The Balance Between Geometry and Topology

Peter van Oosterom, Jantien Stoter, Wilko Quak and Sisi Zlatanova

Department of Geodesy, Delft University of Technology, Thijsseweg 11, 2629 JA Delft, The Netherlands, {oosterom|stoter|quak|zlatanova}@geo.tudelft.nl

Abstract

The architecture of Geographic Information Systems (GISs) is changing: more and more systems are based on the integrated architecture, i.e. storing geometric data in the Data Base Management System (DBMS) together with administrative data. The first step in building a Geo-DBMS is by having data types and operators for simple features (i.e. geometric primitives): point, line and polygon. This has reached a level of standardisation and is now implemented in several commercial DBMSs. The next step is to have support for the topologically structured features in the DBMS, i.e. complex features. The DBMS can then check and guarantee consistency. In addition, complex operations can be executed within the DBMS. Despite the fact that topologically structured models are well known and that it is not difficult to store the topological references, it still remains an unresolved issue as to how to effectively implement these models within a relational DBMS. In this paper, we describe the design and implementation of a topologically structured management at the DBMS level. Our focus is to translate topological structures into geometric primitives. It is then possible to define a DBMS view on a topological primitive, which makes this appear as a geometric primitive. This process supports the best of both worlds: on the one hand there are advantages of the topological structure (no redundancy) and on the other hand the ease of explicit geometric primitives in querying, analysis and presentation is available.
Keywords: Topology, DBMS, OpenGIS, spatial modelling, SQL queries

1 Introduction

The integrated architecture (Vijlbrief & van Oosterom1992) of storing geometric data together with administrative data in the Geo-DBMS can be contrasted to traditional GIS approaches such as the dual architecture (separate DBMSs for geometric and administrative data) and the layered architecture (all data stored in a single DBMS with spatial knowledge contained in a layer between the application and the DBMS), for example ESRI's SDE. In the integrated

architecture, the DBMS is extended with spatial data types (point, polyline and polygon), and functions (overlap, distance, area and length). The first DBMSs offering these capabilities were experimental systems, such as Postgres (Stonebraker et al.1990), O2, Gral (Güting1989), and others (DeWitt et al.1990) and, of course, the functionality was not standardised in, for example, SQL (Date & Darwen1997, ISO1992). Immediately after the availability of first spatial DBMSs, the first GISs based on these DBMSs became available. These were either based on an extended (object) relational database (GEO++) or on a pure object oriented database (GeO2).

The importance of the integrated architecture was recognised by industry and the OpenGIS Consortium (Buehler & McKee1998) standardised the basic spatial types and functions, or in the OpenGIS terminology the Simple Feature Specification (SFS). The SQL/SFS implementation specification (Open GIS Consortium, Inc., 1998) will also be part of the future ISO SQL3 standard (ISO/IEC 13249-31999). In 1999 the first implementations of the OpenGIS SQL/SFS became available, which marked an important step forward in the maturing of GIS to become part of the mainstream ICT. Currently several commercial DBMSs are available with support for spatial data types (some support the OpenGIS standard): Ingres (1994), *Oracle 9i Spatial (2*001 and Hebert & Murray1999), Informix (2000) or IBM DB2 (2000). As an illustration and proof that geo-information is now part of standard ICT product, an example is shown below. In Informix the, OpenGIS compliant, SQL statement to select objects from the table `lki_boundary` where the `shape` attribute overlaps a given polygon (with two holes) provides the example:

```
select * from lki_boundary
where ST_Intersects (shape, ST_PolyFromText('polygon \
 ((103654574 460970880,104323607 460885924,104769627 460885924, \
   105523616 461013359,105544856 461395663,105061624 461741343, \
   104089976 461777967,103474041 461639912,103474041 461162032, \
   103654574 460970880), \
  (104610334 461108935,104610334 461459380,105056356 461459380, \
   105056356 461108935,104610334 461108935), \
  (103792627 461119555,103792627 461470000,104206790 461470000, \
   104206790 461119555,103792627 461119555))',28992));
```

