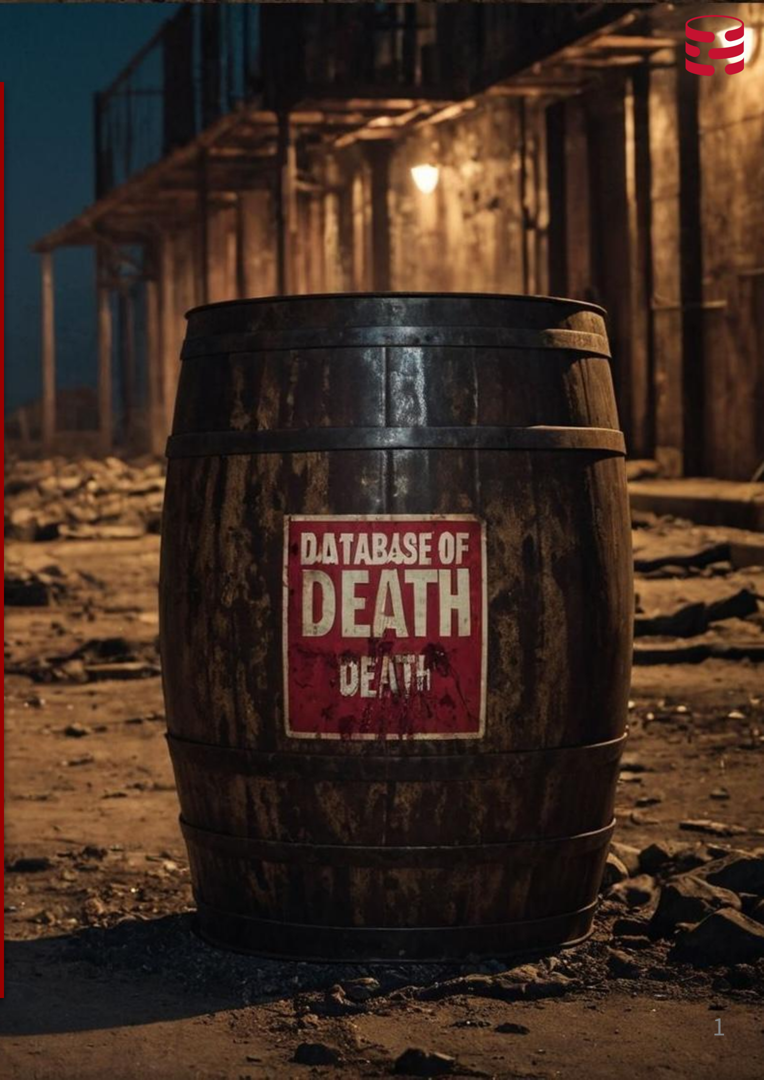


Anarchy in the Database

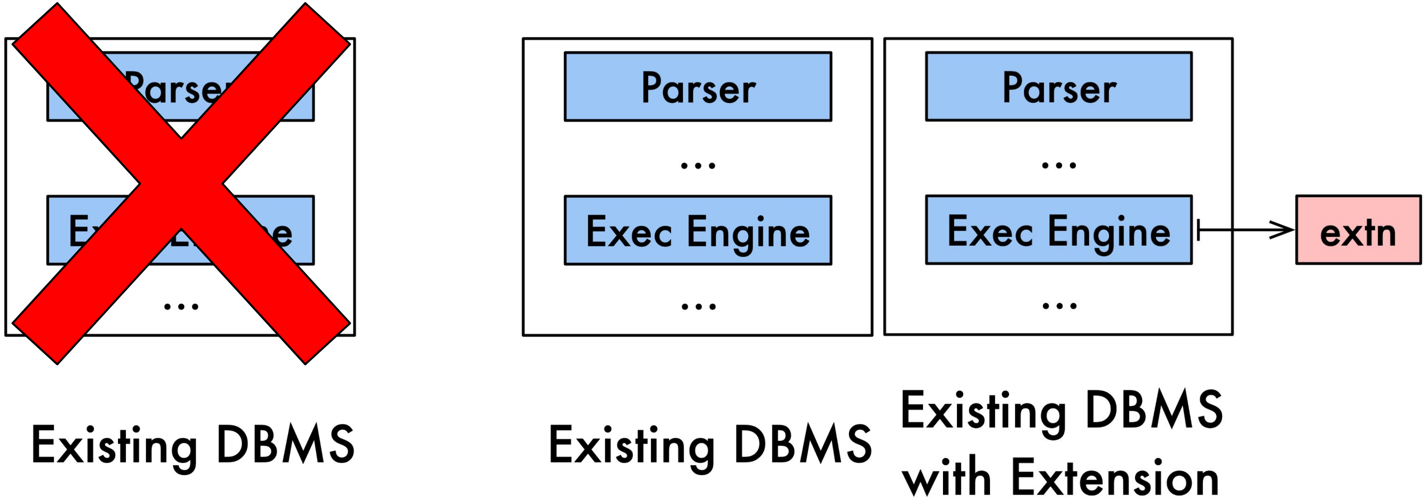
Survey and Evaluation of Database Management System Extensibility

Abigale Kim
University of Wisconsin–Madison
abigale@cs.wisc.edu

Collaborators
Andy Pavlo (pavlo@cs.cmu.edu)
Dave Andersen (dga@cs.cmu.edu)
Marco Slot (marco.slot@crunchydata.com)

POV: You want to find a DBMS for your use case...

Hmm.. existing DBMSs don’t work for our special new use case.



What is extensibility?

