August 7, 2015

PostgreSQL 9.5 New Features

With Examples

Hewlett-Packard Japan, Ltd.

Noriyoshi Shinoda

Index

Index 2

1. About This Document 4

  1.1 Purpose 4

  1.2 Audience 4

  1.3 Scope 4

  1.4 Software Version 4

  1.5 Question, Comment, and Responsibility 4

  1.6 Notation 5

2. New Feature Overview 6

  2.1 Improve Performance 6

  2.2 Added Features 6

  2.3 SQL Improvements 6

3. New Feature Details 8

  3.1 Architecture 8

    3.1.1 New WAL format 8

    3.1.2 WAL compression 8

    3.1.3 Added System Catalogs 8

    3.1.4 Modified Catalogs 11

    3.1.5 Modified Contrib modules 13

    3.1.6 Interface, API, Hook 14

    3.1.7 OOM Killer support 15

    3.1.8 Archiving of last WAL segment 15

  3.2 Utilities 16

    3.2.1 pg_rewind 16

    3.2.2 pg_ctl 18

    3.2.3 vacuumdb 19

    3.2.4 pg_dump 21

    3.2.5 psql 21

    3.2.6 pgbench 23

    3.2.7 pg_receivexlog 25

    3.2.8 reindexdb 25

  3.3 Changes of parameters 27

    3.3.1 Added Parameters 27

    3.3.2 Changed Parameters 27

3.3.3 Parameters changed the default value 29
3.3.4 Obsoleted Parameter 29
3.3.5 Changed parameters in recovery.conf File 29
3.4 Enhancement for SQL statement 31
3.4.1 INSERT ON CONFLICT statement 31
3.4.2 ALTER TABLE SET UNLOGGED statement 37
3.4.3 IMPORT FOREIGN SCHEMA statement 38
3.4.4 SELECT SKIP LOCKED statement 41
3.4.5 CREATE FOREIGN TABLE INHERITS statement 42
3.4.6 ALTER USER CURRENT_USER statement 44
3.4.7 UPDATE SET statement enhancement 45
3.4.8 SELECT TABLESAMPLE statement 45
3.4.9 REINDEX SCHEMA statement 46
3.4.10 REINDEX (VERBOSE) statement 47
3.4.11 ROLLUP, CUBE, GROUPING SETS statement 48
3.4.12 CREATE FOREIGN TABLE (CHECK) 48
3.4.13 Enhance CREATE EVENT TRIGGER statement 50
3.4.14 PL/pgSQL ASSERT statement 52
3.4.15 Additional operators and functions for jsonb type 53
3.4.16 Additional Functions 57
3.5 Row Level Security 60
3.5.1 What is Row Level Security 60
3.5.2 Preparation 60
3.5.3 Create POLICY object 61
3.5.4 Parameter Settings 64
3.6 BRIN Index 65
3.6.1 What is BRIN Index 65
3.6.2 Examples 65
3.6.3 Information confirming 70
3.7 Other New Features 72
3.7.1 Process Name 72
3.7.2 Output of EXPLAIN statement 73
3.7.3 Log for Replication Command 73
3.7.4 Cast Type 74
Bibliography 75
Modification History 76

1. About This Document

1.1 Purpose

The purpose of this document is to provide information of the major new features of PostgreSQL 9.5, the current Alpha 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 9.4 and PostgreSQL 9.5 Alpha 2. 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 9.4.4 (for comparison) PostgreSQL 9.5 Alpha 2 (Aug. 3, 2015, 8:41 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 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@hp.com) Hewlett-Packard Japan Co, Ltd.

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
<u>underline</u>	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

2. New Feature Overview

PostgreSQL 9.5 has many new features and improvements.

2.1 Improve Performance

Performance has been improved in the following part:

- Use 128-bit integer for the aggregate functions
- Index-Only Scan on the GiST index
- Lock reduction for BTree index
- Improve performance of the CRC calculation algorithm
- Improve performance for sort of text and numeric data type
- Improve performance for KNN-GiST
- Aggressive update of the local xmin
- Abbreviated Keys
- and so forth

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.

- Row Level Security (3.5)
- BRIN Index (3.6)
- WAL compression at the Full Page Write (3.3.1)
- Process name and the database cluster corresponding (3.7.1)
- Pg_rewind command (3.2.1)
- Improvement the various utilities (3.2)
- Added Contrib modules (3.1.5)
- Add information for EXPLAIN statement (3.7.2)
- Output log regarding replication command (3.7.3)
- Standby Actions (3.3.5)
- ASSERT statement on PL/pgSQL (3.4.14)
- Split parameters checkpoint_segments (3.3.4)
- and so forth

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.

- INSERT ON CONFLICT (3.4.1)

- ALTER TABLE SET UNLOGGED (3.4.2)
- IMPORT FOREIGN SCHEMA (3.4.3)
- SEELECT SKIP LOCKED (3.4.4)
- CREATE FOREIGN TABLE INHERITS (3.4.5)
- ALTER USER CURRENT_USER (3.4.6)
- UPDATE SET enhancement (3.4.7)
- TABLESAMPLE (3.4.8)
- REINDEX SCHEMA (3.4.9)
- REINDEX VERBOSE (3.4.10)
- GROUPING SETS, CUBE and ROLLUP (3.4.11)
- Add jsonb operators (3.4.12)
- CREATE POLICY (3.5)
- CREATE FOREIGN TABLE (CHECK) (3.4.15)
- CREATE EVENT TRIGGER enhancement (3.4.16)
- CREATE | ALTER | COMMENT ON TRANSFORM
- CREATE | ALTER DATABASE ALLOW_CONNECTIONS
- CREATE | ALTER DATABASE IS_TEMPLATE
- Add IF NOT EXISTS Clause for some statement
- and so forth

Other new features are described in the PostgreSQL 9.5 Alpha Documentation Appendix E. Release Notes (http://www.postgresql.org/docs/9.5/static/release-9-5.html).

3. New Feature Details

3.1 Architecture

3.1.1 New WAL format

XLogRecordBlockHeader structure is newly defined in src/include/access/xlogrecord.h. It is used in the output of the WAL file.

3.1.2 WAL compression

Specifying the parameter wal_compression to "on", the WAL of Full Page Write (during the first update after CHECKPOINT completed) will be compressed and written. WAL pages are compressed by a PGLZ method, which is implemented in pglz_compress functions in the source code src/common/pg_lzcompress.c.

Since the default value for the parameter wal_compression is "off", this feature does not work under normal condition.

3.1.3 Added System Catalogs

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

Table 4 Added system Catalogs

Catalog Name	Description
pg_file_settings	Parameter file and setting information
pg_policy	Row level security policies for tables
pg_policies	Row level security applied table information
pg_replication_origin	Unverified
pg_replication_origin_status	Unverified
pg_stat_ssl	Connected user information which use SSL
pg_transform	Unverified

□ pg_file_settings Catalog

Information of the loaded parameter file name (postgresql.conf, postgresql.auto.conf, etc) and parameters are stored in this catalog. The entity of this catalog is pg_show_all_file_settings function.

Table 5 pg_file_settings Catalog

Column Name	Data Type	Description
sourcefile	text	Parameter filename (Full-path)
sourceline	integer	Line number in the parameter file
seqno	integer	Serial number through the multiple files
name	text	Parameter name
setting	text	Parameter value
applied	boolean	True if the value can be applied successfully
error	text	If not null, an error message indicating why this entry could not be applied

This catalog has the following features:

- Every time search is performed for pg_file_settings catalogs, the parameter file is parsed.
- Requires SUPERUSER privileges to search.
- This catalog also supports 'include' syntax in postgresql.conf file.

Example 1 Reload parameter file

```
postgres=# ALTER SYSTEM SET max_connections = 1000 ;
ALTER SYSTEM
postgres=# SELECT sourcefile, name, setting FROM pg_file_settings    WHERE
                        name = 'max_connections' ;
                  sourcefile                  |      name       | setting
----------------------------------------+-----------------+---------
  /usr/local/pgsql/data/postgresql.auto.conf | max_connections | 1000
(1 row)
```

□ pg_policy Catalog

This catalog provide the information about POLICY objects to be used in the Row Level Security feature. This catalog is created for each database.

Table 6 pg_policy Catalog

Column Name	Data Type	Description
polname	name	The name of the policy
polrelid	oid	OID of the table to which the policy is applies
polcmd	char	The command type to which the policy is applied: ○ SELECT = r ○ INSERT = a ○ UPDATE = w ○ DELETE = d ○ ALL = *
polroles	oid[]	Array of role OID to which the policy is applied
polqual	pg_node_tree	Conditions checked in the USING clause
polwithcheck	pg_node_tree	Conditions checked in the WITH CHECK clause

□ pg_policies Catalog

In this catalog, information on the table, to which the policy has been applied, is stored. This catalog created for each database.

Table 7 pg_policies Catalog

Column Name	Data Type	Description
schemaname	name	Schema name of the policy applied table
tablename	name	Policy applied table name
policyname	name	POLICY object name
roles	name[]	Policy target roll
cmd	text	Policy target command
qual	text	WHERE clause to check the policy
with_check	text	WITH check clause

□ pg_stat_ssl Catalog

This catalog provides the SSL information about sessions that are connected to the instance.

Table 8 pg_stat_ssl Catalog

Column Name	Data Type	Description	Note
pid	integer	Backend process ID	Same as pg_stat_activity.pid
ssl	boolean	Is SSL connection?	
version	text	SSL version	
cipher	text	SSL cipher	
bits	integer	SSL bit	
compression	boolean	Is compressed?	
clientdn	text	Client DN	

□ pg_transform Catalog

This catalog stores transform information that you created in the CREATE TRANSFORM statement. Details of the CREATE TRANSFORM statement and this catalog are unverified.

Table 9 pg_transform Catalog

Column Name	Data Type	Description
trftype	oid	OID of the data type to which this transform is applied
trflang	oid	OID of the language to which this transform is applied
trffromsql	regproc	OID of the input function which converts the data type
trftosql	regproc	OID of the output function which converts the data type

3.1.4 Modified Catalogs

The following System catalogs have been changed.

Table 10 Modified catalogs

Catalog Name	Changed
pg_replication_slots	Add active_pid column
pg_settings	Add pending_restart column
pg_tables	Add rowsecurity column
pg_stat_statements	Add some statistic columns (pg_stat_statements Contrib module)
pg_authid	Add rolbypassrls column, remove rolcatupdate column

□ pg_replication_slots Catalog

Process ID column of wal sender process has been added. This makes it possible of binding to pg_stat_replication catalog.

Table 11 Added column to pg_replication_slots Catalog

Column	Data Type	Description
active_pid	integer	Process ID of wal sender process

Following example shows join of pg_stat_replication and pg_replication_slots catalog, and display the correspondence of host name and the slot name of the slave instance. Since in some cases the slot is not used, LEFT OUTER JOIN clause is used.

Example 2 Join pg_stat_replication Catalog and pg_replication_slots Catalog

```
postgres=> SELECT r.client_hostname, r.pid, s.slot_name FROM
            pg_stat_replication r
            LEFT OUTER JOIN pg_replication_slots s ON r.pid = s.active_pid ;
 client_hostname |   pid   | slot_name
-----------------+-------+-----------
 dbslv1          | 12915 | slot_1
 dbslv2          | 12934 |
(2 rows)
```

□ pg_settings Catalog

Pending_restart column has been added. In spite of this change, the parameter which wait the restart can be checked. The value of this column becomes true when the parameter file is reloaded after changing the file, or after executing ALTER SYSTEM statement. This value is not changed by executing ALTER SYSTEM statement alone. Once pending_restart column becomes true, it will not be changed if the parameter value will be set to the original value.

