Exercise 13: Integrating Spatial and Non-Spatial Data

Table of Contents

1. Intro-fossgis-umass 1
    1.1. Author Attribution 1
    1.2. Module Licensing Information 1
    1.3. Reviewed by 1
2. Integrating spatial and non-spatial data 2
    2.1. From Intro-fossgis-umass 2
        Introduction 2
        Creating a thematic map of construction intensity in Massachusetts 5
        Additional materials 13
        Data and Data Sources 13

1. Intro-fossgis-umass

1.1. Author Attribution

Major contributors to this curriculum include (alphabetical):

Maria Fernandez

Michael Hamel

Quentin Lewis

James Peters

Charlie Schweik

Alexander Stepanov

1.2. Module Licensing Information

Version 1.0.

1.3. Reviewed by

Quentin Lewis 04/09/07

2. Integrating spatial and non-spatial data

2.1. From Intro-fossgis-umass

Introduction

The purpose of this module is to demonstrate how to join a spatial layer and a non-spatial attribute table, in such way that the attribute information would be available within the GIS package. Some typical examples of situations when you need to perform a join of spatial and non-spatial data are:

1. you have a spatial layer for demographic areas (political boundaries)?
    a. you have statistics for the same area on crime, health coverage or income
    b. you would like to create a thematic map which presents trends in crime, health coverage, etc.
2. you have a spatial layer representing a national park or open space area and a point layer representing observation points
    a. an attribute table with given observation details (time, what happened, etc) is available also.
    b. you would like to create a map showing temporal trends of observed phenomena.

In this module we will learn how to join a spatial layer and a non-spatial attribute table within PostgreSQL/PostGIS. So it is assumed that we have a spatial layer(s) and statistical data(tables) for the same area. These data are stored in PostgreSQL/PostGIS and we would like use these data to create a thematic map with QGIS. To perform this operation it's necessary to understand "communication" among QGIS and PostgreSQL/PostGIS.

A short overview of QGIS, pgAdmin and PosgtreSQL/PostGIS framework

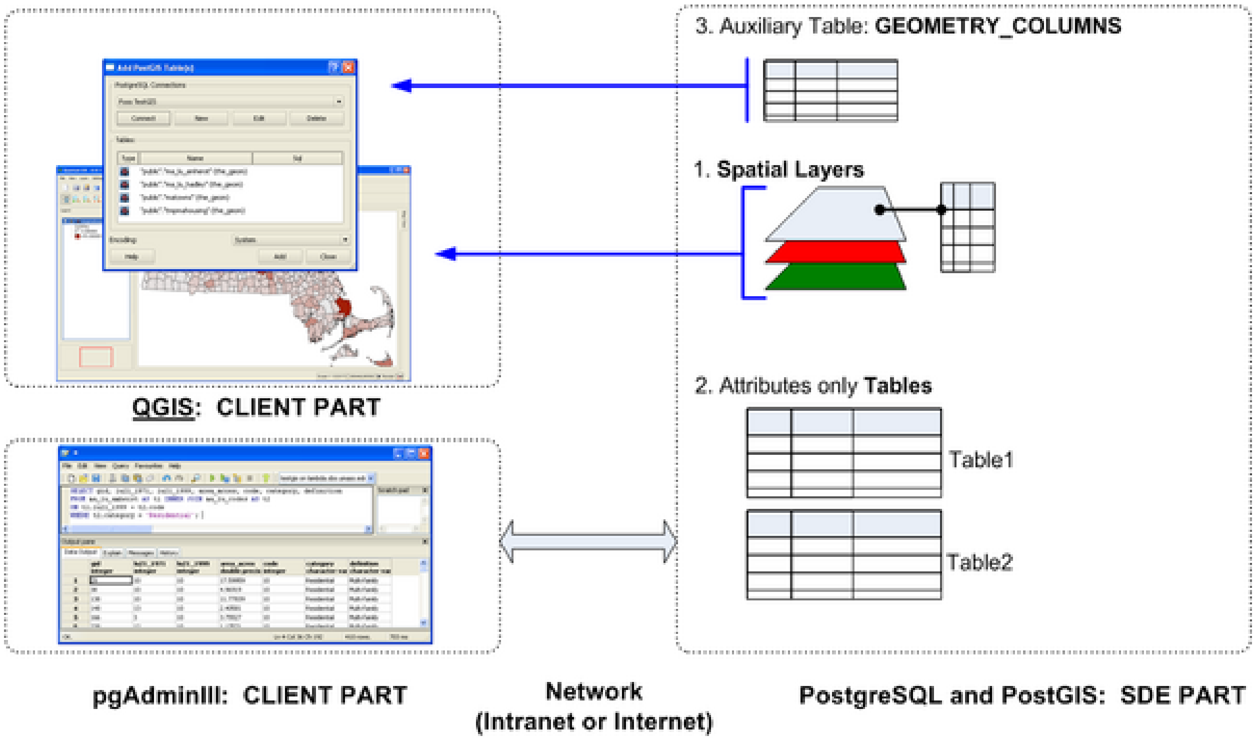


Figure 1. shows a typical and practical setup, where users run QGIS on local machines and shared data is stored on a PostgreSQL/PostGIS server. The spatial database on the server provides access to spatial/gis layers and to non-spatial attribute tables. Users can connect to PostGIS with QGIS. For more advanced work with remote data, pgAdmin can be used. The database provides access to:

