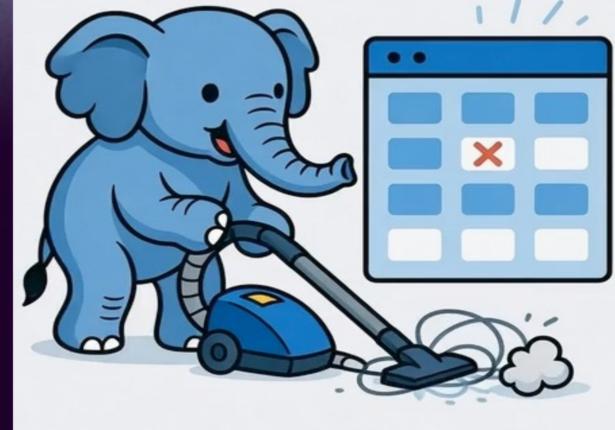
PGCONF SAN JOSE 2026

Autovacuum Blocked, Backlogged, or Slow?!

Understanding Why It Can't Keep Up



Baji Shaik
Sr DB Engineer
Amazon RDS

Mohamed Ali
Sr DB Engineer
Amazon RDS

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Who am I?

- Sr DB Engineer - RDS/Aurora PostgreSQL
- Over a dozen years of experience with PostgreSQL
- Played different roles – DBA, Consultant, Migration Expert, DB Engineer
- Authored multiple books and published several articles on PostgreSQL

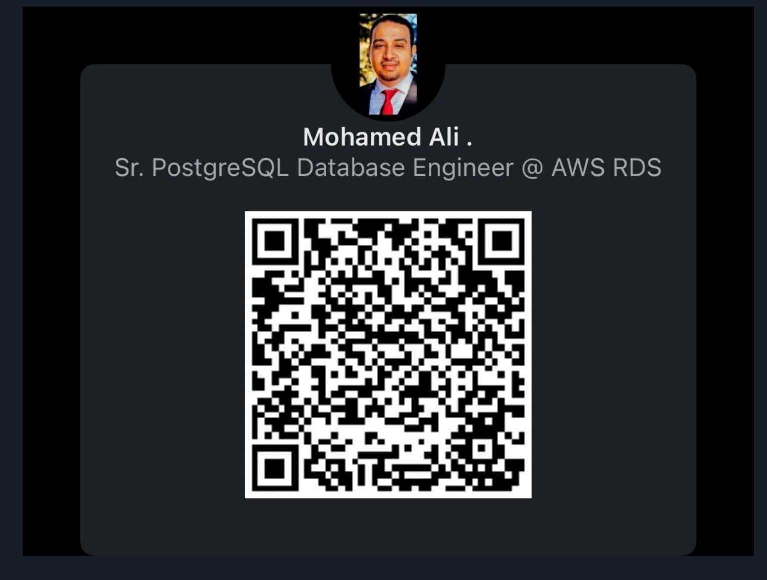
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Who am I?

Senior Database Engineer at Amazon Web Services (AWS), specializing in PostgreSQL and Oracle database technologies. With over 15 years of experience in managing and scaling databases and more than 9 years At AWS RDS, passionate about scalable database systems, performance optimization, and open-source innovation within the PostgreSQL ecosystem

Contributed to many database engineering projects such as:

- The design and integration of Babelfish with PostgreSQL
- The design and development of Aurora PostgreSQL Serverless v2
- Performance Insights and Database Insights
- The design and development of the Aurora PostgreSQL Limitless Database



www.linkedin.com/in/mohamed-ali-97561425

I am the author of two open-source tools pg_collector (https://github.com/awslabs/pg-collector)

and

PG Counter Metrics (PGCM) (https://github.com/awslabs/pg-counter-metrics)

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

How Autovacuum Works

Autovacuum Triggers — When & Why

The Three Critical Problems

Detection & Monitoring

Key Takeaways & Q&A

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Autovacuum: What It Does & What Breaks

Remove dead tuples
Free space for reuse

Update planner stats
Keep plans optimal

Freeze transaction IDs
Prevent wraparound

⚠️ Without autovacuum → bloat, bad plans, and eventually wraparound happens

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

How Autovacuum Works

🚀 Launcher Wakes

- Always-running process
- Wakes every autovacuum_naptime
  (default 1 min)
- Picks a database (wraparound
  risk first, else least recent)
- Starts a worker