Table 12 Added column to pg_settings Catalog

Column Name	Data Type	Description
pending_restart	boolean	Changed in the configuration file and waiting a restart

□ pg_stat_statements Catalog

In the pg_stat_statements catalog, that is created on registering a Contrib module pg_stat_statements, the columns which contain the execution time of maximum / minimum / average / standard derivation of the SQL statements.

Table 13 Added columns to pg_stat_statements Catalog

Column Name	Data Type	Description
min_time	double precision	Minimum time spent in the statement
max_time	double precision	Maximum time spent in the statement
mean_time	double precision	Average time spent in the statement
stddev_time	double precision	Population standard deviation of time spent in the statement

□ pg_tables Catalog

Into the pg_tables catalog, which provides table information, rowsecurity column has been added. This column indicates whether Row Level Security feature is used or not. This column is true for the table of which has a setting of ENABLE ROW LEVEL SECURITY. It is false in the case of policy settings only.

Table 14 Added columns to pg_tables Catalog

Column Name	Data Type	Description
rowsecurity	boolean	True if Row Level Security is enabled on the table

3.1.5 Modified Contrib modules

In PostgreSQL 9.5, some Contrib module have been changed. Some commands widely used before have been moved from the Contrib module to PostgreSQL bin.

Table 15 Changed in Contrib modules

Module Name	Changed	Note
hstore_plperl	Newly added	
hstore_plpython	Newly added	
ltree_plpython	Newly added	
tsm_system_rows	Newly added	
tsm_system_time	Newly added	
pg_archivecleanup	Move to PostgreSQL bin	
pg_test_fsync	Move to PostgreSQL bin	
pg_test_timing	Move to PostgreSQL bin	
pg_upgrade	Move to PostgreSQL bin	
pg_xlogdump	Move to PostgreSQL bin	
pgbench	Move to PostgreSQL bin	
test_parser	Move to test/modules	Change directory
test_shm_mq	Move to test/modules	Change directory
worker_spi	Move to test/modules	Change directory
pg_stat_statements	Added some features	Add some columns
pageinspect	Added some features	Add functions for BRIN index
pgcrypto	Add function	
pg_buffercache	Added some features	

3.1.6 Interface, API, Hook

Infrastructure for expanding the PostgreSQL has been developed.

□ Parallel processing Infrastructure

APIs for performing the parallel processing has been implemented. An explanation of the way to use dynamic shared memory and worker process has been described in the "src/backend/access/transam/README.parallel" file.

□ Custom Scan/Join Interface

Features to call hook functions to replace the execution plan have been provided. The following hooks have been added.

Table 16 Added Hook

Hook	Description	Definition
set_rel_pathlist_hook	Hook function for Custom Scan	src/include/optimizer/paths.h
set_join_pathlist_hook	Hook function for Custom Join	src/include/optimizer/paths.h

3.1.7 OOM Killer support

Environment variables to control Linux OOM killer have been added. However, in Linux, it needs root user privileges to write /proc/self/oom_score_adj file.

Table 17 Environment variables for OOM Killer support

Environment Variable name	Description
PG_OOM_ADJUST_FILE	File to set the weighting
PG_OOM_ADJUST_VALUE	Value to set the weighting

3.1.8 Archiving of last WAL segment

WAL file the slave instance is written partway will be output as ".partial" extension file after standby promotion.

Example 3 Archiving of last segment

```
$ pg_ctl -D data.slave1 promote
server promoting
$
$ psql -c 'SELECT pg_switch_xlog()'
  pg_switch_xlog
----------------
  0/146D2FA8
(1 row)

$ ls -1 arch.slave1
000000010000000000000014.partial
000000020000000000000014
00000002.history
$
```

3.2 Utilities

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

3.2.1 pg_rewind

Pg_rewind command has been added in PostgreSQL 9.5.

□ Overview

Pg_rewind command is a tool to build replication environment. Unlike pg_basebackup command, this command can synchronize to an existing database cluster. It will be used to resynchronization of the promoted slave instance and the old master instance.

□ Command Parameters

Following parameters can be specified to pg_rewind command.

Table 18 Command-line parameters

Parameter Name	Description
-D / --target-pgdata	Existing data directory to modify
--source-pgdata	Source data directory to sync with
--source-server	Source server to sync with
-P / --progress	Output progress messages
-n / --dry-run	Stop before modifying anything
--debug	Output a lot of debug messages
-V / --version	Output version information, then exit
-? / --help	Output usage information

□ Conditions for executing

To run the pg_rewind command, there are some conditions. pg_rewind command is to check the conditions to run from the contents of the pg_control file of source and target. It is necessary either to specify the parameters wal_log_hints of PostgreSQL instance to "on" (the default value: "off"), or to enable the function of the data checksum. The parameters full_page_writes is also need to be set to "on" (the default value: "on").

Example 4 Error messages on parameter settings

```
$ pg_rewind --source-server='host=remhost1 port=5432 user=postgres'
                --target-pgdata=data.p -P
connected to remote server

target server need to use either data checksums or "wal_log_hints = on"
Failure, exiting
```

The instance of the target database cluster must be stopped normally.

Example 5 Error messages during target instance start

```
$ pg_rewind --source-server='host=remhost1 port=5432 user=postgres'
                --target-pgdata=data.p -P
target server must be shut down cleanly
Failure, exiting
```

Data copy procedure, as well as pg_basebackup command, use the connection to the wal sender process. It is necessary to set the max_wal_senders parameter and pg_hba.conf file of connection destination (data provider) instance.

□ Execution procedure

Pg_rewind should be executed according to the procedure listed below. In the following example, the old master instance is set to be the new slave instance using the promotion of the slave instance.

1. Setup the parameters of the master instance

   Setup the parameters in pg_hba.conf and postgresql.conf file. Reload the parameter file, if necessary.

2. Stop the target instance

   Stop the old master instance.

3. Execute pg_rewind command

   Run the pg_rewind command on the host of the old master instance. First specify with the -n parameter, and after the test, run without the -n parameter.

Example 6 Execute pg_rewind command

```
$ pg_rewind --source-server='host=remhost1 port=5432 user=postgres'
              --target-pgdata=data.p
connected to server
The servers diverged at WAL position 0/9000060 on timeline 1.
Rewinding from last common checkpoint at 0/8000060 on timeline 1
reading source file list
reading target file list
reading WAL in target
need to copy 53 MB (total source directory size is 76 MB)
54294/54294 kB (100%) copied
creating backup label and updating control file
Done!
```

4. Create the recovery.conf

   Pg_rewind command does not create recovery.conf file in the target database cluster. Therefore you should create recovery.conf file in the new slave instance.

5. Modify the postgresql.conf

   By the execution of pg_rewind command, postgresql.conf file has been overwritten and copied from the remote host. Edit the parameters if necessary.

6. Start the new slave instance

   Start the new slave instance. And check the pg_stat_replication catalog on the new master instance.

□ Exit status

Pg_rewind command returns 0 when the process is completed successfully. If it fails then exits with 1.

3.2.2 pg_ctl

The default value for the instance stop mode (-m option) has been changed to 'fast' from 'smart'.

3.2.3 vacuumdb

The --jobs parameter has been added to use multiple cores actively. You can specify the number of jobs of parallel processing for the --jobs parameter. The range of the number of the jobs is more than one, less than "macro FD_SETSIZE - 1" (in Red Hat Enterprise Linux 7 1,023 or less).

Example 7 --jobs parameter's limit

```
$ vacuumdb --jobs=-1
vacuumdb: number of parallel "jobs" must be at least 1
$ vacuumdb --jobs=1025
vacuumdb: too many parallel jobs requested (maximum: 1023)
```

□ Relationship between --jobs parameter and the number of session

When a number is specified to --jobs parameter, the same number of sessions as the number specified in the parameter are created. In case that VACUUM is performed for all database (--all specified), this command processes each database in parallel. In case that VACUUM is performed on a single database, it processes each table in parallel. The default value for this parameter is 1; it leads the same behavior as previous versions.

In the following example, 10 postgres processes have started because '—jobs=10' is specified.

Example 8 --jobs parameter and processes

```
$ vacuumdb --jobs=10 -d demodb &
vacuumdb: vacuuming database "demodb"
$ ps -ef | grep postgres
postgres 14539     1   0 10:59 pts/2   00:00:00 /usr/local/pgsql/bin/postgres -D data
postgres 14540 14539   0 10:59 ?       00:00:00 postgres: logger process
postgres 14542 14539   0 10:59 ?       00:00:00 postgres: checkpointer process
postgres 14543 14539   0 10:59 ?       00:00:00 postgres: writer process
postgres 14544 14539   0 10:59 ?       00:00:00 postgres: wal writer process
postgres 14545 14539   0 10:59 ?       00:00:00 postgres: stats collector process
postgres 14569 14539   6 11:00 ?       00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14570 14539   0 11:00 ?       00:00:00 postgres: postgres demodb [local] idle
postgres 14571 14539   5 11:00 ?       00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14572 14539   7 11:00 ?       00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14573 14539   0 11:00 ?       00:00:00 postgres: postgres demodb [local] idle
postgres 14574 14539   0 11:00 ?       00:00:00 postgres: postgres demodb [local] idle
postgres 14575 14539   9 11:00 ?       00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14576 14539   5 11:00          00:00:00 postgres: postgres demodb [local] VACUUM
postgres 14577 14539   0 11:00 ?       00:00:00 postgres: postgres demodb [local] idle
postgres 14578 14539   1 11:00 ?       00:00:00 postgres: postgres demodb [local] idle
```

It the value specified in the --jobs parameter is larger than the number of --table parameters, the upper limit of the degree of parallelism is the number of tables. In addition, since the PostgreSQL max_connections parameter is not considered in the calculation of the number of sessions, in case that the excess of the number of sessions is detected, "FATAL: sorry, too many clients already" error will occur.

3.2.4 pg_dump

The following enhancements have been implemented.

□ Add --enable-row-security parameter

At default, pg_dump command gets a dump by specifying the row_security parameter to "off". Data acquisition will result in an error in case of connecting with user privileges that cannot bypass the Row Level Security. By specifying the --enable-row-security parameter, the dump file contains only the data that has the authority contained. Error in Alpha 2 will occur.

□ Add --snapshot parameter

Specify the snapshot ID created with pg_export_snapshot function. You can get a dump using the specified snapshot ID.

□ Enhance output at the --verbose parameter

Schema name is included in the log that is output when you specify the --verbose parameter.

□ Obsolete --ignore-version parameter

--ignore-version parameter was made obsolete. pg_dumpall command, pg_restore command has also been modified as well.

3.2.5 psql

The following features have been added to the psql command.

□ pager_min_lines parameter

Pager_min_lines parameter has been added. If this parameter is specified, pager is not used when the number of rows of output is less than the specified value. The default value is 0, and pager works in accordance with the number of rows of the terminal. This setting did not work at the author environment.

Example 9 Setting the pager_min_lines

```
postgres=> \pset pager_min_lines 20
Pager won't be used for less than 20 lines
```

□ \set ECHO errors

If you specify a \set ECHO errors, statement which you are trying to run is displayed in case of an error.

Example 10 \set ECHO errors

```
postgres=> SELECT * FROM notexists1 ;
ERROR:  relation "notexists1" does not exist
LINE 1: SELECT * FROM notexists1;
                      ^
postgres=> \set ECHO errors
postgres=> SELECT * FROM notexists1 ;
ERROR:  relation "notexists1" does not exist
LINE 1: SELECT * FROM notexists1;
                      ^
STATEMENT:  SELECT * FROM notexists1 ;
```

□ Display schema name error

A message is added to indicate the error occurrence location in the case a schema name which does not exist is specified.

