Lua в PostgreSQL

из alpha в beta

Евгений Сергеев

PL/Lua

https://github.com/pllua/pllua

Обо мне

Desktop/Database developer (Qt/C++) + СУБД + javascript(embedding and extending), python…

Недвижимость

Медицина

Среднего размера проекты (до 500 пользователей)

СУБД: Oracle, Firebird... Postgresql

Hack and develop pllua

Мотивация

Проблемы (или не проблемы вовсе?)

Код размазывается по схемам (private/public)

Нетривиальная обработка структурированных данных

Невозможно использовать сторонние библиотеки/функции в БД не доработав их напильником либо интегрируя в свою систему (создание C библиотек/оберток для регистрации в БД, возможность совпадения сигнатур фукций при импорте кода)

Скорость разработки (субъективно)

Модульная разработка, require ?

Дублирование функционала

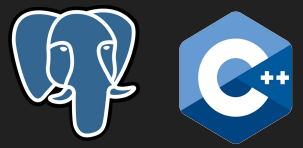
Migrations/generate sql

клиент

chakra shim?

сервер

СУБД



Lua

from Portuguese: lua [ˈlu.(w)ɐ] meaning moon

Появился в 1993, университет Рио-де-Жанейро (Бразилия)

Автор: Роберту Иерузалимски, Валдемар Селиш, Луиш Энрике ди Фигейреду

Мультипарадигмальный: процедурный, функциональный, объектно-ориентированный (прототипный), **встраиваемый**

Тип исполнения: интерпретируемый, JIT-компилируемый

Система типов: динамическая, строгая, «утиная»

Реализация: ANSI C

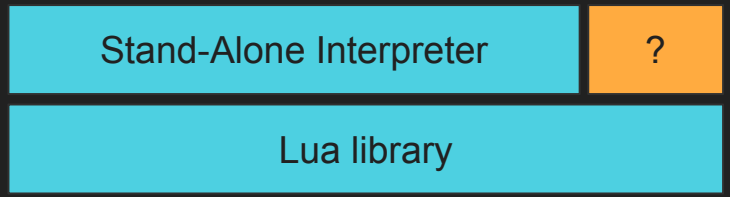
www.lua.org

Lua in 15 Minutes

http://tylerneylon.com/a/learn-lua/

Lua

Lua как встраиваемый язык: lua.c



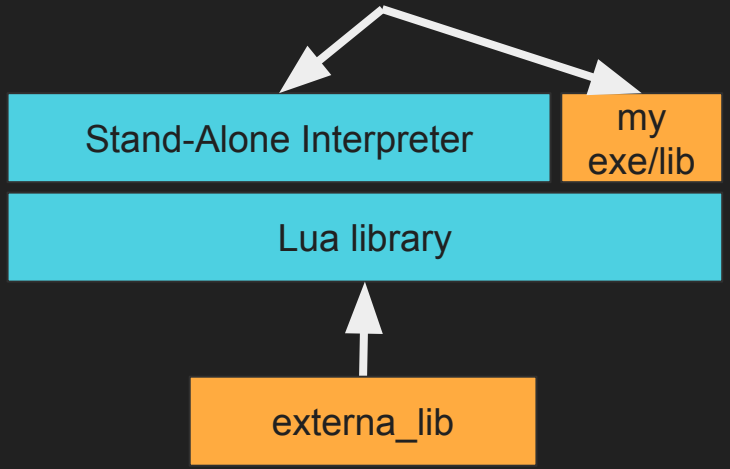
lua_State - можно использовать “неограниченно” в одном или нескольких потоках. Непотокобезопасен.

```
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>
int main(void)
{
    const char * source = “print(‘hello’)”
    lua_State *L = lua_open();
    luaL_openlibs(L);
    luaL_loadbuffer(L, source...)
    lua_pcall(L, 0, 0, 0)
    ...
    return 0;
}
```

