Embedding animal density information into the Swiss Feed Database

Facharbeit im Nebenfach Informatik

Lukas Urech
Zürich, Schweiz
09-713-819

Department of Informatics – Database Technology
Binzmühlenstrasse 14
CH-8050 Zürich
Prof. Dr. Michael Böhlen,

Betreuer: Andrej Taliun

19.04.2013

Content

1. Introduction

1.1. Overview

The goal of my task was to import given animal density data in the Swiss Feed Database and allocate them with geographical data. The data should be integrated in form of new tables which are not directly connected to the already existing ones in the Swiss Feed Database. In the end several different queries can be made about the new data and density information of the animals bovines, cows and pigs can be assigned to the area of the community they were taken. This opens many possibilities about queriyng the database geographically

The results were three additional tables to the Swiss Feed Database with which querries could be made about animal densities belonging to several Swiss communities. In the end of my Report there are 5 specific querries about the new data and a short description, what the main benefit of the additional tables is. One can for example define a polygon, which lies within the Swiss boundaries, and calculate the animal density (bovines, cows, pigs, different years) for that area.

The main problem therefor was to find proper geographical data of Swiss communities, to which the animal density data could be assigned.

During my work it became clear, that the best dataset for this task is the GG25 (Bundesamt für Statistik, 2013), which is provided by the Swiss government. The dataset contains the digital borders of all the Swiss communities.

In the first part of my report, the Swiss Feed Database is shortly explained. The second part is about the data I was using and the description of it. The third part shows the methodology. This means, how the data was exactly imported in the database. The final result and the description of the new tables comes in the fourth part. There one can find also the specific queries. In the end a short conclusion is given and a forecast at future work is made.

1.2. Description of the Swiss Feed Database.

The Swiss Feed Database is a public service for companies, private farmers and research institutions (Universität Zürich, 2013). The data in this database consists of measurements of different nutrients contained in feed samples, which were taken all over Switzerland. The goal of the Swiss Feed Database project is to design and implement new database techniques for modelling, aggregating and analysing feed data. Users can access the database over an online interface. This interface is available in three different languages and it allows users to search for detailed data as well as for aggregated data (Swiss Feed Database, 2013).

The Swiss Feed Database contains of several different tables. In the next part I will present the most important ones and explain them shortly.

- feed categories:

Feeds are identified by an artificial id. The name of the feed is also stored there in three languages (English, German and French). If many feeds belong to the same feed category, the name of this feed group can also be stored in the same table. The name of this table is d_feed.

id	parent_id	name_de	name_en	name_fr

- nutrients:

Nutrients are stored with name, abbreviations and descriptions, all three in English, German and French. Here also the major part of nutrients can be grouped.

id	name_de	name_en	name_fr	abbreviation_de/en/fr	description_de/en/fr

- origin:

This table contains information about the location from where one feed sample was taken. The table holds all the geographical information such as postal code, city, canton, altitude, latitude and longitude. The id is in the table to refer it to other information about the feed sample.

id	postal_code	city	canton	altitude	latitude	longitude

- feed sample:

The table consists of an id, which is the primary key of this relation. The other important information are in the column lims_number, which prevents a string that uniquely identifies the sample within the Lims system at Agroscope, and the preparation, which describes the preparation of the sample before the analyses.

id	lims_number	preparation

- time information:

Each sample can have up to 4 dates: harvesting, sampling, arrival and analyses dates. But only harvesting date is present for the most of the feed samples, so there were used four attributes to store time information. These are the day, the year the month and the season in the four languages. The id of the sample is again the primary key.

id	day	year	month	season_de	season_en	season_fr

- measurement:

Here, the chemical analysed quantity of a nutrient in a particular sample is stored. The time and the feed type is stored as well in this table to identify every measurement uniquely.

id	quantity	nutrient	sample	origin	time	feed

The central table of the Swiss Feed Database is though the fact_table. This table contains all the foreign keys to the other tables. For every new entry in the database, there will also be a new entry in the fact_table.

id	lims_nr	Quantity	time_key	nutrient_key	origin_key	sample_key	feed_key

2. Explaining the data

2.1. The density data

I received the density data in a table called Animal_density. After opening this table with a spreadsheet program like Microsoft Excel, one could see, that the table consists of three different parts. The first part shows the total amount of cows, bovines and pigs per hectare farmland of a community. The second and the third part of the table just counted the farmland belonging to a farm, which contains the actual animal. This means farms with only bovines are not concerned in the table about pigs. The second part shows the amount of bovines per hectare farmland belonging to farms with bovines (Table 2) and the third one shows the amount of pigs per hectare farmland belonging to farms with pigs (Table 3).

