PostgreSQL as an integrated data analysis platform

FOSS4G.be 2015 – Oslandia Team

Let 's try to Think Different (about PostgreSQL)

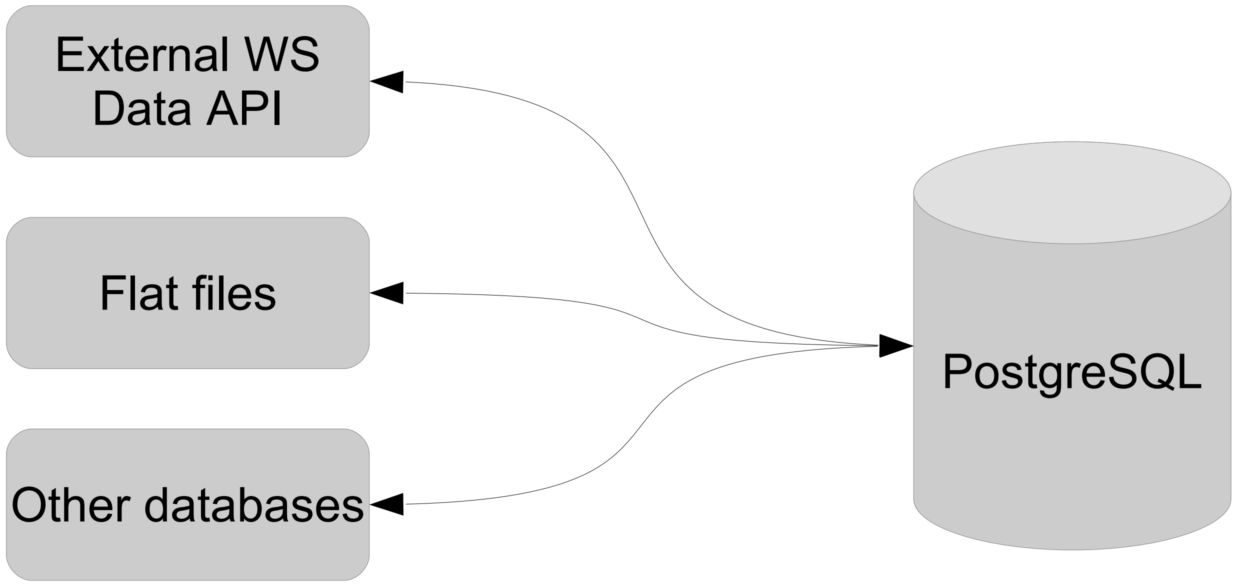
Database is not only a place to store data
(and use basic SQL to access it)

Let 's try to Think Different (about PostgreSQL)

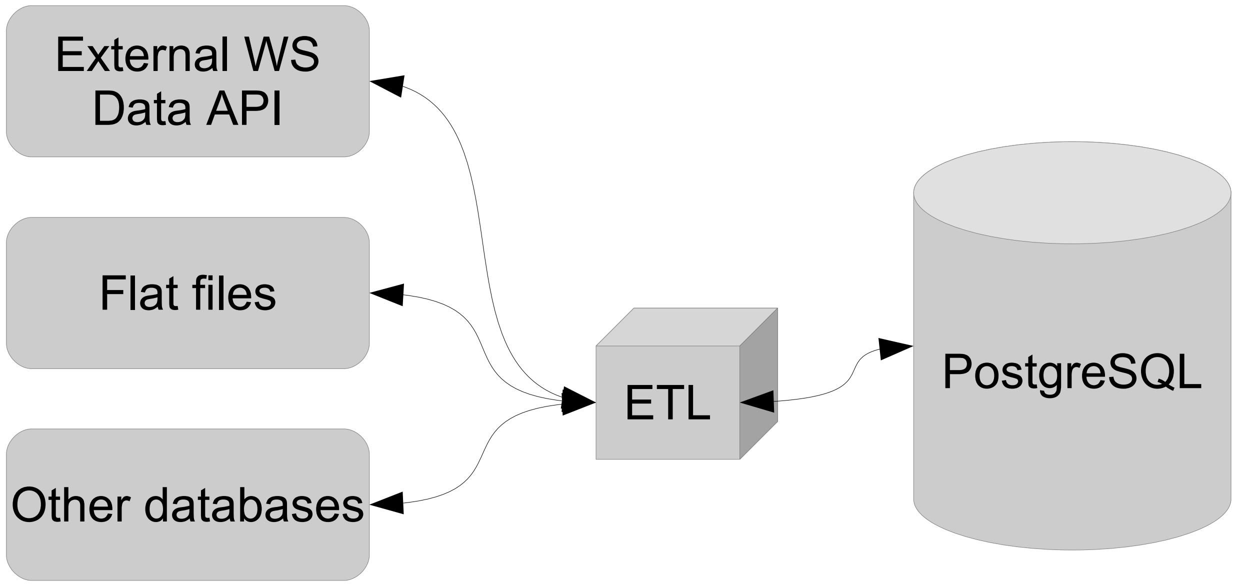
Database is not only a place to store data
(and use basic SQL to access it)

PostgreSQL is far more than an enhanced filesystem
PostgreSQL by design is extensible

