PG 插件框架

开源开发实践-第十周

David & Cary

AGENDA
目 录

- What is a PG Extension
- PG Extension Framework
- Create Your Own Extension
- Test Your Extension

What is a PG Extension

What is PG Extension

PG Extension - A modular design

- Postgres is a huge database system consisting of a wide range of data types, functions, features and operators that can be utilized to solve many common to complex problems.
- However, in the world full of complex problems, sometimes these are just not enough depending on the use case complexities.
- Worry not, since Postgres version 9, it is possible to extend Postgres' s existing functionalities with the use of "extensions"

PostgreSQL uses "modular" design, like modular pipe homes in Hong Kong

https://www.scmp.com/lifestyle/interiors-living/article/2121891/how-hong-kongs-low-cost-housing-pipe-dream-became

What Extensions Do?

Anything you want ... really

- PostgreSQL source repository provides you with a list of extensions that you can use
- These default extensions allow you to do things like:
  - Check data corruption
  - Analyze query performance
  - Additional index methods
  - New data types
  - New functions to inspect your buffer blocks
  - Encrypt or decrypt user inputs
  - Pre-load data into memory from disk
  - …much, much more

Benefits of Extension

The Good and the Bad

The Good	The Bad
• No need to maintain the entire PostgreSQL source repository	• Cannot change the existing way How PostgreSQL core works
• No extra effort needed to remain compatible with new PostgreSQL versions	• Can only be loaded after the PostgreSQL server is started. So it is not possible to use an extension during initdb.
• Extremely flexible – Load only what you need	

PG Extension Framework

The Default Extensions

Know what you have first

- All the default PG extensions are located in the contrib/ folder in your source repository.
- They are by default "not built" with the rest of the PostgreSQL modules.
- So, in order to use them, you need to navigate to the "contrib/" folder and issue the build yourself
  - cd contrib
  - make
  - make install

Appendix F. Additional Supplied Modules

Table of Contents

https://www.postgresql.org/docs/current/contrib.html

2 Ways To Make an Extension

Know what you have first

- An extension can be created using…
  - C Programming language
    - Basically to create one or more additional SQL functions that PG can use to perform certain tasks
  - PL/pgSQL procedural language
    - Basically a collection of SQL instructions to complete certain tasks

```c
/*
 * Function returning data from the shared buffer cache - buffer number,
 * relation node/tablespace/database/blocknum and dirty indicator.
 */
PG_FUNCTION_INFO_V1(pg_buffercache_pages);

Datum
pg_buffercache_pages(PG_FUNCTION_ARGS)
{
    FuncCallContext *funcctx;
    Datum       result;
    MemoryContext oldcontext;
    BufferCachePagesContext *fctx;  /* User function context. */
    TupleDesc   tupledesc;
    TupleDesc   expected_tupledesc;
    HeapTuple   tuple;

    if (SRF_IS_FIRSTCALL())
    {
        int         i;

        funcctx = SRF_FIRSTCALL_INIT();
```

```postgresql
CREATE FUNCTION char_count(TEXT, CHAR)
RETURNS INTEGER
LANGUAGE plpgsql IMMUTABLE STRICT
  AS $$
    DECLARE
      charCount INTEGER := 0;
      i INTEGER := 0;
      inputText TEXT := $1;
      targetChar CHAR := $2;
    BEGIN
    WHILE i <= length(inputText) LOOP
      IF substring( inputText from i for 1) = targetChar THEN
        charCount := charCount + 1;
      END IF;
        i := i + 1;
      END LOOP;

    RETURN(charCount);
    END;
  $$;
```

PL/pgSQL – Based Extension

Extensions made in PL/pgSQL

- Each extension is created as a SQL file with suffix “.sql”
- You need to use the name convention:

  [name]--version.sql

- For example:
  - pageinspect--1.5.sql
  - dblink--1.2.sql
  - hstore--1.4.sql
- These procedural scripts can be loaded to PG at run time.

