Create procedural language extension for the Julia programming language

Personal Details

Full Name	Konstantina Skovola
Email	konskov@gmail.com
GitHub	https://github.com/konskov
LinkedIn	www.linkedin.com/in/konstantina-skovola
Nationality	Greek
Location and timezone	Greece, GMT+2

Introduction

PostgreSQL allows developers more freedom to extend the provided capabilities in an application-specific manner compared to other popular open-source Relational Databases. One example of this is the ability to write user-defined functions in other languages besides SQL and C.

Julia is still a new language but with a growing community and increasing popularity. Among its strong points are the straightforward Python-like syntax, the high speed and of course, its interoperability with other programming languages that is also leveraged by the Pl/julia extension which makes use of Julia's C API.

Of course, Julia is nowhere near as established as Python (for which the handler and Pl support comes included in the standard releases of PostgreSQL). Still, as it becomes more widely adopted, I think that adding a more functional pl/Julia to the list of pl handlers available to developers will soon be seen as a useful contribution to the Postgres community.

In this project I will implement the tasks defined in the project description on the wiki, some of which are also listed as open issues on GitHub, namely:

- Support triggers
- Support event triggers

- Support inline code execution, a.k.a. DO statement
- Support input parameters as arrays
- Cache procedural language code instead of looking it up every time

Deliverables

Pljulia-1.0.sql will look like this:

```
CREATE FUNCTION pljulia_call_handler() RETURNS language_handler
  LANGUAGE c AS 'MODULE_PATHNAME';
CREATE FUNCTION pljulia_inline_handler(internal) RETURNS void
  STRICT LANGUAGE c AS 'MODULE_PATHNAME';
/* The following declaration is optional, depending on the time available
for implementing the validator */
CREATE FUNCTION pljulia_validator(oid) RETURNS void
  STRICT LANGUAGE c as 'MODULE_PATHNAME';
CREATE LANGUAGE  pljulia
  HANDLER pljulia_call_handler
  INLINE pljulia_inline_handler
 /* The following line is optional. */
  VALIDATOR pljulia_validator;
COMMENT ON LANGUAGE pljulia IS 'PL/Julia procedural language';
```

Currently, pl/julia has a pljulia_call_handler that supports regular function calls, but it is missing a trigger_handler(), an event_trigger_handler(). An inline handler will also be added to allow the execution of anonymous code blocks via the DO command.

Furthermore, it is good practice to add a validator to allow language-specific checking to be done during CREATE FUNCTION. For this purpose, one more subgoal can be defined and implemented if time allows it, so it is included in a separate section (Time permitting) in the deliverables table.

Subgoal	**Deliverable**
Trigger support	A function `static Datum pljulia_trigger_handler(PG_FUNCTION_ARGS)` called by pljulia_call_handler when CALLED_AS_TRIGGER(fcinfo)
Event trigger support	A function `static void pljulia_event_trigger_handler(PG_FUNCTION_ARGS)` called by pljulia_call_handler when

	CALLED_AS_EVENT_TRIGGER(fcinfo)
Inline code execution support	A function `pljulia_inline_handler`
Support input parameters as arrays	Type conversion functions
Cache procedural language code	Structs pljulia_proc_desc that holds information for each function and a lookup hash table pljulia_hash_table for quick lookups. Pljulia_compile will be modified to use them
Time permitting	
Check correctness of Julia function definition (before execution time)	A function `pljulia_validator(oid)`

Detailed project description and approximate timeline

The coding period is eleven weeks, which translates to approximately two weeks for each subgoal. I expect that some will turn out to be simpler/trickier than others, some (i.e. triggers) might be somewhat similar in implementation so the two week period assigned to each deliverable is definitely not strict. This is an approximate schedule and likely to change according to the mentors' suggestions.

