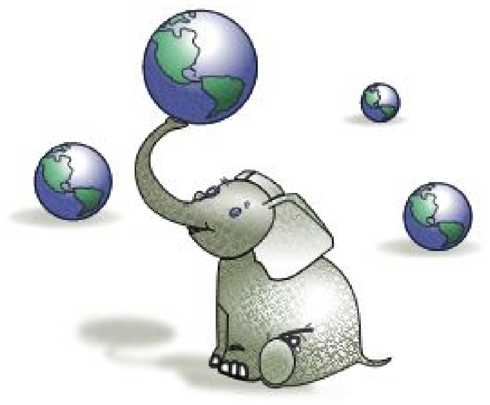
Introduction to Spatial Data Management with Postgis

PostGIS

Geographic Objects for PostgreSQL



A Spatial
Database
for the
Rest of Us

Content

Abstract: What is PostGIS?

- PostGIS is a spatial language extension module to the PostgreSQL backend server
- The OGC WKT (Well-Known Text) and WKB (Well-Known Binary) form define type and coordinates of an object
- Data from a PostgreSQL/PostGIS database can be used as data source for spatial server software like MapServer and GeoServer
- PostGIS is licensed under the GNU GPL and operated as a Free Software Project
- PostGIS is developed by Refractions Research Inc, a GIS and database consulting company in Victoria, British Columbia, Canada
- http://postgis.refractions.net

Who did it? Credits to...

Sandro Santilli (strk@refractions.net) coordinates all bug fixing and maintainance effort, integration of new GEOS functionality, and new function enhancements.

Chris Hodgson (chodgson@refractions.net) Maintains new functions and the 7.2 index bindings.

Paul Ramsey (pramsey@refractions.net) Maintains the JDBC objects and keeps track of the documentation and packaging.

Jeff Lounsbury (jeffloun@refractions.net) Original development of the Shape file loader/dumper.

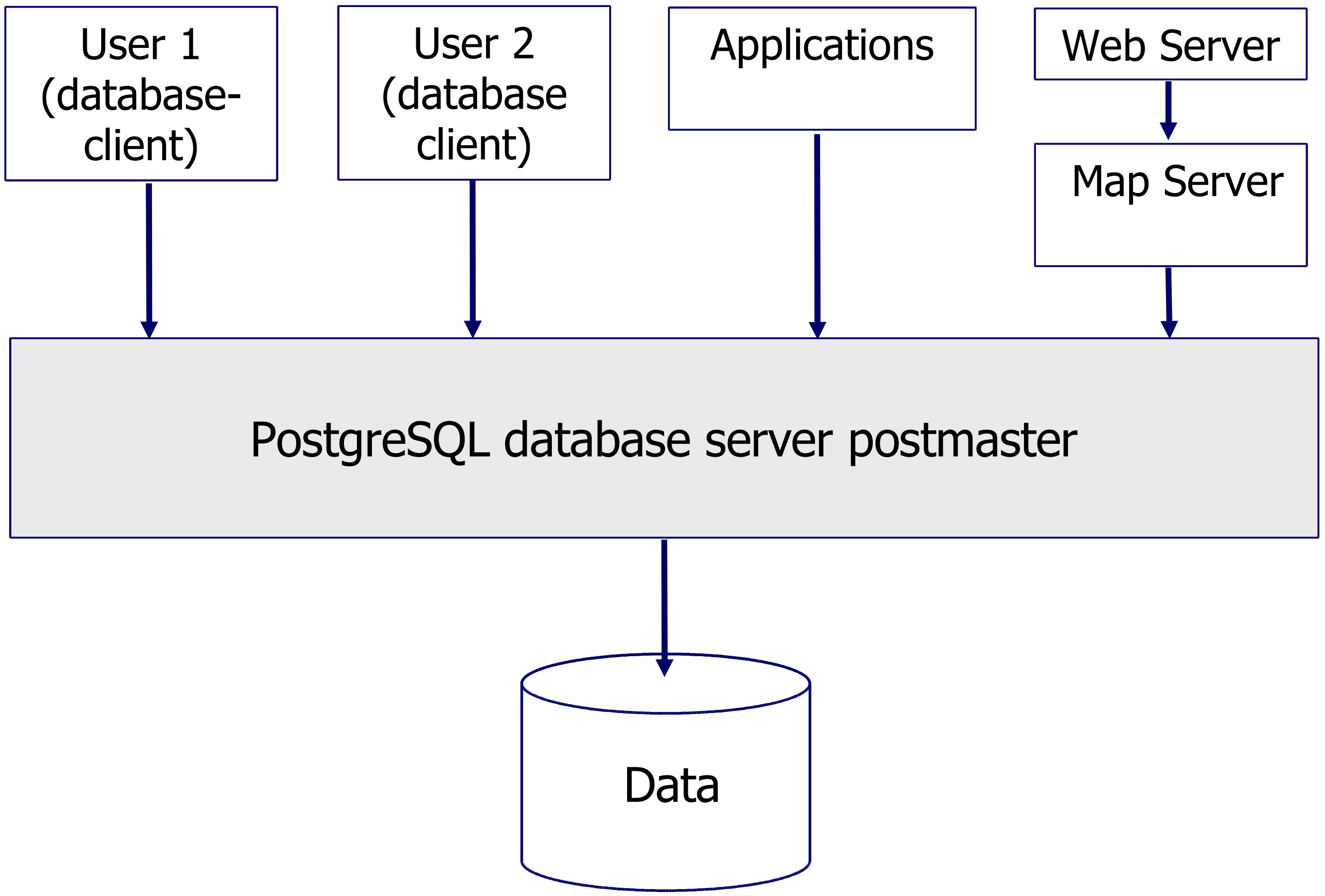
Dave Blasby (dblasby@gmail.com) The original developer of PostGIS. Dave wrote the server side objects, index bindings, and many of the server side analytical functions.

