PERCONA®

Hidden Dangers of Extensions

Jobin Augustine
PostgreSQL Escalation Specialist
Tech Lead for PostgreSQL

Courtesy : Marco Slot , Frank Patchot

Common perceptions and Expectations

- Adds features and makes PostgreSQL cool.
- PostgreSQL's built-in features works as of documented
- No considerable performance degradation
- Not much resource consumption.

430+ extensions available on pgxn.org

joelonsql / PostgreSQL-EXTENSIONs.md

Last active 2 days ago • Report abuse

Subscribe | Star 333 | Fork 60

Code | Revisions 25 | Stars 332 | Forks 60

Embed | <script src="https:/ | Download ZIP

1000+ PostgreSQL EXTENSIONs

PostgreSQL-EXTENSIONs.md | Raw

🗺️ 🐘 1000+ PostgreSQL EXTENSIONs

This is a list of URLs to PostgreSQL EXTENSION repos, listed in alphabetical order of parent repo, with active forks listed under each parent.

⭐ >= 10 stars
⭐⭐ >= 100 stars
⭐⭐⭐ >= 1000 stars
Numbers of stars might not be up-to-date.

If some repo is missing, please write a comment with the url.

https://gist.github.com/joelonsql/e5aa27f8cc9bd22b8999b7de8aee9d47

- Access Methods
- Aggregate Functions
- Data Types
- Dictionaries
- Foreign Data Wrappers
- Procedural Languages
- Spatial and Geographic Objects

Complexity Varies:

Peripheral Extensions - adds user functions, simple types, FDWs

...

Extensions which need to implement many hooks and change the way PostgreSQL Works

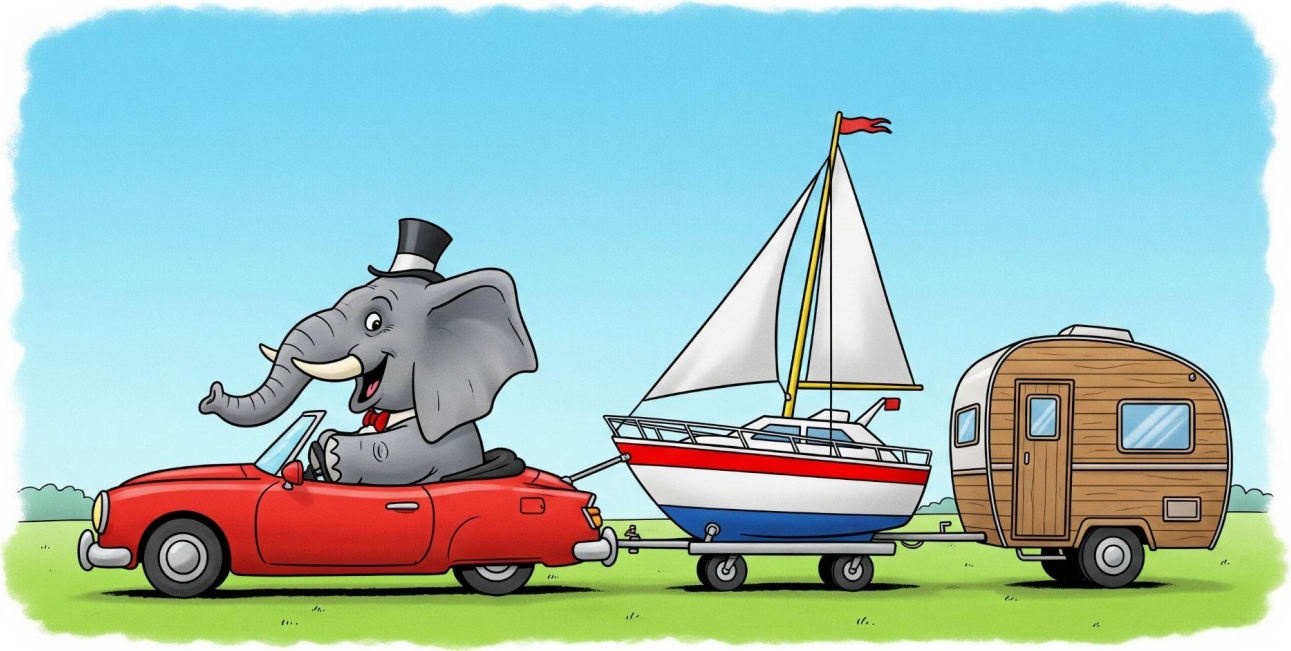
Loading libraries

- Shared_preload_libraries
- Session_preload_libraries
- LOAD

Hooks

- Shared library files are loaded into memory
- Internal C-level hook system
  - A network of function pointers scattered throughout the PostgreSQL source code
- Allows libraries to intercept critical internal operations
  - query planning, execution, client authentication and transaction management
- preloading and hooking enables extensions to modify the very execution flow of the database