1: Total GVE od. Rinder od. Schweine der Gemeinde durch Total Ln der Gemeinde

		On the total land	Cows per ectar	Bovines per ectar	Pigs per ectar
JAHR	GEMEINDE_NUMMER	GEMEINDE_NAME	GVE_TOT_PRO_HA_LN	GVE_RINDER_PRO_HA_LN	GVE_SCHWEINE_PRO_HA_LN
2005	1	Aeugst am Albis	*	*	*
2005	2	Affoltern am Albis	*	*	*
2005	3	Bonstetten	*	*	*
2005	4	Hausen am Albis	*	*	*
2005	5	Hedingen	*	*	*
2005	6	Kappel am Albis	*	*	*
2005	7	Knonau	*	*	*

Table 1: Animals per hectare farmland of a community

The first three columns are for each of this three parts of the table the same. They contain the year, in which the density data is taken, in column one, the number of the community in column two and the name of the community in column three. The other columns contain the actual density value. Tables 1-3 show the first 7 entries in this tables.

2: Total Rinder (gve) der Gemeinde durch total LN der Betriebe mit Rinder

On the farm-land

JAHR	GEMEINDE_NUMMER		GEMEINDE_NAME	GVE_RINDER_PRO_HA_LN_R	
2005		1	Aeugst am Albis		*
2005		2	Affoltern am Albis		*
2005		3	Bonstetten		*
2005		4	Hausen am Albis		*
2005		5	Hedingen		*
2005		6	Kappel am Albis		*
2005		7	Knonau		*

Table 2: Animals per hectare farmland of farms with bovines

3 : Total Schweine (gve) der Gemeinde durch total LN der Betriebe mit Schweine

On the farm-land

JAHR	GEMEINDE_NUMMER		GEMEINDE_NAME	GVE_SCHWEINE_PRO_HA_LN_S	
2005		1	Aeugst am Albis		*
2005		2	Affoltern am Albis		*
2005		3	Bonstetten		*
2005		4	Hausen am Albis		*
2005		5	Hedingen		*
2005		6	Kappel am Albis		*
2005		7	Knonau		*

Table 3: Animals per hectare farmland of farms with pigs

2.2. The geographical information table (GG09_XYZ)

To add geographical information to the animal density data, I had to look for Information, which contain the Swiss communities with their geographical expansion. I found the following table (Table 4) with coordinate information of every Swiss community (Bundesamt für Statistik, 2013). The dataset is called GG09_XYZ. The first 7 entries in this table are shown in Table 4.

KT	KTNAME	BEZ	BEZNAME	GMDE	GMDENAME	XMIN	YMIN	XMAX	YMAX	ZKX	ZKY	ZKZ	ZMIN	ZMAX	ZMEAN	ZMED
1	Zürich	101	Affoltern	1	Aeugst am Albis	678123	234563	681154	238545	679300	235700	700	540	876	685	674
1	Zürich	101	Affoltern	2	Affoltern am Albis	673826	235207	678583	239338	676800	236800	493	448	746	529	502
1	Zürich	101	Affoltern	3	Bonstetten	675744	238998	679641	243158	677800	241000	544	504	702	581	581
1	Zürich	101	Affoltern	4	Hausen am Albis	680391	230183	686462	236411	682900	233100	610	529	902	674	653
1	Zürich	101	Affoltern	5	Hedingen	674863	238053	678972	241035	676400	239000	503	476	740	560	541
1	Zürich	101	Affoltern	6	Kappel am Albis	679049	229407	683714	232608	682400	231300	574	467	618	557	559
1	Zürich	101	Affoltern	7	Knonau	675592	229137	679581	233128	677500	230900	435	414	502	454	454

Table 4: Geographic information of the Swiss communities

According to the density table, columns five and six contain the community number and the name of the community. The first two columns concern the cantonal information. The first column contains the number of the canton (1-26), the second column contains the name of the canton. The third and fourth column contain the same information as the first two, except for the district and not the canton in which the community lies. All the other columns contain the geographical information. Columns seven to ten contain the minimal and the maximal point of the expansion of a community. The number given, is the expansion in the Swiss coordinate system LV95. XMIN means the most southern, YMIN the most western, XMAX the most northern and YMAX the most eastern point.

Columns eleven and twelve are very important. They show the central point of the area of a community. Column ZKX shows the expansion of this point in south-north direction, column ZKY shows the expansion of this point in west-east direction. Column thirteen, ZKZ, gives us the height of this point. Also about the height are the last four columns. Column fourteen shows the lowest height of the community area and column fifteen the highest. The last two columns show the mean height of the area and the median height of the area.

2.3. GG25 - digital community borders 2009

Because the GG09_XYZ allocates just the data as points, which means it won't be possible to illustrate for example the borders of each community, I was looking for another dataset, which allocats this data too. What I found was a dataset called GG25, also procided by the Swiss government (Bundesamt für Landestopografie, swisstopo, 2013). This record includes digital data of the borders of Swiss communities. The data I downloaded from the Toposhop webside was recorded in 2009, but it was the latest data I could receive.

