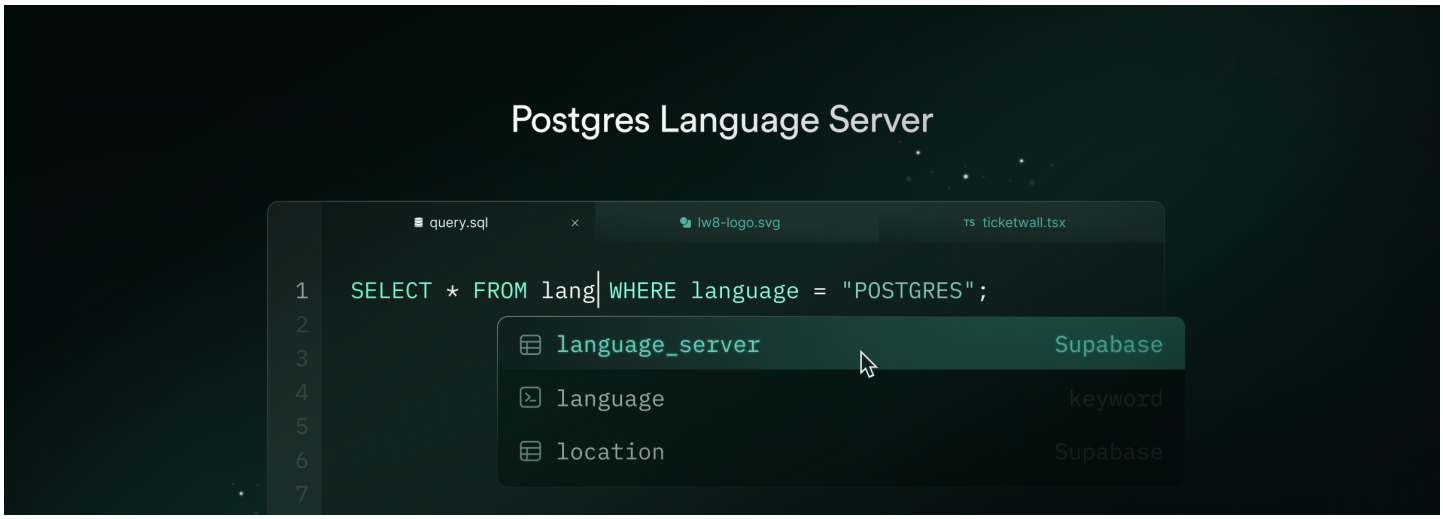
A Language Server for Postgres

supabase.com

MIT license

313 stars 3 forks Activity

README.md



Postgres Language Server

A Language Server for Postgres. Not SQL with flavors, just Postgres.

Status

🚧 This is in active development and is only ready for collaborators. The majority of work is still ahead, but we've verified that the approach works. We're making this public so that we can develop it in the open with input from the community.

Features

The Language Server Protocol is an open protocol between code editors and servers to provide code intelligence tools such as code completion and syntax highlighting. This project implements such a language server for Postgres, significantly enhancing the developer experience within your favorite editor by adding:

- Semantic Highlighting
- Syntax Error Diagnostics
- Show SQL comments on hover
- Auto-Completion
- Code actions such as `Execute the statement under the cursor`, or `Execute the current file`
- Configurable Code Formatting
- ... and many more

Motivation

Despite the rising popularity of Postgres, support for the PL/pgSQL in IDEs and editors is limited. While there are some *generic* SQL Language Servers[1] offering the Postgres syntax as a "flavor" within the parser, they usually fall short due to the ever-evolving and complex syntax of PostgreSQL. There are a few proprietary IDEs[2] that work well, but the features are only available within the respective IDE.

This Language Server is designed to support Postgres, and only Postgres. The server uses libpg_query, therefore leveraging the PostgreSQL source to parse the SQL code reliably. Using Postgres within a Language Server might seem unconventional, but it's the only reliable way of parsing all valid PostgreSQL queries. You can find a longer rationale on why This is the Way™ here. While libpg_query was built to execute SQL, and not to build a language server, any shortcomings have been successfully mitigated in the `parser` crate. You can read the commented source code for more details on the inner workings of the parser.

Once the parser is stable, and a robust and scalable data model is implemented, the language server will not only provide basic features such as semantic highlighting, code completion and syntax error diagnostics, but also serve as the user interface for all the great tooling of the Postgres ecosystem.

Roadmap

This is a proof of concept for building both a concrete syntax tree and an abstract syntax tree from a potentially malformed PostgreSQL source code. The `postgres_lsp` crate was created to prove that it works end-to-end, and is just a very basic language server with semantic highlighting and error diagnostics. Before further feature development, we have to complete a bit of groundwork:

1. *Finish the parser*
    - The parser works, but the enum values for all the different syntax elements and internal conversations are manually written or copied, and, in some places, only cover a few elements required for a simple select statement. To have full coverage without possibilities for a copy and paste error, they should be generated from `pg_query.rs` source code. (#4)
    - There are a few cases such as nested and named dollar quoted strings that cause the parser to fail due to limitations of the regex-based lexer. Nothing that is impossible to fix, or requires any fundamental change in the parser though.
2. *Implement a robust and scalable data model*
    - This is still in a research phase
    - A great rationale on the importance of the data model in a language server can be found here
    - `rust-analyzer`'s `base-db` crate will serve as a role model
    - The `salsa` crate will most likely be the underlying data structure
3. *Setup the language server properly*
    - This is still in a research phase
    - Once again `rust-analyzer` will serve as a role model, and we will most likely implement the same queueing and cancellation approach
4. *Implement basic language server features*
    - Semantic Highlighting
    - Syntax Error Diagnostics
    - Show SQL comments on hover
    - Auto-Completion
    - Code Actions, such as `Execute the statement under the cursor`, or `Execute the current file`
    - ... anything you can think of really
5. *Integrate all the existing open source tooling*
    - Show migration file lint errors from squawk
    - Show plpsql lint errors from plpgsql_check
6. *Build missing pieces*
    - An optionated code formatter (think prettier for PostgreSQL)
7. *(Maybe) Support advanced features with declarative schema management*
    - Jump to definition
    - ... anything you can think of really

Installation

```
- This is not ready for production use. Only install this if you want to help with dev
```

Neovim

Add the postgres_lsp executable to your path, and add the following to your config to use it.

```
require('lspconfig.configs').postgres_lsp = {
  default_config = {
    name = 'postgres_lsp',
    cmd = {'postgres_lsp'},
    filetypes = {'sql'},
    single_file_support = true,
    root_dir = util.root_pattern 'root-file.txt'
  }
}

lsp.configure("postgres_lsp", {force_setup = true})
```

Contributors

- psteinroe (Maintainer)

Acknowledgments

- rust-analyzer for implementing such a robust, well documented, and feature-rich language server. Great place to learn from.
- squawk and pganalyze for inspiring the use of libpg_query.

Footnotes

1. Generic SQL Solutions: sql-language-server, pgFormatter, sql-parser-cst ↩

2. Proprietary IDEs: DataGrip ↩

Releases

No releases published

Sponsor this project

Sponsor

Learn more about GitHub Sponsors

Packages

No packages published

Contributors 4

 **psteinroe** Philipp Steinrötter

 **kiwicopple** Copple

 **brncsk** Ádám Barancsuk

 **shuvroroy** Shuvro Roy

Languages

- **Rust** 100.0%