Hewlett Packard Enterprise

May 24, 2019

PostgreSQL 12 New Features

With Examples (Beta 1)

Hewlett Packard Enterprise Japan Co, Ltd.

Noriyoshi Shinoda

Hewlett Packard
Enterprise

Index

1. About This Document

1.1. Purpose

The purpose of this document is to provide information about the major new features of PostgreSQL 12 Beta 1.

1.2. Audience

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

1.3. Scope

This document describes the major difference between PostgreSQL 11 (11.3) and PostgreSQL 12 Beta 1 (12.0). As a general rule, this document examines the features of behavior change. This document does not describe and verify all new features. In particular, the following new features are not included.

- Bug fix
- Performance improvement by changing internal behavior
- Improvement of regression test
- Operability improvement by psql command tab input
- Improvement of pgbench command
- Improvement of documentation, modify typo in the source code
- Refactoring without a change in behavior

1.4. Software Version

The contents of this document have been verified for the following versions and platforms.

Table 1 Version

Software	Version
PostgreSQL	PostgreSQL 11.3 (for comparison) PostgreSQL 12 (12.0) Beta 1 (May 20, 2019 20:41:05)
Operating System	Red Hat Enterprise Linux 7 Update 5 (x86-64)
Configure option	--with-llvm --with-openssl --with-perl

1.5. Question, comment and Responsibility

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

Notation	Description
Italic	Replaced by the name of the object which users use, or the other syntax.
[ ]	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 Features Summary

More than 150 new features have been added to PostgreSQL 12. Here are some typical new features and benefits. This version focuses to enhance various new features added in PostgreSQL 11.

2.1. Improve analytic query performance

The following features have been added that can be applied to large scale environments:

□ Enhancement of Parallel Query

The scope to which parallel queries are applied has been expanded. Even when the transaction isolation level is SERIALIZABLE, parallel queries may be performed.

□ Enhancement of Partition Table

It is now possible to specify computed values instead of fixed values as partition key values. Partition tables can now be used as foreign key references.

2.2. Improve reliability

In PostgreSQL 12 the checking tool of integrity has been expanded.

□ pg_checksums command

The pg_verify_checksums command added in PostgreSQL 11 has been changed to the pg_checksums command. In addition to checking the integrity, it is now possible to change the enable/disable of the checksum function with the command.

2.3. Improve maintenancibility

The following features that can improve the operability has been added.

□ Abolition of recovery.conf file

The recovery.conf file, used for streaming replication standby instances and recovery, have been consolidated into the postgresql.conf file.

□ Enhancement of REINDEX statement

The CONCURRENTLY clause was added to the REINDEX statement, and the lock range when rebuilding the index became very small.

□ Enhancement of monitoring features

Catalogs have been added that allow you to check the execution status of CLUSTER statement, VACUUM FULL statement and CREATE INDEX statement in real time.

□ Increase in waiting for event type

Several wait events have been added. It can be checked in the pg_stat_activity catalog.

□ pg_promote function

A function pg_promote has been provided to perform promotion from the standby instance to the primary instance.

2.4. Preparing for future new features

PostgreSQL 12 is preparing for the features to be provided in future versions.

□ Multiple storage engine

The basic specification of PLUGGABLE STORAGE ENGINE has been decided that can use multiple storage engines simultaneously. It is expected that the corresponding such as zheap is carried out in the future.

□ 64-bit transaction ID

APIs that obtain 64-bit transaction IDs can now be used.

2.5. Incompatibility

PostgreSQL 12 has the following specifications changed from PostgreSQL 11.

2.5.1. recovery.conf file

Recovery.conf files created in database recovery and standby instances in a streaming replication environment have been discontinued. Various parameters in the recovery.conf file have been integrated into the postgresql.conf file. Details are explained in "3.1.2. Deprecation of recovery.conf file".

2.5.2. Pg_checksums command

The pg_verify_checksums command added in PostgreSQL 11 has been renamed to pg_checksums and the new feature has been added.

2.5.3. To_timestamp / to_date function

Unnecessary space in the template is now ignored.

Example 1 to_date function in PostgreSQL 11

```
postgres=> SELECT TO_DATE('2019/03/23', ' YYYY/MM/DD') ;
  to_date
------------
 0019-03-23
(1 row)
```

Example 2 to_date function in PostgreSQL 12

```
postgres=> SELECT TO_DATE('2019/03/23', ' YYYY/MM/DD') ;
  to_date
------------
 2019-10-20
(1 row)
```

2.5.4. WITH OIDS clause deleted

The WITH OIDS clause of the CREATE TABLE statement is prohibited, and a table with OID cannot be created. Along with this change, the oid column in the system catalog is no longer a hidden column.

The WITHOUT OIDS clause can continue to be used. Along with this change, the default_with_oids parameter cannot be changed to 'on'. With this specification change, the --oids option (-o option) has been removed from the pg_dump command.

Example 3 CREATE TABLE statement in PostgreSQL 12

```
postgres=> CREATE TABLE oid1(c1 INT) WITH OIDS ;
psql: ERROR:  syntax error at or near "OIDS"
LINE 1: CREATE TABLE oid1(c1 INT) WITH OIDS ;
```

Example 4 pg_tablespace catalog in PostgreSQL 12

```
postgres=> \d pg_tablespace
           Table "pg_catalog.pg_tablespace"
   Column   |   Type    | Collation | Nullable | Default
------------+-----------+-----------+----------+---------
 oid        | oid       |           | not null |
 spcname    | name      |           | not null |
 spcowner   | oid       |           | not null |
 spcacl     | aclitem[] |           |          |
 spcoptions | text[]    |           |          |
Indexes:
    "pg_tablespace_oid_index" UNIQUE, btree (oid), tablespace "pg_global"
    "pg_tablespace_spcname_index" UNIQUE, btree (spcname), tablespace
"pg_global"
Tablespace: "pg_global"
```

2.5.5. Timetravel Contrib module

The Contrib module timetravel has been removed.

2.5.6. Removed data type

The datatypes abstime, reltime, and tinterval have been removed. As a result, the data type of column valutil in the system catalog pg_shadow has been changed to "timestamp with time zone".

2.5.7. COPY FREEZE for the partitioned table

The COPY FREEZE statement for partitioned tables has been prohibited. This specification has also been backported to PostgreSQL 11.2 and later.

2.5.8. Change foreign key constraint name

Auto-generated foreign key names are now generated including all the columns contained in the foreign key.

Example 5 FOREIGN KEY constraint name in PostgreSQL 12

```
postgres=> CREATE TABLE ftable2(c1 INT, c2 INT, c3 VARCHAR(10),
       FOREIGN KEY (c1, c2) REFERENCES ftable1(c1, c2)) ;
CREATE TABLE
postgres=> \d ftable2
                  Table "public.ftable2"
 Column |        Type         | Collation | Nullable | Default
--------+---------------------+-----------+----------+---------
 c1     | integer             |           |          |
 c2     | integer             |           |          |
 c3     | character varying(10) |           |          |
Foreign-key constraints:
    "ftable2_c1_c2_fkey" FOREIGN KEY (c1, c2) REFERENCES ftable1(c1, c2)
```

2.5.9. Functions

Current_schema and current_schemas functions were set to be unsafe for the parallel query (Parallel Unsafe).

3. New Features Detail

3.1. Architecture

3.1.1. Modified catalogs

The following catalogs have been changed.

Table 4 Added system catalogs

Catalog name	Description
pg_stat_progress_cluster	Traces the execution status of CLUSTER.
pg_stat_progress_create_index	Traces the execution status of CREATE INEDX.

Table 5 Added information_schema catalog

Catalog name	Description
column_column_usage	Column information that depends on a specific column (stores information on a table with generated columns).

Table 6 System catalogs with columns added

Catalog name	Added column	Data type	Description
*1	oid	oid	Change attribute to regular column.
pg_attribute	attgenerated	char	's' for generated column.
pg_collation	collisdeterministic	boolean	Is the collation deterministic?
pg_statistic	stacol[1-5]	oid	Collation information.
pg_statistic_ext	stxmcv	pg_mcv_list	MCV statistics info.
pg_stat_database	checksum_failures	bigint	Number of blocks for which a checksum error was detected.
	checksum_last_failure	timestamp with time zone	Last timestamp for which a checksum error was detected.
pg_stat_replication	reply_time	timestamp with time zone	Message time from standby instance.
pg_stat_ssl	client_serial	numeric	Client certificate serial number.
	issuer_dn	text	Client certificate issuer DN.

*1 There are many corresponding catalogs.

Table 7 System catalogs with columns deleted

Catalog name	Deleted column	Description
pg_attrdef	adsrc	A human-readable representation of the default value.
pg_class	relhasoids	Whether the table is WITH OIDS specified.
pg_constraint	consrc	A human-readable representation of the expression.

Table 8 System catalogs with column modified

Catalog name	Modified Column	Description
pg_shadow	valuntil	Data type changed to timestamp with time zone.
pg_stat_ssl	client_dn	Column name changed from 'lientdn'.

Among the modified system catalogs, the details of the major catalogs are described below.

□ Pg_stat_progress_cluster catalog

The pg_stat_progress_cluster catalog can trace the execution status of CLUSTER statements or VACUUM FULL statements.

Table 9 pg_stat_progress_cluster catalog

Column name	Data type	Description
pid	integer	Process ID of backend.
datid	oid	OID of the database to which this backend is connected.
datname	name	Name of the database to which this backend is connected.
relid	oid	OID of the table being clustered.
command	text	The command that is running.
phase	text	Current processing phase.
cluster_index_relid	oid	OID of the index being used.
heap_tuples_scanned	bigint	Number of heap tuples scanned.
heap_tuples_written	bigint	Number of heap tuples written.
heap_blks_total	bigint	Total number of heap blocks in the table.
heap_blks_scanned	bigint	Number of heap blocks scanned.
index_rebuild_count	bigint	Number of indexes rebuilt.

□ Pg_stat_progress_create_index catalog

The pg_stat_progress_create_index catalog can trace the execution status of CREATE INDEX statement. The status is also reflected when the REINDEX statement is executed.

Table 10 pg_stat_progress_create_index catalog

Column name	Data type	Description
pid	integer	Process ID of backend.
datid	oid	OID of the database to which this backend is connected.
datname	name	Name of the database to which this backend is connected.
relid	oid	OID of the table on which the index is being created.
index_relid	oid	OID of the index
phase	text	Current processing phase of index creation.
lockers_total	bigint	Total number of lockers to wait for.
lockers_done	bigint	Total number of lockers already waited for.
current_locker_pid	bigint	Process ID of the locker currently being waited for.
blocks_total	bigint	Total number of the blocks to be processed.
blocks_done	bigint	Total number of the blocks already processed.
tuples_total	bigint	Total number of tuples to be processed.
tuples_done	bigint	Number of tuples already processed.
partitions_total	bigint	Total number of partitions.
partitions_done	bigint	Number of partitions on which the index has been completed.

