PostgreSQL 14 New Features

With Examples (Beta 1)
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1</td>
<td>Modified catalogs</td>
<td>13</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Logical Replication</td>
<td>19</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Parallel Query</td>
<td>21</td>
</tr>
<tr>
<td>3.1.4</td>
<td>LZ4 compression of TOAST columns</td>
<td>22</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Execution Plan</td>
<td>23</td>
</tr>
<tr>
<td>3.1.6</td>
<td>Extended Statistics</td>
<td>25</td>
</tr>
<tr>
<td>3.1.7</td>
<td>Data types</td>
<td>25</td>
</tr>
<tr>
<td>3.1.8</td>
<td>BRIN indexes</td>
<td>28</td>
</tr>
<tr>
<td>3.1.9</td>
<td>Wait Events</td>
<td>29</td>
</tr>
<tr>
<td>3.1.10</td>
<td>Stored Procedure</td>
<td>29</td>
</tr>
<tr>
<td>3.1.11</td>
<td>Pg_hba.conf file</td>
<td>30</td>
</tr>
<tr>
<td>3.1.12</td>
<td>LIBPQ library</td>
<td>31</td>
</tr>
<tr>
<td>3.1.13</td>
<td>ECPG</td>
<td>32</td>
</tr>
<tr>
<td>3.1.14</td>
<td>Roles</td>
<td>32</td>
</tr>
<tr>
<td>3.1.15</td>
<td>Logfile</td>
<td>33</td>
</tr>
<tr>
<td>3.1.16</td>
<td>Process name</td>
<td>34</td>
</tr>
<tr>
<td>3.1.17</td>
<td>Regular Expression</td>
<td>35</td>
</tr>
<tr>
<td>3.1.18</td>
<td>Text search</td>
<td>35</td>
</tr>
<tr>
<td>3.1.19</td>
<td>Test module</td>
<td>35</td>
</tr>
<tr>
<td>3.1.20</td>
<td>Maximum password length</td>
<td>36</td>
</tr>
<tr>
<td>3.1.21</td>
<td>LLVM</td>
<td>36</td>
</tr>
<tr>
<td>3.2</td>
<td>SQL Statement</td>
<td>37</td>
</tr>
<tr>
<td>3.2.1</td>
<td>ALTER CURRENT_ROLE</td>
<td>37</td>
</tr>
<tr>
<td>3.2.2</td>
<td>ALTER SUBSCRIPTION</td>
<td>37</td>
</tr>
<tr>
<td>3.2.3</td>
<td>ALTER TABLE</td>
<td>37</td>
</tr>
<tr>
<td>3.2.4</td>
<td>COPY FREEZE</td>
<td>38</td>
</tr>
<tr>
<td>3.2.5</td>
<td>CREATE INDEX</td>
<td>39</td>
</tr>
<tr>
<td>3.2.6</td>
<td>CREATE PROCEDURE/FUNCTION</td>
<td>39</td>
</tr>
<tr>
<td>3.2.7</td>
<td>CREATE TABLE</td>
<td>41</td>
</tr>
<tr>
<td>3.2.8</td>
<td>CREATE TRIGGER</td>
<td>42</td>
</tr>
<tr>
<td>3.2.9</td>
<td>CREATE TYPE</td>
<td>42</td>
</tr>
<tr>
<td>3.2.10</td>
<td>GRANT / REVOKE</td>
<td>43</td>
</tr>
<tr>
<td>3.2.11</td>
<td>INSERT</td>
<td>43</td>
</tr>
<tr>
<td>3.2.12</td>
<td>REINDEX</td>
<td>44</td>
</tr>
<tr>
<td>3.2.13</td>
<td>VACUUM</td>
<td>45</td>
</tr>
<tr>
<td>3.2.14</td>
<td>SELECT</td>
<td>47</td>
</tr>
</tbody>
</table>
3.2.15. TRUNCATE ................................................................. 52
3.2.16. Functions ............................................................... 52
3.3. Configuration parameters ............................................. 61
  3.3.1. Added parameters .................................................. 61
  3.3.2. Changed parameters ............................................... 63
  3.3.3. Removed parameters ............................................... 64
  3.3.4. Parameters with default values changed ....................... 64
3.4. Utilities ................................................................. 65
  3.4.1. Configure ............................................................ 65
  3.4.2. Initdb ............................................................... 65
  3.4.3. Pg_amcheck ......................................................... 67
  3.4.4. Pg_dump ............................................................. 68
  3.4.5. Pg_rewind .......................................................... 68
  3.4.6. Psql ................................................................. 68
  3.4.7. Reindexdb .......................................................... 72
  3.4.8. Vacuumdb .......................................................... 72
3.5. Contrib modules ....................................................... 74
  3.5.1. Amcheck ............................................................ 74
  3.5.2. Bool_plperl ........................................................ 74
  3.5.3. Btree_gist .......................................................... 75
  3.5.4. Cube ............................................................... 75
  3.5.5. Hstore ............................................................. 75
  3.5.6. Old_snapshot ....................................................... 76
  3.5.7. Pageinspect ......................................................... 76
  3.5.8. Passwordcheck ...................................................... 78
  3.5.9. Pgstattuple ......................................................... 78
  3.5.10. Pg_stat_statements ............................................... 78
  3.5.11. Pg_trgm ........................................................... 79
  3.5.12. Pg_surgery ......................................................... 80
  3.5.13. Postgres_fdw ...................................................... 81
URL list ................................................................................. 84
Change history ...................................................................... 85
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) Beta 1.