These are your PL/pgSQL extensions!

```
caryh@HGPC01:~/highgo/git/postgres.community2/postgres/highgo/share/extension$ ls
adminpack--1.0--1.1.sql          hstore--1.1--1.2.sql             pg_prewarm--1.1.sql
adminpack--1.0.sql               hstore--1.2--1.3.sql             pg_prewarm.control
adminpack--1.1--2.0.sql          hstore--1.3--1.4.sql             pgrowlocks--1.0--1.1.sql
adminpack.control                hstore--1.4--1.5.sql             pgrowlocks--1.1--1.2.sql
amcheck--1.0--1.1.sql            hstore--1.4.sql                  pgrowlocks--1.2.sql
amcheck--1.0.sql                 hstore--1.5--1.6.sql             pgrowlocks.control
amcheck--1.1--1.2.sql            hstore.control                   pgrowlocks--unpackaged--1.0.sql
amcheck.control                  hstore--unpackaged--1.0.sql      pg_stat_statements--1.0--1.1.sql
autoinc--1.0.sql                 insert_username--1.0.sql         pg_stat_statements--1.1--1.2.sql
autoinc.control                  insert_username.control          pg_stat_statements--1.2--1.3.sql
autoinc--unpackaged--1.0.sql     insert_username--unpackaged--1.0.sql pg_stat_statements--1.3--1.4.sql
bloom--1.0.sql                   intagg--1.0--1.1.sql             pg_stat_statements--1.4--1.5.sql
bloom.control                    intagg--1.1.sql                  pg_stat_statements--1.4.sql
btree_gin--1.0--1.1.sql          intagg.control                   pg_stat_statements--1.5--1.6.sql
btree_gin--1.0.sql               intagg--unpackaged--1.0.sql      pg_stat_statements--1.6--1.7.sql
btree_gin--1.1--1.2.sql          intarray--1.0--1.1.sql           pg_stat_statements.control
btree_gin--1.2--1.3.sql          intarray--1.1--1.2.sql           pg_stat_statements--unpackaged--1.0.sql
btree_gin.control                intarray--1.2.sql                pgstattuple--1.0--1.1.sql
btree_gin--unpackaged--1.0.sql   intarray.control                 pgstattuple--1.1--1.2.sql
btree_gist--1.0--1.1.sql         intarray--unpackaged--1.0.sql    pgstattuple--1.2--1.3.sql
btree_gist--1.1--1.2.sql         isn--1.0--1.1.sql                pgstattuple--1.3--1.4.sql
```

PL/pgSQL Extensions are installed to “$INSTALLDIR/share/extension”

C-Based Extension

Extensions made in C

- An extension can be written in "c" or in a procedural language called "PL/pgSQL"
- A C-based extension is built into a "shared library" denoted with suffix ".so"
- For example:
  - pgcrypto.so
  - pg_buffercache.so
  - pg_prewarm.so
- C-based extensions normally provide more complex functionalities to PostgreSQL comparing to PL/pgSQL-based extensions
- And PG can dynamically load them at run time.

These are your C extensions!

```
caryh@HGPC01:~/highgo/git/postgres.community2/postgres/highgo/lib$ ls
adminpack.so          euc_tw_and_big5.so      libpgcommon.a           pageinspect.so          tsm_system_rows.so
amcheck.so            file_fdw.so             libpgcommon_shlib.a     passwordcheck.so        tsm_system_time.so
ascii_and_mic.so      fuzzystrmatch.so        libpgfeutils.a          pg_buffercache.so       unaccent.so
auth_delay.so         hstore.so               libpgport.a             pgcrypto.so             utf8_and_ascii.so
auto_explain.so       insert_username.so      libpgport_shlib.a       pg_freespacemap.so      utf8_and_big5.so
autoinc.so            _int.so                 libpgtypes.a            pgoutput.so             utf8_and_cyrillic.so
bloom.so              isn.so                  libpgtypes.so           pg_prewarm.so           utf8_and_euc2004.so
btree_gin.so          latin2_and_win1250.so   libpgtypes.so.3         pgrowlocks.so           utf8_and_euc_cn.so
btree_gist.so         latin_and_mic.so        libpgtypes.so.3.12      pg_stat_statements.so   utf8_and_euc_jp.so
citext.so             libecpg.a               libpgtypes.so.3.13      pgstattuple.so          utf8_and_euc_kr.so
cube.so               libecpg_compat.a        libpgtypes.so.3.14      pg_trgm.so              utf8_and_euc_tw.so
cyrillic_and_mic.so   libecpg_compat.so       libpq.a                 pg_visibility.so        utf8_and_gb18030.so
dblink.so             libecpg_compat.so.3     libpq.so                pgxs                    utf8_and_gbk.so
dict_int.so           libecpg_compat.so.3.12  libpq.so.5              pkgconfig               utf8_and_iso8859_1.so
dict_snowball.so      libecpg_compat.so.3.13  libpq.so.5.12           plpgsql.so              utf8_and_iso8859.so
dict_xsyn.so          libecpg_compat.so.3.14  libpq.so.5.13           postgres_fdw.so         utf8_and_johab.so
earthdistance.so      libecpg.so              libpq.so.5.14           refint.so               utf8_and_sjis2004.so
euc2004_sjis2004.so   libecpg.so.6            libpqwalreceiver.so     seg.so                  utf8_and_sjis.so
euc_cn_and_mic.so     libecpg.so.6.12         lo.so                   tablefunc.so            utf8_and_uhc.so
euc_jp_and_sjis.so    libecpg.so.6.13         ltree.so                tcn.so                  utf8_and_win.so
euc_kr_and_mic.so     libecpg.so.6.14         moddatetime.so          test_decoding.so
```

