Paper to be presented at JEC-GI'97, Vienna,16-18 April 1997.

Geographic Query Tool

Peter van Oosterom and Bart Maessen
Cadastre Netherlands
P.O. Box 9046, 7300 GH Apeldoorn, The Netherlands.
Email: oosterom@kadaster.nl, maessen@kadaster.nl

In this paper the requirements, design and lessons learned from a GIS specialized as a geographic query tool for a relational database are presented. The integration of geometric and administrative data is realized by using views based on joins without changing the original data models. Map overviews are created by the spatial or administrative aggregation of the data. Several generic issues are discussed: architecture, coupling of (non-)spatial data from different sources, thematic maps, visualization of time and aggregation over space and time.

1 Introduction

The *Geographic Query Tool*-project is part of a multi-year process of technological renewal and improvement of the information systems at the Dutch Cadastre. This process is called IT-2000 and focuses on developing an architectural design of the new cadastral systems based on a client-server architecture; see Section 2. In this new architecture, large scale topographic and cadastral data will be stored and maintained in CA-OpenIngres, with OME/SOL (Object Management Extension/Spatial Object Library)[2, 9]. Legal and other administrative information related to parcels are stored in another database. This database will be referred to as the *administrative* database in contrast to the *geometric* database, which contains the topographic and cadastral map data. Starting from this year the database will keep track of all changes over time, that is, it is a spatio-temporal database. Every object is extended with two additional attributes: `tmin` and `tmax`. The objects are starting from and including `tmin` and remain valid until and excluding `tmax`. Current objects get a special `tmax` value: `TMAX_VALUE`, indicating they are valid now.

A prototype of the geographic query tool has been developed which tests the issues described in this paper. A test database has been filled with data

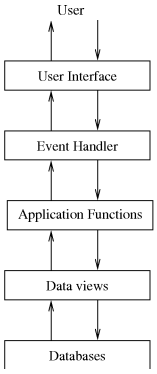

Figure 1: CSA

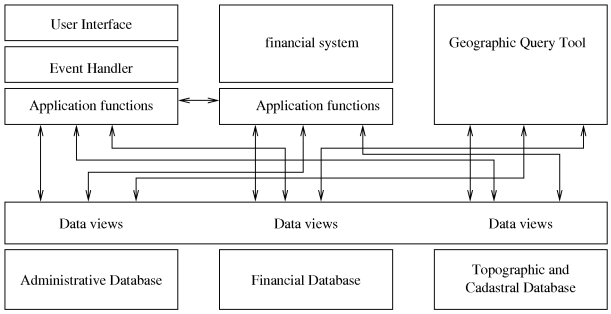

Figure 2: Integration of applications

from the province of Utrecht; geometric data: 1,100,000 parcel boundaries, 320,000 parcels, 2,400,000 topographic lines, 480,000 cartographic symbols and 200,000 text labels; administrative data: 460,000 owners, 560,000 addresses, 440,000 administrative parcel records. The total size of the test database is about 3 Gb. Spatial clustering and indexing are required in order to make interactive querying possible. The prototype is based on GEO++ [5, 8, 10] and CA-OpenIngres. The integration and querying of administrative data and geometric data is discussed in Section 3. Visualizing spatio-temporal data is the topic of Section 4. Finally, conclusions and future work can be found in the last section.

2 System Architecture

The Cadastre has chosen for a client-server architecture (CSA)[3]; see Figure 1. In the future all the Cadastral systems are integrated with each other; see Figure 2. The CSA allows this to be done in a flexible and open manner, which allows sharing of components between (sub) systems. There will be a mix of bought packages and developed systems. Data views have the ability to combine two or more physical databases and make them look like one integrated database.

Because of the CSA, a system such as a query tool must be open, i.e. it can not operate on a local vendor specific data format. Most of the current GIS-systems do not support this open database approach. The traditional approach is that data has to be read into the GIS which stores it in a local proprietary format. However, there are GISs, which have the capability to access an external database system. Usually this means that the geometric information still has to be stored locally, but with help of a

key, the corresponding administrative data can be found. This is still not good enough, since complete integration of geometric and administrative data is the most efficient and consistent data management architecture. A geographic query tool must function as a fronted to the database backend. The geographic query tool assists the user in posing questions through a friendly user interface. Integration is realized at the level of the data view layer: the query tool generates ANSI SQL dynamically and processes the responses of the database system. GEO++ is a GIS which is designed to deal with this.