1.2 Audience

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

1.3 Scope

This document describes the major difference between PostgreSQL 13 (13.3) and PostgreSQL 14 Beta 1 (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>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>PostgreSQL 13.3 (for comparison)</td>
</tr>
<tr>
<td></td>
<td>PostgreSQL 14 (14.0) Beta 1 (May 17, 2021 20:15:58)</td>
</tr>
<tr>
<td>Operating System</td>
<td>Red Hat Enterprise Linux 7 Update 8 (x86-64)</td>
</tr>
<tr>
<td>Configure option</td>
<td>--with-llvm --with-ssl=openssl --with-perl --with-lz4</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Shell prompt for Linux root user.</td>
</tr>
<tr>
<td>$</td>
<td>Shell prompt for Linux general user.</td>
</tr>
<tr>
<td><strong>Bold</strong></td>
<td>The user input string.</td>
</tr>
<tr>
<td>postgres=#</td>
<td>psql command prompt for PostgreSQL administrator.</td>
</tr>
<tr>
<td>postgres=&gt;</td>
<td>psql command prompt for PostgreSQL general user.</td>
</tr>
<tr>
<td><strong>Underline</strong></td>
<td>Important output items.</td>
</tr>
<tr>
<td>&lt;&lt;PASSWORD&gt;&gt;</td>
<td>Replaced by password string.</td>
</tr>
</tbody>
</table>

The syntax is described in the following rules:

Table 3 Syntax rules

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Italic</em></td>
<td>Replaced by the name of the object which users use, or the other syntax.</td>
</tr>
<tr>
<td>[]</td>
<td>Indicate that it can be omitted.</td>
</tr>
<tr>
<td>{ A</td>
<td>B }</td>
</tr>
<tr>
<td>…</td>
<td>General syntax. It is the same as the previous version.</td>
</tr>
</tbody>
</table>
2. New Features Summary

More than 200 new features have been added to PostgreSQL 14 Beta 1. 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! ;
ERROR:  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.
Table 4 Warning of transaction ID wraparound

<table>
<thead>
<tr>
<th>Comparison</th>
<th>PostgreSQL 13</th>
<th>PostgreSQL 14</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warning</td>
<td>11 million</td>
<td>40 million</td>
<td></td>
</tr>
<tr>
<td>Shutdown</td>
<td>One million</td>
<td>3 million</td>
<td></td>
</tr>
</tbody>
</table>

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 also 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)

```sql
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

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

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

Table 6 Added views in the information schema (information_schema)

<table>
<thead>
<tr>
<th>View name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>routine_column_usage</td>
<td>Identifies the column used by the function or procedure. Currently not tracked.</td>
</tr>
<tr>
<td>routineroutine_usage</td>
<td>Identifies the function used by the function or procedure. Currently, some are not tracked.</td>
</tr>
<tr>
<td>routine_sequence_usage</td>
<td>Identifies the sequence used by a function or procedure. Currently, some are not tracked.</td>
</tr>
<tr>
<td>routine_table_usage</td>
<td>Identifies the table used by the function or procedure. Currently not tracked.</td>
</tr>
</tbody>
</table>
Table 7 System catalogs/views with additional columns

<table>
<thead>
<tr>
<th>Catalog/View name</th>
<th>Added column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_attribute</td>
<td>attcompression</td>
<td>char</td>
<td>Current compression method of the column.</td>
</tr>
<tr>
<td>pg_inherits</td>
<td>inhdetachpending</td>
<td>boolean</td>
<td>True for a partition that is in the process of being detached</td>
</tr>
<tr>
<td>pg_locks</td>
<td>waitstart</td>
<td>timestamp with time zone</td>
<td>Time when the server process started waiting for this lock.</td>
</tr>
<tr>
<td>pg_prepared_statements</td>
<td>generic_plans</td>
<td>bigint</td>
<td>Number of times generic plan was chosen.</td>
</tr>
<tr>
<td></td>
<td>custom_plans</td>
<td>bigint</td>
<td>Number of times custom plan was chosen</td>
</tr>
<tr>
<td>pg_proc</td>
<td>prosqlbody</td>
<td>pg_node_tree</td>
<td>Pre-parsed SQL function body.</td>
</tr>
<tr>
<td>pg_range</td>
<td>rngmultitypid</td>
<td>oid</td>
<td>OID of the multirange type.</td>
</tr>
<tr>
<td>pg_replication_slots</td>
<td>two_phase</td>
<td>boolean</td>
<td>Enabled for decoding prepared transactions.</td>
</tr>
<tr>
<td>pg_stat_activity</td>
<td>query_id</td>
<td>bigint</td>
<td>Identifier of this backend's most recent query.</td>
</tr>
<tr>
<td>pg_stat_database</td>
<td>session_time</td>
<td>double precision</td>
<td>Time spent by database sessions in this database.</td>
</tr>
<tr>
<td></td>
<td>active_time</td>
<td>double precision</td>
<td>Time spent executing SQL statements in this database.</td>
</tr>
<tr>
<td></td>
<td>idle_in_transaction</td>
<td>double precision</td>
<td>Time spent idling while in a transaction in this database.</td>
</tr>
<tr>
<td></td>
<td>sessions</td>
<td>bigint</td>
<td>Total number of sessions.</td>
</tr>
<tr>
<td></td>
<td>sessions_abandoned</td>
<td>bigint</td>
<td>Number of lost sessions.</td>
</tr>
<tr>
<td></td>
<td>sessions_fatal</td>
<td>bigint</td>
<td>Number of sessions caused by fatal error.</td>
</tr>
<tr>
<td></td>
<td>sessions Killed</td>
<td>bigint</td>
<td>Number of sessions terminated by the operator.</td>
</tr>
</tbody>
</table>
### Table 8 System catalogs/views with columns removed

<table>
<thead>
<tr>
<th>Catalog/View name</th>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_stat_ssl</td>
<td>compression</td>
<td>It was deleted because compression was disabled.</td>
</tr>
</tbody>
</table>

### Table 9 System catalogs/views with modified output

<table>
<thead>
<tr>
<th>Catalog/View name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_class</td>
<td>The initial value of the reltuples column has been changed to -1.</td>
</tr>
<tr>
<td>pg_type</td>
<td>The typtype column may output 'm' to indicate a multi-range type.</td>
</tr>
<tr>
<td>pg_subscription_rel</td>
<td>'f' = finished table copy may be output in the srssubstrate column.</td>
</tr>
<tr>
<td>pg_stat_activity</td>
<td>Information of the archiver process and the walsender process will be output.</td>
</tr>
<tr>
<td>pg_statistic_ext</td>
<td>Extended statistics with the 'e' expression may be output in the stxkind column.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>text</td>
<td>Name of the memory context.</td>
</tr>
<tr>
<td>ident</td>
<td>text</td>
<td>Identification information of the memory context.</td>
</tr>
<tr>
<td>parent</td>
<td>text</td>
<td>Name of the parent of this memory context.</td>
</tr>
<tr>
<td>level</td>
<td>integer</td>
<td>Distance from TopMemoryContext.</td>
</tr>
<tr>
<td>total_bytes</td>
<td>bigint</td>
<td>Total bytes allocated for this memory context.</td>
</tr>
<tr>
<td>total_nblinks</td>
<td>bigint</td>
<td>Total number of blocks allocated for this memory context.</td>
</tr>
<tr>
<td>free_bytes</td>
<td>bigint</td>
<td>Free space in bytes.</td>
</tr>
<tr>
<td>free_chunks</td>
<td>bigint</td>
<td>Total number of free chunks.</td>
</tr>
<tr>
<td>used_bytes</td>
<td>bigint</td>
<td>Used space in bytes.</td>
</tr>
</tbody>
</table>

Example 9 Query pg_backend_memory_contexts view

