Oracle 10g Topology

Testing Oracle 10g Topology using cadastral data

ir. Friso Penninga

GISt Report No. 26 September 2004

Summary

This report describes the results of a first exploration of functionality and performance of the topological data structure as provided by Oracle's newest DBMS release, Oracle 10g Topology (Oracle10g Enterprise Edition Release 10.1.0.2.0 64bit Production version). The overall conclusion should be that this product is the first attempt to deal with topology in a geoDBMS environment and thus has to be welcomed. However, based on this research several remarks about the offered functionality can be made. Most remarks result from certain design decisions that are in themselves understandable, but have some less favourable consequences.

Although performance tests were not very comprehensive, they did indicate that queries with a topological nature (such as neighbour queries) are potentially much faster when performed on the topological data structure. However, there is - due to some 'overhead' in the topology structure - a break even point where the topological and geometrical form are equally fast. The larger the data set, the better the relative topological query performance (time savings up to 50% were attained). Queries with a geometrical nature (such as line intersections and query windows) were on average 2-3 times slower in topological form, but huge deviations occurred regularly, most likely due to some validation problems with the test data sets.

During the tests the author received help from Wilko Quak, Theo Tijssen and Peter van Oosterom. Concept versions were reviewed by Peter van Oosterom, Wilko Quak and Elfriede Fendel. Special thanks goes out to Siva Ravada, Senior Development Manager Spatial Products at Oracle for his fast and accurate responses on questions and problems.

ISBN:
ISSN: 1569-0245

This publication is the result of the research programme Sustainable Urban Areas, carried out by Delft University of Technology.

Contents

1 Introduction

This report describes the research performed on data storage and retrieval in a topological data structure as provided by the new release of the Oracle DBMS. The main advantages of working with a topological data structure are the avoidance of redundant data storage and the ability to maintain data consistency by the application of validation rules. The objective of this research is to test whether the new Oracle DBMS is able to live up to these expectations. The tests were performed on the Oracle10g Enterprise Edition Release 10.1.0.2.0 64bit Production version. The database is tested with cadastral data, as it is already available in the topological winged edge data structure. The aim was to load and query a large cadastral data set; in the test data of the Rotterdam and Zoetermeer cadastral offices were used. Together they hold all cadastral data of the province Zuid-Holland (without history), just over one million parcels. In order to be able to demonstrate the connection between the size of the data set and the DBMS performance two more data sets were used, i.e. the cadastral municipalities of Apeldoorn (one of the largest cadastral municipalities, over 30,000 parcels) and Eck en Wiel (one of the smaller cadastral municipalities, about 600 parcels).

This report is structured as follows: first the topological data model is explained in chapter 2. The used procedure for loading cadastral data is described in chapter 3 and the results of this procedure are subject of chapter 4. The actual testing is discussed in chapter 5 and the report ends with conclusions in chapter 6.

2 Topological data model

2.1 Topology elements

The basic topology elements in an Oracle topology are its nodes, edges and faces. These elements are all two-dimensional. A node is represented by a point and can be used to model an isolated point feature or to bound edges. Every node has a coordinate pair associated with it to describe the spatial location of the node. An edge is bounded by a start and an end node and has a coordinate string associated that describes the spatial representation. Each edge can consist of multiple vertices, represented by linear as well as circular arc strings. As each edge is directed, it is possible to determine which faces are located at the left and right hand side of the edge. A face is represented by a polygon (that can be reconstructed from the several edge strings) and has references to a directed edge on its outer and (if any) inner boundaries. Each topology has a universal face that contains all other nodes, edges and faces in the topology.

Nodes, edges and faces are the building blocks by which every real world object can be constructed. These real world objects are modelled as features or (as Oracle likes to call them) topology geometries. Each topology geometry is stored as a set of topological elements, e.g. a parcel can consist of several faces. Oracle distinguishes five different topology geometry types: point, line strings, (multi)polygons and (combining the four previous topology geometry types) a heterogeneous collection. To lay down the type of topology geometry, each topology geometry has a reference to its topology geometry layer. These layers consist of collections of topology geometries of a specific type, for instance 'parcels' or 'roads'. Oracle automatically assigns unique IDs to topology geometries and topology geometry layers. Although not used during the research, it is possible to define a hierarchy in topology geometry layers. This hierarchy indicates that a topology geometry of the type of the topmost hierarchy level consists of topology geometries at the next level down and so on. On a cadastral data set this could be used to model the relationship between a cadastral municipality, cadastral sections, cadastral sheets and parcel numbers.

In Oracle10 the relationships (generally 1:n) between topology geometries and the nodes, edges and faces are stored as illustrated in figure 2.1. As there is only one topology geometry type ('parcels') in the test, which furthermore has a 1:1 relationship with the face table, one can say that the DBMS creates a lot of overhead given our specific purpose. This overhead might influence DBMS performance significantly.

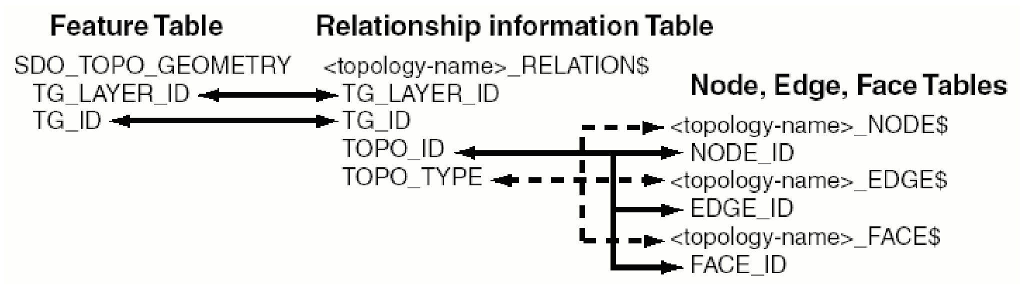

Figure 2.1 **Relationship between topology geometries and nodes, edges and faces [3]**

As mentioned earlier the node, edge and face tables contain references to the associated spatial location of the nodes, edges and faces. The edge table contains several topological relationships as the Oracle topology model uses a winged edge representation. As illustrated in figure 2.2 there are some differences in the way Oracle defines the winged edge model and the way the Dutch cadastre does. As a result the following relationships exist:

Oracle		**Dutch Cadastre**
next_left	=	ll_line
previous_left	=	- fr_line
previous_right	=	- lr_line
next_right	=	fl_line

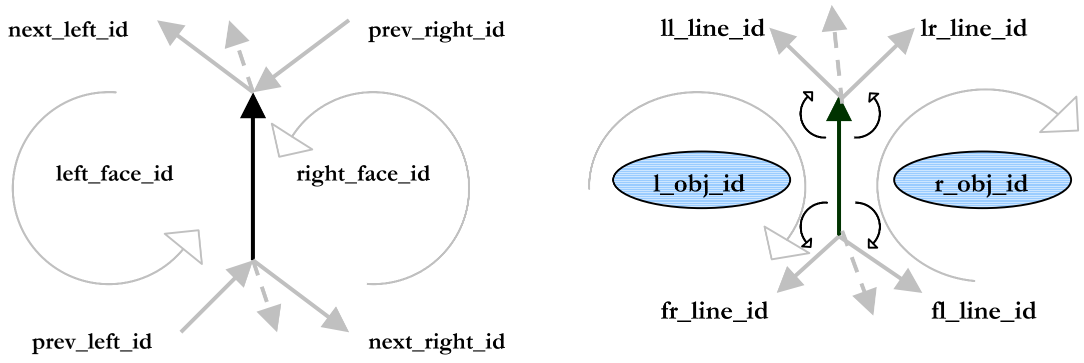

Figure 2.2 Winged-edge representation: Oracle definition (left) and Dutch Cadastre definition (right)

2.2 Topology related tables

In order to be able to work with the collection of tables, the DBMS creates some extra tables. The relationship table is already mentioned as it stores the relationship between topology geometries at the one side and the nodes, edges en faces on the other side. Another automatically generated table is the history table, which holds track of all changes to the topology over time (this approach differs from the cadastre as the cadastre stores historical data in the same tables, see [1] for more details). The metadata tables store the properties of the tables, as well as references to all indexes. The initialize metadata procedure creates most of these indexes on the topology tables.

The procedure to create and fill a topology with data consists of six steps:

1. Creating a new topology

   The sdo_topo.create_topology procedure creates a topology with a user-defined name of the topology. This name combined with _NODE$, _EDGE$, _FACE$ and _HISTORY$ give the names of the created topology tables.

2. Loading data into the node, edge and face table

   This would normally be done in a bulk load, but can also be done using insert-statements.

3. Creating topology geometry tables

   For each topology geometry (feature) type a topology geometry table has to be created.

4. Associate topology geometry tables with topology by defining the topology geometry layers

   By adding topology geometry layers to the topology the relationships between the topology geometry tables and the node, edge or face table are stored in the new <topology-name>_RELATION$ table

5. Initialize metadata and create indexes on topology tables

   The sdo_topo.initialize_metadata procedure creates indexes on the node, edge, face, history and relationship tables.

6. Constructing features from nodes, edges and faces

   Using the topo_geometry constructor one can build the topology geometries from nodes, edges or faces.

The created tables are defined as follows:

```
SQL> describe kadaster_node$
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------------
 NODE_ID                                   NOT NULL NUMBER
 EDGE_ID                                            NUMBER
 FACE_ID                                            NUMBER
 GEOMETRY                                           SDO_GEOMETRY

SQL> describe kadaster_edge$
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------------
 EDGE_ID                                   NOT NULL NUMBER
 START_NODE_ID                                      NUMBER
 END_NODE_ID                                        NUMBER
 NEXT_LEFT_EDGE_ID                                  NUMBER
 PREV_LEFT_EDGE_ID                                  NUMBER
 NEXT_RIGHT_EDGE_ID                                 NUMBER
 PREV_RIGHT_EDGE_ID                                 NUMBER
 LEFT_FACE_ID                                       NUMBER
 RIGHT_FACE_ID                                      NUMBER
 GEOMETRY                                           SDO_GEOMETRY

SQL> describe kadaster_face$
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------------
 FACE_ID                                   NOT NULL NUMBER
 BOUNDARY_EDGE_ID                                   NUMBER
 ISLAND_EDGE_ID_LIST                                SDO_LIST_TYPE
 ISLAND_NODE_ID_LIST                                SDO_LIST_TYPE
 MBR_GEOMETRY                                       SDO_GEOMETRY
```

```
SQL> describe kadaster_history$
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------------
 TOPO_TX_ID                                NOT NULL NUMBER
 TOPO_SEQUENCE                             NOT NULL NUMBER
 TOPOLOGY                                           VARCHAR2(20)
 TOPO_ID                                            NUMBER
 TOPO_TYPE                                          NUMBER
 TOPO_OP                                            VARCHAR2(3)
 PARENT_ID                                          NUMBER

SQL> describe kadaster_relation$
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------------
 TG_LAYER_ID                               NOT NULL NUMBER
 TG_ID                                     NOT NULL NUMBER
 TOPO_ID                                   NOT NULL NUMBER
 TOPO_TYPE                                 NOT NULL NUMBER
 TOPO_ATTRIBUTE                                     VARCHAR2(100)
```

The feature table looks like:

```
SQL> describe parcels
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------------
 OBJECT_ID                                          NUMBER
 FEATURE                                            SDO_TOPO_GEOMETRY
```

where the object type SDO_TOPO_GEOMETRY is defined as:

```
CREATE TYPE sdo_topo_geometry AS OBJECT
   (tg_type           NUMBER,
    tg_id             NUMBER,
    tg_layer_id       NUMBER,
    topology_id       NUMBER);
```

Attribute tg_type defines the type of topology geometry (point, line, polygon/multipolygon or heterogeneous collection). Tg_id is a unique ID of the topology geometry and tg_layer_id the unique ID of the layer the topology geometry belongs to. Topology_id is the unique ID of the topology the layer is in.

