# VACUUM and Autovacuum: a short overview

# Overview

1) Why do we need VACUUM and autovacuum?
2) Types of VACUUM:
   - VACUUM
   - VACUUM FULL
   - FREEZE
   - ANALYZE
3) Autovacuum
4) Possible changes in newer versions of Postgres

# Purpose of VACUUM and autovacuum

In Postgres, DELETE does not remove rows from the relation.

UPDATE = INSERT + DELETE. This creates dead tuples.

**VACUUM:**

- Prevents table bloat by removing dead tuples
  (tuples not visible to any of the active transactions)
- Prevents crashes due to XID wraparound
- Gathers information for future query optimization

Vacuuming must happen frequently enough to perform all these tasks.
Manually launching VACUUM all the time would be inconvenient.
**Autovacuum** automates this process.

=> it is useful to have an idea of how VACUUM + autovacuum work for efficiently preventing possible problems with table bloat and wraparound.

# VACUUM

- Restructures pages and reclaims space taken by dead rows (rows that were **deleted BEFORE any of the current transactions started**)
- Removes dead rows from indexes and TOAST tables
- **Having long-running transactions can mess everything up** (including long transactions on replica if hot_standby_feedback == on)
- Truncates the table if possible
- Updates free space map
- **Done to avoid needing VACUUM FULL**

**NOT NEEDED:**

- On replica
- After TRUNCATE

# VACUUM FULL

- **Shrinks table size**
  (rewrites all “alive” tuples into a new file as compactly as possible)
- **Can only be launched manually** (not by autovacuum)
- OID of the relation stays the same, relfilenode (on-disk name) changes

**Cons:**

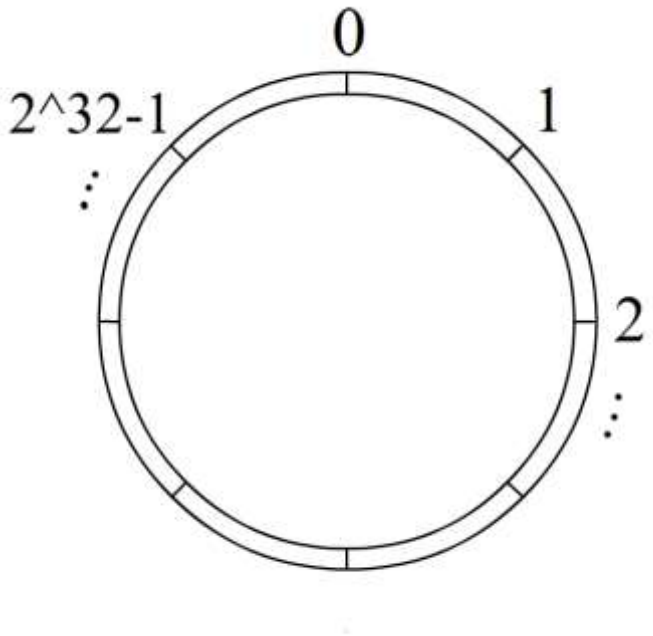
- **ACCESS EXCLUSIVE LOCK** (no reading or writing allowed)
- **table size ≤** needed space **≤ table size * 2**
- Need a **REINDEX**
- Takes a long time

**Alternative:**

**pg_repack** - does allow reads and writes, but needs more space (**≥ table size * 2**)

# VACUUM FREEZE (1)

- **Scans the table (without skipping blocks)** and freezes rows to prevent XID wraparound



