WORKFLOW AUTOMATION WITH POSTGIS TOPOLOGY

Anders Møller Jacobsen
Sweco
a.mller.jacobsen@gmail.com

Traditionally, a process of changing district borders can be a long and cumbersome process involving several layers of political decision-making. In a way, the GIS-coordinator is caught in the middle, supporting the technical changes and hence being a necessary resource when wanting to see the consequences of a revised district layout.
By using PostGIS topology and database triggers the workflow can be optimised and the setup automated, ultimately presenting the end-user with a fluent and fully automatic workflow without any complicated data interactions allowing the end-users to make the changes themselves. This moves the responsibility from the GIS department out in the organisation.
This paper presents the findings of creating this setup with the case of school districts, and creates a foundation for an automated editing process to be integrated with a webGIS application.

Keywords: Topology, Triggers, PostGIS, WebGIS

Changing school district borders in a Danish municipality can be a long and cumbersome process since it involves several departments and different layers of decision-making of both technical and political nature. The changes are nowadays made in desktop GIS software and involve several statistics calculations to support the argument for district changes. By automating the geometry and statistics calculations in the database and moving to a webGIS application, the elaborate work can be changed to be easy, efficient and less

error-prone for the end-user. Moreover, the changes can be made by an end-user without specialised GIS knowledge, moving the responsibility of the districts editing from the GIS department to the school department.

The findings in this article originates from my Master's Thesis where the main goal was to develop a dynamic polygon statistics tool with the specific case of editing school districts. To work flawlessly with a webGIS editing environment, a vital part of the setup is configuration of a database setup with geospatial topology and triggers in a PostGIS database. This article will solely focus on the database setup with editing capabilities in QGIS (QGIS, 2016) and therefore not the specific webGIS application integration made in my thesis due to the user-specific nature of the webGIS software.

METHODOLOGY

The goal of the original setup was to provide an easy and efficient solution for the school department in editing school district borders. The vital part of this setup was developing an automated database setup with geometry and statistic calculations. Topology automates the geometry calculations by creating a ruleset where borders are shared to prevent overlaps and gaps in the geometry (Obe & Hsu, 2015). By having a topology ruleset combined with database triggers that calculates statistics of each student every time the geometry is updated, it can provide a solution automating the data workflow and create an easy editing environment for the end-user.

PostGIS topology

The relations and the hierarchy in topology is built upon the OGC simple feature standard with points, linestrings and polygons (Open Geospatial Consortium (OGC), 2011). In general, two representations of vector data exist in PostGIS. The traditional geometry model is where each feature is controlled as a separate entity; polygon borders are duplicated and possibly overlap or have gaps in between, whereas the topology model shares borders to prevent overlaps and gaps in your geometry (Obe & Hsu, 2015), essentially what you wish in a districts layer that has to maintain full area coverage.

The topology model goes one step back from traditional geometry in GIS since it frees the geometries from the dependency of a coordinates system; this is due to its origin in graph theory (Obe & Hsu, 2015; Yeung & Hall, 2007). Therefore, a different terminology is also used for the primitives; point is a node, linestring is an edge and polygon is

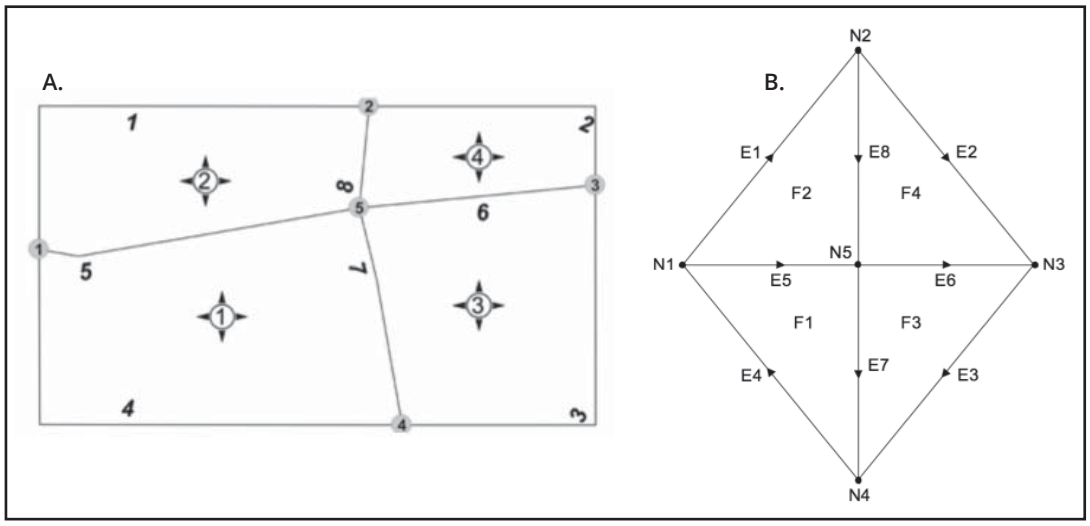

Figure 1. A topology network (A) and a simplified model (B).

