Hewlett Packard
Enterprise

October 1, 2021

PostgreSQL 14 New Features

With Examples (GA)

Hewlett Packard Japan, G.K.

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 14 (14.0) GA.

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 difference between PostgreSQL 13 (13.4) and PostgreSQL 14 (14.0). 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 psql command tab input
- Improvement of pgbench command
- Improvement of documentation, modify typo in the source code
- Refactoring without a change in behavior

1.4 Software Version

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

Table 1 Version

Software	Version
PostgreSQL	PostgreSQL 13.4 (for comparison) PostgreSQL 14 (14.0) (Sep 27, 2021 21:02:15)
Operating System	Red Hat Enterprise Linux 7 Update 8 (x86-64)
Configure option	--with-llvm --with-ssl=openssl --with-perl --with-lz4

1.5. The Question, comment, and Responsibility

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

1.6 Notation

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

Table 2 Examples notation

Notation	Description
#	Shell prompt for Linux root user.
$	Shell prompt for Linux general user.
Bold	The user input string.
postgres=#	psql command prompt for PostgreSQL administrator.
postgres=>	psql command prompt for PostgreSQL general user.
<u>Underline</u>	Important output items.
<<PASSWORD>>	Replaced by password string.

The syntax is described in the following rules:

Table 3 Syntax rules

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

Hewlett Packard
Enterprise

2. New Features Summary

More than 200 new features have been added to PostgreSQL 14. Here are some major new features and benefits.

2.1. Improve analytic query performance

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

□ Logical Streaming Replication enhancements

Logical replication can now transfer decoded updates to a standby instance without waiting for the transaction to complete. In addition, it is now possible to send transfer data that was previously converted to text in binary.

□ Parallel query enhancement

Parallel queries now work for REFRESH MATERIALIZED VIEW statements.

□ LZ4 compression

LZ4 compression is now available for TOAST columns.

□ BRIN index

Bloom filters are now available.

2.2. Improve reliability

PostgreSQL 14 implements the following enhancements to improve reliability.

□ Enhanced data structure checking

Contrib module amcheck now includes a function to check object structure. Also, add pg_amcheck command to run checks by the amcheck module from the command line

□ Elimination of password length restrictions

The restriction on password length has been removed, and very long passwords can now be used.

2.3. Improved maintainability

The following features that can improve operability have been added.

□ Enhancements to the REINDEX statement

It is now possible to execute a REINDEX statement on a partitioned table. This eliminates the need to run REINDEX on each partition. In addition, it will be able to move the index to be re-created with the TABLESPACE clause.

□ Enhancement of monitoring feature

A real-time view of COPY command execution status has been added. A view has also been added to check WAL usage and memory status of backend processes. In the Contrib module pg_stat_statements, it is possible to acquire information on utility statements such as REFRESH MATERIALIZED VIEW statement, and the statistical information that can be acquired has increased significantly.

2.4. Preparing for future new features

PostgreSQL 14 is now ready for features that will be provided in future versions.

□ Data Access by Subscripts

The infrastructure for subscript data access to new data types has been implemented, and the hstore module can take advantage of this feature.

2.5. Incompatibility

In PostgreSQL 14, the following specifications have been changed from PostgreSQL 13.

2.5.1. Functions

The following functions have been changed.

□ LEAD / LAG

The data type of some function return values has been changed from anyelement type to anycompatible type.

□ EXTRACT

The return data type of the EXTRACT function has been changed from double precision to numeric. In previous versions, the EXTRACT function internally called the DATE_PART function, but now it's independent. The data type of the return value of the DATE_PART function remains double precision.

□ VAR_SAMP, STDDEV_SAMP

The return value specified as a NaN value of type numeric has changed from NaN to NULL.

2.5.2. Regular Expression

In newline-sensitive mode, the escape \W and \D now match the linefeed code.

Example 1 Match newline (PostgreSQL 13)

```
postgres=> SELECT regexp_match(E'A\nC', '\D\D\D', 'n') ;
 regexp_match
--------------
 null
(1 row)
```

Example 2 Match newline (PostgreSQL 14)

```
postgres=> SELECT regexp_match(E'A\nC', '\D\D\D', 'n') ;
 regexp_match
--------------
 {"A          +
 C"}
(1 row)
```

2.5.3. Factorial calculation

The factorial calculation result and operator have been changed.

□ Factorial of negative value

Factorial calculation of negative values will result in an error.

Example 3 Factorial calculation (PostgreSQL 13)

```
postgres=> SELECT factorial(-4) ;
factorial
-----------
        1
(1 row)
```

Example 4 Factorial calculation (PostgreSQL 14)

```
postgres=> SELECT factorial(-4) ;
ERROR:  factorial of a negative number is undefined
```

□ Removed operator

The factorial operator (! and !!) has been removed.

Example 5 Factorial calculation (PostgreSQL 14)

```
postgres=> SELECT 4! ;
RROR:  syntax error at or near ";"
LINE 1: SELECT 4! ;
                 ^
```

□ Removed function

The numeric_fac function has been removed.

2.5.4. Operators

The deprecated operators @ and ~ have been removed from the Contrib modules cube, hstore, intarray and seg.

2.5.5. Manuals

The Default Roles in the manual and the program source have been changed to Predefined Roles.

2.5.6. Transaction wraparound warning

The threshold points of the remaining transactions for the wraparound of the transaction ID have been changed as follows.

Hewlett Packard
Enterprise

Table 4 Warning of transaction ID wraparound

Comparison	PostgreSQL 13	PostgreSQL 14	Note
Warning	11 million	40 million	
Shutdown	One million	3 million	

2.5.7. Start-up options

The -o option, which specifies additional options when starting postmaster, has been removed.

2.5.8. Extended Statistics

CREATE STATISTICS statement for system catalogs can no longer be executed. This limit is backported to older versions.

Example 6 CREATE STATISTICS statement for system catalog

```
postgres=# CREATE STATISTICS stat1_collation ON collname,
       collowner FROM pg_collation ;
ERROR:  permission denied: "pg_collation" is a system catalog
```

2.5.9. Protocol

The version 2 client-backend communication protocol has been removed. This protocol has been used since PostgreSQL 7.4.

2.5.10. ALTER OPERATOR

Removed support for postfix (right-unary) operators on ALTER OPERATOR statement and DROP OPERATOR statement.

2.5.11. ALTER TABLE

The DROP EXPRESSION clause can no longer be specified in the ALTER TABLE ONLY statement.

2.5.12. CREATE LANGUAGE

Specifying a name in single quotes will result in an error in the CREATE LANGUAGE statement or DROP LANGUAGE statement.

Example 7 CREATE LANGUAGE statement (PostgreSQL 14)

```
postgres=# CREATE LANGUAGE 'plperl' ;
ERROR:  syntax error at or near "'plperl'"
LINE 1: CREATE LANGUAGE 'plperl';
```

2.5.13. LIBPQ connection string

The connection strings authtype and tty have been removed. The client connection setting sslcompression for SSL compression is ignored by the backend.

2.5.14. Findoidjoins

The findoidjoins command has been removed.

2.5.15. Intarray

Prevent the containment operators (<@ and @>) for intarray module from using GiST indexes.

2.5.16. Pg_hba.conf file

Values "0", "1", and "no-verify" are no longer supported of the clientcert option in the pg_hba.conf file.

2.5.17. Pg_standby

The pg_standby command has been removed.

2.5.18. Tablefunc

Negative values can no longer be specified for the first parameter of the normal_rand function.

Example 8 Normal_rand function

```
postgres=# CREATE EXTENSION tablefunc ;
CREATE EXTENSION
postgres=# SELECT normal_rand(-1, 1.2, 1.3) ;
ERROR:  number of rows cannot be negative
```

Hewlett Packard
Enterprise

3. New Feature Detail

3.1. Architecture

3.1.1. Modified catalogs

The following catalogs have been changed.

Table 5 Added system catalogs and views

Catalog/View name	Description
pg_backend_memory_contexts	Displays all the memory contexts of the server process attached to the current session.
pg_stat_progress_copy	Outputs the execution status of the COPY statement.
pg_stat_replication_slots	Outputs the usage status of the replication slot.
pg_stat_wal	Outputs the WAL data output status.
pg_stats_ext_exprs	Provides information about all expressions included in extended statistics objects

Table 6 Added views in the information schema (information_schema)

View name	Description
routine_column_usage	Identifies the column used by the function or procedure. Currently not tracked.
routine_routine_usage	Identifies the function used by the function or procedure. Currently, some are not tracked.
routine_sequence_usage	Identifies the sequence used by a function or procedure. Currently, some are not tracked.
routine_table_usage	Identifies the table used by the function or procedure. Currently not tracked.

Table 7 System catalogs/views with additional columns

Catalog/View name	Added column	Data type	Description
pg_attribute	attcompression	char	Current compression method of the column.
pg_inherits	inhdetachpendin g	boolean	True for a partition that is in the process of being detached
pg_locks	waitstart	timestamp with time zone	Time when the server process started waiting for this lock.
pg_prepared_statements	generic_plans	bigint	Number of times generic plan was chosen.
	custom_plans	bigint	Number of times custom plan was chosen
pg_proc	prosqlbody	pg_node_tre e	Pre-parsed SQL function body.
pg_range	rngmultitypid	oid	OID of the multirange type.
pg_replication_slots	two_phase	boolean	Enabled for decoding prepared transactions.
pg_stat_activity	query_id	bigint	Identifier of this backend's most recent query.
pg_stat_database	session_time	double precision	Time spent by database sessions in this database.
	active_time	double precision	Time spent executing SQL statements in this database.
	idle_in_transacti on_time	double precision	Time spent idling while in a transaction in this database.
	sessions	bigint	Total number of sessions.
	sessions_abando ned	bigint	Number of lost sessions.
	sessions_fatal	bigint	Number of sessions caused by fatal error.
	sessions_killed	bigint	Number of sessions terminated by the operator.

Catalog/View name	Added column	Data type	Description
pg_stats_ext	exprs	text[]	Expression to get extended statistics
pg_statistic_ext	stxexprs	pg_node_tree	Expression trees for statistics object attributes.
pg_statistic_ext_data	stxdexpr	pg_statistic []	A list of any expressions covered by this statistics object.
pg_subscription	subbinary	boolean	Publisher send data in binary format.
	substream	boolean	Streaming of in-progress transactions.
pg_type	typsubscript	regproc	Subscripting handler function's OID.

Table 8 System catalogs/views with columns removed

Catalog name	Column name	Description
pg_stat_ssl	compression	It was deleted because compression was disabled.

Table 9 System catalogs/views with modified output

Catalog/View name	Description
pg_attribute	The column definition order has changed.
pg_class	The initial value of the reltuples column has been changed to -1.
pg_settings	pending_restart column shows as true when a pertinent entry in postgresql.conf is removed.
pg_statistic_ext	Extended statistics with the 'e'= expression may be output in the stxkind column.
pg_stat_activity	Information of the archiver process and the walsender process will be output.
	walsender now show their latest replication command. This feature is backported to older versions.
pg_subscription_rel	'f' = finished table copy may be output in the srsubstate column.
pg_type	The typtype column may output 'm' to indicate a multi-range type.

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

□ Pg_backend_memory_contexts

The pg_backend_memory_contexts view provides a view of the memory context usage of the server process corresponding to the current session. This view can only be viewed by users with the SUPERUSER attribute.

