Scenarios That Trigger Autovacuum in PostgreSQL

PostgreSQL is widely known for its Multi-Version Concurrency Control (MVCC) model, which allows multiple transactions to occur simultaneously without interfering with each other.

However, one side effect of MVCC is the creation of dead tuples—old versions of data rows that are no longer needed but still occupy space.

Dead tuples also lead to a phenomenon known as table bloat, which refers to the excessive unused space in a table caused by dead tuples that haven't been cleaned up, resulting in inefficient storage and reduced performance

To address the issues of dead tuples and table bloat, autovacuum comes into play. It's an automatic process designed to clean up these dead tuples and maintain optimal database performance.

In this blog, we will explore the main situations when autovacuum should run

Types of Vacuuming Processes

In PostgreSQL, there are two main ways to remove dead tuples or removing table bloats. and keep the database optimized: autovacuum and manual vacuum. Let's break down the differences between them in simple terms:

Autovacuum

Autovacuum is an automatic process that PostgreSQL runs in the background to clean up dead tuples. It works without user intervention and is designed to run regularly, checking for dead tuples and cleaning them up when needed. The autovacuum process helps prevent your database from becoming bloated, which could slow down performance.

The key benefits of autovacuum are:

- Automatic: It runs on its own without requiring you to manually trigger it.
- Background Task: It doesn't block other database operations, so it can run without interrupting your work.
- Configurable: You can set the threshold for when autovacuum should run, based on factors like table size and how much data is updated.

However, autovacuum may not always run as frequently as you need or may be blocked by long-running transactions. It's also possible that it might not clean up dead tuples fast enough in some situations, which is where manual vacuum comes in.

Manual Vacuum

Manual vacuum is when you, as the database administrator, decide to run the vacuum process yourself. This can be useful when you want more control over the vacuuming process or need to clean up a large amount of dead tuples quickly.

The key benefits of manual vacuum are:

- Control: You can run it exactly when you need it, such as during maintenance windows or after large data updates.
- Immediate Cleanup: If autovacuum hasn't run quickly enough or if you need to reclaim space right away, manual vacuum lets you take care of it immediately.

The downside of manual vacuum is that it can be more resource-intensive. Running vacuum manually during peak usage times could lead to performance issues since it locks tables while cleaning them. It also requires you to remember to run it, whereas autovacuum works automatically.

Which One to Use?

Autovacuum is ideal for everyday maintenance and handling routine cleanup automatically. It helps keep the database running smoothly without constant intervention.

Manual vacuum is better for special cases where you need immediate action, like after a huge data update, when autovacuum might not run quickly enough, or when you want to optimize a table that hasn't been cleaned in a while.

In general, both processes are important, and they complement each other. Autovacuum should be configured properly to handle most of the workload, but manual vacuum can be helpful for specific situations that need more immediate attention.

When Should Autovacuum Be Triggered?

Well technically there are many cases for instance here is the list

When relfrozenxid is more than autovacuum_freeze_max_age transactions old

relfrozenxid is a marker that tells PostgreSQL the oldest transaction ID in a table that hasn't been frozen yet. Freezing is a process that helps prevent transaction ID wraparound issues by marking old transaction IDs as safe to reuse.

Here's what happens when relfrozenxid is more than autovacuum_freeze_max_age transactions old

- PostgreSQL keeps track of how old the transaction IDs are in each table. If the oldest transaction ID (relfrozenxid) is older than a certain limit (autovacuum_freeze_max_age), it means the table needs attention.
- When this age limit is exceeded, PostgreSQL's autovacuum process is triggered to run on that table. The goal is to freeze old transaction IDs, making them safe to reuse and preventing any risk of transaction ID wraparound.

When relfrozenxid is more than vacuum_freeze_table_age transactions old

Bit similar to the above but this condition is more of a guideline for when you should consider running a manual VACUUM operation. vacuum_freeze_table_age is a setting that indicates when a table's relfrozenxid is getting old enough that you should proactively vacuum the table.

When the number of tuples obsoleted since the last VACUUM exceeds the vacuum threshold

When we update or delete a row, the old version of that row becomes obsolete because it's no longer the current version. PostgreSQL has a setting that determines when it should automatically run the VACUUM process.

This is based on how many obsolete rows (or tuples) have accumulated since the last time VACUUM was run.

So, when the number of obsolete rows since the last VACUUM goes over this threshold, PostgreSQL decides it's time to run VACUUM again.

