

# Open boundaries in GIS

**OpenGIS and the SDO and SDE implementation**

Godefridus C. Cattenstart

Dissertation submitted in part fulfilment of requirements for the degree of

Master of Science  
in  
Geographical Information Systems

Presented:    may 1998

Mentors:	Dr. D.I. Heywood	Manchester Metropolitan University
	Prof. Dr. H.J. Scholten	Free University of Amsterdam

## Abstract

In 1994 the Open GIS Consortium (OGC) was launched with the mission of improving the interoperability of GIS. At the heart of this mission was a need to find a common standard for the structuring of topological features, among which the structuring in a SQL environment. This common standard has evolved in to OGC "Simple Features Specification For SQL". However, from a practical point of view this specification is only of value once it is embodied within systems that are used in practice. In 1997 two major commercial packages came on the market. ORACLE Corporation Spatial Data Option (SDO) and Environmental Systems Research Institute Spatial Data Engine (SDE). Both with the intention of using the OGC standards as the basic building block for structuring vector databases.

The main aim of this thesis was to evaluate how well SDO and SDE apply the OGC specification and furthermore establish whether or not the practical application of these tools and standards meets the needs of users for the topological structuring of geographic data. To do this a case study data set from the Dienst Landelijk Gebied in the Netherlands was implemented in both SDO and SDE. The next set of four major aspects were found.

Simple and non simple geometry's can be incorporated in SDO and SDE. However, not all OGC simple geometry's are supported and more importantly many non simple topological structures are not accommodated at all. OGC is recommended to address rules on conversion of non simple to simple geometry's.

SDO and SDE differ greatly in their set of geometry validation rules and the way they interpret the OGC specification. This means that both systems will react differently when data are inserted. Inserting the same set of data in both packages might result in different implementations in both packages. From a user point of view this is a most unwanted situation. Standards on converting non simple to simple geometry's is of great importance.

The storage architecture and physical data structure of SDO, in numeric format, and SDE, in binary format, is in accordance with the OGC specification. Storage format in its self is not that much a users concern. However, in order to compose efficient retrieval statements the user must have knowledge about the implemented data structure of geometry's and the spatial indexing mechanism.

Retrieval of data in SDO and SDE is highly bound to the concepts of the developers. In SDO this is a full SQL integrated environment. In SDE it is a defined Application Program interface. Retrieval is an aspect OGC is recommended to address.

From a users point the positive message is that with proper care, spatial data, can be incorporated, managed and set available in both packages, following the rules of OpenGIS. Still there is much work to be done before the OGC specification and its implementation in products like SDO and SDE will provide GIS products which are truly interoperable in character.

# Contents

<b>Prologue.....</b>	<b>1</b>
<b>1. Problem definition .....</b>	<b>2</b>
1.1 The context.....	2
1.1.1 A wish to integrate spatial and attribute data .....	2
1.1.2 Storage characteristics of data in relational databases .....	7
1.1.3 Functional characteristics of SQL.....	8
1.1.4 Why this research.....	9
1.2 The problem .....	9
1.3 Document overview .....	11
1.4 Research extent and methodology .....	12
1.5 Benefits of RDBMS to the GIS community .....	14
1.6 Contribution of this study to the GIS community .....	15
1.7 Summary .....	16
<b>2. The ideal of OpenGIS: the OGC concept.....</b>	<b>17</b>
2.1 OGC strategy .....	17
2.1.1 The application developers benefit.....	17
2.1.2 The users benefit .....	18
2.2 The theory of OpenGIS.....	18
2.3 The Open Geodata Model .....	21
2.3.1 The SQL92 Representation Model .....	25
2.3.2 The SQL92 Implementation Model Data Architecture .....	27
2.3.3 The SQL92 Implementation Model Storage Architecture.....	32
2.3.4 The SQL92 Relational Operators.....	34
2.3.5 The SQL92 Retrieval Model.....	37
2.4 Discussion .....	39
2.4.1 The OGC collaboration .....	39
2.4.2 The SQL92 representation Model .....	39
2.4.3 The SQL92 Implementation Model Data Architecture .....	40
2.4.4 The SQL92 Implementation Model Storage Architecture.....	41
2.4.5 The SQL92 Relational operators.....	44
2.4.6 The SQL92 Retrieval Model.....	46
2.5 Summary .....	47
<b>3. The OGC concept in SDO and SDE .....</b>	<b>48</b>
3.1 SDO .....	48
3.1.1 Data Model .....	49
3.1.2 The Database Storage Structure .....	50
3.1.3 Spatial index .....	53
3.1.4 Data manipulation.....	54
3.1.5 Data retrieval .....	55
3.1.5.1 Spatial operators.....	55
3.1.5.2 The Query Model concept.....	57
3.1.5.3 The Query Model implementation .....	59
3.2 SDE.....	60
3.2.1 Data Model .....	61
3.2.2 The Database Storage Structure .....	63
3.2.3 Spatial index .....	64
3.2.4 Data manipulation.....	66
3.2.5 Data retrieval .....	66
3.2.5.1 Spatial operators.....	66
3.2.5.2 The Query Model concept.....	68
3.2.5.3 The Query Model implementation .....	69

