Exposing PostgreSQL Internals with User-Defined Functions

Greg Smith

2ndQuadrant US

05/20/2010

2ndQuadrant
Professional PostgreSQL

About this presentation

- The master source for these slides is
  `http://projects.2ndquadrant.com`
- You can also find a machine-usable version of the source code to the later internals sample queries there

2ndQuadrant
Professional PostgreSQL

Hacking on PostgreSQL

- The bigger the patch, the less likely the commit

2ndQuadrant
Professional PostgreSQL

User Defined Functions

- UDFs allow writing simple functions in C that access internals
- Many parts of the database are built as functions
- "the standard internal function library is a rich source of coding examples for user-defined C functions"

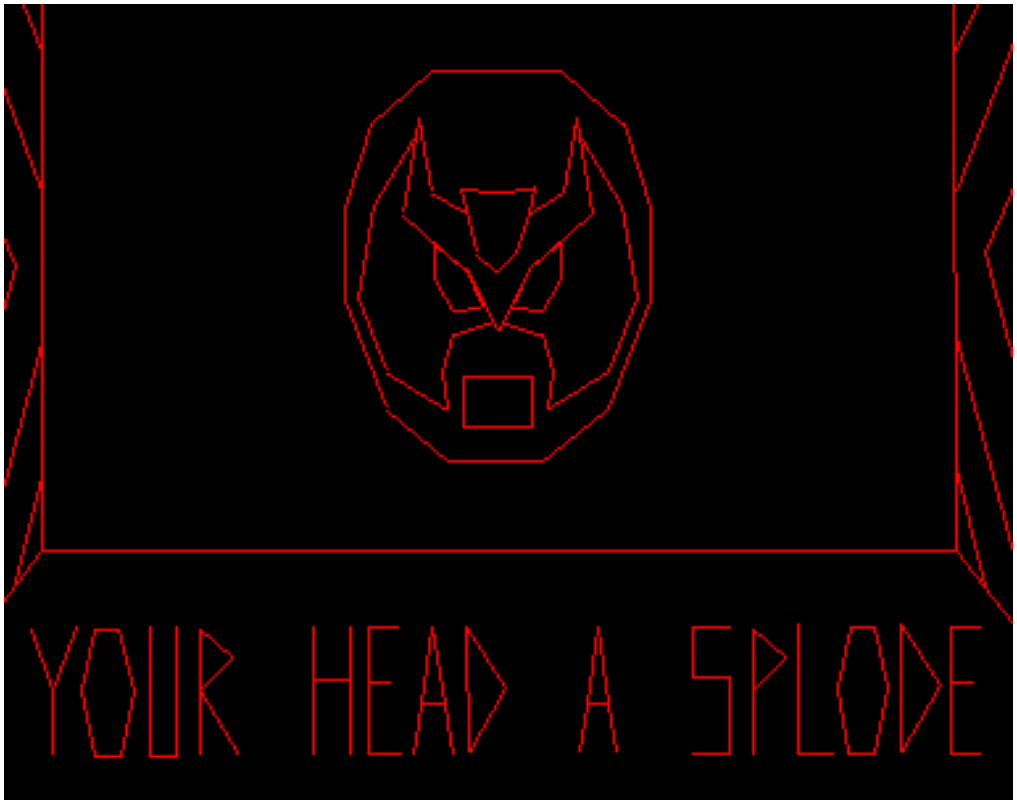
2ndQuadrant
Professional PostgreSQL

Existing Tutorials

- http://www.joeconway.com/web/guest
- "Power PostgreSQL: Extending the Database with C" by Joe Conway
- 100 slides; by page 21, examples are undecipherable
- http://neilconway.org/talks/hacking/
- "Introduction to Hacking PostgreSQL" by Neil Conway and Gavin Sherry
- "Patch to add WHEN clause to the CREATE TRIGGER statement"
- Adds new syntax and query execution
- 117 slides; expect to get lost no later than slide 33, "Semantic Analysis"

2ndQuadrant
Professional PostgreSQL

YOUR HEAD A SPLODE



2ndQuadrant
Professional PostgreSQL

Beginner Resources

- Cover basic setup like compiling
- http://www.postgresql.org/docs/current/static/xfunc-c.html
- http://www.postgresql.org/docs/current/interactive/xtypes.html
- http://linuxgazette.net/139/peterson.html
- http://tldp.org/LDP/LGNET/142/peterson.html

2ndQuadrant
Professional PostgreSQL

Question #1

- What is a tuple?
- Wikipedia: "a tuple represents the notion of an ordered list of elements"

2ndQuadrant
Professional PostgreSQL

tuple=row

- Tuples are how rows are stored in memory, basically

Question #2

- What's a Datum?
- Wikipedia: "Datum is the singular form of data"

2ndQuadrant
Professional PostgreSQL

Simple Datum

- src/include/postgres.h
- A Datum can be a boolean, a character
- It can be an integer (holding up to at least 4 bytes), or some other small integer type
- These are pass by value: the bytes allocated to the Datum contain the data

Reference Datum

- Larger types of data are passed by reference
- Memory is allocated by palloc
- The Datum is a pointer to that data
- Over 8 bytes is definitely too big for a Datum
- Exact transition point depends on architecture and PostgreSQL version
- Macros like DatumGetInt64 hide if you're passing a 8 byte integer by value (64-bit platform) or reference (32-bit)
- Similar macros to hide implementation of float and date/time types

Strings

- C string: standard null-terminated string

```
#define DatumGetPointer(X) ((Pointer) (X))

#define PointerGetDatum(X) ((Datum) (X))

#define DatumGetCString(X) ((char *) DatumGetPointer(X))

#define CStringGetDatum(X) PointerGetDatum(X)
```

StringInfo

- String with some metadata
- Current length, maximum length
- Not necessarily a true string
- Can be a series of binary bytes

Which type of Datum do you have?