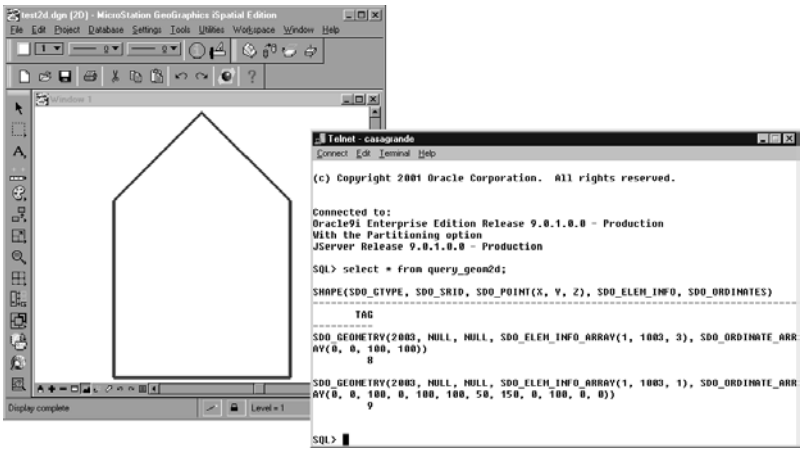

Fig. 1. Polygon stored in Oracle Spatial visualised with Bentley MicroStation

In addition, increasing numbers of commercial GIS packages support the integrated architecture: *MapInfo Professional v6.5 (*2001), *Intergraph GeoMedia Professional 4.0 (*2001) or *Bentley MicroStation GeoGraphics ISpatial edition,J 7.2.x (*2001). However, the above referenced DBMSs do not natively support topology and none of the mentioned GIS packages use topology from a DBMS. Fig. 1 shows a polygon in Bentley MicroStation, which was read from the Oracle Spatial DBMS.

The OpenGIS Consortium mentions topology several times in its abstract specifications (Open GIS Consortium, Inc.1999). What is still missing is the implementation specification for a DBMS environment. The OpenGIS abstract specification is very similar, and more or less at the same abstraction level as the work of the ISO TC 211 (ISO/DIS 19107, 2000). These two standards are now relatively well harmonized and both make a distinction between *geometric primitives* (point, curve, surface and solid) and *topological primitives* (node, edge, face and solid). The latter only have meaning within a topological structure. Note the 3D geometric and topological primitives are both called solid. We therefor propose to use the term *volume* for the 3D topological primitive instead.

Section 2 introduces the data model (options) and motivation behind topological structures. Section 3 argues why topology management should be a task of the DBMS. The design of a DBMS topology management extension is described in Section 4. A partial implementation of this design based on Oracle 9i is given in Section 5. This implementation is based on an additional topological meta data table together with PL/SQL functionality using this meta data. Special attention is paid to the aspect of spatial indexing (and clustering), which is very important when dealing with large, real world, data sets. Querying either the geometric or the topological 'side' of the system is illustrated in Section 6. Finally, conclusions, discussions and future research are described in Section 7.

2 Topological Structures

Having spatial types and operators is one part of the DBMS services required by a GIS. Two other components are 1. spatial indexing (quadtree, r-tree (Guttman1984, Samet1989)) and spatial clustering, and 2. representing and manipulating topological structures. Topological structures are used, among others, to represent planar partitions without redundancy. Another possible use of a topological structure is for the representation of a linear network. In this paper, we will focus on the topological structure for a planar partition. Topological structures for planar partitions have been well known for a long time, e.g. in TIGER (Boudriault1987, Kinnear1987), DIME (US Bureau of the Census1970), the Arc/Info system (Morehouse1989), the Netherlands Cadastre LKI (van Oosterom1997), and many other systems. They have also been studied extensively e.g. by Molenaar (Molenaar1998, de Hoop et al.1993). Relational DBMSs can effectively store the topological references: area left and right of a boundary, boundary to boundary references, treatment of islands, among other aspects., i.e. the modelling via topology. However, they do not support the topology structure management as they are not able to achieve consistency checks (area closed?, no intersecting boundaries?) and they do not provide the necessary operators, such as:

- the basic edit operations (split or merge area/insert or delete edge)
- computing the perimeter and area of topologically represented area
- solving the question: which areas are crossed by this query?
- map-overlay of two (or more) planar partitions
- selection of neighbouring features
- route planning in a linear network represented by a complex feature.

The problem with a standard relational DBMS, is that the declarative language SQL can not handle the 'navigational access' needed for the functionality described above. In SQL it is not possible to express the statement: 'Follow the *next* references of the boundary until we are back at the beginning'. Of course, this is a very simple operation in a programming language using one of the basic iterator concepts. This programming functionality is also available in every object-oriented DBMSs for implementing methods associated with classes.

Though some operations in a relational DBMS are impossible on a topologically structured area feature (e.g. compute area), this structure has many advantages:

- it avoids redundant storage (more compact than a full-polygon model);
- it is easier to maintain consistency of the data after editing;
- it is more efficient during the visualisation in some kind of front-end, because less data has to be read from disk;
- it is the natural data model for certain applications; e.g. during surveying an edge is collected (together with attributes belonging to a boundary); and
- it is efficient for certain query operations (e.g. find neighbours).