```
postgres=# SELECT name, parent, level, total_bytes FROM pg_backend_memory_contexts ;
```

<table>
<thead>
<tr>
<th>name</th>
<th>parent</th>
<th>level</th>
<th>total_bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TopMemoryContext</td>
<td>null</td>
<td>0</td>
<td>68704</td>
</tr>
<tr>
<td>TopTransactionContext</td>
<td>TopMemoryContext</td>
<td>1</td>
<td>8192</td>
</tr>
<tr>
<td>CFuncHash</td>
<td>TopMemoryContext</td>
<td>1</td>
<td>8192</td>
</tr>
<tr>
<td>Operator lookup cache</td>
<td>TopMemoryContext</td>
<td>1</td>
<td>24576</td>
</tr>
<tr>
<td>TableSpace cache</td>
<td>TopMemoryContext</td>
<td>1</td>
<td>8192</td>
</tr>
<tr>
<td>Type information cache</td>
<td>TopMemoryContext</td>
<td>1</td>
<td>24376</td>
</tr>
<tr>
<td>Record information cache</td>
<td>TopMemoryContext</td>
<td>1</td>
<td>8192</td>
</tr>
<tr>
<td>RowDescriptionContext</td>
<td>TopMemoryContext</td>
<td>1</td>
<td>8192</td>
</tr>
<tr>
<td>MessageContext</td>
<td>TopMemoryContext</td>
<td>1</td>
<td>32768</td>
</tr>
</tbody>
</table>

Pg_stat_progress_copy

This view will output the execution status of the COPY statement.
Table 11 Pg_stat_progress_copy view

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

Example 10 Query pg_stat_progress_copy view

```sql
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

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot_name</td>
<td>text</td>
<td>Identifier for the replication slot.</td>
</tr>
<tr>
<td>spill_txns</td>
<td>bigint</td>
<td>Number of transactions spilled to disk.</td>
</tr>
<tr>
<td>spill_count</td>
<td>bigint</td>
<td>Number of times transactions were spilled to disk.</td>
</tr>
<tr>
<td>spill_bytes</td>
<td>bigint</td>
<td>Amount of decoded transaction data spilled to disk.</td>
</tr>
<tr>
<td>stream_txns</td>
<td>bigint</td>
<td>Number of in-progress transactions streamed.</td>
</tr>
<tr>
<td>stream_count</td>
<td>bigint</td>
<td>Number of times in-progress transactions.</td>
</tr>
<tr>
<td>stream_bytes</td>
<td>bigint</td>
<td>Amount of decoded in-progress transaction data streamed.</td>
</tr>
<tr>
<td>total_txns</td>
<td>bigint</td>
<td>Number of decoded transactions.</td>
</tr>
<tr>
<td>total_bytes</td>
<td>bigint</td>
<td>Amount of decoded transactions data.</td>
</tr>
<tr>
<td>stats_reset</td>
<td>timestamp</td>
<td>Time at which these statistics were last reset.</td>
</tr>
</tbody>
</table>

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'.