The GG25 data is in form of a shapefile. In this format, it is easy to import the data to a simple GIS program and display the boarders of the Swiss communities.

A shapefile is not one file by itself, it is actually a collection of files. These files have the extensions .shp, .shx, .dbf or in our case also .prj. The actual shapefile which contains the geometry itself is the .shp, the shape format. Alone this file is incomplete; it needs also the .shx, the shape index format, which gives a positional index of the feature geometry and the .dbf, the attribute format, which contains the columnar attributes for each shape in the database. The .prj, which remains in our case, is the projection format. It includes the coordinate system and projection information for the data. It is a plain text file, describing the projection using well-known text format (OpenGeo, 2013).

If it is possible to Import this shapefile into the Swiss Feed Database, the GG09_XYZ is no longer needed, because the GG25 contains the same data, but with more functionality.

3. Import of the data into the database

3.1 Importing the density data

The animal density data were given to me in an excel worksheet. My goal was now to examine, which data is necessary to be in the Swiss Feed Database. I decided to take, for each cow, bovine and pig, the count of animals per hectare for the whole area of the community. I arranged a new table in excel with the columns number of community, cows per hectare, bovines per hectare, pigs per hectare and the year.

To integrate a table from a spreadsheet program, as excel is, into a PostgreSQL database there is a simple formula in SQL. Before one can import this table in PostgreSQL, one has to create a comma-separated values file (.csv). The created file takes the name d_density.csv With the following code (stackoverflow, 2013), the table could be imported as simple text file (.txt) to the database:

```
COPY d_koord FROM '/path_to_file/d_density.txt' DELIMITERS ',' CSV;
```

Now there exists a new table in the Swiss Feed Database with the name d_density.

3.2. Giving the density data a local value

To display the density data in google maps, they have to be referenced to a special point or area. The first idea was to reference every density data of a community to the exact centre of the area of this community. This solution would be possible with the GG09_XYZ dataset.

The GG25 dataset contains much more local and spatial information about a community. With this data, one does not need the centre of a community area, because the centre can easily be derived from the digital border data.

Another problem was, that the reference system, which Google Maps uses, is another one than the one in the data given by the Swiss government. In Switzerland, the local reference system LV95 is used. With the dataset GG25 it is easy to swich between different reference systems by changing a variable, to understand this transformation, I explained it in the first part of the paragraph.

3.2.1. Transformation of LV95 to WGS84

The transformation, I used here, is an approximate solution and has an accuracy of about 0.1 arc seconds. Given the LV95 coordinates of the Swiss reference system, one can calculate the coordinates of the WGS84 with simple mathematics. There are four main steps to get the right coordinates (Bundesamt für Landestopographie swisstopo, 2008).

In the first step, one has to take the y coordinate (East-West) and the x coordinate (North-South) to the civil system (Bern = 0/0), and then move it to the unit of 1000 km. For a point

700000 m (y)/ 100000 m (x) this means the new value for y is (700000-600000)/100000 = 0.1 and for x it would be (100000-200000)/100000 = -0,1.

In the second step ones has to calculate the longitude and latitude in the unit of 10000''. This means for y the new y:

$$2.6779094 + 4.728982 * y + 0.791484 * y * x + 0.1306 * y * x^2 - 0.0436 * y^3$$

For the example y = 700000 m, the new y is 3.14297976

For the new x it means:

$$16.9023892 + 3.238272 * x - 0.270978 * y^2 - 0.002528 * x^2 - 0.0447 * y^2 * x - 0.0140 * x^3$$

This means for our example x= 100000, the new x is 16.57588564

In the third and last step one just has to convert the latitude and longitude to the unit degrees [°]. One only has to take the x- and y- value, multiply it by 100 and divide the result by 36. This means, the coordinates 700000/100000 in the system LV95 are 8° 43' 49.80''/ 46° 02' 38.86'' in the system WGS84. In this form the data is now stored in the database.

3.2.2. Solution with GG25 – digital community borders

According to Paragraph 2.3 the GG25 dataset is optimal to give the density data a local value. The density can be assigned to each community in the dataset and so manipulated and displayed in a map. This could be in form of a choropleth map with one value for each community, but it also can be chosen a single representative point for one community as flag and aggregate the density values over the whole country. With the GG09_XYZ data, only a solution with the mean points of the communities would have been possible, so I skipped this idea. Consecutively the Import of the dataset GG25 into PostgreSQL is described.

3.2.2.1. PostGIS

The first task to import the GG25 dataset was, to look for a program which allows to import GIS data into PostgreSQL. Although it would be possible to display the polygon data with a simple GIS program like Jump (OpenJump, 2011) and extract all the data in a text file, the polygons would be too big to import it in the same way as d_density and d_koord and not all information would stay in the file.