1. Spatial layer(s) and their attribute tables

2. Non-spatial tables.

PostGIS databases have a special auxiliary table **"GEOMETRY_COLUMNS"** which consists of a list of spatial layers available for QGIS. In other words, when you use QGIS and add a spatial layer from a PostGIS database to a current project, you see a list of available layers to add which matches a list of layers in the **"GEOMETRY_COLUMNS"** table.

Taking into the account this observation, let's outline the major steps in the process of joining a spatial layer and a non-spatial table (Figure 2).

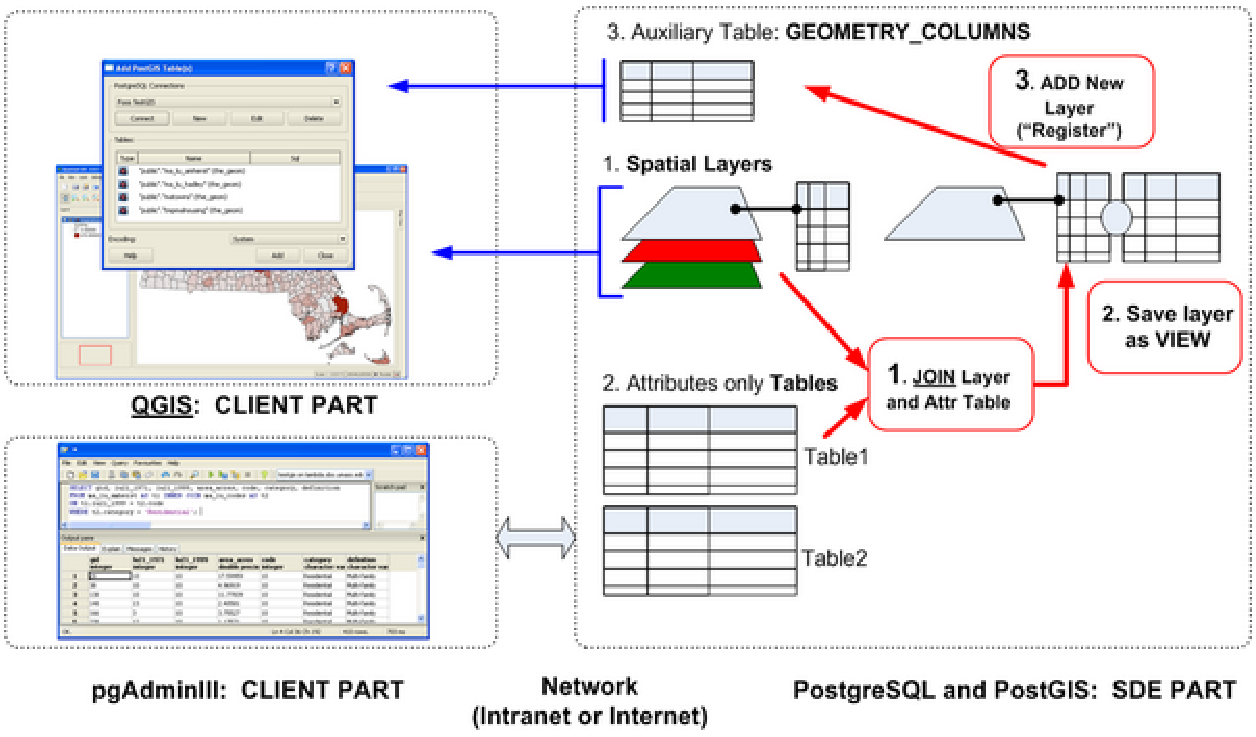

3. Auxiliary Table: GEOMETRY_COLUMNS
3. ADD New Layer ("Register")
1. Spatial Layers
2. Save layer as VIEW
1. JOIN Layer and Attr Table
2. Attributes only Tables
Table1
Table2
QGIS: CLIENT PART
pgAdminIII: CLIENT PART
Network (Intranet or Internet)
PostgreSQL and PostGIS: SDE PART

Figure 2

The major steps are:

1. To perform a?JOIN of a table representing spatial layer and attribute table data

2. To store result of the JOIN (as **view**)

3. To add a name for the new layer/view into the **"GEOMETRY_COLUMNS"** table, then the new layer can be added to QGIS projects.

Step 1, You already know how to perform a JOIN of two tables (with the JOIN keyword or with WHERE clause). We will discuss steps 2 and 3 in more details below.