3 Loading cadastral data - procedure

3.1 Loading nodes, edges and faces

The complete collection of scripts that are needed to load a cadastral data set into a topological data structure can be found in appendix A. In this chapter the basic ideas behind the procedure will be discussed. The relevant source data is stored in the xfio_parcel, xfio_parcelover and xfio_boundary tables. A view that selects only the current cadastral situation is defined on each table (lki_parcel, lki_parcelover and lki_boundary). These three views are queried to select the relevant data. The edge and face references in these tables have to be altered to ensure correct reference to the universal face F0. After these preparations a new topology is created. The DBMS creates the node, edge and face tables. Filling the edge table is the first step:

```
execute SDO_TOPO.CREATE_TOPOLOGY('KADASTER', 0.001);
insert into KADASTER_EDGE$
    select object_id,
    null,
    null,
    ll_line_id,
    -fr_line_id,
    fl_line_id,
    -lr_line_id,
    l_obj_id,
    r_obj_id,
    geo_polyline as geometry
from lki_boundary;
```

The Oracle topology model requires a start and an end node for each edge. These nodes are not explicitly available in the cadastral topology and have to be derived from the geo_polyline. Just using the first and last point of every polyline is not an option, because it will lead to multiple nodes on the same spatial location, as at least two edges share a node. These duplicates have to be filtered out. First a table called vertices is created. In this table for each potential node a unique ID, an empty new_id, a geometry containing the point, the edge_id they originated from and an indicator if the node is a start or an end node are stored.

```
create table vertices(
        id number,
        newid number,
        geometry mdsys.sdo_geometry,
        fromedge number,
        isstart number);

declare
        rownmbr number;
        edgeid number;
        edge mdsys.sdo_geometry;
        start_node mdsys.sdo_geometry;
        end_node mdsys.sdo_geometry;
begin
        rownmbr := 1;
        for e in
               (select geometry, edge_id into edge, edgeid
               from kadaster_edge$) loop
               start_node := startPoint(e.geometry);
```

```
                end_node := endPoint(e.geometry);
                insert into vertices values (
                        rownmbr,
                        1,
                        start_node,
                        e.edge_id,
                        0);
                rownmbr := rownmbr+1;
                insert into vertices values (
                        rownmbr,
                        1,
                        end_node,
                        e.edge_id,
                        1);
                rownmbr := rownmbr+1;
        end loop;
end;
/
```

As the table vertices is filled with all possible nodes, it has to be determined which possible nodes are identical. The basic idea is to select the node with the highest ID from each subset of possible nodes with the same coordinates (found by the sdo_join operator that uses the spatial index to identify points that interact) and use that node as a node in the topological model. Table verts contains all IDs of the possible nodes (id2) and the selected node_id (id), which will be used instead of the possible nodes.

```
create table verts as
 (select distinct(max(b.id)) id, a.id id2  from
vertices a,
vertices b,
table(sdo_join('VERTICES','GEOMETRY','VERTICES','GEOMETRY',
               'mask=ANYINTERACT')) c
    where c.rowid1 = a.rowid and c.rowid2 = b.rowid
    group by a.id)
;
```

This table can be used as a translation table, as it contains the information which node to add to the topology for every vertex. The ID of the node that goes into the topology will be stored as newid in the table vertices:

```
update vertices v
set newid = (
        select vt.id
        from verts vt
        where (v.id = vt.id2))
;
```

As table vertices still contains all possible nodes, only the selected nodes (where ID = newid) have to be inserted into the node table of the topology:

```
insert into KADASTER_NODE$
        select  v.id  as  NODE_ID,  null  as  EDGE_ID,  null  as  FACE_ID,
v.GEOMETRY
from vertices v where (v.id=v.newid) ;
```

The complete table vertices can now be used to update the edge table, as each edge needs a start and an end node. Therefore the start node_id is selected as the newid from the vertices table where the edge_id of the topology matches the source edge ID of the vertex:

```
update KADASTER_EDGE$ e
set start_node_id = (
        select newid
```

```
            from vertices v
            where (e.edge_id=v.fromedge)and(v.isstart='0')),
            end_node_id = (
            select newid
            from vertices v
            where (e.edge_id=v.fromedge)and(v.isstart='1'));
```

The processing described above is time consuming and only necessary since Oracle requires start and end node fields in the edge table. Oracle states in the documentation that the start and end node are necessary to determine the edge's direction. However, it is also possible to determine the direction based on the coordinates in the geometry field and therefore it is not strictly necessary to demand reference to these nodes in the topology model. This choice also causes some redundant data storage, as the coordinates of the nodes are stored in the node table as well as in the edge table (as part of the coordinates of the polyline).

As a last step in loading the topological data the faces are filled with data from lki_parcel and lki_parcelover. The geometry itself is not stored in this table; only the bounding box is stored explicitly in order to be able to create the R-tree faster as it is not necessary to calculate the bounding boxes anymore. However, this incorporates the risk of conflicts between the actual geometry and an erroneously inserted bounding box. As the bounding box is present in the cadastral database this risk is eliminated in the performed tests.

```
insert into KADASTER_FACE$ f
    select
object_id as face_id,
-line_id1 as boundary_edge_id,
null as island_edge_id_list,
null as island_node_id_list,
geo_bbox as mbr_geometry
    from lki_parcel;
```

3.2 Defining topology geometries (features)

As the topology tables are filled, one can start defining the topology geometries. First a topology geometry layer named 'parcels' and of type polygon is created. Other layers like for instance 'boundaries' might be defined in the same way.

```
declare
begin
   sdo_topo.add_topo_geometry_layer('KADASTER','PARCELS','FEATURE',
   'POLYGON');
end;
/
```

The topology geometry table parcels can now be filled by iterating over all rows in the kadaster_face$ table:

```
declare
   fid number;
   tli number;
begin
   select tg_layer_id into tli from user_sdo_topo_info where topology =
   'KADASTER' and table_name = 'PARCELS';
for a in
        (select face_id into fid from kadaster_face$ where face_id <> -1)
       loop

        insert into parcels values
               (a.face_id,
                  sdo_topo_geometry (
                         'KADASTER',           -- topology name
                         3,                    -- topo-geomtry type polygon
                         tli,                  -- TG_LAYER_ID
                  sdo_topo_object_array ( sdo_topo_object (a.face_id,3) )));
end loop;
end;
/
```

It is possible to add extra attributes to the parcels table using standard SQL-functionality to add and fill columns. Inserting topology geometries is quite laborious, as one has to work with the topology geometry layer ID. As a layer is created, the DBMS assigns a unique ID to this layer. This ID is stored in the user_sdo_topo_info table. The insert statement requires this unique ID, so it is necessary to query the info table using the arguments topology_name and table_name. Both arguments can also be found in the insert-statement, but for some unknown reason the DBMS is not able to find this ID on its own. Since the layer ID is used hard-coded in the topology example in the Oracle documentation, it took a while to find out how to retrieve this layer ID. In retrospect it would have been easier if the other topology geometry constructor was used, as this constructor requires the topology name, table name, column name and tg_type as arguments and thus not the layer ID. The complete procedure to define the parcels is very time consuming (over 40 hours for one million parcels), which is even worse if one realizes that in the cadastral case a 1:1 relationship exists between a topology geometry and a face.

One might say that the offered functionality does not match very well with our quite straight forward data needs. The mandatory use of nodes causes a lot of work, but does not contribute anything to the cadastral case, as a lack of clear linkage between a node and a cadastral object. In a cadastral data set boundaries and parcels are the relevant objects, not the vertices in the boundaries.

4 Loading cadastral data – results

The loading process was first tested with a very small cadastral municipality called Eck en Wiel (ECK00). This municipality consists of 586 parcels and it takes less than five minutes to build the topology, create the parcel features and build a table with the polygons of the parcel boundaries (needed for comparison between queries performed on the topological data set and on a geometrical version). The topology consists of 587 faces, 2,046 edges, 1,467 nodes and is illustrated in figure 4.1.

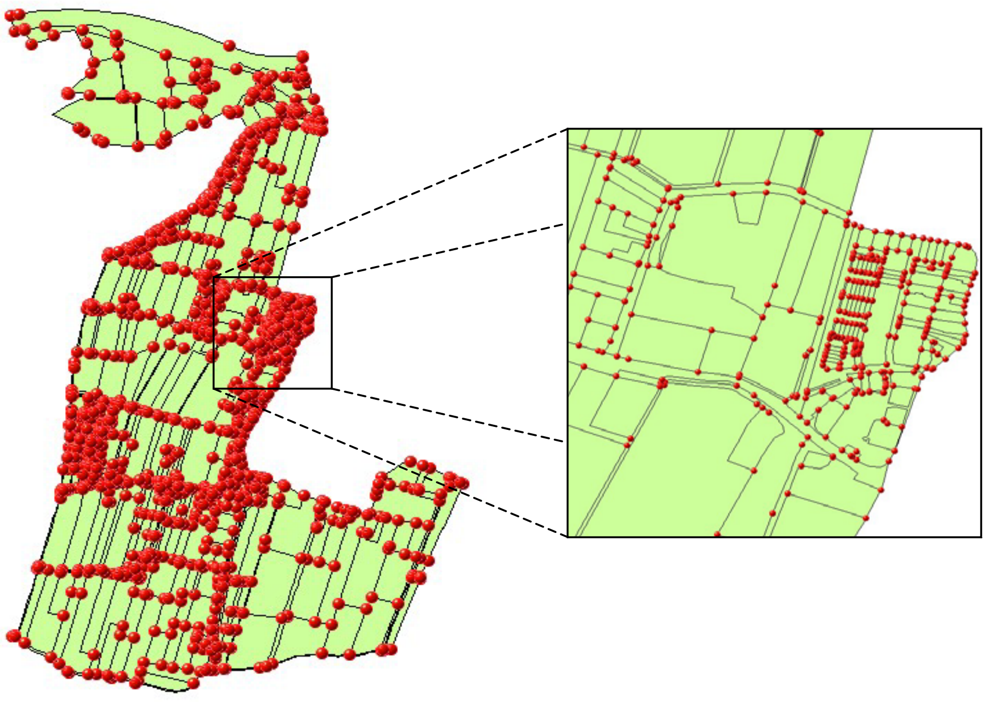

Figure 4.1 Small cadastral data test area: cadastral municipality of Eck en Wiel

As loading small data sets went smoothly, a large cadastral municipality was used as a next test data set. Apeldoorn (cadastral municipality APD01) consists of 30,174 parcels and was loaded in about two hours. The Apeldoorn topology consists of 30,175 faces, 95,792 edges and 66,123 nodes. As illustrated in figure 4.2 on the next page the cadastral municipality APD01 consists of two non-touching parts, separated by municipality SRN00 (Hoog Soeren). One of the parts has a hole; this is the cadastral municipality APD04.

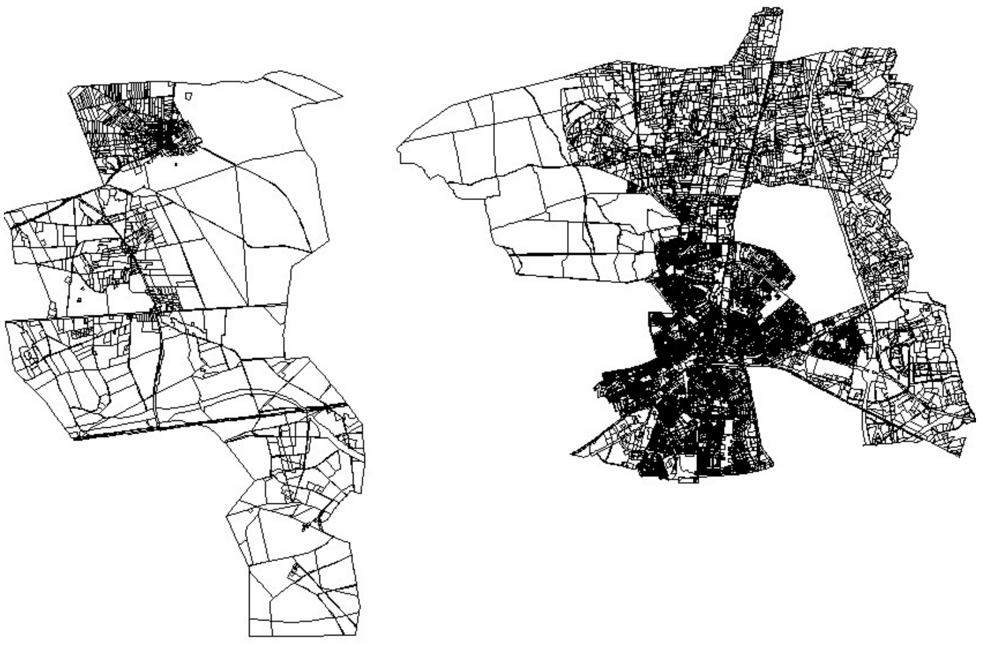

Figure 4.2 Test data set Apeldoorn (APD01)

Although the loading process did not result in any error messages, a visual check (figure 4.3) of the results of the get_geometry() function shows that some errors occurred in parcel 140154608 (APD01 Z 03448 G0000). The validation was not performed yet.

