Topology versus non-topology storage structures

Functional analysis and performance test using LaserScan Radius Topology

J.H. Louwsma

Cover:
Query elements for the performance test

Topology versus non-topology storage structures

Functional analysis and performance test using LaserScan Radius Topology

J.H. Louwsma
Delft, June 2003
Technical University Delft
Faculty of Civil Engineering and Geosciences
Department of Geodesy

Preface

This report describes the case-study to the functional analysis and performance test using LaserScan Radius Topology. This kind of case-study can be done by geodesy students who are in the last stage of the study. The goal of the case-study is to get acquainted with doing independently a relative small investigation in the terrain of geo-DBMS's.

This report can be read by people who are not familiar with topological structures and Radius Topology but it is mostly interesting for people who already worked with spatial data stored in databases. Radius Topology is the first program or supplement on a program which uses topological storage structures instead of the so far used plain geometry storage structure.

Chapter 2 is interesting for people who want to know how Radius Topology is implemented and functions. Chapter 3 and 5 are of interest for who wants to know more about the functionalities and the performance of Radius Topology. In the Appendices more detailed information can be found as background for this report.

I owe gratitude to Peter van Oosterom, Jantien Stoter, Wilko Quak and especially to Theo Tijssen who helped me a lot to get acquainted with Oracle and sqlplus. They all work at the section GIS Technology of the study Geodesy at the technical university of Delft. I also owe gratitude to Alexandra Bade who worked for LaserScan in England.

Delft, June 2003
J.H. Louwsma

Abstract

Databases management systems have their roots in managing administrative data. However, data can contain spatial components and storing this type of data is not that easy. Topological structures can help by storing spatial data in a proper way. The basis of topological strucutures are topological primitives (nodes, edges and faces) and references between these primitives. E.g. an area is stored by references to the edges of that area and the edges are represented by references to the end nodes, where several edges meet. Only the nodes are stored in x and y coordinates. A topological structure is an elegant and robust way to model relationships between geospatial features. Some of the advantages of topological structured data are:

- it avoids redundant storage;
- it is easier to maintain consistency of the data after editing;
- it is efficient for certain query operations.

LaserScan's Radius Topology is the first commercial system that is able to manage topological structured data in a database. Radius Topology is not a stand alone program, but extends the Oracle Spatial database. The main question of this investigation is:

Is a geo-DBMS that uses topological structured data better than geo-DBMS's that uses non-topological structured data?

To find an answer on this question a functional analysis and a performance test of the storage of topological structures using Radius Topology are made. A 1:50.000 data set is used for this research. This data set is stored twice; As plain geometry (normally used storage structure) and as topological structured data. The primitives of the topological structure and the references between these primitives are stored in separate tables.

The functional analysis of Radius Topology consists of an investigation to the functionality. Querying and editing of the topological structured data are the most important functionalities for this case-study. The querying posibilities of Radius Topology are almost equivalent with the Oracle query possibilities.

The editing possibilities of the topological structured data in Radius Topology are restricted by some rules. These rules are defined to prevent inconsistencies in the data set and to guarantee a good data quality.

The performance test of Radius Topology consists of a comparison in storage requirements (in terms of disk space use) for the plain geometry version and the topology version and a comparison of the results of some spatial questions.

In theory the storage requirements for topological structured data should be less than the storage requirements of plain geometry data. This is because only the geometry of the nodes and references are stored once using topological structures in stead of storing geometry redundant using plain geometry. Comparing the storage requirements of topological structured data by Radius Topology and plain geometry data of the same area, the conclusion can be drawn that the storage requirements of the topological structured data is much larger than the storage requirements for the plain geometry. An explanation can be found in the way Radius Topology implements the topological structure; a large overhead exist. Radius Topology uses a large number of tables for the primitives and references and ID's are stored in many tables. Also many indexes are necessary because many tables exist.

Most of the questions for the actual performance test concern geometry and only one is a topology relationship question. Using the topological structured data, geometry has to be constructed real time to find answers on questions concerning geometry. Using the plain geometry version, the geometry of the features is already stored and does not have to be constructed real time. This is immediately the explanation why the performance of Radius Topology concerning geometry questions turned out to be very bad in this study. Using plain geometry data answers were found at least twice as fast as finding answers using topological structured data. Only the relationship question turned out to be much faster using topological structured data. The explanation therefore is, that only looking at the references is enough for finding an answer using toplogical structured data.

The conclusion of this case study after the functional analysis and performance test of Radius Topology is, that at this moment the performance of a geo-DBMS that uses non-topological data is better than a geo-DBMS that uses topological data, but the toplogical rules defined for editing and guaranteeing a consistent dataset, are a great benefit for maintaining a spatial dataset. The choice of which type of geo-DBMS is better, is dependent of the mainly use; querying (non-topological data storage is preferred for a better performance) or managing (topological data storage is preferred for data quality and consistency).

Content

1. Introduction

Today's Data Base Management Systems (DBMSs) have their roots in managing administrative data. However, data can contain spatial components and it is much harder to store these spatial data in a proper way, but the last couple of years several systems offer support for spatial data types. In addition to the spatial data types, topological structures exist and these structures can be stored by references in a DBMS. An topology model uses topological primitives to describe the features in a geometry table. These (2D) primitives are nodes, edges and faces. Nodes represent the end of a line, any intersection of a line or area boundary and point features. Edge primitives are used to represent lines, area boundaries and boundaries of holes in areas. Faces represent areas. E.g. an area is stored by references to the edges of that area and the edges are represented by references to the nodes which the edges consist of. There are two important models of topological structure. The nodes and edges are used to support one-dimensional network topology, and the nodes, edges *and* faces are used to support the two-dimensional planar topology. The DBMS does not only have to store the references, but also has to understand the references used within the topological structure. LaserScan Radius Topology understands these references and can manage topological structures.

Radius Topology is not a stand-alone program, but extends an Oracle database by adding a dynamic topology management layer to the Oracle server. Radius Topology transforms Oracle Spatial data into topologically-aware information that can be queried to find out, for example, what an object touches or what it contains. Answers are received fast and accurate because a topology model is an efficient way to model relationships between geospatial features.

Some of the advantages of topological structured data are:
- it avoids redundant storage;
- it is easier to maintain consistency of the data after editing;
- it is efficient for certain query operations.

The aim of this research will be to do a functional analysis and a performance test of the storage of topological structures in LaserScan Radius Topology. The *main question* of this investigation will be:

Is a geo-DBMS that manages spatial data by means of topology better than geo-DBMSs that do not use topologically structured data, both in terms of functionality and performance?

To come to an answer on this main question, a separation into smaller questions or steps can be made:
- In which way is the topological structure stored or implemented in a DBMS?
- What functionalities does Radius Topology provide (modelling, storing, batch loading, structuring, querying, editing…)?
- What are the storage requirements for topological and non-topological structured data (in terms of disk space use)?
- Are the results from a representative set of spatial questions the same using topological structured data instead of non-topological structured data (redundant storage of polygons)?

  Types of questions to be asked:
  - a find all objects (faces of the area objects to be returned as polygons, so they can be displayed) within a certain rectangular area;

  b find all objects that overlap with a given polygon;
  c find and clip all objects with a given search polygon
  d find all objects that intersect with a (possibly very long) query polyline;
  e find all objects that contain a given set of search points (one or more);
  f find all objects that are adjacent to a given object (also maintained with the topology structure) (polygon).
- Are the results of the spatial questions (mentioned above) obtained faster using topological structured data instead of non-topological structured data?

The initially test data used was a set of cadastral parcel (LKI) data (1:10.000) of Almere (a municipality in the province Flevoland). Because this dataset appeared to have some validity problems after the converting process into topological data, we decided to use another dataset for the comparison.

Eventually a 1:50.000 data set containing area features of the Dutch Topographic Service ('Topografische Dienst Nederland') of the Flevoland region in the Netherlands was used. A comparison is made between a topologically structured data set and a plain geometry data set (using 'standard' spatial storage) of Flevoland. At the end the results of the questions mentioned above are compared and conclusions are drawn.

2. Architecture of Radius Topology

2.1 Implementation of Radius Topology in Oracle

Radius Topology extends the Oracle Spatial database by adding a dynamic topology management layer to the Oracle server (see figure 2.1, from Radius Topology Database Administrator's Guide). The Radius Topology Manager works constantly behind the scenes to ensure that the data is consistent with the spatial model.

