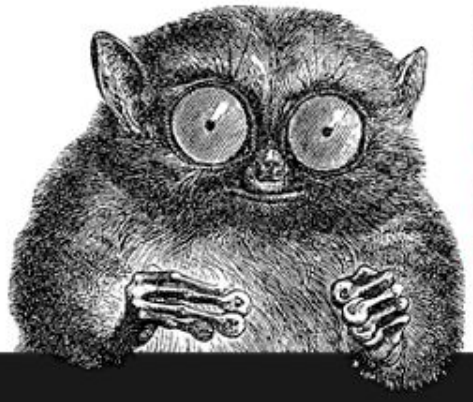
Power PostgreSQL: Extending the Database with C

Joe Conway

mail@joeconway.com

O'Reilly Open Source Convention
July 26–30, 2004



O'REILLY®
OPEN SOURCE CONVENTION™

Extensibility

- Server operation is catalog-driven
- Can incorporate user-written code into itself through dynamic loading

Procedural Language Support

- PostgreSQL has quite an impressive array of procedural languages that can be used to write functions
  - Distributed with PostgreSQL: C, SQL, PL/pgSQL, PL/Perl, PL/Tcl, PL/Python
  - Distributed separately: PL/sh, PL/R, PL/Java, PL/Ruby, PL/PHP, PL/mono

Agenda

- Getting started
  - build system: Makefile, other project files
- Functions returning one row
  - calling conventions
  - arguments
  - return values
  - error handling
  - memory allocation and contexts
  - data persistence
- Set returning functions (a.k.a. SRFs, table functions)
- Q&A

Getting Started

- Use “contrib” as a model
- Typical Makefile

```
subdir = contrib/myfunc
top_builddir = ../..
include $(top_builddir)/src/Makefile.global

MODULES = myfunc
DATA_built = myfunc.sql
DOCS = README.myfunc
REGRESS = myfunc

include $(top_srcdir)/contrib/contrib-global.mk
```

myfunc Files

- Minimal files
  - Makefile
  - myfunc.c
  - myfunc.sql.in
- Optional files
  - myfunc.h
  - README.myfunc
  - sql/myfunc.sql
  - expected/myfunc.out

Single Row Functions

Calling Conventions

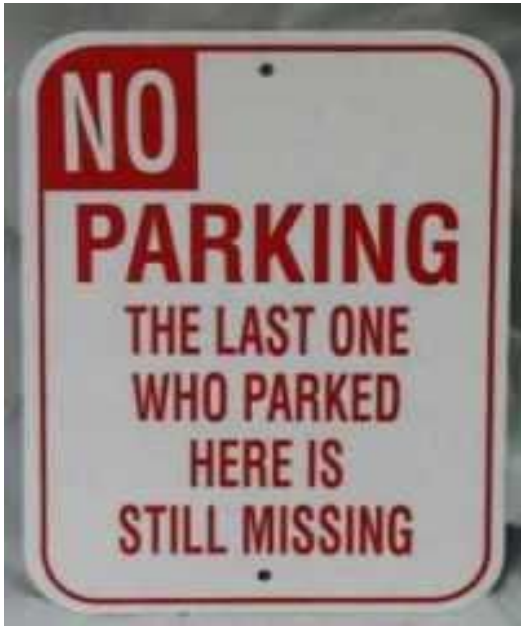
- Version 0
- Version 1
  - Function info macro
  - Function declaration
    - fcinfo
    - FmgrInfo

Calling Conventions

- Version 0
- Version 1
  - Function info macro
  - Function declaration
    - fcinfo
    - FmgrInfo

Version 0 Calling Convention

- Deprecated
- Has been since the 7.1 release.
- Don't use it.



Calling Conventions

- Version 0
- Version 1
  - Function info macro
  - Function declaration
    - fcinfo
    - FmgrInfo

Version 1 Calling Convention

- Function info macro

```
PG_FUNCTION_INFO_V1(myfunc_example_01);


extern Pg_finfo_record *
pg_finfo_myfunc_example_01 (void);

Pg_finfo_record *
pg_finfo_myfunc_example_01 (void)
{
  static Pg_finfo_record my_finfo = { 1 };
  return &my_finfo;
}
```

Version 1 Calling Convention

- Function info macro

```
PG_FUNCTION_INFO_V1(myfunc_example_01);


extern Pg_finfo_record *
pg_finfo_myfunc_example_01 (void);

Pg_finfo_record *
pg_finfo_myfunc_example_01 (void)
{
  static Pg_finfo_record my_finfo = { 1 };
  return &my_finfo;
}
```

Version 1 Calling Convention

- Function declaration

```
Datum
myfunc_example_01(PG_FUNCTION_ARGS)
{
     /* myfunc code here */
}
```

fcinfo

```
typedef struct FunctionCallInfoData
{
     FmgrInfo          *flinfo;
     struct Node       *context;
     struct Node       *resultinfo;
     bool               isnull;
     short              nargs;
     Datum              arg[FUNC_MAX_ARGS];
     bool               argnull[FUNC_MAX_ARGS];
} FunctionCallInfoData;
```

FmgrInfo

```
typedef struct FmgrInfo
{
     PGFunction        fn_addr;
     Oid               fn_oid;
     short             fn_nargs;
     bool              fn_strict;
     bool              fn_retset;
     void             *fn_extra;
     MemoryContextfn_mcxt;
     struct Node      *fn_expr;
} FmgrInfo;
```

Handling Function Arguments

- Argument related macros
- Scalar arguments
- Decomposing arrays
- Decomposing composite (tuple) types
- Resolving polymorphic types

