May 25, 2020

PostgreSQL 13 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 13 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 12 (12.3) and PostgreSQL 13 Beta 1 (13.0). 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 12.3 (for comparison) PostgreSQL 13 (13.0) Beta 1 (May 18, 2020 20:12:53)
Operating System	Red Hat Enterprise Linux 7 Update 5 (x86-64)
Configure option	--with-llvm --with-openssl --with-perl --with-python

1.5. The 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.
<u>Underline</u>	Important output items.
<<PASSWORD>>	Replaced by password string.

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 200 new features have been added to PostgreSQL 13. Here are some major new features and benefits.

2.1. Improve analytic query performance

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

□ Enhancement of Parallel operations

If more than one of the indexes has been created in the table, now VACUUM processing is performed in parallel.

□ Deduplication of B-Tree index

Deduplication is now executed by default for B-Tree indexes. Storage capacity and I/O can be reduced.

□ Reduce WAL output

The statement that suppresses WAL output when the wal_level parameter is set to 'minimal' has been increased.

2.2. Improve reliability

PostgreSQL 13 implements the following enhancements to improve reliability.

□ Improve backup reliability

Backup consistency can now be checked. Base backups, such as the pg_basebackup command, take the size and checksum of each file and save it to the backup destination. The consistency of the obtained backup can be checked with the pg_verifybackup command.

□ Dynamic configuration changes for streaming replication

Various parameters used on the standby instance of streaming replication can now be changed dynamically. This new feature eliminates the need to restart the standby instance if the primary instance fails.

Hewlett Packard
Enterprise

2.3. Improved maintainability

The following features that can improve operability have been added.

□ Disk-based Hash Aggregation

Operations that manipulate hash tables now are performed on storage when the memory usage limit is exceeded. Previously, there was no upper limit on memory usage, which sometimes led to OOM Killer.

□ Logical replication of partitioned tables

Partitioned tables can now be used in logical replication environments.

□ Enhanced monitoring features

Catalogs which can check the execution status of the ANALYZE statement or pg_basebackup command in real-time have been added. Also, the cache hit ratio and I/O status can be checked.

□ Execution plan creation and WAL output tracking

The amount of WAL output can now be tracked when creating an execution plan or executing an SQL statement. It can be checked by pg_stat_statements module and EXPLAIN statement.

□ More wait events

Several wait events have been added that can be monitored. Wait events can be checked in the pg_stat_activity catalog.

2.4. Preparing for future new features

PostgreSQL 13 is now ready for features that will be provided in future versions.

□ Supports 64-bit transaction ID

The xid8 data type that indicates a 64-bit transaction ID and various functions that handle this data type have been added.

□ Lock contention

Page locks and extended locks can now conflict between parallel worker processes. With this implementation, it is now possible to implement parallelization such as COPY, INSERT, VACUUM, etc.

□ Enhancement of PUBLICATION

It is now ready to add non-table objects to PUBLICATION.

2.5. Incompatibility

In PostgreSQL 13, the following specifications have been changed from PostgreSQL 12.

2.5.1. configure

The --disable-float4-byval option has been removed from the 'configure' command. Also, the pkg-config command is now used to search for the libxml2 library.

2.5.2. createuser

The --adduser and --no-adduser options have been removed from the createuser command.

2.5.3. CSV log format

Backend type is added at the end of CSV format log (log_destination = 'csvlog').

Example 1 CSV log (part)

```
2020-05-21 15:01:46.351 JST,,,30744,,5ec6194a.7818,1,,2020-05-21 15:01:46 JST,,0,LOG,00000,"database system was shut down at 2020-05-21 15:01:29 JST",,,,,,,,,"","startup"

2020-05-21 15:01:46.407 JST,,,30734,,5ec61949.780e,6,,2020-05-21 15:01:45 JST,,0,LOG,00000,"database system is ready to accept connections",,,,,,,,,"","postmaster"

2020-05-21 15:02:07.142 JST,"postgres","postgres",30753,"[local]",5ec61953.7821,1,"SELECT",2020-05-21 15:01:55 JST,3/2,0,ERROR,42P01,"relation ""notexists"" does not exist",,,,,,"SELECT * FROM notexists;",15,,"psql","client backend"
```

2.5.4. Extension

Remove support for upgrading "unpackaged" extensions.

2.5.5. opaque

The opaque pseudo-data type has been removed. This data type was used for compatibility from PostgreSQL 7.3 and earlier.

2.5.6. Require version of OpenSSL

OpenSSL 1.0.0 and below are no longer supported. OpenSSL 1.0.1 or higher is required. With this fix, the minimum value of the parameter ssl_min_protocol_version has been changed from TLSv1 to TLSv1.2.

2.5.7. pg_regress

The --load-language option has been removed from the pg_regress command.

2.5.8. psql default prompt

The default prompt settings PROMPT1 and PROMPT2 of the psql command now include %x to indicate the status of the transaction.

Table 4 Changing default settings

Variable name	PostgreSQL 12	PostgreSQL 13	Note
PROMPT1	"%/%R%# "	"%/%R%x%# "	
PROMPT2	"%/%R%# "	"%/%R%x%# "	
PROMPT3	">> "	">> "	no change

Example 2 Default prompt

```
postgres=> BEGIN ;
BEGIN
postgres=*> -- In the active transaction
postgres=*> ERROR ;
ERROR: syntax error at or near "ERROR"
LINE 1: ERROR
        ^
postgres=!> -- In the failed transaction
postgres=!> ROLLBACK ;
ROLLBACK
```

2.5.9. to_date/to_timestamp

The output contents of the format string TM change according to the locale. Previously, it was ignored.

Example 3 Execute to_date function

```
postgres=> SET lc_time='ru_RU' ;
SET
postgres=> SELECT to_date('01 фев 2020', 'DD TMMON YYYY') ;
  to_date
------------
 2020-02-01
(1 row)
```

2.5.10. Promotion during recovery

Previously, if a promotion was made while recovery was paused, the paused state would continue. In PostgreSQL 13, if a promotion is performed while recovery is paused, the promotion takes precedence.

2.5.11. Partition key

Disallow partition key expressions that return pseudo-types. This specification has been backported to PostgreSQL 12.2 and later.

Example 4 Partition key with pseudo-types

```
postgres=> CREATE TABLE part1 (c1 INT, c2 INT)
       PARTITION BY RANGE(((c1, c2))) ;
ERROR:  partition key column 1 has pseudo-type record
```

2.5.12. Wait Event

Many wait events have been renamed. The event_name column in the pg_stat_activity catalog and the output string in the locktype column in the pg_locks catalog have been changed[1].

[1] See 3.1.8 Wait events

2.5.13. SIMILAR TO ESCAPE

SIMILAR TO… ESCAPE NULL clause returns NULL. In previous versions, the ESCAPE NULL clause was ignored.

Example 5 PostgreSQL 12 behavior

```
postgres=> SELECT 'ABC' SIMILAR TO 'ABC' ESCAPE NULL ;
 ?column?
----------
 t
(1 row)
```

Example 6 PostgreSQL 13 behavior

```
postgres=> \pset null null
postgres=> SELECT 'ABC' SIMILAR TO 'ABC' ESCAPE NULL ;
?column?
----------
 null
(1 row)
```

Hewlett Packard Enterprise

3. New Feature Detail

3.1. Architecture

3.1.1. Modified catalogs

The following catalogs have been changed.

Table 5 Added system catalogs

Catalog name	Description
pg_shmem_allocations	View the breakdown of shared memory.
pg_stat_progress_analyze	Tracking progress of ANALYZE statement.
pg_stat_progress_basebackup	Tracking progress of base backup.
pg_stat_slru	Tracking SLRU cache statistics.

Table 6 Dropped catalog

Catalog name	Description
pg_pltemplate	Procedural language templates.

Table 7 Dropped views in information_schema schema

Catalog name	Description
sql_languages	SQL statement standards compliance levels, options, and dialects.
sql_packages	List of standard packages.
sql_sizing_profiles	List of defined size information.

Table 8 System catalogs with columns added

Catalog name	Added column	Data type	Description
pg_available_extension_versions	trusted	boolean	The extension can be installed by non-superusers.
pg_publication	pubviaroot	boolean	Convert partition updates to route tables.
pg_replication_slots	wal_status	text	WAL file status.
	min_safe_lsn	pg_lsn	Currently effective LSN.
pg_stat_activity	leader_pid	integer	Leader PID for parallel query.
pg_stat_{all|sys|user}_tables	n_ins_since_vacuum	bigint	Number of tuples inserted since last VACUUM.
pg_stat_replication	spill_txns	bigint	Number of transactions that exceeded memory requests in logical replication.
	spill_count	bigint	Number of times that logical replication leaked to disk.
	spill_bytes	bigint	Number of bytes written to disk for decoded transactions.
pg_stat_wal_receiver	written_lsn	pg_lsn	Last WAL location already received and written to disk, but not flushed.
	flushed_lsn	pg_lsn	Last WAL location already received and flushed to disk.
pg_statistic_ext	stxstattarget	integer	SET STATISTIC value.
pg_trigger	tgparentid	oid	OID of parent trigger.

Table 9 System catalogs with columns dropped

Catalog name	Dropped column	Description
pg_stat_wal_receiver	received_lsn	Split into written_lsn and flushed_lsn.

Table 10 System catalog with changed output tuples

Catalog name	Description
pg_locks	'spectoken' was added to the value output to the 'locktype' column.
pg_stat_ssl	Process information other than client connection has been removed.
pg_stat_gssapi	Process information other than client connection has been removed.

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

□ Pg_shmem_allocations catalog

The pg_shmem_allocations catalog allows you to see the breakdown in shared memory. However, it does not include the area of dynamic shared memory. Only users with the SUPERUSER attribute can see this catalog.

Table 11 pg_shmem_allocations catalog

Column name	Data type	Description
name	text	Name of memory area, 'null' is unused area.
off	bigint	Offset from the start position.
size	bigint	Reserved bytes.
allocated_size	bigint	Reserved bytes including padding.