→

📋 Check Thresholds

- Worker scans pg_class twice
  (tables + TOAST)
- For each table: checks 3 triggers
  against cumulative statistics
- Details on next 3 slides

→

🗂️ Build Work List

- Pre-PG19: pg_class scan order
  (roughly OID/creation order)
- PG19+: sorted by score
  (most urgent first)
- Wraparound-risk tables
  always get priority

→

⚡ Start Worker

- Max autovacuum_
  max_workers (3)
- Cost limit shared (autovacuum_vacuum_cost_limit)
- PG18: autovacuum_worker_slots

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Trigger 1: Update/Delete (Vacuum & Analyze)

🔄 Vacuum Trigger

• dead_tuples > autovacuum_vacuum_threshold
 + (autovacuum_vacuum_scale_factor × reltuples)

• Defaults: threshold=50, scale_factor=0.2

• 1K rows → after 250 dead tuples
• 100M rows → after 20M dead tuples ⚠️

📊 Analyze Trigger

• modified_tuples > autovacuum_analyze_threshold
 + (autovacuum_analyze_scale_factor × reltuples)

• modified = inserts + updates + deletes
• Defaults: threshold=50, scale_factor=0.1

• 1K rows → after 150 modifications

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Trigger 2: Insert Only (PG13+)

➕ Formula

- new_inserts > autovacuum_vacuum_insert_threshold
  + (autovacuum_vacuum_insert_scale_factor
  × reltuples)
- Defaults: insert_threshold=1000, insert_scale_factor=0.2

❓ Why vacuum insert-only tables?

- No dead tuples — but still need vacuum:
- Visibility map updates → enables index-only scans

aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Trigger 3: XID Freeze (Mandatory)

🛡️ Freeze Triggers

- autovacuum_freeze_max_age (200M)
  → forced, runs even if AV disabled – instance/table

- Eager freezing (PG18+): normal vacuum opportunistically freezes all-visible pages

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Blocked Autovacuum

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Vacuum Blockers

Vacuum can't clean what's still visible to an active snapshot

1. ⏳ **Active** Statements

   Long-running queries

2. 💤 **Idle in** Transaction

   Open BEGIN waiting for client

3. 📦 **Prepared** Transactions

   Two-phase commit left unresolved

4. 📡 **Logical Replication.** Slots

   Inactive or slow slots/lag

5. 🔄 **Read** Replicas

   hot_standby_feedback is on

6. 📋 **Orphan** Temp Tables

7. 🗄️ **Invalid** Database

   Can't connect to vacuum

8. 🐛 **Bugs**

   Stuck workers, launcher crashes, edge cases

9. 💥 **Corruption**

   Invalid pages abort vacuum entirely

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Blockers #1 & #2: Long-Running & Idle in transaction Sessions

⏳ Long-Running & Idle in transaction Sessions

- Long-running SELECT holds a snapshot — vacuum can't clean
- idle in transaction — BEGIN with no COMMIT, snapshot held open based on the transaction isolation level
- BufferPin: vacuum stuck on buffer pinned by another backend
  Does NOT show in pg_locks — check wait_event instead
- Detection: pg_stat_activity WHERE state IN ('active','idle in transaction')
  ORDER BY age(backend_xmin) DESC
- For BufferPin: WHERE query LIKE 'autovacuum:%' AND wait_event = 'BufferPin'

✅ Prevention & Fix

- idle_in_transaction_session_timeout = '30min' — auto-terminate idle sessions
- statement_timeout for long queries — prevent runaway SELECTs
- Monitor: oldest age(backend_xmin) — the oldest xmin is your blocker
- For BufferPin: identify pinning backend → pg_terminate_backend()

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Blockers #3 & #6: Prepared Transaction and Orphaned Temporary Table

📦 Prepared Transactions

- PREPARE TRANSACTION — survives restarts
- Holds XID → can't freeze past it

- Detection: pg_prepared_xacts
  age(transaction) DESC

📋 Orphaned Temporary Tables

- AV drops orphaned temp tables automatically (since PG8)
- Real danger: active temp table + wraparound hit
- Can't drop (needs XID) → single-user mode required

✅ Fix

- Prepared Tx: COMMIT PREPARED 'name' or ROLLBACK PREPARED 'name'
- Temp Tables: kill owning session or DROP early — active temp tables can cause wraparound

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Blockers #4 & #5: Replication

📡 Logical Replication Slots

- Inactive slots hold catalog_xmin forever
- Active but slow → delayed cleanup

- Detection: pg_replication_slots
  age(catalog_xmin) DESC

🔄 Read Replicas (physical replica)