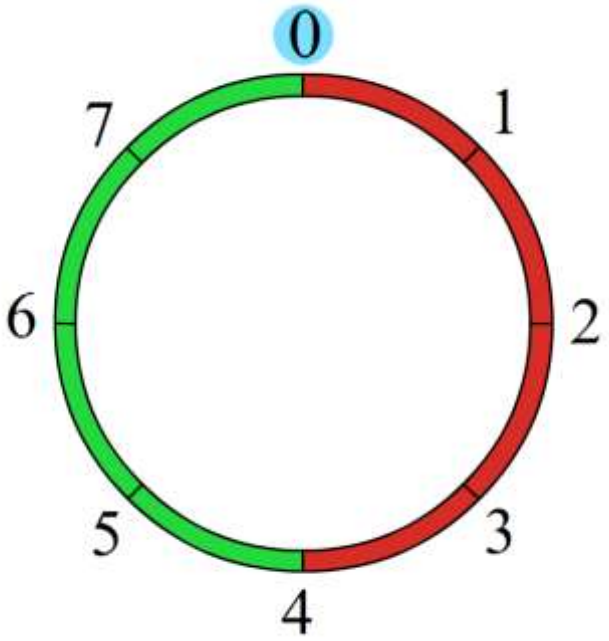

XIDs are 32-bit numbers

They make up a “ring”: after XID=2^32-1 we have XID=0 again.

For the current XID:
Half the ring “clockwise” = future
Half the ring “counterclockwise” = past

# VACUUM FREEZE (2)

- **Scans the table (without skipping blocks)** and freezes rows to prevent XID wraparound



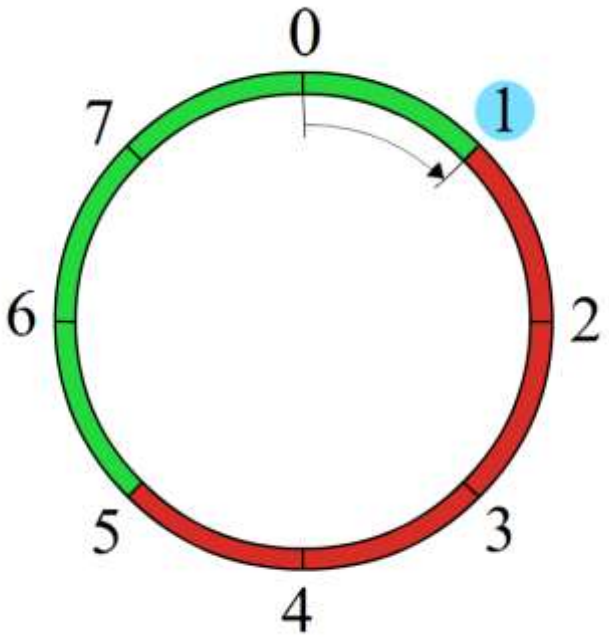

XIDs are 32-bit numbers

They make up a “ring”: after XID=2^32-1 we have XID=0 again.

For the current XID:
Half the ring “clockwise” = future
Half the ring “counterclockwise” = past

# VACUUM FREEZE (3)

- **Scans the table (without skipping blocks)** and freezes rows to prevent XID wraparound



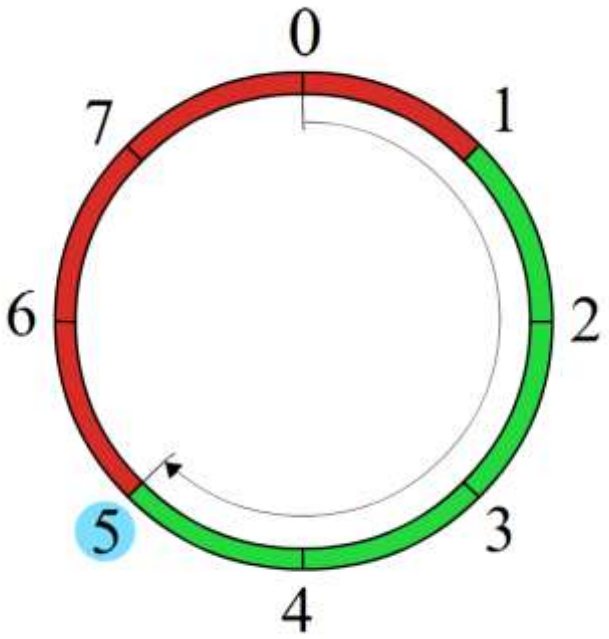

XIDs are 32-bit numbers

They make up a “ring”: after XID=2^32-1 we have XID=0 again.

For the current XID:
Half the ring “clockwise” = future
Half the ring “counterclockwise” = past

# VACUUM FREEZE (4)

- **Scans the table (without skipping blocks)** and freezes rows to prevent XID wraparound




XIDs are 32-bit numbers

They make up a “ring”: after XID=2^32-1 we have XID=0 again.

