



Know the less known: A PostgreSQL Glossary

Devrim Gündüz
Postgres Expert @ EDB
Data Stack Conf 2026, Sofia

Self introduction

- PostgreSQL Major Contributor
- Responsible for PostgreSQL RPM repos (Red Hat, Rocky, AlmaLinux, Amazon Linux, Fedora and SLES)
- Fedora and EPEL contributor
- PostgreSQL community member
- Postgres expert @ EDB
- “The guy with the PostgreSQL tattoo”
- London, UK.

...and nowadays:

DJing!



DJing!



Agenda

- Motivation
- MVCC
- Glossary
- What's new in PG 18 and 19
- Bonus: hidden and read-only parameters

Motivation

Motivation

- Conferences and meetups: lots of great talks

Motivation

- Conferences and meetups: lots of great talks
- There are great tech talks

Motivation

- Conferences and meetups: lots of great talks
- There are great tech talks
- You are new-ish, or not used to some of the terms

Motivation

- Conferences and meetups: lots of great talks
- There are great tech talks
- You are new-ish, or not used to some of the terms
- So, welcome to this talk!

Motivation

- Conferences and meetups: lots of great talks
- There are great tech talks
- You are new-ish, or not used to some of the terms
- So, welcome to this talk!
- Resources: Source code, documentation, blog posts

“*”

“*”

- 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

Glossary

xact

- “Transaction”

Transaction id

- “txid”

Transaction id

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

Transaction id

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

ctid

- “The physical location of the row version within its table.”

ctid

- “The physical location of the row version within its table.”
- “block number” and “location of the tuple in the block”

ctid

- “The physical location of the row version within its table.”
- “block number” and “location of the tuple in the block”
- Do not depend on it

ctid

- “The physical location of the row version within its table.”
- “block number” and “location of the tuple in the block”
- Do not depend on it
- UPDATE or VACUUM FULL will change it

xmin

- “The identity (transaction ID) of the inserting transaction for this row version.”

xmax

- “The identity (transaction ID) of the deleting or updating transaction”

xmax

- “The identity (transaction ID) of the deleting or updating transaction”
 - or zero for an undeleted row version.

xmax

- “The identity (transaction ID) of the deleting or updating transaction”
 - or zero for an undeleted row version.
- May be non-zero in a visible row version:

xmax

- “The identity (transaction ID) of the deleting or updating transaction”
 - or zero for an undeleted row version.
- May be non-zero in a visible row version
 - Deleting transaction has not been committed *yet*

xmax

- “The identity (transaction ID) of the deleting or updating transaction”
- or zero for an undeleted row version.
- May be non-zero in a visible row version:
 - Deleting transaction has not been committed *yet*
 - Deleting transaction was rolled back
 - Row is locked (SELECT ... FOR UPDATE/SHARE)

cmin

- The command identifier (starting at zero) within the inserting transaction.

cmax

- The command identifier within the deleting transaction

cmax

- The command identifier within the deleting transaction
 - or zero.

Tuple header

- Stored in the header of each row
 - xmin: txid of the INSERT
 - xmax: txid of the UPDATE or DELETE; also set by row locks (SELECT ... FOR UPDATE/SHARE)
 - (0 when this does not apply)
 - cmin and cmax share one field in the header (combo CID)

Back to txid

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

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
 - xmax is set to the txid of the deleting transaction
 - Dead tuple only after that transaction commits
 - ...and no snapshot needs the row version any more
 - Long-running transactions delay the cleanup

INSERT, DELETE and UPDATE

- Session one:

```
[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

- Session two:

```
[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

```
[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)

pg_xact

- “Transaction metadata logs”

pg_xact

- “Transaction metadata logs”
- Per docs: “Subdirectory containing transaction commit status data”

pg_xact

- “Transaction metadata logs”
- Per docs: “Subdirectory containing transaction commit status data”
- Formerly pg_clog

pg_xact

- “Transaction metadata logs”
- Per docs: “Subdirectory containing transaction commit status data”
- Formerly pg_clog (renamed in PG 10)
- Keeps growing if old XIDs are not frozen
- VACUUM truncates it (see datfrozenxid)

relfrozenxid

- Per docs: “All transaction IDs before this one have been replaced with a permanent (“frozen”) transaction ID in this table”

relfrozenxid

- Per docs: “All transaction IDs before this one have been replaced with a permanent (“frozen”) transaction ID in this table”
- Tracks vacuum needs to prevent txid wraparound and to allow shrinking of pg_xact
- Freezing sets a flag bit and keeps the original xmin; it does not overwrite it with 2

datfrozenxid

- All about VACUUM
- All transaction IDs before this one have been replaced with a permanent transaction ID in this database.

datfrozenxid

- All about VACUUM
- All transaction IDs before this one have been replaced with a permanent transaction ID in this database.
- Used to track whether the database needs to be vacuumed in order to prevent transaction ID wraparound or to allow pg_xact to be shrunk.

datfrozenxid

- All about VACUUM
- All transaction IDs before this one have been replaced with a permanent transaction ID in this database.
- Used to track whether the database needs to be vacuumed in order to prevent transaction ID wraparound or to allow pg_xact to be shrunk.
- It is the minimum of the per-table pg_class.relFrozenxid values

datfrozenxid

- `SELECT datname, age(datfrozenxid) FROM pg_database;`

multixact

- Used to support row locking by multiple transactions

multixact

- Used to support row locking by multiple transactions
- Tuple headers: 23 bytes (24 with padding)
 - Space is limited

multixact

- Used to support row locking by multiple transactions
- Tuple headers: 23 bytes (24 with padding)
 - Space is limited
- Lock information is stored in “multixact ID” (multiple transaction id) (remember: xact = transaction)

multixact

- Used to support row locking by multiple transactions
- Tuple headers: 23 bytes (24 with padding)
 - Space is limited
- Lock information is stored in “multixact ID” (multiple transaction id) (remember: xact = transaction)
- Concurrent locking of a row

multixact

- Used to support row locking by multiple transactions
- Tuple headers: 23 bytes (24 with padding)
 - Space is limited
- Lock information is stored in “multixact ID” (multiple transaction id) (remember: xact = transaction)
- Concurrent locking of a row
- pg_multixact

multixact ID

- Implemented as 32-bit counter, very much like txid
- PG 19: multixact members are 64-bit
- \$PGDATA/pg_multixact/members: members of each multixact
- \$PGDATA/pg_multixact/offsets: where each multixact starts
- relminmxid / datminmxid: like relfrozenxid / datfrozenxid
- VACUUM removes old files from members and offsets

WAL

WAL

- Write Ahead Log

WAL

- Write Ahead Log
- Logging of transactions

WAL

- Write Ahead Log
- Logging of transactions
- Designed to prevent data loss in most of the situations

WAL

- Write Ahead Log
- Logging of transactions
- Designed to prevent data loss in most of the situations
- OS crash, hardware failure, PostgreSQL crash.

WAL

- Write Ahead Log
- Logging of transactions
- Designed to prevent data loss in most of the situations
- OS crash, hardware failure, PostgreSQL crash.
- Built-in feature

WAL

- Crash recovery (REDO)

WAL

- Crash recovery (REDO)
- Replication

WAL

- Crash recovery (REDO)
- Replication
- PITR

WAL

- Crash recovery (REDO)
- Replication
- PITR
- Sequential availability is a must: no gaps

WAL

- Crash recovery (REDO)
- Replication
- PITR
- Sequential availability is a must: no gaps
- REDO only, no UNDO: old row versions stay in the heap (MVCC)

WAL

- Crash recovery (REDO)
- Replication
- PITR
- Sequential availability is a must: no gaps
- REDO only, no UNDO: old row versions stay in the heap (MVCC)
- No WAL for temp tables and unlogged tables

LSN

LSN

- Log Sequence Number

LSN

- Log Sequence Number
- Byte position of a record in the WAL stream

LSN

- Log Sequence Number
- Byte position of a record in the WAL stream
- Provides uniqueness for each WAL record

LSN

- Log Sequence Number
- Byte position of a record in the WAL stream
- Provides uniqueness for each WAL record
- 64-bit integer, shown as two 32-bit hex halves (e.g. 0/16B3748)

LSN

- Log Sequence Number
- Byte position of a record in the WAL stream
- Provides uniqueness for each WAL record
- 64-bit integer, shown as two 32-bit hex halves (e.g. 0/16B3748)
- Per docs: “Pointer to a location in the WAL”

LSN

- Log Sequence Number
- Byte position of a record in the WAL stream
- Provides uniqueness for each WAL record
- 64-bit integer, shown as two 32-bit hex halves (e.g. 0/16B3748)
- Per docs: “Pointer to a location in the WAL”
- The WAL file name and the offset in it are derived from the LSN

LSN

- Log Sequence Number
- Byte position of a record in the WAL stream
- Provides uniqueness for each WAL record
- 64-bit integer, shown as two 32-bit hex halves (e.g. 0/16B3748)
- Per docs: “Pointer to a location in the WAL”
