Hewlett Packard Enterprise

June 6, 2016

PostgreSQL 9.6 New Features

With Examples

Hewlett-Packard Enterprise Japan, Ltd.

Noriyoshi Shinoda

© 2016 Hewlett-Packard Enterprise.

Hewlett Packard
Enterprise

Index

© 2016 Hewlett-Packard Enterprise.

© 2016 Hewlett-Packard Enterprise.

Hewlett Packard
Enterprise

1. About This Document

1.1 Purpose

The purpose of this document is to provide information of the major new features of PostgreSQL 9.6, the Beta 1 version being published.

1.2 Audience

This document is written for engineers who already have knowledge of PostgreSQL, such as installation, basic management, and so forth.

1.3 Scope

This document describes the major difference between PostgreSQL 9.5 and PostgreSQL 9.6 Beta 1. It does not mean that all of the new features is examined.

1.4 Software Version

This document is intended for the following versions of the software as a general rule:

Table 1 Version

Software	Version
PostgreSQL	PostgreSQL 9.5.3 (for comparison) PostgreSQL 9.6 Beta 1 (May 9, 2016, 9:04 p.m.)
Operating System	Red Hat Enterprise Linux 7 Update 1 (x86-64)

1.5 Question, Comment, and Responsibility

The contents of this document is not an official opinion of the Hewlett-Packard Enterprise Corporation. The author and affiliation company do not take any responsibility about the problem caused by the mistake of contents. If you have any comments for this document, please contact to Noriyoshi Shinoda (noriyoshi.shinoda@hpe.com) Hewlett-Packard Enterprise Japan Co, Ltd.

© 2016 Hewlett-Packard Enterprise.

1.6 Notation

This document contains examples of the execution of the command or SQL statement. Execution examples are described according to the following rules:

Table 2 Examples notation

Notation	Description
#	Shell prompt for Linux root user
$	Shell prompt for Linux general user
bold	User input string
postgres=#	psql command prompt for PostgreSQL administrator
postgres=>	psql command prompt for PostgreSQL general user
underline	Important output items

The syntax is described in the following rules:

Table 3 Syntax rule

Notation	Description
Italic	Replaced by the name of the object which users use, or the other syntax
[ABC]	Indicate that it can be omitted
{ A | B }	Indicate that it is possible to select A or B
…	General syntax, it is the same as the previous version

© 2016 Hewlett-Packard Enterprise.

2. New Feature Overview

PostgreSQL 9.6 has many new features and improvements.

2.1 Improve Performance

Performance has been improved in the following part:

- Use quicksort to external sort
- Estimate accuracy improvement of the GROUP BY clause
- Use a foreign key to the predicted join
- Writing performance improvement such as checkpoint and bgwriter
- The execution of the Index Only Scan with partial indexes
- Speed up CREATE INDEX CONCURRENTLY statement
- And so force

2.2 Added Features

The major additional features are listed below. The number in parentheses is the number of the chapter in this document for details.

- Parallel Seq Scan (3.5)
- Monitoring Wait Stats (3.6)
- Enhancement of FOREIGN DATA WRAPPER object (3.7)
- Synchronous replication with multiple synchronous standby servers (3.8)
- Avoid Full-Table Vacuum (3.1.4)
- Snapshot Too Old implementation by configurable timeout (3.3.1)
- Improvement of various Contrib module (3.1.3)
- Improvement of various utilities (3.2)
- Catalog for Activity of WAL receiver (3.1.1)
- Generic WAL Records
- Non-exclusive base online backup (3.4.7)
- And so force

© 2016 Hewlett-Packard Enterprise.

2.3 SQL Improvements

The following SQL statements are newly supported. The number in parentheses is the number of the chapter in this document for details.

- COPY enhancement (3.4.1)
- ALTER TABLE ADD COLUMN enhancement (3.4.2)
- ALTER TABLESPACE SET enhancement (3.4.3)
- CREATE EXTENSION enhancement (3.4.4)
- ALTER OPERATOR enhancement (3.4.5)
- CREATE / ALTER FUNCTION PARALLEL SAFE / UNSAFE (3.5)
- CREATE ACCESS METHOD statement added
- CREATE TABLE ... LIKE enhancement
- Added functions (3.4.6 / 3.4.7)
- And so force

Other new features are described in the PostgreSQL 9.6 Beta Documentation Appendix E. Release Notes (http://www.postgresql.org/docs/9.6/static/release-9-6.html).

© 2016 Hewlett-Packard Enterprise.

3. New Feature Details

3.1 Architecture

3.1.1 Added System Catalogs

Along with the addition of new features, the following system catalogs have been added.

Table 4 Added system Catalogs

Catalog Name	Description
pg_config	PostgreSQL binary's install information
pg_stat_wal_receiver	Activity of wal receiver process
pg_stat_progress_vacuum	Activity of vacuum process progress
pg_init_privs	Initial privilege settings for object

□ pg_config Catalog

This catalog exposes the information of various macros that are specified at compile time of PostgreSQL binary. Previously, pg_config command provides the information. Entity of this catalog is pg_config function. This catalog is only readable by a superuser.