Handling Function Arguments

- Argument related macros
- Scalar arguments
- Decomposing arrays
- Decomposing composite (tuple) types
- Resolving polymorphic types

Argument Related Macros

- PG_FUNCTION_ARGS
- PG_NARGS()
- PG_ARGISNULL(n)
- PG_GETARG_DATUM(n)
- PG_DETOAST_DATUM(datum)
- PG_DETOAST_DATUM_COPY(datum)
- PG_DETOAST_DATUM_SLICE(datum,f,c)
- PG_FREE_IF_COPY(ptr,n)
- PG_GETARG_<type>(n): examples
  - PG_GETARG_TEXT_P(n) & PG_GETARG_TEXT_P_COPY(n)
  - PG_GETARG_INT32(n)
  - PG_GETARG_ARRAYTYPE_P(n)

Handling Function Arguments

- Argument related macros
- Scalar arguments
- Decomposing arrays
- Decomposing composite (tuple) types
- Resolving polymorphic types

Scalar Arguments

```
#define PG_TEXT_GET_CSTR(textp_) \
    DatumGetCString(DirectFunctionCall1(textout, \
                    PointerGetDatum(textp_)))

PG_FUNCTION_INFO_V1(my_func);
Datum
my_func(PG_FUNCTION_ARGS)
{
    char        *cptr;
    int32        pos;

    cptr = PG_TEXT_GET_CSTR(PG_GETARG_TEXT_P(0));
    pos = PG_GETARG_INT32(1);
```

Handling Function Arguments

- Argument related macros
- Scalar arguments
- Decomposing arrays
- Decomposing composite (tuple) types
- Resolving polymorphic types

Decomposing Arrays

```
Datum
array_to_elems(PG_FUNCTION_ARGS)
{
    ArrayType    *v = PG_GETARG_ARRAYTYPE_P(0);
    int          nitems,
                 *dims,
                 ndims;
    char         *p;
    Oid          element_type;
    int          typlen;
    bool         typbyval;
    char         typalign;
    char         typdelim;
    Oid          typelem;
    Oid          typiofunc
    FmgrInfo     proc;
    int          i;

    p = ARR_DATA_PTR(v);
    ndims = ARR_NDIM(v);
    dims = ARR_DIMS(v);
    nitems = ArrayGetNItems(ndims, dims);
```

Decomposing Arrays (cont.)

```
element_type = ARR_ELEMTYPE(v);

get_type_io_data(element_type, IOFunc_output,
                 &typlen, &typbyval,
                 &typalign, &typdelim,
                 &typelem, &typiofunc);

fmgr_info_cxt(typiofunc,
              &proc,
              fcinfo->flinfo->fn_mcxt);
```

Decomposing Arrays (cont.)

```
for (i = 0; i < nitems; i++)
{
    Datum    itemvalue;
    char     *value;

    itemvalue = fetch_att(p, typbyval, typlen);

    value = DatumGetCString(FunctionCall3(&proc,
                    itemvalue,
                    ObjectIdGetDatum(typelem),
                    Int32GetDatum(-1)));

    /* Do something with value here */

    p = att_addlength(p, typlen,
                      PointerGetDatum(p));
    p = (char *) att_align(p, typalign);
}
```

Handling Function Arguments

- Argument related macros
- Scalar arguments
- Decomposing arrays
- Decomposing composite (tuple) types
- Resolving polymorphic types

Decomposing Composite Types

```
Datum
extract_field_as_text(PG_FUNCTION_ARGS)
{
     TupleTableSlot *slot;
     char           *desired_col_name;
     HeapTuple       tuple;
     TupleDesc       tupdesc;
     int             nc;
     int             j = 0;

     slot = (TupleTableSlot *) PG_GETARG_POINTER(0);
     tuple = slot->val;
     tupdesc = slot->ttc_tupleDescriptor;
     nc = tupdesc->natts;

     desired_col_name =
             PG_TEXT_GET_CSTR(PG_GETARG_TEXT_P(1));
```

Decomposing Composite Types (cont.)

```
for (j = 0; j < nc; j++)
{
    char   *this_col_name;

    if (tupdesc->attrs[j]->attisdropped)
        continue;

    this_col_name = SPI_fname(tupdesc, j + 1);

    if (strcmp(this_col_name,
               desired_col_name) == 0)
    {
            char   *value;

            value = SPI_getvalue(tuple,
                                 tupdesc, j + 1);
            if (value)
                PG_RETURN_TEXT_P
                    (PG_CSTR_GET_TEXT(value));
            else
                PG_RETURN_NULL();
    }
}
```

Decomposing Composite Types (cont.)

```
    ereport(ERROR,
            (errcode(ERRCODE_INVALID_PARAMETER_VALUE),
             errmsg("column not found: %s",
                    desired_col_name)));

    PG_RETURN_NULL();
}

CREATE TABLE myfoo (f1 int, f2 text, f3 float8, f4 int[]);
INSERT INTO myfoo VALUES(1, 'abc', 3.14159, ARRAY[1,2,3]);
INSERT INTO myfoo VALUES(2, 'def', 2.8, ARRAY[6,7,8]);

CREATE OR REPLACE FUNCTION extract_field_as_text(myfoo, text)
RETURNS text
AS '$libdir/myfunc','extract_field_as_text'
LANGUAGE 'C' IMMUTABLE STRICT;

SELECT extract_field_as_text(myfoo, 'f4') from myfoo;
 extract_field_as_text
-----------------------
 {1,2,3}
 {6,7,8}
(2 rows)
```

Handling Function Arguments

- Argument related macros
- Scalar arguments
- Decomposing arrays
- Decomposing composite (tuple) types
- Resolving polymorphic types

