Why FDWs?

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

CREATE EXTENSION IF NOT EXISTS informix_fdw;

CREATE SERVER sles11_tcp FOREIGN DATA WRAPPER informix_fdw OPTIONS (
    informixdir '/Applications/IBM/informix',
    informixserver 'ol_informix1170'
);

CREATE USER MAPPING FOR bernd SERVER sles11_tcp OPTIONS (
    password 'informix',
    username 'informix'
);

CREATE FOREIGN TABLE bar (  
id integer,
  value text  
)
SERVER sles11_tcp
OPTIONS (  
    client_locale 'en_US.utf8', database 'test',
    db_locale 'en_US.819', query 'SELECT * FROM bar'
);

SELECT * FROM bar;
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
Implementation example

https://github.com/credativ/informix_fdw
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, example:

Datum

defx_fdw_handler(PG_FUNCTION_ARGS)
{
    FdwRoutine *fdwRoutine = makeNode(FdwRoutine);
    fdwRoutine->ExplainForeignScan = defxExplainForeignScan;
    fdwRoutine->BeginForeignScan = defxBeginForeignScan;
    fdwRoutine->IterateForeignScan = defxIterateForeignScan;
    fdwRoutine->EndForeignScan = defxEndForeignScan;
    fdwRoutine->ReScanForeignScan = defxReScanForeignScan;

#if PG_VERSION_NUM < 90200
    fdwRoutine->PlanForeignScan = defxPlanForeignScan;
#else
    fdwRoutine->GetForeignRelSize = defxGetForeignRelSize;
    fdwRoutine->GetForeignPaths = defxGetForeignPaths;
    fdwRoutine->GetForeignPlan = defxGetForeignPlan;
#endif

    PG_RETURN_POINTER(fdwRoutine);
}
FDW handler

...and much more callbacks for DML with PostgreSQL 9.3

Datum
ifx_fdw_handler(PG_FUNCTION_ARGS)
{
  [...]

  #if PG_VERSION_NUM >= 90300

  fdwRoutine->AddForeignUpdateTargets = ifxAddForeignUpdateTargets;
  fdwRoutine->PlanForeignModify = ifxPlanForeignModify;
  fdwRoutine->BeginForeignModify = ifxBeginForeignModify;
  fdwRoutine->ExecForeignInsert = ifxExecForeignInsert;
  fdwRoutine->ExecForeignDelete = ifxExecForeignDelete;
  fdwRoutine->ExecForeignUpdate = ifxExecForeignUpdate;
  fdwRoutine->EndForeignModify = ifxEndForeignModify;
  fdwRoutine->IsForeignRelUpdatable = ifxIsForeignRelUpdatable;

  #endif

  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

- Have a look into foreign/foreign.h for various helper functions
FDW API callback routines (1)

```c
#ifdef PG_VERSION_NUM < 90200

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

static void GetForeignRelSize(PlannerInfo *root,
                               RelOptInfo *baserel,
                               Oid foreignTableId);
static void GetForeignPaths(PlannerInfo *root,
                             RelOptInfo *baserel,
                             Oid foreignTableId);
static ForeignScan *GetForeignPlan(PlannerInfo *root,
                                    RelOptInfo *baserel,
                                    Oid foreignTableId,
                                    ForeignPath *best_path,
                                    List *tlist,
                                    List *scan_clauses);
#endif
```
FDW API callback routines (2)

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

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

static TupleTableSlot *IterateForeignScan(ForeignScanState *node);

static void EndForeignScan(ForeignScanState *node);
9.2 has callbacks for ANALYZE, too:

```c
bool
AnalyzeForeignTable (Relation relation,
                      AcquireSampleRowsFunc *func,
                      BlockNumber *totalpages);
```

```c
int
AcquireSampleRowsFunc (Relation relation, int elevel,
                        HeapTuple *rows, int targrows,
                        double *totalrows,
                        double *totaldeadrows);
```
9.3 introduces callbacks for DML actions

#if PG_VERSION_NUM >= 90300

fdwRoutine->AddForeignUpdateTargets = ifxAddForeignUpdateTargets;
fdwRoutine->PlanForeignModify = ifxPlanForeignModify;
fdwRoutine->BeginForeignModify = ifxBeginForeignModify;
fdwRoutine->ExecForeignInsert = ifxExecForeignInsert;
fdwRoutine->ExecForeignDelete = ifxExecForeignDelete;
fdwRoutine->ExecForeignUpdate = ifxExecForeignUpdate;
fdwRoutine->EndForeignModify = ifxEndForeignModify;
fdwRoutine->IsForeignRelUpdatable = ifxIsForeignRelUpdatable;
fdwRoutine->ExplainForeignModify = ifxExplainForeignModify;