#1 Data Integration

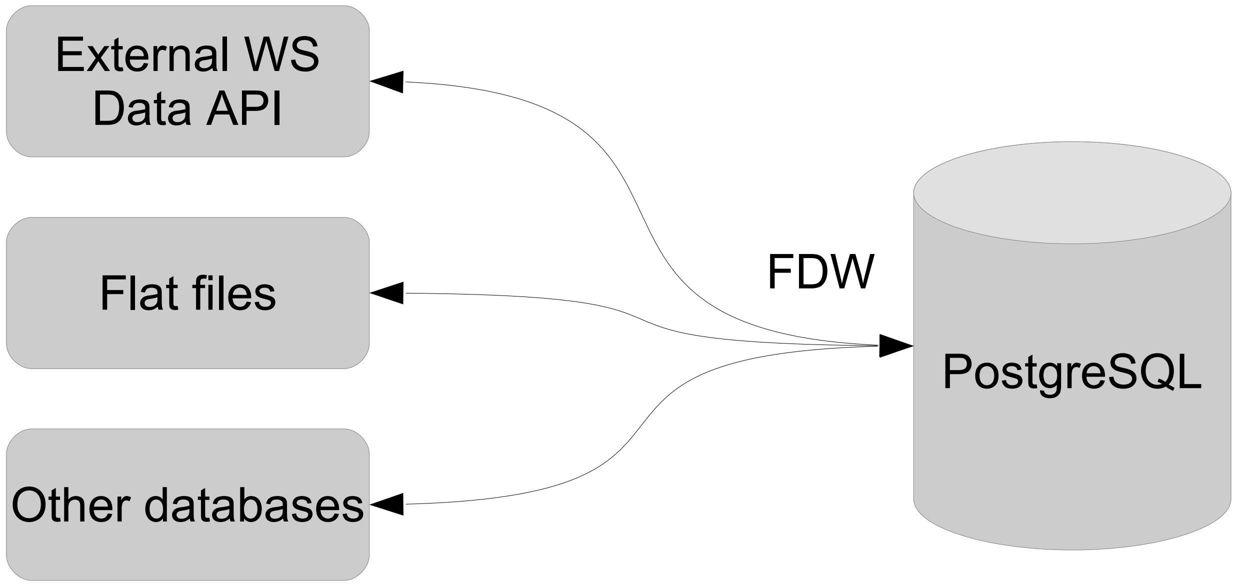


#1 Data Integration



Common answer is « Use an ETL »

#1 Data Integration

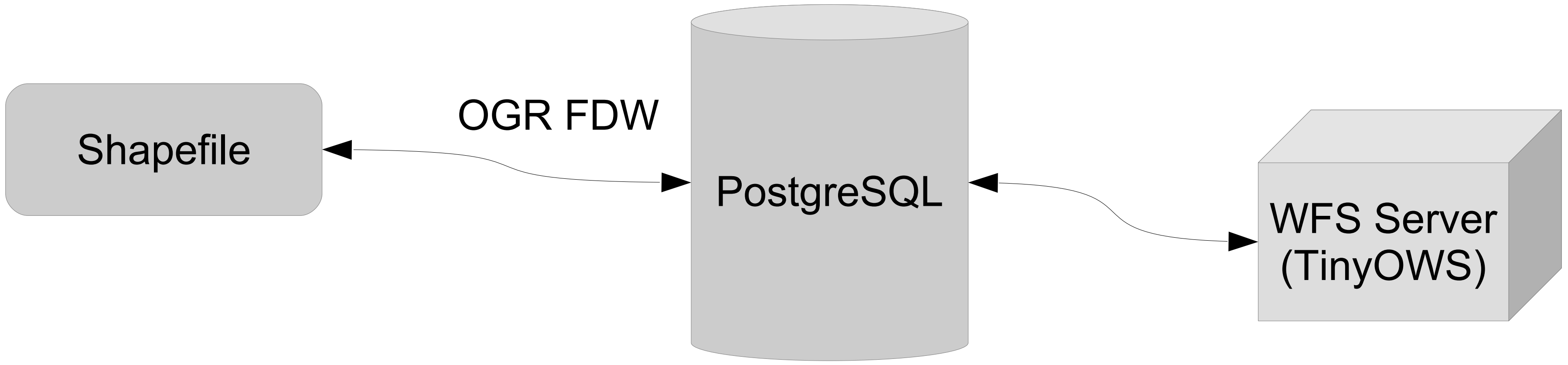


Alternate answer is
« Use PostgreSQL Foreign Data Wrapper »

https://wiki.postgresql.org/wiki/Foreign_data_wrappers

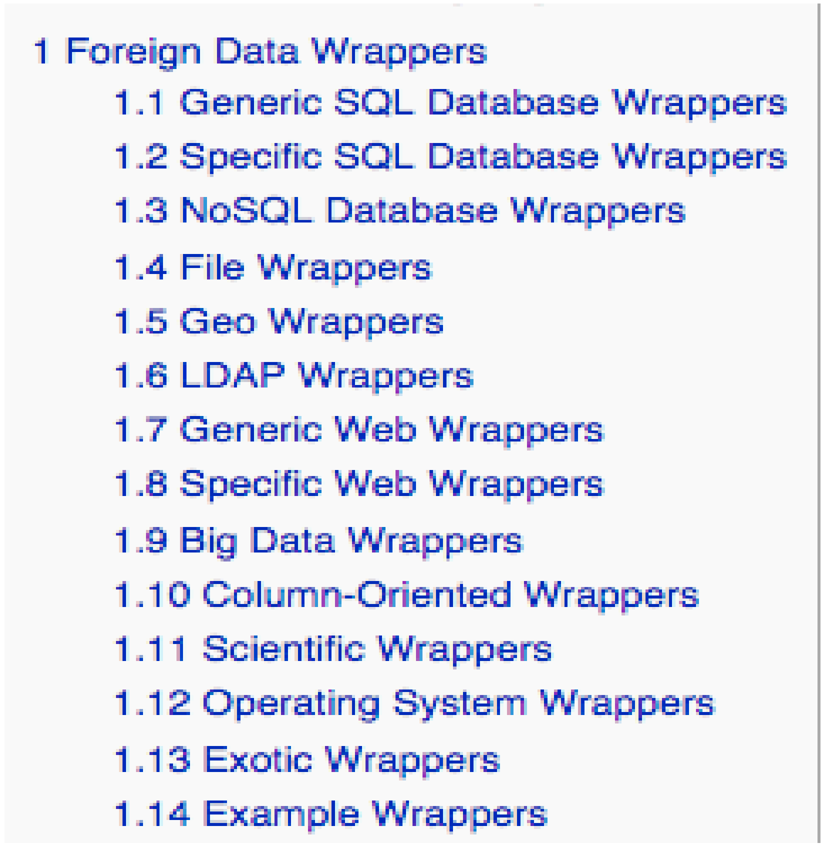
SQL Management of External Data (SQL/MED) added to the SQL standard
Handling access to remote objects from SQL databases
Available in PostgreSQL since 9.3

