Workshop

GRASS GIS and RDBMS

by Marco Ciolli, Paolo Zatelli and Clara Tattoni

Department of Civil and Environmental Engineering
University of Trento
Italy

UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria Civile
e Ambientale

GIS and DBMS

GISs can manage all the features of geographic information: geometry, topology and attributes.

Sometimes it is more efficient to manage some or all of these features with an external Data Base Management System (DBMS).

A DBMS can manage the attributes, a DBMS with spatial extension can manage the geometry as well.

It is possible to use different DBMS, databases and configurations for different maps in the same dataset or for different layers of the same map.



GRASS, QGIS and DBMS

GRASS and QGIS can read all formats supported by the OGR library, that is data from the more used DBMS

If using OGR Library with GRASS, data can be:

- imported (GRASS native format)
- linked as read-only external data with pseudotopology (*v.external*)

OGR Vector Formats

Format Name	Creation	Georeferencing
Arc/Info Binary Coverage	No	Yes
Comma Separated Value (.csv)	Yes	No
DODS/OPeNDAP	No	Yes
DWG	Yes	No
DXF	Yes	No
ESRI Personal GeoDatabase	No	Yes
ESRI ArcSDE	No	Yes
ESRI Shapefile	Yes	Yes
FMEObjects Gateway	No	Yes
GML	Yes	No
GRASS	No	Yes
INTERLIS	No	Yes
KML	Yes	No
Mapinfo File	Yes	Yes
Microstation DGN	No	No
MySQL	No	No
OGDI Vectors	No	Yes
ODBC	No	Yes
Oracle Spatial	Yes	Yes
PostgreSQL	Yes	Yes
S-57 (ENC)	No	Yes
SDTS	No	Yes
SQLite	Yes	No
UK .NTF	No	Yes
U.S. Census TIGER/Line	No	Yes
VRT - Virtual Datasource	No	Yes

QGIS and DBMS

QGIS can access directly PostgreSQL/PostGIS data both attributes and geometry.

In order to use a PostgreSQL/PostGIS layer in QGIS, it must be present a key column of *int4.*

This layer processing speed can be enhanced by indexing that column (if it is a PostgreSQL *primary key* it is indexed by default*).*

If a view is used, it must contain a type *int4* column or a *primary key,* possibly indexed*.*


QGIS and PostgreSQL/PostGIS

The advantage of using PostgreSQL/PostGIS relies in the possibilities of spatial indexing and of features selection on a map both by spatial and attribute queries.

In order to use a PostgreSQL/PostGIS layer in QGIS it is necessary to:

- define a connection to the PostgreSQL/PostGIS database;
- connect to the database;
- select a layer to add to the map in use;

A SQL query can be run to select which features to load from a layer and a PostGIS layer can be edited (but not created).

The QGIS SPIT plug-in (Shapefile to PostGIS Import Tool) can import shapefiles into PostGIS.

