VACUUM

Devrim Gündüz
Postgres Expert @ EDB
Twitter: @DevrimGunduz

27 Jun 2023

Self introduction

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

Nowadays: *Also* DJ’ing!

Before I start:

- PGDay.UK !
- September 12, 2023
- London, UK
- 1- day single track community conference
- CfP and CfS open:
- https://2023.pgday.uk

Agenda

- MVCC: The basics
- Data snapshots
- VACUUM
- VACUUM processing
- FREEZE
- VACUUM tuning
- VACUUM FULL

“*”

“*”

- 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 to not block writers, writer do not block readers”.

MVCC

- Multi Version Concurrency Control
  - Implementation of concurrency in Postgres
  - Snapshot isolation
- “Readers to not block writers, writer 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 to not block writers, writer 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
    - We will get there ;)

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
    - **2: Frozen**

Transaction id

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

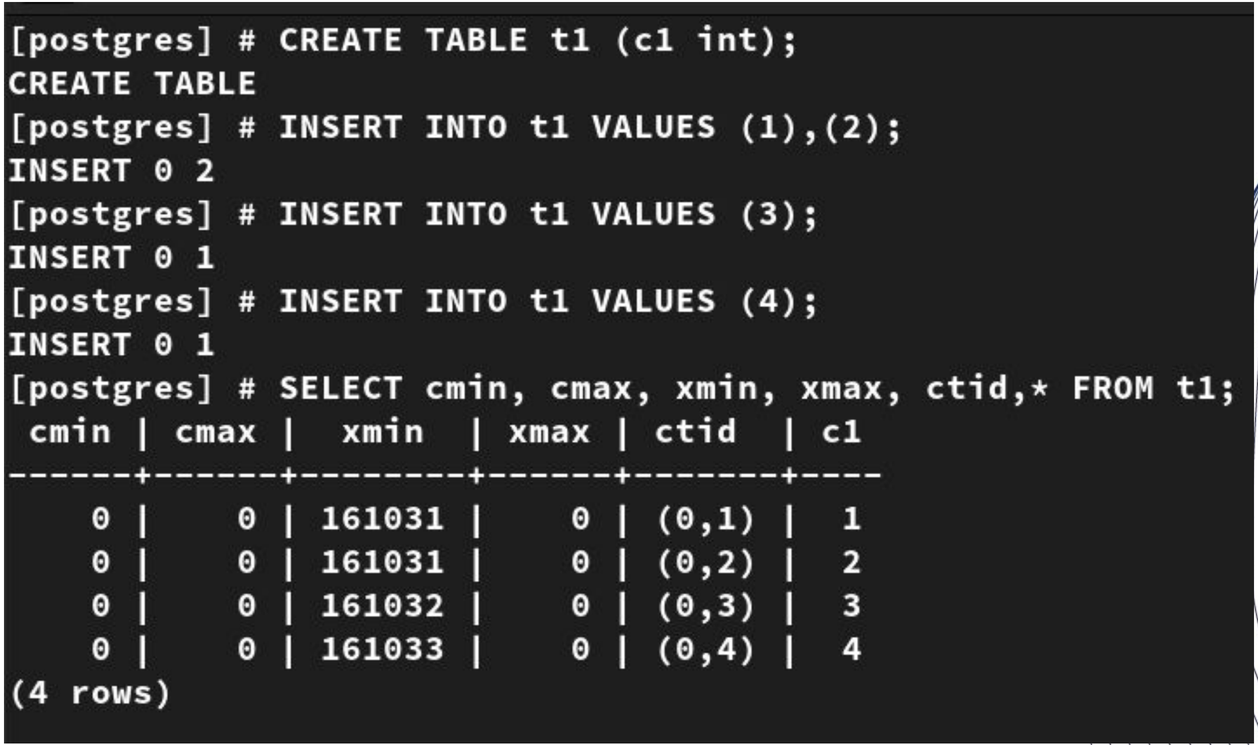
Transaction id

- 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



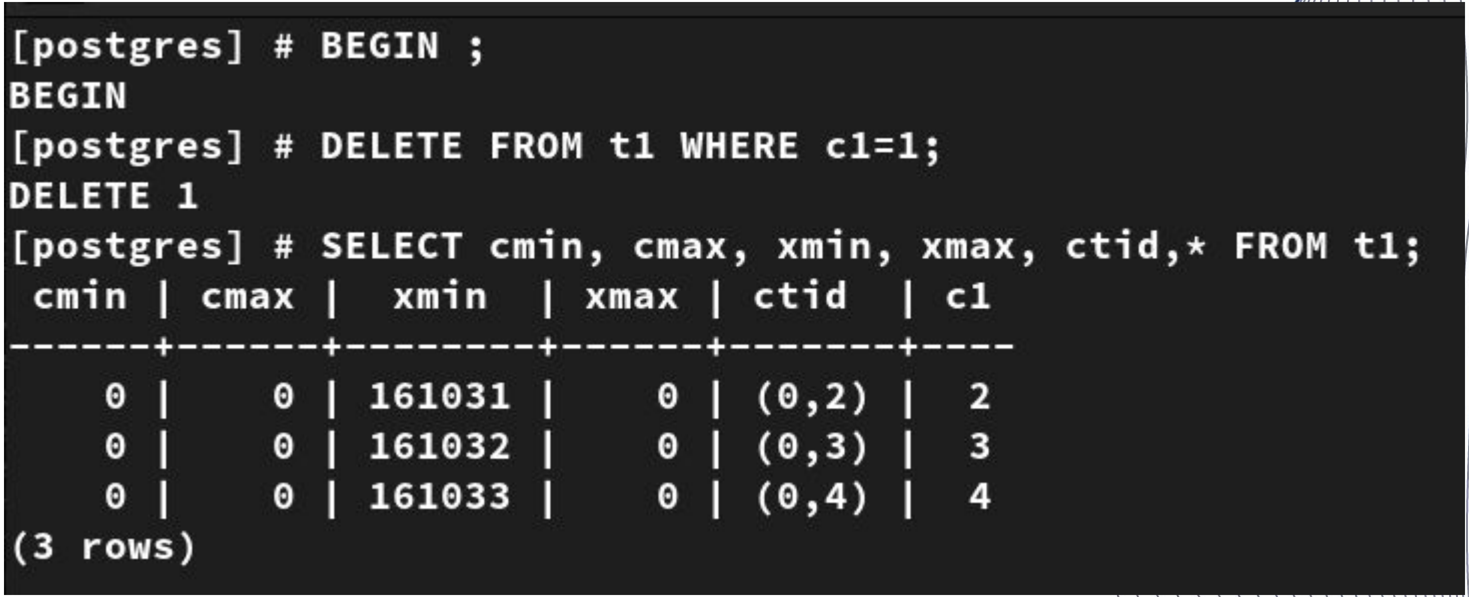
```
[postgres] # CREATE TABLE t1 (c1 int);
CREATE TABLE
[postgres] # INSERT INTO t1 VALUES (1),(2);
INSERT 0 2
[postgres] # INSERT INTO t1 VALUES (3);
INSERT 0 1
[postgres] # INSERT INTO t1 VALUES (4);
INSERT 0 1
[postgres] # SELECT cmin, cmax, xmin, xmax, ctid,* FROM t1;
 cmin | cmax |  xmin  | xmax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

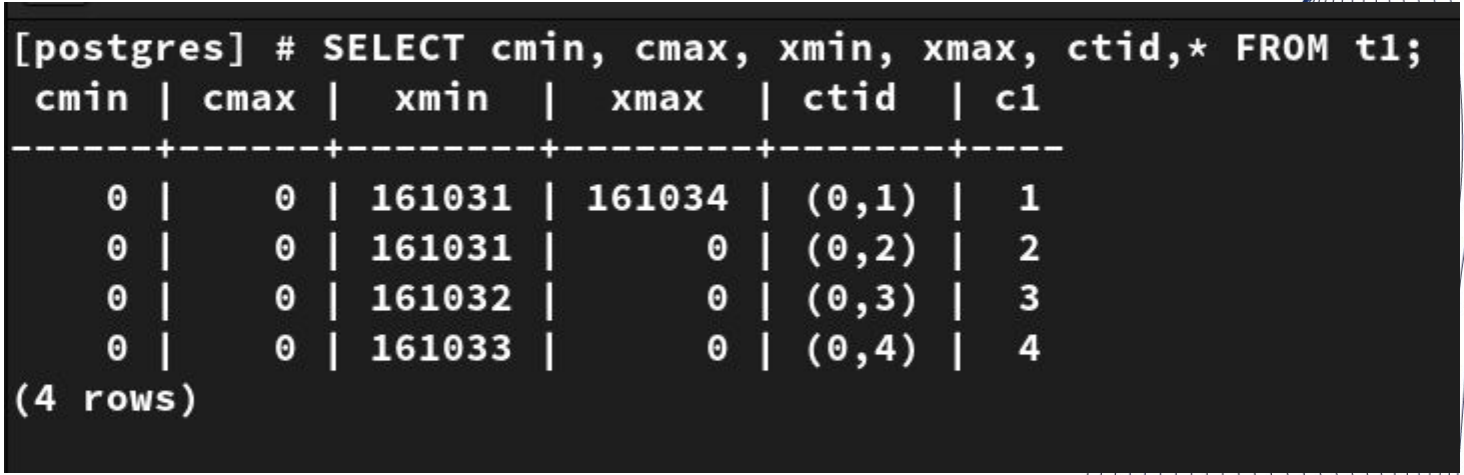
First session:



```
[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:

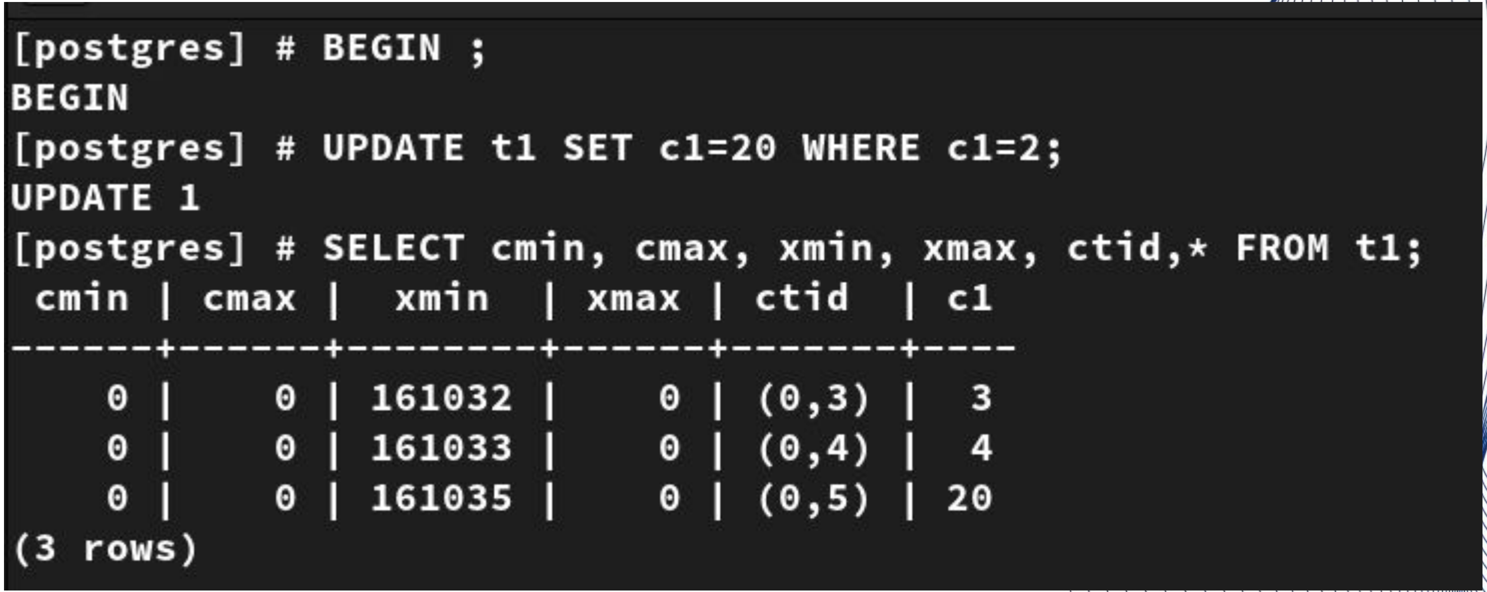


```
[postgres] # SELECT cmin, cmax, xmin, xmax, ctid,* FROM t1;
 cmin | cmax |  xmin  |  xmax  | ctid  | c1
------+------+--------+--------+-------+----
    0 |    0 | 161031 | 161034 | (0,1) |  1
    0 |    0 | 161031 |      0 | (0,2) |  2
    0 |    0 | 161032 |      0 | (0,3) |  3
    0 |    0 | 161033 |      0 | (0,4) |  4
(4 rows)
```