Resolving Polymorphic Types

```
Datum
array_push(PG_FUNCTION_ARGS)
{
     ArrayType *v;
     Datum     newelem;
     Oid       element_type;
     [...snip...]
     Oid       arg0_typeid;
     Oid       arg1_typeid;
     Oid       arg0_elemid;
     Oid       arg1_elemid;

     arg0_typeid = get_fn_expr_argtype(fcinfo->flinfo, 0);
     arg1_typeid = get_fn_expr_argtype(fcinfo->flinfo, 1);

     if (arg0_typeid == InvalidOid ||
          arg1_typeid == InvalidOid)
          ereport(ERROR,
               (errcode(ERRCODE_INVALID_PARAMETER_VALUE),
                errmsg("could not determine input data "
                   "types")));
```

Resolving Polymorphic Types (cont.)

```
arg0_elemid = get_element_type(arg0_typeid);
arg1_elemid = get_element_type(arg1_typeid);

if (arg0_elemid != InvalidOid)
{
     v = PG_GETARG_ARRAYTYPE_P(0);
     element_type = ARR_ELEMTYPE(v);
     newelem = PG_GETARG_DATUM(1);
}
else if (arg1_elemid != InvalidOid)
{
     v = PG_GETARG_ARRAYTYPE_P(1);
     element_type = ARR_ELEMTYPE(v);
     newelem = PG_GETARG_DATUM(0);
}
else
{
     ereport(ERROR,
          (errcode(ERRCODE_DATATYPE_MISMATCH),
           errmsg("neither input type is an array")));
     PG_RETURN_NULL();
}
```

Handling Return Values

- Return related macros
- Scalar return values
- Constructing and returning arrays
- Constructing and returning composite (tuple) types
- Resolving polymorphic types

Handling Return Values

- Return related macros
- Scalar return values
- Constructing and returning arrays
- Constructing and returning composite (tuple) types
- Resolving polymorphic types

Return Related Macros

- PG_RETURN_NULL()
- PG_RETURN_VOID()
- PG_RETURN_DATUM(x)
- PG_RETURN_POINTER(x)
- PG_RETURN_<type>(x): examples
    - PG_RETURN_TEXT_P(x)
    - PG_RETURN_INT32(x)
    - PG_RETURN_ARRAYTYPE_P(x)

Handling Return Values

- Return related macros
- Scalar return values
- Constructing and returning arrays
- Constructing and returning composite (tuple) types
- Resolving polymorphic types

Scalar Return Values

Example: pass-by-reference string types

```
#define PG_CSTR_GET_TEXT(cstrp) \
  DatumGetTextP(DirectFunctionCall1(textin, CStringGetDatum(cstrp)))

Datum
extract_field_as_text(PG_FUNCTION_ARGS)
{
     [...]

     value = SPI_getvalue(tuple, tupdesc, j + 1);
     if (value)
          PG_RETURN_TEXT_P(PG_CSTR_GET_TEXT(value));
     else
          PG_RETURN_NULL();

     [...]
}
```

Alternatively:

```
     PG_RETURN_DATUM(DirectFunctionCall1(textin,
                                         CStringGetDatum(value)));
```

Scalar Return Values (cont.)

Other Examples: pass-by-value

```
Datum
levenshtein(PG_FUNCTION_ARGS)
{
     [...]
     int      cols = 0;

     [...]

     if (rows == 0)
          PG_RETURN_INT32(cols);
     [...]
}
```

Scalar Return Values (cont.)

Other Examples: pass-by-reference numeric types

```
Datum
float8pl(PG_FUNCTION_ARGS)
{
     float8   arg1 = PG_GETARG_FLOAT8(0);
     float8   arg2 = PG_GETARG_FLOAT8(1);
     float8   result;

     result = arg1 + arg2;

     CheckFloat8Val(result);
     PG_RETURN_FLOAT8(result);
}
```

Handling Return Values

- Return related macros
- Scalar return values
- Constructing and returning arrays
- Constructing and returning composite (tuple) types
- Resolving polymorphic types

Constructing and Returning Arrays

```
Datum
create_array(PG_FUNCTION_ARGS)
{
     ArrayType  *v;
     Datum        dvalues[1];
     int16        typlen;
     bool         typbyval;
     char         typalign;
     int          dims[MAXDIM], lbs[MAXDIM], i, ndims;
     Oid          element_type, array_type;

     if (PG_NARGS() == 2)
     {
          ndims = PG_GETARG_INT32(0);
          element_type = get_fn_expr_argtype(fcinfo->flinfo, 1);
          dvalues[0] = PG_GETARG_DATUM(1);
     }
     else
     {
          ndims = 1;
          element_type = get_fn_expr_argtype(fcinfo->flinfo, 0);
          dvalues[0] = PG_GETARG_DATUM(0);
     }
```

Constructing and Returning Arrays (cont.)

```
    array_type = get_array_type(element_type);
    if (array_type == InvalidOid)
        ereport(ERROR,
                (errcode(ERRCODE_INVALID_PARAMETER_VALUE),
                 errmsg("invalid array element type OID: %u",
                        element_type)));

    for (i = 0; i < ndims; i++)
    {
        dims[i] = 1;
        lbs[i] = 1;
    }

    get_typlenbyvalalign(element_type, &typlen,
                         &typbyval, &typalign);

    v = construct_md_array(dvalues, ndims, dims, lbs,element_type,
                           typlen, typbyval, typalign);

    PG_RETURN_ARRAYTYPE_P(v);
}
```

