PostgresPro

# Создание расширения в PostgreSQL

## *Первый шаг на пути к профессиональной разработке БД*

Николай Шаплов

Август 2025

# О докладчике

PostgresPro

**Шаплов Николай**

Фаззинг инженер в компании
Postgres Professional

- C
- C++
- Perl

Последовательный сторонник свободного ПО
в течении более чем 20 лет

# Ссылка на слайды

PostgresPro

https://gitlab.com/dhyannataraj/articles/-/tree/master/slides

# Что такое расширение Postgres

PostgresPro

Расширение – набор объектов Postgres, которые могут быть созданы одной их команд `CREATE`: `CREATE FUNCTION`, `CREATE TYPE`, `CREATE OPERATOR` и т.п.

Этот «набор» создается обновляется и удаляется как единый «пакет».

Когда вы даете команду `CREATE EXTENSION` вы создаете все объекты из этого «пакета».

Когда вы даете команду `DROP EXTENSION` все объекты пакета автоматически удаляются.

Функции могут быть написаны на интерпретируемом `PL/pgSQL` или на любом из `PL/*` языков. Или могут быть загружены из разделяемой `.so` библиотеки

Функцию можно зарегистрировать как обработчики hook'а. Это позволит переопределить поведение PostgreSQL.

# План доклада

PostgresPro

## В этом докладе мы пошагово  рассмотрим создание простейшего расширения.

## В основу доклада легли следующие источники:

https://github.com/IshaanAdarsh/Postgres-extension-tutorial/blob/main/SGML/Main.md

https://www.highgo.ca/2020/01/10/how-to-create-test-and-debug-an-extension-written-in-c-for-postgresql/

# Подготовка системы

PostgresPro

Все примеры проверены на Debian 12. Для не-debian based дистрибутивов могут быть малые отличия.

1. Сделайте себя админом Postgres. Это не правильно, но это упростит примеры

```
$ whoami
nataraj
```

```
$ sudo -u postgres createuser -s nataraj
```

2. Убедитесь что `-dev` паекты PostgreSQL установлены

```
sudo apt-get install postgresql-server-dev-all
```

Эта команда добавит заголовочные файлы, например `postgtres.h`

# Создаем основу расширения

PostgresPro

Создайте директорию для расширения и войдите в нее

```
$ mkdir my_first_extension

$ cd my_first_extension
```

Создайте в ней файл `my_first_extension.control` со следующим содержанием

```
comment = 'My Very First Extension'
default_version = '0.0.1'
```

# Создаем файл-схемы

PostgresPro

Создайте файл `my_first_extension--0.0.1.sql`:

```
CREATE TABLE my_first_extension_table (
  id SERIAL PRIMARY KEY,
  my_string VARCHAR NOT NULL
);

INSERT INTO my_first_extension_table (my_string)
VALUES('Hello, World');
```

# Создаем сборочный файл

PostgresPro

Создайте `Makefile`:

```
EXTENSION = my_first_extension
DATA = my_first_extension--0.0.1.sql

PG_CONFIG  ?= pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

# Все готово. Можно проверять

PostgresPro

## Сборка и установка

```
$ make
$ sudo make install
```

## Проверяем что все работает

```
$ sudo -u postgres psql

=# CREATE EXTENSION my_first_extension;
CREATE EXTENSION

=# SELECT * FROM my_first_extension_table;
 id |   my_string
----+--------------
  1 | Hello, World
(1 row)
```

# А если расширение удалить таблица исчезнет?

PostgresPro

В теории при удалении расширения таблица `my_first_extension_table` должна исчезнуть. Давайте проверим.

```
=# DROP EXTENSION my_first_extension;
DROP EXTENSION

=# SELECT * FROM my_first_extension_table;
ERROR:  relation "my_first_extension_table" does not exist
СТРОКА 1: SELECT * FROM my_first_extension_table
```

# Автоматическое тестирование

PostgresPro

Проверить руками – хорошо, но лучше когда проверка регулярно делается автоматически

Создайте файл `sql/my_first_extension-main-test.sql`:

```
CREATE EXTENSION my_first_extension;
SELECT * FROM my_first_extension_table;

DROP EXTENSION my_first_extension;
-- There should be no my_first_extension_table anymore
SELECT * FROM my_first_extension_table;
```

# Автоматическое тестирование

PostgresPro

Отредактируйте `Makefile`