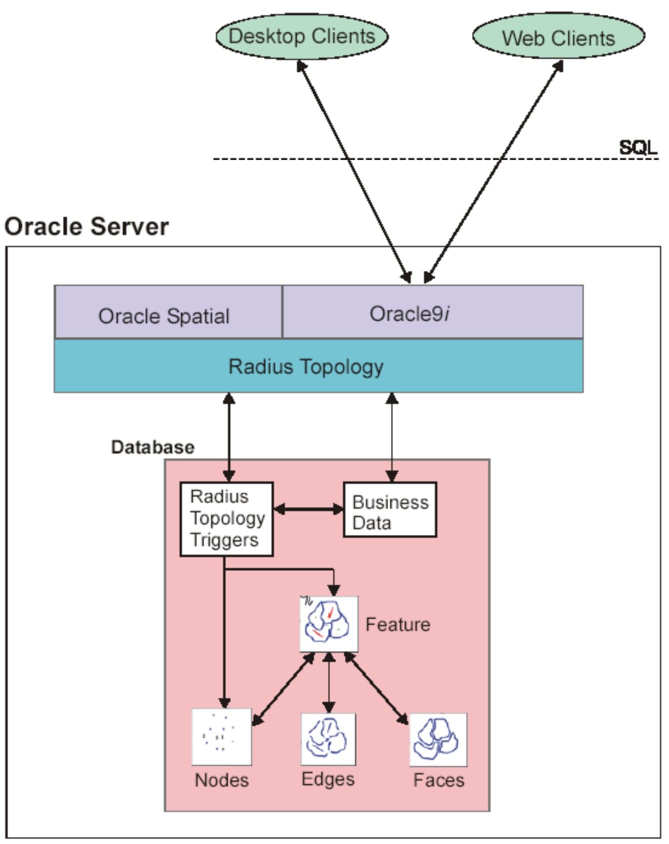


figure 2.1: Radius Topology is an additional layer on the Oracle database (source: Radius Topology Database Administrator's Guide).

A client can update or import data in the database. Normally this data is stored as plain geometry data, but Radius Topology can convert these non-topological data into sets of topological primitives and relationships between the primitives. The primitives are held in standard tables in the database. Relationships are maintained through join tables, so enabling efficient querying of these relationships. The Radius Topology Triggers prevent adding errors when a user modifies existing data or adds data.

The feature tables or geometry tables that already exist, can be converted into topologically structured data tables. During this process a manifold ID and Topology ID are added to the concerning table and these ID's refer to the other tables that are constructed by Radius Topology. The structure of the tables that are created by Radius Topology and the links between these tables can be found in figure 2.2 (from Radius Topology Database Administrator's Guide).

Because it is sometimes desirable to build and hold independent sets of topological relationships within one dataset. An example can be a combination of real boundaries and administrative boundaries. If these datasets are independent, separate manifolds can be created. A manifold is thus some sort of independent set of relationships . Each manifold gets its own name and id.

During the creations of a manifold also the other tables are created that store the nodes, edges, faces and the 'edge to node', 'edge to edge', 'line to edge', 'face to edge' and 'area to face' relationships. These tables are initially empty, but are filled when topology is added (constructed using the geometry data) by Radius Topology.

The links between the geometry table and the primitives can be found following the lines in figure 2.2. The Topology ID column from the geometry table refers to the id from the lsl_topo$nnn table where the nnn stands for the number of the manifold. The lsl_topo table is related to the lsl_topo_part$nnn table. This table contains id's of all the lines and areas (features). The geometries of the lines and areas can be found using the reference and primitive tables (only geometries of nodes and edges are stored). For example the geometry of an endnode of a line is found by following the reference from the ID in the LSL_Topo_Part table to the Line ID in the LSL_Line_To_Edge table and further from the corresponding Edge ID to the Edge ID in the LSL_Edge table to the Edge_ID in the LSL_Edge_To_Node table. Then the corresponding Node ID can be found and with this Node ID finally the corresponding geometry (x,y) can be found in the LSL_Node table.

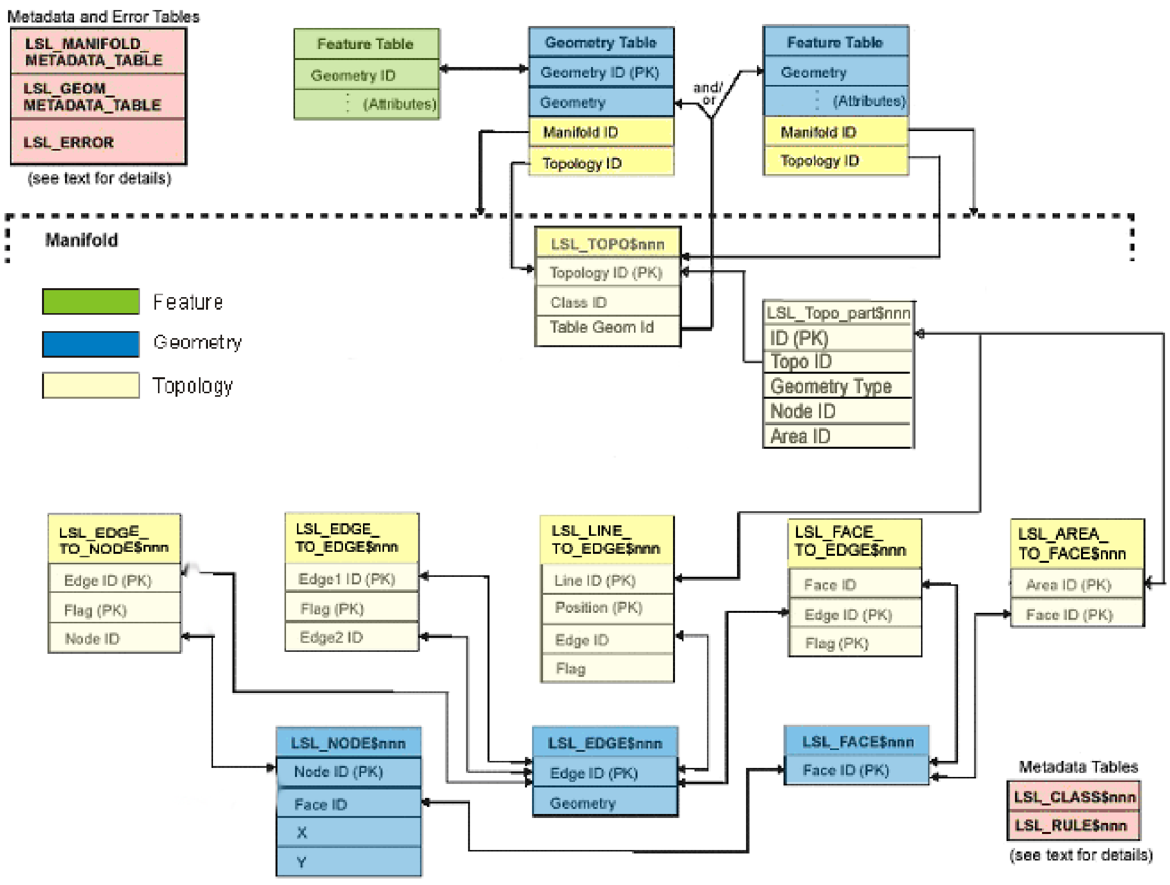


figure 2.2: structure of radius topology database tables
(Source: Radius Topology Database Administrator's Guide)

The functions of all the reference tables are described hereafter to give an idea of the structure of Radius Topology tables. The edge_to_node table is used to find nodes at the start and end of an edge and on the opposite to find edges that emanate from (or end at) a node. The edge_to_edge table defines the ordered set of directed edges that define a line or ring (the edge-to-edge-clockwise and edge-to-edge-anticlockwise references). The line_to_edge table implements references between lines/areas and Edges. The face_to_edge table is used to find the faces on either side of an edge, or the face that contains an edge and on the opposite to find the edges on the boundary of a face or inside a face. The lsl_area_to_face table is used to find the faces that define an area and on the opposite to find areas that share a face. In the Administrator's guide of Radius Topology a more detailed explanation of the reference tables and examples can be found.

In figure 2.2 also a metadata table can be found. This table contains information about the classes of the features and the rules.

After creating a new manifold, classes should also be created into which feature geometries will be topologically structured. This is because triggers added to the feature table when one activates a column, will only function correctly when the classes exist in the manifold. The rules should be applied when data is added or modified, so the topological structure and the integrity of the data stay intact.