The Solution for this is PostGIS, an extension to PostgreSQL which allows to handle spatial data. PostGIS is a software program that adds support for geographic objects in PostgreSQL (Dalluege, 2006). It was developed by the Canadian company Refractions Research and underlies the General Public License GNU. It is based on OpenGIS standards and there exists over 200 functions to make spatial analyses or to edit and manipulate geometry objects. PostGIS is downloadable from the main page of PostGIS (2013).

3.2.2.2. Import of a Shapefile in PostgreSQL

With the PostGIS extension installed it is possible to import a shapefile directly into Postgres.

With the following command (GISTutor, 2011) one can translate the shapefile (.shp) into Structured Query Language (.sql).

```
shp2pgsql -s <SRID> <shapefile> <tablename> <db_name> > filename.sql
```

The shp2pgsql command comes hereby with the installation of postGIS. The SRID is the Information of the reference system of the data. The SRID of the Swiss reference system is 21781, it is necessary to write this number in the code otherwise it won't be possible to swich from the Swiss system (LV95) to the one google uses (WGS84). If no SRID is set, it will be set to -1 which means that it is unknown.

The output of this command is a .sql file. This file is used to insert the shapefile into PostgreSQL. Therefore the following line of code is needed (GISTutor, 2011):

psql -d gisdatabase –U username –h hostname –p port -f filename.sql

Now the data should be in the specified database.

4. Results

After the import of the two datasets, there are three new tables needed in the Swiss Feed Database. The d_density, with the information about the animal density, the created table called d_geometry, which contains the geometry information of every community and a new tabble I called d_gemeinden, with which the two other tables can be connected. Those tables are shown in the first part of this paragraph.

In the second part, the functionality of the geometry type is shortly demonstrated and a few sample queries for this new tables are given.

4.1. New SQL-Scheme in the Swiss Feed Database

Consecutively, the three new tabes in the Swis Feed Database are presented and explained.

4.1.1. d_density

gem_nr	year	animal	density
1	2005	cow	*
1	2005	rind	*
1	2005	pig	*
1	2006	cow	*
1	2006	rind	*
1	2006	pig	*
1	2007	cow	*
1	2007	rind	*
1	2007	pig	*

Table 5: First 9 columns of the table d_density

This table simply holds the animal density information. Primary key here is the gem_nr, the year and the animal all together. The gem_nr is a statistic number of the Bundesamt für Statistik. It is unique for every community in Switzerland. This column is in the table, because I got the data in this way (in the Excel-Sheet) and it is easier to store a number for a community, than a name. The column year shows the year for which the animal density was measured. The animal shows the animal, either pig, cow or bovine and the density gives the actual animal density for a community and a specific animal.
The number of the communities is also set as foreign key to the table d_gemeinden. As one can see, the data is sorted according to its ascending values. For every gem_nr, there exist three times the same year, one for every animal.

4.1.2. d_geometry

id	gemnr	gemteil	gem_name	the_geom
1	3851	0	Davos	0106000020155500000100000001030000....
2	6031	0	Bagnes	0106000020155500000100000001030000....
3	6252	0	Anniviers	0106000020155500000100000001030000....
4	6300	0	Zermatt	0106000020155500000100000001030000....
5	6083	0	Evolène	0106000020155500000100000001030000....
6	3746	0	Zernez	0106000020155500000100000001030000....
7	5049	0	Blenio	0106000020155500000100000001030000....
8	782	0	Guttannen	0106000020155500000100000001030000....
9	3847	0	Val Müstair	0106000020155500000100000001030000....

Table 6: First 9 columns of the table d_geometry

In the d_geometry table, not all information of the imported shapefile from GG25 is maintained in the database. Especially all the information, which can be derived from the the_geom column. Those are for example special columns to store the area of a community or the perimeter.

Also columns like the objectID, the object origin, the year of change and the seenr were skipped. This information is useless for our task and makes working with the table more difficult. Further information about the canton or the district were skipped too, because the focus lies here in the communities. If this information is yet desired, one can join the data

with the table dd_places from the Swiss Feed Database. There information about the canton is included.

In the final table, the ID is the primary key of the relation. The gem_nr is set as a foreign key to the table d_gemeinden. The gem_name, the name of each community, has to stay in the table, to make connections to other tables and to display them. The column gemteil couldn't be deleted because otherwise data of communities with a splitted area were lost. This column is important for all the exclaves of a community.

The most important feature of this table is the column the_geom. From this column all the important data can be derived with special PostGIS queries. The the_geom column is stored in a sort of hexadecimal numbers. With the commands following in the next part, it is possible to display the data as text and then to manipulate it in nearly every way we want.