Saving results of query with VIEW

As you remember, the result of any SQL query on table(s) is a table (expressing it in simple terms). The natural question after doing so many SQL queries is how to save results of the SELECT query? For example, if we need to repeat our analysis on a table after adding new records, or if some processed data are needed for users. It can be done with the "CREATE TABLE" or "CREATE VIEW" SQL statements. The general format of these statements are:

```
CREATE TABLE <nameOfNewTable> AS
SELECT <select statement>
```

or

```
CREATE VIEW <nameOfNewView> AS
SELECT <select statement>
```

The "idea" of both statements is to perform the SELECT statement on table(s) and to **store** the results in a table or in a **view**. What is a view? What is the difference between a table and a view? When you use the "CREATE TABLE" statement, it will create a new table and populate it with the data according to the SELECT statement. What is important to understand is that data will be **cloned/copied** into the new table permanently. We can think of **VIEW** as a filter or a recipe. There is no copying of data or replication of data, data are stored in the original tables, but "view" is constructed and data gathered as you need them. If we use the "cooking recipe" analogy:

1. when we "create table" we actually cook a dish and store it.
2. when we "create view" we use 'recipe' to create a dish "on-the-fly" (so we don't need to store additional ingredients, as they are stored in the original tables). We can keep only recipes.
3. you can keep stock of coffee, sugar, milk, spices and everyone can prepare coffee according to his/her taste and preferences (this is VIEW). The same with data, VIEWs provide an effective and economical way to process data, you can also create different VIEWs for different group of users. E.g. people in your organization can have "Views" with one set of data, and you can provide another set of "VIEWs" for general public with more restrictions.

Adding new data to a table

To add new data to a table (in other words, to add a new row/record to a table), the SQL **INSERT** statement can be used. This statement 'adds/inserts' new records and assign values to fields/attributes. The format of the INSERT statement is:

INSERT INTO VALUES(<list of values>) Please, note that list of values should repeat the order of columns. This general explanation will be more clear later. For now just remember that new records can be added with the **INSERT**statement. Now we know all of the operations to join the spatial layer and non-spatial attribute table. [edit [http://linuxlab.sbs.um?ss.edu/intro-fossgis-umass/index.php?title=Integrating_spatial_and_non-spatial_data&action=edit&section=5]]

Creating a thematic map of construction intensity in Massachusetts

FossGIS server stores a spatial layer of Massachusetts towns (table: **matowns**) and info on building permits issued in MA towns in 2005 (table: **ma_bldngs2005**). What we would like to do is create a thematic map of construction activities, showing the number of building permits issued in Massachusetts. This is a very typical project/question if you work in regional planning, policy analysis or consulting.

Reviewing data/tables

Let's study our data before proceeding further. Please start up the pgAdminIII tool and connect to the TestGIS database on the FossGIS server. If you are having trouble connecting to the database, please look back at this exercise: Using Data Layers from Spacial Data Engines [http://linuxlab.sbs.umass.edu/introFossgisUmass/index.php?title=Using_data_layers_from_Spatial_Data_Engines_%28SDE%29:_Pos

Please, run the following queries to get a feel for the data. Pay attention to the execution time while running queries:

```
SELECT *
FROM ma_bldngs2005;
```

```
SELECT *
FROM matowns;
```

(It usually takes about 25 seconds to execute the query, this can however differ greatly depending of your internet connection).

We will narrow the select query since we don't need all fields/attributes. The following attributes will be used: gid, objectid (both are internal fields for the spatial layer), town_id (id of town), town (town name), pop2000 (population according to the 2000 Census), shape_area (area of each town in sq.meters).

```
SELECT gid, objectid, town_id, town, pop2000, shape_area
FROM matowns;

execute time ~0.7sec
```

We will also use a subset of attributes from the building permits table. We will use town_id(id of town), building (number of buildings allowed to construct), units (number of units) and constrcost (estimated cost of construction). Please run the following query:

```
SELECT town_id, building, units, constrcost
FROM ma_bldngs2005;
```

```
execute time ~0.5sec
```

It's important to note that tables have fields which store 'common' information for both tables: **field town_id**. This field is a primary key for the building permit table and the foreign key for the matowns table. We can use this field to join tables.

Joining two tables together

At this stage of our project, lets perform a SELECT query to JOIN the spatial layer (which is table matowns) and attribute table **ma_bldngs2005**. Please analyze the query below:

```
SELECT gid, objectid, t1.town_id, town, pop2000, shape_area, t2.town_id, building, units,
FROM matowns AS t1, ma_bldngs2005 AS t2
WHERE t1.town_id = t2.town_id;
```

Please note:

1. how we defined?a list of fields after SELECT keyword (all fields enumerated as is, but town_id fields, we use names of tables to distinguish them: t1.town_id and t2.town_id)

2. how we defined *aliases* for table names in the part after FROM keywords. We note matowns AS t1 and ma_bldngs2005 AS t2. So now we can address the tables with shorter names(aliases) t1 and t2.

3. how we set the JOINING condition: t1.town_id = t2.town_id.

Please run this query and see the results. Now the question is how could we *save* the results of the query for future use with PostGIS or with QGIS?