2.2 Installing Radius Topology and converting data

The installation process of Radius Topology can be done quickly if the release notes are studied before. There are some pre-installation requirements mentioned in this release note. One of the pre-installation is a particular hard- and software configuration (or something similar). The hard- and software that is used for this research consist of a 64-bit Sun SPARC hardware platform and a 64-bit version of Oracle9*i* Database Enterprise Edition with Spatial options. The Radius Topology Configuration Manager requires JRE 1,3 and the Oracle 9*i* JDBC driver and SQLJ runtime. Other pre-installation requirements are correctly configured Oracle External Procedures (EXTPROC). The external procedures installed and enabled for this research can be found in appendix A. When the pre-installation requirements are correct, the installation process can start. A more detailed description of this process can be found in the release notes of Radius Topology [3].

After installation has succeeded, Radius Topology is ready for use. Data can be loaded in the database and after loading, the (plain geometry) data can be converted into topologically structured data. But before converting the data, it has to be sure that the data does not contain any arcs because Radius Topology does not support this. First the arcs have to be converted into straight line segments. For the data which is used for this test this converting process was done with a tolerance of 1 mm (to make sure that also very small arcs are not converted into points, which will happen if the complete arc lies within the tolerance).

Using the Radius Topology Configuration Manager one can now convert the plain geometry into topological primitives and relationships. The configuration manager has a very friendly and understandable user interface (especially in comparison with the Oracle sqlplus interface).

When the 1:10.000 plain geometry data of Flevoland (about 350.000 objects) was converted into topological structured data, the converting process stopped due to memory leaks in Oracle library functions. According to LaserScan a patch could be installed and this should solve the lack of memory problem. Or we could split up the data into smaller pieces (40.000 objects) and convert these pieces one by one. Eventually the 9.2.0.2 patch set for Oracle 9.0.2.2 was installed (even though we used Oracle 9.0.2.1) and again an attempt was made to convert the data of Flevoland into topological structured data. The create topology process for the province Flevoland now stopped (after eight days) at 44% because Oracle crashed (internal error). This was done using the version 1.0 release of Radius Topology.

To check if the create topology process went fine for the 44% of the data that was structured, a comparison was made between the plain geometry and the topology primitives. The dataset of Almere consist of boundaries and parcels. The boundaries (plain geometry) of Almere were compared with the edges (topological geometry), the relationship between these two geometries has to be a 1:1 relationship. A small difference was found within the data and this difference was not due to a mistake from Radius Topology, but it was a fault in the original data of the Cadastre.

A boundary between two parcels was missing in the data, so that two parcels shared the same face. However this problem was already known at the Cadastre, but still a good example of the way topology helps to improve data quality.

Meanwhile version 2.0 of Radius Topology was ready for use. We decided to go on with the 1:50.000 dataset of Flevoland. With this data set and the new version of Radius Topology, the create topology process went fine and was much faster.

The conclusion may be drawn from these results that the convertion process from non-topological data into topological data went fine for Flevoland. Also loading large datasets goes fine, but the converting process of large datasets takes much time using version 1.0 of Radius Topology in combination with the used hardware configuration. The available version 2.0 of Radius Topology is much faster in converting data into topological primitives and references, but with the used hardware configuration for this test, it was still not optimal. Maybe later versions and a better hardware configuration will offer a solution.

3. Functionality of Radius Topology

In the previous chapter the implementation of Radius Topology in Oracle, the structure of Radius Topology and all its tables, the loading and converting of the data and the way that topological structured data is stored (using primitives and references) are described. What can be done with the topologically structured data is discussed in this chapter. Querying and editing of data are the most important functionality for this case-study so in 3.1 the Querying of the data is discussed, followed by the editing possibilities and rules in 3.2.

3.1 Querying

Spatial queries that can be asked using a geo-DBMS, can be translated in operations. The operations provided by Radius Topology are:

Share_node	AnyInteract	Inside
Share_edge	AnyInteract_node	Inside_point
Share_outer_edge	AnyInteract_edge	Inside_line
Share_face	AnyInteract_face	Inside_area
		Inside_line_area

Also a combination of these operations can be made. The combinations that are valid are union, intersect and minus.

The operations can be invoked by the command LSL_TOPO_RELATE. An index on the topology_id column of the geometry table is required for this command. More details about the LSL_TOPO_RELATE command and the different operations can be found in the users guide of Radius Topology [5]. Most of the operations can be compared with the Oracle SDO_RELATE command possibilities.

There is a difference between the topo_relate and the sdo_relate command and this difference concerns the subject of the query. The topo_relate command uses an existing subject, whereas the sdo_relate command can also use a self-specified subject. So in the queries that are going to be used in the performance test (see a-e, chapter 1), most of the subjects are self defined. Only the 'adjacency query' (f, chapter 1) uses an existing subject. Considering this, only the share_outer_edge is of interest for this case study and all the other operations that are provided by Radius Topology are of minor interest. The share_outer_edge operation is a restricted share_edge relationship. Edges of the inner rings of area features are not queried and therefore this operation can be described as a test for adjacency. Of course many other interesting relationship queries can be made by these operations, but besides the adjacency query, none other relationship queries are asked within this research.

3.2 Editing

The way that nodes are generated when a feature is edited or created in a dataset that is already topologically structured, is governed by a set of topological rules. Radius Topology allows these rules to determine how the primitives will be represented topologically when they interact.

- When a S**hare Node** rule is defined, new features will share existing feature nodes if they are within a defined tolerance.

- If a N**ode-Split-Edge** rule is defined, the node of a new feature can split the edge of an existing feature if it is within tolerance, causing the features to share a node. Likewise, an existing node can split the edge of a new feature if within tolerance.
- If E**dge-Split-Edge** rule is defined, the edge of a new feature can split the edge of an existing feature if it intersects or comes within tolerance, creating a new shared node.

All the rules and tolerances can be accessed via the configuration manager. Data integrity problems are eliminated by the central application of these topological rules, against which data is automatically validated and cleaned.

If a tolerance has not been set for the Node-Split-Edge rule, it inherits the tolerance defined for the Edge-Split-Edge rule. And if the Share Node rule does not have a defined tolerance, it inherits the tolerance defined for the Node-Split-Edge rule. An explicit tolerance can override an implicit one, but can only be bigger, not smaller.

The data of Almere which is used for this case-study was also converted into topologically structured data. This converting process was done according to the topological rules determined using the configuration manager. For this test data only was loaded and converted into topology and adaptations afterwards were not made. So during the rest of the time these rules were not of interest anymore. But if one creates new data or alters the existing data these changes always run according to the topological rules.

4. Storage requirement analysis

Non-topological structured data and topological structured data have different storage requirements for the same dataset; plain geometry versus primitives and relationships. The proportion between these different storage requirements depends on the complexity of the dataset self, especially the number of points per edge. Using the topologically structured data, the primitives and relationships have to be stored once. Using the non-topologically structured data, the geometry of each feature has to be stored separate, so boundaries are stored twice. So the more nodes per feature the less disk space needed for the storage requirements using the topological structure in comparison with non-topological data storage. This is also illustrated in figure 4.1 below.

The benefit of using topological structured data is larger when the dataset contains objects that have a complex shape instead of objects that have a simple shape. This is because the larger redundancy in the data storage (using the non-topological structure) of the complex objects in stead of the simple objects. This implies that the storage requirements for a large scale data set with many simple objects (many boundaries with only four nodes) is much more using the topology structure in stead of the 'normal' plain geometry.

figure 4.1: left the non-topological storage structure using separate geometry for each feature and right the topological storage structure only using nodes and edges as basis.

To make a good comparison between the storage requirements for the topology structured data as well as for the plain geometry data, not only looking at the tables filled with primitives, relationships and geometry is sufficient, but also the different indexes have to be taken into account.

The storage requirements for the geometry of the 1:50.000 data set used for this test for the plain geometry and the geometry of the topology primitives are:

geometry, polygon column:	15.38 Mb
topology, edge column:	17.95 Mb
topology, node column:	1.89 Mb

So the total storage requirement for plain geometry is 15.4 Mb and for topology primitives it is 19.8 Mb. However these storage requirements are only for the geometry and not for:

- the indexes (plain geometry and topology);
- references (topology);
- ID's (topology).

The storage requirements for the index on the plain geometry data is 2,61 Mb (R-tree index). So the total storage requirements for the **plain geometry** variant are **18.0 Mb**.

The storage requirements for the topolgy tables (primitives and ID's (overhead)) and the associated indexes are:

Tabel	Mb	Index Mb	
node	3.57	3.47	
edge	27.27	5.18	
face	0.61	3.60	
topo	1.73	14.91	(including the table lsl_topo_part and the primitive index)
edge_to_node	5.46	16.74	
edge_to_edge	5.77	15.90	
face_to_edge	5.65	16.62	
line_to_edge	5.80	12.80	
area_to_face	0.69	2.70	
	56.55	91.92	

The total storage requirements for the **topology variant** becomes then **148.5 Mb**. This is more than eight times as much!

