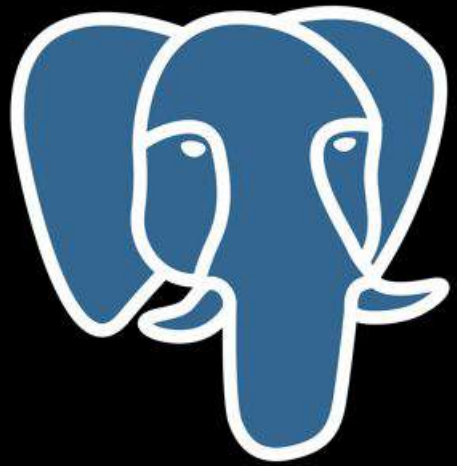
What's New in PostgreSQL 18



Vinoth Kanna RS

Founding Partner, Mydbops LLP

Mydbops MyWebinar 47

Mydbops
Scaling Databases

About Me

Vinoth Kanna R S

- Making Database Management Simple at Scale
- Leading remote database team at Mydbops
- I enjoy working with various database technologies and scaling backend systems.
- Tech Speaker on Open Source Events
- Databases for living (13 years)

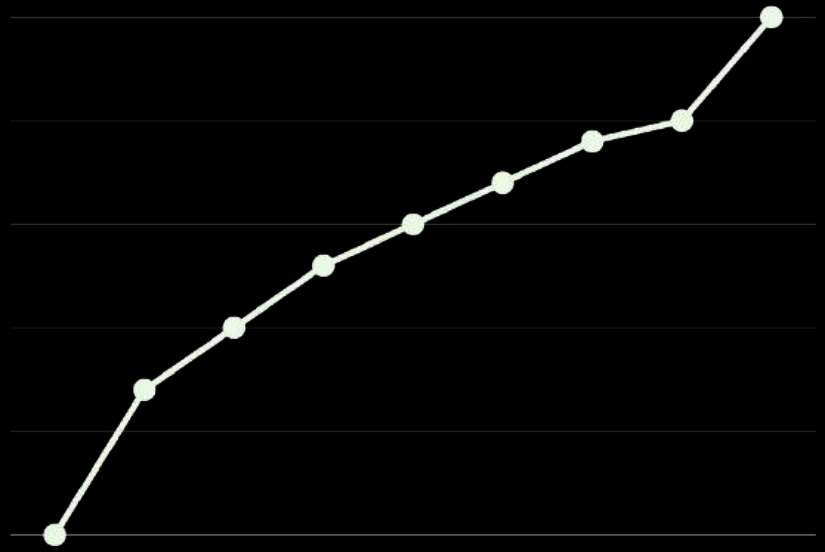
Mydbops
Scaling Databases

Mydbops

Scaling Databases

Your Trusted Open Source Database Management Partner

With 9+ Years of Expertise



Mydbops by the Numbers

9+ years
Of Expertise

800 +
Happy Clients

10 B +
DB Transactions Handled per Day

6000 +
Servers Monitored

3000 +
Tickets Handled per Day

Database Technologies

MySQL · MongoDB · PostgreSQL · MariaDB · TiDB · cassandra

Mydbops
Scaling Databases

Agenda

- Asynchronous IO
- Vacuuming
- Persistent statistics on Upgrade
- UUIDV7
- Virtual Columns
- Index Skip Scans
- Non Overlapping range Constraints
- Other Features
- Cloud Readiness
- Q&A

Mydbops
Scaling Databases

Asynchronous IO

Asynchronous IO: The Foundations (PgSQL 17)

- Streaming and Vectored I/O
- 1 preadv() (128kB) call can replace 16 pread() (8kB) calls
- Uses posix_fadvise() to prefetch to OS page cache
- ReadBuffer is replaced by Streaming API (ReadStream)
  - pg_prewarm - loads a full relation (table, index) into shared buffers
  - Faster ANALYZE
  - Faster sequential scans, OLAP workloads

Asynchronous IO: The Present (PgSQL 18)

- Introduced variable “io_method” to control, configure and optimise
- Extended ReadSteam usage for
  - Bitmap heap scans
  - Vacuum
- “Effective_io_concurrency” directly controls the prefetch
- Stats are available via table **pg_aio** to monitor the performance

Asynchronous IO: io_method

Method	How It Works	Best For
sync	Disables AIO.	Baseline testing and troubleshooting.
worker (Default)	Uses a pool of background workers to handle I/O. Portable and robust.	The recommended setting for most environments.
io_uring	A high-performance, Linux-only method using a shared buffer with the kernel for minimal overhead.	Modern Linux systems where Postgres is compiled with liburing support for maximum speed.

