TESTING CURRENT DBMS PRODUCTS WITH REAL SPATIAL DATA

Peter van Oosterom, Wilko Quak & THEO TIJSSEN

Department of Geodesy
Faculty of Civil Engineering and Geosciences
Delft University of Technology
The Netherlands

ABSTRACT

This paper analyses three commercially available geo-DBMSs, several different recent versions of Oracle, Informix and Ingres, with respect to their offered functionality and their performance (and correctness of the results). The functionality analysis is based on the manuals of the different systems and comparing this to the OpenGIS Simple Feature Specifications for SQL. In addition this paper touches on the topics, which are currently outside the scope of this specification: 3D data, raster (geo-image) data, topology structures, spatial indexing and clustering. The performance of the three different systems is evaluated by loading two large cadastral data sets (one with 21 million spatial records and one with 122 million spatial records) and firing a set of representative queries on these data sets in order to compare the results and test the scalability. It is very difficult to draw overall conclusions but it is clear that Ingres has the least richness with respect to the functionality, that Informix is the only one compliant to the OpenGIS specifications and that Oracle does not always have the best performance, but it is the richest with respect to the functionality.

1. INTRODUCTION

Support for spatial data types is becoming more and more commonplace in standard DBMSs (Oracle, IBM/Informix, Ingres, Postgres). The expected users of this functionality will come from two groups: First, maintainers of big spatial datasets will migrate from proprietary GIS solutions. For them the benefits will be that they can integrate their spatial data with administrative data that was already stored inside a DBMS. The second group of users do not maintain spatial data themselves but use spatial data from third parties to analyse their own data. Both groups will only migrate if functionality and performance of the DBMS is sufficient. The first group will focus more on the edit functionality, while the second group will focus on the analysis functionality. In the functional analysis we try to give an overview and comparison of the functionality of the different databases. The functional analysis will consist of a few parts: 1. What are the data types that are supported: Point, Line, Polygon, Networks, Linear-Referencing, and Topological Maps? 2. Which spatial operators are supported by the database:

Distance, buffer, etc.? and 3. Do the spatial types and operators comply with the OpenGIS specification?

In a previous paper (*Quak et al, 2002*), we tested the performance of several commercial spatial DBMSs using a large cadastral data set consisting of the data of 3 cadastral offices. In the performance analysis presented in this paper we now also load and query a second, even larger, data set: one consisting of the data of 15 cadastral offices, including the complete cadastral (and large scale topographic) map of the Netherlands. This paper is structured as follows: Section 2 gives an overview of the test configuration, used hard- and software and the test data sets (3 and 15 cadastral offices). Loading the test data in the databases is discussed in section 3, including the creation of spatial indices. The performance benchmark results are given in Section 4 for both data sets. Section 5 presents an overview of the functionality available within the different geo-DBMSs. The main results and future work can finally be found in the conclusion, section 6.

2. TEST CONFIGURATION AND TEST DATA

In this section we give an overview of the test configuration (hardware and software used described in subsection 2.1) and the used test data sets (described in subsection 2.2).

2.1 Test configuration

All experiments have been performed on a Sun Enterprise E3500 server at the GDMC (Geo-Database Management Center) in Delft. This computer has two 400 MHz UltraSPARC CPUs, 2 Gb of main memory, and 600Gb of disk space, on several RAID0 and RAID5 sets. Details can be found in (*Tijssen et al. 2001*). In the experiments the following DBMSs were tested: Oracle 8.1.7, Oracle 9.0.1, Oracle 9.2.0, Ingres 2.0, Ingres 2.5 and Informix Spatial 8.11. All of the databases were installed on the Unix file system; no raw devices were used.
All DBMSs have an almost infinite amount of parameters that can be tuned for optimal performance. In the experiments the same amount of time has been spent on tuning the different systems with the assistance of the involved DBMS vendors.

2.2 Test data

In the experiments we used two 'test' data sets from the Dutch Cadastre. The first data set is dated 1 October 1999 and consists of the geographic and administrative data of three cadastral offices. The second test data set is dated 1 June 2001 and contains the geographic and administrative data of fifteen cadastral offices, covering the whole Netherlands.
The cadastral map data, a part of the geographic data set, is stored in a topological model (*Van Oosterom 1997*). The first test data set contains the geometry of almost 4 million parcels (faces) and 10 million boundary lines (edges). In the second data set these numbers are 17 million parcels and 46 million boundary lines. The total number of spatial objects is 21 million and 122 million rows in respectively the first and second data set. This database also contains historic data since 1997. This is implemented by adding a time stamp attribute to all spatial data (*Van Oosterom, 1997*).

