Hacking PostgreSQL

PGCon 2013
Ottawa, Canada

Stephen Frost
sfrost@snowman.net

Stephen Frost

- PostgreSQL
  - Major Contributor
  - Implemented Roles in 8.3
  - Column-Level Privileges in 8.4
  - Contributions to PL/pgSQL, PostGIS
- Resonate, Inc.
  - Principal Database Engineer
  - Online Digital Media Company
  - We're Hiring! * techjobs@resonateinsights.com

Do you read...

- planet.postgresql.org

PostgreSQL Source

- Overall PG source tree structure

```
contrib - contrib modules (Might become extensions, one day..)
doc     - Documentation (SGML)
src     - PostgreSQL "core" (C code, mostly)
...
src/backend     - PostgreSQL server ("Back-End")
src/bin         - psql, pg_dump, initdb, etc ("Front-End")
src/common      - Code common to front & back
src/include     - .h files, and friends
src/interfaces  - libpq, ecpg
src/pl          - Core procedural languages (plpgsql, plperl, tcl, etc)
src/port        - Platform-specific hacks
src/tools       - Developer tools (pgindent, etc)
```

Down the Rabbit Hole..

- Components of the backend (src/backend/...)

```
access      - Methods for accessing different types of data (heap, btree indexes, gist/gin, etc).
catalog     - Definition of the PG tables (pg_catalog.*)
commands    - User-level SQL commands (ALTER, CREATE TABLE, VACUUM, etc)
executor    - Duh, the Executor- runs the queries after planning / optimization
foreign     - Handles Foreign Data Wrappers, user mappings, etc
lib         - "General Purpose" / "Misc" functions (but they are elsewhere too..)
libpq       - Backend interface to talk to libpq, aka the wireline protocol
main        - main(), determines how the backend PG process is starting and hands off to the 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
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
snowball    - Snowball stemming, used with full-text search
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
```

So you have an idea..

- Where to begin?
  - Depends on your idea, but I prefer the parser
  - Grammar drives a lot of things
  - Also one of the hardest items to get agreement on
- The grammar is in src/backend/parser/
  - scan.l - lexer, handles tokenization
  - gram.y - actual grammar
  - Built with flex (lexer) and bison (parser)
  - Rarely have to change the lexer

Modifying the grammar

- Grammar is a set of productions
  - "main" is the 'stmt' production
  - Lists all the top-level commands
  - Each is its own production then

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

```
CopyStmt : COPY opt_binary qualified_name opt_column_list opt_oids
           copy_from opt_program copy_file_name copy_delimiter opt_with copy_options
           {
               CopyStmt * n = makeNode(CopyStmt);
               n->relation = $3;
```

Modifying CopyStmt

- Add it into the COPY production
- Modify the C template code as needed
  - C code is extracted by bison
  - Run through a set of changes (eg: changes "$3")
  - Compiled as part of the overall parser (gram.c)
- Remember to update the keywords list (kwlist.h)
- Also remember to add to unreserved_keywords
- Try to avoid creating new *reserved* keywords

Adding an option to COPY

```
--- a/src/backend/parser/gram.y
+++ b/src/backend/parser/gram.y
@@ -521,8 +521,8 @@ static void processCASbits(int cas_bits, int location, const char * constrType,
-   COMMITTED CONCURRENTLY CONFIGURATION CONNECTION CONSTRAINT CONSTRAINTS
-   CONTENT_P CONTINUE_P CONVERSION_P COPY COST CREATE
+   COMMITTED COMPRESSED CONCURRENTLY CONFIGURATION CONNECTION CONSTRAINT
+   CONSTRAINTS CONTENT_P CONTINUE_P CONVERSION_P COPY COST CREATE
@@ -2403,6 +2403,10 @@ copy_opt_item:
                {
                    $$ = makeDefElem("header", (Node * )makeInteger(TRUE));
                }
+           | COMPRESSED
+               {
+                   $$ = makeDefElem("compressed", (Node * )makeInteger(TRUE));
+               }
            | QUOTE opt_as Sconst
                {
                    $$ = makeDefElem("quote", (Node * )makeString($3));
@@ -12471,6 +12475,7 @@ unreserved_keyword:
            | COMMITTED
+           | COMPRESSED
            | CONFIGURATION
```

What about the code?