Example 7 Refer pg_shmem_allocations catalog

```
postgres=# SELECT * FROM pg_shmem_allocations ;
              name                |    off    |   size    | allocated_size
-------------------------------------+------------+------------+-------------
 Buffer IO Locks                   | 140660096 |   524288 |         524288
 Buffer Descriptors                |   5393792 |  1048576 |        1048576
 Backend SSL Status Buffer         | 146585088 |     42312 |          42368
 Async Queue Control               | 147119360 |      2492 |           2560
 Wal Sender Ctl                    | 147112832 |      1280 |           1280
 AutoVacuum Data                   | 147104384 |      5368 |           5376
 PROCLOCK hash                     | 143136000 |      2904 |           2944
 FinishedSerializableTransactions  | 146097664 |        16 |            128
 XLOG Ctl                          |     53504 |  4208272 |        4208384
 Shared MultiXact State            |   5392640 |      1028 |           1152
…
```

□ Pg_stat_progress_analyze catalog

The pg_stat_progress_analyze catalog allows you to check the execution status of ANALYZE statement. This catalog can be viewed by ordinary users, but ordinary users cannot see the status of other users' command execution.

Table 12 pg_stat_progress_analyze catalog

Column name	Data type	Description
pid	integer	Backend process ID.
datid	oid	OID of the connection database.
datname	name	Connection database name.
relid	oid	Table OID executing ANALYZE statement.
phase	text	Execution phase.
sample_blks_total	bigint	Total number of blocks sampled.
sample_blks_scanned	bigint	Number of sampled blocks.
ext_stats_total	bigint	Extended statistics.
ext_stats_computed	bigint	Number of extended statistics computed.
child_tables_total	bigint	Number of child tables.
child_tables_done	bigint	Number of child tables for ANALYZE done.
current_child_table_relid	oid	ANALYZE executing child table OID.

Example 8 Refer pg_stat_progress_analyze catalog

```
postgres=# SELECT * FROM pg_stat_progress_analyze ;
-[ RECORD 1 ]-------------+---------------------
pid                       | 30932
datid                     | 13578
datname                   | postgres
relid                     | 16388
phase                     | computing statistics
sample_blks_total         | 54055
sample_blks_scanned       | 54055
ext_stats_total           | 0
ext_stats_computed        | 0
child_tables_total        | 0
child_tables_done         | 0
current_child_table_relid | 0
```

□ Pg_stat_progress_basebackup catalog

The pg_stat_progress_basebackup catalog allows you to check the status of the backups performed by tools such as the pg_basebackup command. This catalog can be viewed by non-superuser users, but only the 'pid' column can be viewed by non-connected users.

Table 13 pg_stat_progress_basebackup catalog

Column name	Data type	Description
pid	integer	WAL sender process ID.
phase	text	The string indicating the execution phase.
backup_total	bigint	Total backup size.
backup_streamed	bigint	Streaming data volume.
tablespaces_total	bigint	Number of total tablespaces.
tablespaces_streamed	bigint	Number of streaming tablespaces.

Example 9 Refer pg_stat_progress_basebackup catalog

```
postgres=# SELECT * FROM pg_stat_progress_basebackup ;
-[ RECORD 1 ]--------+-------------------------
pid                  | 31080
phase                | streaming database files
backup_total         | 467913216
backup_streamed      | 302026752
tablespaces_total    | 1
tablespaces_streamed | 0
```

□ Pg_stat_slru catalog

Pg_stat_slru catalog can check the usage status of SLRU cache. Displays statistics on accesses to cached pages for each SLRU cache area.

Table 14 pg_stat_slru catalog

Column name	Data type	Description
name	text	SLRU name.
blks_zeroed	bigint	Number of blocks initialized to zero.
blks_hit	bigint	Number of blocks that hit the cache.
blks_read	bigint	Number of blocks read.
blks_written	bigint	Number of blocks written.
blks_exists	bigint	Number of blocks checked for existence in cache.
flushes	bigint	Number of dirty blocks flushed.
truncates	bigint	Number of blocks truncated.
stats_reset	timestamp with time zone	Date and time when statistics were reset.

The counter values in this view will persist across instance reboots. Execute pg_stat_reset_slru function[2] to reset the counter value.

Example 10 Refer pg_stat_slru catalog

```
postgres=# SELECT name, blks_hit, blks_read, blks_written
        FROM pg_stat_slru ;
     name        | blks_hit | blks_read | blks_written
-----------------+----------+-----------+--------------
 CommitTs        |        0 |         0 |            0
 MultiXactMember |        0 |         0 |            0
 MultiXactOffset |        0 |         0 |            0
 Notify          |        0 |         0 |            0
 Serial          |        0 |         0 |            0
 Subtrans        |        0 |         0 |            0
 Xact            |       14 |         0 |            0
 other           |        0 |         0 |            0
(8 rows)
```

3.1.2. Data types

The following data types have been added:

□ Regcollation

The regcollation type that indicates the Collation name has been added. A to_regcollation function has been added to convert from text type to regcollation type.

Example 11 Execute to_regcollation function

```
postgres=> SELECT to_regcollation('"POSIX"') ;
 to_regcollation
-----------------
 "POSIX"
(1 row)
```

[2] See 3.2.19 Functions

□ Xid8

The xid8 type that indicates a 64-bit transaction ID has been added. Functions that use the xid8 type have been added. The functions that return the conventional 32-bit transaction ID are kept for compatibility. However, they may be removed in the future.

Table 15 Functions that use 64-bit transaction IDs

Legacy function name	New function name
txid_current	pg_current_xact_id
txid_current_if_assigned	pg_current_xact_id_if_assigned
txid_current_snapshot	pg_current_snapshot
txid_snapshot_xip	pg_snapshot_xip
txid_snapshot_xmax	pg_snapshot_xmax
txid_snapshot_xmin	pg_snapshot_xmin
txid_visible_in_snapshot	pg_visible_in_snapshot
txid_status	pg_xact_status

□ Pg_snapshot

The pg_snapshot type stores information about the transaction ID visibility (xmin, xmax, xip_list) at a specific point in time.

□ Other abstract data types

In addition, the following abstract data types have been added.

Table 16 Added abstract data types

Data type name	Description
anycompatible	The function accepts any data type and is automatically promoted.
anycompatiblearray	The function accepts any array type and is automatically promoted.
anycompatiblenonarray	The function accepts any non-array type and is automatically promoted.
anycompatiblerange	The function accepts any range type and is automatically promoted.
table_am_handler	Table access method handler.

3.1.3. Disk based hash aggregation

A hash table may be created in memory when executing a GROUP BY or DISTINCT clause. PostgreSQL 13 performs storage-based hash aggregation if the hash table cannot be stored in work memory. This feature can be controlled by the parameter enable_hashagg_disk (default 'on') and the parameter enable_groupingsets_hash_disk (default 'off') used in the GROUPING SETS clause. In previous versions, work memory was created on the assumption that it could be stored in memory, which could lead to unexpected increases in memory usage.

Example 12 Disk-based hash aggregation

```
postgres=> EXPLAIN ANALYZE SELECT COUNT(*) FROM data1 GROUP BY c1 ;
                                                     QUERY PLAN
-----------------------------------------------------------------
 HashAggregate  (cost=516555.00..694680.00 rows=10000000 width=14)
(actual time=2893.873..6379.011 rows=10000000 loops=1)
   Group Key: c1
   Planned Partitions: 128
   Peak Memory Usage: 4153 kB
   Disk Usage: 322544 kB
   HashAgg Batches: 640
   ->  Seq Scan on data1  (cost=0.00..154055.00 rows=10000000
width=6) (actual time=0.170..628.013 rows=10000000 loops=1)
 Planning Time: 0.059 ms
 Execution Time: 6693.200 ms
(9 rows)
```

3.1.4. Incremental sort

Incremental sorting is a way to reduce the resources used for multi-column sorting. It is used when another column needs to be sorted in addition to the already sorted column. This feature can be controlled by the enable_incrementalsort parameter. The default value is 'on'.

Example 13 Incremental sort

```
postgres=> EXPLAIN ANALYZE SELECT * FROM data1 ORDER BY c1, c2 ;
                                                            QUERY PLAN
-----------------------------------------------------------------
 Incremental Sort  (cost=0.48..763746.44 rows=10000000 width=12)
(actual time=0.116..2447.842 rows=10000000 loops=1)
   Sort Key: c1, c2
   Presorted Key: c1
   Full-sort Groups: 312500   Sort Method: quicksort   Average
Memory: 26kB  Peak Memory: 26kB
   ->  Index Scan using idx1_data1 on data1  (cost=0.43..313746.43
rows=10000000 width=12) (actual time=0.020..1216.827 rows=10000000
loops=1)
 Planning Time: 1.029 ms
 Execution Time: 2674.266 ms
(7 rows)
```

3.1.5. Backup manifests

Consistency checks can now be executed on base backups created with the pg_basebackup command, etc. The size and checksum are calculated for each file in the database cluster.

□ Checksum

Checksum calculation method can be selected from SHA224, SHA256, SHA384, SHA512, and CRC32C. CRC32C is the default value.

□ Manifest file

The backup manifest is stored as backup_manifest in the backup directory. It is a file in JSON format.

Example 14 backup_manifest file

```
$ pg_basebackup -D back
$ cat back/backup_manifest
{ "PostgreSQL-Backup-Manifest-Version": 1,
"Files": [
{ "Path": "backup_label", "Size": 226, "Last-Modified": "2020-05-
21  06:20:15  GMT",  "Checksum-Algorithm":  "CRC32C",  "Checksum":
"0e45028a" },
{ "Path": "global/1262", "Size": 8192, "Last-Modified": "2020-05-
21  05:50:10  GMT",  "Checksum-Algorithm":  "CRC32C",  "Checksum":
"6a2131f8" },
…
"WAL-Ranges": [
{   "Timeline":   1,   "Start-LSN":   "0/2D000028",   "End-LSN":
"0/2D0001E8" }
],
"Manifest-Checksum":
"8832de70e00fd7612125364088a945ff5cacca3befb4004eea255e2b498a0ca6
"}
```

If the backup is performed in tar format, the manifest file is not included in the tar file.

Example 15 Manifest file for backup in tar file format

```
$ pg_basebackup --format=tar -D back
$ ls back
backup_manifest  base.tar  pg_wal.tar
$
```

□ Check consistency

The pg_verifybackup command is provided to recheck the integrity of the obtained backup. Specify the backup destination directory name to the pg_verifybackup command[3].