Create a VIEW

To save data which we have merged from two tables, we can add the "CREATE VIEW <nameOfyourViewHere> AS" statement before SELECT query:

```

CREATE VIEW tmpMaHousing AS
SELECT gid, objectid, shape_area, shape_len, the_geom, t1.town_id, town, pop2000, buildi
FROM matowns AS t1, ma_bldngs2005 AS t2
WHERE t1.town_id = t2.town_id; ( Please Note: this file already exists on the sever)
```

This query creates a VIEW with the name tmpMaHousing which is defined by the SELECT statement. So this view is a recipe which 'recreates' and merges data for you. The important point is that you can use the VIEW (in this case tmpMaHousing) in your queries in the same way you use TABLES.

You can also see a list of available views in the database with the pgAdmin application. Please see the Figure below:

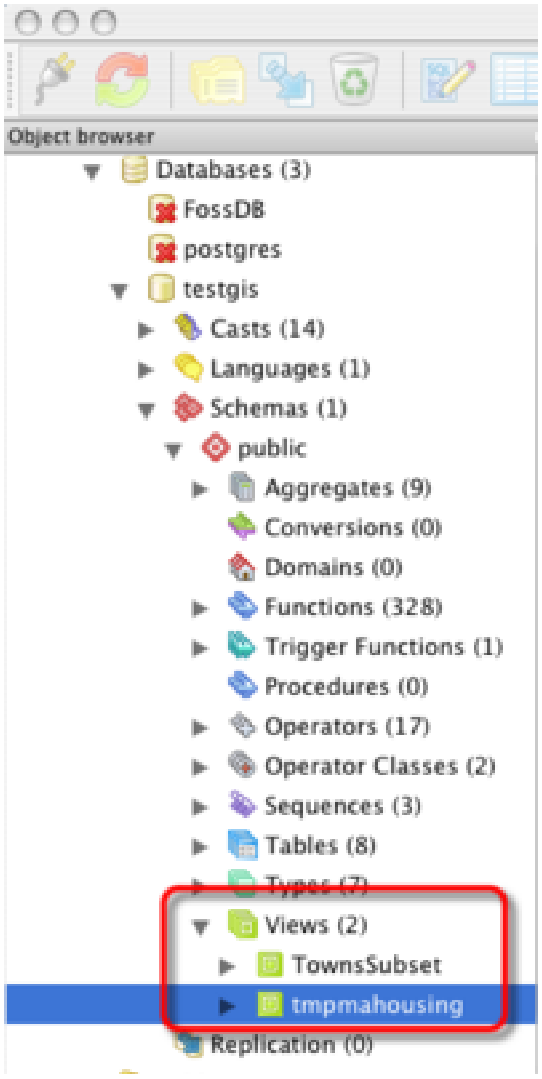


Figure 3. List of VIEWS in pgAdmin console

Registering new spatial layer

At this point in our project, we have defined a new spatial layer/VIEW which combines data from the town spatial layer and the building permits attribute table.

To make this layer visible/available for QGIS users, we need to *register* the VIEW in **GEOMETRY_COLUMNS** table. So we need to add a record with information about our new layer. After that the layer will be available for usage in QGIS (in other words, when a user connects to the PostGIS server, this new layer will appear in the list of available spatial layers to add).

We will add new records with the following INSERT statement:

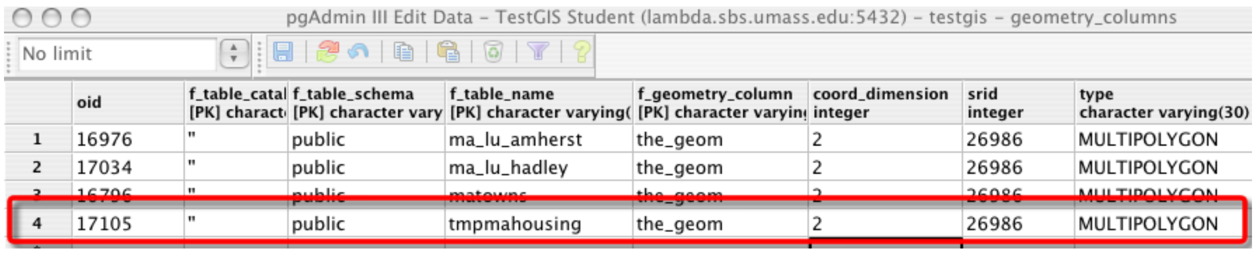
```
INSERT INTO geometry_columns
VALUES(' ','public','tmpmahousing','the_geom',2,26986,'MULTIPOLYGON');
```

Figure 4. INSERTing new record into GEOMETRY_COLUMNS table

Please note that you need to substitute 'tmpmahousing' with the name of your view. Here are some details about this query:

1. we insert data into the table "GEOMETRY_COLUMNS" which is defined after the INSERT INTO keywords
2. the order of values in the VALUES() part, corresponds to the order of the columns in the table. In other words, the position of data in the list of values defines which column it will be written to.
3. there are single quotation marks before ,'public'. It means that we don't record any data in that field.

