VACUUM FULL in PostgreSQL – What you need to be mindful of

If you have worked with PostgreSQL for a while, you have probably come across the command VACUUM FULL. At first glance, it might seem like a silver bullet for reclaiming disk space and optimizing tables. After all, who would not want to tidy things up and make their database more efficient, right?

But here is the thing: while VACUUM FULL can be useful in some situations, it is not the hero it might seem. In fact, it can cause more problems than it solves if you are not careful.

Table of Contents

What Does VACUUM FULL Actually Do?

PostgreSQL uses something called Multi-Version Concurrency Control (MVCC). Without getting too technical, MVCC keeps multiple versions of rows around to handle updates and deletes efficiently. These older versions of rows – called dead tuples – are cleaned up by a process called vacuuming.

- A regular VACUUM removes those dead tuples so the space can be reused.
- VACUUM FULL, however, goes further. It rewrites the entire table to remove dead space completely. It also rebuilds all the indexes on the table. Essentially, it is like dumping all your clothes out of the closet, refolding everything, and putting it back in neatly.
Sounds great, right? So, why not use it all the time?

When Should You Actually Use VACUUM FULL?

There are a few very specific situations where VACUUM FULL makes sense:

After Massive Deletions

Imagine you delete millions of rows from a table. Regular vacuuming might not reclaim that disk space immediately, and the table could still look bloated. In this case, VACUUM FULL can shrink the table and give you that disk space back.

Disk Space Crunch

If your database server is running out of disk space and you need to reclaim it fast, VACUUM FULL can help (though it is still not ideal—more on that later).

Post-Migration Cleanup

If you have migrated a large table or reorganized your data, VACUUM FULL can clean things up during planned downtime.

Outside of these scenarios, though, VACUUM FULL is usually not your best option. Why? Let us break it down.

The Downsides of VACUUM FULL

VACUUM FULL is not a gentle operation – it is more like the nuclear option. Here are the key reasons to think twice before running it:

1. It Locks the Table

This is the big one. VACUUM FULL acquires an exclusive lock on the table, meaning:

- No one can read or write to the table while it is running.
- If your application relies on that table, it will just … stop.
  For high-traffic production systems, this is a nightmare. Imagine locking your sales database during peak hours. Your users (and probably your boss) will not be happy.

2. It Needs a Lot of Disk Space

Here is something many people do not realize: VACUUM FULL needs enough disk space to hold a second copy of your table while it rewrites everything. For large tables, this can be a problem:

- If your table is 500 GB, you need another 500 GB free to run VACUUM FULL.
- If you do not have that space, the operation will fail, and you might end up worse off than before.

3. It Takes a Long Time

VACUUM FULL is not exactly speedy. It rewrites the entire table and all its indexes, which can take hours for large datasets. During this time, the table is locked (remember point 1), and your users are waiting.

4. Indexes Add Overhead

Every index on the table has to be rebuilt during VACUUM FULL. While this ensures your data is optimized, it also adds extra CPU and I/O overhead. If you have a table with lots of indexes, this can turn a slow operation into an agonizingly slow one.

5. It Hogs System Resources

Because VACUUM FULL rewrites everything, it is very I/O-intensive. On a busy server, this can cause performance issues for other queries and operations. You might be fixing one problem but creating five new ones.

6. It Is a Band-Aid, Not a Cure

Finally, let us address the root issue: bloat. If your database is bloated, VACUUM FULL will fix it temporarily. But if you do not address the underlying cause – like improper autovacuum settings or inefficient queries – the bloat will just come back. Think of it as treating the symptom, not the disease.

Why Does Bloat Happen?

Understanding bloat is key to avoiding it. PostgreSQL's MVCC model creates dead tuples during:

- **Deletes:** Old rows are marked as dead but not removed.
- **Updates:** A new version of the row is written, but the old version stays behind.

These dead tuples pile up until a vacuum process removes them. However, if vacuuming does not happen often enough, or your table sees heavy updates and deletes, bloat can spiral out of control.

How to Prevent and Manage Bloat (without VACUUM FULL)

The good news is that you can manage bloat effectively without resorting to VACUUM FULL. Here are some strategies:

1. Tune Autovacuum Settings

Autovacuum is PostgreSQL's built-in process for cleaning up dead tuples. By default, it might not run frequently enough for high-churn tables. Adjusting settings like autovacuum_vacuum_scale_factor and autovacuum_vacuum_threshold can make a big difference.

2. Use Partitioning

Partitioning breaks large tables into smaller, more manageable pieces. Each partition acts as its own table, which makes vacuuming faster and reduces bloat overall.

3. Monitor and Analyze Bloat

Use tools like pgstattuple or extensions like pg_bloat_check to monitor which tables are bloated. Knowing where the problem is helps you target the right solution.

4. Consider CLUSTER or REINDEX

For tables with performance issues but less bloat, CLUSTER (to reorganize data) or REINDEX (to rebuild indexes) can be effective alternatives to VACUUM FULL.

5. Plan Maintenance Windows

If you absolutely must run VACUUM FULL, schedule it during low-traffic periods to minimize the impact.

Alternatives to VACUUM FULL

When faced with bloat or performance issues, these alternatives can often achieve the same result with fewer downsides:

1. Regular VACUUM

Standard VACUUM does not lock the table and is sufficient for most cases. It reclaims space for reuse but does not shrink the table size.

2. Autovacuum

Ensure autovacuum is enabled and properly configured. It is your best defense against bloat.

3. Logical Replication

For severely bloated databases, consider logical replication. This allows you to migrate your data to a new table or database without locking the original table.

When VACUUM FULL Makes Sense

Let me be clear: VACUUM FULL is not evil. It is just a tool, and like any tool, it works best in specific situations:

- You have deleted 80% of a table's rows and need to reclaim disk space urgently.
- Disk space is critically low, and other options have failed.
- You are performing a post-migration cleanup during planned downtime.

If you decide to use it, go in with your eyes open and understand the trade-offs.

Wrapping Up

VACUUM FULL is often misunderstood. While it can fix extreme cases of bloat, its downsides – table locking, high resource usage, and time consumption – make it unsuitable for most scenarios. Instead, focus on proactive maintenance:

- Tune your autovacuum settings.
- Use partitioning and indexing wisely.
- Monitor bloat and act before it becomes a problem.

By taking these steps, you can keep your PostgreSQL database healthy and performant – no VACUUM FULL required.

Need help optimizing your PostgreSQL database?
At Stormatics, we specialize in performance tuning, bloat management, and high-availability solutions for PostgreSQL. Let us help you build a database that runs like clockwork.