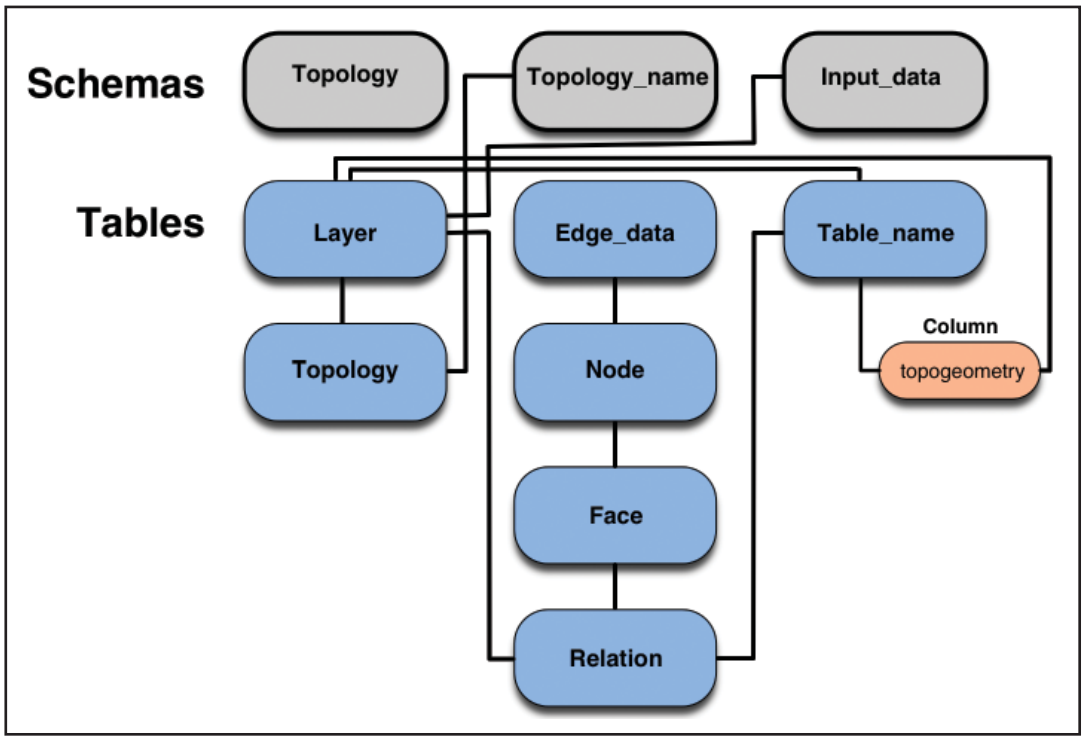

Figure 2. The database design and relations of a topology ruleset created on one table (Table_name) in the schema Input_data

a face. The main difference of topology to 'traditional geometry' is that it is not concerned of describing the geometry itself, but how the geometry interrelates. Moreover, it reduces storage requirements because boundaries are shared, and it is aesthetically more pleasing and correct since data is freed from digitising and editing errors (Yeung & Hall, 2007). It also enables more sophisticated spatial analyses because it eliminates the need to develop programming routines to calculate these spatial relationships every time a new dataset is used (Yeung & Hall, 2007). On figure 1a a topology network can be seen next to a simplified model of the same topology (figure 1b), where Nx refers to node x, Ex refers to edge x and Fx refers to face x in the simplified model (right). The topology network (figure 1a) looks identical to what you can define as 'traditional' geometry but has to be thought as working as a network of nodes and edges.

In a database setup, sorting is done in schemas. Each database can contain multiple schemas, which in turn contain tables, functions, operators and more. Schemas are used for reasons of logical grouping of data and to avoid third-party extensions to collide with existing data (The PostgreSQL Global Development Group, 2016). This is especially visible with the use of topology where two new schemas are created besides the schema consisting your input data. The table for which the topology ruleset is created gets a new geometry column called topogeometry, containing a range of information. It could for example look like this: (1,6,2,3) where 1 is the topology_id in the topology. topology table, 6 is the layer_id in the topology. layer table, 2 is a unique row id within the respective table (table_name), and 3 is the data type in this case (multi)polygon. The full database design of a topology setup with relations can be seen in figure 2 and can be further explored in The PostGIS Development Group (2016).