Example 11 Error message

```
postgres=> CREATE TABLE badschema.table1(c1 NUMERIC, c2 VARCHAR(10)) ;
ERROR:  schema "badschema" does not exist
LINE 1: CREATE TABLE badschema.table1(c1 NUMERIC, c2 VARCHAR(10)) ;
```

□ \watch output \timing information

The output of the \watch command, are now added to the results of \timing command.

Example 12 \watch output \timing information

```
postgres=> \timing
Timing is on.
demodb=> \watch 1
Watch every 1s    Fri Aug    7 11:56:58 2015


                    now
-------------------------------
  2015-08-07 11:56:59.931996+09
(1 row)


Time: 0.458 ms
```

□ Backslash Command

The following enhancements have been implemented. Require superuser privilege for \db+ command. In previous version, general user can execute \db+ command.

Table 19 Added / modified backslash command

Command	Modification	Description
\db+	Additional information	The Size of the tablespace has been added.
\dT+	Additional information	The data type of the owner has been added.

3.2.6 pgbench

Some new features have been provided for pgbench command.

□ --latency-limit parameter

Parameter --latency-limit (-L) can now be specified in the parameters of the pgbench command. If you specify a value in milliseconds for this parameter, the transaction rate of which the duration is shorter than the specified time is displayed.

Example 13 Output of –latency-limit parameter

```
$ pgbench -c 10 -d pgbench -U pgbench -L 10
pghost:    pgport:    nclients: 10 nxacts: 10 dbName: pgbench
starting vacuum...end.
……………………………………………..
number of transactions actually processed: 100/100
number of transactions above the 10.0 ms latency limit: 64 (64.000 %)
latency average: 17.065 ms
latency stddev: 12.816 ms
tps = 479.927051 (including connections establishing)
tps = 546.603406 (excluding connections establishing)
```

□ Behavior change at the specified -f parameter

Behavior of the command in case of using a custom script (-f option), and not specifying the -n option, has been changed. In previous versions, VACUUM is performed to each table which pgbench created, and ended the command if the table does not exist. In PostgreSQL 9.5, it has been changed to continue the processing the custom scripts even if VACUUM process fails.

Example 14 Behavior at the specified -f parameter

```
$ pgbench -f bench.sql –U user1 postgres
starting vacuum...ERROR:    relation "pgbench_branches" does not exist
(ignoring this error and continuing anyway)
ERROR:    relation "pgbench_tellers" does not exist
(ignoring this error and continuing anyway)
ERROR:    relation "pgbench_history" does not exist
(ignoring this error and continuing anyway)
end.
transaction type: Custom query
…………………………………………………………
```

Some of the new features other than the above has been implemented to pgbench command.

3.2.7 pg_receivexlog

To the pg_receivexlog command, parameters for controlling the replication slot was added. Slot name is specified using --slot parameters as in the previous version.

□ --create-slot

Create a slot with the name specified in –slot parameter, and continues to receive WAL data. If there is no margin in the parameter max_replication_slots of the PostgreSQL instance, or if the slot having the same name already exists, then an error occurs.

□ --drop-slot

Remove the specified slot, and exit the command.

□ --synchronous

Flush the WAL data to disk immediately after it has been received

3.2.8 reindexdb

"-v" "-verbose" option has been added to output a detailed information. "REINDEX (VERBOSE) DATABASE database_name" statement is executed internally.

Example 15 reindexdb -v Option

```
$ reindexdb –v demodb
INFO:    index "pg_class_oid_index" was reindexed
DETAIL:    CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:    index "pg_class_relname_nsp_index" was reindexed
DETAIL:    CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:    index "pg_class_tblspc_relfilenode_index" was reindexed
DETAIL:    CPU 0.00s/0.00u sec elapsed 0.00 sec.
………………………………..
```

3.3 Changes of parameters

The following parameters have been changed in PostgreSQL 9.5.

3.3.1 Added Parameters

The following parameters have been added.

Table 20 Added Parameters

Parameter Name	Description	Default Value
cluster_name	The name that is included in process titles	''
gin_pending_list_limit	Maximum size of the GIN pending list	4MB
row_security	Enable Row Level Security feature. available value: on, off and force	on
track_commit_timestamp	Output the last commit time of the transaction to WAL	off
wal_compression	WAL compression at the full page write	off
log_replication_commands	Output log about replication command	off
max_wal_size	WAL size to start the checkpoint; the range of values: 2 - 2147483647	1GB
min_wal_size	WAL size to start the recycling the WAL files; the range of values: 2 - 2147483647	80MB
operator_precedence_warning	Output a warning to the SQL that has a possibility of changing that the result due to the change in priorities	off
wal_retrieve_retry_interval	The re-acquisition interval of the WAL data in milliseconds	5s

□ wal_retrieve_retry_interval parameter

Wal_retrieve_retry_interval has been parameterized from the fixed value setting in the previous version. The default value of 5 seconds is the same behavior as the previous version.

3.3.2 Changed Parameters

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

Table 21 Changed Parameters

Parameter Name	Modification
log_autovacuum_min_duration	Can be specified for each table
archive_mode	"always" can be specified
trace_sort	The output of the additional information
debug_assertions	Became read-only
local_preload_libraries	Allow set by ALTER ROLE SET statement

□ log_autovacuum_min_duration

Parameter log_autovacuum_min_duration can now be specified for each table.

Example 16 Setting parameter for each table

```
postgres=> ALTER TABLE vacuum1 SET (log_autovacuum_min_duration = 1000) ;
ALTER TABLE
postgres=> \d+ vacuum1
                              Table "public.vacuum1"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+--------
 c1     | numeric               |           | main     |              |
 c2     | character varying(10) |           | extended |              |
Options: log_autovacuum_min_duration=1000
```

□ archive_mode

"always" has been added to the choice of the parameter archive_mode. There is no difference between "on" and "always" except the slave instance in the replication environment. If this parameter is specified to "always" in the slave instance, archiver process is started, and the archive log is output.

3.3.3 Parameters changed the default value

The default values of the following parameters have been changed.

Table 22 Parameters the default value of which is changed

Parameter Name	PostgreSQL 9.4	PostgreSQL 9.5
server_version	9.4.4	9.5alpha2
server_version_num	90404	90500
search_path	"$user",public	"$user", public

3.3.4 Obsoleted Parameter

The following parameters are obsoleted.

□ Obsoleted checkpoint_segments

Parameter checkpoint_segments, which determine the occurrence time of the checkpoint, is obsolete and has been divided into parameter max_wal_size and min_wal_size.

Table 23 Alternative parameters for checkpoint_segments

Parameter Name	Description	Default Value
max_wal_size	Maximum size to let the WAL grow to between automatic WAL checkpoints.	1GB
min_wal_size	As long as WAL disk usage stays below this setting.	80MB

In previous version, parameters checkpoint_segments, was used for both starting the checkpoint and recycling the WAL file. Therefore, when the interval of the checkpoint was extended, a lot of WAL file was created. This feature has been changed so as to control the two functions by separate parameters in the PostgreSQL 9.5.

□ Obsoleted ssl_renegotiation_limit

Parameter ssl_renegotiation_limit is obsoleted.

3.3.5 Changed parameters in recovery.conf File

The following parameters of the recovery.conf file has been changed.

□ Added recovery_target_action parameter

Parameter recovery_target_action has been added. This parameter has extended the pause_at_recovery_target parameter of the previous version more generally. The operation in case it reaches the recovery target can be specified.

You can specify the following values.

- pause (default value)

  Wait in a state where instance has reached the target.
- promote

  Complete the recovery, and accept the user's connection.
- shutdown

  Complete the recovery, and stop the instance. Renaming recovery.conf file will not be performed.

□ Obsoleted parameter

Parameter pause_at_recovery_target has been removed. As an alternative parameter, above-mentioned recovery_target_action is used. When the parameter pause_at_recovery_target is specified in recovery.conf file, the following message is logged and instance cannot be started.

Example 17 Error by pause_at_recovery_target setting

```
$ cat data/recovery.conf
restore_command = 'cp /usr/local/pgsql/arch/%f %p'
pause_at_recovery_target = on
$
$ pg_ctl -D data -w start
waiting for server to start....LOG:  redirecting log output to logging collector process
HINT:  Future log output will appear in directory "pg_log".
.... stopped waiting
pg_ctl: could not start server
Examine the log output
$
$ cat data/pg_log/postgresql-2015-08-07_111751.log
LOG:  database system was shut down at 2015-08-07 11:17:20 JST
FATAL:  unrecognized recovery parameter "pause_at_recovery_target"
LOG:  startup process (PID 12233) exited with exit code 1
LOG:  aborting startup due to startup process failure
```

3.4 Enhancement for SQL statement

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

3.4.1 INSERT ON CONFLICT statement

When INSERT statement is executed under condition that if becomes a constraint violation, automatic switch to the UPDATE statement is now available (called UPSERT statement). To use this feature, specify the ON CONFLICT clause in the INSERT statement.

Syntax

```
INSERT INTO …
ON CONFLICT [ { (column_name, …) | ON CONSTRAINT constraint_name }]
{ DO NOTHING | DO UPDATE SET column_name = value }
[ WHERE … ]
```

In the ON CONFLICT clause, specify where the constraint violation will occur.

- Specify a list of column names, or a constraint name in the syntax of the "ON CONSTRAINT constraint_name".
- In case of specifying constraints consisting of multiple columns, it is necessary to specify all of the column names that are included in the constraint.
- If you omit the par after ON CONFLICT, all of the constraint violation is checked. This omission is available only in case of using DO NOTHING clause.
- When the constraint violation occurs for other than the specified columns or constraints in the ON CONFLICT clause, INSERT statement will result in an error.

After the ON CONFLICT clause, the behavior when constraint violation has occurred is described. If you specify a DO NOTHING clause, nothing is done for constraint violations occurred (constraint violation does not occur). If you specify a DO UPDATE clause, UPDATE to the specific columns is executed. Following is the execution examples.

Example 18 Prepare table and data

```
postgres=> CREATE TABLE upsert1 (key NUMERIC, val VARCHAR(10)) ;
CREATE TABLE
postgres=> ALTER TABLE upsert1 ADD CONSTRAINT pk_upsert1 PRIMARY KEY (key) ;
ALTER TABLE
postgres=> INSERT INTO upsert1 VALUES (100, 'Val 1') ;
INSERT 0 1
postgres=> INSERT INTO upsert1 VALUES (200, 'Val 2') ;
INSERT 0 1
postgres=> INSERT INTO upsert1 VALUES (300, 'Val 3') ;
INSERT 0 1
```

The following is a description example of the ON CONFLICT clause. Since DO NOTHING is specified in the handling part, nothing is done even if constraint violation occurs.

Example 19 ON CONFLICT clause samples

```
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
           ON CONFLICT DO NOTHING ; -- omit constraint name or column
INSERT 0 0
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
           ON CONFLICT(key) DO NOTHING ; -- set column name
INSERT 0 0
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
           ON CONFLICT(val) DO NOTHING ; -- error when no constraint column set
ERROR:     there is no unique or exclusion constraint matching the ON CONFLICT specification
postgres=> INSERT INTO upsert1 VALUES (200, 'Update 1')
           ON CONFLICT ON CONSTRAINT pk_upsert1 DO NOTHING ; -- set constraint
INSERT 0 0
```

In the DO UPDATE clause, you describes the update process. This is basically the same as the part subsequent the SET clause in the UPDATE statement. If you use an alias EXCLUDED, you can access the records that could not be stored when INSERT statement is executed.