Table 5 pg_config Catalog

Column Name	Data Type	Description
name	text	Macro name
setting	text	Setting value

Example 1 Query for pg_config Catalog

```
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
…
```

□ pg_stat_wal_receiver Catalog

This catalog provides information about the state of a slave instance's WAL receiver process in replication environment. This catalog is readable by the general user.

Table 6 pg_stat_wal_receiver Catalog

Column Name	Data Type	Description
pid	integer	Process ID of wal receiver
status	text	Activity Status
receive_start_lsn	pg_lsn	First transaction log position
receive_start_tli	integer	First timeline number
received_lsn	pg_lsn	Last transaction log position already received and flushed to disk
received_tli	integer	Timeline number of last transaction log position received and flushed to disk
last_msg_send_time	timestamp with time zone	Send time of last message received
last_msg_receipt_time	timestamp with time zone	Receipt time of last message received
latest_end_lsn	pg_lsn	Last transaction log position reported
latest_end_time	timestamp with time zone	Time of last transaction log position reported
slot_name	text	Replication slot name

This catalog is made up by the result of pg_stat_get_wal_receiver function.

Example 2 Query for pg_stat_wal_receiver Catalog

```
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
…
```

□ pg_stat_progress_vacuum Catalog

This catalog provides information about the state of the progress of vacuum processing. This catalog is readable by the general superuser.

Table 7 pg_stat_progress_vacuum Catalog

Column Name	Data Type	Description
pid	integer	Process ID of backend
datid	oid	OID of database
datname	name	Connected database name
relid	oid	OID of the table being vacuumed
phase	text	Current processing phase of vacuum
heap_blks_total	bigint	Total number of heap blocks in the table
heap_blks_scanned	bigint	Number of heap blocks scanned
heap_blks_vacuumed	bigint	Number of heap blocks vacuumed
index_vacuum_count	bigint	Number of completed index vacuum cycles
max_dead_tuples	bigint	Number of dead tuples that we can store before needing to perform an index vacuum cycle
num_dead_tuples	bigint	Number of dead tuples collected since the last index vacuum cycle

Example 3 Query for pg_stat_progress_vacuum Catalog

```
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 Catalog

The pg_init_privs catalog stores the information about the initial privileges of objects which have non-default value in the database. This catalog is readable by the general superuser.

Table 8 pg_init_privs Catalog

Column Name	Data Type	Description
objoid	oid	The OID of the specific object
classoid	oid	The OID of the system catalog the object is in
objsubid	integer	For a table column, this is the column number
privtype	char	The type of initial privilege of this object
initprivs	aclitem[]	The initial access privileges

3.1.2 Modified Catalogs

The following system catalogs have been changed.

Table 9 Modified catalogs

Catalog Name	Changed
pg_replication_slots	Add confirmed_flush_lsn column
pg_stat_activity	Remove waiting column
	Add wait_event_type column and wait_event column
pg_proc	Add proparallel column
pg_aggregate	Stores information about aggregate functions
pg_am	Stores information about index access methods

□ pg_replication_slots Catalog

The confirmed_flush_lsn column has been added.

Table 10 Added column to pg_replication_slots Catalog

Column Name	Data Type	Description
confirmed_flush_lsn	pg_lsn	Receiving LSN information of the logical slot

□ pg_stat_activity Catalog

The "waiting" column that shows only the waiting state is replaced with the wait_event column and the wait_event_type column that show the wait events.

Table 11 Added columns to pg_stat_activity Catalog

Column Name	Data Type	Description
wait_event_type	text	The type of event for which the backend is waiting
wait_event	text	Wait event name if backend is currently waiting

Please refer to the following URL information about wait events.

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

□ pg_proc Catalog

Proparallel column that shows whether procedure is PARALLEL SAFE or PARALLEL UNSAFE has been added.

Table 12 Added column to pg_proc Catalog

Column Name	Data Type	Description
proparallel	char	Parallel Safe = 's' function, Restricted Parallel Safe function = 'r' (Only leader can execute), Parallel Unsafe function = 'u'

□ pg_aggregate Catalog

The following column has been added.

Table 13 Added columns to pg_aggregate Catalog

Column Name	Data Type	Description
aggcombinefn	regproc	Combine function (zero if none)
aggserialfn	regproc	Serialization function (zero if none)
aggdeserialfn	regproc	Deserialization function (zero if none)
aggserialtype	oid	Return data type of the aggregate function's serialization function

□ pg_am Catalog

pg_am catalog change completely, became simple.

Table 14 Columns of pg_am Catalog

Column Name	Data Type	Description
amname	name	Access method name
amhandler	oid	OID of handler function
amtype	char	Type of access method

© 2016 Hewlett-Packard Enterprise.

At the time of writing (June 6, 2016), there is no description about amtype column on the manual (http://www.postgresql.org/docs/9.6/static/catalog-pg-am.html).

3.1.3 Modified Contrib modules

In PostgreSQL 9.6, some Contrib modules have been changed.

Table 15 Changed in Contrib modules

Module	Changed	Note
auto_explain	Add parameter	Sample_rate parameter has been added
postgres_fdw	Add options	Fetch_size option has been added Extensions option has been added
pg_visibility	Add module	Provide Visibility Map information
bloom	Add module	Index that uses a Bloom filter
sslinfo	Add function	Add ssl_extension_info function
tsearch2	Enhancement	Add operators for phrase search
pg_trgm	Enhancement	Add support for "word similarity". Add configuration parameter pg_trgm.similarity_threshold
pgcrypto	Add function	Add an optional S2K iteration count parameter
pgpageinspect	Add output	Output data added to the heap_page_items function
hstore	Add function	Add functions for json type

□ Enhancement of auto_explain module

Parameter auto_explain.sample_ratio has been added to the auto_explain module. This parameter specifies the percentage of SQL statements to log the execution plan. For example, the execution plan to log by auto_explain.log_min_duration parameter will be reduced to the percentage specified by this parameter. The default value is 1 (= 100%).

□ Enhancement of postgres_fdw module

The fetch size to get tuples by SELECT statement can be specified now. This option can be specified for each SERVER or FOREIGN TABLE. The fetch size in the previous postgres_fdw module was fixed to 100.

Example 4 fetch_size option

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

The following example shows the actually executed SQL statement at the remote instance when "SELECT * FROM table1" statement were executed.

Example 5 Executed SQL statement (from log file)