In this paper, the cadastral map will be used as a case study, because topology plays a key role in this spatial data set consisting of parcels covering the whole country of the Netherlands. The geometric data model for the cadastral *parcel* layer is based on winged-edge topology (Baumgart1975) as described in (van Oosterom1997, Lemmen & van Oosterom1995, Oosterom2001).

3 Does Topological Structure Management Belong in a DBMS?

In an implementation of topological structure management, not all functionality has to be provided by the DBMS. It is possible to provide part of the functionality in a front-end (or middleware) application. This enables the implementation to be based on standard tools without modifying the relational DBMS (server). However, as the support for complex features is quite generic, it should optimally be in the DBMS. This avoids reimplementation of the same functionality in several applications and it is the best guarantee for consistency control. Further, it also allows analysis queries on topologically structured features to be executed within the DBMS. Thus, no unnecessary data transfer to a front-end application takes place. Currently, the OODBMSs do seem to offer the most flexible platform for implementing the complex features. OODBMS offer the possibility of navigating through the database, controlled by programming the implementation methods. Their relatively weak acceptance by the market, the lack of a standard query language and the fuzzy separation between the DBMS and the application (as more and more user programming code ends up in the DBMS) form the motivation to include support for complex features in (extended/object) relational DBMSs.

Since the subject of implementing complex features in a relational DBMS is a fairly unexplored field, various issues have to be taken into account:

- The possibility of implementing a fully operational solution, that covers the total domain of all possible topological structures should be assessed. Since we will first focus on a subset of the topological structures - the planar partitions - the question will also be how far this solution will solve the total domain, and how this subset can be used as a basis to extend to other complex data types.
- The chosen solution of implementing the topology management in the database will have to be tested in an effort to establish the real benefits above other scenarios. Further, whether this functionality is best implemented in the front-end or in a middleware application, or that it be distributed over the DBMS will need to be explored
- A solution must have the possibility of being extended. Users must have the freedom to add on their own applications or variants of topology management structures, without creating conflicts with the database.
- Having an abstract standard is one important step, but the implementation forms the next required step in practice. The OpenGIS Consortium mentions

topology several times in its abstract specifications (Open GIS Consortium, Inc.1999). What is still missing is the implementation specification of topology (or sometimes also called complex features) for specific platforms. The question remains whether this could be possible for standard SQL.

- The rule of thumb whether certain functionality belongs to the DBMS or to a specific application is whenever it concerns general and reusable aspects then this belongs to the DBMS. In this section it was argued that topological management is a generic functionality and therefore belongs to the DBMS.

4 Design of DBMS Topological Structure Management

Subsection 4.1 describes an earlier attempt to extend (post)quel (a language similar to SQL, used in the Postgres DBMS) with 'prototypes' and the 'create layer' statements for the management of topology structures. The drawback of this approach is that it requires an extension of the DBMS query language, which has a serious impact on the DBMS kernel implementation. In practice (and in theory), there are many different types of topology structures used and described. Subsection 4.2 tries to categorise these different topological structures. In Subsection 4.3, this paper suggests a different approach than the 'prototype' approach for topology management having less impact on the DBMS kernel. Most relational DBMSs are currently enriched with object-oriented technology. This, among others, allows the DBMS to be extended with types and functions, which exist and run within the DBMS.

4.1 Query Language Extension Based on Prototypes

For the modelling part of topology and one of the most important operators, i.e. map-overlay, an attempt to extend (post)quel (a language similar to SQL, used in the Postgres DBMS) was described (Schenkelaars & van Oosterom1995). It was suggested to use 'prototypes' to define topological roles and the 'create layer' statement to define (declare) the roles of the different tables based on the prototypes. A translation was given from the postquel to the SQL syntax (Oosterom et al.2000),:

```
create prototype face(id=oid, bnd=edge.id[]);
create prototype edge(id=oid, line=polyline2, left=face.id,
  right=face.id);
```

The prototypes are used to model the different roles that entities can play in a topological structure. It also standardises names of the attributes with specific meaning; e.g. the object identifiers, the topological references and the base geometry. As will be described in the next subsection, there are several ways to implement topology, the method described in this subsection is based on edges and faces (no explicit nodes). There are references from a face to all its boundaries (exterior and possibly also interior) and there are references from a boundary to

the left and right face. This type of topology is sometimes called, wheel topology (NNI NEN-36101995). Other topology implementations (with other restrictions) are possible (see also Subsection 4.2), but can be based on different prototypes. Based on the abstract prototypes given above, it is now possible to create specific tables that inherit the topological structure from the prototypes:

```
create table parcel(owner=text) inherit face;
create table boundary(quality=integer) inherit edge;
create layer (layer_id='cad_map', topology_type='wheel_topol',
                face_role='parcel', edge_role='boundary');
```

The last statement '`create layer`' defines which tables are used to implement a certain topological structure, the parameter '`topology_type`' specifies the type of topology. In a way the '`create layer`' statement could be compared to the '`create index`' statement, because it has important (under water) side effects later on during the updating.