Constructing and Returning Arrays (cont.)

```
CREATE OR REPLACE FUNCTION create_singleton_array(int, anyelement)
RETURNS anyarray
AS '$libdir/myfunc','create_array'
LANGUAGE 'C' IMMUTABLE STRICT;

select create_singleton_array(3, '(6,42)'::point);
 create_singleton_array
------------------------
 {{{"(6,42)"}}}
(1 row)

select t.f[1][1][1] from (select create_singleton_array(3,
'(6,42)'::point) as f) as t;
   f
--------
 (6,42)
(1 row)
```

Handling Return Values

- Return related macros
- Scalar return values
- Constructing and returning arrays
- Constructing and returning composite (tuple) types
- Resolving polymorphic types

Returning Composite Types

```
Datum
build_tuple_from_textarray(PG_FUNCTION_ARGS)
{
    ArrayType          *v = PG_GETARG_ARRAYTYPE_P(0);
    Oid                element_type = ARR_ELEMTYPE(v);
    int                ndims = ARR_NDIM(v);
    int                *dim_counts = ARR_DIMS(v);
    int                *dim_lower_bounds = ARR_LBOUND(v);
    int16              typlen;
    bool               typbyval;
    char               typalign;
    ReturnSetInfo*rsinfo = (ReturnSetInfo *)fcinfo->resultinfo;
    AttInMetadata*attinmeta;
    TupleDesc          tupdesc;
    Tuplestorestate    *tupstore = NULL;
    HeapTuple          tuple;
    MemoryContextper_query_ctx;
    MemoryContextoldcontext;
    Datum              dvalue;
    char               **values;
    int                ncols = 0, nrows = 0, indx[MAXDIM], j;
```

Returning Composite Types (cont.)

```
nrows = 1;
ncols = dim_counts[0];

get_typlenbyvalalign(element_type, &typlen,
                     &typbyval, &typalign);

tupdesc = rsinfo->expectedDesc;
attinmeta = TupleDescGetAttInMetadata(tupdesc);

rsinfo->returnMode = SFRM_Materialize;

per_query_ctx = fcinfo->flinfo->fn_mcxt;
oldcontext = MemoryContextSwitchTo(per_query_ctx);
tupstore = tuplestore_begin_heap(true, false, SortMem);

values = (char **) palloc(ncols * sizeof(char *));
```

Returning Composite Types (cont.)

```
    for (j = 0; j < ncols; j++)
    {
        bool     isnull;

        indx[0] = j + dim_lower_bounds[0];
        dvalue = array_ref(v, ndims, indx, -1, typlen,
                           typbyval, typalign, &isnull);
        values[j] = DatumGetCString(DirectFunctionCall1(textout,
                                                        dvalue));
    }

    tuple = BuildTupleFromCStrings(attinmeta, values);
    tuplestore_puttuple(tupstore, tuple);
    tuplestore_donestoring(tupstore);

    rsinfo->setResult = tupstore;
    rsinfo->setDesc = tupdesc;

    MemoryContextSwitchTo(oldcontext);
    return (Datum) 0;
}
```

Returning Composite Types (cont.)

```
CREATE OR REPLACE FUNCTION build_row(text[])
RETURNS setof record
AS '$libdir/myfunc','build_tuple_from_textarray'
LANGUAGE 'C' IMMUTABLE STRICT;

select a,b,c from build_row('1'::text || array['abc','today']) as t(a
int, b text, c timestamp);
 a |  b  |          c
---+-----+---------------------
 1 | abc | 2004-05-21 00:00:00
(1 row)
```

Handling Return Values

- Return related macros
- Scalar return values
- Constructing and returning arrays
- Constructing and returning composite (tuple) types
- Resolving polymorphic types

Polymorphic Return Types

- Explicit resolution usually not needed
  - Polymorphic return type requires at least one polymorphic argument, therefore return type is predetermined by runtime type of the argument
  - When constructing an array, the array's data type does not even need to be explicitly known; only the element type does.
- Use case: polymorphic-by-signature function
  - Overloaded SQL functions, with differing return types, sharing one implementing C function
  - Resolve return type with get_fn_expr_rettype(finfo)

Polymorphic Return Types (cont.)

From ./src/backend/executor/functions.c:init_sql_fcache()

```
[...]
Oid           foid = finfo->fn_oid;
HeapTuple     procedureTuple;
HeapTuple     typeTuple;
Form_pg_proc procedureStruct;
[...]
procedureTuple = SearchSysCache(PROCOID, ObjectIdGetDatum(foid),
                                                    0, 0, 0);
if (!HeapTupleIsValid(procedureTuple))
     elog(ERROR, "cache lookup failed for function %u", foid);
procedureStruct = (Form_pg_proc) GETSTRUCT(procedureTuple);

rettype = procedureStruct->prorettype;
if (rettype == ANYARRAYOID || rettype == ANYELEMENTOID)
{
     rettype = get_fn_expr_rettype(finfo);
     if (rettype == InvalidOid)
          ereport(ERROR,[...]
}
[...]
ReleaseSysCache(procedureTuple);
[...]
```

Error Handling

- elog()
- ereport()
- Context callbacks

Error Handling

- elog()
- ereport()
- Context callbacks

elog()

```
elog(level, fmt, ...);

elog(ERROR, "cache lookup failed for function %u", foid);
```

Error Handling

- elog()
- ereport()
- Context callbacks

ereport()

```
ereport(level,
          (errcode(errcode),
           errmsg(fmt, ...),
           errdetail(fmt, ...),
           errhint(fmt, ...)));

ereport(ERROR,
         (errcode(ERRCODE_INVALID_PARAMETER_VALUE),
          errmsg("column not found: %s", desired_col_name)));
```