Example 6 Refer pg_stat_progress_create_index catalog

```
postgres=> SELECT * FROM pg_stat_progress_create_index ;
-[ RECORD 1 ]------+-------------------------------------
pid               | 13233
datid             | 16385
datname           | postgres
relid             | 16386
index_relid       | 0
phase             | building index: loading tuples in tree
lockers_total     | 0
lockers_done      | 0
current_locker_pid | 0
blocks_total      | 0
blocks_done       | 0
tuples_total      | 10000000
tuples_done       | 6207353
partitions_total  | 0
partitions_done   | 0
```

□ Pg_stat_replication catalog

The reply_time column has been added to the pg_stat_replication catalog. This column stores the sending time of the last reply message received from the standby instance. This column can only be displayed by the SUPERUSER attribute or the holder of the pg_monitor role.

Example 7 Refer pg_stat_replication catalog

```
postgres=# SELECT * FROM pg_stat_replication ;
-[ RECORD 1 ]----+------------------------------
pid              | 12497
usesysid         | 10
usename          | postgres
application_name | walreceiver
client_addr      |
client_hostname  |
client_port      | -1
backend_start    | 2019-05-24 20:13:15.551032+09
backend_xmin     |
state            | streaming
sent_lsn         | 0/3000060
write_lsn        | 0/3000060
flush_lsn        | 0/3000060
replay_lsn       | 0/3000060
write_lag        |
flush_lag        |
replay_lag       |
sync_priority    | 0
sync_state       | async
reply_time       | 2019-05-24 20:15:45.68363+09
```

□ Pg_indexes catalog

The pg_indexes catalog now includes partitioned indexes. Specifically, the relkind column in the pg_class catalog now includes an index of 'I'.

□ Pg_stat_database catalog

The checksum_failures column and checksum_last_failure column in the pg_stat_database catalog are now updated when a block checksum error is detected. The value of checksum_failures column is updated each time when a checksum error is detected, so it does not indicate the actual number of blocks corrupted.

Example 8 Refer pg_stat_database catalog

```
\postgres=# SELECT * FROM data1 ;
psql: WARNING:  page verification failed, calculated checksum 27311 but expected
12320
ERROR:  invalid page in block 0 of relation base/13567/16384
postgres=#
postgres=# SELECT datname, checksum_failures , checksum_last_failure
        FROM pg_stat_database WHERE datname='postgres' ;
-[ RECORD 1 ]-----+------------------------------
datname          | postgres
checksum_failures | 1
checksum_last_failure | 2019-05-24 20:06:23.747741+09
```

3.1.2. Abolition of recovery.conf file

The recovery.conf file used to build standby instances in a streaming replication environment, and for recovery from backup has been obsolete. Various settings in the recovery.conf file have been integrated into the postgresql.conf file. Some parameters have been changed.

Table 11 Modified parameters

recovery.conf	postgresql.conf	Note
standby_mode	-	Discontinued.
trigger_file	promote_trigger_file	

In case of recovery, create a recovery.signal file in the database cluster and start the instance. This file is deleted when the recovery process is complete. In a streaming replication environment, create a standby.signal file in the database cluster on slave instances. This file is deleted when promoted to primary.

If the recovery.conf file exists in the database cluster, instance startup fails.

Example 9 Deploy recovery.conf file and start instance

```
$ ls data/recovery.conf
data/recovery.conf
$ pg_ctl -D data start
waiting for server to start....2019-05-24 18:59:43.296 JST [41021] LOG:
starting PostgreSQL 12beta1 on x86_64-pc-linux-gnu, compiled by gcc (GCC)
4.8.5 20150623 (Red Hat 4.8.5-28), 64-bit
2019-05-24 18:59:43.297 JST [41021] LOG:  listening on IPv6 address
"::1", port 5432
2019-05-24 18:59:43.297 JST [41021] LOG:  listening on IPv4 address
"127.0.0.1", port 5433
2019-05-24 18:59:43.468 JST [41021] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.5432"
2019-05-24 18:59:43.709 JST [41022] LOG:  database system was
interrupted; last known up at 2019-05-23 18:55:36 JST
2019-05-24 18:59:43.920 JST [41022] FATAL:  using recovery command file
"recovery.conf" is not supported
2019-05-24 18:59:43.921 JST [41021] LOG:  startup process (PID 41022)
exited with exit code 1
2019-05-24 18:59:43.921 JST [41021] LOG:  aborting startup due to startup
process failure
2019-05-24 18:59:43.921 JST [41021] LOG:  database system is shut down
 stopped waiting
pg_ctl: could not start server
Examine the log output.
$
```

3.1.3. Instance startup log

The version number is now output to the log at instance startup. If the parameter logging_collector is specified as 'on', it is output to the log file.

Example 10 Instance startup log

```
$ pg_ctl -D data start
waiting for server to start....2019-05-24 18:09:33.291 JST [89769] LOG:
starting PostgreSQL 12beta1 on x86_64-pc-linux-gnu, compiled by gcc (GCC)
4.8.5 20150623 (Red Hat 4.8.5-28), 64-bit
2019-05-24 18:09:33.291 JST [89769] LOG:  listening on IPv4 address
"0.0.0.0", port 5432
2019-05-24 18:09:33.291 JST [89769] LOG:  listening on IPv6 address "::",
port 5432
…
```

3.1.4. Maximum number of connections

The number of connections for replication is now obtained using the max_wal_senders parameter without depending on the max_connections parameter. As a result of this modification, information of max_wal_senders has been added to the output of the pg_controldata command.

Example 11 Output of pg_controldata command

```
$ pg_controldata -D data | grep max_wal_senders
max_wal_senders setting:            10
```

3.1.5. Parallel query enhancement

Parallel queries now work even if the transaction isolation level of the session is SERIALIZABLE. The following example uses the auto_explain module to output the execution plan of the SQL statement executed in the transaction of the SERIALIZABLE isolation level.

Example 12 Execute query

```
postgres=# LOAD 'auto_explain' ;
LOAD
postgres=# SET auto_explain.log_min_duration = 0 ;
SET
postgres=# BEGIN ISOLATION LEVEL SERIALIZABLE ;
BEGIN
postgres=# SELECT COUNT(*) FROM data1 ;
  count
---------
 1000000
(1 row)
postgres=# COMMIT ;
COMMIT
```

Example 13 Execution plan on PostgreSQL 11

```
2019-05-24 21:52:48.785 JST [1789] LOG:  duration: 45.472 ms  plan:
        Query Text: SELECT COUNT(*) FROM data1 ;
        Aggregate  (cost=17906.00..17906.01 rows=1 width=8)
          ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=0)
```

Example 14 Execution plan of PostgreSQL 12

```
2019-05-24 21:53:22.509 JST [79911] LOG:  duration: 58.141 ms  plan:
        Query Text: SELECT COUNT(*) FROM data1 ;
        Finalize Aggregate  (cost=11614.55..11614.56 rows=1 width=8)
          ->  Gather  (cost=11614.33..11614.54 rows=2 width=8)
                Workers Planned: 2
                ->  Partial Aggregate  (cost=10614.33..10614.34 rows=1
width=8)
                      ->  Parallel Seq Scan on data1  (cost=0.00..9572.67
rows=4
```

3.1.6. Jsonb type and GIN index

The "jsonb @@ jsonpath" and "jsonb @? jsonpath" operators have been added to the GIN index created for the jsonb type. The operator class of GIN index can be used either jsonb_ops or json_path_ops. The following is the execution plan of the SELECT statement described in the manual.

Example 15 jsonb type and GIN index

```
postgres=> CREATE INDEX idxgin ON api USING GIN (jdoc) ;
CREATE INDEX
postgres=> EXPLAIN (COSTS OFF) SELECT jdoc->'guid', jdoc->'name' FROM
api WHERE jdoc @@ '$.tags[*] == "qui"' ;
                          QUERY PLAN
---------------------------------------------------------------
 Bitmap Heap Scan on api
   Recheck Cond: (jdoc @@ '($."tags"[*] == "qui")'::jsonpath)
   ->  Bitmap Index Scan on idxgin
         Index Cond: (jdoc @@ '($."tags"[*] == "qui")'::jsonpath)
(4 rows)

postgres=> EXPLAIN (COSTS OFF) SELECT jdoc->'guid', jdoc->'name' FROM
api WHERE jdoc @@ '$.tags[*] ? (@ == "qui")' ;
                          QUERY PLAN
---------------------------------------------------------------------
 Bitmap Heap Scan on api
   Recheck Cond: (jdoc @@ '$."tags"[*]?(@ == "qui")'::jsonpath)
   ->  Bitmap Index Scan on idxgin
         Index Cond: (jdoc @@ '$."tags"[*]?(@ == "qui")'::jsonpath)
(4 rows)
```

3.1.7. Wait Event

The following changes were made to the wait event output to the wait_event column of the pg_stat_activity catalog.

Table 12 Modified wait events

Wait event name	Description	Modify
BackendRandomLock	Waiting to generate a random number.	Deleted
GSSOpenServer	Waiting for connect to GSSAPI server.	Added
CheckpointDone	Waiting for a checkpoint to complete.	Added
CheckpointStart	Waiting for a checkpoint to start.	Added
Promote	Waiting for standby promotion.	Added
WALSync	Waiting for a WAL file to reach stable storage.	Added

3.1.8. ECPG

The following features have been added to ECPG.

□ DECLARE STATEMENT

SQL statements can now be declared as variables using the DECLARE statement.

Example 16 DECLARE STATEMENT

```
{
    EXEC SQL BEGIN DECLARE SECTION ;
        char *selectString = "SELECT to_char(current_date, 'YYYY/MM/DD') cd" ;
        char today[20] ;
        short today_ind = 0 ;
    EXEC SQL END DECLARE SECTION ;

    memset(today, 0, sizeof(today)) ;

    EXEC SQL DECLARE stmt_1 STATEMENT ;
    EXEC SQL PREPARE stmt_1 FROM :selectString ;
    EXEC SQL DECLARE cur_1 CURSOR FOR stmt_1 ;
    EXEC SQL OPEN cur_1 ;
    EXEC SQL FETCH cur_1 INTO :today :today_ind ;
    EXEC SQL CLOSE cur_1 ;
}
```

□ Bytea data type

The variable of type bytea is defined in the DECLARE SECTION and can be used for data input/output.

Example 17 DECLARE bytea

```
{
    EXEC SQL BEGIN DECLARE SECTION ;
        bytea data[1024] ;
        short data_ind = 0 ;
    EXEC SQL END DECLARE SECTION ;

    memset(data.arr, 0, sizeof(data.arr)) ;
    data.len = 0 ;

    EXEC SQL SELECT col1 INTO :data FROM data1 ;

    for (i = 0; i < data.len; i++)
    {
        printf("data[%d] = %c\n", i, data.arr[i]) ;
    }
}
```

3.1.9. Pluggable Storage Engine

The basic specifications for using multiple storage engines have been decided. The access method for the table specifies the TYPE TABLE clause in the CREATE ACCESS METHOD statement.

