Postgres 16: What's New ?

A comprehensive overview of the Latest Features

Shruthi K C

PGConf India 2024

EDB™

Shruthi K C

Database Developer

EnterpriseDB

I have been working in Databases for past 15 years and started working on Postgres from past 3 years

About Open Source PostgreSQL

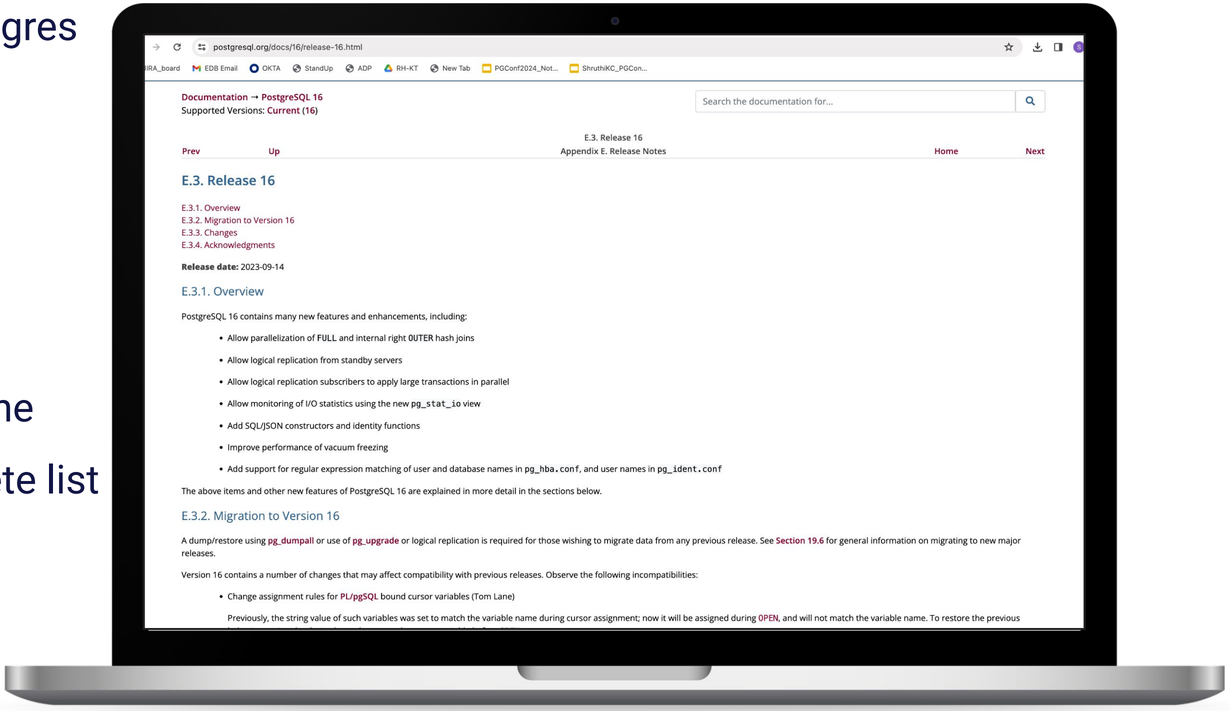
- A powerful, open source relational database system with over 35 years of active development
- The power of an Open Source is in the **community** that surrounds it
- When developers and even companies from different backgrounds, unique perspectives collaborate around making a technology, the result is a code that is richer, more secure and more innovative
- Strong reputation for reliability, feature robustness, and performance
- **Most Admired and Desired Database**
- All new features and fixes are thoroughly vetted by a community of contributors and committers
- There is a wealth of information to be found describing how to install and use PostgreSQL through the official documentation
- Postgres has many mailing lists where you can connect and participate in the community
- Connect with other PostgreSQL users through events and local user groups

PostgreSQL Releases and Lifecycle

- The PostgreSQL Global Development Group releases a new "major version" once a year
- Each major version receives bug fixes and, if need be, security fixes that are released at least once every 3 months in what we call a "minor release"
- Major versions are supported for 5 years after the initial release date
- One last minor version is released with fixes before the version goes end of life
- If you are building a new application, it is recommended that you start with the latest major version of Postgres
- This will guarantee the latest and greatest features, and a continuous flow of minor releases that fix bugs and improve the security of your database

PostgreSQL 16 Released!

- With every new release, Postgres becomes more and more compelling and adds great features that improve the experience of its users
- Refer to press release and the release notes for the complete list of new features and improvements



Documentation → PostgreSQL 16
Supported Versions: Current (16)

E.3. Release 16
Prev Up Appendix E. Release Notes Home Next

E.3. Release 16

E.3.1. Overview
E.3.2. Migration to Version 16
E.3.3. Changes
E.3.4. Acknowledgments

Release date: 2023-09-14

E.3.1. Overview

PostgreSQL 16 contains many new features and enhancements, including:

- Allow parallelization of `FULL` and internal right `OUTER` hash joins
- Allow logical replication from standby servers
- Allow logical replication subscribers to apply large transactions in parallel
- Allow monitoring of I/O statistics using the new `pg_stat_io` view
- Add SQL/JSON constructors and identity functions
- Improve performance of vacuum freezing
- Add support for regular expression matching of user and database names in `pg_hba.conf`, and user names in `pg_ident.conf`

The above items and other new features of PostgreSQL 16 are explained in more detail in the sections below.

E.3.2. Migration to Version 16

A dump/restore using pg_dumpall or use of pg_upgrade or logical replication is required for those wishing to migrate data from any previous release. See Section 19.6 for general information on migrating to new major releases.