```
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
```

EXTENSIONS option for the remote instance can be specified now.

Example 6 Specify extensions option

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

© 2016 Hewlett-Packard Enterprise.

□ Add pg_visibility module

Pg_visibility modules that can get the information of the Visibility Map has been added. The following functions are provided. The execution requires superuser privileges.

Table 16 Functions in the pg_visibility module

Function name	Description
pg_visibility	View the status of each block of the specified table
pg_visibility_map	View the status of each block of the specified table
pg_visibility_map_summary	Display the status of the specified table

In the following example, the number of VISIBLE blocks and FREEZEed blocks is gotten by the pg_visibility_map_summary function .

Example 7 pg_visibility module

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

□ Add bloom module

Bloom module has been added to the Contrib modules. Indexes that use the Bloom Filter can be created by loading the bloom module. Using this module, it is possible to create an index for a large number of column at the same time that uses only a relatively small capacity of the storage. By simple verification, if the BTree index is created on the column described in the WHERE clause, the BTree index is used.

To use the bloom module, it is necessary to specify the USING bloom clause in the CREATE INDEX statement. There are "size", "col1", "col2", … options, but use of them has not yet been verified.

Example 8 Creating a bloom module and the execution plan

```
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) (actual time=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)
```

□ Enhancement of pageinspect module

The real data (t_data) column has been added to the output of the heap_page_items function.

Example 9 Execution of heap_page_items function

```
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 Avoid Full-Table Vacuum

PostgreSQL manages the age of transaction (XID) by the unsigned 32-bit integer. When huge number of transactions are executed, it may be run out of 32-bit integer. For this reason, before running out of the transaction ID, PostgreSQL updates old XIDs in the database to the special XID (FrozenXID = 2). This process is called FREEZE. In FREEZE process of previous version, when it exceeds the specified age with the parameter autovacuum_freeze_max_age, full scan for the table was executed regardless of whether the table was updated or not.

In PostgreSQL 9.6, by extending the Visibility Map to identify the block to be FREEZEed, now it is possible to prevent the large-scale I/O due to the full scan.

3.1.5 Improve CHECKPOINT

Checkpoint of the previous PostgreSQL searched dirty pages in the shared buffer randomly and wrote them to file. In PostgreSQL 9.6, it divides the dirty pages for each table space, sort them by file and block number and writes them. This will allow checkpoint process to write more sequentially.

© 2016 Hewlett-Packard Enterprise.

3.2 Utilities

The following sections describe the major enhancements of the utility commands.

3.2.1 psql

The following features in psql command has been added.

□ Prompt setting

The Process ID of the backend process can be set now in the prompt of the psql command. To set the Process ID , Specify the %p to the variable PROMPT1, PROMPT2, PROMPT3.

Example 10 Specifying the prompt

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

□ Backslash commands

The following enhancements have been made.

Table 17 Added and modified backslash commands

Command	Modification	Description
\ev *view_name*	Added	Change the View definition by an external editor
\sv *view_name*	Added	Show the View definition
\sv+ *view_name*	Added	Show the View definition with line numbers
\gexec	Added	Run the SQL statements in the output result
\errverbose	Added	Show the error information that occurred just before
\crosstabview *column_name*	Added	Show the cross tabulation
\x auto	Changed	Does not use extended table format with EXPLAIN ANALYZE statement
\watch and \pset title	Changed	Show the value of the \pset title when executing the \watch command

□ Show and edit the VIEW definition.

