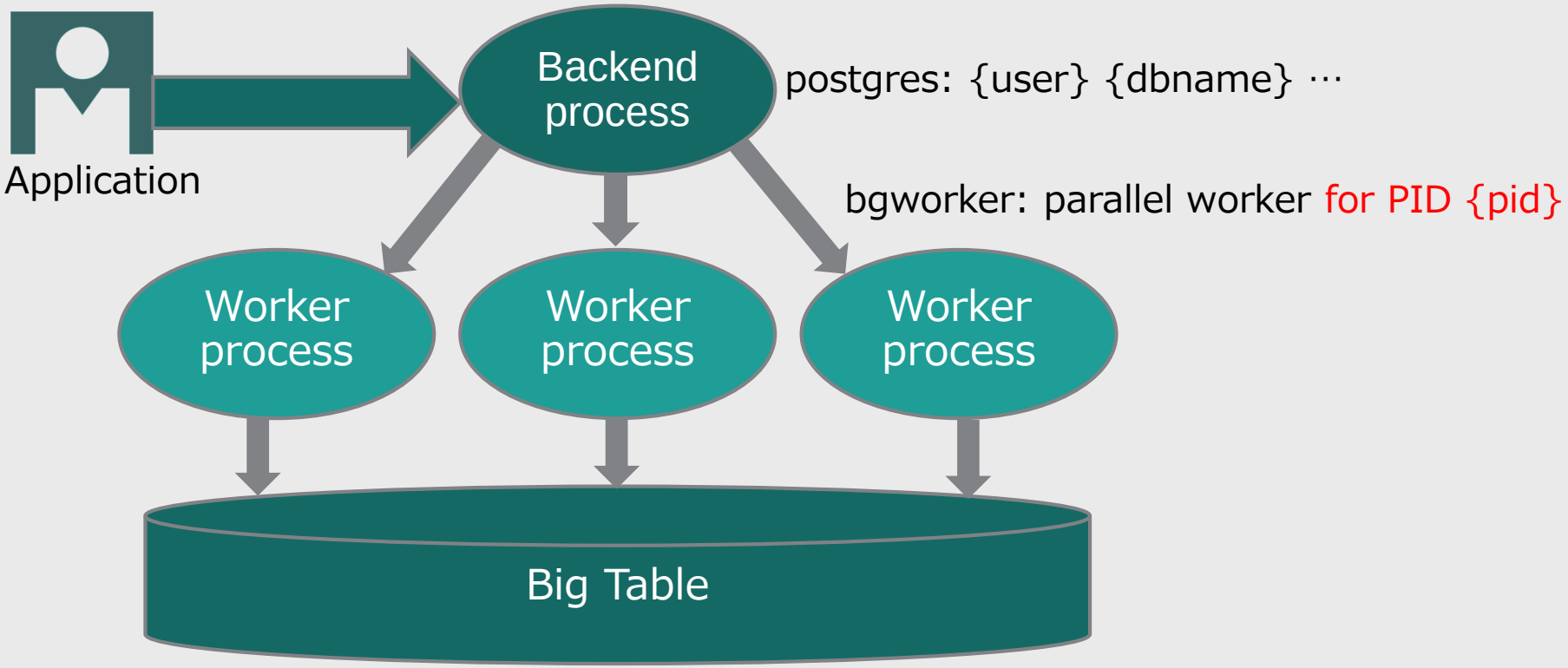
2. Parallel Query

Parallel Query
マルチプロセスによる並列処理

- 概要説明
  - 従来のバージョンでは、セッションに対応する単一のバックエンド・プロセスがすべてのSQL文を処理していました。
  - PostgreSQL 9.6では、Seq Scan / Join / Aggregate処理を複数プロセスを使って処理できるようになりました。
  - 継承テーブルによるパーティション化にも対応しています。

- アーキテクチャ
  - パラレル処理はBackground Worker (9.3～) を使います。
  - プロセス間通信にはDynamic Shared Memory (9.4～) を使います。
  - パラレル処理API (9.5～) を使います。