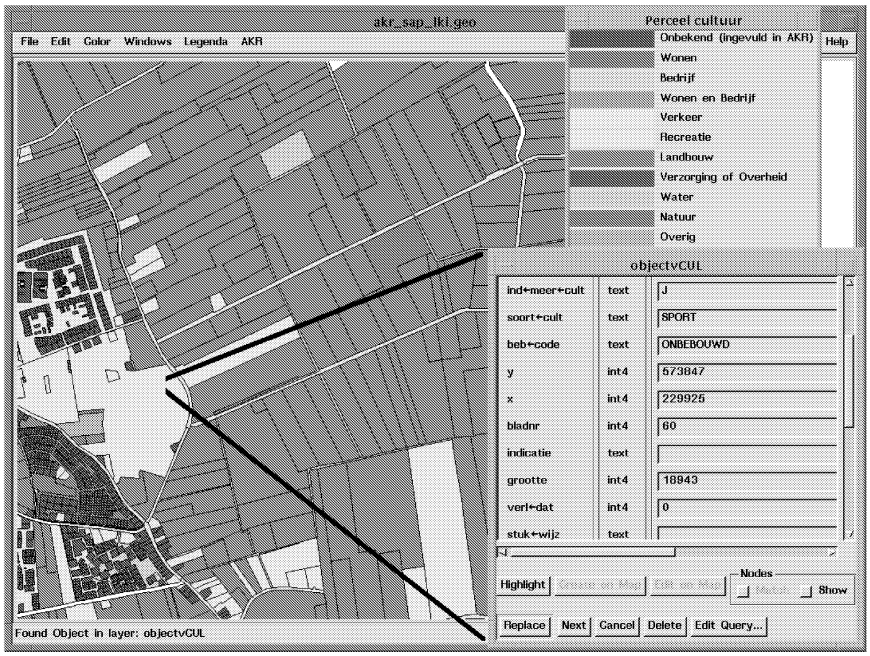

Figure 3: Land use of parcels (right: clicking on a parcel)

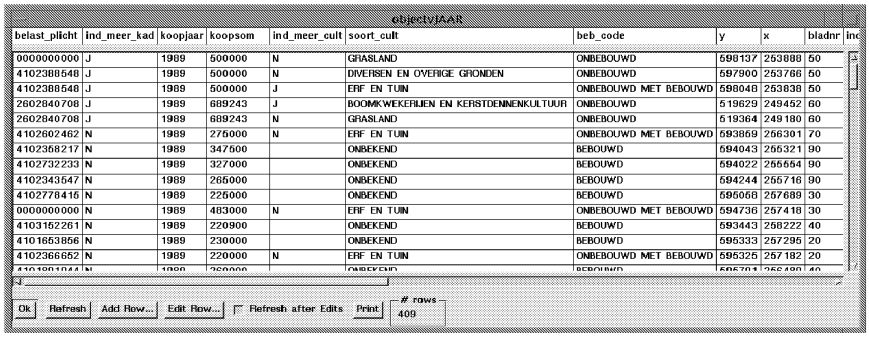

Figure 4: GEO++ dump: showing tabular format of land use

3 Coupling data

Traditionally access to data is done with help of a key, such as an identification number. An alternative is browsing or viewing large tables of information which are ordered; e.g. alphabetically. Modern approaches use user interfaces based on forms as an entry to the data. Filling in a search field, e.g. a name, results in a list of found hits. Using administrative and geometric data in an integrated manner gives a new entrance: the map.