Additionally to the d_geometry table a new entry in a table called geometry_column was generated. This table shows some kind of description for geometric tables. The most important rows are f_table_name, f_geometry_column, coord_dimension, srid and type. The name of the table is in our case d_geometry. f_geometry_column shows the name of the column in d_geometry, in which the geometry information lies. The coordinate dimension is 2. This means our geometry information is all in the second dimension. The SRID is the number for the Swiss reference system. It was set within the import of the shapefile. In the table it is 21781. In the end remains the column type. For d_polly the type is multipolygon. This means that the geometry information, the boundaries of the Swiss communities, are stored as multipolygon in the database.

4.1.3. d_gemeinden

gem_nr	gem_name
1	Aeugst am Albis
2	Affoltern am Albis
3	Bonstetten
4	Hausen am Albis
5	Hedingen
6	Kappel am Albis
7	Knonau
8	Maschwanden
9	Mettmenstetten

Table 7: First 9 columns of the table d_gemeinden

To normalize the two tables, I had to create a third one. This third one is called d_gemeinden and it holds the number and the according name of the communities. The primary key here is the column gem_nr, to which the other two tables are connected. In Figure 1, one can see the relation between these three tables.

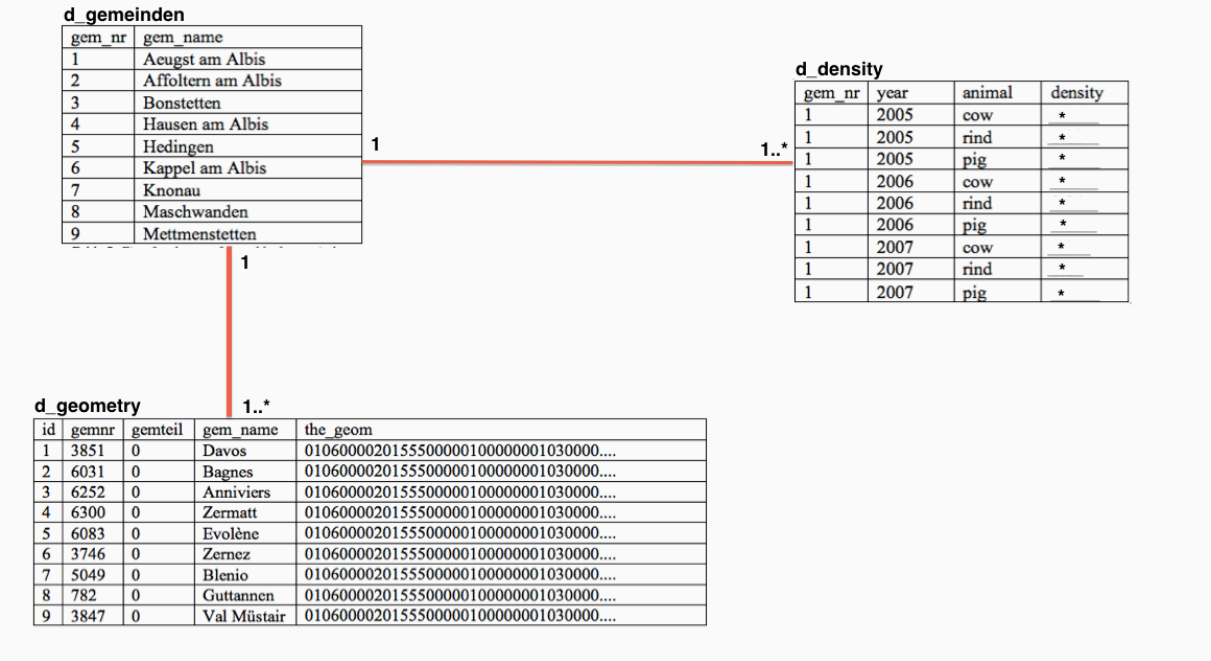

Figure 1: Relation between the new tables

4.2. Use of the new tables

In this part it is shown, how the new tables can be used. First I give a short overview over different functions in PostGIS and show, how the geometry data type can be manipulated. In the second part, I define five different queries and their SQL statements, which can now be answered with the new defined tables.

4.2.1. Functionality of the geometry type in PostGIS

In this part the most important SQL commands for the the_geom column in our table are given. They are all taken from the PostGIS References (2013).

Change of the hexadecimal numbers to text format

To work properly with the the_geom column one has to change the format to text format. In this way our data is shown in multipolygon format. The formula to change it that way is:

ST_AsText(geometry)

The format of our data, after applying the formula is for example:
MULTIPOLYGON(((552529 141944 ,552818 141660,552688 141808,552640 141850,552575 141923,552543 141938,552529 141944)))

Change of the Multipolygon to Polygon format

Multipolygon consist of several, grouped together polygons. In Figure 1 there are examples how multipolygon look.