3.3 Discussion .....	71
3.3.1 OpenGIS and the SDO implementation .....	71
3.3.2 OpenGIS and the SDE implementation .....	74
3.3.3 SDO and SDE compared .....	77
3.4 Summary .....	80
<b>4. Implementing geodata in SDO and SDE: a case study .....</b>	<b>82</b>
4.1 Geometry morphology's from the Dutch National Topographic Map .....	82
4.2 Discussion .....	88
4.3 Summary .....	91
<b>5. Conclusions .....</b>	<b>92</b>
5.1 OGC .....	92
5.1.1 Geodata Model .....	93
5.1.2 OGC storage Architecture .....	94
5.1.3 OGC spatial operations and retrieval .....	95
5.2 OpenGIS in practise .....	96
5.2.1 SDO .....	96
5.2.2 SDE .....	98
5.3 Future developments .....	100
5.3.1 OpenGIS .....	100
5.3.2 SDO .....	101
5.3.3 SDE .....	103
5.4 Summary .....	104
<b>Epilogue .....</b>	<b>106</b>
<b>References .....</b>	<b>107</b>
<b>Appendix A OGC geometry types.....</b>	<b>A 1</b>
<b>Appendix B Storage in the SQL implementation Data Architecture.....</b>	<b>B 1</b>
<b>Appendix C SDO and SDE geometry validation rules.....</b>	<b>C 1</b>
<b>Appendix D Example of data storage in SDO and SDE.....</b>	<b>D 1</b>
<b>Glossary.....</b>	<b>E 1</b>

## List of Tables

Table 1: Comparison of spatial operators between the OGC concept and the SDO implementation. ....	73
Table 2: Comparison of spatial operators between the OGC concept and the SDE implementation. ....	76
Table 3: Data Model.....	77
Table 4: Storage structure.....	78
Table 5: Spatial indexing.....	78
Table 6: Data Retrieval .....	79
Table 7: Special geometry morphology's and there validation in OGC, SDO and SDE. ....	85
Table 8: Acceptance of Line type geometry's.....	89
Table 9: Acceptance of Polygon type geometry's.....	89

## List of Figures

Figure 1: Structure of this document.....	12
Figure 2: Relation between OpenGIS, developers and users. ....	13
Figure 3: Elements of the OpenGIS concept. ....	19
Figure 4: The layered model of geographic element representation.....	22
Figure 5: Relation between location and geometry. ....	23
Figure 6: Feature building block hierachy.....	26
Figure 7: Types of LineStrings.....	29
Figure 8: Multi building block geometry's.....	30
Figure 9: Examples of polygons. ....	30
Figure 10: Examples of objects that are not representable as a single instance of a polygon. ...	31
Figure 11: Examples of valid Multipolygons. ....	32
Figure 12: Simple feature table storage schema. ....	33
Figure 13: A Schema on the structure of normalised tables holding vector data. ....	42
Figure 14: SDO architecture .....	49
Figure 15: The SDO two tier retrieval model. ....	58
Figure 16: SDE architecture .....	61
Figure 17: The SDE two tier/two request retrieval model.....	69
Figure 18: Possible developement of the ESRI product line.....	103

## **Disclaimer**

The results presented in this thesis are based on my own research in the Department of Regional Economics of the Free University of Amsterdam the Netherlands in co-operation with the Department of Environmental and Geographical Sciences of The Manchester Metropolitan University England and the Dienst Landelijk Gebied the Netherlands. All assistance received from other individuals and organisations has been acknowledged and full reference is made to all published and unpublished sources used.

This Thesis has not been submitted previously for a degree at any Institution.

Nieuwegein, may 1998,

Frits Cattenstart.

## Acknowledgements

I would like to express my gratitude to the Amsterdam UNIGIS team. Thanks to Bart Kusse who has read *The Tao of Pooh*. Thanks to Arienne Mahieu for her secretary work. And Many thanks to Mathilde Molendijk. Her sense for human interaction and understanding I greatly appreciate. Special thanks to Ian Heywood, who guided me through the realms of science, and for being my patient mentor. Through his teaching I experienced learning to be great fun. And at the team was Henk Scholten. He took the “risk” to supervise me. His enthusiasm and energy are a continuous source of inspiration. But most of all, he made me more conscious that learning is in every day’s practise, in every subject. Thank you Henk.

Thanks to Han Wammes from ORACLE Corporation for his SDO implementation support and constructive remarks on the theme of this thesis.

Thanks to Jeroen van Winden from Logisterion BV for his SDE implementation and programming support.