Asynchronous IO: io_method

```
vinoth=# select version();

 PostgreSQL 18.0 (Homebrew), compiled by Apple clang version 17.0.0 (clang-1700.0.13.3), 64-bit
(1 row)

vinoth=# show io_method;
 io_method
-----------
 worker
(1 row)

vinoth=# show io_workers;
 io_workers
------------
 3
(1 row)
```

Mydbops
Scaling Databases

Asynchronous IO: io_method

```
vinoth=# select datid, datname, pid, usename, wait_event, state, left(query,10), backend_type from
pg_stat_activity;
 datid | datname |  pid  | usename |     wait_event      | state  |    left    |         backend_type
-------+---------+-------+---------+---------------------+--------+------------+-----------------------------
----
 16393 | vinoth  | 92569 | vinoth  |                     | active | select dat | client backend
       |         | 92204 |         | AutovacuumMain      |        |            | autovacuum launcher
       |         | 92190 |         | IoWorkerMain        |        |            | io worker
       |         | 92194 |         | IoWorkerMain        |        |            | io worker
       |         | 92196 |         | IoWorkerMain        |        |            | io worker
       |         | 92200 |         | CheckpointerMain    |        |            | checkpointer
       |         | 92201 |         | BgwriterMain        |        |            | background writer
       |         | 92203 |         | WalWriterMain       |        |            | walwriter
(8 rows)
```

Mydbops
Scaling Databases

Asynchronous IO: The Future (PgSQL 19+)

- AIO for writes (Checkpointers, BG writers, Backend Writes Etc)
- AIO integration with Network IO
- Expose other OS methods like IOCP for windows, Just like IO_URING For Linux
- POSIX aio_read/aio_write for solaris

FAQ ?

- Will AIO in PgSQL 18 make the writes faster ?
  - No, Current AIO implementation aimed to speed up the reads, Implementation of AIO for writes is in progress, Will be shipped in the later releases.
- What value i can set for io_method to get best performance ?
  - OS other than linux, only supports worker model, Tune the threads according to server capacity, CPU. Watch on memory as you increase worker threads.
  - In case of Linux, IO_Uring would be recommended, Benchmark claims it’s suited for Random IO, Watch on Connections and Open files limits.

Vacuuming

Vacuum Flow

01 Scanning Heap

02 Vacuuming Indexes

03 Vacuuming Heap

04 Cleaning Up Indexes

05 Truncating Heap

06 Update Statistics

Mydbops
Scaling Databases

Vacuum: (PgSQL 18)

- **Uses read_steams** in all lazy vacuum phases (1, 2 and 3)
- Supported in B-Tree, GIST, SPGIST indexes
- New columns total vacuum, auto vacuum, analyze, auto analyze added on pg_stat_*tables

Vacuum: (PgSQL 19+)

- **Parallel vacuum in Phase 1, 3**
- Decoupling vacuum for table and index
- Running specific vacuum phases independently