#1 Data integration : OGR FDW



https://wiki.postgresql.org/wiki/Foreign_data_wrappers

SQL Management of External Data (SQL/MED) added to the SQL standard
Handling access to remote objects from SQL databases
Available in PostgreSQL since 9.3



1 Foreign Data Wrappers
- 1.1 Generic SQL Database Wrappers
- 1.2 Specific SQL Database Wrappers
- 1.3 NoSQL Database Wrappers
- 1.4 File Wrappers
- 1.5 Geo Wrappers
- 1.6 LDAP Wrappers
- 1.7 Generic Web Wrappers
- 1.8 Specific Web Wrappers
- 1.9 Big Data Wrappers
- 1.10 Column-Oriented Wrappers
- 1.11 Scientific Wrappers
- 1.12 Operating System Wrappers
- 1.13 Exotic Wrappers
- 1.14 Example Wrappers

~50 native connectors already available
(And more throught Multicorn extension)

https://github.com/pramsey/pgsql-ogr-fdw

Install OGR FDW

```
git clone https://github.com/pramsey/pgsql-ogr-fdw.git
cd pgsql-ogr-fdw
make
sudo make install
```

Define a FDW wrapper

```
CREATE EXTENSION postgis;
CREATE EXTENSION ogr_fdw;

CREATE SERVER shapefile_france
  FOREIGN DATA WRAPPER ogr_fdw
  OPTIONS (
    datasource '/tmp/fdw_ogr/france.shp',
    format 'ESRI Shapefile'
  );
```

Retrieve shapefile attributes list (metadata)

```
ogrinfo -al -so /tmp/fdw_ogr/france.shp
```

Create Foreign table

```
CREATE SCHEMA shp;

CREATE FOREIGN TABLE shp.france (
  id_geofla integer,
  geom geometry,
  code_chf_l varchar,
  nom_chf_l varchar,
  x_chf_lieu varchar,
  y_chf_lieu varchar,
  x_centroid integer,
  y_centroid integer,
  nom_dept varchar,
  code_reg varchar,
  nom_region varchar,
  code_dept varchar
)

SERVER shapefile_france
OPTIONS (layer 'france');
```

Check it

```
SELECT id_geofla, ST_AsEWKT(ST_Centroid(geom)) AS geom
FROM shp.france LIMIT 1 ;
```

Create VIEW from Foreign Table

https://github.com/pramsey/pgsql-ogr-fdw/issues/11

```
CREATE OR REPLACE VIEW shp.france_wfs AS

SELECT id_geofla,
       ST_Multi(ST_SetSRID(geom,27572))::geometry(MultiPolygon,27572) AS geom,
       code_dept,
       nom_dept
FROM france;
```