- Extensibility: the capability of a database system to let custom software extend its capabilities
- Extension: an instance of this software

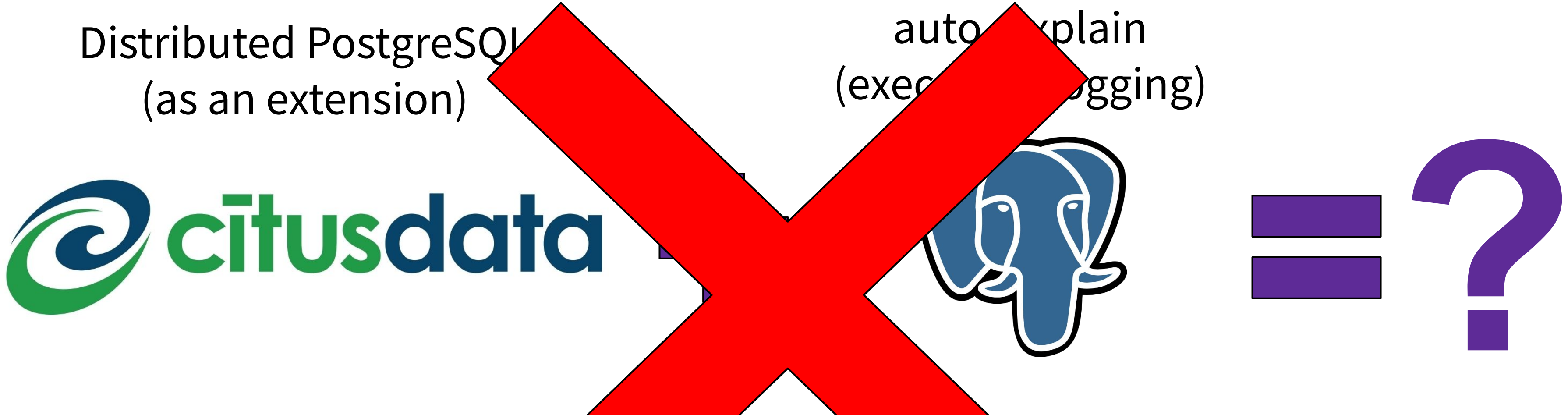
Extensibility benefits

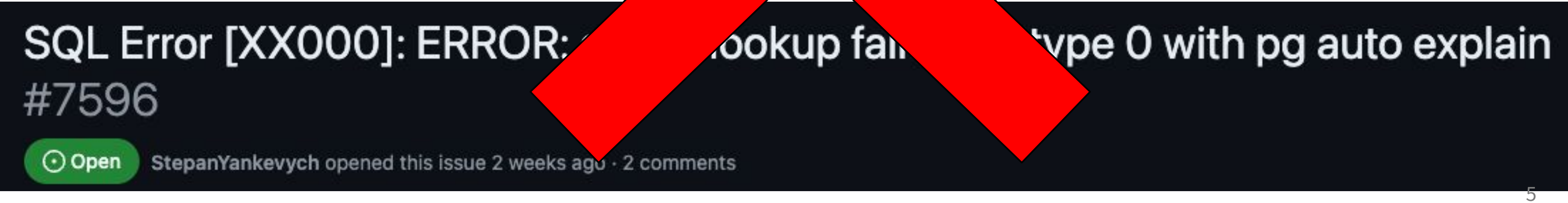
- Support more use cases with less code ✅
- Streamlined developer efforts ✅
- Extensions can be merged into main system ✅
  - autovacuum (merged in 2005)

What happens when we combine extensions?

Distributed PostgreSQL
(as an extension)

auto_explain
(execution logging)





SQL Error [XX000]: ERROR: cache lookup failed for type 0 with pg auto explain #7596

○ Open StepanYankevych opened this issue 2 weeks ago · 2 comments

Research questions

- How well-designed is current DBMS extensibility?
- What design decisions caused conflicts like this (and similar)?
- What can we do to improve the design of DBMS extensibility?

] this paper!

- Can we design extensibility to not have these conflicts?] future work

Research Layout

Taxonomy and Survey

- Categorize extensibility design and ecosystem (taxonomy)
- Figure out how modern DBMSs support extensibility (survey)

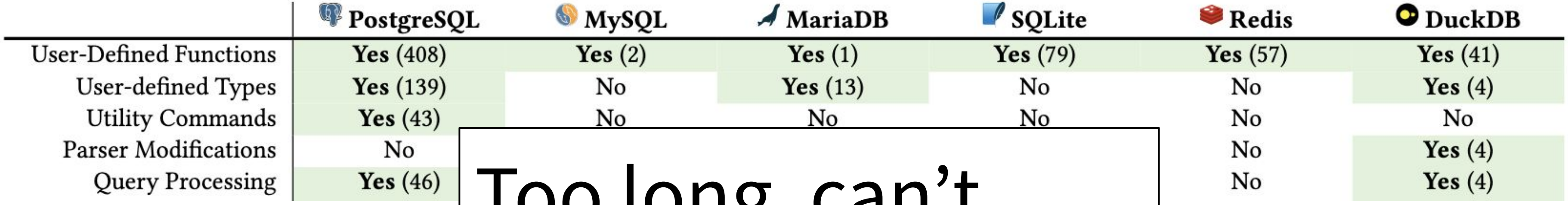
Evaluation

- Collect data on PostgreSQL extensions and analyze the extensibility ecosystem (analysis)