Table 10 Pg_backend_memory_contexts view

Column name	Data type	Description
name	text	Name of the memory context.
ident	text	Identification information of the memory context.
parent	text	Name of the parent of this memory context.
level	integer	Distance from TopMemoryContext.
total_bytes	bigint	Total bytes allocated for this memory context.
total_nblocks	bigint	Total number of blocks allocated for this memory context.
free_bytes	bigint	Free space in bytes.
free_chunks	bigint	Total number of free chunks.
used_bytes	bigint	Used space in bytes.

Example 9 Query pg_backend_memory_contexts view

```
postgres=# SELECT name, parent, level, total_bytes FROM
               pg_backend_memory_contexts ;
         name           |      parent       | level | total_bytes
--------------------------+--------------------+-------+------------
 TopMemoryContext        |                   |    0 |      68704
 TopTransactionContext    | TopMemoryContext   |    1 |        8192
 Record information cache | TopMemoryContext   |    1 |        8192
 TableSpace cache        | TopMemoryContext   |    1 |        8192
 Type information cache   | TopMemoryContext   |    1 |       24376
 Operator lookup cache    | TopMemoryContext   |    1 |       24576
 RowDescriptionContext    | TopMemoryContext   |    1 |        8192
 MessageContext          | TopMemoryContext   |    1 |       32768
 Operator class cache     | TopMemoryContext   |    1 |        8192
…
```

□ Pg_stat_progress_copy

This view will output the execution status of the COPY statement.

Table 11 Pg_stat_progress_copy view

Column name	Data type	Description
pid	integer	Process ID of backend.
datid	oid	OID of the database
datname	name	Name of the database
relid	oid	OID of the table on which the COPY command is executed.
command	text	Command name, COPY FROM or COPY TO.
type	text	The IO types.
bytes_processed	bigint	Number of bytes already processed.
bytes_total	bigint	Size of source file for COPY FROM command.
tuples_processed	bigint	Number of tuples already processed.
tuples_excluded	bigint	Number of tuples not processed because they were excluded.

Example 10 Query pg_stat_progress_copy view

```
postgres=> SELECT * FROM pg_stat_progress_copy ;
-[ RECORD 1 ]---+----------
pid              | 83409
datid            | 13887
datname          | postgres
relid            | 16385
command          | COPY FROM
type             | FILE
bytes_processed  | 22806528
bytes_total      | 138888897
tuples_processed | 1703999
tuples_excluded  | 0
```

□ Pg_stat_replication_slots

This view outputs the state of the logical replication slot. To reset the contents of the view, run the pg_stat_reset_replication_slot function.

Table 12 Pg_stat_replication_slots view

Column name	Data type	Description
slot_name	text	Identifier for the replication slot.
spill_txns	bigint	Number of transactions spilled to disk.
spill_count	bigint	Number of times transactions were spilled to disk.
spill_bytes	bigint	Amount of decoded transaction data spilled to disk.
stream_txns	bigint	Number of in-progress transactions streamed.
stream_count	bigint	Number of times in-progress transactions.
stream_bytes	bigint	Amount of decoded in-progress transaction data streamed.
total_txns	bigint	Number of decoded transactions.
total_bytes	bigint	Amount of decoded transactions data.
stats_reset	timestamp with time zone	Time at which these statistics were last reset

Example 11 Query pg_stat_replication_slots view

```
postgres=> SELECT * FROM pg_stat_replication_slots ;
-[ RECORD 1 ]+----------
slot_name    | sub1
spill_txns   | 3
spill_count  | 9
spill_bytes  | 416939406
stream_txns  | 0
stream_count | 0
stream_bytes | 0
total_txns   | 0
total_bytes  | 0
stats_reset  |
```

□ Pg_stat_wal

Only one tuple of WAL output status is output to the pg_stat_wal view. The value of this view can be reset by pg_stat_reset_shared function with 'wal' specified. The values of the wal_write_time and wal_sync_time columns are output only when the track_wal_io_timing parameter to 'on'.

Table 13 Pg_stat_wal view

Column name	Data type	Description
wal_records	bigint	Total number of WAL records.
wal_fpi	bigint	Total number of WAL full page images.
wal_bytes	numeric	Total amount of WAL generated.
wal_buffers_full	bigint	Number of times WAL data was written to disk because WAL buffers became full
wal_write	bigint	Number of times WAL buffers were written.
wal_sync	bigint	Number of times WAL files were synced to disk.
wal_write_time	double precision	Total amount of time spent writing WAL buffers
wal_sync_time	double precision	Total amount of time spent syncing WAL files.
stats_reset	timestamp with time zone	Time at which these statistics were last reset.

Example 12 Query pg_stat_wal view

```
postgres=> SELECT * FROM pg_stat_wal ;
-[ RECORD 1 ]----+------------------------------
wal_records      | 52417581
wal_fpi          | 98
wal_bytes        | 4139801371
wal_buffers_full | 341903
wal_write        | 342723
wal_sync         | 876
wal_write_time   | 0
wal_sync_time    | 0
stats_reset      | 2021-10-01 10:21:41.173493+09
```

3.1.2. Logical Replication

The following features have been added to Logical Replication.

□ Logical Streaming Replication

Traditional logical replication transfers updates to the subscription instance on every transaction commit. PostgreSQL 14 offers a logical streaming replication feature that transfers data without waiting for the transaction to be committed. For logical streaming replication, set the streaming attribute to 'on' when creating or modifying SUBSCRIPTION. The default value for the STREAMING attribute is 'off'.

Example 13 Streaming replication setting

```
postgres=# CREATE SUBSCRIPTION sub1 CONNECTION 'host=remhost1
        dbname=postgres user=postgres password=<<PASSWORD>>'
        PUBLICATION pub1 WITH (streaming = on) ;
CREATE SUBSCRIPTION
postgres=# SELECT subname, substream  FROM pg_subscription
        WHERE subname = 'sub1' ;
subname | substream
---------+-----------
 sub1    | t
(1 row)
```

□ Binary transfer

Traditional logical replication has converted data into text and transfers it. Binary transfer is possible by setting the attribute binary of SUBSCRIPTION to 'on'. This can be expected to reduce the amount of network transfer. If no attribute is specified, it will be transferred in text format as before.

Example 14 Binary transfer setting

```
postgres=# CREATE SUBSCRIPTION sub2 CONNECTION 'host=remhost1
        dbname=postgres user=postgres password=<<PASSWORD>>'
        PUBLICATION pub2 WITH (binary = on) ;
CREATE SUBSCRIPTION
postgres=# SELECT subname, subbinary  FROM pg_subscription
        WHERE subname = 'sub2' ;
subname | subbinary
---------+-----------
 sub2    | t
(1 row)
```

□ Separation of initial synchronization and differential update

Data initial synchronization and update information transactions have been separated. This eliminates the need to start over from the initial data synchronization when an error occurs. An additional temporary logical replication slot is required when migrating initial data. The name of this replication slot is "pg_%u_sync_%u_%llu" (SUBSCRIPTION oid, table relid, System ID).

□ Wait Events

The following wait events can now be confirmed.

Table 14 Added Wait Events

Wait Event name	Description
LogicalChangesRead	Waiting for reading from a logical change file.
LogicalChangesWrite	Waiting for writing to the logical change file.
LogicalSubxactRead	Waiting for a read from a logical sub-transaction file.
LogicalSubxactWrite	Waiting to write to a logical sub-transaction file.

□ Add Messages

The output plugin accepts a new parameter 'Messages' that controls if logical decoding messages are written into the replication stream.

3.1.3. Parallel Query

SELECT part of the REFRESH MATERIALIZED VIEW statement can now be executed in parallel. The example below is a log of the auto_explain module when the REFRESH MATERIALIZED VIEW statement is executed.

Example 15 Execution plan log

```
LOG:  duration: 6649.485 ms  plan:
        Query Text: REFRESH MATERIALIZED VIEW mview1;
        Gather  (cost=0.00..95721.67 rows=10000000 width=12)
          Workers Planned: 2
          ->  Parallel Seq Scan on data1  (cost=0.00..95721.67
rows=4166667 width=12)
```

3.1.4. LZ4 compression of TOAST columns

It is now possible to specify LZ4 as the compression method for TOAST column data. In order to perform LZ4 compression, PostgreSQL with --with-lz4 specified in the option of the 'configure' command is required. To specify the compression method for TOAST columns in the CREATE TABLE statement, specify pglz or lz4 in the COMPRESSION clause of the column attribute. The default value is determined by default_toast_compression. In the ALTER TABLE statement (or ALTER MATERIALIZED VIEW statement), the SET COMPRESSION clause can be specified in the column definition change syntax. Changing the compression method with the ALTER TABLE statement does not change the existing tuple.

Example 16 LZ4 compression

```
postgres=> CREATE TABLE compress1(c1 INT, c2 TEXT COMPRESSION lz4) ;
CREATE TABLE
postgres=> \d+ compress1
                                 Table "public.compress1"
 Column |  Type   | Collation | Nullable | Default | Storage  | Compression | …
--------+---------+-----------+----------+---------+----------+-------------+ …
 c1     | integer |           |          |         | plain    |             | …
 c2     | text    |           |          |         | extended | lz4         | …
Access method: heap

postgres=> ALTER TABLE compress1 ALTER COLUMN c2 SET COMPRESSION pglz ;
ALTER TABLE
postgres=> \d+ compress1
                                 Table "public.compress1"
 Column |  Type   | Collation | Nullable | Default | Storage  | Compression | …
--------+---------+-----------+----------+---------+----------+-------------+ …
 c1     | integer |           |          |         | plain    |             | …
 c2     | text    |           |          |         | extended | pglz        | …
Access method: heap
```

In the CREATE TABLE (LIKE) statement, specify the "LIKE table name INCLUDING COMPRESSION" clause to copy the table definition up to the compressed definition. Compression attribute of each column can be found in the attcompression column of pg_attribute catalog. 'p' is pglz compression and 'l' is LZ4 compression. To check the compression method of the column, execute the pg_column_compression function with the column name.

Example 17 Pg_column_compression function

```
postgres=> SELECT c1, pg_column_compression(c2) FROM compress1 ;
 c1  | pg_column_compression
-----+-----------------------
 100 | pglz
 200 | lz4
(2 rows)
```

3.1.5. Execution Plan

The following enhancements have been implemented to the SQL statement execution plan creation feature.

□ Asynchronous execution

Asynchronous processing can now be executed in ForeignScan processing. Improve performance by running ForeignScan in parallel when accessing data on remote servers. To use this feature, set the async_capable option to 'on' in the postgres_fdw module (default value is 'off').

Example 18 Asynchronous execution setting

```
postgres=# CREATE SERVER remsvr1 FOREIGN DATA WRAPPER postgres_fdw
       OPTIONS (host 'remhost1', port '5432', dbname 'postgres',
       async_capable 'on') ;
CREATE SERVER
```

It is displayed as Async Foreign Scan in the execution plan. In the example below, the partitions part1v1 and part1v2 contained in the partition table part1 are implemented as FOREIGN TABLE respectively.

Example 19 Execution plan for asynchronous execution

```
postgres=> EXPLAIN SELECT * FROM part1 ;
                                     QUERY PLAN
--------------------------------------------------------------------------------
 Append  (cost=100.00..306.87 rows=2482 width=42)
   ->  Async Foreign Scan on part1v1 part1_1  (cost=100.00..147.23 rows=1241
width=42)
   ->  Async Foreign Scan on part1v2 part1_2  (cost=100.00..147.23 rows=1241
width=42)
(3 rows)
```

□ Memoize

The new executor node type named Memoize has been added. The planner can include this node type in the plan to have the executor cache the results from inside the parameterized nested loop join. The use of this execution plan can be controlled by the parameter enable_memoize. The default value is 'on'. Currently, this execution plan is considered for regular joins and LATERAL type joins.

