Know the less known: A PostgreSQL Glossary

Devrim Gündüz
Postgres Expert @ EDB

PGConf.EU 2023

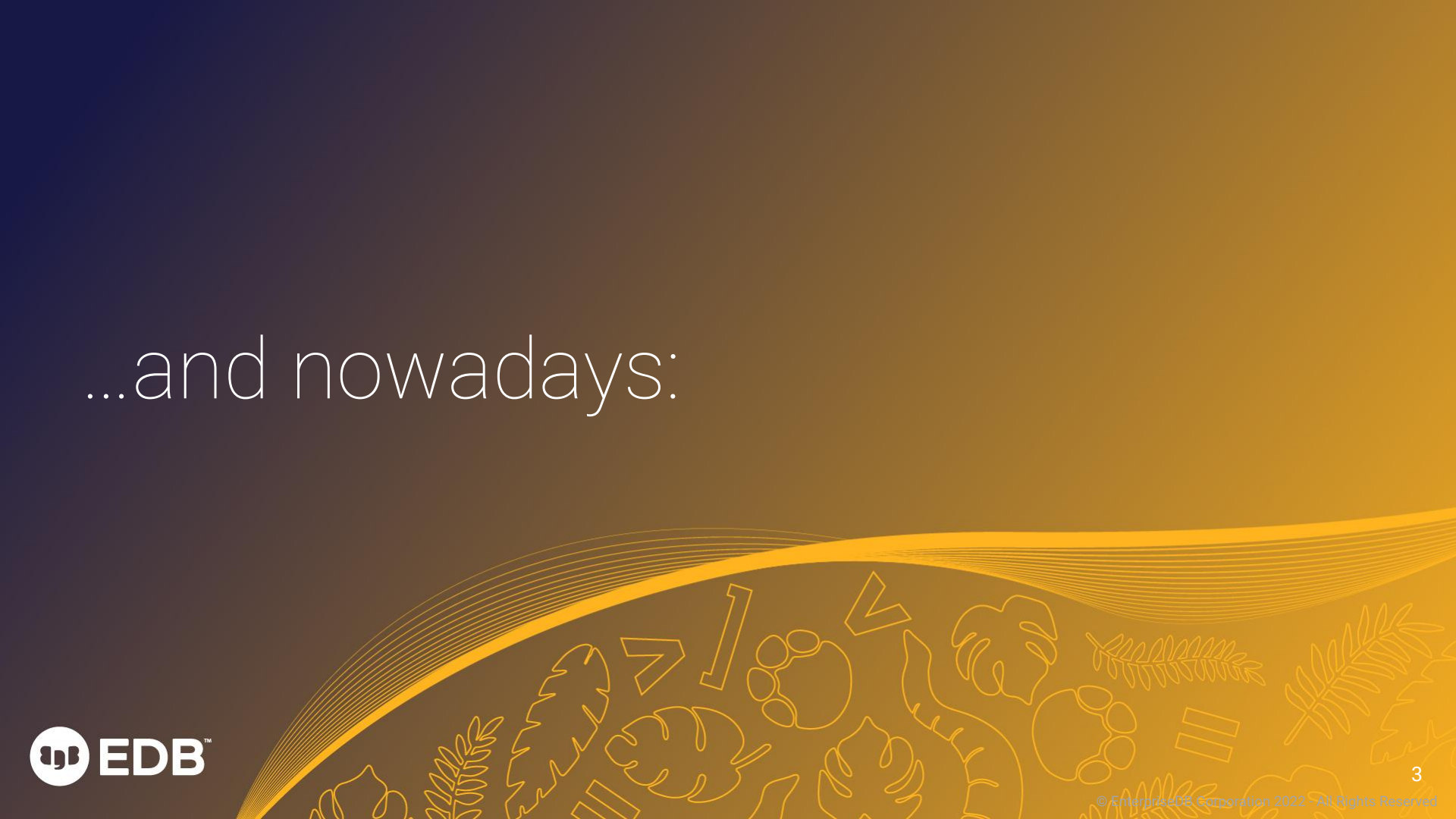
EDB

Self introduction

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

EDB

...and nowadays:



DJing!