Thanks to my colleague Jeroen Baltussen for being my discussion partner.

Thanks to my superior Pieter Arends for providing resources at the Service to make this study possible.

During my UNIGIS study I met a colleague who was at the Service on a temporary assignment. It is from her that I learned the relativity of science. It has greatly influenced my study and the way I approached it. Thank you Janien.

Marianne is not familiar with the intestines of computers and programs. But her influence on my study is evident. She was the one who was involved and supported the decisions upon spending time between family, work, study and relaxation. Let love reside between us.

Frits.

## Prologue

Captured in the blizzard, surrounded by a wall of downward forced white, we were alone, pressed against the rocks and only had each other, just man and woman, and our thoughts...

*Love reveals itself in affection.  
It is this affection that makes the exchange of both souls and  
the unification of both bodies the dearest gift ever to exist.  
This makes Love the most mystic, deepest and elementary  
secret Mother Earth preserves; the attraction and  
devotion of individual creatures towards each other.  
Of all phenomena known there is no one more beautiful  
and with such charisma and magic.  
It remains uncaptured.*

... at a place filled with fire and ice, impossible to retrieve again.



# 1. Problem definition

## 1.1 The context

### 1.1.1 A wish to integrate spatial and attribute data

At the Dienst Landelijk Gebied (Government Service for Land and Water Management) in the Netherlands topographic data are held in ARC/INFO coverage's composing the digital Dutch National Topographic Map (TOP10vector) scale 1 : 10,000 (Cattenstart, 1992). An example of the map is given on page 4. The geometric elements are related to attribute data in ORACLE databases. This dual storage environment exists because, until recently, in the Geographic Information Systems (GIS) world there were no efforts by developers and users to integrate data and parts of software.

Tele-communication of data of different kinds of formats, resources and origin in the GIS world is still cumbersome. But there is the wish to incorporate the geometric and attribute data into a Data Base Management System (DBMS). Now what?

It is a fact that in the last two decades, Database Management Systems (DBMS) have grown to industrial strength. This is due to many issues but there are three elements to be mentioned here that are of great importance.

1. The alphanumeric world managed to create **and** implement just one concept for converting real world features and phenomena into computer held data. This makes access and exchange of these data across computer platforms and software systems easy. It is the concept of Normalised tables based on the "set-theory".
2. The alphanumeric world managed to create **and** implement just one uniform declarative interface language to retrieve and operate on the stored data. A declarative language can be

supported by many types of physical computer and operating system depended implementations but the user only has to be familiar with the language itself. The user only has to formulate **what** to do and not **how** to do it. It is the Structured Query Language (SQL) interface that is used world wide.

3. The alphanumeric world managed to create and implement extremely fast indexing techniques for swift retrieval of parts of data out of voluminous data sets.

The GIS world is still struggling with these aspects for the last two decades. It has not developed a uniform implementation of world spatial phenomena in computer held data, nor a uniform exchangeable method of operating on and retrieval of the data.

From a developers point of view there are three main reasons why storage of spatial data differs from software package to software package:

1. Due to the focus on the research of efficient spatial algorithms storage of data was performed in such a physical format that suited best in the scheme of the type of spatial analysis concept.
2. The data sets them selves are extremely large which forced software developers to implement data formats best suited for the indexing techniques available.
3. Commercially, data format and operation routines offer a great opportunity to distinguish one software package from the other. The cost of research en development had (and is) to be protected for commercial reasons, making users more or less dependent towards a specific software package.

# TOP10vector topographic data



500 0 500 1000 1500 Meters

## Legend

- Symb
- △ Symb
- House
- Polygons
- House
- Block of houses
- Large building
- Flat building
- Wall
- Barn
- Greenhouse
- Store
- Pole
- Shooting range
- Fwyway
- Highway
- Highway
- Highway
- Highway
- Freeway
- Highway for mixed traffic, width between 4 and 7 m
- Highway for mixed traffic, width between 4 and 7 m
- Highway for mixed traffic, width smaller than 4 m
- Car road, separated lanes
- Car road, separated lanes
- Car road, separated lanes
- Car road, width greater than 7 m
- Road, separated lanes
- Road, width greater than 7 m'
- Road, width greater than 7 m'
- Road, width between 4 and 7 m'
- Road, width between 4 and 7 m'
- Road, width smaller than 4 m'
- Road, width smaller than 4 m'
- Other roads
- Semi paved road
- Unpaved road
- Passage
- Promenade
- Street
- cycle track
- Broad-leaved wood
- Coniferous wood
- Mixed leaved wood
- Reed land
- Poplar
- Arable land
- Meadow
- Orchard
- Tree cultivation
- Moor
- Sand
- Other land use
- Cemetery
- Fruit cultivation
- water, sea
- water, shoreline
- Tide land
- Rock
- Landing stage
- Dock



The three reasons have led to the existence of many different spatial data formats. Compatibility was mainly resolved by importing routines which converted foreign spatial data formats physically and conceptually to the format and concepts of the used package. Until the early 1990's, the management of storage of data had not much attention. GIS are primarily a toolbox of efficiently implemented operations rather than management of data ( Wildschut, 1998).