C Extensions are installed to "$INSTALLDIR/lib"

C-Based Extension - cont.

Extensions made in C

- Though most extension logics are written in "C", it still needs to have some "SQL" to set up your new logics.
- For example, you define a new function in C, but you need to run "CREATE FUNCTION" sql commands to PG to register your new function...
- In other words...
  - A C-based extension still requires its own ".sql" scripts to act as entry point to your new C function.
  - A PL/pgSQL-based extension does not really require to have any C implementations because it is possible to implement everything using the ".sql" scripts

Example: pg_buffercache

An extension to see buffer manager status

- This extension has both “C” and “sql” part of implementation

The control file. Will explain later

The Makefile to build the shared library, pg_buffercache.so

The “.sql” driver scripts including all previous versions

The source file

```
caryh@HGPC01:~/highgo/git/postgres.community2/postgres/contrib/pg_buffercache$ ls
Makefile                    pg_buffercache.control
pg_buffercache--1.0--1.1.sql  pg_buffercache_pages.c
pg_buffercache--1.1--1.2.sql  pg_buffercache_pages.o
pg_buffercache--1.2--1.3.sql  pg_buffercache.so
pg_buffercache--1.2.sql       pg_buffercache--unpackaged--1.0.sql
```

contrib/pg_buffercache

The Control File – pg_buffercache example

The Information about your extension

- The control file is required, and it acts like a meta data for your extension.
- It follows this name convention

  [name].control

- It tells PG:
  - comment: The description of what your extension does
  - default_version: the default version of extension to load
  - module_pathname: where the C shared library files are located (without .so). Use $libdir variable here.
  - relocatable: whether the extension can be relocated. Put true here.

```
# pg_buffercache extension
comment = 'examine the shared buffer cache'
default_version = '1.3'
module_pathname = '$libdir/pg_buffercache'
relocatable = true
```

The SQL File – pg_buffercache example

The entry point to your extension

- After PG figures out the version information about the extension, it will first load your C shared library (if required) and then your .sql entry point script.
- In this script:
  - You want to put logics on how to properly use your extension
  - You want to hook up PG with the new C functions you created
  - You want to limit the script to run only when it is loaded via "CREATE EXTENSION"
  - Anything you want PG to do...really.