© 2020-2021 Hewlett Packard Enterprise Japan Co, Ltd.
Table 13 Pg_stat_wal view

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

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 | 2021-05-20 23:21:41.173493+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 settings

```sql
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)
```

square 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 settings

```sql
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**

<table>
<thead>
<tr>
<th>Wait Event name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogicalChangesRead</td>
<td>Waiting for reading from a logical change file.</td>
</tr>
<tr>
<td>LogicalChangesWrite</td>
<td>Waiting for writing to the logical change file.</td>
</tr>
<tr>
<td>LogicalSubxactRead</td>
<td>Waiting for a read from a logical sub-transaction file.</td>
</tr>
<tr>
<td>LogicalSubxactWrite</td>
<td>Waiting to write to a logical sub-transaction file.</td>
</tr>
</tbody>
</table>

□ 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

<table>
<thead>
<tr>
<th>c1</th>
<th>pg_column_compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>pglz</td>
</tr>
<tr>
<td>200</td>
<td>lz4</td>
</tr>
</tbody>
</table>

(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 settings

```sql
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

```sql
postgres=> EXPLAIN SELECT * FROM part1 ;

QUERY PLAN

Append  (cost=100.00..280.97 rows=1742 width=70)
  ->  Async Foreign Scan on part1v1 part1_1  (cost=100.00..136.13 rows=871 width=70)
  ->  Async Foreign Scan on part1v2 part1_2  (cost=100.00..136.13 rows=871 width=70)
(3 rows)
```

Result Cache

The new executor node type named Result Cache 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_resultcache. The default value is 'on'. Currently, this execution plan is considered for regular joins and LATERAL type joins.

Example 20 Result Cache node

```sql
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=508.86..508.87 rows=1 width=40)
  ->  Nested Loop  (cost=0.30..503.86 rows=1000 width=4)
    ->  Seq Scan on tenk1 t2  (cost=0.00..470.00 rows=1000 width=4)
        Filter: (unique1 < 1000)
    ->  Result Cache  (cost=0.30..0.43 rows=1 width=4)
        Cache Key: t2.twenty
    ->  Index Only Scan using tenk1_unique1 on tenk1 t1
        (cost=0.29..0.42 rows=1 width=4)
        Index Cond: (unique1 = t2.twenty)
(8 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

```sql
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>
<thead>
<tr>
<th>Data type name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datemultirange</td>
<td>Date type range</td>
</tr>
<tr>
<td>int4multirange</td>
<td>Int type range</td>
</tr>
<tr>
<td>int8multirange</td>
<td>Bigint type range</td>
</tr>
<tr>
<td>nummultirange</td>
<td>Numeric type range</td>
</tr>
<tr>
<td>tsmultirange</td>
<td>Timestamp type range</td>
</tr>
<tr>
<td>tstzmultirange</td>
<td>Timestamp with time zone type range</td>
</tr>
<tr>
<td>anymultirange</td>
<td>Multi-range pseudo data type</td>
</tr>
<tr>
<td>anycompatiblemultirange</td>
<td>Any pseudo-type multiple range type</td>
</tr>
</tbody>
</table>

The following is an example of range specification by bigint type.
Example 23 Using multirange type

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

```sql
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 expanded.

Example 24 Extend jsonb type

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

```sql
postgres=> SELECT '{}"age": 25"::jsonb)["age"] ;
jsonb
------
25
(1 row)
```

```sql
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() |
|---------------+--------------------|
| 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

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_distinct_per_range</td>
<td>Specifies the estimated number of non-null values in the block range. The default value is 0.1.</td>
</tr>
<tr>
<td>false_positive_rate</td>
<td>Specifies the desired false positive rate used by the index. The default value is 0.01.</td>
</tr>
</tbody>
</table>

Example 26 Using the Bloom filter in the BRIN index

<table>
<thead>
<tr>
<th>postgres=&gt; CREATE INDEX idx1_data1 ON data1 USING brin (c1 numeric_bloom_ops (false_positive_rate = 0.05, n_distinct_per_range = 100)) ;</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE INDEX</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>values_per_range</td>
<td>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.</td>
</tr>
</tbody>
</table>

3.1.9. Wait Events

The following wait events have been added.

Table 18 Added Wait Events

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

The following wait events have been retyped.

Table 19 Wait event with changed type

<table>
<thead>
<tr>
<th>Wait Event name</th>
<th>Before</th>
<th>After</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>WalReceiverWaitStart</td>
<td>Client</td>
<td>IPC</td>
<td></td>
</tr>
</tbody>
</table>

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

```c
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

```c
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

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

SPI_scroll_cursor_move

Move a cursor.

Syntax

```c
void SPI_scroll_cursor_move(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.
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
```

### 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>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read-only</td>
<td>The destination database cannot be updated. The opposite of read-write.</td>
</tr>
<tr>
<td>primary</td>
<td>Connect to the primary instance of streaming replication.</td>
</tr>
<tr>
<td>standby</td>
<td>Connect to a standby instance of streaming replication.</td>
</tr>
<tr>
<td>prefer-standby</td>
<td>Connect if a standby instance exists. If the standby instance does not exist, connect to the primary instance.</td>
</tr>
</tbody>
</table>

- **Connection string sslcrldir**
  - It is now possible to set the sslcrldir to specify the CRL storage directory for SSL connections.

- **Connection string sslsni**
  - Configure Server Name Indication (SNI) for SSL connections.

- **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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PQpipelineStatus</td>
<td>Returns the status of the current pipeline mode.</td>
</tr>
<tr>
<td>PQenterPipelineMode</td>
<td>Enter pipeline mode.</td>
</tr>
<tr>
<td>PQexitPipelineMode</td>
<td>Exit pipeline mode.</td>
</tr>
<tr>
<td>PQpipelineSync</td>
<td>Request pipeline synchronization.</td>
</tr>
</tbody>
</table>

- **Debug log**
  The PQtraceSetFlags function has been added.

### 3.1.13. ECPG

DECLARE STATEMENT statement can now be specified for ECPG.