Error Handling

- elog()
- ereport()
- Context callbacks

Context Callbacks

```
Datum
myfunc_errcontext(PG_FUNCTION_ARGS)
{
     char    *msg = PG_TEXT_GET_CSTR(PG_GETARG_TEXT_P(0));
     ErrorContextCallback  errcontext;

     errcontext.callback = myfunc_error_callback;
     errcontext.arg = (void *) "myfunc_errcontext";
     errcontext.previous = error_context_stack;
     error_context_stack = &errcontext;

     elog(ERROR, "%s", msg);

     error_context_stack = errcontext.previous;
     PG_RETURN_VOID();
}

static void
myfunc_error_callback(void *arg)
{
     if (arg)
          errcontext("In myfunc function %s", (char *) arg);
}
```

Context Callbacks (cont.)

```
CREATE OR REPLACE FUNCTION myfunc_throw_error(text)
RETURNS void
AS '$libdir/myfunc','myfunc_errcontext'
LANGUAGE 'C' IMMUTABLE STRICT;

select myfunc_throw_error('foobar');
ERROR:  foobar
CONTEXT:  In myfunc function myfunc_errcontext
```

Memory Allocation and Contexts

- Macros/functions
- Memory Contexts

Memory Allocation and Contexts

- Macros/functions
- Memory Contexts

Memory Allocation Macros/Functions

- palloc(size)
- palloc0(size)
- pstrdup(str)
- repalloc(void *pointer, Size size)
- pfree(void *pointer)

Memory Allocation and Contexts

- Macros/functions
- Memory Contexts

Memory Contexts

- Overview
  - All individual memory allocations are done in "memory contexts"
  - Memory contexts are created, reset, and deleted en masse, at appropriate times for the given context
  - Contexts are hierarchical. Each can have zero or one parent, and many children. Resetting or deleting a node also affects all of its children
  - This greatly simplifies memory "bookkeeping" for function authors
  - In most cases memory leaked by a function is reclaimed by the backend quickly enough that it does not pose a problem

Memory Contexts (cont.)

- MemoryContextAlloc(MemoryContext ctx, Size size)
- MemoryContextAllocZero(MemoryContext ctx, Size size)
- MemoryContextStrdup(MemoryContext ctx, const char *str)
- MemoryContextSwitchTo(MemoryContext context)
    - CurrentMemoryContext
    - TopMemoryContext
    - TopTransactionContext
    - fcinfo->flinfo->fn_mcxt
    - funcctx->multi_call_memory_ctx

Data Persistence

- Call-to-call
  - switch to fcinfo->flinfo->fn_mcxt
  - switch to funcctx->multi_call_memory_ctx
- Session
  - switch to TopMemoryContext
  - use a dynahash

DynaHash Example

```
#include "utils/hsearch.h"

static char *getStrByName(const char *name);
static HTAB *createStrHash(void);
static void createNewStr(const char *name, char *str);
static void deleteStr(const char *name);

static HTAB *strHash = NULL;
typedef struct strHashEnt
{
     char     name[NAMEDATALEN];
     char    *str;
}    strHashEnt;

#define NUMSTR 16
```

DynaHash Example (cont.)

```
static HTAB *
createStrHash(void)
{
     HASHCTL       ctl;
     HTAB    *ptr;

     ctl.keysize = NAMEDATALEN;
     ctl.entrysize = sizeof(strHashEnt);

     ptr = hash_create("str hash", NUMSTR, &ctl, HASH_ELEM);

     if (!ptr)
          ereport(ERROR,
                  (errcode(ERRCODE_OUT_OF_MEMORY),
                   errmsg("out of memory")));

     return (ptr);
}
```

DynaHash Example (cont.)

```
static void
createNewStr(const char *name, char *str)
{
     strHashEnt *hentry;
     bool       found;
     char       key[NAMEDATALEN];

     if (!strHash)
          strHash = createStrHash();

     MemSet(key, 0, NAMEDATALEN);
     snprintf(key, NAMEDATALEN - 1, "%s", name);
     hentry = (strHashEnt *) hash_search(strHash, key,
                                         HASH_ENTER, &found);
     if (!hentry)
          ereport(ERROR,(errcode([...]), errmsg("out of memory")));
     if (found)
          ereport(ERROR,(errcode([...]), errmsg("duplicate name")));

     hentry->str = MemoryContextStrdup(TopMemoryContext, str);
     strncpy(hentry->name, name, NAMEDATALEN - 1);
}
```

DynaHash Example (cont.)

```
static char *
getStrByName(const char *name)
{
     strHashEnt *hentry;
     char     key[NAMEDATALEN];

     if (!strHash)
          strHash = createStrHash();

     MemSet(key, 0, NAMEDATALEN);
     snprintf(key, NAMEDATALEN - 1, "%s", name);
     hentry = (strHashEnt *) hash_search(strHash, key,
                                         HASH_FIND, NULL);

     if (hentry)
          return (hentry->str);

     return (NULL);
}
```

DynaHash Example (cont.)

```
static void
deleteStr(const char *name)
{
     strHashEnt *hentry;
     bool       found;
     char       key[NAMEDATALEN];

     if (!strHash)
          strHash = createStrHash();

     MemSet(key, 0, NAMEDATALEN);
     snprintf(key, NAMEDATALEN - 1, "%s", name);
     hentry = (strHashEnt *) hash_search(strHash, key,
                                         HASH_REMOVE, &found);
     if (!hentry)
          ereport(ERROR,
                  (errcode(ERRCODE_UNDEFINED_OBJECT),
                   errmsg("undefined string name")));

     if (hentry->str)
          pfree(hentry->str);
}
```