The SQL File - pg_buffercache example

The entry point to your extension

Limit the script to run only when loaded from "CREATE EXTENSION"

Tell PG about a new SQL function defined in the C shared library called "pg_buffercache_pages()"

Create a VIEW to beautify the output from pg_buffercache_pages()

Set restriction on function visibility if required

```sql
-- complain if script is sourced in psql, rather than via CREATE EXTENSION
\echo Use "CREATE EXTENSION pg_buffercache" to load this file. \quit

-- Register the function.
CREATE FUNCTION pg_buffercache_pages()
RETURNS SETOF RECORD
AS 'MODULE_PATHNAME', 'pg_buffercache_pages'
LANGUAGE C PARALLEL SAFE;

-- Create a view for convenient access.
CREATE VIEW pg_buffercache AS
        SELECT P.* FROM pg_buffercache_pages() AS P
        (bufferid integer, relfilenode oid, reltablespace oid, reldatabase oid,
         relforknumber int2, relblocknumber int8, isdirty bool, usagecount int2,
         pinning_backends int4);

-- Don't want these to be available to public.
REVOKE ALL ON FUNCTION pg_buffercache_pages() FROM PUBLIC;
REVOKE ALL ON pg_buffercache FROM PUBLIC;
```

contrib/pg_buffercache/pg_buffercache—1.2.sql

The source file - pg_buffercache example

The entry point to your extension

You have to put the keyword "PG_MODULE_MAGIC" macro on your C file to indicate that this is an extension's C file

```
PG_MODULE_MAGIC;
```

This is the implementation of the new SQL function that your extension will provide

Remember, we register this file in the .sql script?

```
/*
 * Function returning data from the shared buffer cache - buffer number,
 * relation node/tablespace/database/blocknum and dirty indicator.
 */
PG_FUNCTION_INFO_V1(pg_buffercache_pages);

Datum
pg_buffercache_pages(PG_FUNCTION_ARGS)
{
    FuncCallContext *funcctx;
    Datum       result;
    MemoryContext oldcontext;
    BufferCachePagesContext *fctx;  /* User function context. */
    TupleDesc   tupledesc;
    TupleDesc   expected_tupledesc;
    HeapTuple   tuple;

    if (SRF_IS_FIRSTCALL())
    {
```

contrib/pg_buffercache/pg_buffercache.c

Loading an Extension

Use "CREATE/DROP EXTENSION" Syntax

- Now that you have installed all the default extensions from PG, you are able to use them in your psql session
- If you don't know the name of extension, you can use the following to find out:

  SELECT pg_available_extensions();

  SELECT pg_available_extension_versions();

- Use "CREATE EXTENSION [$name]" syntax to load an extension to PG, where $name is the name of the extension without any suffix like (.so) or (.sql).
- For example:

  CREATE/DROP EXTENSION pg_buffercache;

```
postgres=# SELECT pg_available_extensions();
                                  pg_available_extensions
-----------------------------------------------------------------------------------------
 (pg_trgm,1.4,"text similarity measurement and index searching based on trigrams")
 (pg_stat_statements,1.7,"track execution statistics of all SQL statements executed")
 (earthdistance,1.1,"calculate great-circle distances on the surface of the Earth")
 (btree_gin,1.3,"support for indexing common datatypes in GIN")
 (intarray,1.2,"functions, operators, and index support for 1-D arrays of integers")
 (insert_username,1.0,"functions for tracking who changed a table")
 (file_fdw,1.0,"foreign-data wrapper for flat file access")
 (seg,1.3,"data type for representing line segments or floating-point intervals")
 (unaccent,1.1,"text search dictionary that removes accents")
 (dblink,1.2,"connect to other PostgreSQL databases from within a database")
 (dict_xsyn,1.0,"text search dictionary template for extended synonym processing")
```

```
postgres=# SELECT pg_available_extension_versions();
                                  pg_available_extension_versions
-----------------------------------------------------------------------------------------
 (pg_trgm,1.3,t,t,,,"text similarity measurement and index searching based on trigrams")
 (pg_trgm,1.4,t,t,,,"text similarity measurement and index searching based on trigrams")
 (pg_stat_statements,1.4,t,t,,,"track execution statistics of all SQL statements executed")
 (pg_stat_statements,1.6,t,t,,,"track execution statistics of all SQL statements executed")
 (pg_stat_statements,1.7,t,t,,,"track execution statistics of all SQL statements executed")
 (pg_stat_statements,1.5,t,t,,,"track execution statistics of all SQL statements executed")
 (earthdistance,1.1,t,t,,{cube},"calculate great-circle distances on the surface of the Earth")
 (btree_gin,1.0,t,t,,,"support for indexing common datatypes in GIN")
 (btree_gin,1.1,t,t,,,"support for indexing common datatypes in GIN")
 (btree_gin,1.2,t,t,,,"support for indexing common datatypes in GIN")
```

```
postgres=# CREATE EXTENSION pg_buffercache;
CREATE EXTENSION
```