The used (large scale) dataset can be seen as a worst case scenario for Radius Topology, so there are probably datasets for which the storage requirements are less using Radius Topology in stead of non-topologically structured data. Where the break even point lies that topological structured data is stored more efficient than non-topological data, is not investigated. If there is a break even point at all, because the overhead (for example the many tables with the same ID's) using the topological storage structure is very large. So maybe it is even possible that in the current version of Radius Topology the storage requirements for the topological structured data are always larger than the storage requirements for the plain geometry.

5. Performance tests with Radius Topology

The actual performance test of Radius Topology including the comparison between the plain geometry version and the topology version, can only be done if there is a plain geometry table and a topology table. The table filled with topological structured data of faces is the table lsl_face$1 (Flevoland has manifold number 1) and the table filled with non-topological structured data of areas is flevo_areas.

The non-topological structured data table flevo_areas, only consist of four colums (OID, Shape, Topo_ID and Mani_ID). In the topological structured data table lsl_face$3, there is not geometry stored, but only the face_ID and the geometry version. So when questions concerning geometry are asked, the geometry has to be constructed real time using the ID's and reference tables. In chapter 4 it was clear that the storage requirements for the topology version were very large, this does slow down the process of constructing the geometry real time.

The questions to be asked for the performance test are:

- Are the results from some spatial questions (queries a-f in chapter 1) the same using topological structured data instead of non-topological structured data (redundant storage of polygons)?
- Are the results of the spatial questions (queries a-f in chapter 1) obtained faster using topological structured data instead of non-topological structured data?

Questions a to e from chapter 1, concern geometry. A feature that is specified in the questions is the domain of the query and is not stored topologically in an existing table. That is why the geometry has to be really constructed if the topological structured data is used. This takes extra time in comparison to the flevo_areas table which already contains a shape column filled with the geometry. The domain of question f is an existing object within the topology structure and the answer on this question can be found using the references when the topological structured data is used.

For the questions a to e the sdo_relate function is used and for question f the topo_relate function is used. Using the topo_relate function, only relations between existing features in the topological structured data can be found. With the sdo_relate function a feature can be defined and relationships between this feature and a dataset can be found. So with the sdo_relate function, relationships between a specified feature and for example the feature geometries constructed using the LSL_FACE$3 table can be found. Using the LSL_FACE$3 table the topological structure is used.

The main subject of the performance test is the time and result comparison between answers on the queries a to f that are based on topological data versus non-topological data. The questions that are eventually asked are:

```
QUERY 1.1: select in relatively empty rectangle
QUERY 1.2: select in relatively crowded rectangle
QUERY 2.1: select in relatively empty polygon
QUERY 2.2: select in relatively crowded polygon
QUERY   3: clip to polygon with two holes
QUERY 4.1: select on long line through relatively empty area
QUERY 4.2: select on long line through relatively crowded area
QUERY 5.1: select on 3 points in relatively empty areas
QUERY 5.2: select on 6 points in relatively crowded areas
QUERY 6.1: select all objects adjacent to a simple area feature
QUERY 6.2: select all objects adjacent to a complex area feature
```

An example of the script of query 1.1 is given in figure 5.1. The script for all questions can be found in appendix B.

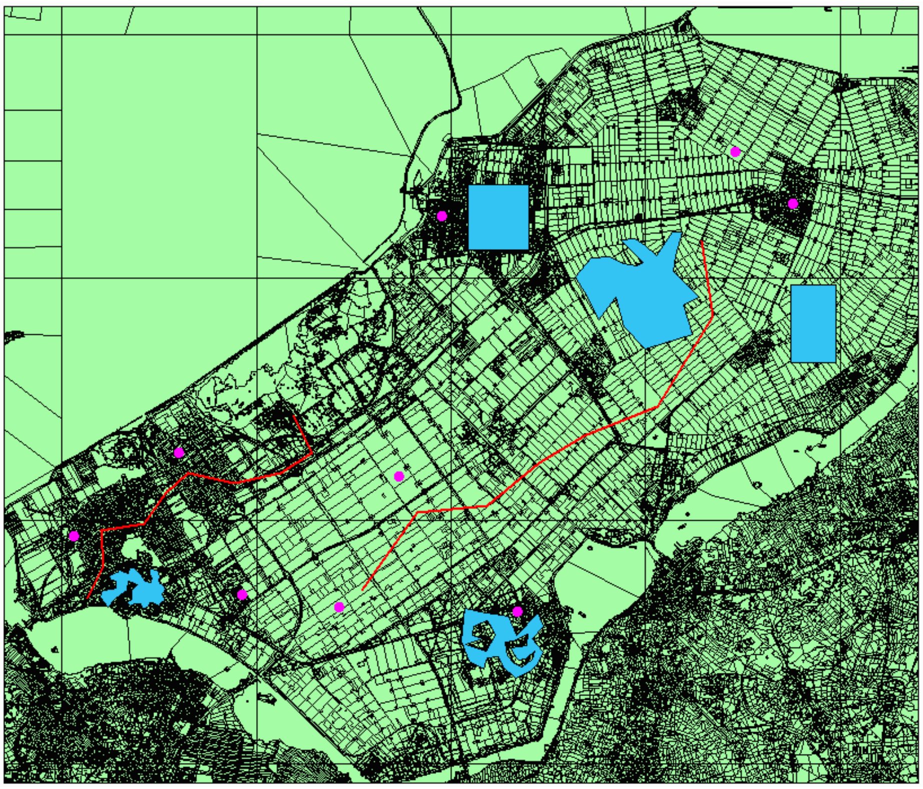
```
-- QUERY 1.1: select in relatively empty rectangle

-- Geometry version
select oid from flevo_areas
   where mdsys.sdo_relate (shape,
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,3),
         mdsys.sdo_ordinate_array(177450.82,495672.131,
         179811.475,499672.131)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

-- Topology version
select oid from flevo_areas
    where mdsys.sdo_relate (lsl_topo_rwo.get_geom(mani_id,topo_id),
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,3),
         mdsys.sdo_ordinate_array(177450.82,495672.131,
         179811.475,499672.131)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';
```

figure 5.1: script of query 1.1

Figure 5.2 is added to give an idea of the used query elements. This is the same picture as on the cover.



Figuur 5.2: overview of the query elements.
The points, lines, simple and complex polygons can be seen.

Needed time

The times needed for finding answers to the questions are mentioned in table 5.1. These times are the times of the 'hot' state of the database. This means that the queries are already asked once before, so the needed data is already in the memory. If the data is already in the memory, the answers on the questions are found faster. All queries are asked twice to make sure that all queries are in a 'hot' situation.

Table 5.1: times (in seconds) needed for finding answers on the different queries for the performance test.

Query	Query type	# rows returned	Plain geometry version	Topology version	Relation of time
1.1	Rectangle	91	00.07	00.16	1:2
1.2	Rectangle	962	00.29	00.43	2:3
2.1	Polygon	296	00.28	01.02	1:4
2.2	Polygon	573	00.64	02.25	1:4
3	Clip	339	24.74	42.34	1:2
4.1	Polyline	106	00.27	01.32	1:5
4.2	Polyline	225	00.57	02.59	1:5
5.1	Points	3	00.13	00.19	2:3
5.2	Points	6	00.22	00.38	1:2
6.1	Adjacency	5	00.22	00.04	5:1
6.2	adjacency	41	01.22	00.12	10:1

There are huge differences between the times that are needed for finding answers on the questions. For queries 1-5 the needed time for topology is much more than the needed time for plain geometry questions. Query 6.1 and 6.2 concern only topological relations and are therefore much faster using the topological structured data.

From these results it is very obvious that (using Radius Topology) the topological questions are much faster and geometry questions are much slower using topological structured data. This is only logical considering the real time geometry construction for geometry questions and only looking up references for the topology questions.

As can be seen in table 5.1 the relationships between the times for the geometry version and the topology version is not everywhere the same. The relationship for some queries is 1:2 but for other queries it is 1:5. Why this is not a fixed relationship is not investigated, maybe this is interesting to find out for other people.

Looking at table 5.1 one can easily conclude that Radius Topology is not that good at all for geometry queries, however for topology queries Radius Topology is much faster. The question now is which queries are more common to ask? Radius Topology is for the performance not an improvement if an organisation often has geometry queries, but the data quality and consistency is a lot better using topological structured data. If an organisation is very interested in topology queries, Radius Topology can be a solution.