- hot_standby_feedback holds back primary vacuum
- Long running query on replica → bloat on primary

- Detection: pg_stat_activity

✅ Fix

- Slots: drop inactive slots, PG16+ idle_replication_slot_timeout
- Replicas: monitor long query, set max_standby_streaming_delay
- Monitor: pg_replication_slots + pg_stat_activity

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Blockers #7, #8, #9: Edge Cases

🗄️ Invalid Database

- Can't connect → can't vacuum
- XID age keeps growing

- **Detection:** `pg_database WHERE datconnlimit = '-2'` from 11.21, 12.16, 13.12, 14.9, 15.4

- Fix: DROP DATABASE

🐛 Bugs

- Detection: set log_autovacuum_min_duration and log_min_messages
- Stuck workers – pg_stat_activity + pg_stat_progress_vacuum

Fix: stay current on minor releases

💥 Corruption

- ERROR: invalid page in block 155 of relation…
- Vacuum aborts entirely

- Fix: recover table from backup

Emergency: zero_damaged_pages = on (data loss risk!)

- Prevention: data_checksums

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Blocked: Quick Diagnostic

Start: SELECT pid, state, age(backend_xmin) FROM pg_stat_activity ORDER BY age(backend_xmin) DESC

Active statements	pg_stat_activity WHERE state = 'active'
Idle in transaction	pg_stat_activity WHERE state = 'idle in transaction'
Prepared transactions	pg_prepared_xacts
Logical replication slots	pg_replication_slots WHERE slot_type = 'logical'
Read replicas	hot_standby_feedback on replica
Temporary tables	pg_class WHERE relpersistence = 't'
Invalid database	pg_database WHERE datconnlimit = '-2'
Bugs (stuck workers)	pg_stat_activity + pg_stat_progress_vacuum
Corruption	PostgreSQL logs: 'invalid page' errors

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Backlogged Autovacuum

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Why Is Autovacuum Backlogged?

More work than workers — the queue grows faster than processing

Too Many Objects,
Too Few Workers

Tables + partitions + catalogs
vs default 3 workers

Dead Tuple Rate
Exceeds Throughput

Bursty workloads + shared
cost limit throttle

Large Tables &
Conservative Thresholds

500GB eats up workers +
scale_factor=0.2 too late

Aggressive Freeze
Vacuum Takes Over

Steals workers from
normal cleanup

Bloated pg_class
Slows List Building

AV scans pg_class TWICE
DDL churn bloats catalogs

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Backlog #1: Too Many Objects, Too Few Autovacuum Workers

3

Default
max_workers

vs

500+

Tables needing
vacuum

📊 User Tables

- 500+ tables with active writes
- 3 workers can't keep up
- If workers = max consistently → backlogged
- Monitor: pg_stat_activity
  WHERE query LIKE 'autovacuum:%'
- Monitor: pg_stat_progress_vacuum
- Monitor: eligible tables

Fix

- Increase autovacuum_max_workers
  (+ cost_limit proportionally)
- PG18: autovacuum_worker_slots
  (dynamic, no restart)
- VACUUM pg_class manually if bloated
- Reduce DDL churn (fewer CREATE/DROP)

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Backlog #2: Dead Tuple Rate Exceeds Vacuum Throughput

💥 Bursty Workloads

- Bulk loads followed by mass deletes
- High-frequency UPDATE on hot tables
  (sessions, counters, queues)
- Dead tuples generated faster than vacuum can clean
(ex: queue tables)

🔧 Shared Cost Limit Throttle

- autovacuum_vacuum_cost_limit = 200
  SHARED across all workers
- 3 workers → each gets ~67 budget
- Adding workers without increasing
  cost_limit = slower per worker

Fix

- Increase cost_limit while increasing the workers
- Schedule manual VACUUM off-peak for bulk load tables
- Monitor dead tuple generation rate vs vacuum throughput

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Backlog #3: Large Tables & Conservative Thresholds

⏳ Large Tables Eat up Workers

- 500GB table ties up a worker for hours
- 200+ smaller tables wait in queue
- Single worker, single-threaded per table
- Monitor: pg_stat_progress_vacuum for long-running autovacuum workers

📏 Thresholds Too Conservative

- Default scale_factor = 0.2 (20%)
- 100M rows → 20M dead tuples to trigger!
- 1B rows → 200M dead tuples!
- Vacuum starts way too late on large tables
- PG18: autovacuum_vacuum_max_threshold (default 100M)