Two key parameters based on them we decide the autovacumm threshold for a table are

- autovacuum_vacuum_threshold (50 default value)
- autovacuum_vacuum_scale_factor (0.2 default value)

```
autovacuum_threshold = autovacuum_vacuum_threshold + (Rows inside the table *
autovacuum_vacuum_scale_factor)Copy to Clipboard
```

So if you have 100 rows inside the table autovacuum will trigger when there are 70+ dead tuples inside the table interesting yeah! autovacuum threshold=50+(100×0.2)=50+20=70

When the number of tuples inserted since the last VACUUM exceeds the insert threshold

PostgreSQL has a setting that determines when it should automatically run the VACUUM process based on the number of new rows inserted since the last time VACUUM was run

So, when the number of new rows added to a table since the last VACUUM exceeds this insert threshold, PostgreSQL decides it's time to run VACUUM again.

The formula for threshold

```
insert_threshold = autovacuum_vacuum_threshold + (Rows inside the table *
autovacuum_vacuum_scale_factor)Copy to Clipboard
```

If we have only 100 tuples then threshold would be 70 and if we insert 70 more records inside the table then autovacuum process will be triggered

When the total number of tuples inserted, updated, or deleted since the last ANALYZE exceeds the analyze threshold

ANALYZE is the PostgreSQL operation that collects statistics about the contents of tables. These statistics help the query planner make informed decisions about the most efficient way to execute queries.

PostgreSQL uses a threshold to decide when to automatically run the ANALYZE operation. This threshold is based on the number of tuples that have been inserted, updated, or deleted since the last time ANALYZE was run.

The formula for threshold

```
analyze_threshold = autovacuum_analyze_threshold + (Rows inside the table *
autovacuum_analyze_scale_factor)Copy to Clipboard
```

- autovacuum_analyze_threshold: 50 (default value)
- autovacuum_analyze_scale_factor: 0.1 (10%)
- number_of_tuples: 200 (total rows in the table)

analyze_threshold = 50 + (0.1 × 200) = 70

The analyze threshold is 70. This means that if there are at least 70 updates or deletes to the tuples in the table, PostgreSQL's autovacuum will trigger an analysis to update the statistics for the query planner

The number of transactions that have passed since the last aggressive vacuum exceeds vacuum_freeze_table_age minus vacuum_freeze_min_age.

PostgreSQL tracks how many transactions have occurred since the last aggressive vacuum. If this number exceeds vacuum_freeze_table_age minus vacuum_freeze_min_age, it signals that it's time for another aggressive vacuum.

```
vacuum_freeze_table_age – vacuum_freeze_min_age 150,000,000 – 50,000,000 = 100,000,000Copy to Clipboard
```

Suppose the last aggressive vacuum was performed when the transaction ID was at 1,000,000,000. Now, the transaction ID has reached 1,100,000,000. Since 100 million transactions have passed since the last aggressive vacuum, PostgreSQL will trigger another aggressive vacuum to ensure that no transaction IDs are getting too old.

If the age of the oldest multixact ID (tracked by relminmxid) exceeds autovacuum_multixact_freeze_max_age

In PostgreSQL, a multixact ID is used to handle scenarios where multiple transactions need to lock the same row simultaneously. This is common in situations involving shared locks. If the age of the oldest multixact ID (relminmxid) exceeds autovacuum_multixact_freeze_max_age, PostgreSQL will trigger an autovacuum process.

If the storage occupied by multixact members exceeds 2GB

PostgreSQL uses a special area to store multixact members. If this storage exceeds 2GB, it can be a sign that there are many multixacts being used, potentially due to long-running transactions or high concurrency.

If multixact-age is greater than autovacuum_multixact_freeze_max_age.

multixact-age refers to the age of the oldest multixact ID in the database When the multixact-age exceeds autovacuum_multixact_freeze_max_age, it indicates that the oldest multixact ID in the database has reached an age where PostgreSQL needs to take action to prevent potential issues

Additional Notes

- If there are old prepared transactions or long-running open transactions causing unvacuumed states, it may prompt an autovacuum. Ending these transactions is necessary to prevent blockage.
- If autovacuum fails to clear old XIDs or MXIDs, the system will issue warnings, indicating that a manual VACUUM is necessary. The absence of such maintenance can lead to forced autovacuum events.
- Autovacuum will process table partitions, but it does not run ANALYZE on partitioned tables automatically.
- Autovacuum does not process temporary tables, so manual vacuum and analyze operations should be performed

In conclusion, monitoring autovacuum and understanding when it should trigger is essential for maintaining the health and performance of your PostgreSQL database. While autovacuum runs automatically to clean up dead tuples, there may be times when it gets blocked or doesn't run as expected. By regularly checking autovacuum stats and addressing any issues, you can prevent table bloat and ensure your database continues to run efficiently. Remember, both autovacuum and manual vacuuming play important roles in database maintenance, and a well-maintained database will lead to better performance and reliability.