Performance drag- Demo



Alibaba Cloud

Cart Log In Free Account

Community Blog Events Webinars Tutorials Forum

Search

Community > Blog > PostgreSQL v12: How pg_stat_statements Causes High-concurrency Performance Issues

PostgreSQL v12: How pg_stat_statements Causes High-concurrency Performance Issues

digoal May 25, 2021 4,742 0

In this article, the author discusses how enabling PostgreSQL pg_stat_statements in PG v12 causes high-concurrency performance issues.

digoal

285 posts | 26 followers

Follow

Related Products

Enable pg_stat_statements

```
pgbench -M prepared -n -r -P 1 -c 104 -j 104 -T 120 -S

transaction type: <builtin: select only>
scaling factor: 1000
query mode: prepared
number of clients: 104
number of threads: 104
duration: 120 s
number of transactions actually processed: 67468127
latency average = 0.185 ms
latency stddev = 0.348 ms
tps = 562224.744452 (including connections establishing)
tps = 562300.147713 (excluding connections establishing)
statement latencies in milliseconds:
         0.001  \set aid random(1, 100000 * :scale)
         0.185  SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
```

Disable pg_stat_statements

```
pgbench -M prepared -n -r -P 1 -c 104 -j 104 -T 120 -S

transaction type: <builtin: select only>
scaling factor: 1000
query mode: prepared
number of clients: 104
number of threads: 104
duration: 120 s
number of transactions actually processed: 187563515
latency average = 0.066 ms
latency stddev = 0.014 ms
tps = 1562993.591525 (including connections establishing)
tps = 1563258.811725 (excluding connections establishing)
statement latencies in milliseconds:
         0.001  \set aid random(1, 100000 * :scale)
         0.065  SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
```

Nikolay Samokhvalov • 1st

Let's make *your* Postgres healthy and awesome: nik@postgres.ai // I sta...

1yr • Edited •

First time I see 3 million TPS on a single machine with #PostgreSQL!! A SELECT-only pgbench experiment conducted by the PostgresAI bot https://lnkd.in/g_f_r9rc.

Anarchy in the Database: A Survey and Evaluation of Database Management System Extensibility

	PostgreSQL
User-Defined Functions	**Yes** (408)
User-defined Types	**Yes** (139)
Utility Commands	**Yes** (43)
Parser Modifications	No
Query Processing	**Yes** (46)
Storage Managers	**Yes** (44)
Index Access Methods	**Yes** (67)
Client Authentication	**Yes** (17)

	PostgreSQL
Adding Components	Yes
Overriding Components	Yes
State Modification	All state
Isolation/Security	None
Background Workers	Yes
Memory Allocation	Yes
Configuration Options	Yes
Source Code	Yes
Programming Languages	C, C++, Rust
Installation Interface	SQL, configs
Build & Test Tooling	Both
Package Manager	Yes (community)

https://www.vldb.org/pvldb/vol18/p1962-kim.pdf
https://www.linkedin.com/pulse/postgresql-extensions-anarchy-database-vldb-paper-franck-pachot-mjo4e/

Major areas

- Lacking Safety Mechanisms
  - highly permissive, low-level API through the database's C code (No restrictions on what can be achieved)
- Extensions can interfere with each other
  - Loading order matters
  - unpredictable and non-deterministic outcomes
- No sandboxing of execution
  - malicious extension to crash the server, corrupt data, or compromise security

- Compatibility & Interoperability Issues
  - PostgreSQL extension pairs can be incompatible, potentially leading to errors, server crashes, or inaccurate outcomes when used together.
- No built-in system for extension interoperability
- Hook chaining is fragile.
  - failures frequently occur when hooks are overwritten or not appropriately chained

Security Risks

- Extensions can bypass PostgreSQL's permission system
  - Full access to your data.
- Extension can read user credentials (check_password_hook)
- Extensions are not developed, tested or audited by wider community
- Extensions can link to external libraries

APIs

- Nothing to offer stable APIs
  - Many extension APIs are essentially internal C functions and structures, not stable or versioned interfaces
- Extension upgrades become tricky (ALTER EXTENSION … UPDATE)
- Duplicate and alter significant portions of PostgreSQL's source code
  - For example, pg_hint_plan uses a hook for the whole query planner and must fork all the related code

