Hacking PostgreSQL

Stephen Frost
Crunchy Data
stephen@crunchydata.com

PGConf.EU 2018
October 24, 2018

Stephen Frost

- Chief Technology Officer @ Crunchy Data
- Committer, PostgreSQL
- Major Contributor, PostgreSQL
- PostgreSQL Infrastructure Team
- Default roles
- Row-Level Security in 9.5
- Column-level privileges in 8.4
- Implemented the roles system in 8.3
- Contributions to PL/pgSQL, PostGIS

Top Level Source Directory

Directory	Description
config	Config system for driving the build
contrib	Source code for Contrib Modules, aka, Extensions
doc	Documentation (SGML)
src/backend	PostgreSQL Server ("Back-End")
src/bin	psql, pg_dump, initdb, pg_upgrade, etc ("Front-End")
src/common	Code common to the front and back ends
src/fe_utils	Code useful for multiple front-end utilities
src/include	Header files for PG, mainly back-end
src/include/catalog	Definition of the PostgreSQL catalog tables (pg_catalog.* tables)
src/interfaces	Interfaces to PG, including libpq, ECPG
src/pl	Core Procedural Languages (plpgsql, plperl, plpython, tcl)
src/port	Platform-specific hacks
src/test	Regression tests
src/timezone	Timezone code from IANA
src/tools	Developer tools (including pgindent)

Backend Code - Down the Rabbit Hole

Directory	Description
access	Methods for accessing different types of data (heap, btree indexes, gist/gin, etc).
bootstrap	Routines for running PostgreSQL in "bootstrap" mode (by initdb)
catalog	Routines used for modifying objects in the PG Catalog (pg_catalog.*)
commands	User-level DDL/SQL commands (CREATE/ALTER, VACUUM/ANALYZE, COPY, etc)
executor	Executor, runs queries after they have been planned/optimized
foreign	Handles Foreign Data Wrappers, user mappings, etc
jit	Provider independent Just-In-Time Compilation infrastructure
lib	Code useful for multiple back-end components
libpq	Backend code for talking the wire protocol
main	main(), determines how the backend PG process is starting and starts right subsystem
nodes	Generalized "Node" structure in PG and functions to copy, compare, etc
optimizer	Query optimizer, implements the costing system and generates a plan for the executor
parser	Lexer and Grammar, how PG understands the queries you send it
partitioning	Common code for declarative partitioning in PG
po	Translations of backend messages to other languages

Backend Code - Part 2

Directory	Description
port	Backend-specific platform-specific hacks
postmaster	The "main" PG process that always runs, answers requests, hands off connections
regex	Henry Spencer's regex library, also used by TCL, maintained more-or-less by PG now
replication	Backend components to support replication, shipping WAL logs, reading them in, etc
rewrite	Query rewrite engine, used with RULEs, also handles Row-Level Security
snowball	Snowball stemming, used with full-text search
statistics	Extended Statistics system (CREATE STATISTICS)
storage	Storage layer, handles most direct file i/o, support for large objects, etc
tcop	"Traffic Cop"- this is what gets the actual queries, runs them, etc
tsearch	Full-Text Search engine
utils	Various back-end utility components, cacheing system, memory manager, etc

What do you want to change?

- Is your idea a new backend command?
- Or a new backslash command for psql?
- Maybe an improvement to pgbench?
- Looking for a way to improve performance?
- Add a new authentication method?
- Support another TLS/SSL/Encryption library?

Let's chat about changing an existing backend command...

Hacking the backend

Where to start when thinking about hacking the backend?

- Depends on your idea, but I prefer the grammar
- Grammar drives a lot
- Also one of the harder places to get agreeement
- Where is the grammar? It's in the parser.

What is a Parser?

Parser vs. Grammar

- Parser consists of two pieces- the Lexer and the Grammar
- Lexer determines how to tokenize the input
- Grammar defines what tokens can be used with each other and how
- While parsing, the grammar collects information about the command
- Once a full command is parsed, a function is called from the grammar

Where is the parser?