For the current XID:
Half the ring “clockwise” = future
Half the ring “counterclockwise” = past

# VACUUM FREEZE (5)

- **Scans the table** and freezes rows to prevent XID wraparound
- Uses visibility map to possibly skip over blocks
- VACUUM FULL performs freeze either way
- Cannot be performed separately from the normal VACUUM (for now)

# VACUUM ANALYZE

- Updates visibility map
- Updates table statistics
- Can be launched on its own (just ANALYZE)

# Autovacuum

- Consists of Autovacuum launcher + workers

- Runs VACUUM FREEZE to prevent XID wraparound
  (even if autovacuum=off)
- Runs VACUUM and ANALYZE to prevent bloat
  (if autovacuum=on and track_counts=on)
- Does not remove existing bloat
  (use VACUUM FULL or pg_repack for that)

- Turning it off is a bad idea (unless you really know what you're doing)
- Can be configured to be more effective

# Newer versions of PostgreSQL

What could change for VACUUM and Autovacuum in the near future:

1) FAST FREEZE

2) Faster or disabled table truncation at VACUUM

3) Block level parallel vacuum

4) Improved VACUUM for GiST

# FAST FREEZE

What could change for VACUUM and Autovacuum in the near future:
https://commitfest.postgresql.org/22/1817/

**PROBLEM:**

VACUUM FREEZE is critical for avoiding crashes due to XID wraparound, but
It can only be conducted alongside a normal VACUUM, which makes it slower

**PROPOSED SOLUTION:**

FAST_FREEZE (FREEZE_ONLY / WITHOUT_INDEX_CLEANUP) option that:

- Doesn't reclaim dead tuples
- Doesn't cleanup indexes

**STATUS:**

Achieved a significant speedup of FREEZE, discussing implementation details
Such a mode could also be used by autovacuum (as a separate patch).

# Table truncation at VACUUM

What could change for VACUUM and Autovacuum in the near future:
https://commitfest.postgresql.org/22/1981/

**PROBLEM:**

Table truncation at the end of VACUUM requires:

- Taking an ACCESS EXCLUSIVE LOCK
- Scanning shared buffers (can be slow)

If shared_buffers are big, other transactions have to wait

In some cases, table grows back right away

**PROPOSED SOLUTIONS:**

- Add a storage parameter for disabling table truncation for VACUUM
- Speed up shared buffers scan (useful for TRUNCATE and DROP)
- Memorize the buffers that we need to discard in advance

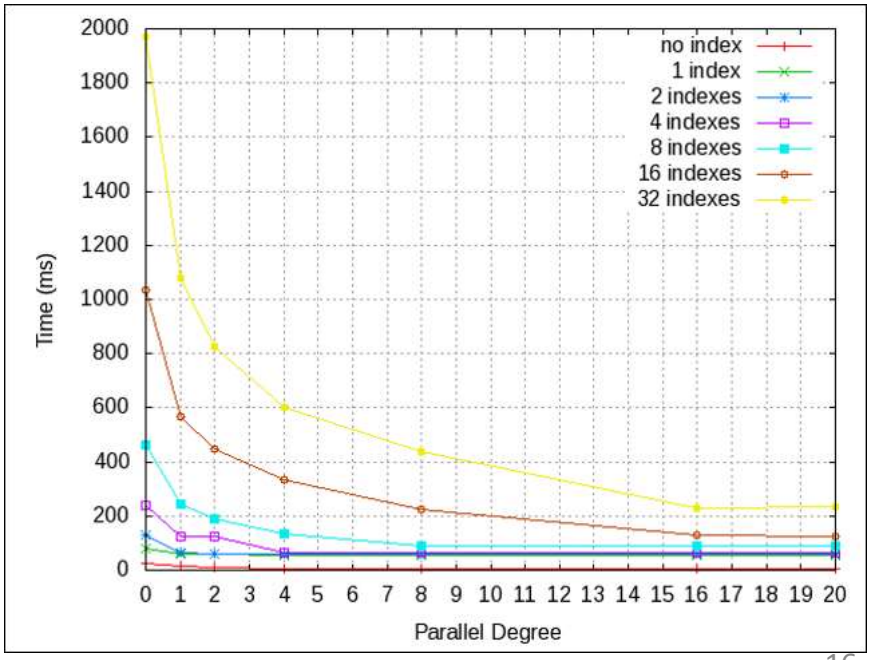
**STATUS**: Long-term solution might appear in PG 13, short-term is being discussed