\ev and \sv command has been added to the psql command. \ev command edits the specified view definition in the editor. \sv command shows the specified view definition. The \sv+ command displays the line number in the view definition.

Example 11 Reference of the VIEW definition

```
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=>
```

□ Execute result(s) of previous query as new queries.

\gexec command executes the result(s) of the SQL Statement just before as new SQL Statements. This is useful when creating the CREATE statement to generate the object using the SELECT statement.

Example 12 Executes the output result

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

□ \watch and \pset title

At the time of execution in the \watch command, the string specified in the \pset title command is now displayed.

Example 13 \watch and \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)
```

□ Multiple -c option and -f option

-c option and -f option that specify the SQL statement to be executed at the time of connection, can now be specified multiply.

3.2.2 pg_basebackup

--slot (or -S) option that specify the slot name to be used for the backup has been added to pg_basebackup command. If non-existent slot name is specified, a warning will be shown, but the backup process will be executed. When this option is specified along with the --write-recovery-conf (-R) option, primary_slot_name option will be added to the recovery.conf file in the backup destination. In order to use the --slot option, it is required to use --xlog-method = setting of the stream (-Xs) option at the same time.

Example 14 Specify the --slot option

```
$ pg_basebackup -D back -x -v -R --slot=slot_1  ← no -Xs option
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 ← not exists slot name
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  ← normal execution
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     ← Check the recovery.conf file
standby_mode = 'on'
primary_conninfo    =    'user=postgres    port=5432    sslmode=disable
sslcompression=1'
primary_slot_name = 'slot_1'
$
$ ls -l back/pg_replslot/ ← check the slot directory
total 0
$
```

The slot is not created at the backup destination for the database cluster.

© 2016 Hewlett-Packard Enterprise.

3.2.3 pg_rewind

Pg_rewind command can work when the target timeline changes. This feature is not verified. Please refer to the following URL.

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

3.2.4 pg_dump / pg_restore

The --strict-names option has been added to the pg_dump command and pg_restore command. In addition, -t option of pg_restore command also match relation other than the normal table. This feature is not verified.

3.2.5 pgbench

Some of the new features in pgbench command has been added, but it does not have to verify.

© 2016 Hewlett-Packard Enterprise.

3.3 Changes of parameters

The following parameters have been changed in PostgreSQL 9.6.

3.3.1 Added Parameters

The following parameters have been added.

Table 18 Added Parameters

Parameter Name	Description (context)	Default Value
backend_flush_after	When single backend writes more than specified size, attempt to force the OS to flush these writes to disk. (user)	128kB
bgwriter_flush_after	Bgwriter force the flush at the time of writing occurrence greater than or equal to the specified size (sighup)	512kB
checkpoint_flush_after	Checkpointer force the flush at the time of writing occurrence greater than or equal to the specified size (sighup)	256kB
enable_fkey_estimates	Use the foreign key to estimate join cost. (user)	on
force_parallel_mode	Force parallel processing (user)	off
idle_in_transaction_session_timeout	Idle transaction timeout (user)	0
max_parallel_degree	The maximum value of the degree of parallelism (user)	2
old_snapshot_threshold	The minimum time that a snapshot is guaranteed valid. (postmaster)	-1
replacement_sort_tuples	The maximum number of tuples to be sorted using Replacement Selection (user).	150000
parallel_setup_cost	Start cost of parallel processing (user)	1000
parallel_tuple_cost	Tuple processing cost of parallel processing (user)	0.1
syslog_sequence_numbers	Add a sequence number to the SYSLOG message (sighup)	on
syslog_split_messages	Split long SYSLOG messages (sighup)	on
wal_writer_flush_after	Wal writer is forced to flush at the time of writing occurrence greater than or equal to the specified size (sighup)	1MB

© 2016 Hewlett-Packard Enterprise.

□ idle_in_transaction_session_timeout parameter

When the transaction is idle for a specified period of time in milliseconds, specify the time to forcibly disconnect the session. The default value is 0 in the time-out does not occur. Specify the parameters from psql of the session in the following example and is running a COMMIT statement from a while waiting after the BEGIN statement is executed.

Example 15 Automatic termination of the idle transaction

```
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 parameter, syslog_split_messages parameter

In a message to be output to the SYSLOG to add the information. By setting the syslog_sequence_numbers parameter to on, the sequence number for each process in the message that is output to the SYSLOG will be added. The default value is on. The syslog_split_messages parameter splits the message if the message has exceeded the PG_SYSLOG_LIMIT (900) bytes.

The example below is the log of the instance startup. Process of process ID 3155 will being output the message. [1-1], [1-2], [2-1], has been added in the syslog_sequence_numbers parameter is part of the [2-2].

