Hooks in PostgreSQL

Who's Guillaume Lelarge?

- French translator of the PostgreSQL manual
- Member of pgAdmin's team
- Vice-treasurer of PostgreSQL Europe
- CTO of Dalibo

- Mail: guillaume@lelarge.info
- Twitter: g_lelarge
- Blog: http://blog.guillaume.lelarge.info

PostgreSQL

- Well known for its extensibility
- For example, a user can add
  - Types
  - Functions
  - Operators
  - Languages
  - Etc
- Extensions in 9.1
- Less known is the hook system

Hooks

- Interrupt, and modify behaviour
- Not known because
  - Not explained in the documentation
  - Usually quite recent
- Four kinds of hooks
  - Planner hooks
  - Executor hooks
  - Security/permissions hooks
  - PL/pgsql hooks

Planner hooks

Hook	Used in	Initial release
explain_get_index_name_hook		8.3
ExplainOneQuery_hook	IndexAdvisor	8.3
get_attavgwidth_hook		8.4
get_index_stats_hook		8.4
get_relation_info_hook	plantuner	8.3
get_relation_stats_hook		8.4
join_search_hook	saio	8.3
planner_hook	planinstr	8.3

Executor hooks

Hook	Used in	Initial release
ExecutorStart_hook	pg_stat_statements	8.4
ExecutorRun_hook	pg_stat_statements	8.4
ExecutorFinish_hook	pg_stat_statements	8.4
ExecutorEnd_hook	pg_stat_statements	8.4
ProcessUtility_hook	pgextwlist, pg_stat_statements	9.0

Security/permissions hooks

Hook	Used in	Initial release
check_password_hook	passwordcheck	9.0
ClientAuthentication_hook	auth_delay, sepgsql, etc	9.1
ExecutorCheckPerms_hook	sepgsql	9.1
fmgr_hook	sepgsql	9.1
needs_fmgr_hook	sepgsql	9.1
object_access_hook	sepgsql	9.1

PL/pgsql hooks

Hook	**Initial release**
func_setup	8.2
func_beg	8.2
func_end	8.2
stmt_beg	8.2
stmt_end	8.2

Used by

- pldebugger,
- plprofiler,
- log_functions.

And yet another one

Hook	Used in	Initial release
shmem_startup_hook	pg_stat_statements	8.4

How do they work inside PG

- Hooks consist of global function pointers
- Initially set to NULL
- When PostgreSQL wants to use a hook
  - It checks the global function pointer
  - And executes it if it is set

How do we set the function pointer?

- A hook function is available in a shared library
- At load time, PostgreSQL calls the _PG_init() function of the shared library
- This function needs to set the pointer
  - And usually saves the previous one!

How do we unset the function pointer?

- At unload time, PostgreSQL calls the _PG_fini() function of the shared library
- This function needs to unset the pointer
  - And usually restores the previous one!

Example with ClientAuthentication_hook

- Declaration of the function type
  - extract from src/include/libpq/auth.h, line 27

```
/* Hook for plugins to get control in ClientAuthentication() */
typedef void (*ClientAuthentication_hook_type) (Port *, int);
```

Example with ClientAuthentication_hook

- Declare, and set the global function pointer
  - extract from src/backend/libpq/auth.c, line 215

```
/*
 * This hook allows plugins to get control following client authentication,
 * but before the user has been informed about the results.  It could be used
 * to record login events, insert a delay after failed authentication, etc.
 */
ClientAuthentication_hook_type ClientAuthentication_hook = NULL;
```

Example with ClientAuthentication_hook

- Check, and execute
  - extract from src/backend/libpq/auth.c, line 580

```
if (ClientAuthentication_hook)
    (*ClientAuthentication_hook) (port, status);
```

Writing hooks

- Details on some hooks
  - ClientAuthentication
  - Executor_End
  - check_password
  - func_beg
- And various examples

ClientAuthentication_hook details

- Get control
  - After client authentication
  - But before informing the user
- Usefull to
  - Record login events
  - Insert a delay after failed authentication

ClientAuthentication_hook use

- Modules using this hook
  - auth_delay
  - sepgsql
  - connection_limits (https://github.com/tvondra/connection_limits)

ClientAuthentication_hook function

- Two parameters
  - f (Port *port, int status)
- Port is a complete structure described in include/libpq/libpq-be.h
  - remote_host, remote_hostname, remote_port, database_name, user_name, guc_options, etc.
- Status is a status code
  - STATUS_ERROR, STATUS_OK

Writing a ClientAuthentication_hook

- Example: forbid connection if a file is present
- Needs two functions
  - One to install the hook
  - Another one to check availability of the file, and allow or deny connection

Writing a ClientAuthentication_hook

- First, initialize the hook

```
static ClientAuthentication_hook_type prev_client_auth_hook = NULL;

/* Module entry point */
void
_PG_init(void)
{
    prev_client_auth_hook = ClientAuthentication_hook;
    ClientAuthentication_hook = my_client_auth;
}
```