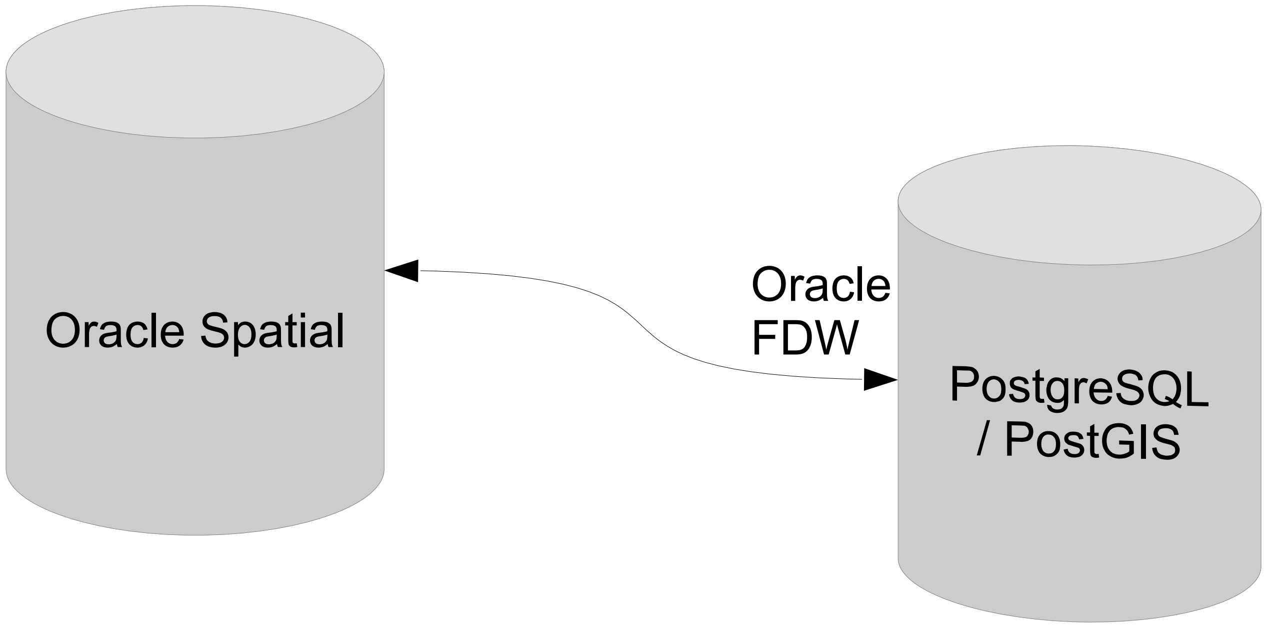
TinyOWS configuration

```
<tinyows online_resource="http://127.0.0.1/cgi-bin/tinyows"
         schema_dir="/usr/local/share/tinyows/schema/"
         estimated_extent="1"
         display_bbox="0">

  <pg host="127.0.0.1" user="pggis" password="***" dbname="db" />

  <metadata name="TinyOWS WFS Server"
            title="TinyOWS Server – OGR FDW Service" />

  <layer retrievable="1"
         writable="0"
         ns_prefix="tows"
         ns_uri="http://www.tinyows.org/"
         schema="shp"
         name="france_wfs"
         title="france" />

</tinyows>
```

Check it

```
wget -O out http://127.0.0.1/cgi-bin/tinyows?
SERVICE=WFS&REQUEST=GetFeature&Typename=tows:france_wfs
```

#1 Data integration : Oracle FDW



http://pgxn.org/dist/oracle_fdw/

```
CREATE EXTENSION postgres_fdw;
CREATE EXTENSION oracle_fdw;

CREATE SERVER orcl FOREIGN DATA WRAPPER oracle_fdw
OPTIONS (dbserver '${ORACLE_URI}');
```

Oracle user Mapping

```
GRANT USAGE ON FOREIGN SERVER orcl TO ${PGUSER};

CREATE USER orcl_map FOR ${PGUSER}
SERVER orcl
OPTIONS (user '${ORAUSER}', password '${ORAPWD}');
```

```
CREATE SCHEMA fdw;

CREATE FOREIGN TABLE fdw.foo (
     id double precision,
     label varchar,
     last_update date,
     geom geometry(POINT, 2154),
)
SERVER orcl
OPTIONS (schema '${ORAUSER}', table 'FOO');
```

```
CREATE SCHEMA mat;
CREATE MATERIALIZED VIEW mat.foo AS SELECT * FROM fdw.foo;
```

```
CREATE UNIQUE INDEX ON mat.foo(id);
CREATE INDEX ON mat.foo USING GIST(geom);
```

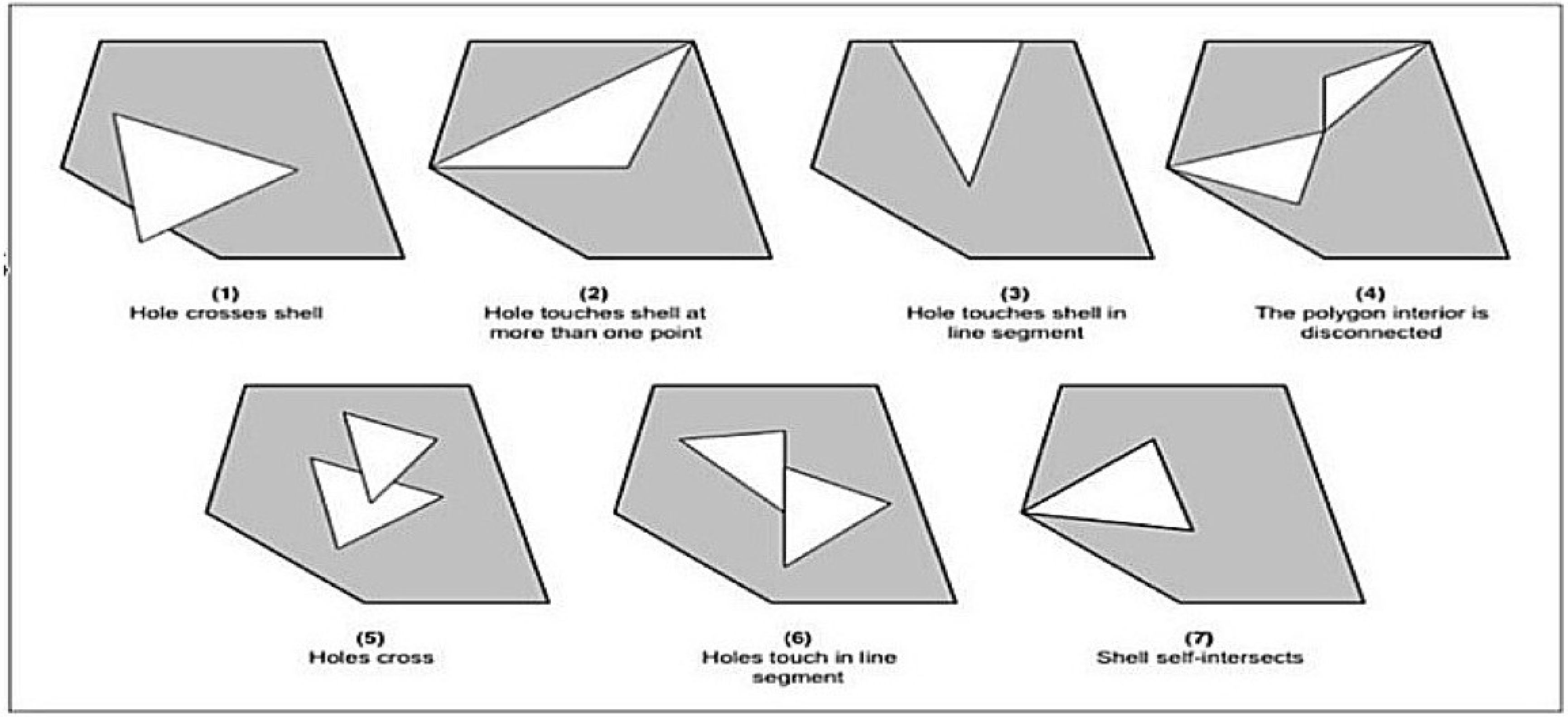
```
REFRESH MATERIALIZED VIEW CONCURRENTLY mat.foo;
```