From a users point of view there are two main reasons why different types of data storage is inconvenient:

1. It is a fact that in current practice computer data exchange requires major efforts. These efforts are marked unnecessary.
2. The costs of “field collecting” and “update” of data is extremely time and money consuming. Making data useful for other research in any type of software package on any computer platform can reduce cost due to the ability of sharing data.

These reasons in part triggered the concept of “open” systems. This concept also applies to the field of GIS, where friendly exchange or shared use of data and program functionality arose based upon a scientific ideal of smooth incorporation of data and GIS-functionality in all kinds of projects at any location (Wildschut, 1998).

The idea of storing spatial data into a DBMS seems simple. But the structure of data stored in a GIS differs greatly from that of an “ordinary” DBMS. The first efforts are made. In 1997 two major packages are available. Environmental Systems Research Institute (ESRI) promoted Spatial Data Engine (SDE). ORACLE Corporation supported Spatial Data Option (SDO). Both packages store spatial vector data in a Relational DBMS (RDBMS). These packages can

hold and manage spatial data within the RDBMS environment. They are not based on the object oriented concept, but on holding elementary data in columns. In concept and implementation the packages differ a great deal. So, which one is the most suitable?

The Open GIS Consortium (OGC) was founded in 1994 and envisions the full integration of geospatial data and geoprocessing resources into mainstream computing and the widespread use of interoperable, commercial geoprocessing software throughout the information infrastructure. OGC involves developers and users to collaborate in the development of interoperable geoprocessing technology specifications, and work to promote the delivery of certifiable interoperable products (NN, 1996). For this purpose OGC has, amongst others, defined several concepts for data storage, spatial operations and retrieval. The question now is: which of the two packages best meets the criteria OGC has set?

It must be odd to observe that since the foundation of OGC in 1994 two packages of such different nature, dealing with the same matter, are brought on the market in 1997. From OGC work it can be concluded that there are different sets of concepts for storage, operations and retrieval. Why do these differences in the OGC specifications exist? It leaves the user community behind with an uncertain favour of one package over the other. And thus is the GIS community still searching in a diverged way for direction instead of converging to a more uniform approach.

What is it that makes spatial data in RDBMS's so special? Has it some relation with storage and retrieval?

### **1.1.2 Storage characteristics of data in relational databases**

Relational databases are designed upon the concepts of “set theory” mathematics. Every set represents a collection of entities which all have (a) common characteristic(s) that place them into that set. Attribute types are related to the entity type(s) that make up the set and for every entity type they are not related to each other.

Normalised, every set (table) in the database represents an entity type as a collection of independent entities (rows) which all have a finite set of attribute types (columns) which all apply to each entity in the set. This is why there is no necessary order in rows and columns.

In the alphanumeric environment the mentioned aspects of the set theory are fully implemented. In the spatial environment, however, entities do are related to each other since they have location towards each other and the geometric attributes that make up the morphology are related to each other as a sequence determining the shape of the geometry. More over, spatial entities have a temporal relationship.

Searching for entities in the alpha numeric environment (table) requires a one dimensional search algorithm which acts upon the rows in the table. The search for spatial entities in a N-dimensional space environment requires a N-dimensional search algorithm that can be related to a one dimensional algorithm (back and forth) since a table that holds the entities as rows has to be searched.

Alpha numeric entities of the same type are supposed to be independent of each other. They do not require a set of functions that determine there relationships. Spatial entities of a certain type are related and require such a set of functions in order to reveal meaningful information from the data set as a whole.

### 1.1.3 Functional characteristics of SQL

Within the context of the “set theory” and normalised tables SQL has proved to be a powerful declarative user interface. The next set of characteristics apply to this interface:

1. It has a consistent syntax, unambiguous semantics and uniform appearance in all functions it provides,
2. Input is command line oriented,
3. Output is generated as a set of values in a table,
4. It is not capable of returning Boolean values as a result,
5. It has no extension for dealing with spatial information,
6. It has no graphical input or output facility's,
7. It is not capable of referring to an object undergoing temporal changes.

From these characteristics Egenhofer (1992) concludes that SQL is not a straight forward solution to the integration and retrieval of spatial data in RDBMS's. He also states that SQL is not capable of retrieving sets of potential valid values from a partial set. In it self this is true but this is not directly related to SQL. In general it is not possible to derive the exact potential valid set from a partial set of values. Further he states that the development of a spatial extended SQL interface is diverse and each package has its own implementation. This, now, is the realm of OpenGIS and OGC. Early “set theory” based retrieval interfaces, like the ARC/INFO ARCPLOT module RESELECT command, have proven that a command line interface which combines attribute and spatial data, alphanumeric and graphical input, table and 2D-visualisation functionality can be a powerful tool.

