The present and future of VACUUM and Autovacuum

Akenteva Anna
Postgres Professional

Postgres PROFESSIONAL

postgrespro.ru

Overview

1. The role of VACUUM and Autovacuum
2. Issues and workarounds
3. Future prospects

1. The role of VACUUM and Autovacuum

Types of VACUUM

Main operations:

- VACUUM FULL (or CLUSTER)
- VACUUM
- VACUUM FREEZE
- VACUUM ANALYZE

What is the purpose?

Postgres PROFESSIONAL

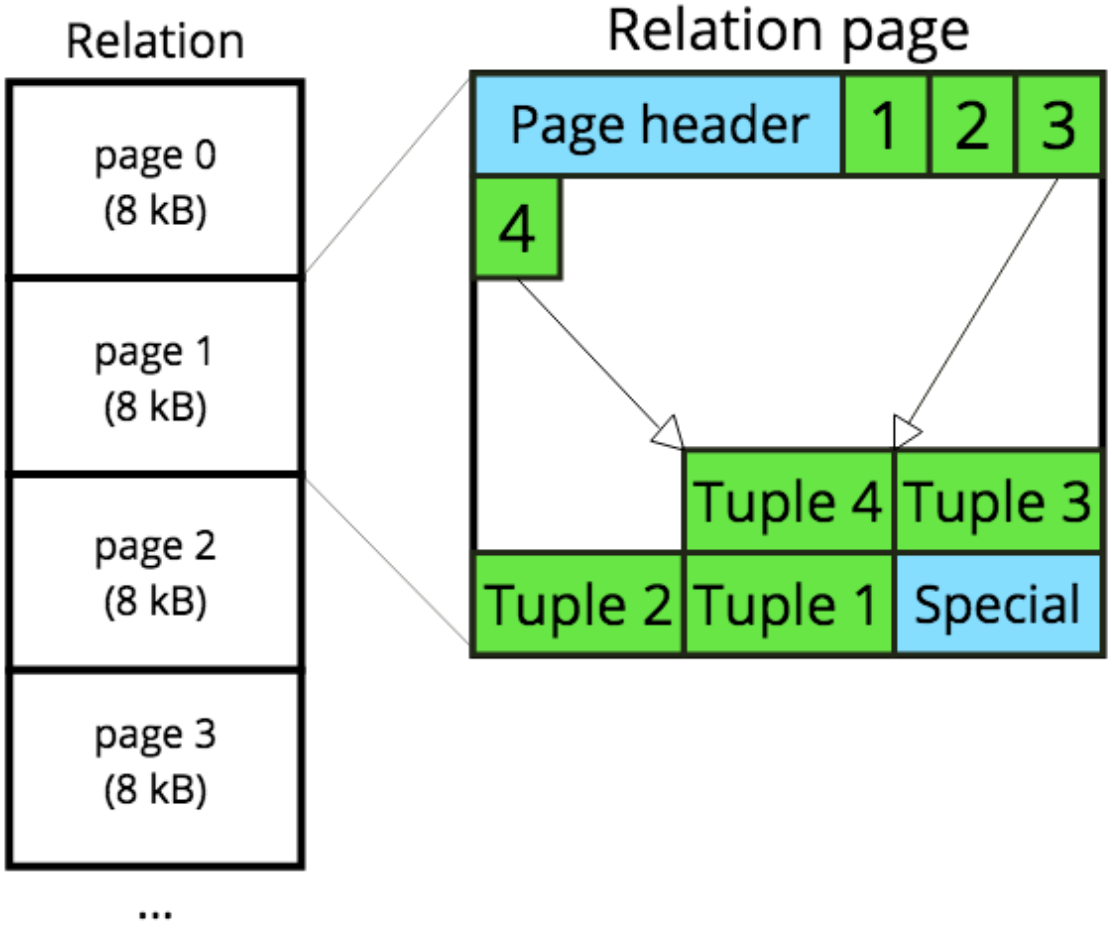
Vacuuming:

- **Cleans out dead rows** (VACUUM)
- **Keeps database functional** (FREEZE)
- **Updates info about relations** (ANALYZE)

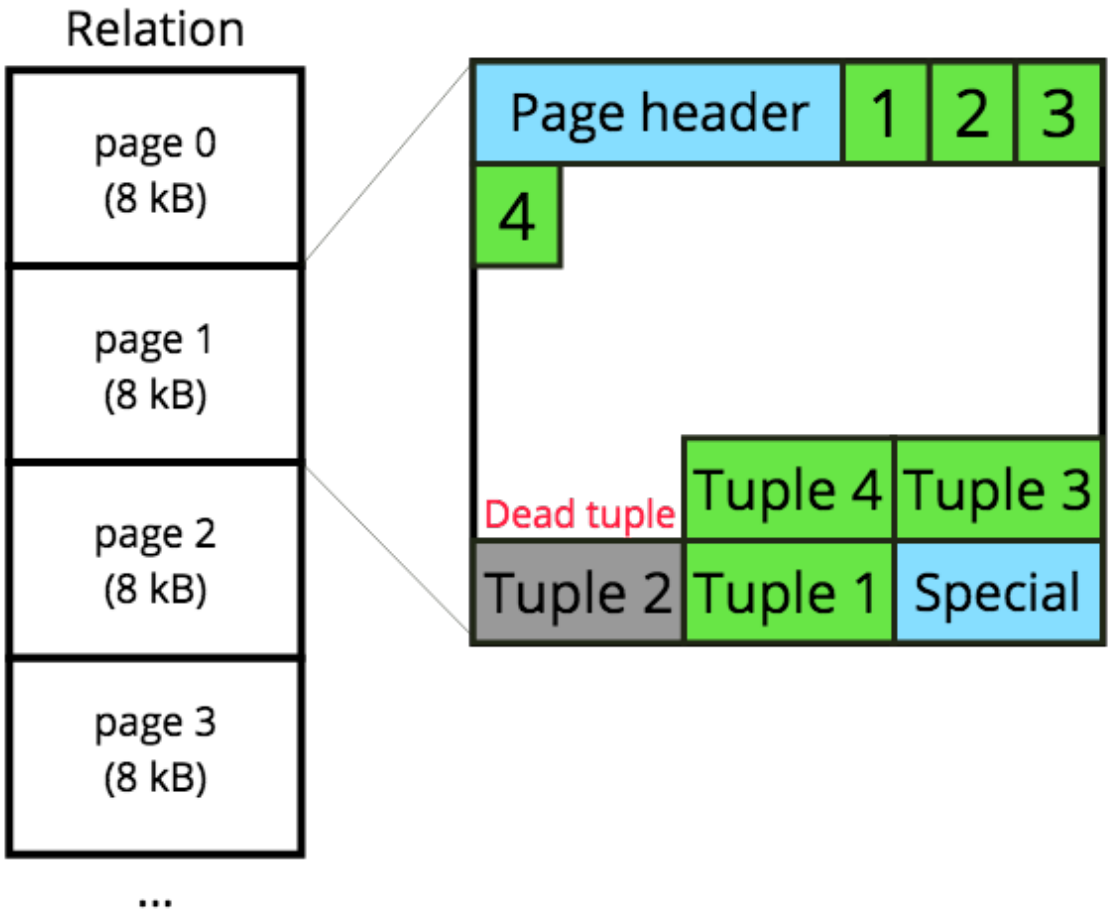
Autovacuum: makes vacuuming happen regularly