#2 Cleaning Data: Validity

```
SELECT count(*) FROM my_schema.my_table WHERE NOT ST_IsValid(geom);
```

#2 Cleaning Data: Validity

```
SELECT count(*) FROM my_schema.my_table WHERE NOT ST_IsValid(geom);
```


(1)
Hole crosses shell
(2)
Hole touches shell at more than one point
(3)
Hole touches shell in line segment
(4)
The polygon interior is disconnected
(5)
Holes cross
(6)
Holes touch in line segment
(7)
Shell self-intersects

```
UPDATE my_schema."my_table"

SET geom=ST_CollectionExtract(ST_MakeValid(geom), 3)

WHERE ST_IsValidReason(geom) != 'Valid Geometry'
        AND (GeometryType(geom) = 'POLYGON'
        OR GeometryType(geom) = 'MULTIPOLYGON');
```

```
UPDATE my_schema."my_table"

SET geom=ST_CollectionExtract(ST_MakeValid(geom), 3)

WHERE ST_IsValidReason(geom) != 'Valid Geometry'
        AND (GeometryType(geom) = 'POLYGON'
        OR GeometryType(geom) = 'MULTIPOLYGON');
```

Still to deal with :

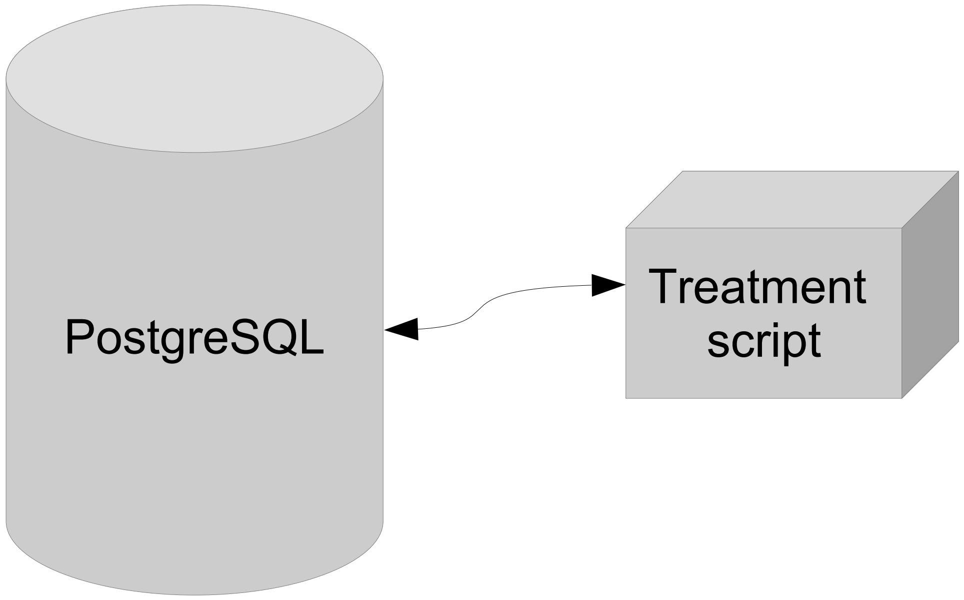
- Null Surface → Empty
- Single Point Line → infitesimal ending point translation

#2 Cleaning Data: Reclassify

```
SELECT    id, input,
          CASE WHEN input = 'yes' THEN 1::boolean
                    WHEN input = 'no'  THEN 0::boolean
                    ELSE NULL
          END reclass
FROM data;
```

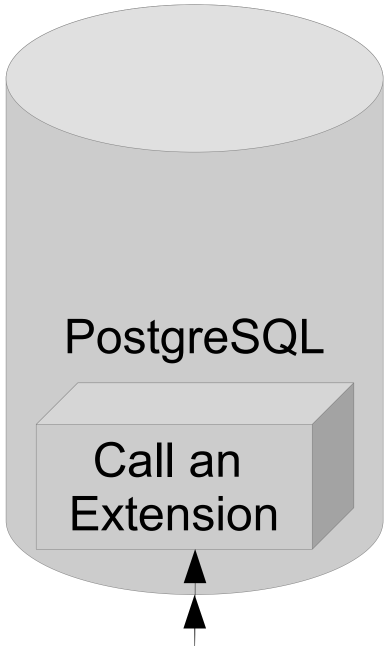
```
 id | input | reclass
----+---------+---------
  1 | yes    | t
  2 | no      | f
  3 | NC     |
```

#3 Data Processing



Common answer is :
« Develop an external script »

#3 Data Processing



Alternate answer is :
«Hey it's already there !»

