Writing A Foreign Data Wrapper

Bernd Helmle, bernd.helmle@credativ.de

17. Mai 2012

Why FDWs?

- ...it is in the SQL Standard (SQL/MED)
- ...migration
- ...heterogeneous infrastructure
- ...integration of remote non-relational datasources
- ...fun

Access remote datasources as PostgreSQL tables...

```
CREATE SERVER centosifx_tcp
FOREIGN DATA WRAPPER informix_fdw
OPTIONS (informixserver 'centosifx_tcp',
         connection_costs '250');

CREATE USER MAPPING FOR CURRENT_USER
SERVER centosifx_tcp
OPTIONS (username 'informix', password 'informix');

CREATE FOREIGN TABLE ifx_fdw_test (
       id integer,
       value integer
    )
SERVER centosifx_tcp
OPTIONS ( query 'SELECT * FROM foo',
          database 'test',
          informixdir '/Applications/IBM/informix');

SELECT * FROM ifx_fdw_test ;
```

What we need...

- ...a C-interface to our remote datasource
- ...knowledge about PostgreSQL's FDW API
- ...an idea how we deal with errors
- ...how remote data can be mapped to PostgreSQL datatypes
- ...time and steadiness

Python-Gurus also could use `http://multicorn.org/`.

Before you start your own...

Have a look at
`http://wiki.postgresql.org/wiki/Foreign_data_wrappers`

Let's start...

```
extern Datum ifx_fdw_handler(PG_FUNCTION_ARGS);
extern Datum ifx_fdw_validator(PG_FUNCTION_ARGS);

CREATE FUNCTION ifx_fdw_handler() RETURNS fdw_handler
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;

CREATE FUNCTION ifx_fdw_validator(text[], oid) RETURNS void
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;

CREATE FOREIGN DATA WRAPPER informix_fdw
  HANDLER ifx_fdw_handler
  VALIDATOR ifx_fdw_validator;
```

FDW handler

Creates and initializes a `FdwRoutine` structure:

```
Datum
ifx_fdw_handler(PG_FUNCTION_ARGS)
{
    FdwRoutine *fdwRoutine = makeNode(FdwRoutine);

    fdwRoutine->PlanForeignScan    = ifxPlanForeignScan;
    fdwRoutine->ExplainForeignScan = ifxExplainForeignScan;
    fdwRoutine->BeginForeignScan   = ifxBeginForeignScan;
    fdwRoutine->IterateForeignScan = ifxIterateForeignScan;
    fdwRoutine->EndForeignScan     = ifxEndForeignScan;
    fdwRoutine->ReScanForeignScan  = ifxRescanForeignScan;

    PG_RETURN_POINTER(fdwRoutine);
}
```

FDW validator callback

- Called via `CREATE FOREIGN TABLE` or `ALTER FOREIGN TABLE`
- Validates a `List *` of FDW options.
- Use `untransformRelOptions()` to get a list of FDW options
- Don't forget to test for duplicated options!
- Up to you which options you want to support

Helper functions

Functions to ease access to FDW options

```
foreign/foreign.h

extern ForeignServer *GetForeignServerByName(const char *name,
                                             bool missing_ok);

extern UserMapping *GetUserMapping(Oid userid, Oid serverid);

extern ForeignDataWrapper
*GetForeignDataWrapperByName(const char *name,
                             bool missing_ok);

extern ForeignTable *GetForeignTable(Oid relid);

extern Oid  get_foreign_data_wrapper_oid(const char *fdwname,
                                         bool missing_ok);
extern Oid  get_foreign_server_oid(const char *servername,
                                   bool missing_ok);
```

FDW API callback routines

```
static FdwPlan *PlanForeignScan(Oid foreignTableOid,
                                PlannerInfo *planInfo,
                                        RelOptInfo *baserel);

static void
ExplainForeignScan(ForeignScanState *node,
                           ExplainState *es);

static void
BeginForeignScan(ForeignScanState *node, int eflags);

static TupleTableSlot *IterateForeignScan(ForeignScanState *node);

static void EndForeignScan(ForeignScanState *node);
```

PlanForeignScan() (1)

```
FdwPlan *
PlanForeignScan (Oid foreigntableid,
                 PlannerInfo *root,
                 RelOptInfo *baserel);
```

- Plans a scan on a foreign datasource
- E.g. establish and cache remote connection
- Initialize required supporting structures for remote access
- Planner info and cost estimates via `baserel` and `root` parameters.
- Significant changes with 9.2

PlanForeignScan() (2)

FDW Cost estimates for the planner

```
FdwPlan *plan;
plan = makeNode(FdwPlan);

if (!conn_cached)
    plan->startup_cost = 500;
else
    plan->startup_cost = 100;

baserel->rows = ifxGetEstimatedNRows(&coninfo);
baserel->width = ifxGetEstimatedRowSize(&coninfo);

/* very crude */
plan->total_cost = plan->startup_cost
                   + ifxGetEstimatedCosts(&coninfo,
                                          baserel->rows, baserel->width);
```

