



VACUUM: From your head down to your shoes (©Buddy Guy)

Devrim Gündüz
Postgres Expert @ EDB

22 Sep 2026
Illinois Prairie PostgreSQL User Group

Self introduction

- PostgreSQL Major Contributor
- Responsible for the PostgreSQL RPM repos (Red Hat, Rocky, AlmaLinux, Fedora and SLES)
- Fedora and Rocky Linux contributor
- PostgreSQL community member
- Postgres expert @ EDB
- London, UK.

Nowadays:



Agenda

- MVCC: The basics
- Data snapshots
- Visibility
- VACUUM
- VACUUM processing
- FREEZE
- WAL
- Replication
- Performance
- AUTOVACUUM
- VACUUM tuning
- VACUUM FULL
- REPACK (its past and future)

“✱”

“✖”

- Basic question first ;)
- What does ✖ sign represent in **SELECT ✖ FROM t1;**

What is MVCC?

MVCC

- **Multi Version Concurrency Control**
 - Implementation of concurrency in Postgres
 - Snapshot isolation

MVCC

- **Multi Version Concurrency Control**
 - Implementation of concurrency in Postgres
 - Snapshot isolation
- “Readers do not block writers, writers do not block readers”.

MVCC

- **Multi Version Concurrency Control**
 - Implementation of concurrency in Postgres
 - Snapshot isolation
- **“Readers do not block writers, writers do not block readers”.**
- **Multiple versions of the same row may occur**
 - New versions are created during updates
 - Uncommitted transactions
 - Dead tuples (see next slides)

MVCC

- **Multi Version Concurrency Control**
 - Implementation of concurrency in Postgres
 - Snapshot isolation
- “Readers do not block writers, writers do not block readers”.
- **Multiple versions of the same row may occur**
 - New versions are created during updates
 - Uncommitted transactions
 - Dead tuples (see next slides)
- **Side effect: VACUUM**
 - We will get there ;)

Transaction id

- “txid”

Transaction id

- “txid”
- Unique identifier
 - 32-bits, ~ 4 billion
 - 64-bits txid is under review

Transaction id

- “txid”
- Unique identifier
 - 32-bits, ~ 4 billion
 - 64-bits txid is under review
 - “Circle”
 - 2 billion in the past, 2 billion in the future



Transaction id

- “txid”
- Unique identifier
 - 32-bits, ~ 4 billion
 - 64-bits txid is under review
 - “Circle”
 - 2 billion in the past, 2 billion in the future
 - 3 special (reserved) txids
 - 0: Invalid
 - 1: Bootstrap
 - **2: Frozen**

Transaction id

- **SELECT**
 - Utilizes “virtual txid”
 - `pg_current_xact_id_if_assigned()`

Transaction id

- **SELECT**
 - Utilizes “virtual txid”
 - `pg_current_xact_id_if_assigned()`
- **Stored in the header of each row**
 - xmin: INSERT
 - xmax: UPDATE or DELETE and also set by row locks (SELECT ... FOR UPDATE/SHARE)
 - (0, when this does not apply)

INSERT, DELETE and UPDATE

- **INSERT**

- Insertion is done to the first available space
 - xmin: set to the txid
 - xmax: 0

INSERT, DELETE and UPDATE

```
[postgres] # CREATE TABLE t1 (c1 int);
CREATE TABLE
[postgres] # INSERT INTO t1 VALUES (1),(2);
INSERT 0 2
[postgres] # INSERT INTO t1 VALUES (3);
INSERT 0 1
[postgres] # INSERT INTO t1 VALUES (4);
INSERT 0 1
[postgres] # SELECT cmin, cmax, xmin, xmax, ctid,* FROM t1;
```

cmin	cmax	xmin	xmax	ctid	c1
0	0	161031	0	(0,1)	1
0	0	161031	0	(0,2)	2
0	0	161032	0	(0,3)	3
0	0	161033	0	(0,4)	4

(4 rows)

INSERT, DELETE and UPDATE

- **DELETE**

- Logical deletion
- Long lasting transactions?
- xmax is set to the txid
- → **dead tuple: Only after the deleting transaction commits and no snapshot can still see it.**