Since PostgreSQL 9.1 : EXTENSION handling

Using existing extension is that easy, UUID generation example :

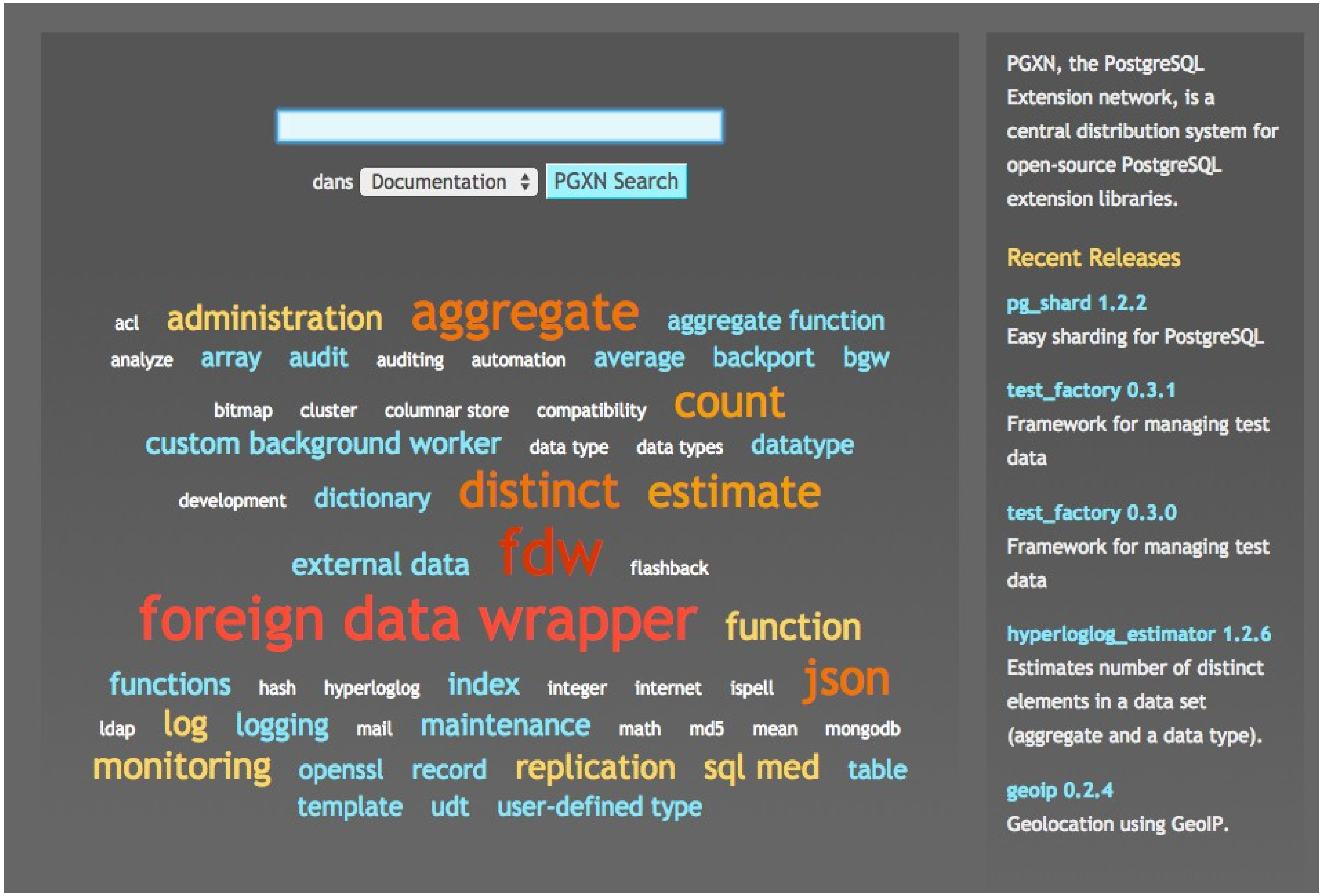
```
foo=# CREATE EXTENSION "uuid-ossp";
CREATE EXTENSION

foo=# SELECT uuid_generate_v4();

6953879c-3aae-4d42-a470-6d430305e173
```

Lot of PostgreSQL extensions available (really)

To display those already available on your server :

```
SELECT * FROM pg_available_extensions ;
```

An PostgreSQL extension repository:
http://pgxn.org/


dans Documentation
PGXN Search
acl administration aggregate aggregate function
analyze array audit auditing automation average backport bgw
bitmap cluster columnar store compatibility count
custom background worker data type data types datatype
development dictionary distinct estimate
external data fdw flashback
foreign data wrapper function
functions hash hyperloglog index integer internet ispell json
ldap log logging mail maintenance math md5 mean mongodb
monitoring openssl record replication sql med table
template udt user-defined type

PGXN, the PostgreSQL Extension network, is a central distribution system for open-source PostgreSQL extension libraries.

