May 27, 2025

PostgreSQL 18 New Features

With Examples (Beta 1)

Noriyoshi Shinoda

Index

1. About This Document

1.1. Purpose

The purpose of this document is to provide information about the major new features of open-source RDBMS PostgreSQL 18 (18.0) Beta 1.

1.2. Audience

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

1.3. Scope

This document describes the major differences between PostgreSQL 17 (17.5) and PostgreSQL 18 (18.0). 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.

- Bugfix
- Performance improvement by changing internal behavior
- Improvement of regression test
- Operability improvement by the psql command tab input
- Improvement of the pgbench command
- Improvement of documentation, modify typos 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 17.5 (for comparison) PostgreSQL 18 Beta 1 (18.0) (2025/05/05 20:41)
Operating System	Red Hat Enterprise Linux 9 Update 5 (x86-64)
Configure options	--with-ssl=openssl --with-lz4 --with-zstd --with-llvm --with-libxml --enable-injection-points --with-libnuma --with-liburing

1.5. The Question, Comment, and Responsibility

The contents of this document are not an official opinion of PostgreSQL Global Development Group. 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@gmail.com).

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 of notation

Notation	Description
#	Shell prompt for Linux root user.
$	Shell prompt for Linux general user.
Bold	The user input string.
postgres=#	The psql command prompt used by PostgreSQL users with the SUPERUSER attribute.
postgres=>	The psql command prompt used by PostgreSQL users without the SUPERUSER attribute.
Underline	Important output items.
<<PASSWORD>>	Replaced by password string.
…	Indicates that it is omitted.

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 18. This chapter provides an overview of typical new features and benefits. Details of the new features will be explained in "3. New Feature Detail".

2.1. Improvements to adapt to large scale environments

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

□ Creating Virtual Columns

Allows the creation of virtual columns without data storage as columns of a table.

□ VACUUM / ANALYZE statements for partitioned tables

The VACUUM and ANALYZE statements can now specify an ONLY clause that does not recursively process partitions.

2.2. Improved reliability

PostgreSQL 18 implements the following enhancements to improve reliability.

□ Output formats for the pg_dumpall command

The pg_dumpall command can now specify output formats other than text.

□ Tar format support for pg_verifybackup command

The pg_verifybackup command can now be executed for tar format backups

2.3. Improved maintainability

The following features that can improve operability have been added.

□ Statistics Migration

Optimizer statistics can now be included in dump files for the pg_dump and pg_dumpall commands. Functions have also been added to set and remove optimizer statistics for tables and columns.

□ OAUTH SASL

Supports authentication by OAUTHBearer/SASL.

□ EXPLAIN statement

Many options have been added to the EXPLAIN statement.

2.4. Improvements in Programming

The following functions have been added to SQL statements.

□ Access to Before Update Data

The RETURNING clause of the update DML can now specify an identifier that indicates the data before (OLD) and after (NEW) the update.

□ UUID v7 function

A function is provided that returns a UUID v7 value with a fixed leading portion.

2.5. Preparing for future new features

PostgreSQL 18 is now ready for features that will be provided in future versions.

□ Asynchronous I/O Infrastructure-

The infrastructure for using asynchronous I/O has been implemented. The parameter io_method specifies the asynchronous I/O method to be used.

□ NUMA support

Functions and views to use the basic NUMA environment have been added.

2.6. Incompatibility

In PostgreSQL 18, the following specifications have been changed from PostgreSQL 17.

2.6.1. Support Information

PostgreSQL 18 changes the supported versions for the following platforms and tools

The following platforms and tools are no longer supported. [edadeb0, 972c2cd, 6c66b74, 45363fc]

- PA-RISC processor
- LLVM 13 and below
- OpenSSL 1.1.0 and below
- Python 3.2

The changes in component support versions required to build PostgreSQL 18 are as follows. [a70e01d, 6c66b74]

- OpenSSL 1.1.1 or higher required
- Python 3.6.8

The following components are now supported in PostgreSQL 18.

- Tcl 9 support added. [32a2aa7]

2.6.2. MD5 Password

MD5 passwords are now deprecated. Generating an MD5 password will generate a warning by default. This warning can be suppressed by setting the parameter md5_password_warnings to 'off'. [db6a4a98]

Example 1 MD5 Password Warning

```
postgres=# SET password_encryption = md5 ;
SET
postgres=# SHOW md5_password_warnings ;
 md5_password_warnings
-----------------------
 on
(1 row)

postgres=# CREATE USER user1 PASSWORD 'user1' ;
WARNING:  setting an MD5-encrypted password
DETAIL:  MD5 password support is deprecated and will be removed in a future release of PostgreSQL.
HINT:  Refer to the PostgreSQL documentation for details about migrating to another password type.
CREATE ROLE
```

2.6.3. Configure command

The configure command has been changed as follows.

□ Option --disable-spin-locks

This option has been removed. [e256266]

□ Option --disable-atomics

This option has been removed. [8138526]

□ Command flex

Version checks have been removed. [0869ea4]

2.6.4. Initdb command

The data checksum feature is enabled by default. To disable checksums, specify the --no-data-checksums option. [04bec89]

Example 2 Checksum enabled by default

```
$ initdb data
The files belonging to this database system will be owned by user "postgres".
...
The default text search configuration will be set to "simple".

Data page checksums are enabled.
...
```

2.6.5. COPY statement

The following changes have been implemented for the COPY statement.

□ Prohibition of the COPY FREEZE statement for external tables

The COPY FREEZE statement for external tables is now explicitly prohibited. [401a695]

Example 3 Prohibition of COPY FREEZE statement for external tables

```
postgres=# CREATE FOREIGN TABLE data1(col1 INT, col2 VARCHAR(10))
                SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=# COPY data1 FROM stdin FREEZE ;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself, or an EOF signal.
>> 1,data1
>> \.
ERROR:  cannot perform COPY FREEZE on a foreign table
```

□ CSV mode specification change

When executing the COPY statement in CSV mode and reading data from a source other than STDIN, '\.' is no longer considered data completion. [7702337]

Example 4 Contents of CSV file data1.csv

```
Data1
Data2
\.
Data3
```

Example 5 PostgreSQL 17 behavior

```
postgres=> COPY data11 FROM '/home/postgres/data1.csv' CSV ;
COPY 2
postgres=> SELECT * FROM copy1 ;
  col1
-------
 Data1
 Data2
(2 rows)
```

Example 6 PostgreSQL 17 behavior

```
postgres=> COPY data1 FROM '/home/postgres/data1.csv' CSV ;
COPY 4
postgres=> SELECT * FROM data1 ;
  col1
-------
 Data1
 Data2
 \.
 Data3
(4 rows)
```

2.6.6. CREATE OPERATOR CLASS statement

The RECHECK clause in the CREATE OPERATOR CLASS and ALTER OPERATOR FAMILY statements is no longer accepted. The RECHECK clause in these statements was valid prior to PostgreSQL 8.4 and did not cause an error if specified until PostgreSQL 17. [7da1bdc]

2.6.7. CREATE SUBSCRIPTION statement

The default value of the 'streaming' option when creating SUBSCRIPTION has been changed from 'off' to 'parallel'. [1bf1140]

Example 7 Change default value when creating SUBSCRIPTION

```
postgres=# CREATE SUBSCRIPTION sub1 CONNECTION 'host=dbsvr1 port=5432
                dbname=postgres' PUBLICATION pub1 ;
NOTICE:  created replication slot "sub1" on publisher
CREATE SUBSCRIPTION
postgres=# SELECT subname, substream FROM pg_subscription ;
 subname | substream
---------+-----------
 sub1    | p
(1 row)
```

2.6.8. CREATE TABLE statement

Partition tables can no longer be created as UNLOGGED tables. Partitions can be created as UNLOGGED tables. [e2bab2d]

Example 8 Error creating UNLOGGED partition table

```
postgres=> CREATE UNLOGGED TABLE part1(col1 INT, col2 VARCHAR(10))
                PARTITION BY RANGE(col1) ;
ERROR:  partitioned tables cannot be unlogged
postgres=> CREATE TABLE part1(col1 INT, col2 VARCHAR(10)) PARTITION BY
                RANGE(col1) ;
CREATE TABLE
postgres=> CREATE UNLOGGED TABLE part1v1 PARTITION OF part1 FOR VALUES
                FROM (0) TO (10) PARTITION BY LIST(col2) ;
ERROR:  partitioned tables cannot be unlogged
```

2.6.9. EXPLAIN statement

The BUFFERS clause is now enabled by default. [c2a4078]

Example 9 PostgreSQL 17 behavior

```
postgres=> EXPLAIN ANALYZE SELECT * FROM data1 WHERE col1=1000 ;
                            QUERY PLAN
-------------------------------------------------------------------------------…
 Index Scan using data1_pkey on data1  (cost=0.42..8.44 rows=1 width=10) (act…
   Index Cond: (col1 = 1000)
 Planning Time: 0.062 ms
 Execution Time: 0.018 ms
(4 rows)
```

Example 10 PostgreSQL 18 behavior

```
postgres=> EXPLAIN ANALYZE SELECT * FROM data1 WHERE col1=1000 ;
                             QUERY PLAN
-------------------------------------------------------------------------------…
 Index Scan using data1_pkey on data1  (cost=0.42..8.44 rows=1 width=10) (act…
   Index Cond: (col1 = 1000)
   Buffers: shared hit=4
 Planning Time: 0.065 ms
 Execution Time: 0.023 ms
(5 rows)
```

2.6.10. GRANT statement

The RULE privilege on objects has been removed since PostgreSQL 8.2, but the GRANT statement did not raise an error. The RULE privilege has been completely removed from this version. [fefa76f]

Example 11 Remove RULE privileges

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 VARCHAR(10)) ;
CREATE TABLE
postgres=> GRANT RULE ON data1 TO PUBLIC ;
ERROR:  unrecognized privilege type "rule"
```

2.6.11. Psql

The output format of the \conninfo metacommand has been changed. [bba2fbc]

Example 12 PostgreSQL 17 psql command output

```
postgres=> \conninfo
You are connected to database "postgres" as user "demo" on host "dbsvr1" (address
"192.168.1.100") at port "5442".
```

Example 13 PostgreSQL 18 psql command output

```
postgres=> \conninfo
        Connection Information
      Parameter       |     Value
----------------------+---------------
 Database             | postgres
 Client User          | demo
 Host                 | dbsvr1
 Host Address         | 192.168.1.100
 Server Port          | 5432
 Options              |
 Protocol Version     | 3
 Password Used        | true
 GSSAPI Authenticated | false
 Backend PID          | 160057
 TLS Connection       | false
 Superuser            | off
 Hot Standby          | off
(13 rows)
```

2.6.12. Pg_backend_memory_contexts view

The starting number of the level column in the pg_backend_memory_contexts view has been changed from zero to one. The same change has been implemented for the output of the pg_log_backend_memory_contexts function. [d9e0386]

Example 14 PostgreSQL 17 pg_backend_memory_contexts view

```
postgres=# SELECT level,name FROM pg_backend_memory_contexts order by 1
               LIMIT 2 ;
 level |         name
-------+----------------------
     0 | TopMemoryContext
     1 | TopTransactionContext
(2 rows)
```

Example 15 PostgreSQL 18 pg_backend_memory_contexts view

```
postgres=# SELECT level,name FROM pg_backend_memory_contexts order by 1
               LIMIT 2 ;
 level |                      name
-------+-------------------------------------------------
     1 | TopMemoryContext
     2 | Map from relid to OID of cached composite type
(2 rows)
```

3. New Feature Detail

3.1. Architecture

Describes the improved architecture and features in PostgreSQL 18.

3.1.1. Modified System catalogs

The following catalogs and views have been changed. [12227a1, 32d3ed8, f0d1127, 559efce, 17cc5f6, 7054186, e7a9496, 02a8d0c, a0f1fce, ca87c41, f92c854, 30a6ed0, 75eb976, a051e71, 83ea6c5, bb8dff9, ac0e331, 2421e9a, 99f8f3f, 60f566b, 89f908a, 73eba50, 8cc139b, 0d6c477]

Table 4 Add system catalogs and views

Catalog/View name	Description
pg_aios	The pg_aios view lists all asynchronous I/O handles.
pg_shmem_allocations_numa	Shows how shared memory allocations in the server's main shared memory segment are distributed across NUMA nodes.

Table 5 System catalogs/views with additional columns

Catalog/View name	Add column	Data type	Description
pg_backend_memory_contexts	type	text	Context type
	path	integer[]	Array showing context hierarchy
pg_class	relallfrozen	integer	Number of pages marked as all frozen
pg_constraint	conenforced	boolean	Is the constraint specified as ENFORCED?
	conperiod	boolean	Specify WITHOUT OVERLAPS or PERIOD
pg_publication	pubgencols	boolean	Replicate the values of the generated columns?
pg_replication_slots	two_phase_at	pg_lsn	Address from which the decoding of prepared transactions is enabled
pg_stat_{all|user|sys}_tables	total_vacuum_time	double precision	Total time for VACUUM execution
	total_autovacuum_time	double precision	Total time for automatic VACUUM execution

Catalog/View name	Add column	Data type	Description
	total_analyze_time	double precision	Total time of ANALYZE execution
	total_autoanalyze_time	double precision	Total time for automatic ANALYZE execution
pg_stat_checkpointer	num_done	bigint	Number of checkpoints completed
	slru_written	bigint	Number of SLRU buffers written during checkpoint
pg_stat_database	parallel_workers_to_launch	bigint	Number of parallel workers attempted to be activated
	parallel_workers_launched	bigint	Number of parallel workers activated
pg_stat_io	read_bytes	numeric	Number of bytes read
	write_bytes	numeric	Number of bytes write
	extend_bytes	numeric	Number of extended bytes
pg_stat_progress_analyze	delay_time	double precision	Sleep time due to cost-based delay
pg_stat_progress_vacuum	delay_time	double precision	Sleep time due to cost-based delay
pg_stat_subscription_stats	confl_update_origin_differs	bigint	Number of tuples modified by other sources during execution of the UPDATE statement
	confl_update_exists	bigint	Number of tuples that violated the unique constraint during execution of the UPDATE statement
	confl_update_missing	bigint	Number of tuples not found to be changed during execution of the UPDATE statement
	confl_delete_origin_differs	bigint	Number of tuples modified by other sources during execution of the DELETE statement
	confl_delete_missing	bigint	Number of tuples not found for deletion during execution of the DELETE statement

Catalog/View name	Add column	Data type	Description
	confl_multiple_unique_conflicts	bigint	Number of tuples that violated the unique constraint when executing a multi-tuple INSERT statement
	confl_insert_exists	bigint	Number of tuples that violated the unique constraint when the INSERT statement was executed

Table 6 System catalogs/views with removed columns

Catalog/View name	Remove column	Description
pg_backend_memory_contexts	parent	Due to the addition of the PATH column, it is no longer needed.
pg_attribute	attcacheoff	No longer needed due to changes in cache structure
pg_stat_wal	wal_write	For integration into pg_stat_io view
	wal_sync	For integration into pg_stat_io view
	wal_write_time	For integration into pg_stat_io view
	wal_sync_time	For integration into pg_stat_io view
pg_stat_io	op_bytes	Because it was a fixed value

Table 7 Views in the information_schema schema with modified content

Catalog/View name	Column name	Description
table_constraints	enforced	Outputs YES or NO depending on column setting (previously only YES)

Table 8 System catalogs/views with modified output

Catalog / View name	Description
pg_aggregate	Includes max/min functions for user-defined types
pg_attribute	Outputs 'v' indicating virtual column in attgenerated column
pg_class	Outputs information on TOAST table in pg_index catalog
pg_constraint	Includes information on NOT NULL constraints for columns
pg_default_acl	L (Large Object) output in defaclobjtype column
pg_proc	Includes max/min functions for user-defined types
pg_replication_slots	Outputs idle_timeout in invalidation_reason column
pg_stat_io	WAL information added

□ Pg_aios view

The pg_aios view outputs the status of the currently used asynchronous I/O handle. [60f566b]

Table 9 Pg_aios view column structure

Column name	Data type	Description
pid	integer	Server process ID performing I/O
io_id	integer	I/O handle ID
io_generation	bigint	I/O handle generation
state	text	I/O handle status
operation	text	I/O handle operations
off	bigint	I/O operation offset
length	bigint	I/O operation length
target	text	I/O execution target
handle_data_len	smallint	I/O operation data length
raw_result	integer	Low-level I/O results
result	text	High-level I/O results
target_desc	text	I/O target description
f_sync	boolean	I/O execution flags
f_localmem	boolean	Local memory flag
f_buffered	boolean	Flag indicating buffer I/O

□ Pg_shmem_allocations_numa view

This view shows how shared memory segments are distributed among NUMA nodes. [8cc139b]

Table 10 Pg_shmem_allocations_numa view column structure

Column name	Data type	Description
name	text	Memory area name
numa_node	integer	NUMA node number
size	bigint	Memory size

Example 16 Retrieving the pg_shmem_allocations_numa view

