Vacuum and autovacuum in a managed PostgreSQL service

Anastasia Lubennikova, neon.tech

Berlin PostgreSQL meetup, July 16, 2024

NEON

Why this talk?

I work on the Neon compute team.

We keep Neon PostgreSQL fleet healthy and performant.

>500.000 projects on a platform	~7000 active computes at a time	7 engineers in the team	* variable load patterns

Neon architecture provides unique opportunities and challenges

for PostgreSQL maintenance and tuning

NEON

Overview

1. What is VACUUM and why is it needed?
2. How the autovacuum background process works and how to tune it?
3. What do PostgreSQL service maintainers need to set up?
4. What can users do to improve database performance?
5. Neon-specific database maintenance challenges

Why do we need GC in Postgres?

- DELETEs and UPDATEs only mark table rows as deleted, because they can still be visible to concurrent transactions.
- Failed or rolled back INSERTs also leave behind rows that are not visible to anyone.
- Eventually, rows end up being visible to either everyone or no one.
- Invisible tuples are called "dead" and can be removed to reclaim space.
- All-visible tuples can be marked as such ("frozen") to optimize performance and xid counter use.

VACUUM - SQL command
that performs maintenance operations on database tables

VACUUM FREEZE

- Mark tuples visible to all transactions with special FrozenXID.
- Reclaim XIDs for reuse
- Prevent XID wraparound
- Not optional

VACUUM

- Mark space occupied by “dead” tuples as free for reuse
- Update visibility map to use index-only scans

VACUUM ANALYZE

- Collect data statistics used by the query planner to improve query performance

VACUUM FULL, CLUSTER and REINDEX

- VACUUM FULL and CLUSTER reclaim disk space by moving visible data into a new file.
  These operations lock the table exclusively (no concurrent writes or reads!).

- pg_repack and pg_squeeze extensions provide non-locking solution
  at the cost of extra space (2x table size) and some overhead.

- REINDEX locks the table for writes.
  REINDEX CONCURRENTLY doesn’t block writes but has some limitations.

Maintenance operations

VACUUM FREEZE **VACUUM** **VACUUM ANALYZE**

- Need to read the whole table page by page.
- Need to run on all active tables regularly to keep up with data changes.

autovacuum

- autovacuum launcher - periodically spins workers in every database

  ```
  postgresql.conf autovacuum = on (requires restart)
  ```

- autovacuum workers - runs VACUUM (FREEZE, ANALYZE) on database tables

  ```
  postgresql.conf autovacuum_max_workers = 3 (requires restart)
  ```

- Launcher tries to start a worker in each database once per naptime interval.

  ```
  postgresql.conf autovacuum_naptime = 1min (requires reload)
  ```

autovacuum

	Increase	Decrease
autovacuum_max_workers	if autovacuum is not keeping up	if autovacuum uses too much memory
autovacuum_naptime	if autovacuum consumes too much CPU because of large number of databases	if you have large number of tables or small tables that need to be checked more often