The administrative data contains the ownership information of all current parcels. In this paper we will focus on the geographic tables and specifically on the table containing the parcel boundaries.

3. PREPARING THE TEST ENVIRONMENT

This section describes the bulk loading of the spatial data into the DBMS. The loading of the data consists of two parts. First, the data is loaded into the system (subsection 3.1), and then index structures are created on the data (subsection 3.2). The dataset consists of a set of base tables that contain the real data. In this section we will concentrate on the spatial data, which are stored in 7 tables, the so-called 'LKI' tables. On top of these tables many database views exist. In subsection 3.3 we discuss the spatial clustering of the data as it turned out that this has a major influence on query performance.

3.1 Loading the data

In the first stage, the loading of the data, we researched the time needed to load the data, and the size of the loaded data. It was attempted to keep the hardware conditions as equal as possible for the three geo-DBMS, e.g. all systems used 3 RAID5 sets for the data storage. In Table 1 the overall results of loading the first test data set (3 offices) in the different systems are given. As performance of the initial loading of the data and the creation of indices were not considered to be the most critical aspect of a geo-DBMS, we did not try to optimise the speed of these processes. Also, during data loading and index creation, the system load was not controlled and other processes could have influenced the reported times. However, the timings are given, just in order to get a 'feeling' of the amount of data, which has to be loaded. The poor performance of the Ingres database can be explained by the fact that the Ingres bulk loading tools cannot handle spatial data; this means that all spatial data needs to be inserted with separate 'insert' queries.

	Loading time	Database size
Ingres 2.0	35 hours	4234 Mb
Oracle 8.1.7	7 hours	7646 Mb
Informix 8.11	12 hours	9272 Mb

Table 1: Database loading time (rounded) and database size 3 offices (tables containing spatial data: totals of 7 LKI tables; 21.165.946 records).

Table 2 shows the result of loading the second test data set (15 offices). Unfortunately, we did not yet have the time before the writing of this paper to load the second data set into Informix. Remarkable are the differences in the storage requirements for the base tables for basically the same data models (as much as possible the same SQL create table statements are used): roughly stated Ingres uses about half the space as needed by Oracle and Informix. Part of this difference can be explained by the fact that Ingres stores the co-ordinates with integer (32bit) precision where the others store the co-ordinates with more precision. The load time and database size scale well (more or less linear to the amount of data). It is a bit dangerous to compare the results from Table 1 and Table 2, because also different versions of the software were used

	Loading time	Database size
Ingres 2.5	96 hours	24.296 Mb
Oracle 9.2.0	77 hours	48.151 Mb

Table 2: Database loading time (rounded) and database size 15 offices (tables containing spatial data: totals of 7 LKI tables; 122.203.421 records).

3.2 Creating spatial indices

After the data is loaded, index structures that accelerate query operations on the data are built. Different DBMSs use different structures to accelerate the queries. All tested DBMSs offer R-tree indexing (*Guttman 1984*) or variants on it. Oracle also provides a quad-tree like index (*Samet 1990*) and an index structure based on regular grids.
The creation of index structures is not yet standardised in SQL as can be seen from the statements that are needed for the different DBMSs below.

```
/* Oracle */
/* first populate table user_sdo_geometry_metadata with an entry
*/
    insert into user_sdo_geom_metadata
    values ('xfio_boundary','shape',mdsys.sdo_dim_array(
    mdsys.sdo_dim_element('X',-25000000,325000000,0.5),
    mdsys.sdo_dim_element('Y',275000000,625000000,0.5)),NULL);
/* then define the spatial index */
   create index xfiox_boundary_0
   on xfio_boundary (shape)
   indextype is mdsys.spatial_index
   parameters ('sdo_fanout=64 sdo_rtr_pctfree=10');

/* Informix */
   create index xfiox_boundary_0
    on xfio_boundary (shape st_geometry_ops)
    using rtree (NO_SORT='FALSE', FILL_FACTOR='90',
     BOUNDING_BOX_INDEX='NO');

/* Ingres */
   create index xfiox_boundary_0
    on xfio_boundary(shape)
    with structure=rtree,
    range=((-25000000,275000000),(325000000,625000000));
```