```
postgres=# SELECT * FROM pg_shmem_allocations_numa ;
                      name                      | numa_node |   size
------------------------------------------------+-----------+----------
 subtransaction                                 |         0 |   270336
 notify                                         |         0 |   139264
 Shared Memory Stats                            |         0 |   319488
 serializable                                   |         0 |   270336
 PROCLOCK hash                                  |         0 |     4096
 FinishedSerializableTransactions               |         0 |     4096
...
```

□ NOT NULL constraint

NOT NULL constraints on table columns are now included in the pg_constraint catalog. The constraint name is automatically generated from the table and column names. The value of the contype column of the NOT NULL constraint is specified as 'n'. [14e87ff]

Example 17 Cataloging NOT NULL Constraints

```
postgres=> CREATE TABLE data1(col1 INT NOT NULL, col2 VARCHAR(10)) ;
CREATE TABLE
postgres=> \d+ data1
                                                Table "public.data1"
 Column |         Type          | Collation | Nullable | Default | Storage …
--------+-----------------------+-----------+----------+---------+---------…
 col1   | integer               |           | not null |         | plain   …
 col2   | character varying(10) |           |          |         | extended…
Not-null constraints:
    "data1_col1_not_null" NOT NULL "col1"
Access method: heap
postgres=> SELECT conname FROM pg_constraint WHERE contype='n' AND
                conname LIKE 'data1%' ;
      conname
--------------------
 data1_col1_not_null
(1 row)
```

□ Pg_stat_{all|user|sys}_tables view

VACUUM and ANALYZE execution times have been added to the pg_stat_{all|user|sys}_tables view. The manually executed and automatically executed cases are cumulated, respectively. [30a6ed0]

Example 18 Cumulative VACUUM / ANALYZE execution time

```
postgres=> ANALYZE data1 ;
ANALYZE
postgres=> VACUUM data1 ;
VACUUM
postgres=> SELECT relname, total_vacuum_time, total_autovacuum_time,
         total_analyze_time, total_autoanalyze_time FROM pg_stat_user_tables ;
-[ RECORD 1 ]----------+------
relname                | data1
total_vacuum_time      | 1826
total_autovacuum_time  | 14607
total_analyze_time     | 125
total_autoanalyze_time | 1475
```

3.1.2. Logical Replication Enhancements

The following new features have been implemented for logical replication

□ Log output when conflicts occur

Information about conflicts that occur when updating against the standby database are now output to the log. The reason for the conflict is output in the format “conflict={reason}”. The parameter track_commit_timestamp must be set to 'on' on the subscriber side (default value 'off') to detect the conflict reasons update_differ and delete_differ. [9758174, 73eba50]

Table 11 Reason for conflict output

Output text	Description (DETAIL log)	Replication
insert_exists	INSERT statement was executed but a tuple with the same key already exists (Key already exists in unique index)	Stop
update_differ	UPDATE statement target tuple was updated by another origin (Updating the row that was modified locally)	Continue
update_exists	UPDATE statement was executed but a tuple with the same key exists (Key already exists in unique index)	Stop
update_missing	UPDATE statement was executed but the target tuple does not exist (Could not find the row to be updated)	Continue
delete_differ	DELETE statement was executed but the target tuple was updated by another origin (Updating the row that was modified locally)	Continue
delete_missing	DELETE statement executed but target tuple does not exist (Could not find the row to be deleted)	Continue
multiple_unique_conflicts	Multiple tuples violate unique constraints (Key already exists in unique index)	Stop

Example 19 Error log when executing INSERT statement

```
ERROR:  conflict detected on relation "public.data1": conflict=insert_exists
DETAIL:  Key already exists in unique index "data1_pkey", modified in transaction 776.
        Key (col1)=(100); existing local tuple (100, standby1); remote tuple (100, primary1).
CONTEXT:  processing remote data for replication origin "pg_24577" during message type "INSERT" for replication target relation "public.data1" in transaction 761, finished at 0/3059248
```

□ Propagation of Generated Columns

The option PUBLISH_GENERATED_COLUMNS has been added to the CREATE PUBLICATION statement to determine whether the values of generated columns (GENERATED ALWAYS AS STORED specified columns) are to be replicated. The default value for this option is 'none', which means that the values of the generated columns will not be propagated. To propagate generated columns, specify 'stored' for this option. The value of the specified option is stored in the pubgencols_type column of the pg_publication catalog. [7054186, 745217a, 7054186, e65dbc9, 117f9f3]

Example 20 Propagation settings for generated columns (PUBLICATION side)

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY,
            col2 INT GENERATED ALWAYS AS (col1*2) STORED, col3 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE PUBLICATION pub1 FOR TABLE data1(col1, col2, col3)
               WITH (publish_generated_columns = stored) ;
CREATE PUBLICATION
postgres=> SELECT pubname, pubgencols FROM pg_publication ;
-[ RECORD 1 ]----
pubname    | pub1
pubgencols | s
postgres=> INSERT INTO data1(col1, col3) VALUES (100, 'data1') ;
INSERT 0 1
```

Example 21 Confirmation of propagation of generated sequence (SUBSCRIPTION side)

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 INT,
               col3 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE SUBSCRIPTION sub1 CONNECTION 'port=5432 dbname=postgres'
        PUBLICATION pub1 ;
NOTICE:  created replication slot "sub1" on publisher
CREATE SUBSCRIPTION
postgres=> SELECT * FROM data1 ;
 col1  | col2  |  col3
-------+-------+-------
   100 |   200 | data1
(1 row)
```

3.1.3. Optimizer Statistics Management

Functions to set and clear optimizer statistics for tables and columns are now provided. Options have been added to the pg_dump, pg_dumpall, and pg_upgrade commands to use this function to migrate statistics. See “3.4 Utilities” for additional command options.

□ Pg_restore_relation_stats function

The pg_restore_relation_stats function sets the optimizer statistics for a table. It returns false in case of minor errors. [d32d146]

Syntax 1 Pg_restore_relatoin_stats function

```
boolean pg_restore_relation_stats(VARIADIC kwargs "any")
```

Example 22 Execute pg_restore_relation_stats function

```
postgres=> SELECT * FROM pg_catalog.pg_restore_relation_stats(
        'version', '180000'::integer,
        'schemaname', 'public',
        'relname', 'data1',
        'relpages', '5406'::integer,
        'reltuples', '1e+06'::real,
        'relallvisible', '0'::integer,
        'relallfrozen', '0'::integer) ;
pg_restore_relation_stats
---------------------------
 t
(1 row)
```

□ Pg_clear_relation_stats function

The function pg_clear_relation_stats clears the optimizer statistics for the specified table. Executing this function resets the corresponding column in the pg_class catalog. [e839c8e, 650ab8a]

Syntax 2 Pg_clear_relation_stats function

```
void pg_clear_relation_stats(schemaname text, relname text)
```

Example 23 Execute pg_clear_relation_stats function

```
postgres=> SELECT pg_clear_relation_stats('public', 'data1') ;
 pg_clear_relation_stats
-------------------------


(1 row)

postgres=> SELECT relname, reltuples FROM pg_class WHERE relname='data1' ;
 relname | reltuples
---------+-----------
 data1   |        -1
(1 row)
```

□ Pg_restore_attribute_stats function

The function pg_restore_attribute_stats sets the column optimizer statistics. It returns false in case of minor errors. [d32d146]

Syntax 3 Pg_restore_attribute_stats function

```
boolean pg_restore_attribute_stats(VARIADIC kwargs "any")
```

Example 24 Execute pg_restore_attribute_stats function

```
postgres=> SELECT * FROM pg_catalog.pg_restore_attribute_stats(
        'version', '180000'::integer,
        'schemaname', 'public',
        'relname', 'data1',
        'attname', 'col1',
        'inherited', 'f'::boolean,
        'null_frac', '0'::real,
        'avg_width', '4'::integer,
        'n_distinct', '-1'::real) ;
 pg_restore_attribute_stats
----------------------------
 t
(1 row)
```

□ Pg_clear_attribute_stats function

The function pg_clear_attribute_stats clears the optimizer statistics for the specified column. Executing this function resets the corresponding column in the pg_stats catalog. [ce207d2, 650ab8a]

Syntax 4 Pg_clear_attribute_stats function

```
void pg_clear_attribute_stats(schemaname text, relname text, attname text,
inherited boolean)
```

Example 25 Execute pg_clear_attribute_stats function

```
postgres=> SELECT pg_clear_attribute_stats('public', 'data1', 'col1', true) ;
-[ RECORD 1 ]-------------+-
pg_clear_attribute_stats |
```

3.1.4. Roles and Permissions

The role pg_signal_autovacuum_worker has been added as a predefined role. This role allows signaling to the Autovacuum worker process. [ccd3802]

3.1.5. I/O Processing

PostgreSQL 18 adds large enhancements to the Storage I/O.

□ Streaming I/O coverage enhancement

Streaming I/O, introduced in PostgreSQL 17, is now more widely applicable. Streaming I/O is used in the following cases

- VACUUM statement for a table [9256822, c3e775e, 9256822, c3e775e]
- CREATE DATABASE STRATEGY=WAL_LOG statement [8720a15]
- VACUUM for B-Tree index [c5c239e]
- VACUUM for GiST index [69273b8]
- VACUUM for SP-GiST index [e215166]
- Amcheck module [043799f]
- Autoprewarm module [d9c7911]

□ Providing Asynchronous I/O

The infrastructure for using asynchronous I/O is now in place. The parameter io_method has been added to specify the method used for asynchronous I/O. The default value of this parameter is 'worker', which performs asynchronous I/O by an I/O worker process; the number of I/O worker processes started is determined by the parameter io_workers (default value 3). If using Linux's io_uring, specify io_uring for this parameter. [da72269, 55b454d, 247ce06, c325a76]

Example 26 Confirmation of I/O worker process

```
$ ps -ef | grep 'io worker' | grep -v grep
postgres  148065  148063  0 10:08 ?        00:00:00 postgres: io worker 0
postgres  148066  148063  0 10:08 ?        00:00:00 postgres: io worker 2
postgres  148067  148063  0 10:08 ?        00:00:00 postgres: io worker 1
```

Table 12 Parameter io_method setting value

Value	Description	Note
worker	Asynchronous I/O is executed by the io worker process.	Default
sync	Synchronous I/O is executed. This is the same behavior as in the previous version.	
io_uring	Use io_uring for asynchronous I/O.	

□ Prefetch support

Prefetching is now possible with PostgreSQL running on macOS. [6654bb9]

3.1.6. Locale Provider

The locale PG_UNICODE_FAST has been added to the builtin locale provider. This locale uses code point sort order and conforms to Unicode's full case mapping. However, conversions may cause changes in the number of characters, such as “ß” being lowercased to “ss”. [d3d0983, 286a365]

Example 27 Database cluster creation with PG_UNICODE_FAST locale specified

```
$ initdb --locale-provider=builtin --builtin-locale=PG_UNICODE_FAST
    --encoding=utf8 data

...
The database cluster will be initialized with this locale configuration:
  locale provider:   builtin
  default collation: PG_UNICODE_FAST
...
```

Example 28 Database creation with PG_UNICODE_FAST locale specified

```
postgres=# CREATE DATABASE fastuni1 LOCALE_PROVIDER builtin
        LOCALE 'PG_UNICODE_FAST' lc_collate 'C' LC_CTYPE 'C' ENCODING 'UTF8'
        TEMPLATE template0 ;
CREATE DATABASE
```

3.1.7. OAUTH SASL

OAUTHBearer/SASL can now be used for database user authentication. To use OAUTHBearer/SASL authentication, specify 'oauth' in the pg_hba.conf authentication method. To use this authentication, the client module uses the libcurl library, so a binary with --with-libcurl to the 'configure' command is required. The parameter oauth_validator_libraries specifies the validator libraries. If this parameter is not specified, authentication by the oauth authentication method will be failed. [b3f0be7]

3.1.8. Faster hash joins

Deform processing during hash join execution has been optimized. [adf97c1]

Example 29 Deforms during hash join