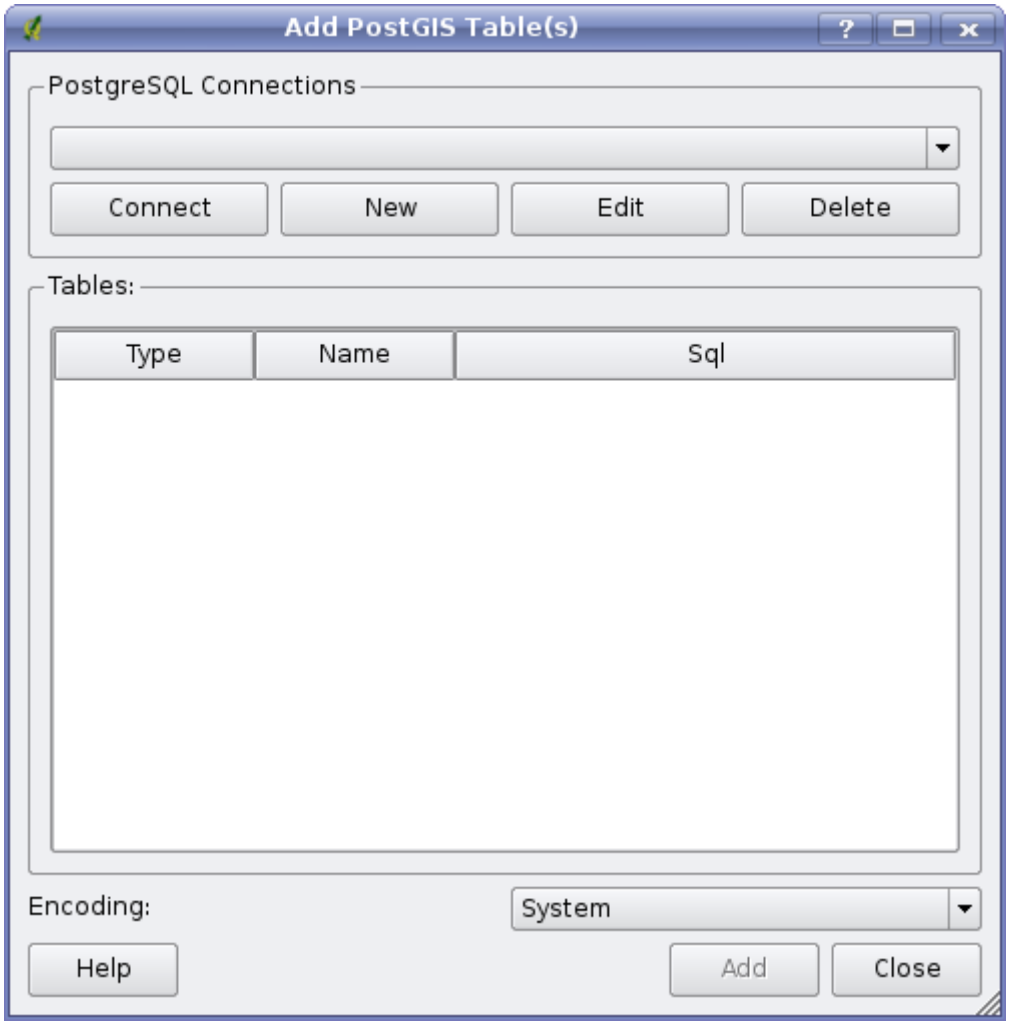
QGIS and PostgreSQL/PostGIS

Defining a connection to PostgreSQL:

- select
  Layer->
  Add PostGIS Layer

OR

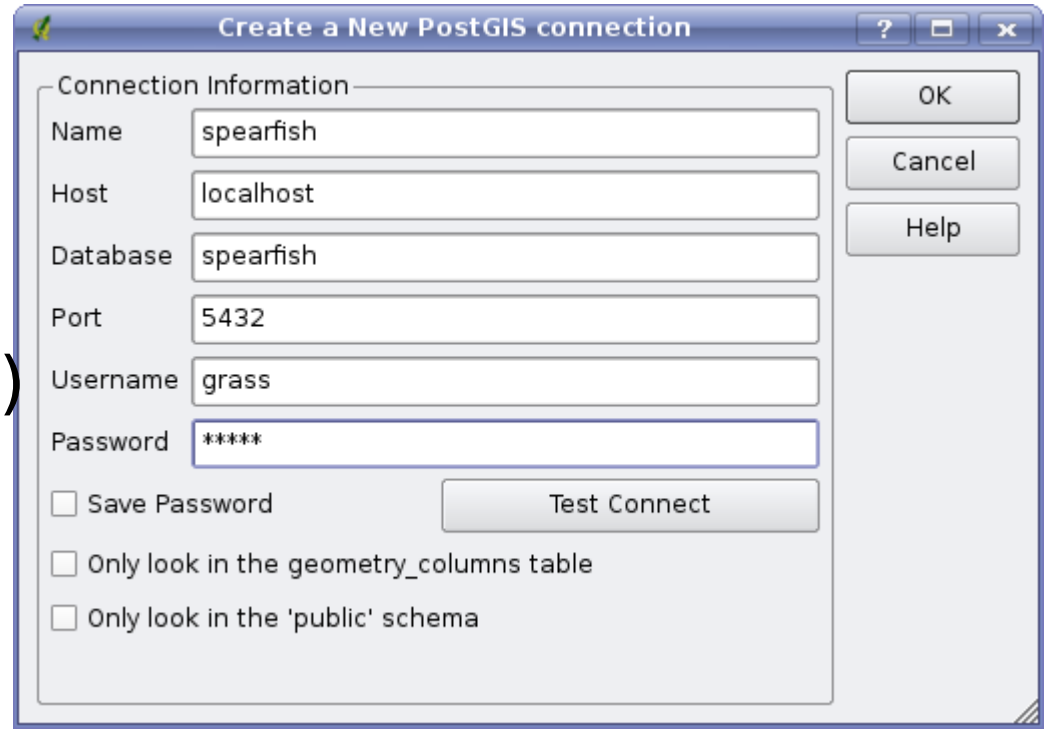
- Click the icon 

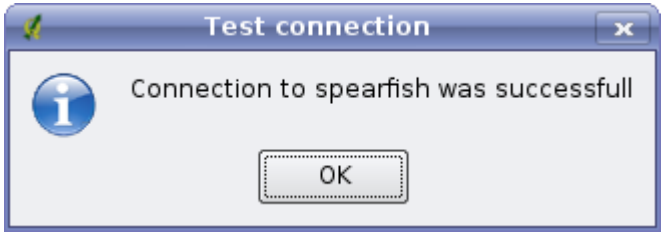


QGIS and PostgreSQL/PostGIS

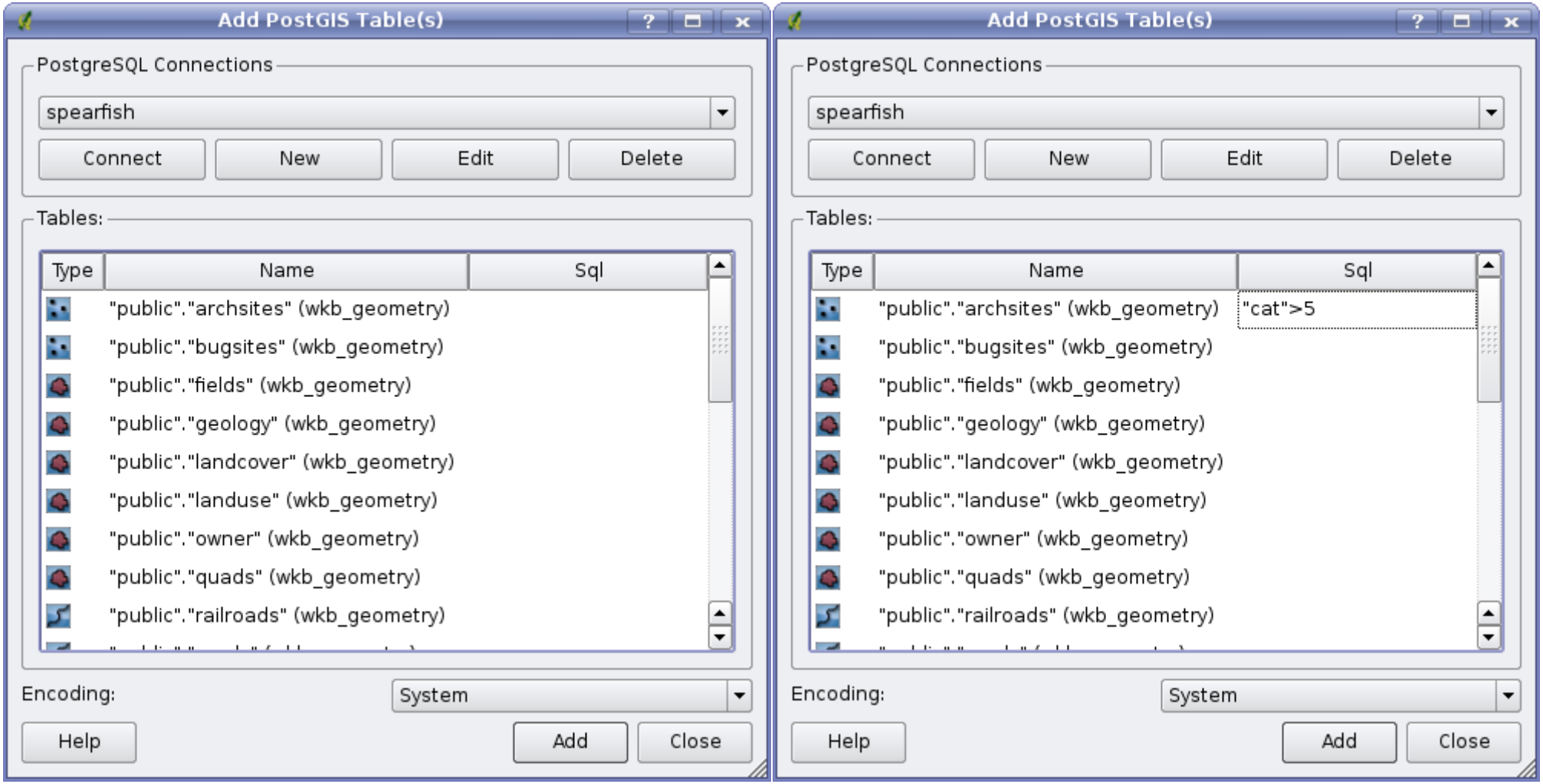
Connection settings

- Name: spearfish
- Host: localhost
- Database: spearpg
- Port: 5432 (default)
- User: grass
- password: grass





QGIS and PostgreSQL/PostGIS



GRASS and DBMS

Default driver for GRASS is DBF:

- SQL support is limited, only a few types and functions are available, constraints also on concurrent use, integrity, etc...
- The database is the directory where .dbf files are stored, usually $GISDBASE/$LOCATION_NAME/$MAPSET/dbf/ for example /home/ubuntu/grass/spearfish60/user1/dbf/
- Each table corresponds to one DBF file;
- Column names are limited to 10 characters (DBF API definition).