Syntax

```
CREATE ACCESS METHOD am_name TYPE TABLE HANDLER handler_name
```

The default storage engine is specified by the default_table_access_method parameter. The default value for this parameter is 'heap'. As a result of this modification, it has been added tuple to the pg_am catalog.

Example 18 Refer pg_am catalog

```
postgres=# SELECT amname, amhandler, amtype FROM pg_am ;
 amname |      amhandler       | amtype
--------+----------------------+--------
 heap   | heap_tableam_handler | t
 btree  | bthandler            | i
 hash   | hashhandler          | i
 gist   | gisthandler          | i
 gin    | ginhandler           | i
 spgist | spghandler           | i
 brin   | brinhandler          | i
(7 rows)
```

□ Specify engine when creating table

To specify a storage engine for a table, specify the USING clause in the CREATE TABLE statement. It can also be specified with the CREATE TABLE AS SELECT statement or the CREATE MATERIALIZED VIEW statement.

Example 19 CREATE TABLE statement

```
postgres=> CREATE TABLE data1(c1 NUMERIC, c2 VARCHAR(10)) USING heap ;
CREATE TABLE
postgres=> CREATE TABLE data2 USING heap AS SELECT * FROM data1 ;
SELECT 100000
postgres=> CREATE MATERIALIZED VIEW mview1 USING heap AS SELECT COUNT(*)
       cnt FROM data1 ;
SELECT 1
```

□ Display table definition by psql command

When the "\d+ table_name" command is executed, the storage engine name of the table is output.

Example 20 \d+ command output

```
postgres=> \d+ data1
                                        Table "public.data1"
 Column |         Type          | Collation | Nullable | Default | Storage  | Stats target |
Description
--------+-----------------------+-----------+----------+---------+----------+--------------+-
 c1     | numeric               |           |          |         | main     |              |
 c2     | character varying(10) |           |          |         | extended |              |
Access method: heap
```

Specifying on for the psql variable HIDE_TABLEAM (default value off) can suppress the output of the Access Method item.

Example 21 \d+ command output

```
postgres=> \set HIDE_TABLEAM on
postgres=> \d+ data1
                                        Table "public.data1"
 Column |         Type          | Collation | Nullable | Default | Storage  | Stats target |
Description
--------+-----------------------+-----------+----------+---------+----------+--------------+-
---------
 c1     | numeric               |           |          |         | main     |              |
 c2     | character varying(10) |           |          |         | extended |              |
```

□ Display of access method by psql command

Table access methods are also displayed with the \dA command. Previously, only the index access method was output.

Example 22 \dA command output

```
postgres=> \dA
List of access methods
  Name  | Type
--------+-------
 brin   | Index
 btree  | Index
 gin    | Index
 gist   | Index
 hash   | Index
 heap   | Table
 spgist | Index
(7 rows)
```

3.1.10. Pg_hba.conf file

The following new features have been added to the pg_hba.conf file.

□ New parameter of clientcert item

A new setting "verfy-full" can be specified for the clientcert parameter. This option also ensures that the certificate's "cn" (common name) matches the username or proper mapping.

□ GSSAPI authentication

Generic Security Standard Application Programming Interface (GSSAPI) is available for client authentication. An entry of hostgssenc / hostnogssenc can be specified in the pg_hba.conf file. In order to enable GSSAPI authentication, it is necessary to specify --with-gssapi as an option of 'configure' command at installation.

3.1.11. Text search

Supported languages for text search increased. It was 16 languages in PostgreSQL 11 but has increased to 22 languages in PostgreSQL 12.

Example 23 Text search configurations

```
postgres=# \dF
              List of text search configurations
   Schema   |    Name    |             Description
------------+------------+-------------------------------------
 pg_catalog | arabic     | configuration for arabic language      << new
 pg_catalog | danish     | configuration for danish language
 pg_catalog | dutch      | configuration for dutch language
 pg_catalog | english    | configuration for english language
 pg_catalog | finnish    | configuration for finnish language
 pg_catalog | french     | configuration for french language
 pg_catalog | german     | configuration for german language
 pg_catalog | hungarian  | configuration for hungarian language
 pg_catalog | indonesian | configuration for indonesian language << new
 pg_catalog | irish      | configuration for irish language      << new
 pg_catalog | italian    | configuration for italian language
 pg_catalog | lithuanian | configuration for lithuanian language << new
 pg_catalog | nepali     | configuration for nepali language
 pg_catalog | norwegian  | configuration for norwegian language  << new
 pg_catalog | portuguese | configuration for portuguese language
 pg_catalog | romanian   | configuration for romanian language
 pg_catalog | russian    | configuration for russian language
 pg_catalog | simple     | simple configuration
 pg_catalog | spanish    | configuration for spanish language
 pg_catalog | swedish    | configuration for swedish language
 pg_catalog | tamil      | configuration for tamil language     << new
 pg_catalog | turkish    | configuration for turkish language
(22 rows)
```

3.1.12. Libpq API

The following functions have been added to the C language interface to PostgreSQL.

□ PQresultMemorySize

"size_t PQresultMemorySize (const PGresult * res)" API has been added. This function returns the amount of memory allocated by PGresult. It can be used to manage application memory.

□ GetForeignDataWrapperExtended

"ForeignDataWrapper * GetForeignDataWrapperExtended (Oid fwdid, bits16 flags)" API has been added.

□ GetForeignServerExtended

"ForeignServer * GetForeignServerExtended (Oid fwdid, bits16 flags)" API has been added.

3.1.13. Transaction ID

The 64-bit transaction ID is available. API "GetTopFullTransactionId" and "GetCurrentFullTransactionId" are provided. However, the current heap does not use them.

3.1.14. Client environment variable

Environment variables PG_COLOR and PG_COLORS have been added. Specify whether to use the color of the diagnostic message in PG_COLOR. Possible values are always, auto or never. PG_COLORS is specified by combining the category of escape sequence and code with an equal sign (=). When specifying multiple categories, separate them with a colon (:).

Table 13 Environment variable PG_COLORS settings

Category	Default value	Note
error	01;31	
warning	01;35	
locus	01	

3.2. SQL statement

This section explains new features related to SQL statements.

3.2.1. ALTER TABLE

It is now possible to change the attributes of some system catalogs.

Example 24 Alter system catalog

```
postgres=# SHOW allow_system_table_mods ;
 allow_system_table_mods
-------------------------
 on
(1 row)

postgres=# ALTER TABLE pg_attribute SET
       (autovacuum_vacuum_scale_factor=0) ;
ALTER TABLE
```

3.2.2. ALTER TYPE ADD VALUE

The ALTER TYPE ADD VALUE statement is now available within a transaction block. However, the added value cannot be used in the transaction block.

Example 25 ALTER TYPE ADD VALUE statement in the transaction block

```
postgres=> BEGIN ;
BEGIN
postgres=> ALTER TYPE t1 ADD VALUE 'v3' ;
ALTER TYPE
```

3.2.3. COMMIT/ROLLBACK AND CHAIN

It became possible to add CHAIN clause which start a new transaction right after committing or discarding (ROLLBACK) the transaction. Specify the AND CHAIN clause in the COMMIT statement or ROLLBACK statement. Specify "AND NO CHAIN" to explicitly deny the CHAIN clause. These statements can also be used in PROCEDUREs using PL/pgSQL.

Example 26 COMMIT AND CHAIN

```
postgres=> BEGIN ;
BEGIN
postgres=> INSERT INTO data1 VALUES (100, 'data1') ;
INSERT 0 1
postgres=> COMMIT AND CHAIN ;
COMMIT
postgres=> INSERT INTO data1 VALUES (200, 'data2') ;
INSERT 0 1
postgres=> ROLLBACK AND CHAIN ;
ROLLBACK
postgres=> INSERT INTO data1 VALUES (300, 'data3') ;
INSERT 0 1
postgres=> COMMIT ;
COMMIT
```

In transactions started with the CHAIN clause, attributes such as transaction isolation level are maintained from the previous transaction.

Example 27 Transaction attribute

```
postgres=> SHOW transaction_isolation ;
 transaction_isolation
-----------------------
 read committed
(1 row)

postgres=> BEGIN ISOLATION LEVEL SERIALIZABLE ;
BEGIN
postgres=> COMMIT AND CHAIN ;
COMMIT
postgres=> SHOW transaction_isolation ;
 transaction_isolation
-----------------------
 serializable
(1 row)
postgres=> COMMIT ;
COMMIT
postgres=> SHOW transaction_isolation ;
 transaction_isolation
-----------------------
 read committed
(1 row)
```

3.2.4. COPY

The following enhancements have been made to the COPY statement.

□ COPY FROM statement

It has become possible to store only the table data that matches the conditions specified in the COPY FROM statement.

Example 28 COPY FROM WHERE statement

```
postgres=# COPY data1 FROM '/home/postgres/data1.csv' CSV DELIMITER ','
               WHERE mod(c1, 2) = 0 ;
COPY 50000
```

It can be executed similarly by \copy command of psql command.

Example 29 \copy WHERE command

```
postgres=> \copy data1 FROM '/home/postgres/data1.csv' CSV DELIMITER ','
               WHERE mod(c1, 2) = 0 ;
COPY 50000
```

□ COPY FREEZE statement

Executing a COPY FREEZE statement on partitioned table results in an error. This specification also applies to PostgreSQL 11.2.

Example 30 COPY FREEZE statement on PostgreSQL 12

```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY
RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES FROM (0) TO
(100) ;
CREATE TABLE
postgres=> CREATE TABLE part1v2 PARTITION OF part1 FOR VALUES FROM (100)
TO (200) ;
CREATE TABLE
postgres=# COPY part1 FROM '/home/postgres/part1.csv' CSV FREEZE ;
ERROR:  cannot perform FREEZE on a partitioned table
```

3.2.5. CREATE AGGREGATE

It is able to use the OR REPLACE clause in the CREATE AGGREGATE statement.

Syntax

```
CREATE [ OR REPLACE ] AGGREGATE name ( [ argmode ] [ argname ]
arg_data_type [ , ... ] )
```

3.2.6. CREATE INDEX

□ Creation of GiST index

Covering indexes are now available for GiST indexes.

Example 31 Covering index for GiST index

```
postgres=> CREATE TABLE data1(c1 INT, c2 box, c3 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE INDEX idx1_data1 ON data1 USING gist (c2) INCLUDE (c1) ;
CREATE INDEX
postgres=> \d data1
                    Table "public.data1"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 c1     | integer               |           |          |
 c2     | box                   |           |          |
 c3     | character varying(10) |           |          |
Indexes:
    "idx1_data1" gist (c2) INCLUDE (c1)
```

□ GiST Index and VACUUM

Free pages are now reused by VACUUM.