DJing!

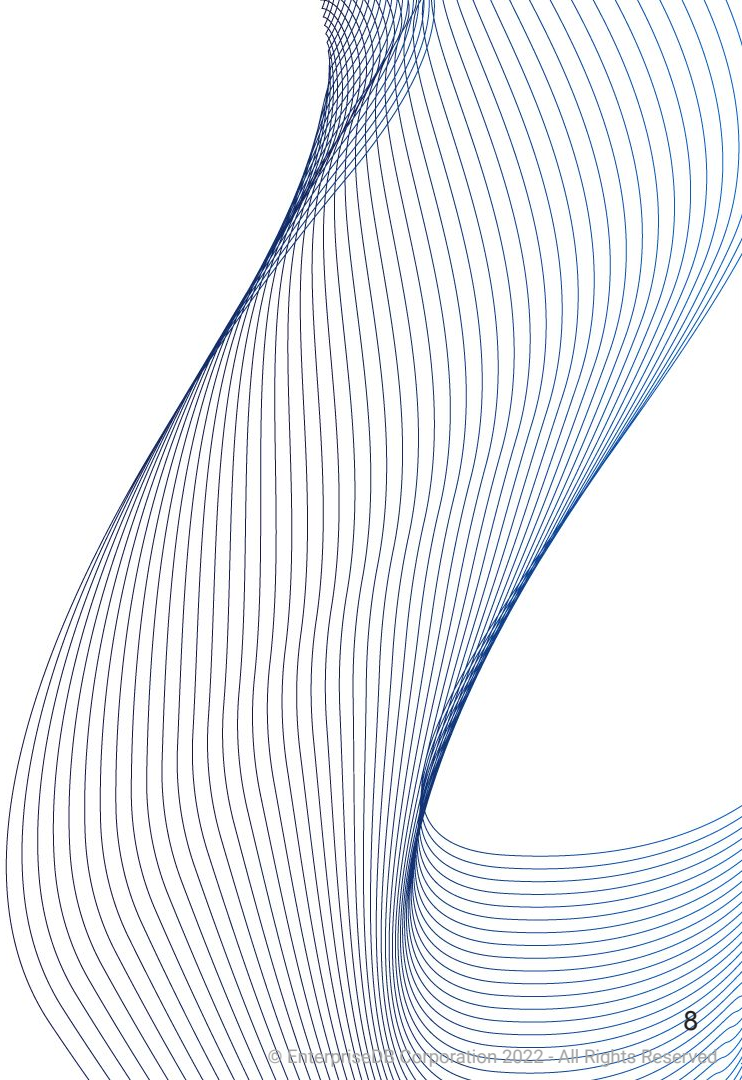
Agenda

- Motivation
- Glossary
- Hidden parameters

Motivation

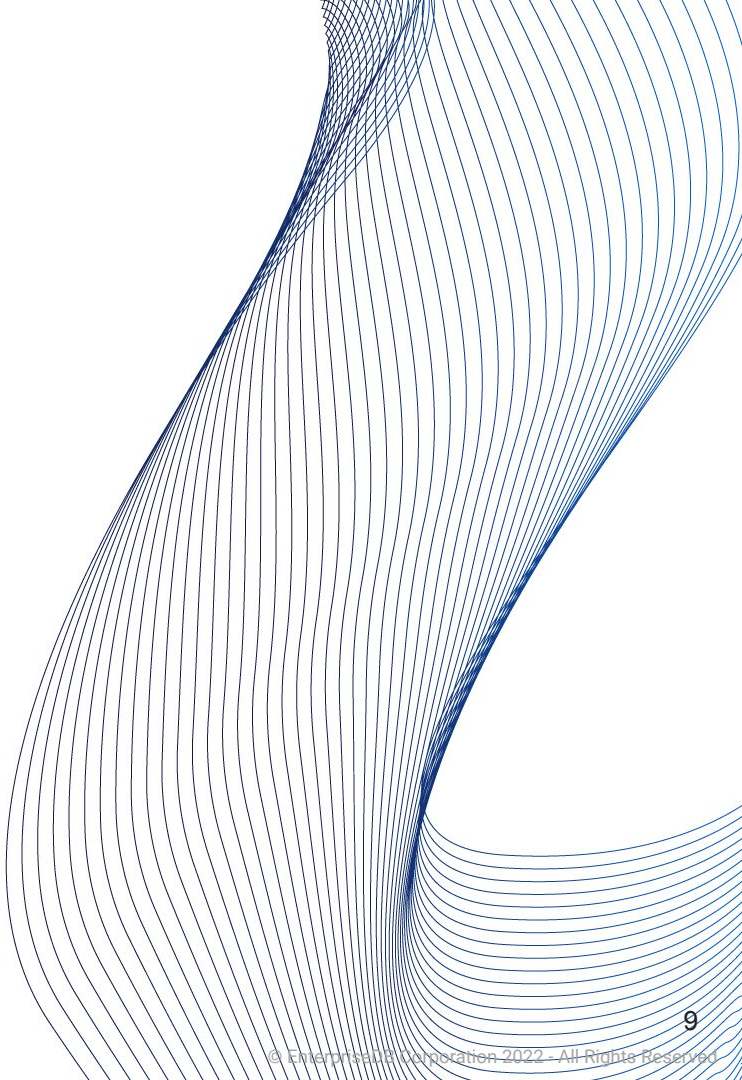
Motivation

- 3 days, 4 tracks, lots of great talks



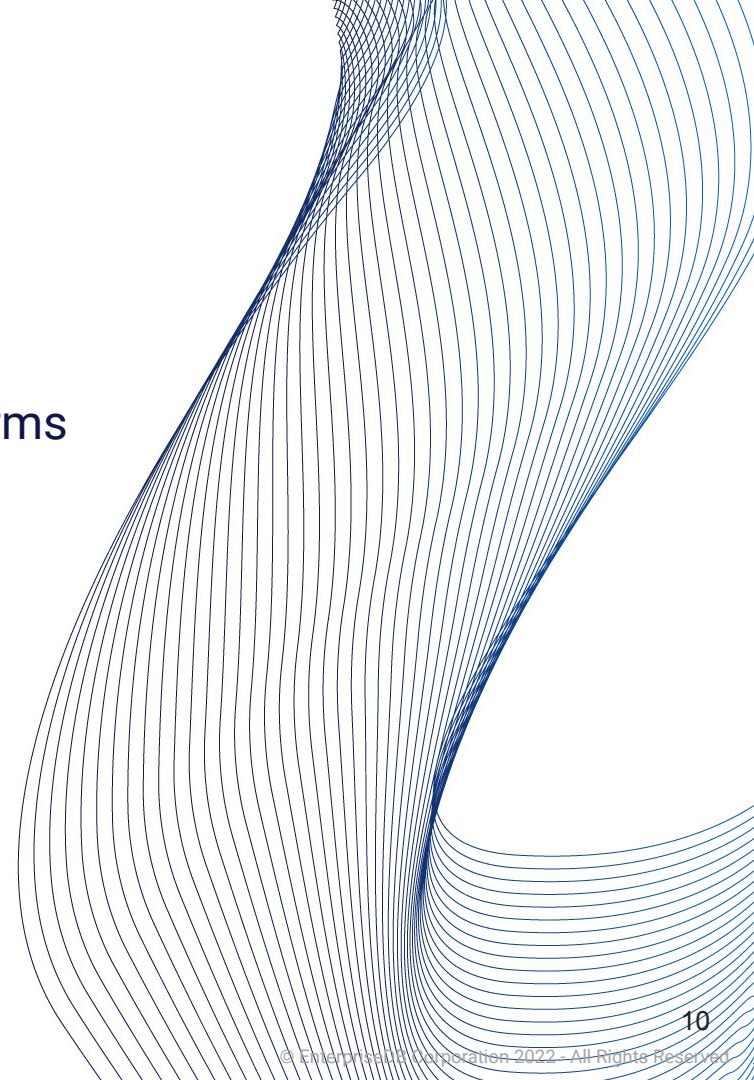
Motivation

- 3 days, 4 tracks, lots of great talks
- There are great tech talks



Motivation

- 3 days, 4 tracks, lots of great talks
- There are great tech talks
- You are new-ish, or not used to some of the terms



Motivation

- 3 days, 4 tracks, 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

- 3 days, 4 tracks, 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 version 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 version 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-bits txid is being discussed

Transaction id

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

Transaction id

- “txid”
- Unique identifier
  - 32-bits, ~ 4 billion
    - 64-bits txid is being discussed
  - “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 visible, always active)**

Transaction id

- SELECT
  - Utilizes “virtual txid”
    - txid_current_if_assigned()

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

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.