INSERT, DELETE and UPDATE

First session:

```
[postgres] # BEGIN ;
BEGIN
[postgres] # DELETE FROM t1 WHERE c1=1;
DELETE 1
[postgres] # SELECT cmin, cmax, xmin, xmax, ctid,* FROM t1;
 cmin | cmax |  xmin  |  xmax  | ctid  | c1
-----+-----+-----+-----+-----+---
      0 |      0 | 161031 |      0 | (0,2) | 2
      0 |      0 | 161032 |      0 | (0,3) | 3
      0 |      0 | 161033 |      0 | (0,4) | 4
(3 rows)
```

INSERT, DELETE and UPDATE

Another session:

```
[postgres] # SELECT cmin, cmax, xmin, xmax, ctid,* FROM t1;
```

cmin	cmax	xmin	xmax	ctid	c1
0	0	161031	161034	(0,1)	1
0	0	161031	0	(0,2)	2
0	0	161032	0	(0,3)	3
0	0	161033	0	(0,4)	4

(4 rows)

INSERT, DELETE and UPDATE

- **UPDATE:**
 - “Expensive” operation
 - INSERT + DELETE
 - Dead tuple (as a part of deletion)

INSERT, DELETE and UPDATE

```
[postgres] # BEGIN ;  
BEGIN  
[postgres] # UPDATE t1 SET c1=20 WHERE c1=2;  
UPDATE 1  
[postgres] # SELECT cmin, cmax, xmin, xmax, ctid,* FROM t1;  
 cmin | cmax |  xmin  |  xmax  | ctid  | c1  
-----+-----+-----+-----+-----+-----  
    0 |    0 | 161032 |    0   | (0,3) | 3  
    0 |    0 | 161033 |    0   | (0,4) | 4  
    0 |    0 | 161035 |    0   | (0,5) | 20  
(3 rows)
```

INSERT, DELETE and UPDATE

Another session:

```
[postgres] # SELECT cmin, cmax, xmin, xmax, ctid,* FROM t1;
```

cmin	cmax	xmin	xmax	ctid	c1
0	0	161031	161035	(0,2)	2
0	0	161032	0	(0,3)	3
0	0	161033	0	(0,4)	4

(3 rows)

INSERT, DELETE and UPDATE

- Consider huge side effects of excessive DELETES (and UPDATES)

Combo CID, cmin, cmax

- pre-8.3: cmin and cmax were separate
- Per combocid.c:
 - To reduce the header size, cmin and cmax are now overlaid in the same field in the header. That usually works because you rarely insert and delete a tuple in the same transaction, and we don't need either field to remain valid after the originating transaction exits.

0		0		208611		0		(0,4)		4
0		0		208612		0		(0,5)		5
2		2		208612		0		(0,7)		7
4		4		208612		0		(0,8)		8

https://doxygen.postgresql.org/combocid_8c_source.html

Data snapshots

Data snapshots

- **Data snapshots**
 - Not a physical snapshot

Data snapshots

- Data snapshots

- Not a physical snapshot

- Isolation

- REPEATABLE READ / SERIALIZABLE:
 - one snapshot, taken at the transaction's first statement
 - READ COMMITTED (the default):
 - each statement gets its own snapshot.
 - Contains committed data
 - Uncommitted data is ignored.

Data snapshots

- Data snapshots
 - Not a physical snapshot
- Isolation
 - REPEATABLE READ / SERIALIZABLE:
 - one snapshot, taken at the transaction's first statement
 - READ COMMITTED (the default):
 - each statement gets its own snapshot.
 - Contains committed data
 - Uncommitted data is ignored.
- Also determines *VACUUM-able* rows or *non-VACUUM-able* rows

Data snapshots

- Long running transactions
 - pg_dump

Data snapshots

- Long running transactions
 - pg_dump
- Some parameters:
 - idle_in_transaction_session_timeout (disabled by default)
 - old_snapshot_threshold (removed as of PG17)
 - transaction_timeout (new in PG 17)

Visibility

Visibility

- **Tuple visibility**
 - xmin,xmax

Visibility

- **Tuple visibility**
 - xmin,xmax
- Only one version is available in a snapshot

Visibility

- Tuple visibility
 - xmin,xmax
- Only one version is available in a snapshot
- Visibility definition:
 - That row version is already committed “before the snapshot was taken”
 - Could be INSERT, or UPDATE
 - UPDATE waiting?