DynaHash Example (cont.)

```
Datum
myfunc_setvar(PG_FUNCTION_ARGS)
{
     char    *name = NULL;
     char    *str = NULL;

     name = PG_TEXT_GET_CSTR(PG_GETARG_TEXT_P(0));
     str = PG_TEXT_GET_CSTR(PG_GETARG_TEXT_P(1));

     createNewStr(name, str);

     PG_RETURN_TEXT_P(PG_CSTR_GET_TEXT("OK"));
}
```

DynaHash Example (cont.)

```
Datum
myfunc_getvar(PG_FUNCTION_ARGS)
{
     char    *name = NULL;
     char    *str;

     name = PG_TEXT_GET_CSTR(PG_GETARG_TEXT_P(0));

     str = getStrByName(name);
     if (str)
          PG_RETURN_TEXT_P(PG_CSTR_GET_TEXT(str));
     else
          PG_RETURN_NULL();
}
```

DynaHash Example (cont.)

```
Datum
myfunc_rmvar(PG_FUNCTION_ARGS)
{
     char    *name = NULL;

     name = PG_TEXT_GET_CSTR(PG_GETARG_TEXT_P(0));

     deleteStr(name);

     PG_RETURN_TEXT_P(PG_CSTR_GET_TEXT("OK"));
}
```

DynaHash Example (cont.)

```
CREATE OR REPLACE FUNCTION myfunc_setvar(text, text)
RETURNS text
AS '$libdir/myfunc','myfunc_setvar'
LANGUAGE 'C' IMMUTABLE STRICT;

CREATE OR REPLACE FUNCTION myfunc_getvar(text)
RETURNS text
AS 'MODULE_PATHNAME','myfunc_getvar'
LANGUAGE 'C' STABLE STRICT;

CREATE OR REPLACE FUNCTION myfunc_rmvar(text)
RETURNS text
AS '$libdir/myfunc','myfunc_rmvar'
LANGUAGE 'C' IMMUTABLE STRICT;
```

DynaHash Example (cont.)

```
select myfunc_setvar('var1','a1b2');
 myfunc_setvar
---------------
 OK
(1 row)

select myfunc_getvar('var1');
 myfunc_getvar
---------------
 a1b2
(1 row)

select myfunc_rmvar('var1');
 myfunc_rmvar
--------------
 OK
(1 row)
```

Multirow Functions

Multirow Functions

- Overview
- Value-per-call
- Materialized

Multirow Functions

- Overview
- Value-per-call
- Materialized

Overview of Multirow Functions

- Typically called Set-Returning Functions, SRFs, or Table Functions
- Return type may be scalar or composite
- Used in FROM clause similar to tables, views, or subqueries
- May return type `record`, which then requires use of a column definition list in the FROM clause
- In value-per-call mode, the executor calls an SRF repeatedly until it signals “done”
- In materialized mode, the executor calls the function once

Multirow Functions

- Overview
- Value-per-call
- Materialized

Value-per-call Pseudocode

```
Datum
my_Set_Returning_Function(PG_FUNCTION_ARGS)
{
     FuncCallContext       *funcctx;
     Datum                 result;
     MemoryContext     oldcontext;
     <user defined declarations>

     if (SRF_IS_FIRSTCALL())
     {
          funcctx = SRF_FIRSTCALL_INIT();
          oldcontext = MemoryContextSwitchTo(
                                    funcctx->multi_call_memory_ctx);
          <user defined code>
          <if returning composite>
               <obtain slot>
               funcctx->slot = slot;
          <endif returning composite>
          <user defined code>
          MemoryContextSwitchTo(oldcontext);
     }
```

Value-per-call Pseudocode (cont.)

```
    <user defined code>
    funcctx = SRF_PERCALL_SETUP();
    <user defined code>

    if (funcctx->call_cntr < funcctx->max_calls)
    {
        <user defined code>
        <obtain result Datum>
        SRF_RETURN_NEXT(funcctx, result);
    }
    else
        SRF_RETURN_DONE(funcctx);
}
```

Value-per-call Example

```
typedef struct
{
     ArrayType          *v;
     int16              typlen;
     bool               typbyval;
     char               typalign;
}    myfunc_fctx;

Datum
build_tuples_from_textarray_funcapi(PG_FUNCTION_ARGS)
{
     FuncCallContext    *funcctx;
     myfunc_fctx        *fctx;
     ArrayType          *v;
     int                ndims, *dim_counts, ncols = 0;
     int16              typlen;
     bool               typbyval;
     char               typalign;
     ReturnSetInfo*rsinfo = (ReturnSetInfo *) fcinfo->resultinfo;
     AttInMetadata*attinmeta;
     TupleDesc          tupdesc;
     HeapTuple          tuple;
     TupleTableSlot     *slot;
```

Value-per-call Example (cont.)

```
/* begin first-call section */

if (SRF_IS_FIRSTCALL())
{
    int                 rsinfo_ncols;
    int                 nrows = 0;
    Oid                 element_type;
    MemoryContextoldcontext;

    funcctx = SRF_FIRSTCALL_INIT();
    oldcontext = MemoryContextSwitchTo(
                                funcctx->multi_call_memory_ctx);

    v = PG_GETARG_ARRAYTYPE_P_COPY(0);
    element_type = ARR_ELEMTYPE(v);
    ndims = ARR_NDIM(v);
    dim_counts = ARR_DIMS(v);
```

