Importance of PostgreSQL Vacuum Tuning

To Optimize Database Performance

PRESENTED BY

Suyog Pagare

Lead DBA – PostgreSQL

Datavail

datAvail

Suyog Pagare

Lead DBA - PostgreSQL, Datavail

- 15 years in Database Administration
- Specialties: Database Migrations, Performance tuning
- Microsoft Certified in Azure Solution Architect Expert
- AWS Certified Solutions Architect
- AWD DB specialty

Connect or follow me on LinkedIn:

https://www.linkedin.com/in/suyog-pagare-5a226918/

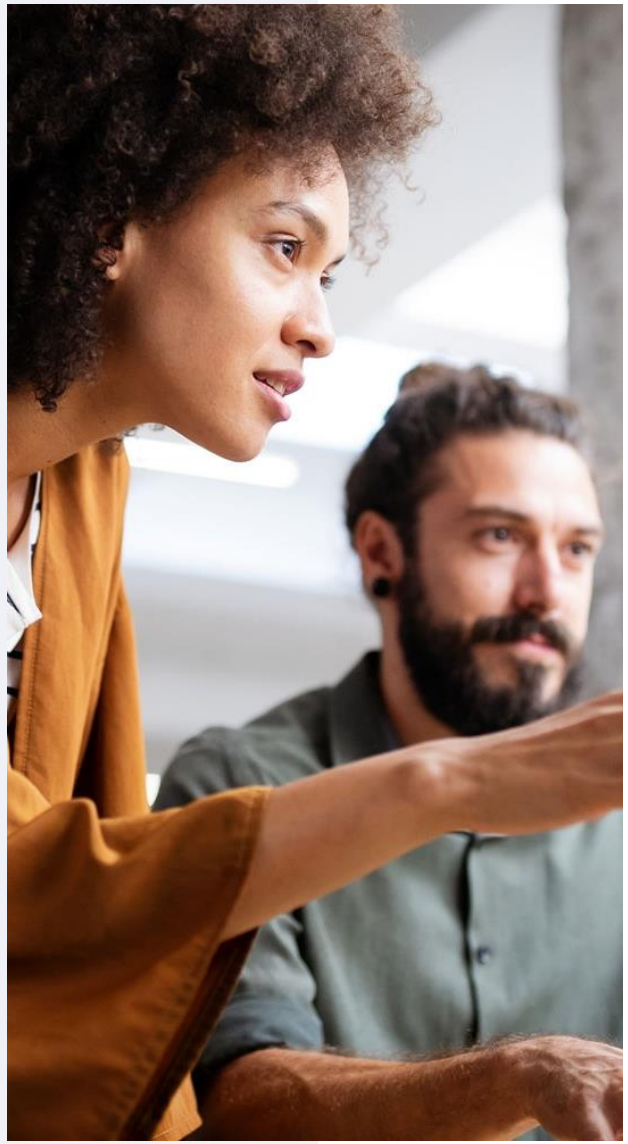
Put the PostgreSQL Pieces Together

datAvail

- Fill out your session evaluation form for a chance to win a LEGO set.

Datavail at a Glance

Delivering a superior approach to leverage data through application of a tech-enabled global delivery model & deep specialization in databases, data management, and application services.



16+

Years

building and operating mission critical data and application systems

$25M

Invested

in IP that improves the service experience and drives efficiency

datAvail ServiceBoost®

datAvail TechBoost®

1,000+

Employees

staffed 24x7, resolving over 2,000,000 incidents per year

Agenda

- Database Bloat
- What is Vacuum and Autovacuum
- Best Practices for Tuning Autovacuum
- Dead tuples Not Yet Removable
- Pg_repack
- Other DB Parameter Tuning

Introduction to PostgreSQL Maintenance

datAvail

Overview

- PostgreSQL is an advanced, open-source relational database system.
- Like any database, PostgreSQL requires regular maintenance to perform optimally.