#### Example 28 DECLARE STATEMENT statement

```sql
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.
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.

The `pg_read_all_data` and `pg_write_all_data` roles have been added. These roles allow reading all objects in the database and writing, respectively.

### 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
    system usage: CPU: user: 0.26 s, system: 0.00 s, elapsed:
    0.56 s
    WAL usage: 21687 records, 0 full page images, 5144230 bytes
```
- 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/pqsql/data/pg_hba.conf:91) |
| LOG: connection authorized: user=demo database=postgres |
| application_name=psql |

### 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

<table>
<thead>
<tr>
<th>Language name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>armenian</td>
<td>Armenian language</td>
</tr>
<tr>
<td>basque</td>
<td>Basque language</td>
</tr>
<tr>
<td>catalan</td>
<td>Catalan language</td>
</tr>
<tr>
<td>hindi</td>
<td>Hindi language</td>
</tr>
<tr>
<td>serbian</td>
<td>Serbian language</td>
</tr>
<tr>
<td>yiddish</td>
<td>Yiddish language</td>
</tr>
</tbody>
</table>

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 May 18 05:14 expected
-rw-r--r-. 1 postgres postgres 440 May 18 05:11 Makefile
-rw-r--r-. 1 postgres postgres 469 May 18 05:11 plsample--1.0.sql
-rw-r--r-. 1 postgres postgres 5046 May 18 05:11 plsample.c
-rw-r--r-. 1 postgres postgres 177 May 18 05:11 plsample.control
-rw-r--r-. 1 postgres postgres 228 May 18 05:11 README
drwxrwxr-x. 2 postgres postgres 26 May 18 05:14 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.
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

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

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

The following new features have been implemented in the ALTER TABLE / CREATE TABLE statement.

- DETACH PARTITION clause
  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 ]
```

- **Automatic VACUUM for partitioned table**

  In the past, automatic VACUUM has ignored partitioned tables, but by propagating the analysis results of leaf partitions to higher levels, automatic VACUUM is made aware of these tables. The table options `autovacuum_enabled`, `autovacuum_analyze_scale_factor`, and `autovacuum_analyze_threshold` can now be specified for partitioned tables.

**Example 36 Automatic VACUUM configuration for partitioned table**

```
postgres=> CREATE TABLE part1(c1 NUMERIC, c2 VARCHAR(10))
    PARTITION BY RANGE(c1) ;
CREATE TABLE
postgres=> ALTER TABLE part1 SET (autovacuum_enabled = off) ;
ALTER TABLE
```

### 3.2.4. COPY FREEZE

Visibility Map is now updated with the execution of COPY FREEZE statement.
Example 37 Check visibility map after executing COPY FREEZE statement

```
begin;
begin
truncate table visible1;
truncate table *
COPY visible1 FROM '/home/postgres/visible1.csv' CSV FREEZE;
COPY 3
commit;
select * from pg_visibility_map('visible1', 0);
```

<table>
<thead>
<tr>
<th>all_visible</th>
<th>all_frozen</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>

(1 row)

### 3.2.5. CREATE INDEX

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

Example 38 INCLUDE clause with SP-GiST index creation

```
create index idx1_gist1 on gist1 using spgist (c1)
    include (c2);
create index idx1_gist1

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 39 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.
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 41 DROP TABLE CASCADE statement

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

3.2.7. CREATE TABLE

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

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

Example 43 CREATE OR REPLACE TRIGGER statement

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

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

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

Syntax

```sql
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

<table>
<thead>
<tr>
<th>GRANT role_name [, ...] TO role_specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ WITH ADMIN OPTION ] [ GRANTED BY role_specification ]</td>
</tr>
</tbody>
</table>

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

Example 44 GRANTED BY clause

```sql
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 45 Table qualified with ON CONFLICT clause

```sql
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 46 Multiple tuple DEFAULT clause

```sql
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 47 Execute REINDEX statement on a partitioned table

```sql
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 48 Specifying tablespace

```
postgres=> REINDEX (TABLESPACE ts1, VERBOSE) INDEX idx1_datal ;
INFO:  index "idx1_datal" 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_datal" 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 49 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.
```

- **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**
  Bypasses VACUUM processing on indexes if pages with dead tuples are less than 2% of the table.

Example 50 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: 10000 removed, 990000 remain, 0 are dead but not yet removable, oldest xmin: 610
     buffer usage: 10803 hits, 2 misses, 4 dirtied
     index scan not needed: 55 pages from table (1.02% of total)
     had 10000 dead item identifiers removed
     avg read rate: 0.095 MB/s, avg write rate: 0.191 MB/s
```
### 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.

**Example 51 No AS clause required**

```sql
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 52 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 53 SEARCH clause (DEPTH FIRST)

```sql
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 ;
```

<table>
<thead>
<tr>
<th>f</th>
<th>t</th>
<th>label</th>
<th>seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>arc 1 -&gt; 2</td>
<td>&quot;(1,2)&quot;</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>arc 2 -&gt; 3</td>
<td>&quot;(1,2)&quot;,&quot;(2,3)&quot;</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>arc 1 -&gt; 3</td>
<td>&quot;(1,3)&quot;</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>arc 1 -&gt; 4</td>
<td>&quot;(1,4)&quot;</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>arc 4 -&gt; 5</td>
<td>&quot;(1,4)&quot;,&quot;(4,5)&quot;</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>arc 2 -&gt; 3</td>
<td>&quot;(2,3)&quot;</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>arc 4 -&gt; 5</td>
<td>&quot;(4,5)&quot;</td>
</tr>
</tbody>
</table>

(7 rows)
Example 54 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 ;
```