Visibility

- Tuple visibility
 - xmin,xmax
- Only one version is available in a snapshot
- Visibility definition:
 - That row version is already committed “before the snapshot was taken”
 - Could be INSERT, or UPDATE
 - UPDATE waiting?
- Tip: Commit time is not stored
 - unless track_commit_timestamp = on

Visibility

- **Tuple visibility**
 - xmin,xmax
- **Only one version is available in a snapshot**
- **Visibility definition:**
 - That row version is already committed “before the snapshot was taken”
 - Could be INSERT, or UPDATE
 - UPDATE waiting?
- **Tip: Commit time is not stored**
 - unless track_commit_timestamp = on
- **Tip: Rollback segments are not available in PostgreSQL**
 - No chance for seeing a past consistent state (lively).

VACUUM

VACUUM

- A must-do maintenance process for PostgreSQL

VACUUM

- A must-do maintenance process for PostgreSQL
- Cleaning up **no-more-needed** dead tuples

VACUUM

- A must-do maintenance process for PostgreSQL
- Cleaning up **no-more-needed** dead tuples
- Can run against:
 - A single table
 - A few tables
 - A database
 - A few databases (via vacuumdb)
 - All databases (via vacuumdb)

VACUUM

- A must-do maintenance process for PostgreSQL
- Cleaning up **no-more-needed** dead tuples
- Can run against:
 - A single table
 - A few tables
 - A database
 - A few databases (via vacuumdb)
 - All databases (via vacuumdb)
- Two main tasks:
 - Removing dead tuples
 - Freezing transaction ids

VACUUM

- Does **not** block most of the queries
 - Concurrent vacuums to the same table are not allowed
 - Cannot create index (concurrently or regular)
 - Cannot create trigger
 - Cannot refresh MV
 - Cannot add/remove columns from table
 - Cannot drop table ;)

VACUUM

- Does **not** block most of the queries
 - Concurrent vacuums to the same table are not allowed
 - Cannot create index (concurrently or regular)
 - Cannot create trigger
 - Cannot refresh MV
 - Cannot add/remove columns from table
 - Cannot drop table ;)
- I/O
 - Creates I/O (we will get there)

VACUUM tasks

- **Removes dead tuples**
 - Clean up dead tuples
 - Also cleans up index pages (pointing to the dead tuples)

VACUUM tasks

- **Removes dead tuples**
 - Clean up dead tuples
 - Also cleans up index pages (pointing to the dead tuples)
- **Freezing**
 - Freeze “old” transaction ids
 - Update some catalog tables

VACUUM tasks

- **Removes dead tuples**
 - Clean up dead tuples
 - Also cleans up index pages (pointing to the dead tuples)
- **Freezing**
 - Freeze “old” transaction ids
 - Update some catalog tables
- **Update VM and FSM**

VACUUM tasks

- **Removes dead tuples**
 - Clean up dead tuples
 - Also cleans up index pages (pointing to the dead tuples)
- **Freezing**
 - Freeze “old” transaction ids
 - Update some catalog tables
- **Update VM and FSM**
- **Update statistics (optional)**

VACUUM process

- VACUUMing is done per table, per page.

VACUUM process

- VACUUMing is done per table, per page.
 - Postgres doesn't allow multiple VACUUMs concurrently on a single relation
- Scan pages for dead tuples

VACUUM process

- VACUUMing is done per table, per page.
 - Postgres doesn't allow multiple VACUUMs concurrently on a single relation
- Scan pages for dead tuples
- Remove index entries pointing to the dead tuples

VACUUM process

- VACUUMing is done per table, per page.
 - Postgres doesn't allow multiple VACUUMs concurrently on a single relation
- Scan pages for dead tuples
- Remove index entries pointing to the dead tuples
- Update Visibility Map (VM) and Free Space Map (FSM)

VACUUM process

- **VACUUMing is done per table, per page.**
 - Postgres doesn't allow multiple VACUUMs concurrently on a single relation
- Scan pages for dead tuples
- Remove index entries pointing to the dead tuples
- Update Visibility Map (VM) and Free Space Map (FSM)
- Truncate last page(s) of the table
 - If the page is empty

VACUUM process

- **VACUUMing is done per table, per page.**
 - Postgres doesn't allow multiple VACUUMs concurrently on a single relation
- Scan pages for dead tuples
- Remove index entries pointing to the dead tuples
- Update Visibility Map (VM) and Free Space Map (FSM)
- Truncate last page(s) of the table
 - If the page is empty
- Update stats, update catalog tables

VACUUM: First phase

- Scan the table, and create list of the dead tuples.