Version 16 contains a number of changes that may affect compatibility with previous releases. Observe the following incompatibilities:

- Change assignment rules for PL/pgSQL bound cursor variables (Tom Lane)

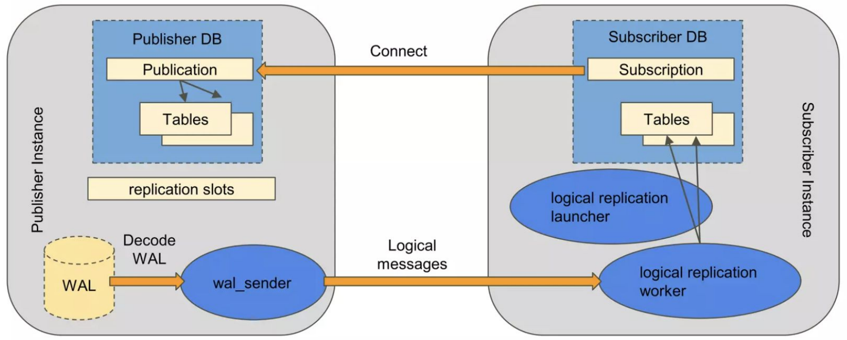
  Previously, the string value of such variables was set to match the variable name during cursor assignment; now it will be assigned during `OPEN`, and will not match the variable name. To restore the previous

Agenda

Improvements to Logical Replication

- Support Bi-directional Replication
- Allow Logical Replication from Standbys
- Apply large transactions in parallel
- Copy tables in binary format
- Flexible Index usage in the subscriber
- Control Enhancements

The architecture



Understanding the Bi-directional Logical Replication

Improvements to Logical Replication

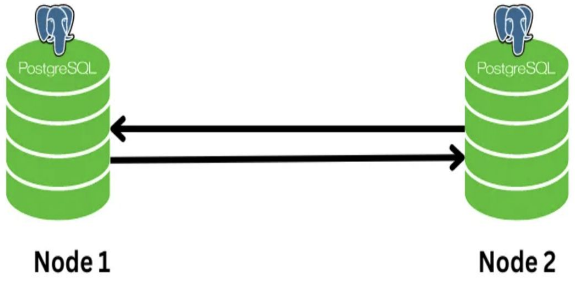
Bidirectional replication is a two-way data exchange, where Node 1 not only receives data from Node 2 but also seamlessly sends its own data back to Node 2.

What happens prior to PostgreSQL16?

- Node 1 executes SQL and replicates it to Node 2
- Node 2 receives the SQL and also executes it
- The same SQL is then sent back to Node 1, resulting in an infinite replication of the same data

Why it happens?

The Apply Worker lacks awareness of whether the data originates locally or comes from replication.



Bidirectional Replication Support in Postgres 16

Improvements to Logical Replication

CREATE SUBSCRIPTION syntax

```
CREATE SUBSCRIPTION <sub-name>
 CONNECTION 'conninfo'
 PUBLICATION <pub-list> [WITH (origin = NONE|ANY)]
```

```
ALTER SUBSCRIPTION <sub-name>
 SET (origin = NONE|ANY)
```

Example:

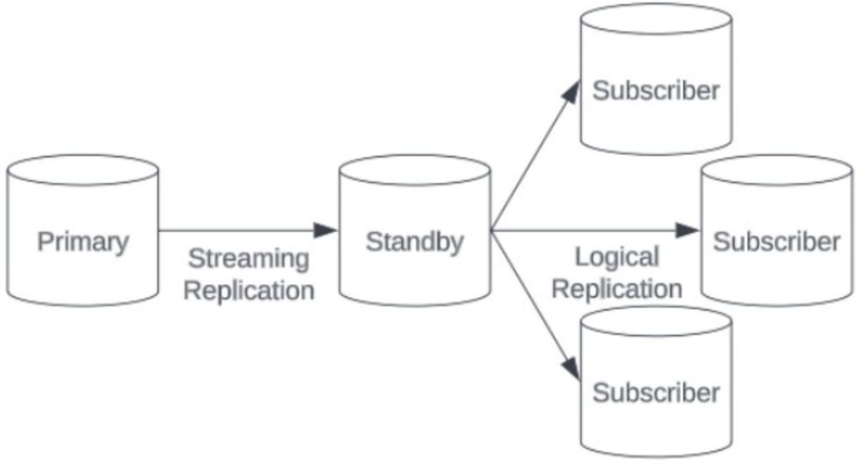
```
CREATE SUBSCRIPTION mysub
CONNECTION 'dbname=postgres port=5434'
PUBLICATION mypub WITH (origin = NONE);
```

- Locally generated data will not have a Replication Origin and the data replicated from another source will have a Replication Origin
- The origin argument accept two values :-
  - **NONE** - The subscription will request the publisher to only send changes that have no origin associated
  - **ANY** - The publisher sends changes regardless of their origin(Writes made on the server + Replicated data from other servers). The default is ANY

Logical Replication from Standbys

Improvements to Logical Replication

- This requires wal_level=logical on both primary and standby
- Allows to create a logical replication slot on a standby node
- Create a SUBSCRIPTION to a standby node
- Create logical decoding from a read-only standby
- Subscribers can continue with the connection to the standby even if the standby gets promoted to the primary
- Reduce the workload on the primary server



Parallel Application of Large Transactions

Improvements to Logical Replication

- The **streaming** option can be used to enable streaming of in-progress transactions for a given subscription. The default value is off.
- If set to parallel, incoming changes are directly applied via one of the parallel apply workers, if available. The worker remains assigned until the transaction completes.
- Performance improvement in the range of 25-40% has been observed for bulk loads with streaming = parallel
- max_parallel_apply_workers_per_subscription determines the maximum number of parallel apply workers per subscription

```
CREATE SUBSCRIPTION mysub CONNECTION 'dbname=postgres port=5434'
              PUBLICATION mypub WITH (streaming = parallel);
```

Binary Copy and Index use in Subscriber

Improvements to Logical Replication

- **Binary Copy**
  - Allow logical replication initial table synchronization to copy rows in binary format
  - Improves performance by 30-40% during SUBSCRIPTION initialization

```
CREATE SUBSCRIPTION mysub CONNECTION … PUBLICATION mypub WITH (binary = true);
```

- **Allow indexes other than PK and REPLICA IDENTITY on the subscriber**
  - Using REPLICA IDENTITY FULL on the publisher can lead to a full table scan per tuple change on the subscriber when REPLICA IDENTITY or PK index is not available
  - Subscribers can now utilize B-tree indexes instead of sequential scans to locate rows
  - This change is expected to boost performance of logical replication

EDB

Logical replication control enhancements

Improvements to Logical Replication

- **Non-superusers can create subscriptions**
  - **pg_create_subscription** - a new predefined role is added in Postgres 16
  - The non-superuser much have been granted pg_create_subscription role
  - The non-superusers are required to specify a password for authentication
  - Superusers can set password_required = false for non-superusers that own the subscription
- **Perform operations with table owner's privileges**
  - The apply process can be configured to perform operations with the table owner's privileges instead of subscription owner's privileges

```
CREATE SUBSCRIPTION mysub CONNECTION … PUBLICATION mypub WITH (run_as_owner = false);
```