<table>
<thead>
<tr>
<th>f</th>
<th>t</th>
<th>label</th>
<th>seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>arc 1 -&gt; 2</td>
<td>(0,1,2)</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>arc 1 -&gt; 3</td>
<td>(0,1,3)</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>arc 1 -&gt; 4</td>
<td>(0,1,4)</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>arc 2 -&gt; 3</td>
<td>(0,2,3)</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>arc 4 -&gt; 5</td>
<td>(0,4,5)</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>arc 2 -&gt; 3</td>
<td>(1,2,3)</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>arc 4 -&gt; 5</td>
<td>(1,4,5)</td>
</tr>
</tbody>
</table>

(7 rows)
Example 55 CYCLE clause

```sql
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 ;
```

<table>
<thead>
<tr>
<th>f</th>
<th>t</th>
<th>label</th>
<th>is_cycle</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>arc 1 -&gt; 2</td>
<td>f</td>
<td>&quot;(1,2)&quot;</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>arc 1 -&gt; 3</td>
<td>f</td>
<td>&quot;(1,3)&quot;</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>arc 2 -&gt; 3</td>
<td>f</td>
<td>&quot;(2,3)&quot;</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>arc 1 -&gt; 4</td>
<td>f</td>
<td>&quot;(1,4)&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(25 rows)</td>
</tr>
</tbody>
</table>

□ 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 56 JOIN USING AS clause

```sql
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 57 TRUNCATE statement for FOREIGN TABLE

```sql
postgres=> CREATE FOREIGN TABLE remotel(c1 NUMERIC, c2 VARCHAR(10)) SERVER remsvrl ;
postgres=> TRUNCATE TABLE remotel ;
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**

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

Example 58 Bit_count function

```sql
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 59 Bit_xor function

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

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 60 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
```

Make_timestamp / make_timestamptz

These functions can now specify a negative value indicating the BC to the specified year.
Example 61 Create a date BC

```sql
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 62 Split_part function

```sql
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 63 Substring function

```sql
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 64 Rtrim function

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

```
postgres=> \df rtrim
```

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Result data type</th>
<th>Argument data types</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_catalog</td>
<td>rtrim</td>
<td>bytea</td>
<td>bytea, bytea</td>
<td>func</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>rtrim</td>
<td>text</td>
<td>text</td>
<td>func</td>
</tr>
<tr>
<td>pg_catalog</td>
<td>rtrim</td>
<td>text</td>
<td>text, text</td>
<td>func</td>
</tr>
</tbody>
</table>

(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 65 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 66 Unistr function**

```
postgres=> SELECT unistr('\0441\+0043B\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 67 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 68 Date format in ISO 8601 format

```sql
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

```sql
text PG_GET_WAL_REPLAY_PAUSE_STATE()```

Example 69 Pg_log_backend_memory_contexts function

```sql
postgres=# SELECT pg_wal_replay_pause() ;
pg_wal_replay_pause
----------------------
(1 row)
postgres=# SELECT pg_get_wal_replay_pause_state() ;
paget_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.
The `pg_terminate_backend` function now has a parameter to specify a timeout. The `pg_wait_for_backend_termination` function will wait for the session to terminate until the timeout (default 5 seconds) without requesting the backend process to stop. The function will return true if the session exits within the timeout.

### Syntax

```c
bool PG_TERMINATE_BACKEND(pid integer, timeout bigint DEFAULT 5000)
bool PG_WAIT_FOR_BACKEND_TERMINATION(pid integer, timeout bigint DEFAULT 5000)
```

### 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

```c
record PG_XACT_COMMITTED_TIMESTAMP_ORIGIN(xid xid, OUT timestamp timestamp with time zone, OUT roident oid)
```
Example 71 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
-----------+--------
2021-05-21 11:08:58.549866+09 | 0
(1 row)
```

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

**Syntax**

```sql
record PG_LAST_COMMITTED_XACT()
```

Example 72 Pg_last_committed_xact function

```
postgres=# SELECT * FROM pg_last_committed_xact() ;
xid | timestamp | roident
-----+-----------+--------
515 | 2021-05-21 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 73 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>
<thead>
<tr>
<th>Parameter name</th>
<th>Description (context)</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_connection_check_interval</td>
<td>The time interval between optional checks that the client is still connected. (user)</td>
<td>0</td>
</tr>
<tr>
<td>compute_query_id</td>
<td>Compute query identifiers. (superuser)</td>
<td>auto</td>
</tr>
<tr>
<td>debug_invalidate_system_caches_always</td>
<td>Aggressively invalidate system caches for debugging purposes. (superuser)</td>
<td>0</td>
</tr>
<tr>
<td>default_toast_compression</td>
<td>Default compression for new columns. (user)</td>
<td>pglz</td>
</tr>
<tr>
<td>enable_async_append</td>
<td>Use of async append plans. (user)</td>
<td>on</td>
</tr>
<tr>
<td>enable_resultcache</td>
<td>Use of result caching. (user)</td>
<td>on</td>
</tr>
<tr>
<td>huge_page_size</td>
<td>The size of huge page. (postmaster)</td>
<td>0</td>
</tr>
<tr>
<td>idle_session_timeout</td>
<td>The maximum allowed idle time. (user)</td>
<td>0</td>
</tr>
<tr>
<td>in_hot_standby</td>
<td>Shows whether hot standby is currently active. (internal)</td>
<td>off</td>
</tr>
<tr>
<td>log_recovery_conflict_waits</td>
<td>Logs standby recovery conflict waits. (sighup)</td>
<td>off</td>
</tr>
<tr>
<td>min_dynamic_shared_memory</td>
<td>Amount of dynamic shared memory reserved at startup. (postmaster)</td>
<td>0</td>
</tr>
<tr>
<td>recovery_init_sync_method</td>
<td>Sets the method for synchronizing the data directory before crash recovery. (postmaster)</td>
<td>fsync</td>
</tr>
<tr>
<td>remove_temp_files_after_crash</td>
<td>Remove temporary files after backend crash. (sighup)</td>
<td>on</td>
</tr>
<tr>
<td>ssl_crl_dir</td>
<td>Location of the SSL certificate revocation list directory. (sighup)</td>
<td>&quot;</td>
</tr>
<tr>
<td>track_wal_io_timing</td>
<td>Collects timing statistics for WAL I/O activity. (superuser)</td>
<td>off</td>
</tr>
<tr>
<td>vacuum_failsafe_age</td>
<td>Age at which VACUUM should trigger failsafe. (user)</td>
<td>1600000000</td>
</tr>
<tr>
<td>vacuum_multixact_failsafe_age</td>
<td>Multixact age at which VACUUM should trigger failsafe. (user)</td>
<td>1600000000</td>
</tr>
</tbody>
</table>
□ 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 74 Display query ID