Please see the results of the query in Figure below:


pgAdmin III Edit Data - TestGIS Student (lambda.sbs.umass.edu:5432) - testgis - geometry_columns

	oid	f_table_catal [PK] charact	f_table_schema [PK] character vary	f_table_name [PK] character varying(	f_geometry_column [PK] character varyin	coord_dimension integer	srid integer	type character varying(30)
1	16976	''	public	ma_lu_amherst	the_geom	2	26986	MULTIPOLYGON
2	17034	''	public	ma_lu_hadley	the_geom	2	26986	MULTIPOLYGON
3	16796	''	public	matowns	the_geom	2	26986	MULTIPOLYGON
4	17105	''	public	tmpmahousing	the_geom	2	26986	MULTIPOLYGON

Figure 5. Result of the insert query

The value for the field **oid** was generated automatically, we specified all other values.

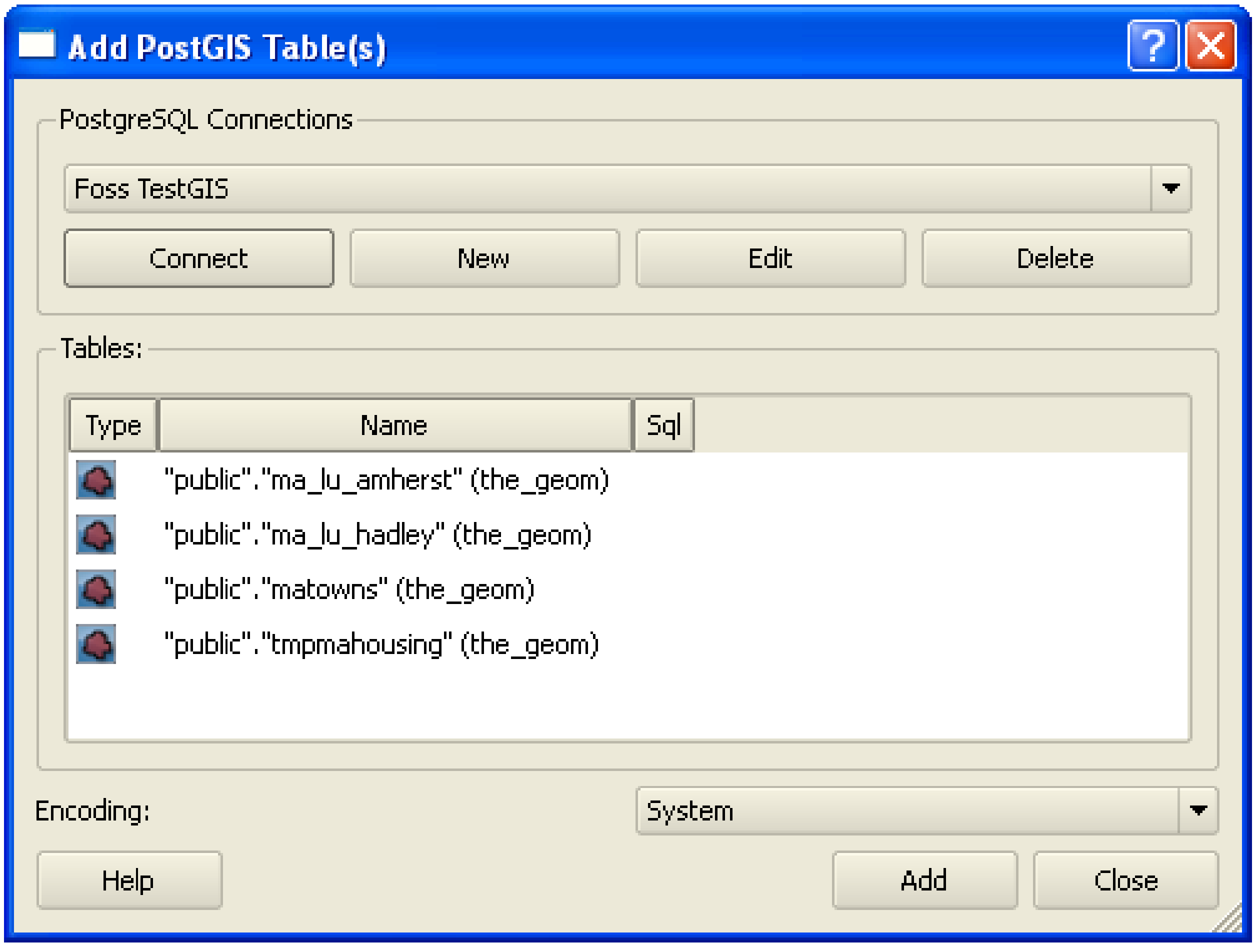
So let's just review what we did:

1. we have joined spatial and non-spatial tables and 'saved' result as a view
2. we registered the view, which made this new layer available to use with QGIS

Now let's try to use the layer with QGIS. You can close the pgAdmin application and start QGIS application.