Monitoring Enhancements

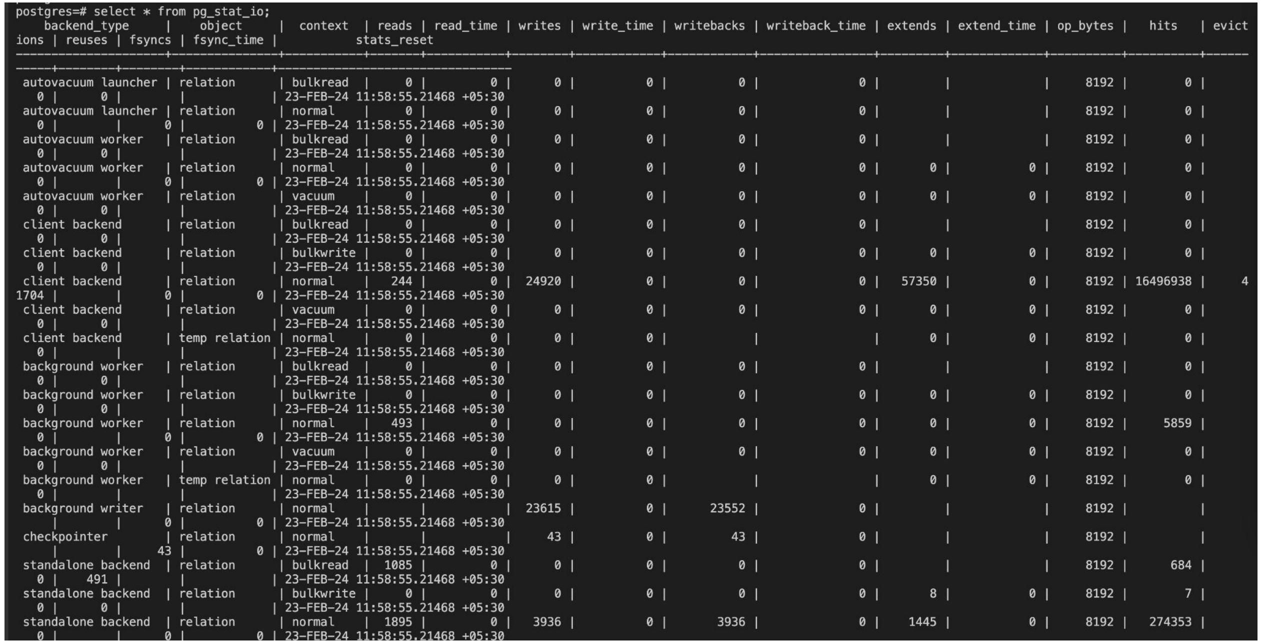
A monitoring boon – pg_stat_io

- comprehensive I/O statistics for troubleshooting performance issues and database optimization
- The view shows cluster-wide I/O statistics that has one row for each combination of backend type, target I/O object and I/O context
    - backend types: background worker, autovacuum worker, checkpointer, etc
    - target I/O objects: permanent or temporary relations
    - I/O context: normal, vacuum, bulkread and bulkwrite
- The view tracks various I/O operations like reads, writes, extends, hits, evictions, reuses and fsyncs
- A high evictions count can indicate that shared buffers should be increased
- Large numbers of fsyncs by client backends could indicate misconfiguration of shared buffers or of the checkpointer
- This release adds new fields last_seq_scan and last_idx_scan to the pg_stat_all_tables view that records a timestamp representing when a table was last scanned
- PostgreSQL 16 also improves the accuracy of the query tracking algorithm used by pg_stat_activity

Monitoring Enhancements

A monitoring boon – pg_stat_io

Here is an example of the statistics you can see in pg_stat_io:



```
postgres=# select * from pg_stat_io;
    backend_type     |    object     |  context  | reads | read_time | writes | write_time | writebacks | writeback_time | extends | extend_time | op_bytes |   hits    | evict
ions | reuses | fsyncs | fsync_time |          stats_reset
---------------------+---------------+-----------+-------+-----------+--------+------------+------------+----------------+---------+-------------+----------+-----------+------
-----+--------+--------+------------+----------------------------------
 autovacuum launcher | relation      | bulkread  |     0 |         0 |      0 |          0 |          0 |              0 |         |             |     8192 |         0 |
   0 |      0 |        |            | 23-FEB-24 11:58:55.21468 +05:30
 autovacuum launcher | relation      | normal    |     0 |         0 |      0 |          0 |          0 |              0 |         |             |     8192 |         0 |
   0 |        |      0 |          0 | 23-FEB-24 11:58:55.21468 +05:30
 autovacuum worker   | relation      | bulkread  |     0 |         0 |      0 |          0 |          0 |              0 |         |             |     8192 |         0 |
   0 |      0 |        |            | 23-FEB-24 11:58:55.21468 +05:30
 autovacuum worker   | relation      | normal    |     0 |         0 |      0 |          0 |          0 |              0 |       0 |           0 |     8192 |         0 |
   0 |        |      0 |          0 | 23-FEB-24 11:58:55.21468 +05:30
 autovacuum worker   | relation      | vacuum    |     0 |         0 |      0 |          0 |          0 |              0 |       0 |           0 |     8192 |         0 |
   0 |      0 |        |            | 23-FEB-24 11:58:55.21468 +05:30
 client backend      | relation      | bulkread  |     0 |         0 |      0 |          0 |          0 |              0 |         |             |     8192 |         0 |
   0 |      0 |        |            | 23-FEB-24 11:58:55.21468 +05:30
 client backend      | relation      | bulkwrite |     0 |         0 |      0 |          0 |          0 |              0 |       0 |           0 |     8192 |         0 |
   0 |      0 |        |            | 23-FEB-24 11:58:55.21468 +05:30
 client backend      | relation      | normal    |   244 |         0 |  24920 |          0 |          0 |              0 |   57350 |           0 |     8192 |  16496938 |     4
1704 |        |      0 |          0 | 23-FEB-24 11:58:55.21468 +05:30
 client backend      | relation      | vacuum    |     0 |         0 |      0 |          0 |          0 |              0 |       0 |           0 |     8192 |         0 |
   0 |      0 |        |            | 23-FEB-24 11:58:55.21468 +05:30
 client backend      | temp relation | normal    |     0 |         0 |      0 |          0 |            |                |       0 |           0 |     8192 |         0 |
   0 |        |        |            | 23-FEB-24 11:58:55.21468 +05:30
 background worker   | relation      | bulkread  |     0 |         0 |      0 |          0 |          0 |              0 |         |             |     8192 |         0 |
   0 |      0 |        |            | 23-FEB-24 11:58:55.21468 +05:30
 background worker   | relation      | bulkwrite |     0 |         0 |      0 |          0 |          0 |              0 |       0 |           0 |     8192 |         0 |
   0 |      0 |        |            | 23-FEB-24 11:58:55.21468 +05:30
 background worker   | relation      | normal    |   493 |         0 |      0 |          0 |          0 |              0 |       0 |           0 |     8192 |      5859 |
   0 |        |      0 |          0 | 23-FEB-24 11:58:55.21468 +05:30
 background worker   | relation      | vacuum    |     0 |         0 |      0 |          0 |          0 |              0 |       0 |           0 |     8192 |         0 |
   0 |      0 |        |            | 23-FEB-24 11:58:55.21468 +05:30
 background worker   | temp relation | normal    |     0 |         0 |      0 |          0 |            |                |       0 |           0 |     8192 |         0 |
   0 |        |        |            | 23-FEB-24 11:58:55.21468 +05:30
 background writer   | relation      | normal    |       |           |  23615 |          0 |      23552 |              0 |         |             |     8192 |           |
     |        |      0 |          0 | 23-FEB-24 11:58:55.21468 +05:30
 checkpointer        | relation      | normal    |       |           |     43 |          0 |         43 |              0 |         |             |     8192 |           |
     |        |     43 |          0 | 23-FEB-24 11:58:55.21468 +05:30
 standalone backend  | relation      | bulkread  |  1085 |         0 |      0 |          0 |          0 |              0 |         |             |     8192 |       684 |
   0 |    491 |        |            | 23-FEB-24 11:58:55.21468 +05:30
 standalone backend  | relation      | bulkwrite |     0 |         0 |      0 |          0 |          0 |              0 |       8 |           0 |     8192 |         7 |
   0 |      0 |        |            | 23-FEB-24 11:58:55.21468 +05:30
 standalone backend  | relation      | normal    |  1895 |         0 |   3936 |          0 |       3936 |              0 |    1445 |           0 |     8192 |    274353 |
   0 |        |      0 |          0 | 23-FEB-24 11:58:55.21468 +05:30
```