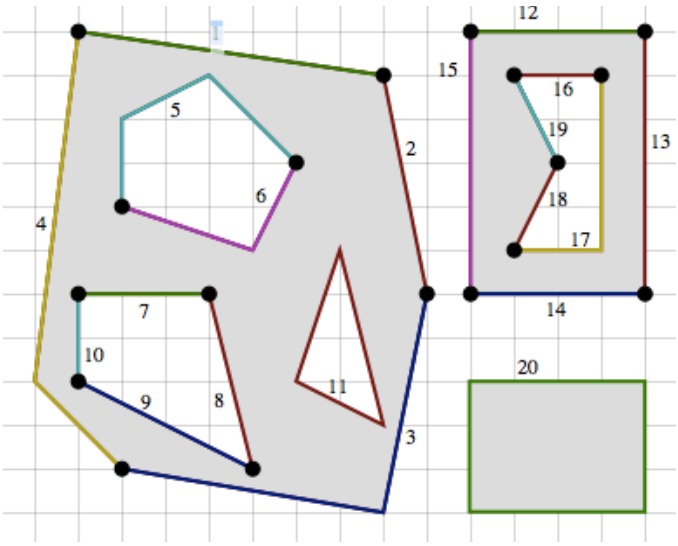

Figure 2: Different Multipolygons

In PostGIS there exists a formula to convert the multipolygons to polygons. It would be easier to work with polygons, than with multipolygons, although data is lost. The formula therefor is:

ST_geometryn(geometry,1)

Changing the reference system

To transform the reference system from one into another, for example from LV95 to WGS84, one has to know the SRID of the reference systems. To get the SRID of the reference system, the data already is, the following formula should be used:

ST_SRID(geometry)

This returns the integer SRID number of the spatial reference system of the geometry. To transform this reference system to another, the following formula is used:

ST_Transform(geometry g1, integer srid)

This makes transformations to other reference systems very easy. No long conversion has to be done like in part 3.2.1.1.

Displaying the area of the communities

To get the Area of a community, the formula

ST_Area(geometry)

can be used. It returns the area of the geometry, if it is a polygon or multipolygon.

Displaying the centre point of the communities

Similar to the area, also the centre point of the communities can be displayd. This is also the point, why the table d_koord was no longer needed. The formula to get the centroid of the geometry as a point is:

ST_Centroid(geometry)

This functionality is exactly the reason, why the GG09_XYZ was no longer needed. With that function, one can easily compute the center point of a Swiss community and the result is the same points as in the dataset GG09_XYZ.

Displaying length of the (multi-)polygon

The length of the polygon is the same value as the row perimeter in the d_polly displayd. Because of the existence of this formula, the column could be deleted. The formula therefor is:

ST_Length(geometry)

Examine if geometries touch eachother

This function shows if two geometries, in our case two polygons, spatially touch eachother. The function returns true if the do. This function can be usefull to see if two polygons are neighbors:

ST_Touches(geometry, geometry)

Examine if a geometry contains another

To examine if a geometry A spatially contains a geometry B this function is used. In our case this formula can be usefull to see if a community contains enclaves in its area.

ST_Contains(geometry A, geometry B)

This formula also returns true, if geometry A contains geometry B.

Examine if a geometry intersects another

To examine if a geometry A intersects a geometry B this function is used. This function can be usefull, if we have a defined polygon over some communities in Switzerland, to heve the intersection of it.

ST_Intersects(geometry A, geometry B)

This formula returns true, if geometry A intersects geometry B.

5.2.2. Five specific queries

1. What is the total number of a certain animal in a certain Swiss community?

```
select gem_name, year, animal, st_area(the_geom)/10000*density as amount_of_animals
        from d_geometry join d_density on gemnr = gem_nr;
```

After joining the d_density table with the d_geometry table on the number of communities, all the necessary information is in one table. To get the amoung of animals, one has to select the area and multiply it by the animal density. This new column is renamed as amount_of_animals. To get the right measuring unit, one has to divide the area by 10000.

An example would be: How many cows existed in 2008 in the village Rafz?

```
select gem_name, year, animal, st_area(the_geom)/10000*density as amount_of_animals
        from d_geometry join d_density on gemnr = gem_nr
                where gem_name = 'Rafz' and animal = 'cow' and year = 2008;
```

The result to this specific query is:

gem_name	year	animal	amount_of_animals
Rafz	2008	cow	*

Table 8: Result of the first query

2. Given a specivic polygon area, what is the animal density of this area?

```
create view schnitt_gemeinden as
        select gem_name, gemnr, the_geom
                from d_geometry
                        where st_intersects(st_setSRID(st_geometryfromtext
                        ('POLYGON((X1 Y1, X2 Y1, X2 Y2, X1 Y2, X1 Y1))'), 21781),
                        the_geom) = true;

 create view area_of_schnitt_gemeinden as
        select gemnr, st_area(st_intersection(st_setSRID (st_geometryfromtext
        ('POLYGON((X1 Y1, X2 Y1, X2 Y2, X1 Y2, X1 Y1))'), 21781), the_geom)) as
        area_gem
                from schnitt_gemeinden;

create view schnitt_density as
        select gemnr, area_gem/ st_area(st_setSRID (st_geometryfromtext ('POLYGON((X1
        Y1, X2 Y1, X2 Y2, X1 Y2, X1 Y1))'), 21781))* density  as schnitt_dens
                from area_of_schnittgemeinden, d_density
                        where gemnr = gem_nr and animal = 'XXX' and year = XXXX;

select sum(schnitt_dens) as density_in_polygon
        from schnitt_density;
```