```
postgres=> EXPLAIN ANALYZE SELECT * FROM data1, data2
                WHERE data1.col1 = data2.col1 ;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=32789.00..76628.00 rows=1000000 width=20) …
   Hash Cond: (data2.col1 = data1.col1)
   ->  Seq Scan on data2  (cost=0.00..15440.00 rows=1000000 width=10) …
   ->  Hash  (cost=15406.00..15406.00 rows=1000000 width=10) …
         Buckets: 262144  Batches: 8  Memory Usage: 7179kB
         ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=10) …
 Planning Time: 0.069 ms
 JIT:
   Functions: 10
   Options: Inlining false, Optimization false, Expressions true, …
   Timing: Generation 0.728 ms (Deform 0.395 ms), Inlining 0.000 ms,
Optimization 0.250 ms, Emission 3.996 ms, Total 4.974 ms
 Execution Time: 396.935 ms
(12 rows)
```

3.1.9. Libpq

The following APIs have been added and enhanced.

□ Connection string

The minimum / maximum protocol version and SSL connection log file name have been added to the connection string. Corresponding environment variables have also been added. [285613c, 2da74d8]

Table 13 Added connect string

Connect string	Environment variables	Description
max_protocol_version	PGMAXPROTOCOLVERSION	Maximum version
min_protocol_version	PGMINPROTOCOLVERSION	Minimum version
sslkeylogfile	-	SSL key log file

□ Other API

The following APIs have been added. [4b99fed, cdb6b0f]

Table 14 Added API functions

API Name	Description
PQservice	Return service name from current connection
PQfullProtocolVersion	Return protocol version including minor version

3.1.10. Access Method

The following enhancements have been added to the index access methods. [56fead4, af4002b, ce62f2f, af4002b]

Table 15 Added Access Method Element

Element name	Description
amgettreeheight	Calculates the height of the index tree
amconsistentequality	Consistent equivalence semantics. Split old version of amcancrosscompare.
amconsistentequality	Has consistent ordering semantics. Split out old version of amcancrosscompare.
amcanhash	Support hashing

3.1.11. NUMA

The basic NUMA environment-aware API, SQL functions and pg_shmem_allocations_numa view have been added. The --with-libnuma option to the 'configure' command is required to enable this feature. [65c298f, 8cc139b]

Table 16 Added APIs

API names	Description
pg_numa_init	Initialize NUMA environment
pg_numa_get_max_node	Get maximum number of nodes
pg_numa_get_pagesize	Get page size
pg_numa_query_pages	Identify NUMA nodes in individual memory pages

Example 30 Execute pg_numa_available function

```
postgres=> SELECT pg_numa_available() ;
 pg_numa_available
-------------------
 t
(1 row)
```

3.1.12. Accelerated numerical calculations

Acceleration of the following basic calculations and CRC32C functions are implemented.

- Faster numeric multiplication [c4e4422, c4e4422, ca481d3]
- Faster division of numeric types [9428c00]
- Intel AVX-512 use of CRC32C functions [3c6e8c1]

3.1.13. Scope of Use of Unique Key Constraints

Previously, only B-Tree indexes could be used for foreign keys, materialized views, and partition keys, but now other index access methods can be used to create indexes as long as they are unique. [bfe21b7, 9d6db8b, f278e1f]

3.1.14. Hook

The following hooks for the EXPLAIN statement have been added. [fd02bf7, 50ba65e]

- Hook explore_per_plan_hook for EXPLAIN statement
- Hook explain_per_node_hook for per-node execution
- Hook for option validation, explore_validate_options_hook

3.2. SQL Statement

This section explains new features related to SQL statements.

3.2.1. ALTER DEFAULT PRIVILEGE

Default permissions can now be set for large objects. The defaclobjtype column in the pg_default_acl catalog will output the L indicating a large object. [0d6c477]

Example 31 Default privileges for LARGE OBJECT

```
postgres=> ALTER DEFAULT PRIVILEGES GRANT SELECT ON LARGE OBJECTS TO PUBLIC ;
ALTER DEFAULT PRIVILEGES
```

3.2.2. ALTER SUBSCRIPTION

The two_phase option can now be altered in the ALTER SUBSCRIPTION statement; the SUBSCRIPTION object must be disabled once in order to alter the two_phase option. [1462aad]

Example 32 Modify the two_phase option

```
postgres=# SELECT subname, subtwophasestate FROM pg_subscription ;
 subname | subtwophasestate
---------+------------------
 sub1    | e
(1 row)

postgres=# ALTER SUBSCRIPTION sub1 DISABLE ;
ALTER SUBSCRIPTION
postgres=# ALTER SUBSCRIPTION sub1 SET (two_phase=off) ;
ALTER SUBSCRIPTION
postgres=# SELECT subname, subtwophasestate FROM pg_subscription ;
 subname | subtwophasestate
---------+------------------
 sub1    | d
(1 row)
```

3.2.3. ALTER TABLE

The following syntax has been added to the ALTER TABLE statement.

□ ALTER TABLE ALTER CONSTRAINT

The SET INHERIT clause (or SET NO INHERIT clause) can be specified in the ALTER CONSTRAINT clause. [f4e53e1]

Example 33 Specifying the SET INHERIT clause

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 VARCHAR(10) NOT NULL) ;
CREATE TABLE
postgres=> ALTER TABLE data1 ALTER CONSTRAINT data1_col2_not_null NO INHERIT ;
ALTER TABLE
```

□ ALTER TABLE NOT NULL NOT VALID

The NOT VALID clause can be specified for NOT NULL constraints in an ALTER TABLE statement. [a379061]

Example 34 Specifying the NOT VALID clause

```
postgres=> ALTER TABLE data1 ADD CONSTRAINT nn_data1_col2 NOT NULL
               col2 NOT VALID ;
ALTER TABLE
postgres=> \d+ data1
                                                  Table "public.data1"
 Column |         Type          | Collation | Nullable | Default | Storage  …
--------+-----------------------+-----------+----------+---------+--------- …
 col1   | integer               |           | not null |         | plain    …
 col2   | character varying(10) |           | not null |         | extende  …
Indexes:
    "data1_pkey" PRIMARY KEY, btree (col1)
Not-null constraints:
    "data1_col1_not_null" NOT NULL "col1"
    "nn_data1_col2" NOT NULL "col2" NOT VALID
Access method: heap
```

□ ALTER TABLE ONLY DROP CONSTRAINT

ALTER TABLE DROP CONSTRAINT statements with ONLY clause for partitioned tables can now be executed. In previous versions, this would have resulted in an error. [4dea33c]

Example 35 Specifying the ONLY clause

```
postgres=> CREATE TABLE part1(c1 INT PRIMARY KEY, c2 INT CHECK(c2 > 0))
               PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES
               FROM (0) TO (1000000) ;
CREATE TABLE
postgres=> ALTER TABLE ONLY part1 DROP CONSTRAINT part1_c2_check ;
ALTER TABLE
```

3.2.4. ANALYZE

The following new features have been implemented for the ANALYZE statement.

□ ONLY clause

In previous versions, executing an ANALYZE statement on a partition table would recursively collect statistics for the partition. By specifying the ONLY clause, statistics can be collected only for partition tables. [62ddf7e]

Example 36 Specifying the ONLY clause

```
postgres=> ANALYZE VERBOSE ONLY part1 ;
INFO:  analyzing "public.part1" inheritance tree
INFO:  "part1v1": scanned 5406 of 5406 pages, containing 500000 live rows and
0 dead rows; 30000 rows in sample, 500000 estimated total rows
INFO:  finished analyzing table "postgres.public.part1"
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 5445 hits, 0 reads, 0 dirtied
WAL usage: 5 records, 0 full page images, 1195 bytes, 0 buffers full
system usage: CPU: user: 0.04 s, system: 0.00 s, elapsed: 0.04 s
ANALYZE
```

□ Output of I/O statistics and WAL statistics

The output of the ANALYZE VERBOSE statement now includes I/O and WAL statistics. The I/O statistics output is similar to the logs for the parameter log_autovacuum_min_duration setting. [4c1b4cd, bb77752]

Example 37 Output of the ANALYZE VERBOSE statement

```
postgres=> ANALYZE VERBOSE data1 ;
INFO:  analyzing "public.data1"
INFO:  "data1": scanned 5406 of 5406 pages, containing 333333 live rows and
666667 dead rows; 30000 rows in sample, 333333 estimated total rows
INFO:  finished analyzing table "postgres.public.data1"
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 5421 hits, 0 reads, 0 dirtied
WAL usage: 4 records, 0 full page images, 928 bytes, 0 buffers full
system usage: CPU: user: 0.02 s, system: 0.00 s, elapsed: 0.02 s
ANALYZE
```

Lines beginning with “INFO:” are I/O statistics and lines beginning with “WAL usage:” are WAL statistics.

□ Output of delay time

If the parameter track_cost_delay_timing is set to 'on', the ANALYZE VERBOSE statement will now output "delay time". The same information is also output in the log of automatic VACUUM. [7720082]

Example 38 Output of the ANALYZE VERBOSE statement

```
postgres=> ANALYZE VERBOSE data1 ;
INFO:  analyzing "public.data1"
INFO:  "data1": scanned 5406 of 5406 pages, containing 500000 live rows and 0
dead rows; 30000 rows in sample, 500000 estimated total rows
INFO:  finished analyzing table "postgres.public.data1"
delay time: 0.000 ms
I/O timings: read: 25.770 ms, write: 0.000 ms
...
WAL usage: 4 records, 1 full page images, 8240 bytes, 0 buffers full
system usage: CPU: user: 0.01 s, system: 0.03 s, elapsed: 0.05 s
ANALYZE
```

□ Output of WAL buffer full

The WAL buffer full count is now output when the ANALYZE VERBOSE statement is executed. [6a8a7ce]

Example 39 Output of the ANALYZE VERBOSE statement

```
postgres=> ANALYZE VERBOSE data1 ;
INFO:  analyzing "public.data1"
INFO:  "data1": scanned 5406 of 5406 pages, containing 500000 live rows and 0
dead rows; 30000 rows in sample, 500000 estimated total rows
INFO:  finished analyzing table "postgres.public.data1"
avg read rate: 0.000 MB/s, avg write rate: 0.300 MB/s
buffer usage: 5425 hits, 0 reads, 1 dirtied
WAL usage: 3 records, 1 full page images, 3544 bytes, 0 buffers full
system usage: CPU: user: 0.02 s, system: 0.00 s, elapsed: 0.02 s
```

3.2.5. COPY

The following features have been added to the COPY statement

□ COPY TO statement from a materialized view

The COPY TO statement can now specify the materialized view. [534874f]

Example 40 COPY TO statement for materialized view

```
postgres=> CREATE MATERIALIZED VIEW mview1 AS SELECT col1, col2 FROM data1 ;
SELECT 1000000
postgres=> COPY mview1 TO '/tmp/data1.csv' CSV ;
COPY 1000000
```

□ Addition of the reject_limit option

The COPY FROM statement now has a reject_limit option. This option allows the maximum number of errors to be specified when errors are ignored (on_error=ignore). If the number of errors exceeds the specified value, the COPY statement will fail. [4ac2a9b, a39297e]

Example 41 Specifying the REJECT_LIMIT option

```
postgres=> \! cat data1.csv
  1,data1
  2,data2
ABC,data3
  4,data4
DEF,data5
postgres=> COPY data1 FROM '/home/postgres/data1.csv' WITH (REJECT_LIMIT 1,
                ON_ERROR ignore , FORMAT csv, LOG_VERBOSITY verbose) ;
NOTICE:  skipping row due to data type incompatibility at line 3 for column
"col1": "ABC"
NOTICE:  skipping row due to data type incompatibility at line 5 for column
"col1": "DEF"
ERROR:  skipped more than REJECT_LIMIT (1) rows due to data type incompatibility
CONTEXT:  COPY data1, line 5, column col1: "DEF"
```

□ Added value for log_verbosity option

The log_verbosity option can now be 'silent' in the COPY statement. Specifying this value suppresses log output. [e7834a1

3.2.6. CREATE FOREIGN TABLE

The LIKE clause now allows copying column definitions from other tables (or foreign tables). [302cf157]

Example 42 Specifying the LIKE Clause for External Tables

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE FOREIGN TABLE data2(LIKE data1 EXCLUDING ALL) SERVER remsvr1 ;
CREATE FOREIGN TABLE
```

The values that can be specified in the INCLUDING / EXCLUDING clause are partially different from those in the CREATE TABLE statement.

Table 17 Values that can be specified in the INCLUDING / EXCLUDING clause

Clause	CREATE TABLE statement	CREATE FOREIGN TABLE statement
COMMENTS	○	○
COMPRESSION	○	-
CONSTRAINTS	○	○
DEFAULTS	○	○
GENERATED	○	○
IDENTITY	○	-
INDEXES	○	-
STATISTICS	○	○
STORAGE	○	-
ALL	○	○

3.2.7. CREATE INDEX

Creation of GIN indexes is now processed in parallel with B-Tree and BRIN indexes. [8492feb]

3.2.8. CREATE TABLE

The following enhancements have been implemented for the CREATE TABLE statement

□ WITHOUT OVERLAPS clause

It is now possible to specify a WITHOUT OVERLAPS clause that does not allow duplicate ranges for primary key constraints (PRIMARY KEY), unique constraints (UNIQUE) and foreign key constraints (FOREIGN KEY). At least two columns are required when specifying this clause for a primary key constraint. It can also be specified when adding a primary key in an ALTER TABLE statement. The column CONPERIOD has been added to the pg_constraint catalog to represent the WITHOUT OVERLAPS clause. The ON {UPDATE, DELETE} clause is not yet supported. [fc0438b, 89f908a]

Syntax 5 PRIMARY KEY clause

```
PRIMARY KEY (column_name1, …, column_nameN WITHOUT OVERLAPS)
```

Example 43 WITHOUT OVERLAPS clause in the primary key

```
postgres=> CREATE TABLE temporal_range1 (
    id int4range,
    valid_at daterange,
    CONSTRAINT temporal_rng_pk PRIMARY KEY (id, valid_at WITHOUT OVERLAPS)
) ;
CREATE TABLE
postgres=> \d temporal_range1
            Table "public.temporal_range1"
  Column  |   Type    | Collation | Nullable | Default
----------+-----------+-----------+----------+---------
 id       | int4range |           | not null |
 valid_at | daterange |           | not null |
Indexes:
    "temporal_rng_pk" PRIMARY KEY (id, valid_at WITHOUT OVERLAPS)
```

□ ENFORCED clause

Allows the ENFORCED or NOT ENFORCED clause to be specified for CHECK constraints and foreign key constraints. If nothing is specified or the ENFORCED clause is specified, PostgreSQL will behave as before, with CHECK constraints operating at the appropriate time. If the NOT ENFORCED clause is specified, the CHECK constraint will not work. The column CONFORCED has been added to the pg_constraint catalog to indicate this specification. [ca87c41, eec0040]

Example 44 CHECK constraints that do not operate

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY,
                col2 INT CHECK (col2 < 10) NOT ENFORCED) ;
CREATE TABLE
postgres=> INSERT INTO data1 VALUES (100, 100) ;
INSERT 0 1
postgres=> SELECT * FROM data1 ;
 col1  | col2
-------+-------
   100 |   100
(1 row)

postgres=> SELECT conenforced FROM pg_constraint
                WHERE conname='data1_col2_check' ;
 conenforced
-------------
 f
(1 row)
```

□ NOT VALID clause

Allows the NOT VALID clause to be specified as a foreign key for partition tables. [b663b94]

Example 45 NOT VALID clause for foreign keys in partition tables

```
postgres=> CREATE TABLE fkdata1(col1 INT PRIMARY KEY, col2 VARCHAR(10)) ;
CREATE TABLE
postgres=> CREATE TABLE part1(col1 INT PRIMARY KEY, col2 VARCHAR(10))
                PARTITION BY RANGE(col1) ;
CREATE TABLE
postgres=> ALTER TABLE part1 ADD FOREIGN KEY (col1) REFERENCES fkdata1
                NOT VALID ;
ALTER TABLE
```