For more details: postgresql.org/docs/12/routine-vacuuming.html

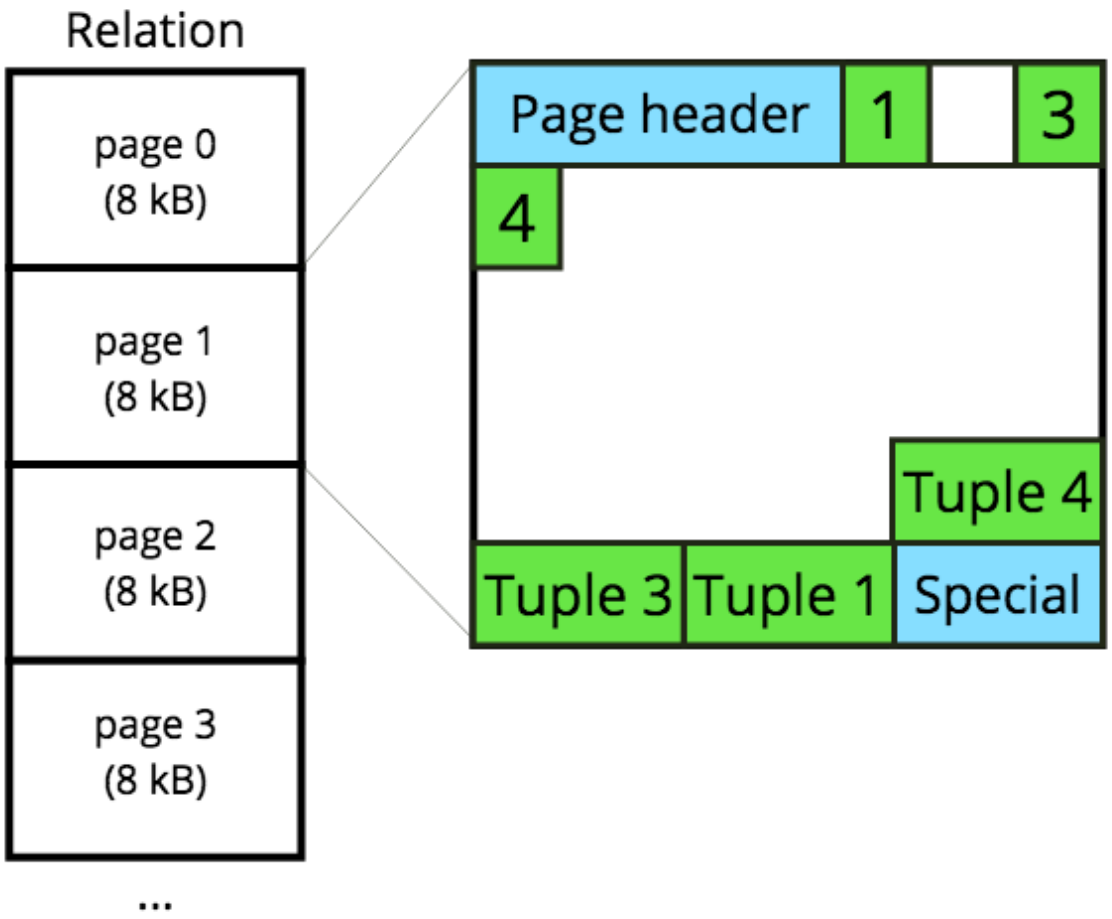
VACUUM cleanup



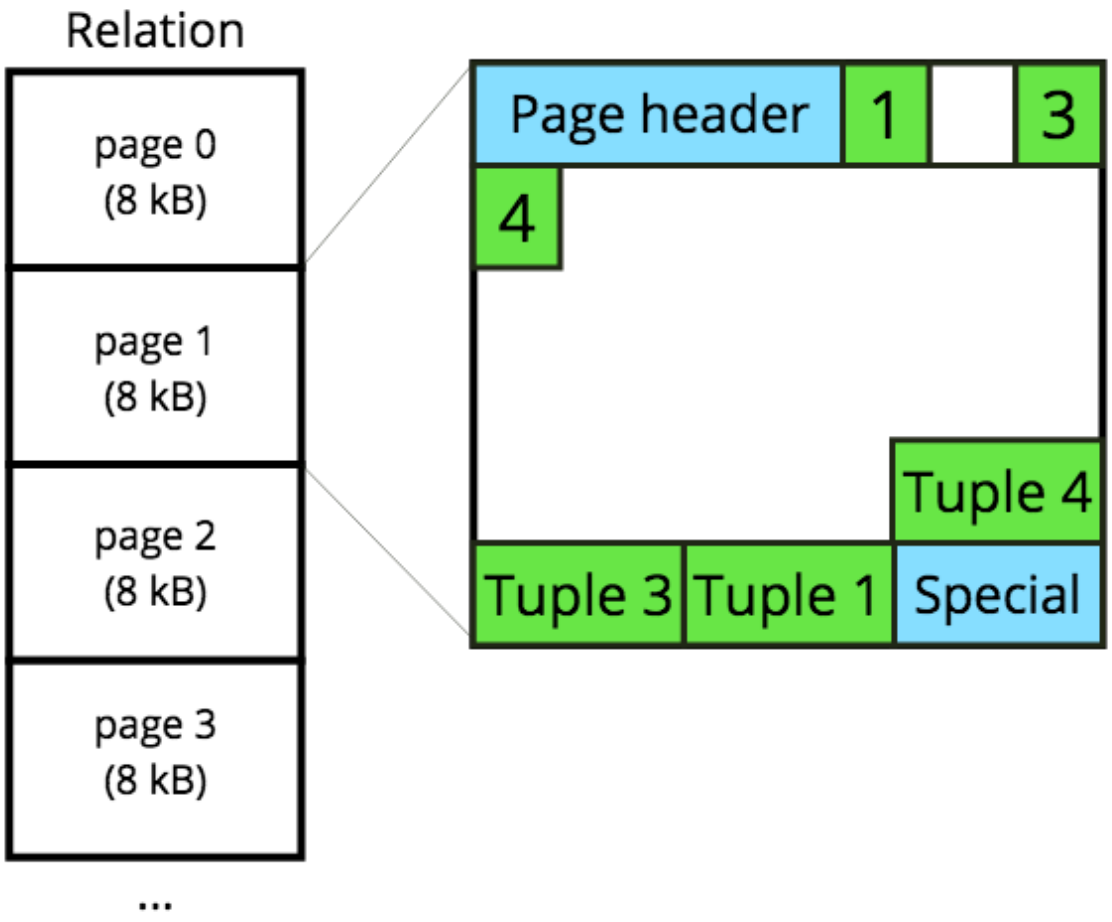
VACUUM cleanup



VACUUM cleanup

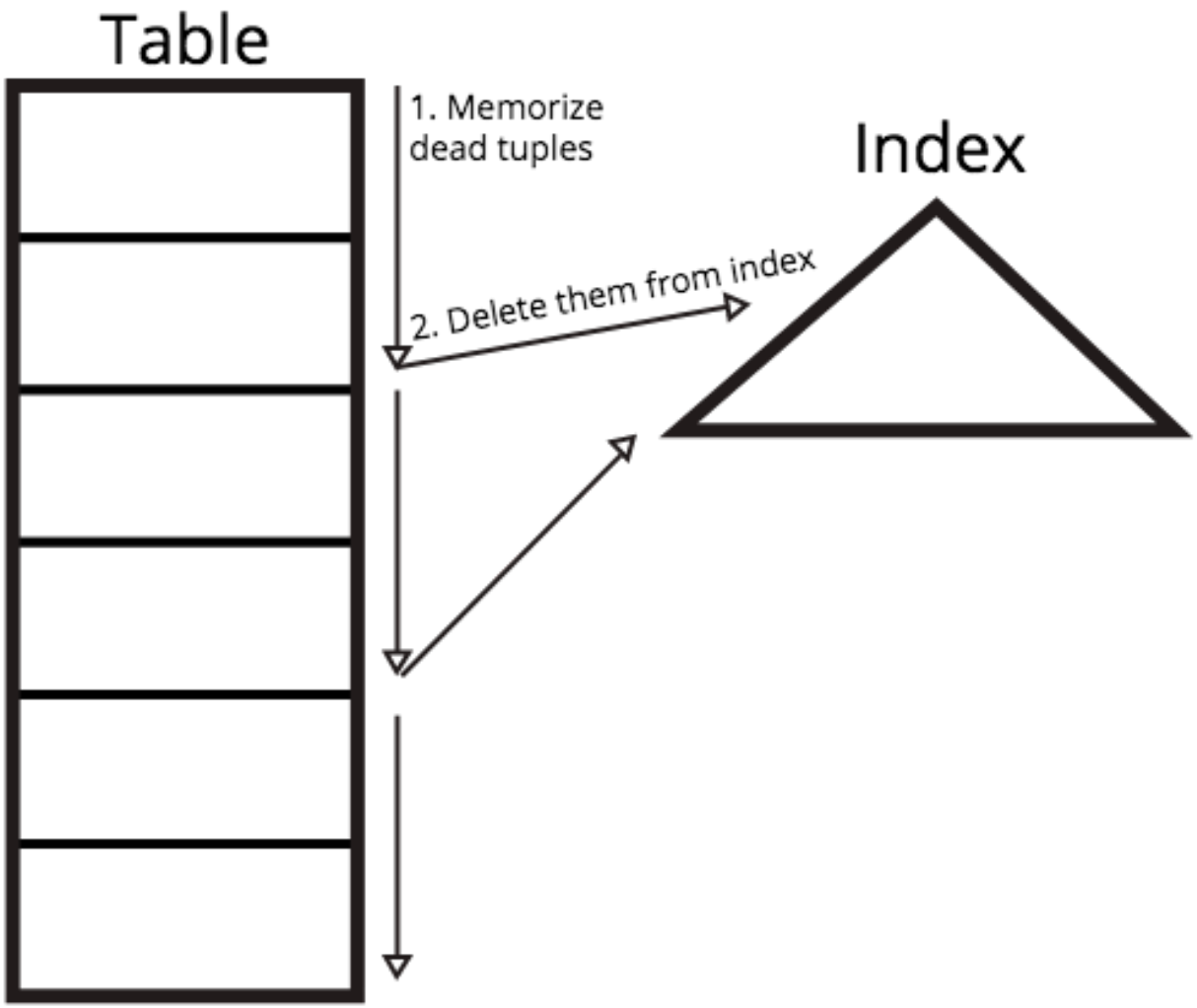