Example 20 Memoize node

```
postgres=> EXPLAIN SELECT COUNT(*),AVG(t1.unique1) FROM tenk1 t1 INNER JOIN
        tenk1 t2 ON t1.unique1 = t2.twenty WHERE t2.unique1 < 1000 ;
                                          QUERY PLAN
--------------------------------------------------------------------------------
Aggregate  (cost=450.10..450.11 rows=1 width=40)
   ->  Nested Loop  (cost=24.33..445.10 rows=1000 width=4)
         ->  Bitmap Heap Scan on tenk1 t2  (cost=24.04..381.54 …
               Recheck Cond: (unique1 < 1000)
               ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..23.79 …
                     Index Cond: (unique1 < 1000)
         ->  Memoize  (cost=0.30..1.85 rows=1 width=4)
               Cache Key: t2.twenty
               ->  Index Only Scan using tenk1_unique1 on tenk1 t1  (cost=…
                     Index Cond: (unique1 = t2.twenty)
(10 rows)
```

3.1.6. Extended Statistics

It is now possible to define extended statistics on expressions. The pg_stats_ext_exprs system view has been added to reference extended statistics with expressions.

Example 21 Extended statistics using expressions

```
postgres=> CREATE STATISTICS extstat1_data1 ON MOD(c1, 10),
       MOD(c2, 10) FROM data1 ;
CREATE STATISTICS
postgres=> ANALYZE data1 ;
ANALYZE
postgres=> SELECT * FROM pg_stats_ext_exprs ;
-[ RECORD 1 ]----------+----------------------------------------
schemaname              | public
tablename               | data1
statistics_schemaname   | public
statistics_name         | extstat1_data1
statistics_owner        | demo
expr                    | mod(c1, 10)
null_frac               | 0
avg_width               | 4
…
```

3.1.7. Data types

The following data types have been added / extended.

□ Maximum / minimum value of numeric data type

Infinity and -Infinity, which indicate the maximum and minimum values, can now be specified for numeric and float types. The value obtained by dividing a non-zero floating point number by the Infinity value, or the return value of zero when -Infinity is specified for the exp function and power function.

Example 22 Infinity/-Infinity value

```
postgres=> CREATE TABLE numeric1(c1 NUMERIC, c2 float) ;
CREATE TABLE
postgres=> INSERT INTO numeric1 VALUES ('Infinity', '-Infinity') ;
INSERT 0 1
postgres=> SELECT * FROM numeric1 WHERE c1='Infinity' ;
    c1    |    c2
----------+-----------
 Infinity | -Infinity
(1 row)

postgres=> SELECT 0.1 / '-Infinity', exp('-Infinity'),
              power(2, '-Infinity') ;
 ?column? | exp | power
----------+-----+-------
        0 |   0 |     0
(1 row)
```

□ Multirange type

A data type has been provided that indicates multiple ranges that do not overlap. The specific data types are as follows.

Table 15 Added data types

Data type name	**Description**
datemultirange	Date type range
int4multirange	Int type range
int8multirange	Bigint type range
nummultirange	Numeric type range
tsmultirange	Timestamp type range
tstzmultirange	Timestamp with time zone type range
anymultirange	Multi-range pseudo data type
anycompatiblemutirange	Any pseudo-type multiple range type

The following is an example of range specification by bigint type.

Example 23 Using multirange type

```
postgres=> SELECT '{[1, 5), (10, 20]}'::int8multirange ;
int8multirange
-----------------
 {[1,5),[11,21)}
(1 row)
postgres=> SELECT nummultirange( numrange(1, 10, '()'),
       numrange(15, 20, '[]')) ;
  nummultirange
------------------
 {(1,10),[15,20]}
(1 row)
```

□ Jsonb type

The access method by subscript for jsonb type has been enhanced.

Example 24 Enhancement of jsonb type

```
postgres=> SELECT ('[1, "2", null]'::jsonb)[1] ;
 jsonb
-------
 "2"
(1 row)

postgres=> SELECT ('{"age": 25}'::jsonb)['age'] ;
 jsonb
-------
 25
(1 row)
postgres=> SELECT ('{"email":"pgsql-hackers@postgresql.org",
              "phone":"+3012345678"}'::jsonb) ['phone'] ;
    jsonb
----------------
 "+3012345678"
(1 row)
```

□ Pg_lsn type

It is now possible to perform addition / subtraction calculations using the numeric type for values of the pg_lsn type.

Example 25 Calculation of pg_lsn type

```
postgres=> SELECT pg_current_wal_lsn(), pg_current_wal_lsn()
       + 256 ;
 pg_current_wal_lsn | ?column?
--------------------+-----------
 0/15D1F88          | 0/15D2088
(1 row)
```

□ Point type

The operators (<<|, >>|) that get strictly below/above can now be used for point types.

3.1.8. BRIN indexes

The following operator classes are now available for BRIN indexes.

□ Bloom Filter

The operator classes "{data type}_bloom_ops" that use Bloom filters have been added. The following attributes can be specified for this operator class.

Table 16 Added attributes

Attribute name	Description
n_distinct_per_range	Specifies the estimated number of non-null values in the block range. The default value is -0.1.
false_positive_rate	Specifies the desired false positive rate used by the index. The default value is 0.01.

Example 26 Using the Bloom filter in the BRIN index

```
postgres=> CREATE INDEX idx1_data1 ON data1 USING brin
       (c1 numeric_bloom_ops (false_positive_rate = 0.05,
        n_distinct_per_range = 100)) ;
CREATE INDEX
```

□ Min/Max-Multi Index

The operator classes "{data type}_minmax_multi_ops" have been added to store multiple minimum and maximum values. The following attributes can be specified for this operator class.

Table 17 Added attribute

Attribute name	Description
values_per_range	Specifies the maximum number of values to be stored by the index. The default value is 32, and values from 8 to 32 can be specified.

3.1.9. Wait Events

The following wait events have been added.

Table 18 Added Wait Events

Wait Event name	Type	Description
AppendReady	IPC	Waiting for subplan node preparation.
BaseBackupRead	IO	Waiting for base backup file to be read.
BufferIO	IPC	Waiting for buffer I/O to complete.
BufFileTruncate	IO	Waiting for buffer file truncation.
LogicalChangesRead	IO	Waiting to read the logical change file.
LogicalChangesWrite	IO	Waiting to write a logical change file.
LogicalSubxactRead	IO	Waiting to load a logical subtransaction.
LogicalSubxactWrite	IO	Waiting for a logical subtransaction to write.
WalReceiverExit	IPC	Waiting for the WAL receiver process to finish.

The following wait events have been retyped.

Table 19 Wait event with changed type

Wait Event name	Before	After	Note
WalReceiverWaitStart	Client	IPC	

3.1.10. Stored Procedure

The following functions have been added.

□ SPI_execute_extended

This API replaces the deprecated SPI_execute_plan_with_paramlist. It is possible to specify the re-owner used to fix the execution plan.

Syntax

```
int SPI_execute_extended(const char *src, const SPIExecuteOptions
*options)
```

□ SPI_cursor_parse_open

Set up a cursor using a query string and parameters.

Syntax

```
Portal SPI_cursor_parse_open(const char *name, const char *src,
       const SPIParseOpenOptions *options)
```

□ SPI_scroll_cursor_fetch

Fetch some rows from a cursor

Syntax

```
void SPI_scroll_cursor_fetch(Portal portal, FetchDirection
        direction, long count)
```

3.1.11. Pg_hba.conf file

By adding a backslash (\) at the end of a line, a single setting can now be described across multiple lines. The backslash must be the last character on the line.

Example 27 Line breaks in the pg_hba.conf file

```
# IPv4 local connections:
host    all            all            127.0.0.1/32          trust
host    postgres       postgres       192.168.1.219/32       \
                                                         md5
```

Hewlett Packard
Enterprise

3.1.12. LIBPQ library

The following extensions have been implemented in the libpq library.

□ Connection string target_session_attrs

The following values have been added to the settings of the target_session_attrs parameter.

Table 20 Added setting values

Value	Description
read-only	The destination database cannot be updated. The opposite of read-write.
primary	Connect to the primary instance of streaming replication.
standby	Connect to a standby instance of streaming replication.
prefer-standby	Connect if a standby instance exists. If the standby instance does not exist, connect to the primary instance.

□ Connection string sslcrldir

It is now possible to set the sslcrldir to specify the CRL storage directory for SSL connections. It can also be specified by the environment variable PGSSLCRLDIR.

□ Connection string sslsni

Configure Server Name Indication (SNI) for SSL connections. It can also be specified by the environment variable PGSSLSNI.

□ Pipeline mode

The libpq pipeline mode allows applications to avoid FE/BE protocol synchronization messages implicitly included in the old libpq API after each query. The following functions have been added.

Table 21 Added functions for pipeline mode

Function name	Description
PQpipelineStatus	Returns the status of the current pipeline mode.
PQenterPipelineMode	Enter pipeline mode.
PQexitPipelineMode	Exit pipeline mode.
PQpipelineSync	Request pipeline synchronization.

□ Debug log

The PQsetTraceFlags function has been added.

□ Sync message

The PQsendFlushRequest function has been added.

3.1.13. ECPG

DECLARE STATEMENT statement can now be specified for ECPG.

Example 28 DECLARE STATEMENT statement

```
EXEC SQL BEGIN DECLARE SECTION;
    char *selectStr = "SELECT c1 FROM data1";
    long f1;
EXEC SQL END DECLARE SECTION;

int main()
{
    EXEC SQL CONNECT TO postgres USER postgres;

    EXEC SQL DECLARE stmt1 STATEMENT;
    EXEC SQL PREPARE stmt1 FROM :selectStr;
    EXEC SQL DECLARE cur1 CURSOR FOR stmt1;
    EXEC SQL OPEN cur1;
    …
}
```

3.1.14. Roles

The following roles have been added.

□ Pg_database_owner

The pg_database_owner role has been added. Members of this role implicitly consist of the database owner. It is usually expected to be used in a template database.

□ Pg_read_all_data / pg_write_all_data

Added pg_read_all_data role, which allows reading all objects in the database, and pg_write_all_data role, which allows writing.

Hewlett Packard
Enterprise

3.1.15. Logfile

Additional information is now output to the log file.

□ Log_autovacuum_min_duration

Detailed information for each index is now output to the automatic VACUUM log (log_autovacuum_min_duration).

Example 29 Index information for automatic VACUUM log

```
LOG:  automatic vacuum of table "postgres.public.data1": index scans: 1
        pages: 0 removed, 5406 remain, 0 skipped due to pins, 0 skipped frozen
        tuples: 500000 removed, 500000 remain, 0 are dead but not yet removable, oldest xmin: 566
        buffer usage: 21775 hits, 0 misses, 1 dirtied
        index scan needed: 5406 pages from table (100.00% of total) had 500000 dead item identifiers removed
        index "idx1_data1": pages: 2745 in total, 0 newly deleted, 0 currently deleted, 0 reusable
        index "idx2_data1": pages: 2745 in total, 0 newly deleted, 0 currently deleted, 0 reusable
        avg read rate: 20.330 MB/s, avg write rate: 20.239 MB/s
…
```

□ Log_connection

Additional information about authentication is output to the connection log (log_connection).