It is assumed that the updates are correct and consistent with the topological rules. Otherwise a transaction can not be committed. This is checked by the DBMS as a result of the '`create layer`' statement. After this first layer `cad_map` is created, a second layer '`soils`' could be created in a similar manner. Having created two layers, it is now possible to perform the complex map-overlay operation within the DBMS. The DBMS must be extended to support this overlay operation, which returns the number of faces as a result and as a side effect is able the compute the new (topological) layer as overlay of the input layers. Again, note that the support of topology is more or less the same level in the DBMS as the support of indices or referential integrity constraints.

4.2 Topological Structures Inventory

Without claiming to be complete, this subsection tries to make an inventory of the most common topological structures. It has already been stated several times that there are many different ways to implement topology. In the previous subsection the wheel topology was discussed. Other topology implementations (with other restrictions) are possible and can be implemented in the DBMS. For example, the Dutch Cadastre uses a planar partition for the parcel layer based on edges and faces, but with different references, i.e. the chain topology (also called the winged-edge topology). The different topological structures can be characterised by the following 'parameters':

- what is the dimension of the embedding space: 2D, 2.5D, 3D, time added?
- which topological elements (primitives) are used: node, edge, face, volume?
- are the elements considered directed (oriented) or not?
- which explicit topological relationships (part_of, in, on) are stored?
- what are the topological 'rules': crossing edges allowed?, dangling elements allowed?, same topological primitive on both sides of boundary allowed?, etc.

Table 1. Inventory of different topological structures

	Topol. type	Dimension	Primitives used	Topological tables	Explicit Relationships	All tables	Rules
TIN	221	2D	Node,edge	node,edge	no	2	Planar Partition
Wing-edge	222	2D	Edge,face	edge,face	no	2	Planar Partition
Wheel (chain)	223	2D	Edge,face	edge,face	no	2	Planar Partition
3DFDS	381	3D	node,arc, edge,face	arc,edge, face	node-on-face node-in-volume arc-partof-line arc-on-face arc-in-volume	8	Space Partition
TEN	352	3D	node,arc, triangle, tetrahedron	arc,triangle, tetrahedron	tri-partof-surf arc-partof-line	5	Space Partition
Cell-tuple	313	3D	0-cell,1-cell 2-cell,3-cell	cells	no	1	Space Partition
SSS	364	3D	Node,face	face,line surface,volume	node-in-volume face-in-volume	6	Space Partition

Table 1 illustrates some 'parameters' of different topological structures.

The topological type is obtained by encoding the dimension of the data structure, the total number of topological tables and the topological 'rules'. As can be seen from in Table 1 many types of topology are possible and even more are possible; e.g. structures not for partitions such as a linear network. First of all, the dimension may be 2D or 3D. Second, not all primitives have to be used. For example, in the 2D winged-edges structure, neither the node nor the volume 'roles' are used, but only the edge and face 'roles'. In the TIN, only the node and the edge 'roles' are used. In SSS (Zlatanova & Tempfli2000) the roles that are not explicit are the edge and volume 'roles'. 3DFDS (Molenaar1998) does not use the volume 'role', while TEN (Pilouk1996) exploits the 'role' of all the primitives. It depends on the type of topology as to which roles are expected. Note, that the node table in the presented 3D topological structures contains only the coordinates of the points, i.e. links to other topological primitives are not provided. Indeed, 3D topological structures that assign topological functions to the node are also available (de la Losa & Cervelle1999).

4.3 Topology Management using Meta Information

A lighter approach than extending the DBMS kernel with topological functionality involves creating a table with meta information describing the topological structures in the DBMS. This topological meta information can than be used both within the DBMS (back-end) and outside the DBMS (front-end or middleware). In general, meta information (or system catalogs) of a DBMS contains descriptions of the data stored in the database: tables, attributes and types, and contains

descriptions of the available types and operators. This enables dynamic SQL applications. A traditional example of the use of meta-data is to obtain a description of the tables within Oracle such as:

```
SELECT owner,table_name,column_id,column_name,data_type
FROM all_tab_columns
ORDER BY owner,table_name,column_id
```

Oracle spatial also uses a specific meta data table to describe geometric attributes in more detail: `USER_SDO_GEOM_METADATA`, which contains information about the number of dimensions, the extent of the domain and the resolution. In case the (relational) DBMS has to support topology management, the structural knowledge has to be stored (and be accessible for applications). For example, topological layer names, boundary tables, area tables and relevant attributes with metrical and topological information must be determined and accessible. One solution for this problem is providing prototypes as a basis for the possible topological structures; see Subsection 4.1. A drawback of this solution is that the topology elements (object ids, references and also the metric attributes) have fixed names.