#endif
FDW Flow

- GetForeignRelSize()
- GetForeignPaths()
- GetForeignPlan()
- PlanForeignModify()
- IsForeignRelUpdatable()
- BeginForeignScan()
- BeginForeignModify()
- IterateForeignScan()
- ExecForeignUpdate()
- ExecForeignDelete()
- ExecForeignInsert()
- EndForeignScan()
- EndForeignModify()

AddForeignTargets()

Query Cost Estimates
Query Planning
Query Planning - Finalize
Prepare / Metadata

Materialize

Free resources
AddForeignTargets

```c
static void
ifxAddForeignUpdateTargets(Query *parsetree,
    RangeTblEntry *target_rte,
    Relation target_relation);
```

- *Injects* a custom column into the parsetree for DELETE and UPDATE commands.

- Can be NULL in case DELETE is not supported or UPDATE actions rely on unchanged unique keys.
FDW Query Planning

- Setup and Planning a scan or modify action 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.
- Big differences between 9.1 and 9.2 API, DML introduced with 9.3 API
Size estimates for remote datasource (table size, number of rows, ...)

- root: Query Information Structure

- baserel: Table Information Structure, carry your FDW private information in baserel->fdw_private.

- baserel->fdw_private is a pointer to your state data structure.
baserel->rows = ifxGetEstimatedNRows(&coninfo);
baserel->width = ifxGetEstimatedRowSize(&coninfo);
baserel->fdw_private = (void *) planState;
GetForeignPaths() (1)

- Create access path for foreign datasource.

- ForeignPath access path required at least.

- Multiple paths possible (e.g. presorted results, ...)

- Arbitrarily complex
planState = (IfxFdwPlanState *) baserel->fdw_private;

/*
 * Create a generic foreign path for now. We need to consider any
 * restriction quals later, to get a smarter path generation here.
 *
 * For example, it is quite interesting to consider any index scans
 * or sorted output on the remote side and reflect it in the
 * chosen paths (helps nested loops et al.).
 */
add_path(baserel, (Path *)
    create_foreignscan_path(root, baserel,
        baserel->rows,
        planState->coninfo->planData.costs,
        planState->coninfo->planData.costs,
        NIL,
        NULL,
        NIL));
GetForeignPlan() (1)

- Creates a final ForeignScan plan node based on paths created by GetForeignPaths()

- Additional parameters

- ForeignPath *best_path: Chosen foreign access path (best)

- List *tlist: Target list

- List *scan_clauses: Restriction clauses enforced by the plan
ForeignScan plan node should be created by `make_foreignscan()`:

```c
ForeignScan *
make_foreignscan(List *qptlist,
    List *qpqual,
    Index scanrelid,
    List *fdw_exprs,
    List *fdw_private)
```

- **fdw_exprs**: Expressions to be evaluated by the planner
- **fdw_private**: Private FDW data
Save parameters in the `foreignScan->fdw_private` pointer.

Must be copiable with `copyObject`.

Use a `List *` with either bytea or/and constant values (via `makeConst`).

```c
List *plan_values;

plan_values = NIL;
plan_values = lappend(plan_values,
    makeConst(BYTEAOID, -1, InvalidOid, -1,
        PointerGetDatum(ifxFdwPlanDataAsBytea(coninfo)),
        false, false));
```
Challenge: Filter the data on the remote dataset before transferring them

- Nobody wants to filter thousands of rows to just get one
- Idea: push filter conditions down to the foreign datasource (if possible)
- Ideally done during planning phase (GetForeignRelSize(), GetForeignPaths())
- baserel->baserestrictinfo
- Hard to get it right
Predicate Pushdown

- `baserel->baserestrictinfo`: List of predicates belonging to the foreign table (logically AND'ed)
- `baserel->reltargetlist`: List of columns belonging to the foreign table
- Have a look at `expression_tree_walker()` and `ruleutils` API
  (include/nodes/nodeFuncs.h, include/utils/ruleutils.h)

```c
ListCell *cell;

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

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

```sql
SELECT COUNT(*) FROM sles11.inttest;
  count
  -------
  10001
(1 row)

EXPLAIN SELECT * FROM foo JOIN centosifx.inttest t
  ON (t.f1 = foo.id)
  WHERE t.f1 = 104 AND t.f2 = 120;