Other contributors in alphabetical order:

Alex Bodnaru, Bernhard Reiter, Bruno Wolff III, Carl Anderson, David Skea, David Techer, IIDA Tetsushi, Geographic Data BC, Gerald Fenoy, Gino Lucrezi, Klaus Foerster, Kris Jurka, Mark Cave-Ayland, Mark Sondheim, Markus Schaber, Nikita Shulga, Norman Vine, Olivier Courtin, Ralph Mason, Steffen Macke.

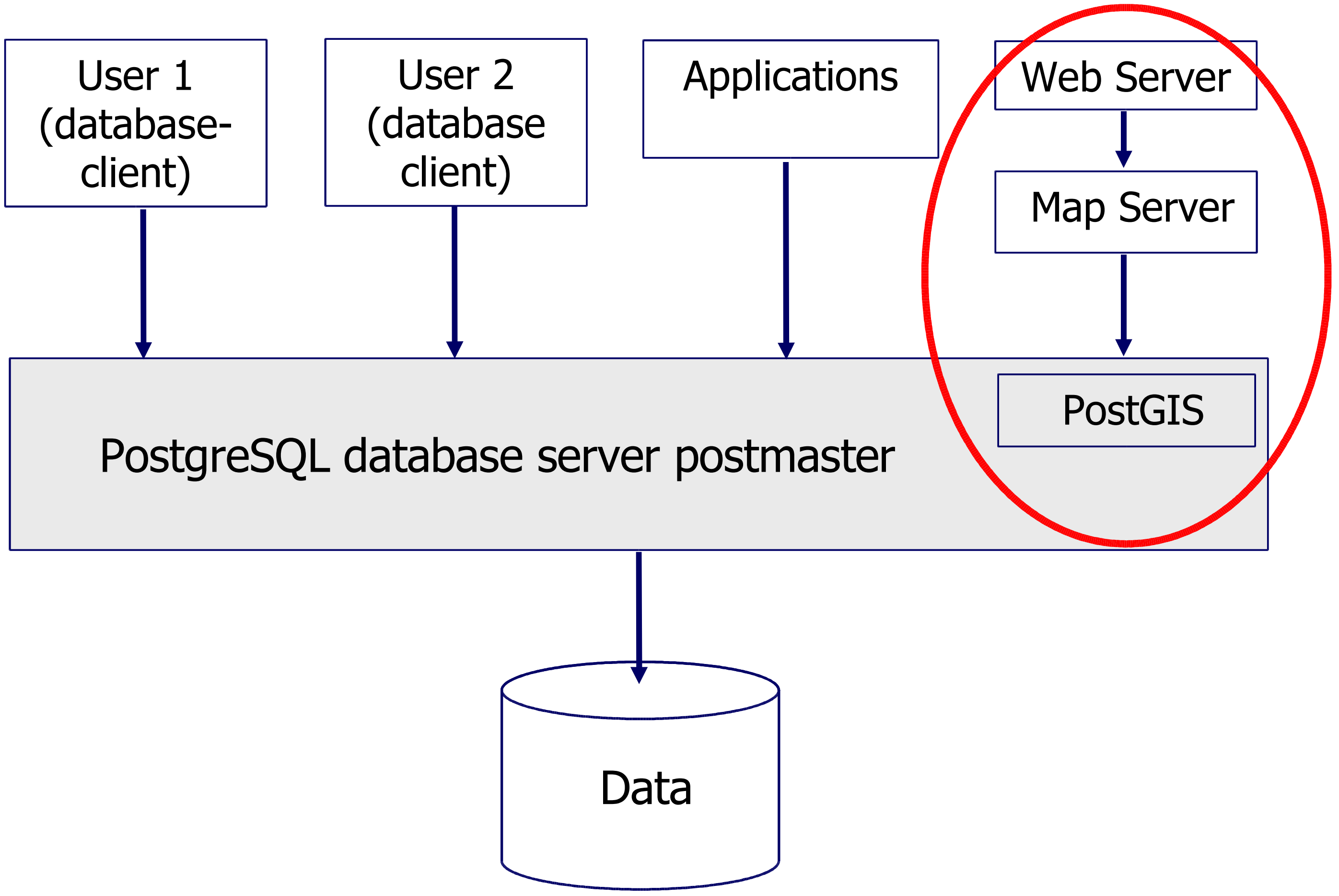
PostgreSQL Architecture with Client Applications

PostgreSQL is implemented as a client server system



Foucs of Interest for this Presentation

PostGIS spatial language extension module to the backend server



Overview: PostGIS Spatial Data Management

- PostGIS is implemented compliant to the OGC Simple Feature Specifications for SQL standard
- The OGC specification defines operations and the SQL schema to insert, query, manipulate and delete spatial objects
- The coordinates of the spatial objects are stored in Feature Tables
- One Feature Table can contain one type of geometry (point, line, polygon, multiple of each and geometry collection)
- The coordinates of each object is stored in a field of a special type
- The field type for a set of coordinates is WKT (Well Known Text)
- Meta data is collected for each Feature Table to organize the type and the coordinate system of the contained geometry
- The meta data for each Feature Table is stored in the special table geometry_columns

Spatial Data Type WKT as defined by the OGC

WKT Examples:

```
POINT(2572292.2 5631150.7)


LINESTRING (2566006.4 5633207.9,  2566028.6 5633215.1, 2566062.3 5633227.1)


MULTILINESTRING((2566006.4 5633207.9,  2566028.6 5633215.1),  (2566062.3
5633227.1,  2566083 5633234.8))


POLYGON (2568262.1 5635344.1,  2568298.5 5635387.6, 2568261.04 5635276.15,
2568262.1 5635344.1);


MULTIPOLYGON(((2568262.1 5635344.1,  2568298.5 5635387.6, 2568261.04 5635276.15,
 2568262.1 5635344.1), (2568194.2 5635136.4,  2568199.6 5635264.2, 2568200.8
5635134.7,  2568194.2 5635136.4 )))
```