An alternative is to describe this topological information in another meta data table. Again, somewhere it must be declared which tables and which attributes carry the topological information. An example of the extension of the meta information of the DBMS was given in the context of GEO++ (`geo_dyn_info table`) in (Vijlbrief & van Oosterom1992). Below the table definition of a more generic topological meta data table is shown. This model we now propose as the basis of our solution:

```
                                | /* example entry for winged-edge
                                |    topology of Dutch cadastre */
create table topo_meta_data(    | insert into topo_meta_data values
 topol_layer_name  varchar(32)  |   (
                     PRIMARY KEY,|    'parcel layer',
 topol_type        integer,     |    222,
 node_table_name   varchar(32), |    NULL,
 node_info         varchar(256),|    NULL,
 edge_table_name   varchar(32), |    'lki_boundary',
 edge_info         varchar(256),|    'object_id,l_obj_id,r_obj_id,
                                |     fr_line_id,lr_line_id,
                                |     geo_polyline',
 face_table_name   varchar(32), |    'lki_parcel',
 face_info         varchar(256),|    'object_id,line_id1',
 volume_table_name varchar(32), |    NULL,
 volume_info       varchar(256));|    NULL);
```

The topology type is encoded in a number and stored in the meta data attribute '`topol_type`'. Knowing which tables to use for which role is not sufficient. It must be explicit about which attributes in these tables contain the needed identifier, references and other information. This depends again on the topology type. Standardising the topologically structured knowledge makes it possible to implement functionality (within the DBMS). Several categories of functionality can be considered. Initially we focus on the 'bridge' between topology and

geometry by implementing a function which realises (materialises) geometric primitives from topological primitives.

5 Implementation in Oracle 9i

The Oracle 9i spatial DBMS will be used to test the topological structure management proposal. The manner in which the relational model is extended (PL/SQL, stored procedures) to support topological structure is described. The interface of a hard coded (table and attribute names are fixed) example of a PL/SQL function which translates a topological structure into a geometry is given below:

```
CREATE OR REPLACE FUNCTION return_polygon
  (i_object_id kadtest.lki_parcel.object_id%type)
RETURN mdsys.sdo_geometry
IS
...../* body of the function */
  polygon:=mdsys.sdo_geometry(2003, NULL, NULL,
          mdsys.SDO_ELEM_INFO_ARRAY(1, 1003,1), list_coordinates);
  return polygon;
END return_polygon;
```

This function uses the 'lki_boundary' table in order to obtain the coordinates of the polygon realised for the face in the 'lki_parcel' table. In our current prototype implementation, the boundaries are accessed based on their 'object_id', on which a btree index is created. For fast access, spatial clustering of the boundary table is needed. The clustering supports fast access in case a large number of boundary records, forming one polygon are to be retrieved in a sequence and are spatially related A spatial index is not needed, per se, for this purpose. However, sometimes boundaries are spatially selected independent of the polygons and for this reason also a spatial index, an rteee, is created on the boundary table. Now, the function 'return_polygon' can be used to create a view in the following way:

```
create view lki_parcel_pgn_vw as
  select municip, osection, parcel, oarea,
    return_polygon(object_id) shape
   from lki_parcel;
```

Fig. 2 shows an example that uses the function to view topological data in a viewer that knows that the data it displays is in a topological model (Wilko Quak's Quick GIS). Currently, table names and attribute names are hard coded in the function 'return_polygon'. One might wonder what is the role of the 'from' clause in this situation, besides being necessary in a valid SQL 'select' statement. The answer is that the 'from' clause provides the iteration over the 'object_id's' from the 'lki_parcel' table.