autovacuum_work_mem

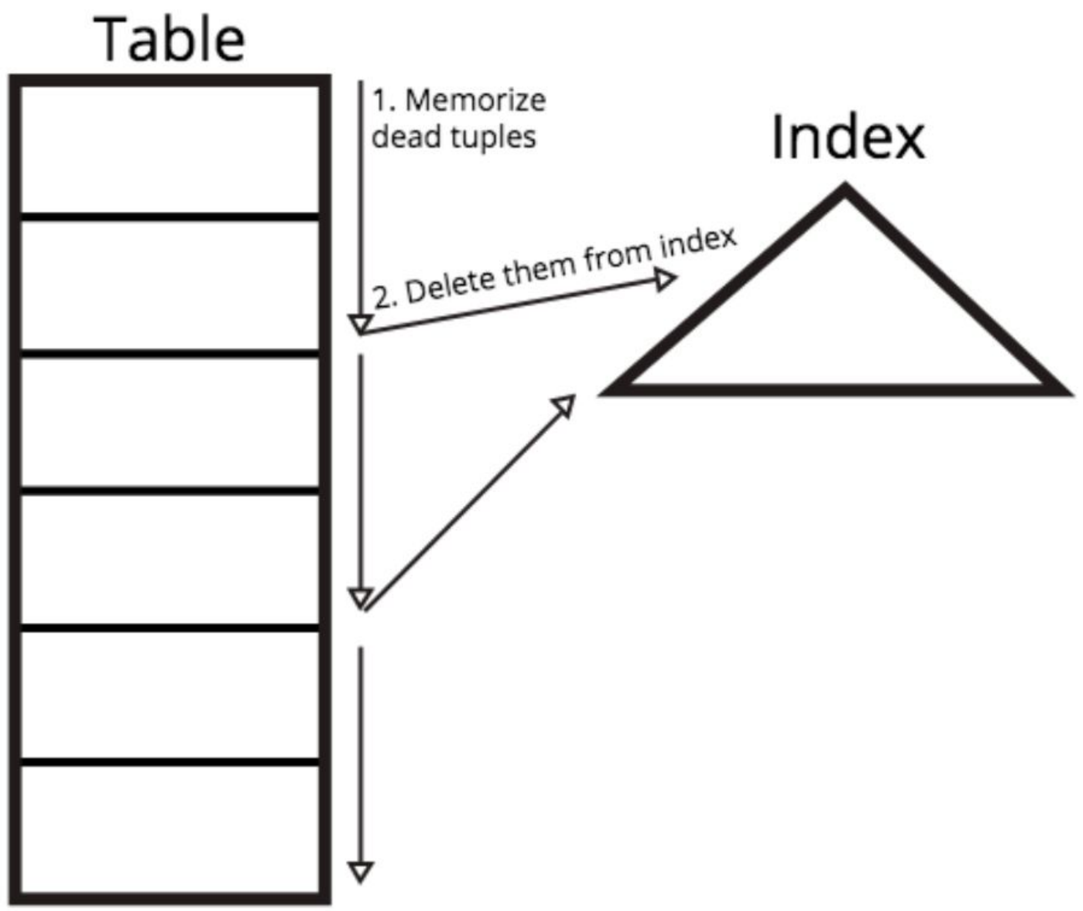
- Maximum amount of memory to be used by **each** autovacuum worker process

```
postgresql.conf autovacuum_work_mem = -1 (requires reload)
postgresql.conf maintenance_work_mem = 64MB
```

What is this memory used for?

If a table has indexes, VACUUM happens in 3 steps:

1) Scan heap, memorize "dead" tuples
2) Vacuum index
3) Vacuum heap



NEON

How to decide if a table needs autovacuum FREEZE, VACUUM, or ANALYZE?