□ GENERATED ALWAYS VIRTUAL clause

Allows the creation of auto-generated columns that do not hold the data. Specify a GENERATED ALWAYS VIRTUAL clause in the column definition. Previously, only GENERATED ALWAYS STORED clauses that retained actual data were supported. The VIRTUAL clause can be confirmed by the output of 'v' in the attgenerated column of the pg_attribute column. Indexes cannot be created on generated columns with VIRTUAL clause specified. [83ea6c5]

Example 46 GENERATED ALWAYS VIRTUAL clause

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 INT
               GENERATED ALWAYS AS (col1 * 2) VIRTUAL, col3 VARCHAR(10)) ;
CREATE TABLE
postgres=> \d data1
                                Table "public.data1"
 Column |         Type          | Collation | Nullable |           Default
--------+-----------------------+-----------+----------+------------------...
 col1   | integer               |           | not null |
 col2   | integer               |           |          | generated always as
(col1 * 2)
 col3   | character varying(10) |           |          |
Indexes:
    "data1_pkey" PRIMARY KEY, btree (col1)

postgres=> INSERT INTO data1(col1, col3) VALUES (100, 'data1') ;
INSERT 0 1
postgres=> SELECT * FROM data1 ;
 col1  | col2  |  col3
-------+-------+-------
   100 |   200 | data1
(1 row)
```

If the generated value overflows the column definition, it is not checked in the INSERT or UPDATE statement. The error occurs when the corresponding record is retrieved. CHECK constraints are checked when executing INSERT and UPDATE statements.

Example 47 Behavior in case of overflow

```
postgres=> CREATE TABLE overflow1(col1 VARCHAR(3), col2 VARCHAR(3)
                GENERATED ALWAYS AS (col1 || 'X')) ;
CREATE TABLE
postgres=> INSERT INTO overflow1(col1) VALUES ('ABC') ;
INSERT 0 1
postgres=> SELECT * FROM overflow1 ;
ERROR:  value too long for type character varying(3)
```

3.2.9. EXPLAIN

The following features have been added to the EXPLAIN statement.

□ Disabled Nodes Output

The output of the EXPLAIN statement now includes information on the number of disabled nodes (Disabled Nodes:). For example, disabling some execution plans by setting the parameter enable_seqscan to 'false', which was previously handled by increasing the cost, is now clearly recognized as a disabled node in PostgreSQL 18. [e222534, c01743a, 161320b]

Example 48 Disabling Nodes (PostgreSQL 18)

```
postgres=> SET enable_seqscan = off ;
SET
postgres=> EXPLAIN SELECT * FROM data1 ;
                            QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=10)
   Disabled: true
(2 rows)
```

Example 49 Disabling Nodes (PostgreSQL 17)

```
postgres=> SET enable_seqscan = off ;
SET
postgres=> EXPLAIN SELECT * FROM data1 ;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on data1  (cost=10000000000.00..10000015406.00 rows=1000000 width=10)
(1 row)
```

Example 50 src/backend/optimizer/path/costsize.c（PostgreSQL 17）

```
cost_seqscan(Path *path, PlannerInfo *root,
                         RelOptInfo *baserel, ParamPathInfo *param_info)
{
...
        if (!enable_seqscan)
                startup_cost += disable_cost;
...
```

□ Storage Clause Output

Temporary storage information used during Window Aggregation and CTE Scan runs is now output as a Storage clause. [95d6e9a, 40708ac]

Example 51 Windows Aggregation temporary storage information

```
postgres=> EXPLAIN (ANALYZE, COSTS OFF) SELECT COUNT(*) OVER() FROM data1 ;
                                QUERY PLAN
---------------------------------------------------------------------------
 WindowAgg (actual time=130.172..189.102 rows=1000000 loops=1)
   Storage: Disk  Maximum Storage: 9766kB
   ->  Seq Scan on data1 (actual time=0.008..38.947 rows=1000000 loops=1)
 Planning Time: 0.037 ms
 Execution Time: 206.478 ms
(5 rows)
```

Example 52 CTE Scan Temporary Storage Information

```
postgres=> EXPLAIN (ANALYZE, COSTS OFF) WITH cte1 AS MATERIALIZED (SELECT col1,
col2 FROM data1) SELECT * FROM cte1 WHERE col1 < 10000 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 CTE Scan on cte1 (actual time=0.012..115.005 rows=9999 loops=1)
   Filter: (col1 < 10000)
   Rows Removed by Filter: 990001
   Storage: Disk  Maximum Storage: 19528kB
   CTE cte1
     ->  Seq Scan on data1 (actual time=0.009..34.816 rows=1000000 loops=1)
 Planning Time: 0.064 ms
 Execution Time: 116.189 ms
(8 rows)
```

□ Parallel Bitmap Heap Scan Statistics

Statistics for Parallel Bitmap Heap Scan execution are now output for each worker node. [5a1e6df]

Example 53 Parallel Bitmap Heap Scan Detailed Information

```
postgres=> EXPLAIN (COSTS OFF, ANALYZE, VERBOSE) SELECT AVG(col1) FROM data1
                WHERE col1 BETWEEN 1000000 AND 2000000 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Finalize Aggregate (actual time=98.946..99.991 rows=1 loops=1)
   Output: avg(col1)
   Buffers: shared hit=8125 read=16
   ->  Gather (actual time=98.801..99.981 rows=3 loops=1)
…
               ->  Parallel Bitmap Heap Scan on public.data1 (actual time=46…
                     Output: col1, col2
                     Recheck Cond: ((data1.col1 >= 1000000) AND (data1.col1 …
                     Heap Blocks: exact=1598
                     Buffers: shared hit=8125 read=16
…
```

□ Record Count Output Format

Record counts (actual rows) are now output to two decimal places. [ddb17e3, 95dbd827]

Example 54 Loop processing line count output

```
postgres=> EXPLAIN (ANALYZE, VERBOSE) SELECT * FROM data1 WHERE col1 < 1000 ;
                                QUERY PLAN
-------------------------------------------------------------------------------
Index Scan using data1_pkey on public.data1   (cost=0.42..39.77 rows=1048
width=10) (actual time=0.008..0.084 rows=999.00 loops=1)
   Output: col1, col2
   Index Cond: (data1.col1 < 1000)
   Buffers: shared hit=11
 Planning:
   Buffers: shared hit=4
 Planning Time: 0.096 ms
 Execution Time: 0.115 ms
(8 rows)
```

□ Index Searches

The “Index Searches” item indicating the number of index searches is now output. [0fbceae]

Example 55 Index Search Count

```
postgres=> EXPLAIN (ANALYZE, COSTS OFF) SELECT COUNT(*) FROM data1 WHERE
               col1 IN (100, 1000, 10000) ;
                                        QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate (actual time=0.021..0.021 rows=1.00 loops=1)
   Buffers: shared hit=10
   ->  Index Only Scan using data1_pkey on data1 (actual time=0.015..0.019 …
         Index Cond: (col1 = ANY ('{100,1000,10000}'::integer[]))
         Heap Fetches: 0
         Index Searches: 3
…
```

□ Window Function Output

The “Window:” clause is now output in the execution plan when using Window functions. [8b1b342]

Example 56 Window function output

```
postgres=> EXPLAIN SELECT AVG(col1) OVER (PARTITION BY col2) FROM data1 ;
                                QUERY PLAN
-------------------------------------------------------------------------
 WindowAgg  (cost=130488.33..146238.34 rows=1000000 width=36)
   Window: w1 AS (PARTITION BY col2)
   ->  Sort  (cost=128738.34..131238.34 rows=1000000 width=8)
         Sort Key: col2
         ->  Seq Scan on data1  (cost=0.00..15406.00 rows=1000000 width=8)
(5 rows)
```

3.2.10. INSERT / UPDATE / DELETE / MERGE

The RETURNING clause of the update DML statement can now specify an OLD clause for pre-update information and a NEW clause for post-update information. The pre-update data (OLD) in the INSERT statement and the post-update data (NEW) in the DELETE statement will be NULL. [80feb72]

Syntax 6 RETURNING Clause

```
RETURNING OLD.column1, NEW.column2
RETURNING OLD.*, NEW.*
RETURNING WITH (OLD AS alias1, NEW AS alias2) alias1.column1, …
```

Example 57 Output of old and new data in RETURNING clause

```
postgres=> UPDATE data1 SET col2='after' WHERE col1=100
                RETURNING WITH (OLD AS o, NEW AS n) o.col1, o.col2, n.col2 ;
 col1  |   col2   |  col2
-------+----------+--------
   100 | before   | after
(1 row)

postgres=> \pset null
Null display is "null".

postgres=> DELETE FROM data1 WHERE col1=200 RETURNING OLD.*, NEW.* ;
 col1  |   col2   |  col1  |  col2
-------+----------+--------+--------
 200   | before   | null   | null
(1 row)
```

3.2.11. VACUUM

The following new features have been added to the VACUUM statement.

□ ONLY clause

ONLY clause can now be specified for partitioned tables. No VACUUM processing is performed. [62ddf7e]

Example 58 Execute the VACUUM ONLY statement

```
postgres=> VACUUM VERBOSE ONLY part1 ;
WARNING:  VACUUM ONLY of partitioned table "part1" has no effect
VACUUM
```

□ VERBOSE / Log output

When 'on' is specified for the parameter track_io_timing, 'delay time' is now output in the ANALYZE VERBOSE statement. The same information is also output to the log of automatic VACUUM. [7720082]

Example 59 Execute the VACUUM VERBOSE statement

```
postgres=> VACUUM VERBOSE data1 ;
INFO:  vacuuming "postgres.public.data1"
INFO:  finished vacuuming "postgres.public.data1": index scans: 0
pages: 0 removed, 5406 remain, 1 scanned (0.02% of total), 0 eagerly scanned
tuples: 0 removed, 500000 remain, 0 are dead but not yet removable
removable cutoff: 786, which was 0 XIDs old when operation ended
frozen: 0 pages from table (0.00% of total) had 0 tuples frozen
visibility map: 0 pages set all-visible, 0 pages set all-frozen (0 were all-visible)
index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed
delay time: 0.000 ms
I/O timings: read: 0.000 ms, write: 0.000 ms
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 13 hits, 0 reads, 0 dirtied
WAL usage: 0 records, 0 full page images, 0 bytes
system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
VACUUM
```

□ Output WAL buffer full

The WAL buffer full count is now output when the VACUUM VERBOSE statement is executed. [6a8a7ce]

Example 60 Execute the VACUUM VERBOSE statement

```
postgres=> VACUUM VERBOSE data1 ;
INFO:  vacuuming "postgres.public.data1"
INFO:  finished vacuuming "postgres.public.data1": index scans: 0
pages: 0 removed, 5406 remain, 1 scanned (0.02% of total), 0 eagerly scanned
tuples: 0 removed, 500000 remain, 0 are dead but not yet removable
removable cutoff: 763, which was 0 XIDs old when operation ended
frozen: 0 pages from table (0.00% of total) had 0 tuples frozen
visibility map: 0 pages set all-visible, 0 pages set all-frozen (0 were all-visible)
index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 13 hits, 0 reads, 0 dirtied
WAL usage: 0 records, 0 full page images, 0 bytes, 0 buffers full
system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
VACUUM
```

□ Release of end-of-table free block

The parameter vacuum_truncate has been added. This parameter determines the behavior when the table attribute vacuum_truncate or the TRUNCATE option of the VACUUM statement is not set. [0164a0f]

Example 61 Setting the vacuum_truncate parameter

```
postgres=> SET vacuum_truncate = off ;
SET
postgres=> VACUUM data1 ;
VACUUM
```

3.2.12. Data Types

The following enhancements have been implemented for PostgreSQL 18 data types.

□ Bytea and integer type casts

Bytea and integer type (int2, int4, int8) data can now be cast to each other. [6da469b]

Example 62 Bytea and integer type casts

```
postgres=> SELECT '\x1122334455667788'::bytea::int8 ;
        int8
---------------------
 1234605616436508552
(1 row)

postgres=> SELECT 1234567890::int8::bytea ;
       bytea
--------------------
 \x00000000499602d2
(1 row)
```

3.2.13. Functions

The following functions have been added/enhanced.

□ Array_reverse function

The function array_reverse has been added, which returns an array with the first dimension of the array inverted. [49d6c7d]

Syntax 7 Array_reverse function

```
anyarray array_reverse(anyarray)
```

Example 63 Execution of the array_reverse function

```
postgres=> SELECT array_reverse(ARRAY['AB', 'CD', 'EF']) ;
-[ RECORD 1 ]-+----------
array_reverse | {EF,CD,AB}

postgres=> SELECT array_reverse(ARRAY[[1,2],[3,4],[5,6]]) ;
-[ RECORD 1 ]-+--------------------
array_reverse | {{5,6},{3,4},{1,2}}
```

□ Casefold function

The casefold function is almost identical to the lower function, but it performs lower case conversion according to collation. Its main purpose is to facilitate case-insensitive string comparisons. It can only be used when the database encoding is UTF-8. Some characters may change the character length. If the locale PG_UNICODE_FAST is used, ß (U+00DF) will be converted to 'ss'. [bfc5992]

Example 64 If the database collation is PG_C_UTF8

```
postgres=> SELECT casefold('Σ') ;
 casefold
----------
 σ
(1 row)
```

Example 65 If the database collation is PG_UNICODE_FAST

```
postgres=> SELECT casefold('ß') ;
 casefold
----------
 ss
(1 row)
```

□ Crc32 function

The functions crc32 to calculate CRC-32 and crc32c to calculate CRC-32C have been added. [760162f]

Syntax 8 Crc32 / Crc32c functions

```
bigint crc32(bytea)
bigint crc32c(bytes)
```

Example 66 Execution of the crc32 and crc32c functions

```
postgres=> SELECT crc32('ABC'::bytea) ;
   crc32
------------
 2743272264
(1 row)

postgres=> SELECT crc32c('ABC'::bytea) ;
   crc32c
------------
 2285480319
(1 row)
```

□ Extract function

The WEEK clause can now be specified for INTERVAL types. [6be39d7]

Example 67 Execution of the extract function

```
postgres=> SELECT EXTRACT(WEEK FROM INTERVAL '13 days 24 hours') ;
 extract
---------
       1
```

□ Gamma / lgamma function

The functions gamma and lgamma have been added to calculate gamma and log gamma values. [a3b6dfd]

Syntax 9 Gamma / Lgamma functions

```
double precision gamma(double precision)
double precision lgamma(double precision)
```

Example 68 Execution of the gamma / lgamma function

```
postgres=> SELECT gamma(2.5), lgamma(3.5) ;
       gamma        |       lgamma
--------------------+--------------------
 1.329340388179137 | 1.2009736023470743
(1 row)
```

□ Has_largeobject_privilege function

The has_largeobject_privilege function has been added to check for the existence of privileges on large objects. [4eada20]

Syntax 10 Has_largeobject_privilege function

```
boolean has_largeobject_privilege(name, oid, text)
boolean has_largeobject_privilege(oid, oid, text)
boolean has_largeobject_privilege(oid, text)
```

Example 69 Execution of the has_largeobject_privilege function