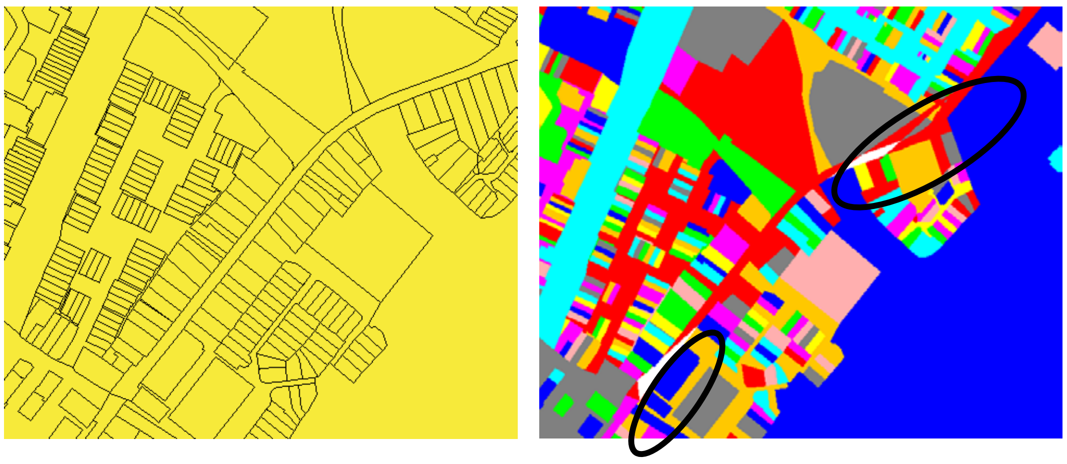

Figure 4.3 Gaps and spikes in erroneous road parcel

At the left side the correct cadastral situation, at the right hand side the features in the topology. One can see that something went wrong with the parcel containing the street from the bottom left to the top right. Two holes appear and some strange peaks occur. After examining this case thoroughly it turned out that the actual geometry was not the same shape as illustrated before, but was in fact an invalid geometry. The visualization tool created the two peaks and holes, but in reality the geometry was of type multipolygon, but none of the polygons were properly closed. All this was caused by two erroneous left–right pointers in the edge table. By mistake two boundaries (marked in red circles, IDs 142755121 and 142754803) point not to the street parcel, but to the parcel marked by the arrow (see figure 4.4). In the most recent cadastral data set these errors are corrected, but not in the test data set.

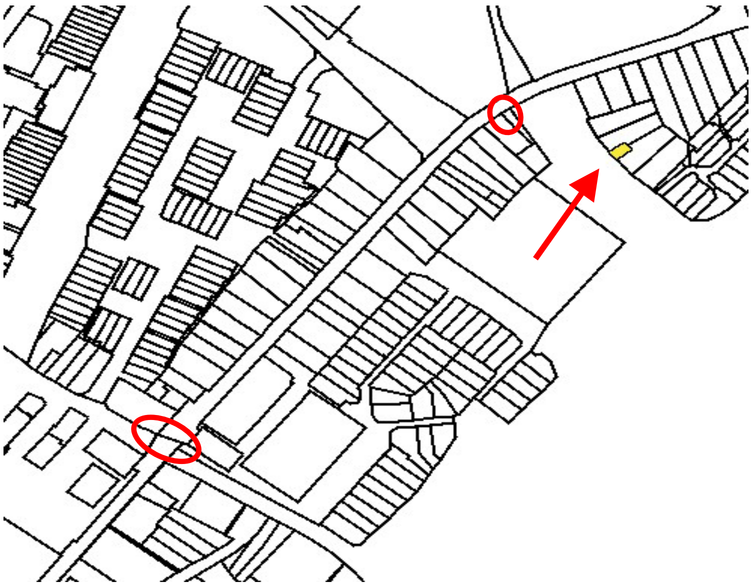

Figure 4.4 Two left-right pointers of edges point to the wrong parcel

To be able to illustrate the encountered problem in an error report to Oracle, a small topology was built as illustrated in figure 4.5. Based on this topology four topology geometries were defined, each consisting of one of the faces. The left-right face pointers of edges e2 and e11 were changed to P2-P3 instead of P4-P1. As a result the get_geometry() function returns some invalid geometries (even two multipolygons). Contacting Oracle led to the announcement to fix this bug in the next release: "The fix is in the next release and the validate on this data would now come back with this message: 'Consecutive boundary edges -3 and -2 of face 1 do not meet at common node'." (E-mail Siva Ravada, March 1st, 2004)

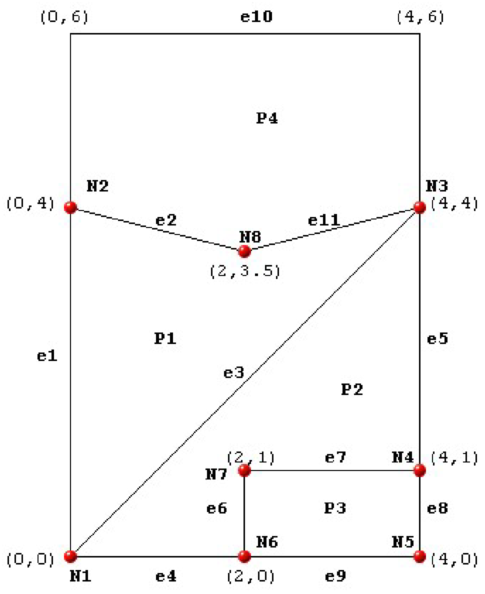

Figure 4.5 Small topology to illustrate invalid geometries due to erroneous left-right references

```
SQL> select f.feature.get_geometry()
  2  from prcls f
  3  ;

F.FEATURE.GET_GEOMETRY()(SDO_GTYPE,    SDO_SRID,    SDO_POINT(X,    Y,    Z),
SDO_ELEM_INFO,
-----------------------------------------------------------------------------
SDO_GEOMETRY(2003,    NULL,    NULL,    SDO_ELEM_INFO_ARRAY(1,    1003,    1),
SDO_ORDINATE_ARRAY(4, 4, 0, 0, 0, 4))

SDO_GEOMETRY(2007, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1, 13, 1003,
1), SDO_ORDINATE_ARRAY(4, 1, 4, 4, 0, 0, 2, 0, 2, 1, 4, 1, 4, 4, 2, 3.5, 0,
4))

SDO_GEOMETRY(2007, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1, 7, 1003, 1),
SDO_ORDINATE_ARRAY(4, 4, 2, 3.5, 0, 4, 4, 1, 2, 1, 2, 0, 4, 0, 4, 1))

SDO_GEOMETRY(2003,    NULL,    NULL,    SDO_ELEM_INFO_ARRAY(1,    1003,    1),
SDO_ORDINATE_ARRAY(4, 4, 4, 6, 0, 6, 0, 4))

4 rows selected.
```

Then the validation process was tried for this erroneous topology and it seemed to validate the topology, but later it was discovered that there was a mistake in the way we carried out the validation process. In order to be able to validate the topology, a topomap object has to be created first. A topomap object is associated with an in-memory cache that is associated with a topology. After creation the topology has to be loaded into the topomap, before it can be validated. As this is not explicitly mentioned in the chapter of [3] with the topo_map-procedures, this loading was omitted at first. As the validate procedure did not return a message like 'topomap is empty' or something like that, this error was not directly discovered. Note that the entire topology has to be loaded into the topo_map object, which requires a lot of memory in case of large data sets. The complete validation process is given below.

```
variable res1 varchar2(4);
variable res2 varchar2(4);
call sdo_topo_map.create_topo_map('<TOPOLOGY_NAME>','TEMP');
call sdo_topo_map.load_topo_map('TEMP','TRUE') into :res1;
print res1;
call sdo_topo_map.validate_topo_map('TEMP') into :res2;
print res2;
```

Variable res2 will be true if the topology is marked valid. However, as opposed to what one might expect, if the topology is not valid, res2 will not be false nor a complete list of found topological problems will be returned. In fact the validate_topo_map procedure will crash and produce an error message that points to the first encountered topological problem:

```
call sdo_topo_map.validate_topo_map('TEMP') into :res
     *
ERROR at line 1:
ORA-29532: Java call terminated by uncaught Java exception:
oracle.spatial.topo.TopoValidationException: Consecutive boundary edges -3
and -2 of face 1 don't meet at common node
```

A clear drawback of this approach is that in order to fix ten topological errors, one has to run the validate process eleven times. On the other hand this increases the DBMS's robustness, but still a list with errors would come in handy, even more if one considers the possible number of topological errors in a data set containing over one million parcels. This number of errors, combined with the runtime of a single validation process on very large data sets points out the necessity of a validation process, which returns all possible errors at once. As a result of this validation problem a non-validated data set is used during the rest of the testing, which later proved to be a bit dangerous.

In order to be able to test the DBMS query performance a large data set was loaded into the topological data structure. As a test data set the cadastral data of the cadastral offices Den Haag and Zoetermeer were used. Together they hold all cadastral data of the province Zuid-Holland, which consists of 1,001,133 parcels, 3,363,761 boundaries and 2,401,586 nodes. The entire loading process requires about five days. As we like to compare query performances between topological and geometrical data structures the following statement was used to acquire a polygon data set of Zuid-Holland:

```
create table PG as
       select
              p.object_id,
              p.feature.get_geometry() as GEOM
       from PARCELS p;
call createMetaData('PG','GEOM',0.1);
create index idx on
       PG(GEOM)
       indextype is mdsys.spatial_index;
```

During this process an error occurred:

```
      p.feature.get_geometry()
      *
ERROR at line 4:
ORA-06531: Reference to uninitialized collection
ORA-06512: at "MDSYS.SDO_UTIL", line 2327
ORA-06512: at "MDSYS.SDO_TOPO_GEOMETRY", line 524
ORA-06512: at line 1
```

The cause of this error was unclear. Although some kind of data error was a possibility, we suspected the process ran out of memory, as memory usage during the operation was very high and almost as high as the available amount of memory. In order to be able to find the cause the input was split in ten and later in hundred separate insert-statements, selecting on the last digits of the object_id:

```
insert into PG
       select
          p.object_id,
          p.feature.get_geometry()
       from PARCELS p where (p.object_id like '%01' );
```

This time some of the insert-statements (with approximately 10.000 rows at a time) went well, but some others produced the same error as mentioned above. More precisely, the first 29 statements were successfully executed (last digits 00-28), the next set (29-63) returned the error messages, 64-66 were successful again, 67 failed and 68-99 were executed properly. The way in which the error messages appeared showed some similarities, every time in the set 29-63 the error was returned after 7 minutes and 10 or 20 seconds. At this moment we still expected some kind of memory leak which appeared after a certain amount of inserted rows. Restarting the process after allowing the DBMS to use more memory did not solve the problem and it showed that Oracle always uses about the maximal allowed memory capacity, so at this time memory problems did not seem to be the most likely cause. In an attempt to detect one of the exact parcels causing the problem, the first data set (last digits 29) was divided in ten parts, the process was restarted and repeated after the next error. This finally led to a parcel ID of an erroneous parcel. Looking up the cadastral situation showed that the erroneous parcel was one of the 34 very small, round parcels as they appear in the purple parcel (ID 340644065, SKN00 D08605 G0000), see figure 4.6.