Working with new aggregated spatial layer within QGIS

Now let's switch our role from database gurus to QGIS users. Please switch to QGIS and add the PostGIS layer you have created . Click on the "add PostGIS layer" button and connect to the TestGIS server. You will see a list of available spatial layers in the window (Figure 6):



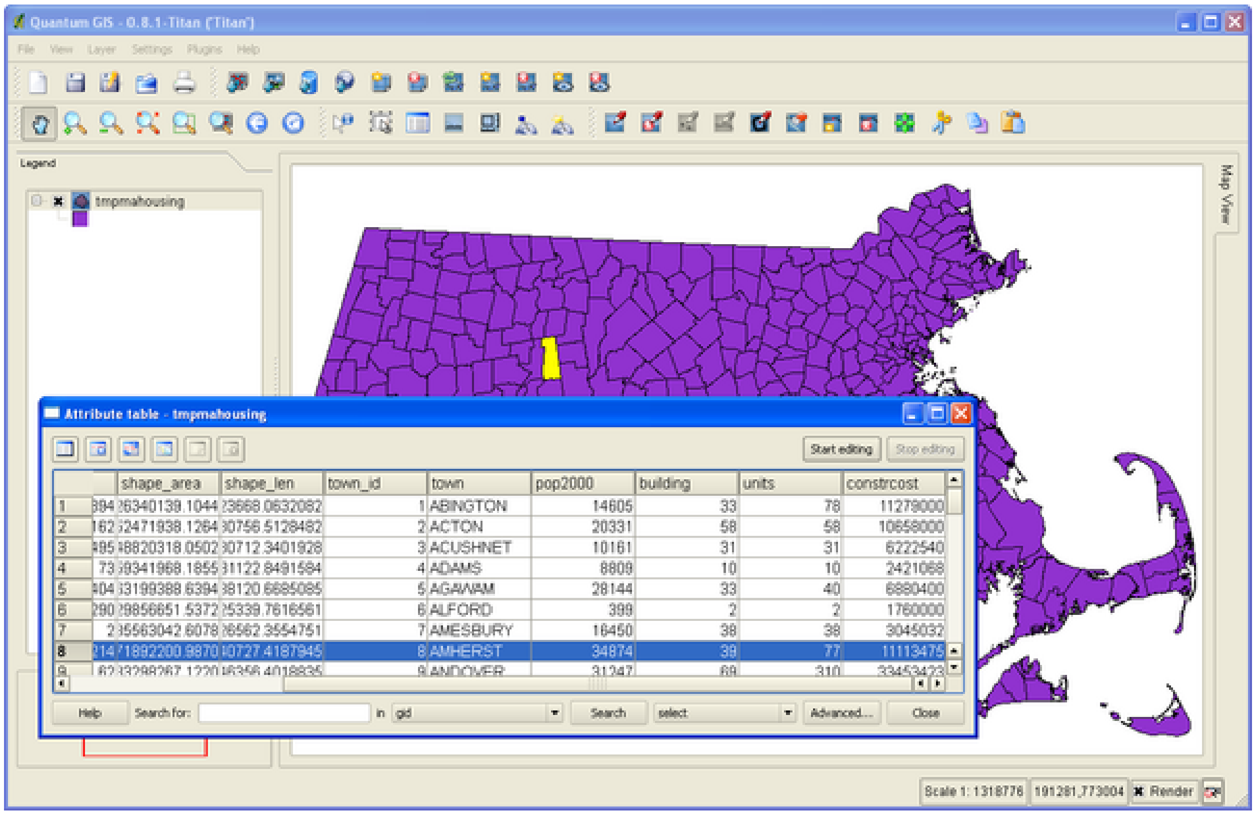
Foss TestGIS

Type	Name	Sql
●	"public"."ma_lu_amherst" (the_geom)	
●	"public"."ma_lu_hadley" (the_geom)	
●	"public"."matowns" (the_geom)	
●	"public"."tmpmahousing" (the_geom)	

Encoding: System

Figure 6. A list of spatial layers stored on PostGIS server and available for QGIS