- The parser is in src/backend/parser
- In that directory are:
  - scan.l - Lexer, handles tokenization
  - gram.y - Definition of the grammar
  - parse_*.c - Specialized routines for parsing things
  - analyze.c - Tranforms raw parse tree into a Query
  - scansup.c - Support routines for the lexer

Modifying the grammar

- The grammar is a set of "productions" in gram.y
- "main()" is the "stmt" production
- Lists the productions for all of the top-level commands
- "—" is used to indicate "this OR that"

```
stmt :
            AlterEventTrigStmt
            | AlterCollationStmt
            | AlterDatabaseStmt
...
| CopyStmt
...
```

What about the COPY statement?

- These are the top-level COPY productions
- They refer to other productions though...

```
CopyStmt:   COPY opt_binary qualified_name opt_column_list opt_oids
            copy_from opt_program copy_file_name copy_delimiter opt_with copy_options
            ...
            | COPY '(' PreparableStmt ')' TO opt_program copy_file_name opt_with copy_options
            ...
```

COPY productions

- These are the other COPY productions

```
copy_from:
            FROM                                { $$ = true; }
            | TO                                { $$ = false; }
        ;

opt_program:
            PROGRAM                             { $$ = true; }
            | /* EMPTY */                       { $$ = false; }
        ;
...
copy_file_name:
            Sconst                              { $$ = $1; }
            | STDIN                             { $$ = NULL; }
            | STDOUT                            { $$ = NULL; }
        ;

copy_options: copy_opt_list                     { $$ = $1; }
            | '(' copy_generic_opt_list ')'     { $$ = $2; }
        ;
...
```

COPY productions

- Multi-value productions look like this

```
copy_generic_opt_list:
            copy_generic_opt_elem
                {
                    $$ = list_make1($1);
                }
            | copy_generic_opt_list ',' copy_generic_opt_elem
                {
                    $$ = lappend($1, $3);
                }
        ;

copy_generic_opt_elem:
            ColLabel copy_generic_opt_arg
                {
                    $$ = makeDefElem($1, $2, @1);
                }
        ;

copy_generic_opt_arg:
            opt_boolean_or_string           { $$ = (Node *) makeString($1); }
            | NumericOnly                   { $$ = (Node *) $1; }
            | '*'                           { $$ = (Node *) makeNode(A_Star); }
```

COPY productions

- Note the C template code in the grammar
- Compiled as part of the overall parser in gram.c
- "$$" is "this node"
- "$1" is the whatever the first value resolves to
- "$3" is the whatever the third value resolves to

```
copy_generic_opt_list:
            copy_generic_opt_elem
                {
                    $$ = list_make1($1);
                }
            | copy_generic_opt_list ',' copy_generic_opt_elem
                {
                    $$ = lappend($1, $3);
                }
        ;
```

COPY options list

- Production of COPY options

```
copy_opt_item:
            BINARY
                {
                    $$ = makeDefElem("format", (Node *)makeString("binary"), @1);
                }
            | OIDS
                {
                    $$ = makeDefElem("oids", (Node *)makeInteger(true), @1);
                }
            | FREEZE
                {
                    $$ = makeDefElem("freeze", (Node *)makeInteger(true), @1);
                }
            ...
```

Adding a new COPY option

- Add to the copy_opt_item production
- Modify the C template(s) as needed
- Also need to update the list of tokens / key words, kwlist.h
- Has to be added to unreserved_keyword production
- Always try to avoid adding any kind of reserved keyword

```
copy_opt_item:
              BINARY
                 {
                     $$ = makeDefElem("format", (Node *)makeString("binary"), @1);
                 }
 +            | COMPRESSED
 +               {
 +                   $$ = makeDefElem("compressed", (Node *)makeInteger(true), @1);
 +               }
              | OIDS
                 {
                     $$ = makeDefElem("oids", (Node *)makeInteger(true), @1);
                 }
              ...
```

What about the code?