Example 30 Authentication information log at the time of connection

```
LOG:  connection received: host=192.168.1.219 port=51524
LOG:  connection authenticated: identity="demo" method=md5 (/usr/local/pgsql/data/pg_hba.conf:91)
LOG:  connection authorized: user=demo database=postgres application_name=psql
```

Hewlett Packard
Enterprise

3.1.16. Process name

The information is appended to some process names in the instance.

□ WAL Receiver

The status is now output to the process name of the wal receiver process.

Example 31 Process name of WAL receiver

```
$ ps -ef|grep receiver | grep -v grep
postgres  93255  93249  0 14:34 ?        00:00:01 postgres:
walreceiver streaming 0/5D744F8
$
```

□ WAL sender

The replication command name is now output to the process name of the WAL sender process for logical replication.

Example 32 Process name of WAL sender

```
$ ps -ef|grep sender | grep -v grep
postgres  92995  92944  0 14:28 ?        00:00:00 postgres:
walsender postgres [local] START REPLICATION
$
```

3.1.17. Regular Expression

The complement-class escapes \D, \S, \W are now allowed within bracket expressions. Also, 'word' has been added as a character class. This is the equivalent of \w.

Example 33 'Word' character class

```
postgres=> SELECT 'abc def' ~ '[[:word:]] [[:word:]]' ;
 ?column?
----------
 t
(1 row)
```

3.1.18. Text search

The languages for text search have been increased. The following languages have been added to PostgreSQL 14.

Table 22 Added languages

Language name	Description
armenian	Armenian language
basque	Basque language
catalan	Catalan language
hindi	Hindi language
serbian	Serbian language
yiddish	Yiddish language

3.1.19. Test module

The PL/Sample module, which can be used as a template for procedure languages, has been added.

Example 34 PL/Sample module

```
$ cd src/test/modules/plsample
$ ls -l
total 24
drwxrwxr-x. 2 postgres postgres   26 Sep 28 06:00 expected
-rw-r--r--. 1 postgres postgres  440 Sep 28 05:57 Makefile
-rw-r--r--. 1 postgres postgres  469 Sep 28 05:57 plsample--1.0.sql
-rw-r--r--. 1 postgres postgres 5046 Sep 28 05:57 plsample.c
-rw-r--r--. 1 postgres postgres  177 Sep 28 05:57 plsample.control
-rw-r--r--. 1 postgres postgres  228 Sep 28 05:57 README
drwxrwxr-x. 2 postgres postgres   26 Sep 28 06:00 sql
```

3.1.20. Maximum password length

The maximum length limit for password strings used for client authentication has been removed.

3.1.21. LLVM

Now supports LLVM 12.

Hewlett Packard
Enterprise

3.2. *SQL Statement*

This section explains new features related to SQL statements.

3.2.1. ALTER CURRENT_ROLE

The statements that can specify CURRENT_USER clause, such as the ALTER AGGREGATE, ALTER CONVERSION, and ALTER DATABASE statements, can now also specify a CURRENT_ROLE clause.

Example 35 ALTER DATABASE statement

```
postgres=# ALTER DATABASE demodb OWNER TO CURRENT_ROLE ;
ALTER DATABASE
```

3.2.2. ALTER SUBSCRIPTION

The syntax for adding/removing publications for existing subscriptions has been added.
Specify ADD PUBLICATION clause or DROP PUBLICATION clause in the ALTER SUBSCRIPTION statement.

Syntax

```
ALTER SUBSCRIPTION subscription_name [ ADD | DROP ] PUBLICATION
        publication_name
```

3.2.3. ALTER TABLE

CONCURRENTLY clause can now be specified in the ALTER TABLE DETACH PARTITION statement. If CONCURRENTLY is specified, it will be executed at a lower lock level to avoid blocking other sessions that may be accessing the partitioned table. In this mode, two transactions are used internally and cannot be executed within a transaction block. Also, it cannot be executed on a partitioned table that contains the default partition.

If FINALIZE is specified, the previous DETACH CONCURRENTLY call that was canceled or interrupted is completed.

Syntax

```
ALTER TABLE table_name DETACH PARTITION partition_name
 [ FINALIZE | CONCURRENTLY ]
```

3.2.4. COPY FREEZE

Visibility Map is now updated with the execution of COPY FREEZE statement.

Example 36 Check visibility map after executing COPY FREEZE statement

```
postgres=# BEGIN ;
BEGIN
postgres=*# TRUNCATE TABLE visible1 ;
TRUNCATE TABLE
postgres=*# COPY visible1 FROM '/home/postgres/visible1.csv'
       CSV FREEZE ;
COPY 3
postgres=*# COMMIT ;
COMMIT
postgres=# SELECT * FROM pg_visibility_map('visible1', 0) ;
 all_visible | all_frozen
-------------+------------
 t           | t
(1 row)
```

3.2.5. CREATE INDEX

The INCLUDE clause can now be specified when creating SP-GiST indexes.

Example 37 INCLUDE clause with SP-GiST index creation

```
postgres=> CREATE INDEX idx1_gist1 ON gist1 USING spgist (c1)
       INCLUDE (c2) ;
CREATE INDEX
postgres=> \d+ idx1_gist1
                Index "public.idx1_gist1"
 Column | Type | Key? | Definition | Storage  | Stats target
--------+------+------+------------+----------+--------------
 c1     | text | yes  | c1         | extended |
 c2     | text | no   | c2         | extended |
spgist, for table "public.gist1"
```

3.2.6. CREATE PROCEDURE/FUNCTION

The following enhancements have been implemented for the CREATE PROCEDURE and CREATE FUNCTION statements.

□ OUT clause

It is now possible to specify the OUT clause as a parameter of PROCEDURE.

Example 38 CREATE PROCEDURE statement

```
postgres=> CREATE PROCEDURE sum_n_product(x int, y int, OUT sum
 int, OUT prod int) AS $$
    BEGIN
         sum := x + y ;
         prod := x * y ;
     END $$ LANGUAGE plpgsql ;
CREATE PROCEDURE
postgres=> SELECT proargmodes FROM pg_proc WHERE
       proname='sum_n_product' ;
 proargmodes
-------------
 {i,i,o,o}
(1 row)
postgres=> CALL sum_n_product(2, 4, NULL, NULL) ;
 sum | prod
-----+------
   6 |    8
(1 row)
```

□ LANGUAGE SQL

When the LANGUAGE SQL clause is specified in the CREATE FUNCTION statement and CREATE PROCEDURE statement, it is not necessary to enclose it in character literals, and SQL standard-compliant syntax is supported. The source for FUNCTION / PROCEDURE created with this syntax is stored in the prosqlbody column of the pg_proc catalog.

Example 39 LANGUAGE SQL

```
postgres=> CREATE PROCEDURE insert_data1(id INTEGER, val TEXT)
   LANGUAGE SQL
   BEGIN ATOMIC
      INSERT INTO data1 VALUES (id, val) ;
      INSERT INTO data1 VALUES (id, val) ;
   END ;
CREATE PROCEDURE
```

FUNCTION and PROCEDURE created in the standard format are aware of the dependencies of the objects they are used on. For this reason, when the DROP TABLE CASCADE statement is executed, FUNCTION with dependencies will be dropped at the same time.

Example 40 DROP TABLE CASCADE statement

```
postgres=> CREATE PROCEDURE insert_data1(id NUMERIC, val TEXT)
       LANGUAGE SQL
       BEGIN ATOMIC
        INSERT INTO data1 VALUES (id, val);
       END ;
CREATE PROCEDURE
postgres=> DROP TABLE data1 ;
ERROR:  cannot drop table data1 because other objects depend on it
DETAIL:   function  insert_data1(numeric,text)  depends  on  table
data1
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
postgres=> DROP TABLE data1 CASCADE ;
NOTICE:  drop cascades to function insert_data1(numeric,text)
DROP TABLE
```

3.2.7. CREATE TABLE

It is now possible to specify different values of COLLATION for partition boundaries.

Example 41 CREATE TABLE statement

```
postgres=>  CREATE TABLE part_coll  (c1 TEXT)  PARTITION BY RANGE
        (c1 COLLATE "POSIX") ;
CREATE TABLE
postgres=> CREATE TABLE part_coll_p1 PARTITION OF part_coll FOR
       VALUES FROM ('a' collate "C") to ('g') ;
CREATE TABLE
```

3.2.8. CREATE TRIGGER

The OR REPLACE clause can now be specified in the CREATE TRIGGER statement.

Example 42 CREATE OR REPLACE TRIGGER statement

```
postgres=> CREATE OR REPLACE TRIGGER data1_trig1 AFTER INSERT
       ON data1 FOR EACH ROW EXECUTE FUNCTION data1_ins_func1 () ;
CREATE TRIGGER
```

3.2.9. CREATE TYPE

SUBSCRIPT clause that specifies the MULTIRANGE_TYPE_NAME clause and generic subscript access for the function you specify more than one range type has been added to the CREATE TYPE statement.

Syntax

```
CREATE TYPE name AS RANGE (
  SUBTYPE = subtype
  [ , MULTIRANGE_TYPE_NAME = multirange_type_name ]
)
```

Syntax

```
CREATE TYPE name (
  INPUT = input_function,
  OUTPUT = output_function
  [ , SUBSCRIPT = subscript_function ]
)
```

3.2.10. GRANT / REVOKE

It is now possible to specify the GRANTED BY clause for granting roles with GRANT and REVOKE statements in accordance with the SQL standard. The GRANTED BY clause can only specify the current role that executes the GRANT statement, so it behaves the same as in previous versions.

Syntax

```
GRANT role_name [, …] TO role_specification
    [ WITH ADMIN OPTION ] [ GRANTED BY role_specification ]

REVOKE [ ADMIN OPTION FOR ] role_name [, …] FROM role_specification
    [ GRANTED BY role_specification ] [ CASCADE | RESTRICT ]
```

Example 43 GRANTED BY clause

```
postgres=> GRANT SELECT ON data1 TO user1 GRANTED BY CURRENT_USER ;
GRANT
postgres=> GRANT SELECT ON data1 TO user1 GRANTED BY postgres ;
ERROR:  grantor must be current user
```

3.2.11. INSERT

The following enhancements have been implemented in the INSERT statement.

□ ON CONFLICT clause

It is now possible to qualify a table name for a column specification in the WHERE clause in the ON CONFLICT clause.

Example 44 Table qualified with ON CONFLICT clause

```
postgres=> INSERT INTO data1 AS d1 VALUES (10, 'data1') ON CONFLICT
       (c1) WHERE d1.c2 = 'data1' DO NOTHING ;
INSERT 0 1
```

□ GENERATED ALWAYS clause

It is now possible to specify multi-tuple DEFAULT clause when executing an INSERT statement on a table that has a column with a GENERATED ALWAYS clause.

Example 45 Multiple tuple DEFAULT clause

```
postgres=> CREATE TABLE gen1(c1 INT PRIMARY KEY, c2 INT
       GENERATED ALWAYS AS (c1+1) STORED) ;
CREATE TABLE
postgres=> INSERT INTO gen1 VALUES (100, DEFAULT),(200, DEFAULT) ;
INSERT 0 2
postgres=> SELECT * FROM gen1 ;
 c1  | c2
-----+-----
 100 | 101
 200 | 201
(2 rows)
```

3.2.12. REINDEX

The following enhancements have been implemented to the REINDEX statement.

□ Partitioned table

It is now possible to specify a partitioned table for the REINDEX statement.