✅ Fix

- Partition large tables — smaller partitions = faster vacuum
- Per-table tuning: ALTER TABLE hot_table SET (autovacuum_vacuum_scale_factor = 0.01)
- Schedule manual VACUUM on large tables during off-peak
- Per-table settings > global changes — don't change global for one table's behavior
- Drop unused/duplicate indexes
- HOT (Heap Only Tuples) - UPDATE-heavy workloads, it can be a life saver to avoid indexing the updated columns and setting a fillfactor of less than 100

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Backlog #4: Aggressive Freeze Vacuum Takes Over

⚠️ What Happens

- When age(relfrozenxid) exceeds autovacuum_freeze_max_age (default 200M)
- Aggressive anti-wraparound vacuum kicks in
- Steals workers from normal cleanup
- Regular vacuum backlog grows

⚠️ track_counts = off? OR autovacuum disabled?

- Autovacuum has no stats — flying blind
- Dead tuple / insert triggers won't fire
- Only XID wraparound vacuum works
- Check: SHOW track_counts;
- Must be ON for autovacuum to function
- Autoanalyze never run

🔍 Detection

- SELECT relname, age(relfrozenxid)
  FROM pg_class WHERE relkind = 'r'
  ORDER BY age(relfrozenxid) DESC
- pg_stat_activity: look for
  '(to prevent wraparound)'

✅ Prevention

- Monitor age(relfrozenxid) with alerts
  Warning: 500M, Critical: 1B
- Don't set freeze_max_age too low or too high

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Backlog #5: Bloated pg_class Slows Table List Building

⚠️ The Problem

- AV worker scans pg_class TWICE to build work list (tables + TOAST)
- Bloated pg_class → slow scan → delays entire autovacuum cycle

Common causes of pg_class bloat:

- Repeated CREATE/DROP of temp tables
- Excessive DDLs

🔍 Detection & Fix

Detection:

- SELECT n_dead_tup, n_live_tup FROM pg_stat_all_tables WHERE relname = 'pg_class'
- High n_dead_tup = bloated catalog

Fix:

- VACUUM pg_class (manual)
- Reduce DDL churn — reuse temp tables or use CTEs
- Use ON COMMIT DROP sparingly
- Consolidate partition maintenance

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Slow Autovacuum

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Why Is Autovacuum Slow?

Autovacuum is running, not blocked, not backlogged — but each run takes too long

I/O
Bottlenecks

Slow storage, shared bandwidth, WAL amplification

Cost-Based
Delay Throttle

Intentional slowdown shared across workers

Insufficient
maintenance_work_mem for index cleanup

Multiple index passes

Large Table
Challenges

Many indexes, TOAST, no parallelism

More Indexes

Bloated/unused indexes slow vacuum, slow vacuum causes more bloat

Slow vacuum freeze

Approaching wraparound

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Slow #1 & #2: I/O Bottlenecks & The Vacuum Throttle

💾 I/O Bottlenecks

- Slow storage / under-provisioned IOPS
- Vacuum competes with queries for disk
- WAL write amplification from vacuum

⏱️ Cost-Based Delay (The Throttle)

- Vacuum sleeps 2ms after 200 cost units
- Cost limit SHARED across all workers
- 3 workers sharing 200 = ~67 each
- Cost: hit=1, miss=2, dirty=20

✅ Fix

- Increase autovacuum_vacuum_cost_limit to 600-1000 for modern SSDs – decrease autovacuum_vacuum_cost_delay
- vacuum_buffer_usage_limit (PG16+): ring buffer size (default 2MB)— if set to 0, uses shared_buffers – 1/8 is cap
  Increase for large tables, decrease to protect cache
- Schedule vacuum during off-peak to reduce I/O contention

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Slow #3: Insufficient maintenance_work_mem for index cleanup

```
SET maintenance_work_mem = '64MB';
VACUUM VERBOSE larget;
-[ RECORD 1 ]-------------+------------------
 duration                 | 00:27:52
 scanned_pct              | 9.9%
 index_vacuum_count       | 28
 num_dead_tuples          | 51,751,802
 max_dead_tuples/cycle    | 174,762
 cycles_required          | 296
```

➔

```
SET maintenance_work_mem = '1GB';
VACUUM VERBOSE larget;
-[ RECORD 1 ]-------------+------------------
 duration                 | 00:00:22
 scanned_pct              | 100.0%
 index_vacuum_count       | 0
 num_dead_tuples          | 51,751,802
 max_dead_tuples/cycle    | 110,495,028
 cycles_required          | 0
```

❌ 174K/cycle → 296 cycles → 28 index passes in 27 min

✅ 110M/cycle → 1 pass → 100% in 22 sec — 76× faster!