- The WAL file name and the offset in it are derived from the LSN
- Each data page stores the LSN of the last WAL record that changed it

LSN

- Log Sequence Number
- Byte position of a record in the WAL stream
- Provides uniqueness for each WAL record
- 64-bit integer, shown as two 32-bit hex halves (e.g. 0/16B3748)
- Per docs: “Pointer to a location in the WAL”
- The WAL file name and the offset in it are derived from the LSN
- Each data page stores the LSN of the last WAL record that changed it
- Recovery replays a record only if page LSN < record LSN; otherwise it is skipped

What's new in PG 18 and **19**

PG 18: Freezing and checksums

- VACUUM may now freeze all-visible pages early (eager freezing)
- Tuned by vacuum_max_eager_freeze_failure_rate (default 0.03)
- Less work for the later aggressive vacuum
- initdb enables data checksums by default (--no-data-checksums)
- pg_upgrade needs matching checksum settings; it keeps optimizer statistics

PG 19: Multixact and wraparound

- Multixact members are 64-bit; xid itself is still 32-bit
- `pg_get_multixact_stats()` reports multixact activity
- Wraparound warnings start at 100 million remaining (was 40 million)
- Applies to both xid and multixid
- Reference: `autovacuum_freeze_max_age` = 200 million, `autovacuum_multixact_freeze_max_age` = 400 million
- `vacuum_failsafe_age` = 1.6 billion

PG 19: Autovacuum

- Tables are now prioritised by a score
- Earlier versions process tables in catalog order
- `autovacuum_vacuum_score_weight`, `..._vacuum_insert_`, `..._analyze_`, `..._freeze_`, `..._multixact_freeze_score_weight`
- `autovacuum_max_parallel_workers`: parallel workers per autovacuum worker
- Per table: `autovacuum_parallel_workers` (default -1)

PG 19: REPACK

- Supersedes CLUSTER and VACUUM FULL (both still exist)
- VACUUM FULL is like REPACK without USING INDEX
- Took ideas from pg_squeeze: now in core
- CONCURRENTLY: one at a time across the whole cluster
- Heap tables only, with a primary key or index-based replica identity
- max_repack_replication_slots (default 5, restart needed); MAINTAIN privilege
- Progress: pg_stat_progress_repack

PG 19: WAL and LSN

- WAIT FOR LSN '0/0306EE20': wait until that LSN is written, flushed or replayed
- Handy for read-your-writes on standbys
- wal_compression = on now uses the best available method (was pglz)
- effective_wal_level (read-only): logical decoding can switch on with wal_level = replica
- Full-page-image bytes in pg_stat_wal and EXPLAIN (ANALYZE, WAL)

PG 18 and 19: more changes

- PG 18: asynchronous I/O (io_method), pg_aios, WAL rows in pg_stat_io
- PG 18: idle_replication_slot_timeout
- PG 19: JIT is off by default (see the jit_* parameters)
- PG 19: max_locks_per_transaction default 128 (was 64)
- PG 19: default TOAST compression is lz4

Thank you! Now it is time for
questions

Bonus:postgresql.conf parameters

Wait, what?

“Hidden” parameters in postgresql.conf

- Mainly for developers
 - ...and/or advanced users

“Hidden” parameters in postgresql.conf

- Mainly for developers
 - ...and /or advanced users
 - ...or for DBAs who know what they are doing

“Hidden” parameters in postgresql.conf

- Mainly for developers
 - ...and /or advanced users
 - ...or for DBAs who know what they are doing
 - Not included in postgresql.conf

In short:

In short:



In short:



“Hidden” parameters in postgresql.conf

- allow_system_table_mods (boolean)
- ignore_checksum_failure (boolean)
- zero_damaged_pages (boolean)
- ignore_invalid_pages (boolean)
- ignore_system_indexes (boolean)

“Hidden” parameters in postgresql.conf

- post_auth_delay (integer)
- pre_auth_delay (integer)
- wal_consistency_checking (string)
- wal_debug (boolean)
- backtrace_functions (string)
- debug_deadlocks (boolean)
- log_btree_build_stats (boolean)

“Hidden” parameters in postgresql.conf

- trace_notify (boolean)
- trace_recovery_messages (enum)
- trace_sort (boolean)
- trace_locks (boolean)
- trace_lwlocks (boolean)
- trace_userlocks (boolean)
- trace_lock_oidmin (integer)
- trace_lock_table (integer)

“Hidden” parameters in postgresql.conf

- jit_debugging_support (boolean)
- jit_dump_bitcode (boolean)
- jit_expressions (boolean)
- jit_profiling_support (boolean)
- jit_tuple_deforming (boolean)

More “hidden” parameters

- debug_discard_caches (integer)
- debug_io_direct (string)
- debug_parallel_query (enum)
- debug_logical_replication_streaming (enum)
- remove_temp_files_after_crash (boolean)
- send_abort_for_crash, send_abort_for_kill (boolean)
- Read-only, PG 19: effective_wal_level

“Read-only” parameters in PostgreSQL

- data_checksums (boolean)
 - initdb: on by default since PG 18
- block_size (integer)
 - 8192 byte (8kB)
- debug_assertions (boolean)
 - off

“Read-only” parameters in PostgreSQL

- lc_*

“Read-only” parameters in PostgreSQL

- max_function_args (integer)
 - 100
- max_identifier_length (integer)
 - 63
 - (bytes, not characters)
- max_index_keys (integer)
 - 32
- segment_size (integer)
 - 1GB

“Read-only” parameters in PostgreSQL

- server_encoding (string)
 - initdb, UTF-8
- server_version (string)
 - 18.x
 - 19beta
- server_version_num (integer)
 - 180000+
 - 190000

“Read-only” parameters in PostgreSQL

- wal_block_size (integer)
 - 8192 byte
 - Not the same as block_size
- wal_segment_size (integer)
 - 16777216 (bytes)
 - → 16MB (default)



Know the less known: A PostgreSQL Glossary

Devrim Gündüz
Postgres Expert @ EDB
Data Stack Conf 2026, Sofia