Tables 3 and 4 contain an overview of the resources needed for the spatial indices on the two data sets. Because of the different architectures of the DBMSs it is sometimes hard to give good comparison. In Tables 3 and 4 the time for index creation and analysis are added together. Informix has an option to include the actual shape in the R-tree index besides the boxes always stored in the R-tree. This results in a bigger, but very efficient 'index' for queries only retrieving the actual shapes based on a spatial selection as the base table does not have to be accessed. As many indices are created, spatial and non-spatial, the results of a selection of 8 representative indices are shown. Out of the total of 8 indices, 4 are spatial indices (on boundary, parcel and text tables); see Table 7 of (*Tijssen et al, 2001*). With respect to the index sizes, it can be concluded that Oracle and Informix use about 2.5 times more space. When in Informix the shape is included in the index its size grows even more of course. Further analysis reveals that this is due to the spatial indices and not to the alphanumerical indices. Again, it is difficult to compare the results from Table 3 (3 offices) and Table 4 (15 offices), because also newer versions of the software were used. However, there was a remarkable improvement in the creation time of indices between Ingres version 2.0 and 2.5.

	Index creation time	Index size
Ingres 2.0	6 hours	820 Mb
Oracle 8.1.7	12 hours	2.171 Mb
Informix 8.11 (only bbox)	6 hours	2.523 Mb
Informix 8.11 (with shapes)	7 hours	3.645 Mb

Table 3: Index creation time and index size (3 offices, totals of 8 'LKI example indices').

	Index creation time	Index size
Ingres 2.5	5 hours	4.589 Mb
Oracle 9.2.0	75 hours	10.486 Mb

Table 4: Index creation time and index size (15 offices, totals of 8 'LKI example indices').

3.3 Spatial clustering of the data

The way the spatial data is physically distributed on disk appears to be very important for response time of queries on this data. This means that the data on the disk is organised in such a way, that objects that are spatially close to each other are also close together on disk as most queries also involve spatially close objects. For this reason we tried to spatially cluster the data in all DBMSs. Depending on the DBMS there are, in theory, three different ways of achieving this disk clustering, which are handled in the next three paragraphs:

Clustering using a spatial index: Most DBMSs can use a so-called primary index to determine the physical storage of the data and in this way determine the clustering of the data. However, currently this option is not available in any of the DBMSs for the spatial index as primary storage structure of a table. Only standard index structures (like B-trees) can be used for clustering.

Clustering using a non-spatial attribute: As the DBMSs do not allow direct clustering on a spatial attribute (primary index), but do allow the clustering on an alpha numerical attribute (which can have a strong relation to location; e.g. a zip-code), a form of spatial clustering can be obtained. This is possible in all three of the tested DBMSs. However, in Oracle it is then impossible to define a (secondary) spatial index on such a table (which is called an index organized table in Oracle terms). This makes this type of spatial clustering in Oracle useless in practice. Below an Ingres example, which shows how the parcels table in the system can be clustered on the alphanumerical Spatial Location Code (SLC) with the following command:

```
modify parcels to btree on slc;
```

Clustering before Load Time: If the DBMS does not offer support for the clustering of spatial data, it can still be attempted to cluster the data by controlling the order in which the objects are loaded into the database. Although it is not documented, and not guaranteed by the system, most systems insert the objects on disk in the order in which they are loaded. So by ordering the data before it is loaded into the system a very good clustering can sometimes be

achieved. Drawback of this method is that after updates, deletes and inserts the clustering will get worse over time. (and thus the query performance).

Sometimes partitioning of one table into a number a partitions based on some kind of attributes is suggested. For example in the case of the second data set the partitions could be based on office code, will result in 15 partitions with their own physical storage of the data. This will indeed avoid a totally random distribution of the data on the disk, but will be a too coarse organization for efficient spatial retrieval.

In this subsection we discussed the clustering of the base tables. However, when spatial indices become very large, also the clustering of the indices themselves becomes relevant as a random distribution of the nodes of an R-tree on disk would slow down the use of the spatial index itself. As far as we could check only Informix offers clustering of the spatial indexed itself (via the 'NO_SORT' parameter in the 'create index' statement).

4. QUERYING THE DATA

The Cadastre provided us with sample questions from cases they encountered regularly and which they wanted to be solved by the database. In subsection 4.1 we give an overview of the different types of spatial queries relevant to this research. Subsection 4.2 reports the results of the selected queries, both in terms of correctness (number of reported objects) and in terms of performance (elapse times in a system with no other significant load).