Localization

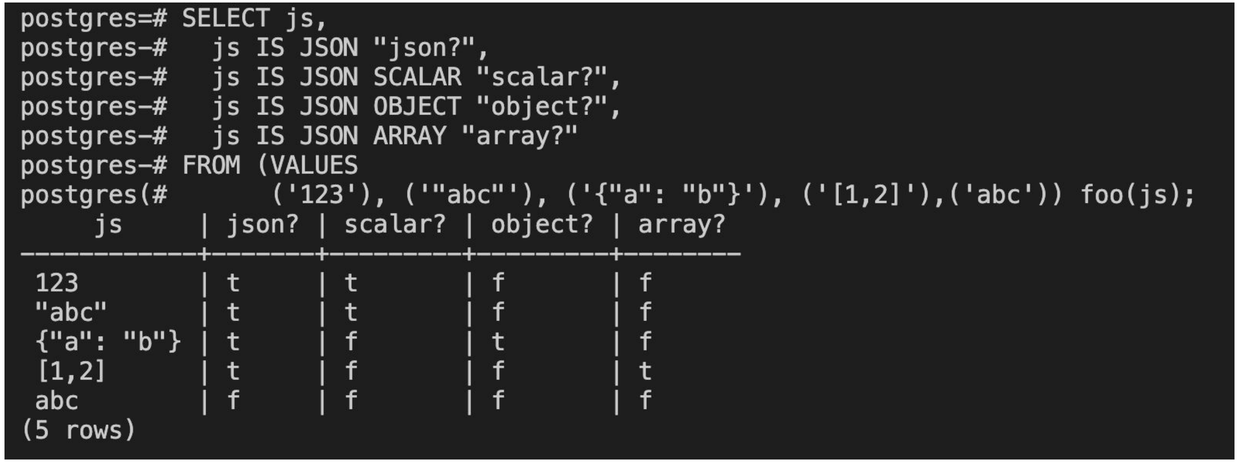
- PostgreSQL 16 improves general support for text collations, which provide rules for how text is sorted.
- PostgreSQL 16 builds with ICU support by default and can determine the default ICU locale from the environment.
- Allows users to define custom ICU collation rules.

Enhanced JSON Support

- PostgreSQL 16 adds more syntax from the SQL/JSON standard, including constructors such as
  - JSON_ARRAY()
  - JSON_ARRAYAGG()
  - JSON_OBJECT()
  - JSON_OBJECTAGG()
- Introduces the ability to use underscores for thousands separators (e.g. 5_432_000) and non-decimal integer literals, such as 0x1538, 0o12470, and 0b1010100111000.
- Introduced \bind command in psql which allows users to prepare parameterized queries and use \bind to substitute the variable inside a psql terminal

Enhanced JSON Support

- Introduce SQL standard IS JSON predicate
  - IS [NOT] JSON VALUE
  - IS [NOT] JSON SCALAR
  - IS [NOT] JSON ARRAY
  - IS [NOT] JSON OBJECT



```
postgres=# SELECT js,
postgres-#   js IS JSON "json?",
postgres-#   js IS JSON SCALAR "scalar?",
postgres-#   js IS JSON OBJECT "object?",
postgres-#   js IS JSON ARRAY "array?"
postgres-# FROM (VALUES
postgres(#       ('123'), ('"abc"'), ('{"a": "b"}'), ('[1,2]'),('abc')) foo(js);
     js     | json? | scalar? | object? | array?
------------+-------+---------+---------+--------
 123        | t     | t       | f       | f
 "abc"      | t     | t       | f       | f
 {"a": "b"} | t     | f       | t       | f
 [1,2]      | t     | f       | f       | t
 abc        | f     | f       | f       | f
(5 rows)
```

Performance Improvements

- **Bulk loading**
  - This release brings significant improvements to bulk loading using COPY command, applicable to both single and concurrent operations
  - Tests have shown remarkable performance enhancements of up to 300% in certain scenarios
- **CPU Acceleration using SIMD in x86 and ARM architectures**
  - notable performance gains when processing ASCII and JSON strings, as well as array and subtransaction searches
- **Allow VACUUM/ANALYZE to specify buffer usage limit**
  - A new option BUFFER_USAGE_LIMIT has been added, which allows user to control the size of shared buffers. Larger values can make vacuum run faster at the cost of slowing down other concurrent queries
  - Also, a new GUC named vacuum_buffer_usage_limit is added to controls how large to make the access strategy when the buffer size is not explicitly specified in VACUUM/ANALYZE command
- **Improved VACUUM**
  - During non-freeze operations, perform page freezing where appropriate. This makes full-table freeze vacuums less necessary.