[3] See 3.4.6 pg_verifybackup

Example 16 pg_verifybackup command

```
$ pg_verifybackup back
backup successfully verified
```

3.1.6. Partitioned table

The following features have been added to the partitioned table.

□ Logical replication support

The partitioned tables can now be added to PUBLICATION in a logical replication environment. In the previous version, the CREATE PUBLICATION statement in the example below failed. In order for replication to succeed, a table with the same structure is required on the SUBSCRIPTION side.

Example 17 Add partitioned table to PUBLICATION

```
postgres=> CREATE TABLE part1(c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10))
        PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES
        FROM (0) TO (100000) ;
CREATE TABLE
postgres=> CREATE TABLE part1v2 PARTITION OF part1 FOR VALUES
        FROM (100000) TO (200000) ;
CREATE TABLE
postgres=> CREATE PUBLICATION pub1 FOR TABLE part1 ;
CREATE PUBLICATION
postgres=> \dRp+ pub1
                     Publication pub1
 Owner | All tables | Inserts | Updates | Deletes | Truncates | Via root
-------+------------+---------+---------+---------+-----------+----------
 demo  | f          | t       | t       | t       | t         | t
Tables:
   "public.part1"
```

By default, PUBLICATION converts updates to partitions into updates to the partitioned table and sends them to SUBSCRIPTION. This allows the SUBSCRIPTION side to perform replication even on non-partitioned tables. This behavior can be changed by setting the PUBLICATION attribute publish_via_partition_root to 'off'.

Example 18 Non-partitioned table on SUBSCRIPTION side

```
postgres=> ALTER PUBLICATION pub1 SET (publish_via_partition_root = on) ;
ALTER PUBLICATION
postgres=> \dRp+ pub1
                        Publication pub1
 Owner | All tables | Inserts | Updates | Deletes | Truncates | Via root
-------+------------+---------+---------+---------+-----------+----------
 demo  | f          | t       | t       | t       | t         | t
Tables:
    "public.part1"

# SUBSCRIPTION side
postgres=> CREATE TABLE part1(c1 NUMERIC PRIMARY KEY, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=# CREATE SUBSCRIPTION sub1 CONNECTION 'host=remhost1 dbname=postgres
user=postgres password=<<PASSWORD>>' PUBLICATION pub1 ;
NOTICE:  created replication slot "sub1" on publisher
CREATE SUBSCRIPTION
postgres=#
```

□ BEFORE INSERT trigger

ROW level BEFORE INSERT triggers can now be set. However, tuple is not possible to change the partition to be stored.

Example 19 CREATE TRIGGER statement for a partitioned table

```
postgres=> CREATE TABLE part1 (c1 INT, c2 INT, c3 VARCHAR(10))
                PARTITION BY LIST(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES IN (10) ;
CREATE TABLE
postgres=> CREATE OR REPLACE FUNCTION fnc1_part1() RETURNS TRIGGER
             LANGUAGE plpgsql AS $$
             BEGIN
                NEW.c3 = 'TRIGGER';
                RETURN NEW;
             END; $$ ;
CREATE FUNCTION
postgres=> CREATE TRIGGER trg1_part1 BEFORE INSERT ON part1
                FOR EACH ROW EXECUTE FUNCTION fnc1_part1() ;
CREATE TRIGGER
postgres=> INSERT INTO part1(c1, c2) VALUES (10, 20) ;
INSERT 0 1
postgres=> SELECT * FROM part1 ;
 c1 | c2 |   c3
----+----+---------
 10 | 20 | TRIGGER
(1 row)
```

□ Allow partition wise join in more case

Partition-wise join can now be performed even if the partition structures are not exactly the same. This feature does not work with hash partitioned tables.

□ Partitioned column using the entire table

It is now possible to specify an entire column to the definition of the partitioning columns.

Example 20 Partitiond column expression

```
postgres=> CREATE TABLE part2(c1 INT, c2 INT) PARTITION
       BY LIST((part2)) ;
CREATE TABLE
postgres=> CREATE TABLE part2v1 PARTITION OF part2 FOR VALUES
       IN ('(1, 2)') ;
CREATE TABLE
postgres=> CREATE TABLE part2v2 PARTITION OF part2 FOR VALUES
       IN ('(2, 4)') ;
CREATE TABLE
```

3.1.7. Log output for Autovacuum

WAL statistics are now output to the automatic VACUUM log.

Example 21 Log output for Autovacuum

```
2020-05-21 15:59:21.872 JST [31752] LOG:  automatic vacuum of table
"postgres.public.data1": index scans: 1
       pages: 0 removed, 54055 remain, 0 skipped due to pins, 0
skipped frozen
       tuples: 10000000 removed, 0 remain, 0 are dead but not yet
removable, oldest xmin: 514
       buffer usage: 234548 hits, 119331 misses, 130854 dirtied
       avg read rate: 2.780 MB/s, avg write rate: 3.048 MB/s
       system usage: CPU: user: 3.24 s, system: 4.13 s, elapsed:
335.39 s
       WAL usage: 244232 records, 93518 full page images, 303298890
bytes
```

3.1.8. Wait events

Wait events that are output to the wait_event column of pg_stat_activity catalog had the following changes.

Table 17 Added wait events

Wait event name	Description
BackupWaitWalArchive	Waiting for archive creation.
RecoveryConflictSnapshot	Waiting for recovery conflict resolution during VACUUM cleanup.
RecoveryConflictTablespace	Waiting for resolution of recovery conflict when deleting table space.
RecoveryPause	Waiting for promotion of standby instance.
VacuumDelay	Cost-based VACUUM delay.
ProcSignalBarrier	Waiting for a barrier event to be processed by all backends.

Table 18 Wait event that has been renamed

Wait event name (PostgreSQL 12)	Wait event name (PostgreSQL 13)
AsyncCtlLock	NotifySLRU
AsyncQueueLock	NotifyQueue
CLogControlLock	XactSLRU
ClogGroupUpdate	XactGroupUpdate
CommitTsControlLock	CommitTsSLRU
Hash/Batch/Allocating	HashBatchAllocate
Hash/Batch/Electing	HashBatchElect
Hash/Batch/Loading	HashBatchLoad
Hash/Build/Allocating	HashBuildAllocate
Hash/Build/Electing	HashBuildElect
Hash/Build/HashingInner	HashBuildHashInner
Hash/Build/HashingOuter	HashBuildHashOuter
Hash/GrowBatches/Allocating	HashGrowBatchesAllocate
Hash/GrowBatches/Deciding	HashGrowBatchesDecide
Hash/GrowBatches/Electing	HashGrowBatchesElect
Hash/GrowBatches/Finishing	HashGrowBatchesFinish
Hash/GrowBatches/Repartitioning	HashGrowBatchesRepartition
Hash/GrowBuckets/Allocating	HashGrowBucketsAllocate
Hash/GrowBuckets/Electing	HashGrowBucketsElect
Hash/GrowBuckets/Reinserting	HashGrowBucketsReinsert
MultiXactOffsetControlLock	MultiXactOffsetSLRU
MultiXactMemberControlLock	MultiXactMemberSLRU
OldSerXidLock	SerialSLRU

Wait event name (PostgreSQL 12)	Wait event name (PostgreSQL 13)
RecoveryWalAll	RecoveryWalStream
RecoveryWalStream	RecoveryRetrieveRetryInterval
SerializablePredicateLockListLock	SerializablePredicateList
SubtransControlLock	SubtransSLRU
speculative token	spectoken

There are many event name changes other than the above table, such as deleting 'Lock' from the end of the event name.

3.1.9. libpq connection string

The following libpq connection strings have been added / changed.

□ Channel_binding parameter

The channel_binding parameter has been added to the client connection string to control channel binding. The values that can be specified are as follows.

Table 19 channel_binding parameter values

Value	Description
require	Required.
prefer	Client selection.
disable	Disable.

'Prefer' is the default value when SSL is enabled. 'Disable' is the default value if SSL is disabled. The environment variable PGCHANNELBINDING can specify the same value as this parameter.

□ Sslkey parameter

ASN.1 DER format files can now be specified for the sslkey parameter.

□ Ssl_min_protocol_version, ssl_max_protocol_version parameter

Specify the minimum version (ssl_min_protocol_version) and maximum version (ssl_max_protocol_version) of the SSL / TLS protocol. Possible values are TLSv1, TLSv1.1, TLSv1.2, and TLSv1.3. If not specified, the backend specification will be used. It is also possible to use the environment variables PGSSLMINPROTOCOLVERSION and PGSSLMAXPROTOCOLVERSION instead of using parameters.

□ Sslpassword parameter

Specify the password of the private key specified by the sslkey parameter.

3.1.10. libpq functions

The following libpq functions have been added.

- BufferUsageAccumDiff
- TupleHashTableHash
- LookupTupleHashEntryHash
- PQsetSSLKeyPassHook_OpenSSL
- LogicalTapeSetExtend

3.1.11. Hook

□ TLS initialization hook

The TLS initialization hook was provided. Specify the callback function in openssl_tls_init_hook.

□ TRUNCATE hook

The hook is executed to allow mandatory access control (MAC) for the TRUNCATE statement. Contrib module sepgsql can control access to TRUNCATE statement.

3.1.12. Column trigger

In a logical replication environment, column triggers are now executed on the subscription side.

3.1.13. Local connection key

Conventionally, the key information of the shared memory used by the instance was determined based on the connection waiting port number (parameter 'port'). PostgreSQL 13 now uses the i-node number of the database cluster.

Example 22 Shared memory key value

```
$ ls -lid data
33832487 drwx------. 20 postgres postgres 4096 Jan 16 00:17 data
$ ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x02043e27 196610     postgres   600        56         6
```

In the above example, the i-node number is 33,832,487, which is 0x02043e27 when converted to hexadecimal.

3.1.14. Trusted Extension

The 'trusted' attribute described in the extension control file ({extension}.control) file has been added. The default value for this attribute is 'off'. Extensions with this attribute value set to 'on' can execute the CREATE EXTENSION statement even if the user does not have the SUPERUSER attribute. User must have CREATE privilege on the database.

