DALIBO
L'expertise PostgreSQL

Multicorn: writing foreign data wrappers in python

PostgreSQL Conference Europe 2013

Ronan Dunklau <ronan.dunklau@dalibo.com>

Table des matières

1 Multicorn: writing foreign data wrappers in python

1.1 Slides license

- Creative Common BY-NC-SA
- You are free
  - to Share
  - to Remix
- Under the following conditions
  - Attribution
  - Noncommercial
  - Share Alike

1.2 Author

- Ronan Dunklau
- Work
  - DBA at Dalibo
  - email: ronan.dunklau@dalibo.com

1.3 Agenda

- General FDW overview
- Multicorn installation and usage
- Implement your own FDW in python
- Differences with C FDWs (internals)

1.4 FDW overview

- Access remote datasources as tables
- Four object types
- Defined by SQL/MED specification

1.5 Foreign Data Wrapper

- Set of routines, implementing an API
- Usually installed as an extension

1.6 Server

- Object defining connection options
- Attached to a foreign data wrapper
- Own foreign tables

1.7 Foreign Table

- Attached to a server
- Can define more options
- Looks like a regular table
- Supports SELECT statement (9.1) as well as DML statements (9.3)

1.8 User Mapping

- Maps user settings to a server
- Useful for storing passwords

1.9 File fdw example

```
  CREATE EXTENSION file_fdw;
  CREATE server file_server FOREIGN DATA WRAPPER
file_fdw;
  CREATE FOREIGN TABLE file_table (
     city varchar, country varchar
  ) SERVER file_server OPTIONS (
     filename '/tmp/zipcodes.csv',
     encoding 'UTF8',
    delimiter ','
  )
```

2 Presentation and usage

2.1 What is Multicorn ?

- PostgreSQL extension
- Allows you to write FDW in python
- License: PostgreSQL licensed
- Developed at Kozea by Florian Mounier, Ronan Dunklau
- Code: http://github.com/Kozea/Multicorn
- Documentation: http://multicorn.org

2.2 Why Multicorn ?

- FDW development is complex
- Ease of prototyping
- Python language and ecosystem

2.3 How does it work ?

- One extension
- Offers a python API on top of the C-API
- Bundled with some wrappers / examples

2.4 What is in it ?

- SqlAlchemy (RDBMS)
- LDAP
- IMAP
- Filesystem
- Google

2.5 Installation

- Get the sources:
  1. From github: http://github.com/Kozea/Multicorn
  2. From pgxn: http://pgxn.org/dist/multicorn/

```
make && make install
CREATE EXTENSION multicorn;
```

2.6 Usage

```
  CREATE SERVER test_srv FOREIGN DATA WRAPPER multicorn
  OPTIONS (wrapper 'multicorn.testfdw');
  CREATE FOREIGN TABLE test_table (id varchar) SERVER
test_srv OPTIONS (...);
```

Specific FDW options documented at http://multicorn.org/foreign-data-wrappers/

3 Implement your own FDW in python

- Really simple API
- Inherit multicorn.ForeignDataWrapper
  - One instance per table per backend

3.1 Minimalist example

- Project setup
  - Use standard python packaging (setup.py)
  - logfdw/init.py: only the class definition

```
ro@ronan_laptop logfdw % ls -R
.:
logfdw  setup.py

./logfdw:
__init__.py
```

3.2 Setup.py

```python
import subprocess
from setuptools import setup, find_packages, Extension

setup(
 name='logfdw',
 version='0.0.1',
 author='Ronan Dunklau',
 license='Postgresql',
 packages=['logfdw']
)
```

3.3 logfdw/__init__.py

```python
from multicorn import ForeignDataWrapper

class LogFDW(ForeignDataWrapper):

    def execute(self, quals, columns):
        pass
```

3.4 Let's test it !

- Install the code
- Install the extension
- Create the server
- Create the table
- Test it !

