PostgreSQL FDW

Your own database backend
...in 8 lines of code

Foreign Data Wrappers

- SQL/MED
- Supports read and write
- Fast - pushing down: WHERE-clause, joins, aggregates (coming in PostgreSQL 10)
- Official contrib extensions: file_fdw, pgsql_fdw

Foreign Data Wrappers

Pros:

- online data
- easy data mapping in app
- joins with existing data
- use SQL for data processing

Use cases:

- remote, various protocols
- local, various file formats
- other databases
- data migration
- logs
- INSERT INTO SELECT FROM, MATERIALIZED VIEW

File FDW

```
CREATE EXTENSION file_fdw;

CREATE SERVER report FOREIGN DATA WRAPPER file_fdw;

CREATE FOREIGN TABLE report (
   report_date timestamp(3) with time zone,
   value integer
) SERVER report OPTIONS (filename '/tmp/report.csv', format 'csv');

IMPORT FOREIGN SCHEMA reports FROM SERVER report INTO reports;
```

Multicorn

```
# pgxn install multicorn

# cat custom_fdw.py

from multicorn import ForeignDataWrapper

class CustomForeignDataWrapper(ForeignDataWrapper):
  def __init__(self, options, columns):
     super(CustomForeignDataWrapper, self).__init__(options, columns)
     self.columns = columns

  def execute(self, quals, columns):
     for index in range(20):
        yield {column_name: '%s %s' % (column_name, index) for column_name in self.columns}
```

Multicorn

```
CREATE SERVER multicorn_custom FOREIGN DATA WRAPPER multicorn
options (
 wrapper 'custom_fdw.CustomForeignDataWrapper'
);

CREATE FOREIGN TABLE constant_table (
   test character varying,
   test2 character varying
) server multicorn_custom options (max_length 10);

SELECT test, test2 FROM constant_table WHERE 1=1;
```

```
  test    |  test2
----------+----------
 test 0   | test2 0
 test 1   | test2 1
 test 2   | test2 2
 test 3   | test2 3
 test 4   | test2 4
 test 5   | test2 5
 test 6   | test2 6
 test 7   | test2 7
 test 8   | test2 8
 test 9   | test2 9
(10 lines)
```

Multicorn

Provided wrappers:

- fsfdw
- gitfdw
- imapfdw
- rssfdw
- sqlalchemyfdw

Pros:

- low entry level
- use Python libs

Cons:

- slow
- extra serialize/deserialize step, requires mapping

HTTP API - Server

```
CREATE TABLE users (
  id INTEGER PRIMARY KEY,
  name CHARACTER VARYING
);

CREATE TABLE groups (
  id INTEGER PRIMARY KEY,
  name CHARACTER VARYING
);

CREATE TABLE group_user (
  user_id INTEGER REFERENCES users(id),
  group_id INTEGER REFERENCES groups(id),
  PRIMARY KEY (user_id, group_id)
);
```

HTTP API - Server

```
# http http://localhost:8080/groups name=default
# http http://localhost:8080/users name=jan groups_collection:='[1]'
# http http://localhost:8080/users?relations=_all
{
  "results": [
    {
      "groups_collection": [
        {
          "id": 1,
          "name": "default"
        }
      ],
      "id": 3,
      "name": "jan"
    }
  ],
  "returned": 3,
  "total": null
}
```

HTTP API - Client

```
CREATE EXTENSION multicorn;
CREATE SERVER multicorn_falcon FOREIGN DATA WRAPPER multicorn OPTIONS (
  wrapper 'pystok-fdw.falcon-api.FalconApiFDW'
);
CREATE FOREIGN TABLE groups (
 id integer,
 name character varying
) SERVER multicorn_falcon OPTIONS (url 'http://localhost:8080/groups');

CREATE FOREIGN TABLE users (
 id integer,
 name character varying,
 groups_collection json
) SERVER multicorn_falcon OPTIONS (
 url 'http://localhost:8080/users',
 params 'relations=groups_collection'
);
```

HTTP API - Client

```
import requests
from multicorn import ForeignDataWrapper

class FalconApiFDW(ForeignDataWrapper):
  def __init__(self, options, columns):
    super(FalconApiFDW, self).__init__(options, columns)
    self.options = options

  def execute(self, quals, columns):
    response = requests.get(self.options['url'])
    return response.json()['results']
```

HTTP API - Client

```
postgres=# set client_min_messages = debug;
SET
postgres=# select * from users;
DEBUG:  Request: GET http://api:80/users?relations=groups_collection
DEBUG:  Response:
{"returned":2,"total":null,"results":[{"id":1,"groups_collection":[{"id":1,"name":"default"}],"name":"janw"},{"id":3,"groups_collection":[],"name":"second"}]}
 id |  name  |        groups_collection
----+--------+--------------------------------
  1 | janw   | [{"id": 1, "name": "default"}]
  3 | second | []
(2 rows)
```

HTTP API - Client

```
postgres=# select id, name, groups_collection->0->'name' as first_group_name from users;
 id |  name   | first_group_name
----+---------+------------------
  1 | janw    | "default"
  3 | second |
(2 rows)

postgres=# explain select * from users where name like 'john%';
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Foreign Scan on users  (cost=20.00..30000000000.00 rows=100000000 width=68)
   Filter: ((name)::text ~~ 'john%'::text)
(2 rows)
```

HTTP API

What's missing:

- sending quals to the server
- insert, update, delete
- logging, error handling

Full example:

- https://github.com/nineinchnick/pystok-fdw
- src/pystok-fdw/falcon-api.py
- only 97 lines of code

Summary

- uniform access to all databases and resources
- having full SQL or joins is beneficial - see JSON support in PostgreSQL
- start easy (Multicorn), add quals, move to native if needed
- use contrib file_fdw or pgsql_fdw
- google for exotic drivers

https://github.com/nineinchnick/pystok-fdw
https://www.postgresql.org/docs/9.6/static/ddl-foreign-data.html
https://www.postgresql.org/docs/9.6/static/file-fdw.html
https://wiki.postgresql.org/wiki/Foreign_data_wrappers
http://multicorn.org/implementing-an-fdw/