- COPY has a function to process options
  - Surprise, it's called "ProcessCopyOptions"
  - COPY is defined in backend/commands/copy.c
- COPY state info
  - Local state structure CopyStateData also in copy.c
  - Not in a .h because only COPY needs it
  - Define structures in .c files near the top

Option handling in copy.c

```
@@ -109,6 +119,7 @@ typedef struct CopyStateData
    bool        binary;         /* binary format? * /
+   bool        compressed;     /* compressed file? * /
    bool        oids;           /* include OIDs? * /
@@ -889,6 +1186,20 @@ 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)
```

That's it, right?

- Not hardly.
- Lots of changes to copy.c
  - New 'COMPRESSED' state
  - Tracking gzFile instead of FILE*
  - Using gzread / gzwrite instead of read/write
- Data in and out
  - All is buffered with 2 buffers
  - Uncompressed data
  - Compressed data

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(-)
```

- Documentation updates in doc/src
- Modify fd.c to support compressed files
  - fd.c provides file descriptor cacheing
  - Added: AllocateFileGz, FreeFileGz
- Regression test updates

COPY PIPE

- Follow the mailing lists
- Watch for others working on similar capabilities
- Try to think about general answers, not specific
- Be supportive of other ideas and approaches
- Send and receive COPY data from program instead
- E.g. for gzipped files

```
postgres=# COPY t FROM PROGRAM 'zcat /tmp/t.csv.gz'
```

Hacking the PG way

- PG has specific ways to do
  - Memory management
  - Error logging / cleanup
  - Linked lists
  - Catalog lookups
  - Nodes
  - Datums
  - Code Style
- How to submit your patch

Memory Handling

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

Logging from PG

- 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")));
```

Catalog Lookups

- SysCache
  - General function 'SearchSysCache'
  - Defined in utils/cache/syscache.c
  - Also some convenience routines in lsyscache.c
- Scanning catalog tables and Snapshots
  - Beware of SnapshotNow semantics
  - Viewing exactly what is in the heap
  - Heap can change while scanning it

resonate

Nodes

- PG has a node structure for expression trees
- Each node has a 'type' plus appropriate data
- 'type' is stored in the node, allows IsA() testing
- Backend memory only, never out on disk, etc
- Create nodes using makeNode(TYPE)
- Adding node type
  - Node types defined in include/nodes/nodes.h
  - make / copy / equality funcs in backend/nodes/

Datums

- General data type structure
- Defined in postgres.h
- Helper macros also in postgres.h
  - Example helpers, theres a bunch of them
  - Int32GetDatum(int) - Returns Datum of int
  - DatumGetInt32(Datum) - Returns int from Datum

Tuples

- Heap Tuple defined in include/access/htup.h
- HeapTupleData is in-memory construct
- Provides length of tuple, pointer to header
- Used in multiple ways
  - Pointer to disk buffer (must be pin'd)
  - Empty
  - Single pmalloc'd chunk
  - Seperately allocated
  - Minimal Tuple structure

Tuples (more)

- HeapTupleHeaderData and friends 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

Toast

- Large values can be compressed
- May also get "TOASTed" and moved to "toast" table
- Handled as a stored-out-of-line Datum
- Need to be careful with variable length Datums
- Typically try to avoid de-TOASTing Datums until absolutely required to

Other subsystems

- Many things have already been done
- Eg: linked list implementation (llist.h)
- Generalized code should go in common area
- Look at existing code
  - Real examples help immensely
  - Chances are, you will find what you need
  - Portability considerations

resonate

Code Style

- Try to make your code 'fit in'
- Follow the PG style guide in the FAQ
- Beware of copy/paste
- 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

Submitting Patches

- Patch format
  - Context diff or git-diff
  - Ideally, pick which is better
- Include in email to -hackers
  - Description of the patch
  - Regression tests
  - Documentation updates
  - pg_dump support
- Register on commitfest.postgresql.org

resonate

Thank you!

Stephen Frost
sfrost@snowman.net
@net_snow