PostGIS Interfaces to GIS Data

- PostGIS ships with a Shape file loader an dumper
- Various file types (Shape, MapInfo, DGN, GML, ...) can be read, converted and inserted to a PostGIS database using the OGC libraries
- A PostGIS Feature Table can be used as data source for a growing variety of map and feature server software like UMN MapServer, GeoServer, uDGI, deegree , JUMP, etc...
- The data can be accessed using standard ODGB or JDBC connectors
- Several projects have evolved around PostGIS transforming and inserting geometries from highly specialized formats like SICAD C60, EDBS, DXF, WLDGE and many more

Creating a PostGIS Database

Create database:

```
createdb <dbname>
```

Load PL/pgsql language for PostGIS:

```
createlang plpgsql <dbname>
```

Load PostGIS and object definitions:

```
psql -d <dbname> -f postgis.sql
```

This file also contains the CreateTable SQL for the metadata table geometry_columns

CreateTable spatial_ref_sys (coordinate system codes):

```
psql -d <dbname> -f spatial_ref_sys.sql
```

Insert Coordinates using SQL

SQL with PostGIS Function – Example:

```
create table user_locations (gid int4, user_name varchar);

select AddGeometryColumn
('db_mapbender','user_locations','the_geom','4326','POINT',2);

insert into user_locations values ('1','Morissette',GeometryFromText
('POINT(-71.060316 48.432044)', 4326));

insert into user_locations values ('2', 'Sperb',GeometryFromText
('POINT(-48.6764 -26.8916)', 4326));

...
```

`AddGeometryColumn()`

this function adds the meta information of this field to the table geometry_columns

`DropGeometryColumn()`

removes the meta information from the table geometry_columns

Representation of the Data in the Table

- Meta information in geometry_columns

```
f_table_catalog    | spatial
f_table_schema     | db_mapbender
f_table_name       | user_locations
f_geometry_column  | the_geom
coord_dimension    | 2
srid               | 4326
type               | POINT
attrelid           | 8751803
varattnum          | 11
stats              |
```

- Data from Feature Table <user_locations>

```
gid       | 1
user_name | Sperb
the_geom  | SRID=4326;POINT(-48.6764 -26.8916)
```

Loading Flat File GIS Data

PostGIS Shape Loader

- Create SQL input files:

```
shp2pgsql -s <epsgcode> <ShapeFileName> <TableName> <dbName> > <filename>
```

Make sure that the EPSG code of the data you load is correct! If you do not know which coordinate system the data comes in you cannot combine it with any other spatial data!

- Load input file:

```
psql -d <dbname> -f <filename>
```

Convert Non-Spatial X and Y fields to PostGIS

First select both X and Y coordinates into one field of type character `objgeom = `"-48.6764 -26.8916". Then call the function GeometryFromText, add the geometry type, brackets and append the EPSG code of the coordinates.

```
UPDATE test SET the_geom =  GeometryFromText(
'POINT '(|| objgeom || ')"', 31467)
```

Beware of brackets andd quotes, the code might look slightly nauseating. If you add brackets to the coordinate strings `objgeom = `"**(**-48.6764 -26.8916**)**" then you do not have to add them in the INSERT string

```
UPDATE test SET the_geom =  GeometryFromText(
'POINT '|| objgeom || '"', 31467)
```

Converting binary to WKT

Since PostGIS version 1.0 the internal storage of coordinates in the geometry column has changed to the binary format WKB (Well Known Binary). It is not readable but provides better performance.

You can convert the binary WKB form with SQL into a readable WKT string using the PostGIS function <asewkt> (as extended WKT).

The PostGIS extended WKT carries the coordinate system additionally to the basic WKT string syntax. That will effectively prevent anybody from ever forgetting the coordinate system ever again.

```
Select asewkt(<geometrycolumn>) from <table name>
```

```
Select user_name, asewkt(the_geom)from user_locations
```

```
gid       | 1
user_name | Sperb
the_geom  | SRID=4326;POINT(-48.6764 -26.8916)
```

Spatial Operators in PostGIS

- Query and manipulation of data using the SQL interface
- Implementation of these spatial operators was the original domain of every GIS software.