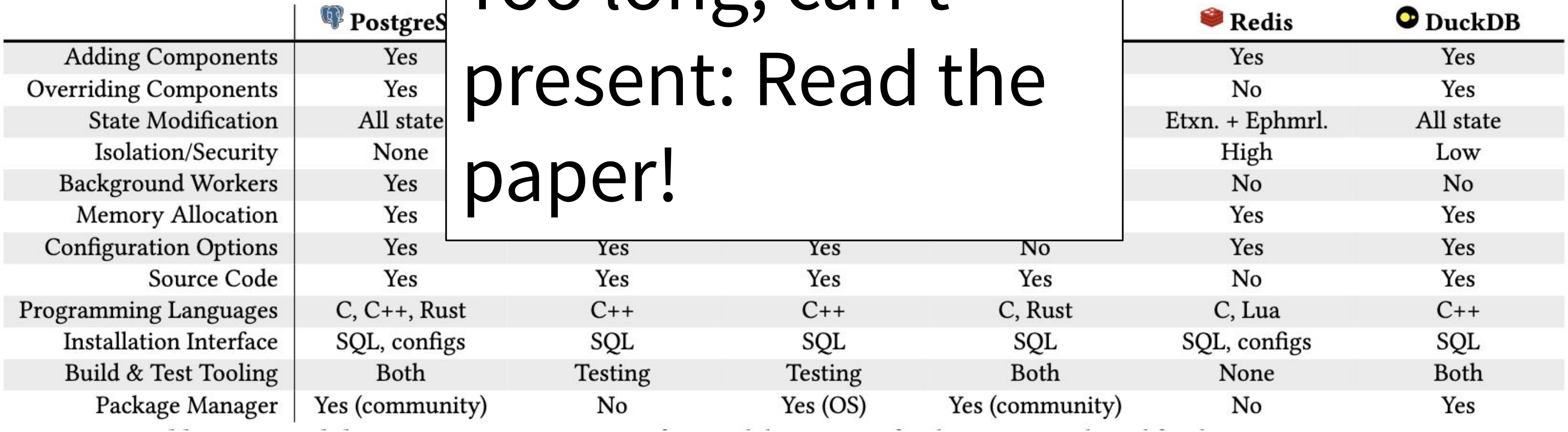
Survey

- Goal: gain larger understanding of the DBMS extensibility design space
- Examined six different DBMSs (PostgreSQL, MySQL, MariaDB, SQLite, DuckDB, Redis OSS)
    - Open source, more comprehensive support for extensibility
- Read extensibility implementation and extensions

Survey Results



	PostgreSQL	MySQL	MariaDB	SQLite	Redis	DuckDB
User-Defined Functions	**Yes** (408)	**Yes** (2)	**Yes** (1)	**Yes** (79)	**Yes** (57)	**Yes** (41)
User-defined Types	**Yes** (139)	No	**Yes** (13)	No	No	**Yes** (4)
Utility Commands	**Yes** (43)	No	No	No	No	No
Parser Modifications	No	[illegible]	[illegible]	[illegible]	No	**Yes** (4)
Query Processing	**Yes** (46)	[illegible]	[illegible]	[illegible]	No	**Yes** (4)



	PostgreS[illegible]	[illegible]	[illegible]	[illegible]	Redis	DuckDB
Adding Components	Yes	[illegible]	[illegible]	[illegible]	Yes	Yes
Overriding Components	Yes	[illegible]	[illegible]	[illegible]	No	Yes
State Modification	All state	[illegible]	[illegible]	[illegible]	Etxn. + Ephmrl.	All state
Isolation/Security	None	[illegible]	[illegible]	[illegible]	High	Low
Background Workers	Yes	[illegible]	[illegible]	[illegible]	No	No
Memory Allocation	Yes	[illegible]	[illegible]	[illegible]	Yes	Yes
Configuration Options	Yes	Yes	Yes	No	Yes	Yes
Source Code	Yes	Yes	Yes	Yes	No	Yes
Programming Languages	C, C++, Rust	C++	C++	C, Rust	C, Lua	C++
Installation Interface	SQL, configs	SQL	SQL	SQL	SQL, configs	SQL
Build & Test Tooling	Both	Testing	Testing	Both	None	Both
Package Manager	Yes (community)	No	Yes (OS)	Yes (community)	No	Yes

Too long, can't present: Read the paper!

ExtAnalyzer!

Code: https://github.com/cmu-db/ext-analyzer

- Tests compatibility between different extensions
  - Compatibility: two extensions work as intended when installed together
- Collects information about extensions
  - Types of extensibility and system components used
  - Source code analysis on bad practices (e.g. duplicate code in extension and DB codebases)

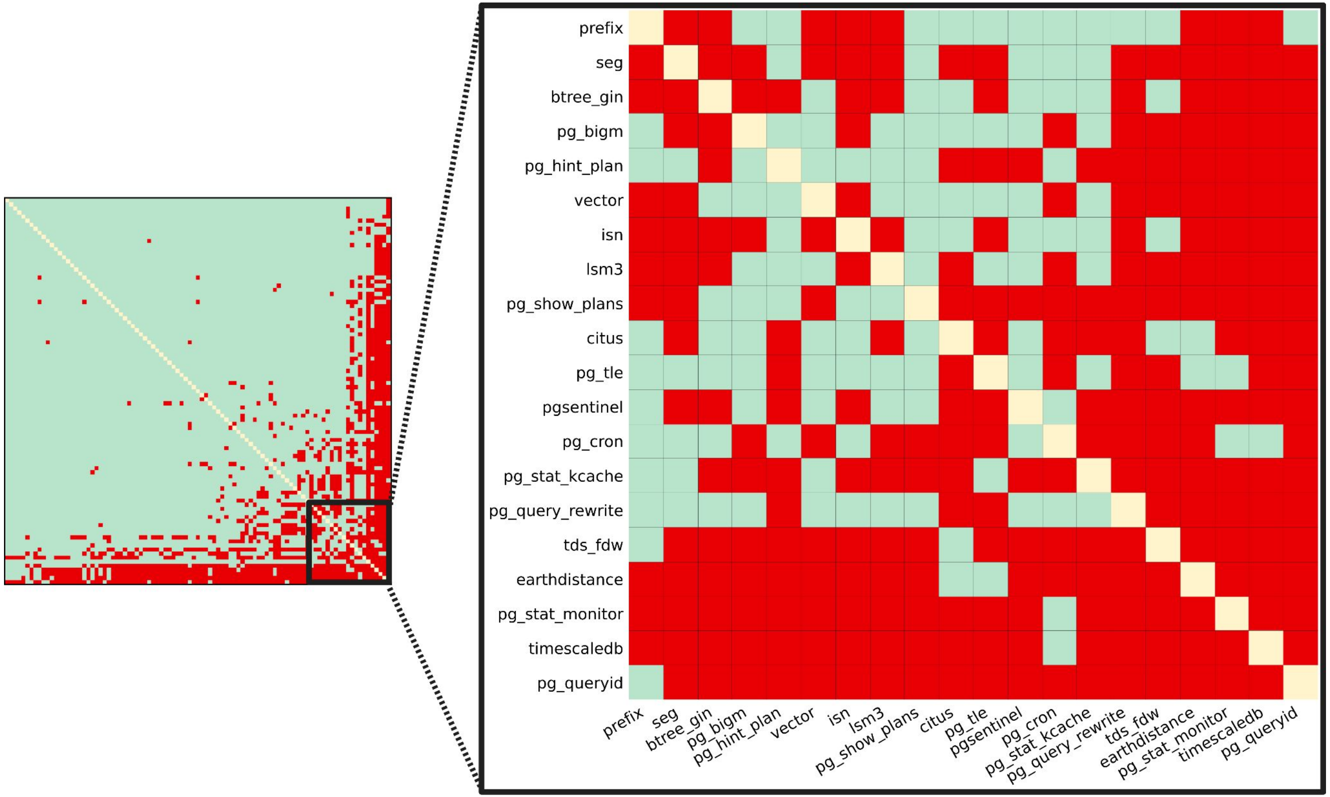
We wrote an analysis framework for all the DBMSs we surveyed, but PostgreSQL has the largest extensibility ecosystem and provided the most fruitful results.