Example 20 DO UPDATE clause samples

```
postgres=> INSERT INTO upsert1 VALUES (400, 'Upd4')
              ON CONFLICT DO UPDATE SET val = EXCLUDED.val ; -- No constraint error
ERROR:    ON CONFLICT DO UPDATE requires inference specification or constraint name
LINE 2: ON CONFLICT DO UPDATE SET val = EXCLUDED.val;
              ^
HINT:    For example, ON CONFLICT ON CONFLICT (<column>).
postgres=> INSERT INTO upsert1 VALUES (300, 'Upd3')
     ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val ; -- Use EXCLUDED alias
INSERT 0 1
postgres=> INSERT INTO upsert1 VALUES (300, 'Upd3')
              ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val WHERE upsert1.key = 100 ;
INSERT 0 0 -- Can determin UPDATE conditions when specify the WHERE clause
```

□ Relation between ON CONFLICT Clause and Trigger

How the trigger works during the execution of INSERT ON CONFLICT statement is verified. BEFORE INSERT trigger has always been executed. If the record is updated by DO UPDATE statement, BEFORE INSERT trigger and BEFORE / AFTER UPDATE trigger worked. Only BEFORE INSERT trigger was executed if the UPDATE is not performed by the WHERE clause.

Table 24 Executed triggers

Triggers	Success INSERT	DO NOTHING	DO UPDATE (UPDATED)	DO UPDATE (NO UPDATE)
BEFORE INSERT	Execute	Execute	Execute	Execute
AFTER INSERT	Execute	-	-	-
BEFORE UPDATE	-	-	Execute	-
AFTER UPDATE	-	-	Execute	-

□ ON CONFLICT clause and Execution Plan

By executing ON CONFLICT clause, the execution plan will change. When you run the EXPLAIN statement, Conflict Resolution, Conflict Arbiter Indexes, and Conflict Filter appear in the execution plan. Actual output is shown in the following example.

Example 21 ON CONFLICT Clause and Execution Plan

```
postgres=> EXPLAIN INSERT INTO upsert1 VALUES (200, 'Update 1')
              ON CONFLICT(key) DO NOTHING ;
                                     QUERY PLAN
----------------------------------------------------
 Insert on upsert1   (cost=0.00..0.01 rows=1 width=0)
   Conflict Resolution: NOTHING
   Conflict Arbiter Indexes: pk_upsert1
   ->   Result   (cost=0.00..0.01 rows=1 width=0)
(4 rows)
postgres=> EXPLAIN INSERT INTO upsert1 VALUES (400, 'Upd4')
              ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val ;
                                     QUERY PLAN
----------------------------------------------------
 Insert on upsert1   (cost=0.00..0.01 rows=1 width=0)
   Conflict Resolution: UPDATE
   Conflict Arbiter Indexes: pk_upsert1
   ->   Result   (cost=0.00..0.01 rows=1 width=0)
(4 rows)
postgres=> EXPLAIN INSERT INTO upsert1 VALUES (400, 'Upd4')
              ON CONFLICT(key) DO UPDATE SET val = EXCLUDED.val WHERE upsert1.key = 100 ;
                                     QUERY PLAN
----------------------------------------------------
 Insert on upsert1   (cost=0.00..0.01 rows=1 width=0)
   Conflict Resolution: UPDATE
   Conflict Arbiter Indexes: pk_upsert1
   Conflict Filter: (upsert1.key = '100'::numeric)
   ->   Result   (cost=0.00..0.01 rows=1 width=0)
(5 rows)
```

Current version does not support ON CONFLICT DO UPDATE statement to the remote instance using postgres_fdw module.

□ ON CONFLICT clause and the partition table

ON CONFLICT clause for partition table using the INSERT trigger it will be ignored.

Example 22 INSERT ON CONLICT statement for the partition table#1

```
postgres=> CREATE TABLE main1 (key1 NUMERIC, val1 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE TABLE main1_part100 (CHECK(key1 < 100)) INHERITS (main1) ;
CREATE TABLE
postgres=> CREATE TABLE main1_part200 (CHECK(key1 >= 100 AND key1 < 200))
        INHERITS (main1) ;
CREATE TABLE
postgres=> ALTER TABLE main1_part100 ADD CONSTRAINT pk_main1_part100
        PRIMARY KEY (key1);
ALTER TABLE
postgres=> ALTER TABLE main1_part200 ADD CONSTRAINT pk_main1_part200
        PRIMARY KEY (key1);
ALTER TABLE
postgres=> CREATE OR REPLACE FUNCTION func_main1_insert()
        RETURNS TRIGGER AS $$
         BEGIN
           IF    (NEW.key1 < 100) THEN
                 INSERT INTO main1_part100 VALUES (NEW.*) ;
           ELSIF (NEW.key1 >= 100 AND NEW.key1 < 200) THEN
                 INSERT INTO main1_part200 VALUES (NEW.*) ;
           ELSE
                  RAISE EXCEPTION 'ERROR! key1 out of range.' ;
           END IF ;
           RETURN NULL ;
         END ;
        $$ LANGUAGE 'plpgsql';
CREATE FUNCTION
```

Example 23 INSERT ON CONLICT statement for the partition table#2

```
postgres=> CREATE TRIGGER trg_main1_insert BEFORE INSERT ON main1
             FOR EACH ROW EXECUTE PROCEDURE func_main1_insert() ;
CREATE TRIGGER
postgres=> INSERT INTO main1 VALUES (100, 'DATA100') ;
INSERT 0 0
postgres=> INSERT INTO main1 VALUES (100, 'DATA100') ;
ERROR:  duplicate key value violates unique constraint "pk_main1_part200"
DETAIL:  Key (key1)=(100) already exists.
CONTEXT:  SQL statement "INSERT INTO main1_part200 VALUES (NEW.*)"
PL/pgSQL function func_main1_insert() line 6 at SQL statement
postgres=> INSERT INTO main1 VALUES (100, 'DATA100')
        ON CONFLICT DO NOTHING ;
ERROR:  duplicate key value violates unique constraint "pk_main1_part200"
DETAIL:  Key (key1)=(100) already exists.
CONTEXT:  SQL statement "INSERT INTO main1_part200 VALUES (NEW.*)"
PL/pgSQL function func_main1_insert() line 6 at SQL statement
```

3.4.2 ALTER TABLE SET UNLOGGED statement

When the update transaction occurs on the table, change log is written to WAL. In the standard setting COMMIT statement from the user will wait until the writing to WAL completes. In previous versions of PostgreSQL, if reliability is not important, we were able to create a table that does not perform the writing of the WAL, using the CREATE UNLOGGED TABLE statement. In PostgreSQL 9.5, the control of WAL writing comes to be modified to each table.

Syntax - To UNLOGGED TABLE

```
ALTER TABLE table_name SET UNLOGGED
```

Syntax - To ordinary (LOGGED) TABLE

```
ALTER TABLE table_name SET LOGGED
```

Example 24 Change to UNLOGGED TABLE

```
postgres=> CREATE TABLE logtbl1 (c1 NUMERIC, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> \d+ logtbl1
                          Table "public.logtbl1"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+------------
 c1     | numeric               |           | main     |              |
 c2     | character varying(10) |           | extended |              |

postgres=> ALTER TABLE logtbl1 SET UNLOGGED ;
ALTER TABLE
postgres=> \d+ logtbl1
                     Unlogged table "public.logtbl1"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+------------
 c1     | numeric               |           | main     |              |
 c2     | character varying(10) |           | extended |              |
```

Using the \d+ command, you can see that the normal table is changed to UNLOGGED table.

□ Implementation

Internally, a new UNLOGGED TABLE (or TABLE) with the same structure is created, and the data are copied. relfilenode columns and relpersistence column of the pg_class catalog is changed.

Example 25 Change of pg_class Catalog

```
postgres=> SELECT relname, relfilenode, relpersistence FROM pg_class WHERE
               relname='logtbl1' ;
  relname | relfilenode | relpersistence
---------+-------------+----------------
  logtbl1 |         16483 | p
(1 row)

postgres=> ALTER TABLE logtbl1 SET UNLOGGED ;
ALTER TABLE
postgres=> SELECT relname, relfilenode, relpersistence FROM pg_class WHERE
               relname='logtbl1' ;
  relname | relfilenode | relpersistence
---------+-------------+----------------
  logtbl1 |         16489 | u
(1 row)
```

□ Changes to INHERIT table

If you change the setting of the inheritance table, each table maintains the independent setting. Even if one of the child table is set to UNLOGGED, the other tables are not affected.

3.4.3 IMPORT FOREIGN SCHEMA statement

In order to access the external table using the FOREIGN DATA WRAPPER, create an external table using the CREATE FOREIGN TABLE statement after you create a SERVER and USER MAPPING. In previous versions, CREATE FOREIGN TABLE statement is executed for each remote reference table, and it is necessary to define the definition of the same column as the remote table again. IMPORT FOREIGN SCHEMA statement performs this in a collectively per schema. IMPORT FOREIGN SCHEMA statement can import not only the table, also views and materialized views that are stored in the schema of the remote instance.

Syntax

```
IMPORT FOREIGN SCHEMA
[ { LIMIT TO | EXCEPT } (table_name, …) ]
FROM SERVER server_name
INTO local_schema
[ OPTIONS (option 'value' , …) ]
```

Table 25 Syntax for IMPORT FOREIGN SCHEMA statement

Clause	Description
LIMIT TO EXCEPT	Used to limit the table to be imported. If omitted import all tables
FROM SERVER	Specify the SERVER object that represents the remote instance witch perform the import.
INTO	Specify the schema name of the local instance which perform import.

Example 26 Execution of the IMPORT FOREIGN SCHEMA statement

```
postgres=# CREATE EXTENSION postgres_fdw ;
CREATE EXTENSION
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw
           OPTIONS (host '192.168.1.200', port '5432', dbname 'demodb') ;
CREATE SERVER
postgres=# CREATE USER MAPPING FOR public SERVER remsvr1
           OPTIONS (user 'user1', password 'secret') ;
CREATE USER MAPPING
postgres=# CREATE SCHEMA schema2 ;
CREATE SCHEMA
postgres=# IMPORT FOREIGN SCHEMA schema1 FROM SERVER remsvr1 INTO schema2 ;
IMPORT FOREIGN SCHEMA
```

- Schema to store the imported FOREIGN TABLE must be created in advance.
- In the case that the table, which has the same name as the remote schema, exists in the local schema specified to import, error occurs and FOREIGN TABLE is not created.
- If the remote instance does not require a password (trust), IMPORT FOREIGN SCHEMA statement issued by a general user will fail.

Example 27 Behavior that cause an error

```
postgres=# IMPORT FOREIGN SCHEMA public FROM SERVER remsvr1 INTO schemax ;
ERROR:    schema "schemax" does not exist    -- No Local Schema

postgres=# IMPORT FOREIGN SCHEMA schema1 FROM SERVER remsvr1 INTO schema1 ;
ERROR:    relation "data1" already exists    -- Exists same name table
CONTEXT:    importing foreign table "table2"

postgres=> IMPORT FOREIGN SCHEMA public FROM SERVER remsvr1 INTO schema1 ;
ERROR:    password is required
DETAIL:    Non-superuser cannot connect if the server does not request a password.
HINT:    Target server's authentication method must be changed.
```

□ OPTIONS clause

Following options can be specified in the IMPORT FOREIGN SCHEMA statement with OPTIONS clause.

Table 26 IMPORT FOREIGN SCHEMA statement OPTIONS clause

Option Name	Default	Description
import_collate	true	Import COLLATE clause
import_default	false	Import DEFAULT clause
import_not_null	true	Import NOT NULL constraint

If DEFAULT clause that uses sequence objects to the remote table has been used with the import_default option set "true", IMPORT FOREIGN SCHEMA statement becomes to error.