4.1 Types of spatial queries

The Cadastre questions were translated into SQL queries, which were tested on all databases. All operations in the experiments focus on querying the data. This means update, delete or insert operations are not considered. Possible types of the queries in the test data set are:

- Type 1: Find and display all objects within a certain rectangular area of the map. Normally this very common query is solved in the database by first filtering based on the Minimum Bounding Rectangles (MBR) of all the objects and then using the actual shape of the objects to produce the final result.
- Type 2: Find all objects that overlap with a given polygon. In this case the query polygon can be irregular that is, it can be defined by many points, be non-convex and may also have one or more holes. This means that the DBMS has to perform an intersection test with all objects in the database. If the query polygon has many points, this operation can become very expensive. Therefore, usually the database also first performs some kind of filtering based on the MBRs of both the search polygon and the shapes of the objects in order to reduce the amount of expensive computation. For the user, this filtering is and should be invisible and it is during this phase that the spatial clustering and indexing plays an important role in the internals of the DBMS.
- Type 3: Find and clip all objects with a given search polygon. Now the objects have to be clipped as well based on the computed intersection with the search polygon. Again, this is an expensive operation and filtering may be useful.

- Type 4: Find all objects that intersect with a (possibly very long) query polyline. Although the number of objects returned can be small, the standard filtering on MBR will not work very well, because the filter will return far too many objects.

In this paper we present the benchmark results of query types 2 and 4 on one of the larger tables, the boundary table. The spatial selection is based on overlap of the search geometry with that of the shape attribute (and not the MBR of the boundary object). This implies that for the 'select count(*)' type of queries, as will be presented in this section, both the index and the boundary table have to be accessed and fetched from disk in the case of Oracle and Ingres. In the case of Informix, with shapes also stored in the R-tree index, the query can be solved by only accessing the index. In order to obtain reproducible results, it was decided to perform 'cold' tests by rebooting the machine and starting the DBMS server just before a test is run. 'Hot' is the situation where some or all data has been retrieved before and is possibly present in one of the caches. Note that this caching can occur at several levels: the DBMS level, the Operating System level and the RAID (disks) level. Below an example query is shown (Informix, that is according to the OpenGIS SFS standard):

```
select count(*) from xfio_boundary
where ST_Intersects (shape, ST_PolyFromText(
  'polygon ((78550000 457900000,78550000 458025000,\
  78650000 458025000,78650000 457900000,78550000
457900000))',28992));
```

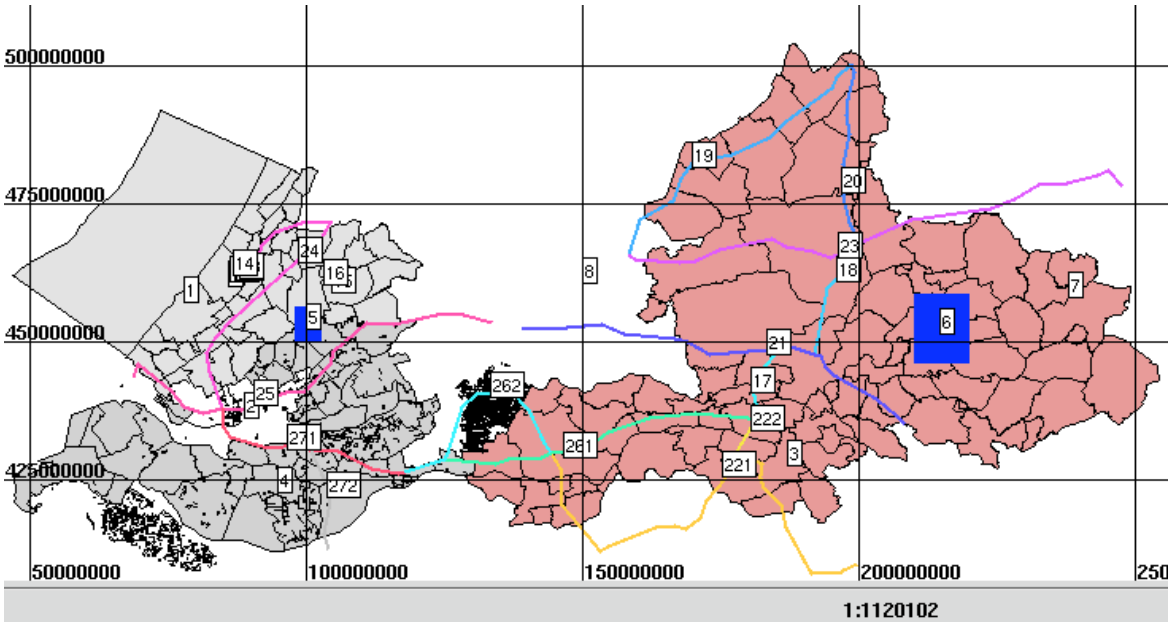

Figure 1: Overview of cadastral selection/query geometries.

