All the dirt on VACUUM – Postgres 11

Jim Nasby, Sr. Database Engineer
PostgresConf US 2019
Intro

In-depth talk

But... here's some quick tips
Intro

Postgres Multi-Version Concurrency Control is like a credit card

Every UPDATE, DELETE and ROLLBACK leaves “debt” that must be repaid

Not paying off credit cards leads to bankruptcy

Not vacuuming leads to a “death spiral”
Intro

Postgres Multi-Version Concurrency Control is like a credit card

Every UPDATE, DELETE and ROLLBACK leaves “debt” that must be repaid

Vacuum is how this debt is repaid

You **WANT** vacuum running in your database
I LOVE vacuum! How can I get more of it?
**Intro**

By default, autovacuum limited to 4MB/s write
Increase `vacuum_cost_limit` from 200 to 2000

Ensure `maintenance_work_mem` is as close to 1GB as possible
Heap Only Tuples

Applies when an update does not change any index values* and the new tuple will fit on the same page

Allows for updating just the heap, without touching indexes

Dead HOT tuples can be removed at any time, without need for vacuum

HOT is like paying cash instead of using a credit card

See Grant McAlister’s talk from yesterday

* Indexed values means any column referenced anywhere in an index, including predicates and functions. See src/backend/access/heap/README.HOT.
Intro

Beware of:
- Long-running transactions (including idle in transaction)
- Prepared transactions (best to set $\text{max}\_\text{prepared}\_\text{transactions} = 0$)
- Stuck replicas

Targeted manual vacuums help a lot
- Vacuum small, frequently modified tables once a minute
- Vacuum the entire instance once a day / week
  (possibly with $\text{vacuum}\_\text{cost}\_\text{delay} > 0$)
VACUUM

VACUUM FULL: completely rebuilds table and indexes

VACUUM FREEZE: sets freeze table age limits to 0

VACUUM: regular manual vacuum

VACUUM ANALYZE: also runs analyze after vacuum

VACUUM VERBOSE: provides status and stats

See also vacuumdb shell command

Autovacuum: built-in background automatic vacuum process
VACUUM FULL

Rebuilds table and indexes from scratch, similar to CLUSTER

Takes an exclusive lock on the table

Since it’s a table rebuild, doesn’t actually vacuum anything

https://github.com/reorg/pg_repack is another alternative
**vacuum()**

Can not be run in a transaction (or function/procedure)

For each table, call vacuum_rel() (and analyze_rel() if requested)

Update datfrozenxmin and datminmxid
vacuum_rel()

vacuum() > vacuum_rel()

Vacuums a single relation

Non-aggressive autovac will skip relation if locked

Does a bunch of mundane stuff then calls either cluster_rel() (VACUUM FULL) or lazy_vacuum_rel()

If not autovac, call itself to vacuum the TOAST table
There was a bug in some old versions where relfrozenxid and relminmxid were updated even if the whole table hadn’t been scanned, potentially resulting in data loss.

See vacuum_set_xid_limits()
Vacuum Process (lazy_vacuum_rel())

- Acquire locks, set limits
- Loop through heap, possibly skipping pages
- Per-page activity
- *Serially* loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap
- Index cleanup
- Attempt relation truncation
- Update pg_class info
Vacuum Process \texttt{(lazy\_vacuum\_rel())}

- Acquire locks, set limits
- Loop through heap, possibly skipping pages
- Per-page activity
- *Serially* loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap

- Index cleanup
- Attempt relation truncation
- Update \texttt{pg\_class info}

SETUP

MAIN LOOP

CLEANUP
What tuples can be vacuumed?

Only rows that are not visible to currently running transactions

Generally* limited by the **oldest running transaction** in the database

Can be changed by **old_snapshot_threshold**

**Streaming replication** (vacuum_defer_cleanup_age, hot_standby_feedback), **prepared transactions**, and logical decoding can also affect it

**Special handling** for current XIDs and locks

See HeapTupleSatisfiesVacuum() for details.
What tuples can be vacuumed?

BEGIN;
-- Go to lunch

Lots of UPDATEs and DELETEs

-- Back from lunch
COMMIT;

Vacuum and HOT can’t do any cleanup here ...