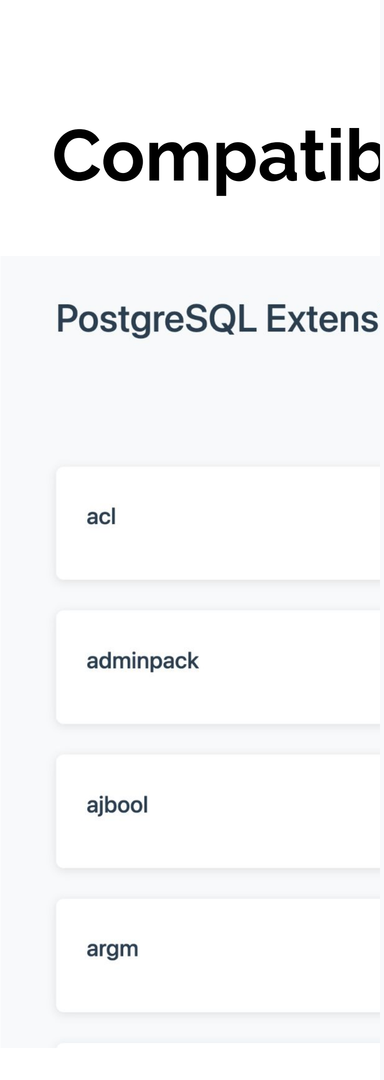
ExtAnalyzer!

- Tests compatibility between different extensions
  - Compatibility: two extensions work as intended when installed together
- Collects information about extensions
  - Types of extensibility and system components used
  - Source code analysis on bad practices (e.g. duplicate code in extension and DB codebases)

We wrote an analysis framework for all the DBMSs we surveyed, but PostgreSQL has the largest extensibility ecosystem and provided the most fruitful results.

Compatibility Failure Matrix





Compatib

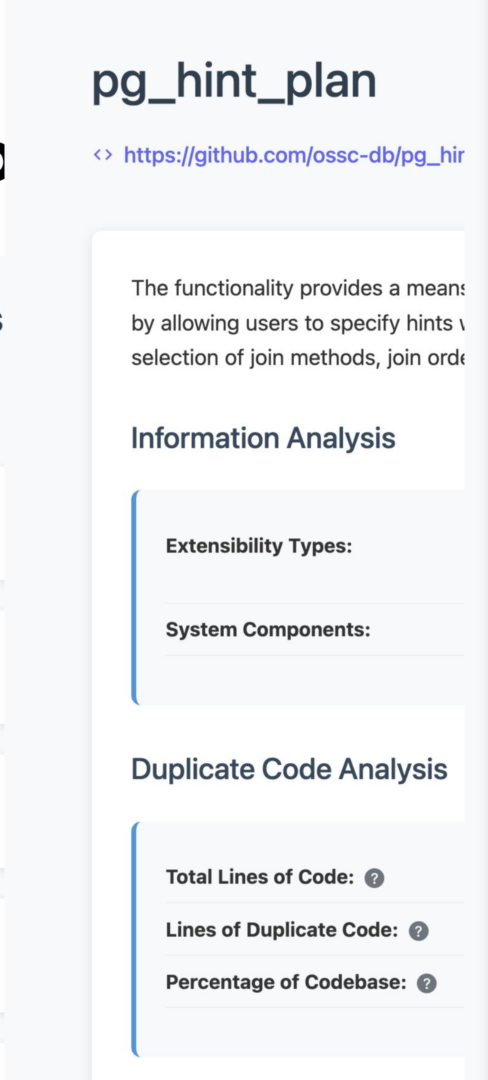
PostgreSQL Extens

acl

adminpack

ajbool

argm



pg_hint_plan

<> https://github.com/ossc-db/pg_hir

The functionality provides a means
by allowing users to specify hints v
selection of join methods, join orde

Information Analysis

Extensibility Types:

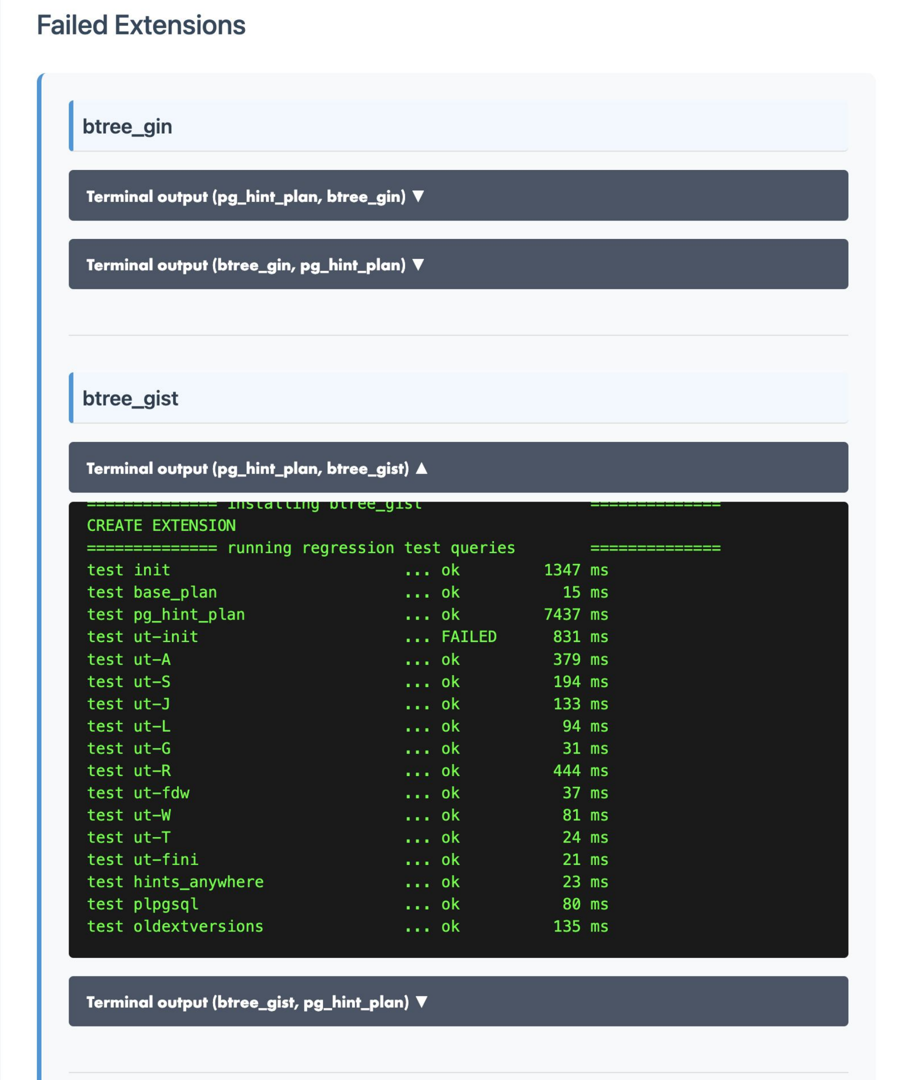
System Components:

Duplicate Code Analysis

Total Lines of Code: ?

Lines of Duplicate Code: ?

Percentage of Codebase: ?



Failed Extensions

btree_gin

Terminal output (pg_hint_plan, btree_gin) ▼

Terminal output (btree_gin, pg_hint_plan) ▼

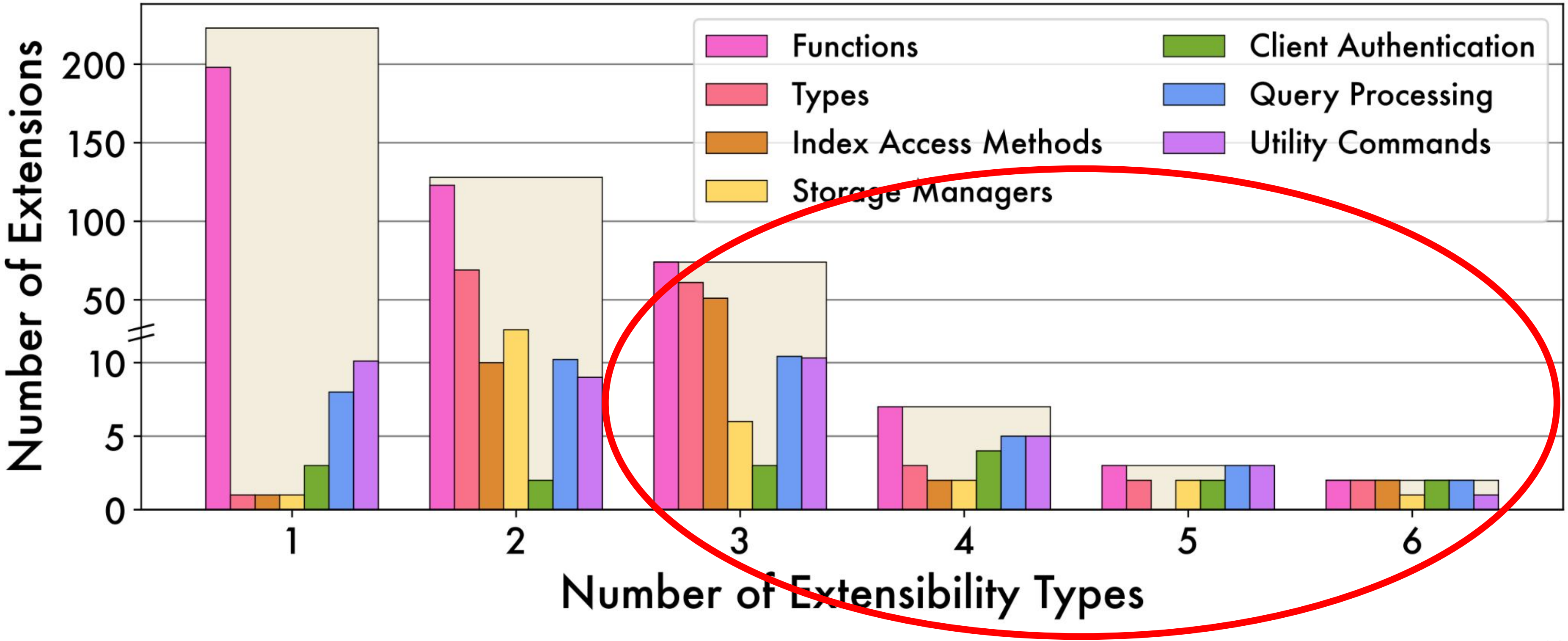
btree_gist

Terminal output (pg_hint_plan, btree_gist) ▲

```
============== installing btree_gist             ==============
CREATE EXTENSION
============== running regression test queries        ==============
test init                         ... ok         1347 ms
test base_plan                    ... ok           15 ms
test pg_hint_plan                 ... ok         7437 ms
test ut-init                      ... FAILED      831 ms
test ut-A                         ... ok          379 ms
test ut-S                         ... ok          194 ms
test ut-J                         ... ok          133 ms
test ut-L                         ... ok           94 ms
test ut-G                         ... ok           31 ms
test ut-R                         ... ok          444 ms
test ut-fdw                       ... ok           37 ms
test ut-W                         ... ok           81 ms
test ut-T                         ... ok           24 ms
test ut-fini                      ... ok           21 ms
test hints_anywhere               ... ok           23 ms
test plpgsql                      ... ok           80 ms
test oldextversions               ... ok          135 ms
```