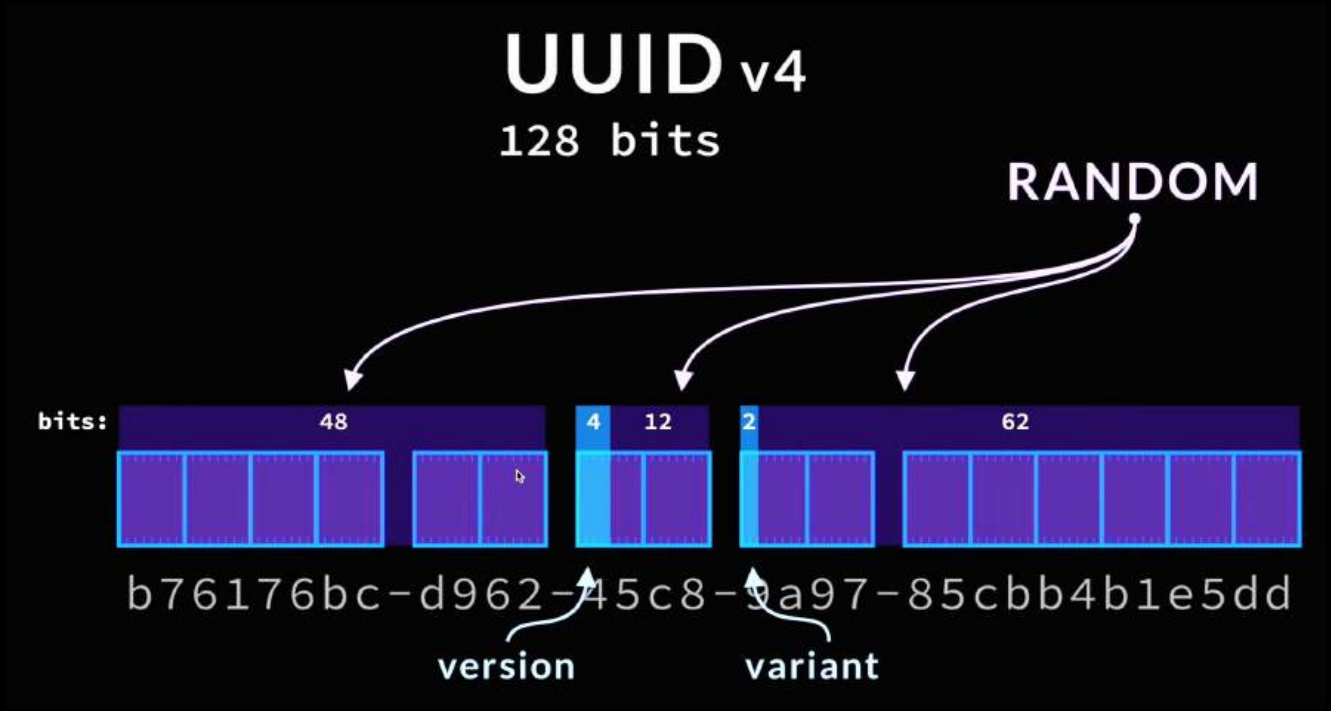
Persistent statistics on Upgrade

Persistent statistics on Upgrade

- On execution of pg_upgrade, statistics will be automatically copied
  - We have tested this during the upgrade between PostgreSQL 17 to PostgreSQL 18
- We will be able to dump the statistics manually from old version to restore in the new version
  - You should be using the pg_dump and pg_restore client tools that comes with PgSQL 18
  - pg_dump --statistics-only -Fc <DB> > stats.dump
  - pg_restore --statistics-only -d <DB> stats.dump
- Execute vacuumdb --missing-stats-only
  - compute only missing optimizer statistics

UUIDV7

UUIDV4 (PgSQL 17 and Older)