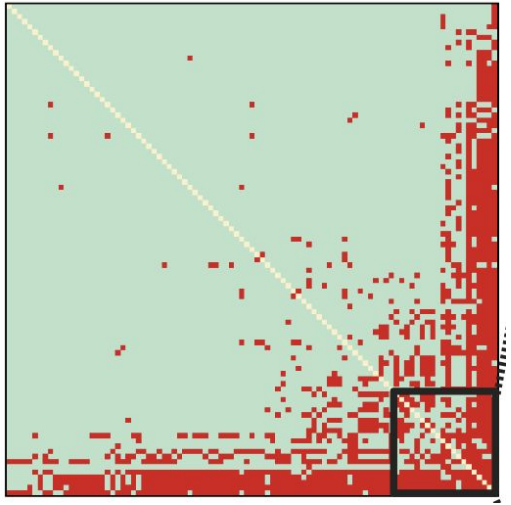
Order of loading

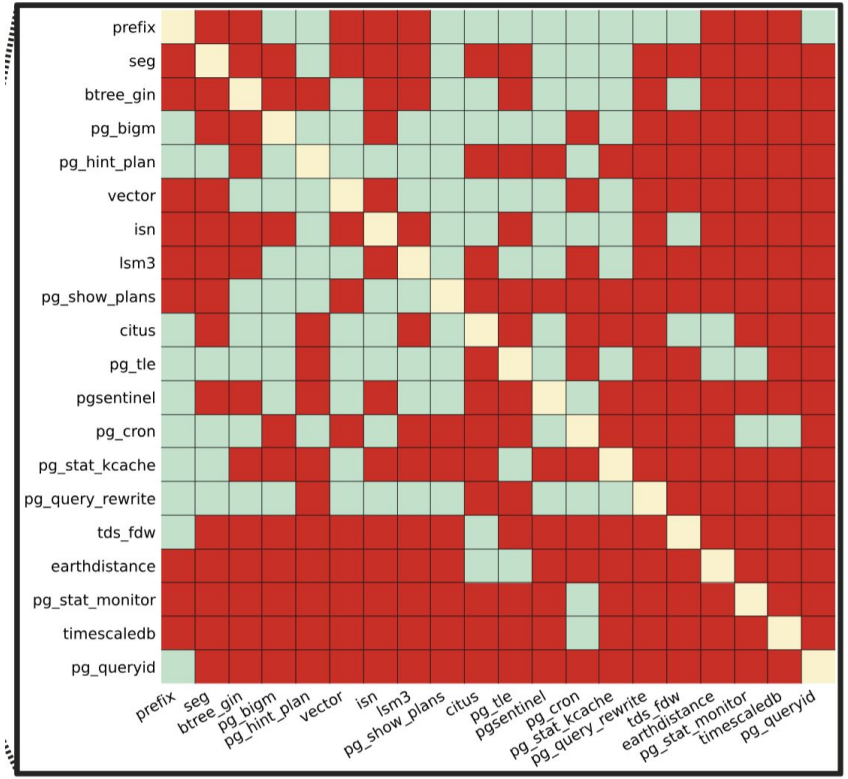
shared_preload_libraries = ‘pg_stat_statements,auto_explain,pg_qualstats,hypopg,pgstattuple,pg_stat_kcache,pgfincore,pg_prewarm,pg_wait_sampling,pg_buffercache,pg_repack,btree_gist,vector,pglogical’

- The extension loaded last will be called first for hook implementation
- Extension can chain the hooks and cause unpredictable behaviour
  - & Crashes if you are lucky!

Interoperability compatibility

https://www.vldb.org/pvldb/vol18/p1962-kim.pdf





- 16.8% of extension pairs failed to work together
- Of these tests, 255 of 380 (67.1%) fail and about a quarter (29.2%) of those fail due to brittle test cases

Testing is near impossible

- Most PostgreSQL extensions use the built-in testing harness (pg_regress)
  - Works on string comparison.
  - Not really suitable for testing extensions
- Hundreds of extensions
  - If there are 100 extensions, there could be 100! ways to load.

$$9.3 \times 10^{157}$$

Operational Challenges

- One of the Biggest hurdle in PostgreSQL upgrade
- Architecture Lock-in
  - Eg: pglogical

Outages

- Major cause of Memory leaks, Segmentation faults
- Malfunction causes PostgreSQL process to crashes

Summary

- Add custom code throughout the PostgreSQL and alter its behaviour
- Call any PG function and modify global variables
- Conflict with other extensions
- Crash the database
- Corrupt the memory and storage
- Leak memory
- Break PostgreSQL features
- Cause severe performance degradation
- Cause security problems.
- Vendor lock ins
- Unmaintainable systems if extensions are not actively developed anymore
- Upgrade failures
- Can link external libraries and additional processes.
- Extensions can implement APIs and network communications.

Recommendations

Recommendations

- Have a critical view of extensions used
  - Especially those which implement hooks
  - Those which are not part of contrib modules
  - Those which are not community in general recommends
- Remove extensions which are not regularly used
  - Don’t leave the extension in the system
- Security team should be involved. Audit the extension code.
  - Avoid extensions which are not widely used.

Additional Reference

- https://www.pgevents.ca/events/pgconfdev2025/schedule/session/410-the-trouble-with-extensions/
- https://www.youtube.com/watch?v=0dyBfg-By80 (Marco Slot)

Thank You

Q & A