```
postgres=> SELECT lo_create(1001) ;
-[ RECORD 1 ]---
lo_create | 1001

postgres=> SELECT has_largeobject_privilege(1001, 'SELECT') ;
-[ RECORD 1 ]-------------+--
has_largeobject_privilege | t
```

□ Json[b]_strip_nulls function

The json[b]_strip_nulls function now has a parameter strip_in_arrays to remove NULL values from an array. The default value is FALSE. [4603903]

Syntax 11 Json_strip_nulls / Jsonb_strip_nulls functions

```
json json_strip_nulls(target json, strip_in_arrays boolean DEFAULT false)
jsonb jsonb_strip_nulls(target jsonb, strip_in_arrays boolean DEFAULT false)
```

Example 70 Execution of the json_strip_nulls function

```
postgres=> SELECT json_strip_nulls('[1,2,null,4]', true) ;
-[ RECORD 1 ]----+--------
json_strip_nulls | [1,2,4]
```

□ Max / min function

Max and min functions can now be used for values of type bytea. [2d24fd9]

Syntax 12 Max / min functions

```
bytea max(bytea)
bytea min(bytea)
```

Example 71 Execution of the max / min function for bytea type

```
postgres=> CREATE TABLE bytea1(col1 bytea) ;
CREATE TABLE
postgres=> INSERT INTO bytea1 VALUES
                (decode('abcdef', 'hex')), (decode('123456', 'hex')) ;
INSERT 0 2
postgres=> SELECT max(col1), min(col1) FROM bytea1 ;
   max    |   min
----------+---------
 \xabcdef | \x123456
(1 row)
```

□ Reverse function

The version corresponding to bytea type has been added. [0697b23]

Syntax 13 Reverse function

```
bytea reverse(bytea)
```

Example 72 Execution of the reverse function

```
postgres=> SELECT reverse('\xabcdef12'::bytea) ;
  reverse
------------
 \x12efcdab
(1 row)
```

□ Pg_get_acl function

The function pg_get_acl has been added to retrieve the ACL settings of a database object. [4564f1c, d898665]

Syntax 14 Pg_get_acl function

```
aclitem[] pg_get_acl(classid oid, objid oid, objsubid integer)
```

Example 73 Execution of the pg_get_acl function

```
postgres=> CREATE TABLE data1(col1 INT PRIMARY KEY, col2 VARCHAR(10)) ;
CREATE TABLE
postgres=> GRANT SELECT, MAINTAIN ON data1 TO user1 ;
GRANT
postgres=> SELECT UNNEST(
  pg_get_acl('pg_class'::regclass, 'data1'::regclass::oid, 0)) ;
       unnest
--------------------
 demo=arwdDxtm/demo
 user1=rm/demo
(2 rows)
```

□ Pg_get_sequence_data function

The function pg_get_sequence_data, which returns a tuple of sequences, has been added. This function returns a NULL value if a sequence with no SELECT privilege is specified. This function is intended primarily for use in optimizing pg_dump commands. [c8b06bb, bd15b7d, a83a944]

Syntax 15 Pg_get_sequence_data function

```
record pg_get_sequence_data(sequence_oid regclass)
```

Example 74 Execution of the pg_get_sequence_data function

```
postgres=> CREATE SEQUENCE seq1 START WITH 100 ;
CREATE SEQUENCE
postgres=> SELECT NEXTVAL('seq1') ;
 nextval
---------
     100
(1 row)

postgres=> SELECT * FROM pg_get_sequence_data('seq1') ;
 last_value | is_called
------------+-----------
        100 | t
(1 row)
```

□ Pg_get_process_memory_contexts function

The function pg_get_process_memory_contexts has been added to retrieve memory context information for the specified process. [042a662]

Syntax 16 Pg_get_process_memory_context function

```
record pg_get_process_memory_contexts(pid integer, summary boolean, timeout
double precision)
```

Example 75 Execution of the pg_get_process_memory_contexts function

```
postgres=# SELECT name, total_bytes FROM
 pg_get_process_memory_contexts(pg_backend_pid(), false, 0) ORDER BY path ;
                      name                       | total_bytes
-------------------------------------------------+-------------
 TopMemoryContext                                |      140496
 Map from relid to OID of cached composite type |        8192
 Type information cache                          |       24624
 Record information cache                        |        8192
...
```

□ Pg_ls_summariesdir function

The function pg_ls_summariesdir has been added to PostgreSQL 18. This function provides a list of WAL summary files. The function can be executed by a user with the SUPERUSER attribute or a user authorized by the pg_maintain role. If the WAL summarize feature is disabled, no records are returned. [4e1fad3]

Syntax 17 Pg_ls_summariesdir function

```
SETOF record pg_ls_summariesdir()
```

Table 18 Column name and description of the return value of the pg_ls_summariesdir function

Column name	Data type	Description
name	text	WAL Summary File Name
size	bigint	Bytes in file
modification	timestamp with time zone	Last Modified

Example 76 Execution of the pg_ls_summariesdir function

```
postgres=# SELECT * FROM pg_ls_summariesdir() ;
                      name                       | size  |      modification
-------------...--------------------------------+-------+------------------------
 00000001…100002800000000010B2FC8.summary |  4808 | 2025-05-09 15:42:38+09
 00000001…10B2FC800000000015416F8.summary |  4994 | 2025-05-09 15:42:38+09
 00000001…15416F800000000015417F8.summary |    56 | 2025-05-09 15:42:38+09
 00000001…15417F800000000015481A0.summary |   230 | 2025-05-09 15:42:38+09
 00000001…15481A000000000015482A0.summary |    56 | 2025-05-09 15:42:38+09
(5 rows)
```

□ Pg_stat_get_backend_wal function

The function pg_stat_get_backend_wal has been added to retrieve WAL output information for a specified backend process. The result of executing this function is similar to the pg_stat_wal view, but on a per-backend-process basis. [76def4c]

Syntax 18 Pg_stats_get_backend_wal function

```
record pg_stat_get_backend_wal(backend_pid integer)
```

Table 19 Column name and description of the return value of the pg_stat_get_backend_wal function

Column name	Data type	Description
wal_records	bigint	Number of WAL records
wal_fpi	bigint	Number of WAL full page images
wal_bytes	numeric	Number of WAL bytes
wal_buffers_full	bigint	Number of storage outputs due to WAL buffer full
stats_reset	timestamp with time zone	Data reset time

Example 77 Execution of the pg_stat_get_backend_wal function

```
postgres=> SELECT * FROM pg_stat_get_backend_wal(pg_backend_pid()) ;
 wal_records | wal_fpi | wal_bytes | wal_buffers_full | stats_reset
-------------+---------+-----------+------------------+-------------
     4005580 |      28 | 262315709 |            28954 |
(1 row)
```

□ Regular Expression Function

Argument names have been added to the regular expression function (regexp*). [580f872]

Example 78 Argument name of the regexp_count function

```
postgres=> \df regexp_count
                                        List of functions
   Schema   |     Name     | Result data type |                  Argument data
------------+--------------+------------------+-------------------------------
 pg_catalog | regexp_count | integer          | string text, pattern text   …
 pg_catalog | regexp_count | integer          | string text, pattern text,  …
 pg_catalog | regexp_count | integer          | string text, pattern text,  …
(3 rows)
```

□ To_number function and to_char function

The to_number function and to_char function now accept an RN indicating a Roman numeral as the format specification character. The case of the input string and the format specification character is ignored. [172e6b3]

Example 79 Specify Roman numerals in the to_number function

```
postgres=> SELECT to_number('MMXXV', 'rn'), to_char(1560, 'RN') ;
 to_number |     to_char
-----------+-----------------
      2025 |            MDLX
(1 row)
```

□ Uuidv7／uuidv4 functions

Uuidv7 function supporting UUID v7 and uuidv4 function supporting UUID v4 are added. uuidv7 is expected to suppress randomization of index storage pages by specifying the timestamp of function execution in the first part of uuidv7. The uuidv7 function is expected to suppress randomization of index storage pages by specifying a timestamp at the beginning of the function. [78c5e14]

Example 80 Execution of the uuidv7 function and the uuidv4 function

```
postgres=> SELECT uuidv7() FROM generate_series(1, 5) ;
                uuidv7
--------------------------------------
 0193b90c-0818-772e-8fe2-ea74b2446cb2
 0193b90c-0818-775d-b20b-4551f631cd81
 0193b90c-0818-7761-a496-972341cb066c
 0193b90c-0818-7763-b66a-511858d491d7
 0193b90c-0818-7766-b1b9-7a5c2d05ee74
(5 rows)

postgres=> SELECT uuidv4() ;
                uuidv4
--------------------------------------
 9aaf5cc9-893e-46da-9690-dda84405ffe8
(1 row)
```

3.2.14. PL/pgSQL

The operator "=>" can now be used to specify parameters for named cursors. Previously, the operator ":=" was used. [246dedc]

Example 81 Named cursor parameter specification

```
CREATE FUNCTION func1(INTEGER) RETURNS TEXT AS $$
DECLARE
  cur1 CURSOR (v INTEGER) FOR SELECT col2 FROM data1 WHERE col1=v;
  val1 TEXT;
BEGIN
  OPEN cur1 (v => $1);
  FETCH cur1 INTO val1;
  RETURN val1;
END;
$$ LANGUAGE plpgsql;
```

3.2.15. Optimizer

Faster execution plans are now selected.

□ ANY conversion of OR conditions

OR conditions for indexed columns are now automatically converted to ANY clauses. [d4378c0, ae45691]

Example 82 Execution Plan of PostgreSQL 17

```
postgres=> EXPLAIN SELECT * FROM data1 WHERE col1=1000 OR col1=2000 ;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Bitmap Heap Scan on data1  (cost=8.87..16.78 rows=2 width=10)
   Recheck Cond: ((col1 = 1000) OR (col1 = 2000))
   ->  BitmapOr  (cost=8.87..8.87 rows=2 width=0)
         ->  Bitmap Index Scan on data1_pkey  (cost=0.00..4.43 rows=1 width=0)
               Index Cond: (col1 = 1000)
         ->  Bitmap Index Scan on data1_pkey  (cost=0.00..4.43 rows=1 width=0)
               Index Cond: (col1 = 2000)
(7 rows)
```

Example 83 Execution Plan of PostgreSQL 18

```
postgres=> EXPLAIN SELECT * FROM data1 WHERE col1=1000 OR col1=2000 ;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using data1_pkey on data1  (cost=0.42..12.88 rows=2 width=10)
   Index Cond: (col1 = ANY ('{1000,2000}'::integer[]))
(2 rows)
```

□ ANY Conversion of Join OR Conditions

When an OR is used in a join condition, it can now be converted to an ANY clause. [627d634]

Example 84 Execution Plan of PostgreSQL 17

```
postgres=> EXPLAIN (COSTS OFF) SELECT COUNT(*) FROM data1 AS d1, data2 AS d2
                WHERE d1.col1 = d2.col1 OR d1.col1 = d2.col2 ;
                              QUERY PLAN
-----------------------------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Workers Planned: 2
         ->  Partial Aggregate
               ->  Nested Loop
                     ->  Parallel Seq Scan on data1 d1
                     ->  Bitmap Heap Scan on data2 d2
                           Recheck Cond: ((d1.col1 = col1) OR (d1.col1 = col2))
                           ->  BitmapOr
                                 ->  Bitmap Index Scan on data2_pkey
                                       Index Cond: (col1 = d1.col1)
                                 ->  Bitmap Index Scan on idx1_data2
                                       Index Cond: (co;2 = d1.col1)
(13 rows)
```

Example 85 Execution Plan of PostgreSQL 18

```
postgres=> EXPLAIN (COSTS OFF) SELECT COUNT(*) FROM data1 AS d1, data2 AS d2
                WHERE d1.col1 = d2.col1 OR d1.col2 = d2.col2 ;
                              QUERY PLAN
-----------------------------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Workers Planned: 2
         ->  Partial Aggregate
               ->  Nested Loop
                     ->  Parallel Seq Scan on data2 d2
                     ->  Index Only Scan using data1_pkey on data1 d1
                           Index Cond: (col1 = ANY (ARRAY[d2.col1, d2.col2]))
(8 rows)
```

□ IN (VALUES) Clause Converted to ANY Clause

The WHERE clause with IN (VALUES) is converted to an ANY clause. [c0962a1]

Example 86 Execution Plan of PostgreSQL 17

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1
                WHERE col1 IN (VALUES(100), (200)) ;
                  QUERY PLAN
------------------------------------------
 Nested Loop
   ->  Unique
         ->  Sort
               Sort Key: "*VALUES*".column1
               ->  Values Scan on "*VALUES*"
   ->  Index Scan using data1_pkey on data1
         Index Cond: (col1 = "*VALUES*".column1)
(7 rows)
```

Example 87 Execution Plan of PostgreSQL 18

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1
                WHERE col1 IN (VALUES(100), (200)) ;
                    QUERY PLAN
-----------------------------------------------
 Index Scan using data1_pkey on data1
   Index Cond: (col1 = ANY ('{100,200}'::integer[]))
(2 rows)
```

□ Right Semi Join

Right Semi Join is now available as the execution plan. It may be adopted for hash joins or merge joins. [aa86129e]

Example 88 Adoption of Right Semi Join

```
postgres=> EXPLAIN (COSTS OFF, VERBOSE)
  SELECT COUNT(*) FROM tenk1 a WHERE (unique1, two) IN
    (SELECT unique1, ROW_NUMBER() OVER() FROM tenk1 b) ;
                                    QUERY PLAN
------------------------------------------------------------------------------
 Aggregate
   Output: count(*)
   ->  Hash Right Semi Join
         Hash Cond: ((b.unique1 = a.unique1) AND ((row_number() OVER (?))…
         ->  WindowAgg
               Output: b.unique1, row_number() OVER (?)
               ->  Seq Scan on public.tenk1 b
                     Output: b.unique1
         ->  Hash
               Output: a.unique1, a.two
               ->  Seq Scan on public.tenk1 a
                     Output: a.unique1, a.two
(12 rows)
```

□ UNIQUE Constraints and GROUP BY

When a column specified by a NOT NULL constraint and a UNIQUE index is included in a GROUP BY clause, non-target columns are now excluded from GROUP BY. The following example shows a col1 column with UNIQUE and NOT NULL constraints specified. [bd10ec5]

Example 89 Execution Plan of PostgreSQL 17

```
postgres=> EXPLAIN (COSTS OFF) SELECT col1, col2 FROM data1
                GROUP BY col1, col2 ;
       QUERY PLAN
------------------------
 HashAggregate
   Group Key: col1, col2
   ->  Seq Scan on data1
(3 rows)
```

Example 90 Execution Plan of PostgreSQL 18