Parallel Query
マルチプロセスによる並列処理



Parallel Query
実行計画

```
postgres=> EXPLAIN ANALYZE VERBOSE SELECT COUNT(*) FROM data1 ;
                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=107137.83..107137.84 rows=1 width=8)
       (actual time=847.788..847.788 rows=1 loops=1)
   Output: count(*)
   ->  Gather  (cost=107137.62..107137.83 rows=2 width=8)
               (actual time=847.523..847.645 rows=3 loops=1)
         Output: (PARTIAL count(*))
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial Aggregate  (cost=106137.62..106137.63 rows=1 width=8)
                     (actual time=823.624..823.625 rows=1 loops=3)
               Output: PARTIAL count(*)
               Worker 0: actual time=811.215..811.216 rows=1 loops=1
               Worker 1: actual time=812.571..812.571 rows=1 loops=1
               ->  Parallel Seq Scan on public.data1  (cost=0.00..95721.10 rows=4166610 width=0)
                           (actual time=89.482..542.489 rows=3333300 loops=3)
                     Worker 0: actual time=0.073..502.833 rows=3817475 loops=1
                     Worker 1: actual time=268.352..605.484 rows=2208715 loops=1
 Planning time: 0.086 ms
 Execution time: 848.692 ms
(15 rows)
```

Parallel Query
実行計画の説明

オペレータ	説明	表示
Finalize Aggregate	集約の最終処理	ALL
Gather	ワーカー間の集計処理	ALL
Workers Planned	計画されたワーカー数	ALL
Workers Launched	実行されたワーカー数	ANALYZE
Partial Aggregate	ワーカーによる集約	ALL
Partial HashAggregate		
Partial GroupAggregate		
Worker N (N=0, 1, …)	各ワーカーの処理時間	ANALYZE, VERBOSE
Parallel Seq Scan	並列検索	ALL
Single Copy	単一プロセスで実行	ALL

Parallel Query
並列度の決め方

- 並列度は基本的にはデータのサイズに依存します。
  - 1,000ブロック以下の場合、パラレル処理は行われません（最新のスナップショットではmin_parallel_relation_sizeでサイズを定義）。
  - 3,000ブロック、9,000ブロック、27,000ブロックで並列度を上げます。以下３倍ごとに並列度を追加します。
- 並列度の最大値は、MIN (max_worker_processes, max_parallel_degree) となります。
- 実際の起動ワーカー数は実行中のワーカーを除外するため、実行計画作成時の計算よりも小さくなる場合があります。
- テーブル・オプションparallel_degree（最新スナップショットではparallel_workers）が指定されている場合はテーブルのサイズに依存せずに並列度が決定されます（最大 1024）。

```
postgres=> ALTER TABLE data1 SET (parallel_degree = 5) ;
ALTER TABLE
```

Parallel Query
関連パラメーター

– 並列処理に関係するパラメーター

パラメーター名（最新スナップショット）	デフォルト値	説明（コンテキスト）
max_parallel_degree (max_parallel_workers_per_gather)	2	実行計画決定時の最大並列度（user）
parallel_setup_cost	1000	並列処理の開始コスト（user）
parallel_tuple_cost	0.1	並列処理のタプル処理コスト（user）
force_parallel_mode	off	並列処理の実行計画作成を強制（user）
(min_parallel_relation_size)	8MB	Parallel Queryを検討する最小サイズ（user）
max_worker_processes	8	ワーカー・プロセスの最大値（postmaster）
dynamic_shared_memory_type	posix	ダイナミック共有メモリーの種別（postmaster）

Parallel Query
制約

- リーダーのみアクセス可能
  - Temporary Table
  - CTE Scan
  - 外部テーブル（更新をサポートする場合以外）
  - Restricted Parallel Safeな関数の使用
- シリアル処理になるケース
  - DELETE文 / UPDATE文の検索処理
  - 分離レベルがSERIALIZABLE
  - カーソルの使用
  - Parallel Unsafeな関数の使用
- パラメータidle_in_transaction_session_timeoutとの併用
  - 一部のワーカーが終了した場合は？（未検証）

Parallel Query
Parallel Unsafe Function

– Parallel Unsafe Function
  – SQL文内で使われていると、並列処理を行う実行計画が作られません。
  – pg_proc.proparallel='u' である標準関数
  – CREATE FUNCTION文で作成された関数のデフォルト設定

– 主なParallel Unsafe関数

カテゴリー	関数名
シーケンス関連	nextval, currval, setval, lastval
LOB関連	lo_*, loread, lowrite
レプリケーション関連	pg_create_*_slot, pg_drop_*_slot, pg_logical_*, pg_replication_*
その他	pg_advisory_*, pg_try_advisory_*, plpgsql_*_handler, pg_extension_config_dump, pg_*_backup, set_config, txid_current, query_to_xml*, txid_current