VACUUM: First phase

- Scan the table, and create list of the dead tuples.
- Read data into the memory (I/O)
- Freeze tuples (we will get there)

VACUUM: First phase

- Scan the table, and create list of the dead tuples.
- Read data into the memory (I/O)
- Freeze tuples (we will get there)
- Cleanup of index tuples
(which point to the dead and removed tuples)

VACUUM: First phase

- Scan the table, and create list of the dead tuples.
- Read data into the memory (I/O)
- Freeze tuples (we will get there)
- Cleanup of index tuples
(which point to the dead and removed tuples)
- NOTE: Dead tuple cleanup is not done at this phase.

VACUUM: First phase

- **Some parameters:**
 - `maintenance_work_mem`
 - Can also be set per-session
 - v17+ stores dead tuples more efficiently

VACUUM: Second phase

- Removal of dead tuples

VACUUM: Second phase

- Removal of dead tuples
- FSM and VM are updated (per page)

VACUUM: Second phase

- Removal of dead tuples
- FSM and VM are updated (per page)
- Repairs fragmentation (per page)

VACUUM: Third phase

- Final phase

VACUUM: Third phase

- Final phase
- Index cleanup

VACUUM: Third phase

- Final phase
- Index cleanup
- Updates stats and system catalogs (per table)

VACUUM: Third phase

- Final phase
- Index cleanup
- Updates stats and system catalogs (per table)
- Truncation (if applicable)

VACUUM: Ring buffers

- **Buffer Access Strategy**

- saves shared_buffers
- Uses them circularly
- PG 16 added the ability to set its size (BUFFER_USAGE_LIMIT / vacuum_buffer_usage_limit).
- <https://www.postgresql.org/docs/current/glossary.html#GLOSSARY-BUFFER-ACCESS-STRATEGY>

- Helps keep shared buffers “hot”

VACUUM: FREEZE

- “Transaction ID wraparound problem”
 - Time to recall “circle”
 - A must-avoid problem

VACUUM: FREEZE

- “Transaction ID wraparound problem”
 - Time to recall “circle”
 - A must-avoid problem
- FREEZE
 - frozen txid

VACUUM: FREEZE

- “Transaction ID wraparound problem”
 - Time to recall “circle”
 - A must-avoid problem
- FREEZE
 - frozen txid
- An aggressive vacuum skips pages the visibility map marks as all-frozen.

VACUUM: FREEZE

- “Transaction ID wraparound problem”
 - Time to recall “circle”
 - A must-avoid problem
- FREEZE
 - frozen txid
- An aggressive vacuum skips pages the visibility map marks as all-frozen.
- Specially reserved txid: 2
 - freezing sets a flag bit and keeps the original xmin, instead of overwriting it with 2.
 - “Always older than other transaction IDs”
 - “Always visible”

VACUUM: FREEZE

- “Transaction ID wraparound problem”
 - Time to recall “circle”
 - A must-avoid problem
- FREEZE
 - frozen txid
- An aggressive vacuum skips pages the visibility map marks as all-frozen.
- Specially reserved txid: 2
 - freezing sets a flag bit and keeps the original xmin, instead of overwriting it with 2.
 - “Always older than other transaction IDs”
 - “Always visible”
- vacuum_freeze_min_age

VACUUM: FREEZE

- “Limited number ($N = 2^{32}$) of XID's required to do vacuum freeze to prevent wraparound every $N/2$ transactions.”
- This causes performance degradation due to the need to read and rewrite all not yet frozen pages tables while being vacuumed.