Lua

Lua как расширяемый язык

```
local lib = require ‘external_lib’
```



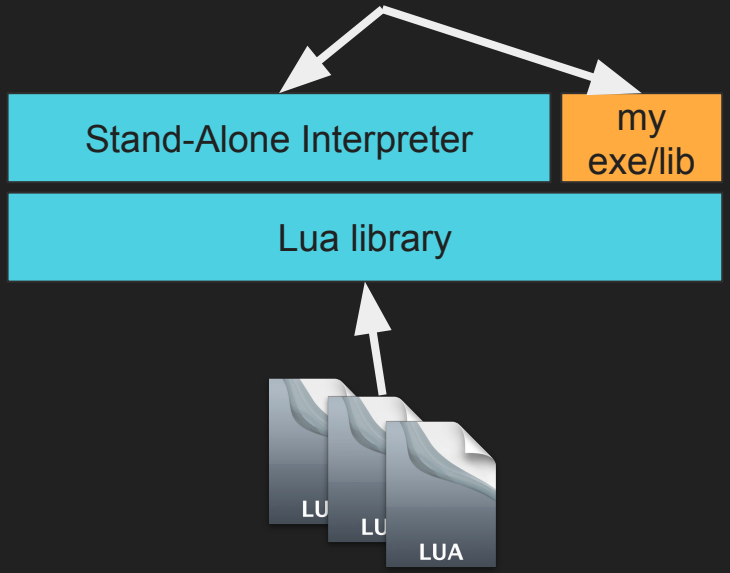
export.c

```
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>
LUALIB_API int luaopen_external_lib(lua_State *L)
{
        ...
        return 1;
}
```

Lua

Lua как расширяемый язык

```
local m = require ‘mymodule’
```




```lua
-- mymodule.lua
local M = {} -- public interface

-- private
local x = 1
local function baz() print 'test' end

function M.foo() print("foo", x) end

function M.bar()
  M.foo()
  baz()
  print "bar"
end

return M
```