autovacuum FREEZE thresholds

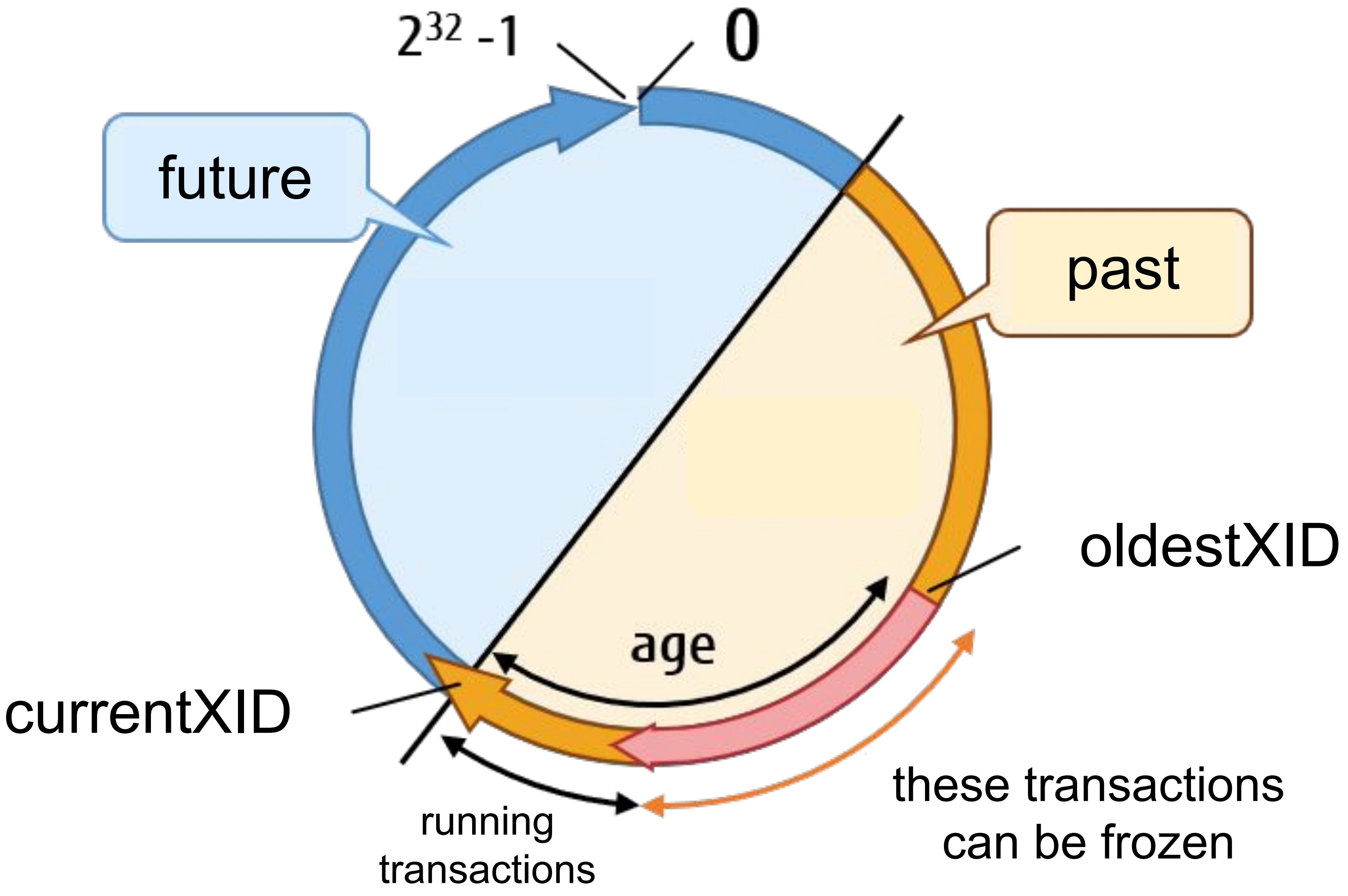
- `autovacuum_freeze_max_age = 200.000.000`
- `autovacuum_multixact_freeze_max_age = 400.000.000`

```
number of xids left =
        2^32 / 2 - oldest_xid
```

XID counter is 32-bit

For each XID:

- half the numbers **before** it is the **past**
- half the numbers **after** it is the **future**



https://www.fujitsu.com/jp/products/software/resources/feature-stories/postgres/article-index/tuningrule9-base/