Example 23 plpgsql.control file contents

```
$ cat plpgsql.control
# plpgsql extension
comment = 'PL/pgSQL procedural language'
default_version = '1.0'
module_pathname = '$libdir/plpgsql'
relocatable = false
schema = pg_catalog
superuser = true
trusted = true
```

Example 24 CREATE EXTENSION statement executed by a general user

```
postgres=# CREATE USER demo PASSWORD '<<PASSWORD>>' ;
CREATE ROLE
postgres=# GRANT CREATE ON DATABASE postgres TO demo ;
GRANT
postgres=# \connect postgres demo
You are now connected to database "postgres" as user "demo".
postgres=> CREATE EXTENSION hstore ;
CREATE EXTENSION
```

The following extensions have the trusted setting set to 'on'.

- bool_plperl
- btree_gin
- btree_gist
- citext
- cube
- dict_int
- earthdistance
- fuzzystrmatch
- hstore
- hstore_plperl
- intarray.
- isn
- jsonb_plperl
- lo
- ltree
- pgcrypto
- pg_trgm
- plperl
- plpgsql
- seg
- tablefunc
- tcn
- tsm_system_rows
- tsm_system_time
- unaccent

3.1.15. Replication slot

If the parameter primary_slot_name is not specified on the standby instance in a streaming replication environment, a temporary replication slot can be created on the primary instance. The standby instance parameter wal_receiver_create_temp_slot must be 'on' (default 'off').

Example 25 Information on automatically created replication slots

```
postgres=# SELECT * FROM pg_replication_slots ;
-[ RECORD 1 ]-------+--------------------
slot_name           | pg_walreceiver_4273
plugin              |
slot_type           | physical
datoid              |
database            |
temporary           | t
active              | t
active_pid          | 4273
xmin                |
catalog_xmin        |
restart_lsn         | 0/390005A0
confirmed_flush_lsn |
wal_status          | normal
min_safe_lsn        |
```

3.1.16. Text search

Languages for text search have been increased. In PostgreSQL 12, there were 22 languages, but in PostgreSQL 13, Greek has been added.

Example 26 Text search languages

```
postgres=> \dF
             List of text search configurations
   Schema   |    Name    |              Description
------------+------------+---------------------------------------
 pg_catalog | arabic     | configuration for arabic language
 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 | greek      | configuration for greek language << new
 pg_catalog | hungarian  | configuration for hungarian language
…
```

3.2. SQL statement

This section explains new features related to SQL statements.

3.2.1. ALTER NO DEPENDS ON

ALTER statements for functions, indexes, materialized views, and triggers can now remove dependency settings for extensions.

Syntax

```
ALTER object_name NO DEPENDS ON EXTENSION extension_name
```

Example 27 ALTER NO DEPENDS ON statement

```
postgres=> ALTER FUNCTION func1 DEPENDS ON EXTENSION cube ;
ALTER FUNCTION
postgres=> ALTER FUNCTION func1 NO DEPENDS ON EXTENSION cube ;
ALTER FUNCTION
```

3.2.2. ALTER STATISTICS SET STATISTICS

Specify the statistics collection target when executing ANALYZE. The value that can be set is from 0 to 10,000. If you specify 10,000 or more, it is assumed to be 10,000. Specify -1 to reset to the default value.

Syntax

```
ALTER STATISTICS statistic_name SET STATISTICS new_target
```

The specified value can be checked in the stxstattarget column of the pg_statistic_ext catalog.

Example 28 ALTER STATISTICS SET STATISTICS statement

```
postgres=> CREATE TABLE data1(c1 INT, c2 INT, c3 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE STATISTICS stat1 ON c1, c2 FROM data1 ;
CREATE STATISTICS
postgres=> ALTER STATISTICS stat1 SET STATISTICS 10000 ;
ALTER STATISTICS
postgres=> SELECT stxname, stxstattarget FROM pg_statistic_ext
       WHERE stxname='stat1' ;
 stxname | stxstattarget
---------+---------------
 stat1   |         10000
(1 row)
```

3.2.3. ALTER TABLE

The following enhancements have been added to the ALTER TABLE statement.

□ ALTER TABLE ALTER COLUMN DROP EXPRESSION

By executing the ALTER TABLE ALTER COLUMN DROP EXPRESSION statement, it is now possible to drop an auto-created definition from a column specified with the GENERATED ALWAYS clause. Simply drop the calculation information and the column values will remain. IF EXISTS clause can be specified.

Syntax

```
ALTER TABLE table_name ALTER COLUMN column_name DROP EXPRESSION
[IF EXISTS]
```

Example 29 ALTER TABLE DROP EXPRESSION statement

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

postgres=> INSERT INTO gen1(c1, c2) VALUES (100, 200) ;
INSERT 0 1
postgres=> ALTER TABLE gen1 ALTER COLUMN c3
       DROP EXPRESSION IF EXISTS ;
ALTER TABLE
postgres=> \d gen1
              Table "public.gen1"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 c1     | integer |           |          |
 c2     | integer |           |          |
 c3     | integer |           |          |

postgres=> SELECT * FROM gen1 ;
 c1  | c2  | c3
-----+-----+-----
 100 | 200 | 300
(1 row)
```

□ ALTER TABLE ALTER COLUMN SET STORAGE

Alters to a column's STORAGE attribute are now propagated to the index.

Example 30 ALTER TYPE SET statement

```
postgres=> CREATE TABLE attr1(c1 CHAR(10)) ;
CREATE TABLE
postgres=> \d+ attr1
                              Table "public.attr1"
 Column |     Type      | Collation | Nullable | Default | Storage  | …
--------+---------------+-----------+----------+---------+----------+ …
 c1     | character(10) |           |          |         | extended | …
Access method: heap

postgres=> CREATE INDEX idx1_attr1 ON attr1(c1) ;
CREATE INDEX
postgres=> ALTER TABLE attr1 ALTER COLUMN c1 SET STORAGE EXTERNAL ;
ALTER TABLE
postgres=> \d+ idx1_attr1
                  Index "public.idx1_attr1"
 Column |     Type      | Key? | Definition | Storage  | Stats target
--------+---------------+------+------------+----------+--------------
 c1     | character(10) | yes  | c1         | external |
btree, for table "public.attr1"
```

3.2.4. ALTER TYPE

In the ALTER TYPE statement, attributes can be changed using the SET clause.

Syntax

```
ALTER TYPE type_name SET (attribute = value)
```

The attributes that can be changed are as follows. Part of the attributes that can be specified in the CREATE TYPE statement.

Table 20 Attributes that can be changed

Attribute name	Description
RECEIVE	Function that converts external representation into an internal representation.
SEND	Function to convert from internal representation to binary.
TYPMOD_IN	Modifier support function.
TYPMOD_OUT	Modifier support function.
ANALYZE	Statistics collection function.
STORAGE	Function that determines how to store variable-length data.

Example 31 ALTER TYPE SET statement

```
postgres=# ALTER TYPE box SET (SEND = myboxsend) ;
ALTER TYPE
```

3.2.5. ALTER VIEW

Column names in views can now be changed. Execute an ALTER VIEW RENAME COLUMN statement.

Syntax

```
ALTER VIEW view_name RENAME COLUMN old_name TO new_name
```

Example 32 Rename view column name

```
postgres=> CREATE VIEW customer_name AS SELECT c_customer_id,
        c_first_name || ',' || c_last_name c_full_name FROM customer ;
CREATE VIEW
postgres=> \d customer_name
               View "public.customer_name"
   Column     |     Type      | Collation | Nullable | Default
---------------+---------------+-----------+----------+---------
 c_customer_id | character(16) |           |          |
 c_full_name   | text          |           |          |

postgres=> ALTER VIEW customer_name RENAME COLUMN c_full_name TO full_name ;
ALTER VIEW
postgres=> \d customer_name
               View "public.customer_name"
   Column     |     Type      | Collation | Nullable | Default
---------------+---------------+-----------+----------+---------
 c_customer_id | character(16) |           |          |
 full_name     | text          |           |          |
```

3.2.6. CREATE DATABASE

The LOCALE clause can now be specified as an option in the CREATE DATABASE statement. It cannot be specified at the same time as LC_CTYPE clause or LC_COLLATE clause.

Syntax

```
CREATE DATABASE database_name [[WITH] LOCALE [=] 'locale_name']
```

Example 33 Specify LOCALE clause

```
postgres=# CREATE DATABASE localedb1 WITH LOCALE='ja_JP.eucjp' ENCODING='eucjp'
        TEMPLATE=template0 ;
CREATE DATABASE
postgres=# \l localedb1
                             List of databases
  Name    |  Owner   | Encoding |  Collate    |   Ctype     | Access privileges
-----------+----------+----------+-------------+-------------+-----------------
 localedb1 | postgres | EUC_JP   | ja_JP.eucjp | ja_JP.eucjp |
(1 row)
```

3.2.7. CREATE INDEX

In-page deduplication has been added to the B-Tree index. This feature is enabled by default. To disable this feature, set the deduplicate_items attribute to 'off'.

Example 34 B-Tree index deduplication

```
postgres=> CREATE TABLE data1(c1 INT, c2 VARCHAR(10), c3 VARCHAR(10)) ;
CREATE TABLE
postgres=> INSERT INTO data1 VALUES (generate_series(1, 1000000),
        'data1', 'data1') ;
INSERT 0 1000000
postgres=> CREATE INDEX idx1_dup ON data1(c2)
        WITH (deduplicate_items = off) ;
CREATE INDEX
postgres=> CREATE INDEX idx2_dedup ON data1(c3)
        WITH (deduplicate_items = on) ;
CREATE INDEX
postgres=> \d+ idx1_dup
                      Index "public.idx1_dup"
 Column |         Type          | Key? | Definition | Storage  | Stats target
--------+-----------------------+------+------------+----------+--
 c2     | character varying(10) | yes  | c2         | extended |
btree, for table "public.data1"
Options: deduplicate_items=off
```

Example 35 Effect of index deduplication

```
postgres=> SELECT pg_size_pretty(pg_relation_size('idx1_dup')) ;
 pg_size_pretty
----------------
 21 MB
(1 row)
postgres=> SELECT pg_size_pretty(pg_relation_size('idx2_dedup')) ;
 pg_size_pretty
----------------
 6792 kB
(1 row)
```

3.2.8. CREATE TABLE

Some attributes[4] can be specified in CREATE TABLE/ALTER TABLE statement.

Example 36 Specifying additional attributes for tables