BLOAT

See HeapTupleSatisfiesVacuum() for details.
What tuples can be vacuumed?

- Very expensive `SELECT`
- Lots of `UPDATEs` and `DELETEs`

`SELECT finishes`

See `HeapTupleSatisfiesVacuum()` for details.
What tuples can be vacuumed?

- Very expensive `SELECT` on a replica with `hot_standby_feedback`
  Logical replica stops replaying

- Lots of `UPDATE`es and `DELETE`es

`SELECT` finishes

See `HeapTupleSatisfiesVacuum()` for details.
Setting limits: Freezing

Transaction ID (XID) and MultiXact ID (MXID) values are limited to 31 effective bits.

Allowing these values to roll over would result in data loss.

Old values must be “frozen”.

An “aggressive” vacuum is run when a table contains XIDs or MXIDs in need of freezing, as determined by `relfrozenoid` and `relinvmxmidxid` in `pg_class`.

XIDs and MXIDs are 32 bit unsigned values, but XIDs need to accommodate transactions that are considered to be “in the future”, which means there can’t be more than 31 effective bits. MXIDs don’t have a concept of “in the future”, but are artificially limited to 31 bits.
Freezing

A non-aggressive vacuum can also freeze tuples

\texttt{vacuum\_freeze\_min\_age* and vacuum\_multixact\_freeze\_min\_age}
determine how old a XID/MXID must be to be considered for freezing

\texttt{(auto)vacuum\_freeze\_table\_age* and}
\texttt{(auto)vacuum\_multixact\_freeze\_table\_age} determine when a
vacuum will become aggressive

\*\texttt{min\_age} are computed from the oldest running XID in the system.
\*\texttt{table\_age} are computed from pg\_class\_realfrozenxid and
pg\_class\_relminxid.
Freezing

Vacuum is aggressive  Vacuum is non-aggressive

Tuples will be frozen  Tuples will not be frozen

*_freeze_table_age  XIDs / time  *_freeze_min_age

Age
What is a MultiXact?

MultiXacts occur when multiple transaction IDs need to lock and invalidate a tuple, most commonly due to updates on a Foreign Key parent.

Subtransactions (from savepoints & plpgsql EXCEPTION handlers) create their own transaction IDs, so a single backend can create MultiXacts
Vacuum Process (`lazy_vacuum_rel()`)  

- Acquire locks, set limits  
- Loop through heap, possibly skipping blocks  
- Per-page activity  
- *Serially* loop through all indexes  
- Remove dead tuples from heap  
- Vacuum FreeSpaceMap  
- Index cleanup  
- Attempt relation truncation  
- Update `pg_class` info

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Loop through heap, possibly skipping blocks

```c
for (blkno = 0; blkno < nblocks; blkno++)
```

`nblocks` is determined on entry to `lazy_scan_heap()`.

Newer blocks will not be scanned.
Block skipping

Vacuum can skip blocks that are all-visible
An aggressive vacuum can skip blocks that are all-frozen

While reading heap, vacuum will skip blocks if at least 32 blocks would be skipped

Skipping can be disabled by adding the DISABLE_PAGE_SKIPPING option to VACUUM.
Block skipping

4 heap blocks

Visibility Map

4 all-frozen bits
4 all-visible bits
Block skipping

4 heap blocks

Visibility Map

4 all-frozen bits

4 all-visible bits
Block skipping

64 heap blocks

All-visible bits
Block skipping

> 32 all-visible blocks; skip ahead

All-visible bits
Block skipping

> 32 all-visible blocks; skip ahead

All-visible bits

All-frozen bits

An aggressive vacuum can not skip blocks that are not all-frozen
Block skipping

> 32 all-visible blocks; skip ahead

All-visible bits

All-frozen bits

By definition, blocks that are not all-visible must also not be all-frozen
Vacuum Process (\texttt{lazy\_vacuum\_rel()})

- Acquire locks, set limits
- Loop through heap, possibly skipping blocks
  - Per-page activity
  - *Serially* loop through all indexes
  - Remove dead tuples from heap
  - Vacuum FreeSpaceMap
- Index cleanup
- Attempt relation truncation
- Update \texttt{pg\_class} info