PlanForeignScan() (3)

- Save parameters in the `plan->fdw_private` pointer.
- Must be copieable with `copyObject`.
- Use a `List *` with either `bytea` or/and constant values (via `makeConst`).

```
List *plan_values;

plan_values = NIL;
plan_values = lappend(plan_values,
                      makeConst(BYTEAOID, -1, InvalidOid, -1,
                                PointerGetDatum(ifxFdwPlanDataAsBytea(coninfo)),
                                false, false));
plan->fdw_private = plan_values;
```

PlanForeignScan() (4)

- baserel->baserestrictinfo: List of predicates belonging to the foreign table (logically AND'ed)
- baserel->reltargetlist: List of Var expressions specified in the SELECT target list belonging to the foreign table

```
List *restrictInfos;
ListCell *cell;

foreach(cell, baserel->baserestrictinfo)
{
    RestrictInfo *info;
    info = (RestrictInfo *) lfirst(cell);

    if (IsA(info->clause, OpExpr))
    {
        /* examine right and left operand */
    }
}
```

BeginForeignScan()

```
void
BeginForeignScan (ForeignScanState *node,
                  int eflags);
```

- Execute startup callback for the FDW.
- Basically prepares the FDW for executing a scan.
- `ForeignScanState` saves function state values.
- Use `node->fdw_state` to assign your own FDW state structure.
- Must handle `EXPLAIN` and `EXPLAIN ANALYZE` by checking `eflags & EXEC_FLAG_EXPLAIN_ONLY`

ExplainForeignScan()

```
void
ExplainForeignScan (ForeignScanState *node,
                    ExplainState *es);
```

- Only ran when `EXPLAIN` is used.
- "Injects" `EXPLAIN` information.
- If there's no additional information, just return
- E.g. calculated connection costs, timings etc.

IterateForeignScan() (1)

```
TupleTableSlot *
IterateForeignScan (ForeignScanState *node);
```

- Fetches data from the remote source.
- Data conversion
- Materializes a physical or virtual tuple to be returned.
- Needs to return an empty tuple when done.

IterateForeignScan() (2)

Returning a virtual tuple

```
TupleTableSlot *slot = node->ss.ss_ScanTupleSlot;

slot->tts_isempty = false;
slot->tts_nvalid = number_cols;;
slot->tts_values = (Datum *)palloc(sizeof(Datum) * slot->tts_nvalid);
slot->tts_isnull = (bool *)palloc(sizeof(bool) * slot->tts_nvalid);

for (i = 0; j < attrCount - 1; i)
{
    tupleSlot->tts_isnull[i] = false;
    tupleSlot->tts_values[i] = PointerGetDatum(val);
}
```

ReScanForeignScan()

```
void ReScanForeignScan (ForeignScanState *node);
```

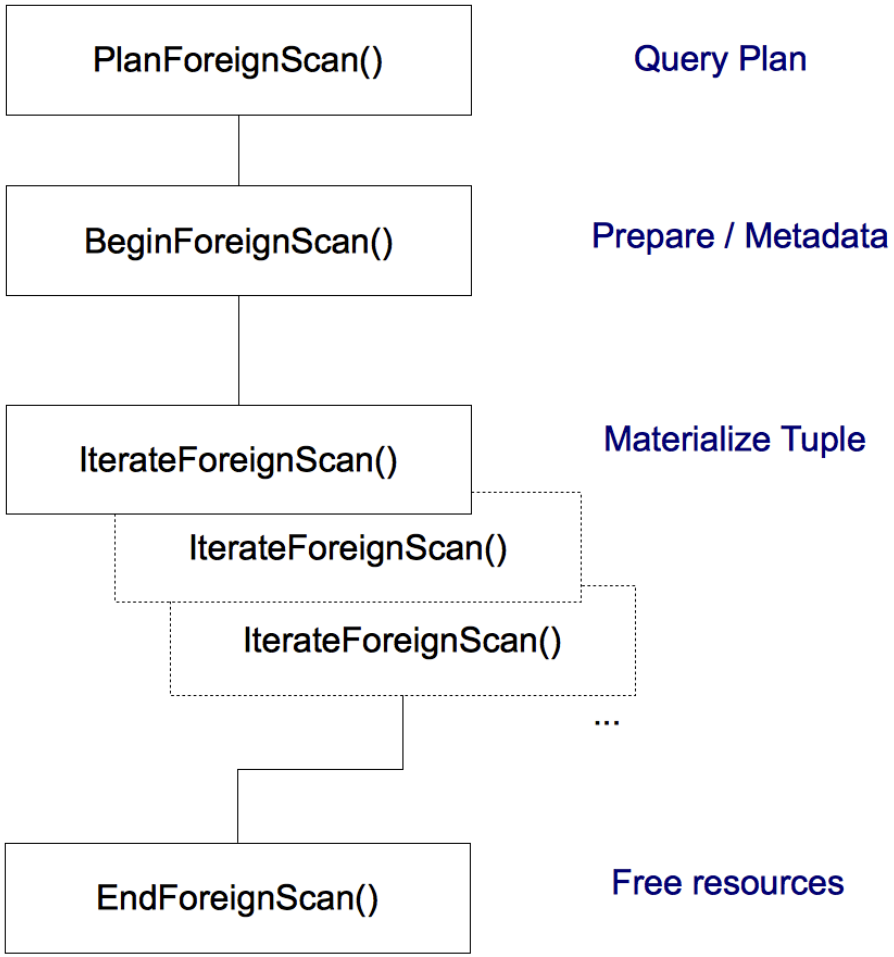
- Prepares the FDW to handle a rescan
- Begins the scan from the beginning
- Must take care for changed query parameters!
- Better to just "instruct" `IterateForeignScan()` to do the right thing (tm)