```
postgres=> CREATE TABLE data1(c1 NUMERIC, c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> ALTER TABLE data1 SET (toast.vacuum_index_cleanup = OFF) ;
ALTER TABLE
postgres=> ALTER TABLE data1 SET (autovacuum_vacuum_insert_threshold = 10000) ;
ALTER TABLE
postgres=> ALTER TABLE data1 SET (autovacuum_vacuum_insert_scale_factor =
0.1) ;
ALTER TABLE
```

3.2.9. CREATE TABLESPACE

Maintenance_io_concurrency[5] can now be specified for tablespace attributes.

[4] See 3.3.1 Added parameters

[5] See 3.3.1 Added parameters

Example 37 Specify maintenance_io_concurrency attribute

```
postgres=# CREATE TABLESPACE ts1 LOCATION '/home/postgres/ts1'
               WITH (maintenance_io_concurrency = 20) ;
CREATE TABLESPACE
postgres=# ALTER TABLESPACE ts1 SET (maintenance_io_concurrency = 30) ;
ALTER TABLESPACE
postgres=# \db+ ts1
                                          List of tablespaces
 Name |  Owner   |      Location      | Access privileges |            Options
|  Size   | Description
------+----------+--------------------+-------------------+--------------
--------------------+---------+-------------
 ts1    | postgres  | /home/postgres/ts1  |                                  |
{maintenance_io_concurrency=30} | 0 bytes |
(1 row)
```

3.2.10. DROP DATABASE FORCE

The database can now be forcibly dropped even if there are connected session exists. The connected session is forcibly closed. The database to which the session issuing the DROP DATABASE statement is connected cannot be dropped.

Syntax

```
DROP DATABASE database_name [[WITH] (FORCE)]
```

Example 38 DROP DATABASE statement

```
postgres=# DROP DATABASE demodb WITH (FORCE) ;
DROP DATABASE
postgres=# DROP DATABASE postgres WITH (FORCE) ;
ERROR:  cannot drop the currently open database
```

3.2.11. EXPLAIN ANALYZE

The following extensions have been implemented in the EXPLAIN ANALYZE statement.

□ Output of cache information

By specifying BUFFERS, the status of the shared buffer when creating an execution plan is now output.

Example 39 EXPLAIN (ANALYZE, BUFFERS) statement

```
postgres=> EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM data1
       WHERE c1 = 10000 ;
                                               QUERY PLAN
-------------------------------------------------------------------
 Index Scan using idx1_data1 on data1  (cost=0.43..8.45 rows=1
width=12) (actual time=15.359..15.362 rows=1 loops=1)
   Index Cond: (c1 = '10000'::numeric)
   Buffers: shared read=4 dirtied=1
 Planning Time: 0.498 ms
   Buffers: shared hit=70 read=1 dirtied=2
 Execution Time: 15.412 ms
(6 rows)
```

□ Output sort information

The output of the EXPLAIN ANALYZE statement when executing a parallel query has changed. JIT function information is now output for each worker. Also, the output of sort information has changed.

Example 40 Output of the EXPLAIN ANALYZE statement (JIT)

```
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT * FROM data1 WHERE c1 < 10000 ;
                                                            QUERY PLAN
-----------------------------------------------------------------------------
 Gather          (cost=0.00..107056.40     rows=9188     width=12)     (actual
time=12.326..410.535 rows=9999 loops=1)
   Output: c1, c2
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel  Seq  Scan  on  public.data1   (cost=0.00..106137.60  rows=3828
width=12) (actual time=228.802..360.446 rows=3333 loops=3)
         Output: c1, c2
         Filter: (data1.c1 < '10000'::numeric)
         Rows Removed by Filter: 3330000
         Worker 0:  actual time=330.989..330.989 rows=0 loops=1
           JIT:
             Functions: 2
             Options:  Inlining  false,  Optimization  false,  Expressions  true,
Deforming true
             Timing: Generation 2.370 ms, Inlining 0.000 ms, Optimization 1.562
ms, Emission 25.957 ms, Total 29.889 ms
         Worker 1:  actual time=343.317..343.317 rows=0 loops=1
           JIT:
             Functions: 2
             Options:  Inlining  false,  Optimization  false,  Expressions  true,
Deforming true
             Timing: Generation 1.655 ms, Inlining 0.000 ms, Optimization 1.265
ms, Emission 18.974 ms, Total 21.894 ms
…
```

Example 41 Output of the EXPLAIN ANALYZE statement (Sort)

```
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT * FROM data1 ORDER BY c1 ;
                                                       QUERY PLAN
----------------------------------------------------------------------------
 Gather  Merge   (cost=752323.83..1726047.95  rows=8345624  width=12)  (actual
time=1471.139..4003.430 rows=10000000 loops=1)
   Output: c1, c2
   Workers Planned: 2
   Workers Launched: 2
   ->    Sort    (cost=751323.81..761755.84  rows=4172812  width=12)  (actual
time=1328.103..1849.320 rows=3333333 loops=3)
        Output: c1, c2
        Sort Key: data1.c1
        Sort Method: external merge  Disk: 74864kB
        Worker 0:  actual time=1466.829..1989.939 rows=3335746 loops=1
          Sort Method: external merge  Disk: 75096kB
        Worker 1:  actual time=1360.627..1885.937 rows=3338879 loops=1
          Sort Method: external merge  Disk: 75168kB
        ->    Parallel   Seq   Scan   on   public.data1    (cost=0.00..149837.12
rows=4172812 width=12) (actual time=60.222..279.624 rows=3333333 loops=3)
             Output: c1, c2
             Worker 0:  actual time=58.974..280.404 rows=3335746 loops=1
             Worker 1:  actual time=58.948..277.968 rows=3338879 loops=1
 Planning Time: 0.056 ms
 Execution Time: 4261.051 ms
(18 rows)
```

□ Output of WAL information

If the WAL option is specified in the EXPLAIN ANALYZE statement, the generated WAL information will be output.

Example 42 EXPLAIN (ANALYZE, WAL) statement

```
postgres=> EXPLAIN (ANALYZE, WAL) DELETE FROM data1 WHERE c1 = 20000 ;
                                                QUERY PLAN
-------------------------------------------------------------------
 Delete on data1  (cost=0.00..233293.36 rows=1 width=6) (actual
time=1094.412..1094.412 rows=0 loops=1)
   WAL: records=2 bytes=110
   ->  Seq Scan on data1  (cost=0.00..233293.36 rows=1 width=6) (actual
time=175.959..1094.402 rows=1 loops=1)
         Filter: (c1 = '20002'::numeric)
         Rows Removed by Filter: 9999997
         WAL: records=1 bytes=56
 Planning Time: 0.065 ms
 Execution Time: 1094.429 ms
(8 rows)
```

3.2.12. INSERT

Specifying a value using the INSERT OVERRIDING USER VALUE statement for an IDENTITY column with the GENERATED ALWAYS clause specified is now allowed. However, the value specified for the IDENTITY column is ignored.

Example 43 INSERT OVERRIDING USER VALUE statement

```
postgres=> CREATE TABLE gen1(c1 INT GENERATED ALWAYS AS IDENTITY,
       c2 VARCHAR(10)) ;
CREATE TABLE
postgres=> INSERT INTO gen1 OVERRIDING USER VALUE VALUES
       (100, 'data1') ;
INSERT 0 1
postgres=> SELECT * FROM gen1 ;
 c1 |  c2
----+-------
  1 | data1
(1 row)
```

3.2.13. JSON

The following JSON related features have been added.

□ Allow Unicode escape

Unicode escapes are now allowed for character literals and identifiers, even if the server's character encoding is not UTF-8.

□ Datetime method of jsonpath

The datetime method that converts a character string to a date/time type has been added.

Example 44 datetime method

```
postgres=> SELECT
 jsonb_path_query(c1,   '$[*]   ?   (@.datetime()   <   "2015-08-
2".datetime())') FROM json1 ;
 jsonb_path_query
------------------
 "2015-8-1"
(1 row)
```

3.2.14. MAX/MIN pg_lsn

MAX / MIN function can be used for pg_lsn data type.

Example 45 MAX(pg_lsn) function

```
postgres=# SELECT MIN(flush_lsn), MAX(flush_lsn)
        FROM pg_stat_replication ;
    min     |    max
-----------+-----------
 0/94FD1E0 | 0/94FD1E0
(1 row)
```

3.2.15. ROW

ROW can now be directly extracted fields in some data types from the expressions.

Example 46 Extracting values from ROW expressions

```
postgres=> SELECT (ROW(2, 3.1)).f1, (ROW(4, 5.1)).f2 ;
 f1 | f2
----+-----
  2 | 5.1
(1 row)
```

3.2.16. SELECT FETCH FIRST WITH TIES

The WITH TIES clause can be specified in the SELECT FETCH FIRST statement. It outputs the same number of tuples as the number of tuples specified in the ROWS clause.

Example 47 SELECT FETCH FIRST WITH TIES statement

```
postgres=> SELECT * FROM data1 WHERE c1 < 4 ORDER BY c1
             FETCH FIRST 2 ROWS ONLY ;
 c1 |  c2
----+-------
  1 | data1
  1 | data1
(2 rows)

postgres=> SELECT * FROM data1 WHERE c1 < 4 ORDER BY c1
             FETCH FIRST 2 ROWS WITH TIES ;
 c1 |  c2
----+-------
  1 | data1
  1 | data1
  1 | data1
(3 rows)
```

3.2.17. VACUUM PARALLEL

VACUUM statements for indexes can now be parallelized. Parallelism works by default, and the degree of parallelism is determined by the number of indexes created on the table. To specify the degree of parallelism, specify a value between 0 and 1024 in the PARALLEL clause. This feature does not work with automatic VACUUM.

Example 48 PARALLEL specification of VACUUM statement