Lua remote debugger

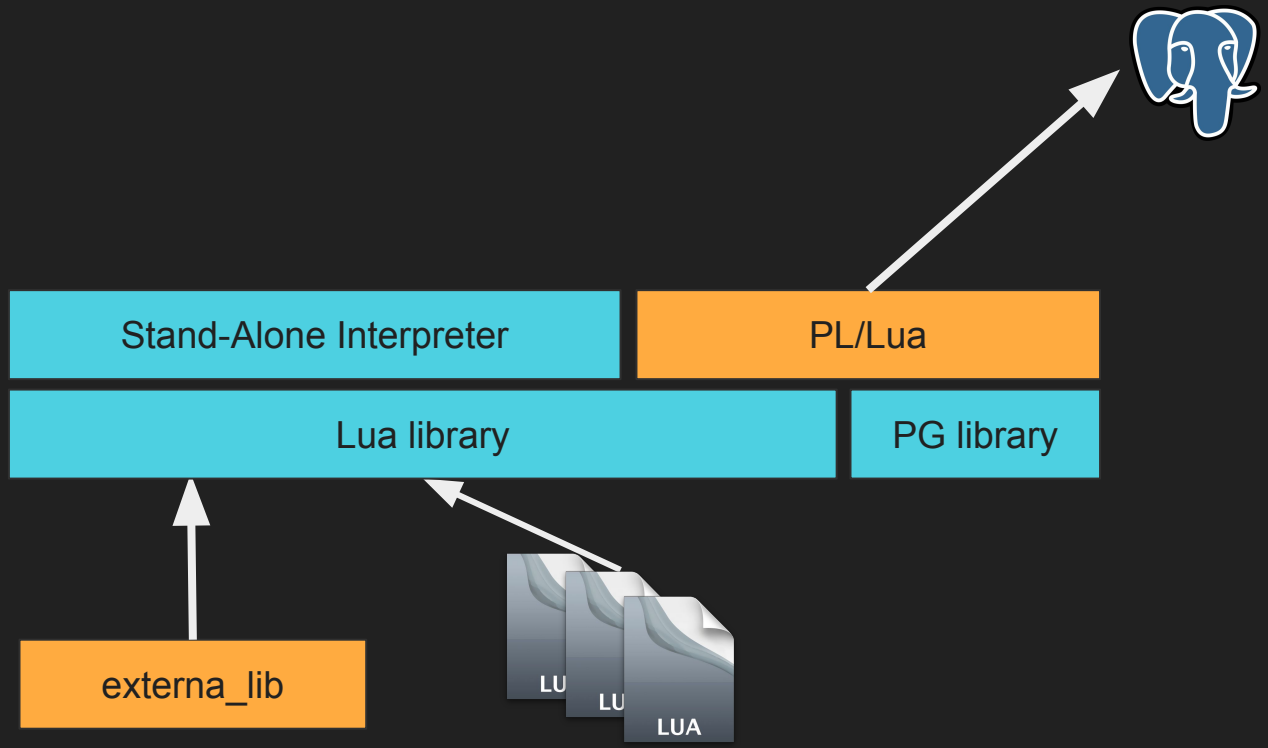
```
$ luarocks install mobdebug
```

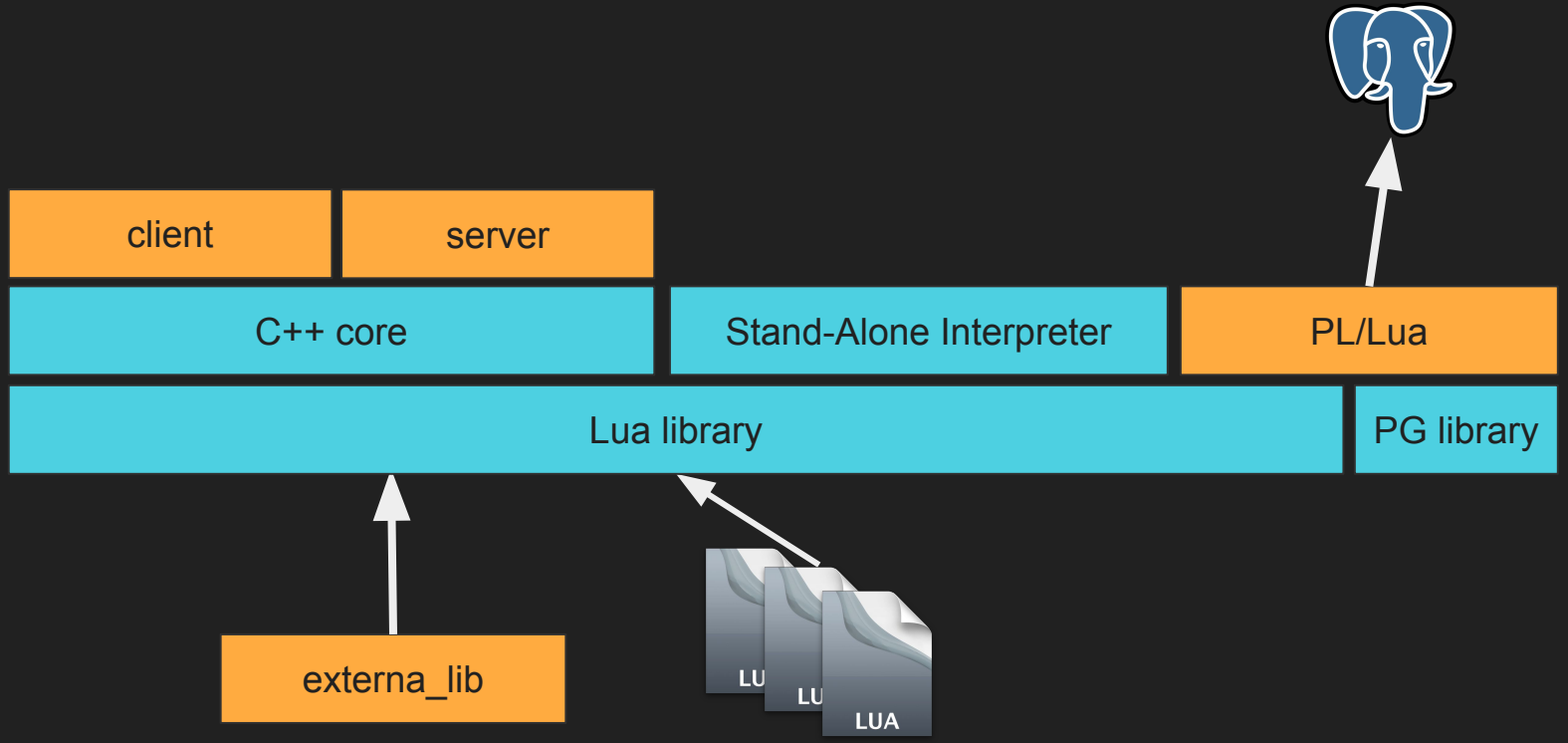
(package managers: luarocks.org, luadist.org)