6. Conclusions and recommendations

The expectation at the start of this case study was that spatial relationship queries should be faster to carry out using Radius Topology instead of non-topologically structured data (because the information can be found without having to perform direct geometry comparisons), but the opposite appeared to be true. This is (afterwards) logical, because the geometry had to be computed real time using topological structured data. This is a disadvantage of Radius Topology. Radius Topology has also great advantages (data quality and consistency), but do these advantages of Radius Topology counterbalance the disadvantages?

6.1 Conclusions

The **functionality** of Radius Topology is good. Many question can be done using Radius Topology. These questions are similar with the sdo_relate possibilities. Also the integrity of the topological structure is guaranteed by the topological rules. The converting process is improved with the new version of Radius Topology but still not optimal (with the used hardware configuration). This process maybe can be improved in later versions to make Radius Topology a good program.

The performance tests give not a positive view of Radius Topology, but it has to be said that the tests for this study are the worst case scenario for Radius Topology.

First of all the **storage requirements** for the topological structured dataset is much more than the requirements for the non-topological dataset, but this is dependent of the structure of the original dataset itself. The used dataset for this study had a very simple structure and the differences in storage requirements were therefore very large. The differences in the requirements between the two versions are probably smaller if another more complex dataset is used, but it is unknown if there exists a break even point.

Comparing the results of the **actual performance test**, one thing seems clear: the topological structured data is much slower in finding answers concerning geometry questions such as question 1-5 (a-e in chapter 1). Finding answers on question 6 (f in chapter 1) concerning topological relationships, Radius Topology is much faster. An explanation is simple: When finding answers on questions concerning geometry the topological structured data has to follow the references and after that the geometry has to be constructed real time. Using non-topological data maybe more stored data has to be searched trough (which wasn't even the case for this study), but the geometry already exist and does not have to be constructed real time.

The relationships between the times for the two scenarios were not the same for all questions. Finding out where these differences come from could for example be done in another case study.

Finally the conclusion can be made that the functionality of version 2.0 of Radius Topology still should be made better. Many organisations are more interested in geometry queries than in topology queries, for example the question which parcels overlap with a planned new road. So the performance concerning geometry queries should improve.

Also the storage requirements of Radius Topology should be improved to make the performance just as good as the performance for the non-topologically structured data, especially for less suitable datasets (with many simple features).

All these necessary improvements do not alter the fact that Radius Topology is the first program that manages topological rules and that it is a good beginning of a new kind of data storage! The quality improvement of the data and the consistency of the data is a very great advantage for managing purposes in comparison with the plain geometry data storage, but for organisations that are interested more in querying data (geometry queries) than managing data, Radius Topology does not offer an improvement at this moment.

6.2 Recommendations

In this case study a few things are become clear. Still there are some questions that could be investigated after this study.

One of these questions concerns the break even point of the storage requirements for plain geometry data and topologically structured data. Is there a break even point at all en if yes, which kind of dataset (complexity) should be used? It is not even sure that a break even point exists with this version of Radius Topology. The number of tables and the amount of overhead (ID's stored twice or more and many indexes are necessary) is very large.

Another point of interest is the relationship of times between the two versions of data storage. It is not a fixed relation as can be seen in table 5.1. Is there a logical explanation for this?

In the end a recommendation can be made of following the developments of Radius Topology. This is one of the first versions of programs that are able to manage topological structured data. There are a lot of improvements possible in the future.

References

1) Spatial DBMS testing with data from the Cadastre and TNO NITG, by drs. T.P.M. Tijssen, drs. C.W. Quak, prof. dr.ir. P.J.M. van Oosterom. Delft, march 2001.

2) The Balance between Geometry and Topology, by Peter van Oosterom, Jantien Stoter, Wilko Quak and Sisi Zlatanova. Delft, 2001.

3) Radius Topology 1.0 (v1-14) Release Notes. Laser-Scan Limited 2002.

4) Radius Topology Administrator's Guide, Issue 1.1 for radius Topology Version 1.0, October 2002. Laser-Scan Limited.

5) Radius Topology User's Guide and Reference, Issue 1.1 for Radius Topology Version 1.0, October 2002. Laser-Scan Limited.

Appendix A – External procedures

In the 'listener.ora' file:

```
CALLOUT_LISTENER =
   (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC) (KEY = EXTPROC64) )
   )

SID_LIST_CALLOUT_LISTENER =
   (SID_LIST =
      (SID_DESC =
         (SID_NAME = PLSExtProc)
         (ORACLE_HOME = /opt/oracle/product/9.2.0.1_64)
         (PROGRAM = extproc)
         (ENVS =
"EXTPROC_DLLS=/opt/oracle/product/9.2.0.1_64/lsl/lib/lsl_topo_lib.so")
      )
   )
```

In the 'tnsnames.ora' file:

```
EXTPROC_CONNECTION_DATA =
   (DESCRIPTION =
      (ADDRESS_LIST =
         (ADDRESS = (PROTOCOL = IPC) (KEY = EXTPROC64))
      )
      (CONNECT_DATA =
         (SID = PLSExtProc)
         (PRESENTATION = RO)
      )
   )
```