```
postgres=> VACUUM (PARALLEL 4, VERBOSE) data1 ;
INFO:  vacuuming "public.data1"
INFO:  launched 2 parallel vacuum workers for index vacuuming (planned: 2)
INFO:  scanned index "idx3_data1" to remove 1000 row versions by parallel
vacuum worker
DETAIL:  CPU: user: 0.39 s, system: 0.11 s, elapsed: 12.45 s
INFO:  scanned index "idx2_data1" to remove 1000 row versions by parallel
vacuum worker
DETAIL:  CPU: user: 0.39 s, system: 0.12 s, elapsed: 12.56 s
INFO:  scanned index "idx1_data1" to remove 1000 row versions
DETAIL:  CPU: user: 0.47 s, system: 0.17 s, elapsed: 53.26 s
INFO:  "data1": removed 1000 row versions in 1000 pages
…
```

3.2.18. Operator <->

Distance operators (<->) can now be used between several data types. GiST index and SP-GiST index can be used in box_ops operator class.

Example 49 Operator <->

```
postgres=> CREATE TABLE POINT_TBL(f1 point) ;
CREATE TABLE
postgres=> CREATE TABLE BOX_TBL (f1 box) ;
CREATE TABLE
postgres=> SELECT p.f1, b.f1, p.f1 <-> b.f1 AS dist_pb,
        b.f1 <-> p.f1 AS dist_bp FROM POINT_TBL p, BOX_TBL b ;
f1 | f1 | dist_pb | dist_bp
----+----+---------+---------
(0 rows)
```

3.2.19. Functions

The following functions have been added / enhanced.

□ Gen_random_uuid

The gen_random_uuid function has been provided to generate a random UUID.

Example 50 gen_random_uuid function

```
postgres=> SELECT gen_random_uuid() ;
           gen_random_uuid
--------------------------------------
 3759712a-e7f6-4348-b62c-85535c831e7e
(1 row)
```

□ Date format

FF1 to FF6 format patterns defined in SQL Standard 2016 have been added to the date format.

Example 51 Date format

```
postgres=> SELECT TO_CHAR(current_timestamp, 'HH:MI:SS.FF6');
     to_char
-----------------
 01:39:56.077670
(1 row)
```

Table 21 Added format pattern

Pattern	Output value	Note
FF1	1/10th of a second.	
FF2	1/100 second.	
FF3	Millisecond.	
FF4	1/10 ms.	
FF5	1/100 ms.	
FF6	Microsecond.	

The pattern SSSSS is now an alias for SSSS.

□ Min_scale, trim_scale

The min_scale function obtains the minimum number of digits from the NUMERIC type, excluding trailing zeros after the decimal point. The trim_scale function gets the value of the NUMERIC type excluding trailing zeros after the decimal point.

Example 52 min_scale, trim_scale function

```
postgres=> SELECT trim_scale(1.234000::numeric),
       min_scale(1.234000::numeric) ;
 trim_scale | min_scale
------------+-----------
     1.234 |         3
(1 row)
```

□ Jsonb_set_lax

The jsonb_set_lax function is the same as the jsonb_set function except that a null_value_treatment parameter (text type) has been added. The following values can be specified for this parameter.

Table 22 null_value_treatment parameter value

Value	Description	Note
use_json_null	Return NULL.	default value
raise_exception	Raise exception.	
delete_key	Delete key.	
return_target	Return target.	

Example 53 jsonb_set_lax function

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

□ Gcd, lcm

The lcm function, which returns the least common multiple, and the gcd function, which returns the greatest common divisor, have been added. Specify two numbers of the same type in the parameter. If both parameters are 0, these functions return 0.

Example 54 gcd, lcm function

```
postgres=> SELECT gcd(1071, 462), lcm(1071, 46) ;
 gcd |  lcm
-----+-------
  21 | 49266
(1 row)
```

□ Get_bit / set_bit

The offset parameter of the get_bit / set_bit function that specifies the bytea type has been changed from integer type to bigint type.

□ Unicode normalization

The normalize function that performs UNICODE normalization and the 'is_normalized' function that checks whether it is normalized have been implemented. You can also use the 'is normalized' syntax to check the specified text.

Syntax

```
text is [not] [form] normalized
```

In the [form] part, NFC (default), NFD, NFKC or NFKD can be specified.

Example 55 is normalized syntax

```
postgres=> SELECT '㋿' IS NFKD NORMALIZED ;
 is_normalized
---------------
 f
(1 row)
postgres=> SELECT normalize('㋿', NFKD) ;
 normalize
-----------
 令和
(1 row)
```

□ Pg_stat_reset_slru

This function resets the counter value in the pg_stat_slru catalog. If NULL is specified for the parameter of this function, all counters will be reset. By specifying the counter name ('name' column of pg_stat_slru catalog) to this function, the specified counter is reset.

Example 56 is normalized syntax

```
postgres=# SELECT pg_stat_reset_slru('subtrans') ;
pg_stat_reset_slru
--------------------


 (1 row)
```

3.3. Configuration parameters

In PostgreSQL 13 the following parameters have been changed.

3.3.1. Added parameters

The following parameters have been added.

Table 23 Added parameters

Parameter name	Description (context)	Default Val
autovacuum_vacuum_insert_scale_factor	Percentage of INSERT records for which automatic VACUUM is executed (sighup).	0.2
autovacuum_vacuum_insert_threshold	Number of INSERT records for which automatic VACUUM is executed (sighup).	1000
backtrace_functions	List of function names that output backtrace when an error occurs (superuser).	''
enable_groupingsets_hash_disk	Enable disk-based hash grouping set (user).	off
enable_hashagg_disk	Enable disk-based hash aggregation (user).	on
enable_incrementalsort	Enable incremental sorting (user).	on
ignore_invalid_pages	Ignore invalid page references during recovery (postmaster).	off
log_min_duration_sample	Time to output SQL statement log by sampling (superuser).	-1
log_parameter_max_length	Bind parameter value in the non-error statement logging messages are trimmed to setting value (superuser).	-1
log_parameter_max_length_on_error	Bind parameter value in the error statement logging messages are trimmed to setting value (user).	0
log_statement_sample_rate	SQL statement log output rate by sampling (superuser)	1.0
logical_decoding_work_mem	Work memory for decoding for the logical replication environment (user).	64MB
maintenance_io_concurrency	Number of I/O concurrent executions used for maintenance (user).	10
max_slot_wal_keep_size	Maximum MB size of WAL held by replication slot (sighup).	-1

Parameter name	**Description (context)**	**Default Val**
wal_receiver_create_temp_slot	Whether the wal receiver process creates a temporary replication slot (sighup).	off
wal_skip_threshold	Write the interval at which fsync is executed when wal_level is 'minimal' (user).	2MB

□ Output back trace

In the parameter backtrace_functions, specify the name of the function that performs trace output when an error occurs. The specified function is not the SQL function, but the function name in the source code.

Example 57 backtrace_functions parameter

```
postgres=# SET backtrace_functions = 'pg_strtoint32' ;
SET
postgres=# SELECT int 'foobar' ;
ERROR:  invalid input syntax for type integer: "foobar"
LINE 1: SELECT int 'foobar' ;
postgres=# \q
$ tail data/log/postgresql.log
2020-05-21 18:36:42.452 JST [2532] BACKTRACE:
        postgres: postgres postgres [local] SELECT() [0x807d2e]
        postgres: postgres postgres [local] SELECT(int4in+0xd) [0x7d27ad]
        postgres:  postgres  postgres  [local]  SELECT(InputFunctionCall+0x67)
[0x884d27]
        postgres: postgres postgres [local] SELECT(OidInputFunctionCall+0x31)
[0x884f31]
        postgres:   postgres   postgres   [local]   SELECT(coerce_type+0x361)
[0x5785f1]
        postgres: postgres postgres [local] SELECT(coerce_to_target_type+0x9b)
[0x577d8b]
...
        postgres:   postgres   postgres   [local]   SELECT(parse_analyze+0x4f)
[0x55874f]
        postgres: postgres postgres [local] SELECT(pg_analyze_and_rewrite+0x13)
[0x76ffa3]
        postgres: postgres postgres [local] SELECT() [0x770483]
        postgres:   postgres   postgres   [local]   SELECT(PostgresMain+0x1037)
[0x7718f7]
        postgres: postgres postgres [local] SELECT() [0x483e46]
        postgres:   postgres   postgres   [local]   SELECT(PostmasterMain+0xe88)
[0x6ff668]
        postgres: postgres postgres [local] SELECT(main+0x46a) [0x484a9a]
        /lib64/libc.so.6(__libc_start_main+0xf5) [0x7f19383693d5]
        postgres: postgres postgres [local] SELECT() [0x484b01]
2020-05-21 18:36:42.452 JST [2532] STATEMENT:  SELECT int 'foobar' ;
```

3.3.2. Changed parameters

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

Table 24 Changed parameters

Parameter name	Changes
allow_system_table_mods	The context changed from postmaster to superuser.
effective_io_concurrency	Changed to use directly specified value instead of using calculated value by internal logic.
max_files_per_process	The minimum value has been changed from 25 to 64.
log_line_prefix	Backend type (%b) has been added.
primary_conninfo	Context changed from postmaster to sighup.
primary_slot_name	Context changed from postmaster to sighup.
ssl_passphrase_command	Only SUPERUSER can be referenced.
track_activity_query_size	The maximum value has been changed from 102,400 to 1,048,576.

3.3.3. Parameters with default values changed

The following parameters have changed default values.

Table 25 Parameters that default value has been changed

Parameter name	PostgreSQL 12	PostgreSQL 13	Note
server_version	12.3	13beta1	
server_version_num	120003	130000	
ssl_min_protocol_version	TLSv1	TLSv1.2	

3.3.4. Removed parameter

The following parameter has been removed:

Table 26 Removed parameter

Parameter name	Note
log_parameters_on_error	Split into log_parameter_max_length and log_parameter_max_length_on_error

Hewlett Packard
Enterprise

3.4. Utilities

Describes the major enhancements of utility commands.

3.4.1. dropdb

The --force option (or -f option) has been added. Forces database deletion even if a connection session exists. The connected session will be closed.

Example 58 dropdb --force option