□ WAL when creating GIN index

The amount of WAL output at indexing time has been significantly reduced.

3.2.7. CREATE STATISTICS

The mcv clause can be used in the CREATE STATISTICS statement. This value indicates Multivariate most-common values (MCV). It expands the regular MCV list and tracks the most frequent value combinations. The acquired statistics are stored in the stxmcv column of the pg_statistic_ext catalog.

Example 32 Create MCV statistics

```
postgres=> CREATE TABLE stat1 (c1 NUMERIC, c2 NUMERIC, c3
VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE STATISTICS mcv_stat1(mcv) ON c1, c2 FROM stat1 ;
CREATE STATISTICS
```

3.2.8. CREATE TABLE

The following enhancements have been made to the CREATE TABLE statement.

□ GENERATED column

The generated column is the column defined as the column based on calculation results to the table. At the time of column definition, specify GENERATED ALWAYS AS (calculation expression) STORED clause following data type.

Example 33 GENERATED column definition

```
postgres=> CREATE TABLE gen1(c1 VARCHAR(10), c2 VARCHAR(10), c3 VARCHAR(20)
GENERATED ALWAYS AS (c1 || c2) STORED) ;
CREATE TABLE
postgres=> \d gen1
                                  Table "public.gen1"
 Column |        Type         | Collation | Nullable |Default
--------+---------------------+-----------+----------+------------------
 c1     | character varying(10) |          |          |
 c2     | character varying(10) |          |          |
 c3     | character varying(20) |          |          | generated always as
(((c1::text || c2::text))) stored
```

Values cannot be specified directly for generated columns in INSERT and UPDATE statements. Only the DEFAULT clause is valid.

Example 34 Update directly for GENERATED column

```
postgres=> INSERT INTO gen1 VALUES ('AB', 'CD', 'EF') ;
psql: ERROR:  cannot insert into column "c3"
DETAIL:  Column "c3" is a generated column.
postgres=> INSERT INTO gen1 VALUES ('AB', 'CD', DEFAULT) ;
INSERT 0 1
```

The generated column values are calculated when executing an INSERT or UPDATE statement, and the calculated values are physically stored.

Example 35 Stored the GENERATED column value

```
postgres=# SELECT heap_page_items(get_raw_page('gen1', 0)) ;
                          heap_page_items
------------------------------------------------------------------------
 (1,8152,1,35,524,0,0,"(0,1)",3,2050,24,,,"\\x07414207434406b41424344")
(1 row)
```

In the above example, it is displayed that the c1 column is 'AB' (= 0x4142), the c2 column is 'CD' (= 0x4344), and the c3 column is 'ABCD' (= 0x41424344).

Information on generated columns can be found by storing "s" in the attgenerated column added to the pg_attrdef catalog. In addition, information_schema column_column_usage table is newly added.

Example 36 GENERATED column information

```
postgres=> SELECT * FROM information_schema.column_column_usage ;
 table_catalog | table_schema | table_name | column_name | dependent_column
---------------+--------------+------------+-------------+-----------------
 postgres      | public       | gen1       | c1          | c3
 postgres      | public       | gen1       | c2          | c3
(2 rows)

postgres=> SELECT attname, attgenerated FROM pg_attribute WHERE attname IN
('c1', 'c2', 'c3') ;
 attname | attgenerated
---------+--------------
 c1      |
 c2      |
 c3      | s
(3 rows)
```

A generated column cannot be specified as a partitioning key. In addition, it is not possible to define a generated column that depends on the other generated columns.

Example 37 Restriction of GENERATED column

```
postgres=> CREATE TABLE pgen1(c1 INT, c2 INT, c3 INT GENERATED ALWAYS AS
       (c1 + c2) STORED) PARTITION BY RANGE(c3) ;
psql: ERROR:  cannot use generated column in partition key
LINE 1: ...NERATED ALWAYS AS (c1 + c2) STORED) PARTITION BY RANGE(c3) ;
                                                                 ^
DETAIL:  Column "c3" is a generated column.

postgres=> CREATE TABLE gen2 (c1 INT, c2 INT GENERATED ALWAYS AS (c1*2)
       STORED, c3 INT GENERATED ALWAYS AS (c2*2) STORED) ;
psql: ERROR:  cannot use generated column "c2" in column generation
expression
LINE 1: ...AYS AS (c1*2) STORED, c3 INT GENERATED ALWAYS AS (c2*2) STOR...
                                                             ^
DETAIL:  A generated column cannot reference another generated column.
```

□ TABLESPACE clause of partition table definition

TABLESPACE clause is now enabled when creating the partition table. In previous versions, the TABLESPACE clause was ignored. Also, the value of the TABLESPACE clause of the partitioned table becomes the default tablespace at the time of partition creation.

Example 38 Partitioned Table Creation and TABLESPACE Clause

```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY
              LIST(c1) TABLESPACE ts1 ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES IN
(100) ;
CREATE TABLE
postgres=> \d part1
              Partitioned table "public.part1"
 Column |        Type          | Collation | Nullable | Default
--------+----------------------+-----------+----------+---------
 c1     | numeric              |           |          |
 c2     | character varying(10) |           |          |
Partition key: LIST (c1)
Number of partitions: 1 (Use \d+ to list them.)
Tablespace: "ts1"

postgres=> \d part1v1
                  Table "public.part1v1"
 Column |        Type          | Collation | Nullable | Default
--------+----------------------+-----------+----------+---------
 c1     | numeric              |           |          |
 c2     | character varying(10) |           |          |
Partition of: part1 FOR VALUES IN ('100')
Tablespace: "ts1"
```

□ FOR VALUES clause of partition table

It is now possible to specify a formula or function instead of a literal in the FOR VALUES clause of a partition. The specified formula is executed only once when the CREATE TABLE statement is executed, and the calculated value is stored in the table definition.

Example 39 FOR VALUES clause of partition table

```
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES IN
       (power(2, 3)) ;
CREATE TABLE
postgres=> \d part1v1
                  Table "public.part1v1"
 Column |        Type         | Collation | Nullable | Default
--------+---------------------+-----------+----------+---------
 c1     | numeric             |           |          |
 c2     | character varying(10) |           |          |
Partition of: part1 FOR VALUES IN ('8')
Tablespace: "ts1"
```

□ A foreign key reference to the partition table

The partition table can be referenced as a foreign key.

Example 40 Refer to partition table as reference table

```
 postgres=> CREATE TABLE fkey1(c1 INT PRIMARY KEY, c2 VARCHAR(10))
PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE fkey1v1 PARTITION OF fkey1 FOR VALUES FROM (0)
TO (1000000) ;
CREATE TABLE
postgres=> CREATE TABLE fkey1v2 PARTITION OF fkey1 FOR VALUES FROM
(1000000) TO (2000000) ;
CREATE TABLE
postgres=> CREATE TABLE ref1(c1 INT REFERENCES fkey1(c1), c2
VARCHAR(10)) ;
```

In PostgreSQL 11, when trying to create the ref1 table, "ERROR: cannot reference partitioned table" fkey1 "error occurred.

□ VACUUM processing for index

By specifying the VACUUM_INDEX_CLEANUP = OFF in WITH clause, it will be able to disable

the VACUUM process for the index. The default value is 'ON', and VACUUM will be performed as before.

Example 41 Suppressing VACUUM on indexes

```
postgres=> CREATE TABLE vacuum1(c1 INT, c2 VARCHAR(10)) WITH (VACUUM_INDEX_CLEANUP = OFF) ;
CREATE TABLE
postgres=> \d+ vacuum1
                                    Table "public.vacuum1"
 Column |        Type         | Collation | Nullable | Default | Storage  | Stats target |
Description
--------+----------------------+-----------+----------+---------+----------+--------------+-
 c1     | integer              |           |          |         | plain    |              |
 c2     | character varying(10) |           |          |         | extended |              |
Access method: heap
Options: vacuum_index_cleanup=off
```

□ Truncate empty block at the end of the table

VACUUM_TRUNCATE has been added to the attributes of the table. Determines whether the truncate empty block at the end of the table is released when performing VACUUM. The default value is 'ON', which truncate empty space as before. When it is specified as 'OFF', this operation is not performed.

Example 42 Truncate empty block at the end of the table

```
postgres=> CREATE TABLE vacuum1(c1 INT, c2 VARCHAR(10)) WITH
        (VACUUM_TRUNCATE = OFF) ;
CREATE TABLE
postgres=> \d+ vacuum1
                                        Table "public.vacuum1"
 Column |        Type         | Collation | Nullable | Default | Storage
| Stats target | Description
--------+---------------------+-----------+----------+---------+--------
------+---------------------+-----------+----------+---------+----+-----
--------
 c1     | integer             |           |          |         | plain
|           |
 c2     | character varying(10) |           |          |         | extended
|           |
Access method: heap
Options: vacuum_truncate=off
```

3.2.9. EXPLAIN

The SETTINGS ON option can now be specified in the EXPLAIN statement. This option outputs information on parameters related to the execution plan being changed from the default value.

Example 43 EXPLAIN (SETTINGS ON)

```
postgres=> SET random_page_cost = 1.0 ;
SET
postgres=> EXPLAIN (SETTINGS ON) SELECT * FROM data1 WHERE c1=100 ;
                              QUERY PLAN
-----------------------------------------------------------------------
 Index Scan using idx1_data1 on data1  (cost=0.29..2.31 rows=1
width=12)
   Index Cond: (c1 = '100'::numeric)
 Settings: random_page_cost = '1'
(3 rows)
```

3.2.10. REINDEX CONCURRENTLY

The CONCURRENTLY option can now be added to the REINDEX statement. By reducing the lock range, application operation can be made to coexist with re-indexing. This feature is realized by temporarily creating a new index ({index_name}_ccnew) and replacing it with the old index.

Example 44 REINDEX CONCURRENTLY statement

```
postgres=> REINDEX (VERBOSE) TABLE CONCURRENTLY data1 ;
psql: INFO:  index "public.idx1_data1" was reindexed
psql: INFO:  index "pg_toast.pg_toast_16385_index" was reindexed
psql: INFO:  table "public.data1" was reindexed
DETAIL:  CPU: user: 3.25 s, system: 0.69 s, elapsed: 13.27 s.
REINDEX
```

In accordance with the change of REINDEX statement, --concurrently option was also added to the reindexdb command.

Example 45 reindexdb command

```
$ reindexdb --dbname postgres --echo --concurrently
SELECT pg_catalog.set_config('search_path', '', false);
REINDEX DATABASE CONCURRENTLY postgres;
WARNING:  concurrent reindex is not supported for catalog relations,
skipping all
$
```

3.2.11. PL/pgSQL additional check

The following values can now be specified for the parameter plpgsql.extra_warnings: Both can output additional warnings or errors when the function is executed.