Writing a ClientAuthentication_hook

- Check availability of the file, and allow or deny connection

```
static void my_client_auth(Port *port, int status)
{
    struct stat buf;

    if (prev_client_auth_hook)
        (*prev_client_auth_hook) (port, status);

    if (status != STATUS_OK)
        return;

    if(!stat("/tmp/connection.stopped", &buf))
        ereport(FATAL, (errcode(ERRCODE_INTERNAL_ERROR),
            errmsg("Connection not authorized!!")));
}
```

Executor hooks details

- Start
  - beginning of execution of a query plan
- Run
  - Accepts direction, and count
  - May be called more than once
- Finish
  - After the final ExecutorRun call
- End
  - End of execution of a query plan

Executor hooks use

- Usefull to get informations on executed queries
- Already used by
  - pg_stat_statements
  - auto_explain
  - pg_log_userqueries
    http://pgxn.org/dist/pg_log_userqueries/
  - query_histogram
    http://pgxn.org/dist/query_histogram/
  - query_recorder
    http://pgxn.org/dist/query_recorder/

ExecutorEnd_hook function

- One parameter
  - f(QueryDesc *queryDesc)
- QueryDesc is a structure described in include/executor/execdesc.h
  - CmdType, sourceTexte, Instrumentation, etc

Writing an ExecutorEnd_hook

- Example: log queries executed by superuser only
- Needs three functions
  - One to install the hook
  - One to uninstall the hook
  - And a last one to do the job :-)

Writing an ExecutorEnd_hook

- First, install the hook

```
/* Saved hook values in case of unload */
static ExecutorEnd_hook_type prev_ExecutorEnd = NULL;

void _PG_init(void)
{
  prev_ExecutorEnd = ExecutorEnd_hook;
  ExecutorEnd_hook = my_ExecutorEnd;
}
```

Writing an ExecutorEnd_hook

- The hook itself:
  - check if the user has the superuser attribute
  - log (or not) the query
  - fire the next hook or the default one

```
static void
my_ExecutorEnd(QueryDesc *queryDesc)
{
  Assert(query != NULL);

  if (superuser())
    elog(LOG, "superuser %s fired this query %s",
         GetUserNameFromId(GetUserId()),
         query);

  if (prev_ExecutorEnd)
    prev_ExecutorEnd(queryDesc);
  else
    standard_ExecutorEnd(queryDesc);
}
```

Writing an ExecutorEnd_hook

- Finally, uninstall the hook

```
void _PG_fini(void)
{
  ExecutorEnd_hook = prev_ExecutorEnd;
}
```

check_password hook details

- Get control
  - When CREATE/ALTER USER is executed
  - But before commiting
- Usefull to
  - Check the password according to some enterprise rules
  - Log change of passwords
  - Disallow plain text passwords
- Major issue
  - Less effective with encrypted passwords :-/

check_password hook use

- Usefull to check password strength
- Already used by
  - passwordcheck

check_password_hook function

- Five parameters
  - const char *username, const char *password, int password_type, Datum validuntil_time, bool validuntil_null
- password_type
  - PASSWORD_TYPE_PLAINTEXT
  - PASSWORD_TYPE_MD5

Writing a check_password_hook

- Example: disallow plain text passwords
- Needs two functions
  - One to install the hook
  - One to check the password type

Writing a check_password_hook

- First, install the hook

```
void _PG_init(void)
{
  check_password_hook = my_check_password;
}
```

Writing a check_password_hook

- The hook itself:
  - check if the password is encrypted

```
static void
my_check_password(const char *username,
  const char *password, int password_type,
  Datum validuntil_time, bool validuntil_null)
{
  if (password_type == PASSWORD_TYPE_PLAINTEXT)
  {
    ereport(ERROR,
      (errcode(ERRCODE_INVALID_PARAMETER_VALUE),
       errmsg("password is not encrypted")));
  }
}
```

func_beg details

- Get control
  - Before BEGIN block of a PL/pgsql function
- Usefull to
  - Log start of each function
  - Profile functions
  - Debug functions

func_beg use

- Modules using this hook
  - pldebugger
  - plprofiler
  - log_functions (https://github.com/gleu/log_functions)

func_beg function

- Two parameters
  - f (PLpgSQL_execstate *estate, PLpgSQL_function *func)
- estate is a complete structure described in src/pl/plpgsql/plpgsql.h
- func is a complete structure described in src/pl/plpgsql/plpgsql.h
  - Name, OID, return type, ...

Writing a func_beg

- Example: log each function executed
- Needs two functions
  - One to install the hook
  - Another one to log the function name

Writing a func_beg

- First, initialize the hook

```
static PLpgSQL_plugin plugin_funcs = { my_func_beg };

void _PG_init(void)
{
  PLpgSQL_plugin ** var_ptr = (PLpgSQL_plugin **)
                  find_rendezvous_variable("PLpgSQL_plugin");
  *var_ptr = &plugin_funcs;
}

void load_plugin(PLpgSQL_plugin *hooks)
{
  hooks->func_beg = my_func_beg;
}
```

Writing a func_beg

- Log function name

```
static void my_func_beg(PLpgSQL_execstate *estate,
                              PLpgSQL_function  *func)
{
    elog(LOG, "Execute function %s", func->fn_name);
}
```