Parallel Query
Parallel Unsafe Function

– Restricted Parallel Safe Function
  – SQL文内で使われるとリーダー・プロセスが指定処理を行います。
  – pg_proc.proparallel='r' である標準関数
– 主なRestricted Parallel Safe関数

カテゴリー	関数名
日付関連	age, now, statement_timestamp
乱数関連	random, setseed
XML関連	cursor_to_xml*, database_to_xml*, schema_to_xml*, table_to_xml*, cursor_to_xml*
アップグレード関連	binary_upgrade*
その他	pg_config, pg_cursor, pg_start_backup, pg_get_viewdef, pg_prepared_statement, pg_notify, pg_stat_*

Parallel Query

Parallel Unsafe Function

```
postgres=> EXPLAIN SELECT * FROM data1 WHERE c1=100 ;
                            QUERY PLAN
------------------------------------------------------------------------
 Gather  (cost=1000.00..107137.72 rows=1 width=11)
   Workers Planned: 2
   ->  Parallel Seq Scan on data1  (cost=0.00..106137.62 rows=0 width=11)
         Filter: (c1 = '100'::numeric)
(4 rows)

postgres=> EXPLAIN SELECT * FROM data1 WHERE c1=nextval('s1') ;
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on data1  (cost=0.00..229052.60 rows=1 width=11)
   Filter: (c1 = (nextval('s1'::regclass))::numeric)
(2 rows)
```

Parallel Query
ユーザー作成関数とParallel Unsafe

– CREATE FUNCTION文で作成する関数のデフォルトはParallel Unsafe

– CREATE FUNCTION文 / ALTER FUNCTION文で設定できます。

– 構文

```
CREATE [ OR REPLACE ] FUNCTION function_name  …
PARALLEL {UNSAFE | RESTRICTED | SAFE}
```

Parallel Query
ユーザー作成関数とParallel Unsafe

```
postgres=> CREATE FUNCTION seqf1() RETURNS numeric AS $$
postgres$>    BEGIN RETURN nextval('seq1'); END;
postgres$>  $$ PARALLEL SAFE LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=> EXPLAIN SELECT * FROM data1 WHERE c1=seqf1() ;
                            QUERY PLAN
-------------------------------------------------------------------------
 Gather  (cost=1000.00..711300.83 rows=1 width=11)
   Workers Planned: 4
   ->  Parallel Seq Scan on data1  (cost=0.00..710300.74 rows=0 width=11)
         Filter: (c1 = seqf1())
(4 rows)
postgres=> SELECT * FROM data1 WHERE c1=seqf1() ;
ERROR:  cannot execute nextval() during a parallel operation
CONTEXT:  PL/pgSQL function seqf1() line 2 at RETURN
```

3. FOREIGN DATA WRAPPERの拡張

FOREIGN DATA WRAPPERの拡張

Remote Push-down処理の追加

– いくつかの処理を外部システムで行うことができるようになりました。

操作	PostgreSQL 9.5	PostgreSQL 9.6
ソート	Local	Remote
更新	カーソル利用 / Tuple単位	直接実行 / SQL単位
結合	Local	Remote
集計	Local	Local

– postgres_fdwを使って実行計画を確認します。

FOREIGN DATA WRAPPERの拡張

Sort Push-down

– PostgreSQL 9.6の場合、ORDER BY句がリモートで実行されます。

```
postgres=> EXPLAIN ANALYZE VERBOSE SELECT * FROM remote1
     ORDER BY 1 ;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
Foreign Scan on public.remote1  (cost=100.00..139.87 rows=871 width=70)
(actual time=1060.858..3216.021 rows=1000000 loops=1)
   Output: c1, c2
   Remote SQL: SELECT c1, c2 FROM public.remote1
         ORDER BY c1 ASC NULLS LAST
 Planning time: 0.370 ms
 Execution time: 3257.071 ms
(5 rows)
```