Example 16 Change of SYSLOG output

```
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 parameter

This parameter specifies the survival time of the snapshot in seconds. Unnecessary tuple exceeds the threshold value can be released by vacuum. The number of seconds that can be specified for this parameter is from -1 to 86,400. However, you can specify up to 60d is to specify the number of days, such as "1d". The default value of -1 in the same operation as the previous version, disable this feature. Referring to the deleted snapshot error "ERROR: snapshot too old" will occur.

3.3.2 Changed Parameters

The following parameters are changed in the range of setting or the options.

Table 19 Changed Parameters

Parameter Name	Modification
log_line_prefix	Set to output the time stamp of the numeric types have been added
effective_io_concurrency	It can now be specified in the ALTER TABLESPACE SET statement
wal_level	Setting archive and hot_standby has been unified in the "replica"
synchronous_commit	The remote_apply can now be specified
autovacuum_max_workers	The maximum value has been changed to 262,143 from 8,388,607
max_connections	The maximum value has been changed to 262,143 from 8,388,607
max_replication_slots	The maximum value has been changed to 262,143 from 8,388,607
max_wal_senders	The maximum value has been changed to 262,143 from 8,388,607
max_worker_processes	The maximum value has been changed to 262,143 from 8,388,607
superuser_reservrd_connections	The maximum value has been changed to 262,143 from 8,388,607
wal_writer_delay	Short_desc column of pg_settings catalog has been changed

© 2016 Hewlett-Packard Enterprise.

□ log_line_prefix parameter

Can now specify the % n which is numerical representation of a time stamp.

Example 17 Parameter 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
```

□ wal_level parameter

"Archive" and "hot_standby" value for this parameter have been unified to "replica". However, when archive or hot_standby is specified , the error doesn't occur and it is considered to be a replica.

Example 18 Parameter wal_level

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

□ synchronous_commit parameter

Value "remote_apply" can now be specified for this parameter. Set this value, in a synchronous replication environment, transaction of the master instance will wait until the WAL is applied on the slave instance. This makes it possible to read the updated data on the slave instance at the time of the completion of the transaction. If recovery_min_apply_delay parameter is specified in the recovery.conf file on the slave instance, the completion of the COMMIT statement on the master instance will wait for the specified time.

3.3.3 Parameters changed the default value

The default values of the following parameters have been changed.

Table 20 Parameters the default value of which is changed

Parameter Name	PostgreSQL 9.5	PostgreSQL 9.6
server_version	9.5.2	9.6beta1
server_version_num	90502	90600

3.4 Enhancement for SQL statement

The new features on the SQL statement, are explained here.

3.4.1 Enhancement of the COPY statement

In the prior version it was able to use the COPY statement to output the results of the SELECT statement to a file or standard output. In PostgreSQL 9.6, by specifying the UPDATE / DELETE / INSERT statements to the COPY statement, it will be able to output the affected records.

Example 19 Specify the DELETE statement to COPY statement

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

In the above example 5 records that have been deleted by the DELETE statement are written to the file. When specifying the DELETE, UPDATE, and INSERT statement RETURNING clause is required .

Example 20 No RETURNING clause error

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

3.4.2 Enhancement of the ALTER TABLE ADD COLUMN statement

The ALTER TABLE ADD COLUMN statement to add a column, IF NOT EXISTS clause to check for the existence of the column can now be specified. Only if the specified column does not exist, add column operation will be performed.

Syntax

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

Example 21 ALTER TABLE ADD COLUMN IF NOT EXISTS statement

```
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 Enhancement of the ALTER TABLESPACE SET statement

In PostgreSQL 9.6 it is now possible to set the parameter effective_io_concurrency for each tablespace. Parameters that can be specified for each tablespace in the old version was only random_page_cost and seq_page_cost.

Example 22 Enhancement of the ALTER TABLESPACE statement

```
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 Enhancement of the CREATE EXTENSION statement

To CREATE EXTENSION statement, CASCADE clause to automatically load the relevant module can now be specified.

Syntax

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

Example 23 CREATE EXTENSION CASCADE statement

```
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 Enhancement of the ALTER OPERATOR statement

ALTER OPERATOR statement to make the change of the operator have been significantly enhanced. It can be added use the following syntax as well as the CREATE OPERATOR statement.

Syntax

```
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 Added function for jsonb type

Function for jsonb type has been added.

□ Function jsonb_insert

Jsonb_insert function to perform additional elements have been provided.

Example 24 Function jsonb_insert

```
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 Additional Functions

Following function have been enhanced in PostgreSQL 9.6.

□ scale function

By specifying a number of numeric type as the parameter, this function returns the number of decimal places. It cannot be used for a float type value. If NULL is passed, it returns NULL.

□ num_nulls / num_nonnulls functions

Returns the number of NULL values from any number of arguments (num_nulls), returns the number of non-NULL value (num_nonnulls).

Example 25 num_nulls / num_nonnulls functions

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

□ current_setting function

The overloaded function that has second parameter has been created. When specified true for the second parameter, no error occurs if specified a non-existent parameter.

Example 26 current_setting function

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

□ pg_control_* functions

It is now possible to get the information by SQL function, conventionally that has been acquired by the pg_controldata command by SQL function. The following functions can be used. General users can use these functions.

Table 21 Added functions

Function Name	Description
pg_control_init	Acquiring of database cluster information
pg_control_checkpoint	Acquiring of checkpoint information
pg_control_recovery	Acquiring of recovery information

Example 27 Execute pg_control_init function