Terminal output (btree_gist, pg_hint_plan) ▼

Extensibility Distribution



Complex Extensions (>= 3 extensibility types)

- Used K-modes clustering to determine types of complex extensions
- New indexes (67.9%)
  - UDFs, UDTs, index access methods
- Query processing features (21.0%)
  - UDFs, query processing, utility commands
- UDT-optimized query engine (4.9%)
  - UDFs, UDTs, Storage Manager, Query Processing
- New storage manager (4.9%)
  - UDFs, storage manager, utility commands
- Full-featured extensions (1.2%)
  - All extensibility types

Connecting properties to incompatibility

- Ran paired T-tests on extensions and their characteristics
- Higher complexity causes lower incompatibility
  - Using many extensibility types, system components, hooks
  - Duplicate code usage, having a larger codebase
- Larger scope query planner and execution hooks
  - Modifying the planner or executor decreases compatibility

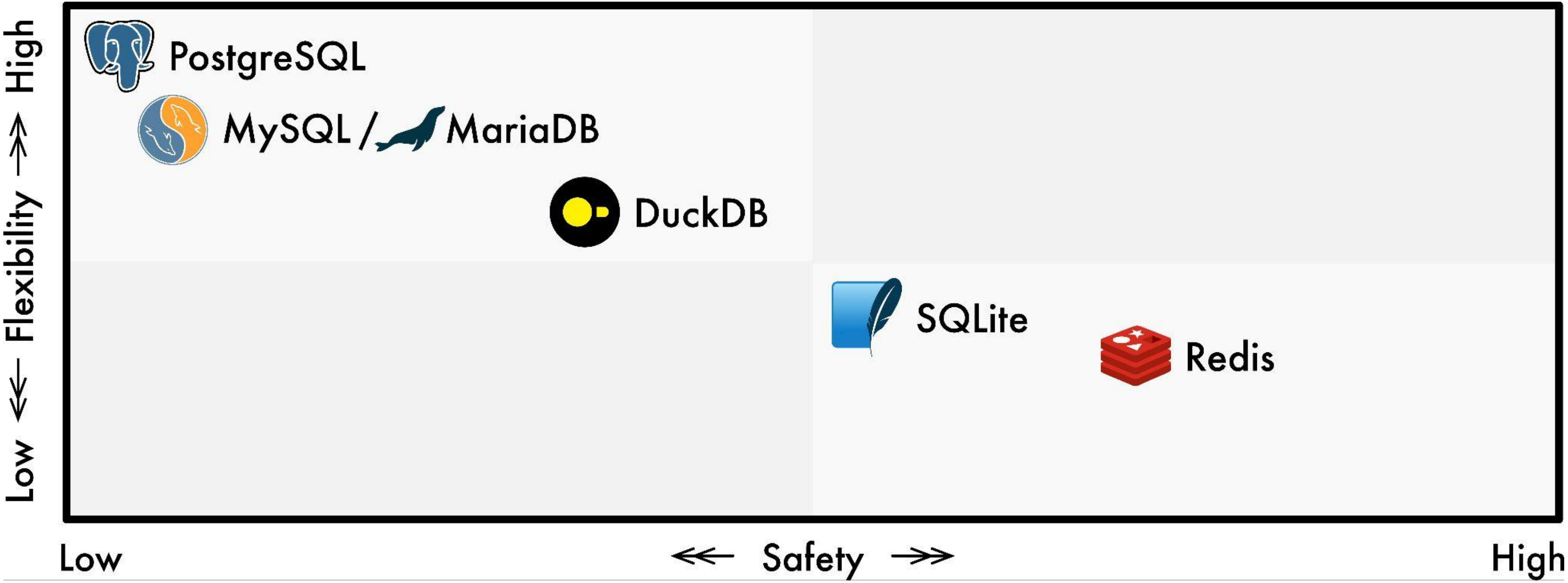
Discussion

- Safety vs. flexibility trade-off
- Extensions naturally compose each other
- Similar extensions exist in different systems
- PostgreSQL API Lessons

Discussion

- Safety vs. flexibility trade-off
- PostgreSQL API Lessons
- Extensions naturally compose each other
- Similar extensions exist in different systems

Safety vs. Flexibility



PostgreSQL API Lessons

- DBMS should expose finer-grained hooks for common purposes ✅
- DBMS should have helper APIs to help extensions manage internal state ✅
- Standardized build and test infrastructure leads to more development ✅