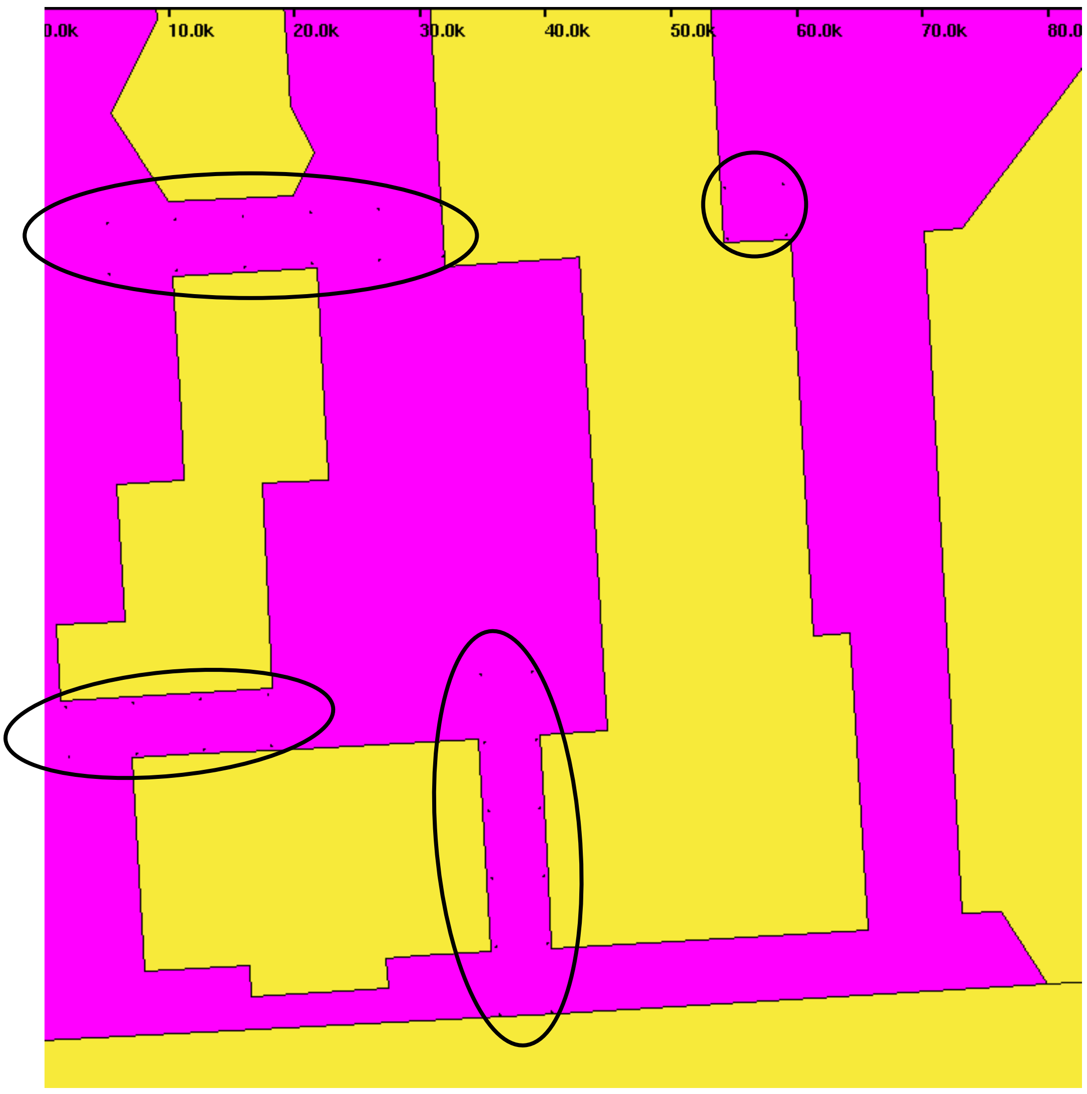

Figure 4.6 Thirty-four small parcels with missing left-right references

These parcels are really small (7-10 centimetres in diameter) and their right_municip_IDs were empty. As an empty left or right municipality reference normally indicates that a boundary is part of the office boundary, these empty references are replaced with references to the universal face F0. In this case this replacement is obviously erroneous, but situations exist where this can be correct, for instance in Baarle-Nassau where a Belgian enclave is situated in the Netherlands. It looks like the DBMS cannot handle situations like these.

It turned out that the parcels in this case have consecutive object_IDs, causing errors in the set with parcel IDs ending with 29 until 63. The large parcel (marked in purple) is owned by the municipality of Spijkenisse. The four island parcels are owned by an investment company and they contain a shopping mall. The small circle-shaped parcels are also owned by Spijkenisse, but also contain a building right of the investment company. It is very likely that the small parcels contain the standards of several coverings to create a dry passage to all customers between the different parts of the shopping mall. Two similar errors were found on another location (id 340633554, SKN00 D08530 G0000) (see figure 4.7) in the same municipality, probably again because of the founding of a covering. This time the small parcels are located just outside of a home for the elderly. In this case only two parcels have boundaries with incomplete references to their left and right faces.

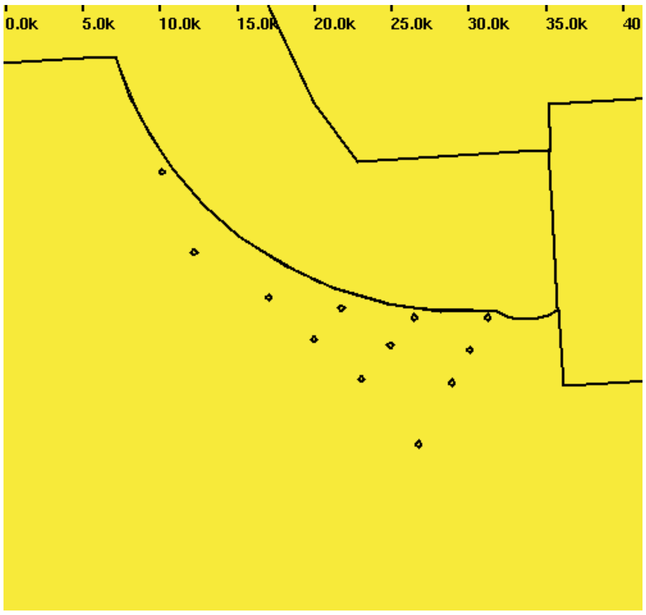

Figure 4.7 Very small parcels with missing references

One might reason that if missing left or right hand side pointers caused the earlier problems, the absence of further problems indicates that all other pointers are correct. However, this is not very likely to be true as it is not expected that all other references are correct. Probably only references to the universal face cause problems like described above.

Earlier experiences with cadastral data led to the assumption that the errors with the left and right pointers as described above perhaps could be avoided by the usage of the combination of the left-right municipality, cadastral section, sheet and parcel number as it is expected that these alternative topology references are better filled by the Dutch Cadastre than the left-right IDs. As this might be true for some pointers this would not solve the problems in Spijkenisse.

5 Testing with cadastral data

As mentioned in the introduction one of the major advantages of using a topological data structure is the avoidance of redundant data storage. The risk of redundant data storage is the possibility of data inconsistencies. Besides preventing data inconsistencies the avoidance of redundancy might also reduce storage capacity requirements, as double storage requires double storage capacity. However, the question is whether the saved storage capacity is larger than the additional storage capacity required for storing the topological relationships. This is also dependant of the type of data, e.g. the number of points per edge.

The chapter on the topological data structure already showed that the current DBMS topology implementation does not completely avoid redundant data storage, as point geometry is stored both in node and in edge tables. The tables 5.1 and 5.2 illustrate that due to storing extended topological relationships the topological data structure requires about five times more storage capacity compared to storing the parcels as polygons. One can also see that the indexes require about the same storage capacity as the data itself. There are eight different indexes on the node, edge and face tables: on the node table an index on the associated face ID and a spatial index on the node geometry, on the edge table indexes on the start node, end node, left and right face ID and a spatial index on the geometry and on the face table a spatial index on the bounding box of the face. Note that on both data structures only the geometry is stored, administrative data is not taken into account.

Topological data structure	Eck en Wiel tables	Eck en Wiel indexes	♯	Apel-doorn tables	Apel-doorn indexes	♯	Zuid-Holland Tables	Zuid-Holland indexes	♯
<topology>_node$	0.08	0.16	2	2.78	5.86	2	100.61	276.14	2
<topology>_edge$	0.48	0.40	5	20.19	17.33	5	732.20	715.83	5
<topology>_face$	0.12	0.08	1	5.57	2.65	1	192.56	109.94	1
<topology>_relation$	0.02	0.04	2	0.74	2.04	2	24.74	66.61	2
<topology>_history$	0.00	0.00	1	0.00	0.00	1	0.00	0.00	1
feature_table	0.02	0.02	1	0.94	0.69	1	30.23	22.61	1
Sum	*0.72*	*0.70*		*30.22*	*28.57*		*1080.34*	*1191.13*	
Total topological		**1.42**			**58.79**			**2271.47**	

Table 5.1 Data storage (in Mb) (size of table, size of indexes (sum) and number of indexes) of the topological data structure

Geometrical data structure	Eck en Wiel table	Eck en Wiel index	♯	Apel-doorn table	Apel-doorn index	♯	Zuid-Holland Table	Zuid-Holland index	♯
Polygons	0.21	0.08	1	9.52	2.65	1	310.97	109.90	1
Total geometrical		**0.29**			**12.17**			**420.87**	

Table 5.2 Data storage (in Mb) (size of table, size of index and number of indexes) of the geometrical data structure

The main objective of the query testing is to determine the relative performance of querying on a topological data structure. All queries are executed both topologically and geometrically. One can expect that queries with a geometrical nature, such as 'which parcels are in this query window' or 'which parcels intersect with this line' are faster in the geometrical form. At the same time one can expect queries based on relationships, such as 'return all neighbours of this parcel' to be faster when performed on a topological data structure. Below two queries, one query window and one neighbour query, both described in topological and in geometrical form.

Query window in geometrical and topological form:

```
select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
SDO_ORDINATE_ARRAY(75000000,425000000,76000000,426000000)))='TRUE';

select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
SDO_ORDINATE_ARRAY(75000000,425000000,76000000,426000000)))='TRUE';
```

Neighbour query in geometrical and topological form:

```
select p.object_id from pg p, pg g where g.object_id = 340633554 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select q.object_id from parcels q, parcels r where r.object_id = 340633554
and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;
```

Note that the possibility exists to query directly on the edge and faces tables by using the left-right references instead of using the sdo_anyinteract statement. The small cadastral data set of Eck en Wiel was the first one to become available for testing, although it was not validated. In figure 5.1 the results of both the topological and geometrical version of 15 queries are illustrated. The first four queries are query windows, the next six are line queries and the last five queries are neighbour queries. One can see that only the neighbour queries are sometimes as fast in the topological version as in the geometrical form.

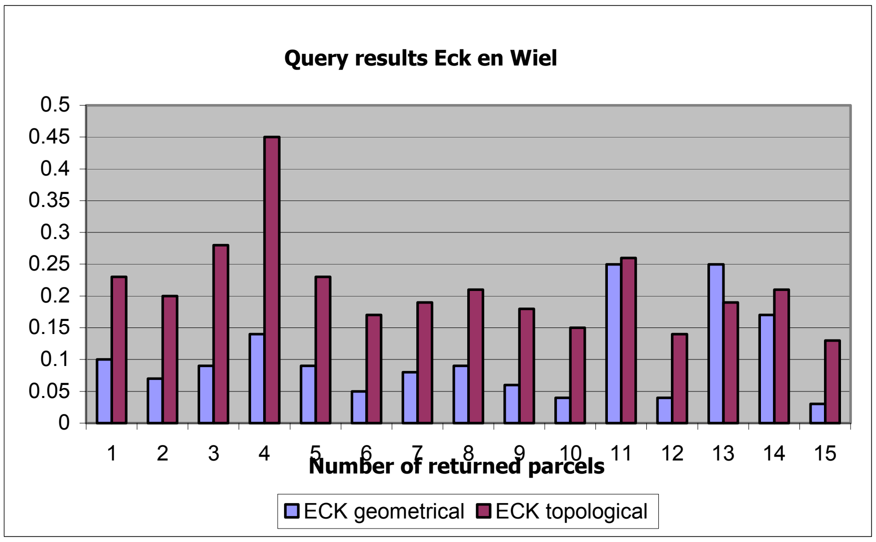

Figure 5.1 **Query performance in geometrical form and in topological form on the small data set**

The length of the query time is more or less related to the size of the query itself, i.e. to the number of returned parcels, as can be seen in figure 5.2. The topological form seems to have a constant offset in comparison to the geometrical form.