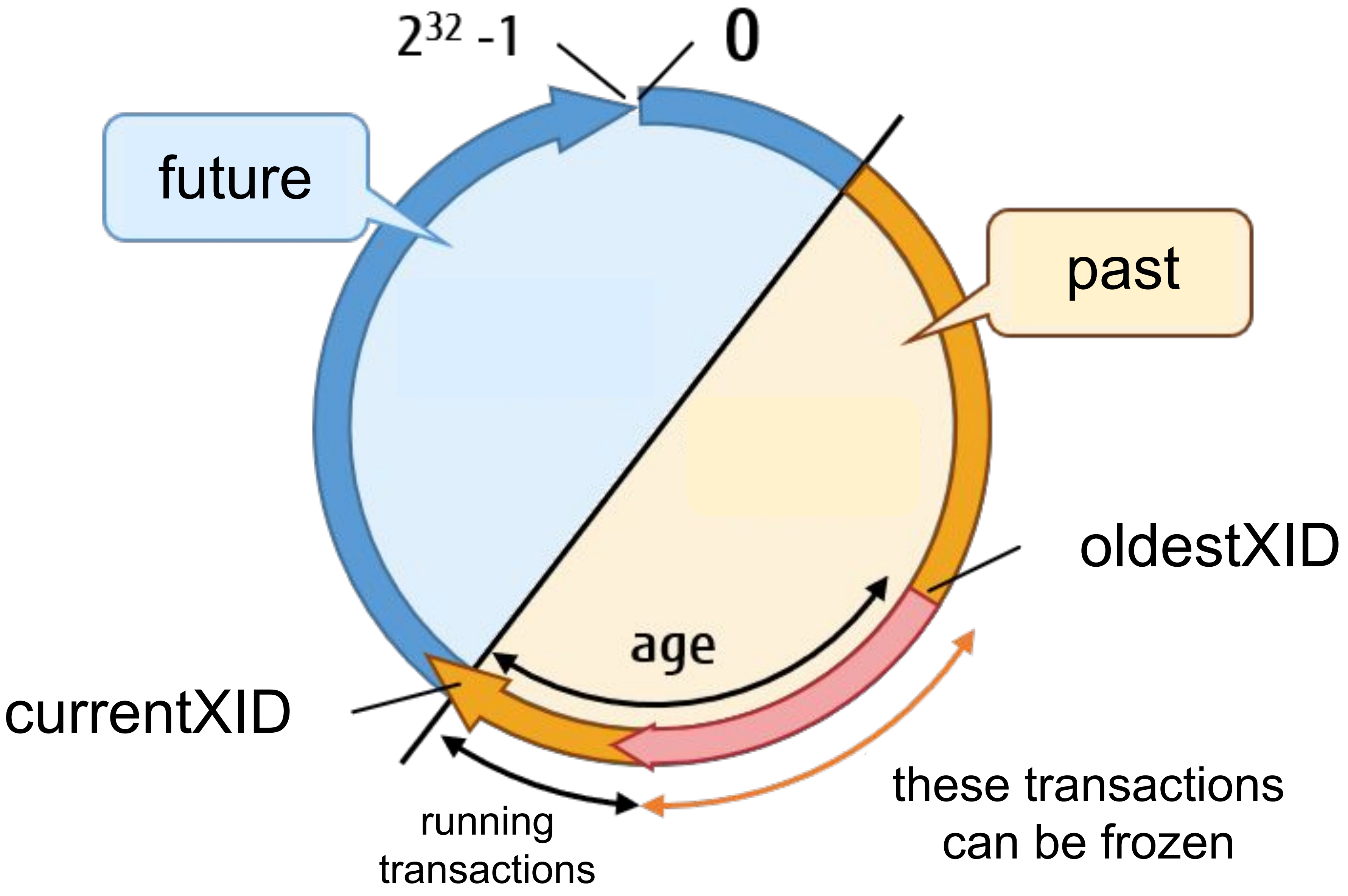
autovacuum FREEZE thresholds

- `autovacuum_freeze_max_age = 200.000.000`
- `autovacuum_multixact_freeze_max_age = 400.000.000`

table_age = oldest used xid of a row in a table

When table_age >= max_age
autovacuum must kick in.

The closer table_age to 2^32 / 2 the
more urgent is FREEZE.



https://www.fujitsu.com/jp/products/software/resources/feature-stories/postgres/article-index/tuningrule9-base/

database xid age monitoring

```
-- if database is freeze_overdue by a lot, it means that autovacuum is not keeping up

with data as (
    select
      oid,
      datname,
      age(datfrozenxid) as xid_age,
      mxid_age(datminmxid) as mxid_age,
      pg_database_size(datname) as db_size
    from pg_database
  )
  select
    xid_age >= current_setting('autovacuum_freeze_max_age')::INTEGER as freeze_overdue,
    mxid_age >= current_setting('autovacuum_multixact_freeze_max_age')::INTEGER as mxid_freeze_overdue,
    *,
    pg_size_pretty(db_size) as db_size_hr
  from data
  order by greatest(xid_age, mxid_age) desc;
```

https://x.com/samokhvalov/status/1722585894430105822

per-table xid age monitoring

```
-- find tables overdue for vacuum freeze

WITH tables AS (
  SELECT
    c.relname,
    GREATEST(age(c.relfrozenxid), age(t.relfrozenxid)) AS age
  FROM pg_class AS c
    LEFT JOIN pg_class AS t ON c.reltoastrelid = t.oid
  WHERE c.relkind IN ('r', 'm')
)
SELECT age >= current_setting('autovacuum_freeze_max_age')::INTEGER as freeze_overdue, *
FROM tables
ORDER BY age DESC;
```

https://github.com/heap/pg-autovacuum-diagnostics/blob/master/scripts/overdue_for_aggressive_autovacuum.sql