Example: Query for the minimum bounding box rectangle

```
geodata2=# SELECT EXTENT(the_geom) FROM tbl_spatial;
                   extent
--------------------------------------------
 BOX3D(-48.57 -26.89 0, -47.64 -25.16 0)
(1 row)
```

- PostGIS ships with all functions and operators required to comply to the OGC SFS (Simple Feature Specification for SQL) specification
- The Implementation of all relevant functionality has been done in the JTS (Java Topology Suite) which has been ported to GEOS and is used by most Open Source tools

Example: touches(), intersects(), overlaps(), buffer()...

Spatial Index

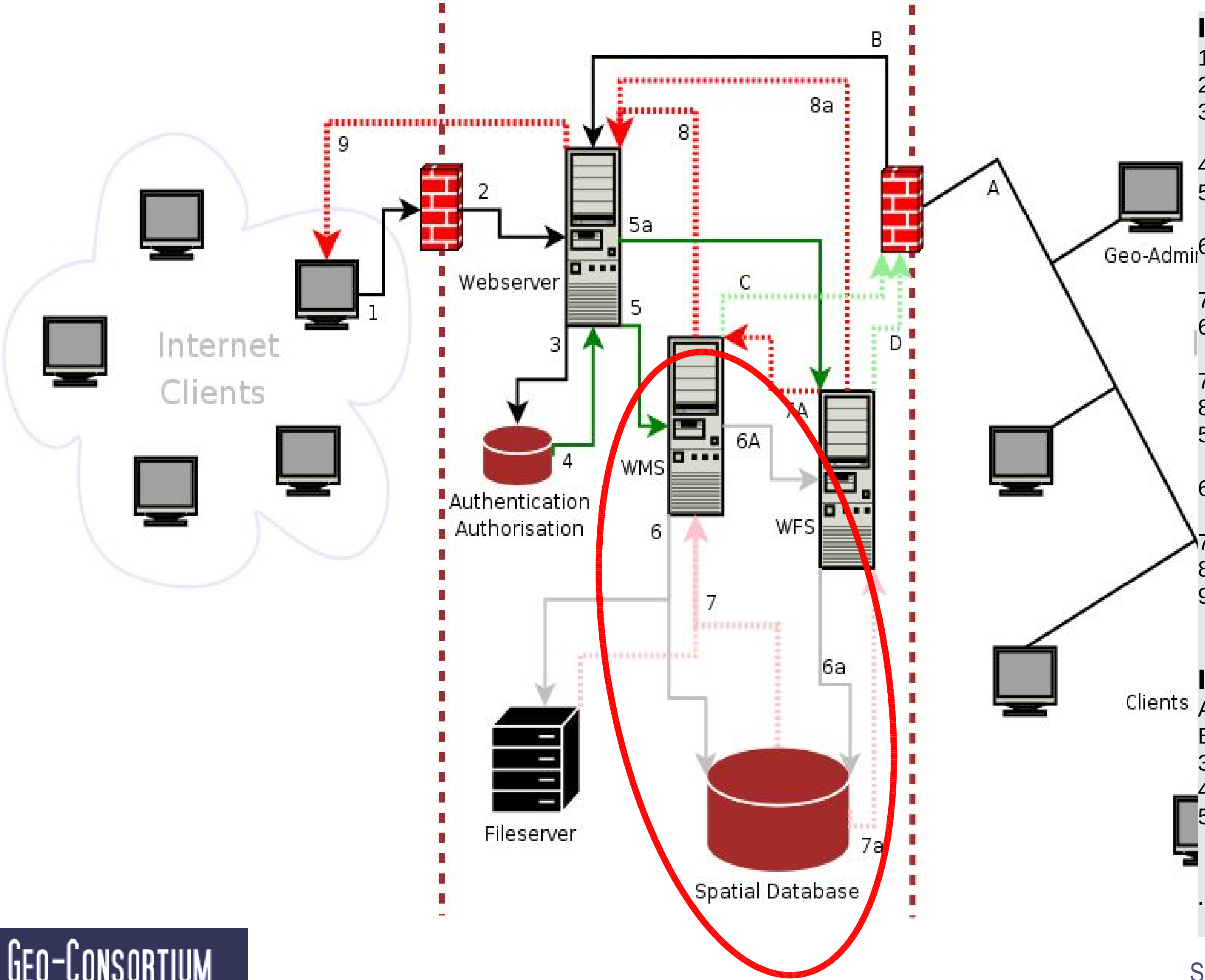
Spatial data has a slightly different need for indexing than common relational data. The efficiency of an index also greatly varies according to the kind of data that is being stored. Polygons with several thousand vertices that extend throughout the whole area can have devastating effects on performance because they appear in every section of interest. In those cases it might make sense to clip the geometry into several pieces.