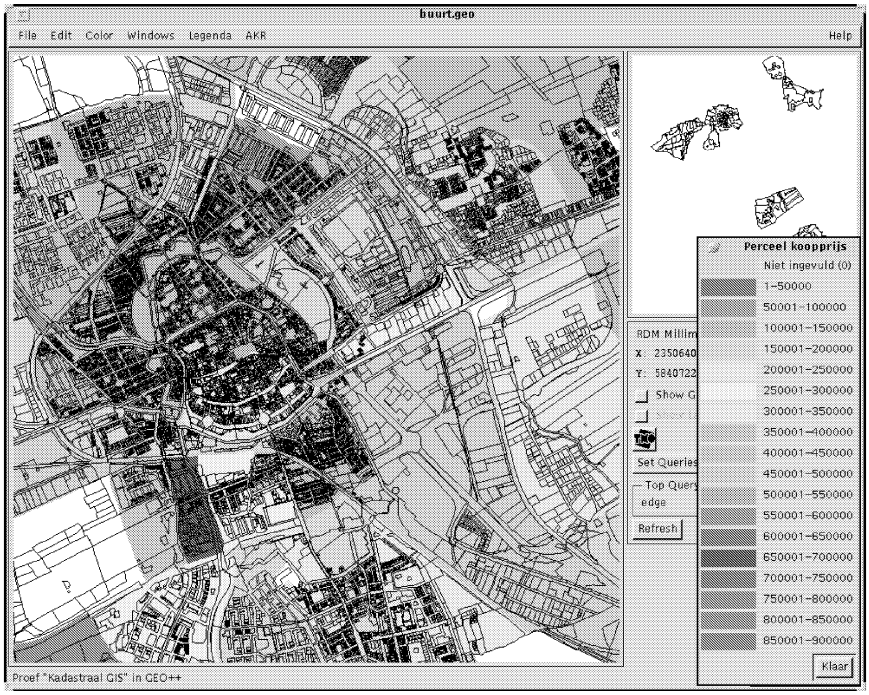

Figure 5: The average parcel price per CBS census region

Talking about integration of data in a relational database system environment means talking about integration of data models. Each data model represents a part of the real world, and is maintained by a specific application. Therefore, the data models may not be changed. Integration is done at the level of data views and is implemented by using the relational *join* mechanism. The left hand side of Figure 3 colors the land use of parcels, based on the attribute `soort_cult`. The parcels themselves are stored in the geometric data model. The `soort_cult` is stored in the administrative data model together with other administrative information such as prices and owners. Figure 4 also shows the land use of parcel, but now in a tabular format. The integrated model can be realized with the following view:

```
create view culture, location as select ap.soort_cult, gp.location
    from   geometric_parcel gp, administrative_parcel ap
    where  gp.municip = ap.municip and
           gp.osection = ap.osection and
           gp.parcel = ap.parcel
```

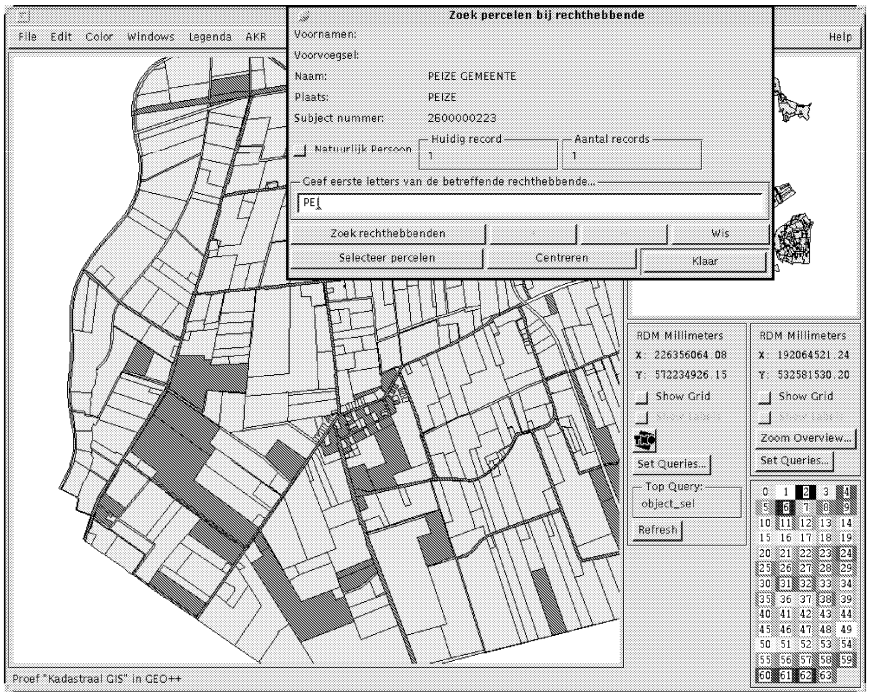

Figure 6: GEO++ dump: searching for parcels by owner

In the previous example you could see that municip(ality), (o)section and Parcel (number) are part of the key[1] on which the geometric and administrative tables are joined. Overview or summary maps can be based on spatial aggregation of detail information. If the data is aggregated in terms of relational queries, then this will easily translate into a map. Figure 5 shows the average price of the parcels per census region of the CBS (Central Bureau for Statistics). Assume the table with census regions is created:

```
create table census_region (name text(20), region long polygon);
```

The corresponding view is:

[1]The Cadastral map in the Netherlands is divided into municipalities. These municipalities consist of sections. In turn these sections are divided into parcels, which form the basic elements of the Cadastral map. Therefore, it is implicitly stored to which municipality a parcel belongs.

```
create view avg_price as
  select average(ap.price), cr.name, cr.region
  from administrative_parcel ap, census_region cr
  where inside(ap.location, cr.region)=1
  group by cr.name,cr.region;
```

This section started by stating that the map is a new entry to the data. Based on this principal, simply clicking on a parcel must give the corresponding administrative properties; see right hand side of Figure 3. Filling in forms is a more traditional method for searching data. If this is combined with a map interface, then even better browsing possibilities are obtained. Figure 6 shows a form to enter the name of the owner of parcels. In this case parcels of the 'Municipality of Peize' are in dark gray.