GRASS and DBMS

GRASS also supports the following DBMS

- SQLite
- PostgreSQL
- MySQL
- ODBC

Spatial enabled DBMS

- PostGIS

GRASS and DBMS

In order to use GRASS with these DBMS, GRASS must be compiled with the appropriate support.

GRASS module *db.drivers* lists the DBMS drivers supported by the current GRASS version.
db.drivers -f from the command line.

Binary (package) installation usually provides:

- GRASS compiled with support for all DBMSs (and more..);
- Package manager handles the dependencies and install all the necessary libraries to interact with DBMS together with GRASS;
- DBMS have to be installed separately.

GRASS and DBMS

Compilation:

- But for DBF, GRASS must be configured with the support for the chosen DB, enabling the option with

  `--with_DBMSname` for ex. `--with-postgres`

- Both runtime and development packages must be installed. For example: *postgres* for using the DBMDS and *postgres-dev* for compling GRASS;

- If libraries and/or includes are located in non standard directories, full paths must be provided when configuring GRASS compilation. For ex.

```
--with-postgres-includes=/usr/include/postgresql/
```

GRASS and DBMS

Possible configuration options are:

--with-postgres-includes=DIRS PostgreSQL include files are in DIRS

--with-postgres-libs=DIRS PostgreSQL library files are in DIRS

--with-mysql-includes=DIRS MySQL include files are in DIRS

--with-mysql-libs=DIRS MySQL library files are in DIRS

--with-sqlite-includes=DIRS SQLite include files are in DIRS

--with-sqlite-libs=DIRS SQLite library files are in DIRS

--with-odbc-includes=DIRS ODBC include files are in DIRS

--with-odbc-libs=DIRS ODBC library files are in DIRS

GRASS and DBMS

Commands available in GRASS:

- *db.** to interact with the database and set default values for vector maps
  db.columns db.copy db.drivers db.login db.tables db.connect db.describe db.execute db.select db.test
- *v.db.** to set values for a single map
  v.db.addcol v.db.droptable v.db.update v.db.addtable v.db.reconnect.all v.db.connect v.db.select
- Values set with db.* command are saved as defaults until the next edit, ex. *db.connect* sets the default database. Values for single maps can be different from the default value, ex. with *v.db.connect* a different DB can be specified for a given map.



GRASS and SQLite

SQLite does not need any further external configuration. The DB is a single file that can have any name and can be saved in any directory.

Use in GRASS:

- Set the default database

```
db.connect database=/tmp/spearfish.db driver=sqlite
```

- A new database will be created automatically by GRASS when inserting first table.
  The new DB will be created in current directory if none provided.
- Show the default database connections

```
db.connect -p
```

- All newly created vector map will use the above database for attribute storage



GRASS and SQLite

For example in GRASS, location spearfish, mapset user1;

- Copy file roads from PERMANENT to file roads2 in current mapset (files from different mapsets cannot be edited)

```
g.copy vect=roads,roads2
```

- Copy the table from roads2 to SQLite-db using db.copy

```
db.copy from_driver=dbf from_table=roads2 \
from_database=$GISDBASE/$LOCATION_NAME/$MAPSET/dbf/ \
to_driver=sqlite to_table=roads2 \
to_database=$HOME/spearfish.db
```

- Connect vector map to SQLite table

```
v.db.connect map=roads2 driver=sqlite \
database=spearfish.db table=roads2 key=cat layer=1 -o
```

- Show connection details `v.db.connect -p roads2`



GRASS and ODBC

- ODBC (unixodbc) must be installed, together with the necessary drivers (for example odbc-postgresql) and GRASS support enabled, then drivers and connection (DSN Data Source Name) have to be configured.
- Set-up can be done editing the file /etc/odbc.ini, (drivers and DSN) or $HOME/odbc.ini (DSN only). Usually the latter settings have priority over the first ones.
- Interfaces exists for editing the files, such as ODBCConfig.