\textbf{SETUP} \hspace{1cm} \textbf{MAIN LOOP} \hspace{1cm} \textbf{CLEANUP}
Per-page activity

- Attempt to lock page
  If aggressive vacuum and any tuples need freezing, wait for lock
- Perform HOT pruning (heap_page_prune())
- Scan items on page, deciding how to handle each tuple
- Freeze items (if any)
- If no indexes, vacuum page (lazy_vacuum_page())
- Update visibility map if needed (all-visible & all-frozen)
Every time a backend takes a reference to a buffer, it gets a “pin”. See src/backend/storage/buffer/README.
Vacuum Process (*lazy_vacuum_rel()*

- Acquire locks, set limits
- Loop through heap, possibly skipping blocks
- Per-page activity
  - *Serially* loop through all indexes
  - Remove dead tuples from heap
  - Vacuum FreeSpaceMap
- Index cleanup
- Attempt relation truncation
- Update `pg_class` info

**SETUP**

**MAIN LOOP**

**CLEANUP**

© 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Loop through all indexes

Index lookups can involve user-defined code
Postgres does not trust that for something as critical as vacuum
Lots of index probes could also be quite expensive

Instead of probing indexes as each tuple to be removed is discovered, vacuum remembers each TupleID

Each index method implements its own routine for scanning the index, checking each index tuple against the list of remembered heap TupleIDs
Loop through all indexes

autovacuum_work_mem
maintenance_work_mem

Scan items on page
Find another thread as needed. Remember to maintain its TID.
Loop through all indexes

autovacuum_work_mem
maintenance_work_mem

TID TID TID TID TID TID \[ \ldots \] TID TID TID TID TID TID TID
Loop through all indexes

For each index, scan through entire index checking each TID against the list of remembered TIDs: `lazy_vacuum_index()`

Is TID 2300 in list?  
Is TID 45 in list?  
Is TID 301 in list?
Vacuum Process *(lazy_vacuum_rel())*

- Acquire locks, set limits
- Loop through heap, possibly skipping blocks
- Per-page activity
- "Serially" loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap

**SETUP**

**MAIN LOOP**

**CLEANUP**

- Index cleanup
- Attempt relation truncation
- Update *pg_class* info

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Remove dead tuples from heap

```python
lazy_vacuum_heap()
```

Using the list of TIDs
- Go to each block with tuples to be vacuumed
- Remove the tuples from the block
- Repair page fragmentation
- Update visibility map
- Update FreeSpaceMap
Vacuum FreeSpaceMap

Update non-leaf data in the FreeSpaceMap
Vacuum Process (`lazy_vacuum_rel()`)
Vacuum Process (lazy_vacuum_rel())

- Loop through heap, possibly skipping blocks
- Per-page activity
- *Serially* loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap

MAIN LOOP

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Running out of memory for TIDs

```
auto vacuum work mem
maintenance work mem
(values capped at 1GB, TIDs are 6 bytes)
```

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Vacuum Process (lazy_vacuum_rel())

- Loop through heap, possibly skipping blocks
- If about to run out of TID memory:
  - *Serially* loop through all indexes
  - Remove dead tuples from heap
  - Vacuum FreeSpaceMap

MAIN LOOP

- Per-page activity
- *Serially* loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap
Vacuum Process (lazy_vacuum_rel())

- Loop through heap, possibly skipping blocks
- If about to run out of TID memory:
  - *Serially* loop through all indexes
  - Remove dead tuples from heap
  - Vacuum FreeSpaceMap
- Per-page activity
- *Serially* loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap
Vacuum Process (lazy_vacuum_rel())

- Acquire locks, set limits
- Loop through heap, possibly skipping blocks
- Per-page activity
- *Serially* loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap

- Index cleanup
- Attempt relation truncation
- Update pg_class info

SETUP

MAIN LOOP

CLEANUP

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Index Cleanup
(lazy_cleanup_index())