Figure 1 shows the different query geometries against a background of municipalities from the 3 offices test data set. Table 5 gives an overview of the selected queries, of which the results are presented in this paper. In this table a short verbal description of each query geometry is given, together with some numbers indicated that the amount of data involved in the boundary table: '# objects inside MBR' is the number of geometries from the table against which the queries are

made (containing large scale cadastral parcel boundaries) which have their own MBR overlap with the MBR of the query geometry. The bigger numbers for the '15 offices' data result from a later moment in time (the cadastral data contains history) and/or inclusion of data of more offices.

Query#	Query geometry description	# objects inside MBR (3 offices)	# objects inside MBR (15 offices)
1	Rectangle 100 x 125 m	423	546
4	Rectangle 1000 x 1250 m	784	964
6	Rectangle 10000 x 12500 m	142405	175144
10	Regular polygon with 8 points	2114	2264
14	Irregular polygon with 50+ points	1082	1215
16	"glasses" shaped polygon with 2 holes	13699	31247
20	Vertical, long polyline	94963	137389
24	Diagonal, long polyline	1932126	2500267
26	Relatively horizontal, long multi-polyline	1028776	1981833

Table 5: Selected queries and the number of reported objects based the MBR-overlap filter.

4.2 Query results

Basic windowing queries are very important and in general are performed well by all DBMSs. This query selects all objects that overlap with a given rectangle. It is executed when you draw a map on the rectangular screen. We tested the DBMSs by selecting rectangles with different sizes (ranging from a few thousand square meters to an area of 45km x 45km). Some of these rectangles are in densely populated areas and some not. Tables 6 and 7 show the results of these queries for respectively the 3 and 15 offices data sets. It can be concluded that the DBMSs are fast enough for interactive mapping.

One of the query types in the experiments that showed the biggest difference in response time for the different databases was the following: Find all parcels that overlap with a very long linear object (for example a centreline of a road). The line (labelled '20' in Figure 1) intersects with 670 parcel boundaries in the first data set (3 offices) and 1357 in the second data set (15 offices). As can be seen from the test results in Table 6 the difference in response time between the different DBMSs is huge, especially Ingres and Oracle were very slow for the very long polyline queries. After reporting the results of this query to Oracle they responded with sending a patch. After installing the patch, performance for this query increased tremendously, which can also be seen from the results from Table 7 (column Oracle 9.2.0). Apparently the different DBMSs have very different query plans for this query.

Query	# result objects	Ingres 2.0	Oracle 8.1.7	Informix 8.11
Small rectangle	415	1	4	0
Medium sized rectangle	782	0	0	0
Large rectangle	142405	22	44	22
Regular polygon	1680	1	1	0
Irregular polygon	549	0	2	0
Glasses shaped polygon	10035	2	17	3
Vertical polyline	670	11	99	1
Diagonal polyline	1753	213	3525	4
Multi-polyline	2385	709	1482	4

*Table 6: Query results for 3 offices, timings in seconds (rounded), production software (table 'lki_boundary', attribute 'shape', operation 'select count *').*

Query	# result objects	Ingres 2.5	Oracle 9.0.1	Oracle 9.2.0
Small rectangle	415	1	3	1
Medium sized rectangle	804	0	2	1
Large rectangle	175098	35	183	23
Regular polygon	1704	1	3	1
Irregular polygon	550	1	1	1
Glasses shaped polygon	22402	6	23	12
Vertical polyline	(9i: 1412) 1357	17	16	6
Diagonal polyline	(9i: 2716) 2561	473	208	43
Multi-polyline	(9i: 4657) 4514	3424	195	63

*Table 7: Query results for 15 offices, timings in seconds (rounded), production software (table 'lki_boundary', attribute 'shape', operation 'select count *').*

The correctness of the results where checked by some visual inspection and also, and this did give the greater confidence, by checking the results from these completely different DBMSs to each other. In all cases the results were identical with exception of one situation: Oracle 9.0.1 and 9.2.0 report different numbers of resulting objects for all polyline queries compared to Ingres 2.5. Investigating some of the additionally reported objects showed that these correspond to circular arcs represented by 3 points on a straight line, which can be considered as a data error. Why did we not see this with Oracle 8.1.7? The reason for this is that in the tests on the first data set (3 offices), arcs were treated as straight lines and this was corrected in the Oracle case of the second test data set (15 offices). Ingres does not support the circular arc type (only circles, but these are irrelevant in this context). For more information with respect of the supported data types in the different DBMSs, see subsection 5.1.