(Extracted from 64-bit xid patch)

VACUUM parameters

VACUUM parameters

- FREEZE [boolean]
- VERBOSE [boolean]
- ANALYZE [boolean]
- DISABLE_PAGE_SKIPPING [boolean]
- SKIP_LOCKED [boolean]
- INDEX_CLEANUP { AUTO | ON | OFF }
- PROCESS_MAIN [boolean]
- PROCESS_TOAST [boolean]
- TRUNCATE [boolean]
- PARALLEL integer
- SKIP_DATABASE_STATS [boolean]
- ONLY_DATABASE_STATS [boolean]
- BUFFER_USAGE_LIMIT size
- FULL [boolean]

VACUUM parameters

- TRUNCATE [boolean]
- PARALLEL integer
- v16+:
- SKIP_DATABASE_STATS [boolean]
 - skip updating the database-wide statistics about oldest unfrozen XIDs
- ONLY_DATABASE_STATS [boolean]
 - Just update database statistics
- BUFFER_USAGE_LIMIT size
 - vacuum_buffer_usage_limit
 - Max 16 GB

VACUUM and WAL

WAL

- Logging of transactions

WAL

- Logging of transactions
- All “modifications” are logged

WAL

- Logging of transactions
- All “modifications” are logged
- VACUUM -> page modifications -> WAL
 - Crash recovery
 - Also required for replica servers

WAL

- Logging of transactions
- All “modifications” are logged
- VACUUM -> page modifications -> WAL
 - Crash recovery
 - Also required for replica servers
- So, VACUUM causes extra I/O pressure on WAL

WAL

- Logging of transactions
- All “modifications” are logged
- VACUUM -> page modifications -> WAL
 - Crash recovery
 - Also required for replica servers
- So, VACUUM causes extra I/O pressure on WAL
 - backups!

WAL

- Logging of transactions
- All “modifications” are logged
- VACUUM -> page modifications -> WAL
 - Crash recovery
 - Also required for replica servers
- So, VACUUM causes extra I/O pressure on WAL
 - backups!
- Full Page Writes
 - tuning

VACUUM and replication

VACUUM and replication

- Long running (SELECT) queries on standby

VACUUM and replication

- Long running (SELECT) queries on standby
- Row is / rows are modified on primary

VACUUM and replication

- Long running (SELECT) queries on standby
- Row is / rows are modified on primary
- VACUUM kicks in

VACUUM and replication

- Long running (SELECT) queries on standby
- Row is / rows are modified on primary
- VACUUM kicks in
- Standby: “ERROR: canceling statement due to conflict with recovery”

VACUUM and replication

- Long running (SELECT) queries on standby
- Row is / rows are modified on primary
- VACUUM kicks in
- Standby: “ERROR: canceling statement due to conflict with recovery”
- Parameter: hot_standby_feedback

VACUUM and replication

- Long running (SELECT) queries on standby
- Row is / rows are modified on primary
- VACUUM kicks in
- Standby: “ERROR: canceling statement due to conflict with recovery”
- Parameter: hot_standby_feedback
- Side effect: VACUUMs will delay, bloat will increase.

VACUUM performance

VACUUM performance

- `vacuum_cost_delay` (0, disabled by default)

VACUUM performance

- `vacuum_cost_delay` (0, disabled by default)
- `vacuum_cost_page_hit` (1 by default)

VACUUM performance

- `vacuum_cost_delay` (0, disabled by default)
- `vacuum_cost_page_hit` (1 by default)
- `vacuum_cost_page_miss` (2 by default)

VACUUM performance

- `vacuum_cost_delay` (0, disabled by default)
- `vacuum_cost_page_hit` (1 by default)
- `vacuum_cost_page_miss` (2 by default)
- `vacuum_cost_page_dirty` (20 by default)

VACUUM performance

- `vacuum_cost_delay` (0, disabled by default)
- `vacuum_cost_page_hit` (1 by default)
- `vacuum_cost_page_miss` (2 by default)
- `vacuum_cost_page_dirty` (20 by default)
- `vacuum_cost_limit` (200 by default)

VACUUM performance

- Changing `vacuum_cost_delay` will result in less I/O over the time, but then VACUUM will take longer.

VACUUM performance

- Changing `vacuum_cost_delay` will result in less I/O over the time, but then VACUUM will take longer.
- This is the way to throttle VACUUM process.