Recent Releases

pg_shard 1.2.2
Easy sharding for PostgreSQL

test_factory 0.3.1
Framework for managing test data

test_factory 0.3.0
Framework for managing test data

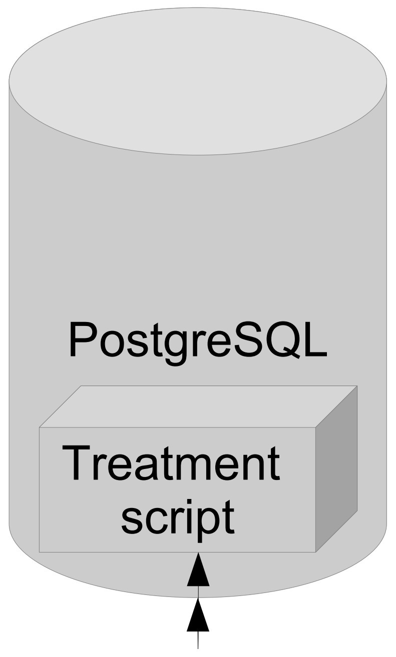
hyperloglog_estimator 1.2.6
Estimates number of distinct elements in a data set (aggregate and a data type).

geoip 0.2.4
Geolocation using GeoIP.

Some useful PostgreSQL extensions (among others)

- pg_trgm
  Use trigram matching to evaluate string similarity (for natural language texts search)
- Fuzzystrmatch
  Alternates well known string similarity functions (levenshtein, soundex...)
- Unnacent
  Deal with accentuated text
- xml2
  Xpath functions facilities (use libxml2)
- Pgcrypto
  Cryptographic functions
- Hstore
  Storing and manipulation of key/value pairs inside a single PostgreSQL value

#3 Data Processing



Alternate answer is :
«Put your scripts inside PostgreSQL»

#3 Data Processing : PL/Python

Using existing Python Library from PostgreSQL
Throught SQL function

#3 Data Processing : PL/Python

Using existing Python Library from PostgreSQL
Call throught SQL function

An example with GeoPy, Installation :

```
sudo apt-get install postgresql-plpython-9.4 python3-geopy

createdb db
createlang plpython3u db
psql db -c "CREATE EXTENSION postgis"
```

Register on GeoNames
Enable your account to use the free WebService

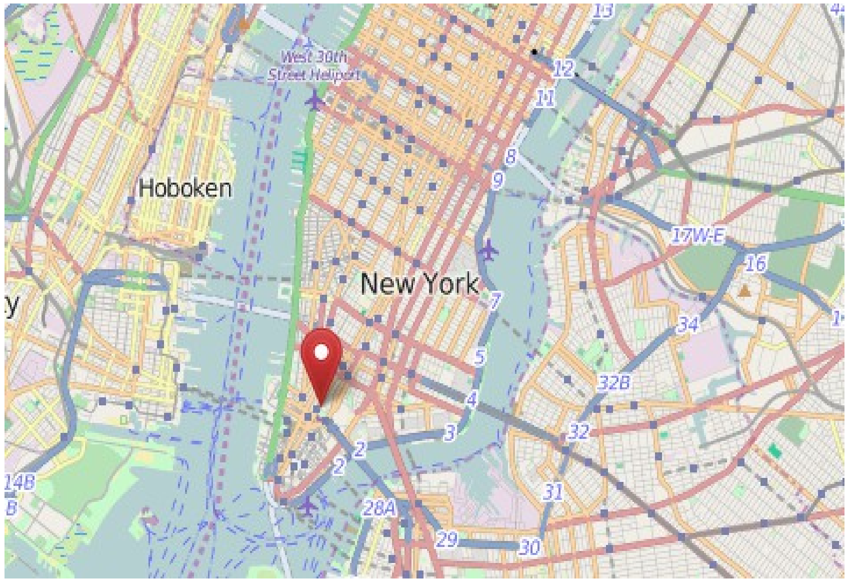
Pl/Python basic Geocoder function

```
CREATE OR REPLACE FUNCTION geoname(toponym text)
                   RETURNS geometry(Point,4326)
AS $$

    from geopy import geocoders
    g = geocoders.GeoNames(username="YOUR_USERNAME")

    try:
        place, (lat, lng) = g.geocode(toponym)

        result = plpy.execute(
        "SELECT 'SRID=4326;POINT(%s %s)'::geometry(Point, 4326) AS geom"
        % (lng, lat), 1)

        return result[0]["geom"]

    except:
        plpy.warning('Geocoding Error')
        return None


$$ LANGUAGE plpython3u;
```

Check it :

```
psql db -c
"SELECT ST_AsGeoJSON(geoname('New York, NY 10022'))"

{"type":"Point","coordinates":[-74.00597,40.71427]}
```