GRASS and ODBC

ODBC-driver parameters are configured in the file /etc/odbc.ini:

```
[PostgreSQL]
Description     = ODBC for PostgreSQL
Driver          = /usr/lib/odbc/psqlodbc.so
Setup           = /usr/lib/odbc/libodbcpsqlS.so
FileUsage       = 1


[MySQL]

Description     = MySQL driver
Driver          = /usr/lib/odbc/libmyodbc.so
Setup           = /usr/lib/odbc/libodbcmyS.so
```



GRASS and ODBC

DSN connection parameters are set in $HOME/odbc.ini or /etc/odbc.ini

```
[spearfish]
Description     = PostgreSQL for ODBC
Driver          = PostgreSQL  # as in odbcinst.ini
Trace           = No
TraceFile       =
Database        = spearfishodbc
Servername      = localhost
UserName        = grass
Password        =
Port            = 5432
Protocol        = 8.1
ReadOnly        = No
RowVersioning   = No
ShowSystemTables = No
ShowOidColumn   = No
FakeOidIndex    = No
ConnSettings    =
```



GRASS and ODBC

- Create a new database *spearfishodbc* in PostgreSQL

```
createdb -U grass spearfishodbc
```

- Connection

```
db.connect driver=odbc database=spearfishodbc
```

- Login

```
db.login user=grass pass=grass
```

- Show current connection

```
v.db.connect -p roads2
```

All tables are stored in *spearfishodbc* in PostgreSQL database through ODBC.

GRASS and MySQL

- Mysql-database and user “grass” must be created

```
sudo mysql -u root
```

```
mysql> CREATE DATABASE spearfish;
```

```
mysql> GRANT ALL ON spearfish.* to grass identified by
'grass';
```

Use in GRASS

- Set the default database in GRASS

```
db.connect driver=mysql
database=“host=myhost.com,dbname=spearfish“
```

- Set username and password (stored in cleartext in $HOME)

```
db.login user=grass password=secret driver=mysql\
database=“host=myhost.com,dbname=spearfish“
```

- Show connection details and tables

```
db.connect -p ; db.tables -p
```

GRASS and MySQL

- Copy the table roads2 from DBF to MySQL

```
db.copy from_driver=dbf from_table=roads2 \
from_database=$GISDBASE/$LOCATION_NAME/$MAPSET/dbf/ \
to_driver=mysql to_table=roads2 \
to_database="host="myhost.com,dbname=spearfish"
```

- Connect vector map to MySQL

```
v.db.connect -o map=roads2 driver=mysql \
database="host=myhost.com,dbname=spearfish"
table=roads2 key=cat layer=1
```

- Show connection details

```
v.db.connect -p roads2
```

GRASS and PostgreSQL

- Create Postgres user and database for use within GRASS

```
sudo createuser -d -U postgres grass

createdb -U grass spearpg
```

Use in GRASS

- Set the default database

```
db.connect driver=pg \
database="host=myhost.com,dbname=spearpg"
```

- Set username and password

```
db.login user=grass password=secret
```

- Show connection details

```
db.connect -p
```

GRASS and PostgreSQL

- Copy the table roads2 from DBF to PostgreSQL

```
db.copy from_driver=dbf from_table=roads2 \
from_database=$GISDBASE/$LOCATION_NAME/$MAPSET/dbf/ \
to_driver=pg to_table=roads2 \
to_database="host="myhost.com,dbname=spearpg"
```

- Connect vector map to PostgreSQL

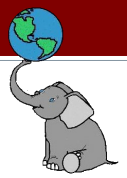
```
v.db.connect -o map=roads2 driver=pg \
database="host=myhost.com,dbname=spearpg"
table=roads2 key=cat layer=1
```

- Show connection details

```
v.db.connect -p roads2
```

Hands-on part!



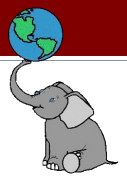


GRASS and PostgreSQL/PostGIS

GRASS synergy with PostgreSQL and its spatial extension PostGIS are explored:

- creating a spatial database (PostGIS);
- dumping a GRASS vector map into the spatial database;
- using PostgreSQL to analyze and transform semantic data (attributes);
- attaching attributes to a GRASS vector map;
- editing PostgreSQL data from GRASS;
- displaying a PostGIS layer within GRASS;
- dumping a PostGIS layer into GRASS.





GRASS and PostgreSQL/PostGIS

Prerequisites

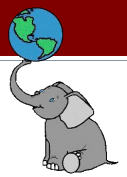
- PostgreSQL installed and configured;
- PostGIS installed and ready to use;
- GRASS 6.x built with PostgreSQL extension;
- PostgreSQL client available;
- PostgreSQL (super)user.