□ strict_multi_assignment setting

A warning is output when the number of columns output by the SELECT INTO statement does not match the number of input variables. In the example below, two warnings occur in the function.

Example 46 strict_multi_assignment setting

```
postgres=> SET plpgsql.extra_warnings = 'strict_multi_assignment' ;
SET
postgres=> CREATE OR REPLACE FUNCTION strict1()
  RETURNS void
  LANGUAGE plpgsql
AS $$
  DECLARE
     x INTEGER ;
     y INTEGER ;
  BEGIN
     SELECT 1 INTO x, y ;
     SELECT 1, 2, 3 INTO x, y ;
  END ;
$$ ;
CREATE FUNCTION
postgres=> SELECT strict1() ;
psql: WARNING:  number of source and target fields in assignment do not
match
DETAIL:  strict_multi_assignment check of extra_warnings is active.
HINT:  Make sure the query returns the exact list of columns.
psql: WARNING:  number of source and target fields in assignment do not
match
DETAIL:  strict_multi_assignment check of extra_warnings is active.
HINT:  Make sure the query returns the exact list of columns.
 strict1
---------

(1 row)
```

□ too_many_rows setting

An error is generated when multiple records are returned by the SELECT INTO statement, and execution of the procedure is stopped.

Example 47 too_many_rows setting

```
postgres=> SET plpgsql.extra_errors to 'too_many_rows' ;
SET
postgres=> DO $$
       DECLARE x INTEGER ;
       BEGIN
           SELECT generate_series(1,2) INTO x ;
           RAISE NOTICE 'test output' ;
       END ;
       $$ ;
psql: ERROR:  query returned more than one row
HINT:  Make sure the query returns a single row, or use LIMIT 1
CONTEXT:  PL/pgSQL function inline_code_block line 4 at SQL statement
```

3.2.12. VACUUM / ANALYZE

The following features have been added to the VACUUM and ANALYZE statements.

□ SKIP_LOCKED clause

When executing a VACUUM or ANALYZE statement on a locked table, it has been waiting for the lock to be released. In PostgreSQL 12, an option SKIP_LOCKED clause has been added to skip locked tables. When processing is skipped, a log of WARNING level (LOG level for automatic VACUUM) is output. SQLSTATEs are considered successful even if skipped.

Example 48 Lock table

```
postgres=> BEGIN ;
BEGIN
postgres=> LOCK TABLE lock1 IN EXCLUSIVE MODE ;
LOCK TABLE
```

Example 49 Skip VACUUM for locked table

```
postgres=> VACUUM (SKIP_LOCKED) lock1 ;
psql: WARNING:  skipping vacuum of "lock1" --- lock not available
VACUUM
postgres=> \echo :SQLSTATE
00000
```

□ Option specification syntax

In the VACUUM statement and ANALYZE statement, it is now possible to specify the action to be performed with TRUE/FALSE or ON/OFF.

Example 50 Operation specification by ON/OFF

```
postgres=> VACUUM (VERBOSE OFF, FULL ON, ANALYZE ON) data1 ;
VACUUM
postgres=> VACUUM (VERBOSE TRUE, FULL TRUE, ANALYZE FALSE) data1 ;
psql: INFO:  vacuuming "public.data1"
psql: INFO:  "data1": found 0 removable, 1000000 nonremovable row versions
in 5406 pages
DETAIL:  0 dead row versions cannot be removed yet.
CPU: user: 0.23 s, system: 0.28 s, elapsed: 0.72 s.
VACUUM
```

□ Suppress VACUUM for index option

By specifying the OFF to INDEX_CLEANUP clause in VACUUM statement, it will be able to suppress the VACUUM process for the index. If this specification is omitted, the operation depends on the VACUUM_INDEX_CLEANUP attribute of the table.

Example 51 Suppress VACUUM for index

```
postgres=> VACUUM (VERBOSE ON, INDEX_CLEANUP OFF) data1 ;
psql: INFO:  vacuuming "public.data1"
psql: INFO:  "data1": found 0 removable, 0 nonremovable row versions in 0
out of 0 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 493
There were 0 unused item pointers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
0 tuples and 0 item identifiers are left as dead.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
VACUUM
```

□ Suppression of free page truncation processing at the end of the table

The TRUNCATE clause has been added to the VACUUM statement. By specifying OFF for this attribute, it is possible to suppress free area deletion processing at the end of the table. If omitted, it depends on the VACUUM_TRUNCATE attribute of the table.

Example 52 Suppress truncation processing

```
postgres=> VACUUM (VERBOSE ON, TRUNCATE OFF) data1 ;
psql: INFO:  vacuuming "public.data1"
psql: INFO:  "data1": removed 50000 row versions in 541 pages
psql: INFO:   "data1": found 50000 removable, 50000 nonremovable row
versions in 541 out of 541 pages
…
There were 0 unused item pointers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
VACUUM
```

3.2.13. WITH SELECT

In the past, common table expressions (CTE) specified in the WITH clause were all MATERIALIZED. In PostgreSQL 12, the default behavior has been changed to NOT MATERIALIZED. In order to change these behaviors, it is possible to specify MATERIALIZED or NOT MATERIALIZED in the WITH clause. Specifying the NOT MATERIALIZED clause allows the specification of the WHERE clause to be pushed down into the WITH clause.
In the example below, using the NOT MATERIALIZED clause shows that index search is selected and the cost is reduced.

Example 53 WITH NOT MATERIALIZED

```
postgres=> EXPLAIN WITH s AS NOT MATERIALIZED (SELECT * FROM data1)
       SELECT * FROM s WHERE c1=100 ;
                            QUERY PLAN
------------------------------------------------------------------------
--
 Index Scan using data1_pkey on data1  (cost=0.42..8.44 rows=1
width=12)
   Index Cond: (c1 = '100'::numeric)
(2 rows)
```

Example 54 WITH MATERIALIZED

```
postgres=> EXPLAIN WITH s AS MATERIALIZED (SELECT * FROM data1)
       SELECT * FROM s WHERE c1=100 ;
                            QUERY PLAN
------------------------------------------------------------------------
 CTE Scan on s  (cost=15406.00..37906.00 rows=5000 width=70)
   Filter: (c1 = '100'::numeric)
   CTE s
     ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=12)
(4 rows)
```

3.2.14. Functions

The following functions have been added/enhanced.

□ SQL/JSON

Some functions related to SQL / JSON proposed in SQL 2016 standard are provided.

Example 55 jsonb_path_query_array function

```
postgres=> SELECT jsonb_path_query_array('{"a":[1,2,3,4,5]}', '$.a[*] ?
(@ >= $min && @ <= $max)', '{"min":2,"max":4}') ;
 jsonb_path_query_array
------------------------
 [2, 3, 4]
(1 row)
```

The following functions have been added.

Table 14 JSON/SQL functions

Function name	Description
jsonb_path_exists	Checks whether the JSON path returns any item for the specified JSON value.
jsonb_path_match	Returns JSON path predicate result for the specified JSON value. Only the first result item is taken into account.
jsonb_path_query	Gets all JSON items returned by JSON path for the specified JSON value.
jsonb_path_query_array	Gets all JSON items returned by JSON path for the specified JSON value and wraps result into an array.
jsonb_path_query_first	Gets the first JSON item returned by JSON path for the specified JSON value.

□ Pg_partition_tree

Pg_partition_tree is a function to display the tree structure of the partition table. It also supports hierarchical partition structure. Specify a partition table in the parameter. Specifying an object name other than a partition table or partition returns NULL.

Example 56 pg_partition_tree function

```
postgres=> SELECT * FROM pg_partition_tree('part1') ;
  relid  | parentrelid | isleaf | level
---------+-------------+--------+-------
 part1   |             | f      |     0
 part1v1 | part1       | t      |     1
 part1v2 | part1       | t      |     1
(3 rows)

postgres=> SELECT pg_size_pretty(sum(pg_relation_size(relid))) AS total
FROM pg_partition_tree('part1') ;
 total
-------
 84 MB
(1 row)
```

□ Pg_partition_root

The pg_partition_root is a function that returns the top-level partition table name of a specified partition. In the following example, a sub partition is created and the pg_partition_root function is executed.

Example 57 pg_partition_root function

```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 NUMERIC, c3 VARCHAR(10))
PARTITION BY LIST(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES IN (100)
PARTITION BY LIST (c2) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1v2 PARTITION OF part1v1 FOR VALUES IN (200) ;
CREATE TABLE
postgres=> SELECT pg_partition_root('part1v1v2') ;
 pg_partition_root
-------------------
 part1
(1 row)
```

□ Pg_partition_ancestors

The pg_partition_ancestors function prints the list towards the parent of the partition table that contains the specified partition.

Example 58 pg_partition_ancestors function

```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY
LIST(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES IN (100) ;
CREATE TABLE
postgres=> SELECT pg_partition_ancestors('part1v1') ;
 pg_partition_ancestors
------------------------
 part1v1
 part1
(2 rows)
```

□ Pg_promote

This function promotes a standby instance to a primary instance. Previously, it was necessary to execute the pg_ctl promote command. Specify the parameter which determines whether to wait (default true) and the number of seconds to wait (default 60 seconds). This function returns false if processing fails or if the promotion is not completed within the waiting time, otherwise, it returns true.

Example 59 pg_promote function

```
postgres=# SELECT pg_promote(true, 90) ;
 pg_promote
------------
 t
(1 row)
```

□ Pg_ls_tmpdir

The pg_ls_tmpdir function has been added to return a list of file names where temporary data has been saved. Specify the OID of the TABLESPACE in the parameter. If omitted, it is assumed that pg_default is specified. This function requires the SUPERUSER privilege or the pg_monitor role.

Example 60 pg_ls_tmpdir function

```
postgres=# SELECT * FROM pg_ls_tmpdir() ;
      name       |   size    |      modification
-----------------+-----------+------------------------
 pgsql_tmp36911.6 | 148955136 | 2019-05-24 11:57:36+09
 pgsql_tmp37050.0 | 139722752 | 2019-05-24 11:57:36+09
 pgsql_tmp37051.0 | 138403840 | 2019-05-24 11:57:36+09
(3 rows)
```

□ Pg_ls_archive_statusdir

The pg_ls_archive_statusdir function has been added to obtain the status of the archive file. This function searches the ${PGDATA}/pg_wal/archive_status directory and outputs the file name, size, and modification date. It does not output the actual information of archived WAL files. This function requires the SUPERUSER privilege or the pg_monitor role.

Example 61 pg_ls_archive_statusdir function

```
postgres=# SELECT * FROM pg_ls_archive_statusdir() ;
            name              | size |      modification
------------------------------+------+------------------------
 00000001000000000000000D.done |    0 | 2019-05-24 19:33:00+09
 00000001000000000000000E.done |    0 | 2019-05-24 19:33:01+09
 00000001000000000000000F.done |    0 | 2019-05-24 19:33:02+09
(3 rows)
```

□ Date_trunc

This function can now set Timezone.