```
$ dropdb --force --echo demodb
SELECT pg_catalog.set_config('search_path', '', false);
DROP DATABASE demodb WITH (FORCE);
```

3.4.2. pg_basebackup

The following enhancements have been implemented in the pg_basebackup command.

□ --no-verify-checksums option

Do not verify checksums.

□ --no-manifest option

Disables generation of a backup manifest.

□ --manifest-force-encode option

Outputs the file name to be written to the manifest in hexadecimal. By default, only non-UTF-8 file names are represented in hexadecimal.

□ --manifest-checksums option

Specify the algorithm for creating the checksum. The options that can be specified are CRC32C, SHA224, SHA256, SHA384, SHA512 or NONE. The default value is CRC32C. If NONE is specified, the manifest will not include the checksum.

□ --no-estimate-size option

The pg_basebackup command has been changed to estimate the data size without specifying the --progress option. Specify the option --no-estimate-size when not calculating the data amount estimate. This option cannot be used with --progress.

Example 59 pg_basebackup ---no-estimate-size option

```
$ pg_basebackup --verbose --no-estimate-size -D back
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 1/44000028 on timeline 1
pg_basebackup: starting background WAL receiver
pg_basebackup: created temporary replication slot "pg_basebackup_3822"
pg_basebackup: write-ahead log end point: 1/44000138
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: syncing data to disk ...
pg_basebackup: renaming backup_manifest.tmp to backup_manifest
pg_basebackup: base backup completed
```

3.4.3. pg_dump

The following enhancements have been implemented in the pg_dump command.

□ --include-foreign-data option

Added --include-foreign-data option to dump FOREIGN SERVER data. This option specifies the FOREIGN SERVER name pattern. This option can be specified more than once.

Example 60 pg_dump --include-foreign-data option

```
$ pg_dump -d demodb --include-foreign-data=svr1
--
-- PostgreSQL database dump
…
CREATE FOREIGN TABLE public.foreign1 (
    c1 numeric,
    c2 character varying(10)
)
SERVER svr1;
…
--
-- Data for Name: data1; Type: TABLE DATA; Schema: public; Owner:
demo
--

COPY public.foreign1 (c1, c2) FROM stdin;
100     data1
\.
…
```

□ Output ALTER DEPEND ON EXTENSION statement

ALTER object DEPENDS ON EXTENSION statement is now output.

Example 61 ALTER DEPENDS ON EXTENSION statement

```
$ pg_dump -d demodb | grep EXTENSION
-- Name: cube; Type: EXTENSION; Schema: -; Owner: -
CREATE EXTENSION IF NOT EXISTS cube WITH SCHEMA public;
-- Name: EXTENSION cube; Type: COMMENT; Schema: -; Owner:
COMMENT ON EXTENSION cube IS 'data type for multidimensional
cubes';
ALTER FUNCTION public.func1() DEPENDS ON EXTENSION cube;
$
```

3.4.4. pg_rewind

The following three options have been added to the pg_rewind command.

□ --no-ensure-shutdown option

The pg_rewind will start in single-user mode if it detects a database cluster that has not shut down properly. Specify the --no-ensure-shutdown command and do not perform startup processing.

□ --write-recovery-conf option

The --write-recovery-conf option (or -R option) configures replication-related settings. This option must be specified with the --source-server option.

□ --restore-target-wal option

The --restore-target-wal option (or -c option) gets the restore_command parameter value from the target cluster and restores the required WAL segments from the archive.

Example 62 pg_rewind command

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

Usage:
  pg_rewind [OPTION]...

Options:
  -c,  --restore-target-wal              use  restore_command  in  target
configuration to
                               retrieve WAL files from archives
  -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
  -R, --write-recovery-conf      write configuration for replication
                               (requires --source-server)
      --debug                   write a lot of debug messages
      --no-ensure-shutdown       do not automatically fix unclean shutdown
  -V, --version                 output version information, then exit
  -?, --help                    show this help, then exit

Report bugs to <pgsql-bugs@lists.postgresql.org>.
PostgreSQL home page: <https://www.postgresql.org/>
```

3.4.5. pg_verifybackup

The pg_verifybackup command checks consistency by comparing the contents of the backup and the manifest. This command performs the following checks:

- Manifest file version
- The checksum of the manifest file itself
- File size

Hewlett Packard Enterprise

- File checksum
- WAL file integrity

Syntax

```
$ pg_verifybackup [option] directory
```

Table 27 pg_verifybackup command option

Long Option	Short Option	Description
--exit-on-error	-e	Exit immediately on error.
--ignore=PATH	-i	Ignore indicated path.
--manifest=PATH	-m	Use specified path for manifest.
--no-parse-wal	-n	Do not try to parse WAL files.
--quiet	-q	Do not print any output, except for errors.
--skip-checksums	-s	Skip checksum verification.
--wal-directory=PATH	-w	Use specified path for WAL files.
--version	-V	Output version information.
--help	-?	Output usage information.

Example 63 pg_verifybackup command

```
$ pg_basebackup -D back
$ pg_verifybackup back
backup successfully verified
$ echo $?
0
```

□ Validation error

The pg_verifybackup command outputs an error message if the size or content of the file has changed.

Example 64 pg_verifybackup command with error

```
$ pg_basebackup -D back
$ vi back/postgresql.conf
$ vi back/pg_hba.conf
$ pg_verifybackup back
pg_verifybackup: error: "pg_hba.conf" has size 4548 on disk but
size 4513 in the manifest
pg_verifybackup:    error:    checksum    mismatch    for    file
"postgresql.conf"
$
```

☐ Execution error

An error occurs when executing a backup for which a manifest has not been created or a backup created in tar format.

Example 65 pg_verifybackup command with error

```
$ pg_basebackup -D back --no-manifest
$ pg_verifybackup back
pg_verifybackup:     fatal:     could     not     open     file
"back/backup_manifest": No such file or directory
$ rm -r back
$ pg_basebackup -D back -Ft
$ pg_verifybackup back
pg_verifybackup: error: "base.tar" is present on disk but not in
the manifest
pg_verifybackup: error: "pg_wal.tar" is present on disk but not in
the manifest
pg_verifybackup:  error:  "base/13577/3433"  is  present  in  the
manifest but not on disk
…
Try "pg_waldump --help" for more information.
pg_verifybackup: error: WAL parsing failed for timeline 1
$
```

3.4.6. pg_waldump

The --quiet option (or -q option) has been added to the pg_waldump command. This option suppresses non-error screen output.

3.4.7. psql

The following features have been added to the psql command.

□ \dAc, \dAf, \dAo, \dAp commands

Commands for outputting operator information have been added.

Table 28 Added command

Command	Output contents	Note
\dAc	AM, Input Type, Storage Type, Operator class, Default?.	
\dAo	AM, Opfamily Name, Operator.	
\dAp	AM, Left arg type, Right arg type, Number, Proc name.	

Example 66 \dAc command

```
postgres=# \dAc
                          List of operator classes
  AM   |         Input type          | Storage type |     Operator class      | Default?
--------+-----------------------------+--------------+-------------------------+------
 brin   | bytea                       |              | bytea_minmax_ops        | yes
 brin   | "char"                      |              | char_minmax_ops         | yes
 brin   | name                        |              | name_minmax_ops         | yes
 brin   | bigint                      |              | int8_minmax_ops         | yes
 brin   | smallint                    |              | int2_minmax_ops         | yes
 brin   | integer                     |              | int4_minmax_ops         | yes
 brin   | text                        |              | text_minmax_ops         | yes
 brin   | oid                         |              | oid_minmax_ops          | yes
 brin   | tid                         |              | tid_minmax_ops          | yes
 brin   | box                         |              | box_inclusion_ops       | yes
…
```

□ \g, \gx commands

The feature has been added to the \g and \gx commands to temporarily change variables specified by the \pset command.

Example 67 \g command

```
postgres=> SELECT * FROM data1 WHERE c1=10 ;
 c1 |  c2
----+-------
 10 | data1
 10 | data1
 10 | data1
(3 rows)

postgres=> \g (format=csv)
c1,c2
10,data1
10,data1
10,data1
```

□ \warn command

The \warn command that outputs a message to standard error has been added.

Example 68 \warn command

```
postgres=> \warn `date`
Thu May 21 21:10:09 JST 2020
```

□ \e command

After closing the editor, the edited SQL statement is now displayed.

Example 69 \e command

```
postgres=> \e
postgres=> SELECT * FROM data1
postgres->
```

□ \d+ command

The table type is now output as a 'Persistence' column.

Example 70 \d command

```
postgres=> \d+
                              List of relations
 Schema    |       Name         | Type  | Owner    | Persistence |    Size    | Description
-----------+--------------------+-------+----------+-------------+------------+------
 pg_temp_3 | temp1              | table | demo     | temporary   | 8192 bytes |
 public    | data1              | table | demo     | permanent   | 42 MB      |
 public    | unlogged1          | table | demo     | unlogged    | 16 kB      |
(3 rows)
```

□ PROMPT2 value

Variable PROMPT2 now a %w can be used. If this value is specified, it will be converted to a space of the same length as PROMPT1.

Example 71 PROMPT2 variable value

```
postgres=> \set PROMPT2 '>%w'
postgres=> CREATE TABLE data1(c1 INT, c2 VARCHAR(10))
>          TABLESPACE pg_default ;
CREATE TABLE
```

□ I/O error detection

Errors when outputting files can now be detected.

□ Default value of PROMPT1/PROMPT2

Default prompt now includes %x to indicate transaction status.[6]。

3.4.8. reindexdb

The --jobs option to execute parallel processing has been added to the reindexdb command. The default value is 1 and no parallel processing is performed.

[6] Refer 2.5.7 Psql default prompt psql

Example 72 reindexdb --jobs option

```
$ reindexdb --concurrently --jobs 2 postgres
reindexdb: warning: cannot reindex system catalogs concurrently,
skipping all
```

3.4.9. vacuumdb

The --parallel option (or -P option) has been added. Specify the degree of parallelism (0 to 1024) in the option. This option cannot be specified together with the --full or
--analyze-only option.

Example 73 vacuumdb --parallel option

```
$ vacuumdb --parallel=4 postgres
vacuumdb: vacuuming database "postgres"
```

3.4.10. Other

The URL to www.postgresql.org has been added to the output of the --help option for many commands.

Example 74 --help option

```
$ pg_controldata --help
pg_controldata displays control information of a PostgreSQL database cluster.

Usage:
  pg_controldata [OPTION] [DATADIR]

Options:
 [-D, --pgdata=]DATADIR  data directory
  -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>.
PostgreSQL home page: <https://www.postgresql.org/>
```

3.5. Contrib modules

Describe new features related to the Contrib modules.