To make this query clearer I decided to create views. In the first view, there will be all communities selected, which intersect with a certain polygon. The st_setSRID statement is used to have for the polygon and the the_geom column the same SRID (21781), the st_geometryfromtext is used to read the the_geom column. In the select statement, the name, the number and the geometry of the communities are selected. This view is called schnitt_gemeinden and shows all the communities, which are intersected by the given polygon.

The second view, area_of_schnitt_gemeinden, gives selects the number of the intersected communities and the exact area of the intersected part of the community area.

The third view computes now the density for this area part. Here also the community number is selected. In addition the area of the intersected part of the community is divided by the area of the given polygon and multiplied by the desired animal density. The selection comes from a join between the view area_of_schnitt_gemeinden and the d_density table.

In the end remains just one selection from this new view. The sum of the column schnitt_dens of the view schnitt_density is made and gives us the result to the asked question. This column is now the result for the density in the given polygon.

An example would be: What is the exact cow – density in the year 2009 over the Polygon (600000 200000, 602000 200000, 602000 197000, 600000 197000, 600000 200000) ?

```
create view schnitt_gemeinden as
    select gem_name, gemnr, the_geom
        from d_geometry  where st_intersects(st_setSRID
        (st_geometryfromtext ('POLYGON((600000 200000, 602000 200000, 602000
        197000, 600000 197000, 600000 200000))'), 21781), the_geom) = true;

create view area_of_schnitt_gemeinden as
    select gemnr, st_area(st_intersection(st_setSRID (st_geometryfromtext
    ('POLYGON((600000 200000,  602000 200000, 602000 197000, 600000 197000,
    600000 200000))'), 21781), the_geom)) as area_gem
        from schnitt_gemeinden

create view schnitt_density as
    select gemnr, area_gem/ st_area(st_setSRID (st_geometryfromtext
    ('POLYGON((600000 200000,  602000 200000, 602000 197000, 600000 197000,
    600000 200000))'), 21781))* density  as schnitt_dens
        from area_of_schnittgemeinden, d_density
            where gemnr = gem_nr and animal = 'cow' and year = 2009;

select sum(schnitt_dens) as density_in_polygon
    from schnitt_density;
```

The result to this specific query is:

density_in_polygon
*

Table 9: Result of the second query

3. In which Swiss communities did the animal density increase? In which did it decrease between two meassurement years?

```
Select gem_nr
        from d_density
                where animal = 'XXX' and year = XXXX  and not exists
                        (select new_tab.year, new_tab.density
                        from d_density new_tab
                                where new_tab.year = YYYY
                                and new_tab.animal = 'XXX' and d_density.gem_nr =
                                new_tab.gem_nr and new_tab.density <= d_density.density)
                                        order by gem_nr asc;
```

In this query the number of the communities is selected from the d_density table, where no density value exists for a given year and a given animal, which is smaller, than another density value of the same animal, but a different year. The number of communities should be ordered asceding.

An example for this query would be: What is the number of the communities, in which the cow-density increased between the years 2008 and 2009?

```
Select gem_nr
        from d_density
                where animal = 'cow' and year = 2009  and not exists
                        (select new_tab.year, new_tab.density
                        from d_density new_tab
                                where new_tab.year = 2008
                                and new_tab.animal = 'cow' and d_density.gem_nr =
                                new_tab.gem_nr and new_tab.density <= d_density.density)
                                        order by gem_nr asc;
```

gem_nr
*
*
*
*
*
*
*
*
*
*

Table 10: First 10 Results of the third query

4. Which community has the highest animal density?

```
select gem_name, density
    from d_density, d_geometry
        where gemnr = gem_nr and animal = 'XXX' and year = XXXX and density =
        (select max(density)
            from d_density
                where animal = 'XXX' and year = XXXX);
```

The name and the density of a Swiss community are selected from a join between d_density and d_geometry on the number of the community. Also a specific year and an animal should be selected there. The density should be equal to the maximum density for this specific animal and year.