Mydbops
Scaling Databases

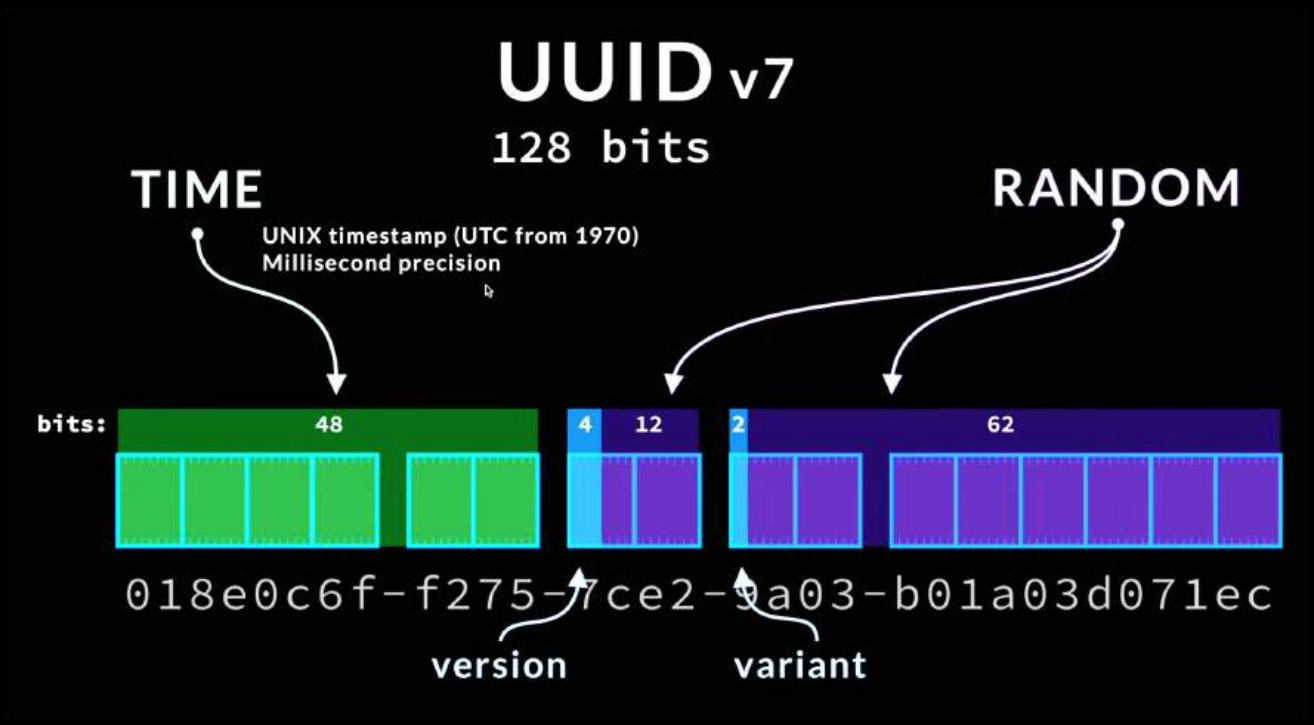
UUIDV4 (PgSQL 17 and Older)

- Issues when using random generated UUIDV4
  - Not possible to sort, Usually need other timestamp columns
  - Diverse index locality due to random ordering
  - Table bloating / Fragmentation due to unsorted nature of the primary key
- Blog reference on bloating between random UUID4 and sequentially generated UUID's
  - https://www.enterprisedb.com/blog/sequential-uuid-generators

UUIDV4 (PgSQL 17 and Older)

- How sequential UUID was generated on the older versions ?
  - Extensions
    - https://github.com/dverite/postgres-uuidv7-sql/blob/main/README.md
  - Custom Function
    - https://instagram-engineering.com/sharding-ids-at-instagram-1cf5a71e5a5c
  - Application logic

UUIDV7 (PgSQL 18+)



Mydbops
Scaling Databases

UUIDV7 (PgSQL 18+)

```
vinoth=# SELECT uuidv7();
               uuidv7
--------------------------------------
 019a38ce-4774-7b52-a4d3-1155e5d7e058
(1 row)

vinoth=# select uuid_extract_timestamp(uuidv7());
    uuid_extract_timestamp
-------------------------------
 2025-10-31 11:17:24.123+05:30
(1 row)

vinoth=# select now();
              now
---------------------------------
 2025-10-31 11:17:31.807555+05:30
(1 row)
```

Mydbops
Scaling Databases

UUIDV7 (PgSQL 18+)

```
vinoth=# CREATE TABLE test (
    id uuid DEFAULT uuidv7() PRIMARY KEY,
    name text
);
INSERT INTO test (name) VALUES ('foo');
INSERT INTO test (id, name) VALUES (uuidv7(INTERVAL '-1 day'), 'old3');
INSERT INTO test (id, name) VALUES (uuidv7(INTERVAL '-7 day'), 'old2');
INSERT INTO test (id, name) VALUES (uuidv7(INTERVAL '-24 day'), 'old1');

vinoth=# select uuid_extract_timestamp(id), id, name from test;
    uuid_extract_timestamp     |                  id                  | name
-------------------------------+--------------------------------------+------
 2025-10-31 11:17:57.729+05:30 | 019a38ce-f1a1-7deb-8617-2e17342e38b5 | bar
 2025-10-30 11:17:57.73+05:30  | 019a33a8-95a2-7089-a576-5a9d5d00a6d7 | old3
 2025-10-24 11:17:57.73+05:30  | 019a14c2-6da2-72c7-9eeb-dc05308ebb91 | old2
 2025-10-07 11:17:57.73+05:30  | 0199bd36-51a2-7473-ae01-3dc84c5b0fb0 | old1
```