Example 62 date_trunc function with timezone

```
postgres=> SELECT date_trunc('day', TIMESTAMP WITH TIME ZONE '2019-05-
24 20:38:40+00', 'Asia/Tokyo') ;
      date_trunc
------------------------
 2019-05-25 00:00:00+09
(1 row)
```

□ Hyperbolic functions

The following hyperbolic functions included in SQL:standards 2016 have been added

Table 15 Hyperbolic functions

Function name	Description	Note
log10	base 10 logarithm	
sinh	hyperbolic sine	
cosh	hyperbolic cosine	
tanh	hyperbolic tangent	
asinh	inverse hyperbolic sine	
acosh	inverse hyperbolic cosine	
atanh	inverse hyperbolic tangent	

Example 63 Hyperbolic functions

```
postgres=> SELECT log10(20), sinh(1) ;
      log10       |       sinh
------------------+------------------
 1.3010299956639813 | 1.1752011936438014
(1 row)
```

□ Copy replication slot

The function that performs a copy of an existing replication slot has been provided. The pg_copy_physical_replication_slot function and the pg_copy_logical_replication_slot function are provided according to the type of replication slot. To make a copy, it is necessary to replication slot is being used.

Example 64 Copy replication slot

```
postgres=# SELECT pg_create_physical_replication_slot('slot_1') ;
 pg_create_physical_replication_slot
-------------------------------------
 (slot_1,)
(1 row)
postgres=#      SELECT      pg_copy_physical_replication_slot('slot_1',
'slot_c') ;
psql: ERROR:  cannot copy a replication slot that doesn't reserve WAL
```

3.3. Configuration parameters

In PostgreSQL 12 the following parameters have been changed.

3.3.1. Added parameters

The following parameters have been added.

Table 16 Added parameters

Parameter name	Description (context)	Default Val
archive_cleanup_command	Migrate from recovery.conf (sighup)	-
data_sync_retry (Also added to 11.2)	Behavior when fsync system call fails (postmaster)	off
default_table_access_method	Default storage engine name	heap
log_statement_sample_rate	Rate of output of SQL statement to log (superuser)	1
log_transaction_sample_rate	Rate of output of transaction control statement to log (superuser)	0
plan_cache_mode	Changed the behavior of caching execution plans (user)	auto
primary_conninfo	Migration from recovery.conf file (postmaster)	-
primary_slot_name	Migrate from recovery.conf (postmaster)	-
promote_trigger_file	Migrate from recovery.conf (sighup)	-
recovery_end_command	Migrate from recovery.conf (sighup)	-
recovery_min_apply_delay	Migrate from recovery.conf (sighup)	0
recovery_target	Migrate from recovery.conf (postmaster)	-
recovery_target_action	Migrate from recovery.conf (postmaster)	pause
recovery_target_inclusive	Migrate from recovery.conf (postmaster)	on
recovery_target_lsn	Migrate from recovery.conf (postmaster)	-
recovery_target_name	Migrate from recovery.conf (postmaster)	-
recovery_target_time	Migrate from recovery.conf (postmaster)	-
recovery_target_timeline	Migrate from recovery.conf (postmaster)	latest
recovery_target_xid	Migrate from recovery.conf (postmaster)	-
restore_comand	Migrate from recovery.conf (postmaster)	-
shared_memory_type	Type of shared memory (postmaster)	OS 依存
ssl_library	Name of library providing SSL function	-

Parameter name	Description (context)	Default Val
	(internal)	
ssl_max_protocol_version	Maximum version of SSL protocol to support (sighup)	-
ssl_min_protocol_version	Minimum version of SSL protocol to support (sighup)	TLSv1
tcp_user_timeout	TCP timeout specification (user)	0
wal_init_zero	Fill-in WAL file to zero (superuser)	on
wal_recycle	Recycle WAL File (superuser)	on

□ Primary_conninfo parameter

Specifies the connection string for the primary instance. Previously this was specified in the recovery.conf file. The default value for the application_name entry has traditionally been "walreceiver", but when the cluster_name parameter is specified, the value for cluster_name has been changed to the default value. The value of the application_name column in the pg_stat_replication catalog changes.

□ Ssl_library parameter

This parameter indicates the name of the library that provides the SSL feature. The parameter value is "OpenSSL" when --with-openssl is specified when executing configure command in Red Hat Enterprise Linux environment.

Example 65 ssl_library parameter

```
postgres=# SHOW ssl_library ;
 ssl_library
-------------
 OpenSSL
(1 row)
```

□ Shared_memory_type parameter

This parameter specifies the type of shared memory (such as shared_buffers).

Table 17 shared_memory_type parameter value

Setting	Description	System call
mmap	Use an anonymous memory map.	mmap
sysv	Use System V shared memory.	shmget
windows	Use Windows shared memory.	CreateFileMapping

The default value for this parameter on Linux is mmap. This is the same behavior as PostgreSQL 9.3 and later. Very small System V shared memory and most shared memory are configured using memory map file (mmap). Setting this parameter to sysv allows you to revert to the pre-PostgreSQL 9.2 behavior. In this case, configure all shared memory using System V shared memory.

□ Plancache_mode parameter

This parameter sets the method for caching the execution plan of a prepared statement (created by the PREPARE statement). The default value is auto, which is the same behavior as previous versions. Normally, an execution plan is generated each time the SQL statement created by the PREPARE statement is executed by the EXECUTE statement. In the following example, it can be seen that the execution plan changes depending on the parameters specified in the EXECUTE statement. It indicates that the data stored in column C2 is uneven, there is little plan0 data (does an index search), and there is a lot of plan1 data (a whole table is searched).

Example 66 Execution plan changes with each execution of EXECUTE statement

```
postgres=> PREPARE sel1(VARCHAR) AS SELECT * FROM plan1 WHERE c2=$1 ;
PREPARE
postgres=> EXPLAIN EXECUTE sel1('plan0') ;
                              QUERY PLAN
-----------------------------------------------------------------------
 Index Scan using idx1_plan1 on plan1  (cost=0.42..4.44 rows=1 width=12)
   Index Cond: ((c2)::text = 'plan0'::text)
(2 rows)
postgres=> EXPLAIN EXECUTE sel1('plan1') ;
                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on plan1  (cost=0.00..17906.01 rows=1000001 width=12)
   Filter: ((c2)::text = 'plan1'::text)
(2 rows)
```

If the same SQL statement is executed 5 times or more, the execution plan may be cached, and the cached execution plan (general execution plan) may be used even if the parameter is changed from the next time. In the example below, the display in the execution plan changes from a literal value to $1 at the execution of the sixth EXPLAIN statement.

Example 67 Execution plan is cached

```
postgres=> EXPLAIN EXECUTE sel1('plan1') ; -- Repeat 5 times
                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on plan1  (cost=0.00..23311.01 rows=1000001 width=12)
   Filter: ((c2)::text = 'plan1'::text)
(2 rows)
postgres=> EXPLAIN EXECUTE sel1('plan1') ; -- Sixth (execution plan
changed)
                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on plan1  (cost=0.00..23311.01 rows=1000001 width=12)
   Filter: ((c2)::text = ($1)::text)
(2 rows)
```

The newly added parameter plan_cache_mode changes this behavior. Setting the parameter value to "force_custom_plan" turns off execution plan caching. On the other hand, setting the parameter value to "force_generic_plan" immediately enables execution plan caching.

Example 68 Setting value force_generic_plan

```
postgres=> SET plan_cache_mode = force_generic_plan ;
SET
postgres=> PREPARE sel1(VARCHAR) AS SELECT * FROM plan1 WHERE c2=$1 ;
PREPARE
postgres=> EXPLAIN EXECUTE sel1('plan0') ;
                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on plan1  (cost=0.00..23312.50 rows=1000120 width=12)
   Filter: ((c2)::text = ($1)::text)
(2 rows)
```

□ Data_sync_retry parameter

This parameter determines what happens if the fsync system call issued during a checkpoint fails. In previous versions, the fsync function was re-executed (data_sync_retry = "on"), and the default behavior of the new version (data_sync_retry = "off") caused an instance stoppage by PANIC if the fsync system call failed. This parameter has been added since PostgreSQL 11.2.

3.3.2. Changed parameters

The setting range and options were changed for the following configuration parameters.

Table 18 Changed parameters

Parameter name	Changes
client_min_messages	It is no longer possible to set it to a higher level than ERROR.
dynamic_shared_memory_type	The setting value "none" has been deleted.
log_autovacuum_min_duration	Log output contents now change according to VACUUM execution status.
log_connections	Application_name information has been added to the log.
plpgsql.extra_warnings	The following parameter values have been added: - too_many_rows - strict_multi_assignment
trace_sort	The log output message has been changed.
wal_level	It is now checked for proper level at startup.
wal_sender_timeout	The context has been changed from sighup to user.
default_with_oids	It cannot be set to "on".
recovery_target_timeline	"Current" has been added as a setting value, the default value has been changed to "latest".
autovacuum_vacuum_cost_delay	Data type changed from integer to real.

The default_with_oids parameter cannot be referenced from the pg_settings catalog.

Example 69 default_with_oids parameter

```
postgres=> SHOW default_with_oids ;
default_with_oids
-------------------
 off
(1 row)
postgres=> SET default_with_oids = on ;
psql: ERROR:  tables declared WITH OIDS are not supported
postgres=> SELECT COUNT(*) FROM pg_settings WHERE
name='default_with_oids' ;
 count
-------
     0
(1 row)
```

□ Wal_sender_timeout parameter

This parameter can now be changed by the user on a per session basis. This allows changing parameters on a per-connection basis from slave instances in a streaming replication environment.

Example 70 wal_sender_timeout parameter

```
$ grep primary_conninfo data/postgresql.conf
primary_conninfo = 'host=svrhost1 port=5432 user=postgres
password=password options=''-c wal_sender_timeout=5000'''
```

□ Log_connections parameter

The value of the application_name parameter is added to the log that is output when this parameter is set to 'on'.

Example 71 log_connections parameter

```
□ Connection from psql command
LOG:  connection authorized: user=postgres database=postgres
application_name=psql

□ Connection from pg_basebackup command
LOG:  replication connection authorized: user=postgres
application_name=pg_baseback

□ Connection from Streaming Replication slave instance
LOG:  replication connection authorized: user=postgres
application_name=walreceiver
```

□ Trace_sort parameter

The output log format has been changed when this parameter is set to 'on'.