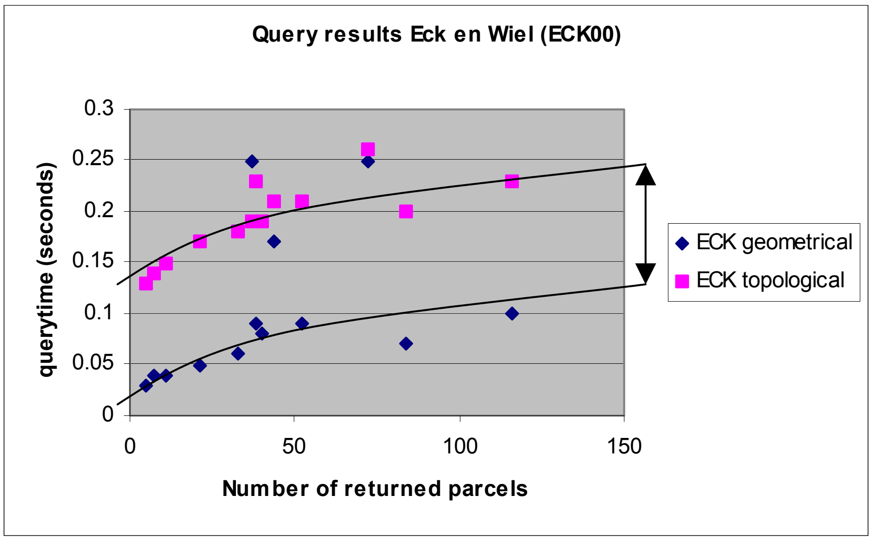

Figure 5.2 Relation between query performance and number of returned parcels

The next test data set is the larger municipality of Apeldoorn (again not validated). It turns out that this time the topological form is a bit quicker for neighbour questions, as again the last five queries are neighbour queries (figure 5.3). A relationship between the number of returned parcels and the query time is less obvious, most likely due to the less homogeneous character of the Apeldoorn parcel data set (figure 5.4). Combining the results of both Eck en Wiel and Apeldoorn makes this even clearer (figure 5.5).

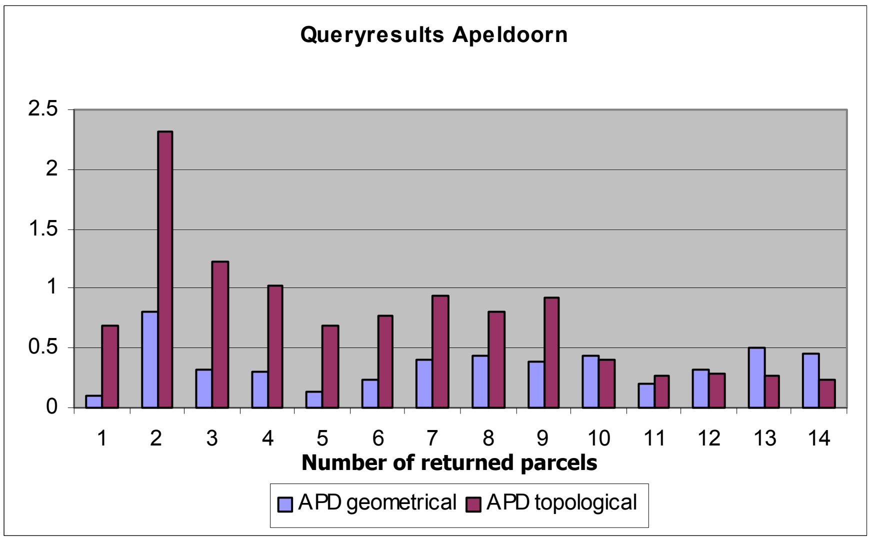

Figure 5.3 Query performance in geometrical form and in topological form on Apeldoorn test set

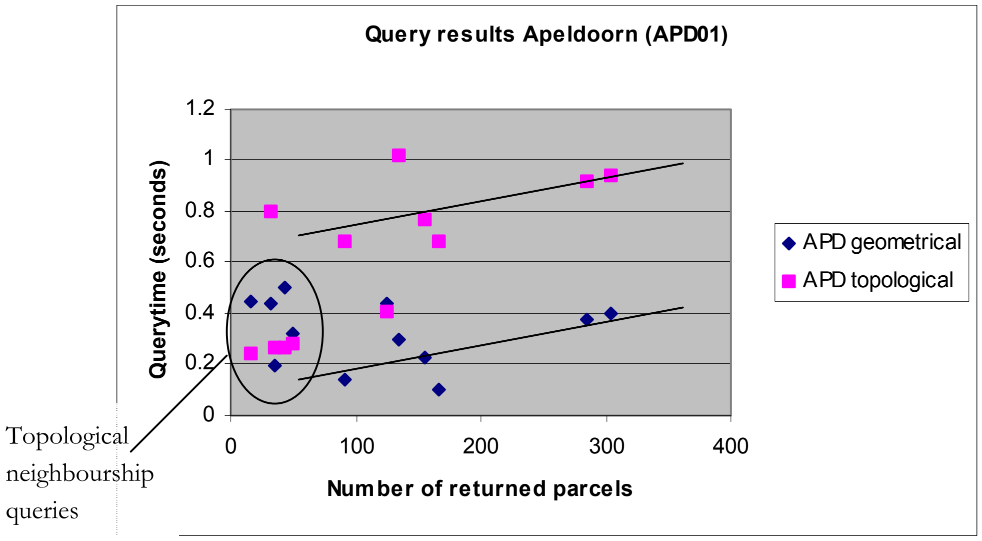

Figure 5.4 Relationship between query time and number of returned parcels

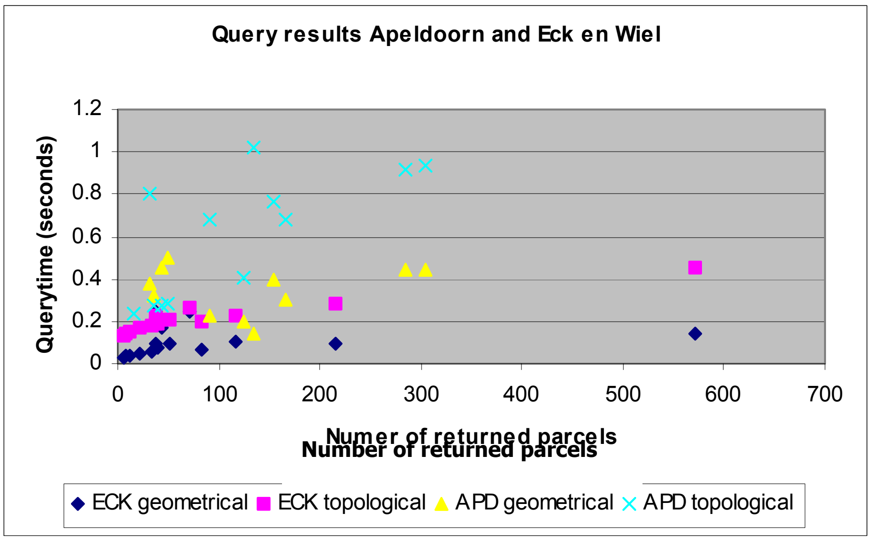

Figure 5.5 Relationship between query time and number of returned parcels for all data sets combined

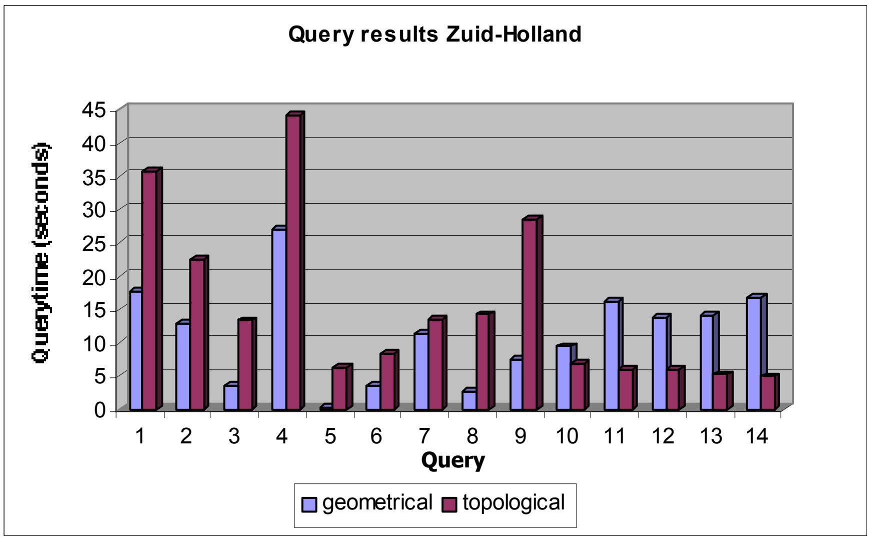

Figure 5.6 Query performance on the large data set

As the results of the neighbour queries on Apeldoorn indicated an increase of relative performance of the topological form with the size of the data set, high hopes were set for the results on the large Zuid-Holland data set. Figure 5.6 on the previous page shows that the neighbour queries (10-14) are indeed over two times faster in topological form than in geometrical form. As the topological form on the smallest data set (586 parcels) was about two times slower than the geometrical form, it was 14% faster on the larger data set (30,174 parcels) and on average even 53% faster on the very large (one million parcels) data set. This relative increase in performance is illustrated in figure 5.7, where the neighbour queries on the three data sets are compared. Although the absolute query times are different for each data set, these three pictures clearly show the relative increase of the topological query performance.

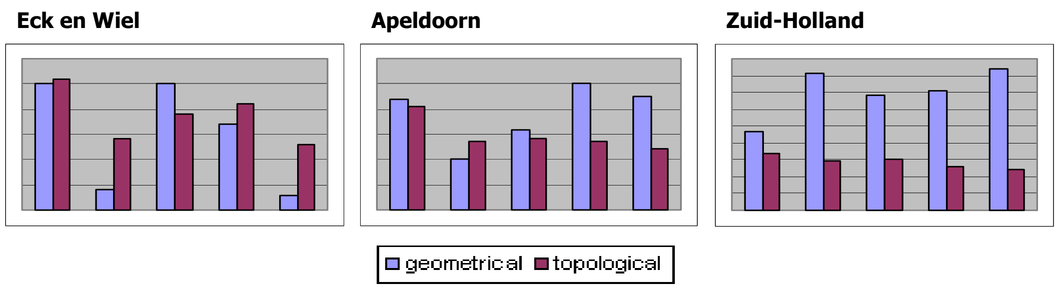

Figure 5.7 Relative increase of topological query performance

Using the topological data storage the query window and line queries turned out to be slower than on the geometrical data storage, just as one would predict. On average the topological form is about two or three times slower, but some extreme peaks occur as sometimes the topological form can be up to 30 times slower or two times faster. A plausible explanation for the occurrence of these peaks could not be found, but the assumption is that, generally speaking, these peaks will occur more often in case of small query results, i.e. just a few returned parcels, although this is difficult to prove due to the low absolute query times and differences. In order to give an idea about query performance, query size and the difference between the geometrical and topological results the results of a single run are in table 5.3.

Query name	L1	L2	L3	L4	QW1	QW2	QW3	QW4
Geometrical (seconds)	17.6	12.48	3.48	27.21	0.27	3.39	10.64	72.37
Topological (seconds)	33.8	21.7	13.02	41.29	5.97	7.85	12.96	37.93
# parcels queried	1378	1291	268	1772	40	720	2714	20629

Query name	QW5	QW6	C1	C2	N1	N2	N3	N4	N5
Geometrical (s)	94.42	271.74	2.26	7.01	9.64	13.04	9.79	11.66	12.56
Topological (s)	168.8	1264.86	14.05	28.16	5.62	6.80	5.42	4.76	4.71
# parcels queried	112011	524254	3615	2124	161	501	191	5	40

Table 5.3 Query times and number of returned parcels. The query names correspond to the names in the figure on the next page

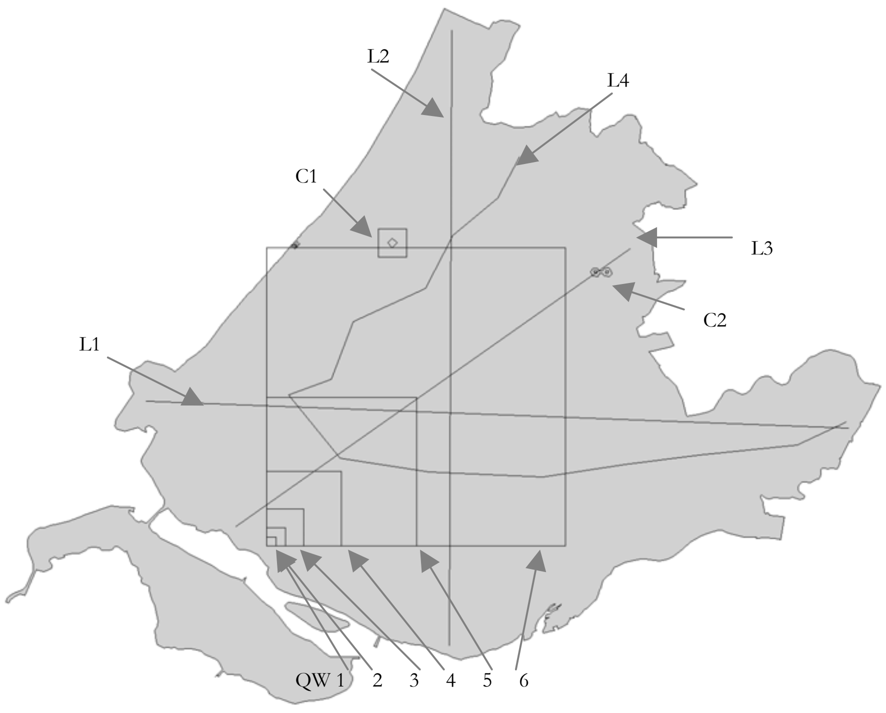

Figure 5.8 Queries on the Zuid-Holland data set

These queries are illustrated in figure 5.8. L1 until L4 are line queries, with L4 consisting of 13 line segments whereas the other lines consist of one segment only. The query windows QW 1-6 dimensions are 1x1, 2x2, 4x4, 8x8, 16x16 an 32x32 kilometres. The complex shape C1 consists of a polygon of 3x3 kilometres with a rhombic hole of 0.5 square kilometres. Second complex shape C2 has more fanciful boundaries, but can be described best as something similar to a pair of glasses.