FOREIGN DATA WRAPPERの拡張
Direct Modify

– PostgreSQL 9.6の場合、DELETE文/UPDATE文が直接実行されます。

```
postgres=> EXPLAIN ANALYZE VERBOSE DELETE FROM remote1
      WHERE c1 < 1000 ;
                                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
Delete on public.remote1  (cost=100.00..162.32 rows=910 width=6)
(actual time=149.975..149.975 rows=0 loops=1)
   ->  Foreign Delete on public.remote1  (cost=100.00..162.32 rows=910
width=6) (actual time=149.971..149.971 rows=999 loops=1)
         Remote SQL: DELETE FROM public.remote1 WHERE ((c1 <
1000::numeric))
 Planning time: 0.511 ms
 Execution time: 162.910 ms
(5 rows)
```

FOREIGN DATA WRAPPERの拡張
Direct Modify

– PostgreSQL 9.5以前の場合はカーソルを作成し、1レコード単位で更新されます。

```
LOG:  execute <unnamed>: DECLARE c1 CURSOR FOR SELECT ctid
      FROM public.remote1 WHERE ((c1 < 1000::numeric)) FOR UPDATE
LOG:  statement: FETCH 100 FROM c1
LOG:  execute pgsql_fdw_prep_1: DELETE FROM public.remote1 WHERE ctid = $1
DETAIL:  parameters: $1 = '(0,1)'
LOG:  execute pgsql_fdw_prep_1: DELETE FROM public.remote1 WHERE ctid = $1
…
LOG:  execute pgsql_fdw_prep_1: DELETE FROM public.remote1 WHERE ctid = $1
DETAIL:  parameters: $1 = '(5,74)'
LOG:  statement: DEALLOCATE pgsql_fdw_prep_1
LOG:  statement: CLOSE c1
```

FOREIGN DATA WRAPPERの拡張

Join Push-down

– PostgreSQL 9.6の場合、同一SERVER上の結合処理はリモートで実行されます。

```
postgres=> EXPLAIN ANALYZE VERBOSE SELECT COUNT(*) FROM
remote1, remote2 WHERE remote1.c1 = remote2.c1 AND remote1.c1 =
1000 ;
                                    QUERY PLAN
-------------------------------------------------------------------------------
Aggregate  (cost=157.90..157.91 rows=1 width=8)
      (actual time=388.830..388.830 rows=1 loops=1)
   Output: count(*)
   ->  Foreign Scan  (cost=100.00..157.78 rows=49 width=0)
      (actual time=388.824..388.825 rows=1 loops=1)
         Relations: (public.remote1) INNER JOIN (public.remote2)
         Remote SQL: SELECT NULL FROM (public.remote1 r1 INNER JOIN
public.remote2 r2 ON (((r2.c1 = 1000::numeric)) AND ((r1.c1 =
1000::numeric))))
 Planning time: 0.155 ms
 Execution time: 389.493 ms
(7 rows)
```

4. レプリケーション関連の新機能

Multiple Synchronous Replication
複数インスタンスに対する同期レプリケーション

– synchronous_standby_namesの構文変更

num_sync (*application_name1*, *application_name2*, …)

– num_sync部分には同期インスタンス数を指定します。

```
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)
```

Synchronous Apply
WALの適用までCOMMIT待機

– synchronous_commitパラメーターに設定値 remote_apply 追加
  – スレーブ・インスタンスでWAL適用が完了するまでCOMMIT文を待機します。

設定値	Master	Slave memory	Slave wal	Slave apply
off	非同期			
local	同期	非同期		
remote_write	同期		非同期	
on	同期			非同期
remote_apply	同期			

– recovery_min_apply_delayとの併用
  – スレーブ・インスタンスのrecovery.confファイルにrecovery_min_apply_delay指定があるとCOMMIT文の完了が指定時間待機されます。

5. モニタリング

モニタリング

pg_stat_wal_receiver

列名	データ型	説明
pid	integer	wal receiverプロセスID
status	text	ステータス（starting, streaming, stopped, waiting, …）
receive_start_lsn	pg_lsn	WAL受信開始 LSN
receive_start_tli	integer	WAL受信開始Time Line
received_lsn	pg_lsn	受信LSN
received_tli	integer	受信Time Line
last_msg_send_time	timestamp	最終メッセージ送信時刻
last_msg_receipt_time	timestamp	最終メッセージ受信時刻
latest_end_lsn	pg_lsn	WAL Senderからの最終LSN
latest_end_time	timestamp	WAL Senderからの最終時間
slot_name	text	スロット名