Value-per-call Example (cont.)

```
/* first call section */

if (ndims == 1)
{
     nrows = 1;
     ncols = dim_counts[0];
}
else if (ndims == 2)
{
     nrows = dim_counts[0];
     ncols = dim_counts[1];
}
else
     ereport(ERROR,[...]);

get_typlenbyvalalign(element_type, &typlen,
                                   &typbyval, &typalign);
```

Value-per-call Example (cont.)

```
        tupdesc = rsinfo->expectedDesc;
        rsinfo_ncols = tupdesc->natts;
        if (rsinfo_ncols != ncols)
            ereport(ERROR, [...]);

        slot = TupleDescGetSlot(tupdesc);
        attinmeta = TupleDescGetAttInMetadata(tupdesc);
        fctx = (myfunc_fctx *) palloc(sizeof(myfunc_fctx));

        fctx->typlen = typlen;
        fctx->typbyval = typbyval;
        fctx->typalign = typalign;
        fctx->v = v;

        funcctx->user_fctx = (void *) fctx;
        funcctx->attinmeta = attinmeta;
        funcctx->slot = slot;
        funcctx->max_calls = nrows;

        MemoryContextSwitchTo(oldcontext);
    }

    /* end first-call section */
```

Value-per-call Example (cont.)

```
/* begin per-call setup section */

funcctx = SRF_PERCALL_SETUP();
slot = funcctx->slot;
attinmeta = funcctx->attinmeta;
tupdesc = attinmeta->tupdesc;
fctx = (myfunc_fctx *) funcctx->user_fctx;

typlen = fctx->typlen;
typbyval = fctx->typbyval;
typalign = fctx->typalign;
v = fctx->v;
ndims = ARR_NDIM(v);
dim_counts = ARR_DIMS(v);
if (ndims == 1)
     ncols = dim_counts[0];
else if (ndims == 2)
     ncols = dim_counts[1];

rsinfo->returnMode = SFRM_ValuePerCall;
rsinfo->setDesc = tupdesc;

/* end per-call setup section */
```

Value-per-call Example (cont.)

```
/* begin per-call return section */
if (funcctx->call_cntr < funcctx->max_calls)
{
    Datum        dvalue;
    Datum        result;
    int          *dim_lower_bounds = ARR_LBOUND(v);
    int          indx[MAXDIM];
    int          j;
    int          i = funcctx->call_cntr;
    char         **values = (char **)
                            palloc(ncols * sizeof(char *));

    for (j = 0; j < ncols; j++)
    {
        bool isnull;

        if (ndims == 1)
            indx[0] = j + dim_lower_bounds[0];
        else
        {
            indx[0] = i + dim_lower_bounds[0];
            indx[1] = j + dim_lower_bounds[1];
        }
```

Value-per-call Example (cont.)

```
            dvalue = array_ref(v, ndims, indx, -1, typlen,
                               typbyval, typalign, &isnull);
            if (!isnull)
                values[j] = DatumGetCString(
                             DirectFunctionCall1(textout, dvalue));
            else
                values[j] = NULL;
        }
        tuple = BuildTupleFromCStrings(attinmeta, values);

        result = TupleGetDatum(slot, tuple);

        SRF_RETURN_NEXT(funcctx, result);
    }
    else
        SRF_RETURN_DONE(funcctx);

    /* end per-call return section */
}
```

Value-per-call Example (cont.)

```
CREATE OR REPLACE FUNCTION build_rows_funcapi(text[])
RETURNS setof record
AS '$libdir/myfunc','build_tuples_from_textarray_funcapi'
LANGUAGE 'C' IMMUTABLE STRICT;

select a,b,c
from build_rows_funcapi(array[
     ['1','abc','today'],
     ['2','def',('today'::date + '1 week'::interval)::text]
])
as t(a int, b text, c timestamp);

 a |  b  |          c
---+-----+---------------------
 1 | abc | 2004-05-30 00:00:00
 2 | def | 2004-06-06 00:00:00
(2 rows)
```

Multirow Functions

- Overview
- Value-per-call
- Materialized

Materialized Example

```
Datum
build_tuples_from_textarray(PG_FUNCTION_ARGS)
{
    ArrayType          *v = PG_GETARG_ARRAYTYPE_P(0);
    Oid                element_type = ARR_ELEMTYPE(v);
    int                ndims = ARR_NDIM(v);
    int                *dim_counts = ARR_DIMS(v);
    int                *dim_lower_bounds = ARR_LBOUND(v);
    int                ncols = 0, nrows = 0;
    int                indx[MAXDIM];
    int16              typlen;
    bool               typbyval;
    char               typalign;
    ReturnSetInfo*rsinfo = (ReturnSetInfo *) fcinfo->resultinfo;
    AttInMetadata*attinmeta;
    TupleDesc          tupdesc;
    Tuplestorestate    *tupstore = NULL;
    HeapTuple          tuple;
    MemoryContextper_query_ctx;
    MemoryContextoldcontext;
    Datum              dvalue;
    char               **values;
    int                rsinfo_ncols, i, j;
```

Materialized Example (cont.)

```
if (ndims == 1)
{
     nrows = 1;
     ncols = dim_counts[0];
}
else if (ndims == 2)
{
     nrows = dim_counts[0];
     ncols = dim_counts[1];
}
else
     ereport(ERROR, [...]);

if (!rsinfo || !(rsinfo->allowedModes & SFRM_Materialize))
     ereport(ERROR, [...]);
```

Materialized Example (cont.)

```
get_typlenbyvalalign(element_type, &typlen,
                                          &typbyval, &typalign);

tupdesc = rsinfo->expectedDesc;
rsinfo_ncols = tupdesc->natts;
if (rsinfo_ncols != ncols)
    ereport(ERROR, [...]);
attinmeta = TupleDescGetAttInMetadata(tupdesc);

rsinfo->returnMode = SFRM_Materialize;
per_query_ctx = fcinfo->flinfo->fn_mcxt;
oldcontext = MemoryContextSwitchTo(per_query_ctx);

tupstore = tuplestore_begin_heap(true, false, SortMem);
values = (char **) palloc(ncols * sizeof(char *));
```

Materialized Example (cont.)