3.5.1. adminpack

The pg_file_sync function has been added to the extension adminpack. This function guarantees a reliable write to the specified file or directory.

Example 75 pg_file_sync function

```
postgres=# CREATE EXTENSION adminpack ;
CREATE EXTENSION
postgres=# SELECT
    pg_file_sync('postgresql-2020-05-21_120622.log') ;
 pg_file_sync
--------------


(1 row)
```

3.5.2. auto_explain

The log_wal parameter to output WAL information has been added to the extension auto_explain. The default value is 'off', and WAL information is not output. If this parameter is set to 'on', auto_explain.log_analyze must also be set to 'on'.

Example 76 wal_log parameter

```
postgres=# LOAD 'auto_explain' ;
LOAD
postgres=# SET auto_explain.log_wal = on ;
SET
postgres=# SET auto_explain.log_analyze = on ;
SET
postgres=# SET auto_explain.log_min_duration = 0 ;
SET
postgres=# DELETE FROM data1 WHERE c1=100 ;
DELETE 1
postgres=# \! tail -7 log/postgres.log
2020-05-21 21:16:28.707 JST [4731] LOG:  duration: 0.064 ms  plan:
        Query Text: DELETE FROM data1 WHERE c1=100 ;
        Delete  on  data1   (cost=0.00..4.50  rows=2  width=6)  (actual
time=0.063..0.063 rows=0 loops=1)
          WAL: records=2 fpi=2 bytes=8966
          ->  Seq Scan on data1  (cost=0.00..4.50 rows=2 width=6) (actual
time=0.018..0.026 rows=2 loops=1)
                Filter: (c1 = '100'::numeric)
                Rows Removed by Filter: 198
```

3.5.3. dict_int

The dict_int extension now has an ABSVAL attribute. If this attribute is specified, the sign specified for the integer value will be ignored.

Example 77 ABSVAL attribute

```
postgres=# CREATE EXTENSION dict_int ;
CREATE EXTENSION
postgres=# ALTER TEXT SEARCH DICTIONARY intdict (ABSVAL = true) ;
ALTER TEXT SEARCH DICTIONARY
postgres=# \dFd+ intdict
                        List of text search dictionaries
 Schema |  Name   |        Template         |  Init options    |      Description
--------+---------+-------------------------+------------------+-------------
 public | intdict | public.intdict_template | absval = 'true'  | dictionary
for integers
(1 row)
```

3.5.4. ltree

The ltree, lquery, and ltxtquery types now support binary I/O.

Example 78 information on the ltree data type

```
postgres=> SELECT typreceive, typsend FROM pg_type WHERE typname='ltree' ;
-[ RECORD 1 ]----------
typreceive | ltree_recv
typsend    | ltree_send
```

3.5.5. pageinspect

The following enhancements have been implemented in pageinspect extension.

□ Heap_tuple_infomask_flags function

This function converts the values of t_infomask, t_infomask2 into a human readable form.

Example 79 heap_tuple_infomask_flags function

```
postgres=# SELECT t_infomask, t_infomask2, raw_flags, combined_flags
             FROM heap_page_items(get_raw_page('data1', 0)),
             LATERAL heap_tuple_infomask_flags(t_infomask, t_infomask2)
             m(raw_flags, combined_flags) ;
-[ RECORD 1 ]--+-------------------------------------------------------
t_infomask     | 2306
t_infomask2    | 2
raw_flags      | {HEAP_HASVARWIDTH,HEAP_XMIN_COMMITTED,HEAP_XMAX_INVALID}
combined_flags | {}
```

□ Bt_metap function

An allequalimage column has been added to the output of the bt_metap function to indicate the status of duplicate data in the index.

Example 80 Output of bt_metap function

```
postgres=# SELECT * FROM bt_metap('idx1_data1') ;
-[ RECORD 1 ]-----------+-------
magic                   | 340322
version                 | 4
root                    | 3
level                   | 1
fastroot                | 3
fastlevel               | 1
oldest_xact             | 0
last_cleanup_num_tuples | -1
allequalimage           | t
```

□ Bt_page_items function

The dead, htid, and tids columns have been added to the output of the bt_page_items function.

Example 81 Output of bt_page_items function

```
postgres=# SELECT * FROM bt_page_items('idx1_data1', 1) ;
-[ RECORD 1 ]-----------------------
itemoffset | 1
ctid       | (0,1)
itemlen    | 16
nulls      | f
vars       | t
data       | 0b 00 80 01 00 00 00 00
dead       | f
htid       | (0,101)
tids       |
```

Table 29 Added columns

Column name	Description
dead	A boolean value indicating the value of the LP_DEAD bit.
htid	Indicates a single heap TID value for each tuple.
tids	Array of TID.

3.5.6. pg_stat_statements

The features to output execution plan and WAL information have been added to the pg_stat_statements extension.

□ Track_planning parameter

This parameter specifies whether to track the execution plan. The default value is 'on'. If this parameter is specified, information about the execution plan will be added to the pg_stat_statements view.

□ Pg_stat_statements view

The following columns have been added to the pg_stat_statements view:

Table 30 Added columns

Column name	Data type	Description
plans	bigint	Number of times the statement was planned.
total_plan_time	double precision	Total time spent planning the statement.
min_plan_time	double precision	Minimum time spent planning the statement, in milliseconds.
max_plan_time	double precision	Maximum time spent planning the statement, in milliseconds.
mean_plan_time	double precision	Meantime spent planning the statement, in milliseconds.
stddev_plan_time	double precision	Population standard deviation of time spent planning the statement, in milliseconds.
wal_records	bigint	Total count of WAL records.
wal_fpi	bigint	Total number of WAL full page writes.
wal_bytes	numeric	Total amount of WAL bytes.

The column names in the pg_stat_statements view have been changed as follows.

Table 31 Modified columns

PostgreSQL 12	PostgreSQL 13	Description
total_time	total_exec_time	Total time spent executing the statement.
min_time	min_exec_time	Minimum time spent executing the statement.
max_time	max_exec_time	Maximum time spent executing the statement.
mean_time	mean_exec_time	Mean time spent executing the statement.
stddev_time	stddev_exec_time	Population standard deviation of time spent executing the statement.

Example 82 Refer the pg_stat_statements view

```
postgres=# SELECT queryid, plans, total_plan_time, min_plan_time
        FROM pg_stat_statements WHERE plans != 0;
      queryid        | plans |   total_plan_time    | min_plan_time
----------------------+--------+----------------------+----------------
  1809546128015582813 |     1 |               0.9533 |       0.9533
  1580010417825738145 |   900 |   29.697499999999998 |        0.0194
   726988806503882440 |     1 |               0.0889 |       0.0889
  6547867665616429211 |     1 |               0.3364 |       0.3364
  1518237015596383425 |     1 |               0.0522 |       0.0522
  6090531631178874043 |   900 |    51.09360000000001 |        0.0221
 -8944905077122340531 |     6 |  0.22340000000000002 |        0.0333
  -261743227044297956 |    10 | 0.061900000000000004 |        0.0054
…
```

□ SQL statement normalization

The SELECT statement with the FOR UPDATE clause specified and the SELECT statement without it are now separated. Previously, the presence or absence of the FOR UPDATE clause was ignored.

3.5.7. postgres_fdw

The following options have been added to the postgres_fdw extension:

□ Password_required option

By default, general users are refused connections without a password. This behavior can now be changed by specifying the password_required attribute when creating / changing a user mapping. Only superusers can change this option.

Example 83 Create USER MAPPING object without password

```
postgres=# CREATE USER MAPPING FOR demo SERVER remhost1
        OPTIONS (user 'demo') ;
CREATE USER MAPPING
```

Example 84 Creation of external tables by general users and errors

```
postgres=> CREATE FOREIGN TABLE data1(c1 INT, c2 VARCHAR(10))
       SERVER remhost1 ;
CREATE FOREIGN TABLE
postgres=> SELECT * FROM data1 ;
ERROR:  password is required
DETAIL:   Non-superusers must provide a password in the user
mapping.
```

By setting the option password_required to 'false', password setting for user mapping can be avoided.

Example 85 Settings to allow password omission

```
postgres=# ALTER USER MAPPING FOR demo SERVER remhost1 OPTIONS
              (password_required 'false');
ALTER USER MAPPING
```

□ Additional authentication options

The sslkey and sslcert options can be specified as options when creating a user mapping. These options can only be specified if the superuser is creating/modifying USER MAPPING.

3.5.8. bool_plperl

The extension bool_plperl was added. Use this extension with PL/Perl extension. Used to correctly transfer the value of a bool value to a Perl program.

Example 86 bool_plperl extension

```
postgres=# CREATE EXTENSION plperl ;
CREATE EXTENSION
postgres=# CREATE EXTENSION bool_plperl ;
CREATE EXTENSION
postgres=# CREATE FUNCTION perl_and(bool, bool) RETURNS bool
         TRANSFORM FOR TYPE bool
         AS $$
            my ($a, $b) = @_;
            return $a && $b;
         $$ LANGUAGE plperl ;
CREATE FUNCTION
```

URL List

The following websites are references to create this material.

□ Release Notes

https://www.postgresql.org/docs/13/release.html

□ Commitfests

https://commitfest.postgresql.org/

□ PostgreSQL 13 Beta Manual

https://www.postgresql.org/docs/13/index.html

□ Git

git://git.postgresql.org/git/postgresql.git

□ GitHub

https://github.com/postgres/postgres

□ PostgreSQL 13 Open Items

https://wiki.postgresql.org/wiki/PostgreSQL_13_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 13 Beta 1

https://www.postgresql.org/about/news/2040/

□ PostgreSQL Developer Information

https://wiki.postgresql.org/wiki/Development_information

□ Slack - postgresql-jp

https://postgresql-jp.slack.com/

Hewlett Packard
Enterprise

Change history

Change history

Version	Date	Author	Description
0.1	Apr 09, 2020	Noriyoshi Shinoda	Create an internal review version. Reviewers: Tomoo Takahashi Akiko Takeshima (Hewlett Packard Enterprise Japan)
1.0	May 27, 2020	Noriyoshi Shinoda	Modification completed according to PostgreSQL 13 Beta 1 Reviewer: Satoshi Nagayasu (Uptime Technologies, LCC.)