```
EXTENSION = my_first_extension
DATA = my_first_extension--0.0.1.sql
REGRESS = my_first_extension-main-test

PG_CONFIG  ?= pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

# Автоматическое тестирование

PostgresPro

Запускаем тестирование

```
make installcheck
```

и получаем ошибку:

```
my_first_extension/expected/my_first_extension-main-
test.out: No such file or directory
```

Но вместе с этим появится директория `results` с файлом `my_first_extension-main-test.out` внутри

# Автоматическое тестирование

PostgresPro

Проверим содержимое `results/my_first_extension-main-test.out`

```
CREATE EXTENSION my_first_extension;
SELECT * FROM my_first_extension_table;
 id |  my_string
----+-------------
  1 | Hello, World
(1 row)

DROP EXTENSION my_first_extension;
-- There should be no my_first_extension_table anymore
SELECT * FROM my_first_extension_table;
ERROR:  relation "my_first_extension_table" does not exist
LINE 1: SELECT * FROM my_first_extension_table;
                      ^
```

# Автоматическое тестирование

PostgresPro

Содержимое файла соответствует нашим ожиданиям поэтому копируем `my_first_extension-main-test.out` из директории `results` в директорию `expected`

```
$ mkdir expected
$ cp results/my_first_extension-main-test.out expected
```

# Автоматическое тестирование

PostgresPro

## Снова запускаем тесты, теперь они должны пройти успешно

```
$ make installcheck
echo "+++ regress install-check in  +++" && /usr/lib/postgresql/15/lib/pgxs/src/makefiles/../..
+++ regress install-check in  +++
(using postmaster on Unix socket, default port)
============== dropping database "contrib_regression" ==============
SET
DROP DATABASE
============== creating database "contrib_regression" ==============
CREATE DATABASE
ALTER DATABASE
ALTER DATABASE
ALTER DATABASE
ALTER DATABASE
ALTER DATABASE
ALTER DATABASE
============== running regression test queries        ==============
test my_first_extension-main-test ... ok           54 ms

======================
 All 1 tests passed.
======================
```

# Вторая версия. Добавляем plpgsql функцию

PostgresPro

Давайте добавим простейшую plpgsql функцию в наше расширение.

```
CREATE FUNCTION my_one()
RETURNS INT
LANGUAGE plpgsql
AS
$$
BEGIN
    RETURN 1;
END;
$$;
```

Для этого понадобится увеличить номер версии расширения

# Вторая версия. Добавляем plpgsql функцию

PostgresPro

Так как мы изменяем объекты входящие в расширение, у нас изменится версия схемы. Новая версия схемы будет `0.0.2` и мы должны отразить эти изменния в коде проекта

1. Переименуйте основной файл схемы из `my_first_extension--0.0.1.sql` в `my_first_extension--0.0.2.sql`

2. Создайте файл для update'а схемы с версии `0.0.1` на вресию `0.0.2`: `my_first_extension--0.0.1--0.0.2.sql`

3. В оба этих файла добавьте текст функции `CREATE FUNCTION my_one()` приведенный на предыдущем слайде.

# Вторая версия. Добавляем plpgsql функцию

PostgresPro

Обновляем номер версии в `my_first_extension.control`

```
comment = 'My Very First Extension'
default_version = '0.0.2'
```

# Вторая версия. Добавляем plpgsql функцию

PostgresPro

Обновляем `Makefile`, добавив туда актуальные имена файлов схемы

```
EXTENSION = my_first_extension
DATA = my_first_extension--0.0.2.sql \
my_first_extension--0.0.1--0.0.2.sql

REGRESS = my_first_extension-main-test

PG_CONFIG  ?= pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

# Вторая версия. Добавляем plpgsql функцию

PostgresPro

Запускаем сборку, установку, создаем расширение и проверяем что новая функция работает как задумывалось.

