PERCONALIVE

Cleaning the Room Everything You Need To Know About Vacuum

Charly Batista

Who am I?

I’m Charly Batista :)

- PostgreSQL Tech Leader @ Percona
- Database lover
- Working with IT for over 20 years
- Craft beer and coffee lover
- You can find me at:

  https://github.com/elchinoo

  https://www.linkedin.com/in/charlyfrankl

Agenda

MVCC

- What are and why do we need transactions?
- MVCC, what exactly is it?
- MVCC on Postgres

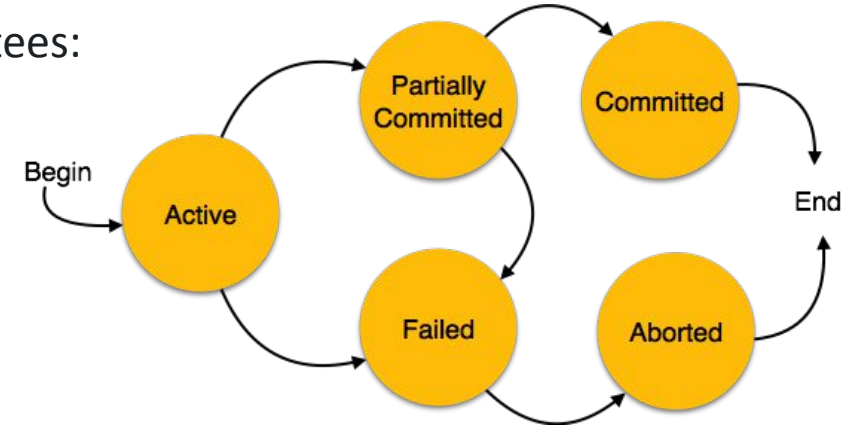
Vacuum and Autovacuum

- What is vacuum?
- How about autovacuum?
- Settings and tuning

MVCC

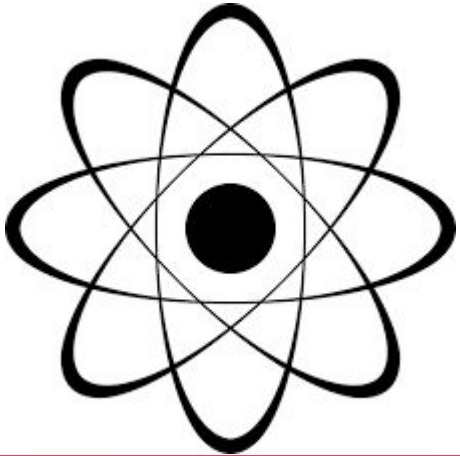
What are transactions?

- A transaction is a logical, atomic unit of work that contains one or more SQL
- Transaction are all about guarantees:
  - Atomicity
  - Consistency
  - Isolation
  - Durability



Atomicity

- An atomic transaction is an indivisible and irreducible series of database operations such that either all occur, or nothing occurs:
  - Indivisible
  - All or nothing



Atomicity

- Example bank transfer
  - John transfers $100 to Alice
    - *Withdraw $100 from John's bank account*
    - *Deposit $100 into Alice's bank account*

Consistency

- A transaction must keep the database in a valid state:
  - Any transactions started in the future necessarily see the effects of other transactions committed in the past
  - Database constraints are not violated, particularly once a transaction commits
  - Operations in transactions are performed accurately, correctly, and with validity, with respect to application semantics

Isolation

- Determines how transaction integrity is visible to other users
- Makes the transaction think it is the only transaction working inside the database
- Ideally prevent race conditions
- Also help prevent anomalies like dirty read, phantom read, etc

Isolation

- There are different levels of isolation
  - Read Uncommitted
  - Read Committed
  - Repeatable Read
  - Serializable
- Postgres default transaction isolation = 'read committed'

Durability

- Committed transactions can not be rolled back
- Committed transactions must survive

How about MVCC?

- Multiversion concurrency control
- Concurrency control method commonly used by databases
- Optimistic - means no locking
  - Readers and Writers do not block each other
- First perform changes in a protected area then change the database state
- Main idea: Version your database (Multiversion :-D)

MVCC on Postgres

MVCC on Postgres

- Read uncommitted not really implemented on Postgres
- No Rollback segments for UNDO
  - UNDO management is within tables
- Rows are never really deleted
- A tuple contains the hidden columns to help managing transactions
  - xmin, xmax, cmin, cmax, …

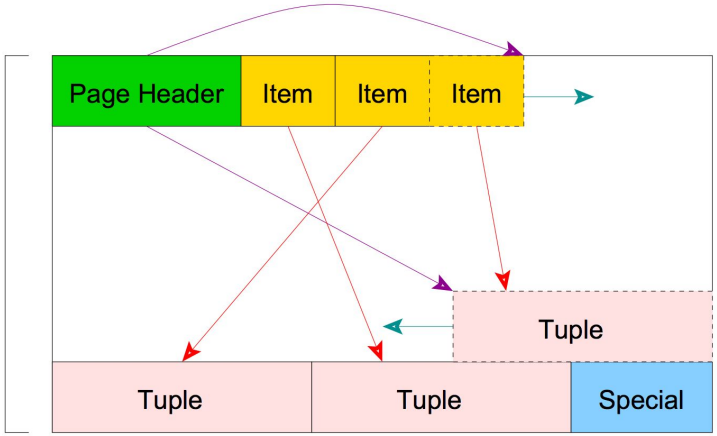
MVCC on Postgres

- Transaction identifiers (xid or transaction IDs) are 32-bit unsigned integer
- ID's 0, 1 and 2 are reserved.
- Can be inspected, for example "`select xmin, xmax, cmin, cmax, a from tb1;`"