An alternative way of searching is to use the positions of objects: it must be possible to 'jump' to a specific coordinate (or to a specific parcel or to a specific postal code with or without 'auto-scale'). Finding all parcels, that cross a planned road is another way of selecting parcels based on a geometric attribute. Figure 7 shows a route and the corresponding parcels that are crossed. In SQL this would look like:

```
select 'wanted attributes of parcel'
    from geometric_parcel, line
    where overlaps(line.shape, geometric_parcel.shape) = 1
    and line.name = 'planned road'
```

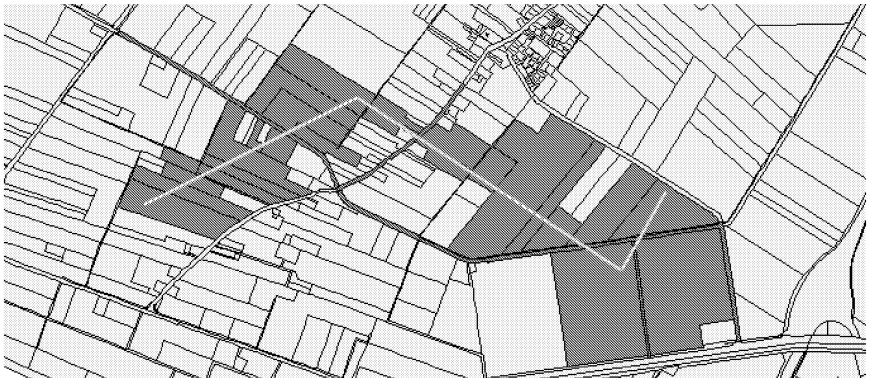

Figure 7: GEO++ dump: all the parcels which a route crosses

Integration with other geographic data sets is also possible by visualizing the data from different sources in one and the same map. In this case integration is not achieved on database level, but on map level. Figures 8 and 9 show the combination of the Cadastral map data with respectively soil data and an aerial photograph.

Figure 8: Integrated soil map

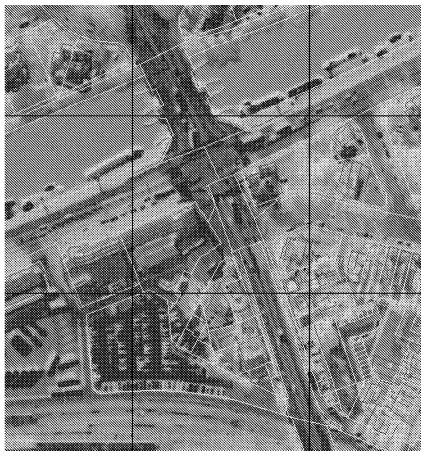

Figure 9: Aerial photograph

4 Visualizing changes over time

To visualize spatio-temporal data [1, 6, 7] specific techniques are required in the geographic query tool. An easy method is a 'snap-shot' map showing thematic symbols for a temporal theme; e.g. depicting dates, change rates, order of occurrence, etc. These maps can be created through 'temporal' SQL queries to obtain (average) age of the features, to retrieve predecessors, to select the oldest feature in the region, and so on. The following is an overview of a few more advanced techniques.

Double map: Display besides each other the same region with the same object types but related to two different dates. Windows are linked with respect to region and object types, but the user can modify the dates for the two windows with time-sliders. A general and flexible way to present data from the past is to create views on the base tables, which present the situation at given points in time. The views at time moments `t_beg` and `t_end` for the table with topographic line objects can be defined as:

```
create view line_t_beg as
   select * from line
   where tmin <= t_beg and t_beg < tmax;

create view line_t_end as
   select * from line
   where tmin <= t_end and t_end < tmax;
```

Change map: Display the changed, new and deleted objects over a specified time interval `(t_beg, t_end]` on top of the map. Again, the time interval

is specified with sliders. The reference map, for presenting the changes, can be the map at time t_end. Views for really new, deleted, and updated object versions in the interval (t_beg, t_end] are defined; e.g. the view for really new object versions is [2]:

```
create view line_new as
  select * from line l_new
  where t_beg < l_new.tmin and l_new.tmin <= t_end and
    not l_new.oid in
    /* make sure that there are no predecessors */
    (select l_pred.oid from line l_pred
    where l_pred.oid = l_new.oid and
          l_pred.tmax = l_new.tmin);
```