QUERY PLAN
---------------------------------------------------------------------------------
Nested Loop (cost=2.00..1695.03 rows=1 width=18)
  ->  Seq Scan on foo (cost=0.00..1693.01 rows=1 width=4)
      Filter: (id = 104)
  ->  Foreign Scan on inttest t (cost=2.00..2.01 rows=1 width=14)
      Filter: ((f1 = 104) AND (f2 = 120))
      Informix costs: 2.00
      Informix query: SELECT * FROM inttest WHERE (f1 = 104) AND (f2 = 120)
(7 rows)
```
PlanForeignModify

- Create modify state information, e.g. get a connection, describe remote metadata, ...
- Private state information attached to ModifyTable plan node
- State information pushed down to BeginForeignModify
- Same with planning a scan: private state information must be in a List * format.
- **INSERT** probably needs more work here (connection preparing, ...)
Possible to access the scan state created during planning phase

```c
static List *
ifxPlanForeignModify(PlannerInfo *root,
    ModifyTable *plan,
    Index resultRelation,
    int subplan_index)
{
    if ((resultRelation < root->simple_rel_array_size)
        && (root->simple_rel_array[resultRelation] != NULL))
    {
        RelOptInfo *relInfo = root->simple_rel_array[resultRelation];
        IfxFdwExecutionState *scan_state;

        /*
         * Extract the state of the foreign scan.
         */
        scan_state = (IfxFdwExecutionState *)
            ((IfxFdwPlanState *)relInfo->fdw_private)->state;
    }
}
```
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
BeginForeignModify

- Like BeginForeignScan(), starting callback for DML actions into the executor.
- Depending on the DML action, might be differences in preparing stuff
- E.g. setup connection or get connection from cache
- Deserialize plan data pushed down from PlanForeignModify()
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.
- Private FDW data located in node->fdw_state
Returning a virtual tuple

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

- Prepares the FDW to handle a rescan

- Begins the scan from the beginning, e.g. when used in scrollable cursors

- Must take care for changed query parameters!

- Better to just “instruct” IterateForeignScan() to do the right thing (tm)
Execute a modify action(1)

```c
static TupleTableSlot *
ifxExecForeignUpdate(EState *estate,
    ResultRelInfo *rinfo,
    TupleTableSlot *slot,
    TupleTableSlot *planSlot)

static TupleTableSlot *
ifxExecForeignInsert(EState *estate,
    ResultRelInfo *rinfo,
    TupleTableSlot *slot,
    TupleTableSlot *planSlot);

static TupleTableSlot *
ifxExecForeignDelete(EState *estate,
    ResultRelInfo *rinfo,
    TupleTableSlot *slot,
    TupleTableSlot *planSlot);
```
Execute a modify action (2)

ExecForeignUpdate, ExecForeignDelete and ExecForeignInsert callbacks

- Called for every tuple to work on
- An INSERT action doesn’t employ a remote scan!
- Does all the necessary data conversion if necessary
- slot carries all information for the target tuple
- Private FDW data stored in rinfo->ri_FdwState
void EndForeignScan (ForeignScanState *node);

void EndForeignModify(EState *estate,
                       ResultRelInfo *rinfo)

- EndForeignModify() run when no more tuple to modify
- EndForeignScan() run when IterateForeignScan returns no more rows, but after EndForeignModify()
- Finalizes the remote scan and modify action
- Close result sets, handles, connection, free memory, etc...
Connect local transactions with remote transactions and savepoints

RegisterXactCallback(pgfdw_xact_callback, NULL);
RegisterSubXactCallback(pgfdw_subxact_callback, NULL);

static void fdw_xact_callback(XactEvent event, void *arg);

static void fdw_subxact_callback(SubXactEvent event, SubTransactionId mySubid,
                                  SubTransactionId parentSubid, void *arg)
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!
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

```c
regproc typinputfunc;
Datum result;
HeapTuple type_tuple;

type_tuple = SearchSysCache1(TYPEOID, inputOid);
if (!HeapTupleIsValid(type_tuple))
{
    /*
     * Oops, this is not expected...
     */
    ...
}

ReleaseSysCache(type_tuple);

typinputfunc = ((Form_pg_type) GETSTRUCT(type_tuple))->typinput;
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/9.3/static/errcodes-appendix.html
- Alternative: map remote error conditions to PostgreSQL errors
- Be careful with elog(ERROR, ...).
Example (there is no FDW_WARNING SQLSTATE):

```c
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)

```c
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!