```
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
```

© 2016 Hewlett-Packard Enterprise.

Example 28 Execute pg_control_checkpoint function

```
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
```

Example 29 Execute pg_control_recovery function

```
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
```

© 2016 Hewlett-Packard Enterprise.

□ pg_current_xlog_flush_location function

Function pg_current_xlog_flush_location that returns the LSN indicating the writing location of the WAL file has been added. General users can execute this function, but cannot execute in slave instance of the replication environment.

Example 30 pg_current_xlog_flush_location function

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

□ parse_ident function

This function decompose a string indicating the name of the object that contains the schema name into an array consists of schema and object names. The presence of the specified object is not checked. Also search_path parameter is not considered.

Example 31 parse_ident function

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

□ pg_size_bytes function

Pg_size_bytes function returns the number of bytes from the string specified in units of kB, MB, GB and TB. This is the opposite behavior of pg_size_pretty function. It can be put a space between the number and the unit.

Example 32 pg_size_bytes function

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

□ pg_blocking_pids function

Returns a list of processes that are blocking the process specified by process ID.

Example 33 pg_blocking_pids function

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

□ Extend pg_start_backup / pg_stop_backup functions

The pg_start_backup function and the pg_stop_backup function, parameters for the exclusive control "exclusive" (boolean) has been added. The default value is false, the behavior is the same as previous versions. It does not create the backup_label files and tablespace_map file if you specify the "exclusive" parameter to false. Pg_stop_backup function you must specify the same mode as the pg_start_backup function.

Example 34 pg_start_backup / pg_stop_backup functions

```
postgres=# SELECT pg_start_backup('back1', true, false) ;
 pg_start_backup
-----------------
 0/4C000028
(1 row)

postgres=# -- Do online backup
postgres=# SELECT pg_stop_backup(true) ;
ERROR:  non-exclusive backup in progress
HINT:  did you mean to use pg_stop_backup('f')?
postgres=#
postgres=# SELECT pg_stop_backup(false) ;
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
                              pg_stop_backup
-------------------------------------------------------------------------------
 (0/4C0000F8,"START WAL LOCATION: 0/4C000028 (file 00000001000000000000004C)+
 CHECKPOINT LOCATION: 0/4C000060                                            +
 BACKUP METHOD: streamed                                                    +
 BACKUP FROM: master                                                        +
 START TIME: 2016-06-01 09:51:03 JST                                        +
 LABEL: back1                                                               +
 ","")
(1 row)
```

□ Unverified functions

The following functions have been added, but the behavior is not verified.

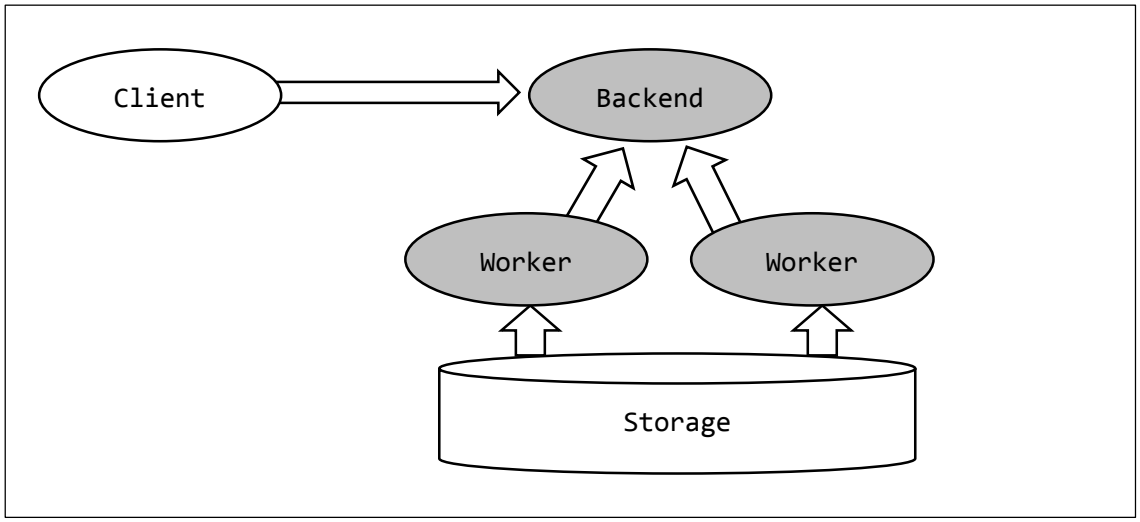
- pg_notification_queue_usage
- acosd / asind / atan2d / atand / cosd / cotd / sind / tand

3.5 Parallel Seq Scan

3.5.1 Overview

In the conventional PostgreSQL, all of the SQL statements ware executed only by the back-end process that accepts the connection. In PostgreSQL 9.6 now it is possible to perform parallel processing by multiple worker processes.

Figure 1 Parallel Seq Scan / Parallel Aggregate



Parallel processing can be executed only for Seq Scan, Join and Aggregate. The degree of parallelism depends on the size of the table. Processes executing parallel processing use the mechanism of the Background Worker. The maximum value of the degree of parallelism is determined by the parameter max_parallel_degree or max_worker_processes, whichever is smaller. Parameter max_parallel_degree can be changed by general users by per-session.

Table 22 Related parameters for parallel processing

Parameter Name	Description (context)	Default value
max_parallel_degree	The maximum value of the degree of parallelism (user)	2
parallel_setup_cost	Start cost of parallel processing (user)	1000
parallel_tuple_cost	Tuple cost of parallel processing (user)	0.1
max_worker_processes	The maximum value of the worker process (postmaster)	8
force_parallel_mode	Force parallel processing (user)	off

© 2016 Hewlett-Packard Enterprise.

□ Parameter force_parallel_mode

Parallel processing is executed only when the cost is considered lower than the normal serial processing. By specifying the parameter force_parallel_mode to on, parallel processing is forced (Also value 'regress' is for the regression test). However, the parallel processing is executed only when the parameter max_parallel_degree is 1 or more.

□ Related table option

Table option parallel_degree determines the degree of parallelism for each table. When the value is set to 0, parallel processing is prohibited. If not set, the parameters max_parallel_degree of the session will be the upper limit.

If parallel_degree is set to greater than the max_parallel_degree, the upper limit of the actual degree of parallelism cannot exceed the max_parallel_degree.

Example 35 Execution plan of parallel processing.