- PostGIS implements the GiST-Index (Generalized Search Tree)
- The index is created using standard SQL

```
    CREATE INDEX <indexname>
    ON <tablename>
    USING GIST ( <geometrycolumn> GIST_GEOMETRY_OPS );
```

It is necessary to vacuum-analyze the database before the index takes effect. Frequent read/write operations might slow performance.

Web Mapping a Spatial Data Infrastructure



Internet
1 Client request map
2 Firewall admits request
3 Webserver authenticates user
4 Authorisation of user
5 Standard WMS getMap request
6 Spatial data query (SFS SQL)
7 Native data
6A OGC WFS getFeature request
7A ...returns GML
8 ... returns map image
5a OGC WFS getFeature request
6a Spatial data query (SFS SQL)
7a ...returns WKT
8a ...returns GML
9 GML and iamge are returned to the client

Intranet
A OWS requests
B Webserver,
3 Authentication
4 Authorisation
5, 6, 7, C, A Standard WMS request 5a, 6a, 7a, D, A WFS request
...

PostGIS tables as datasource for UMN MapServer

Use a version of UMN MapServer that supports PostGIS. Enter the connection type and credentials (you might want to add the map server engine as a trusted client in the pg_hba.conf)

```
LAYER

...

    CONNECTIONTYPE postgis

    CONNECTION "user=<username> dbname=<database> host=<server>
    port=5432 password=<password>"

    DATA "<geometry_column> from <table>"

...

END
```

Example of a most simplistic SQL selection:

```
DATA "the_geom from nutzung"
```

Using Views as Data Source for UMN MapServer

If you need more complex queries, create a view.

Create view:

```
CREATE VIEW qry_users AS  SELECT *  FROM users WHERE user_type
LIKE 'professional' ORDER BY gid;
```

You can use the USING pseudo-SQL clause to add more complex queries. For views or subselects MapServer can neither determine the unique identifier for each row nor the SRID for the table because they are not identified in the meta data table <geometry_columns>.

DATA string in MAP configuration file:

```
DATA "the_geom from (select * from qry_ nutzung) as foo using
unique gid using SRID=31467"
```

Required information:

- Coordinate system as <SRID=number> (case sensitive)
- Field which contains unique identifier <using unique> (case sensitive)
- Alias for the table <as foo> (usage)

Using SQL Clauses in the MAP Configuration File

Queries can also be added to the MAP configuration file directly. In that case the corresponding SQL with joins accross tables has to be added to the DATA section.

Example:
Select street names from table <str_name> as Label for geometries from tbale <str>

```
DATA "the_geom from (SELECT landuse.object_id, landuse.area,
forest.type, forest.the_geom FROM landuse INNER JOIN forest ON
landuse.gid = forest.gid) as foo using unique gid using
SRID=4326"
```

Notice

Always add a field that conatins the unique ID and explicitly specify the coordinate reference system EPSG code as shown in the example. UMN MapServer needs this information to be able to convert geometries to the correct coordinate system and query for each object with unique ID.