Compiling hooks

- Usual Makefile

```
MODULE_big = your_hook
OBJS = your_hook.o

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

Compiling hooks – example

- Make is your friend (and so is pg_config)

```
$ make USE_PGXS=1
gcc -O2 -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-
   statement -Wendif-labels -Wformat-security -fno-strict-aliasing -fwrapv
   -fexcess-precision=standard -fpic -I. -I. -I/opt/postgresql-
   9.1/include/server -I/opt/postgresql-9.1/include/internal -D_GNU_SOURCE
   -c -o your_hook.o your_hook.c
gcc -O2 -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-
   statement -Wendif-labels -Wformat-security -fno-strict-aliasing -fwrapv
   -fexcess-precision=standard -fpic -shared -o your_hook.so
   only_encrypted_passwords.o -L/opt/postgresql-9.1/lib -Wl,--as-needed -Wl,-
   rpath,'/opt/postgresql-9.1/lib',--enable-new-dtags
```

- Can't use PGXS with PL/pgsql plugins
  - But will be possible in 9.2 (thanks to Heikki for working on the patch)

Installing hooks – from source

- Make is still your friend

```
$ make USE_PGXS=1 install
/bin/mkdir -p '/opt/postgresql-9.1/lib'
/bin/sh /opt/postgresql-9.1/lib/pgxs/src/makefiles/../../config/install-sh -c
   -m 755  your_hook.so '/opt/postgresql-9.1/lib/your_hook.so'
```

PGXS

- It's better to rely only on PGXS (if possible)
- Makefile looks like this:

```
MODULE_big = your_hook
OBJS = your_hook.o

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

- So much simpler...

Using hooks with shared_preload_libraries

- Install the shared library
- In postgresql.conf
  - shared_preload_libraries
  - And possibly other shared library GUCs
- Restart PG

Using hooks – example

- Install the hook...
- In postgresql.conf

```
shared_preload_libraries = 'only_encrypted_passwords'
```

- Restart PostgreSQL

```
$ pg_ctl start
server starting
2012-01-28 16:01:32 CET  LOG:  loaded library "only_encrypted_passwords"
```

Using hooks – example

- Use the hook...

```
postgres=# CREATE USER u1 PASSWORD 'supersecret';
ERROR:  password is not encrypted

postgres=# CREATE USER u1 PASSWORD 'md5f96c038c1bf28d837c32cc62fa97910a';
CREATE ROLE

postgres=# ALTER USER u1 PASSWORD 'f96c038c1bf28d837c32cc62fa97910a';
ERROR:  password is not encrypted

postgres=# ALTER USER u1 PASSWORD 'md5f96c038c1bf28d837c32cc62fa97910a';
ALTER ROLE
```

Using hooks with LOAD statement

- Install the shared library
- LOAD the library
- ... and use it

Using hooks – example

- Install the hook...
- Create the function, and use it:

```
postgres=# CREATE FUNCTION f1() RETURNS boolean LANGUAGE plpgsql AS $$
postgres$# BEGIN
postgres$# PERFORM pg_sleep(5);
postgres$# RETURN true;
postgres$# END
postgres$# $$;
CREATE FUNCTION
hooks=# SET client_min_messages TO log;
LOG:  duration: 0.132 ms  statement: SET client_min_messages TO log;
SET
hooks=# SELECT f1();
LOG:  duration: 5003.180 ms  statement: SELECT f1();
 f1
----
 t
(1 row)
```

Using hooks – example

- LOAD the shared library, and use it...

```
hooks=# LOAD 'logplpgsql';
LOG:  duration: 0.373 ms  statement: LOAD 'logplpgsql';
LOAD
hooks=# SELECT f1();
LOG:  Execute function f1
LOG:  duration: 5001.466 ms  statement: SELECT f1();
[...]

hooks=# SELECT f1() FROM generate_series(1, 5);
LOG:  Execute function f1
LOG:  Execute function f1
LOG:  Execute function f1
LOG:  Execute function f1
LOG:  Execute function f1
LOG:  duration: 25006.701 ms  statement: SELECT f1() FROM generate_series(1,
   5);
 [...]
```

9.2 hooks

- One old hook with enhanced capability
- PGXS support for PL/pgsql hooks
- Two new hooks
  - A logging hook
  - And another planer hook

9.2 – Enhanced object_access_hook

- DROP statement support for object_access_hook
- Used by sepgsql

9.2 hooks – the logging hook

- Logging hook, by Martin Pihlak
  - emit_log_hook
  - Intercept messages before they are sent to the server log
  - Custom log filtering
  - Used by pg_journal (http://www.pgxn.org/dist/pg_journal/0.1.0/)

9.2 hooks – the planner hook

- Planner hook, by Peter Geoghegan
  - post_parse_analyze_hook
  - Get control at end of parse analysis
  - Query normalisation within pg_stat_statements

Conclusion

- Hooks are an interesting system to extend the capabilities of PostgreSQL
- Be cautious to avoid adding many of them
- We need more of them :-)

- Examples and slides available on:
  - https://github.com/gleu/Hooks-in-PostgreSQL