Land (1997) considers that leaving the SQL interface, although spatial data are stored in the RDBMS, will lead to an undesired divergent search for other solutions. This should indeed be avoided.

#### **1.1.4 Why this research**

In order to integrate spatial and attribute data into a RDBMS a research has to be performed on the practical aspects of the different OGC concepts and the implementation of the OGC concepts in the SDO and SDE packages. In this way the GIS community is helped to distinguish one concept from the other, to assess the practical aspects of the implementation of a concept and support the search for a uniform approach of capturing and retrieving real world spatial phenomena in computer held data.

On that bases TOP10vector topographic data can be integrated into the attribute holding RDBMS's.

### **1.2 The problem**

ESRI SDE and ORACLE Corporation SDO are two products that intend to make a step towards OpenGIS. Data is stored in a RDBMS. Retrieval is performed by SQL, an Application Programming Interface (API), Object Linking and Embedding (OLE), or an Open DataBase Connection (ODBC). Each package is based on a different concept (to be discussed in the next chapter) of the OGC specifications. As there is the wish to store spatial data into a RDBMS it raises the question:



Does the implementation of OpenGIS concepts according to OGC in SDE and SDO limit the practical use of geodata sets?

The question incorporates a broad spectrum of fields of interest. There are the technical aspects, the developers view and the users view. In this study the technical aspects are emphasised. But since even the technical aspects are numerous three fields are discussed:

1. The feasibility of OGC defined morphologies of spatial geometry's,
2. An uniform data storage environment,
3. An uniform language interface.

In practice this would lead to the next set of actual questions:

1. What are the differences between the OGC concepts implementations in SDE and SDO:
  - 1.a. Which OGC geometry's are defined and (not) accepted in SDE and SDO,
  - 1.b. Which OGC storage architecture is implemented in SDE and SDO,
  - 1.c. Which OGC Spatial operations are defined and implemented in SDE and SDO,
  - 1.d. What OGC data retrieval interface is defined and implemented in SDE and SDO,
2. What are the practical implications of spatial data in SDE and SDO.

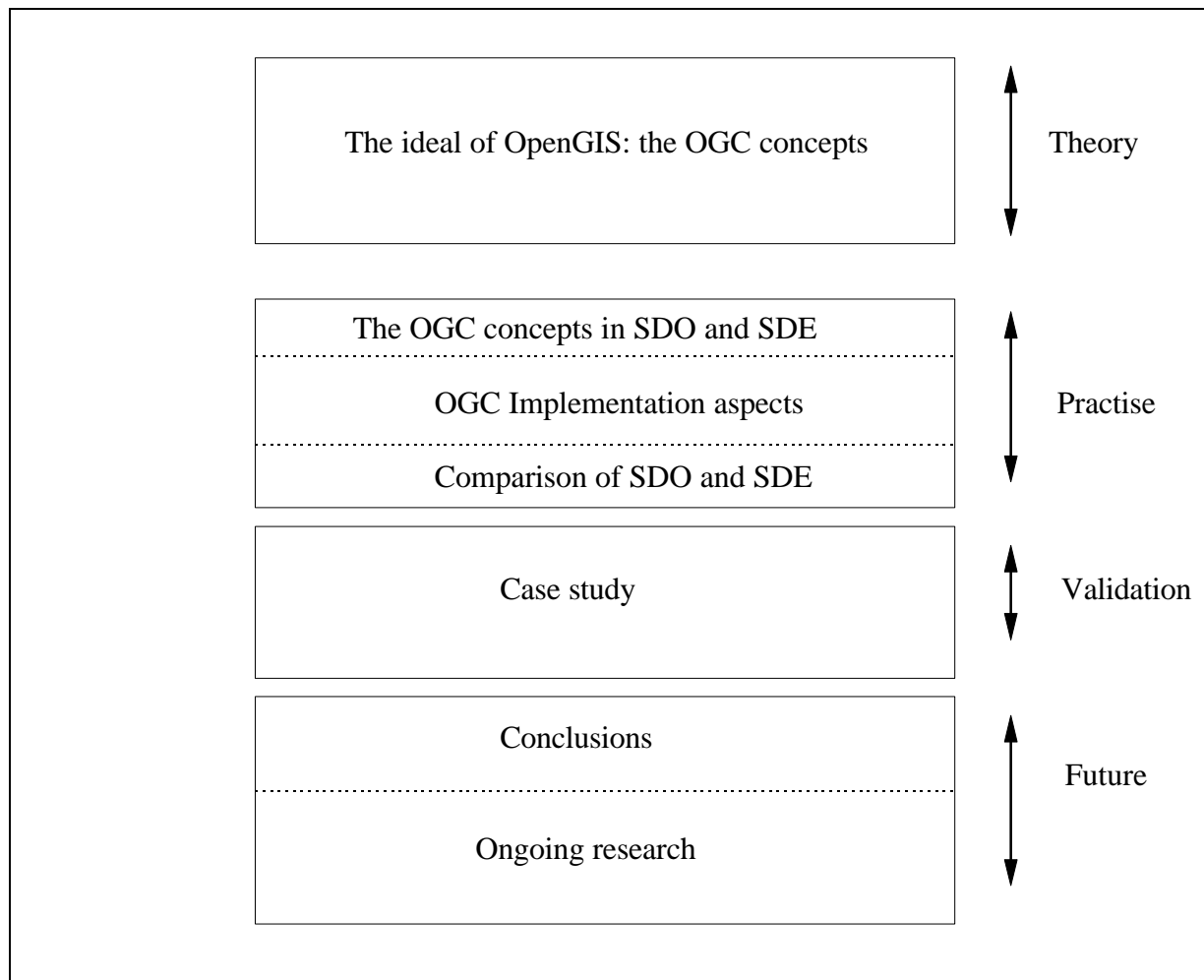
The first set of questions asks for a comparison of OGC concepts and the extent to which these concepts are implemented in SDE and SDO. It also involves to what extent these packages are a superset or subset of the OGC specifications. The second question refers to an actual case in which spatial data are to be incorporated into both packages.