Now the methods for adding the topology ruleset to a traditional geometry table will briefly be explained. A topology ruleset is created on a per table basis. After the extension has been installed in the database, a new schema (Topology_name – figure 2) for the topology is created automatically. By adding a new geometry column, topogeometry, specifically for the topology to an existing table, the topology is created by converting the existing geometry to the new topogeometry column. In this step, the ruleset is created with a range of tables placed in the topology schema (Topology_name – figure 2) containing the different primitives (nodes, edges and faces). Moreover, another topology-related schema (Topology – figure 2) is added to the database. After successful creation of the topology ruleset for a table, it is possible to use QGIS for viewing and editing it. To view data, the 'DBManager – Topoviewer' is used. When editing it has to be done in the edge_data table to work properly. After the edits has been saved the topology ruleset is updated to do so and is reflected in all tables.

After converting the existing geometry to topogeometry, it will most likely have geometric errors. A range of steps can solve these issues. First, it is checked whether some districts consist of more than one face. Some places tiny faces might have been created in the conversion process. Secondly, a function that removes edges that form these small faces is executed. If the edge splits two faces, the original face is destroyed and a new face that is the union of these is created. This function only removes edges that do not affect the geometric definition of your districts, which means this function will not destroy your topology. If these two functions cannot remove all the small faces, they have to be located manually in for example QGIS and removed. As of now (Obe & Hsu, 2015) a fully automatic workflow to remove errors doesn't exists.

PostGIS triggers

A trigger is a specification or function that the database will automatically run whenever a certain type of operation is performed. Triggers are attached to tables or views, and can be executed before or after an INSERT, UPDATE or DELETE operation either once per modified row, or once per SQL statement (The PostgreSQL Global Development Group, 2016). The basis of creating triggers is to automate processes that otherwise the user has to execute every time changes are made.

In the original setup the trigger calculated statistics about the student every time the geometry of the district borders was edited. The statistics can be anything, in this specific case an age, gender and ethnicity was available for each student in the research area. The automated function (trigger) calculated a total count of students, number of male/females, age- and ethnicity distribution per district. These statistics calculated on-the-fly could then be used by the school department in the decision process of editing the district borders. Since the edits were done in a webGIS environment, the end-user would not have access to the database so an automated calculation process in form of a database trigger was necessary to use in this setup.

RESULTS AND DISCUSSION

The final result of a working setup is simply a working topology ruleset and automatic statistics calculations for the students. For the end-user, the editing process will be experienced as smooth and easy and without any data interaction, other than editing the district borders; see figure 3 for the result of a border edit and the automatically calculated statistics done by the trigger. In the following sections are some of the findings briefly presented and discussed.

Automating feature editing

The trigger to recalculate the statistics on the students was developed and explored throughout the project, and optimised several times during the process. In the final testing, it was found that the statistics recalculations were up to 1 minute long due to the trigger, which is excessively long for a