Test Your Extension

Simple Extension Example - char_count

Simple Extension - char_count

- Let’s create a simple extension that providesth is new SQL function:

  integer **char_count**(text, char)

- It will count the number of occurrence of “char” in the string “text” and returns the count to the caller

For example:

```
SELECT char_count('aaaabbbbbbbcc1111222222233333335555590','x');
 char_count
------------
          0
(1 row)

SELECT char_count('aaaabbbbbbbcc1111222222233333335555590','c');
 char_count
------------
          2
```

Simple Extension Example – char_count

Simple Extension – char_count

- Step 1: Create your own contrib folder and change to the new directory
  - mkdir contrib/char_count
  - cd contrib/char_count
- Step 2: Create these folders in your extension folder to hold regression test cases
  - mkdir sql
  - mkdir expected

Simple Extension Example – char_count

Simple Extension – char_count

- Step 3: Create a control file for your extension with these contents:
  - char_count.control

```
# char_count extension
comment = 'function to count number of specified characters'
default_version = '1.0'
module_pathname = '$libdir/char_count'
relocatable = true
```

Simple Extension Example - char_count

Simple Extension - char_count

- Step 4: Create a SQL driver script for char_count. Start with version 1.0
  - char_coun--1.0.sql

PL/pgSQL Language

```
\echo Use "CREATE EXTENSION char_count" to load this file. \quit
CREATE FUNCTION char_count(TEXT, CHAR)
RETURNS INTEGER
LANGUAGE plpgsql IMMUTABLE STRICT
  AS $$
    DECLARE
      charCount INTEGER := 0;
      i INTEGER := 0;
      inputText TEXT := $1;
      targetChar CHAR := $2;
    BEGIN
    WHILE i <= length(inputText) LOOP
      IF substring( inputText from i for 1) = targetChar THEN
        charCount := charCount + 1;
      END IF;
        i := i + 1;
      END LOOP;

    RETURN(charCount);
    END;
  $$;
```

C Language

```
\echo Use "CREATE EXTENSION char_count" to load this file. \quit
CREATE FUNCTION char_count(TEXT, TEXT) RETURNS INTEGER
AS '$libdir/char_count'
LANGUAGE C IMMUTABLE STRICT
```

Simple Extension Example – char_count

Simple Extension – char_count

- Step 5: Create a Makefile for char_count
  - This Makefile is relatively simple
  - It is simply a bunch of variable declaration to tell the main Makefile what to do.

You need to put one more line here to specify the list of c files (without the .c prefix) here.

PL/pgSQL Language

```
# contrib/char_count/Makefile

EXTENSION = char_count
DATA = char_count--1.0.sql
PGFILEDESC = "char_count - count number of specified character"
REGRESS = char_count

ifdef USE_PGXS
PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
else
subdir = contrib/char_count
top_builddir = ../..
include $(top_builddir)/src/Makefile.global
include $(top_srcdir)/contrib/contrib-global.mk
endif
```

C Language

```
MODULES = char_count
EXTENSION = char_count
DATA = char_count--1.0.sql
PGFILEDESC = "char_count - count number of specified character"
REGRESS = char_count

ifdef USE_PGXS
PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
else
subdir = contrib/char_count
top_builddir = ../..
include $(top_builddir)/src/Makefile.global
include $(top_srcdir)/contrib/contrib-global.mk
endif
```

Simple Extension Example – char_count

Simple Extension – char_count

- Step 5 – 1 (C Language Only): Create the char_count.c source file with the implementation