Example 28 DEFAULT with Sequence

```
(remote) postgres=> CREATE SEQUENCE seq1 ;
CREATE SEQUENCE
(remote)postgres=> CREATE TABLE data1(c1 NUMERIC DEFAULT nextval('seq1')) ;
CREATE TABLE

(local) postgres=# IMPORT FOREIGN SCHEMA public FROM SERVER remsvr2 INTO schema1
            OPTIONS (import_default 'true') ;
ERROR:   relation "public.seq1" does not exist
CONTEXT:   importing foreign table "data1"
```

3.4.4 SELECT SKIP LOCKED statement

When performing an access to the tuple holding the conflicting lock, SQL statement usually waits until the lock is released. NOWAIT clause can be used as a way to generate an error without waiting. In PostgreSQL 9.5, it is possible to perform search skipping a locked tuple. To use this feature, specify the SKIP LOCKED clause in the SELECT statement.

Example 29 Prepare data and lock tuple

```
postgres=> CREATE TABLE lock1 (key NUMERIC, val TEXT) ;
CREATE TABLE
postgres=> INSERT INTO lock1 VALUES (generate_series(1, 2000), 'initial') ;
INSERT 0 2000
postgres=> BEGIN ;
BEGIN
postgres=> SELECT * FROM lock1 WHERE key = 1000 FOR SHARE ;
 key  |   val
------+---------
 1000 | initial
(1 row)
```

The above example create a table and locked a tuple of the "key = 1000" with the FOR SHARE clause. In the following example, we performed a search using SKIP LOCKED clause in the other session. Since tuple of the "key = 1000" conflicts with SELECT statement specifying the FOR UPDATE clause, it becomes wait state under the standard condition, or it causes an error in case that NOWAIT is specified. By specifying the SKIP LOCKED clause, tuples other than "key = 1000" has been successfully retrieved.

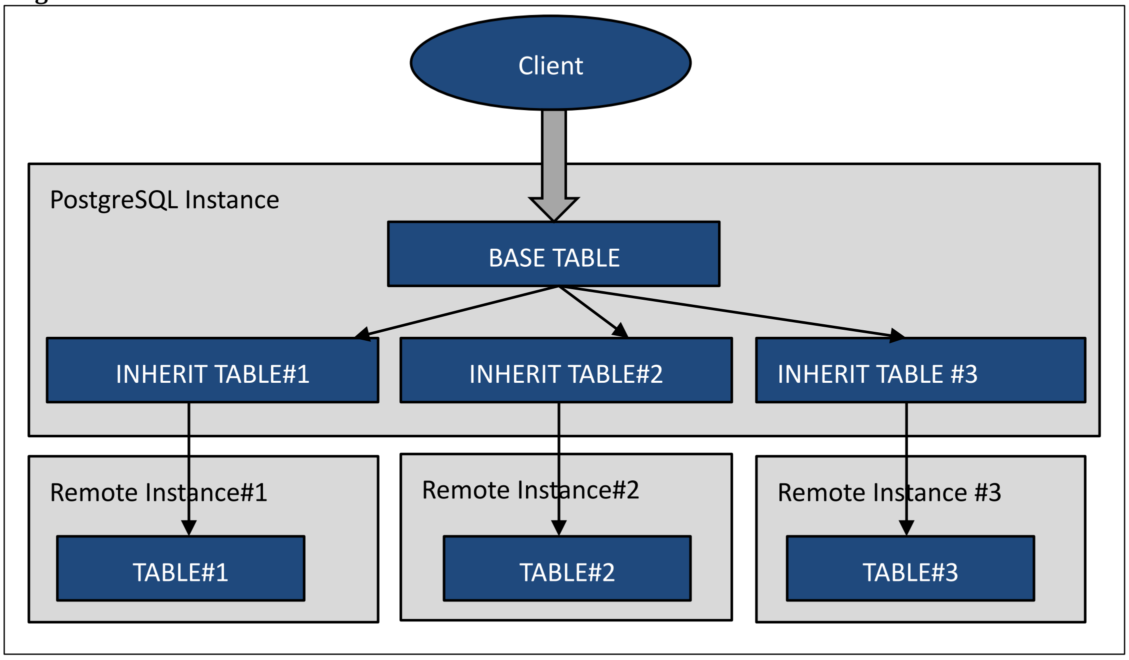
Example 30 SELECT by the other session

```
postgres=> SELECT * FROM lock1 WHERE key IN (999, 1000, 1001) FOR UPDATE SKIP LOCKED ;
  key    |      val
------+---------
    999 | initial
  1001 | initial
(2 rows)
```

3.4.5 CREATE FOREIGN TABLE INHERITS statement

As inheritance table of an existing table, you can create an external table (FOREIGN TABLE). This feature makes it possible to distribute the processing to multiple hosts with a combination of partitioning and external table.

Figure 1 FOREIGN TABLE INHERIT



Syntax

```
CREATE FOREIGN TABLE table_name (check_constraints …)
INHERITS (parent_table)
SERVER server_name
OPTIONS (option = 'value' …)
```

INHERITS clause is different from the previous version. Parent table is specified in INHERITS clause. Example of the implementation and the verification of the execution plan are shown below.

Example 31 Create parent table

```
postgres=> CREATE TABLE parent1(key NUMERIC, val TEXT) ;
CREATE TABLE
```

Create a table (inherit1, inherit2, inherit3) on the remote instance.

Example 32 Create child table on the remote instance

```
postgres=> CREATE TABLE inherit1(key NUMERIC, val TEXT) ;
CREATE TABLE
```

Execute the CREATE FOREIGN TABLE statement to create an external table to the table (inherit1, inherit2, inherit3) on each remote instance.

Example 33 Create External Tables

```
postgres=# CREATE EXTENSION postgres_fdw ;
CREATE EXTENSION
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw
            OPTIONS (host 'remsvr1', dbname 'demodb', port '5432') ;
CREATE SERVER
postgres=# CREATE USER MAPPING FOR public SERVER remsvr1
            OPTIONS (user 'demo', password 'secret') ;
CREATE USER MAPPING
postgres=# GRANT ALL ON FOREIGN SERVER remsvr1 TO public ;
GRANT
postgres=> CREATE FOREIGN TABLE inherit1(CHECK(key < 1000))
            INHERITS (parent1) SERVER remsvr1 ;
CREATE FOREIGN TABLE
```

The changes from the previous versions are: CREATE FOREIGN TABLE statement specified CHECK constraint rather than column definition, and INHERITS clause specified the original table. The above is an example of only one instance, but in fact, you should execute CREATE SERVER statement, CREATE USER MAPPING statement, and CREATE FOREIGN TABLE statement for multiple instances.

Example 34 Check FOREIGN TABLE Definition with INHERITS Clause

```
postgres=> \d+ inherit2
                              Foreign table "public.inherit2"
 Column |  Type   | Modifiers | FDW Options | Storage  | Stats target | Description
--------+---------+-----------+-------------+----------+--------------+------------
 key    | numeric |           |             | main     |              |
 val    | text    |           |             | extended |              |
Check constraints:
    "inherit2_key_check" CHECK (key >= 1000::numeric AND key < 2000::numeric)
Server: remsvr2
Inherits: parent1
```

When you check the execution plan, you can see that SQL access only to a specific instance due to CHECK constraints.

Example 35 Check the Execution Plan

```
postgres=> EXPLAIN SELECT * FROM parent1 WHERE key = 1500 ;
                              QUERY PLAN
---------------------------------------------------------------------
 Append  (cost=0.00..121.72 rows=6 width=64)
   ->  Seq Scan on parent1  (cost=0.00..0.00 rows=1 width=64)
         Filter: (key = '1500'::numeric)
   ->  Foreign Scan on inherit2  (cost=100.00..121.72 rows=5 width=64)
(4 rows)
```

3.4.6 ALTER USER CURRENT_USER statement

Now it is possible to specify the CURRENT_USER and CURRENT_ROLE for the user name in the ALTER USER / ALTER ROLE statement. You can execute ALTER USER statement for several purpose such as password changes of the connecting user.

Example 36 CURRENT_USER Clause

```
postgres=> ALTER USER CURRENT_USER PASSWORD 'secret' ;
ALTER ROLE
```

3.4.7 UPDATE SET statement enhancement

In UPDATE statement which uses the result of SELECT, the FROM clause are no longer required.

Example 37 Preparation of tables

```
postgres=> CREATE TABLE upd1(c1 NUMERIC, c2 NUMERIC, c3 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE TABLE upd2(c1 NUMERIC, c2 NUMERIC, c3 VARCHAR(10)) ;
CREATE TABLE
```

This will update the value of c2, c3 column of the table upd2 with c2, c3 column of the table upd1. At that time, the tuple which has the same value in c1 column is used.

Example 38 Syntax up to PostgreSQL 9.4

```
postgres=> UPDATE upd2 SET c2 = upd1.c2, c3 = upd1.c3
             FROM (SELECT * FROM upd1) AS upd1
             WHERE upd1.c1 = upd2.c1 ;
UPDATE 2
```

Example 39 Syntax for PostgreSQL 9.5

```
postgres=> UPDATE upd2 SET (c2, c3) =
             (SELECT c2, c3 FROM upd1 WHERE upd1.c1 = upd2.c1) ;
UPDATE 2
```

3.4.8 SELECT TABLESAMPLE statement

TABLESAMPLE clause, sampling a certain percentage of the record from the table, is now available.

Syntax

```
SELECT … FROM table_name …
TABLESAMPLE {SYSTEM | BERNOULLI} (percent)
[ REPEATABLE (seed) ]
```

You can choose SYSTEM or BERNOULLI as sampling method. SYSTEM uses the tuple entire block sampled. BERNOULLI choose a more constant rate of tuples from sampled block. In 'percent' you specify the sampling percentage (1-100). When you specify the value other than 1 - 100 SELECT statement results in an error. REPEATABLE clause is optional the seed of sampling can be specified.

□ Execution Plan

Execution plan of TABLESAMPLE is the followings. The cost increases when BERNOULLI is specified.

Example 40 Execution Plan for TABLESAMPLE SYSTEM

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 TABLESAMPLE SYSTEM (10) ;
                                    QUERY PLAN
--------------------------------------------------------------------------------
Aggregate   (cost=341.00..341.01 rows=1 width=0) (actual time=4.914..4.915 rows=1 loops=1)
   ->   Sample  Scan  (system)  on  data1    (cost=0.00..316.00  rows=10000  width=0)
   (actualtime=0.019..3.205 rows=10090 loops=1)
 Planning time: 0.106 ms
 Execution time: 4.977 ms
(4 rows)
```

Example 41 Execution Plan for TABLESAMPLE BERNOULLI

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 TABLESAMPLE
                        BERNOULLI (10) ;
                                   QUERY PLAN
--------------------------------------------------------------------------------
 Aggregate   (cost=666.00..666.01 rows=1 width=0) (actual time=13.654..13.655 rows=1 loops=1)
   ->  Sample Scan (bernoulli) on data1   (cost=0.00..641.00 rows=10000 width=0) (actual
   time=0.013..12.121 rows=10003 loops=1)
 Planning time: 0.195 ms
 Execution time: 13.730 ms
(4 rows)
```

3.4.9 REINDEX SCHEMA statement

It is now possible to specify the SCHEMA clause to REINDEX statement. All indexes in the schema will be re-created. If there is an index of other users own in the schema, the indexes to which the execution user of REINDEX can access are reconstructed.

Example 42 REINDEX SCHEMA statement

```
postgres=> REINDEX SCHEMA schema1 ;
REINDEX
postgres=> REINDEX SCHEMA schema2 ;
ERROR:    must be owner of schema schema2
postgres=> BEGIN ;
BEGIN
postgres=> REINDEX SCHEMA schema1 ;
ERROR:    REINDEX SCHEMA cannot run inside a transaction block
postgres=>
```

To execute REINDEX SCHEMA statement, the following conditions are required.