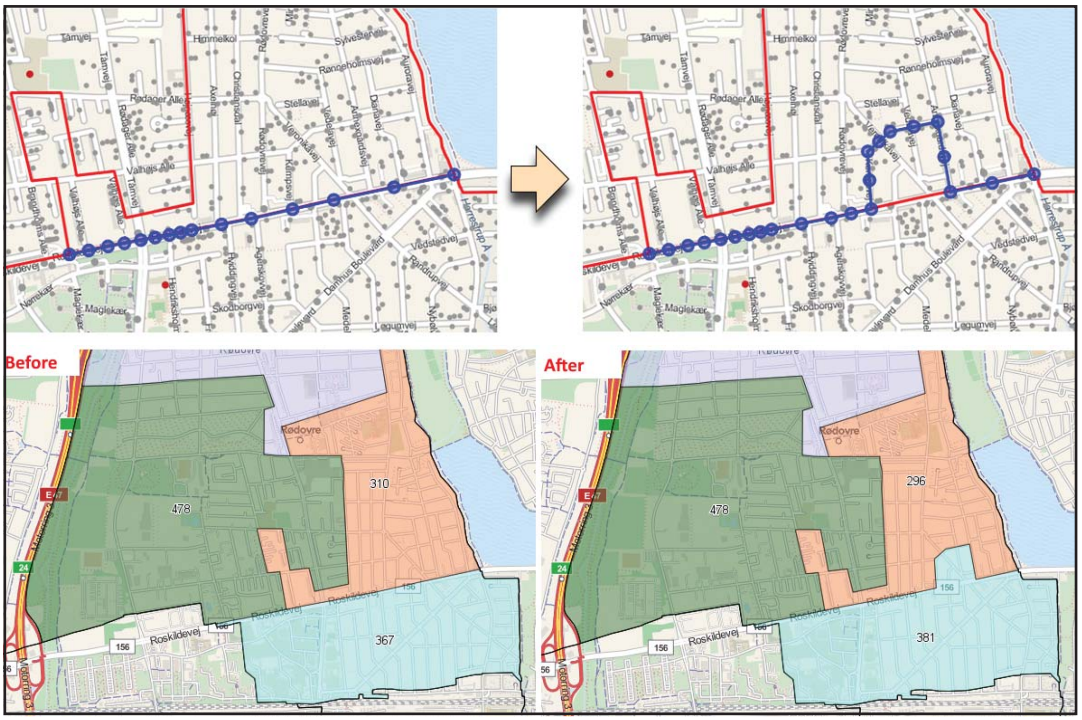

Figure 3. The editing workflow of changing a districts border, and the districts before and after the edits. Note the number of students counted per district has been updated automatically by the trigger.

final product. Different changes were tried, and one attempt where the whole code structure was changed made it possible to do the calculations in just around 15 seconds. This is still long, but the trigger is also doing an overlay analysis between students and school districts, and calculates 11 columns every time a change is saved. It was also tested to disable the trigger, and this made the saving of districts changes instant. The triggers' purpose is solely used in this project to reflect realistic data in the decision-making, so without that, the final setup would work much faster.

When building the topology ruleset some processes were a bit more complicated and locating and fixing all errors could unfortunately not be completed automatically, as mentioned earlier. These leftover errors are all related to when converting an existing 'traditional' polygon feature dataset into topology geometry (topogeometry). The different steps to fix these errors (as of now manually) can maybe be automated in a script but the problem is the outcome of each step that sometimes requires a 'human decision' about the result data. By this, it is meant that some errors will change the geometric definition if corrected automatic, and therefore is not allowed with the existing functions. Whether these issues are data-related or simply lies in the constraints in the topology cannot be ensured fully. Worth mentioning is though that this process of fixing errors is a one-time thing that only has to be completed when importing new data, making is less painful for small datasets like the one used in this project. With larger datasets, a script workflow has to be considered, and hopefully an automatic process of fixing errors will be available in a future PostGIS version.

Future developments

In the current version, the editing options of the topology are limited mainly due to the strict constraint the topology ruleset applies to a table.

From the beginning, it was thought that editing options would be similar to those for traditional features. One problem experienced was the editing of the topology nodes, that when edited would 'destroy' the topology in the surrounding districts. This was observed as; the districts categorised in colours would lose their colour, and removed from the tables in the topology_name schema (figure 2). Therefore, the editing of topology nodes, is not working currently when edited with for example QGIS and has yet to be solved with either a more advanced QGIS plugin or changes in the PostGIS topology definition.

Other editing situation such as adding a new district or combining existing districts also came up during the development. As of now, these options are also not possible and can only be done by removing and reapplying the topology ruleset. Basically, that removes the purpose of topology, since you would have to do the edits in a 'traditional' geometry setup. These options will also have to be solved in an appropriate way in a future version.

CONCLUDING THOUGHTS

The vital part in the technical setup of editing school districts through a webGIS application has been the database configuration with geospatial topology and trigger automation. Topology automates the geometry calculations by creating a ruleset where a polygon feature's borders are shared to prevent overlaps and gaps in the geometry. Triggers help to calculate statistics every time the geometry is edited. With these combined, the end-user will be presented with a fluent and fully automatic workflow without any complicated data interactions. This setup will moreover limit the time spend on communication between the GIS and school department when changes are to be made, as well as making the process less error-prone due to the automatic data workflow. As of now, the possibilities with topology in PostGIS are great and can certainly provide a setup aiding a better data workflow, although there is still room for improvements when editing.

REFERENCER

- Obe, R. O., & Hsu, L. S. (2015). *PostGIS In Action* (2nd ed.). Shelter Island: Manning Publications Co.
- Open Geospatial Consortium (OGC). (2011). *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture* (1.2.1 ed.): John R. Herring.
- QGIS. (2016). QGIS - *A Free and Open Source Geographic Information System*, at http://qgis.org, [accessed September 25th 2016]
- The PostGIS Development Group. (2016). *PostGIS Manual,* at http://postgis.net/docs/, [accessed September 25th 2016]
- The PostgreSQL Global Development Group. (2016). *PostgreSQL 9.2.18 Documentation,* at http://www.postgresql.org/docs/9.2/static/index.html, [accessed September 25th 2016]
- Yeung, A. K. W., & Hall, G. B. (2007). *Spatial Database Systems: Design, Implementation and Project Management.* Dordrecht, The Netherlands: Springer.