Software version used here

- PostgreSQL 8.1.8
- PostGIS 1.1.6
- GRASS 6.2.1
- pgAdminIII 1.4.3

All software is available on the live DVD and the PostgreSQL *grass user* is already setup.





Creation and setup of a spatial database

PostgreSQL server must be available. **From a system console:**

•Create a new PostgreSQL database, called *spearpg,* as user *grass* belonging to user *grass* on *localhost*

```
createdb spearpg -O grass -h localhost -U grass
```

•Add the *plpgsql* procedural language

```
createlang plpgsql spearpg -U grass
```

•Load PostGIS spatial extension (system tables, function, geometric types) for later use

```
psql spearpg -U grass -f /usr/share/postgresql-8.1-
postgis/lwpostgis.sql
```

•Load the datum definitions

```
psql spearpg -U grass -f /usr/share/postgresql-8.1-
postgis/spatial_ref_sys.sql
```

Dumping GRASS vector dataset into PostGIS

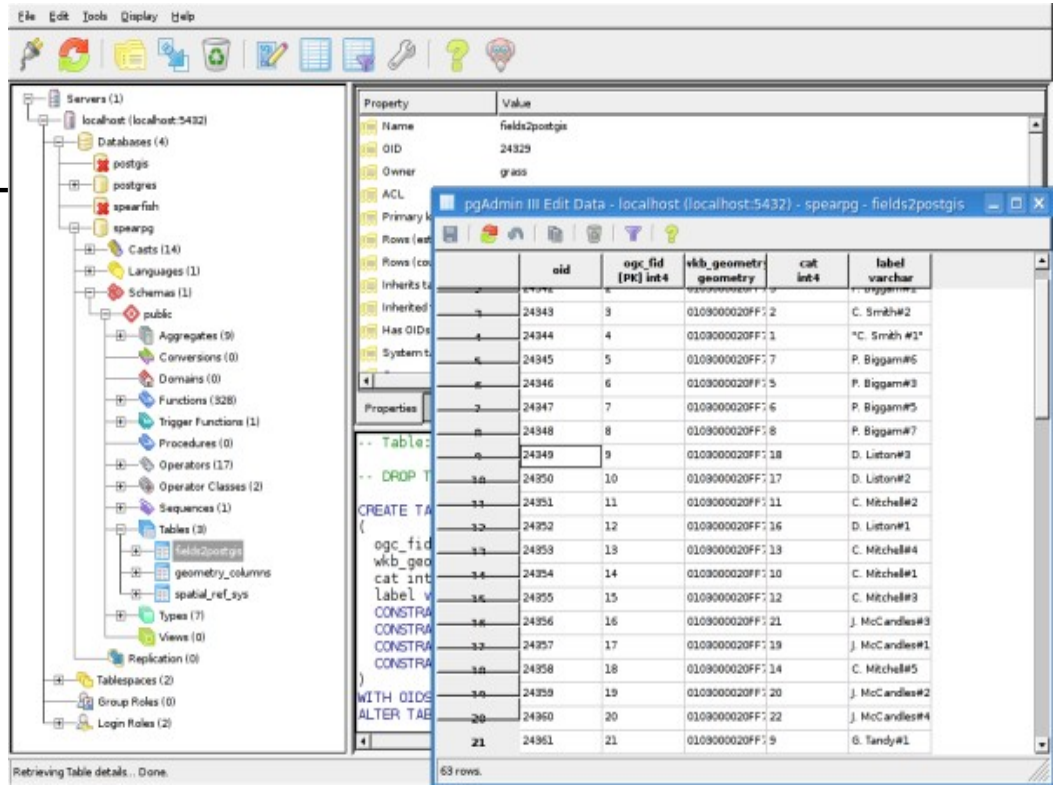
- Start GRASS clicking on this desktop icon and select LOCATION spearfish60 and MAPSET user1

- Copy vector map fields from MAPSET PERMANENT (ro) to fields2 in mapset user1

```
g.copy vect=fields,fields2
```

- Dump fields2 into PostGIS, inserting only features with category (-c)

```
v.out.ogr -c input=fields2
   type=area
   'dsn=PG:host=localhost
   dbname=spearpg user=grass
   password=grass'
   olayer=fields2postgis layer=1
   format=PostgreSQL
```



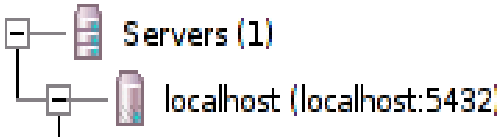
- The result can be viewed via pgAdmin III: [Menu->Developement->Database->pgAdmin III]

Transform semantic data with PosgreSQL

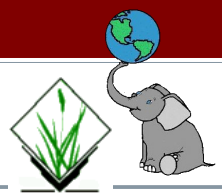
PostgreSQL can be used to modify the information stored in the tables. It grants data consistency, concurrent accesses, ACID support and high performance on large datasets; it provides lots of functions for working on data types. Any client can be used for tables editing via SQL, here PgAdmin III is used.