```
postgres=> EXPLAIN (COSTS OFF) SELECT col1, col2 FROM data1
                GROUP BY col1, col2 ;
       QUERY PLAN
--------------------------
 HashAggregate
   Group Key: col1
   ->  Seq Scan on data1
(3 rows)
```

□ Optimization of the DISTINCT clause

Since the order of columns specified in the DISTINCT clause is not meaningful, optimization has been implemented to reorder them as needed. A new parameter, enable_distinct_reordering, has been added to change this behavior. The default value of this parameter is 'on', which causes the optimization to occur; to revert to PostgreSQL 17 behavior, set this parameter to 'off'. In the example below, column col1 is modified in the execution plan because it is a primary key, and the processing of column col2 is disabled. [a8ccf4e]

Example 91 Execution Plan of PostgreSQL 17

```
postgres=> EXPLAIN (COSTS OFF) SELECT DISTINCT col2, col1 FROM data1 ;
       QUERY PLAN
--------------------------
 HashAggregate
   Group Key: col2, col1
   ->  Seq Scan on data1
(3 rows)
```

Example 92 Execution Plan of PostgreSQL 18

```
postgres=> EXPLAIN (COSTS OFF) SELECT DISTINCT col2, col1 FROM data1 ;
                    QUERY PLAN
-----------------------------------------------
 Unique
   ->  Incremental Sort
         Sort Key: col1, col2
         Presorted Key: col1
         ->  Index Scan using data1_pkey on data1
(5 rows)
```

□ Self-join optimization

The optimization of replacing self-join operations with unjoin scans when the execution result is unaffected has been implemented. This behavior is controlled by the parameter enable_self_join_elimination. The default value is 'on' and the optimization is executed. In the following example, column col1 of table data1 is the primary key. [fc069a3]

Example 93 Execution Plan of PostgreSQL 17

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1 d1, data1 d2 WHERE
                d1.col1 = d2.col1 ;
            QUERY PLAN
--------------------------------
 Hash Join
   Hash Cond: (d1.col1 = d2.col1)
   ->  Seq Scan on data1 d1
   ->  Hash
         ->  Seq Scan on data1 d2
(5 rows)
```

Example 94 Execution Plan of PostgreSQL 18

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1 d1, data1 d2 WHERE
                d1.col1 = d2.col1 ;
      QUERY PLAN
----------------------
 Seq Scan on data1 d2
(1 row)
```

□ Index Skip Scan

For an index consisting of multiple columns, if a match condition other than the first column is specified, the execution plan that uses the corresponding index is now created. In the following example, a composite index is created for columns col3 and col2, in that order.

Example 95 Execution Plan of PostgreSQL 17

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1 d1, data1 d2 WHERE
                d1.col1 = d2.col1 ;
            QUERY PLAN
---------------------------------
 Hash Join
   Hash Cond: (d1.col1 = d2.col1)
   ->  Seq Scan on data1 d1
   ->  Hash
         ->  Seq Scan on data1 d2
(5 rows)
```

Example 96 Execution Plan of PostgreSQL 18

```
postgres=> EXPLAIN (COSTS OFF) SELECT * FROM data1 d1, data1 d2 WHERE
                d1.col1 = d2.col1 ;
      QUERY PLAN
----------------------
 Seq Scan on data1 d2
(1 row)
```

□ Transformation of HAVING clauses into WHERE clauses

The optimization of transforming some HAVING clauses used in GROUPING SETS clauses into WHERE clauses has been implemented. [67a54b9]

Example 97 Execution Plan of PostgreSQL 17

```
postgres=> EXPLAIN (COSTS OFF) SELECT c1, c2, COUNT(c1) FROM group1
        GROUP BY GROUPING SETS ((c1, c2), (c1)) HAVING c1 > 1000 AND c2 > 1000 ;
               QUERY PLAN
----------------------------------------
 GroupAggregate
   Group Key: c1, c2
   Group Key: c1
   Filter: ((c1 > 1000) AND (c2 > 1000))
   ->  Sort
         Sort Key: c1, c2
         ->  Seq Scan on group1
(7 rows)
```

Example 98 Execution Plan of PostgreSQL 18

```
postgres=> EXPLAIN (COSTS OFF) SELECT c1, c2, COUNT(c1) FROM group1
        GROUP BY GROUPING SETS ((c1, c2), (c1)) HAVING c1 > 1000 AND c2 > 1000 ;
            QUERY PLAN
-----------------------------------
 GroupAggregate
   Group Key: c1, c2
   Group Key: c1
   Filter: (c2 > 1000)
   ->  Sort
         Sort Key: c1, c2
         ->  Seq Scan on group1
               Filter: (c1 > 1000)
(8 rows)
```

3.3. Configuration parameters

In PostgreSQL 18, the following parameters have been changed.

3.3.1. Added parameters

The following parameters have been added. [0dcaea5, db6a4a9, c758119, a8ccf4e, 306dc52, 052026c, bb8dff9, 45188c2, 3d1ef3a, fc069a3, ac0e331, b3f0be7, 6c349d8, 6d376c3, 0284401, 4f7f7b0, 10f6646, 0164a0f, 247ce06, 55b454d, 62d712e, 9fbd53d, f78ca6f]

Table 20 Added parameters

Parameter name	Description (context)	Default value
autovacuum_worker_slots	Maximum number of autovacuum_max_workers settings (postmaster)	16
autovacuum_vacuum_max_threshold	Maximum number of tuples required to trigger VACUUM on a table (sighup)	100000000
enable_distinct_reordering	Whether to use column reordering optimization specified in the DISTINCT clause (user)	on
enable_self_join_elimination	Enable self-join optimization feature (user)	on
extension_control_path	Extended control file path (superuser)	$system
file_copy_method	Specify how files are copied (user)	copy
idle_replication_slot_timeout	Number of idle minutes to disable replication slots (sighup)	0
io_max_combine_limit	Maximum value for io_combine_limit (postmaster)	16
io_max_concurrency	Maximum number of simultaneous I/Os a single process can perform (postmaster)	64
io_method	Asynchronous I/O execution method (postmaster)	worker
io_workers	Number of IO worker processes （sighup）	3
log_lock_failure	Log output when lock fails (superuser)	off
max_active_replication_origins	Maximum number of replication origins (postmaster)	10

Parameter name	Description (context)	Default value
md5_password_warnings	Output warning when generating MD5 passwords (user)	on
num_os_semaphores	Number of semaphores required to run the server (internal)	計算値
oauth_validator_libraries	Library used to validate OAuth connection tokens (sighup)	"
vaccum_truncate	Whether to release empty end-of-table pages (user)	on
ssl_groups	Curve name used for EDCH key exchange (sighup)	X25519:prime256v1
ssl_tls13_ciphers	List of cipher suites that can be used for connections with TLS version 1.3 (sighup)	"
track_cost_delay_timing	Cost-based VACUUM delay tracking (superuser)	off
vacuum_max_eager_freeze_failure_rate	Percentage of pages where VACUUM disables EAGER scans (user)	0.03
vacuum_truncate	Enables vacuum to truncate empty pages at the end of the table (user)	on

□ Maximum number of autovacuum worker processes

The autovacuum_max_workers parameter can now be changed dynamically by executing the pg_reload_conf function. The parameter autovacuum_worker_slots (default 16) has been added to indicate the upper limit for dynamic changes. autovacuum_max_workers parameter setting can be larger than autovacuum_worker_slots, but the autovacuum_max_workers parameter setting will not be changed. but it will not launch more worker processes than autovacuum_worker_slots. [c758119, 6d01541]

Example 99 Automatic VACUUM worker limit

```
postgres=# SELECT name, setting, context FROM pg_settings WHERE name
               LIKE 'autovacuum%worker%' ;
          name           | setting |  context
-------------------------+---------+------------
 autovacuum_max_workers  | 3       | sighup
 autovacuum_worker_slots | 16      | postmaster
(2 rows)
```

3.3.2. Modified Parameters

The following parameters have been changed in terms of setting range and choices. [d0c2860, e48b19c, 7c3fb50, c758119, d4c3a6b, 9219093, 3516ea7]

Table 21 Modified Parameters

Parameter name	Change
autovacuum_max_workers	Changed to dynamically updatable (context=sighup)
log_rotation_size	Upper limit changed to 2,147,483,647
log_connections	Changed to string type from boolean type
log_line_prefix	Escape %L indicating local IP address can be used
max_files_per_process	Exclude the number of files inherited by the backend
ssl_groups	Changed name from ssl_ecdh_curve

□ Parameter log_connections is changed

The parameter log_connections has been changed from a boolean type to a text type, allowing multiple settings of log output to be comma-separated. [9219093, 18cd15e]

Table 22 Setting value of parameter log_connections

Setting value	Output contents
''	No logging (default value)
receipt	Log when a connection is received
authentication	Log the ID that identifies the user
authorization	Log the completion of authentication
setup_durations	Log the time between when the connection is made and when the first query is ready to be executed
all	Same as receipt,authentication,authorization

Example 100 Log output by setting log_connections parameter

```
# receipt log
LOG:  connection received: host=192.168.1.123 port=37374

# authentication log (Success)
LOG:  connection authenticated: identity="demo" method=scram-sha-256
(/data/pg_hba.conf:119)
# authentication log (Failure)
FATAL:  password authentication failed for user "demo"
DETAIL:  Connection matched file "/data/pg_hba.conf" line 119: "host     all
all             192.168.1.0/24          scram-sha-256"

# authorization log (Success)
LOG:  connection authorized: user=demo database=postgres application_name=psql
# authorization log (Failure)
FATAL:  password authentication failed for user "demo"
DETAIL:  Connection matched file "/data/pg_hba.conf" line 119: "host     all
all             192.168.1.0/24          scram-sha-256"

# setup_durations log (Success)
LOG:  connection ready: setup total=9.915 ms, fork=0.248 ms,
authentication=6.329 ms
# setup_durations log (Failure)
FATAL:  password authentication failed for user "demo"
DETAIL:  Connection matched file " /data/pg_hba.conf" line 119: "host     all
all             192.168.1.0/24          scram-sha-256"
```

The previous settings of true/false, on/off, and 0/1 can also be set for compatibility purposes. true,0,on settings are the same as “receipt,authentication,authorization”.

3.3.3. Parameters with default values changed

The following parameters have changed default values. [98f320e, 9219093, ff79b5b, cc6be07, daa02c6]

Table 23 Parameters with default values changed

Parameter name	PostgreSQL 17	PostgreSQL 18	Note
server_version	17.5	18beta1	
server_version_num	170005	180000	
data_checksums	off	on	
log_connections	off	''	
effective_io_concurrency	1	16	
maintenance_io_concurrency	10	16	
ssl_groups	prime256v1	X25519:prime256v1	Parameter name change

3.4. Utilities

Describes the major enhancements of utility commands.

3.4.1. Configure

The following new features have been added to the configure command.

□ Adding the --with-liburing option

The option --with-liburing has been added to use io_uring for asynchronous I/O on Linux. Enabling this option allows specifying io_uring for the parameter io_method. [c325a76]

□ Addition of --with-libcurl option

The option --with-libcurl option has been added for OAUTHBeare/SASL functionality. [b3f0be7]

□ Addition of --with-libnuma option

The option --with-libnuma has been added to detect NUMA environments. [65c298f]

3.4.2. Initdb

The following options have been added to the initdb command.

□ Option --no-data-checksums

Generation of page checksums is now enabled by default in the initdb command. With this specification, the option --no-data-checksums, which does not generate page checksums, has been added. If used at the same time as the --data-checksums option, which generates checksums, the option specified later will take effect. [983a588]

Example 101 Specifying the --no-data-checksums option

```
$ initdb --no-data-checksums data
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.UTF-8".
The default database encoding has accordingly been set to "UTF8".
initdb: could not find suitable text search configuration for locale "en_US.UTF-8"
The default text search configuration will be set to "simple".

Data page checksums are disabled.

creating directory data ... ok
...
```

□ Option --no-sync-data-files

Added new option --no-sync-data-files. This option skips synchronization of directories and files in the database cluster. Unless the --no-sync option is specified, directories such as pg_wal and pg_xact will be written synchronously as in the previous version. [cf131fa]

3.4.3. Pg_combinebackup

The option --link (short form -k) has been added. This option composites backup data using hard links instead of file copies. [99aeb84]

3.4.4. Pg_createsubscriber

The following options have been added to the pg_createsubscriber command.

□ Option --all

The --all option (short form -a) specifies that subscriptions should be created for all databases on the publisher's side to which it connects. [fb2ea12]

□ Option --enable-two-phase

The option --enable-two_phase has been added to enable the two_phase option for the subscriptions being created (short form -T). [e117cfb]

□ Option --remove

The --remove option (short form -R) specifies the type of object to be removed from the subscriber. Currently, only 'publications' are supported. [e5aeed4]

Example 102 Verifying Options

```
$ pg_createsubscriber --help
pg_createsubscriber creates a new logical replica from a standby server.

Usage:
  pg_createsubscriber [OPTION]...

Options:
  -a, --all                           create subscriptions for all databases
except template databases or databases that don't allow connections
  -d, --database=DBNAME               database in which to create a subscription
  -D, --pgdata=DATADIR                location for the subscriber data directory
  -n, --dry-run                       dry run, just show what would be done
  -p, --subscriber-port=PORT          subscriber port number (default 50432)
  -P, --publisher-server=CONNSTR      publisher connection string
  -R, --remove=OBJECTTYPE             remove all objects of the specified type from
specified databases on the subscriber; accepts: publications
      --replication-slot=NAME         replication slot name
...
```

3.4.5. Pg_dump

The following options have been added to the pg_dump command. [9c49f0e, bde2fb7]

Table 24 Added options

Option name	Description	Note
--sequence-data	Output sequence data	
--with-data	Dump data	
--with-schema	Dump schema information	
--with-statistics	Dump statistics data	

Example 103 Specifying the --sequence-data option

```
$ pg_dump --sequence-data | grep -i sequence
-- Name: seq01; Type: SEQUENCE; Schema: public; Owner: demo
CREATE SEQUENCE public.seq01
ALTER SEQUENCE public.seq01 OWNER TO demo;
-- Name: seq01; Type: SEQUENCE SET; Schema: public; Owner: demo
$
```

3.4.6. pg_dumpall

Non-text output formats can now be specified, as with the pg_dump command, by specifying the output format in the --format option. [1495eff]

Table 25 Specified value of --format option

Value	Value (Short)	Description	Note
custom	c	Custom format for pg_restore	
directory	d	Directory format for pg_restore	
tar	t	tar format	
plain	p	text format	Default

Example 104 Specify output format for pg_dumpall command

```
$ pg_dumpall --format=c --file=dump_dir
$ ls dump_dir/
databases  global.dat  map.dat
```

3.4.7. Pg_dump / pg_dumpall / pg_restore command

The pg_dump, pg_restore, and pg_dumpall commands now have an option to migrate statistics. The following options have been added. [1fd1bd8, cd3c451, acea3fc]

Table 26 Added options

Option name	Description	pg_dump / pg_dumpall	pg_restore
--no-data	Contains no data	○	○
--no-policies	Contains no Row Level Security settings	○	○
--no-schema	Contains no schema	○	○
--no-statistics	Contains no statistics	○	○
--statistics-only	Contains only statistics	○	-
--sequence-data	Contains sequence data	○	-

Example 105 Output statistics by pg_dump command

```
$ pg_dump -t data1 --statistics-only
--…

SELECT * FROM pg_catalog.pg_restore_relation_stats(
        'relation', 'public.data1'::regclass,
        'version', '180000'::integer,
        'relpages', '6'::integer,
        'reltuples', '1000'::real,
        'relallvisible', '0'::integer
);
SELECT * FROM pg_catalog.pg_restore_attribute_stats(
        'relation', 'public.data1'::regclass,
        'attname', 'col1'::name,
        'inherited', 'f'::boolean,
        'version', '180000'::integer,
        'null_frac', '0'::float4,
        'avg_width', '4'::integer,
        'n_distinct', '-1'::float4,
        'histogram_bounds',
'{1,10,20,30,40,50,60,70,80,90,100,110,120,130,140,150,160,170,180,190,200,210
…
```

3.4.8. Pg_recvlogical

The following features have been implemented to the pg_recvlogical command.

