



EDB

Postgres® for the AI Generation

Know the less known: A PostgreSQL Glossary

Devrim Gündüz

Postgres Expert @ EDB

Berlin PostgreSQL Meetup - July Edition

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.



...and also:



DJ'ing!





Agenda

Last time it took 90 mins to complete this talk, but...



It is really a coincidence...





DI 16.07. POOR BOYZ CLUB



Again, coincidence:

- Thanks Anastasia for covering many parts of my talk :-)



Again, coincidence:

- Thanks Anastasia for covering many parts of my talk :-)
- I had to rewrite the talk



Agenda

- “*”
- MVCC
- Glossary
- WAL
- LSN



“*”



“*”

- Basic question first ;)
- What does * mean in **SELECT * FROM t1;**



What's MVCC?



What's MVCC?

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



What's MVCC?

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



What's 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



What's 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)



Glossary



xact

“Transaction”



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



Back to txid

- SELECT
 - Utilizes “virtual txid”
 - txid_current_if_assigned()
- Stored in the header of each row
 - xmin: INSERT



Back to txid

- SELECT
 - Utilizes “virtual txid”
 - `txid_current_if_assigned()`
- Stored in the header of each row
 - xmin: INSERT
 - xmax: UPDATE or DELETE



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, UPDATE and DELETE

```
[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, UPDATE AND DELETE

- DELETE
 - Logical deletion
 - Long lasting transactions?
 - xmax is set to the txid
 - → **dead tuple!**



INSERT, UPDATE AND DELETE

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, UPDATE AND DELETE

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, UPDATE AND DELETE

```
[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, UPDATE AND DELETE

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”
- 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”
(**multiple 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”
(**multiple 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”
(**multiple 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”
- Tracks vacuum needs to prevent txid wraparound and allowing shrinking of pg_xact



WAL



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!
- 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.





Now it is time for questions!





EDB

Postgres® for the AI Generation

Know the less known: A PostgreSQL Glossary

Devrim Gündüz

Postgres Expert @ EDB

Berlin PostgreSQL Meetup - July Edition