Key Maintenance Tasks

- Autovacuum
- Manual Vacuum
- Analyze
- Index Maintenance

Understanding Database Bloat

datAvail

- Definition: Database bloat refers to the unnecessary consumption of disk space in a database due to outdated or deleted data, and inefficiencies in storage.
- Causes of Bloat
  - Frequent updates and deletes: PostgreSQL uses a multi-version concurrency control (MVCC) system, which can leave behind dead tuples.
  - No Vacuum.
  - Inefficient use of indexes: Unused or poorly maintained indexes can also contribute to bloat.
- Symptoms of Database Bloat
  - Increase Disk Usage
  - Degraded Performance
  - Longer Backup and Restore Times
- How to Identify Bloat
  - Pgstattuple extension
  - Pg_stat_user_tables: To monitor dead tuples and live tuple counts

Query to Find Bloat

datAvail

```
SELECT
   c.oid,
   n.nspname || '.' || c.relname AS ObjectName,
   pg_stat_get_live_tuples(c.oid) AS LiveTuples,
   pg_stat_get_dead_tuples(c.oid) AS DeadTuples,
   CASE
      WHEN pg_stat_get_live_tuples(c.oid) > 0
      THEN (pg_stat_get_dead_tuples(c.oid)::numeric / pg_stat_get_live_tuples(c.oid))::numeric(18,3) * 100
      ELSE 0
   END AS DeadtupleRatio,
   pg_table_size(n.nspname || '.' || c.relname) AS tbl_sz,
   pg_size_pretty(pg_table_size(n.nspname || '.' || c.relname)) AS tbl_sz_prty,
   pg_size_pretty(pg_indexes_size(n.nspname || '.' || c.relname)) AS tbl_idx_sz_prty,
   (SELECT count(1) FROM pg_indexes i WHERE i.schemaname || '.' || i.tablename = n.nspname || '.' || c.relname) AS idx_cnt,
   CASE
      WHEN pg_table_size(n.nspname || '.' || c.relname) > 0
      THEN (pg_indexes_size(n.nspname || '.' || c.relname)::numeric / (pg_indexes_size(n.nspname || '.' || c.relname) + pg_table_size(n.nspname || '.' || c.relname))) * 100
      ELSE 0
   END AS idx_tbl_ratio,
   relhastriggers
FROM
   pg_class c
JOIN
   pg_catalog.pg_namespace n
ON
   c.relnamespace = n.oid
WHERE
   c.relkind = 'r'
   AND n.nspname NOT LIKE 'pg\_%'
ORDER BY
   pg_table_size(n.nspname || '.' || c.relname) DESC
LIMIT 30;
```

Consequences of Ignoring Bloat:

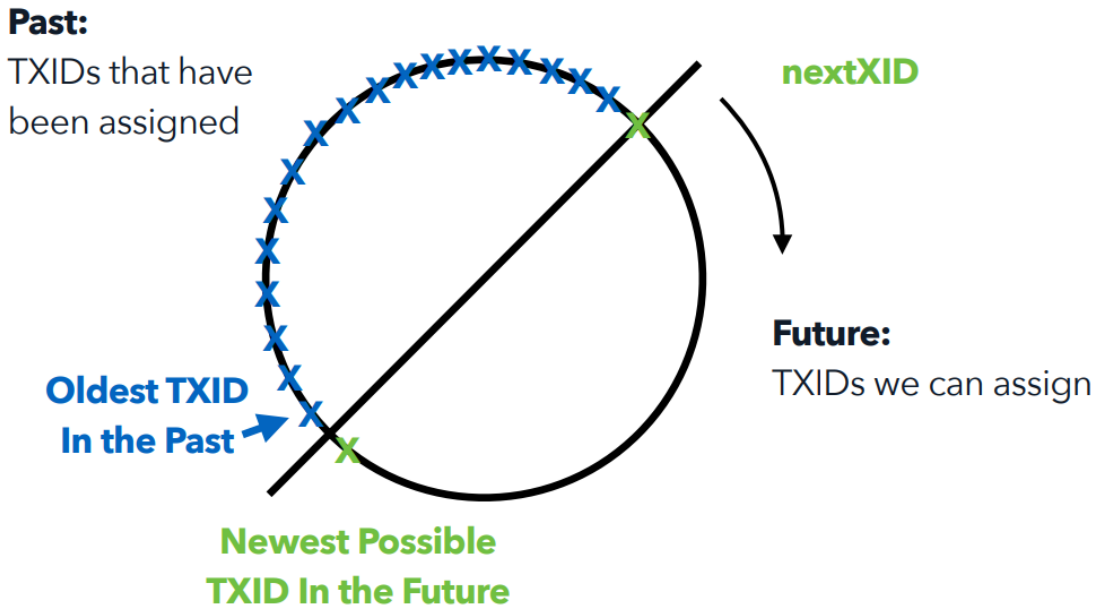
datAvail

- Performance Degradation
- High Storage Costs
- Operations Challenges
- Transaction Id Wraparound

Transaction ID Wraparound

datAvail

- What is Transaction ID (Xid)
- Wraparound Issue



Preventing and Managing Bloat

datAvail

- Vacuum….!!!
- It removes dead tuples
- Reclaim storage back to the OS
- Maintain visibility map