- The code for COPY is in src/backend/commands/copy.c
- COPY has a function to process the options given
- Conveniently, this function is ProcessCopyOptions()
- CopyStateData exists to keep track of the COPY operation
- Not in a .h since only COPY uses it
- When defining a structure in a .c, put it near the top

```
typedef struct CopyStateData
{
...
      bool        binary;         /* binary format? */
      bool        oids;           /* include OIDs? */
 +    bool        compressed;     /* compressed file? */
      bool        freeze;         /* freeze rows on loading? */
...
```

What about the code?

- Add in the code to handle the option passed in

```
ProcessCopyOptions(CopyState cstate,
...
         }
 +       else if (strcmp(defel->defname, "compressed") == 0)
 +       {
 +#ifdef HAVE_LIBZ
 +           if (cstate->compressed)
 +               ereport(ERROR,
 +                       (errcode(ERRCODE_SYNTAX_ERROR),
 +                        errmsg("conflicting or redundant options")));
 +           cstate->compressed = defGetBoolean(defel);
 +#else
 +           ereport(ERROR,
 +                   (errcode(ERRCODE_SYNTAX_ERROR),
 +                    errmsg("Not compiled with zlib support.")));
 +#endif
 +       }
         else if (strcmp(defel->defname, "oids") == 0)
...
```

Is that it?

Not hardly.

- Further changes to copy.c for a COMPRESSED state
- Changes to track gzFile instead of FILE*
- Also have to use gzread()/gzwrite()
- Documentation updates in doc/src/sgml/ref/copy.sgml
- Regression test updates
- Resulting diffstat:

```
doc/src/sgml/ref/copy.sgml          |  12 ++
src/backend/commands/copy.c         | 458 +++++++++++++++++++++++++++++++++++++++++++++++++++++-----
src/backend/parser/gram.y           |   9 +-
src/backend/storage/file/fd.c       |  97 ++++++++++++
src/include/parser/kwlist.h         |   1 +
src/include/storage/fd.h            |   9 ++
src/test/regress/input/copy.source  |  20 +++
src/test/regress/output/copy.source |  18 +++
8 files changed, 583 insertions(+), 41 deletions(-)
```

PostgreSQL Subsystems

PostgreSQL has specific ways of handling

- Memory management
- Error logging / cleanup
- Linked lists (multiple ways...)
- Catalog lookups
- Nodes
- Datums and Tuples

Memory Management

- All memory is part of a memory context
- Allocated through palloc()
- Contexts exist for most of what you would expect
  - CurrentMemoryContext - what palloc() will use
  - TopMemoryContext - Backend Lifetime
  - Per-Query Context
  - Per-Tuple Context

Errors and Asserts

- Internal "can't happen" cases can use elog()
  - Always runs
  - Should not be used where a user might see it
  - May be useful for debugging
- Assert() is also available
  - Only runs in Assert-enabled builds
  - Be wary of making Assert builds act differently from non-Assert builds
  - Useful to make sure other hackers are using function properly

Logging from PostgreSQL

- Use ereport() with errcode() and errmsg()
- error level and errmsg() are required
- PG has a style guide for error messages
- ERROR or higher and PG will handle most cleanup
  - Rolls back transaction
  - Frees appropriate memory contexts

```
+           if (gzwrite(cstate->copy_gzfile, fe_msgbuf->data,
+                       fe_msgbuf->len) != fe_msgbuf->len)
+               ereport(ERROR,
+                       (errcode_for_file_access(),
+                        errmsg("could not write to COPY file: %m")));
```

SysCache and Scanning Catalogs

- General function 'SearchSysCache'
- Defined in utils/cache/syscache.c
  - Search a system catalog based on some key
  - Up to four keys can be used
  - Helper routines for fewer keys available (SearchSysCache1, etc)
  - Must call ReleaseSysCache() when done with a tuple
- Also some convenience routines in lsyscache.c
- Look for existing routines before implementing a new one

Nodes

- PostgreSQL expression trees are made up of Nodes
- Each node has a type, plus appropriate data
- 'type' of a Node is stored in the Node, allowing IsA() function
- Nodes created using makeNode(TYPE)
- Used extensivly by the grammar, but also elsewhere
- To add a new Node type
  - Add to include/nodes/nodes.h
  - Create make / copy / equality funcs in backend/nodes/