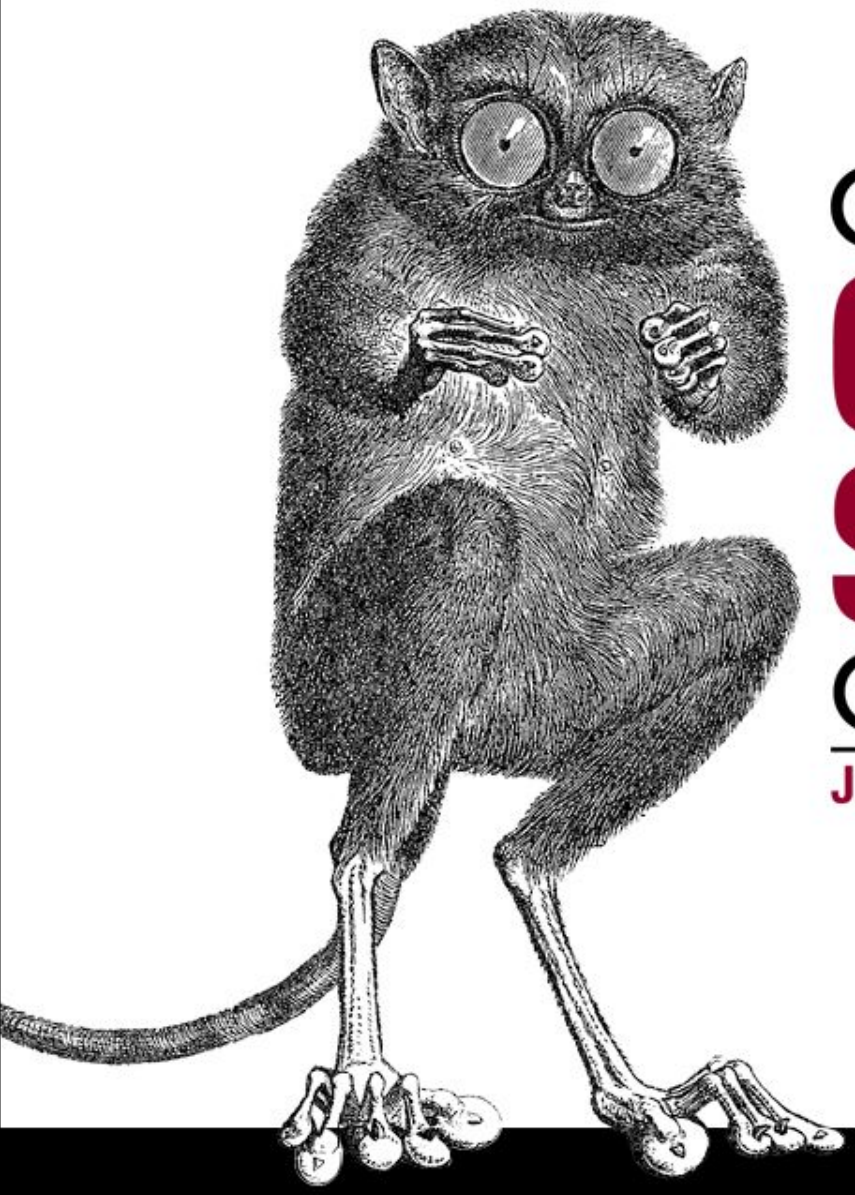
```
for (i = 0; i < nrows; i++)
{
     for (j = 0; j < ncols; j++)
     {
          bool isnull;
          if (ndims == 1)
               indx[0] = j + dim_lower_bounds[0];
          else
          {
               indx[0] = i + dim_lower_bounds[0];
               indx[1] = j + dim_lower_bounds[1];
          }
          dvalue = array_ref(v, ndims, indx, -1, typlen,
                                   typbyval, typalign, &isnull);
          if (!isnull)
               values[j] = DatumGetCString(
                              DirectFunctionCall1(textout, dvalue));
          else
               values[j] = NULL;
     }
     tuple = BuildTupleFromCStrings(attinmeta, values);
     tuplestore_puttuple(tupstore, tuple);
}
```

Materialized Example (cont.)

```
    tuplestore_donestoring(tupstore);
    rsinfo->setResult = tupstore;
    rsinfo->setDesc = tupdesc;
    MemoryContextSwitchTo(oldcontext);

    return (Datum) 0;
}
```

Materialized Example (cont.)

```
CREATE OR REPLACE FUNCTION build_rows_tuplestore(text[])
RETURNS setof record
AS '$libdir/myfunc','build_tuples_from_textarray'
LANGUAGE 'C' IMMUTABLE STRICT;

select a,b,c
from build_rows_tuplestore(array[
     ['1','abc','today'],
     ['2','def',('today'::date + '1 week'::interval)::text]
])
as t(a int, b text, c timestamp);

 a |  b  |          c
---+-----+---------------------
 1 | abc | 2004-05-30 00:00:00
 2 | def | 2004-06-06 00:00:00
(2 rows)
```



O'REILLY®

OPEN SOURCE CONVENTION™

JULY 26–30, 2004 • PORTLAND, OR

License