Performance Improvements

- **Load balancing with multiple hosts in libpq**
  - To balance the load across multiple servers when using libpq, a new feature has been added that lets you specify a connection parameter called load_balance_hosts
  - By setting this parameter to random, libpq will randomly connect to different hosts and their associated IP addresses
  - This helps distribute the workload when there are multiple clients or frequent connection setups
  - It's recommended to configure a reasonable value for connect_timeout so that if one of the nodes is not responding, a new node will be tried

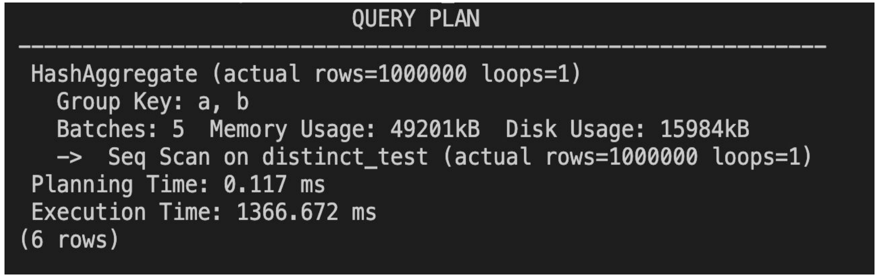
What's new in the Postgres 16 query planner / optimizer

1. Incremental sorts for DISTINCT queries

- Prior to PostgreSQL 16, when the sorting method was chosen for SELECT DISTINCT queries, the planner only considered performing a full sort which was more expensive
- The PostgreSQL 16 query planner now considers performing incremental sorts for SELECT DISTINCT queries

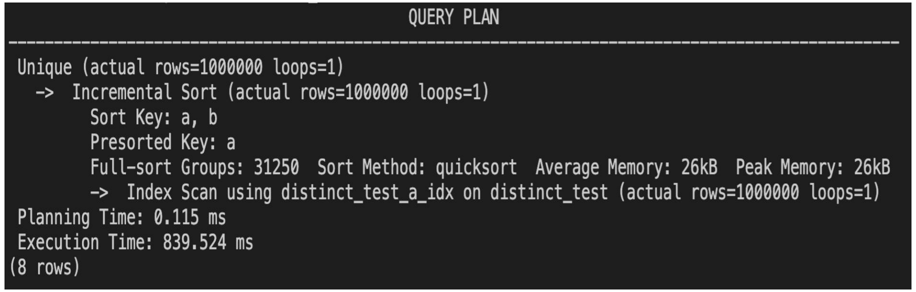
```
-- Test Setup
CREATE TABLE distinct_test (a INT, b INT);
INSERT INTO distinct_test SELECT x,1 FROM
generate_series(1,1000000)x;
CREATE INDEX on distinct_test(a);
VACUUM ANALYZE distinct_test;

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT DISTINCT a, b FROM distinct_test;
```

PG15 EXPLAIN output



```
                             QUERY PLAN
----------------------------------------------------------------
 HashAggregate (actual rows=1000000 loops=1)
   Group Key: a, b
   Batches: 5  Memory Usage: 49201kB  Disk Usage: 15984kB
   ->  Seq Scan on distinct_test (actual rows=1000000 loops=1)
 Planning Time: 0.117 ms
 Execution Time: 1366.672 ms
(6 rows)
```

PG16 EXPLAIN output



```
                                                QUERY PLAN
------------------------------------------------------------------------------------------------------
 Unique (actual rows=1000000 loops=1)
   ->  Incremental Sort (actual rows=1000000 loops=1)
         Sort Key: a, b
         Presorted Key: a
         Full-sort Groups: 31250  Sort Method: quicksort  Average Memory: 26kB  Peak Memory: 26kB
         ->  Index Scan using distinct_test_a_idx on distinct_test (actual rows=1000000 loops=1)
 Planning Time: 0.115 ms
 Execution Time: 839.524 ms
(8 rows)
```

EDB

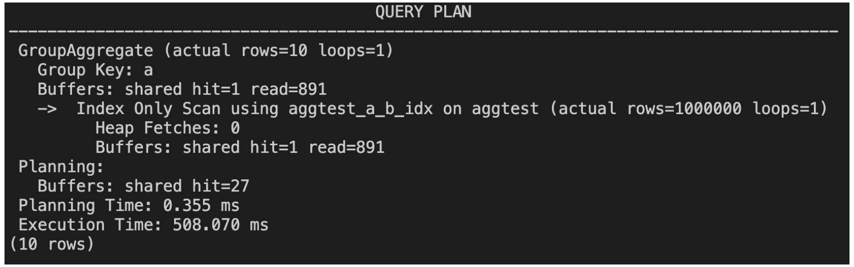
What’s new in the Postgres 16 query planner / optimizer

2. Faster ORDER BY / DISTINCT aggregates

- In PostgreSQL 15 and earlier, aggregate functions containing an ORDER BY or DISTINCT clause would result in the executor always performing a sort inside the Aggregate node of the plan
- The PostgreSQL 16 query planner now tries to form a plan which feeds the rows to the plan’s Aggregate node in the correct order. And the executor is now smart enough to recognize this and forego performing the sort itself when the rows are pre-sorted

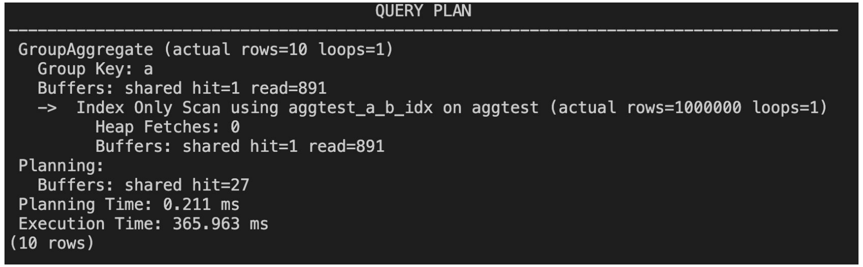
```
-- Test Setup
CREATE TABLE aggtest (a INT, b text);
INSERT INTO aggtest SELECT a,md5((b%100)::text) FROM
generate_series(1,10) a, generate_series(1,100000)b;
CREATE INDEX ON aggtest(a,b);
VACUUM FREEZE ANALYZE aggtest;

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF, BUFFERS)
SELECT a, COUNT(DISTINCT b) FROM aggtest GROUP BY a;
```