INSERT, DELETE and UPDATE

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

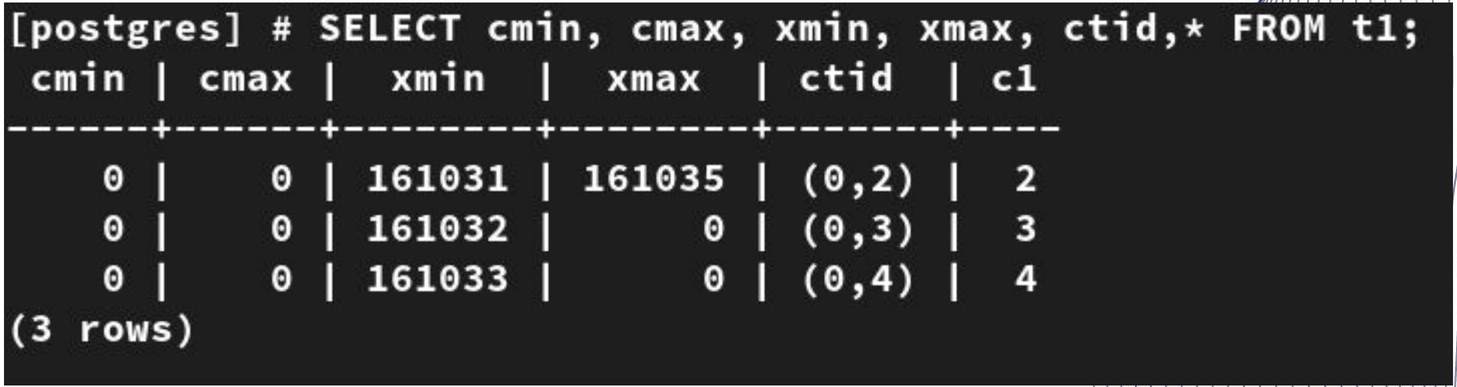
INSERT, DELETE and UPDATE



```
[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:



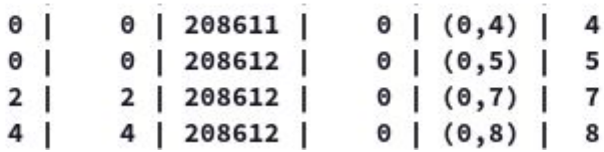
```
[postgres] # SELECT cmin, cmax, xmin, xmax, ctid,* FROM t1;
 cmin | cmax |  xmin  |  xmax  | ctid  | c1
------+------+--------+--------+-------+----
    0 |    0 | 161031 | 161035 | (0,2) |  2
    0 |    0 | 161032 |      0 | (0,3) |  3
    0 |    0 | 161033 |      0 | (0,4) |  4
(3 rows)
```

INSERT, DELETE and UPDATE

- Consider huge side effects of excessive DELETEs (and UPDATEs)

Comboid, cmin, cmax

- pre-8.3: cmin and cmax were separate
- Per comboid.c: “
  - To reduce the header size, cmin and cmax are now overlayed 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
```

Data snapshots

Data snapshots

- Data snapshots
  - Not a physical snapshot

Data snapshots

- Data snapshots
    - Not a physical snapshot
- Isolation
    - Created at the beginning of the transaction
    - Contains committed data
    - Uncommitted data is ignored.

Data snapshots

- Data snapshots
  - Not a physical snapshot
- Isolation
  - Created at the beginning of the transaction
  - 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 (disabled by default)

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 transaction start time
    - 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 transaction start time
    - Could be INSERT, or UPDATE
  - UPDATE waiting?
- Tip: Commit time is not stored.

Visibility

- Tuple visibility
  - xmin,xmax
- Only one version is available in a snapshot
- Visibility definition:
  - That row version is already committed before the transaction start time
    - Could be INSERT, or UPDATE
  - UPDATE waiting?
- Tip: Commit time is not stored.
- 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
  - All databases

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
  - All databases
- Two main tasks:
  - Removing dead tuples
  - Freezing transaction ids

VACUUM

- Does **not** block most of the queries
  - Concurrent vacuums to the same table is 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 is 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.
- Scan pages for dead tuples

VACUUM process

- VACUUMing is done per table, per page.
- Scan pages for dead tuples
- Remove index entries pointing to the dead tuples

VACUUM process

- VACUUMing is done per table, per page.
- 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.
- 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.
- 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.
- Freeze tuples (we will get there)

VACUUM: First phase

- Scan the table, and create list of the dead tuples.
- 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.
- 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
    - VACUUM can utilize up to 1 GB (matches on-disk data file size limit)