モニタリング
pg_stat_activity

– pg_stat_activityカタログの変更点

列名	データ型	変更	説明
waiting	boolean	削除	待機しているか
wait_event_type	text	追加	待機イベントの概要
wait_event	text	追加	待機イベントの名前

– wait_event_type列の値

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

モニタリング

pg_stat_progress_vacuum

列名	データ型	説明
pid	integer	Autovacuum WorkerのPID
datid	oid	データベースOID
datname	name	データベース名
relid	oid	対象テーブルのOID
phase	text	Vacuumのフェーズ
heap_blks_total	bigint	テーブルの総ブロック数
heap_blks_scanned	bigint	スキャンされた総ブロック数
heap_blks_vacuumed	bigint	Vacuumされた総ブロック数
index_vacuum_count	bigint	インデックスの総サイクル数
max_dead_tuples	bigint	maintenance_work_memに格納できる総タプル数
num_dead_tuples	bigint	デッドタプル数

6. パフォーマンスと内部構造

パフォーマンスと内部構造
Freeze Map

- 不定期な全件検索の廃止
  - 従来のバージョンではパラメーターvacuum_freeze_table_ageで指定された時期（age）が過ぎるとフルスキャンが発生します。
  - 自動Vacuumが停止していてもこの処理は止められませんでした。
  - PostgreSQL 9.6ではFreeze処理を行った場所を管理するようになりました。

- Visibility Map (_vmファイル）の拡張により対応します。
  - ページ単位にAll VisibleとAll Frozenを管理する。

パフォーマンスと内部構造
チェックポイントの改善

- ソートしてからダーティ・バッファを同期
  - 従来のバージョンでは、シェアード・バッファ内を端から検索してダーティ・バッファをフラッシュしていました。
  - PostgreSQL 9.6ではファイル単位にブロック番号順にソートしてから書き込みを行います。
  - シーケンシャルに書き込むことでパフォーマンス向上を期待

パフォーマンスと内部構造
時間ベースのスナップショット生存期間制限

- スナップショットの生存期間を指定
  - 制限時間を過ぎたレコードはVACUUM対象になります。

- パラメータold_snapshot_thresholdに秒数（または日数）を指定
  - -1 = 時間ベースの制御を行わない（デフォルト値）
  - 0 = 直ちに無効
  - 1～86,400 = 秒数による指定
  - 1d～60d = 日数による指定
- 発生するエラー

```
ERROR:  snapshot too old
```

パフォーマンスと内部構造
外部ソートにQuick Sortを使用

- 外部ソートをReplacement Selectionで行うタプル数を指定
  - パラメーターreplacement_sort_tuples（デフォルト値 150000）
- 実行計画上の違い（EXPLAIN ANALYZE出力）
  - Quick Sort

    Sort Method: external merge Disk: 2144kB
  - Replacement Selection

    Sort Method: external sort Disk: 2144kB

- ログの違い（trace_sort = on）
  - Quick Sort

    LOG: starting quicksort of run 1: CPU 0.02s/0.02u sec elapsed 0.05 sec
  - Replacement Selection

    LOG: replacement selection will sort 58253 first run tuples

パフォーマンスと内部構造
その他の追加されたパラメーター

パラメーター名	デフォルト値	説明
enable_fkey_estimates	on	外部キーの情報を実行計画作成に使用する
idle_in_transaction_session_timeout	0	アイドル状態のトランザクションを強制終了する時間（ミリ秒）
backend_flush_after	128kB	backendのFlushを強制する閾値
bgwriter_flush_after	512kB	bgwriterのFlushを強制する閾値
checkpoint_flush_after	256kB	checkpointのFlushを強制する閾値
syslog_sequence_numbers	on	SYSLOGにシーケンス番号を付与
syslog_split_messages	on	長いSYSLOGメッセージを分割
wal_writer_flush_after	1MB	wal writerのFlushを強制する閾値

– backend_flush_afterの最新スナップショットのデフォルト値は 0

パフォーマンスと内部構造
その他の変更されたパラメーター

パラメーター名	説明
log_line_prefix	数値を使ったタイムスタンプ（%n）が利用可能に
wal_level	archiveとhot_standbyがreplicaに統合
autovacuum_max_workers	最大値が262,143に縮小
max_connections	最大値が262,143に縮小
max_replication_slots	最大値が262,143に縮小
max_wal_senders	最大値が262,143に縮小
max_worker_processes	最大値が262,143に縮小
superuser_reservrd_connections	最大値が262,143に縮小
wal_writer_delay	pg_settings.short_dsc列を変更