Vacuum

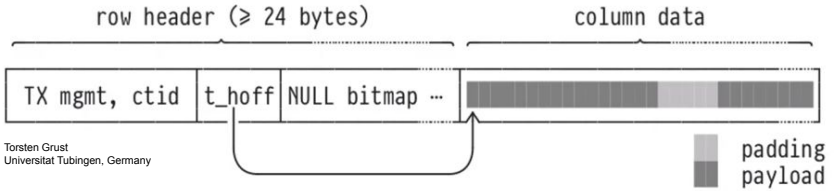
A short pick into a Postgres data file

- A Postgres data file is divided into 8kb pages
- A page has:
  - A 24 bytes header
  - Line pointers (pointer to the data tuple)
  - The data tuple itself
  - A special are in the end of the page



A short pick into a Postgres data file

- The data tuple is where the data lives
- A tuple has:
  - A 24+ bytes header
  - The payloader (data)
  - Padding



Torsten Grust
Universitat Tubingen, Germany

A short pick into a Postgres data file

- The tuple header has information important to us today:
  - t_xmin
  - t_xmax
  - t_infomask
  - t_infomask2

Field	Type	Length	Description
t_xmin	TransactionId	4 bytes	insert XID stamp
t_xmax	TransactionId	4 bytes	delete XID stamp
t_cid	CommandId	4 bytes	insert and/or delete CID stamp (overlays with t_xvac)
t_xvac	TransactionId	4 bytes	XID for VACUUM operation moving a row version
t_ctid	ItemPointerData	6 bytes	current TID of this or newer row version
t_infomask2	uint16	2 bytes	number of attributes, plus various flag bits
t_infomask	uint16	2 bytes	various flag bits
t_hoff	uint8	1 byte	offset to user data

What are the problems we need to solve?

- Postgres doesn’t have undo logs
  - Everything goes to the data files!
- Postgres doesn’t physically remove rows on delete
  - No matter how many rows you remove, the file doesn’t shrink!
- Postgres doesn’t do inplace update but a “delete + insert”
  - Every update duplicate the row and leave the garbage behind!

What are the problems we need to solve?

- Postgres optimizer uses table/index statistics
  - They are not updated after every transaction, too expensive!
  - If the transaction doesn’t update, who update the statistics?
- Postgres uses a 32 bits (4 bytes) transaction identifier
  - It is finite and relatively small, you’ll eventually run out of XID!
  - It can wraparound. Ok, will start from zero again, then what?

Vacuum to rescue

- **VACUUM** — garbage-collect and optionally analyze a database
- Here we'll talk about 4 major variations of vacuum on Postgres:
  - Full
  - Freeze
  - Vacuum
  - Autovacuum

Vacuum to rescue

- **Only rows that are not in any currently running transactions can be vacuumed**
  - It means that long running transactions can prevent dead rows to be removed;
  - Long running transactions can prevent vacuum to freeze old transaction IDs;

Vacuum to rescue

- How does it work?

Live demo time!!!

Autovacuum

- We are here to talk about **vacuum**, not autovacuum, but…
- **Always have the parameter autovacuum set to ON**
- Background Process that tracks the usage and activity information;
- We cannot really control when it runs;
- PostgreSQL identifies the tables needing vacuum or analyze depending on certain parameters, for example threshold and scale factor;

Autovacuum

- Setting global parameters alone may not be appropriate, all the time.
- Regardless of the table size, if the condition for autovacuum is reached, a table is eligible for autovacuum vacuum or analyze.
  - Consider 2 tables with ten records and a million records.
  - Frequency at which a vacuum or an analyze runs automatically could be greater for the table with just ten records.

Autovacuum

- Use table level autovacuum settings instead.
- ALTER TABLE foo.bar SET (autovacuum_vacuum_scale_factor = 0, autovacuum_vacuum_threshold = 100);

- Autovacuum reads block_size pages of a table from disk (default of 8KB), and modifies/writes to the pages containing dead tuples;
- Involves both read and write IO and may be heavy on big tables with huge amount of dead tuples;

Vacuum

Congratulations, we made it to the end!!!

- If we need to learn one thing from this presentation is:
  NEVER DISABLE YOUR AUTOVACUUM
- Transactions are not free, don’t let a transaction open for a long time;
- Transaction IDs are not unlimited, make sure your autovacuum is able to freeze them;
- Vacuum is able to prevent bloating to increase but not able to pack the table;

Congratulations, we made it to the end!!!

- Vacuum FULL is the only one able to return disk space back, but it locks the table;
- Sometimes we need to have per-table autovacuum configuration;
- Again, NEVER DISABLE YOUR AUTOVACUUM. If it’s causing problems is because you didn’t understand how it works and you may need to make it more aggressive!!!


Are you passionate about Open Source?!
We're looking for you!
Join us!
#RemoteWork
APPLY NOW: percona.com/careers

Thank you!