autovacuum VACUUM thresholds

- autovacuum_vacuum_threshold
- autovacuum_vacuum_scale_factor

- autovacuum_vacuum_insert_threshold
- autovacuum_vacuum_insert_scale_factor

```
updates or deleted tuples >
   autovacuum_vacuum_threshold + autovacuum_vacuum_scale_factor * number of tuples

OR

inserted tuples >
   autovacuum_vacuum_insert_threshold + autovavuum_vacuum_insert_scale_factor * number of tuples
```

autovacuum ANALYZE thresholds

- `autovacuum_analyze_threshold`
- `autovacuum_analyze_scale_factor`

number of tuple inserts, updates or deletes >
 autovacuum_analyze_threshold + autovacuum_analyze_scale_factor * number of rel tuples

autovacuum cost based throttling

- `autovacuum_cost_limit`
- `autovacuum_vacuum_cost_delay`

```
vacuum_cost =
vacuum_cost_page_hit (1) + vacuum_cost_page_miss (10)  +
vacuum_cost_page_dirty (20)


if vacuum_cost > autovacuum_cost_limit (200):
    sleep for autovacuum_vacuum_cost_delay (2ms)
```

autovacuum cost based throttling (2)

With the default parameters,
autovacuum will at most
write 4MB/s to disk,
or
read 8MB/s from disk or the OS page cache
or
read 80MB/s from shared buffers

Don’t hesitate to increase

```
  autovacuum_cost_limit (requires reload)
```

“TIDY A LITTLE A DAY AND YOU’LL BE TIDYING FOREVER.”

- MARIE KONDŌ

ITSALLYOUBOO.COM

Monitor last autovacuum time

```
WITH tables AS (
  SELECT
    relname,
    pg_stat_get_last_vacuum_time(c.oid) AS last_vacuum_time,
    pg_stat_get_last_autovacuum_time(c.oid) AS last_autovacuum_time
  FROM pg_class AS c
  WHERE c.relkind IN ('r', 'm')
)
SELECT *
FROM tables
ORDER BY GREATEST(last_vacuum_time, last_autovacuum_time);
```

Monitor bloat with the pgstattuple extension

```
CREATE EXTENSION pgstattuple;

-- full table scan. A lot of I/O !
SELECT relname, (pgstattuple(oid)).* FROM pg_class WHERE relkind = 'r'
ORDER BY dead_tuple_percent desc;

-- way faster, because it skips all-visible pages
SELECT relname, (pgstattuple_approx(oid)).* FROM pg_class WHERE relkind = 'r'
ORDER BY dead_tuple_percent desc;
```

NEON

Permissions to VACUUM and monitor

- By default a table can only be vacuumed or analyzed by its owner or a superuser.

  GRANT VACUUM explicitly per-object (pg16+)

  or GRANT predefined roles `pg_vacuum_all_tables`, `pg_analyze_all_tables` or `pg_maintain`

- By default, system and statistics views expose limited information to regular users

  GRANT predefined roles `pg_read_all_stats` or `pg_monitor` (pg10+)

Permissions to VACUUM and monitor (2)

- Shared catalog tables can only be vacuumed or analyzed by superuser (or by autovacuum)

```
select relname from pg_class where relisshared and relkind in ('r', 'm');

        relname
-----------------------
 pg_authid
 pg_subscription
 pg_database
 pg_db_role_setting
 pg_tablespace
 pg_auth_members
 pg_shdepend
 pg_shdescription
 pg_replication_origin
 pg_shseclabel
 pg_parameter_acl
```

What can users do to improve database performance?

1. Don’t block autovacuum
2. Use advanced PostgreSQL features for special use cases
3. Monitor table bloat and statistics quality and tune thresholds per object

What can users do to improve database performance?

1. Don’t block autovacuum:
   - Avoid long-running transactions

     `idle_in_transaction_session_timeout`
   - Rollback unused PREPARED transactions

     ROLLBACK PREPARED
   - Drop unused replication slots
   - Ensure that replica with `hot_standby_feedback = on` is not lagging
   - Drop unused indexes, they slow down vacuum and block HOT-updates

What can users do to improve database performance?

2. Follow best-practices and use advanced PostgreSQL features for special use cases:

    - Use partitioning
    - Use TRUNCATE to bulk delete data
    - Use COPY FREEZE for data loading
    - Change the workload so that fewer dead tuples are generated
        - tune fillfactor
        - decrease the number of insert conflicts

What can users do to improve database performance?