3.5 Let's test it!

```
pip install .
CREATE EXTENSION multicorn;

CREATE SERVER log_server
  FOREIGN DATA WRAPPER multicorn
  OPTIONS (wrapper 'logfdw.LogFDW');

CREATE FOREIGN TABLE logtable (
  ts TIMESTAMP,
  message VARCHAR
) SERVER log_server;

SELECT * FROM logtable;
```

3.6 Where are we now ?

- Project structure
- Dummy FDW
- But it works

3.7 Getting useful

- Actually parse something
- Return rows
- We need options ! (log file, pattern...)

3.8 Using options

- init method (constructor)
- called whenever needed with the fdw options and the column definition
- instance cached in the backend

3.9 Using options (code)

```
class LogFDW(ForeignDataWrapper):

    def __init__(self, fdw_options, fdw_columns):
        super(LogFDW, self).__init__(fdw_options, fdw_columns)
        self.log_file = fdw_options.get('log_file', None)
        if self.log_file is None:
            raise ValueError('The log_file option is mandatory')
        # Default to matching the whole line.
        self.line_re = re.compile(fdw_options.get('line_pattern',
"(.*)"))
        if len(fdw_columns) != self.line_re.groups:
            raise ValueError('The table should have as much columns
as '
                             'there are groups in the pattern')
```

3.10 Execute method

- Parse the file
- Match lines
- Return matches

3.11 Execute method (code)

```python
    def execute(self, quals, columns):
        with open(self.log_file, 'r') as f:
            for line in f:
                match = self.line_re.match(line)
                if match:
                    yield match.groups()
```

3.12 Where are we now ?

- Simple fdw
- takes advantage of built-in python libraries
- simply gets FDW options

3.13 Optimizing lookup by date

- Assertion: log is ordered by date
- Easy to optimize: condition of the form

```sql
WHERE date < some_date
```

- Need to identify the date column

3.14 Column object

- Column name, type name, type mod
- Column options

```
class ColumnDefinition(object):

    def __init__(self, column_name, type_oid, typmod, type_name,
                 base_type_name,
                 options):
        self.column_name = column_name
        self.type_oid = type_oid
        self.typmod = typmod
        self.type_name = type_name
        self.base_type_name = base_type_name
        self.options = options or {}
```

3.15 Receiving condition

- “quals” argument
- list of “Qual object”
- field_name, operator, value
- all conditions are re-checked by PostgreSQL

3.16 Let's optimize !

- Parse the quals argument
- Stop iterating when the date is bigger than what we need

3.17 Where are we now ?

- Simple optimization for the max date
- Further optimisations possible on the date:
  - Read the file backwards for ">" conditions
  - Dichotomic search to find the lines matching the date
  - Left as an exercise to the public

3.18 Influencing the planner

- get_path_keys method
  - Return a list of possible (keys definition, expected number of rows)
  - Compared by Multicorn against EquivalenceClasses and joined clauses
  - Generate a Parameterized Path
- get_rel_size method
  - Returns a tuple of the form (number_of_rows, average_row_width)

3.19 Influencing the planner

- base table with 100 rows
- foreign table with 100000 rows
- Lookup by a specific key:

```python
def get_path_keys(self):
    return [(('id',), 1)]

def get_rel_size(self, quals, columns):
    return (100000, 100)
```

3.20 Influencing the planner

- What happens ? Without path_keys:

```
explain select * from without_index inner join
ref_values using(id);
```

```
                                    QUERY PLAN
Hash Join  (cost=57.67..4021812.67 rows=615000 width=68)
  Hash Cond: (without_index.id = ref_values.id)
  ->  Foreign Scan on without_index  (cost=20.00..4000000.00 rows=100000 width=40)
  ->  Hash  (cost=22.30..22.30 rows=1230 width=36)
        ->  Seq Scan on ref_values  (cost=0.00..22.30 rows=1230 width=36)
```

3.21 Influencing the planner

- What happens ? With path keys:

```
explain select * from with_index inner join ref_values
using(id);
```

```
                              QUERY PLAN
Nested Loop  (cost=20.00..49234.60 rows=615000 width=68)
  ->  Seq Scan on ref_values  (cost=0.00..22.30 rows=1230 width=36)
  ->  Foreign Scan on with_index  (cost=20.00..40.00 rows=1 width=40)
        Filter: (id = ref_values.id)
```