VACUUM: First phase

- Some parameters:
  - Maintenance_work_mem
    - Can also be set per-session
    - VACUUM can utilize up to 1 GB (matches on-disk data file size limit)

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

- VACUUM uses “ring buffer”
  - temporary
  - small

VACUUM: Ring buffers

- VACUUM uses “ring buffer”
  - temporary
  - small
- Does not use buffer pool

VACUUM: Ring buffers

- VACUUM uses “ring buffer”
  - temporary
  - small
- Does not use buffer pool
- Helps keep shared buffers “hot”

VACUUM: Ring buffers

- VACUUM uses “ring buffer”
  - temporary
  - small
- Does not use buffer pool
- Helps keep shared buffers “hot”
- 256 kB
  - Per docs (src/backend/storage/buffer/README):
  - “For sequential scans, a 256 KB ring is used.
    That's small enough to fit in L2 cache,
    which makes transferring pages from OS cache
    to shared buffer cache efficient.”

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
- Scans all pages (and files, when the table spans more than one file)

VACUUM: FREEZE

- “Transaction ID wraparound problem”
    - Time to recall “circle”
    - A must-avoid problem
- FREEZE
    - frozen txid
- Scans all pages (and files, when the table spans more than one file)
- Specially reserved txid: 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
- Scans all pages (and files, when the table spans more than one file)
- Specially reserved txid: 2
  - "Always older than other transaction IDs"
  - "Always visible"
- vacuum_freeze_min_age

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!

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 rule: Autovacuum will kick of during peak hours.

AUTOVACUUM: Is everything cool?

- No.
- Murphy rule: Autovacuum will kick of during peak hours
- May / will prioritize busy tables
  - Some tables may / will be untouched

AUTOVACUUM: Is everything cool?

- No.
- Murphy rule: Autovacuum will kick of during peak hours
- May / will prioritize busy tables
  - Some tables may / will be untouched
- Anti-wraparound vacuum cannot be stopped.
  - 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: Is everything cool?

- More workers -> more I/O
- More workers -> more RAM usage (maintenance_work_mem)
- Cancels itself when a higher lock level is required by another transaction
    - Some tables may never be autovacuumed.

AUTOVACUUM: parameters

- autovacuum_work_mem = -1
- log_autovacuum_min_duration = 10min
- autovacuum = on
- autovacuum_max_workers = 3
- autovacuum_naptime = 1min
- autovacuum_vacuum_threshold = 50
- autovacuum_vacuum_insert_threshold = 1000
- autovacuum_analyze_threshold = 50

AUTOVACUUM: parameters

- autovacuum_vacuum_scale_factor = 0.2
- autovacuum_vacuum_insert_scale_factor = 0.2
- autovacuum_analyze_scale_factor = 0.1
- autovacuum_freeze_max_age = 200000000
- autovacuum_multixact_freeze_max_age = 400000000
- autovacuum_vacuum_cost_delay = 2ms
- autovacuum_vacuum_cost_limit = -1

Autovacuum: Tuning per table

```
ALTER TABLE t1
 SET (autovacuum_vacuum_scale_factor = 0.05,
     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

VACUUM FULL: Non-blocking Alternative

- Some alternatives exist
  - pg_repack
  - pg_squeeze

pg_stat_progress_vacuum

```
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_tuples         | 107682804
num_dead_tuples         | 149101
```

VACUUM VERBOSE

- INFO: finished vacuuming "onlinedps.pg_toast.pg_toast_20508": index scans: 0
- pages: 0 removed, 0 remain, 0 scanned (100.00% of total)
- tuples: 0 removed, 0 remain, 0 are dead but not yet removable
- removable cutoff: 30184655, which was 3 XIDs old when operation ended
- new relfrozenxid: 30184655, which is 30180246 XIDs ahead of previous value
- new relminmxid: 16, which is 15 MXIDs ahead of previous value
- index scan not needed: 0 pages from table (100.00% of total) had 0 dead item identifiers removed
- I/O timings: read: 0.051 ms, write: 0.000 ms
- avg read rate: 32.150 MB/s, avg write rate: 0.000 MB/s
- buffer usage: 19 hits, 1 misses, 0 dirtied
- WAL usage: 1 records, 0 full page images, 188 bytes
- system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s

THANK YOU

Now it is time for questions!

VACUUM

Devrim Gündüz
Postgres Expert @ EDB
Twitter: @DevrimGunduz

27 Jun 2023