Appendix B – Query script

Per query the script for the geometry version and the script for the topology version can be found.

```
--################################################################################
-- QUERY 1.1: select in relatively empty rectangle

-- Geom
select oid from flevo_areas
   where mdsys.sdo_relate (shape,
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,3),
         mdsys.sdo_ordinate_array(177450.82,495672.131, 179811.475,499672.131)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

-- Topo
select oid from flevo_areas
    where mdsys.sdo_relate (lsl_topo_rwo.get_geom(mani_id,topo_id),
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,3),
         mdsys.sdo_ordinate_array(177450.82,495672.131, 179811.475,499672.131)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

--################################################################################
-- QUERY 1.2: select in relatively crowded rectangle

-- Geom
select oid from flevo_areas
   where mdsys.sdo_relate (shape,
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,3),
         mdsys.sdo_ordinate_array(160860.656,501442.623, 164008.197,504786.885)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

-- Topo
select oid from flevo_areas
    where mdsys.sdo_relate (lsl_topo_rwo.get_geom(mani_id,topo_id),
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,3),
         mdsys.sdo_ordinate_array(160860.656,501442.623, 164008.197,504786.885)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

--################################################################################
-- QUERY 2.1: select in relatively empty polygon

-- Geom
select oid from flevo_areas
   where mdsys.sdo_relate (shape,
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,1),
         mdsys.sdo_ordinate_array( 168860.656,497508.197, 168467.213,499016.393,
            167614.754,497836.066, 167024.59,499016.393,  166368.852,500065.574,
            167221.311,501049.18,  168204.918,501114.754, 168795.082,500721.311,
            169581.967,500655.738, 169844.262,500786.885, 168729.508,501967.213,
            169450.82,502032.787,  169975.41,501704.918,  170565.574,501180.328,
            171352.459,502360.656, 171877.049,502360.656, 171549.18,501180.328,
            171418.033,500327.869, 172795.082,499016.393, 171942.623,498622.951,
            172401.639,497114.754, 170893.443,496655.738, 169975.41,496327.869,
            168860.656,497508.197)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

-- Topo
select oid from flevo_areas
    where mdsys.sdo_relate (lsl_topo_rwo.get_geom(mani_id,topo_id),
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,1),
         mdsys.sdo_ordinate_array( 168860.656,497508.197, 168467.213,499016.393,
            167614.754,497836.066, 167024.59,499016.393,  166368.852,500065.574,
            167221.311,501049.18,  168204.918,501114.754, 168795.082,500721.311,
            169581.967,500655.738, 169844.262,500786.885, 168729.508,501967.213,
            169450.82,502032.787,  169975.41,501704.918,  170565.574,501180.328,
            171352.459,502360.656, 171877.049,502360.656, 171549.18,501180.328,
            171418.033,500327.869, 172795.082,499016.393, 171942.623,498622.951,
            172401.639,497114.754, 170893.443,496655.738, 169975.41,496327.869,
            168860.656,497508.197)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

--##########################################################################
-- QUERY 2.2: select in relatively crowded polygon

-- Geom
select oid from flevo_areas
   where mdsys.sdo_relate (shape,
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,1),
         mdsys.sdo_ordinate_array( 143167.025,483183.015, 143554.018,483464.465,
            143577.472,483898.366, 143870.648,484214.996, 143425.02,484402.629,
            143378.112,483945.274, 142791.759,483534.827, 142357.858,483030.564,
            142170.226,483276.832, 141970.866,483546.554, 142052.955,483921.82,
            142604.126,483957.001, 142791.759,484027.363, 142768.305,484414.356,
            142545.491,484449.537, 142404.767,484625.443, 142662.762,484789.622,
            142862.122,484871.711, 142967.665,484953.8,   143213.933,484824.803,
            143413.293,484777.895, 143460.201,485106.252, 143706.469,485071.071,
            143917.556,484824.803, 144023.1,484742.713,   144175.552,484543.353,
            144269.368,484402.629, 144574.271,484332.266, 144644.634,484543.353,
            144527.363,484801.349, 144503.909,485012.436, 144926.083,484988.981,
            145008.172,484625.443, 144961.264,484426.083, 145031.626,484250.177,
            145207.532,484226.723, 145313.076,483968.728, 145230.986,483781.095,
            145137.17,483699.006,  145242.713,483534.827, 145254.44,483464.465,
            145242.713,483382.375, 145090.262,483276.832, 144984.718,483100.926,
            144879.175,483194.742, 144656.361,483300.286, 144503.909,483300.286,
            144433.547,483089.199, 144410.093,483054.018, 144292.822,482983.655,
            144128.643,482983.655, 144081.735,483065.745, 144058.281,483206.469,
            143741.651,483241.651, 143542.291,483100.926, 143401.566,483229.924,
            143167.025,483183.015)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

-- Topo
select oid from flevo_areas
    where mdsys.sdo_relate (lsl_topo_rwo.get_geom(mani_id,topo_id),
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,1),
         mdsys.sdo_ordinate_array( 143167.025,483183.015, 143554.018,483464.465,
            143577.472,483898.366, 143870.648,484214.996, 143425.02,484402.629,
            143378.112,483945.274, 142791.759,483534.827, 142357.858,483030.564,
            142170.226,483276.832, 141970.866,483546.554, 142052.955,483921.82,
            142604.126,483957.001, 142791.759,484027.363, 142768.305,484414.356,
            142545.491,484449.537, 142404.767,484625.443, 142662.762,484789.622,
            142862.122,484871.711, 142967.665,484953.8,   143213.933,484824.803,
            143413.293,484777.895, 143460.201,485106.252, 143706.469,485071.071,
            143917.556,484824.803, 144023.1,484742.713,   144175.552,484543.353,
            144269.368,484402.629, 144574.271,484332.266, 144644.634,484543.353,
            144527.363,484801.349, 144503.909,485012.436, 144926.083,484988.981,
            145008.172,484625.443, 144961.264,484426.083, 145031.626,484250.177,
            145207.532,484226.723, 145313.076,483968.728, 145230.986,483781.095,
            145137.17,483699.006,  145242.713,483534.827, 145254.44,483464.465,
            145242.713,483382.375, 145090.262,483276.832, 144984.718,483100.926,
            144879.175,483194.742, 144656.361,483300.286, 144503.909,483300.286,
            144433.547,483089.199, 144410.093,483054.018, 144292.822,482983.655,
            144128.643,482983.655, 144081.735,483065.745, 144058.281,483206.469,
            143741.651,483241.651, 143542.291,483100.926, 143401.566,483229.924,
            143167.025,483183.015)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

--##############################################################################
-- QUERY 3: clip to polygon with two holes

drop table query31;
drop table query32;

-- Geom
create table query31 as select
   sdo_geom.sdo_intersection (shape,
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,1, 29,2003,1, 47,2003,1),
         mdsys.sdo_ordinate_array( 163250.772,481287.388, 162780.571,481248.204,
            162907.917,480542.903, 163260.568,480131.477, 163662.198,479984.539,
            164200.97,480395.965,  164308.725,480797.595, 163838.524,480376.373,
            163495.669,480327.394, 163211.589,480689.84,  163103.834,480964.124,
            163887.503,481071.879, 163926.686,481689.018, 163250.772,481287.388,
            161076.092,481600.855, 161644.251,481414.734, 160772.42,481111.062,
            161791.189,480807.391, 162241.799,481591.059, 162085.065,481982.894,
            161849.964,482355.136, 161183.846,482061.261, 161076.092,481600.855,
            160537.319,481306.98,  160713.645,482972.276, 161781.393,482776.358,
            162976.488,482590.237, 163378.118,481630.243, 164475.254,482766.562,
            164739.743,482080.852, 164200.97,481414.734,  164739.743,480738.82,
            164387.092,480072.701, 164063.828,479749.438, 163309.547,479416.379,
            162692.408,479964.947, 162418.124,480493.923, 161869.556,480444.944,
            161703.026,479866.988, 160958.541,480268.619, 160537.319,481306.98)),
   0.00001) clip_geom
from flevo_areas where mdsys.sdo_relate (shape,
   mdsys.sdo_geometry(2003,NULL,NULL,
      mdsys.sdo_elem_info_array(1,1003,1, 29,2003,1, 47,2003,1),
      mdsys.SDO_ORDINATE_ARRAY( 163250.772,481287.388, 162780.571,481248.204,
            162907.917,480542.903, 163260.568,480131.477, 163662.198,479984.539,
            164200.97,480395.965,  164308.725,480797.595, 163838.524,480376.373,
            163495.669,480327.394, 163211.589,480689.84,  163103.834,480964.124,
            163887.503,481071.879, 163926.686,481689.018, 163250.772,481287.388,
            161076.092,481600.855, 161644.251,481414.734, 160772.42,481111.062,
            161791.189,480807.391, 162241.799,481591.059, 162085.065,481982.894,
            161849.964,482355.136, 161183.846,482061.261, 161076.092,481600.855,
            160537.319,481306.98,  160713.645,482972.276, 161781.393,482776.358,
            162976.488,482590.237, 163378.118,481630.243, 164475.254,482766.562,
            164739.743,482080.852, 164200.97,481414.734,  164739.743,480738.82,
            164387.092,480072.701, 164063.828,479749.438, 163309.547,479416.379,
            162692.408,479964.947, 162418.124,480493.923, 161869.556,480444.944,
            161703.026,479866.988, 160958.541,480268.619, 160537.319,481306.98)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE' ;
select count(*) from query31;

-- Topo
create table query32 as select
   sdo_geom.sdo_intersection (lsl_topo_rwo.get_geom(mani_id,topo_id),
      mdsys.sdo_geometry(2003,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1003,1, 29,2003,1, 47,2003,1),
         mdsys.sdo_ordinate_array( 163250.772,481287.388, 162780.571,481248.204,
            162907.917,480542.903, 163260.568,480131.477, 163662.198,479984.539,
            164200.97,480395.965,  164308.725,480797.595, 163838.524,480376.373,
            163495.669,480327.394, 163211.589,480689.84,  163103.834,480964.124,
            163887.503,481071.879, 163926.686,481689.018, 163250.772,481287.388,
            161076.092,481600.855, 161644.251,481414.734, 160772.42,481111.062,
            161791.189,480807.391, 162241.799,481591.059, 162085.065,481982.894,
            161849.964,482355.136, 161183.846,482061.261, 161076.092,481600.855,
            160537.319,481306.98,  160713.645,482972.276, 161781.393,482776.358,
            162976.488,482590.237, 163378.118,481630.243, 164475.254,482766.562,
            164739.743,482080.852, 164200.97,481414.734,  164739.743,480738.82,
            164387.092,480072.701, 164063.828,479749.438, 163309.547,479416.379,
            162692.408,479964.947, 162418.124,480493.923, 161869.556,480444.944,
            161703.026,479866.988, 160958.541,480268.619, 160537.319,481306.98)),
   0.00001) clip_geom
```

```
from flevo_areas where mdsys.sdo_relate (lsl_topo_rwo.get_geom(mani_id,topo_id),
   mdsys.sdo_geometry(2003,NULL,NULL,
      mdsys.sdo_elem_info_array(1,1003,1, 29,2003,1, 47,2003,1),
      mdsys.SDO_ORDINATE_ARRAY(    163250.772,481287.388, 162780.571,481248.204,
            162907.917,480542.903, 163260.568,480131.477, 163662.198,479984.539,
            164200.97,480395.965,  164308.725,480797.595, 163838.524,480376.373,
            163495.669,480327.394, 163211.589,480689.84,  163103.834,480964.124,
            163887.503,481071.879, 163926.686,481689.018, 163250.772,481287.388,
            161076.092,481600.855, 161644.251,481414.734, 160772.42,481111.062,
            161791.189,480807.391, 162241.799,481591.059, 162085.065,481982.894,
            161849.964,482355.136, 161183.846,482061.261, 161076.092,481600.855,
            160537.319,481306.98,  160713.645,482972.276, 161781.393,482776.358,
            162976.488,482590.237, 163378.118,481630.243, 164475.254,482766.562,
            164739.743,482080.852, 164200.97,481414.734,  164739.743,480738.82,
            164387.092,480072.701, 164063.828,479749.438, 163309.547,479416.379,
            162692.408,479964.947, 162418.124,480493.923, 161869.556,480444.944,
            161703.026,479866.988, 160958.541,480268.619, 160537.319,481306.98)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE' ;
select count(*) from query32;
commit;

--##############################################################################
-- QUERY 4.1: select on long line through relatively empty area

-- Geom
select oid from flevo_areas
   where mdsys.sdo_relate (shape,
      mdsys.sdo_geometry(2002,NULL,NULL,
         mdsys.sdo_elem_info_array(1,2,1),
         mdsys.sdo_ordinate_array( 155418.033,483934.426, 158237.705,487934.426,
            161844.262,488262.295, 164532.787,490491.803, 166827.869,491868.852,
            170631.148,493377.049, 173450.82,498032.787,  172860.656,501901.639)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

-- Topo
select oid from flevo_areas
   where mdsys.sdo_relate (lsl_topo_rwo.get_geom(mani_id,topo_id),
      mdsys.sdo_geometry(2002,NULL,NULL,
         mdsys.sdo_elem_info_array(1,2,1),
         mdsys.sdo_ordinate_array( 155418.033,483934.426, 158237.705,487934.426,
            161844.262,488262.295, 164532.787,490491.803, 166827.869,491868.852,
            170631.148,493377.049, 173450.82,498032.787,  172860.656,501901.639)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

--##############################################################################
-- QUERY 4.2: select on long line through relatively crowded area

-- Geom
select oid from flevo_areas
   where mdsys.sdo_relate (shape,
      mdsys.sdo_geometry(2002,NULL,NULL,
         mdsys.sdo_elem_info_array(1,2,1),
         mdsys.sdo_ordinate_array( 141319.672,483540.984, 142106.557,485114.754,
            142040.984,487016.393, 144204.918,487344.262, 145319.672,488918.033,
            146500,489967.213,     148860.656,489442.623, 151221.311,489967.213,
            152860.656,491016.393, 151877.049,492918.033)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

-- Topo
select oid from flevo_areas
   where mdsys.sdo_relate (lsl_topo_rwo.get_geom(mani_id,topo_id),
      mdsys.sdo_geometry(2002,NULL,NULL,
         mdsys.sdo_elem_info_array(1,2,1),
         mdsys.sdo_ordinate_array( 141319.672,483540.984, 142106.557,485114.754,
            142040.984,487016.393, 144204.918,487344.262, 145319.672,488918.033,
            146500,489967.213,     148860.656,489442.623, 151221.311,489967.213,
            152860.656,491016.393, 151877.049,492918.033)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';
```

```
--###############################################################################
-- QUERY 5.1: select on 3 points in relatively empty areas

-- Geom
select oid from flevo_areas
   where mdsys.sdo_relate (shape,
      mdsys.sdo_geometry(2005,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1,3),
         mdsys.sdo_ordinate_array(
            154237.705,483016.393, 157319.672,489704.918, 174631.148,506426.23)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

-- Topo
select oid from flevo_areas
   where mdsys.sdo_relate (lsl_topo_rwo.get_geom(mani_id,topo_id),
      mdsys.sdo_geometry(2005,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1,3),
         mdsys.sdo_ordinate_array(
            154237.705,483016.393, 157319.672,489704.918, 174631.148,506426.23)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

--###############################################################################
-- QUERY 5.2: select on 6 points in relatively crowded areas

-- Geom
select oid from flevo_areas
   where mdsys.sdo_relate (shape,
      mdsys.sdo_geometry(2005,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1,6),
         mdsys.sdo_ordinate_array(
            140598.361,486622.951, 146040.984,490950.82, 149254.098,483606.557,
            163418.033,482754.098, 159549.18,503147.541, 177581.967,503737.705)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

-- Topo
select oid from flevo_areas
   where mdsys.sdo_relate (lsl_topo_rwo.get_geom(mani_id,topo_id),
      mdsys.sdo_geometry(2005,NULL,NULL,
         mdsys.sdo_elem_info_array(1,1,6),
         mdsys.sdo_ordinate_array(
            140598.361,486622.951, 146040.984,490950.82, 149254.098,483606.557,
            163418.033,482754.098, 159549.18,503147.541, 177581.967,503737.705)),
   'MASK=ANYINTERACT QUERYTYPE=WINDOW') = 'TRUE';

--###############################################################################
-- QUERY 6.1: select all objects adjacent to a simple area feature
--            simple: oid=280112, topo_id=208481

-- Geom
select oid from flevo_areas
   where mdsys.sdo_relate(shape,(select shape from flevo_areas where oid=280112),
      'MASK=TOUCH QUERYTYPE=WINDOW') = 'TRUE';

-- Topo
select oid from flevo_areas
   where lslsys.lsl_topo_relate(TOPO_ID,208481,
      'SHARE_NODE MINUS SHARE_FACE',1) = 'TRUE';

--###############################################################################
-- QUERY 6.2: select all objects adjacent to a complex area feature
--            complex: oid=280049, topo_id=220200

-- Geom
select oid from flevo_areas
   where mdsys.sdo_relate(shape,(select shape from flevo_areas where oid=280049),
      'MASK=TOUCH QUERYTYPE=WINDOW') = 'TRUE';
```

```
-- Topo
select oid from flevo_areas
   where lslsys.lsl_topo_relate(TOPO_ID,220200,
      'SHARE_NODE MINUS SHARE_FACE',1) = 'TRUE';

--#############################################################################
spool off
exit
```