Datums

- General structure for a given single value
- Defined in postgres.h
- Lots of helper routines for working with Datums
  - Int32GetDatum(int) - Returns Datum representation of an Int32
  - DatumGetInt32(Datum) - Returns int32 from a Datum
  - Many others for each data type
- Datums may be stored "out-of-line" (aka TOAST'd)

Tuples

- Tuples are essentially "rows", comprised of Datums and other things
- Heap Tuple defined in include/access/htup.h
- HeapTupleData is in-memory construct
- Provides length of tuple, pointer to header
- Many different uses
  - Pointer to disk buffer (must be pin'd)
  - Empty
  - Single pmalloc'd chunk
  - Seperately allocated
  - Minimal Tuple structure

Tuples - continued

- HeapTupleHeaderData and friends are in htup_details.h
- Number of attributes
- Provides various flags (NULL bitmap, etc)
- Data follows the header (not in the struct)
- Lots of macros for working with tuples in details

Other Subsystems

- Many simple things have already been written and generalized
- Generalized code should go into 'src/backned/lib/'
- Look for existing code
  - Existing code is already portable
  - Already been tested
  - Includes regression tests
  - Means you have less to write

Selection of Subsystems

- Simple Linked List implementation - pg_list.h, list.c
- Integrated/inline doubly- and singly- linked lists - ilist.h, ilist.c
- Binary Heap implementation- binaryheap.c
- Hopcroft-Karp maximum cardinality algorithm for bipartite graphs - bipartite_match.c
- Bloom Filter - bloomfilter.c
- Dynamic Shared Memory Based Hash Tables - dshash.c
- HyperLogLog cardinality estimator - hyperloglog.c
- Knapsack problem solver - knapsack.c
- Pairing Heap implementation - pairingheap.c
- Red-Black binary tree - rbtree.c
- String handling - stringinfo.c

pgsql-hackers

Primary mailing list for discussion of PostgreSQL development

- Get a PostgreSQL Account at `https://postgresql.org/account`
- Subscribe at `https://lists.postgresql.org`
- Discuss your ideas and thoughts about how to improve PostgreSQL
- Watch for others working on similar capabilities
- Try to think about general answers, not specific
- Be supportive of other ideas and approaches
- What happened to COPY ... COMPRESSED ?
  - Send and receive COPY data from program instead
  - COPY ... PROGRAM 'zcat ...'
  - Not quite identical but large overlap
  - Simpler in a few ways than direct zlib support

Code Style

- Try to make your code 'fit in'
- Follow the PG style guide in the Developer FAQ
- Beware of copy/paste
- Aim to be C99-compliant (with caveats)
- Comments
  - C-style comments only, no C++
  - Generally on their own lines
  - Describe why, not what or how
  - Big comment blocks for large code blocks
  - Functions, big conditions or loops

Error Message Style

- Three main parts to an error message
  - Primary message
  - Detail information
  - Hint, if appropriate
- Do not make assumptions about formatting
- Do not end an error message with a newline
- Use double-quotes when quoting
- Quotes used for filenames, user identifiers, and other variables
- Avoid using passive voice- use active voice, PostgreSQL is not a human

Git crash-course

- Clone down the repo-
  - git clone https://git.postgresql.org/git/postgresql.git
  - Creates postgresql directory as a git repo
- cd into postgresql
- Create a branch to work on
  - git checkout -b myfeature
  - Creates a local branch called myfeature
- Hack on PostgreSQL! Make changes!
- Commit changes and build a diff
  - git add files changes
  - git commit
  - git branch –set-upstream-to=origin/master myfeature
  - git format-patch @{u} –stdout >myfeature.patch

Submitting Your Patch

- Patch format
  - Context diff or git-diff
  - Ideally, pick which is better
  - Multiple patches in one email- do not multi-email
- Include in email to -hackers
  - Description of the patch
  - Regression tests
  - Documentation updates
  - pg_dump support, if appropriate
- Register patch on `https://commitfest.postgresql.org`

Questions?

Thanks!