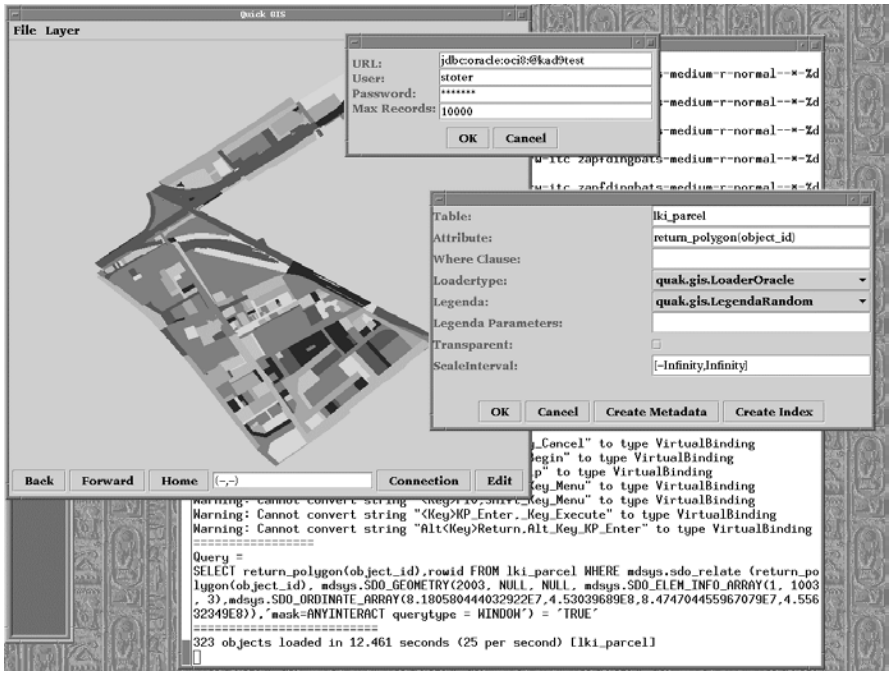

Fig. 2. Viewing topological data in a veiwer that is not topology aware

A function-based spatial index is created in order to optimise the performance. Since version 9i, Oracle offers function-based indices, i.e. an index, which is created on the return value of a function in contrast to a normal index created directly on the value of an attribute. The spatial index is created based on the pre-computed values returned by the function. This is implemented in Oracle 9i in two steps. First, by inserting a row in the table `USER_SDO_GEOM_METADATA` and then the spatial index is created by specifying the function name and parameters.

```
insert into user_sdo_geom_metadata values(
  'LKI_PARCEL', 'return_polygon(object_id)', mdsys.sdo_dim_array (
  mdsys.sdo_dim_element('X', 82291,  84261, 0.0005),
  mdsys.sdo_dim_element('Y', 453039, 455632, 0.0005)), NULL);

create index lki_parcel_idx on
  lki_parcel(return_polygon(object_id))
  indextype is mdsys.spatial_index;
```

The view that was created (`lki_parcel_pgn_vw`) can use this index and thus improve performance and functionality. Without a function-based spatial index it would not have been possible to properly index the faces. During an overlap query (or any other search query using the spatial index, for some examples see the next section), objects are filtered by means of this index. That is, using the pre-computed boxes which are stored in the R-tree. Then the '`return_polygon`' function is executed to obtain the complete geometry of

filtered objects used in the exact overlap test. In Informix (Informix2000) it is possible to build an R-tree index which also includes the shape instead of only the box (as is the case in the standard R-tree), which then avoids executing the function `return_polygon`.

6 Querying Topology and Geometry

In this section examples of queries are given, which use the function to realise a geometry based on stored topology as described in section 5. The first two *geometric* queries return respectively the area and the maximum area (with a tolerance value of 0.5) of realised geometry's.

```
select municip, osection, parcel, sdo_geom.sdo_area(shape,0.5)
  from lki_parcel_pgn_vw;
select max(sdo_geom.sdo_area(shape,0.5))
  from lki_parcel_pgn_vw;
```

The next *geometric* query gives all the parcels, which are within a distance of 100m of the given rectangle (specified by lower-left and upper-right coordinates).

```
select parcel from lki_parcel
  where sdo_within_distance(
    return_polygon(object_id),mdsys.sdo_geometry(2003,NULL,NULL,
      mdsys.sdo_elem_info_array(1,1003,3),
      mdsys.SDO_ORDINATE_ARRAY(832018,453536,832824,453610)),
   'distance = 100')='TRUE';
```

Next a *topological* query selects all neighbour parcels of parcel 'X'.

```
select distinct l_obj_id neighbour_parcel
  from lki_boundary where r_obj_id='X'
union all
select distinct r_obj_id neighbour_parcel
   from lki_boundary where l_obj_id='X';
```

Another pure topological query is to find all boundaries of parcel 'X'.

```
  select distinct object_id parcel_boundary
    from lki_boundary where r_obj_id='X' or l_obj_id='X';
```

Finally, an example of a *complex* query: map-overlay, which computes the intersection of all faces from one layer with all faces from the other layer. Below, one input layer is topologically structured '`lki_parcel`' and the other layer is not topologically structured '`query_geom`'. The result is stored in a non-topologically structured table '`overlay_result`'.

```
create table overlay_result as
select object_id, tag,
  sdo_geom.sdo_intersection(return_polygon(object_id),q.shape,0.5)
    clip_geom
  from lki_parcel, query_geom q
  where mdsys.sdo_relate (return_polygon(object_id),
    q.shape,'mask=ANYINTERACT querytype = JOIN') = 'TRUE';
```

In the target list the actual geometric computation is performed in addition to selecting the identifiers from both input layers. If both input layers are partitions, the output layer will also form a partition again. To execute the map-overlay efficiently, the where clause has been added and tests for overlap (through a spatial index because a spatial operator is used).