PG15 EXPLAIN output



```
                                          QUERY PLAN
-------------------------------------------------------------------------------------------------
 GroupAggregate (actual rows=10 loops=1)
   Group Key: a
   Buffers: shared hit=1 read=891
   ->  Index Only Scan using aggtest_a_b_idx on aggtest (actual rows=1000000 loops=1)
         Heap Fetches: 0
         Buffers: shared hit=1 read=891
 Planning:
   Buffers: shared hit=27
 Planning Time: 0.355 ms
 Execution Time: 508.070 ms
(10 rows)
```

PG16 EXPLAIN output



```
                                          QUERY PLAN
-------------------------------------------------------------------------------------------------
 GroupAggregate (actual rows=10 loops=1)
   Group Key: a
   Buffers: shared hit=1 read=891
   ->  Index Only Scan using aggtest_a_b_idx on aggtest (actual rows=1000000 loops=1)
         Heap Fetches: 0
         Buffers: shared hit=1 read=891
 Planning:
   Buffers: shared hit=27
 Planning Time: 0.211 ms
 Execution Time: 365.963 ms
(10 rows)
```

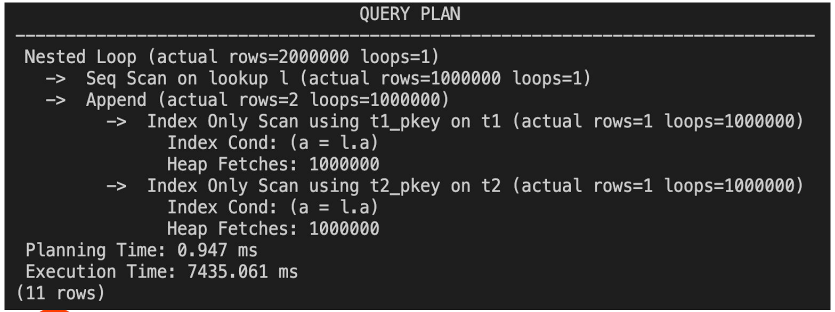
What’s new in the Postgres 16 query planner / optimizer

3. Allow memoize atop a UNION ALL

- When the same value needs to be looked up several times, Memoize can give a nice performance boost as it can skip executing its subnode when the required rows have been queried already and are cached
- The PostgreSQL 16 query planner will now consider using Memoize for UNION ALL query

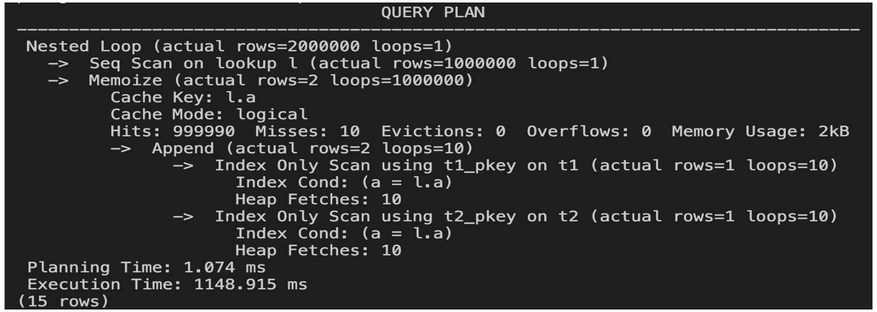
```
-- Test Setup
CREATE TABLE t1 (a INT PRIMARY KEY);
CREATE TABLE t2 (a INT PRIMARY KEY);
CREATE TABLE lookup (a INT);
INSERT INTO t1 SELECT x FROM generate_Series(1,10000) x;
INSERT INTO t2 SELECT x FROM generate_Series(1,10000) x;
INSERT INTO lookup SELECT x%10+1 FROM
generate_Series(1,1000000)x;

ANALYZE t1, t2, lookup;

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT * FROM (SELECT * FROM t1 UNION ALL SELECT * FROM t2) t
INNER JOIN lookup l ON l.a = t.a;
```

PG15 EXPLAIN output



```
                                   QUERY PLAN
-------------------------------------------------------------------------------
 Nested Loop (actual rows=2000000 loops=1)
   ->  Seq Scan on lookup l (actual rows=1000000 loops=1)
   ->  Append (actual rows=2 loops=1000000)
         ->  Index Only Scan using t1_pkey on t1 (actual rows=1 loops=1000000)
               Index Cond: (a = l.a)
               Heap Fetches: 1000000
         ->  Index Only Scan using t2_pkey on t2 (actual rows=1 loops=1000000)
               Index Cond: (a = l.a)
               Heap Fetches: 1000000
 Planning Time: 0.947 ms
 Execution Time: 7435.061 ms
(11 rows)
```

PG16 EXPLAIN output



```
                                   QUERY PLAN
-----------------------------------------------------------------------------------------------
 Nested Loop (actual rows=2000000 loops=1)
   ->  Seq Scan on lookup l (actual rows=1000000 loops=1)
   ->  Memoize (actual rows=2 loops=1000000)
         Cache Key: l.a
         Cache Mode: logical
         Hits: 999990  Misses: 10  Evictions: 0  Overflows: 0  Memory Usage: 2kB
         ->  Append (actual rows=2 loops=10)
               ->  Index Only Scan using t1_pkey on t1 (actual rows=1 loops=10)
                     Index Cond: (a = l.a)
                     Heap Fetches: 10
               ->  Index Only Scan using t2_pkey on t2 (actual rows=1 loops=10)
                     Index Cond: (a = l.a)
                     Heap Fetches: 10
 Planning Time: 1.074 ms
 Execution Time: 1148.915 ms
(15 rows)
```