Example 46 Execute REINDEX statement on a partitioned table

```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10))
        PARTITION BY RANGE (c1) ;
CREATE TABLE
postgres=> CREATE TABLE part1v1 PARTITION OF part1 FOR VALUES
        FROM (0) TO (100000) ;
CREATE TABLE
postgres=> CREATE TABLE part1v2 PARTITION OF part1 FOR VALUES
        FROM (100000) TO (200000) ;
CREATE TABLE
postgres=> CREATE INDEX idx1_part1 ON part1(c1) ;
CREATE INDEX
postgres=> REINDEX TABLE part1 ;
REINDEX
```

□ Specifying the tablespace

Tablespaces for indexes can now be specified in the optional TABLESPACE clause.

Example 47 Specifying the tablespace

```
postgres=> REINDEX (TABLESPACE ts1, VERBOSE) INDEX idx1_data1 ;
INFO:  index "idx1_data1" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
REINDEX
postgres=> REINDEX (TABLESPACE ts1, VERBOSE) TABLE data1 ;
INFO:  index "data1_pkey" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
INFO:  index "idx1_data1" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
INFO:  index "pg_toast_16385_index" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
REINDEX
```

3.2.13. VACUUM

The following enhancements have been implemented in VACUUM.

□ TOAST table

The PROCESS_TOAST option can now be specified in the VACUUM statement. This option performs VACUUM processing on the TOAST table. This parameter is enabled by default. To disable processing of the TOAST table, specify the PROCESS_TOAST FALSE option.

Example 48 VACUUM (PROCESS_TOAST) statement

```
postgres=> VACUUM (PROCESS_TOAST FALSE, VERBOSE) data1 ;
INFO:  vacuuming "public.data1"
INFO:  "data1": found 0 removable, 37 nonremovable row versions in
1 out of 5406 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin:
518
There were 38 unused item identifiers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
VACUUM
```

□ Failsafe Mechanism

If it determines that the table's relfrozenxid or relminmxid is dangerously past, VACUUM bypasses non-freeze processing. The parameters vacuum_failsafe_age and vacuum_multixact_failsafe_age have been added as these thresholds.

□ Bypass index vacuum

Allow vacuum to skip index vacuuming when the number of removable index entries is insignificant. The vacuum parameter INDEX_CLEANUP has a new default of auto that enables this optimization.

Example 49 Autovacuum log at bypass index vacuum

```
LOG:  automatic vacuum of table "postgres.public.data1": index scans: 0
        pages: 0 removed, 5406 remain, 0 skipped due to pins, 0 skipped frozen
        tuples: 15000 removed, 985000 remain, 0 are dead but not yet removable, oldest xmin: 880
        index scan bypassed: 82 pages from table (1.52% of total) have 15000 dead item identifiers
        avg read rate: 0.095 MB/s, avg write rate: 0.191 MB/s
        buffer usage: 10793 hits, 2 misses, 4 dirtied
        WAL usage: 5407 records, 1 full page images, 356929 bytes
…
```

3.2.14. SELECT

The following enhancements have been implemented to the SELECT statement.

□ Support for reserved words

In PostgreSQL 14, AS clause is no longer required even when a reserved word is specified as an alias for a column in the SELECT statement. The number of reserved words that cannot be used for aliases has been significantly reduced.

Example 50 No AS clause required

```
postgres=> SELECT loc analyze FROM dept ;
 analyze
---------
(0 rows)
```

□ GROUP BY DISTINCT

It is now possible to specify DISTINCT to eliminate duplicates in the GROUP BY clause. It is also possible to specify ALL to output all (the default behavior is ALL).

Example 51 GROUP BY DISTINCT

```
postgres=> SELECT a, b, c FROM (VALUES (1, 2, 3),
        (4, NULL, 6), (7, 8, 9)) AS t (a, b, c)
        GROUP BY ALL ROLLUP(a, b), ROLLUP(a, c) ORDER BY a, b, c ;
a | b | c
---+---+---
 1 | 2 | 3
 1 | 2 |
…
   |   |
(25 rows)

postgres=> SELECT a, b, c FROM (VALUES (1, 2, 3),
        (4, NULL, 6), (7, 8, 9)) AS t (a, b, c) GROUP BY
        DISTINCT ROLLUP(a, b), ROLLUP(a, c) ORDER BY a, b, c ;
 a | b | c
---+---+---
 1 | 2 | 3
 1 | 2 |
…
   |   |
(13 rows)
```

□ SEARCH clause / CYCLE clause

The SEARCH and CYCLE clauses defined in the SQL standard can now be specified in the WITH clause of recursive SQL statements. The CYCLE clause is used for closed circuit detection, and the SEARCH clause can be used to specify depth-first search (DEPTH FIRST) or breadth-first search (BREADTH FIRST).

Example 52 SEARCH clause (DEPTH FIRST)

```
postgres=> CREATE TABLE graph0(f INT, t INT, label TEXT) ;
CREATE TABLE
postgres=> INSERT INTO graph0 VALUES (1, 2, 'arc 1 -> 2'),
      (1, 3, 'arc 1 -> 3'), (2, 3, 'arc 2 -> 3'),
      (1, 4, 'arc 1 -> 4'), (4, 5, 'arc 4 -> 5') ;
INSERT 0 5
postgres=> WITH RECURSIVE search_graph(f, t, label) AS (
       SELECT * FROM graph0 g
       UNION ALL
       SELECT g.* FROM graph0 g, search_graph sg WHERE
       g.f = sg.t
   ) SEARCH DEPTH FIRST BY f, t SET seq
   SELECT * FROM search_graph ORDER BY seq ;
 f | t |   label    |        seq
---+---+------------+-------------------
 1 | 2 | arc 1 -> 2 | {"(1,2)"}
 2 | 3 | arc 2 -> 3 | {"(1,2)","(2,3)"}
 1 | 3 | arc 1 -> 3 | {"(1,3)"}
 1 | 4 | arc 1 -> 4 | {"(1,4)"}
 4 | 5 | arc 4 -> 5 | {"(1,4)","(4,5)"}
 2 | 3 | arc 2 -> 3 | {"(2,3)"}
 4 | 5 | arc 4 -> 5 | {"(4,5)"}
(7 rows)
```

Example 53 SEARCH clause (BREADTH FIRST)

```
postgres=> WITH RECURSIVE search_graph(f, t, label) AS (
       SELECT * FROM graph0 g
       UNION ALL
       SELECT g.* FROM graph0 g, search_graph sg WHERE g.f = sg.t
   ) SEARCH BREADTH FIRST BY f, t SET seq
   SELECT * FROM search_graph ORDER BY seq ;
 f | t |    label    |   seq
---+---+------------+---------
 1 | 2 | arc 1 -> 2 | (0,1,2)
 1 | 3 | arc 1 -> 3 | (0,1,3)
 1 | 4 | arc 1 -> 4 | (0,1,4)
 2 | 3 | arc 2 -> 3 | (0,2,3)
 4 | 5 | arc 4 -> 5 | (0,4,5)
 2 | 3 | arc 2 -> 3 | (1,2,3)
 4 | 5 | arc 4 -> 5 | (1,4,5)
(7 rows)
```

Example 54 CYCLE clause

```
postgres=> CREATE TABLE graph1(f INT, t INT, label TEXT) ;
CREATE TABLE
postgres=> INSERT INTO graph1 VALUES
        (1, 2, 'arc 1 -> 2'), (1, 3, 'arc 1 -> 3'),
       (2, 3, 'arc 2 -> 3'),
        (1, 4, 'arc 1 -> 4'), (4, 5, 'arc 4 -> 5'),
       (5, 1, 'arc 5 -> 1') ;
INSERT 0 6
postgres=> WITH RECURSIVE search_graph(f, t, label) AS (
       SELECT * FROM graph1 g
       UNION ALL
       SELECT g.* FROM graph1 g, search_graph sg
       WHERE g.f = sg.t
   ) CYCLE f, t SET is_cycle TO TRUE DEFAULT FALSE USING PATH
   SELECT * FROM search_graph ;
 f | t |   label    | is_cycle |                path
---+---+------------+----------+-----------------------------------
 1 | 2 | arc 1 -> 2 | f        | {"(1,2)"}
 1 | 3 | arc 1 -> 3 | f        | {"(1,3)"}
 2 | 3 | arc 2 -> 3 | f        | {"(2,3)"}
 1 | 4 | arc 1 -> 4 | f        | {"(1,4)"}
…
 5 | 1 | arc 5 -> 1 | t        | {"(5,1)","(1,4)","(4,5)","(5,1)"}
 2   |   3   |   arc   2   ->   3   |   f                        |
{"(1,4)","(4,5)","(5,1)","(1,2)","(2,3)"}
(25 rows)
```

□ USING AS clause

It is now possible to use aliases for columns specified in the JOIN USING clause. Aliases are also available in the SELECT clause for the columns specified in the JOIN. This is a feature defined in SQL:2016 feature F404 "Range variable for common column names".

Example 55 JOIN USING AS clause

```
postgres=> SELECT d1.c1, x.c1, x.c2 FROM data1 d1 JOIN data2 d2
       USING (c1, c2) AS x ;
```

3.2.15. TRUNCATE

TRUNCATE statements can now be executed in FOREIGN TABLE if supported by FOREIGN DATA WRAPPER. The postgres_fdw module now supports the TRUNCATE statement.

Example 56 TRUNCATE statement for FOREIGN TABLE

```
postgres=> CREATE FOREIGN TABLE remote1(c1 NUMERIC, c2
       VARCHAR(10)) SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=> TRUNCATE TABLE remote1 ;
TRUNCATE TABLE
```

3.2.16. Functions

The following functions have been added / extended.

□ Bit_count

Outputs the number of bits that are turned on in the specified bit string or bytea type.

Syntax

```
bigint BIT_COUNT(bit | bytea)
```

Example 57 Bit_count function

```
postgres=> SELECT bit_count(B'0101011001'::bit(10)) ;
 bit_count
-----------
         5
(1 row)
```

□ Bit_xor

An aggregate function bit_xor has been added to calculate the XOR of a bit string. The bit_or and bit_and functions are already provided.

Syntax

```
integer | bit BIT_XOR(bigint | bit | integer | smallint)
```

Example 58 Bit_xor function

```
postgres=> SELECT SUM(c1), BIT_XOR(c1) FROM data1 ;
    sum     | bit_xor
------------+---------
 5000050000 |  100000
(1 row)
```

□ Date_bin

Truncates the entered timestamp at the specified interval. It is similar to the date_trunc function, but can be truncated at any interval.

Syntax

```
timestamp  with  time  zone  DATE_BIN(stride  interval,  source
timestamp with time zone, origin timestamp with time zone)
```

Example 59 Date_bin function

```
postgres=>  SELECT  date_bin('15  minutes',  TIMESTAMP'2020-02-11
15:44:17', TIMESTAMP'2001-01-01 00:02:30') ;
      date_bin
---------------------
 2020-02-11 15:32:30
(1 row)
```

□ Make_timestamp / make_timestamptz

These functions can now specify a negative value indicating the BC to the specified year.

Example 60 Create a date BC

```
postgres=> SELECT make_timestamp(-2, 12, 25, 10, 20, 30) ;
     make_timestamp
------------------------
 0002-12-25 10:20:30 BC
(1 row)
```

□ Split_part

It is now possible to specify a negative value for the third parameter of the split_part function. If a negative value is specified, the part from the right side of the string specified in the first parameter can be cut off.

Example 61 Split_part function

```
postgres=> SELECT split_part('www.postgresql.org', '.', -1) ;
split_part
------------
 org
(1 row)
```

□ Substring

It now conforms to the syntax formulated in the SQL standard, SQL:2003.