Post-application period (April 14 - May 16)

Continue studying source code and documentation in more detail to make sure I understand all the internals and am ready to jump into coding

- Source code:
  Procedural language extensions in the source code of Postgres (particularly plpython, plperl, pltcl)
  Julia.h and jlapi.c (used for embedding Julia in C)
- Julia Documentation

Community bonding (May 17 - June 7)

Establish communication with mentors
Resolve any questions, discuss implementation details and make changes to the schedule and deliverables if needed
Start writing type conversion functions

Weeks 1 - 2 (June 7 - 20)
Handle argument and result types Write functions to convert to/from julia/pg value representations[1] Resolve github currently open issues concerning IN/OUT argument types [2] Add tests for new supported types
Weeks 3 - 4 (June 21 - July 4)
Create structs pljulia_proc_desc, pljulia_hash_table Pljulia_proc_desc could have an entry for the function code and the parsed arguments (this is done in the current version, I would just need to add them to the struct) Modify pljulia_compile to lookup the function code first using the structs. If no hashtable entry is found, then create a new hashtable entry for the function, compile the code and create a pljulia_proc_desc for the newly-encountered function Test performance with and without cached code
Weeks 5 - 6 (July 5 - 18)
Write pljulia_trigger_handler This function will use pljulia_compile Write and run tests
Mid-term Evaluation (July 12 - 16) All tests for weeks 1-4 passing and making progress with the trigger handler Assess whether there is enough time to add a validator function, considering the overall progress
Weeks 7 - 8 (July 19 - August 1)

[1] Will start with standard Julia data types such as numeric, strings, arrays, tuples and dictionaries. At least numeric, character, text, boolean, composite, array pg types will be supported as input and output. Currently, `jl_value_t_to_datum` is the function that handles the OUT types (converts from Julia to pg representation) and it already covers many primitive types (except for boolean which is just one more if case to add). Composite types, tuples and arrays are not covered so write cases to support these types. As for IN types, the type and value of the input arguments is currently determined in pljulia_compile, but no check is performed/action taken for non-primitive types (e.g. array types). Therefore write functions to convert pg arrays etc to appropriate Julia representation.

[2] Handle returning arrays of strings,
Handle returning boolean types from user defined functions,
Handle arrays passed as IN parameters

Write event trigger handler. This function will use pljulia_compile Start writing pljulia_validator Write test cases for the event trigger handler
Weeks 8 - 10 (August 2 - 15)
Write pljulia_inline_handler(PG_FUNCTION_ARGS) This function will save the current execution/memory context, create a new context and call a subroutine that will, in this new context, compile and execute the source code passed to it as an argument Finish pljulia_validator Add tests
Final week (August 16 - 23)
Complete documentation, fix any remaining bugs In view of the final evaluation, prepare a demonstration of the features added over the summer to present to the mentors (For example, set up a database, write trigger, event trigger functions and execute transactions that fire them. Display the results using pgAdmin. Execute DO and SELECT statements to showcase the support of inline code blocks and several argument and return types) Write a report to supplement the demo

About me

I am in the final year of my studies at the School of Electrical and Computer Engineering, National Technical University of Athens. My majors are Computer Science and Computer Systems and my minor is Computer Networks. I am currently working on my Masters thesis on Knowledge Graphs for Sustainable Development.

I enjoy coding in C and working on backend tasks. My interests are primarily databases, knowledge management, and operating systems.

I am confident I can deliver what I have mentioned above within the set timeframe.

- C is a language I have used extensively in my coursework, especially in projects for Operating Systems and Embedded Systems
- Database Systems and Advanced Topics in Database Systems taught me much about SQL and database setup/administration

- For my thesis I work mainly with Python scripts, but I have looked into Julia as a faster alternative to Python, to deal with large datasets. So I already have an understanding of Julia, and I will expand on that as needed.

It will be quite a distinction to become involved in Postgres development and start contributing to Postgres and open-source databases. The time used for researching the codebase, mailing list archives and contribution guides prior to my application has shown me that the code is well-documented and the community welcomes new contributors.

During the coding period, I will adjust my working hours as necessary to make sure my progress follows the schedule that we will set together with the mentors.

I had already planned to allot 40 hours per week to GSoC during the summer before finding out that the scope of this year's projects is roughly 20 hours of work per week. I also don't have any other engagements except for my thesis, which I'm flexible about, so I feel it is safe to say that I can remain within the proposed timeline.