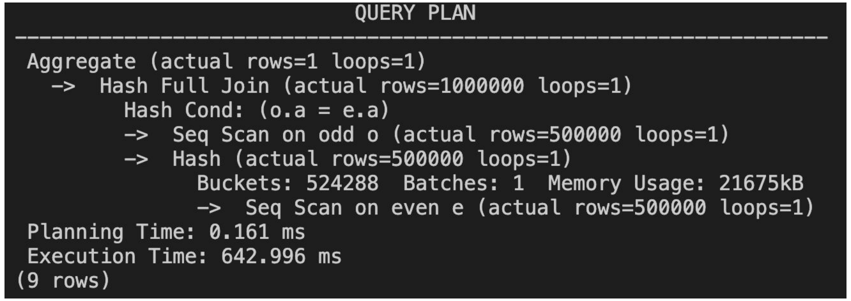
What’s new in the Postgres 16 query planner / optimizer

4. Parallel Hash Full and Right Joins

- In PostgreSQL 16, Parallel Hash Join has been improved and now supports FULL and RIGHT join types
- This allows queries that have a FULL OUTER JOIN to be executed in parallel and also allows Right Joins plans to execute in parallel

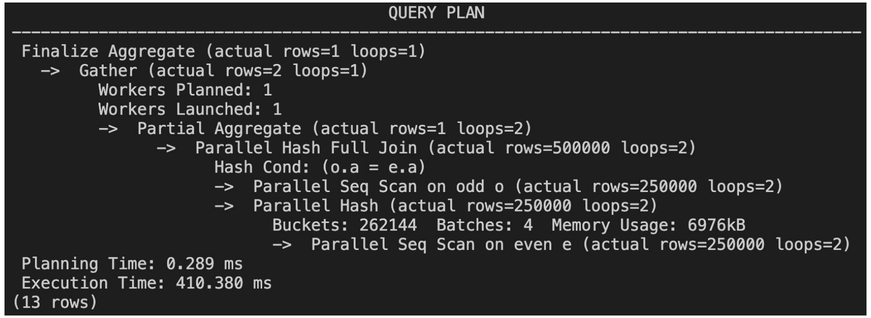
```
-- Setup
CREATE TABLE odd (a INT);
CREATE TABLE even (a INT);
INSERT INTO odd
SELECT a FROM generate_series(1,1000000,2) a;
INSERT INTO even
SELECT a FROM generate_series(2,1000000,2) a;
VACUUM ANALYZE odd, even;

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT COUNT(o.a), COUNT(e.a) FROM odd o FULL JOIN even e ON o.a =
e.a;
```

PG15 EXPLAIN output



```
                            QUERY PLAN
---------------------------------------------------------------------
 Aggregate (actual rows=1 loops=1)
   ->  Hash Full Join (actual rows=1000000 loops=1)
         Hash Cond: (o.a = e.a)
         ->  Seq Scan on odd o (actual rows=500000 loops=1)
         ->  Hash (actual rows=500000 loops=1)
               Buckets: 524288  Batches: 1  Memory Usage: 21675kB
               ->  Seq Scan on even e (actual rows=500000 loops=1)
 Planning Time: 0.161 ms
 Execution Time: 642.996 ms
(9 rows)
```

PG16 EXPLAIN output



```
                                    QUERY PLAN
------------------------------------------------------------------------------------------
 Finalize Aggregate (actual rows=1 loops=1)
   ->  Gather (actual rows=2 loops=1)
         Workers Planned: 1
         Workers Launched: 1
         ->  Partial Aggregate (actual rows=1 loops=2)
               ->  Parallel Hash Full Join (actual rows=500000 loops=2)
                     Hash Cond: (o.a = e.a)
                     ->  Parallel Seq Scan on odd o (actual rows=250000 loops=2)
                     ->  Parallel Hash (actual rows=250000 loops=2)
                           Buckets: 262144  Batches: 4  Memory Usage: 6976kB
                           ->  Parallel Seq Scan on even e (actual rows=250000 loops=2)
 Planning Time: 0.289 ms
 Execution Time: 410.380 ms
(13 rows)
```

EDB

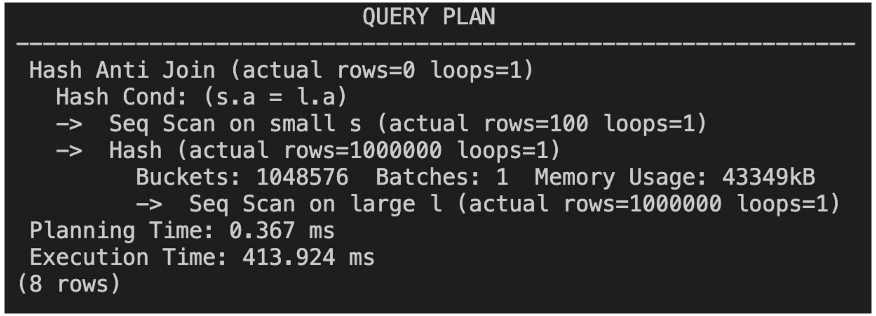
What’s new in the Postgres 16 query planner / optimizer

5. Support Right Anti Join

- In PostgreSQL versions before 16, an Anti Join—as you might see if you use NOT EXISTS in your queries—would always put the table mentioned in the NOT EXISTS part on the inner side of the join. This meant there was no flexibility to hash the smaller of the two tables, resulting in possibly having to build a hash table on the larger table
- The PostgreSQL 16 query planner can now choose to hash the smaller of the two tables. This can now be done because PostgreSQL 16 supports Right Anti Joins

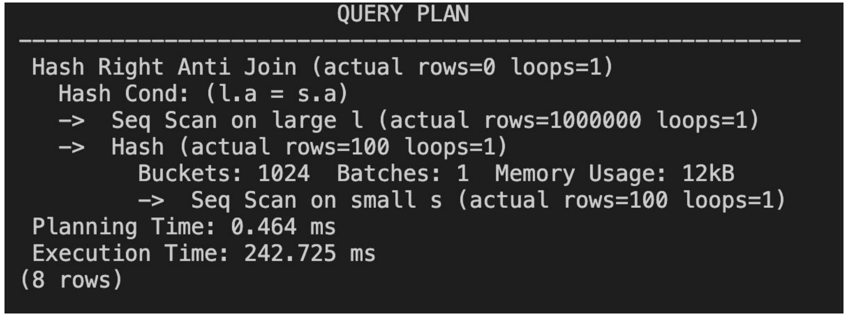
```
-- Setup
CREATE TABLE small(a int);
CREATE TABLE large(a int);
INSERT INTO small
SELECT a FROM generate_series(1,100) a;
INSERT INTO large
SELECT a FROM generate_series(1,1000000) a;
VACUUM ANALYZE small,large;

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT * FROM small s
WHERE NOT EXISTS(SELECT 1 FROM large l WHERE s.a = l.a);
```