Example 72 PostgreSQL 11 log (part)

```
LOG:  -1 switching to external sort with 16 tapes: CPU: user: 0.00 s,
system: 0.00 s, elapsed: 0.00 s
LOG:  -1 using 3951 KB of memory for read buffers among 15 input tapes
LOG:  performsort of -1 done (except 15-way final merge): CPU: user:
0.15 s, system: 0.01 s, elapsed: 0.16 s
```

Example 73 PostgreSQL 12 log (part)

```
LOG:  worker -1 switching to external sort with 16 tapes: CPU: user:
0.00 s, system: 0.00 s, elapsed: 0.00 s
LOG: worker -1 starting quicksort of run 1: CPU: user: 0.00 s, system:
0.00 s, elapsed: 0.00 s
```

□ Wal_level parameter

When a replication slot is used, it became to be checked whether the wal_level parameter is an appropriate value during the instance startup. If the required level is not set, instance startup will fail. In the following example, wal_level is changed to "replica" and the instance is restarted in the Logical Replication environment.

Example 74 wal_level value check

```
postgres=# ALTER SYSTEM SET wal_level=minimal ;
ALTER SYSTEM
postgres=# \q
$
$ pg_ctl -D data restart
waiting for server to shut down.... done
server stopped
waiting for server to start....2019-05-24 23:09:17.918 JST [32486] FATAL:
WAL archival cannot be enabled when wal_level is "minimal"
 stopped waiting
pg_ctl: could not start server
Examine the log output.
$
```

3.3.3. Parameters with default values changed

The default values of the following configuration parameters have been changed.

Table 19 Parameters that default value has been changed

Parameter name	PostgreSQL 11	PostgreSQL 12	Note
autovacuum_vacuum_cost_delay	20	2	
extra_float_digits	0	1	
jit	off	on	
recovery_target_timeline	''	latest	
server_version	11.3	12beta1	
server_version_num	110003	120000	
transaction_isolation	default	read committed	

3.4. Utilities

Describes the major enhancements of utility commands.

3.4.1. configure

The "--disable-strong-random" option has been removed from the "configure" command that is run when installing PostgreSQL from source code. The --with-gssapi option for using GSSAPI has been added.

3.4.2. initdb

The initdb command now refers to the /etc/localtime file when determining the time zone of a database cluster. This file is referenced when the environment variable TZ is not specified.

3.4.3. oid2name

The oid2name command has been reviewed for options, and long-named options are now available.

Table 20 Added options

Short option	Added long-named option	Description
-f	--filenode	File node specification
-i	--indexes	Include index and sequence
-o	--oid	Specify OID
-q	--quiet	Omit header
-s	--tablespaces	Show tablespace OID
-S	--system-objects	Contains system objects
-t	--table	Specify table name
-x	--extended	Output additional information
-d	--dbname	Database to connect to
-h	--host	Hostname to connect to (-H option is deprecated)
-p	--port	Port number to connect to
-U	--username	Database username

3.4.4. pg_basebackup

The behavior of the pg_basebackup command has been changed when the --write-recovery-conf parameter (-R) is specified. The standby.signal file is automatically created in the backup destination folder, and the primary_conninfo parameter is added to the postgresql.auto.conf file. This behavior is only performed when connecting to a PostgreSQL 12 or later instance.

Example 75 -R parameter of pg_basebackup command

```
$ pg_basebackup -D back -R
$ ls back
backup_label      log           pg_ident.conf  pg_replslot   pg_stat_tmp
PG_VERSION           postgresql.conf
base              pg_commit_ts  pg_logical     pg_serial     pg_subtrans
pg_wal               standby.signal
current_logfiles  pg_dynshmem   pg_multixact   pg_snapshots  pg_tblspc
pg_xact
global            pg_hba.conf   pg_notify      pg_stat       pg_twophase
postgresql.auto.conf
$ cat back/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
primary_conninfo  =  'user=postgres  passfile=''/home/postgres/.pgpass''
port=5432    sslmode=disable    sslcompression=0    gssencmode=disable
target_session_attrs=any'
primary_slot_name = 'slot1'
$
```

3.4.5. pg_checksums

The command pg_verify_checksums, which checks checksum integrity, has been renamed to pg_checksums. In addition to checking the integrity of checksums, it is now possible to change checksum activation/deactivation.

Example 76 Usage of pg_checksums command

```
$ pg_checksums --help
pg_checksums enables, disables or verifies data checksums in a PostgreSQL
database cluster.

Usage:
  pg_checksums [OPTION]... [DATADIR]

Options:
 [-D, --pgdata=]DATADIR  data directory
  -c, --check            check data checksums (default)
  -d, --disable          disable data checksums
  -e, --enable           enable data checksums
  -N, --no-sync          do not wait for changes to be written safely to
disk
  -P, --progress         show progress information
  -v, --verbose          output verbose messages
  -r RELFILENODE         check only relation with specified relfilenode
  -V, --version          output version information, then exit
  -?, --help             show this help, then exit

If no data directory (DATADIR) is specified, the environment variable
PGDATA is used.

Report bugs to <pgsql-bugs@lists.postgresql.org>.
$
```

Specify the --check the option to check the integrity of the block. If the consistency error is confirmed, the command ends with a return value of 1.

Example 77 Integrity check

```
$ pg_checksums -D data --check
Checksum operation completed
Files scanned:  973
Blocks scanned: 11187
Bad checksums:  0
Data checksum version: 1
$ echo $?
0
$
```

Specify the --enable option (--disable the option to disable) to enable database cluster checksums. Even if the checksum is enabled, a consistency check is performed.

Example 78 Enable integrity check

```
$ pg_checksums -D data --enable
Checksum operation completed
Files scanned:  973
Blocks scanned: 11187
pg_checksums: syncing data directory
pg_checksums: updating control file
Checksums enabled in cluster
$
```

This command must be run with the instance successfully shut down. If an instance terminates abnormally or cannot be executed for a database cluster during the instance is up.

Example 79 Abend an instance and enable integrity

```
$ pg_ctl -D data -m immediate stop
waiting for server to shut down.... done server stopped
$ pg_checksums -D data --check
pg_checksums: cluster must be shut down
$
```

Specify the --progress option (-P option) to display the processing status.

Example 80 Display command execution status

```
$ pg_checksums -D data --check --progress
87/87 MB (100%) computed
Checksum operation completed
Files scanned:  973
Blocks scanned: 11187
Bad checksums:  0
Data checksum version: 1
```

3.4.6. pg_ctl

The following feature has been added to the pg_ctl command.

□ logrotate option

The parameter logrotate for log rotation has been added to the pg_ctl command. In the past, it was necessary to send a SIGHUP signal to the logger process. To suppress the message, specify the -s parameter at the same time.

Example 81 pg_ctl logrotate command

```
$ pg_ctl -D data logrotate
server signaled to rotate log file
$
```

3.4.7. pg_dump

The following options have been added to the pg_dump command.

□ --on-conflict-do-nothing option

This option automatically assigns the ON CONFLICT DO NOTHING clause to the output INSERT statement. Must be specified with the --inserts option or --column-inserts option.

Example 82 --on-conflict-do-nothing option

```
$ pg_dump -t data1 --inserts --on-conflict-do-nothing
--
-- PostgreSQL database dump
<<途中省略>>
-- Data for Name: data1; Type: TABLE DATA; Schema: public; Owner: demo
--

INSERT INTO public.data1 VALUES (1, 'data1') ON CONFLICT DO NOTHING;
INSERT INTO public.data1 VALUES (2, 'data1') ON CONFLICT DO NOTHING;
<<以下省略>>
```

□ --extra-float-digits option

If an integer value is specified for this parameter, the "SET extra_float_digits = specified value" statement is executed before data acquisition using the pg_dump command. The dump file does not contain SET statements. The range of values that can be specified is -15 to 3. If a non-numeric value is specified, it is regarded as 0.

□ --rows-per-insert option

This option is used with the --inserts option. Multiple tuples can be inserted in a single INSERT statement. The range of values is 1 to 2,147,483,647.

Example 83 –rows-per-insert option

```
$ pg_dump -t data1 --inserts --rows-per-insert=2
--
-- PostgreSQL database dump
<<途中省略>>
-- Data for Name: data1; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO public.data1 VALUES
       (1, 'data1'),
       (2, 'data1');
INSERT INTO public.data1 VALUES
       (3, 'data1'),
       (4, 'data1');
<<以下省略>>
```

3.4.8. pg_dumpall

The following options have been added to the pg_dumpall command.

□ --extra-float-digits option

If an integer value is specified for this parameter, the "SET extra_float_digits = specified value" statement is executed before data acquisition using the pg_dumpall command. The dump file does not contain SET statements. The range of values that can be specified is -15 to 3. If a non-numeric value is specified, it is regarded as 0.

□ --exclude-database option

In PostgreSQL 12, the --exclude-database option has been added. This option specifies which databases to exclude from backup. When specifying multiple databases, use the same pattern as for the psql command. It is also possible to specify the same option multiple times. In the following example, demodb1 and demodb2 are specified as excluded databases.

Example 84 --exclude-database option

```
$ pg_dumpall --exclude-database='demodb[12]' -f alldump.sql
```

□ Additional comments

Comments have been added to the output file for user settings (ALTER USER SET statement) and database settings.

Example 85 Additional comments

```
-- User Configurations
-- User Config {User_name}
-- Databases
-- Database {Database_name} dump
```

3.4.9. pg_rewind

The --no-sync option which does not execute sync system calls for storage has been added to the pg_rewind command.

Example 86 pg_rewind command

```
$ pg_rewind --help
pg_rewind resynchronizes a PostgreSQL cluster with another copy of the
cluster.

Usage:
  pg_rewind [OPTION]...

Options:
  -D, --target-pgdata=DIRECTORY  existing data directory to modify
      --source-pgdata=DIRECTORY  source data directory to synchronize with
      --source-server=CONNSTR    source server to synchronize with
  -n, --dry-run                  stop before modifying anything
  -N, --no-sync                  do not wait for changes to be written
                                safely to disk
  -P, --progress                 write progress messages
      --debug                   write a lot of debug messages
  -V, --version                  output version information, then exit
  -?, --help                     show this help, then exit

Report bugs to <pgsql-bugs@postgresql.org>.
```

3.4.10. pg_restore

When specifying standard output as the data output destination, specify "-f -".

3.4.11. pg_upgrade

The following options have been added to the pg_upgrade command.

□ --socketdir option

The --socketdir option (or -s option) that specifies the directory for creating a local socket has been added.

□ --clone option

The --clone option performs fast cloning using the "reflink" feature. This feature is available only on some operating systems and file systems.

Example 87 pg_upgrade command

```
$ pg_upgrade --help
pg_upgrade upgrades a PostgreSQL cluster to a different major version.

Usage:
  pg_upgrade [OPTION]...

Options:
  -b, --old-bindir=BINDIR       old cluster executable directory
…
  -r, --retain                  retain SQL and log files after success
  -s, --socketdir=DIR           socket directory to use (default CWD)
  -U, --username=NAME           cluster superuser (default "postgres")
  -v, --verbose                 enable verbose internal logging
  -V, --version                 display version information, then exit
  --clone                       clone instead of copying files to new cluster
  -?, --help                    show this help, then exit
…
```

3.4.12. psql

The following features have been added to the psql command.