7. Contribモジュール

bloom
bloom filterを使ったインデックス

– 使用できるデータ型は integer, char, textです。

– 複数列に同時作成できます。

– 等価検索に使用できます。

```
postgres=# CREATE EXTENSION bloom ;
CREATE EXTENSION
postgres=> CREATE INDEX idx1 ON bloom1 USING bloom (c1, c2, c3) ;
CREATE INDEX
postgres=> EXPLAIN SELECT * FROM bloom1 WHERE c1 = 100000 ;
                              QUERY PLAN
-----------------------------------------------------------------------------
 Bitmap Heap Scan on bloom1  (cost=15348.00..15352.01 rows=1 width=20)
   Recheck Cond: (c1 = 100000)
   ->  Bitmap Index Scan on idx1 (cost=0.00..15348.00 rows=1 width=0)
         Index Cond: (c1 = 100000)
(4 rows)
```

pg_visibility
Visibility Mapの情報取得

– 以下の関数が利用できます。

関数名	説明
pg_visibility	指定ブロックのVisible, Frozen状態出力
pg_visibility_map	ブロック番号, Visible, Frozen状態出力
pg_visibility_map_summary	Visible, Frozenブロック数出力

– 実行例

```
postgres=# CREATE EXTENSION pg_visibility ;
CREATE EXTENSION
postgres=# SELECT pg_visibility_map('data1') ;
 pg_visibility_map
-------------------
 (0,f,f)
 (1,f,f)
 (2,f,f)
```

postgres_fdw
パラメーター追加

– fetch_sizeオプション

```
postgres=# CREATE SERVER host1 FOREIGN DATA WRAPPER
postgres_fdw OPTIONS
      (host 'host1', port '5432', dbname 'demodb', fetch_size '300') ;
CREATE SERVER
postgres=# CREATE FOREIGN TABLE table1(c1 INT, c2 VARCHAR(5))
      SERVER host1 OPTIONS(fetch_size '300') ;
CREATE FOREIGN TABLE
```

– extensionsオプション

```
postgres=# CREATE SERVER host1 FOREIGN DATA WRAPPER
      postgres_fdw OPTIONS
      (host 'host1', port '5432', dbname 'demodb', extensions 'hstore') ;
CREATE SERVER
```

pageinspect
実データ出力の追加

– t_data列の追加

```
postgres=# CREATE EXTENSION pageinspect ;
CREATE EXTENSION
postgres=# INSERT INTO data1 VALUES ('A', 'B') ;
INSERT 0 1
postgres=# UPDATE data1 SET c1='C', c2='D' ;
UPDATE 1
postgres=# SELECT lp, t_data FROM
      heap_page_items(get_raw_page('data1', 0)) ;
 lp |   t_data
----+------------
  1 | ¥x05410542      ← 0x05 = ENQ, 0x41 = 'A', 0x42 = 'B'
  2 | ¥x05430544      ← 0x05 = ENQ, 0x43 = 'C', 0x44 = 'D'
(2 rows)
```

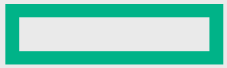
pageinspect
関数の追加（最新のスナップショット）

- pg_check_visible
  - Visibility Bitとページの整合性チェック
- pg_check_frozen
  - Frozen Bitとページの整合性チェック

まとめ

まとめ

- PostgreSQL 9.6には、魅力的な新機能が数多く採用されました。
  - Parallel Seq Scan
  - FOREIGN DATA WRAPPERの改善
  - モニタリングの改善
  - レプリケーション関連の新機能
  - Freeze Mapをはじめとするパフォーマンス改善
- 参考URL
  - Commitfests
    http://commitfest.postgresql.org/
  - ぬこ@横浜さんのブログ
    http://d.hatena.ne.jp/nuko_yokohama/
  - Michael Paquierさんのブログ
    http://michael.otacoo.com/
  - What's New in PostgreSQL 9.6
    https://wiki.postgresql.org/wiki/NewIn96



Hewlett Packard
Enterprise

Thank you

noriyoshi.shinoda@hpe.com