Syntax

```
text SUBSTRING(search_text text SIMILAR pattern text ESCAPE escape
text)
```

The execution result is the same as SUBSTRING(text FROM pattern FOR escape).

Example 62 Substring function

```
postgres=> SELECT substring('Thomas' SIMILAR '%#"o_a#"_' ESCAPE '#') ;
substring
-----------
 oma
(1 row)
```

□ Trim / ltrim / rtrim

The ltrim and rtrim functions can now be used for bytea types, and the trim function can now specify LEADING and TRAILING clauses for bytea types.

Example 63 Rtrim function

```
postgres=> SELECT rtrim('abc'::bytea, 'c'::bytea) ;
 rtrim
--------
 \x6162
(1 row)

postgres=> \df rtrim
                     List of functions
   Schema   | Name  | Result data type | Argument data types | Type
------------+-------+------------------+---------------------+---
 pg_catalog | rtrim | bytea            | bytea, bytea        | func
 pg_catalog | rtrim | text             | text                | func
 pg_catalog | rtrim | text             | text, text          | func
(3 rows)
```

□ Trim_array

The function trim_array has been added to remove the end of an array.

Syntax

```
anyarray TRIM_ARRAY(array anyarray, n integer)
```

Example 64 Trim_array function

```
postgres=> SELECT trim_array(ARRAY[1, 2, 3, 4, 5], 2) ;
trim_array
------------
 {1,2,3}
(1 row)
```

□ Unistr

The unistr function evaluates the specified Unicode escaped string.

Syntax

```
text UNISTR(text)
```

Example 65 Unistr function

```
postgres=> SELECT unistr('\0441\+00043B\u043E\043D') ;
 unistr
--------
 слон
(1 row)
```

□ String_to_table

The string_to_table function separates the string with the specified separator and converts it to a table.

Syntax

```
setoff STRING_TO_TABLE(string text, delimiter text [, null_string
text])
```

Example 66 String_to_table function

```
postgres=> SELECT string_to_table('ABC-+-DEF-+-GHI', '-+-',
 'abc') ;
 string_to_table
-----------------
 ABC
 DEF
 GHI
(3 rows)
```

□ Jsonb type ISO 8601 format date

Supports ISO 8601 date formats. This feature has been backported since PostgreSQL 13.1.

Example 67 Date format in ISO 8601 format

```
postgres => SELECT jsonb_path_query('"2017-03-10T12:34:56+3:10"',
'$.datetime()') ;
     jsonb_path_query
-----------------------------
 "2017-03-10T12:34:56+03:10"
(1 row)
```

□ Pg_get_wal_pause_state

Returns the paused state of recovery. Returns "not paused" if no pause was requested, "pause requested" if pause was requested but recovery is not paused, and "paused" if recovery is paused. In PostgreSQL 14, recovery is now paused if the required parameter values are missing.

Syntax

```
text PG_GET_WAL_REPLAY_PAUSE_STATE()
```

Example 68 Pg_get_wal_pause_state function

```
postgres=# SELECT pg_wal_replay_pause() ;
 pg_wal_replay_pause
---------------------


(1 row)

postgres=# SELECT pg_get_wal_replay_pause_state() ;
 pg_get_wal_replay_pause_state
-------------------------------
 paused
(1 row)
```

□ Pg_log_backend_memory_contexts

The pg_log_backend_memory_contexts function outputs the memory status of the process with the specified ID as a LOG level message.

Syntax

```
bool PG_LOG_BACKEND_MEMORY_CONTEXTS(pid integer)
```

Example 69 Pg_log_backend_memory_contexts function

```
postgres=# SELECT pg_log_backend_memory_contexts(pg_backend_pid()) ;
LOG:  logging memory contexts of PID 67300
STATEMENT:  SELECT pg_log_backend_memory_contexts(pg_backend_pid());
LOG:  level: 0; TopMemoryContext: 68704 total in 5 blocks; 14416 free (10
chunks); 54288 used
…
```

□ Pg_terminate_backend

The pg_terminate_backend function now has a parameter to specify a timeout. The function will return true if the session exits within the timeout.

Syntax

```
boolean PG_TERMINATE_BACKEND(pid integer,
        timeout bigint DEFAULT 0)
```

□ Pg_xact_commit_timestamp_origin

This function outputs the time stamp and replication origin for the transaction ID. There is a need to specify 'on' the track_commit_timestamp the execution of this function.

Syntax

```
record PG_XACT_COMMITTED_TIMESTAMP_ORIGIN(xid xid, OUT timestamp
timestamp with time zone, OUT roident oid)
```

Example 70 Pg_xact_commit_timestamp_origin function

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

postgres=# SELECT * FROM
 pg_xact_commit_timestamp_origin(txid_current()::text::xid) ;
 timestamp | roident
-----------+---------
           |
(1 row)
```

□ Pg_last_committed_xact

The function now outputs a roident column indicating the replication origin in the execution result.

Syntax

```
record PG_LAST_COMMITTED_XACT()
```

Example 71 Pg_last_committed_xact function

```
postgres=# SELECT * FROM pg_last_committed_xact() ;
 xid |           timestamp           | roident
-----+-------------------------------+---------
 515 | 2021-10-01 11:08:58.549866+09 |       0
(1 row)
```

□ Pg_get_catalog_foreign_keys

Added pg_get_catalog_foreign_keys function to retrieve information about foreign keys defined in a system catalog. This function can be run by general users.

□ Pg_create_logical_replication_slot

The parameter twophase has been added to support two-phase commit.

Example 72 Pg_create_logical_replication_slot function

```
postgres=# \dfS pg_create_logical_replication_slot
List of functions
-[ RECORD 1 ]-------+-------------------------------------------------------
Schema              | pg_catalog
Name                | pg_create_logical_replication_slot
Result data type    | record
Argument data types | slot_name name, plugin name, temporary boolean DEFAULT
false, twophase boolean DEFAULT false, OUT slot_name name, OUT lsn pg_lsn
Type                | func
```

3.3. Configuration parameters

In PostgreSQL 14 the following parameters have been changed.

3.3.1. Added parameters

The following parameters have been added.

Table 23 Added parameters

Parameter name	Description (context)	Default value
client_connection_check_interval	The time interval between optional checks that the client is still connected. (user)	0
compute_query_id	Compute query identifiers. (superuser)	auto
debug_discard_caches	Aggressively invalidate system caches for debugging purposes. (superuser)	0
default_toast_compression	Default compression for new columns. (user)	pglz
enable_async_append	Use of async append plans. (user)	on
enable_memoize	Enable Memoize node. (user)	on
huge_page_size	The size of huge page. (postmaster)	0
idle_session_timeout	The maximum allowed idle time. (user)	0
in_hot_standby	Shows whether hot standby is currently active. (internal)	off
log_recovery_conflict_waits	Logs standby recovery conflict waits. (sighup)	off
min_dynamic_shared_memory	Amount of dynamic shared memory reserved at startup. (postmaster)	0
recovery_init_sync_method	Sets the method for synchronizing the data directory before crash recovery. (sighup)	fsync
remove_temp_files_after_crash	Remove temporary files after backend crash. (sighup)	on
ssl_crl_dir	Location of the SSL certificate revocation list directory. (sighup)	''
track_wal_io_timing	Collects timing statistics for WAL I/O activity. (superuser)	off
vacuum_failsafe_age	Age at which VACUUM should trigger failsafe. (user)	1600000000
vacuum_multixact_failsafe_age	Multixact age at which VACUUM should trigger failsafe. (user)	1600000000

□ Compute_query_id

If the parameter value is set to on, a unique ID will be calculated for the query being executed. The query ID will be the same value for all SQLs that have the same meaning in the database. The query ID can be displayed by EXPLAIN VERBOSE statement, queryid column in pg_stat_activity view, %Q in log_line_prefix parameter, and so on. This is an adaptation of the functionality previously used by the pg_stat_statements module. The default value is 'auto', which is automatically enabled if modules such as pg_stat_statements require this parameter.

Example 73 Display query ID

```
postgres=# SET compute_query_id = on ;
SET
postgres=# EXPLAIN VERBOSE SELECT * FROM data1 ;
                              QUERY PLAN
------------------------------------------------------------------
 Seq  Scan  on  public.data1    (cost=0.00..15406.00  rows=1000000
width=12)
   Output: c1, c2
 Query Identifier: 3365166609774651210
(3 rows)
```

□ Idle_session_timeout

Specifies the timeout value in milliseconds for forcibly terminating idle sessions. The default value is 0, which means that no timeout will occur. During transaction execution, idle_in_transaction_session_timeout is used and idle_session_timeout is ignored.

Example 74 Idle timeout

```
postgres=> \set VERBOSITY verbose
postgres=> SET idle_session_timeout = 1000 ;
SET
postgres=> -- Wait one second.
postgres=> SELECT 1 ;
FATAL:  57P05: terminating connection due to idle-session timeout
LOCATION:  ProcessInterrupts, postgres.c:3356
server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
The connection to the server was lost. Attempting reset: Succeeded.
postgres=>
```

□ Log_recovery_conflict_waits

In an environment where the log_recovery_conflict_waits parameter is enabled on the standby instance, conflict information is output to the log when a conflict occurs when deadlock_timeout is exceeded. Also, if the conflict is resolved, it will be output to the log. The following is the output conflict information.

Example 75 Log of conflict state

```
-- When conflict occurs
LOG:  recovery still waiting after 1031.776 ms: recovery conflict
on snapshot
DETAIL:  Conflicting process: 53647.
 [49948]  CONTEXT:   WAL  redo  at  1/8ECE0138  for  Heap2/PRUNE:
latestRemovedXid  895  nredirected  0  ndead  167;  blkref  #0:  rel
1663/14892/17264, blk 108

-- When conflict is resolved
LOG:   recovery  finished  waiting  after  26073.257  ms:  recovery
conflict on snapshot
CONTEXT:  WAL redo at 1/8ECE0138 for Heap2/PRUNE: latestRemovedXid
895 nredirected 0 ndead 167; blkref #0: rel 1663/14892/17264, blk
108
```

□ Remove_temp_files_after_crash

Determines whether to remove the temporary files that were created when the backend process terminated abnormally. Conventionally, these files are not removed until the instance is restarted. The default value is 'on', which means that the temporary files will be deleted if the backend process terminates abnormally.

3.3.2. Changed parameters

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

Table 24 Changed parameters

Parameter name	Changes
log_line_prefix	Added %P to indicate the process ID of the parallel group leader.
	Added %Q to indicate query ID.
password_encryption	The set value on/true/1 has been removed.
restore_command	Dynamic changes are now possible by reloading the parameter file.
unix_socket_directories	Abstract namespaces starting with @ can now be specified.

3.3.3. Removed parameters

The following parameters have been removed. The parameter vacuum_cleanup_index_scale_factor is disabled since PostgreSQL 13.3.

Table 25 Removed parameters

Parameter name	Reason
operator_precedence_warning	It was deemed unnecessary as PostgreSQL 9.4 and earlier are no longer supported.
vacuum_cleanup_index_scale_factor	It can be specified in the index to maintain compatibility, but it has no effect.

3.3.4. Parameters with default values changed

The following parameters have changed default values.

Table 26 Parameters with default values changed

Parameter name	PostgreSQL 13	PostgreSQL 14	Note
checkpoint_completion_target	0.5	0.9	
password_encryption	md5	scram-sha-256	
server_version	13.4	14.0	
server_version_num	130004	140000	
vacuum_cost_page_miss	10	2	

3.4. Utilities

Describes the major enhancements of utility commands.

3.4.1. Configure

The following options have been added to the 'configure' command.

□ Specify SSL library

Added "--with-ssl={library name}" option to specify the SSL library. Also, conventional "--with-openssl" option has been left for compatibility.

Example 76 Configure command for SSL