Autovacuum

AUTOVACUUM

- Since PostgreSQL 8.1

AUTOVACUUM

- Since PostgreSQL 8.1
- Kicks off autovacuum/autoanalyze, per parameters.

AUTOVACUUM

- Since PostgreSQL 8.1
- Kicks off autovacuum/autoanalyze, per parameters.
- Kicks off to prevent transaction ID wraparound.

AUTOVACUUM

- Since PostgreSQL 8.1
- Kicks off autovacuum/autoanalyze, per parameters.
- Kicks off to prevent transaction ID wraparound.
- On by default.

AUTOVACUUM

- Since PostgreSQL 8.1
- Kicks off autovacuum/autoanalyze, per parameters.
- Kicks off to prevent transaction ID wraparound.
- On by default.
 - Do not turn it off!

AUTOVACUUM: Is everything cool?

- No.

AUTOVACUUM: Is everything cool?

- No.
- **Murphy's law: Autovacuum will kick off during peak hours**

AUTOVACUUM: Is everything cool?

- No.
- Murphy's law: Autovacuum will kick off during peak hours
- Cancels itself when a higher lock level is required by another transaction
 - Anti-wraparound vacuum: is not auto-cancelled on lock conflicts; if cancelled manually, it restarts.
 - Will start even if autovacuum is turned off.

AUTOVACUUM: Is everything cool?

- More workers -> more I/O

AUTOVACUUM: Is everything cool?

- More workers -> more I/O
- More workers -> more RAM usage (maintenance_work_mem)

AUTOVACUUM: parameters (in PG 18)

- `autovacuum = on`
- `autovacuum_analyze_scale_factor = 0.1`
- `autovacuum_analyze_threshold = 50`
- `autovacuum_freeze_max_age = 200000000`
- `autovacuum_max_workers = 3`
- `autovacuum_multixact_freeze_max_age = 400000000`
- `autovacuum_naptime = 60`
- `autovacuum_vacuum_cost_delay = 2`

AUTOVACUUM: parameters (in PG 18)

- `autovacuum_vacuum_cost_limit` = -1
- `autovacuum_vacuum_insert_scale_factor` = 0.2
- `autovacuum_vacuum_insert_threshold` = 1000
- `autovacuum_vacuum_max_threshold` = 100000000
- `autovacuum_vacuum_scale_factor` = 0.2
- `autovacuum_vacuum_threshold` = 50
- `autovacuum_work_mem` = -1
- `autovacuum_worker_slots` = 16
- `log_autovacuum_min_duration` = 600000

What is new in PG 19 for autovac?

<code>autovacuum_max_parallel_workers</code>	maximum number of parallel workers a single autovacuum worker can use.
<code>autovacuum_vacuum_score_weight</code>	dead tuples (the normal vacuum threshold)
<code>autovacuum_vacuum_insert_score_weight</code>	inserts (the insert threshold)
<code>autovacuum_analyze_score_weight</code>	analyze
<code>autovacuum_freeze_score_weight</code>	transaction ID wraparound (<code>autovacuum_freeze_max_age</code>)
<code>autovacuum_multixact_freeze_score_weight</code>	multixact wraparound

What is new in PG 19 for autovac?

- **Tables are now prioritised by a score**
 - Earlier versions process tables in catalog order, and prioritisation only arrives with the PG 19 score weights.

AUTOVACUUM: per table parameter, v19

- `autovacuum_parallel_workers` (default -1),
use `autovacuum_max_parallel_workers`

Autovacuum: Tuning per table

```
ALTER TABLE public.t1
```

```
SET (autovacuum_vacuum_scale_factor = 0.05,  
    autovacuum_parallel_workers = 2,      -- PG19  
    autovacuum_vacuum_threshold = 200000,  
    autovacuum_analyze_scale_factor = 0.1,  
    autovacuum_analyze_threshold = 200000);
```

- Can be used to customize autovac settings for some tables

VACUUM and autovacuum

VACUUM and autovacuum

- Can live together.

VACUUM and autovacuum

- Can live together.
- Tuning both of them will help overall performance.

