PostgresPro

Creating own extension

First step to become Postgres Developer

Nikolay Shaplov

May 2025, Nepal

What is Postgres extension

PostgresPro

Extension is a set of postgres objects that can be `CREATE`ed. Like `CREATE FUNCTION`, `CREATE TYPE` or `CREATE OPERATOR`.

This set is `CREATE`ed, `UPDATE`ed and `DROP`ped as a bundle.

When you do `CREATE EXTENSION` you it will create all object included into extension.

When you do `DROP EXTENSION` all that objects will be automatically removed.

Functions may be written in `PL/pgSQL` or other `PL/*` or you can load binary function from shared `.so` library.

You can register a function as a hook handler to change behavior if PostgreSQL.

Talk plan

PostgresPro

I will show step by step how simple extension can be created.

This talk is based on following tutorials

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/

Preparing your system

PostgresPro

I am doing that on Debian 12. For non-debian based linuxes couple of thing may differ.

Make yourself postgres admin. That will simplify some things:

```
$ whoami
nataraj
```

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

Make sure postgres `-dev` package is installed

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

It will bring you `postgtres.h`, you will need it later

Creating extension skeleton

PostgresPro

Create extension directory and enter it

```
$ mkdir my_first_extension

$ cd my_first_extension
```

Create `my_first_extension.control` file

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

Creating schema file

PostgresPro

Create `my_first_extension--0.0.1.sql` file

```
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');
```

Creating build file

PostgresPro

Create `Makefile` file

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

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

Let's check it

PostgresPro

Build and install it

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

Check it working

```
$ 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)
```

Let’s check dropping extension works

PostgresPro

Dropping the extension should destroy all object. Let’s check it

```
=# DROP EXTENSION my_first_extension;
DROP EXTENSION

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

Automatic testing

PostgresPro

Manual check is good, automatic tests are better

Create `sql/my_first_extension-main-test.sql` file

```
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;
```

Automatic testing

PostgresPro

Edit `Makefile` file

```
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)
```

Automatic testing

PostgresPro

Run testing

```
make installcheck
```

and get an error:

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

But you will also get `results` dir with `my_first_extension-main-test.out` in it

Automatic testing

PostgresPro

Let's check content of `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;
                      ^
```

Automatic testing

PostgresPro

Everything looks well so copy `my_first_extension-main-test.out` from `results` dir to `expected`

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

Automatic testing

PostgresPro

Run test again. Now they should pass

```
$ 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.
======================
```

Add plpgsql function. Creating second release

PostgresPro

Lets add a plpgsql function to an extension.

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

This will make us to bump schema version.

Add plpgsql function. Creating second release

PostgresPro

Our schema version will be `0.0.2` we should change it all around the extension code.

You should rename schema file `my_first_extension--0.0.1.sql` to `my_first_extension--0.0.2.sql`

Create a schema update file from `0.0.1` to `0.0.2`:
`my_first_extension--0.0.1--0.0.2.sql`

Add `CREATE FUNCTION my_one()` from the previous slide to both files.

Add plpgsql function. Creating second release

PostgresPro

Update `my_first_extension.control` file

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

Add plpgsql function. Creating second release

PostgresPro

Update `Makefile` file

```
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)
```

Add plpgsql function. Creating second release

PostgresPro

Now install updated version and check is is working properly.

```
$ sudo make install

$ sudo -u postgres psql

=# CREATE EXTENSION my_first_extension;
CREATE EXTENSION

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

Add plpgsql function. Creating second release

PostgresPro

Do not forget to create tests for new functionality!

(Do it yourself)

Add C function. Creating third release

PostgresPro

Create `my_first_extension.c` file

```
#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);
}
```

Add C function. Creating third release

PostgresPro

Update `Makefile` file

```
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)
```

Add C function. Creating third release

PostgresPro

Update `my_first_extension.control` file

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

Add C function. Creating third release

PostgresPro

Now you can build your C file

```
$ 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
```

But this function is still not available from SQL queries.

Add C function. Creating third release

PostgresPro

Add following code to schema to make this function available from SQL

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

Now we have switch to `0.0.3` schema version. Do it yourself. Follow guidelines from `0.0.1` to `0.0.2` update.

Add C function. Creating third release

PostgresPro

Now install updated version and check is is working properly.

```
$ sudo make install

$ sudo -u postgres psql

=# ALTER EXTENSION my_first_extension UPDATE;
ALTER EXTENSION

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

Add C function. Creating third release

PostgresPro

Do not forget about creating tests

(Do it yourself)

PostgresPro

That is only the first step!

PostgresPro

QA

Nikolay Shaplov

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

PostgresPro

Get current version of these slides at

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