3.22 Where are we, now ?

- Simple optimizations
- Inform the planner about said optimizations
- For an actual example, look at the multicorn.imapfdw.ImapFDW class

3.23 Writing

- Available since 9.3
- Simple C-API
- Simpler python API :)

3.24 Writing

- insert(self, value)
- update(self, oldvalue, newvalue)
- delete(self, oldvalue)
- rowid_column attribute

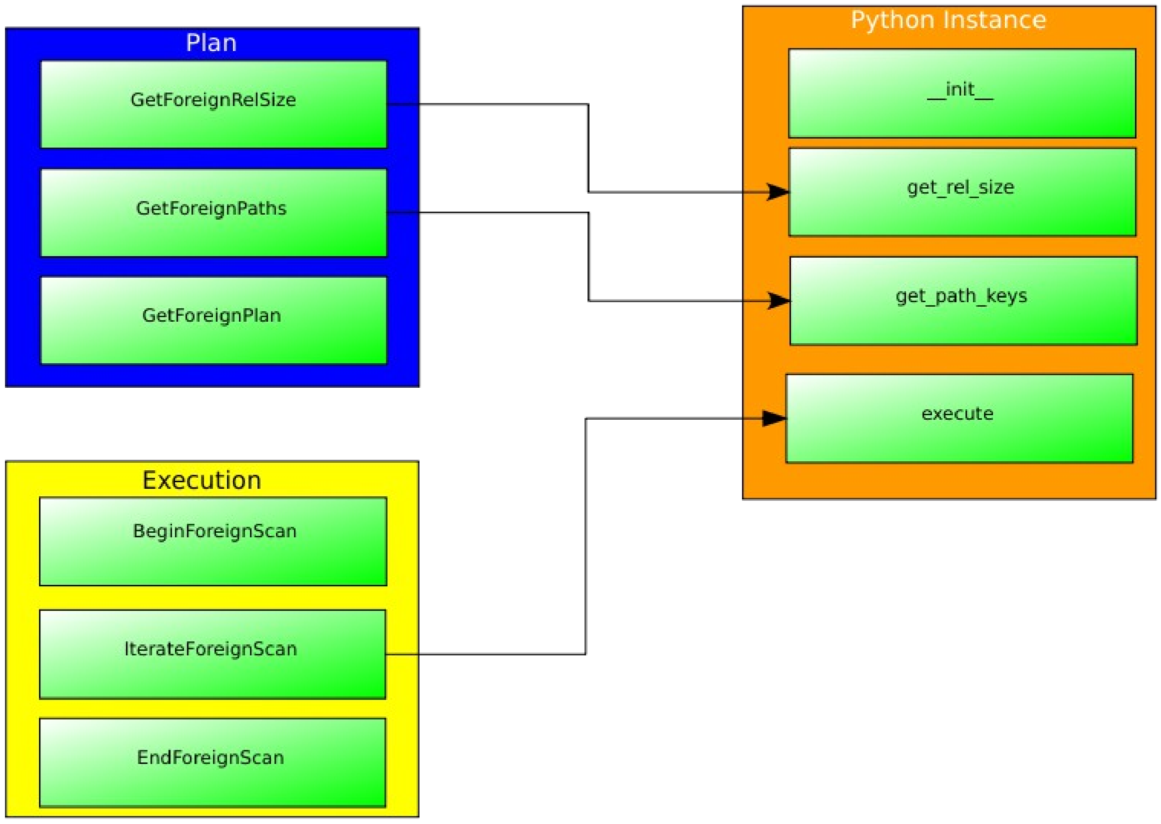
3.25 Transaction support

- pre_commit
- commit
- rollback

3.26 TransactionAwareForeignDataWrapper

- helper class to keep an internal log of what happens before commit
- No MVCC, no nothing
- Not consistent
- Doesn't help with the “core” feature for anything else than a RDBMS.

3.27 Internals



3.28 Questions ?

Thank you !