MAP File Filter Parameters

The MAP configuration file allows to set a FILTER parameter to select features with defined properties. This is the "traditional" (non-standardized) way to show selections.

```
FILTER "[filteritem] = [Kriterium]"
```

Example: Selection of users with type = "professional"

```
DATA "the_geom from (select * from qry_users) as foo using
unique gid using SRID=4326"

FILTER "user_type > 'professional'"
```

Example: Selection of numbered roads from a network (second character is a number like in A42, B52,...)

```
DATA "the_geom from tbl_streets"

FILTER "(substr(street_name, (length(street_name)-1), 1))
BETWEEN 0 and 9"
```

Create PostGIS Schema

PostGIS enabled databases can be organized in schemas as any standard database.

Create schema:

```
z.B.: CREATE SCHEMA spatial_schema;
```

Create table in schema:

```
z.B.: CREATE TABLE spatial_schema.db_user_locations
(id int8, user_name varchar, user_type int4) WITH OIDS;
```

Standard SQL select clause:

```
SELECT * FROM spatial_schema.db_user_locations;
```

Load CSV File

Upload CSV files (comma separated value), for example if oyu need to include data from a DBF file (which contains the alphanumerical attributes of a flat GIS Shape file.

Create CVS file:

Example:

```
Load DBF file with OpenOffice.org and store it as CSV
```

Create table with corresponding fields:

Example:

```
CREATE TABLE mytable
(id int8, area float8, boart varchar) WITH OIDS;
```

Copy CVS file from hard disk

Example:

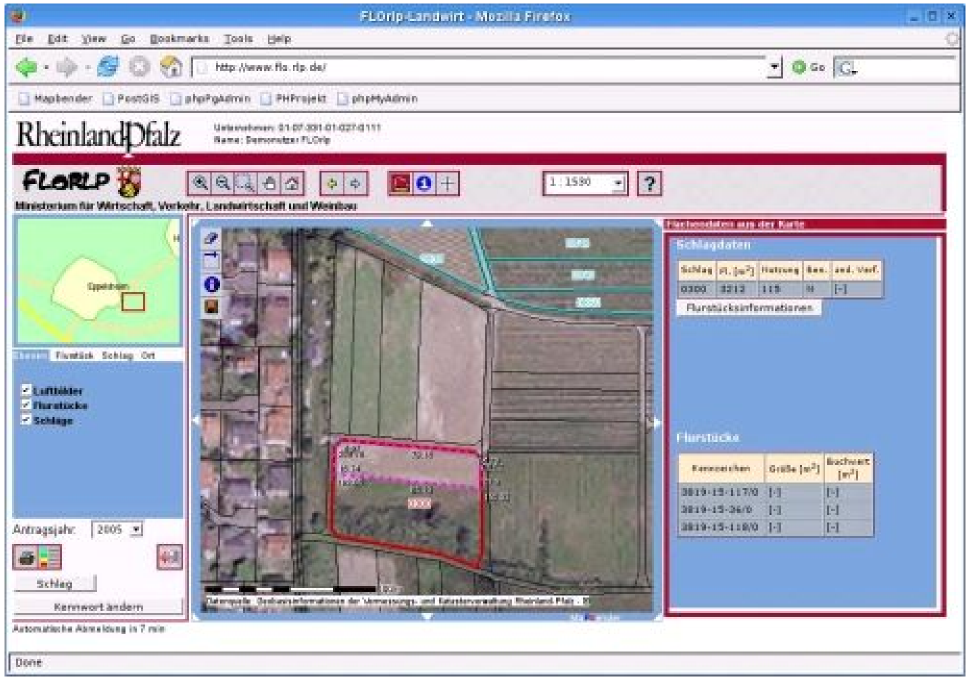
```
COPY mytable FROM '/data/postgis_schulung/poi.csv';
```

Spatial Enterprise System Online Presentation

Most Spatial Data Infrastructure operated at enterprise level that use Free and Open Source Software operate a PostgreSQL database with PostGIS extension. There exisits as yet no viable alternative and currently there seems to be no need to get one.

