Deep Postgres Extensions in Rust: *postgres-extension.rs*

Jeff Davis <jdavis@postgresql.org>

Citus Data / Microsoft <Jeffrey.Davis@microsoft.com>

Motivation

- **Postgres relies on an ecosystem of extensions**
  - This is a good thing!
- **Extensions allow domain-specific or experimental development**
- **We need to encourage new developers to get involved and new types of extension development**
- **Rust offers a different language and environment**
  - And brings new ideas!

Why Rust?

- **More memory safety features than C**
- **Minimal runtime like C:**
  - No garbage collector or refcounting
  - No “extra” code
- **No “extra” data held in structs**
  - Not even a vtable pointer!
- **Modern features**
- **Growing developer community**
- **Awesome ecosystem**

The Postgres World is C

- **Real extensions used to require C:**
  - Foreign Data Wrappers
  - Custom Data Types
  - Index and Sort Support Functions
  - Background Workers
  - UDFs calling internal functions

What About Procedural Languages?

- **PL/pgSQL, Perl, Python, v8, etc.**
- **Essentially sandboxes**
- **Only for UDFs and SPI**
  - SPI: Server Programming Interface allows execution of arbitrary SQL within a UDF
- **We need something more**

Let’s see what rust can do

- **Go beyond the Rust marketing and see how to use it to work with a complex system like postgres:**
  - Memory Contexts
  - Error handling using setjmp/longjmp
  - Global variables
  - Intricate APIs

So what is postgres-extension.rs?

- **Allows close integration into the backend as an extension, just like C**
- **But it’s a pure Rust crate**
- **A collection of function declarations, macros, and utility functions**
  - Link seamlessly with C

Not a Client Driver, PL, or ORM

- **There's already an excellent pure-rust client library: *rust-postgres***
  - Interact with postgresql from client application
  - Thanks Steven Fackler!
- ***postgres-extension.rs* is for deeper integration into the postgres server, like a C extension**

Features 1

- **Can construct and operate directly on Postgres structures**
  - No copying or translation of data going from C to Rust or Rust to C
  - Structure format is declared to be C-compatible
- **Uses palloc()/pfree() for all heap allocations**
  - Even rust standard library calls
  - Means you can safely pass back data that postgres will free with a memory context reset
- **elog()/ereport() support**

Features 2: Solves Error-Handling Mismatch

- **If Rust panics, catch it before it returns to C, and turn it into a postgres ERROR**
- **If postgres calls rust, and rust calls a postgres function, and the postgres function throws an ERROR:**
  - catch it and turn it into a rust panic before skipping over any rust frames
  - Important so that rust destructors are called
- **This problem was a stumbling block preventing better support for C++ extensions, but is solved in *postgres-extension.rs***

Demo 1: UDFs and error handling

- **DEMO**

Demo 2: UDF with SPI

- **DEMO**

Demo 3: Concurrent Server with Tokio

- **Tokio is an async framework**
- **Runtime for futures**
- **Build a background worker extension that:**
  - Accepts simple SQL statements from concurrent connections to port 8080
  - Executes SQL with SPI
  - Returns results

Potential Sources of Overhead

- Array bounds checks
- Catching longjmp() at C→Rust boundary
- Catching rust panics at Rust→C boundary
- Converting rust strings to C strings
- All avoidable if you are careful, just like C

C *and* Rust, not C *or* Rust

- **Make rust developers *more* welcome**
- **Without making C developers *less* welcome**
- **Fitting for a bilingual city like Ottawa!**

Conclusion

- http://github.com/jeff-davis/postgres-extension.rs
- Try out writing extensions in a new language
- Only some internal postgres interfaces are supported for now
- Rust seems to have passed the test for real database internals
- Rust and Postgres have great potential together