Anarchy in the Database: A Survey and Evaluation of DBMS Extensibility

Abigale Kim[1], Marco Slot[2], David G. Andersen[3], Andrew Pavlo[3]
[1]University of Wisconsin—Madison, [2]Snowflake, [3]Carnegie Mellon University

Carnegie Mellon Database Group

Motivation

Extensions, or user-provided logic that augments a DBMS's behavior, allow developers to support more use cases without writing new systems from scratch. Many well-known DBMSs (PostgreSQL, MySQL, SQLite, DuckDB, Redis, SQL Server, Oracle) have extensions. However, suboptimal extensibility design decisions cause unexpected errors (crashes, incorrect results) and bad software engineering practices.

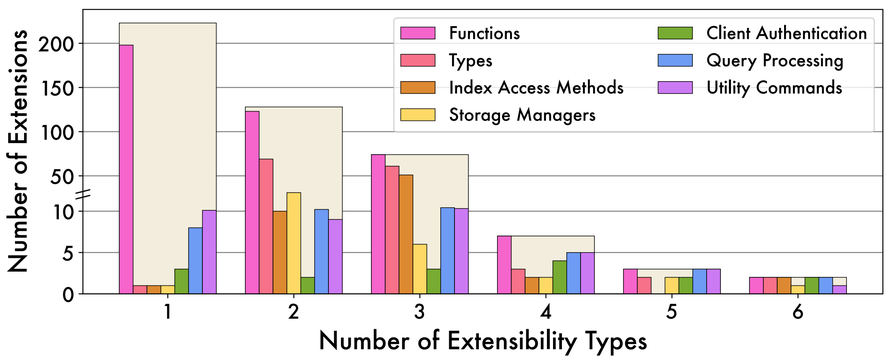
Taxonomy and Survey

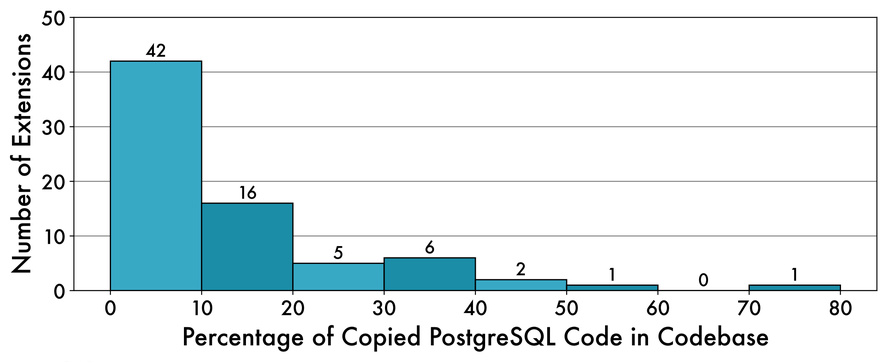
- Interfaces: adding to vs. overriding components of the DBMS
- State modification: database, system, extension (extn), ephemeral (eph)
- System components: Background workers (BW), memory allocation (MA), configuration options (CO), source code (SC)

	PostgreSQL	MySQL	MariaDB	SQLite	Redis	DuckDB
Interfaces	Both	Both	Both	Both	Adding	Both
State Modification	All state	All state	All state	All state	Extn + eph	All state
Isolation / Security	None	Low	Low	Medium	High	Low
System Components	All	All	All	MA, SC	MA, CO	MA, CO, SC
Languages	C, C++, Rust	C++	C++	C, Rust	C	C++
Installation	SQL, configs	SQL	SQL	SQL	SQL, configs	SQL
Build/Test Tooling	Both	Testing	Testing	Both	None	Both
Package Manager	Yes	No	No	Yes	No	Yes

PostgreSQL Extension Analysis

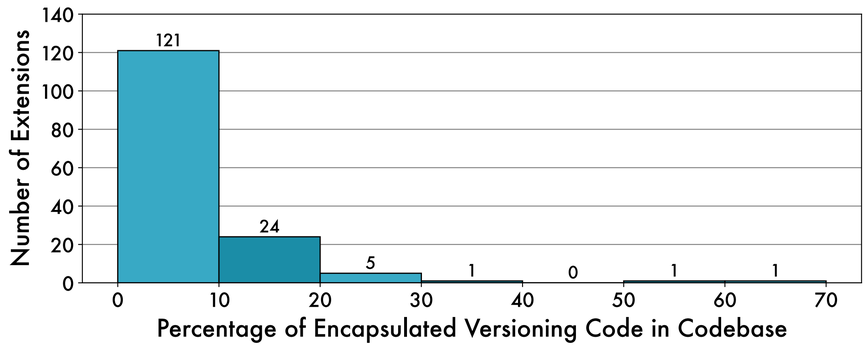
- ExtAnalyzer! tests compatibility between extensions
- Collects information about extensions
  - Types of extensibility and system components used
  - Duplicate source code analysis: whether the extension copies code from PostgreSQL to use in their own extensions
- Versioning code analysis: whether extensions use PG_VERSION_NUM macro to support different logic for each PostgreSQL version