The system FLOrlp of the Ministry of Commerce, Infrastructure, Agriculture and Viticulture in Rhineland Palatinate, Germany allows farmers to use full GIS functionality to process their requests for EU subsidies.

http://www.flo.rlp.de

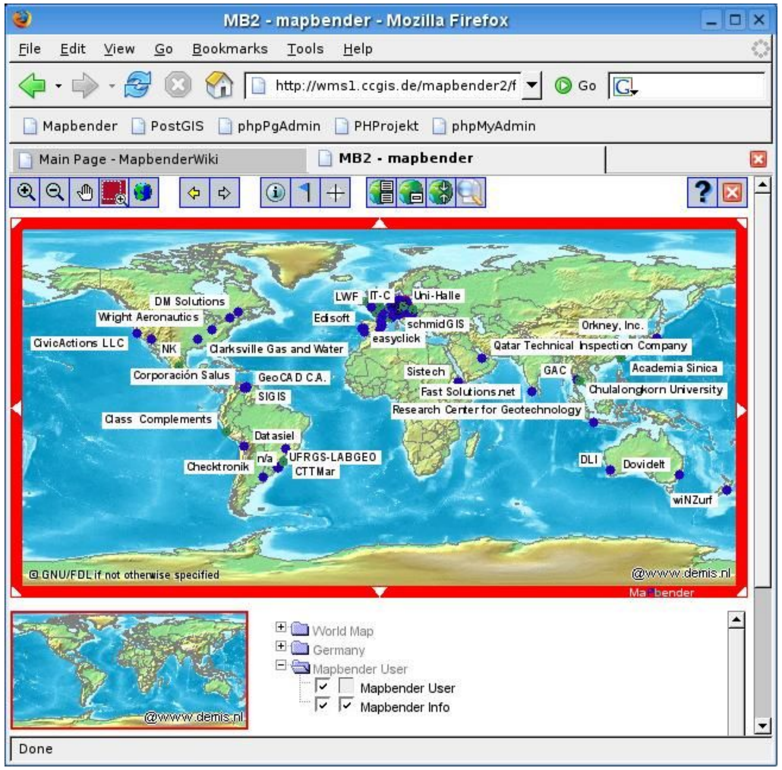

FLOrlp-Landwirt - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.flo.rlp.de/
Mapbender PostGIS phpPgAdmin PHProjekt phpMyAdmin
RheinlandPfalz
FLORLP
Ministerium für Wirtschaft, Verkehr, Landwirtschaft und Weinbau
1 : 1530
Luftbilder
Flurstücke
Schläge
Antragsjahr: 2005
Schlag
Kennwort ändern
Schlagdaten
Flurstücksinformationen
Flurstücke
Done

What can PostGIS do for You Today?

There is a wide field of application for Spatial Data. Recent publications by Google, Yahoo and related commercial providers has brought web mapping applications and satellite imagery to the broad public strongly increasing interest in this technology.

To operate a standardized spatial data infrastructure you need the database, a map server and a client interface. Additionally you might want to include a Web Feature Service that returns GML (Geographic Markup Language) to search, find and navigate to geographic features.

A "geographic feature" might be the position of a car (a dynamic point feature for tracking), that of the Grand Canyon (a polygon feature for sightseeing).



Be creative!

Further Information and Contact

The most important source of further information is the Internet. Check out the following adresses referencing everything that you might need.

- http://www.mapbender.org/ Mapbender (Web Mapping Suite Wiki)
- http://mapserver.gis.umn.edu/ UMN MapServer (Web Map Server)
- http://www.maptools.org/ Resources (Mapping Tools)
- http://freegis.org/ General Information about Free GIS
- http://www.eogeo.org/ Earth Observation and Geospatial Information

Feel free to contact me directly:

Arnulf Christl
arnulf.christl@ccgis.de

CCGIS Christl & Stamm GbR
Siemensstr. 8
53121 Bonn

Homepage: http://www.ccgis.de