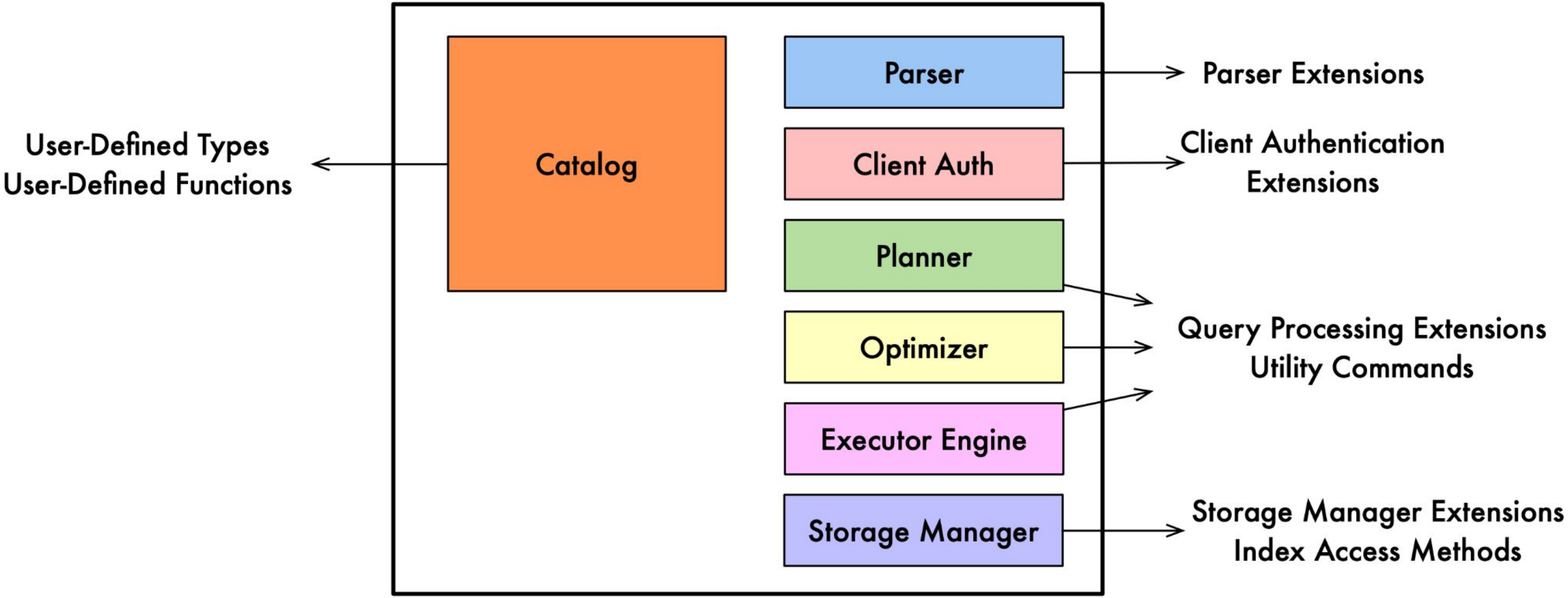
Takeaways

- Survey findings: PostgreSQL's flexible interface, comprehensive support, and usability results in significantly more prolific ecosystem
- Analysis findings: Extensions are commonly incompatible with each other, caused by higher complexity and query execution modules
- Discussion: safety vs. flexibility, PostgreSQL API gives extensions too much responsibility without helper APIs

Github Repo: ExtAnalyzer!

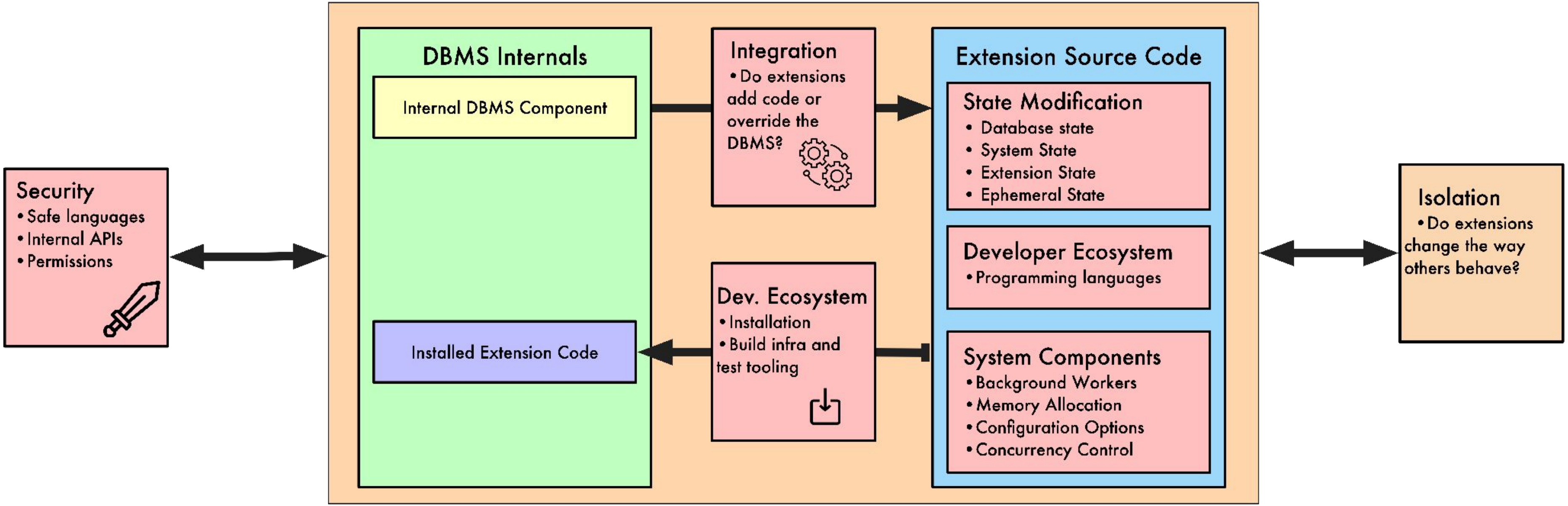
Email: abigale@cs.wisc.edu

Backup Slide: Types of Extensibility



Internal DBMS Architecture

Backup Slide: Extension Taxonomy Diagram



Security
- Safe languages
- Internal APIs
- Permissions

DBMS Internals

Internal DBMS Component

Installed Extension Code

Integration
- Do extensions add code or override the DBMS?