To this point the DBMS performed as predicted, but for some unknown reasons the data set of Zuid-Holland became corrupt. There might be a relationship with the fact that the data set was not valid as validating was impossible due to the DBMS's approach with returning one error at a time. The problem was discovered by comparing the results of a large query test with the results of the same test repeated two days later. Although the query times looked about the same in the second test, it turned out that some topological versions of queries did not return the correct number of parcels anymore. At this time it was unclear what was causing these erroneous query results. In the time between the two query tests nothing was changed in the data set, only a validation process was carried out, but as this process only reads the data, it is not likely to cause such a problem. The different topology tables were all looking good as they still contain all data and all indexes were present and valid. Rebuilding indexes just to make sure that they were available did not make any difference. At this moment the question arises whether the problem originated from the data or from the DBMS itself. Therefore the Apeldoorn data set was queried again and it turned out that these query results were also erroneous. Given the fact that now two independent data sets were causing troubles, all trace and log files were examined closely in order to find out whether there were any indications that something went wrong with the DBMS itself. Since not a single clue could be found the focus shifted again towards the data itself. As Apeldoorn is - compared to the Zuid-Holland data set - a small data set it was possible to rebuild the entire topology (all tables and indexes) in one day, although validating was omitted as it is nearly impossible to validate a data set within the DBMS validation approach. Executing the same queries again now showed that the topological queries were returning the correct answers, but performance seemed to have dropped. As extensive testing was done already on the Zuid-Holland data set it was decided to rebuild the entire Zuid-Holland topology tables and indexes in order to be able to judge whether performance problems occur or not. Rebuilding the topology did solve the problem of erroneous query results, but the topological query performance also dropped dramatically. Generally speaking performance dropped by a factor four. In these tests even the topological form of the neighbour queries was about 70% slower compared to the geometrical form, whereas these queries were over 50% faster in the first test round. Again no clues to the cause of this performance problem could be discovered in the trace and log files.

In order to test whether it was possible to reproduce the same query errors it was reconstructed which actions were performed at the time of the first occurrence of the problem. Re-executing all this different tests and validation processes did not lead to another corrupt data set, which makes it even more obscure what caused the query errors and the performance drop. Although it was very unclear what caused the earlier problems, it was important to make sure whether these problems were caused by some topological errors in the data or not. As mentioned earlier validating using the topo_map objects was quite slow. Therefore the validation process was carried out using the Java api and executed on a faster machine than the Casagrande server. Zuid-Holland could not be validated as a lot of problems arise from the shared boundary between the cadastral offices of Den Haag and Zoetermeer (which are two independent data sets). These identical boundaries occur in both data sets, but they have different object IDs which causes the following validation errors (as the two identical boundaries intersect each other):

```
went wrong:oracle.spatial.topo.TopoValidationException: Edge 310439292
coordinate string has an intersection with another edge
oracle.spatial.topo.TopoValidationException: Edge 310439292 coordinate
string has an intersection with another edge
        at oracle.spatial.topo.TopoMap.validateCache(TopoMap.Java:6680)
        at ValidateTopology.main(ValidateTopology.Java:64)
```

Fixing all these shared boundaries was considered to be too laborious, so it was decided to create a new topology with only the Zoetermeer office. This subset of the Zuid-Holland data set consists of 577,540 parcels (577,541 faces, 1,962,170 edges and 1,407,361 nodes). While query performance on the Zuid-Holland data set was still dramatically, the Zoetermeer data showed good performance of the topological neighbour queries. These queries were performed almost 60% faster than the geometrical version of the neighbour queries. However again it was too laborious to create an entirely error free topological data set. At first the errors indicated some other topological errors in Spijkenisse, again due to missing or incorrect references of the minuscule circle shaped parcels. After fixing these errors, problems arise with arc-shaped boundaries. Erroneously the DBMS documentation states that the DBMS is capable of handling circular arcs as 'an edge may have several vertices making up a line string, circular arc string, or combination' (page 1-3, 1-4 Oracle Topology documentation [3]) it is not capable of handling these boundaries correct as it miscalculates intersections of these arcs. Oracle states that this is a documentation error. However, as for instance our test data sets contain multiple circular arcs, it would be a desirable extension in future DBMS releases when circular arcs would be handled correctly.

After all some important questions remain unanswered. It has not become clear what caused the sudden incorrect answers of the topological queries on the Zuid-Holland data set, why performance decreased dramatically in the rebuild Zuid-Holland data set and why the Zoetermeer data set at the same time was returning promising query times (but without validation, all kinds of side effects of non-robust processing may cause these effects).

6 Conclusions

From this first exploration of Oracle 10g Topology 's functionality and performance several meaningful conclusions can be drawn. As the DBMS is only tested with invalid data sets the conclusions are grouped in two classes, namely the findings unrelated to the invalid state and the findings that might be/are infected by the invalid state of the data set.

6.1 Conclusions independent of validation issues

6.1.1 Functionality

- The current implementation does not completely avoid redundant data storage, as geometry is stored both in node and edge tables. However, as long as the user uses the supplied api for data editing instead of directly updating the node, edge and face tables, data consistency is maintained. This can be considered a suitable solution for the average user, but the expert user probably wants to edit directly on the node, edge and face tables as it is much quicker than using all kinds of extra functionality.

- The insertion statement for the face table requires a bounding box. This bounding box is used to create R-trees faster. This incorporates the risk of input errors with non-fitting bounding boxes. Although it would slow down the topology construction process, it would be better to derive these bounding boxes in the DBMS itself to ensure data integrity.

- As the validation process returns one error at a time, it is not suitable for making a data set valid, but only for keeping it valid during edit operations. This Oracle design decision is based on the fact that usually a topological error causes other errors, thus resulting in a list of cascading orders. This is an understandable argument, but from a user perspective a list of errors might be desirable as one can often recognize the 'root' errors in the list, thus enabling correction of multiple topological errors in one validation round.

- It is not possible to expand the set of topological rules. If one wants for instance to enforce a planar partition of features, a non-overlap rule is needed.

- It is a disadvantage that to validate the entire topology has to be loaded into the topo_map object, as this requires a lot of memory in case of large data sets.

- Oracle Topology is not able to deal with holes in the topology as references to the universal face. The holes has to be modelled as distinct faces. The creation of this additional faces can be difficult, as for instance the cadastral data set does not contain this face for the Belgium exclave within the Netherlands.

- Storing data in a topological data structure requires about 4-5 times more storage capacity, partially caused by the number and size of required indexes. It is important to recognise that this data set was a worst case scenario, as its edges do not consist of a lot of points, thus reducing the effect of avoiding redundant data storage.

- Erroneously the documentation states that the DBMS is capable of handling circular arcs. However, handling circular arcs would be a desirable extension of the functionality in a future release.

6.1.2 Querying

- Topological queries are limited to the any_interact statement. The actual type of interaction can only be found using the geometrical sdo_relate function, which makes a distinction between the interaction types touch, overlapbydisjoint, overlapbyintersect, equal, inside, coveredby, contains, covers, anyinteract and on.

- Queries with a topological nature (like neighbour queries) are potentially much faster when performed on the topological data structure. There is however, due to some 'overhead' in the topology structure, a break-even point where the topological and geometrical approach are equally fast. The larger the data set, the better the relative topological query performance. Time savings of over 50% are attainable when using the topological query form.

- Queries with a geometrical nature (such as line intersections and query windows) are slower when performed on a topological data structure. On average these queries are two or three times slower, but huge deviations occur regularly.

6.2 Conclusions influenced by validation issues

The general conclusion at this stage is that Oracle10g Topology looks promising, but is not very stabile when data contains topological errors.

- Oracle Topology is very sensitive to data errors (for instance the wrongly defined bounding boxes), but at the same time these errors are discovered late (sometimes not until querying) and are accompanied by unclear error messages.

- The get_geometry() function is also very sensitive (for non-validated) data with errors in the left-right references of an edge. Again the error messages are not very clear.

A lesson can also be learned by the Dutch Cadastre as the current situation of duplicating office boundaries in the data sets of both offices causes problems when one wants to combine these data sets. Identical boundaries now have different boundary IDs without any reference to the twin boundary ID.

6.3 Overall conclusion

Oracle 10g Topology is the first attempt to handle topology issues in a DBMS environment and thus has to be welcomed. However, some critical remarks can be made. Erroneously the documentation stated that the DBMS is capable of handling circular arcs. This would be however a desirable extension of the functionality. Other remarks are consequences of certain design decisions that are understandable, although this does not alter the possibility of improving these points in further development. This applies mainly on all remarks on the validation process.

References

[1] Oosterom, P.J.M. van, C.H.J. Lemmen. *Spatial Data Management on a Very Large Cadastral Database.* In: P.J.M. van Oosterom, C.H.J. Lemmen (eds.); Computers, Environment and Urban Systems (CEUS), Jaargang: 2001, vol 25, no. 4-5, 2001, p. 509-528

[2] Oosterom, Peter van, et all. *The Balance Between Geometry and Topology.* In: Advances in Spatial Data Handling, 10th International Symposium on Spatial Data Handling, Dianne Richardson and Peter van Oosterom (eds.), ISBN 3-540-43802-5, Springer-Verlag, Berlin, 2002, pages 209-224.

[3] *Oracle Spatial Topology and Network Data Models, 10g Release 1 (10.1),* Oracle Corp. 2003

Appendix A Scripts

Retrieving data from the cadastral database

```
--
set echo on;
set timing on;
--
-- Make a link to the database with the cadastral data.
--
drop table lki_boundary;
drop table lki_parcel;
drop table lki_parcelover;

create table lki_boundary
as
    select *
    from kadtest.lki_boundary;

create table lki_parcel
as
    select *
    from kadtest.lki_parcel;


create table lki_parcelover
as
    select *
    from kadtest.lki_parcelover
    where object_id in (select distinct object_id from lki_parcel );

create unique index lki_parcel_idx_1 on lki_parcel(object_id);
create index lki_parcelover_idx_1 on lki_parcelover(object_id);
create unique index lki_boundary_1 on lki_boundary(object_id);
create index lki_boundary_2 on lki_boundary(l_obj_id);
create index lki_boundary_3 on lki_boundary(r_obj_id);

analyze table lki_parcel compute statistics;
analyze table lki_parcelover compute statistics;
analyze table lki_boundary compute statistics;

quit;
```

Removing old tables and topologies

```
set echo on;
set timing on;

drop table PARCELS;

execute sdo_topo.delete_topo_geometry_layer('KADASTER','PARCELS','FEATURE');

execute SDO_TOPO.DROP_TOPOLOGY('KADASTER');

select * from user_sdo_topo_info;
select * from user_sdo_topo_metadata;
//select * from kadaster_relation$;

//purge recyclebin;

quit;
```

Alter data to update relationships

```
--
-- Parameters:
-- &1 -- the name of the municipaly.
--
set echo on;
set timing on;



--
-- Setting references to external object to null.
--
update lki_boundary
set l_obj_id = -1
where l_municip = '     ';

update lki_boundary
set r_obj_id = -1
where r_municip = '     ';

--In case one works with only one municipality:

--update lki_boundary
--set ll_line_id = -1
--where abs(ll_line_id) not in (select object_id from lki_boundary);

--update lki_boundary
--set fl_line_id = -1
--where abs(fl_line_id) not in (select object_id from lki_boundary);

--update lki_boundary
--set lr_line_id = 1
--where abs(lr_line_id) not in (select object_id from lki_boundary);

--update lki_boundary
--set fr_line_id = 1
--where abs(fr_line_id) not in (select object_id from lki_boundary);


quit;
```