🆕 PG17 Changes

- 1GB cap on maintenance_work_mem removed (was hardcoded)
- pg_stat_progress_vacuum: max_dead_tuples → max_dead_tuple_bytes
- New columns: indexes_total, indexes_processed, num_dead_item_ids
- Parallel index vacuum: max_parallel_maintenance_workers

🔧 autovacuum_work_mem (PG15+)

- Separate from maintenance_work_mem
- Controls memory for autovacuum only
- Default: -1 (uses maintenance_work_mem)
- Set independently for fine-grained control

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Slow #4 & #5: Large Tables and Too Many Indexes

📦 Large Table Challenges

- TOAST tables need separate vacuum
- No parallelism in autovacuum (PG19 – autovacuum_max_parallel_workers)
- Single worker, single-threaded per table
- 500GB table ties up a worker for hours

📇 Too Many Indexes

- Vacuum scans EVERY index per pass
- 10 indexes = 10 full index scans
- Bloated indexes = more pages to scan
- Unused indexes waste vacuum time

✅ Fix

- DROP unused indexes — they slow vacuum for zero query benefit
- REINDEX CONCURRENTLY — rebuild bloated indexes without locks
- pg_repack — zero-downtime table + index rebuild
- Partition large tables — smaller partitions = faster vacuum
- PG19: autovacuum_max_parallel_workers for parallel index vacuum
- pre-PG19, manual parallel vacuum using " SET MAX_PARALLEL_MAINTENANCE_WORKERS=4;" (hard cap) or "VACUUM (PARALLEL 4, VERBOSE) parallel_vacuum;"

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Slow #6: Slow vacuum freeze

⚡ INDEX_CLEANUP off TRUNCATE off (PG12+)

- Skip index dead-pointer removal
- Skip tail truncation (avoids lock)
- VACUUM (INDEX_CLEANUP off, TRUNCATE off) tablename; -

PG18: vacuum_truncate parameter

Failsafe (PG14+)

- Triggers at vacuum_failsafe_age (default 1.6B XIDs)
- Auto-disables cost delay
- Auto-skips index cleanup + truncate
- Same as manual INDEX_CLEANUP off

Prioritize tables (PG19)

- autovacuum score weigths parameters (autovacuum_freeze_score_weight)

Pre-PG19, prioritizing tables with the highest XID age – “vacuumdb .... -min-xid-age <example 1000000000 >”

⚠️ When Failsafe Fires — What It Means

- Regular vacuum failing for a LONG time (approaching 1.6B XIDs)
- Failsafe = automatic INDEX_CLEANUP off — last resort
- Indexes left bloated → REINDEX CONCURRENTLY needed after
- Prevention: monitor age(relfrozenxid), fix blockers early
- If failsafe fires, treat as incident — investigate root cause

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Detection & Monitoring

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Your Autovacuum Dashboard

Six metrics to monitor — set alerts BEFORE crisis mode

age(datfrozenxid)

⚠️ 500M | 🔴 1B

n_dead_tup > 1M

⚠️ 5 tables | 🔴 20

last_autovacuum

❗ > 24h | 🔴 > 7d

AV workers = max

❗ > 50% | 🔴 > 90%

Slot xmin age

⚠️ 100M | 🔴 500M

idle in transaction

⚠️ > 5 min | 🔴 > 30 min

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Monitoring Queries & Logging

🔍 Key Catalog Views

pg_stat_user_tables:
 n_dead_tup, last_autovacuum
pg_stat_activity:
 WHERE query LIKE 'autovacuum:%'
pg_stat_progress_vacuum:
 phase, pct_scanned, index_vacuum_count
pg_replication_slots:
 xmin age, catalog_xmin, WAL lag
pg_database / pg_class:
 age(datfrozenxid), age(relfrozenxid)

📝 log_autovacuum_min_duration

Set to 0 → log every autovacuum run
Set to 60000 → log runs > 1 min

Log tells you:

- Tuples removed vs remain
- Index scans (passes) count
- I/O rates (read/write MB/s)
- Buffer usage (hits, misses)
- WAL usage and CPU time

Parse logs for trending over time

💡 Always enable logging in production — it's your best debugging tool for vacuum issues