In subsection 3.3 the importance of spatial clustering was stressed. The results presented in this section are based on loading more or less presorted data. As a simple test we also tried to sort the data randomly before inserting the data in the DBMSs. As Ingres and Informix support

some kind of spatial clustering their query performance was identical to the performance reported in this section. However, it turned out that the performance of Oracle was about 30 times slower compared to the Oracle figures presented in this section for test data set 1. For the larger test data set 2 even a larger performance difference is expected, but we did not test this.

5. FUNCTIONALITY COMPARISON

Next to correctness and performance also the richness of the functionality is important for a geographic extension. In this section we will try to give an overview of the functionality of the different extensions. The functionality will be compared with the OpenGIS Simple Features Specification for SQL (*Open GIS Consortium, Inc., 1998*). The functionality supported by the different DBMSs is retrieved from their documentation: Oracle9i (*Murray, 2001*), Informix (*Informix, 2000*) and Ingres (*ASK-OpenIngres, 1994*). Subsection 5.1 gives an overview of the supported data types. The next subsections each discuss a group of operations: data type component access (subsection 5.2), topological relationship operators (subsection 5.3), geometric operations (subsection 5.4) and aggregate functions (subsection 5.5).

5.1 Spatial Types Supported

The first thing to compare are the spatial data types. Table 8 gives an overview of the types. An interesting observation is that Oracle9i supports only one type (SDO_GEOMETRY) that can store a geometry of any type. This means that if you create a table in which you want to store only points, the user has to make sure that only points are inserted. However, the SDO_GEOMETRY type is powerful enough to store any OpenGIS type. The Informix types are OpenGIS compliant. The Ingres extension is from well before the OpenGIS specification and cannot handle objects with islands and multigeometries. It has support for: polygons without islands, line segments and circles. The maximum number of point per polyline or polygon is quite limited, which is not the case for Oracle and Informix. However, Ingres does offer a choice between double precision floating point coordinates or integer coordinates, which can be very efficient. Oracle supports the use of circular arcs in its spatial types. Both Oracle9i and Informix have Geodetic coordinates besides Cartesian coordinates. Oracle9i has support for Linear Referencing.

OpenGIS	Oracle	Informix	Ingres
Point	SDO_GEOMETRY	ST_Point	point
LineString	SDO_GEOMETRY	ST_LineString	line
Polygon	SDO_GEOMETRY	ST_Polygon	polygon
GeomCollection	SDO_GEOMETRY	ST_GeomCollection	-
MultiPoint	SDO_GEOMETRY	ST_MultiPoint	-
MultiLineString	SDO_GEOMETRY	ST_MultiLineString	-
MultiPolygon	SDO_GEOMETRY	ST_MultiPolygon	-

Table 8: Spatial types supported by DBMSs.

5.2 Data type component access

Data type component access functions retrieve a part of a spatial atom, such as the starting point of a LineString, the number of points in a Polygon, etc. Here Oracle9i has an absolute minimum of retrieval functions. However the documentation provides a complete description of the internal representation of SDO_GEOMETRY and the user could develop these functions. Informix has an extensive list of retrieval functions. Ingres has not too many. See Table 9 for a non-complete overview.

Ingres	Box_ll Box_ur Point_x Point_y numpoints
Oracle	GET_DIMS GET_GTYPE
Informix	ST_CoordDim ST_EndPoint ST_PointN ST_SRID ST_StartPoint ST_X ST_Y ST_Z ST_ExteriorRing ST_GeometryN ST_GeometryType ST_InteriorRingN ST_NumGeometries ST_NumInteriorRing ST_NumPoints ST_Point

Table 9: Component access methods.

5.3 Topological relationship operators

Topological relationship operators are used to determine the topological relationship between two geometries. In the OpenGIS specification the relations are defined using the dimensionally extended nine-intersection model, some of these relationships are named and have a separate function. In Ingres the support for topological relationships is minimal, the other databases both support topological operations. An overview is given in Table 10.

In Oracle9i all topological relations are implemented using one function (SDO_GEOM.RELATE) where the type of the relation is passed as a text string. Table 10 gives the value of the text parameter.