ZeroBrane Studio->Start Debugger Server

```
local mobdebug = require('mobdebug')
mobdebug.start()
print('Start')
local foo = 0
...
```

PL/Lua


Stand-Alone Interpreter
PL/Lua
Lua library
PG library
externa_lib
LUA


client
server
C++ core
Stand-Alone Interpreter
PL/Lua
Lua library
PG library
externa_lib
LU
LU
LUA

Lua версии

Lua & lua library

Lua 5.1 - Bytecode and Virtual Machine

Luajit - Lua 5.1 API+ABI, Tracing just-in-time compilation, FFI (luajit.org)

Terra - Terra is a simple, statically-typed, compiled language with manual memory management. But unlike C, it is designed from the beginning to interoperate with Lua. Terra functions are first-class Lua values created using the terra keyword. (LLVM backend terralang.org)

Install PL/Lua

пакет postgresql-9.X-pllua — version 1.0 (почему Alpha?)
(c) 2008 Luis Carvalho

github.com/pllua/pllua — dev version

```
PG_CONFIG ?= pg_config
LUA_INCDIR ?= /usr/include/lua5.1
LUALIB ?= -L/usr/local/lib -llua5.1
```

```
$ make clean && make && sudo make install && make installcheck
$ psql -c "CREATE EXTENSION pllua" mydb
```

read access to users pllua.init

PL/Lua что внутри

```
lua_State *LuaVM[2] = {NULL, NULL};
Datum _PG_init(PG_FUNCTION_ARGS) {
```

trusted pllua

```
lua_State *L = luaL_newstate();
…
 {LUA_TABLIBNAME, luaopen_table},
 {LUA_STRLIBNAME, luaopen_string},
 load all from pllua.init,
…
return L;
```

untrusted plluau

```
lua_State *L = luaL_newstate();
…
 luaL_openlibs(L);
…
return L;
```

PL/Lua create function

```
CREATE FUNCTION func(args)
RETURNS rettype AS $$
    -- Lua function body
$$ LANGUAGE [pllua | plluau];
```

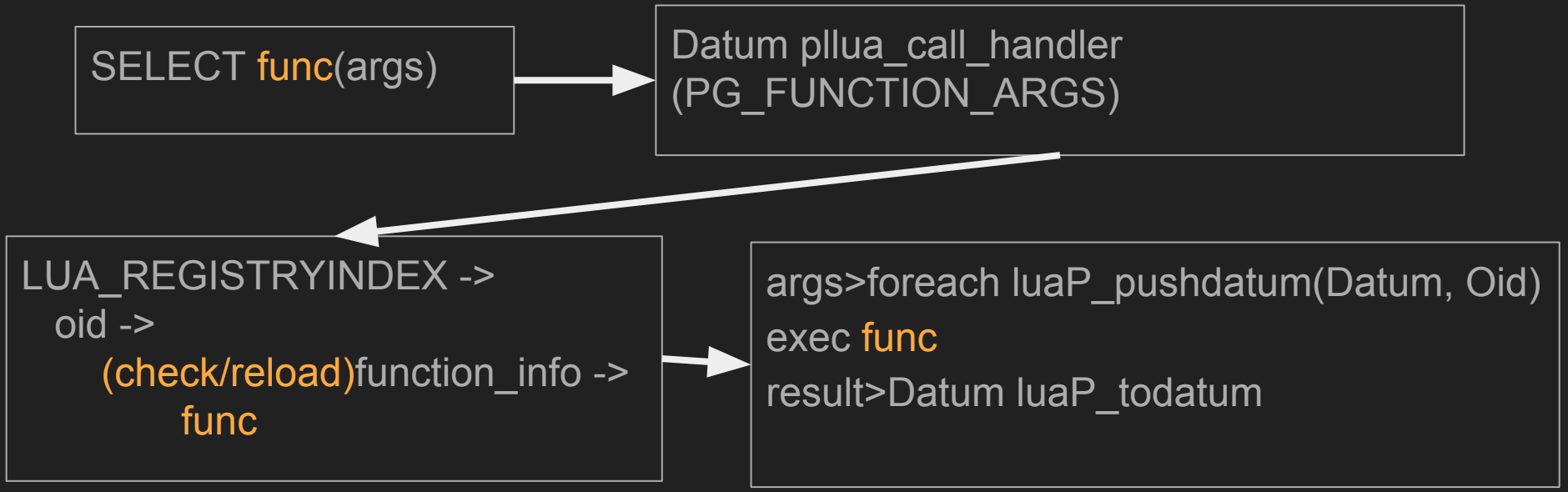
Datum pllua_validator
(PG_FUNCTION_ARGS)

```
local _U, func
func = function(_argnames_)
    -- Lua function body
end
return func
```