UUIDV7 (PgSQL 18+)

```
vinoth=# explain select uuid_extract_timestamp(id), id, name from test where uuid_extract_timestamp(id) <
'2025-10-29';
 Seq Scan on test  (cost=0.00..26.94 rows=357 width=56)
   Filter: (uuid_extract_timestamp(id) < '2025-10-29 00:00:00+05:30'::timestamp with time zone)
(2 rows)

vinoth=# create index idx_uuid_ts on test (uuid_extract_timestamp(id));
CREATE INDEX

vinoth=# explain select uuid_extract_timestamp(id), id, name from test where uuid_extract_timestamp(id) <
'2025-10-29';
 Seq Scan on test  (cost=0.00..1.08 rows=2 width=56)
   Filter: (uuid_extract_timestamp(id) < '2025-10-29 00:00:00+05:30'::timestamp with time zone)
(2 rows)
```

Mydbops
Scaling Databases

Index Skip Scans

Index Skip Scans

- Making use of a non-leading column of an index
- Help to reduce the number of indexes
- Skip scan works by generating a dynamic equality constraint internally, that matches every possible value in an index column
- Eg:

  Index: (is_deleted, status)

  Query: select * from table where status = ‘new’;

Virtual Generated Columns

Virtual Generated Columns

- PostgreSQL 18 makes VIRTUAL the new default for generated columns.
  - STORED (The old way): Computed on write, takes up disk space. Adding one required a full table rewrite and a disruptive lock.
  - VIRTUAL (The new default): Computed on read, takes up zero storage space.
- Adding a virtual column is now an instantaneous, metadata-only operation. No more waiting hours for ALTER TABLE on a huge table.
- Generated virtual column values are calculated on the fly during the execution of the query

Virtual Generated Columns

```
vinoth=# CREATE TABLE users (
    first_name TEXT,
    last_name TEXT,
    full_name TEXT GENERATED ALWAYS AS (first_name || ' ' || last_name) VIRTUAL
);
CREATE TABLE

vinoth=# insert into users values ('vinoth','mydbops');
INSERT 0 1

vinoth=# select * from users;
 first_name | last_name |   full_name
------------+-----------+----------------
 vinoth     | mydbops   | vinoth mydbops
(1 row)
```

Mydbops
Scaling Databases

Extension of RETURNING SQL Clause

Mydbops
Scaling Databases

Extension of RETURNING SQL Clause

- OLD and NEW support for RETURNING clauses in INSERT, UPDATE, DELETE, and MERGE commands.
  - Easier to implement the auditing functionality without needing for triggers

```
UPDATE products SET price = price * 1.10
    WHERE price <= 99.99
    RETURNING name, old.price AS old_price, new.price AS new_price,
              new.price - old.price AS price_change;

DELETE FROM products
  WHERE obsoletion_date = 'today'
  RETURNING *;
```

Data Checksums

Data Checksum

- Data checksum are on by default in PgSQL 18
  - If you’re upgrading from instance with checksum turned off, Ensure to initialize DB on PgSQL 18 by disabling checksum.
  - initdb --no-data-checksums </datadir>
- Best practise is to have the checksum turned on, Logical restore and replication can be used to upgrade with checksum turned on

Other Features

- Add vacuumdb option --missing-stats-only to compute only missing optimizer statistics
  - To be run post upgrade
- Pg_overexplain
  - Understand the internal heuristics behind index, optimisations selection
- Oauth support

Public Cloud Support

Mydbops
Scaling Databases

Public Cloud Support

- AWS
  - Preview, Available only in Ohio Region
  - No option in parameter group to tune io_method
- GCP
  - Available
  - io_uring method not available for io_method, Only sync, worker is available
- Azure
  - Preview, Available only in the East Asia region
  - io_uring method not available for io_method, Only sync, worker is available

Key Takeaways

Key Takeaways

- Linux VM based instances will be best to test all the features
- Validate compatibility for upgrade with pg_upgrade --check
- If you want to explore all the new features, Linux based VM would be the best option
- When tuning for AIO, Benchmark with real time data size, queries, server configuration
- Benchmarks from the synthetic benchmarks from sysbench, pgbench should be only taken for reference

Connect with us !

Reach us at: **info@mydbops.com**

Mydbops
Scaling Databases

Thank You

Mydbops
Scaling Databases