```
$ ./configure --help | grep ssl
  --with-ssl=LIB          use LIB for SSL/TLS support (openssl)
  --with-openssl          obsolete spelling of --with-ssl=openssl
$
```

□ Column compression method

The --with-lz4 option has been added to take advantage of the LZ4 column compression feature. The compiler flag LZ4_CFLAGS and the linker flag LZ4_LIBS have been added as environment variables.

Example 77 Configure command for LZ4

```
$ ./configure --help | grep -i lz4
  --with-lz4              build with LZ4 support
  LZ4_CFLAGS  C compiler flags for LZ4, overriding pkg-config
  LZ4_LIBS    linker flags for LZ4, overriding pkg-config
$
```

3.4.2. Initdb

The following options have been added to the initdb command.

□ --no-instructions option

When this option is specified, a message indicating the instance start-up method will not be output.

Example 78 --no-instructions option

```
$ initdb -D data --no-instructions
The files belonging to this database system will be owned by user
"postgres".
This user must also own the server process.
…
initdb: warning: enabling "trust" authentication for local
connections
You can change this by editing pg_hba.conf or using the option -
A, or
--auth-local and --auth-host, the next time you run initdb.
```

□ --data-checksums option

In the message output when the --help option is specified, the --data-checksums option that specifies the checksum has been promoted from "Less commonly used options:" to the commonly used option "Options:".

Example 79 Checksum specification option

```
$ initdb --help
initdb initializes a PostgreSQL database cluster.


Usage:
  initdb [OPTION]... [DATADIR]


Options:
  -A, --auth=METHOD         default authentication method for local connections
      --auth-host=METHOD    default authentication method for local TCP/IP
connections
      --auth-local=METHOD   default authentication method for local-socket
connections
 [-D, --pgdata=]DATADIR     location for this database cluster
  -E, --encoding=ENCODING   set default encoding for new databases
  -g, --allow-group-access  allow group read/execute on data directory
  -k, --data-checksums      use data page checksums
      --locale=LOCALE       set default locale for new databases
…
```

□ --discard-cache option

Run the bootstrap backend with the debug_discard_caches=1 option. This takes a very long time and is only of use for debugging.

3.4.3. Pg_amcheck

The pg_amcheck command has been added. This command makes it easy to check the structure of tables and indexes against a database with the Contrib module amcheck installed. In the following example, it can be seen that the damaged part of data2 table.

Example 80 Pg_amcheck command

```
$ pg_amcheck --database=postgres --table=data2 --verbose
pg_amcheck: including database: "postgres"
pg_amcheck: in database "postgres": using amcheck version "1.3" in
schema "public"
pg_amcheck: checking heap table "postgres"."public"."data2"
heap table "postgres"."public"."data2", block 0, offset 17:
    xmin 3236212 equals or exceeds next valid transaction ID 0:546
heap table "postgres"."public"."data2", block 0, offset 48:
    line pointer to page offset 6282 is not maximally aligned
pg_amcheck:             checking             btree             index
"postgres"."pg_toast"."pg_toast_16399_index"
pg_amcheck:             checking             heap             table
"postgres"."pg_toast"."pg_toast_16399"
```

Many options can be specified for the pg_amcheck command other than the above example.

Table 27 Major options

Option name	Description
--all	Check all databases.
--index=PATTERN	Check matching index(es).
--schema=PATTERN	Check matching schema(s).
--on-error-stop	Stop checking at end of first corrupt page.
--parent-check	Check index parent/child relationships.
--startblock=BLOCK	Begin checking table(s) at the given block number.
--endblock=BLOCK	Check table(s) only up to the given block number.
--jobs=NUM	Use this many concurrent connections to the server.
--progress	Show progress information.

3.4.4. Pg_dump

The pg_dump command has been following enhancements are implemented.

□ Restore partitions

It is now possible to restore a single partition from the partitioned table as a table.

□ Dumping Extension settings

The --extension option has been added to limit the extension definitions to be dumped. Specify the pattern for the extension name.

Example 81 --extension option

```
$ pg_dump --extension=cube
--
-- PostgreSQL database dump
…
--
-- Name: cube; Type: EXTENSION; Schema: -; Owner: -
--

CREATE EXTENSION IF NOT EXISTS cube WITH SCHEMA public;
…
```

□ Multiple --verbose option

In the pg_dump command, pg_dumpall command, pg_restore command, and pg_rewind command, specifying the -v option multiple times will now increase the output level of the log.

3.4.5. Pg_dumpall

The pg_dumpall command now has an option --no-toast-compression that does not dump the compression settings for TOAST data.

3.4.6. Pg_rewind

It is now possible to specify a standby instance of streaming replication as the connection destination.

3.4.7. Psql

The following features have been added to the psql command.

□ Displaying access methods

The Access Method is now output to the output of the \di+, \dm+, \dt+, and \dtS+ commands.

Example 82 Display access method

```
postgres=> \di+
                                        List of relations
 Schema |    Name     | Type  | Owner | Table | Persistence | Access Method | …
--------+-------------+-------+-------+-------+-------------+---------------+ …
 public | idx1_data1 | index | demo  | data1 | permanent   | btree         | …
 public | idx1_data2 | index | demo  | data2 | permanent   | btree         | …
(2 rows)
postgres=> \dt+
                                           List of relations
 Schema |  Name   |       Type        | Owner | Persistence | Access Method | …
--------+---------+-------------------+-------+-------------+-----------+ …-
 public | data1   | table             | demo  | permanent   | heap          | …
 public | data2   | table             | demo  | permanent   | heap          | …
(2 rows)
```

□ \dt, \di command

It is now possible to specify for TOAST table and TOAST index.

Example 83 Specifying the TOAST table

```
postgres=> \dt pg_toast.pg_toast_16384
                List of relations
  Schema  |      Name      |    Type     |  Owner
----------+----------------+-------------+----------
 pg_toast | pg_toast_16384 | TOAST table | postgres
(1 row)

postgres=> \di pg_toast.pg_toast_1213_index
                        List of relations
  Schema  |        Name         | Type  |  Owner   |     Table
----------+---------------------+-------+----------+---------------
 pg_toast | pg_toast_1213_index | index | postgres | pg_toast_1213
(1 row)
```

□ \dX command

Outputs a list of extended statistics.

Example 84 List of extended statistics

```
postgres=> CREATE STATISTICS stat1_data1 ON c1, c2 FROM data1 ;
CREATE STATISTICS
postgres=> ANALYZE data1 ;
ANALYZE
postgres=> \dX
                      List of extended statistics
 Schema |    Name     |    Definition      | Ndistinct | Dependencies |   MCV
--------+-------------+--------------------+-----------+--------------+------
 public | stat1_data1 | c1, c2 FROM data1 | defined   | defined      | defined
(1 row)
```

□ \d+ command

When the table name is specified in the \d+ command, a compression setting column (Compression) is output between the Storage and Stats target columns. This column can be suppressed by setting the HIDE_TOAST_COMPRESSION variable to 'on'.

Example 85 \d+ command

```
postgres=> \d+ compress1
                                Table "public.compress1"
 Column |  Type   | Collation | Nullable | Default | Storage  | Compression | …
--------+---------+-----------+----------+---------+----------+-------------+ …
 c1     | integer |           |          |         | plain    |             | …
 c2     | text    |           |          |         | extended | lz4         | …
Access method: heap

postgres=> \set HIDE_TOAST_COMPRESSION on
postgres=> \d+ compress1
                           Table "public.compress1"
 Column |  Type   | Collation | Nullable | Default | Storage  | Stats target | …
--------+---------+-----------+----------+---------+----------+--------------+ …
 c1     | integer |           |          |         | plain    |              | …
 c2     | text    |           |          |         | extended |              | …
Access method: heap
```

□ \df, \do command

These commands can now add specify the data type.

Example 86 \df, \do command

```
postgres=> \do - pg_catalog.int4
                             List of operators
   Schema   | Name | Left arg type | Right arg type | Result type | Description
------------+------+---------------+----------------+-------------+-----------
 pg_catalog | -    |               | integer        | integer     | negate
(1 row)

postgres=> \df sum (numeric)
                       List of functions
   Schema   | Name | Result data type | Argument data types | Type
------------+------+------------------+---------------------+------
 pg_catalog | sum  | numeric          | numeric             | agg
(1 row)
```

□ \dT command

The \dT command now accepts alias names for data types.

Example 87 \dT command with alias

```
postgres=> \dT int
                           List of data types
   Schema   |  Name   |                 Description
------------+---------+---------------------------------------------
 pg_catalog | integer | -2 billion to 2 billion integer, 4-byte storage
(1 row)
```

□ Behavior when the editor is closed

The \e command (also \ef and \ev commands) launches the editor and no longer re-executes SQL statements in the file if the file is not modified.

3.4.8. Reindexdb

The --tablespace option has been added to the reindexdb command. It is now possible to specify the tablespace in which to store the indexes to be recreated.

Example 88 REINDEXDB --tablespace option

```
$ reindexdb --table=data1 --tablespace=ts1 --verbose postgres
INFO:  index "idx1_data1" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
INFO:  index "pg_toast_16385_index" was reindexed
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
```

3.4.9. Vacuumdb

The following options have been added to the vacuumdb command.

□ --force-index-cleanup option

Always remove index entries pointing to dead tuples.

□ --no-index-cleanup option

Do not delete unnecessary tuples in the index.

□ --no-truncate option

Do not truncate the free pages at the end of the table.

□ --no-process-toast option

Do not perform VACUUM processing on the TOAST table.

Example 89 Vacuumdb command options

```
$ vacuumdb --help
vacuumdb cleans and analyzes a PostgreSQL database.