Back to txid

- SELECT
  - Utilizes “virtual txid”
    - txid_current_if_assigned()

Back to txid

- SELECT
  - Utilizes “virtual txid”
    - txid_current_if_assigned()
- Stored in the header of each row
  - xmin: INSERT
  - xmax: UPDATE or DELETE
    - (0, when this 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!**

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*
- “bloat”

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: 24 bytes
  - Space is limited

multixact

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

multixact

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

multixact

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

multixact ID

- Implemented as 32-bit counter
- Very much like txid
- $PGDATA/pg_multixact/members: Holds the list of members in each multixact
- VACUUM: Will remove old files from pg_multixact/members and pg_multixact/offsets

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 allowing shrinking of pg_xact

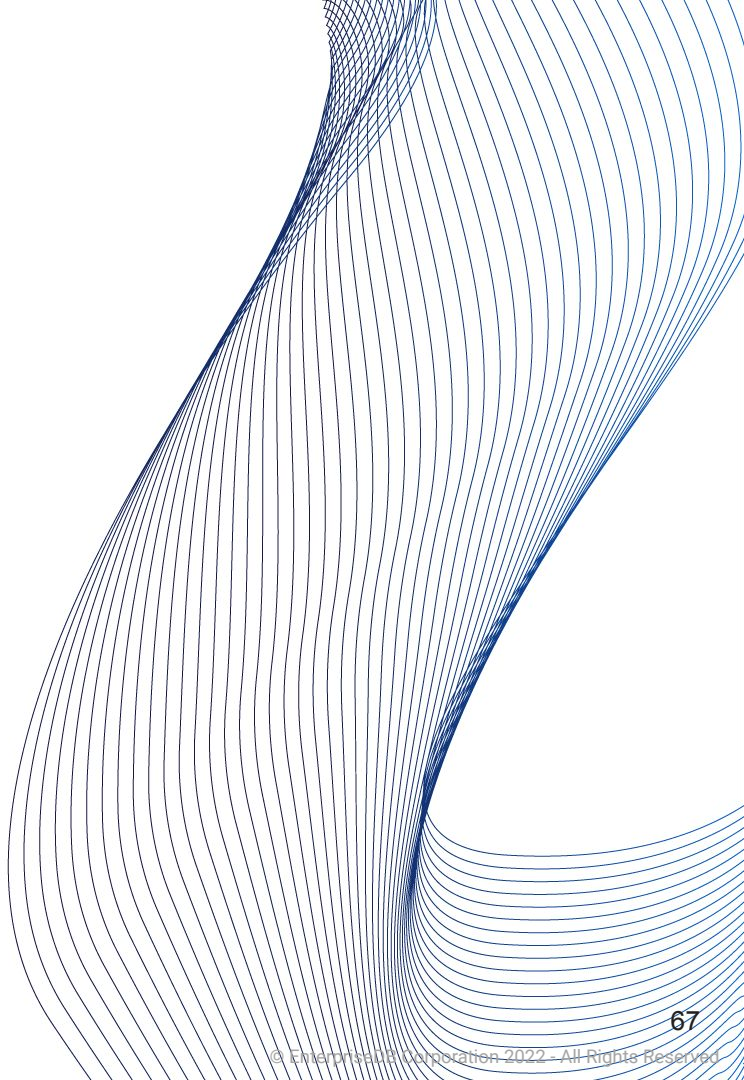
WAL

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

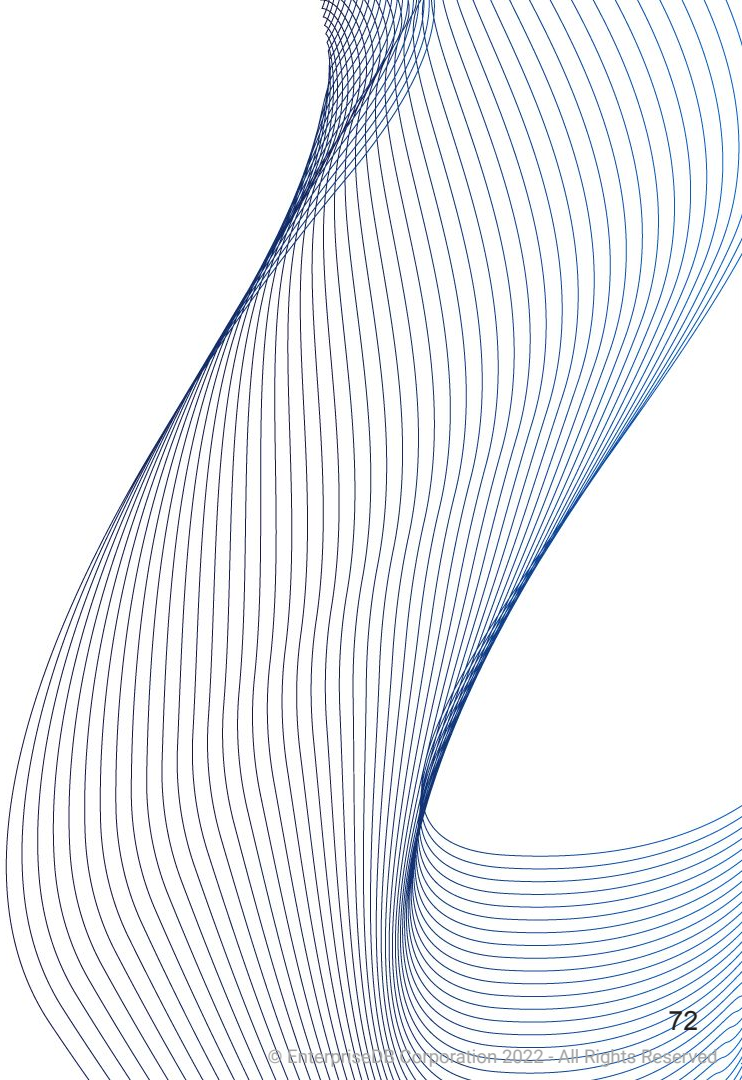
- Write Ahead Log
- Logging of transactions
- Designed to prevent data loss in most of the situations
- OS crash, hardware failure, PostgreSQL crash.
- Built-in feature

WAL

- Transaction logging!