- Being the owner of the schema.
- Run within a transaction is impossible.

3.4.10 REINDEX (VERBOSE) statement

To REINDEX statement, VERBOSE clause can now be specified to output a detailed message. Similarly. The -v option has been added to reindexdb command.

Syntax

```
REINDEX (VERBOSE) {TABLE | DATABASE | SCHEMA} ...
```

Example 43 REINDEX (VERBOSE)

```
postgres=> REINDEX (VERBOSE) TABLE data1 ;
INFO:    index "idx1_data1" was reindexed
DETAIL:    CPU 0.00s/0.00u sec elapsed 0.02 sec.
INFO:    index "pg_toast_16424_index" was reindexed
DETAIL:    CPU 0.00s/0.00u sec elapsed 0.00 sec.
REINDEX
```

3.4.11 ROLLUP, CUBE, GROUPING SETS statement

In SELECT statement, ROLLEUP, CUBE and GROUPING SETS clause is now available to calculate the subtotal value. These specifications are used with GROUP BY clause. Please refer to the manual for the detailed syntax.

Example 44 ROLLUP (Output the subtotal values for each column c1)

```
postgres=> SELECT c1, SUM(c2) FROM role1 GROUP BY ROLLUP (c1) ;
   c1    |    sum
------+-------
  val1 |    5050
  val3 |    5050
  val4 |    5050
         | 15150
(4 rows)
```

3.4.12 CREATE FOREIGN TABLE (CHECK)

In CREATE FOREIGN TABLE statement, now can be describe the CHECK constraints. However constraints specified in the CREATE FOREIGN TABLE statement is not valid for INSERT or UPDATE statement. CHECK constraint becomes effect only in the SELECT statement of the session that set the parameters constraint_exclusion to on.

Example 45 CHECK constraint in the CREATE FOREIGN TABLE statement

```
postgres=> CREATE FOREIGN TABLE data1 (c1 NUMERIC, c2 VARCHAR(10),
               CHECK(c1 > 0)) SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=> INSERT INTO data1 VALUES (-2, 'add') ;
INSERT 0 1
postgres=> SELECT * FROM data1 WHERE c1 = -2 ;
c1    | c2
----+-----
  -2 | add
(1 row)
postgres=> SET constraint_exclusion = on ;
SET
postgres=> SELECT * FROM data1 WHERE c1 = -20 ;
  c1 | c2
----+----
(0 rows)
```

When you set the parameter constraint_execution to "on", the execution plan will change. Not published a SQL statement to a remote instance in the case of conditions that do not match CHECK constraint in the WHERE clause is written.

Example 46 Execution plan and CHECK constraint

```
postgres=> SHOW constraint_exclusion ;
  constraint_exclusion
----------------------
  partition
(1 row)
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 WHERE c1 < 0 ;
                                        QUERY PLAN
------------------------------------------------------------------------------------------
  Aggregate   (cost=178.27..178.28 rows=1 width=0) (actual time=1.411..1.411 rows=1 loops=1)
     ->   Foreign Scan on data1   (cost=100.00..175.42 rows=1138 width=0) (actual
time=1.400..1.401 rows=6 loops=1)
  Planning time: 0.267 ms
  Execution time: 2.585 ms
(4 rows)
postgres=> SET constraint_exclusion = on ;
SET
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 WHERE c1 < 0 ;
                                         QUERY PLAN
------------------------------------------------------------------------------------------
  Aggregate   (cost=0.01..0.02 rows=1 width=0) (actual time=0.005..0.006 rows=1 loops=1)
     ->   Result   (cost=0.00..0.01 rows=1 width=0) (actual time=0.001..0.001 rows=0 loops=1)
               One-Time Filter: false
  Planning time: 0.351 ms
  Execution time: 0.038 ms
(5 rows)
```

3.4.13 Enhance CREATE EVENT TRIGGER statement

Table_rewrite event can now be specified in the ON clause of the CREATE EVENT TRIGGER statement. Table_rewrite trigger is executed on the ALTER TABLE statement or the ALTER TYPE statement.

Example 47 Enhance CREATE EVENT TRIGGER statement

```
postgres=# CREATE FUNCTION rewrite1() RETURNS event_trigger AS $$
BEGIN
   RAISE NOTICE 'Rewriting Table % for reason %',
                           pg_event_trigger_table_rewrite_oid()::regclass,
                           pg_event_trigger_table_rewrite_reason() ;
END;
$$ LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=# CREATE EVENT TRIGGER rewrite_trg1 ON table_rewrite
              EXECUTE PROCEDURE rewrite1() ;
CREATE EVENT TRIGGER
postgres=> ALTER TABLE data1 ALTER COLUMN c1 TYPE VARCHAR(10) ;
NOTICE:   Rewriting Table data1 for reason 4
ALTER TABLE
```

In the trigger function, it is possible to use the following functions, which get the information of the trigger's associated table.

Table 27 Information obtaining functions

Function Name	Description	Return
pg_event_trigger_table_rewrite_oid	Trigger occurrence table	oid
pg_event_trigger_table_rewrite_reason	Trigger occurrence reason	indicates the reason

Pg_event_trigger_table_rewrite_reason function returns the OR of the following values. These values are listed in the source code (src/include/commands/event_trigger.h).

Table 28 Return value of pg_event_trigger_table_rewrite_reason function

Value	Event	Executed SQL statement
1	Change persistence setting	ALTER TABLE SET UNLOGGED, etc
2	Change DEFAULT	ALTER TABLE ADD COLUMN DEFAULT, etc
4	Change column	ALTER TABLE ADD COLUMN, etc
8	Change OID setting	ALTER TABLE SET WITH OIDS, etc

3.4.14 PL/pgSQL ASSERT statement

ASSERT statement has been added to the PL/pgSQL. ASSERT statement can check the assumed value in the stored procedure. This statement can be ignored using parameters.

Syntax

```
ASSERT condition [, 'message' ]
```

Into the 'condition' write a conditional statement to check the value. If the condition becomes FALSE or NULL, ASSERT_EXCEPTION exception is raised.

The 'message' is optional. You can specify a string which is displayed when an exception occurs. If omitted, "assertion failed" is output.

Example 48 ASSERT statement

```
postgres=> CREATE OR REPLACE FUNCTION assert1(NUMERIC)
           RETURNS NUMERIC
           AS $$
           BEGIN
                ASSERT $1 < 20 ;
                ASSERT $1 > 10, 'Assert Message' ;
                RETURN $1 * 2 ;
           END;
           $$ LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=> SELECT assert1(30) ;
ERROR:     assertion failed
CONTEXT:    PL/pgSQL function assert2(numeric) line 3 at ASSERT
postgres=> SELECT assert1(5) ;
ERROR:     Assert Message
CONTEXT:    PL/pgSQL function assert2(numeric) line 4 at ASSERT
```

□ Parameter

Parameter for controlling the operation of the ASSERT statement is plpgsql.check_asserts. The default value is "on", and ASSERT statement will work. If this parameter is set to "off" (false), ASSERT statement will no longer work.

Example 49 Parameter plpgsql.check_asserts

```
postgres=> SELECT assert1(5) ;
ERROR:    Assert Message
CONTEXT:    PL/pgSQL function assert1(numeric) line 3 at ASSERT
postgres=> SET plpgsql.check_asserts = false ;
SET
postgres=> SELECT assert1(5) ;
  assert1
---------
        10
(1 row)
```

3.4.15 Additional operators and functions for jsonb type

Operators and functions for jsonb type have been added.

□ "||" operator

Using the "||" operator, you can perform addition and update of the json elements. Value will be replaced if you add the same key. This operator with the same behavior can be realized also jsonb_concat function.

Example 50 Add and Replace

```
postgres=> SELECT '{"key":"key1", "val1":"1000"}'::jsonb || '{"val2":"2000"}' ;
                    ?column?
-------------------------------------------------------
  {"key": "key1", "val1": "1000", "val2": "2000"}
(1 row)

postgres=> SELECT '{"key":"key1", "val1":"1000"}'::jsonb || '{"val1":"3000"}' ;
             ?column?
-------------------------------------
  {"key": "key1", "val1": "3000"}
(1 row)
```

□ "-" operator, "#-" operator

Using the "-" operator and "#-" operator, you can delete the json element. If the key does not exist, there is no change in the original data. It is also possible to delete the element by specifying number from the array. Element numbers start from 0. These operator with the same behavior can be realized also jsonb_delete function.

Example 51 Delete element

```
postgres=> SELECT '{"key":"key1", "val1":"1000"}'::jsonb - 'key' ;
         ?column?
-------------------
  {"val1": "1000"}
(1 row)

postgres=> SELECT '["red", "green", "blue"]'::jsonb – 1 ;
       ?column?
-----------------
  ["red", "blue"]
(1 row)
postgres=> SELECT '{"key":"key1", "val1":"1000"}'::jsonb - 'test' ;
                      ?column?
---------------------------------
  {"key": "key1", "val1": "1000"}
(1 row)
```

Example 52 Delete elements of the nested structure

```
postgres=> SELECT
              '{"ename":{"first":"Mike", "last":"Stern"}, "gender":"M"}'::jsonb
              #–
              '{"ename","last"}'::text[] ;
                                 ?column?
---------------------------------------------
  {"ename": {"first": "Mike"}, "gender": "M"}
(1 row)
```

□ jsonb_set Function

Jsonb_set function replace or add of elements. new_value parameter value added if create_missing parameter is true and the item designated by path does not exist.

Syntax

```
jsonb jsonb_set(target jsonb, path text[], new_value jsonb [,
                create_missing boolean DEFAULT true])
```

Example 53 Replace element

```
postgres=> SELECT
  jsonb_set('{"key":"key1", "val1":"1000"}'::jsonb,'{"val1"}','2000') ;
             jsonb_set
-------------------------------
  {"key": "key1", "val1": 2000}
(1 row)
```

Example 54 Add element #1

```
postgres=> SELECT
    jsonb_set('{"key":"key1", "val1":"1000"}'::jsonb,'{"val2"}','2000') ;
                            jsonb_set
-----------------------------------------------
  {"key": "key1", "val1": "1000", "val2": 2000}
(1 row)
```

Example 55 Add element #2

```
postgres=> SELECT
    jsonb_set('{"key":"key1", "val1":"1000"}'::jsonb,'{"val2"}','2000', false) ;
                            jsonb_set
-----------------------------------------------
  {"key": "key1", "val1": "1000", "val1": 2000}
(1 row)
```

□ jsonb_pretty Function

Jsonb_pretty function can perform formatting of jsonb data.

Syntax

```
jsonb jsonb_pretty(from_json jsonb)
```

Example 56 Formatting jsonb

```
postgres=> SELECT jsonb_pretty('{"key":"key1", "val1":"1000",
               "arr1":["itm1","itm2"]}'::jsonb) ;
       jsonb_pretty
--------------------
 {                               +
         "key": "key1",+
         "arr1": [          +
                 "itm1",      +
                 "itm2"        +
         ],                       +
         "val1": "1000"+
 }
(1 row)
```

□ jsonb_strip_nulls Function

Jsonb_strip_nulls function removes the null fields from jsonb data.

Syntax

```
jsonb jsonb_strip_nulls(from_json jsonb)
json json_strip_nulls(from_json json)
```

Example 57 Omit null fields.

```
postgres=> SELECT json_strip_nulls('[{"f1":1,"f2":null},2,null,3]') ;
    json_strip_nulls
---------------------
 [{"f1":1},2,null,3]
```

3.4.16 Additional Functions

Following function have been enhanced in PostgreSQL 9.5.

□ width_bucket Function

To width_bucket function, the version with anyelement type and anyarray type parameter has been added.