Create and fill the topology

```
set echo on;
set timing on;

--
-- Create a topology.
--
purge recyclebin;

execute SDO_TOPO.DROP_TOPOLOGY('KADASTER');
execute SDO_TOPO.CREATE_TOPOLOGY('KADASTER', 0.001);

--
-- Load topology data.
--

insert into KADASTER_EDGE$
    select object_id,
    null,
    null,
    ll_line_id,
    -fr_line_id,
    fl_line_id,
    -lr_line_id,
    l_obj_id,
    r_obj_id,
    geo_polyline as geometry
from lki_boundary;


--
-- First create a table with all endpoints of edges.
-- Tp prevent duplicates, the endpoint of a geometry is only added when it's mot
-- the same as the startpoint.
--

drop table vertices purge;
create table vertices(
       id number,
       newid number,
       geometry mdsys.sdo_geometry,
       fromedge number,
       isstart number);

declare
       rownmbr number;
       edgeid number;
       edge mdsys.sdo_geometry;
       start_node mdsys.sdo_geometry;
       end_node mdsys.sdo_geometry;
begin
       rownmbr := 1;
       for e in
              (select  geometry,  edge_id  into  edge,  edgeid  from  kadaster_edge$)
loop
              start_node := startPoint(e.geometry);
              end_node := endPoint(e.geometry);
              insert into vertices values (
                     rownmbr,
                     1,
                     start_node,
                     e.edge_id,
                     0);
              rownmbr := rownmbr+1;
              insert into vertices values (
                     rownmbr,
                     1,
                     end_node,
                     e.edge_id,
                     1);
              rownmbr := rownmbr+1;
       end loop;
end;
/

call createMetaData('VERTICES','GEOMETRY',0.1);

create index vertices_idx1 on vertices(geometry) indextype is mdsys.spatial_index;
create index vertices_idx2 on vertices(id);
analyze table vertices compute statistics;
call analyzeRtree('VERTICES_IDX1');

drop table verts;
create table verts as
 (select distinct(max(b.id)) id, a.id id2  from
       vertices a,
       vertices b,
       table(sdo_join('VERTICES','GEOMETRY','VERTICES','GEOMETRY','mask=ANYINTERAC
T')) c
    where c.rowid1 = a.rowid and c.rowid2 = b.rowid
    group by a.id)
;

drop index verts_idx1;
create index verts_idx1 on verts(id2);
analyze table verts compute statistics;

update vertices v
set newid = (
       select vt.id
       from verts vt
       where (v.id = vt.id2))
;


insert into KADASTER_NODE$
       select v.id as NODE_ID, null as EDGE_ID, null as FACE_ID, v.GEOMETRY
from vertices v where (v.id=v.newid) ;
```

```
--drop table vertices;
--drop table verts purge;
--
-- Create the exterior face.
--
insert into KADASTER_FACE$
values
(
    -1,
    null,
    null,
    null,
    null
);

-- Fill the exterior face.
--
-- TODO: Now we assume that the topology (municipality) consists of one part.
-- TODO: Check about clockwise and ccw.
--
update KADASTER_FACE$
set
    island_edge_id_list = (
            select sdo_list_type(max(object_id))
            from lki_boundary
            where l_municip = '     ')
    where face_id = -1;
insert into KADASTER_FACE$ f
    select
       object_id as face_id,
       -line_id1 as boundary_edge_id,
       null as island_edge_id_list,
       null as island_node_id_list,
       geo_bbox as mbr_geometry
    from lki_parcel;


create or replace function find_islands(
    parcel_id in number)
return sdo_list_type
is
eiland1 number;
rval sdo_list_type := sdo_list_type();
begin
    select line_id2
    into eiland1
    from lki_parcel where object_id = parcel_id;
    if (eiland1 <> 0) then
       rval.extend;
       rval(rval.last) := -eiland1;
    end if;

    for pcover in (
       select line_id1 , line_id2 , line_id3 , line_id4 , line_id5 , line_id6 ,
line_id7 , line_id8 , line_id9 , line_id10
       from lki_parcelover
       where object_id = parcel_id) loop

       if (pcover.line_id1 <> 0) then
           rval.extend;
           rval(rval.last) := -pcover.line_id1;
       end if;
       if (pcover.line_id2 <> 0) then
           rval.extend;
           rval(rval.last) := -pcover.line_id2;
       end if;
       if (pcover.line_id3 <> 0) then
           rval.extend;
           rval(rval.last) := -pcover.line_id3;
       end if;
       if (pcover.line_id4 <> 0) then
           rval.extend;
           rval(rval.last) := -pcover.line_id4;
```

```
end if;
        if (pcover.line_id5 <> 0) then
            rval.extend;
            rval(rval.last) := -pcover.line_id5;
        end if;
        if (pcover.line_id6 <> 0) then
            rval.extend;
            rval(rval.last) := -pcover.line_id6;
        end if;
        if (pcover.line_id7 <> 0) then
            rval.extend;
            rval(rval.last) := -pcover.line_id7;
        end if;
        if (pcover.line_id8 <> 0) then
            rval.extend;
            rval(rval.last) := -pcover.line_id8;
        end if;
        if (pcover.line_id9 <> 0) then
            rval.extend;
            rval(rval.last) := -pcover.line_id9;
        end if;
        if (pcover.line_id10 <> 0) then
            rval.extend;
            rval(rval.last) := -pcover.line_id10;
        end if;

    end loop;

    return rval;
end find_islands;
/
show errors

update KADASTER_FACE$ f
set island_edge_id_list = find_islands(f.face_id)
where f.face_id <> -1;

--
-- Initialize topology metadata.
--
execute sdo_topo.initialize_metadata('KADASTER');

--
-- Connect all edges to their endpoints.
--

drop index vertices_idx3;
drop index vertices_idx4;
create index vertices_idx3 on vertices(fromedge);
create index vertices_idx4 on vertices(isstart);
analyze table vertices compute statistics;

update KADASTER_EDGE$ e
set start_node_id = (
        select newid
        from vertices v
        where (e.edge_id=v.fromedge)and(v.isstart='0')),
        end_node_id = (
        select newid
        from vertices v
        where (e.edge_id=v.fromedge)and(v.isstart='1'));


update KADASTER_NODE$ n
set EDGE_ID = (select min(edge_id) from KADASTER_EDGE$ e where e.start_node_id =
n.node_id)
where edge_id is null;

update KADASTER_NODE$ n
set EDGE_ID = (select -min(edge_id) from KADASTER_EDGE$ e where e.end_node_id =
n.node_id)
where edge_id is null;
```

Create features

```
#!/bin/sh

CONNECT=$1
TOPOLOGY=$2

cat > tmp_topo.sql << EOF
set echo on;
set timing on;

execute sdo_topo.delete_topo_geometry_layer('${TOPOLOGY}','PARCELS','FEATURE');
drop table parcels purge;

create table parcels(
    object_id number,
    feature sdo_topo_geometry);

declare
begin
    sdo_topo.add_topo_geometry_layer('${TOPOLOGY}','PARCELS','FEATURE','POLYGON');
end;
/

--
-- See if this query works.
--

declare
fid number;
tli number;
begin
select tg_layer_id into tli from user_sdo_topo_info where topology = '${TOPOLOGY}'
and table_name = 'PARCELS';
for a in
       (select face_id into fid from kadaster_face$       where  face_id  <>  -1)
loop

       insert into parcels values
              (a.face_id,
              sdo_topo_geometry (
                     '${TOPOLOGY}',        -- topology name
                     3,                    -- topo-geomtry type polygon
                     tli,                  -- TG_LAYER_ID
                     sdo_topo_object_array ( sdo_topo_object (a.face_id,3) ))
               );
end loop;
end;
/

quit;
EOF

sqlplus $1 @tmp_topo

#rm -f tmp_topo.sql
```

Appendix B Test queries

Eck en Wiel and Apeldoorn

```
set echo on;
set timing on;

--
-- geometry queries eck
--
-- query window

select    p.object_id    from    pg_eck    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(15950
0000,442350000,159800000,442650000)))='TRUE';

select    p.object_id    from    pg_eck    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(15880
0000,440700000,160000000,441300000)))='TRUE';

select    p.object_id    from    pg_eck    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(15893
0000,440600000,159470000,442800000)))='TRUE';

select    p.object_id    from    pg_eck    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(15865
0000,440700000,160200000,444000000)))='TRUE';

-- line

select    p.object_id    from    pg_eck    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(158500000,444000000,160000000,441000
000)))='TRUE';

select    p.object_id    from    pg_eck    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(158500000,441800000,
160000000,441800000)))='TRUE';

select    p.object_id    from    pg_eck    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(158900000,440500000,158900000,444000
000)))='TRUE';

select    p.object_id    from    pg_eck    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(159400000,440500000,159400000,444000
000)))='TRUE';

select    p.object_id    from    pg_eck    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(158500000,441000000,160500000,441000
000)))='TRUE';

select    p.object_id    from    pg_eck    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(158500000,443700000,160500000,443700
000)))='TRUE';


-- neighbour

select  p.object_id  from  pg_eck  p,  pg_eck  g  where  g.object_id  =  140560612  and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select  p.object_id  from  pg_eck  p,  pg_eck  g  where  g.object_id  =  140826938  and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';
```

```
select p.object_id from pg_eck p, pg_eck g where g.object_id = 140458960 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select p.object_id from pg_eck p, pg_eck g where g.object_id = 140873993 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select p.object_id from pg_eck p, pg_eck g where g.object_id = 140179548 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';





--
-- topological queries eck
--
-- query window

select q.object_id from parcels_eck q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(15950
0000,442350000,159800000,442650000)))='TRUE';

select q.object_id from parcels_eck q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(15880
0000,440700000,160000000,441300000)))='TRUE';

select q.object_id from parcels_eck q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(15893
0000,440600000,159470000,442800000)))='TRUE';

select q.object_id from parcels_eck q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(15865
0000,440700000,160200000,444000000)))='TRUE';

-- line

select q.object_id from parcels_eck q where sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(158500000,444000000,160000000,441000
000)))='TRUE';

select q.object_id from parcels_eck q where sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(158500000, 441800000, 160000000,
441800000)))='TRUE';

select q.object_id from parcels_eck q where sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(158900000,440500000,158900000,444000
000)))='TRUE';

select q.object_id from parcels_eck q where sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(159400000,440500000,159400000,444000
000)))='TRUE';

select q.object_id from parcels_eck q where sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(158500000,441000000,160500000,441000
000)))='TRUE';

select q.object_id from parcels_eck q where sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(158500000,443700000,160500000,443700
000)))='TRUE';

-- neighbour
select q.object_id from parcels_eck q, parcels_eck r where r.object_id = 140560612
and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select q.object_id from parcels_eck q, parcels_eck r where r.object_id = 140826938
and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select q.object_id from parcels_eck q, parcels_eck r where r.object_id = 140458960
and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;
```

```
select q.object_id from parcels_eck q, parcels_eck r where r.object_id = 140873993
and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select q.object_id from parcels_eck q, parcels_eck r where r.object_id = 140179548
and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

--
-- geometry queries apd
--
-- query window
select    p.object_id    from    pg_apd    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(19530
0000,469460000,195600000,469760000)))='TRUE';

select    p.object_id    from    pg_apd    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(19020
0000,471400000,194800000,475700000)))='TRUE';

select    p.object_id    from    pg_apd    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(18110
0000,467900000,184300000,475400000)))='TRUE';

select    p.object_id    from    pg_apd    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(18210
0000,464300000,196500000,475300000)))='TRUE';

-- line
select    p.object_id    from    pg_apd    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(194400000,472000000,195900000,469000
000)))='TRUE';

select    p.object_id    from    pg_apd    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(194400000,   470000000,   195900000,
470000000)))='TRUE';

select    p.object_id    from    pg_apd    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(180000000,474000000,199000000,474000
000)))='TRUE';

select    p.object_id    from    pg_apd    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(178400000,470200000,200000000,470200
000)))='TRUE';

select    p.object_id    from    pg_apd    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(184200000,462700000,184200000,475800
000)))='TRUE';

-- neighbour
select  p.object_id  from  pg_apd  p,  pg_apd  g  where  g.object_id  =  140187964  and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select  p.object_id  from  pg_apd  p,  pg_apd  g  where  g.object_id  =  140567793  and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select  p.object_id  from  pg_apd  p,  pg_apd  g  where  g.object_id  =  140877486  and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select  p.object_id  from  pg_apd  p,  pg_apd  g  where  g.object_id  =  140499936  and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select  p.object_id  from  pg_apd  p,  pg_apd  g  where  g.object_id  =  140232417  and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

--
-- topological queries apd
--
-- query window

select   q.object_id   from   parcels_apd1   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(19530
0000,469460000,195600000,469760000)))='TRUE';
```

```
select   q.object_id   from   parcels_apd1   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(19020
0000,471400000,194800000,475700000)))='TRUE';

select   q.object_id   from   parcels_apd1   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(18110
0000,467900000,184300000,475400000)))='TRUE';

select   q.object_id   from   parcels_apd1   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(18210
0000,464300000,196500000,475300000)))='TRUE';

-- line
select   q.object_id   from   parcels_apd1   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(194400000,472000000,195900000,469000
000)))='TRUE';

select   q.object_id   from   parcels_apd1   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(194400000,   470000000,   195900000,
470000000)))='TRUE';

select   q.object_id   from   parcels_apd1   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(180000000,474000000,199000000,474000
000)))='TRUE';

select   q.object_id   from   parcels_apd1   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(178400000,470200000,200000000,470200
000)))='TRUE';

select   q.object_id   from   parcels_apd1   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(184200000,462700000,184200000,475800
000)))='TRUE';

-- neighbour
select  q.object_id  from  parcels_apd1  q,  parcels_apd1  r  where  r.object_id  =
140187964 and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select  q.object_id  from  parcels_apd1  q,  parcels_apd1  r  where  r.object_id  =
140567793 and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select  q.object_id  from  parcels_apd1  q,  parcels_apd1  r  where  r.object_id  =
140877486 and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select  q.object_id  from  parcels_apd1  q,  parcels_apd1  r  where  r.object_id  =
140499936 and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select  q.object_id  from  parcels_apd1  q,  parcels_apd1  r  where  r.object_id  =
140232417 and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;
--
-- extra large line queries apd
--     geometry
--

select    p.object_id    from    pg_apd    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(194400000,466600000,194400000,476200
000)))='TRUE';

-- topo
select   q.object_id   from   parcels_apd   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(194400000,466600000,194400000,476200
000)))='TRUE';


quit;
```