PG15 EXPLAIN output



```
                          QUERY PLAN
---------------------------------------------------------------
 Hash Anti Join (actual rows=0 loops=1)
   Hash Cond: (s.a = l.a)
   ->  Seq Scan on small s (actual rows=100 loops=1)
   ->  Hash (actual rows=1000000 loops=1)
         Buckets: 1048576  Batches: 1  Memory Usage: 43349kB
         ->  Seq Scan on large l (actual rows=1000000 loops=1)
 Planning Time: 0.367 ms
 Execution Time: 413.924 ms
(8 rows)
```

PG16 EXPLAIN output



```
                          QUERY PLAN
-----------------------------------------------------------
 Hash Right Anti Join (actual rows=0 loops=1)
   Hash Cond: (l.a = s.a)
   ->  Seq Scan on large l (actual rows=1000000 loops=1)
   ->  Hash (actual rows=100 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 12kB
         ->  Seq Scan on small s (actual rows=100 loops=1)
 Planning Time: 0.464 ms
 Execution Time: 242.725 ms
(8 rows)
```

EDB

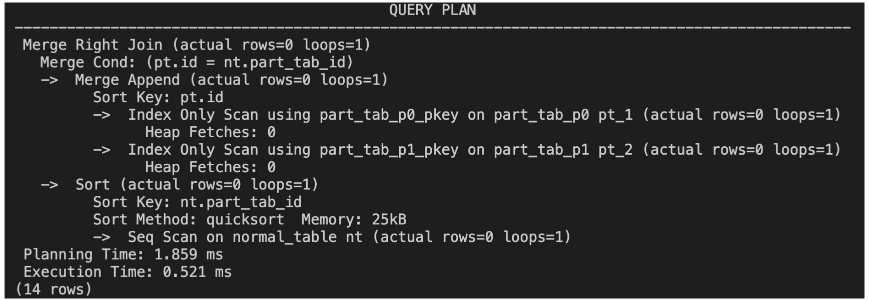
What’s new in the Postgres 16 query planner / optimizer

6. Allow left join removals on partitioned tables

- In versions prior to PostgreSQL 16, there was no support for left join removals on partitioned tables
- The PostgreSQL 16 query planner now allows the LEFT JOIN removal optimization with partitioned tables

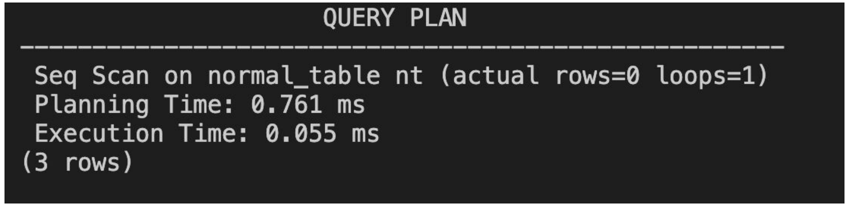
```
-- Setup
CREATE TABLE part_tab (id BIGINT PRIMARY KEY, payload TEXT) PARTITION BY
HASH(id);
CREATE TABLE part_tab_p0 PARTITION OF part_tab FOR VALUES WITH (modulus 2,
remainder 0);
CREATE TABLE part_tab_p1 PARTITION OF part_tab FOR VALUES WITH (modulus 2,
remainder 1);
CREATE TABLE normal_table (id INT, part_tab_id BIGINT);

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT nt.* FROM normal_table nt LEFT JOIN part_tab pt ON nt.part_tab_id = pt.id;
```

PG15 EXPLAIN output



```
                                           QUERY PLAN
------------------------------------------------------------------------------------------------
 Merge Right Join (actual rows=0 loops=1)
   Merge Cond: (pt.id = nt.part_tab_id)
   ->  Merge Append (actual rows=0 loops=1)
         Sort Key: pt.id
         ->  Index Only Scan using part_tab_p0_pkey on part_tab_p0 pt_1 (actual rows=0 loops=1)
               Heap Fetches: 0
         ->  Index Only Scan using part_tab_p1_pkey on part_tab_p1 pt_2 (actual rows=0 loops=1)
               Heap Fetches: 0
   ->  Sort (actual rows=0 loops=1)
         Sort Key: nt.part_tab_id
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on normal_table nt (actual rows=0 loops=1)
 Planning Time: 1.859 ms
 Execution Time: 0.521 ms
(14 rows)
```

PG16 EXPLAIN output



```
                    QUERY PLAN
-----------------------------------------------------
 Seq Scan on normal_table nt (actual rows=0 loops=1)
 Planning Time: 0.761 ms
 Execution Time: 0.055 ms
(3 rows)
```

Access Control & Security

- Add libpq connection option require_auth to specify a list of acceptable authentication methods
- Add libpq option sslcertmode to control transmission of the client certificate. The option values are disable, allow, and require.
- Support for regular expression matching on database and role entries in pg_hba.conf
- Support for Kerberos credential delegation, allowing extensions such as postgres_fdw and dblink to use authenticated credentials to connect to trusted services

Privilege Administration Overhaul

Administering PG Without Having to Be a Superuser

- Allow ALTER GROUP group_name ADD USER user_name to be performed with ADMIN OPTION. Previously CREATEROLE permission was required.
- Allow roles that create other roles to automatically inherit the new role's rights or the ability to SET ROLE to the new role. This is controlled by server variable createrole_self_grant.
- Allow GRANT to use WITH ADMIN TRUE/FALSE syntax. Previously only the WITH ADMIN OPTION syntax was supported.
- Allow users to change the default privileges of only inherited roles.
- Add GRANT to control permission to use SET ROLE. This is controlled by a new GRANT ... SET option.

Postgres 17: What to expect ?

Laying Groundwork for an exciting Postgres 17

- Incremental backups
- Logical replication improvements (DDL Replication, Replication of sequences, Reuse of tablesync workers and so on)
- SQL/JSON improvements to make it more standard-compliant
- Improvements in vacuum technology
- Improvements in partitioning technology
- Improve statistics/monitoring
- SLRU optimizations
- Improve locking for better scalability
- Enhance Table AM APIs
- TOAST improvements

Thank You!

PostgreSQL 16: What’s New ?
A comprehensive overview of the Latest Features

Shruthi K C

EDB™