Vacuum

datAvail

- Types of Vacuum
  - Vacuum
  - Vacuum Full

Syntax:

Vacuum verbose table_name;

vacuum (index_cleanup true) table_name;

- Analyze

```
VACUUM [ ( option [, ...] ) ] [ table_and_columns [, ...] ]
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] [ ANALYZE ] [ table_and_columns [, ...] ]

where option can be one of:

    FULL [ boolean ]
    FREEZE [ boolean ]
    VERBOSE [ boolean ]
    ANALYZE [ boolean ]
    DISABLE_PAGE_SKIPPING [ boolean ]
    SKIP_LOCKED [ boolean ]
    INDEX_CLEANUP { AUTO | ON | OFF }
    PROCESS_MAIN [ boolean ]
    PROCESS_TOAST [ boolean ]
    TRUNCATE [ boolean ]
    PARALLEL integer
    SKIP_DATABASE_STATS [ boolean ]
    ONLY_DATABASE_STATS [ boolean ]
    BUFFER_USAGE_LIMIT size

and table_and_columns is:

    table_name [ ( column_name [, ...] ) ]
```

Autovacuum

datAvail

- Automate solution to execute Vacuum
- No manual intervention needed
- Triggers it based on the defined thresholds
- Does not reclaim the free space

Understanding Autovacuum and Autoanalyze

datAvail

Benefits of Autovacuum

- Automates Maintenance
- Prevents Data Corruption
- Optimize Performance

Formula:

Autovacuum = autovacuum_vacuum_scale_factor * number of rows + autovacuum_vacuum_threshold

Autoanalyze = autovacuum_analyze_scale_factor * number of rows + autovacuum_analyze_threshold

- Check status using table: pg_stat_vacuum_progress

Tuning of Autovacuum Parameters

datAvail

- Important Parameters:
  - Autovacuum_vacuum_threshold/ Autovacuum_scale_factor
  - Autovacuum_analyze_threshold/ Autoanalyze_scale_factor
  - Autovacuum_max_workers
  - Autovacuum_naptime
  - Autovacuum_cost_delay
  - Max_worker_processes
  - Max_parallel_workers
  - Max_parallel_maintenance_workers
  - Default_stats_target

Best Practices

datAvail

- Setup a cronjob daily to execute vacuum based of % of table bloat
- Use vacuumdb utility to run vacuum with parallel jobs
- Setup aggressive Autovacuum for specific tables
- Monitor vacuum process
- For faster operation, set high maintenance_work_mem and execute manual vacuum for faster execution

Dead tuple Not Yet Removed

datAvail

Sometimes even after manual vacuum, dead tuples are not removed.

- Long running transaction
- Abandoned replication slot
- DDL transaction is in process
- Orphan Prepared transactions

```
SELECT gid, prepared, owner, database, transaction AS xmin
FROM pg_prepared_xacts
ORDER BY age(transaction) DESC;
```

- Standby with hot_standby_feedback=on

```
SELECT application_name, client_addr, backend_xmin
FROM pg_stat_replication
ORDER BY age(backend_xmin) DESC;
```

Pg_repack

datAvail

- pg_repack is a PostgreSQL extension used for reclaiming space and optimizing database performance without requiring a full table lock.
- Purpose: Efficiently removes table and index bloat, reducing storage footprint and improving query speed.
- **Online Operation:** Performs maintenance tasks without significant downtime, allowing for continuous database availability.
- **Bloat Reduction:** Reorganizes tables and indexes, eliminating wasted space caused by deleted or updated tuples.
- **Pre-checks:** Ensure sufficient disk space and backups before running.

How to Run pg_repack

datAvail

- Install pg_repack
- Add pg_repack to shared_preload_libraries
- Create extension pg_repack on database
- Run command to execute pg_repack
  - pg_repack -d mydb

  You can execute pg_repack for specific table as well
  - pg_repack -t mytable -d mydb

Other DB Parameter Tuning

datAvail

- https://pgtune.leopard.in.ua/
- https://pgconfigurator.cybertec-postgresql.com/
- https://www.pgconfig.org
- https://www.dbtune.com/

Put the PostgreSQL Pieces Together

datAvail

- Fill out your session evaluation form for a chance to win a LEGO set.


Q&A

Thank You

Suyog Pagare

Lead DBA: PostgreSQL Services

Suyog.Pagare@datavail.com

+1 720-756-4881

datAvail