WAL

- Transaction logging!
- Replication

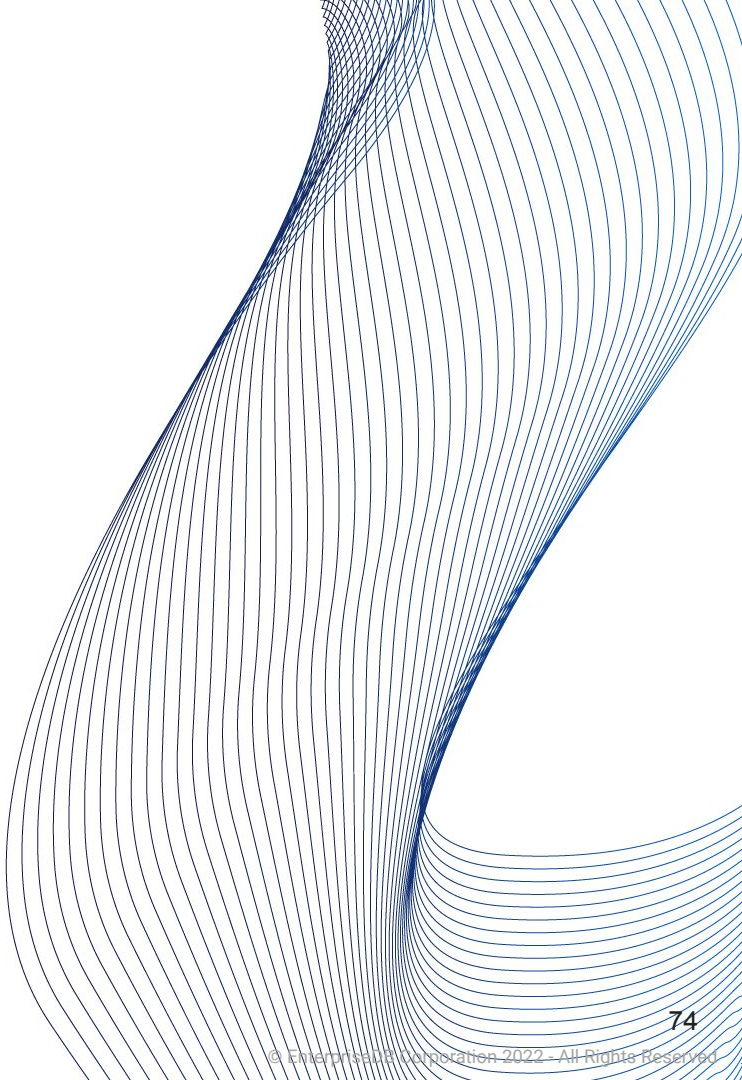


WAL

- Transaction logging!
- Replication
- PITR

WAL

- Transaction logging!
- Replication
- PITR
- REDO



WAL

- Transaction logging!
- Replication
- PITR
- REDO
- Sequentially availability is a must.

WAL

- Transaction logging!
- Replication
- PITR
- REDO
- Sequentially availability is a must.
- REDO vs UNDO

WAL

- Transaction logging!
- Replication
- PITR
- REDO
- Sequentially availability is a must.
- REDO vs UNDO
- No REDO for temp tables and unlogged tables.

LSN

LSN

- Log Sequence Number

LSN

- Log Sequence Number
- Position of the record in WAL file.

LSN

- Log Sequence Number
- Position of the record in WAL file.
- Provides uniqueness for each WAL record.

LSN

- Log Sequence Number
- Position of the record in WAL file.
- Provides uniqueness for each WAL record.
- 64-bit integer (historically 2x32-bit)

LSN

- Log Sequence Number
- Position of the record in WAL file.
- Provides uniqueness for each WAL record.
- 64-bit integer (historically 2x32-bit)
- Per docs: “Pointer to a location in WAL file”

LSN

- Log Sequence Number
- Position of the record in WAL file.
- Provides uniqueness for each WAL record.
- 64-bit integer (historically 2x32-bit)
- Per docs: “Pointer to a location in WAL file”
- LSN: Block ID + Segment ID

LSN

- Log Sequence Number
- Position of the record in WAL file.
- Provides uniqueness for each WAL record.
- 64-bit integer (historically 2x32-bit)
- Per docs: “Pointer to a location in WAL file”
- LSN: Block ID + Segment ID
- During recovery, LSN on the page and LSN in the WAL file are compared.

LSN

- Log Sequence Number
- Position of the record in WAL file.
- Provides uniqueness for each WAL record.
- 64-bit integer (historically 2x32-bit)
- Per docs: “Pointer to a location in WAL file”
- LSN: Block ID + Segment ID
- During recovery, LSN on the page and LSN in the WAL file are compared.
- The larger one wins.

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)

“Read-only” parameters in PostgreSQL

- data_checksums (boolean)
  - Initdb , off by default
- 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
- multibyte
- max_index_keys (integer)
- 32
- segment_size (integer)
- 128

“Read-only” parameters in PostgreSQL

- server_encoding (string)
  - initdb, UTF-8
- server_version (string)
  - 15.2
  - 16devel
- server_version_num (integer)
  - 150002
  - 160000

“Read-only” parameters in PostgreSQL

- wal_block_size (integer)
  - 8192 byte
  - Not the same as block_size
- wal_segment_size (integer)
  - “2”
  - → 16 MB

Know the less known: A PostgreSQL Glossary

Devrim Gündüz
Postgres Expert @ EDB

PGConf.EU 2023

EDB