Call index-specific cleanup method

For B-tree, simply cleans up index free space map
**Attempt Truncation**

See if truncation would save enough space to be worth-while  
(should_attempt_truncation())

- Must be at least 1,000 empty blocks
- Number of empty blocks must be >= 16% of heap
- Truncation is not possible if old_snapshot_threshold is set

**Attempt truncation** (lazy_truncate_heap())

1. Abort if new pages added since vacuum started
2. Try to exclusive-lock table (up to 5 seconds)
3. Scan backwards to find last non-empty page. If our lock is blocking someone, go back to step 1
4. If pages were found, actually truncate relation
**Update pg_class info**
\(\text{vac_update_relstats()}\)

If relpages, reltuples, relallvisible, relfrozenoid or relminoid have changed, then update them in pg_class

If this is a vacuum (and not just an analyze), also update relhasindex, relhasrules and relhastriggers
Vacuum Process (*lazy_vacuum_rel()*)

- Acquire locks, set limits
- Loop through heap, possibly skipping blocks
- Per-page activity
- *Serially* loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap

- Index cleanup
- Attempt relation truncation
- Update *pg_class* info

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
vac_update_datfrozenxid()

vacuum() > vac_update_datfrozenxid()

Update datfrozenxid and datminmxid in pg_database

If new values for either:
  • Truncate Commit LOG files (pg_xact/)
  • Update internal frozen XID and MXID info

MultiXact files (pg_multixact) are truncated during checkpoint
### pg_stat_progress_vacuum

View "pg_catalog.pg_stat_progress_vacuum"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>datid</td>
<td>oid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>datname</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>relid</td>
<td>oid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phase</td>
<td>text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blks_total</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blks_scanned</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blks_vacuumed</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index_vacuum_count</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_dead_tuples</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>num_dead_tuples</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### pg_stat_progress_vacuum

View "pg_catalog.pg_stat_progress_vacuum"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>datid</td>
<td>oid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>datname</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>relid</td>
<td>oid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phase</td>
<td>text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blds_total</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blds_scanned</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blds_vacuumed</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index_vacuum_count</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_dead_tuples</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>num_dead_tuples</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### pg_stat_progress_vacuum

View "pg_catalog.pg_stat_progress_vacuum"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>datid</td>
<td>oid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>datname</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>relid</td>
<td>oid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>phase</strong></td>
<td><strong>text</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_bcks_total</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_bcks_scanned</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_bcks_vacuumed</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index_vacuum_count</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_dead_tuples</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>num_dead_tuples</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
pg_stat_progress_vacuum

7 phases for a vacuum

initializing
scanning heap
vacuuming indexes
vacuuming heap
cleaning up indexes
truncating heap
performing final cleanup
pg_stat_progress_vacuum

- Loop through heap, possibly skipping pages
- Per-page activity
- *Serially* loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap

MAIN LOOP
pg_stat_progress_vacuum

- Loop through heap, possibly skipping pages
- Per-page activity
- *Serially* loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap

scanning heap
scanning heap
vacuuming indexes
vacuuming heap
vacuuming heap
Vacuum Process (*lazy_vacuum_rel()*):

- Loop through heap, possibly skipping blocks
- If about to run out of TID memory:
  - *Serially* loop through all indexes
  - Remove dead tuples from heap
  - Vacuum FreeSpaceMap

- Per-page activity
- *Serially* loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap

OUCH
Vacuum Process (lazy_vacuum_rel())

- Loop through heap, possibly skipping blocks
- If about to run out of TID memory:
  - *Serially* loop through all indexes
  - Remove dead tuples from heap
  - Vacuum FreeSpaceMap
- Per-page activity
- *Serially* loop through all indexes
- Remove dead tuples from heap
- Vacuum FreeSpaceMap

scanning heap
scanning heap
vacuuming indexes
vacuuming heap
scanning heap
vacuuming indexes
vacuuming heap
vacuuming heap

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
### pg_stat_progress_vacuum

View "pg_catalog.pg_stat_progress_vacuum"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>datid</td>
<td>oid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>datname</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>relid</td>
<td>oid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phase</td>
<td>text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blks_total</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blks_scanned</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blks_vacuumed</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index_vacuum_count</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_dead_tuples</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>num_dead_tuples</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### pg_stat_progress_vacuum