In order to determine the morphology and characteristics of geometry's, as they emerge in every day practise, a representative geodata set of large volume is visited. In this case a dataset which is intensively used by the Dutch GIS community is the Dutch National Topographic Map (TOP10vector) scale 1 : 10,000. The dataset occupies 7 Gbyte of disk space and contains over 15 million polygons, 75 million separate lines and 1.5 million points.

### **1.3 Document overview**

In chapter 1 the problem is more specifically defined. Chapter 2 illuminates the concepts of "OpenGIS". It discusses how common use and sharing of data can be implemented and how these data might be retrieved in an uniform way. Chapter 3 deals with the implementation of the OpenGIS concepts in the two mentioned commercial packages SDE and SDO. Chapter 4 gives a practical implementation of topographical data into these packages and discusses the relation of this practical work towards the scientific concepts. Chapter 5 holds conclusions and future research. The structure of this document can also be depicted as shown in Figure 1.

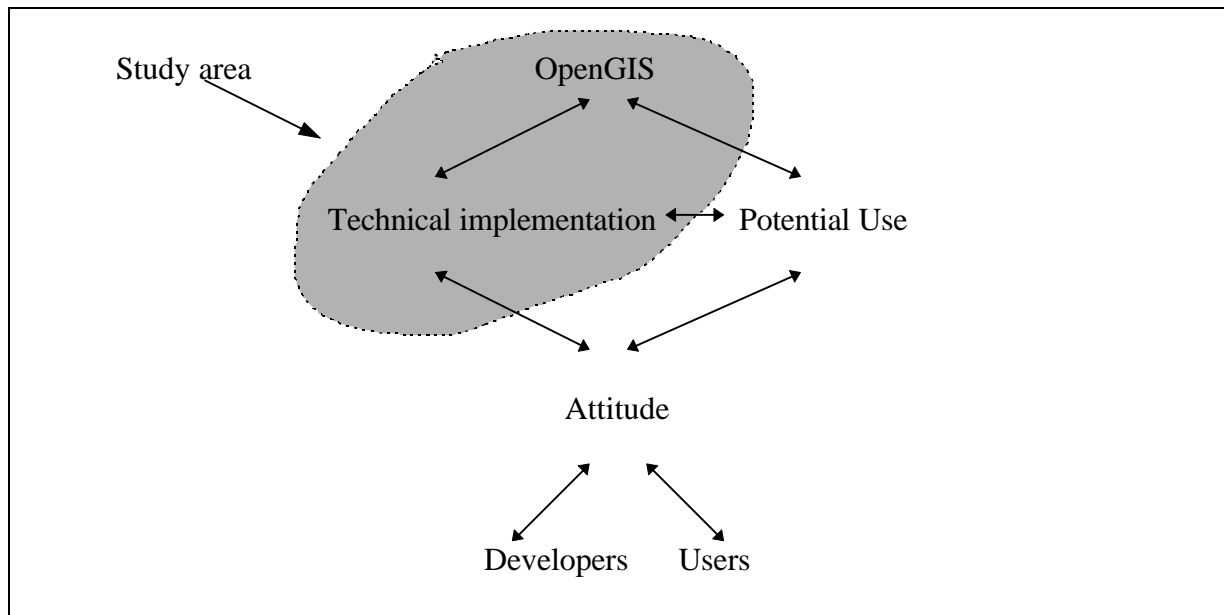
Figure 1: Structure of this document



## 1.4 Research extent and methodology

Implementation of OGC definitions has implications regarding the development and application of OpenGIS. The technical implementation and its potential use determine the attitude of both developers and users. Figure 2 depicts the relation between developers, users and OpenGIS.