The next step is to get the topological layer as a result of the map-overlay operation. A solution may be to make a special purpose map-overlay function (in PL/SQL), which does all the computation and populates the target topology tables.

7 Discussion and Conclusion

In this paper we presented a breakthrough in managing topology and geometry in RDBMSs based on a function, which realises (constructs) a topological primitive into its geometric counter part. The proposed solution is based on a generic topology meta data table and stored procedure (PL/SQL), which uses this meta data to realise the geometry. This makes it possible to have both the advantages of a topological structure (e.g. no redundancy) and advantages of explicit geometric primitives in querying, analysis and presentation. Any GIS package being able to display geometry from the DBMS (called `sdo_geometry` in Oracle) can now also analyse, query and display topologically structured features from the DBMS. An important issue is the connection between the features and their topology and geometry 'aspects'. Our generic approach makes it possible to integrate topology and geometry as attributes of the features or to separate those in two modelling levels. In the second approach, which is also described in (Molenaar1998), topology and geometry aspects are modelled (in tables such as node, edge and face) separately from the features, which are modelled on top of (refer to) the topology tables. This approach also fits in our generic approach and it is up to the application developer to decide on the best model for a given situation. It is our expectation that future ISO and/or OpenGIS standards with respect to topology will also fit in our generic approach.

Future work will focus on keeping the topological structure consistent under updates. This issue is related to the temporal aspect of geo-information as updating is the source of historical information (van Oosterom1997). In the same manner as there should not be gaps and overlaps in space (in a partition), there should also be no gaps and overlaps in time: a next version should start at the moment the previous version becomes outdated.

One can assume that the topological structure is correct and that updates keep the structure correct and consistent within the topological rules. Otherwise a transaction can not be committed. This must be checked by the DBMS. Two approaches can be identified. In the first solution, a group of insert/delete/updates to node/edges/face/volume tables is considered as one transaction and on the commit of this transaction a function is executed, which checks the correctness of the changes. In the second solution it is not allowed to change the node/edge/face/volume tables directly, but only through higher level edit

operations, which bring the DBMS from one consistent state into the other (by changing several tables at the same time). Further research will investigate and compare these different solutions.

Other future work will try to implement the more generic solution, including support for other types of topological structures. This solution should also be applicable for other RDBMSs. The later could perhaps be achieved by not using Oracle's PL/SQL in the server, but the more generic Java/SQL. This might also be beneficial for the performance, due to the more advanced techniques of a JVM, such as just in time compilation. An interesting issue related to other topology types, raised in (de Hoop et al.1993), is whether it is possible to share the geometry in differently structured topological layers. In our model, this is not visible, although an implementation might support this. We did not investigate this, and this is again a good item for future research.

Finally, complex operations such as map-overlay of two (or more) planar partitions, resulting in a new planar partition will be investigated. Of course, as usual everything performed within the DBMS.

Acknowledgments

We would like to thank the participants in our research center, GDMC (Geo Database Management Center) for their cooperation, for making (beta) versions of their software available and for sharing their thoughts on the R&D developments in the Geo-DBMS domain. In this research project, we used Oracle 9i as our main DBMS. We owe our thanks to Tom Vijlbrief (Dutch Cadastre), Siva Ravada (Oracle), and the anonymous SDH reviewers for their valuable remarks.

References

ASK-OpenIngres (1994) INGRES/object management extension user's guide. Release 6.5, Technical report, ASK-OpenIngres

Baumgart BG (1975) A polyhedron representation for computer vision. National Computer Conference, pp 589-596

Bentley MicroStation GeoGraphics ISpatial edition (J 7.2.x) (2001)

Boudriault G (1987) Topology in the TIGER file. Auto-Carto 8:258-269

Buehler K, McKee L (1998). The OpenGIS guide - introduction to interoperable geoprocessing. Technical Report Third edition. The OpenGIS Consortium, Inc.