3. Monitor table bloat and statistics quality and tune thresholds per object:

    - Monitor table and index bloat
    - Fine-tune autovacuum thresholds per database or per table
    - Run VACUUM ANALYZE manually after bulk data loading
    - Use pg_repack or pg_squeeze if necessary

Neon specific

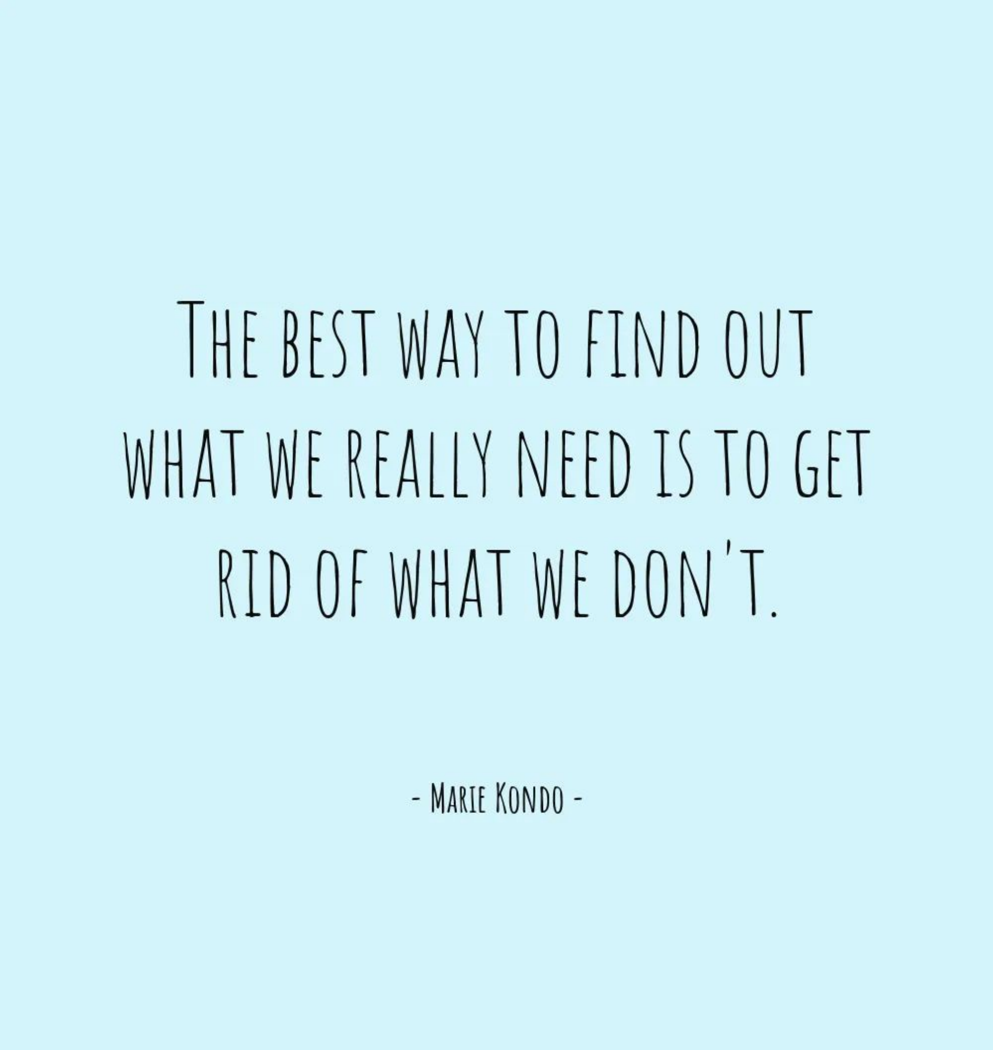
- We shut down idle computes, so we cannot rely on autovacuum running in the background.
  → We need to be intentional about autovacuum and tune it to be more active.

- We can autoscale memory and CPU without restart.
  → We can run more active autovacuum when there is no load to utilize resources more effectively.

- PostgreSQL run-time statistics are stored differently than data
  Statistics are reset after a non-clean shutdown.
  → Extra effort is required to preserve run-time statistics between restarts.

NEON

Thank you!



The best way to find out what we really need is to get rid of what we don't.

- Marie Kondo -