```
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 Execution plan

The example below is the execution plan of the parallel processing SELECT statement. COUNT processing of large-scale table is processed in 3 parallelism.

Example 36 Execution plan of parallel processing

```
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)
```

Following execution plan component can be shown by the EXPLAIN statement about parallel processing.

Table 23 The output of the EXPLAIN statement

Plan component	Description	Explain Statement
Parallel Seq Scan	Parallel search processing	All
Partial Aggregate	Aggregation processing performed by the worker process	All
Gather	Processing to aggregate the worker process	All
Finalize Aggregate	The final aggregation processing	All
Workers Planned:	The number of planned worker processes	All
Workers Launched:	The number of workers that are actually run	ANALYZE
Worker N (N=0,1,…)	Processing time of each worker, etc	ANALYZE, VERBOSE
Single Copy	Processing to be executed in a single process	All

3.5.3 Parallel processing and functions

There are usable functions and unusable functions in parallel processing. When functions which have 'u'(PARALLEL UNSAFE) value for proparallel column in pg_proc catalog are user in SQL statement, parallel processing can not be performed. The following table shows major standard PARALLEL UNSAFE functions.

Table 24 Major PARALLEL UNSAFE standard functions

Category	Function name example
JSON	json_populate_record, json_populate_recordset, jsonb_insert, jsonb_set
SEQUENCE object	nextval, currval, setval, lastval
Large Object	lo_*, loread, lowrite
Replication	pg_create_*_slot, pg_drop_*_slot, pg_logical_*, pg_replication_*
Other	pg_advisory_*, pg_try_advisory_*, plpgsql_*_handler, pg_extension_config_dump, pg_*_backup, set_config, txid_current, query_to_xml*

In the following example, two SQL statements that differ only conditional part of the WHERE clause are executed. SELECT statement with the literal in the WHERE clause will be performed parallel processing but, SELECT statement with the currval of sequence operation function is executed in serial.

© 2016 Hewlett-Packard Enterprise.

Example 37 The difference of the execution plan by the use of PARALLEL UNSAFE function

```
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)
```

In pg_proc in the catalog, functions that are the proparallel column 'r' can only be run on the leader process of parallel processing.

Table 25 Major RESTRICTED PARALLEL SAFE standard functions

Category	Function name example
Date and Age	age, now
Random number	random, setseed
Upgrade	binary_upgrade*
Convert to XML	cursor_to_xml*, database_to_xml*, schema_to_xml*, table_to_xml*
Other	pg_start_backup, inet_client*, current_query, pg_backend_pid, pg_conf*, pg_cursor, pg_get_viewdef, pg_prepared_statement, etc

□ User-defined functions and PARALLEL SAFE

To indicate whether it is possible to perform parallel processing for user-defined functions, can be used PARALLEL SAFE clause or PARALLEL UNSAFE clause in the CREATE FUNCTION statement or ALTER FUNCTION statement. The default is PARALLEL UNSAFE.

Example 38 User-defined functions and 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 Calculation of the degree of parallelism

The degree of parallelism is calculated based on the size of the table, it is increased by one as the size of table is 3,000 blocks, 9,000 blocks and 27,000s block. After that, it is increased as the size of table becomes threefold within the range that does not exceed the parameter max_parallel_degree or parameter max_worker_processes. Actual calculation is executed by create_parallel_paths function in the source code src/backend/optimizer/path/allpaths.c.