□ Option --failover

The option --failover has been added. Specifying this option adds a 'failover' option to the replication slot. This option is specified at the same time as the --create-slot option. [cf2655a]

□ Option --drop-slot

In previous versions, the --dbname option was required when using the --drop-slot option. In PostgreSQL 18, the --dbname option is no longer required. [c68100a]

3.4.9. Pg_resetwal

The option --char-signedness has been added to the pg_resetwal command. This option changes the sign setting of the char type in the control file. This option is expected to be specified within the pg_upgrade command and is not intended to be specified manually. [30666d1]

3.4.10. Pg_restore

The following options have been added to the pg_restore command to correspond to the changes in the pg_dumpall command. With the added --global-only option, only role and tablespace information is restored. [1495eff]

Table 27 Added options

Option name	Option (Short)	Description
--globals-only	-g	Restore global information only
--exclude-database=PATTERN	-	Databases not to be restored

3.4.11. Pg_rewind

The --write-recovery-conf option of pg_rewind has been extended to include the database name (dbname) in the primary_conninfo value generated when specified with the --source-server option. This change allows rewound servers to connect to the primary server without having to manually modify the configuration file when sync_replication_slots is enabled. [4ecdd41]

3.4.12. Pg_upgrade

The following new features have been added to the pg_upgrade command.

□ Option --set-char-signedness

The option --char-signedness has been added to the pg_upgrade command. This option changes the signedness setting of the char type in the control file. [1aab680]

□ Statistics migration

The statistics are now migrated in the default configuration. Specify the --no-statistics option to not migrate statistics. [1fd1bd8]

□ Internally executed command processing

Internally executed pg_dump and pg_dumpall commands are executed with the option --no-sync. Internally executed CREATE DATABASE statement executed with STORATEGY=FILE_COPY clause. [64f34eb, 6e16b1e]

3.4.13. Pg_verifybackup

The pg_verifybackup command can now also verify backups in tar format. The format of the backup target is specified with the --format option (short form -F). When using this option, the --no-parse-wal option must also be specified at the same time. [8dfd312]

Example 106 Execute pg_verifybackup command for tar format backup

```
$ pg_basebackup -D back.1 --format=tar
$ pg_verifybackup --format=tar --no-parse-wal back.1
backup successfully verified
```

3.4.14. Psql

The following new features have been implemented in the psql command.

□ Option --help=variables

The xheader_width description has been added to the output destination of the command option --help=variables. [768dfd8]

Example 107 Output of --help option

```
$ psql --help=variables | grep -e header -e full
  unicode_header_linestyle
  xheader_width
    set the maximum width of the header for expanded output
    [full, column, page, integer value]
```

□ Prepared statement definition

The following metacommands have been added to manipulate the definition of Prepared statements [d55322b]

Table 28 Added metacommands

Metacommand	Description	Note
\parse	Create a Prepared statement	
\bind_named	Bind a value to a parameter	
\close	Close a Prepared statement	

Example 108 Operation of Prepared statement

```
postgres=> INSERT INTO data1 VALUES ($1, $2) \parse stmt1

postgres=> SELECT name, statement FROM pg_prepared_statements ;
 name  |             statement
-------+-----------------------------------
 stmt1 | INSERT INTO data1 VALUES ($1, $2)
(1 row)

postgres=> \bind_named stmt1 100 'data1' \g
INSERT 0 1
postgres=> SELECT * FROM data1 ;
 col1  |  col2
-------+-------
 100   | data1
(1 row)

postgres=> \close stmt1
```

□ Metacommand \JdP+

The name of the access method (Access method) for the partition table is now printed. [978f38c]

Example 109 Output of the \dP+ metacommand

```
postgres=> CREATE TABLE part1(col1 INT PRIMARY KEY, col2 VARCHAR(10))
               PARTITION BY RANGE(col1) USING heap ;
CREATE TABLE
postgres=> \dP+
                                 List of partitioned relations
 Schema |    Name     | Owner |       Type        | Table | Access method |…
--------+-------------+-------+-------------------+-------+---------------+…
 public | part1       | demo  | partitioned table |       | heap          |…
 public | part1_pkey  | demo  | partitioned index | part1 | btree         |…
(2 rows)
```

□ Metacommands \df+, \do+, \dAo+, \dC+

The output of each command now includes a “Leakproof?” column indicating whether it is leakproof. [2355e51]

Example 110 Output of the \dAo+ metacommand

```
postgres=# \dAo+
                        List of operators of operator families
   AM   |      Operator family      | … | Sort opfamily | Leakproof?
--------+---------------------------+ … +---------------+-----------
 brin   | bit_minmax_ops            | … |               | yes
 brin   | bit_minmax_ops            | … |               | yes
 brin   | bit_minmax_ops            | … |               | yes
 brin   | bit_minmax_ops            | … |               | yes
…
```

□ Pipeline

The following metacommands have been added to support the pipeline. [41625ab, 3ce3575, 17caf66]

Table 29 Added metacommand

Metacommand	Description	Note
\startpipeline	Start pipeline	
\sendpipeline	Send extended query	
\syncpipeline	Synchronize pipeline	
\endpipeling	End pipeline	
\flushrequest	Request flush	
\flush	Send unsent data	
\getresults	Get execution results	Allow specifying number of fetches

The following variables are available to show the results of executing the above commands.

Table 30 Added variables

Variable name	Description	Note
PIPELINE_SYNC_COUNT	Number of pipes synchronized	
PIPELINE_COMMAND_COUNT	Number of commands piped	
PIPELINE_RESULT_COUNT	Number of results executed	

Example 111 Pipeline Execution

```
postgres=> \startpipeline
postgres=> SELECT * FROM data1 WHERE col1=$1 \bind 10 \sendpipeline
postgres=*> SELECT * FROM data1 WHERE col1=$1 \bind 20 \sendpipeline
postgres=*> \echo :PIPELINE_COMMAND_COUNT
2
postgres=*> \syncpipeline
postgres=*> \echo :PIPELINE_SYNC_COUNT
1
postgres=*> \flushrequest
postgres=*> \echo :PIPELINE_RESULT_COUNT
2
postgres=*> \getresults 2
 col1 |  col2
------+--------
   10 | data1
(1 row)

 col1 |  col2
------+--------
   20 | data2
(1 row)

postgres=> \endpipeline
```

□ Output in extended table format mode

By specifying x for some metacommands, the execution results are output in extended table format mode. In addition to the \d* metacommand, it can also be specified with \l, \z, or \lo. [00f4c29]

Example 112 Extended table format mode output

```
postgres=> \df bitand
                          List of functions
   Schema   |  Name  | Result data type | Argument data types | Type
------------+--------+------------------+---------------------+------
 pg_catalog | bitand | bit              | bit, bit            | func
(1 row)

postgres=> \dfx bitand
List of functions
-[ RECORD 1 ]-------+-----------
Schema              | pg_catalog
Name                | bitand
Result data type    | bit
Argument data types | bit, bit
Type                | func
```

□ Variable SERVICE

The variable SERVICE is now available to indicate the name of the service used for the connection. Specify %s to include the service name in the prompt string. [477728b]

Example 113 Output service name

```
postgres=> \set PROMPT1 '%/[%n][%s]> '
postgres[demo][]> \connect "service=pg18"
You are now connected to database "postgres" as user "demo" on host "localhost"
(address "::1") at port "5432".
postgres[demo][pg18]> \echo :SERVICE
pg18
postgres[demo][pg18]>
```

□ Title by object type

The titles of object list metacommands such as \dt and \di now change according to object type. In the previous version, they were all “List of relations”. [a14707d]

Example 114 Title according to object type

```
postgres=> \dt
         List of tables
 Schema | Name  | Type  | Owner
--------+-------+-------+-------
 public | data1 | table | demo
(1 row)

postgres=> \di
               List of indexes
 Schema |    Name     | Type  | Owner | Table
--------+-------------+-------+-------+-------
 public | data1_pkey | index | demo  | data1
(1 row)

postgres=> \dm
Did not find any materialized views.
postgres=> \ds
Did not find any sequences.
```

□ Metacommand \dx

The output of the \dx metacommand now includes the "Default version" entry for the extension. [d696406]

Example 115 Output of the \dx metacommand

```
postgres=> \dx
                              List of installed extensions
   Name   | Version | Default version |   Schema   |Description
----------+---------+-----------------+------------+-------------------------...
 pgcrypto | 1.4     | 1.4             | public     | cryptographic functions
 plpgsql  | 1.0     | 1.0             | pg_catalog | PL/pgSQL  procedural ...
(2 rows)
```

3.4.15. Vacuumdb

The vacuumdb command now has a --missing-stats-only option to analyze only columns and indexes with missing object statistics. This option must be specified in conjunction with the --analyze-only or --analyze-in-stages option. [edba754]

Example 116 Specifying the --missing-stats-only option

```
$ vacuumdb --verbose --missing-stats-only --analyze-only
vacuumdb: vacuuming database "postgres"
```

3.5. Contrib modules

Describes new features related to the Contrib modules.

3.5.1. Amcheck

The function gin_index_check, which checks the GIN index, has been added. [14ffaec]

Example 117 Checking the GIN index

```
postgres=# CREATE INDEX idx1_gin1 ON gin1 USING GIN
                (to_tsvector('english', title1)) ;
CREATE INDEX
postgres=# SELECT gin_index_check('idx1_gin1') ;
 gin_index_check
-----------------


(1 row)
```

3.5.2. Bloom

Bloom index usage is now stored in the pg_stat_*_index view. [93063e2]

Example 118 Bloom Index Usage

```
postgres=> CREATE INDEX idx1_bloom ON data1 USING bloom (col1) ;
CREATE INDEX
postgres=> EXPLAIN ANALYZE SELECT * FROM data1 WHERE col1=10000 ;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Bitmap Heap Scan on data1  (cost=15348.00..15352.01 rows=1 width=10) …
   Recheck Cond: (col1 = 10000)
   Rows Removed by Index Recheck: 350
   Heap Blocks: exact=341
   ->  Bitmap Index Scan on idx1_bloom  (cost=0.00..15348.00 rows=1 width=0)…
         Index Cond: (col1 = 10000)
 Planning Time: 0.101 ms
 Execution Time: 3.602 ms
(8 rows)

postgres=> SELECT indexrelname, idx_scan, idx_tup_read FROM
                pg_stat_user_indexes ;
 indexrelname | idx_scan | idx_tup_read
--------------+----------+--------------
 idx1_bloom   |        1 |          351
(1 row)
```

3.5.3. Dblink

The option use_scram_passthrough has been added to the foreign data wrapper dblink_fdw to pass through SCRAM authentication. [3642df2]

Example 119 Pass-through of SCRAM authentication credentials

```
postgres=# CREATE SERVER link1 FOREIGN DATA WRAPPER dblink_fdw OPTIONS
(host 'dbsvr1', port '5432', dbname 'demodb', use_scram_passthrough 'true') ;
CREATE SERVER
```

3.5.4. File_fdw

The following options have been added. These options can have the same values as the options in the COPY statement. [a1c4c8a, 6c8f670]

Table 31 Added options

Option name	Description
on_error	Determines the action to be taken when a minor error occurs. The possible values are as follows • default: Stop execution (default value) • ignore: Continue execution of the process
log_verbosity	This parameter determines the level of log output. The following values can be specified • default: Output the number of errors (default value) • verbose: Output more detailed logs • silent: No log output
reject_limit	Specifies the number of data type conversion error records to allow, with the on_error=ignore option

Example 120 Additional options for file_fdw module (option on_error = 'ignore')

```
postgres=# \! cat /tmp/fs1.csv
100,data1
ABC,data2
300,data3
postgres=# CREATE EXTENSION file_fdw ;
CREATE EXTENSION
postgres=# CREATE SERVER fs FOREIGN DATA WRAPPER file_fdw ;
CREATE SERVER
postgres=# CREATE FOREIGN TABLE fs1 (id INT, val TEXT) SERVER fs
        OPTIONS (filename '/tmp/fs1.csv', FORMAT 'csv',
                ON_ERROR 'ignore', REJECT_LIMIT '1') ;
CREATE FOREIGN TABLE
postgres=# SELECT * FROM fs1 ;
NOTICE:  1 row was skipped due to data type incompatibility
 id  |  val
-----+-------
 100 | data1
 300 | data3
(2 rows)
```

Example 121 Additional options for file_fdw module (option log_verbosity = 'verbose')

```
postgres=# CREATE FOREIGN TABLE fs2 (id INT, val TEXT) SERVER fs
  OPTIONS (filename '/tmp/fs1.csv', FORMAT 'csv', ON_ERROR 'ignore',
        REJECT_LIMIT '1', LOG_VERBOSITY 'verbose') ;
CREATE FOREIGN TABLE
postgres=# SELECT * FROM fs2 ;
NOTICE:  skipping row due to data type incompatibility at line 2 for column
"id": "ABC"
NOTICE:  1 row was skipped due to data type incompatibility
 id  |  val
-----+-------
 100 | data1
 300 | data3
(2 rows)
```

3.5.5. Injection_points

The following new features have been added to the injection_points module

□ Additional custom parameter

The parameter injection_points.stats has been added. This parameter determines whether cumulative statistics are enabled in the module. By default, statistics are disabled (off). [2e35c67]

□ Add callbacks

The callback to initialize the shared memory of the module specified in the shared_preload_libraries parameter has been added. [b2b023a]

3.5.6. Isn

The isn.weak parameter has been added to the isn module. Previous versions used the isn_weak function, which has been retained for compatibility but is now set to VOLATILE. [4489044]

3.5.7. Passwordcheck

The parameter passwordcheck.min_password_length, which indicates the minimum password length, has been added to the passwordcheck module. The default value for this parameter is 8. [f7e1b38]

Example 122 Minimum password length setting

```
postgres=# SET passwordcheck.min_password_length = 12 ;
SET
postgres=# ALTER ROLE demo PASSWORD 'ABCDEFGHIJK' ;
ERROR:  password is too short
DETAIL:  password must be at least "passwordcheck.min_password_length" (12)
bytes long
```

3.5.8. Pgcrypto

The following enhancements have been added to the pgcrypto module

□ Addition of the fips_mode function

The fips_mode function has been added to indicate whether OpenSSL has FIPS enabled. [924d89a]