□ generate_series Function

Parameters of generate_series function were bigint type, integer type, or timestamp type; now numeric type is also supported. Thus it is now possible to include a decimal point in the increment of value.

Example 58 Enhancement of generate_series Function

```
postgres=> SELECT generate_series (1.2, 2.1, 0.3) ;
 generate_series
-----------------
             1.2
             1.5
             1.8
             2.1
(4 rows)
```

□ Unverified Functions

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

- array_agg (support for anynoarray type)
- array_agg_array*
- array_position
- array_positions
- bernoulli
- binary_upgrade_*
- bound_box
- box (support for polygon type)
- brin*
- bttextsortsupport
- dist_cpoint
- dist_polyp

- dist_ppoly
- gist_bbox_distance
- gist_box_fetch
- gist_point_fetch
- gistcanreturn
- inet_gist_fetch
- inet_merge
- inet_same_family
- int8_avg_accum_inv
- json_strip_nulls
- jsonb_agg
- jsonb_agg_finalfn
- jsonb_agg_transfn
- jsonb_build_*
- jsonb_object*
- max (support timestamp without time zone type)
- min (support timestamp without time zone type)
- mxid_age
- network_larger
- network_smaller
- numeric_poly_*
- numeric_sortsupport
- pg_ddl_command_*
- pg_event_trigger_ddl_commands
- pg_event_trigger_table_rewite_*
- pg_get_object_address
- pg_identify_object_as_address
- pg_last_committed_xact
- pg_ls_dir (add missing_ok, include_dot_dirs parameters)
- pg_read_binary_file (add missing_ok parameter)
- pg_read_file (add missing_ok parameter)
- pg_replication_origin*
- pg_show_all_file_settings
- pg_show_replication_origin_status
- pg_stat_file (add missing_ok parameter)
- pg_stat_get_snapshot_timestamp

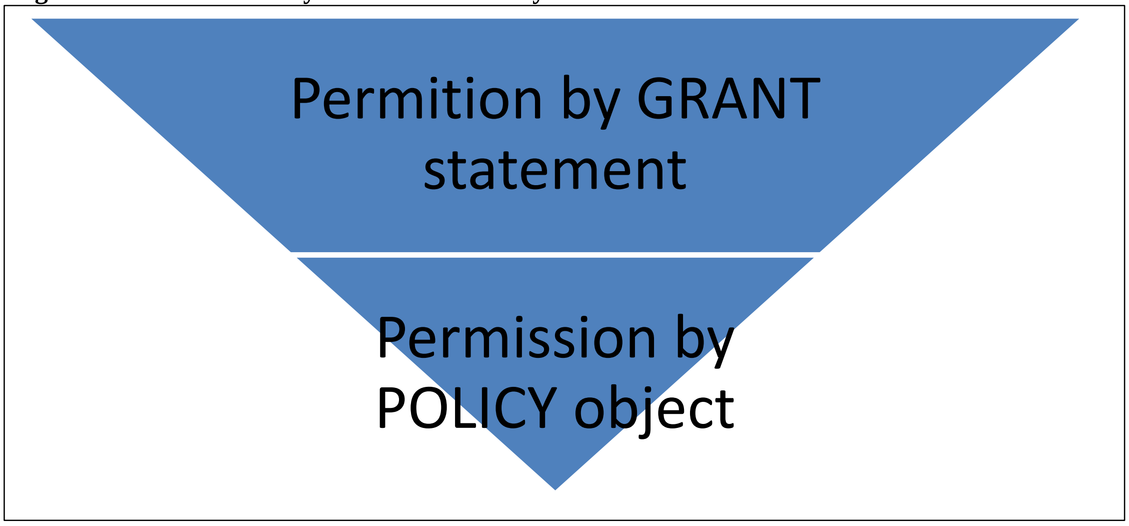
- pg_xact_commit_timestamp
- range_gist_fetch
- range_merge
- regnamespace*
- regrole*
- row_security_active
- system
- timestamp_izone_transform
- timestamp_zone_transform
- to_jsonb
- to_regnamespace
- to_regrole
- tsm_*

3.5 Row Level Security

3.5.1 What is Row Level Security

In previous PostgreSQL, the privilege to access to tables and columns is specified by GRANT statement. This style is available even in PostgreSQL 9.5. Row Level Security is a feature that allows you to limit the tuples (records), which were allowed in the GRANT statement also with tuple level. Row Level Security to by the access restriction you create an object called POLICY.

Figure 2 Access Control by Row Level Security



3.5.2 Preparation

In order to use Row Level Security, execute the ALTER TABLE ENABLE ROW LEVEL SECURITY statement to the table to be restricted by policy. By default Row Level Security setting of the table is disabled. To disable the setting for the table, run the ALTER TABLE DISABLE ROW LEVEL SECURITY statement.

Example 59 Enable Row Level Security to the table

```
postgres=> ALTER TABLE poltbl1 ENABLE ROW LEVEL SECURITY ;
ALTER TABLE
postgres=> \d+ poltbl1
                                   Table "public.poltbl1"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+------------
 c1     | numeric               |           | main     |              |
 c2     | character varying(10) |           | extended |              |
 uname  | character varying(10) |           | extended |              |
Policies (Row Security Enabled): (None)
```

3.5.3 Create POLICY object

To specify access privileges for a table, create the policy. Policy is created with the CREATE POLICY statement. General users can create POLICY.

Syntax

```
CREATE POLICY policy_name ON table_name
[ FOR { ALL | SELECT | INSERT | UPDATE | DELETE } ]
[ TO { roles | PUBLIC, …} ]
USING (condition)
[ WITH CHECK (check_condition) ]
```

Table 29 Syntax for CREATE POLICY statement

Clause	Description
policy_name	Specify the policy name
ON	Specify the table name to which the policy is applied
FOR	Operation name to apply the policy or ALL
TO	Target role name to allow policy or PUBLIC
USING	Specify conditions for permission to access the tuple (Same syntax as WHERE clause). Only the tuples in which the condition specified by USING clause becomes TRUE is returned to the user
WITH CHECK	Specify the conditions for rows that can be updated by the UPDATE statement. You can not specify the CHECK clause in the policy for the SELECT statement

In the example below, the POLICY to the table poltbl1 is created. Since TO clause is omitted, the subject is all users (PUBLIC), the operation is executed to all the SQL (FOR ALL), performed tuple is the one whose "uname" column's value is the same as the current username (current_user function).

Example 60 CREATE POLICY statement

```
postgres=> CREATE POLICY pol1 ON poltbl1 FOR ALL USING (uname = current_user) ;
CREATE POLICY
postgres=> \d+ poltbl1
                          Table "public.poltbl1"
Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+------------
 c1     | numeric               |           | main     |              |
 c2     | character varying(10) |           | extended |              |
 uname  | character varying(10) |           | extended |              |
Policies:
    POLICY "pol1" FOR ALL
      USING (((uname)::name = "current_user"()))
```

Information of the created policy can be checked from pg_policy catalog. The information of the table that has been set a policy can be checked from pg_policies catalog.

In the following example, I verify the effect of the policy as follows:

- ○ User "user1", who is the owner of the table poltbl1, stores 3 records (2-12 lines).
- ○ Although table tblpol1 is searched with the privilege of user "user2", only one record whose "uname" column's value is "user2" is referenced (15-19 lines).
- ○ The user was trying to change the value of "uname" column, however, because of the deviation of the condition specified in USING clause of the CREATE POLICY statement, UPDATE statement failed (20-21 lines).

Example 61 Effect of the policy

```
 1  $ psql -U user1
 2  postgres=> INSERT INTO poltbl1 VALUES (100, 'Val100', 'user1') ;
 3  INSERT 0 1
 4  postgres=> INSERT INTO poltbl1 VALUES (200, 'Val200', 'user2') ;
 5  INSERT 0 1
 6  postgres=> INSERT INTO poltbl1 VALUES (300, 'Val300', 'user3') ;
 7  INSERT 0 1
 8  postgres=> SELECT COUNT(*) FROM poltbl1 ;
 9   count
10  -------
11       3
12  (1 row)
13
14  $ psql -U user2
15  postgres=> SELECT * FROM poltbl1 ;
16   c1  |   c2   | uname
17  -----+--------+-------
18   200 | val200 | user2
19  (1 row)
20  postgres=> UPDATE poltbl1 SET uname='user3' ;
21  ERROR:  new row violates row level security policy for "poltbl1"
```

CREATE POLICY statement, which create a policy, does not result in an error when you run it to the table which is not specified ENABLE ROW LEVEL SECURITY clause. In this case, ROW LEVEL SECURITY feature is not enabled for the corresponding table; only the authorization by the GRANT statement is enabled.

To change or delete policy setting is performed in ALTER POLICY statement or DROP POLICY statement, respectively.

3.5.4 Parameter Settings

Feature of Row Level Security is controlled by the parameter row_security. It can take the following values.

Table 30 Choices for Parameter row_security

Value	Description
on	Enable the feature of Row Level Security. This is the default value.
off	Disable the feature of Row Level Security.
force	It will force the feature of Row Level Security. The policy will be enforced and even the table owner will not be able to access the data which violates the policy. Therefore, even the owner of the table will not be able to access the data with policy violations.

□ User Privileges

User with SUPERUSER privilege will be able to bypass the policy setting. User with BYPASSRLS privilege can bypass the policy by setting the parameter row_security to off (row_security can be set per session). General user who don’t have BYPASSRLS privileged cannot access the table. NOBYPASSRLS is specified as the default of CREATE USER statement.

3.6 BRIN Index

3.6.1 What is BRIN Index

BRIN index (Block Range Index) is the physical format of the newly added index. Traditional BTREE index saves the location of accurate tuples for a column value. While this method is fast for the search by index, there was a disadvantage that the capacity of the index itself increased. BRIN index saves the maximum value and the minimum value of the column values in the block together. This achieved both high query speed and reduction of storage capacity. In particular, it can be expected to be effective as an index to the table that contains a large number of tuples.

Figure 3 Structure of BRIN INDEX

Block Range	Contain NULL?	Minimum Value	Maximum Value
1-128	TRUE	100	1000
129-256	FALSE	200	400
257-384	TRUE	500	2000
385-512	FALSE	100	900

↑ pages_per_range

In the BRIN index, pages_per_range parameter as can be specified the additional option. In this parameter, the number of pages used by the index of the entry is specified. The default value is 128. The minimum value of this option is 1, and the maximum is 131,072.

□ Syntax

To create BRIN index, specify USING BRIN clause in the CREATE INDEX statement.

Syntax

```
CREATE INDEX index_name ON table_name USING BRIN (column_name, …)
[ WITH (pages_per_range = value) ]
```

3.6.2 Examples

The following are examples of creating a BRIN index. First create the table and store the data.

Example 62 Create the table and store the data

```
postgres=> CREATE TABLE brin1 (c1 NUMERIC, c2 NUMERIC, c3 NUMERIC, c4 NUMERIC) ;
CREATE TABLE
postgres=> INSERT INTO brin1 VALUES (
postgres(>      generate_series(1, 10000000),
postgres(>      generate_series(1, 10000000),
postgres(>      generate_series(1, 10000000),
postgres(>      generate_series(1, 10000000)
postgres(>    ) ;
INSERT 0 10000000
```

Next, create an index for each column. BTREE index is created for the c1 column as a comparison. From c2 column to c4 column, three BRIN index with different parameter pages_per_range value are created.