- No way to tell from the Datum itself
- Uses of data have an explicit type inferred by context they are used in
- UDFs label each input and output parameter with an associated type

Mapping function names into calls

- src/include/catalog/pg_proc.h lists every function
- DATA(insert OID = 2077 (current_setting PGNSP PGUID 12 1 0 0 f f f t f s 1 0 25 "25" __ _null_ _null_ _null_ show_config_by_name _null_ _null_ _null_));
- Need to read its source code to learn what all these fields mean
- 25 = OID of text type
- Compiled into source code
- src/backend/catalog/postgres.bki
- Not fixed length; number of columns varies based on number of parameters passed to function

Function call internals

- src/backend/utils/fmgr includes DirectFunctionCall code
- Datum DirectFunctionCall1(PGFunction func, Datum arg1)
- Datum DirectFunctionCall2(PGFunction func, Datum arg1, Datum arg2)
- Up to DirectFunctionCall9 with takes arg1..arg9.
- See src/backend/utils/fmgr/README
- You can call functions from within your UDF using this interface

Internal functions in the database

- Function "library" is large
- Large enough that it's overwhelming
- List in psql:
- df pg_catalog.*
- Doesn't include many of the really useful built-in functions
- List is at src/include/utils/builtins.h
- Everything is in pg_proc.h

Decoding and encoding text in a function

```
Datum show_config_by_name(PG_FUNCTION_ARGS) {

char *varname;

char *varval;

/* Get the GUC variable name */

varname = TextDatumGetCString(PG_GETARG_DATUM(0));

/* Get the value */

varval = GetConfigOptionByName(varname, NULL);

/* Convert to text */

PG_RETURN_TEXT_P(cstring_to_text(varval));}
```

2ndQuadrant
Professional PostgreSQL

Sample hacking exercise

- "On a big server where I allocated a lot of memory for shared_buffers, how can I tell how much has been used?"
- "What is the memory working set size my fully cached application with small tables?"
- Expose this information from the buffer cache internals
- Can solve now by using pg_buffercache and counting buffers used
- Results skewed by ring buffer implementation
- Interesting value despite limitations

2ndQuadrant
Professional PostgreSQL

Who has this data?

- Background writer code scans and needs this information:
- src/backend/storage/buffer/bufmgr.c
- List of free buffers part of the allocation strategy code:
- src/backend/storage/buffer/freelist.c
- Cache use is circular
- If more than a single pass has been made, you've used all of it at some point

2ndQuadrant
Professional PostgreSQL

Game Plan

- Expose the hidden value in the low-level code
- Find a similar UDF to borrow code from
- Write a new UDF wrapper to expose the internals
- Add to the function catalog

src/backend/storage/buffer/freelist.c

```
int32 BuffersUsed(void) {

int used;

LWLockAcquire(BufFreelistLock, LW_EXCLUSIVE);

if (StrategyControl->completePasses == 0)

  used=StrategyControl->nextVictimBuffer;

else

  used=NBuffers;

LWLockRelease(BufFreelistLock);

return (int32) used; }
```

2ndQuadrant
Professional PostgreSQL

src/backend/utils/adt/dbsize.c

```
Datum pg_buffers_used(PG_FUNCTION_ARGS) {

int used;

int64 size;

used=BuffersUsed();

size=used * BLCKSZ;

PG_RETURN_INT64(size); }
```

Catalog and headerinfo

- cd src/include/catalog/
- Run unused_oids in that directory to find an unused value
- src/include/catalog/pg_proc.h
- DATA(insert OID = 3822 (pg_buffers_used PGNSP PGUID 12 1 0 0 f f f t f v 0 0 20 "" _null_ _null_ _null_ _null_ pg_buffers_used _null_ _null_ _null_);
- DESCR("bytes of shared_buffers cache used");
- src/include/storage/buf_internals.h
- extern int64 BuffersUsed(void);
- src/include/utils/builtins.h
- extern Datum pg_buffers_used(PG_FUNCTION_ARGS);

2ndQuadrant
Professional PostgreSQL

Basic Debugging

- ERROR: invalid memory alloc request size 4294967293
- Wrong return type; allocated memory for my function didn't match
- dbsize.c:637: warning: implicit declaration of function BuffersUsed
- Missing function declaration in the header files
- General development logging
- client_min_messages = debug2
- elog(DEBUG1,"Buffers used: %d",used);
- elog(DEBUG1, "Used buffer cache bytes: %lld",size);

2ndQuadrant
Professional PostgreSQL

It runs!

- Function gets called and no compiler warnings
- Is the resulting data useful?

Oops!

- UDF doesn't work at all!
- Data returned is always zero
- Your client process is not the background writer
- Process model in PostgreSQL is fairly complicated

2ndQuadrant
Professional PostgreSQL

Open Item Complexity

- Some features appear easy to build and obviously useful
- Those are done already
- What's left on TODO list often contains hidden complexity and gotchas
- Ask about your idea before writing a lot of code
- Keep the complexity as low as possible

Working example

- No substitute for a real commit to show a proven end result
- http://archives.postgresql.org/pgsql-committers/2010-01/msg00288.php
- Adds pg_table_size and pg_indexes_size functions
- Shows catversion bump
- Even includes docs!
- http://git.postgresql.org/gitweb?p=postgresql.git;a=commitdiff
- ;h=7c3ec9753dbedb00642f0fdfce90f9a11940df99

2ndQuadrant
Professional PostgreSQL

Closing Reminders

- Keep it small
- Read other people's code
- Steal code from the server
- When in doubt, you can always read the source!

2ndQuadrant
Professional PostgreSQL