PostgreSQL internals

Pavel Stěhule

PostgreSQL Meetup

PostgreSQL - subsystems

- Memory management
- Safe storage
- Metadata management (system cache)
- Query parser, optimizer - query compiler
- Execution plan executor, expression executor - interprets
- Runtime statistics collector

History

- Ingres - C language (storage) + Lisp (optimizer) + bison (parser)
- Postgres, PostgreSQL - C language + macros + bison (parser). Lisp removed (memory management issues)
- 966K lines of source code
  - MySQL 3.8M C++, Firebird 1.5M C++

Memory management

- libc malloc/free - simple, fast, but high risks of memory leaks.
- MemoryContext - PostgreSQL solution - hiearchy of memory blocks
  - palloc/pfree - related to current memory context
  - reset - drop - release all nested contexts

Memory management

```
MemoryContext *oldcxt = MemoryContextSwitchTo(estate->func->fn_cxt);
PLpgSQL_row *row = palloc0(sizeof(*row));

row->dtype = PLPGSQL_DTYPE_ROW;
row->varnos = palloc(sizeof(int) * FUNC_MAX_ARGS);
...

MemoryContextSwitchTo(oldcxt);
```

Memory contexts

- TopMemoryContext
- ErrorContext
- TopTransactionContext
- ecxt_per_query_memory
- ecxt_per_tuple_memory

PostgreSQL memory management

- Simple, effective
- Usually only few real memory leaks, sometimes is necessary (and possible) use some short life memory context.
- Source of troubles for beginners - results are returned in wrong short life context. Use --enable-cassert

Datum type

- generic type - used as ancestor type of any custom data type in C code
- same size as pointer
- can holds a some value or pointer
- pg_type .. type cache holds more details - typlen, typbyval, oid, typname, typmod, ...

Varlena type

- generic type - used as ancestor type of any variable length custom data type in C code
- length is encoded in first n (1, 2, 4) bytes
- typlen = -1
- text or binary values: varchar, text, bytea, numeric, ...

TOAST

- varlena values can be:
  - toasted
    - compressed
    - stored in external table
- toasting - when value is stored to page
- detoasting - when value is used - detoasting is lazy

Custom types

- Internally based on Datum (Varlena) type
- Any type has defined in, out, send, recv functions
- There should be macros for conversions Datum x type
  - DatumGetXXX
  - XXXGetDatum
- These macros must be used, sometimes Datum can be a pointer to toasted value. These macros ensure detoasting. These macros are internal - not related to type casts.

Hello world

```
Datum
hello_int(PG_FUNCTION_ARGS)
{
    int a = PG_GETARG_INT32(0);

    PG_RETURN_INT32(a + 100);
}

CREATE OR REPLACE FUNCTION hello_int(int)
RETURNS int AS hello_int LANGUAGE C;
```

Hello world

```
Datum
hello_text(PG_FUNCTION_ARGS)
{
    text *txt = PG_GETARG_TEXT_P(0);
    StringInfoData str;

    initStringInfo(&str);
    appendStringInfo(&str, "Hello, %s", text_to_cstring(txt));

    PG_RETURN_TEXT_P(cstring_to_text_with_len(str.data, str.length));
}

CREATE OR REPLACE FUNCTION hello(text)
RETURNS text AS hello_text LANGUAGE C STRICT;
```

low level macros

```
text *result;
char  *str = "AHOJ";
int   length;

length = strlen(str);
result = (text *) palloc(length + VARHDRSZ);
memcpy(VARDATA(result), str, length);

/* encoded is total length (not only string length) */
SET_VARSIZE(result, length + VARHDRSZ);
```

Passing parameters

- PostgreSQL uses own implementation of passing parameters to SQL functions - V1 convention
- Simple implementation without integration of C compiler functionality. Is not necessary to manipulate with stack. System can call and pass parameters any V1 function, that can be implemented in any language (sometimes is executed runtime of some programming language). V1 functions are equal from C perspective.
- Possible to pass more information about function, parameters, evaluation context to function body.

FunctionCallInfo

```
#define PG_FUNCTION_ARGS FunctionCallInfo fcinfo

#define PG_ARGISNULL(n)  (fcinfo->argnull[n])
#define PG_NARGS()       (fcinfo->nargs)
#define PG_GATARG_DATUM(n)  (fcinfo->args[n])

#define PG_GETARG_TEXT_PP(X)  ((text *) PG_DETOAST_DATUM_PACKED(X))

#define PG_RETURN_NULL() \
  do { fcinfo->isnull = true; return (Datum) 0; } while (0)
```

Nodes - OOP, functional programing in C language

- PostgreSQL uses C89
- Some code has OOP features
- Some code has functional programming features

OOP in C

- struct Node is abstract ancestor
- children nodes are structs, where ancestor is on first position - then cast to ancestor is safe
- There are generic read/write/print/compare/transform/iterate operations over nodes
- AST is tree of parser nodes
- Execution plan is tree of executor nodes

Node

```
typedef struct Node
{
    NodeTag     type;
}

typedef struct A_Const
{
    NodeTag     type;
    Value       val;
    int         location;
}

...
```

Node

```
int
get_location(Node *node)
{
    switch (node->type)
    {
        case T_A_Const:
            return ((A_Const *) node)->location;
         ...
```

LISP, List in C

- linked list of Nodes
- functions for creating, delete, iterate

list_make1

```
set_target: ColId opt_indirection
        {
            $$ = make_node(ResTarget);
            $$->name = $1;
            $$->location = @1;
        }
    ;

set_target_list:
    set_target                              { $$ = list_make1($1); }
    | set_target_list ‘,‘ set_target        { $$ = lappend($1, $3); }
    ;
```

foreach

```
ListCell *lc;
List    *exprList;

foreach(lc, exprList)
{
    Expr *expr = (Expr *) lfirst(lc);

    ...

}
```

PostgreSQL

- Use C language
- + lot of macros
  - hide complexity
  - hide platform specific features
- C is very fast, mature, small language
- There are significant differences between platforms, but can be solved (mostly) by macros. There are not too much #ifdef segments
- Almost all code is working on Unix like systems and Windows