```sql
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
```

□ 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 75 Idle timeout

```sql
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 76 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

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

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

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator_precedence_warning</td>
<td>It was deemed unnecessary as PostgreSQL 9.4 and earlier are no longer supported.</td>
</tr>
<tr>
<td>vacuum_cleanup_index_scale_factor</td>
<td>It can be specified in the index to maintain compatibility, but it has no effect.</td>
</tr>
</tbody>
</table>

3.3.4. Parameters with default values changed

The following parameters have changed default values.

Table 26 Parameters with default values changed

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>PostgreSQL 13</th>
<th>PostgreSQL 14</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkpoint_completion_target</td>
<td>0.5</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>password_encryption</td>
<td>md5</td>
<td>scram-sha-256</td>
<td></td>
</tr>
<tr>
<td>server_version</td>
<td>13.3</td>
<td>14.0beta1</td>
<td></td>
</tr>
<tr>
<td>server_version_num</td>
<td>130003</td>
<td>140000</td>
<td></td>
</tr>
<tr>
<td>vacuum_cost_page_miss</td>
<td>10</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
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 77 Configure command for SSL

```bash
$ ./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 78 Configure command for LZ4

```bash
$ ./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 79 --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 --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 80 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
...
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 81 Pg_amcheck command

```bash
$ 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>
<thead>
<tr>
<th>Option name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--all</td>
<td>Check all databases.</td>
</tr>
<tr>
<td>--index=PATTERN</td>
<td>Check matching index(es).</td>
</tr>
<tr>
<td>--schema=PATTERN</td>
<td>Check matching schema(s).</td>
</tr>
<tr>
<td>--on-error-stop</td>
<td>Stop checking at end of first corrupt page.</td>
</tr>
<tr>
<td>--parent-check</td>
<td>Check index parent/child relationships.</td>
</tr>
<tr>
<td>--startblock=BLOCK</td>
<td>Begin checking table(s) at the given block number.</td>
</tr>
<tr>
<td>--endblock=BLOCK</td>
<td>Check table(s) only up to the given block number.</td>
</tr>
<tr>
<td>--jobs=NUM</td>
<td>Use this many concurrent connections to the server.</td>
</tr>
<tr>
<td>--progress</td>
<td>Show progress information.</td>
</tr>
</tbody>
</table>
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 dump extensions only. Specify the pattern for the extension name.

**Example 82 --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_rewind*

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

3.4.6. *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 83 Added display of access method

```
postgres=> \di+
List of relations
-------------+---------+--------+--------+---------------------------+--------+---------------------------+-------------+--------+---------------------------+
| Schema | Name     | Type   | Owner  | Table       | Persistence | Access Method  | 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 84 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_index |
(1 row)
```
\dX command

Outputs a list of extended statistics.

Example 85 List of extended statistics

```
pages=> CREATE STATISTICS stat1_data1 ON c1, c2 FROM data1 ;
CREATE STATISTICS
postgres=> ANALYZE data1 ;
ANALYZE
postgres=> \dX

List of extended statistics

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Definition</th>
<th>Ndistinct</th>
<th>Dependencies</th>
<th>MCV</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>stat1_data1</td>
<td>c1, c2 FROM data1</td>
<td>defined</td>
<td>defined</td>
<td>defined</td>
</tr>
</tbody>
</table>
```

\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 86 \d+ command

postgres=> \d+ compress1

Table "public.compress1"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
<th>Default</th>
<th>Storage</th>
<th>Stats target</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
<td>plain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c2</td>
<td>text</td>
<td></td>
<td></td>
<td></td>
<td>extended</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Access method: heap

postgres=> \set HIDE_TOAST_COMPRESSION on

postgres=> \d+ compress1

Table "public.compress1"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
<th>Default</th>
<th>Storage</th>
<th>Stats target</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
<td>plain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c2</td>
<td>text</td>
<td></td>
<td></td>
<td></td>
<td>extended</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Access method: heap

\df, \do command

These commands can now add specify the data type.

Example 87 \df, \do command

postgres=> \do - pg_catalog.int4

List of operators

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Left arg type</th>
<th>Right arg type</th>
<th>Result type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_catalog</td>
<td>-</td>
<td>integer</td>
<td>integer</td>
<td>negate</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

postgres=> \df sum (numeric)

List of functions

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Result data type</th>
<th>Argument data types</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_catalog</td>
<td>sum</td>
<td>numeric</td>
<td>numeric</td>
<td>agg</td>
</tr>
</tbody>
</table>

(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.

### 3.4.7. 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**