Figure 2: Relation between OpenGIS, developers and users.



This study focuses directly upon the technical implementation.

The research is conducted using a linear method. The method consists out of the next set of steps:

1. Perform a research on the OGC different concepts towards geometry type definitions, operations and retrieval.
2. Perform a research on the SDO and SDE OpenGIS implementation concepts.
3. Implement typical geometric data into SDO and SDE and determine to what extent SDO and SDE are a superset or subset of OGC concepts implementation.
4. Evaluate SDE and SDO against OGC.

## **1.5 Benefits of RDBMS to the GIS community**

Computing of complex operations, ease of use and avoiding confusion are major aspects of computer aided processing. OpenGIS is focused on these aspects. RDBMS in conjunction with SQL are world wide accepted as standards in data storage and retrieval. If spatial data can be stored in RDBMS and retrieved using SQL or SQL imbedded Application Program Interface's (API) a uniform approach of spatial and alphanumeric data can be gained. Both developers and users gain benefits.

The next set of benefits to the GIS developers can be mentioned. A RDBMS provides:

1. Storage of large volumes of data,
2. Highly efficient one dimensional indexing to minimise response time,
3. Database networking such as “distributed databases” and “cloning”,
4. Strong relationship between spatial and attribute data,
5. Data integrity,
6. User roles upon permitted database actions,
7. Rollback mechanisms,
8. Multi user simultaneous environment,
9. Data transaction locking.

In fact, the GIS developers can concentrate on the skills they are best in, namely spatial operations on sets of data.

The next set of benefits to the GIS users can be mentioned. A RDBMS provides:

1. A familiar environment,
2. A uniform interface,
3. Security of labour intensive data manipulation,
4. Availability of data on other connected resources,
5. Data integrity,
6. Simultaneous use and sharing of data.

## **1.6 Contribution of this study to the GIS community**

The comparison of the GIS implementation of SDO and SDE against the OGC concepts gives an opportunity to reflect the “state of the art” of OpenGIS implementation. The study may provide elements to the mental frameworks that help in constructing OpenGIS as it arises from every day practise. In this context the issues discussed in this study induce elements of answers to the next questions:

1. What is the current status of a more uniform approach in GIS interoperability,
2. What is the current status of parts of the OpenGIS concepts implementations in commercial packages,
3. To what extent does the current OpenGIS implementation in SDO and SDE support and limit the practical use of GIS,
4. “Where to go from here?”.

Although the study is highly temporal, since GIS packages evolve rapidly, it should provide a moment of consideration. Bundling forces available, so they all drive in the same direction, is highly important in a complex environment of scientific, user and commercial interests.

## **1.7 Summary**

At the Dienst Landelijk Gebied (Government Service for Land and Water Management) in the Netherlands there is the wish to integrate topographic (spatial) data with attribute data into a RDBMS. In this environment spatial decision support systems can take advantage of all facility's RDBMS can offer, like data integrity and authorisation, as well as a strong and controlled link between geometric and attribute data. In 1997 two major commercial packages came on the market. ORACLE Spatial Data Option (SDO) and ESRI Spatial Data Engine (SDE). But which one is most suited? Using standards and rules of OpenGIS as a reference SDO and SDE can be evaluated against this reference.

In order to become familiar with OpenGIS and perform such evaluation the next chapter contains the concepts and implementation rules of OpenGIS as defined by the Open GIS Consortium.

## **2. The ideal of OpenGIS: the OGC concept**

The OGC concepts are subject to constant change en evolution. In this chapter the OGC state of the art as published in several documents from 1996 until March 1998 is described, referenced and discussed.

### **2.1 OGC strategy**

The OpenGIS Consortium (OGC) was founded in 1994 and envisions the full integration of geospatial data and geoprocessing resources into mainstream computing and the widespread use of interoperable, commercial geoprocessing software throughout the information infrastructure. OGC involves developers and users to collaborate in the development of interoperable geoprocessing technology specifications, and work to promote the delivery of certifiably interoperable products (NN, 1996).

The OGC makes efforts in order to provide a scientific mental framework of aspects of OpenGIS and generate guidelines towards implementations of OpenGIS. The realm of OGC is to enable applications to interact with other applications that manage, deliver and process geospatial data. It addresses how to provide a service and how to request a service.

#### **2.1.1 The application developers benefit**

Application developers are extremely helped by standards. Using standards they can create program routines for the task at hand making use of other existing routines.



The application developer can more easily and more flexibly:

1. Write software to access geodata,
2. Write software to access geoprocessing resources,
3. Tailor applications to specific user needs,
4. Reuse geoprocessing code,
5. Be more computer platform independent.