Appendix C – Storage requirements

The tables with there storage requirements:

```
SQL> select tablespace_name "TSpace", owner, table_name, (blocks*8/1024) "Size_Mb",
  2       blocks "Blocks", empty_blocks "Empty", Num_rows "Num_rows",
  3       chain_cnt "Chained",avg_row_len "AvRowLen",avg_space "FrSpace"
  4    from all_tables
  5    where (owner = 'FLEVO')
  6    order by owner,table_name
  7  ;

TSpace OWNER     TABLE_NAME                        Size_Mb  Blocks Empty Num_rows
Chained AvRowLen FrSpace
------ --------- -------------------------------- -------- ------- ----- -------- -
------ -------- -------
USERS  FLEVO     FLEVO_AREAS                         14.85    1901    18    41319
0      319    1077
USERS  FLEVO     FLEVO_AREASE                          .01       1    62       14
0      340    3262
USERS  FLEVO     FLEVO_AREASF                        14.86    1902    17    41333
0      319    1078
USERS  FLEVO     FLEVO_AREAST                        14.85    1901    18    41319
0      319    1077
USERS  FLEVO     FLEVO_EDGE_ONLY                     17.95    2297     6   106409
0      152     905
USERS  FLEVO     FLEVO_GEOM_ONLY                     14.63    1872    47    41319
0      314    1078
USERS  FLEVO     FLEVO_NODE_ONLY                      1.89     242    13    68203
0       24     829
USERS  FLEVO     FLEVO_PARCELS                       38.30    4903    24   111235
0      305    1100
USERS  FLEVO     LSL_AREA_TO_FACE$1                    .69      88    39    41319
0       13    1071
USERS  FLEVO     LSL_CLASS$1                           .01       1    62        1
0       25    8051
USERS  FLEVO     LSL_EDGE$1                          27.27    3490    29   106409
0      164    3011
USERS  FLEVO     LSL_EDGE_TO_EDGE$1                   5.77     738    29   212818
0       16    2752
USERS  FLEVO     LSL_EDGE_TO_NODE$1                   5.46     699     4   212818
0       15    2549
USERS  FLEVO     LSL_FACE$1                            .61      78    49    41333
0       11    1210
USERS  FLEVO     LSL_FACE_TO_EDGE$1                   5.65     723    44   212818
0       15    2686
USERS  FLEVO     LSL_LINE_TO_EDGE$1                   5.80     743    24   211850
0       19    2073
USERS  FLEVO     LSL_NODE$1                           3.57     457    54    68203
0       43    1397
       FLEVO     LSL_PRIM_IDX$1                                              188887
0       29
USERS  FLEVO     LSL_RULE$1                            .01       1    62        1
0       20    8056
USERS  FLEVO     LSL_TOPO$1                            .77      98    29    41319
0       15     927
USERS  FLEVO     LSL_TOPO_PART$1                       .96     123     4    44437
0       18     906
USERS  FLEVO     TOP50_FLEVO                         15.88    2033    14    41333
0      342    1084
```