```
$ make
$ sudo make install

$ sudo -u postgres psql

=# CREATE EXTENSION my_first_extension;
CREATE EXTENSION

s=# SELECT my_one();
 my_one
--------
      1
(1 row)
```

# Вторая версия. Добавляем plpgsql функцию

PostgresPro

## Не забудьте про тесты для новой функциональности!

(Самостоятельная работа)

# Третья версия. Добавляем C-функцию

PostgresPro

Создайте `my_first_extension.c`:

```
#include "postgres.h"
#include "fmgr.h"

PG_MODULE_MAGIC;

PG_FUNCTION_INFO_V1(my_get_two);

Datum
my_get_two(PG_FUNCTION_ARGS)
{
    PG_RETURN_INT32(2);
}
```

# Третья версия. Добавляем C-функцию

PostgresPro

Обновите `Makefile`:

```
EXTENSION = my_first_extension
DATA = my_first_extension--0.0.2.sql \
my_first_extension--0.0.1--0.0.2.sql
MODULE_big = my_first_extension
OBJS = my_first_extension.o

REGRESS = my_first_extension-main-test

PG_CONFIG  ?= pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

# Третья версия. Добавляем C-функцию

PostgresPro

Обновите `my_first_extension.control`:

```
comment = 'My Very First Extension'
default_version = '0.0.2'
module_pathname = '$libdir/my_first_extension'
```

# Третья версия. Добавляем C-функцию

PostgresPro

## Теперь можно скомпилировать C-файл

```
$ make
gcc -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement -Werror=vla -Wendif-
labels -Wmissing-format-attribute -Wimplicit-fallthrough=3 -Wcast-function-type -Wformat-security -
fno-strict-aliasing -fwrapv -fexcess-precision=standard -Wno-format-truncation -Wno-stringop-
truncation -g -g -O2 -fstack-protector-strong -Wformat -Werror=format-security -fno-omit-frame-pointer
-fPIC -I. -I./ -I/usr/include/postgresql/15/server -I/usr/include/postgresql/internal  -Wdate-time -
D_FORTIFY_SOURCE=2 -D_GNU_SOURCE -I/usr/include/libxml2   -c -o my_first_extension.o
my_first_extension.c
gcc -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement -Werror=vla -Wendif-
labels -Wmissing-format-attribute -Wimplicit-fallthrough=3 -Wcast-function-type -Wformat-security -
fno-strict-aliasing -fwrapv -fexcess-precision=standard -Wno-format-truncation -Wno-stringop-
truncation -g -g -O2 -fstack-protector-strong -Wformat -Werror=format-security -fno-omit-frame-pointer
-fPIC -shared -o my_first_extension.so my_first_extension.o -L/usr/lib/x86_64-linux-gnu  -Wl,-z,relro
-Wl,-z,now -L/usr/lib/llvm-14/lib  -Wl,--as-needed
/usr/bin/clang-14 -Wno-ignored-attributes -fno-strict-aliasing -fwrapv -Wno-unused-command-line-
argument -Wno-compound-token-split-by-macro -O2  -I. -I./ -I/usr/include/postgresql/15/server
-I/usr/include/postgresql/internal  -Wdate-time -D_FORTIFY_SOURCE=2 -D_GNU_SOURCE
-I/usr/include/libxml2  -flto=thin -emit-llvm -c -o my_first_extension.bc my_first_extension.c
```

Функция `my_get_two` создана, но пока не доступна из SQL-запросов

# Третья версия. Добавляем C-функцию

PostgresPro

Нижеследующий код сделает функцию добавленную доступной из SQL-команд под именем `my_two`

```
CREATE FUNCTION my_two()
RETURNS INT
AS 'MODULE_PATHNAME', 'my_get_two'
LANGUAGE C;
```

Для этого мы должны обновить схему до `0.0.3` версии. Сделайте это самостоятельно, используя обновление схемы с версии `0.0.1` до версии `0.0.2` приведенное выше, в качестве примера.

# Третья версия. Добавляем C-функцию

PostgresPro

Теперь мы можем установить обновленную версию и убедиться что она работает.

```
$ sudo make install

$ sudo -u postgres psql

=# ALTER EXTENSION my_first_extension UPDATE;
ALTER EXTENSION

postgres=# SELECT my_two();
 my_two
--------
      2
(1 row)
```

# Третья версия. Добавляем C-функцию

PostgresPro

## Не забудьте создать тесты для вновь добавленной функциональности

(Самостоятельная работа)

PostgresPro

# И это только начало

PostgresPro

# Вопросы


## Николай Шаплов

E-mail: n.shaplov@postgrespro.ru
Matix: @dhyan:nataraj.su


PostgresPro

# Последняя версия слайдов доступна на

gitlab.com/dhyannataraj/articles/-/tree/main/slides/2025 - Creating own extension