VACUUM and autovacuum

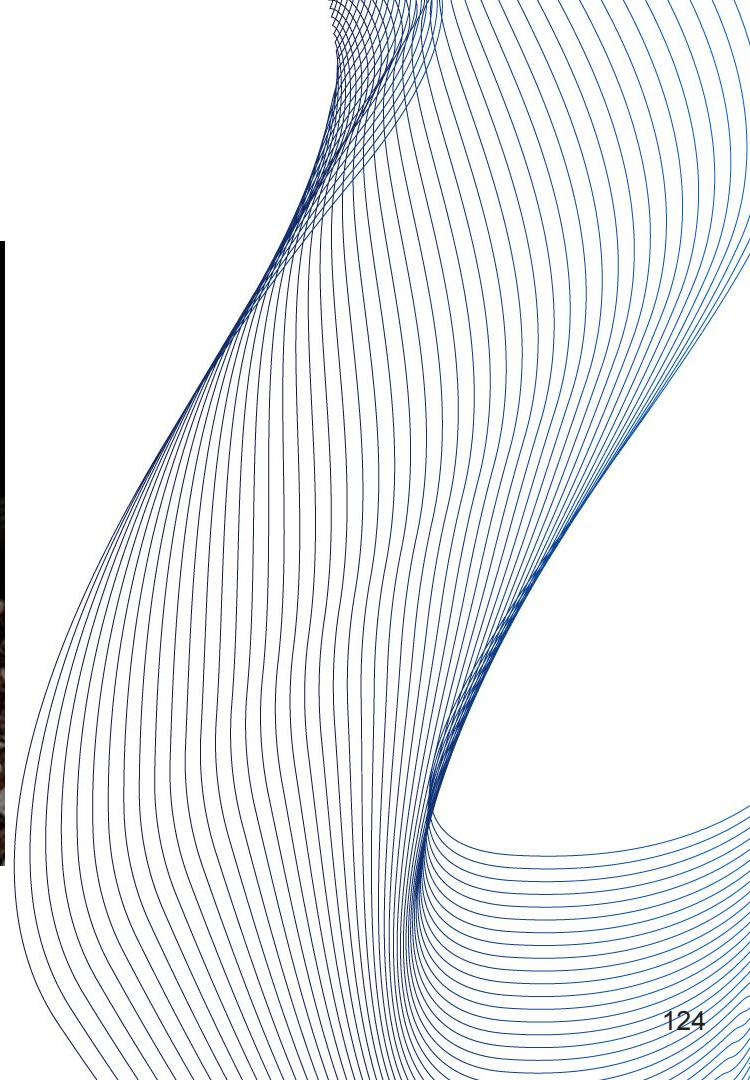
- Can live together.
- Tuning both of them will help overall performance.
- We suggest using cron-based VACUUM.

VACUUM and autovacuum

- Can live together.
- Tuning both of them will help overall performance.
- We suggest using cron-based VACUUM.
 - This will very likely prevent peak-time autovacuum accidents.

VACUUM FULL

VACUUM FULL



VACUUM FULL

- “Cut my life into pieces, this is my **last resort**”.

VACUUM FULL

- “Cut my life into pieces, this is my **last resort**”.
- Last resort.

VACUUM FULL

- “Cut my life into pieces, this is my **last resort**”.
- Last resort.
- Rewrites the table

VACUUM FULL

- “Cut my life into pieces, this is my **last resort**”.
- Last resort.
- Rewrites the table
- Requires ACCESS EXCLUSIVE LOCK
 - The only transaction that runs against the table

VACUUM FULL

- “Cut my life into pieces, this is my **last resort**”.
- Last resort.
- Rewrites the table
- Requires ACCESS EXCLUSIVE LOCK
 - The only transaction that runs against the table
- Requires disk space similar to the table size.

VACUUM FULL

- “Cut my life into pieces, this is my **last resort**”.
- Last resort.
- Rewrites the table
- Requires ACCESS EXCLUSIVE LOCK
 - The only transaction that runs against the table
- Requires disk space similar to the table size.
- Downtime!

VACUUM FULL: Non-blocking Alternative

- Some alternatives exist

VACUUM FULL: Non-blocking Alternative

- **Some alternatives exist**
 - `pg_repack`
 - `pg_squeeze` (cron!)