OpenGIS	**Oracle**	**Informix**	**Ingres**
Equals	'EQUAL'	ST_Equals	==
Disjoint	'DISJOINT'	ST_Disjoint	not overlaps
Intersects	'ANYINTERACT'	ST_Intersects	overlaps
Touches	'TOUCH'	ST_Touches	-
Crosses	'OVERLAPBDYDISJOINT'	ST_Crosses	-
Within	'INSIDE'	ST_Within	inside
Contains	'CONTAINS'	ST_Contains	-
Overlaps	'OVERLAPBDYINTERSECT'	ST_Overlaps	-
Relate	-	ST_Relate	-

Table 10: Topological relationships supported.

5.4 Geometric operations

The geometric operations or spatial analysis functions normally take one or two geometries and calculate a new geometry or value. This is the place where there is the biggest difference between the different DBMSs. Table 11 gives an overview of the OpenGIS operations. Oracle and Informix both implement much more than these functions; see Table 12. We refer to the documentation for more details.

OpenGIS	Oracle	Informix	Ingres
Distance	SDO_DISTANCE	ST_Distance	distance
ConvexHull	SDO_CONVEXHULL	ST_ConvexHull	-
Intersection	SDO_INTERSECTION	ST_Intersection	-
Union	SDO_UNION	ST_Union	-
Difference	SDO_DIFFERENCE	ST_Difference	-
SymDifference	SDO_XOR	ST_SymDifference	-

Table 11: Standard geometric operation (spatial analysis) supported.

Ingres	Area Length Perimeter anypoint bbox
Oracle9I	SDO_AREA SDO_BUFFER SDO_CENTROID SDO_CONVEXHULL SDO_LENGTH SDO_MAX_MBR_ORDINATE SDO_MBR SDO_MIN_MBR_ORDINATE SDO_POINTONSURFACE VALIDATE_GEOMETRY VALIDATE_LAYER WITHIN_DISTANCE
Informix	ST_Area ST_Boundary SE_BoundingBox ST_Buffer ST_Centroid ST_ConvexHull ...

Table 12: Additional geometric operations (spatial analysis) supported.

5.5 Spatial joins and aggregates

Spatial joins can be defined as: Find all pairs of spatial objects in two tables A and B where some spatial relationship between A and B holds. This means that any DBMS that support spatial relationships also supports spatial joins. However because these operations can work on collections of objects, performance of the operation becomes important. DBMSs support the acceleration of these operations using indices (R-tree, Quad-tree) and special functions.

Another class of operations returns an aggregate of a collection of geometries. An overview of the aggregate functions is given in Table 13 (Oracle only). Tough quite different as compared to the other aggregate functions, we classified the nearest neighbour (SDO_NN) as an aggregate function, because it has to have some kind of global knowledge of a table (comparable to the standard 'min' or 'max' aggregate functions for numerical values). The following example in Oracle returns the complete set of aggregated municipality boundaries by selecting and aggregating all parcel boundaries where the municipality to the left is different from the community to the right:

```
create table municipality_boundaries as
  select sdo_aggr_union(mdsys.sdoaggrtype(geo_polyline,0.1)) as geometry,
    r_municip,l_municip
    from lki_boundary
    where r_municip <> l_municip
    group by r_municip,l_municip;
```

Oracle
SDO_AGGR_CENTROID
SDO_AGGR_CONVEXHULL
SDO_AGGR_MBR
SDO_AGGR_UNION
SDO_NN

Table 13: Spatial aggregates supported.

Currently none of the DBMS extensions has active support for the management of topological structures (van Oosterom et al, 2002). Of course it is possible to store topological data but the constraints that enforce topological correctness cannot be expressed easily. Third parties have built extensions for Oracle9i that implement topology management (*Laser-Scan, 2002*).

6. CONCLUSIONS

In this research we successfully loaded huge amounts of spatial (vector) data into 'off the shelf' DBMS products (Informix, Ingres, Oracle) without encountering any real problems. Loading spatial data is still more elaborate than loading alphanumeric data. This is because the standard loading tools are sometimes not able to work with the spatial types, or the creation of indices needs extra handwork. In general the query results indicate scalable systems, and the response times are strongly related to the number of reported objects and hardly to the size of the overall data set. This indicates that spatial clustering and indexing are functioning well. During the experiments it was found that the way the data is clustered on disk is almost as important for response time as having the right index structures on the data. This remains true even if a table would fit in the core memory of the computer.

We can conclude that currently spatial DBMSs are sufficient for storage, retrieval and simple analysis of spatial data. At this moment the support does not include true 3D data types (though z values may be stored in both Oracle and Informix), geo-referenced rasters (images) and topological structure management. Also, real complex analyses remain in the domain of the dedicated GIS. Such a GIS should be able to operate on top of a spatially enabled DBMS. As stated, currently none of the DBMS extensions has active support for the management of topological structures.