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap_blks_total</td>
<td>bigint</td>
<td># of blocks in table at start of vacuum</td>
</tr>
<tr>
<td>heap_blks_scanned</td>
<td>bigint</td>
<td># of table blocks scanned</td>
</tr>
<tr>
<td>heap_blks_vacuumed</td>
<td>bigint</td>
<td># of table blocks vacuumed</td>
</tr>
</tbody>
</table>
### pg_stat_progress_vacuum

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap_blks_total</td>
<td>bigint</td>
<td></td>
</tr>
<tr>
<td>heap_blks_scanned</td>
<td>bigint</td>
<td>scanning heap</td>
</tr>
<tr>
<td>heap_blks_vacuumed</td>
<td>bigint</td>
<td>vacuuming heap</td>
</tr>
</tbody>
</table>
pg_stat_progress_vacuum

index_vacuum_count | bigint

# of times indexes have been looped through (vacuuming indexes phase)

If index_vacuum_count > 0 and phase = 'scanning heap' 😨
## pg_stat_progress_vacuum

View "pg_catalog.pg_stat_progress_vacuum"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>datid</td>
<td>oid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>datname</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>relid</td>
<td>oid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phase</td>
<td>text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blks_total</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blks_scanned</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heap_blks_vacuumed</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index_vacuum_count</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_dead_tuples</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>num_dead_tuples</td>
<td>bigint</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### pg_stat_progress_vacuum

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_dead_tuples</td>
<td>bigint</td>
<td>Size of array (max # of TIDs)</td>
</tr>
<tr>
<td>num_dead_tuples</td>
<td>bigint</td>
<td>Number of remembered TIDs</td>
</tr>
</tbody>
</table>

**autovacuum_work_mem**

**maintenance_work_mem**

![TID TID TID TID TID TID ... TID TID TID](image)
pg_stat_progress_vacuum

max_dead_tuples ~= num_dead_tuples

autovacuum_work_mem
maintenance_work_mem

| TID | TID | TID | TID | TID | TID | ... | TID | TID | TID | TID | TID |

😭
Autovacuum

Two parts: launcher & worker
Autovacuum Launcher

Launcher wakes every \texttt{autovacuum\_naptime} seconds

Prioritizes databases by
- Most in need of XID freeze
- Most in need of MXID freeze
- Least recently autovacuumed, skipping any database less than \texttt{autovacuum\_naptime} ago

Multiple workers can work on a database

Check count of running autovacuums vs \texttt{autovacuum\_max\_workers}

On RDS check MaximumUsedTransactionIDs in CloudWatch

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Autovacuum Worker

Get list of heap tables & materialized views that need vacuum or analyze

Get list of TOAST tables that need vacuuming

TEMP tables are ignored (or removed)

List of tables is not prioritized in any fashion
Autovacuum Worker

For each relation
- attempt to get lock
- skip if unavailable (unless freeze is needed)
- call `vacuum()`. `vacuum()` will terminate if it blocks another process, unless aggressive.

Process work items (currently only BRIN summarize)

`vac_update_datfrozenoid()`

exit
Vacuum Cost Delay

Well documented

**Simple explanation:** once `(auto)` vacuum_cost_limit is hit, sleep for (auto) vacuum_cost_delay. Increasing limit speeds vacuum; increasing delay slows vacuum.

**Auto vacuum default:** 4-8MB/s

Don't slow vacuum too much

**On systems where writing is cheaper than reading,** set vacuum_cost_page_dirty lower than vacuum_cost_page_miss
Adaptive Autovacuum

Makes autovacuum settings more aggressive when maximum transaction ID age gets too high. Resets settings once age drops.

You can monitor via events on the instance.

Reduces risk of instance going read-only to prevent wraparound

Available in RDS Postgres 9.4+
Death-spiral

1. Table, Indexes start small
2. Vacuum can't keep up
3. Vacuum falls farther behind
4. Table, indexes get bloated
5. Bankruptcy! Need to VACUUM FULL or pg_repack

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.