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

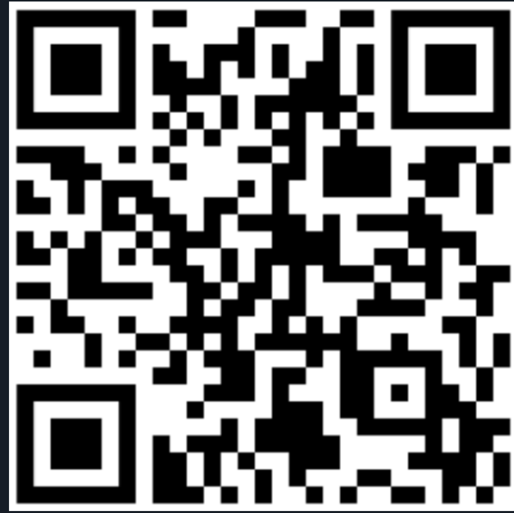
What is PG Collector ?

PG Collector is a diagnostic tool designed to gather comprehensive health and performance data from PostgreSQL databases. It produces a consolidated HTML report for deep-dive analysis and provides an automated database health check to identify critical risks.

PG Collector is safe to run on production environments and does not create any database objects to produce the output.

PG Collector published Under MIT-0 license part of awslabs Github.

https://github.com/awslabs/pg-collector



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

INFO			
Database size	DB parameters	Transaction ID TXID (Wraparound)	Table Size
Index Size	Vacuum & Statistics	Extensions	Memory setting
pg_stat_statements extension	Users & Roles Info	schema Info	Tablespaces Info
Table Access Profile	Unused Indexes	Index Access Profile	Fragmentation (Bloat)
Toast Tables Mapping	Replication	Sessions/Connections Info	Orphaned prepare transactions
PK or FK using numeric or integer data type	public Schema	invalid indexes	Default access privileges
pgaudit extension	Unlogged Tables	Access privileges	ssl
Background processes	Multixact ID MXID (Wraparound)	Temp tables	Large objects
Partition tables	pg_shdepend	FK without index	sequences
pg_hba.conf	Duplicate indexes	Functions statistics	DB Load
Triggers	pg_config	COPY command progress	Invalid databases
Index Creation Progress	Materialized Views	******	******

Transaction ID TXID (Wraparound)

▼ Details

Vacuum Blockers :

NOTE: the vacuum blocker will check if the XID age is above 300 million
No inactive replication slot found
No orphaned prepared transactions found
No active Logical Replication Slots with high XID age found
No long running queries found
No Physical Replication Slots with high XID age found

oldest xid:

oldest_xid
5099

oldest xid per database:

database_name	oldest_xid_per_db
template0	5099
template1	5099
postgres	5099
rdsadmin	5099

percent_towards_emergency_autovac & percent_towards_wraparound :

oldest_current_xid	percent_towards_wraparound	percent_towards_emergency_autovac
5099	0	0

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

PostgreSQL Version Improvements

PG13-16: parallel index vacuum • failsafe mechanism • idle_replication_slot_timeout • autovacuum_work_mem

PG17

- 1GB maintenance_work_mem cap removed
- vacuum_buffer_usage_limit (ring buffer control)
- pg_stat_progress_vacuum: max_dead_tuple_bytes, indexes_total, indexes_processed

PG18

- Async I/O (AIO) — faster heap scans
- Dynamic autovacuum_max_workers (no restart needed!)
- autovacuum_vacuum_max_threshold (hard cap on dead tuples)
- Eager freezing during normal vacuum
- vacuum_truncate parameter

PG19

- Score-based table prioritization
- 5 weight parameters (0.0–10.0): freeze, multixact_freeze, vacuum, vacuum_insert, analyze
- score = MAX(value/threshold)
- Tables near wraparound boosted
- autovacuum_max_parallel_workers (parallel index in autovacuum!)

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Key Takeaways

1. Autovacuum is NOT optional — prevents bloat AND wraparound
2. Blocked → fix the blocker (transactions, DDL, slots)
3. Backlogged → tune thresholds & add workers
4. Slow → tune cost limits, memory, fix index bloat
5. Monitor proactively — n_dead_tup, XID age, worker saturation
6. Log everything — log_autovacuum_min_duration = 0
7. Per-table tuning > global changes for hot tables

aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Baji Shaik
Sr DB Engineer
Amazon RDS

Mohamed Ali
Sr DB Engineer
Amazon RDS