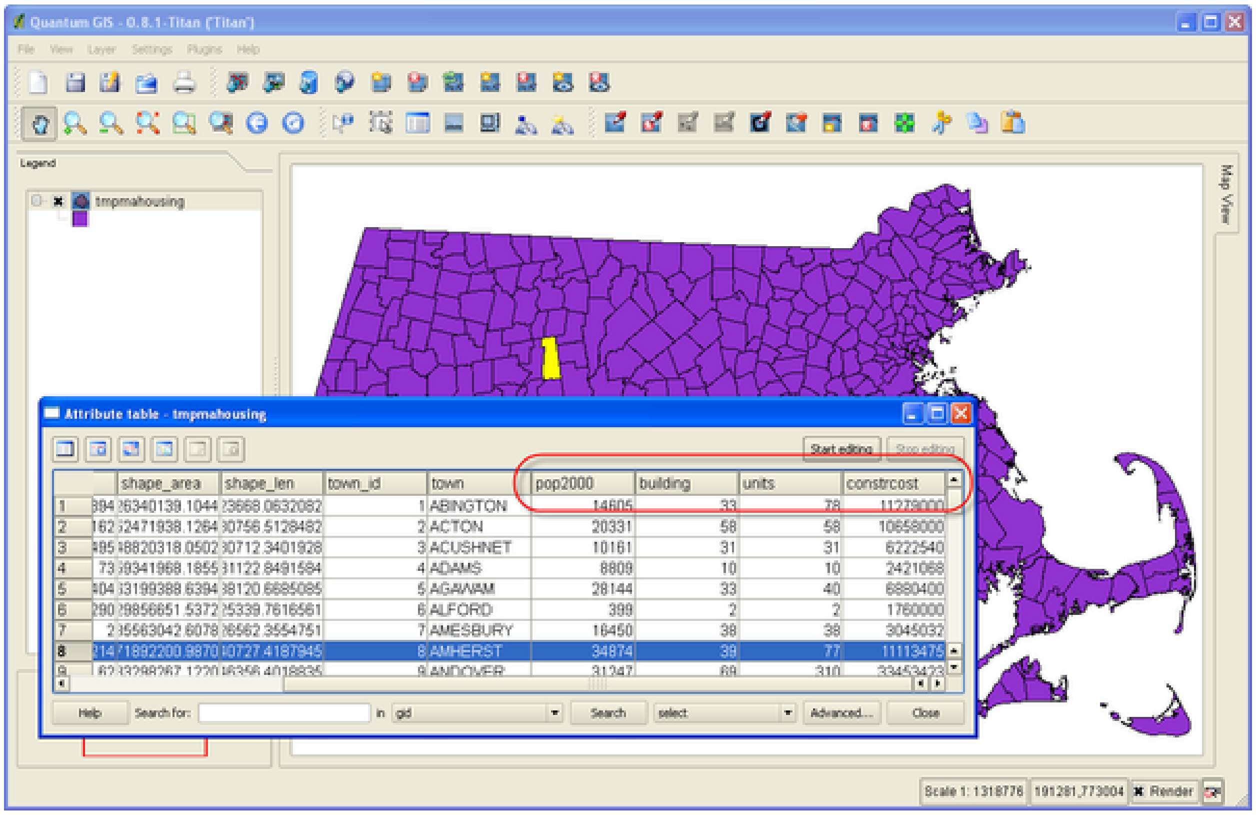
Select the layer you have created and add it to the QGIS project. (you can see our layer 'tmpMaHousing' which the FossGIS team created).



		shape_area	shape_len	town_id	town	pop2000	building	units	constrcost
1	394	26340139.1044	23668.0632082	1	ABINGTON	14605	33	78	11279000
2	162	52471938.1264	30756.5128482	2	ACTON	20331	58	58	10658000
3	495	48820318.0502	30712.3401928	3	ACUSHNET	10161	31	31	6222540
4	73	59341968.1855	31122.8491584	4	ADAMS	8809	10	10	2421068
5	404	63199388.6394	38120.6685085	5	AGAWAM	28144	33	40	6880400
6	290	29856651.5372	25339.7616561	6	ALFORD	399	2	2	1760000
7	2	35563042.6078	26562.3554751	7	AMESBURY	16450	38	38	3045032
8	214	71892200.9870	40727.4187945	8	AMHERST	34874	39	77	11113475
9	62	33298267.1220	46356.4018835	9	ANDOVER	31247	69	310	33453423

Figure 7. Attribute data of the newly created layer

Open the attribute table for the layer (this may take a minute, and we have have observed QGIS .8 for Windows freezing on this step, but wait a couple of minutes before you assume that the application has frozen. If the application does freeze restart QGIS and attempt the step again.). Take a moment to analyze a list of available attributes and the number of records in the dataset. What is important is that you HAVE ACCESS to building permits data in QGIS: check fields *buildings*, *units*, etc. These are data from the building permits table joined to the spatial layer (Figure 8).



		shape_area	shape_len	town_id	town	pop2000	building	units	constrcost
1	384	26340139.1044	23668.0632082	1	ABINGTON	14605	33	78	11279000
2	162	52471938.1264	30756.5128482	2	ACTON	20331	58	58	10658000
3	485	48820318.0502	30712.3401928	3	ACUSHNET	10161	31	31	6222540
4	73	59341968.1855	31122.8491584	4	ADAMS	8809	10	10	2421068
5	404	63199388.6394	38120.6685085	5	AGAWAM	28144	33	40	6880400
6	290	29856651.5372	25339.7616561	6	ALFORD	399	2	2	1760000
7	2	35563042.6078	26562.3554751	7	AMESBURY	16450	38	38	3045032
8	214	71892200.9870	40727.4187945	8	AMHERST	34874	39	77	11113475
9	62	83298287.1220	46358.4018835	9	ANDOVER	31247	69	310	33453423

Figure 8. Building permit data are available within QGIS

Now you can create a thematic map, showing the number of building permits issued by towns in Massachusetts. To do that you need to change the symbology settings and use the "Buildings" field for that. (Please note that this process may also take a couple of minutes, and we have observed QGIS .8 for Windows freezing on this step, please wait a couple of minutes before you assume that QGIS has frozen. If you are sure that it has frozen, restart the application and attempt the step again.)