First of all you have to login as *grass* user and connect to the localhost server:

- right-click on the *localhost* label in the left-side panel, select *properties* and change the username from *postgres* to *grass.*



- Right-click on the *localhost* label in the left-side panel and select *connect .*



Transform semantic data with PosgreSQL

The aim is to create a new ”owner” column containing owners' names without trailing numbers and spurious characters (# and “) so that we can select fields by owner (it could be done using LIKE or ~ operators)

•Open the query-builder clicking on the icon 

•Add a new column to the *field2postgis* table:

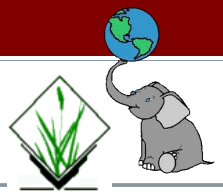
```
ALTER TABLE fields2postgis ADD COLUMN owner varchar
```

•Fill new column with owners' names

```
UPDATE fields2postgis SET owner= trim(both '#"' from
split_part(label,'#',1))
```

•Create a new table without geometry

```
CREATE TABLE fields2att AS SELECT cat, owner FROM
fields2postgis
```



Attaching attributes to a vector layer

•The original vector is connected to a DBF file

```
v.db.connect fields2 -p
```

•The *owner* column is not present as you can see consulting the table via *gis.m* Now detach the DBF table (this is not mandatory if you use the *-o* option)

```
v.db.connect -d map=fields2 driver=dbf
database=$GISDBASE/$LOCATION_NAME/$MAPSET/dbf/ key=cat layer=1
```

•Connect to the PostgreSQL DB and login

```
db.connect driver=pg database=spearpg
```

```
db.login driver=pg database=spearpg user=grass
```

•Attach the PosgreSQL table

```
v.db.connect map=fields2 table=fields2att key=cat layer=1 -o
```

•Now check the connection parameters. The owner column is now available

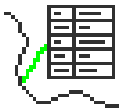
```
v.db.connect fields2 -p
```

Editing PostgreSQL data from GRASS

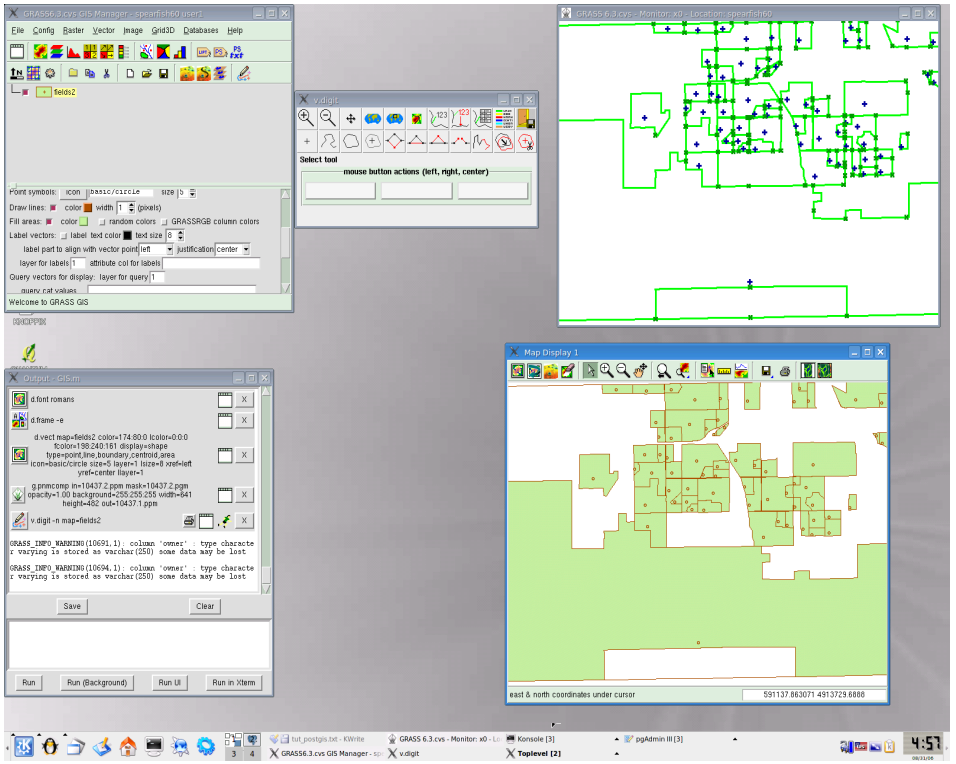
•Data can be displayed and queried via GRASS *gis.m* interface.