□ CSV format output

Output format from psql command can be changed to CSV format. It can be changed in any of the following ways.

- Specify psql command parameter --csv
- Execute "\pset format csv" command from within the psql command

The default value of column separator is a comma (,), but it can be changed by "\pset csv_fieldsep" command. If the output value contains delimiters, the value is enclosed in double quotation marks ("). Column titles can be suppressed with "\pset tuples_only on" command.

Example 88 \pset format csv command

```
postgres=> \pset format csv
Output format is csv.
postgres=> \pset csv_fieldsep
Field separator for CSV is ",".
postgres=> SELECT * FROM data1 ;
c1,c2          <- Title
1,ABC
2,"AB,C"       <- If the data contains column separators.
2,"AB""C"      <- When the data contains double quotations.
```

□ Display of partition table

The partition table is now output explicitly when the \d command is executed.

Example 89 \d command

```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY
LIST(c1) ;
CERATE TABLE
postgres=> \d part1
              Partitioned table "public.part1"
 Column |        Type         | Collation | Nullable | Default
--------+---------------------+-----------+----------+---------
 c1     | numeric             |           |          |
 c2     | character varying(10) |           |          |
Partition key: LIST (c1)
Number of partitions: 0
```

Tablespace of partitioned indexes is now displayed.

Example 90 \d command

```
postgres=> CREATE INDEX idx1_part1 ON part1(c2) TABLESPACE ts1 ;
CREATE INDEX
postgres=> \d idx1_part1
      Partitioned index "public.idx1_part1"
 Column |        Type         | Key? | Definition
--------+---------------------+------+------------
 c2     | character varying(10) | yes  | c2
btree, for table "public.part1"
Tablespace: "ts1"
```

□ Display object privilege

Partitioned tables are now specified in the list of object privileges.

Example 91 \z command

```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10)) PARTITION BY
LIST (c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES IN (10) ;
CREATE TABLE
postgres=> \z
                                Access privileges
 Schema |  Name   |        Type          | Access privileges | Column
privileges | Policies
--------+---------+------------------+-------------------+------------
-------+----------
 public | part1   | partitioned table |                          |
|
 public | part1v1 | table               |                          |
|
(2 rows)
```

□ Display connection information

The TCP / IP address is now output when the \conninfo command is executed.

Example 92 \conninfo command

```
$ psql -p 5432 -h pghost1 -d postgres -U demo
Password for user demo: <<PASSWORD>>
psql (12devel)
Type "help" for help.

postgres=> \conninfo
You are connected to database "postgres" as user "demo" on host "pghost1"
(address "192.168.1.101") at port "5432".
postgres=>
```

□ SQLSTATE with VERBOSITY

SQLSTATE can now be specified in the "\set VERBOSITY" command.

Example 93 \set VERBOSITY command

```
postgres=> \set VERBOSITY sqlstate
postgres=> SELECT * FROM not_exists ;
psql: ERROR:  42P01
```

□ Display of partition table

The "\dP" command to display only the partition table has been added.

Example 94 \dP command

```
ostgres=> CREATE TABLE part1(c1 INT, c2 VARCHAR(2)) PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES FROM (0) TO (1000000) ;
CREATE TABLE
postgres=> \dP
            List of partitioned relations
 Schema | Name  | Owner |      Type        | On table
--------+-------+-------+------------------+----------
 public | part1 | demo  | partitioned table |
(1 row)

postgres=> \dP+
                        List of partitioned relations
 Schema | Name  | Owner |      Type        | On table | Total size | Description
--------+-------+-------+------------------+----------+------------+-------------
 public | part1 | demo  | partitioned table |          | 0 bytes    |
(1 row)
```

3.4.13. vacuumdb

The following options have been added to the vacuumdb command.

□ --disable-page-skipping option

This option is for executing a VACUUM (DISABLE_PAGE_SKIPPING) statement from a command line. This option can be specified for PostgreSQL 9.6 or later instances.

□ --skip-locked option

This option is for executing a VACUUM (SKIPP_LOCKED) statement from a command line. This option can be specified for PostgreSQL 12 or later instances.

Example 95 Execute against old version

```
$ vacuumdb -h remhost11 -d postgres -U postgres --skip-locked
vacuumdb: cannot use the "skip-locked" option on server versions older
than PostgreSQL 12
```

□ --min-mxid-age option

Only execute the vacuum or analyze commands on tables with a multixact ID age of at least specified age.

□ --min-xid-age option

Only execute the vacuum or analyze commands on tables with a transaction ID age of at least specified age.

3.4.14. vacuumlo

The vacuumlo command now has a long name option available.

Table 21 Added long-name option

Short option	Added long-named option	Description
-l	--limit	The upper limit of objects to delete
-n	--dry-run	Not actually run
-v	--verbose	The output of additional information
-h	--host	Connect host
-p	--port	Connect port number
-U	--username	Connect user name
-w	--no-password	Do not output password prompt
-W	--password	Force password input

3.5. Contrib modules

Describes new features related to the Contrib module.

3.5.1. auto_explain

□ auto_explain.log_level parameter

The parameter auto_explain.log_level (default value LOG) has been added. This parameter can specify the level of log output by the auto_explain module. Previously, it was fixed at log level LOG.

Example 96 log_level parameter

```
postgres=# LOAD 'auto_explain' ;
LOAD
postgres=# SET auto_explain.log_level = NOTICE ;
SET
postgres=# SET auto_explain.log_min_duration = 0 ;
SET
postgres=# SELECT * FROM data1 WHERE c1=1000 ;
psql: NOTICE:  duration: 0.025 ms  plan:
Query Text: SELECT * FROM data1 WHERE c1=1000 ;
Index Scan using data1_pkey on data1  (cost=0.42..8.44 rows=1 width=12)
  Index Cond: (c1 = '1000'::numeric)
  c1  |  c2
------+-------
 1000 | data1
(1 row)
```

□ Add JIT compilation information

JIT compilation information is now output to the log (backported to PostgreSQL 11.2).

Example 97 JIT compilation information log

```
2019-05-24 22:25:11.027 JST [17749] LOG:  duration: 47.828 ms  plan:
        Query Text: SELECT COUNT(*) FROM data1 ;
        Aggregate  (cost=17906.00..17906.01 rows=1 width=8)
          ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=0)
        JIT:
          Functions: 2
          Options:  Inlining  false,  Optimization  false,  Expressions  true,
Deforming true
```

3.5.2. citext

Added citext_hash_extended to calculate 64-bit hash value. The second parameter specifies the SEED.

Example 98 citext_hash_extended function

```
postgres=> INSERT INTO hash1 VALUES ('abc') ;
INSERT 0 1
postgres=> SELECT citext_hash_extended(c1, 0) FROM hash1 ;
 citext_hash_extended
----------------------
 -6747756470228489321
(1 row)

postgres=> SELECT citext_hash_extended(c1, 1) FROM hash1 ;
 citext_hash_extended
----------------------
  2125675926272891485
(1 row)
```

3.5.3. hstore

Hstore_hash_extended has been added to calculate 64-bit hash values. The second parameter specifies the SEED.

Example 99 hstore_hash_extended function

```
postgres=> INSERT INTO hstore1 VALUES (hstore('key1', 'val1')) ;
INSERT 0 1
postgres=> SELECT hstore_hash_extended(c1, 0) FROM hstore1 ;
 hstore_hash_extended
----------------------
  4863461342754690677
(1 row)

postgres=> SELECT hstore_hash_extended(c1, 1) FROM hstore1 ;
 hstore_hash_extended
----------------------
  1673650576531873159
(1 row)
```

3.5.4. pg_stat_statements

The pg_stat_statements_reset function of the pg_stat_statements module has added a parameter that limits the deletion range of statistical information. Statistical information can be deleted by specifying database ID, user ID, and query ID. If these parameters are omitted (the default value is 0), all statistics will be discarded as before.

Example 100 pg_stat_statements_reset function

```
postgres=> \dfS pg_stat_statements_reset
List of functions
-[ RECORD 1 ]-------+-------------------------------------------------------------
Schema              | public
Name                | pg_stat_statements_reset
Result data type    | void
Argument data types | userid oid DEFAULT 0, dbid oid DEFAULT 0, queryid bigint
DEFAULT 0
Type                | func
```

Also, this function cannot be executed by users who hold the pg_read_all_stats role.

3.5.5. postgres_fdw

The postgres_fdw module adds an "options" option that specifies the GUC options that can be run in a remote session. For example, settings of work_mem parameter and geqo parameter can be changed. Set values are described in the "-c parameter name = value" format. When specifying multiple parameters, specify the entire "-c parameter_name = value" separated by a space.

Example 101 options parameter

```
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS (host 'remhost1', options '-c work_mem=4MB') ;
CREATE SERVER
postgres=# ALTER SERVER remsvr1 OPTIONS
       (SET options '-c work_mem=16MB -c geqo=off') ;
ALTER SERVER
postgres=# SELECT * FROM pg_foreign_server ;
-[ RECORD 1 ]-----------------------------------------------------
srvname    | remsvr1
srvowner   | 10
srvfdw     | 16412
srvtype    |
srvversion |
srvacl     |
srvoptions | {host=remhost1, "options=-c work_mem=16MB -c geqo=off"}
```

Hewlett Packard
Enterprise

URL List

The following websites are references to create this material.

- Release Notes
  https://www.postgresql.org/docs/devel/static/release-12.html
- Commitfests
  https://commitfest.postgresql.org/
- PostgreSQL 12 Beta Manual
  https://www.postgresql.org/docs/12/index.html
- Git
  https://git.postgresql.org/gitweb/?p=postgresql.git;a=summary
- GitHub
  https://github.com/postgres/postgres
- Open source developer based in Japan (Michael Paquier)
  http://paquier.xyz/
- PostgreSQL 12 Open Items
  https://wiki.postgresql.org/wiki/PostgreSQL_12_Open_Items
- Qiita (Nuko@Yokohama)
  http://qiita.com/nuko_yokohama
- PostgreSQL Deep Dive
  http://pgsqldeepdive.blogspot.jp/ (Satoshi Nagayasu)
- pgsql-hackers Mailing list
  https://www.postgresql.org/list/pgsql-hackers/
- Announce of PostgreSQL 12 Beta 1
  https://www.postgresql.org/about/news/1943/
- Slack - postgresql-jp
  https://postgresql-jp.slack.com/

Hewlett Packard Enterprise

Change history

Change history

Version	Date	Author	Description
0.1	Apr 22, 2019	Noriyoshi Shinoda	Create internal review version. Reviewer： Tomoo Takahashi Akiko Takeshima Takahiro Kitayama (Hewlett Packard Enterprise Japan)
1.0	May 24, 2019	Noriyoshi Shinoda	Modification completed according to PostgreSQL 12 Beta 1. Reviewer: Satoshi Nagayasu (Uptime Technologies, LCC.)