lua exec

```
LUA_REGISTRYINDEX ->
    oid ->
        function_info ->
            func
```

PL/Lua execute function



PL/Lua пример функции

```
CREATE FUNCTION counter()
RETURNS int AS $$
    while true do
        _U = _U + 1
        coroutine.yield(_U)
    end
end
do
    _U = 0
    counter = coroutine.wrap(counter)
$$ LANGUAGE pllua;
```

```
CREATE FUNCTION counter()
RETURNS int AS $$
    _U = _U + 1
    return _U
end
do
    _U = 0
$$ LANGUAGE pllua;
```

PL/Lua пример функции

```
CREATE FUNCTION counter()
RETURNS int AS $$
---------------
    _U = _U + 1
    return _U
end
do
    _U = 0
---------------
$$ LANGUAGE pllua;
```

```
local _U, counter
counter = function()
---------------
    _U = _U + 1
    return _U
end
do
    _U = 0
---------------
end
return counter
```

```
_U = {   saved_plan = server.prepare(cmd, {...}):save()   }
```

PL/Lua shared values

```
CREATE FUNCTION getvalue()
RETURNS integer AS $$
    if shared.value == nil then
        setshared("value", 0)
    end
    return value
$$ LANGUAGE pllua;
```

```
CREATE FUNCTION setvalue(v integer)
RETURNS void AS $$
    if shared.value == nil then
        setshared("value", v)
    else
      value = v
    end
$$ LANGUAGE pllua;
```

setshared актуально для trusted версии, т.к. создание новых не local переменных недоступно. Строки, как результат Select будут отброшены при завершении фунции\транзакции

PL/Lua пример функции - многострочный результат или Set-returning functions (SRFs)

```
CREATE FUNCTION nrows(n int4)
RETURNS setof int AS $$
for i = 1, n do
        coroutine.yield(i)
end
$$ LANGUAGE pllua;
```

PL/Lua поддержка типов

PL/Lua поддержка типов

PostgreSQL type	Lua type
bool	boolean
float4, float8, int2, int4	number
text, char, varchar	string
Base, domain	userdata
Arrays, composite	table

операции над строками могуть потребовать дополнительной библиотеки для работы с UTF-8 (копирование подстрок, вычисление длины...)

PL/Lua & composite type

```
CREATE TYPE person AS (name text, surname text);

CREATE FUNCTION smith () RETURNS  person AS $$
      return {name="John", surname = "Smith"}
 $$ LANGUAGE pllua;

SELECT surname, name FROM smith ()
```

PL/Lua & array type

```
CREATE FUNCTION array_test (list text[])
RETURNS text[] AS $$
list[7] = 'check'
return list
$$ LANGUAGE pllua;

select array_test(ARRAY['q','w','e'])
------------
{q,w,e,NULL,NULL,NULL,check}
```

```
CREATE FUNCTION array_test (list text[])
RETURNS text[] AS $$
list[-2] = 'check'
return list
$$ LANGUAGE pllua;

select array_test(ARRAY['q','w','e'])
------------
[-2:3]={check,NULL,NULL,q,w,e}
```

PL/Lua “неподдерживаемые типы”

```
create extension hstore with schema dt;

select 'foo=>x, bar=>qwerty, zzz=>yyy'::dt.hstore
do $$
local value = fromstring('dt.hstore', 'foo=>x, bar=>qwerty, zzz=>yyy')
local del = pgfunc('dt.delete(dt.hstore,text)')
print (value)
value = del(value, 'bar')
print(value)
$$ language pllua

INFO:  "bar"=>"qwerty", "foo"=>"x", "zzz"=>"yyy"
INFO:  "foo"=>"x", "zzz"=>"yyy"
```