Syntax 19 Fips_mode function

```
boolean fips_mode()
```

Example 123 Execution of the fips_mode function

```
postgres=> SELECT fips_mode() ;
 fips_mode
-----------
 f
(1 row)
```

□ Control of built-in crypto functions

When OpenSSL is used in FIPS mode, cryptographic implementations that are not FIPS certified should not be used. The parameter pgcrypto.built-in_crypto_enabled has been added to disable built-in functions for this purpose. Only SUPERUSER can change this parameter. [035f99c]

Example 124 Disable built-in functions

```
postgres=# SET pgcrypto.builtin_crypto_enabled = off ;
SET
postgres=# SELECT gen_salt('des') ;
ERROR:  use of built-in crypto functions is disabled
postgres=# SELECT crypt('data1', 'salt1') ;
ERROR:  use of built-in crypto functions is disabled
```

□ CFB Mode Added to AES Encryption Functions

CFB (Cipher-FeedBack mode) has been added to the AES encryption mode. [9ad1b3d]

Example 125 Added CFB mode

```
postgres=> SELECT encrypt('foo', '0123456789', 'aes-cfb') ;
 encrypt
----------
 \x6f8ced
(1 row)

postgres=> SELECT encode(decrypt(encrypt('foo', '0123456', 'aes-cfb'),
                '0123456', 'aes-cfb'), 'escape') ;
 encode
--------
 foo
(1 row)
```

□ SHA-2-based hashing and encryption

SHA-2 based hash algorithms sha256crypt and sha512crypt are now available. [749a9e2]

Example 126 SHA-256 / SHA-512 hashing and encryption

```
postgres=> SELECT gen_salt('sha256crypt') ;
            gen_salt
------------------------------
 $5$rounds=5000$yTxNtev7NxPXhErT
(1 row)

postgres=> SELECT gen_salt('sha512crypt') ;
            gen_salt
------------------------------
 $6$rounds=5000$NoKeO.dTtV9RHhBN
(1 row)

postgres=> SELECT crypt('test', gen_salt('sha512crypt')) ;
                                                              crypt
------------------------------------------------------------------------------...
$6$rounds=5000$B6qwrNDTAV6Zq5ZX$D8mfxoa3IO2BUjZWG4fMCs52wcb9wLigkDwfA4ZZ6CE3yW
Mwy5sIZOxLCZI/dctvxQP9Grx7sk.ooQu5GI7hf1
(1 row)
```

3.5.9. Pg_buffercacahe

The following new features have been added to the pg_buffercache module

□ Added the pg_buffercache_numa view

New view pg_buffercache_numa with the following columns has been added. [ba2a3c2]

Table 32 Column structure of the pg_buffercache_numa view

Column name	Data type	Description
bufferid	integer	Buffer ID
os_page_num	bigint	OS page number
numa_node	integer	NUMA node number

□ Added functions

The following functions have been added. [dcf7e16]

Table 33Added functions

Function name	Description
pg_buffercache_evict_relation	Remove specific table information from buffer cache
pg_buffercache_evict_all	Remove all information from buffer cache

Example 127 Execution of the pg_buffercache_evict_relation function

```
postgres=# SELECT * FROM pg_buffercache_evict_relation('data1') ;
 buffers_evicted | buffers_flushed | buffers_skipped
-----------------+-----------------+-----------------
            5410 |            5406 |               0
(1 row)
```

3.5.10. Pg_logicalinspect

The pg_logicalinspect module has been added to PostgreSQL 18. This contrib module provides functions to inspect logical decoding components. The added function specifies a snapshot file of the logical decoding. [7cdfeee]

Syntax 20 Pg_get_logical_snapshot / Pg_get_logical_snapshot_info functions

```
record pg_get_logical_snapshot_meta(filename text)
record pg_get_logical_snapshot_info(filename text)
```

Example 128 Execution of the pg_get_logical_snapshot_meta function

```
postgres=# SELECT * FROM pg_get_logical_snapshot_meta('0-57004740.snap') ;
-[ RECORD 1 ]--------
magic    | 1369563137
checksum | 1831531145
version  | 6
```

Example 129 Execution of the pg_get_logical_snapshot_info function

```
 postgres=# SELECT * FROM pg_get_logical_snapshot_info('0-57004740.snap') ;
-[ RECORD 1 ]------------+-----------
state                    | consistent
xmin                     | 767
xmax                     | 766
start_decoding_at        | 0/570043B8
two_phase_at             | 0/0
initial_xmin_horizon     | 0
building_full_snapshot   | f
in_slot_creation         | f
last_serialized_snapshot | 0/57004708
next_phase_at            | 0
committed_count          | 0
committed_xip            |
catchange_count          | 0
catchange_xip
```

3.5.11. Pg_overexplain

The pg_overexplain module has been added to PostgreSQL 18. When loaded, this module allows the addition of the DEBUG option to the EXPLAIN statement, which can be used to debug the output of an EXPLAIN statement. [8d5ceb1, 9f0c36a]

Example 130 Execution of the EXPLAIN (DEBUG) statement

```
postgres=# LOAD 'pg_overexplain' ;
LOAD
postgres=# EXPLAIN (DEBUG) SELECT * FROM data1 WHERE c1=1000 ;
                                QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using data1_pkey on data1  (cost=0.42..8.44 rows=1 width=10)
   Index Cond: (c1 = 1000)
   Disabled Nodes: 0
   Parallel Safe: true
   Plan Node ID: 0
 PlannedStmt:
   Command Type: select
   Flags: canSetTag
   Subplans Needing Rewind: none
   Relation OIDs: 16389
   Executor Parameter Types: none
   Parse Location: 16 for 33 bytes
(12 rows)
```

3.5.12. Pg_stat_statements

The following features have been added to the pg_stat_statements module.

□ Retrieving Internal SQL

The query ID can now also be calculated for internal SQL statements used for CREATE TABLE AS and DECLARE CURSOR statements. This allows the execution time of internal SQL to be retrieved in environments where the pg_stat_statements.track option is set to 'all'. [6b652e6]

Example 131 Capture internal SQL statements

```
postgres=> SHOW pg_stat_statements.track ;
 pg_stat_statements.track
--------------------------
 all
(1 row)

postgres=> CREATE TABLE data2 AS SELECT * FROM data1 WHERE col1=1000 ;
SELECT 1
postgres=> SELECT queryid, query FROM pg_stat_statements ;
       queryid       |                          query
---------------------+-------------------------------------------------------------
 4147066388115963993 | SHOW pg_stat_statements.track
 3197110067974111441 | SELECT * FROM data1 WHERE col1=$1
 6773306165436199945 | CREATE TABLE data2 AS SELECT * FROM data1 WHERE col1=$1
(3 rows)
```

□ SET Statements as Constants

The SET statement settings are stored as constants with parameter symbols in the pg_stat_statements table. Previously, statements with different specified values were stored separately. If the constants DEFAULT or FROM CURRENT are used, they are not subject to constant conversion. [dc68515]

Example 132 SET statement constant

```
postgres=> SET work_mem = '8MB' ;
SET
postgres=> SET work_mem = '16MB' ;
SET
postgres=> SET work_mem = DEFAULT ;
SET
postgres=> SELECT query, calls FROM pg_stat_statements WHERE query LIKE 'SET%' ;
         query          | calls
------------------------+-------
 SET work_mem = DEFAULT |     1
 SET work_mem = $1      |     2
(2 rows)
```

□ Additional Columns

The following columns have been added to the pg_stat_statements view. The columns added are parallel query execution information and the number of times the WAL buffer was full. [cf54a2c, ce5bcc4]

Table 34 Added columns

Column name	Data type	Description
parallel_workers_to_launch	bigint	Number of parallel workers scheduled to execute
parallel_workers_launched	bigint	Number of parallel workers executed
wal_buffers_full	bigint	Number of times the WAL buffer has been full

Example 133 Verify additional columns (parallel query information)

```
postgres=> SELECT COUNT(*) FROM data1 ;
-[ RECORD 1 ]--
count | 1000000

postgres=> SELECT query, parallel_workers_to_launch, parallel_workers_launched
FROM pg_stat_statements WHERE userid=16384 ;
-[ RECORD 1 ]----------+-------------------------
query                  | SELECT COUNT(*) FROM data1
parallel_workers_to_launch | 4
parallel_workers_launched  | 4
```

□ Compression of Constant Lists

When the number of IN lists in a WHERE clause is variable, previous versions allocated separate Query IDs. PostgreSQL 18 compresses the constant list and assigns a single Query ID. [62d712e, 9fbd53d]

For example, execute the following SQL statement and check it in the pg_stat_statements view.

Example 134 SQL statements to execute

```
SELECT * FROM data1 WHERE col1 IN (10, 20) ;
SELECT * FROM data1 WHERE col1 IN (10, 20, 30, 40, 50) ;
```

Example 135 PostgreSQL 17 behavior

```
postgres=> SELECT calls, query FROM pg_stat_statements WHERE query LIKE
'%data1%' ;
 calls |                          query
-------+------------------------------------------------------------
     1 | SELECT * FROM data1 WHERE col1 IN ($1, $2)
     1 | SELECT * FROM data1 WHERE col1 IN ($1, $2, $3)
(2 rows)
```

Example 136 PostgreSQL 18 behavior

```
postgres=> SELECT calls, query FROM pg_stat_statements WHERE query LIKE
'%data1%' ;
 calls |                       query
-------+-------------------------------------------------
     2 | SELECT * FROM data1 WHERE col1 IN ($1 /*, ... */)
(1 row)
```

3.5.13. Pgstattuple / pageinspect

Each function in the Contrib module now also supports sequences. [05036a3]

Example 137 Pageinspect module

```
postgres=# CREATE EXTENSION pageinspect ;
CREATE EXTENSION
postgres=# SELECT tuple_data_split('seq1'::regclass, t_data, t_infomask,
t_infomask2, t_bits) FROM heap_page_items(get_raw_page('seq1', 0)) ;
-[ RECORD 1 ]----+-----------------------------------------------------
tuple_data_split | {"\\x0100000000000000","\\x0000000000000000","\\x00"}
```

Example 138 Pgstattuple module

```
postgres=# CREATE SEQUENCE seq1 ;
CREATE SEQUENCE
postgres=# CREATE EXTENSION pgstattuple ;
CREATE EXTENSION
postgres=# SELECT * FROM pgstattuple('seq1') ;
-[ RECORD 1 ]------+------
table_len          | 8192
tuple_count        | 1
tuple_len          | 41
tuple_percent      | 0.5
dead_tuple_count   | 0
dead_tuple_len     | 0
dead_tuple_percent | 0
free_space         | 8108
free_percent       | 98.97
```

3.5.14. Postgres_fdw

The following features have been added to the postgres_fdw module.

□ SCRAM Authentication Pass-through

The option USE_SCRAM_PASSThrough has been added to perform authentication by automatically sending SCRAM authentication information. This option can be specified as an attribute of SERVER or USER MAPPING. [761c795]

Example 139 Pass-through settings for SCRAM authentication

```
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw
               OPTIONS (host 'dbsvr1', port '5432', dbname 'postgres') ;
CREATE SERVER
postgres=# CREATE USER MAPPING FOR demo SERVER remsvr1
        OPTIONS (user 'demo', use_scram_passthrough 'true') ;
CREATE USER MAPPING
```

□ Enhancements to the postgres_fdw_get_connections function

The used_in_xact and closed columns have been added to the return value of the postgres_fdw_get_connections function. Also, a check_conn parameter (default false) has been added to the function's parameters to check for connections. [c297a47, 857df3c, 4f08ab5]

Table 35 Added return column

Column name	Data type	Description
user_name	text	Connection mapping username (or public)
used_in_xact	boolean	Is a transaction in progress?
closed	boolean	Is the session closed?

Example 140 Execution of the postgres_fdw_get_connections function

```
postgres=> BEGIN ;
BEGIN
postgres=*> INSERT INTO data1 VALUES (100, 'data1') ;
INSERT 0 1
postgres=*> SELECT * FROM postgres_fdw_get_connections(true) ;
 server_name | user_name | valid | used_in_xact | closed | remote_backend_pid
-------------+-----------+-------+--------------+--------+--------------------
 remsvr1     | public    | t     | t            | f      |             309160
(1 row)

postgres=*> COMMIT ;
COMMIT
postgres=> SELECT * FROM postgres_fdw_get_connections(true) ;
 server_name | user_name | valid | used_in_xact | closed | remote_backend_pid
-------------+-----------+-------+--------------+--------+--------------------
 remsvr1     | public    | t     | f            | f      |             309160
(1 row)
```

□ Function postgres_fdw_get_connections

The remote_backend_pid column (type int4), which indicates the backend process ID of the remote server, has been added to the return value of the postgres_fdw_get_connections function. [fe186bd]

Example 141 Execution of the postgres_fdw_get_connections function

```
postgres=> CREATE FOREIGN TABLE data1(col1 INT, col2 VARCHAR(10))
                SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=> SELECT * FROM data1 ;
 col1 | col2
------+------
(0 rows)

postgres=> SELECT * FROM postgres_fdw_get_connections() ;
 server_name | user_name | valid | used_in_xact | closed | remote_backend_pid
-------------+-----------+-------+--------------+--------+--------------------
 remsvr1     | demo      | t     | f            |        |             370760
(1 row)
```

3.5.15. Miscellaneous

In addition to those described above, the following test extension modules have been added under the src/test/modules directory. [93bc3d7, b3f0be7, b85a9d0]

Table 36 Added modules

Module name	Dscription
oauth_validator	OAUTHBeare/SASL Validator
test_aio	Asynchronous I/O Testing
typecache	Type cache Testing

URL list

The following websites are references to create this material.

- Release Notes
  https://www.postgresql.org/docs/18/release.html
- Commitfests
  https://commitfest.postgresql.org/
- PostgreSQL 18 Manual
  https://www.postgresql.org/docs/18/index.html
- PostgreSQL 18 Open Items
  https://wiki.postgresql.org/wiki/PostgreSQL_18_Open_Items
- Git
  git://git.postgresql.org/git/postgresql.git
- GitHub
  https://github.com/postgres/postgres
- PostgreSQL 18 Beta 1 のアナウンス
  https://www.postgresql.org/about/news/postgresql-18-beta-1-released-3070/
- Qiita（ぬこ@横浜さん）
  http://qiita.com/nuko_yokohama
- pgsql-hackers Mailing list
  https://www.postgresql.org/list/pgsql-hackers/
- PostgreSQL Developer Information
  https://wiki.postgresql.org/wiki/Development_information
- pgPedia
  https://pgpedia.info/postgresql-versions/postgresql-18.html
- Slack - postgresql-jp (Japanese)
  https://postgresql-jp.slack.com/

Change History

Change history

Version	Date	Author	Description
0.1	2025/05/06	Noriyoshi Shinoda	Create an internal review version.
1.0	2025/05/26	Noriyoshi Shinoda	Verification completed in PostgreSQL 18 Beta 1 environment.