VACUUM cleanup



Dead tuples
get removed
from index too

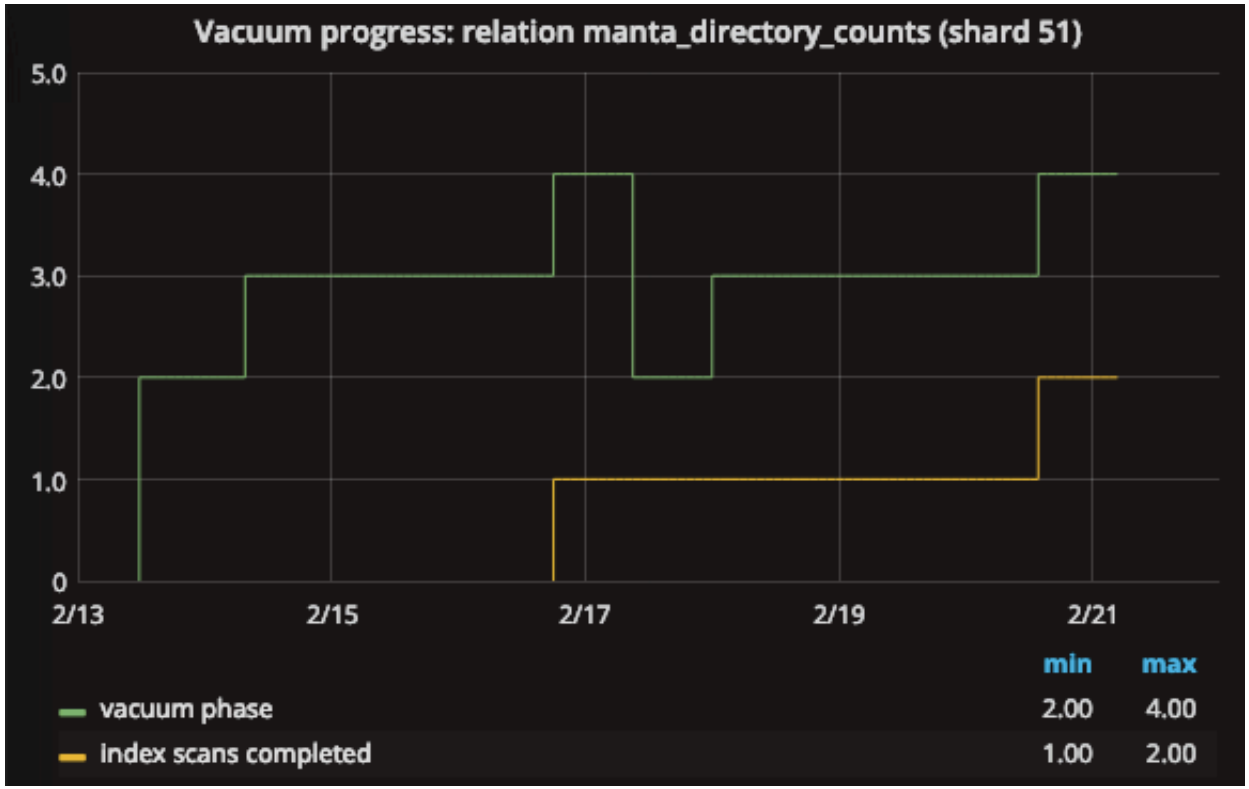
VACUUM cleanup



Index cleanup:

1) Scan heap
2) Vacuum index
3) Vacuum heap

VACUUM cleanup