•Attributes are taken from PostgreSQL (see DB and table name).

•Attributes can be edited from GRASS using *v.digit* using this  icon on the GIS manager.

•The attributes stored in PostgreSQL table can be modified directly with 

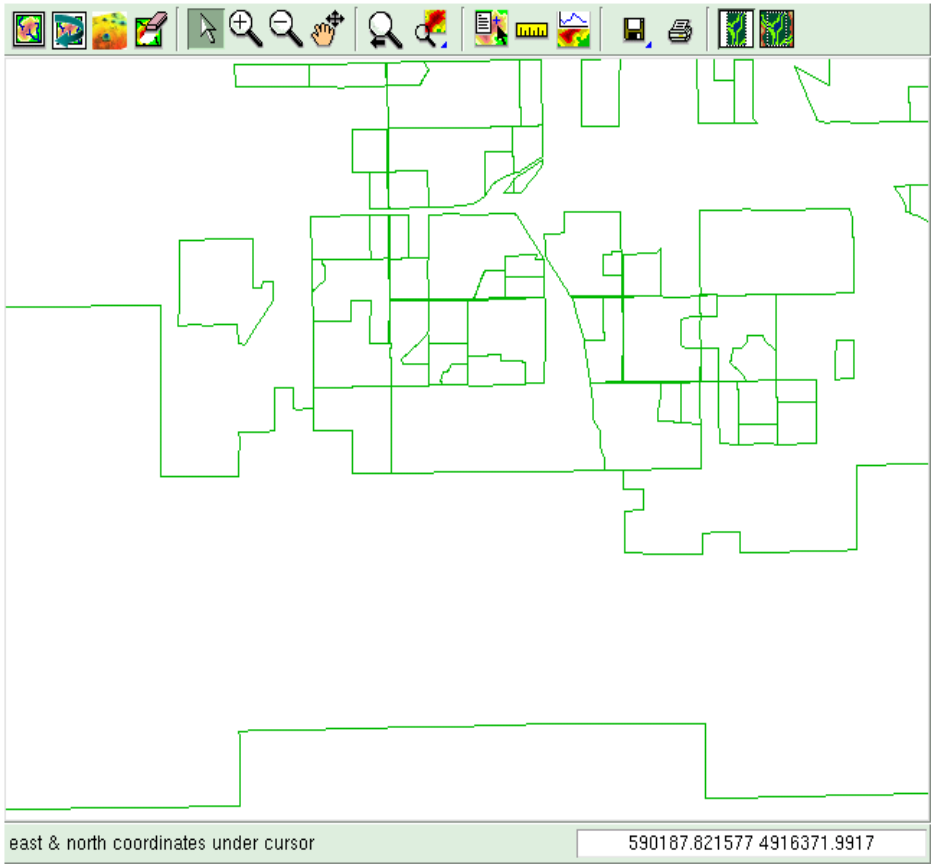
•It is possible add, modify or remove geometry features (stored in GRASS) and related attributes (stored in PostgreSQL) with *v.digit*.



Displaying a PostGIS layer within GRASS

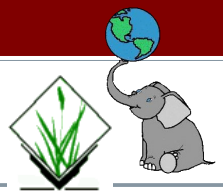
•A PostGIS layer (geometry plus attributes) can be displayed using *v.external* (read-only mode) and *gis.m*

v.external dsn="PG:host=localhost dbname=spearpg user=grass password=grass" output=fields2external layer=fields2postgis



•GRASS can not build topology nor modify new features or attributes (read only mode) to the PostGIS layer via OGR.

•The attributes cannot be queried by selecting geometric features (because topology is not available).



Displaying a PostGIS layer within GRASS

- A PostGIS layer (geometry plus attributes) can be dumped into GRASS via OGR using *v.in.ogr*

```
v.in.ogr dsn="PG:host=localhost dbname=spearpg
  user=grass password=grass" output=fields2ogr
  layer=fields2postgis
```

- Topology is built and features can be added, modified and removed with *v.digit*. However attributes are still uploaded in a new PostgreSQL table while geometry is stored in native GRASS format.