Zoetermeer office

```
set echo on;
set timing on;

--
-- geometry queries
--
-- line

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(62117500,440570000,137367000,4376600
00)))='TRUE';

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(94822500,480343000,94545300,41424000
0)))='TRUE';

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(71679500,427128000,94545300,41424000
0)))='TRUE';

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(102029000,466762000,99672800,4623270
00,94961100,458308000,92050900,452627000,84290400,449024000,81934500,442926000,773
61300,441263000,82904600,434473000,92466600,432948000,104523000,432394000,11339200
0,433780000,121291000,434750000,131824000,435858000,137090000,438353000)))='TRUE';

-- query window

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,76000000,42600
0000)))='TRUE';

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,77000000,42700
0000)))='TRUE';

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,79000000,42900
0000)))='TRUE';

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,83000000,43300
0000)))='TRUE';

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,91000000,44100
0000)))='TRUE';

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,107000000,4570
00000)))='TRUE';

-- complex query shapes

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,1,11,2003,1),SDO_ORDINATE_ARRAY(87000000,456000000,9000
0000,456000000,90000000,459000000,87000000,459000000,87000000,456000000,88000000,4
57500000,88500000,458000000,89000000,457500000,88500000,457000000,88000000,4575000
00)))='TRUE';

select    p.object_id    from    pg_small    p    where    sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,1,33,2003,1,43,2003,1),SDO_ORDINATE_ARRAY(109690000,454
447000,109899000,453966000,110442000,453925000,110630000,454259000,110943000,45425
9000,111131000,453946000,111653000,453904000,111924000,454154000,111904000,4544880
00,111611000,454823000,111152000,454802000,110943000,454468000,110630000,454488000
,110421000,454802000,109962000,454802000,109690000,454447000,110024000,454509000,1
10316000,454509000,110358000,454217000,110045000,454280000,110024000,454509000,111
298000,454509000,111611000,454468000,111569000,454217000,111298000,454259000,11129
8000,45450900)))='TRUE';

-- neighbour

select p.object_id from pg_small p, pg_small g where g.object_id = 340633554 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select p.object_id from pg_small p, pg_small g where g.object_id = 310460084 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select p.object_id from pg_small p, pg_small g where g.object_id = 340621392 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select p.object_id from pg_small p, pg_small g where g.object_id = 310391758 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select p.object_id from pg_small p, pg_small g where g.object_id = 340644065 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

--
-- topological queries
--
--     line

select   q.object_id   from   parcels_small   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(62117500,440570000,137367000,4376600
00)))='TRUE';

select   q.object_id   from   parcels_small   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(94822500,480343000,94545300,41424000
0)))='TRUE';

select   q.object_id   from   parcels_small   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(71679500,427128000,94545300,41424000
0)))='TRUE';

select   q.object_id   from   parcels_small   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(102029000,466762000,99672800,4623270
00,94961100,458308000,92050900,452627000,84290400,449024000,81934500,442926000,773
61300,441263000,82904600,434473000,92466600,432948000,104523000,432394000,11339200
0,433780000,121291000,434750000,131824000,435858000,137090000,438353000)))='TRUE';

-- query window
select   q.object_id   from   parcels_small   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,76000000,42600
0000)))='TRUE';

select   q.object_id   from   parcels_small   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,77000000,42700
0000)))='TRUE';

select   q.object_id   from   parcels_small   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,79000000,42900
0000)))='TRUE';

select   q.object_id   from   parcels_small   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,83000000,43300
0000)))='TRUE';

select   q.object_id   from   parcels_small   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,91000000,44100
0000)))='TRUE';

select   q.object_id   from   parcels_small   q   where   sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,107000000,4570
00000)))='TRUE';

-- complex shapes

select q.object_id from parcels_small q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,1,11,2003,1),SDO_ORDINATE_ARRAY(87000000,456000000,9000
0000,456000000,90000000,459000000,87000000,459000000,87000000,456000000,88000000,4
57500000,88500000,458000000,89000000,457500000,88500000,457000000,88000000,4575000
00)))='TRUE';

select q.object_id from parcels_small q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,1,33,2003,1,43,2003,1),SDO_ORDINATE_ARRAY(109690000,454
447000,109899000,453966000,110442000,453925000,110630000,454259000,110943000,45425
9000,111131000,453946000,111653000,453904000,111924000,454154000,111904000,4544880
00,111611000,454823000,111152000,454802000,110943000,454468000,110630000,454488000
,110421000,454802000,109962000,454802000,109690000,454447000,110024000,454509000,1
10316000,454509000,110358000,454217000,110045000,454280000,110024000,454509000,111
298000,454509000,111611000,454468000,111569000,454217000,111298000,454259000,11129
8000,45450900)))='TRUE';

-- neighbour

select q.object_id from parcels_small q, parcels_small r where r.object_id =
340633554 and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select q.object_id from parcels_small q, parcels_small r where r.object_id =
310460084 and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select q.object_id from parcels_small q, parcels_small r where r.object_id =
340621392 and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select q.object_id from parcels_small q, parcels_small r where r.object_id =
310391758 and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select q.object_id from parcels_small q, parcels_small r where r.object_id =
340644065 and sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

quit;
```

Zuid-Holland

```
set echo on;
set timing on;

--
-- geometry queries
--
-- line

select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(62117500,440570000,137367000,4376600
00)))='TRUE';

select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(94822500,480343000,94545300,41424000
0)))='TRUE';

select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(71679500,427128000,94545300,41424000
0)))='TRUE';

select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(102029000,466762000,99672800,4623270
00,94961100,458308000,92050900,452627000,84290400,449024000,81934500,442926000,773
61300,441263000,82904600,434473000,92466600,432948000,104523000,432394000,11339200
0,433780000,121291000,434750000,131824000,435858000,137090000,438353000)))='TRUE';

-- query window
```

```
select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,76000000,42600
0000)))='TRUE';

select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,77000000,42700
0000)))='TRUE';

select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,79000000,42900
0000)))='TRUE';

select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,83000000,43300
0000)))='TRUE';

select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,91000000,44100
0000)))='TRUE';

select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,107000000,4570
00000)))='TRUE';

-- complex query shapes

select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,1,11,2003,1),SDO_ORDINATE_ARRAY(87000000,456000000,9000
0000,456000000,90000000,459000000,87000000,459000000,87000000,456000000,88000000,4
57500000,88500000,458000000,89000000,457500000,88500000,457000000,88000000,4575000
00)))='TRUE';

select p.object_id from pg p where sdo_anyinteract(p.geom,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,1,33,2003,1,43,2003,1),SDO_ORDINATE_ARRAY(109690000,454
447000,109899000,453966000,110442000,453925000,110630000,454259000,110943000,45425
9000,111131000,453946000,111653000,453904000,111924000,454154000,111904000,4544880
00,111611000,454823000,111152000,454802000,110943000,454468000,110630000,454488000
,110421000,454802000,109962000,454802000,109690000,454447000,110024000,454509000,1
10316000,454509000,110358000,454217000,110045000,454280000,110024000,454509000,111
298000,454509000,111611000,454468000,111569000,454217000,111298000,454259000,11129
8000,45450900)))='TRUE';

-- neighbour

select p.object_id from pg p, pg g where g.object_id = 340633554 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select p.object_id from pg p, pg g where g.object_id = 310460084 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select p.object_id from pg p, pg g where g.object_id = 340621392 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select p.object_id from pg p, pg g where g.object_id = 310391758 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

select p.object_id from pg p, pg g where g.object_id = 340644065 and
sdo_anyinteract(p.geom, g.geom) = 'TRUE';

--
-- topological queries
--
--     line

select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(62117500,440570000,137367000,4376600
00)))='TRUE';
```

```
select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(94822500,480343000,94545300,41424000
0)))='TRUE';

select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(71679500,427128000,94545300,41424000
0)))='TRUE';

select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2002,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(102029000,466762000,99672800,4623270
00,94961100,458308000,92050900,452627000,84290400,449024000,81934500,442926000,773
61300,441263000,82904600,434473000,92466600,432948000,104523000,432394000,11339200
0,433780000,121291000,434750000,131824000,435858000,137090000,438353000)))='TRUE';

-- query window
select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,76000000,42600
0000)))='TRUE';

select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,77000000,42700
0000)))='TRUE';

select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,79000000,42900
0000)))='TRUE';

select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,83000000,43300
0000)))='TRUE';

select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,91000000,44100
0000)))='TRUE';

select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(75000000,425000000,107000000,4570
00000)))='TRUE';

-- complex shapes

select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,1,11,2003,1),SDO_ORDINATE_ARRAY(87000000,456000000,9000
0000,456000000,90000000,459000000,87000000,459000000,87000000,456000000,88000000,4
57500000,88500000,458000000,89000000,457500000,88500000,457000000,88000000,4575000
00)))='TRUE';

select q.object_id from parcels q where sdo_anyinteract(q.feature,
sdo_geometry(2003,NULL,NULL,
SDO_ELEM_INFO_ARRAY(1,1003,1,33,2003,1,43,2003,1),SDO_ORDINATE_ARRAY(109690000,454
447000,109899000,453966000,110442000,453925000,110630000,454259000,110943000,45425
9000,111131000,453946000,111653000,453904000,111924000,454154000,111904000,4544880
00,111611000,454823000,111152000,454802000,110943000,454468000,110630000,454488000
,110421000,454802000,109962000,454802000,109690000,454447000,110024000,454509000,1
10316000,454509000,110358000,454217000,110045000,454280000,110024000,454509000,111
298000,454509000,111611000,454468000,111569000,454217000,111298000,454259000,11129
8000,45450900)))='TRUE';

-- neighbour

select q.object_id from parcels q, parcels r where r.object_id = 340633554 and
sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select q.object_id from parcels q, parcels r where r.object_id = 310460084 and
sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;
```

```
select  q.object_id  from  parcels  q,  parcels  r  where  r.object_id  =  340621392  and
sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select  q.object_id  from  parcels  q,  parcels  r  where  r.object_id  =  310391758  and
sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

select  q.object_id  from  parcels  q,  parcels  r  where  r.object_id  =  340644065  and
sdo_anyinteract(q.feature, r.feature) = 'TRUE'  ;

quit;
```