1. Try this using **Continuous Color**, which is the legend type under symbology.

   a. Set the minimum and maximum color values you would like, and QGIS will create a gradient for you. (In the example below, the minimum value would be white, the maximum value would be red.

   b. Be sure that you have changed the **Classification field** to building.

2. Now try using the **Graduated Symbol** option.

   a. Choose the number of **Classes** or ranges that you would like to see on your map (try creating at least 4).

      i. If you choose equal interval the computer will automatically create intervals for the number of classes you have chosen.

      ii. If you choose empty, you can double click on each classification (Empty1, Empty2, and so on) to assign a high and low number to that classification.

      iii. Remember to set the classification field to building.

iv. Also remember to create a label for each classification.

v. Finally, pay attention to the fill color for each classification, use the default colors to begin, then try to update the colors to immitate the white to red gradiant in the image below.

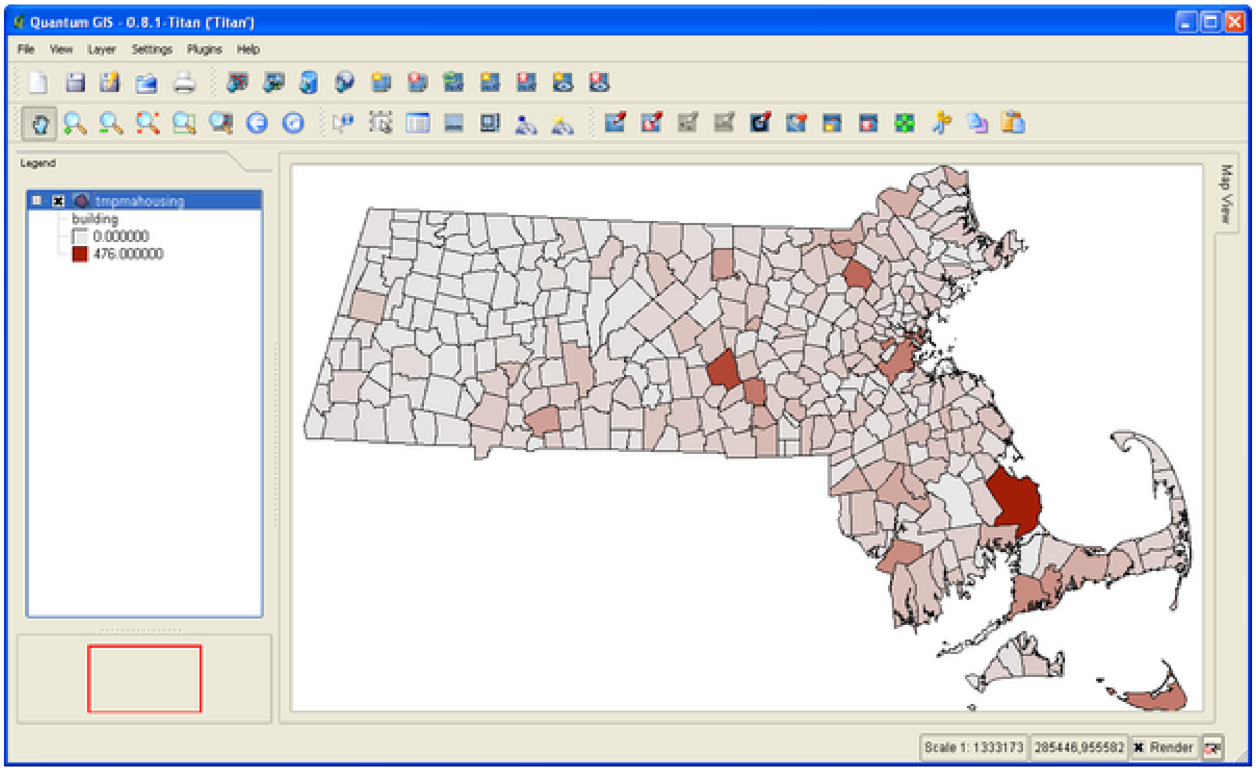


Figure 9. Final thematic map

The final thematic map could look like the Figure above, where darker colors represent greater building activity.

Additional materials

1. An article on SQL on Wikipedia http://en.wikipedia.org/wiki/Sql
2. SQL Tutorial at http://www.w3schools.com/sql/default.asp
3. Another SQL Tutorial at http://www.sql-tutorial.com/

Data and Data Sources

1. Spatial layer of towns' political boundaries is provided by [MassGIS [http://www.mass.gov/mgis/]]
2. Data on building permits for Massachusetts are provided by Census and [MISER [http://www.umass.edu/miser]]