Example 63 Create index and check

```
postgres=> CREATE INDEX btree1 ON brin1 (c1) ;
CREATE INDEX
postgres=> CREATE INDEX brin1_def ON brin1 USING BRIN (c2) ;
CREATE INDEX
postgres=> CREATE INDEX brin1_64 ON brin1 USING BRIN (c3)
          WITH (pages_per_range = 64) ;
CREATE INDEX
postgres=> CREATE INDEX brin1_512 ON brin1 USING BRIN (c4)
          WITH (pages_per_range = 512) ;
CREATE INDEX
postgres=> ANALYZE VERBOSE brin1 ;
INFO:  analyzing "public.brin1"
INFO:  "brin1": scanned 30000 of 73520 pages, containing 4080483 live rows and 0 dead rows;
30000 rows in sample, 9999961 estimated total rows
ANALYZE
postgres=> \d+ brin1
                      Table "public.brin1"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 c1     | numeric |           | main    |              |
 c2     | numeric |           | main    |              |
 c3     | numeric |           | main    |              |
 c4     | numeric |           | main    |              |
Indexes:
    "brin1_512" brin (c4) WITH (pages_per_range=512)
    "brin1_64" brin (c3) WITH (pages_per_range=64)
    "brin1_def" brin (c2)
    "btree1" btree (c1)
```

Finally, checking the usage of storage area. In the case of the example below, BRIN index is found to be constructed with a very small area.

Example 64 Check the size of files (storage)

```
postgres=> SELECT relname, pg_size_pretty(pg_relation_size(oid)) FROM
               pg_class WHERE relname LIKE 'brin1%' OR relname = 'btree1' ;
   relname    | pg_size_pretty
-----------+----------------
  brin1         | 574 MB
  btree1        | 214 MB
  brin1_def     | 32 kB
  brin1_64      | 48 kB
  brin1_512 | 24 kB
(5 rows)
```

Now I validate the performance. Run the same SQL statements, which have the difference in their column name, multiple times to verify the results of using the index.

Example 65 Execution plan for BTREE index

```
postgres=> EXPLAIN ANALYZE SELECT * FROM brin1 WHERE c1 = 5000000 ;
                                           QUERY PLAN
--------------------------------------------------------------------------------
  Index Scan using btree1 on brin1    (cost=0.43..8.45 rows=1 width=24)
               (actual time=0.085..0.086 rows=1 loops=1)
      Index Cond: (c1 = '5000000'::numeric)
  Planning time: 0.389 ms
  Execution time: 0.240 ms
(4 rows)
```

The following are examples of using the BRIN index.

Example 66 Execution plan for BRIN index (default parameter value)

```
postgres=> EXPLAIN ANALYZE SELECT * FROM brin1 WHERE c2 = 5000000 ;
                                QUERY PLAN
------------------------------------------------------------------------------
 Bitmap Heap Scan on brin1   (cost=16.01..20.02 rows=1 width=24)
           (actual time=1.955..9.026 rows=1 loops=1)
   Recheck Cond: (c2 = '5000000'::numeric)
   Rows Removed by Index Recheck: 17407
   Heap Blocks: lossy=128
   ->   Bitmap Index Scan on brin1_def   (cost=0.00..16.01 rows=1 width=0)
           (actual time=0.747..0.747 rows=1280 loops=1)
             Index Cond: (c2 = '5000000'::numeric)
 Planning time: 0.144 ms
 Execution time: 9.222 ms
(8 rows)
```

Example 67 Execution plan for BRIN index (pages_per_range=64)

```
postgres=> EXPLAIN ANALYZE SELECT * FROM brin1 WHERE c3 = 5000000 ;
                              QUERY PLAN
------------------------------------------------------------------------------
Bitmap Heap Scan on brin1   (cost=24.01..28.02 rows=1 width=24)
           (actual time=2.542..5.310 rows=1 loops=1)
   Recheck Cond: (c3 = '5000000'::numeric)
   Rows Removed by Index Recheck: 8703
   Heap Blocks: lossy=64
   ->   Bitmap Index Scan on brin1_64   (cost=0.00..24.01 rows=1 width=0)
           (actual time=1.420..1.420 rows=640 loops=1)
             Index Cond: (c3 = '5000000'::numeric)
 Planning time: 0.140 ms
 Execution time: 5.360 ms
(8 rows)
```

Example 68 Execution plan for BRIN index (pages_per_range=512)

```
postgres=> EXPLAIN ANALYZE SELECT * FROM brin1 WHERE c4 = 5000000 ;
                                          QUERY PLAN
---------------------------------------------------------------------------
  Bitmap Heap Scan on brin1    (cost=12.01..16.02 rows=1 width=24)
              (actual time=29.661..36.501 rows=1 loops=1)
      Recheck Cond: (c4 = '5000000'::numeric)
      Rows Removed by Index Recheck: 69631
      Heap Blocks: lossy=512
      ->   Bitmap Index Scan on brin1_512    (cost=0.00..12.01 rows=1 width=0)
              (actual time=0.258..0.258 rows=5120 loops=1)
                Index Cond: (c4 = '5000000'::numeric)
  Planning time: 0.146 ms
  Execution time: 36.561 ms
(8 rows)
```

3.6.3 Information confirming

To the Contrib module pageinspect, following functions have been added to retrieve information of BRIN index.

Table 31 Functions that have been added to the pageinspect module

Function name	Description
brin_page_type	Get the page type of index page
brin_metapage_info	Get a variety of information in the index metadata
brin_revmap_data	Get a list of tuples from the map page
brin_page_items	Get the stored data from the data page

Example 69 Example of brin_page_items function

```
postgres=# CREATE EXTENSION pageinspect ;
CREATE EXTENSION
postgres=# SELECT itemoffset, blknum, value
    FROM brin_page_items(get_raw_page('brin1_def', 2), 'brin1_def') LIMIT 10 ;
 itemoffset  | blknum   |            value
------------+--------+--------------------
           1 |          0 | {1 .. 18745}
           2 |      128 | {18746 .. 36153}
           3 |      256 | {36154 .. 53561}
           4 |      384 | {53562 .. 70969}
           5 |      512 | {70970 .. 88377}
           6 |      640 | {88378 .. 105785}
           7 |      768 | {105786 .. 123193}
           8 |      896 | {123194 .. 140601}
           9 |     1024 | {140602 .. 158009}
          10 |     1152 | {158010 .. 175417}
(10 rows)
```

Option information in the BRIN index can also be found from reloptions column of pg_class catalog.

Example 70 Obtain of options information

```
postgres=> SELECT relname,reloptions FROM pg_class where relname like 'brin1%' ;
   relname    |          reloptions
-----------+-----------------------
 brin1         |
 brin1_512 | {pages_per_range=512}
 brin1_64    | {pages_per_range=64}
 brin1_def    |
(4 rows)
```

3.7 Other New Features

3.7.1 Process Name

If a character string is specified to the new parameter cluster_name, specified string is given to the process name. Characters that can be specified in the parameter cluster_name is an ASCII characters (0x20 - 0x7E). The other characters are output after being converted to a question mark (?). The following example shown the process name that was specified as "cluster1" to cluster_name parameter. Postmaster process is not affected by this parameter.

Example 71 Parameter cluster_name

```
$ ps –ef | grep postgres
postgres 12364     1  0 06:14 pts/0 00:00:00 /usr/local/pgsql/bin/postgres -D data
postgres 12365 12364  0 06:14 ?     00:00:00 postgres: cluster1: logger process
postgres 12367 12364  0 06:14 ?     00:00:00 postgres: cluster1: checkpointer process
postgres 12368 12364  0 06:14 ?     00:00:00 postgres: cluster1: writer process
postgres 12369 12364  0 06:14 ?     00:00:00 postgres: cluster1: wal writer process
postgres 12370 12364  0 06:14 ?     00:00:00 postgres: cluster1: autovacuum launcher process
postgres 12371 12364  0 06:14 ?     00:00:00 postgres: cluster1: stats collector process
```

It does not mean that cluster_name parameter is checked between database clusters. Even if multiple instance, specified the same value to the cluster_name parameter, are executed in one host, it does not result in an error.

3.7.2 Output of EXPLAIN statement

In the output of EXPLAIN VERBOSE statement, additional information about the sort is now added. Underlined parts of the following example are the output added in PostgreSQL 9.5.

Example 72 Additional sort information

```
postgres=> EXPLAIN VERBOSE SELECT * FROM sort1
                    ORDER BY c1 DESC, c2 COLLATE "C" ;
                              QUERY PLAN
-----------------------------------------------------------------------
 Sort   (cost=65.83..68.33 rows=1000 width=12)
   Output: c1, c2, ((c2)::character varying(10))
   Sort Key: sort1.c1 DESC, sort1.c2 COLLATE "C"
   ->  Seq Scan on public.sort1   (cost=0.00..16.00 rows=1000 width=12)
         Output: c1, c2, c2
```

3.7.3 Log for Replication Command

When you set the parameters log_replication_commands to "on" the master instance of the replication environment, the log of the replication operations which wal sender process executed is output. If this parameter is set to be "on", the message of which the output LOG level log is output. In case of default value ("off"), the log is output at DEBUG1 level. The beginning of the log is "received replication command:", followed by a replication-related command. For example, the following log is output when the slave instance connects.

Example 73 Log of slave instance connection

```
LOG:  received replication command: IDENTIFY_SYSTEM
LOG:  received replication command: START_REPLICATION SLOT "slot_1" 0/7000000 TIMELINE 1
```

Log is not output at the time of the slave instance stop. Since pg_basebackup command also uses the replication feature, the following log is output.

Example 74 Log of pg_basebackup command

```
LOG:    received replication command: IDENTIFY_SYSTEM
LOG:    received replication command: BASE_BACKUP LABEL 'pg_basebackup base backup'    WAL
NOWAIT
```

3.7.4 Cast Type

Newly added Cast type is utilized to convert oid type to object name. The following data types can be used.

Table 32 Data Type

Data Type	Description
regnamespace	Get schema name
regrole	Get role name

In the following example, I have searched the schema name, table name, the owner name from pg_class catalog. Since "relowner" column, "relnamespace" column is type oid, type conversion is executed.

Example 75 search pg_class catalog

```
postgres=> SELECT relnamespace::regnamespace, relname, relowner::regrole
               FROM pg_class WHERE relnamespace = 'public'::regnamespace ;
 relnamespace | relname | relowner
--------------+---------+----------
 public       | data1   | user1
 public       | data2   | user1
(2 rows)
```

Bibliography

I have referred to the following URL.

- Release Notes
  http://www.postgresql.org/docs/9.5/static/release.html
- What's new in PostgreSQL 9.5
  https://wiki.postgresql.org/wiki/What%27s_new_in_PostgreSQL_9.5
- Commitfests
  https://commitfest.postgresql.org/
- PostgreSQL 9.5 Alpha Manual
  http://www.postgresql.org/docs/9.5/static/index.html
- GitHub
  https://github.com/postgres/postgres
- Announcement for PostgreSQL 9.5 Alpha 2
  http://www.postgresql.org/about/news/1604/
- PostgreSQL 9.5 WAL format
  https://wiki.postgresql.org/images/a/af/FOSDEM-2015-New-WAL-format.pdf
- Open source developer based in Japan (Michael Paquier)
  http://michael.otacoo.com/
- Hibi-no kiroku bekkan (Nuko@Yokohama)
  http://d.hatena.ne.jp/nuko_yokohama/
- v9.5 New Features - Custom Scan/Join Interface
  http://www.slideshare.net/kaigai/postgresql-unconference-30may-tokyo
- Tablesample In PostgreSQL 9.5
  http://blog.2ndquadrant.com/tablesample-in-postgresql-9-5/

Modification History

History

Ver#	Date	Author	Description
0.1	July 6, 2015	Noriyoshi Shinoda	Created for HP internal review. (Alpha 1) Reviewers are: Tomoo Takahashi Akiko Takeshima Takahiro Kitayama Yoshiko Shinoda
1.0	August 7, 2015	Noriyoshi Shinoda	Created to be published to the Internet. (Alpha 2)