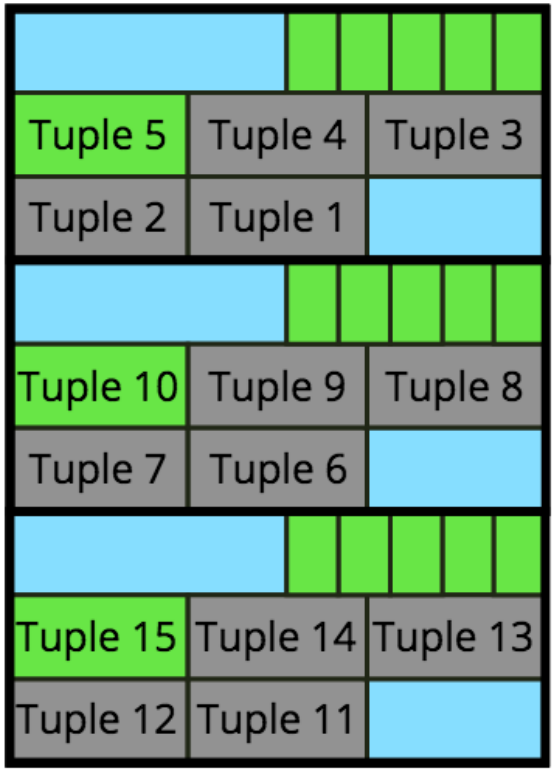
Index cleanup:

1) Scan heap
2) Vacuum index
3) Vacuum heap

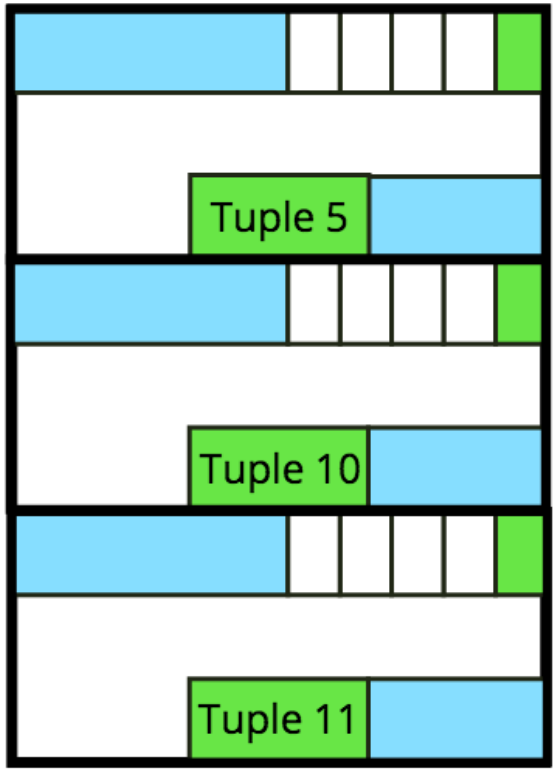
Image source: http://dtrace.org/blogs/dap/2019/05/22/visualizing-postgresql-vacuum-progress/