# Block level parallel VACUUM

What could change for VACUUM and Autovacuum in the near future:
https://commitfest.postgresql.org/22/1774/

**GOAL:** Implement a block-level parallel VACUUM

**STATUS:** Idea is approved, patch gives improvement, discussing details and bugs (possibly will appear in PG13)




(Picture: test by Masahiko Sawada)

# Optimizing VACUUM for GiST

What could change for VACUUM and Autovacuum in the near future:
https://commitfest.postgresql.org/22/1598/

**PROBLEMS**:

- Empty pages for GiST aren't reused, which can lead to bloat
- Algorithm for scanning the relation is not optimal

**GOALS:**

- Make GiST reuse empty pages for new page splits
- Improve scanning algorithm to read GiST pages in physical order

**STATUS:** Active discussion of the code and its possible drawbacks.

# What to keep in mind

- Long-running transactions will mess things up
- Help autovacuum:
  - After inserting a lot of data call FREEZE
  - After deleting/updating a lot of data call VACUUM
- Temporary tables are not vacuumed by autovacuum
- A backend can only clean its own temporary tables

# Configuring autovacuum (1)

**Turning it on and logging:**

autovacuum = on (track_counts = on needed in order for this to work)

log_autovacuum_min_duration = N ms (log actions that took **≥ N ms,** -1 for no logging)

# Configuring autovacuum (2)

**Controlling when to VACUUM a table:**

autovacuum_vacuum_scale_factor (default: 0.2 - works bad with huge tables)

autovacuum_vacuum_threshold (default: 50)

We only decide to vacuum a table if the number of dead rows in it is **≥**

**≥** (number of rows in relation) * autovacuum_vacuum_scale_factor + autovacuum_vacuum_threshold

Sometimes it makes sense to redefine those values for specific relations:

ALTER TABLE tbl SET (autovacuum_vacuum_scale_factor = 0);

ALTER TABLE tbl SET (autovacuum_vacuum_threshold = 10000);

**Controlling when to ANALYZE a table:**

autovacuum_analyze_scale_factor

autovacuum_analyze_threshold

**Controlling when to FREEZE a table:**

autovacuum_freeze_max_age

autovacuum_multixact_freeze_max_age

# Configuring autovacuum (3)

**Frequency of launching workers:**

autovacuum_naptime – MIN delay between running autovacuum workers on ONE db

Between two launches of a worker in the whole cluster we wait
max( 110 , autovacuum_naptime / (number of databases)) ms

**Number of workers and their use of memory:**

autovacuum_work_mem (defaults to maintenance_work_mem)
– max amount of memory used by ONE autovacuum worker

autovacuum_max_workers
– max amount of autovacuum workers for the whole cluster

NOTES:

- Too few workers will lead to poor vacuuming, too many will lead to using too much memory
- You may want to increase autovacuum_work_mem if you see it going through the same indexes many times (or decrease autovacuum_vacuum_[scale_factor|threshold]).
- If you increase autovacuum_max_workers, you might want to increase autovacuum_vacuum_cost_limit too, because the limit is global for all workers

# Configuring autovacuum (4)

**Controlling I/O impact:**

autovacuum_vacuum_cost_limit (defaults to vacuum_cost_limit) - global for all workers!
autovacuum_vacuum_cost_delay (defaults to vacuum_cost_delay)

**ALGORITHM:**

Keep vacuuming until autovacuum_vacuum_cost_limit is reached,
sleep for autovacuum_vacuum_cost_delay ms, resume vacuuming.

Things contributing to reaching the limit are vacuum_cost_page_[ hit | miss | dirty ].

If there is a need to increase the throughput, It's easier to raise
autovacuum_vacuum_cost_limit than to adjust all the other parameters.

# Questions?

You can also ask me by email: **akenteva.anna@yandex.ru**