The indexes on the tables with their storage requirements:

```
SQL> select index_type, index_name, leaf_blocks*8/1024 "Mb_in_Leaf_blk",
  2       table_name, tablespace_name "Tablespace"
  3    from all_indexes
  4    -- where (index_type != 'LOB') and
  5    where (owner = 'FLEVO')
  6    order by table_name, index_name
  7  ;

INDEX_TYPE INDEX_NAME                     Mb_in_Leaf_blk TABLE_NAME
Tablespace
---------- ------------------------------ -------------- -------------------------
------ ----------
DOMAIN     FLEVO_AREAS_RTIDX                             FLEVO_AREAS
FUNCTION-B FLEVO_FUNC_RT_IDX                             FLEVO_AREAS
ASED DOMAI
N
DOMAIN     LSL_FLEVO_AREAS_1_IDX                         FLEVO_AREAS
NORMAL     LSL_FLEVO_AREAS_1_TM                     1.52 FLEVO_AREAS
USERS
NORMAL     LSL_AREA_TO_FACE$1_PK                    1.59 LSL_AREA_TO_FACE$1
USERS
NORMAL     LSL_FACE_TO_AREA_IDX$1                   1.11 LSL_AREA_TO_FACE$1
INDX
NORMAL     LSL_CLASS$1_NAME                          .01 LSL_CLASS$1
USERS
NORMAL     LSL_CLASS$1_PK                            .01 LSL_CLASS$1
USERS
NORMAL     LSL_EDGE$1_PK                            2.46 LSL_EDGE$1
USERS
DOMAIN     LSL_EDGE_IDX$1                                LSL_EDGE$1
NORMAL     LSL_KEY$918647616776                     2.72 LSL_EDGE$1
INDX
NORMAL     LSL_EDGE2_TO_EDGE1_IDX$1                 9.60 LSL_EDGE_TO_EDGE$1
INDX
NORMAL     LSL_EDGE_TO_EDGE$1_PK                    6.30 LSL_EDGE_TO_EDGE$1
USERS
NORMAL     LSL_EDGE_TO_NODE$1_PK                   10.13 LSL_EDGE_TO_NODE$1
USERS
NORMAL     LSL_NODE_TO_EDGE_IDX$1                   6.61 LSL_EDGE_TO_NODE$1
INDX
NORMAL     LSL_FACE$1_PK                             .99 LSL_FACE$1
USERS
NORMAL     LSL_FACE_TO_EDGE$1_PK                   10.42 LSL_FACE_TO_EDGE$1
USERS
NORMAL     LSL_FACE_TO_EDGE_IDX$1                   6.20 LSL_FACE_TO_EDGE$1
INDX
NORMAL     LSL_EDGE_TO_LINE_IDX$1                   6.66 LSL_LINE_TO_EDGE$1
INDX
NORMAL     LSL_LINE_TO_EDGE$1_PK                    6.14 LSL_LINE_TO_EDGE$1
USERS
NORMAL     LSL_FACE_TO_NODE_IDX$1                    .00 LSL_NODE$1
USERS
NORMAL     LSL_KEY$229639288448                     2.09 LSL_NODE$1
INDX
NORMAL     LSL_NODE$1_PK                            1.38 LSL_NODE$1
USERS
DOMAIN     LSL_NODE_IDX$1                                LSL_NODE$1
IOT - TOP  SYS_IOT_TOP_27748                        9.66 LSL_PRIM_IDX$1
INDX
NORMAL     LSL_RULE$1_PK                             .01 LSL_RULE$1
USERS
NORMAL     LSL_TOPO$1_PK                            1.20 LSL_TOPO$1
USERS
DOMAIN     LSL_TOPO_IDX$1                                LSL_TOPO$1
NORMAL     LSL_AREA_TO_RING_IDX$1                    .06 LSL_TOPO_PART$1
INDX
```

```
NORMAL     LSL_NODE_TO_POINT_IDX$1                    .00 LSL_TOPO_PART$1
INDX
NORMAL     LSL_TOPO_PART$1_PK                        1.20 LSL_TOPO_PART$1
USERS
DOMAIN     LSL_TOPO_PART_IDX$1                            LSL_TOPO_PART$1
NORMAL     LSL_TOPO_TO_PART_IDX$1                    1.27 LSL_TOPO_PART$1
INDX
```

The storage requirements of the tables in segments:

```
SQL> select owner, table_name, column_name, segment_name,
  2      (select sum(bytes/(1024*1024))
  3         from sys.dba_extents e where e.segment_name = l.segment_name)
"Size_in_mb",
  4      (select unique e.tablespace_name
  5         from sys.dba_extents e where e.segment_name = l.segment_name) "Tspace"
  6    from all_lobs l
  7     where owner = 'FLEVO'
  8    order BY owner,table_name,column_name,segment_name
  9  ;

OWNER    TABLE_NAME                COLUMN_NAME                         SEGMENT_NAME
Size_in_mb Tspace
-------- ------------------------- ----------------------------------- ------------
-------------- ---------- ------
FLEVO    FLEVO_AREAS                "SHAPE"."SDO_ELEM_INFO"
SYS_LOB0000027701C00008$$          .50 USERS
FLEVO    FLEVO_AREAS                "SHAPE"."SDO_ORDINATES"
SYS_LOB0000027701C00009$$         1.00 USERS
FLEVO    LSL_EDGE$1                 "GEOMETRY"."SDO_ELEM_INFO"
SYS_LOB0000027708C00008$$          .50 USERS
FLEVO    LSL_EDGE$1                 "GEOMETRY"."SDO_ORDINATES"
SYS_LOB0000027708C00009$$          .50 USERS
FLEVO    LSL_NODE$1                 "GEOMETRY"."SDO_ELEM_INFO"
SYS_LOB0000027714C00009$$          .50 USERS
FLEVO    LSL_NODE$1                 "GEOMETRY"."SDO_ORDINATES"
SYS_LOB0000027714C00010$$          .50 USERS
```

The storage requirements of the tables in segments:

```
SQL> select segment_name, COUNT(*) extents, sum(bytes/(1024*1024)) size_in_mb,
  2       sum(blocks) "Blocks", tablespace_name "Tablespace"
  3    from user_extents
  4    --where (segment_name not LIKE 'SYS_IL000%' and segment_name not LIKE
'SYS_LOB000%'
  5    --   and tablespace_name = 'USERS')
  6    GROUP BY SEGMENT_NAME, tablespace_name
  7    order BY SEGMENT_NAME
  8  ;

SEGMENT_NAME                      EXTENTS SIZE_IN_MB  Blocks Tablespace
------------------------------ ---------- ---------- ------- ----------
FLEVO_AREAS                            30      15.00    1920 USERS
LSL_AREA_TO_FACE$1                      2       1.00     128 USERS
LSL_AREA_TO_FACE$1_PK                   4       2.00     256 USERS
LSL_EDGE$1                             24      12.00    1536 USERS
LSL_EDGE$1_PK                           5       2.50     320 USERS
LSL_EDGE_TO_EDGE$1                     12       6.00     768 USERS
LSL_EDGE_TO_EDGE$1_PK                  13       6.50     832 USERS
LSL_EDGE_TO_LINE_IDX$1                 14       7.00     896 INDX
LSL_EDGE_TO_NODE$1                     11       5.50     704 USERS
LSL_EDGE_TO_NODE$1_PK                  21      10.50    1344 USERS
```

```
LSL_FACE$1                             2         1.00      128 USERS
LSL_FACE$1_PK                          3         1.50      192 USERS
LSL_FACE_TO_AREA_IDX$1                 3         1.50      192 INDX
LSL_FACE_TO_EDGE$1                    12         6.00      768 USERS
LSL_FACE_TO_EDGE$1_PK                 21        10.50     1344 USERS
LSL_FACE_TO_EDGE_IDX$1                13         6.50      832 INDX
LSL_FACE_TO_NODE_IDX$1                 1          .50       64 USERS
LSL_LINE_TO_EDGE$1                    12         6.00      768 USERS
LSL_LINE_TO_EDGE$1_PK                 13         6.50      832 USERS
LSL_NODE$1                             8         4.00      512 USERS
LSL_NODE$1_PK                          3         1.50      192 USERS
LSL_NODE_TO_EDGE_IDX$1                14         7.00      896 INDX
LSL_NODE_TO_POINT_IDX$1                1          .50       64 INDX
LSL_TOPO$1                             2         1.00      128 USERS
LSL_TOPO$1_PK                          3         1.50      192 USERS
LSL_TOPO_PART$1                        2         1.00      128 USERS
LSL_TOPO_PART$1_PK                     3         1.50      192 USERS
LSL_TOPO_TO_PART_IDX$1                 3         1.50      192 INDX
```