VACUUM FULL cleanup

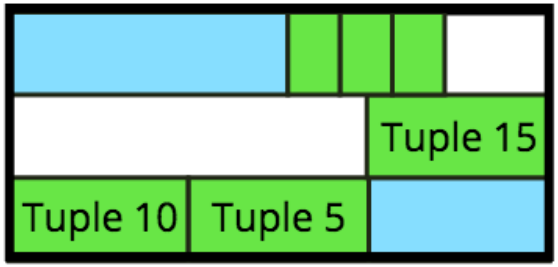
Before VACUUM



After VACUUM



After CLUSTER
or **VACUUM FULL**

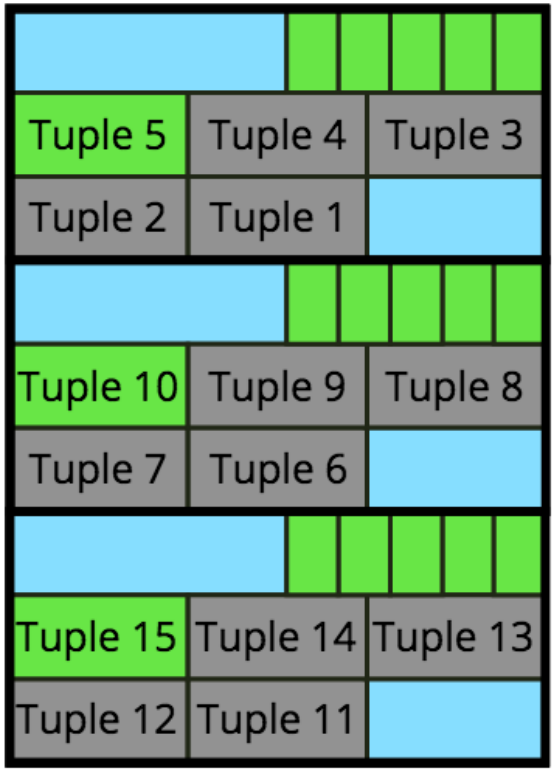


(index gets rebuilt)

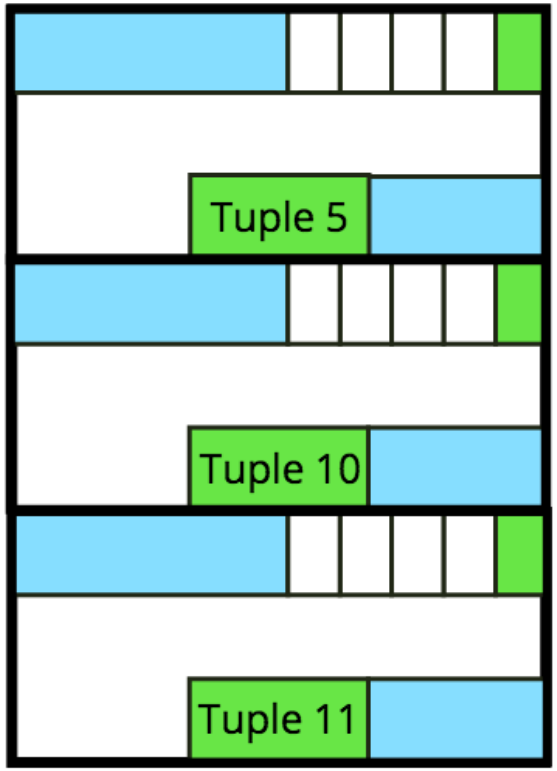
Can be only
run manually

VACUUM FULL cleanup

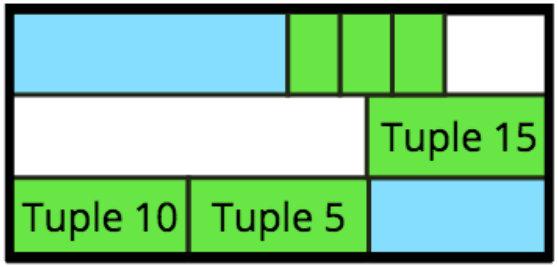
Before VACUUM



After VACUUM



After CLUSTER
or **VACUUM FULL**



(index gets rebuilt)

You can specify
the **fillfactor**

VACUUM and VACUUM FULL: summary

Postgres PROFESSIONAL

VACUUM:

- Makes space for new INSERTs
- Doesn't reduce relation size on disk (usually)

VACUUM FULL / CLUSTER:

- Reduces relation size on disk (usually)
- Can make space for new INSERTs (if fillfactor < 1)
- A heavier operation, can be only launched manually

VACUUM FREEZE
preventing XID wraparound

Each **transaction** is assigned an **ID** (XID).

A XID is a 32-bit number.

Without FREEZE, we'd run out of available XIDs.