Space-time aggregation: Aggregate the (number of) changed, new, and deleted objects to larger units in order to visualize the change rate in different regions. For example, parcels belonging to a municipality and the number of parcel changes over a time interval are aggregated to the level municipalities with the following view:

```
create view municip_new_parcels as
  select count(*), gp.municip from geometric_parcel gp
  where t_beg < gp.tmin and gp.tmin <= t_end
  group by gp.municip;
```

In this case the 'spatial' aggregation is done on the basis of a thematic attribute municip, which indicates the municipality a parcel belongs to. It is also possible to aggregate spatial regions that are not related to the parcels; e.g. census regions of the CBS. The following is an example of true aggregation over space and time:

```
create view census_region_new_parcels as
  select count(gp.oid), cr.name, cr.region
  from geometric_parcel gp, census_region cr
  where t_beg < gp.tmin and gp.tmin <= t_end and
        inside(gp.location, cr.region)=1
  group by cr.name, cr.region;
```

Time animation: Visualize changes through an animation by displaying the same region and object types starting at t_beg in n steps to t_end. Note that an efficient implementation will not retrieve the complete data set over and over again. But it will first make one complete selection at time t_beg and use additional queries to obtain the changes in the time interval sorted on time and oid. Instead of using 'linear' time, e.g. seven days per animation step, it might be more useful to use the discrete 'change' times. Another variant is an animation in which *both* time and location may

[2] We assume in this view that oid, object identifier, is a key.

change to enlarge the perception even more.

Time as third dimension: Visualize changes over time, by using the third dimension for time. The user navigates through this 3D-space; see Figure 10. Looking from the top, the user observes the current situation. Horizontally cutting away above the plane $Z = t$ and looking down again, the situation at t is observed. Making vertical cuts, the users investigate the changes over time (along a line in 2D space). Further, transparency, perspective, stereo, rotation (motion), shading, and other 3D visualization techniques [4] can be used to enlarge the insight.

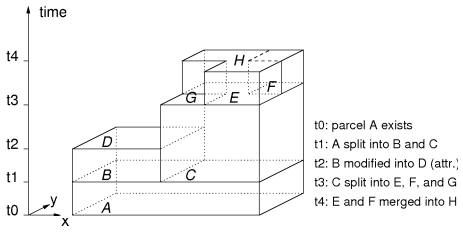

Figure 10: 3D visualization of parcel changes over time

5 Conclusion

In this paper it is demonstrated that the concept of views and relational joins form a powerful mechanism to integrate data models from different sources. If one of these data models stores geometric information, then it is easy to create thematic maps based on the values of the attributes stored in the other model. Aggregation in a relational database environment is a step further to derive new map information. Searching and querying the database can be done with help of regular search forms, by way of the map, using the position of objects or a combination of them. Visualizing historic data requires specific techniques such as animation or the use of the third dimension for time.

Instead of a test database containing only one province, in the future the query tool will operate on a nation-wide database, thus enabling nation-wide queries. Further, external users should be able to query the data over the internet. Limited functionality of the geographic query tool together with accounting functionality is expected to be implemented in Java as a basis for the digital Geo-shop.

Acknowledgments

We would like to thank our colleagues, Harry Uitermark, Maarten Moolenaar, and Koos Huygen for their valuable reviews.

References

[1] C. Armenakis. Mapping of spatio-temporal data in an active cartographic environment. *Geomatica*, 50(4):401–413, 1996.

[2] ASK-OpenIngres. INGRES/Object Magemenent Extention User's Guide, Release 6.5. Technical report, 1994.

[3] Cadastre Netherlands. Detaillering Software Architectuur 2000, Versie 3. Technical report, August 1995. (In Dutch).

[4] James D. Foley and Andries van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Reading, Mass., 1982.

[5] J. A. IJsselstein and A. P. Kap. Het kadastraal perceel: een stevig fundament! *Nederlands Geodetisch Tijdschrift Geodesia*, 37(7/8):343–349, 1995. (In Dutch).

[6] M. J. Kraak and A. M. MacEachren. Visualization of the temporal component of spatial data. In *6th SDH*, pages 391–409, September 1994.

[7] G. Langran. *Time in Geographic Information Systems*. Taylor & Francis, London, 1992.

[8] Professional Geo Systems (PGS). The GEO++ system, version 2.80, Reference manual. Technical report, September 1996.

[9] Peter van Oosterom. Maintaining consistent topology including historical data in a large spatial database. In *Auto-Carto 11*, April 1997. To be published.

[10] T. Vijlbrief and P. van Oosterom. The GEO++ system: An extensible GIS. In *5th SDH*, pages 40–50, August 1992.