This does the same thing as the PL/pgSQL language script we defined earlier but done in C.

```c
#include "postgres.h"
#include "fmgr.h"
#include "utils/builtins.h"

PG_MODULE_MAGIC;

PG_FUNCTION_INFO_V1(char_count_c);

Datum
char_count_c(PG_FUNCTION_ARGS)
{
        int charCount = 0;
        int i = 0;
        text * inputText = PG_GETARG_TEXT_PP(0);
        text * targetChar = PG_GETARG_TEXT_PP(1);

        int inputText_sz = VARSIZE(inputText)-VARHDRSZ;
        int targetChar_sz = VARSIZE(targetChar)-VARHDRSZ;
        char * cp_inputText = NULL;
        char * cp_targetChar = NULL;

        if ( targetChar_sz > 1 )
        {
                elog(ERROR, "arg1 must be 1 char long");
        }

        cp_inputText = (char *) palloc ( inputText_sz + 1);
        cp_targetChar = (char *) palloc ( targetChar_sz + 1);
        memcpy(cp_inputText, VARDATA(inputText), inputText_sz);
        memcpy(cp_targetChar, VARDATA(targetChar), targetChar_sz);

        elog(INFO, "arg0 length is %d, value %s", (int)strlen(cp_inputText), cp_inputText );
        elog(INFO, "arg1 length is %d, value %s", (int)strlen(cp_targetChar), cp_targetChar );

        while ( i < strlen(cp_inputText) )
        {
                if( cp_inputText[i] == cp_targetChar[0] )
                        charCount++;
                i++;
        }

        pfree(cp_inputText);
        pfree(cp_targetChar);
        PG_RETURN_INT32(charCount);
}
```

Simple Extension Example – char_count

Simple Extension – char_count

- Step 6: Build and install the extension within the char_count folder.
  - make
  - make install

- Step 7: Login to PG and use the new char_count extension

```
carytest=# CREATE EXTENSION char_count;
CREATE EXTENSION
carytest=# select char_count('1111111122223555556','1');
 char_count
------------
          8
(1 row)

carytest=#
```

That’s It!

Create Test Cases For Extension

Create Test Case for char_count example

Simple Extension - char_count

- Step 1: Navigate to the "sql" folder we created earlier in last chapter
  - cd contrib/char_count/sql

- Step 2: Create new test char_count.sql

```
CREATE EXTENSION char_count;
SELECT char_count('aaaabbbbbbbcc','a');
SELECT char_count('aaaabbbbbbbcc','b');
SELECT char_count('aaaabbbbbbbcc','c');
SELECT char_count('aaaabbbbbbbcc111122222222333333335555590','x');
SELECT char_count('aaaabbbbbbbcc111122222222333333335555590','c');
SELECT char_count('aaaabbbbbbbcc111122222222333333335555590','b');
SELECT char_count('aaaabbbbbbbcc111122222222333333335555590','5');
SELECT char_count('aaaabbbbbbbcc111122222222333333335555590','3');
SELECT char_count('aaaabbbbbbbcc111122222222333333335555590','2');
SELECT char_count('aaaabbbbbbbcc111122222222333333335555590','1');
SELECT char_count('aaaabbbbbbbcc111122222222333333335555590','0');
SELECT char_count('aaaabbbbbbbcc111122222222333333335555590','asd');
```

Note that we specify the name of our test in the Makefile already `REGRESS = char_count`

Create Test Case for char_count example

Simple Extension - char_count

- Step 3: In the char_count folder, run the regression test:
  - make installcheck
  - And you will find that the test will fail at first because we have not put our expected outputs to the "expected" folder
  - A new folder called "results" will be created that contains the results of the execution of the specified script

Create Test Case for char_count example

Simple Extension - char_count

- Step 4: Examine the results folder and make sure the output is right
  - The function should count number of characters correctly
- Step 5: Copy the result file to the expected folder.
  - cp char_count/results/char_count.out char_count/expected

```
CREATE EXTENSION char_count;
SELECT char_count('aaaabbbbbbbcc','a');
 char_count
------------
          4
(1 row)

SELECT char_count('aaaabbbbbbbcc','b');
 char_count
------------
          7
(1 row)

SELECT char_count('aaaabbbbbbbcc','c');
 char_count
------------
          2
(1 row)

SELECT char_count('aaaabbbbbbbcc111122222223333333555590','x');
 char_count
------------
          0
(1 row)

SELECT char_count('aaaabbbbbbbcc111122222223333333555590','c');
 char_count
------------
          2
(1 row)
```

That's It!

瀚高
HIGH GO

融知与行 瀚且高远

THANKS