In the future we plan to extend the experiments on the different DBMSs, e.g. IBM DB2 and Postgres and perhaps also on middleware based solutions, such as ArcSDE (on top of SQL server). Benchmarking on different hardware platforms is also a very interesting option; e.g. Windows or Linux. We also wish to look further at the implementation of complex functionality (such as clipping, buffering and map overlay), because this is where the DBMSs differ most. Finally, the development of a formal benchmark for spatial DBMSs is considered (*Stonebraker, et al 1993*).

Ingres has the least richness with respect to the functionality, but with the exception of long polyline queries has a very good performance. Informix has also a very good performance (including the long polyline queries) and is the only one compliant to the OpenGIS specifications. Oracle does have reasonable performance (including the long polyline queries in the latest version of Oracle) and is the richest with respect to the functionality. All in all it seems that spatial DBMSs are growing up, but it will still take some time before they are fully mature.

ACKNOWLEDGEMENTS

We would like to thank the three involved vendors of the geo-DBMSs, that is Oracle, CA (Ingres) and IBM (Informix) for their support during this research and specifically the persons involved in the actual activities: Floris Versteeg, Han Wammes and Siva Ravada of Oracle, Herman van der Wal of CA and Robert Uleman of IBM (Informix). We would also like to thank the Netherlands Kadaster (many persons involved, but to name just a few: Paul van der Molen, Caroline Groot and Peter Jansen) for making their data and example queries available for this research. Finally we would like to thank Sun Microsystems for making their hardware available for research purposes at a lower cost. Without their support it would not have been possible to obtain the reported results. Of course, we remain fully responsible for any error either in the benchmarks themselves or in the text of this paper summarizing the main results.

REFERENCES

ASK-OpenIngres (1994): INGRES/object management extention user's guide, release 6.5, Technical report, ASK-OpenIngres.

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

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

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

Laser-Scan (2002): Radius Topology – User's Guide and Reference.

Murray, C. (2001): Oracle Spatial User's Guide and Reference, Oracle Corporation, Redwood City, CA, USA. Release 9.0.1 Part No. A8805-01.

Van Oosterom, P., J. Stoter, W. Quak and S. Zlatanova (2002): The balance between geometry and topology, Advances in Spatial Data Handling, 10th International Symposium on Spatial Data Handling pp. 209-224.

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

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

Quak W., P. van Oosterom and T. Tijssen (2002): Testing current DBMS products with real spatial data, presented at 5th AGILE Conference on Geographic Information Science, Palma (Mallorca, Spain).

Samet, H. (1990): The Design and Analysis of Spatial Data Structures, Addison Wesley.

Stonebraker, M., J. Frew, K. Gardels and J. Meredith (1993): The Sequoia 2000 storage benchmark, in 19th ACM SIGMOD Conf. on the Management of Data, Washington, DC.

Tijssen, T, W. Quak and P. van Oosterom (2001): Spatial DBMS testing with data from the Cadastre and TNO-NITG, Technical report GISt No. 7, TU Delft, Faculty CiTG, Department of Geodesy.

CVs OF THE AUTHORS

Peter van Oosterom is a full professor and head of the 'GIS Technology' Section at the Delft University of Technology. His main research topics include: geo-databases, generalization, distributed GIS architectures, and Cadastral applications. He received his PhD degree from Leiden University based on the thesis entitled 'Reactive Data Structures for GIS'.
Wilko Quak is an assistant researcher of the 'GIS Technology' Section at the Delft University of Technology. His current main research topics are: geo-DBMS performance and modelling of spatial data.
Theo Tijssen is lecturer and researcher of the 'GIS Technology' Section at the Delft University of Technology. His current main research topics are: geo-DBMS implementation and optimization, geo-DBMS and GIS, data exchange.

CO-ORDINATES

Prof.dr.ir.Peter van Oosterom
Drs Wilko Quak
Drs Theo Tijssen
Delft University of Technology
Faculty of Civil Engineering and
Geosciences
Department of Geodesy
Section GIS-technology
Thijsseweg 11
2629 JA Delft
The Netherlands
Tel. +31 15 278 6950
+31 15 278 3756
+31 15 278 3670
Fax +31 15 278 2745
E-mail oosterom@geo.tudelft.nl
quak@geo.tudelft.nl
tijssen@geo.tudelft.nl
Website www.gdmc.nl