```bash
$ 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
```

### 3.4.8. Vacuumdb

The following options have been added to the vacuumdb command.

- `--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
- `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_DEGREE` 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. Bool_plperl

New Contrib modules bool_plperl and bool_plperlu have been added. Since bool_plperl is a trusted module, it can be executed by general users with the CREATE EXTENSION statement.

Example 91 Bool_plperlu module

```
postgres=> CREATE EXTENSION bool_plperlu ;
CREATE EXTENSION
postgres=> CREATE FUNCTION perl2text(text) RETURNS bool
    LANGUAGE plperl
    TRANSFORM FOR TYPE bool
    AS $$
        return shift ;
    $$ ;
CREATE FUNCTION
postgres=> SELECT perl2text('f') ;
perl2text
---------
t
(1 row)
```
3.5.3. Btree_gist
All functions provided by the btree_gist module is now parallel-safe.

3.5.4. Cube
The cube type now supports binary I/O.

Example 92 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)
```

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

Example 93 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["attr1"] |   attr['attr2'] |
--------+-----------------+-----------------|
        1 |    val1         |    val2         |
(1 row)
```
3.5.6. 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 94 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            | 2021-05-21 10:13:00+09 | 533         |
| 1            | 2021-05-21 10:14:00+09 | 534         |
| 2            | 2021-05-21 10:15:00+09 | 535         |
| 3            | 2021-05-21 10:16:00+09 | 535         |

(4 rows)
```

3.5.7. Pageinspect

The following functions for GiST indexes have been added.

Table 28 Added functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gist_pageOpaqueInfo</td>
<td>Returns information about the opaque area.</td>
</tr>
<tr>
<td>gist_page_items</td>
<td>Returns the data in the page as record type.</td>
</tr>
<tr>
<td>gist_page_itemsBytea</td>
<td>Returns the data in the page in bytea type.</td>
</tr>
</tbody>
</table>
Example 95 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 96 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
```
3.5.8. Passwordcheck

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

Example 97 Password check

```
postgres=# CREATE USER demo PASSWORD 'password123' ;
ERROR:  password is easily cracked
postgres=# \! tail -3 data/log/postgresql-2020-05-21_151319.log
ERROR:  password is easily cracked
DETAIL:  cracklib diagnostic: it is based on a dictionary word
STATEMENT:  CREATE USER demo PASSWORD 'password123';
```

3.5.9. Pgstattuple

The pgstattuple_approx function can now be executed on TOAST tables.

Example 98 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)
```

3.5.10. 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

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toplevel</td>
<td>boolean</td>
<td>True if the query was executed as a top level.</td>
</tr>
</tbody>
</table>
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 99 Tracking REFRESH MATERIALIZED VIEW statement

```sql
postgres=# REFRESH MATERIALIZED VIEW mview1 ;
REFRESH MATERIALIZED VIEW
postgres=#
postgres=# SELECT query, rows FROM pg_stat_statements WHERE query LIKE 'REFRESH%';

<table>
<thead>
<tr>
<th>query</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFRESH MATERIALIZED VIEW mview1</td>
<td>1000000</td>
</tr>
</tbody>
</table>

(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 100 Pg_stat_statements_info view

```sql
postgres=\d pg_stat_statements_info
View "public.pg_stat_statements_info"
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dealloc</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stats_reset</td>
<td>timestamp with time zone</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

3.5.11. Pg_trgm

Equality operators are now supported on GiST / GIN indexes.
Example 101 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)

3.5.12. 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 102 Pg_surgery Contrib module

```sql
postges=# SELECT xmin, ctid, c1 FROM data1 ;
xmin | ctid  | c1
-------+-------+-----
517 | (0,1) | 100
(1 row)

postges=# SELECT heap_force_freeze('data1'::regclass, ARRAY['(0, 1)']::tid[]) ;
heap_force_freeze
-------------------
(1 row)

postges=# SELECT xmin, ctid, c1 FROM data1 ;
xmin | ctid  | c1
-------+-------+-----
2 | (0,1) | 100
(1 row)

postges=# SELECT heap_force_kill('data1'::regclass, ARRAY['(0, 1)']::tid[]) ;
heap_force_kill
-----------------
(1 row)

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

3.5.13. 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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>postgres_fdw_disconnect</td>
<td>Disconnect a specified active session.</td>
</tr>
<tr>
<td>postgres_fdw_disconnect_all</td>
<td>Disconnect all active sessions.</td>
</tr>
<tr>
<td>postgres_fdw_get_connections</td>
<td>Return all active sessions.</td>
</tr>
</tbody>
</table>

Example 103 Execution of postgres_fdw related functions

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

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

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

- 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 104 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

<table>
<thead>
<tr>
<th>Interface name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecForeignBatchInsert</td>
<td>Execution of batch transmission process.</td>
</tr>
<tr>
<td>GetForeignModifyBatchSize</td>
<td>Acquisition of batch size.</td>
</tr>
</tbody>
</table>

☐ 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.
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 Beta 1
  https://www.postgresql.org/about/news/postgresql-14-beta-1-released-2213/
☐ 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
☐ SQL Notes
☐ Slack - postgresql-jp
  https://postgresql-jp.slack.com/
## Change history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Apr 18, 2021</td>
<td>Noriyoshi Shinoda</td>
<td>Create an internal review version. Reviewers: Tomoo Takahashi Akiko Takeshima (Hewlett Packard Enterprise Japan)</td>
</tr>
<tr>
<td>1.0</td>
<td>May 21, 2021</td>
<td>Noriyoshi Shinoda</td>
<td>Modification completed according to PostgreSQL 14 Beta 1.</td>
</tr>
</tbody>
</table>