Example 39 The part of create_parallel_paths function

```
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 Monitoring Wait Stats

The information of the wait events that are occurring in the PostgreSQL instance can be get now. Waiting column is deleted from pg_stat_activity catalog, wait_event_type columns and wait_event column are added. The wait_event_type column contains the following values.

Table 26 The value of wait_event_type column

Columns value	Description
LWLockNamed	Waiting by a particular lightweight lock
LWLockTranche	Waiting by the lightweight lock for the group
Lock	Waiting by weight lock (LOCK TABLE statement etc)
BufferPin	PIN waiting for the buffer

Please refer to the following URL for more information.

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

Example 40 Waiting by two LOCK TABLE IN EXCLUSIVE statements

```
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)
```

Example 41 Waiting by SELECT FOR UPDATE statement and UPDATE statement

```
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 Enhancement of FOREIGN DATA WRAPPER

FOREIGN DATA WRAPPER to provide access to external objects has been extended.

3.7.1 Sort Push-down

In previous versions sort processing for FOREIGN TABLE were executed on the local instance after data were transferred from the external system. In PostgreSQL 9.6 the ORDER BY clause can be sent to external objects. In the following example, the SELECT statement with the ORDER BY clause to the remote instance is executed using the postgres_fdw module.

Example 42 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)
```

The example shows that the ORDER BY clause in SQL Statement for table1 which is a FOREIGN TABLE is executed on the remote instance.

3.7.2 Direct Modify

DELETE and UPDATE statements in the old version for FOREIGN TABLE created cursors by SELECT FOR UPDATE statement, the update process was executed for each record in the cursor. In PostgreSQL 9.6 update DML can be executed directly in the remote instance.

In the following example, the DELETE statement to FOREIGN TABLE is executed in PostgreSQL 9.5 and PostgreSQL 9.6. In PostgreSQL 9.5 SELECT FOR UPDATE statement is executed, but it changes to DELETE statement in PostgreSQL 9.6.

Example 43 DELETE statement in PostgreSQL 9.5

```
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)
```

Example 44 DELETE statement in PostgreSQL 9.6

```
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

A process to join the tables on the same FOREIGN SERVER can now be performed on a remote instance.

Example 45 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 Multiple synchronous standby servers

Support synchronous replication with multiple synchronous standby servers. Specify the number of instances to execute synchronous replication to the parameter synchronous_standby_names in primary instance.

Syntax

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

Specify the number of instances to execute synchronous replication to the num_sync in integer of 1 or more. If specified zero or less , if omitted, the instance can't start. If the specified number of instances cannot be ensured, update transaction of the primary instance is stopped. In the example below, there are three instances that can execute synchronous replication, and execute synchronous replication to two instance actually.

Example 46 Multiple synchronous standby server replication

```
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)
```

The parameter synchronous_standby_names can be written in the same format as the previous version. In that case instance that can execute synchronous replication is one.

3.9 Security

3.9.1 Default Role

Pg_signal_backend role is created in default. This role allow the transmission of the signal to the backend processes.

3.9.2 Namespace

Role (user) names that begin with pg_ are no longer able to use because those are reserved.

Example 47 Role names starting from "pg_"

```
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

Bibliography

I have referred to the following Website.

- Release Notes
  http://www.postgresql.org/docs/9.6/static/release-9-6.html
- Commitfests
  https://commitfest.postgresql.org/
- PostgreSQL 9.6 Beta Manual
  http://www.postgresql.org/docs/9.6/static/index.html
- GitHub (Mirror of the official PostgreSQL GIT repository)
  https://github.com/postgres/postgres
- Announce of PostgreSQL 9.6 Beta 1
  http://www.postgresql.org/about/news/1668/
- NewIn96
  https://wiki.postgresql.org/wiki/NewIn96
- Open source developer based in Japan (Michael Paquier)
  http://michael.otacoo.com/
- Hibi-no kiroku bekkan (Nuko@Yokohama) [In Japanese]
  http://d.hatena.ne.jp/nuko_yokohama/

© 2016 Hewlett-Packard Enterprise.

Hewlett Packard
Enterprise

Modification History

History

Version#	Date	Author	Description
0.1	May 20, 2016	Noriyoshi Shinoda	Created for HPE internal. (Beta 1) Reviewers are: Tomoo Takahashi Akiko Takeshima Takahiro Kitayama Takuma Ikeda
1.0	May 30, 2016	Noriyoshi Shinoda	Created to be published to the Internet.
1.0.1	May 31, 2016	Noriyoshi Shinoda	Fix typo
1.0.2	June 2, 2016	Noriyoshi Shinoda	Add pg_start_backup / pg_stop_backup
1.0.3	June 6, 2016	Noriyoshi Shinoda	Fix parallel_degree parameter description. Fix Parallel Unsafe function list. Fix pg_proc description.

© 2016 Hewlett-Packard Enterprise.

Hewlett Packard
Enterprise

© 2016 Hewlett-Packard Enterprise.