An example for this query would be: Which community has the highest cow-density in the year 2009?

```
select gem_name, density
    from d_density, d_geometry
        where gemnr = gem_nr and animal = 'cow' and year = 2009 and density =
        (select max(density)
            from d_density
                where animal = 'cow' and year = 2009);
```

The result of this query is:

gem_name	density
Nidau	*

Table 11: Result of the fourth query

5. What is the over-all animal density in the last years (2005-2009) in a specific canton?

```
create view v1 as
    select d_density.gem_nr, year, sum(density)as d1
        from d_density, dd_places, d_gemeinden
            where  city_long = gem_name and d_density.gem_nr =
            d_gemeinden.gem_nr and canton_abbreviation = 'XX'
                group by d_density.gem_nr, year
                    order by d_density.gem_nr asc;
```

```
create view v2 as
        select gem_nr, sum(d1)/count(*) as over_all_density
                from v1
                        group by gem_nr;

select sum(over_all_density)/count(*) as over_all_in_canton from v2;
```

Because of reasons of clarity, I created here also two different views. The first view v1 selects the number of communities and the sum of the densities of the three animals per year in a certain canton. This means, there remains one density value for each community per year. The canton can be chosen by joining the density table with the table d_gemeinden and the dd_places table of the Swiss Feed Database, which actually holds the information about the cantons. The results are grouped according to the community number.

In the second view, the number of communities in a canton and the sum of their densities of all years divided by the six different years is selected from the first view and the result is grouped by the community number.
In the end, one had just to compute the average of all communities in a certain canton.

There is one problem remaining: There are years or animals, where the value is 0. All these zero-values are now also part of this computation. To involve this thought, the query has to be specified.

An example for the query would be: What is the over-all animal density in the last years (2005-2009) in the canton Zurich?

```
create view v1 as
        select d_density.gem_nr, year, sum(density)as d1
                from d_density, dd_places, d_gemeinden
                        where  city_long = gem_name and d_density.gem_nr =
                        d_gemeinden.gem_nr and canton_abbreviation = 'ZH'
                                group by d_density.gem_nr, year
                                        order by d_density.gem_nr asc;

create view v2 as
        select gem_nr, sum(d1)/count(*) as over_all_density
                from v1
                        group by gem_nr;

select sum(over_all_density)/count(*) as over_all_in_canton from v2;
```

The result of this query is as follows:

over_all_in_canton
*

Table 12: Result for over all animal density in canton Zurich

5. Conclusion and further work

The new two tables offer more flexibility to the database. Now it is not only possible to determine, where which nutrients occur, but also, where the animal density is high or low. This offers space for analysis, for example in which context nutrients and animal density are. Through the import and the creation of this two new tables it is also possible to make researches, for farmers and scientists, about the change in animal density in Switzerland and the distribution of animals all over Switzerland.
A next step in this research field could be the inclusion of the two tables in the user interface of the Swiss Feed Database and to make to display maps for queries about the animal density. A choropleth map of the animal density for every Swiss community would just be the beginning for this.

But there are also other things which should be overthougth in future work. In Switzerland, many fusions of communities took place in the last year. For the correctness, one should check all the data in the database, if the latest community fusions are already considered. Therefore, one has to import the latest version and examine the accurate changes in the two tables, every time a new dataset is made by the Swiss government.
Another point, which one can do as further work, is to complete the animals also with the French and the German name.

References

Bundesamt für Landestopografie, swisstopo, 2013: http://www.toposhop.admin.ch/de/shop/products/landscape/gg25_1 (03.12.2012)

Bundesamt für Landestopografie, swisstopo, 2008: Formeln und Konstanten fpr die Berechung der SChweizerischen schiefachsigen Zylinderprojektion und der Transformation zwischen Koordinatensystemen. Eidgenössisches Departement für Verteidigung, Bevölkerungsschutz und Sport VBS.

Bundesamt für Statistik, 2013: http://www.bfs.admin.ch/bfs/portal/de/index/dienstleistungen/geostat/datenbeschreibung/generalisierte_gemeindegrenzen.html (15.11.2012)

Dalluege, U. 2006: PostGis Tutorial (Grundlagen). HCU Hamburg Dept. Geomatik

GISTutor, 2011: http://www.gistutor.com/postgresqlpostgis/4/18-importing-shapefile-gis-data-into-postgresql.html (13.01.2013)

OpenGeo, 2013: http://workshops.opengeo.org/postgis-intro/loading_data.html (07.01.2013)

OpenJump, 2011: http://www.openjump.org (13.01.2013)

PostGIS, 2013: http://postgis.net/install (10.01.2013)

PostGIS References, 2013: http://postgis.net/docs/manual-1.3/ch06.html (13.01.2013)

Stackoverflow, 2013: http://stackoverflow.com/questions/2987433/how-to-import-csv-file-data-into-a-postgres-table (07.01.2013)

Swiss Feed Database, 2013: http://www.feed-alp.admin.ch/feedbase/ (11.11.2012)

Universität Zürich, 2013: http://www.ifi.uzh.ch/dbtg/research/sfdb.html (11.11.2012)

* This data can not be shown because it is not yet published. Rights of the data belong to agroscope