EndForeignScan()

```
void
EndForeignScan (ForeignScanState *node);
```

- Run when `IterateForeignScan` returns no more rows
- Finalizes the remote scan
- Close result sets, handles, connection, free memory, etc...

FDW Flow


PlanForeignScan()
Query Plan
BeginForeignScan()
Prepare / Metadata
IterateForeignScan()
Materialize Tuple
IterateForeignScan()
IterateForeignScan()
...
EndForeignScan()
Free resources

Memory Management

- PostgreSQL uses `palloc()`
- Memory is allocated in `CurrentMemoryContext`
- Use your own MemoryContext where necessary (e.g. `IterateForeignScan()`)
- Memory allocated in external libraries need special care

Data conversion

- Easy, if the remote datasource delivers a well formatted value string (e.g. date strings formatted as `yyyy-mm-dd`).
- Use type input function directly
- Binary compatible types (e.g integer)
- Binary data should always be `bytea`
- String data must have a valid encoding!

Data conversion - Encoding

- Within a FDW, a backend acts like any other client: ensure encoding compatibility or encode your string data properly.
- A look at `mb/pg_wchar.h` might be of interest.
- `GetDatabaseEncoding()`
- `pg_do_encoding_conversion()`

Data conversion - Get type input function

```
regproc result;
HeapTuple type_tuple;

type_tuple = SearchSysCache1(TYPEOID, inputOid);
if (!HeapTupleIsValid(type_tuple))
{
    /*
     * Oops, this is not expected...
     */
    ifxRewindCallstack(&(state->stmt_info));
    elog(ERROR,
         "cache lookup failed for input function for type %u", inputOid);
}

ReleaseSysCache(type_tuple);
result = ((Form_pg_type) GETSTRUCT(type_tuple))->typinput;
```

Data conversion - Calling type input functions

Once having its OID, any type input function can be called like this:

```
/* errors out */
typinputfunc = getTypeInputFunction(state, PG_ATTRTYPE_P(state, attnum));
result = OidFunctionCall2(typinputfunc,
                          CStringGetDatum(buf),
                          ObjectIdGetDatum(InvalidOid));
```

Error Handling (1)

- Set FDW SQLSTATE according to your error condition *class HV*, see *http://www.postgresql.org/docs/9.1/static/errcodes-appendix.html*
- Alternative: map remote error conditions to PostgreSQL errors
- Be careful with `elog(ERROR, ...)`.

Error Handling (2)

Example (there is no FDW_WARNING SQLSTATE):

```
if (err == IFX_CONNECTION_WARN)
{
    IfxSqlStateMessage message;
    ifxGetSqlStateMessage(1, &message);

    ereport(WARNING, (errcode(WARNING),
                      errmsg("opened informix connection with warnings"),
                      errdetail("informix SQLSTATE %s: \"%s\"",
                                message.sqlstate, message.text)));
}
```

Catching Errors (1)

- Sometimes necessary to catch backend errors
- Synchronize error conditions between PostgreSQL and remote datasource
- Possibility: use a `PG_TRY...PG_CATCH` block.

Catching Errors (2)

```
PG_TRY();
{
    ...
    typinputfunc = getTypeInputFunction(state, PG_ATTRTYPE_P(state, attnum));
    result = OidFunctionCall2(typinputfunc,
      CStringGetDatum(pstrdup(buf)),
      ObjectIdGetDatum(InvalidOid));
}
PG_CATCH();
{
    ifxRewindCallstack(&(state->stmt_info));
    PG_RE_THROW();
}
PG_END_TRY();
```

Thank You!