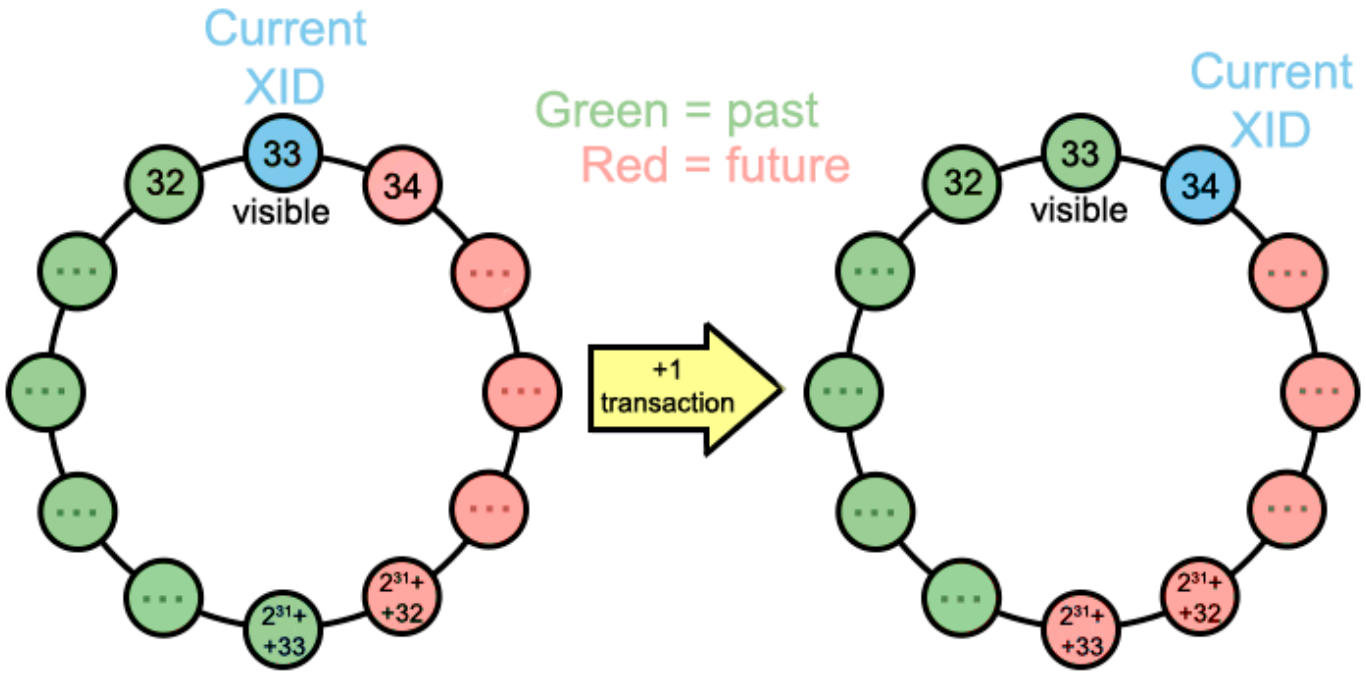
VACUUM FREEZE
preventing XID wraparound

PROFESSIONAL Postgres

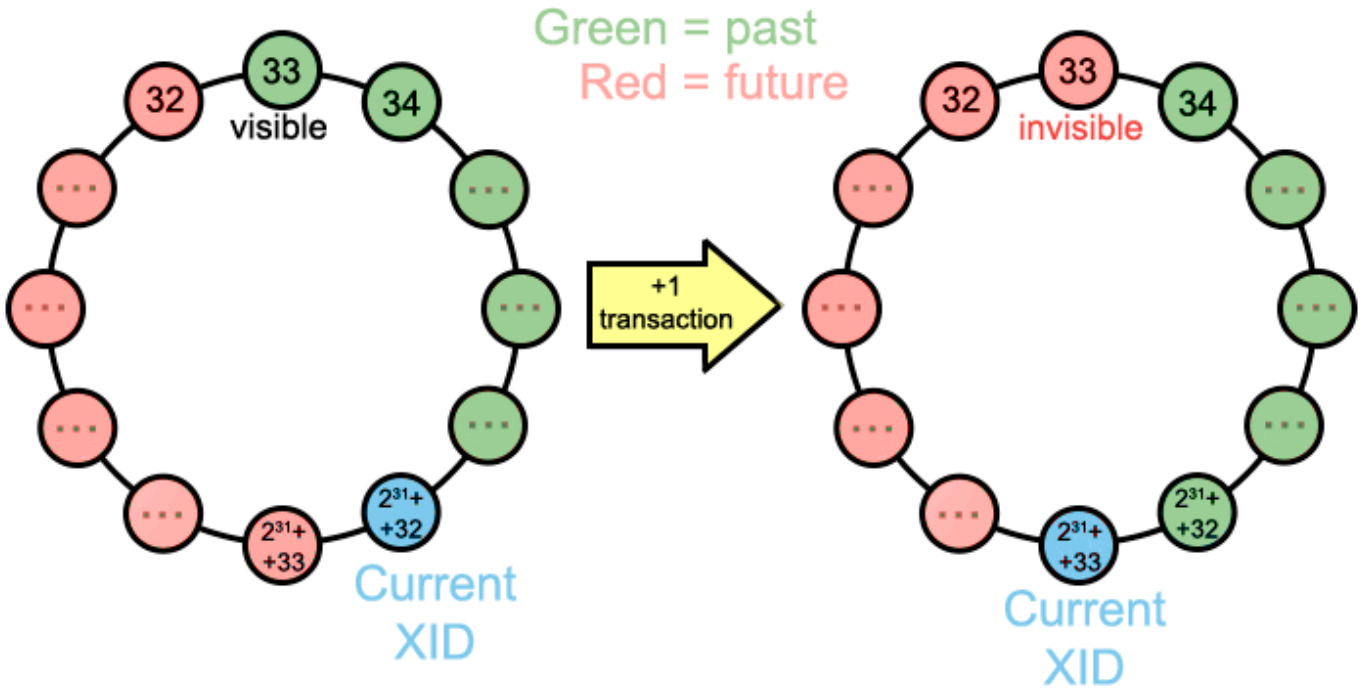
For each XID:

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

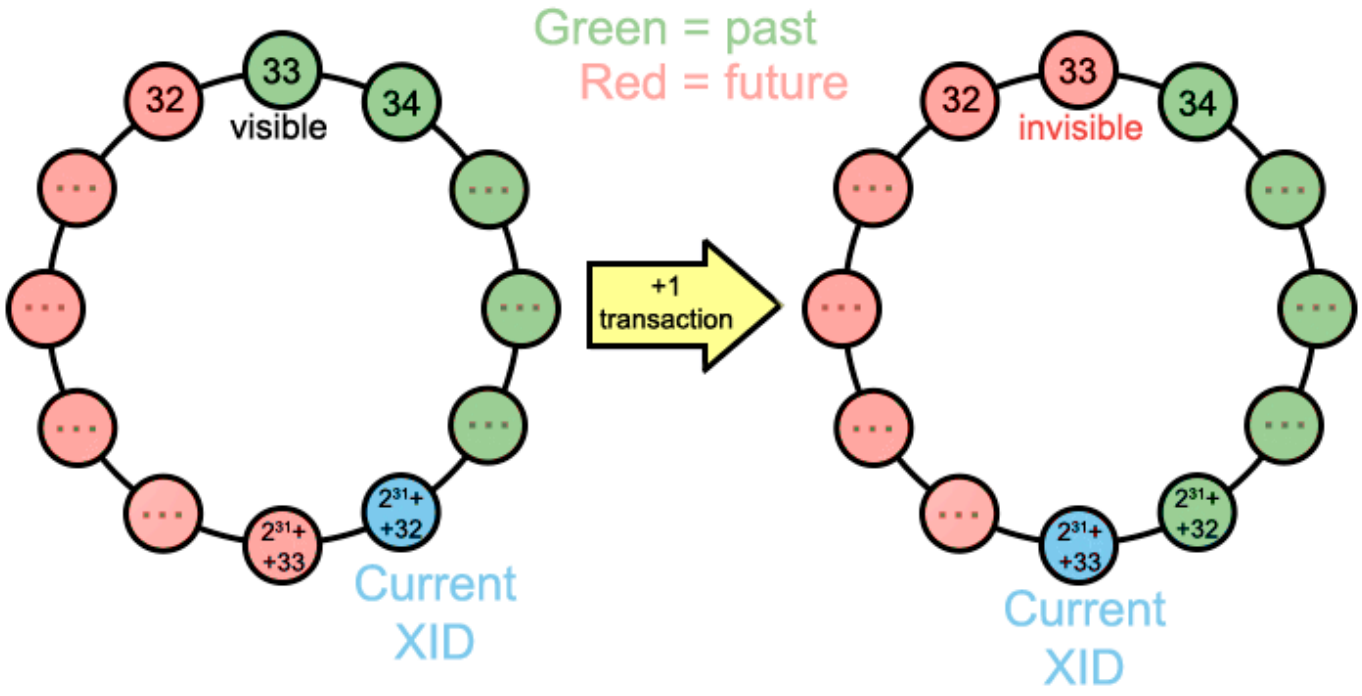
VACUUM FREEZE
preventing XID wraparound



VACUUM FREEZE
preventing XID wraparound



VACUUM FREEZE
preventing XID wraparound



To avoid this, we need to FREEZE old tuples.

VACUUM FREEZE: summary

VACUUM FREEZE:

- **Prevents XID wraparound**, for which it…
- **"Freezes" old tuples**
  that all running transactions can see
  (marks them as existing in the absolute past)

Runs when needed even if Autovacuum is disabled

VACUUM ANALYZE
updating info about relations

Information about relations that should be periodically collected: data statistics, visibility map.

They affect performance:

- **Data statistics:** used by the query planner
- **Visibility map (VM)** speeds up index-only scans

VACUUM ANALYZE: summary

VACUUM ANALYZE:

- **Updates visibility map (VM)**
- **Updates data statistics**

Updating statistics can be run separately (ANALYZE)

Conclusion

Vacuuming prevents problems:

- **Bloat** of tables and indexes
- **XID wraparound**
- **Performance degradation**

when it's launched by Autovacuum regularly enough.

2. Current issues and workarounds

Long-running transactions

PROFESSIONAL Postgres

A long transaction may prevent tuples from:

- Being cleaned out of the table
- Being frozen

=> try to avoid long-running transactions!

Temporary tables

Issues with temp tables:

- Autovacuum doesn't work with them
- A backend can only VACUUM its own temp tables
- Long sessions + temp tables => **wraparound**

How to avoid problems?

- Don't use temp tables for too long
- …or VACUUM them manually in your app

Perfomance issues

PROFESSIONAL Postgres

Too many index scans?

- Disable index cleanup, but use REINDEX later
- Increase amount of memory available to workers (autovacuum_work_mem, vacuum_work_mem)

VACUUMing uses too much memory?

- Decrease the number of workers
- Decrease the amount of memory available to workers

Visibility map not getting updated

Automatic VACUUM can only be triggered by UPDATEs/DELETEs. INSERTs trigger ANALYZE, which doesn't update the visibility map (VM).

This means:

- The VM doesn't get updated **after a big INSERT**
- **Append-only tables** rarely get visited by VACUUM (only to prevent wraparound)

Visibility map not getting updated

PROFESSIONAL Postgres

Consequences:

- VM isn't updated => **degradation of index-only scan**
- Possible **unexpected heavy loads** due to
  - Rare but massive wraparound-preventing VACUUM
  - SELECT setting hint bits after a big INSERT

Visibility map not getting updated

Workarounds:

Calling VACUUM or VACUUM FREEZE manually

- After big inserts
- Periodically for append-only tables

VACCUM will update the visibility map,
FREEZE will help lessen the amount of Autovacuum's work

Visibility map not getting updated

Keep in mind:
VACUUM FULL / CLUSTER don’t create a VM

=> you might want to run VACUUM [ANALYZE] after them to create a VM [and update statistics]

Getting stuck on big relations

PROFESSIONAL Postgres

What is the problem?

- **1 table = 1 autovacuum worker**
  => slow processing of big tables (especially with indexes)
- Vacuuming can be cancelled or interrupted
- It starts from the beginning of the relation each time
- User can’t control the relation order for Autovacuum

Getting stuck on big relations

Which means, big relations might:

- end up never getting fully processed
- block access to other relations

Getting stuck on big relations

Workarounds:

- Reduce bloat by using VACUUM FULL or analogues
- See if you can configure Autovaccum better
- Think of table partitioning
  (https://www.enterprisedb.com/fr/blog/containing-bloat-partitions)

Taking locks

VACUUM FULL/CLUSTER locks the whole relation.