http://www.openstreetmap.org/?mlon=-74.00597&mlat=40.71427&zoom=12



#3 Data Processing : GeoSpatial statistic correlation

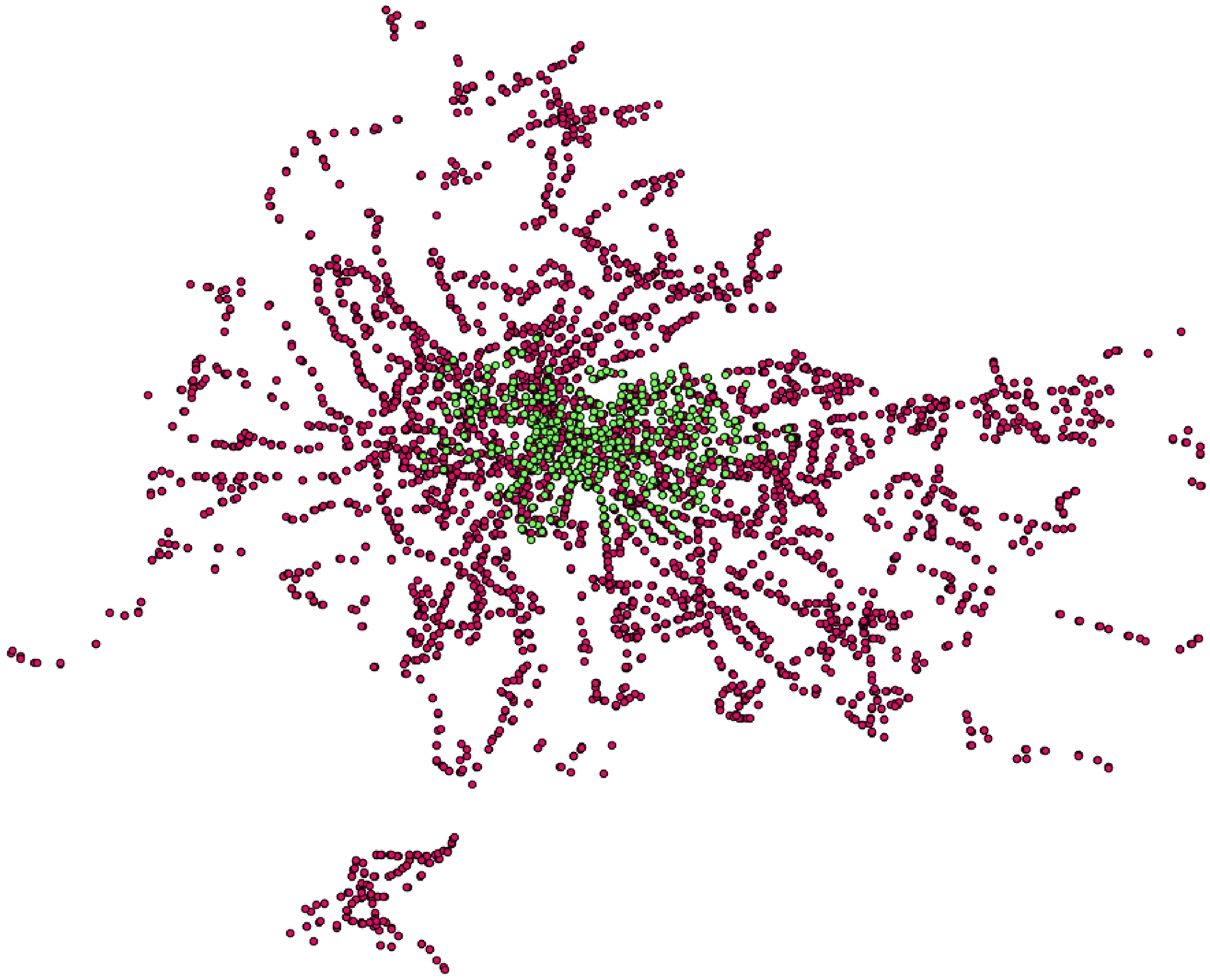
```
WITH lyon AS (SELECT ST_Transform(geoname('Lyon'), 2154) geom),

     tc AS (SELECT DISTINCT ON (tc.gid) tc.gid, v.gid, tc.geom,
                   ST_Distance(tc.geom, v.geom) dist

            FROM data.tc tc, data.velov v

            ORDER BY tc.gid, tc.geom <-> v.geom)


SELECT corr(dist, ST_Distance(l.geom, tc.geom)) FROM lyon l, tc;
```

#3 Data Processing : GeoSpatial statistic correlation

```
WITH lyon AS (SELECT ST_Transform(geoname('Lyon'), 2154) geom),

     tc AS (SELECT DISTINCT ON (tc.gid) tc.gid, v.gid, tc.geom,
                   ST_Distance(tc.geom, v.geom) dist

            FROM data.tc tc, data.velov v

            ORDER BY tc.gid, tc.geom <-> v.geom)


SELECT corr(dist, ST_Distance(l.geom, tc.geom)) FROM lyon l, tc;
```

```
      corr
-------------------
 0.951847587972223
```

#3 Data Processing : GeoSpatial statistic correlation



#3 Data Processing : Same, but via PL/R

```
createlang plr  DATABASE
```

```
CREATE OR REPLACE FUNCTION r_corr(a float[], b float[])
  RETURNS float AS
$$
  return (cor(a, b))
$$ language plr;
```

```
WITH lyon AS (SELECT ST_Transform(geoname('Lyon'), 2154) geom),
     tc AS (SELECT DISTINCT ON (tc.gid) tc.gid,
                   ST_Distance(tc.geom, v.geom) dist, tc.geom
            FROM data.tc tc, data.velov v ORDER BY tc.gid, tc.geom <-> v.geom)



SELECT r_corr(ARRAY(SELECT dist
                    FROM lyon l, tc ORDER BY tc.gid),

              ARRAY(SELECT(ST_Distance(l.geom, tc.geom))
                    FROM lyon l, tc ORDER BY tc.gid)
           );
```

```
      r_corr
-------------------
 0.951847587972213
```

#To Go Further

Write his own PostgreSQL module in C (if needed)

Write your own FDW (with Multicorn or in C)

Use Pl/R to use R advanced statistics

Machine Learning

#Conclusion

PostgreSQL behaves like an extensible and integrated Framework

Allow you to keep all data in the same place

(modern) SQL acting as a glue language

OSLANDIA

www.oslandia.com

Thanks !