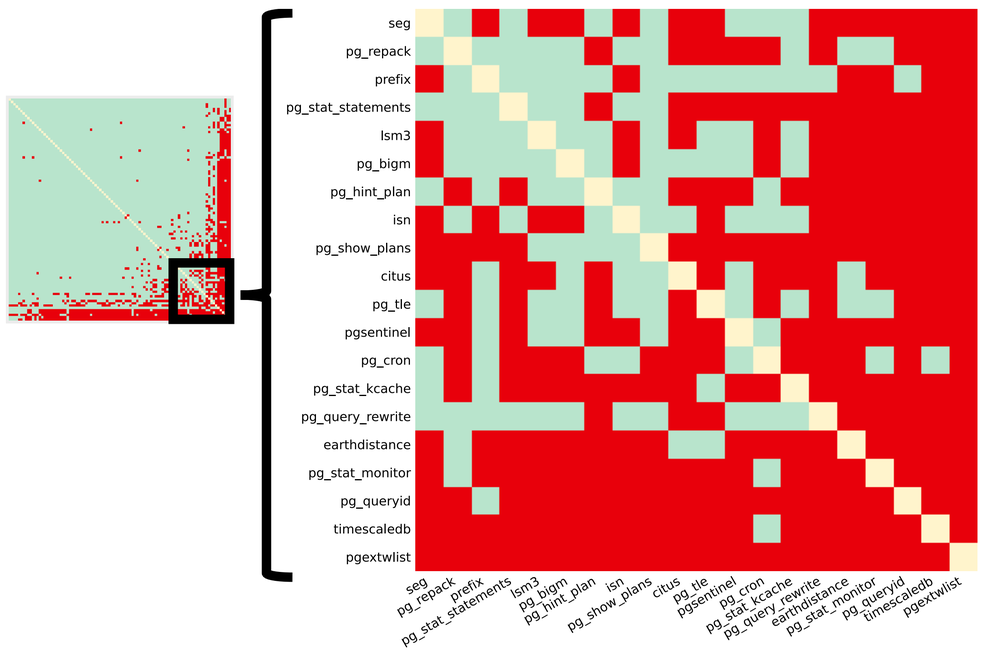


Complex Extension Categories w/K-modes

1. New indexes
2. Query processing features
3. UDT-optimized query engine
4. New storage manager
5. Full-featured extensions (all types)



PostgreSQL Compatibility Analysis



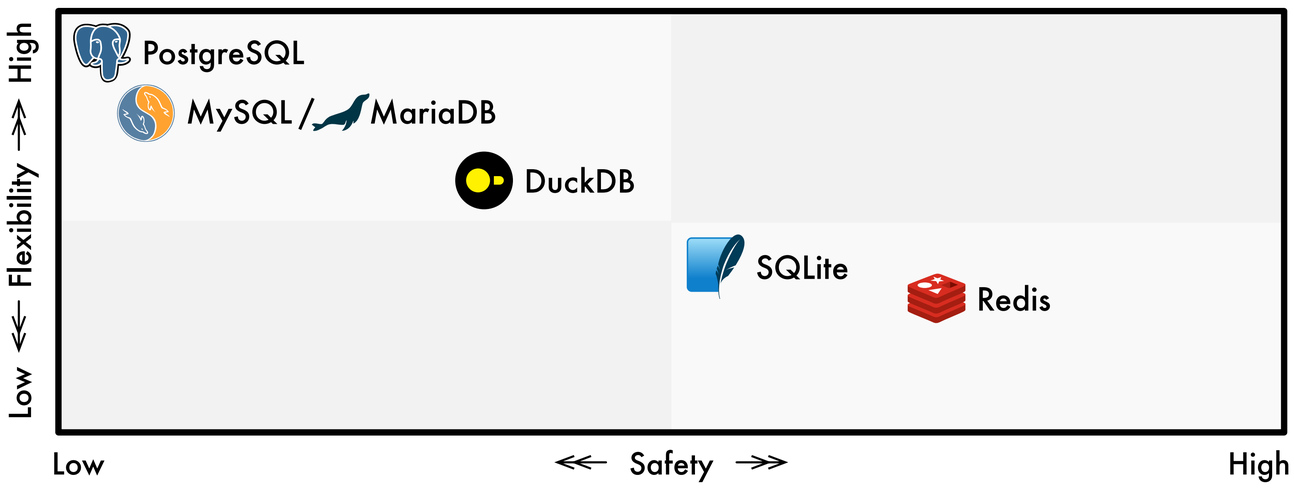
Reasons for Failure

- Extension bugs: deadlocks, memory errors (segfaults, OOM)
- Incompatible configuration settings
- UDT operator incompatibilities
- Query processing bugs (plan tree incompatibilities)
- Installation order (extensions fail when installed in the wrong order)c

Failure Correlations

- Higher complexity (more system components, extensibility types, hooks)
- Query planner and execution hooks (planner_hook, ProcessUtility_hook)
- Small-scale hooks (set_join_pathlist_hook)
- Codebase quality measures (duplicate code usage, larger codebase)

Discussion



Safety vs. flexibility tradeoffs in DBMSs we surveyed. This tradeoff determines the API the DBMS presents to extension developers.

- DBMS extensibility is not inherently composable, but developers still use extensions as dependencies in their implementations
  - Suggestion: formally support composability
- Similar extensions are re-implemented for different systems
  - Suggestion: POSIX-like API for extensions
- PostgreSQL's hook chaining is a problem since it creates a new call path that extensions do not test beforehand
  - Suggestion: use SQL interface to install extensions
- PostgreSQL gives extensions a lot of responsibility to handle internal state without giving them the tools to do so (helper APIs)
- PostgreSQL's building and testing infrastructure makes extensions development easier => contributed to massive popularity