Date CJ, Darwen H (1997) A Guide to the SQL standard. 4th edition, Addison Wesley

DeWitt DJ, Futtersack P, Maier D, Velez F (1990) A study of three alternative workstation-server architectures for object oriented database systems. Technical Report 907, Computer Sciences Department, University of Wisconsin-Madison

Güting, RH (1989) Gral: An extensible relational database system for geometric applications. In: Proceedings of the Fifteenth International Conference on Very Large Data Bases, Amsterdam, ACM, New York, pp 33-44

Guttman A (1984) R-trees: A dynamic index structure for spatial searching. ACM SIGMOD 13: 47-57

Hebert J, Murray C (1999) Oracle Spatial User's Guide and Reference. Oracle Corporation, Redwood City, CA, USA. Part No. A77132- 01

de Hoop S, van Oosterom P & Molenaar M (1993). Topological querying of multiple map layers. In: COSIT'93, Elba Island, Italy, Springer-Verlag, Berlin, pp 139-157

IBM (2000) IBM DB2 Spatial Extender User's Guide and Reference. special web release edition

Informix (2000) Informix Spatial DataBlade Module User's Guide. Part no. 000-6868

Intergraph GeoMedia Professional 4.0 (2001) [online]. Available from: http://www.intergraph.com/gis/ gmpro/

ISO (1992) Database language SQL. Technical Report Document ISO/IEC 9075, International Organization for Standardization

ISO/DIS 19107 (2000) Geographic information - Spatial schema, Technical Report final text of CD 19107 (15046-7). International Organization for Standardization, TC211/WG2

ISO/IEC 13249-3 (1999) Information technology - database languages - SQL multimedia and application packages - part 3: Spatial. ISO/IEC JTC 1 SC 32, Technical Report Part 3, International Organization for Standardization

Kinnear C (1987) The TIGER structure. In: Auto-Carto 8. pp 249-257

Lemmen C, van Oosterom P (1995) Efficient and automatic production of periodic updates of cadastral maps. In: JEC'95, Joint European Conference and Exhibition on Geographical Information, The Hague, The Netherlands, pp 137-142

de la Losa A, Cervelle B (1999) 3D topological modelling and visualisation for 3D GIS. Computer & Graphics 23

MapInfo Professional v6.5 (2001) [online]. Available from: http://dynamo.mapinfo.com/products/index.cfm

Molenaar M (1998) An Introduction to the theory of spatial object Modelling for GIS. Taylor and Francis

Morehouse S (1989) The architecture of Arc/Info. In: Auto-Carto 9. pp 266-277

NNI NEN-3610 (1995) Terreinmodel vastgoed. Termen. Technical report, Nederlands Normalisatie-instituut (In Dutch, English version also available)

van Oosterom P (1997) Maintaining consistent topology including historical data in a large spatial database. In: Auto-Carto 13. pp 327-336

van Oosterom P, Lemmen C (2001) Spatial data management on a very large cadastral database. Computers, Environment and Urban Systems 25(4-5):509-528

van Oosterom P, Verbree E, Kap A (2000) Storing and manipulating simple and complex features in database management systems. In: Proceedings of the 3rd AGILE Conference on Geographic Information Science, Helsinki/Espoo, Finland, pp 178-182

Open GIS Consortium, Inc. (1998) OpenGIS simple features specification for SQL. Technical Report Revision 1.0, OGC

Open GIS Consortium, Inc. (1999) The OpenGIS abstract specification, topic 1: Feature geometry. Technical Report Version 4 (99-101.doc), OGC

Oracle 9i Spatial (2001) [online]. Available from: http://otn.oracle.com/products/spatial/

Pilouk M (1996) Integrated Modelling for 3D GIS. Ph.D. thesis, ITC Enschede, Netherlands

Samet H (1989) The Design and Analysis of Spatial Data Structures. Addison-Wesley, Reading, Mass.

Schenkelaars V, van Oosterom P (1995) Map-overlay within a geographic interaction language. In: Auto-Carto 12, Charlotte, NC, pp 281-290

Stonebraker M, Rowe LA, Hirohama M (1990) The implementation of Postgres. IEEE Transactions on Knowledge and Data Engineering 2(1): 125-142

US Bureau of the Census (1970) The DIME geocoding system. Technical Report 4, Census Use Study, US Department of Commerce, Bureau of the Census, Washington, DC

Vijlbrief T, van Oosterom P (1992) The GEO++ system: An extensible GIS. In: Proceedings of the 5th International Symposium on Spatial Data Handling. Charleston, South Carolina, pp 40-50

Zlatanova S, Tempfli K (2000) Modelling for 3D GIS: spatial analysis and visualisation trough the web. Proceedings of the XIX congress of ISPRS Com. VI/2. 16-23 July, Amsterdam, Vol. XXXIII, B4/3, Comm. IV, pp 1257-1264