Usage:
  vacuumdb [OPTION]... [DBNAME]

Options:
  -a, --all                     vacuum all databases
  -d, --dbname=DBNAME             database to vacuum
      --disable-page-skipping    disable all page-skipping behavior
  -e, --echo                    show the commands being sent to the server
  -f, --full                    do full vacuuming
  -F, --freeze                  freeze row transaction information
      --force-index-cleanup        always remove index entries that point to
dead tuples
  -j, --jobs=NUM                use this many concurrent connections to vacuum
      --min-mxid-age=MXID_AGE    minimum multixact ID age of tables to vacuum
      --min-xid-age=XID_AGE        minimum transaction ID age of tables to
vacuum
      --no-index-cleanup        don't remove index entries that point to dead
tuples
      --no-process-toast          skip the TOAST table associated with the
table to vacuum
      --no-truncate             don't truncate empty pages at the end of the
table
  -P, --parallel=PARALLEL_WORKERS use this many background workers for vacuum,
if available
…
```

3.5. Contrib modules

Describes new features related to the Contrib modules.

3.5.1. Amcheck

The verify_heapam function has been added to check the structure of the table.

Example 90 Verify_heapam function

```
postgres=# CREATE EXTENSION amcheck ;
CREATE EXTENSION
postgres=# SELECT verify_heapam('data1') ;
 verify_heapam
---------------
(0 rows)
```

3.5.2. Btree_gist

All functions provided by the btree_gist module is now parallel-safe.

3.5.3. Cube

The cube type now supports binary I/O.

Example 91 Cube type binary access

```
postgres=> CREATE EXTENSION cube ;
CREATE EXTENSION
postgres=> SELECT typname, typreceive, typsend FROM pg_type
       WHERE typname='cube' ;
 typname | typreceive |  typsend
---------+------------+-----------
 cube    | cube_recv  | cube_send
(1 row)
```

3.5.4. Hstore

It is now possible to access columns of type hstore using subscripts.

Example 92 Access using subscripts

```
postgres=> CREATE EXTENSION hstore ;
CREATE EXTENSION
postgres=> CREATE TABLE hstore1 (id INT, attr hstore) ;
CREATE TABLE
postgres=>  INSERT  INTO  hstore1  VALUES  (1,  '"attr1"=>"val1",
"attr2"=>"val2", "attr3"=>"val3"') ;
INSERT 0 1
postgres=> SELECT id, attr['attr1'], attr->'attr2' FROM hstore1 ;
 id | attr | ?column?
----+------+----------
  1 | val1 | val2
(1 row)
```

3.5.5. Old_snapshot

This is a newly added Contrib module. This module defines the pg_old_snapshot_time_mapping function. When the parameter old_snapshot_threshold is set to something other than -1, the mapping between XIDs and timestamps can be displayed.

Example 93 Pg_old_snapshot_time_mapping function

```
postgres=# SHOW old_snapshot_threshold ;
 old_snapshot_threshold
------------------------
 90min
(1 row)

postgres=# CREATE EXTENSION old_snapshot ;
CREATE EXTENSION
postgres=# SELECT * FROM pg_old_snapshot_time_mapping() ;
 array_offset |     end_timestamp      | newest_xmin
--------------+------------------------+-------------
            0 | 2021-10-01 10:13:00+09 |         533
            1 | 2021-10-01 10:14:00+09 |         534
            2 | 2021-10-01 10:15:00+09 |         535
            3 | 2021-10-01 10:16:00+09 |         535
(4 rows)
```

3.5.6. Pageinspect

The following functions for GiST indexes have been added.

Table 28 Added functions

Function name	Description
gist_page_opaque_info	Returns information about the opaque area.
gist_page_items	Returns the data in the page as record type.
gist_page_items_bytea	Returns the data in the page in bytea type.

Example 94 Information of GiST index

```
postgres=# SELECT * FROM
 gist_page_opaque_info(get_raw_page('idx1_gist1', 1)) ;
    lsn     |    nsn     | rightlink | flags
------------+------------+-----------+--------
 0/2216510 | 0/2216510 |       473 | {leaf}
(1 row)

postgres=# SELECT * FROM
 gist_page_items(get_raw_page('idx1_gist1', 1), 'idx1_gist1') ;
 itemoffset |    ctid     | itemlen |     keys
-------------+-------------+---------+--------------
          1 | (457,65535) |     144 | (c2)=('' '')
          2 | (249,65535) |     144 | (c2)=('' '')
          3 | (487,65535) |     144 | (c2)=('' '')
…
```

□ Output of bt_metap function

Additional information "last cleanup num del pages" is now output.

Example 95 Output of bt_metap function

```
postgres=# SELECT * FROM bt_metap('idx1_data1') ;
-[ RECORD 1 ]-------------+-------
magic                      | 340322
version                    | 4
root                       | 290
level                      | 2
fastroot                   | 290
fastlevel                  | 2
last_cleanup_num_delpages | 0
last_cleanup_num_tuples   | -1
allequalimage              | t
```

3.5.7. Passwordcheck

The reason is now output in the output of the check using the Cracklib library.

Example 96 Password check

```
postgres=# CREATE USER demo PASSWORD 'password123' ;
ERROR:  password is easily cracked
postgres=# \! tail -3 data/log/postgresql-2020-10-01_151319.log
ERROR:  password is easily cracked
DETAIL:  cracklib diagnostic: it is based on a dictionary word
STATEMENT:  CREATE USER demo PASSWORD 'password123';
```

3.5.8. Pgstattuple

The pgstattuple_approx function can now be executed on TOAST tables.

Example 97 Pgstattuple_approx function

```
postgres=#  SELECT  pgstattuple_approx((SELECT  reltoastrelid  FROM
pg_class WHERE relname='data1')) ;
  pgstattuple_approx
-----------------------
 (0,0,0,0,0,0,0,0,0,0)
(1 row)
```

3.5.9. Pg_stat_statements

The following features have been extended to the pg_stat_statements module. When using pg_stat_statements, compute_query_id parameter must be specified as 'on' or 'auto'.

□ Pg_stat_statements view

The following columns have been added.

Table 29 Added columns

Column name	Data type	Description
toplevel	boolean	True if the query was executed as a top level.

□ Track utility commands

Tracking is now available for REFRESH MATERIALIZED VIEW statement, CREATE TABLE AS statement, SELECT INTO statement, CREATE MATERIALIZED VIEW statement, etc.

Example 98 Tracking REFRESH MATERIALIZED VIEW statement

```
postgres=# REFRESH MATERIALIZED VIEW mview1 ;
REFRESH MATERIALIZED VIEW
postgres=#
postgres=# SELECT query, rows FROM pg_stat_statements WHERE
 query LIKE 'REFRESH%';
              query               |  rows
----------------------------------+---------
 REFRESH MATERIALIZED VIEW mview1 | 1000000
(1 row)
```

□ Pg_stat_statements_info view

The pg_stat_statements_info view has been added. This view can be used to check the operational status of the pg_stat_statements module. Currently, the pg_stat_statements view only provides a dealloc column showing the number of SQL statements deleted and a stats_reset column showing the date and time the statistics were reset. The 'dealloc' column is used to validate the pg_stat_statements.max parameter.

Example 99 Pg_stat_statements_info view

```
postgres=# \d pg_stat_statements_info
      View "public.pg_stat_statements_info"
 Column      |  Type                     | Collation | Nullable | Default
-------------+--------------------------+-----------+----------+---------
 dealloc     | bigint                   |           |          |
 stats_reset | timestamp with time zone |           |          |
```

3.5.10. Pg_trgm

Equality operators are now supported on GiST / GIN indexes.

Example 100 Index equation operator

```
postgres=> CREATE INDEX trgm_idx ON test_trgm
        USING gist (t gist_trgm_ops) ;
CREATE INDEX
postgres=> EXPLAIN ANALYZE SELECT * FROM test_trgm
        WHERE t='100000' ;
                              QUERY PLAN
-----------------------------------------------------------------
 Index Scan using trgm_idx on test_trgm  (cost=0.28..8.30 rows=1
width=5) (actual time=0.424..0.732 rows=1 loops=1)
   Index Cond: (t = '100000'::text)
   Rows Removed by Index Recheck: 2
 Planning Time: 0.038 ms
 Execution Time: 0.744 ms
(5 rows)
```

3.5.11. Pg_surgery

A new Contrib module, pg_surgery, has been added. It provides a function heap_force_freeze to force a tuple to freeze, and a function heap_force_kill to force it to be deleted.

Example 101 Pg_surgery Contrib module

```
postgres=# SELECT xmin, ctid, c1 FROM data1 ;
 xmin | ctid  | c1
------+-------+-----
  517 | (0,1) | 100
(1 row)
postgres=# SELECT heap_force_freeze('data1'::regclass,
                ARRAY['(0, 1)']::tid[]) ;
 heap_force_freeze
-------------------

(1 row)
postgres=# SELECT xmin, ctid, c1 FROM data1 ;
 xmin | ctid  | c1
------+-------+-----
    2 | (0,1) | 100
(1 row)
postgres=# SELECT heap_force_kill('data1'::regclass,
 ARRAY['(0, 1)']::tid[]) ;
 heap_force_kill
-----------------

(1 row)
postgres=# SELECT xmin, ctid, c1 FROM data1 ;
 xmin | ctid | c1
------+------+----
(0 rows)
```

3.5.12. Postgres_fdw

The following enhancements have been implemented for the postgres_fdw module.

□ Functions

The following functions have been added to control the active session.

Table 30 Added functions

Function name	Description
postgres_fdw_disconnect	Disconnect a specified active session.
postgres_fdw_disconnect_all	Disconnect all active sessions.
postgres_fdw_get_connections	Return all active sessions.

Example 102 Execution of postgres_fdw related functions

```
postgres=> CREATE FOREIGN TABLE remote1(c1 NUMERIC,
 c2 VARCHAR(10)) SERVER remsvr1 ;
CREATE FOREIGN TABLE
postgres=> SELECT COUNT(*) FROM remote1 ;
  count
---------
 1000000
(1 row)

postgres=> SELECT * FROM postgres_fdw_get_connections() ;
 server_name | valid
-------------+-------
 remsvr1     | t
(1 row)

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

□ TRUNCATE statement

The TRUNCATE statement can now be executed on an foreign table.

□ Supports Bulk Insert

In response to the addition of the Bulk Insert API to the Foreign Data Wrapper, batch_size can now be specified as an option for FOREIGN SERVER and FOREIGN TABLE that use postgres_fdw.

Example 103 Batch_size option setting

```
postgres=> CREATE FOREIGN TABLE data1(c1 NUMERIC, c2 VARCHAR(10)) SERVER svr1
        OPTIONS (batch_size '1000') ;
CREATE FOREIGN TABLE
postgres=> \d data1
                     Foreign table "public.data1"
 Column |        Type          | Collation | Nullable | Default | FDW options
--------+-----------------------+-----------+----------+---------+----------
 c1     | numeric              |          |         |        |
 c2     | character varying(10) |          |         |        |
Server: svr1
FDW options: (batch_size '1000')
```

The following interfaces have been added to the Foreign Data Wrapper for the bulk insert feature.

Table 31 Added Interface functions

Interface name	Description
ExecForeignBatchInsert	Execution of batch transmission process.
GetForeignModifyBatchSize	Acquisition of batch size.

□ keep_connections option

The option keep_connections controls remote connections after a transaction completes. The default value is 'on', which keeps the connection after the transaction is completed. Setting this option to 'off' closes the connection when the transaction completes.

□ IMPORT FOREIGN SCHEMA statement

Partitions can now be specified in the LIMIT TO and EXCEPT clauses.

□ Reconnect

If the session with the remote instance is found to be broken, it will now be reconnected.

Hewlett Packard Enterprise

URL list

The following websites are references to create this material.

□ Release Notes

https://www.postgresql.org/docs/14/release-14.html

□ Commitfests

https://commitfest.postgresql.org/

□ PostgreSQL 14 Manual

https://www.postgresql.org/docs/14/index.html

□ Git

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

□ GitHub

https://github.com/postgres/postgres

□ Announce of PostgreSQL 14 GA

https://www.postgresql.org/about/news/postgresql-14-released-2318/

□ Postgres Professional

https://habr.com/ru/company/postgrespro/blog/541252/

□ PostgreSQL 14 Open Items

https://wiki.postgresql.org/wiki/PostgreSQL_14_Open_Items

□ Qiita (Nuko@Yokohama)

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-14.html

□ SQL Notes

https://sql-info.de/postgresql/postgresql-14/articles-about-new-features-in-postgresql-14.html

□ Slack - postgresql-jp

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

Change history

Change history

Version	Date	Author	Description
0.1	Apr 18, 2021	Noriyoshi Shinoda	Create an internal review version. Reviewers: Tomoo Takahashi Akiko Takeshima (Hewlett Packard Enterprise Japan)
1.0	May 21, 2021	Noriyoshi Shinoda	Modification completed according to PostgreSQL 14 Beta 1.
1.1	Oct 5, 2021	Noriyoshi Shinoda	Modification completed according to PostgreSQL 14 GA.