Workarounds:

- Use alternatives (pg_repack, pgcompacttable)
- See if you can prevent needing VACUUM FULL by:
  - Avoiding long-running transactions
  - Configuring Autovaccum better
  - Using table partitioning

3. Future prospects

Block level parallel VACUUM

Author: Masahiko Sawada

Link to discussion: commitfest.postgresql.org/25/1774/

More details: pgcon.org/2018/schedule/events/1202.en.html

Issue: vacuuming takes long, especially on big tables and tables with many indexes.

Proposed fix: let multiple processes vacuum one table. It will speed up vacuuming, but consume more I/O and CPU.

Block level parallel VACUUM

PROFESSIONAL Postgres

Author: Masahiko Sawada

Link to discussion: commitfest.postgresql.org/25/1774/

More details: pgcon.org/2018/schedule/events/1202.en.html

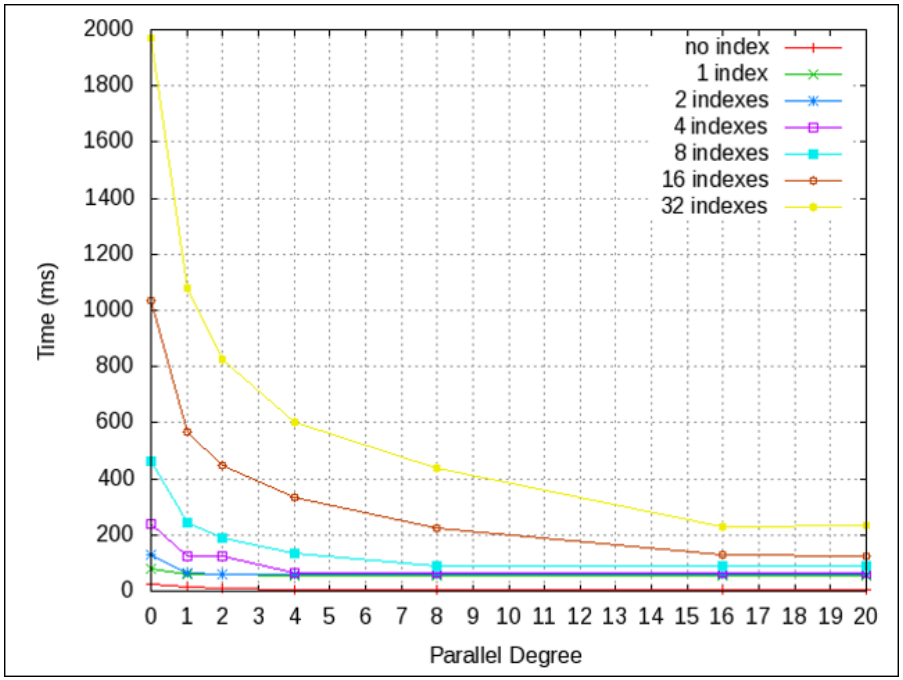


Image by Masahiko Sawada (from the discussion)

Issue: vacuuming takes long, especially on big tables and tables with many indexes.

Proposed fix: let multiple processes vacuum one table. It will speed up vacuuming, but consume more I/O and CPU.

Trigger autovacuum on tuple insertion

Author: Darafei Praliaskouski

Link to discussion: commitfest.postgresql.org/25/2093/

Issue:
For append-only tables, VACUUM is invoked
only when the table gets close to a wraparound.
=> their visibility map gets updated too rarely.

Proposed fix:
Invoke VACUUM based on inserts, not only deletes / updates.
Another option: update visibility map as a separate operation.

Resume [auto]vacuum from interruption and cancellation

Author: Masahiko Sawada

Link to discussion: commitfest.postgresql.org/25/2211/

Issue:
long-running vacuum/autovacuum can be cancelled/interrupted.
Starting from the beginning of the table each time,
vacuum might not ever reach the end of the table.

Proposed fix:
Teach vacuum to start on the block it previously ended on.

Write visibility map during CLUSTER/VACUUM FULL

PROFESSIONAL Postgres

Author: Alexander Korotkov

Link to discussion: commitfest.postgresql.org/25/2273/

Issue:
After CLUSTER / VACUUM FULL, index-only scan can suffer due to visibility map not being automatically created.

Proposed fix:
force CLUSTER and VACUUM FULL to create a visibility map.

Remove size limitations of vacuums dead_tuples array

Author: Ants Aasma

Link to discussion: commitfest.postgresql.org/25/2302/

Issue:
Now maintenance_work_mem has an upper limit of 1GB. Vacuuming large tables may require too many index scans due to this limit, even if there's plenty of memory available.

Proposed fix:
Raise the upper limit of maintenance_work_mem.

What about new storage types?

PROFESSIONAL Postgres

Zheap:

- In-place updates when possible
- Uses UNDO log

Zedstore: a column-oriented storage

The need for VACUUM will likely be minimised for them

Summary

Summary

Hopefully now you know:

- Why vacuuming is needed
- What issues you might run into
- What to look forward to in newer versions of PostgreSQL

…and are motivated to learn more!

Links

- Documentation:
  https://www.postgresql.org/docs/12/routine-vacuuming.html
- Visualisation of VACUUM progress:
  http://dtrace.org/blogs/dap/2019/05/22/visualizing-postgresql-vacuum-progress/
- Details on how VACUUM works:
  http://www.interdb.jp/pg/pgsql06.html
- Tuning autovacuum:
  https://www.2ndquadrant.com/en/blog/autovacuum-tuning-basics/
- Table partitioning:
  https://www.enterprisedb.com/fr/blog/containing-bloat-partitions
- Monitoring and configuring autovacuum
  https://pgconf.ru/en/2018/108354

Thank you!

Akenteva Anna

akenteva.annie@gmail.com

a.akenteva@postgrespro.ru