VACUUM VERBOSE

- VACUUM (VERBOSE) public.film;
- INFO: vacuuming "postgres.public.film"
- INFO: table "film": truncated 64 to 55 pages
- INFO: finished vacuuming "postgres.public.film": index scans: 0
- pages: 9 removed, 55 remain, 64 scanned (100.00% of total), 0 eagerly scanned
- tuples: 0 removed, 1000 remain, 0 are dead but not yet removable
- removable cutoff: 1207, which was 8 XIDs old when operation ended
- new relfrozenxid: 1037, which is 336 XIDs ahead of previous value
- frozen: 1 pages from table (1.56% of total) had 14 tuples frozen
- visibility map: 55 pages set all-visible, 1 pages set all-frozen (0 were all-visible)
- index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed
- index "film_fulltext_idx": pages: 10 in total, 0 newly deleted, 0 currently deleted, 0 reusable
- avg read rate: 0.000 MB/s, avg write rate: 246.322 MB/s
- buffer usage: 262 hits, 0 reads, 60 dirtied
- WAL usage: 63 records, 62 full page images, 497996 bytes, 493964 full page image bytes, 0 buffers full

VACUUM VERBOSE

- memory usage: dead item storage 0.02 MB accumulated across 0 resets (limit 64.00 MB each)
- system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
- INFO: vacuuming "postgres.pg_toast.pg_toast_16457"
- INFO: finished vacuuming "postgres.pg_toast.pg_toast_16457": index scans: 0
- pages: 0 removed, 0 remain, 0 scanned (100.00% of total), 0 eagerly scanned
- tuples: 0 removed, 0 remain, 0 are dead but not yet removable
- removable cutoff: 1207, which was 8 XIDs old when operation ended
- new relfrozenxid: 1207, which is 506 XIDs ahead of previous value
- frozen: 0 pages from table (100.00% of total) had 0 tuples frozen
- visibility map: 0 pages set all-visible, 0 pages set all-frozen (0 were all-visible)
- index scan not needed: 0 pages from table (100.00% of total) had 0 dead item identifiers removed
- avg read rate: 23.965 MB/s, avg write rate: 23.965 MB/s
- buffer usage: 29 hits, 1 reads, 1 dirtied
- WAL usage: 1 records, 1 full page images, 7515 bytes, 7396 full page image bytes, 0 buffers full
- memory usage: dead item storage 0.02 MB accumulated across 0 resets (limit 64.00 MB each)
- system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s

pg_stat_progress_vacuum (v19)

pid	18303
datid	19323
datname	foobar
relid	19870
phase	scanning heap
heap_blks_total	370044
heap_blks_scanned	13443
heap_blks_vacuumed	0
index_vacuum_count	0
max_dead_tuple_bytes	107682804
dead_tuple_bytes	3391
num_dead_item_ids	149101
indexes_total	0
indexes_processed	0
delay_time	0
mode	normal
started_by	manual

REPACK

Its past and future

REPACK

- PostgreSQL 19 feature
- Took ideas from pg_squeeze
- In-core feature

REPACK (and limitations)

- **Supersedes CLUSTER and VACUUM FULL (they still exist)**
 - VACUUM FULL is like REPACK without USING INDEX.
- **CONCURRENTLY**
 - only one REPACK CONCURRENTLY can run at a time across the whole cluster.
 - cannot be run inside a transaction block
 - only works on tables using the heap access method
 - is refused on user catalog tables
 - is refused when the replica identity index has been dropped
 - cannot be run against UNLOGGED tables, partitioned tables, TOAST or system catalogs, or tables without a primary key or index-based replica identity.
- **ANALYZE**
 - cannot be run inside a transaction block
 - works only on a single table that isn't partitioned

REPACK

- **MAINTAIN** privilege is needed.
- **max_repack_replication_slots**:
 - reserves slots for REPACK only; default 5; needs a restart to change
- Plain and concurrent REPACK both error out if the table has any invalid index
- **pg_stat_progress_repack** view:
 - **pg_stat_progress_cluster** still exists and is built on top of it



THANK YOU

Now it is time for questions!



VACUUM: From your head down to your shoes (©Buddy Guy)

Devrim Gündüz
Postgres Expert @ EDB

22 Sep 2026
Illinois Prairie PostgreSQL User Group