Dev. Ecosystem
- Installation
- Build infra and test tooling

Extension Source Code

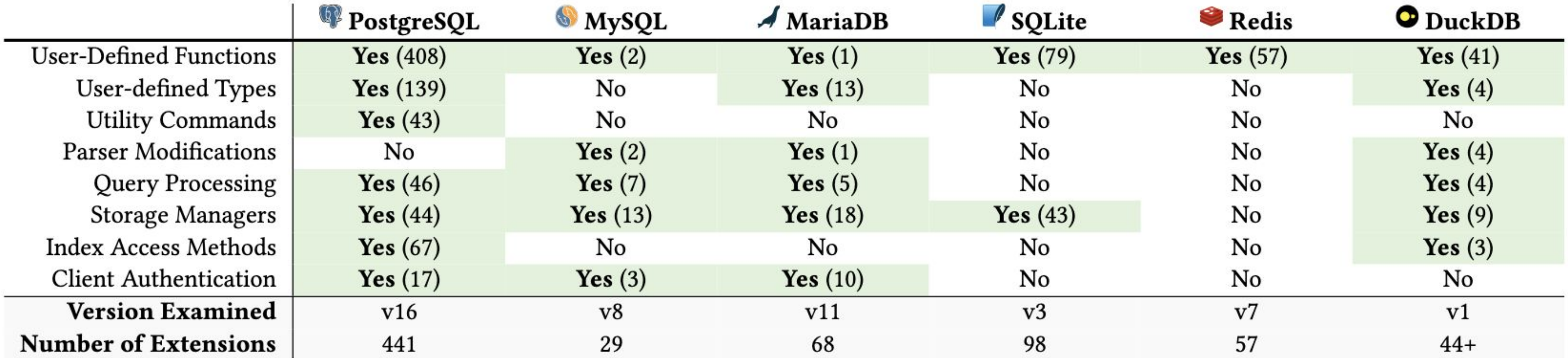
State Modification
- Database state
- System State
- Extension State
- Ephemeral State

Developer Ecosystem
- Programming languages

System Components
- Background Workers
- Memory Allocation
- Configuration Options
- Concurrency Control

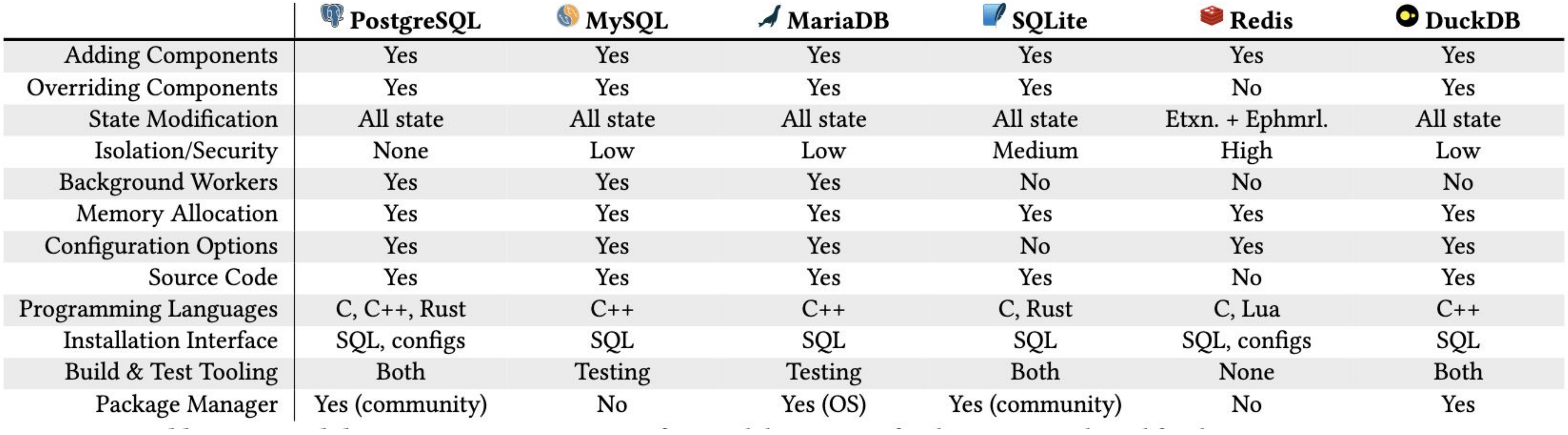
Isolation
- Do extensions change the way others behave?

Backup Slide: Types of Extensibility Survey



	PostgreSQL	MySQL	MariaDB	SQLite	Redis	DuckDB
User-Defined Functions	**Yes** (408)	**Yes** (2)	**Yes** (1)	**Yes** (79)	**Yes** (57)	**Yes** (41)
User-defined Types	**Yes** (139)	No	**Yes** (13)	No	No	**Yes** (4)
Utility Commands	**Yes** (43)	No	No	No	No	No
Parser Modifications	No	**Yes** (2)	**Yes** (1)	No	No	**Yes** (4)
Query Processing	**Yes** (46)	**Yes** (7)	**Yes** (5)	No	No	**Yes** (4)
Storage Managers	**Yes** (44)	**Yes** (13)	**Yes** (18)	**Yes** (43)	No	**Yes** (9)
Index Access Methods	**Yes** (67)	No	No	No	No	**Yes** (3)
Client Authentication	**Yes** (17)	**Yes** (3)	**Yes** (10)	No	No	No
Version Examined	v16	v8	v11	v3	v7	v1
Number of Extensions	441	29	68	98	57	44+

Backup Slide: Survey



	PostgreSQL	MySQL	MariaDB	SQLite	Redis	DuckDB
Adding Components	Yes	Yes	Yes	Yes	Yes	Yes
Overriding Components	Yes	Yes	Yes	Yes	No	Yes
State Modification	All state	All state	All state	All state	Etxn. + Ephmrl.	All state
Isolation/Security	None	Low	Low	Medium	High	Low
Background Workers	Yes	Yes	Yes	No	No	No
Memory Allocation	Yes	Yes	Yes	Yes	Yes	Yes
Configuration Options	Yes	Yes	Yes	No	Yes	Yes
Source Code	Yes	Yes	Yes	Yes	No	Yes
Programming Languages	C, C++, Rust	C++	C++	C, Rust	C, Lua	C++
Installation Interface	SQL, configs	SQL	SQL	SQL	SQL, configs	SQL
Build & Test Tooling	Both	Testing	Testing	Both	None	Both
Package Manager	Yes (community)	No	Yes (OS)	Yes (community)	No	Yes