Lua Error Handling and Exceptions

```
function foo ()
        ...
  if unexpected_condition then error() end
        ...
  print(a[i])    -- potential error: `a' may not be a table
        ...
end

Then, you call foo with pcall:
if pcall(foo) then
  -- no errors while running `foo'
  ...
else
  -- `foo' raised an error: take appropriate actions
  ...
end

local status, err = pcall(function () error({code=121}) end)
print(err.code)  -->  121
```

PostgreSQL Subtransactions

```
do $$
local plan = server.prepare([[
UPDATE accounts
SET balance = balance + $2
WHERE account_name = $1
returning account_name
]], {"text","int4"})
local moneyTransfer = function(from, to, amount)
        local x
        x = plan:execute{from, (-1)*amount}
        assert(x, 'error transfer from '..from)
        x = plan:execute{to, amount}
        assert(x, 'error transfer to '..to)
        return true
end
local status, info = pcall( moneyTransfer , 'mary', 'joe', 30)
print(status, info)
$$ LANGUAGE pllua;
```

PostgreSQL Error Handling and Exceptions

```
do $$
error({message="error message", hint="error hint", detail="error detail"})
$$language pllua;

ERROR:  error message
DETAIL:  error detail
HINT:  error hint
CONTEXT:
stack traceback(trusted):
        [C]: in function 'error'
        [string "anonymous"]:2: in main chunk
```

\+ info, warning, log

PL/Lua stack traceback

```
create or replace function a() returns void as $$
server.execute('select b()')
$$ language pllua

create or replace function b() returns void as $$
server.execute('select c()')
$$ language plluau

create or replace function c() returns void as $$
server.execute('select d()')
$$ language pllua

create or replace function d() returns void as $$
error('Custom exception')
$$ language plluau
```

```
ERROR: Custom exception
CONTEXT:
stack traceback(untrusted):
        [C]: in function 'error'
        [string "d"]:2: in function <[string "d"]:1>
        [C]: in function 'execute'
        [string "b"]:2: in function <[string "b"]:1>
SQL statement "select d()"
SQL statement "select c()"
SQL statement "select b()"
stack traceback(trusted):
        [C]: in function 'execute'
        [string "c"]:2: in function <[string "c"]:1>
        [C]: in function 'execute'
        [string "a"]:2: in function <[string "a"]:1>
SQL statement "select c()"
SQL statement "select b()"
SQL statement "select b()"
```

PL/Lua & modules

```
local m = require ‘mymodule’
```

/usr/local/share/lua/5.1...

Идея из plv8 -> find_function

```
do $$
local m = pgfunc "mymodule(internal)"
m.bar()
$$ language pllua;
```

```
CREATE OR REPLACE FUNCTION mymodule(internal)
  RETURNS internal AS
$BODY$
local M = {} -- public interface
-- private
local x = 1
local function baz() print 'test' end
function M.foo() print("foo", x) end

function M.bar()
  M.foo()
  baz()
  print "bar"
end

return M

$BODY$ LANGUAGE pllua;
```

PL/Lua & доступ к данным (запросы)

Список записей:

```
list = server.execute(“SELECT/UPDATE ...)
```

Итератор:

```
for row in server.rows(cmd) do
    row.column ...
end
```

План:

```
plan = server.prepare(cmd...); plan:execute(...); plan:rows(...)
```

Курсор:

```
cursor = plan:getcursor(args...) ; cursor:fetch(...); cursor:move(...)
```

PL/Lua & trigger value

```
trigger = {
   ["old"] = "tuple: 0xd084d8",
   ["name"] = "trigger_name",
   ["when"] = "after",
   ["operation"] = "update",
   ["level"] = "row",
   ["row"] = "tuple: 0xd244f8",
   ["relation"] = {
      ["namespace"] = "public",
      ["attributes"] = {
         ["test_column"] = 0,
      },
      ["name"] = "table_name",
      ["oid"] = 59059
   }
}
```

Спасибо за внимание