### **2.1.2 The users benefit**

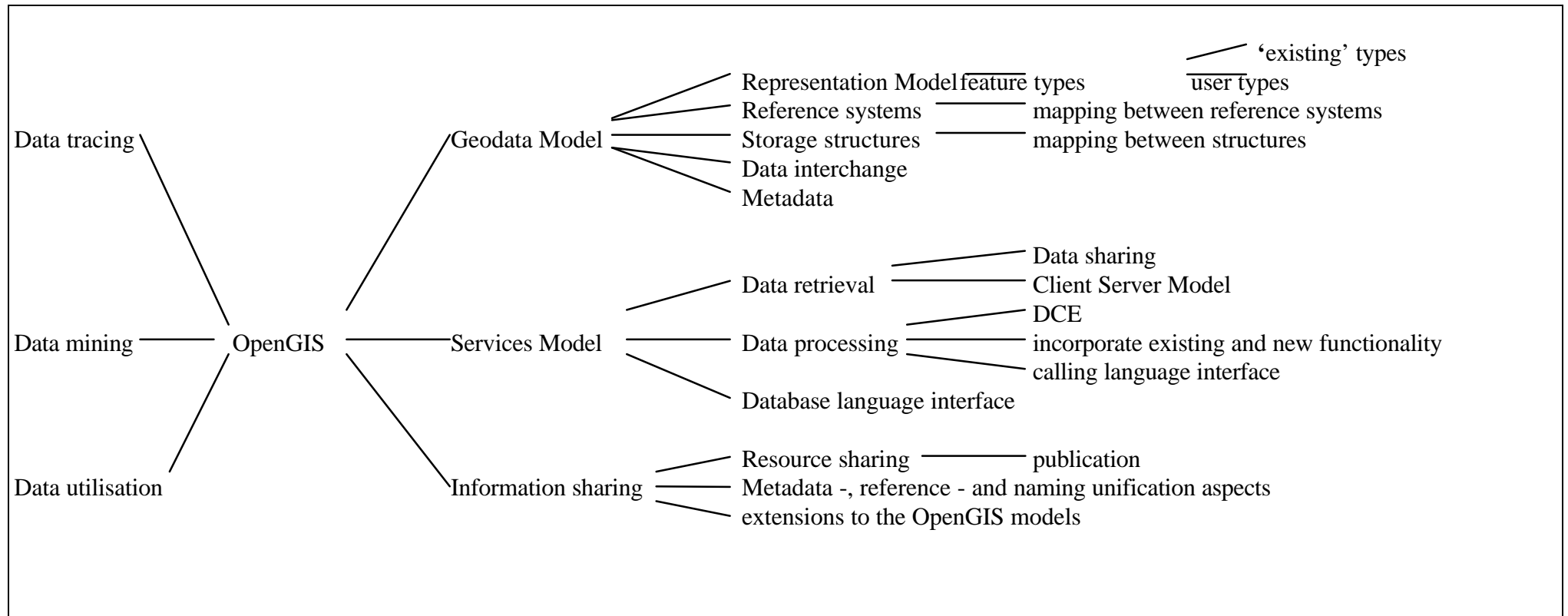
The users should be the ultimate beneficiaries, receiving:

1. Real-time access to universal located geographic data,
2. A platform independent availability of sets of applications,
3. The ability to work with different types of geodata and formats within a single application environment,
4. Spatial information and operations are more easily accessible for those who are already involved and lower the threshold for those who undertake the first steps in automated spatial information supported decision making.

## **2.2 The theory of OpenGIS**

“OpenGIS” can be defined as “the interoperability of geospatial data and geoprocessing resources”. In practise it means that data resources and the calling of process functionality can be shared between different (parts) of organisations and software packages that process spatial data. Figure 3 envisions aspects of OpenGIS as discussed in this chapter.

Figure 3: Elements of the OpenGIS concept.



When starting a computer aided decision project data are required to gain information relevant to the project. In analytical projects it means that data must be found (tracing) and evaluated (mining) regarding suitability for the task at hand. Then suitable data can be used (utilisation). In management projects, which involve continuous available databases, data are to be built up and shared between different sets of procedures. Shared use of actual data is essential in these type of projects.

“Open systems” involves activity from the developers who implement software and structures that facilitate this concept. It also involves activity from the users who operate in this environment.

Using common data involves aspects of data storage, data processing and data sharing. Data must be found, evaluated and finally used. In order to make data accessible to others three major concepts are to be considered. According to OGC they are, the Geodata Model, the Services Model and the Information Sharing Model (NN, 1996).

The Geodata Model is a mental framework in which the representation of spatial phenomena is conceptually defined. The model provides a means to store our world's view into computer held data.

The Service Model is a mental framework to access, manage and manipulate data between systems and projects. In this study specially the spatial operations are considered.

The Information sharing Model is a mental framework that provides all aspects necessary to make the “open” system work properly. Among others it involves concepts for proper data use by setting metadata available, concepts for reachability of computers and data systems and

agreements upon metadata descriptions, naming conventions etc.. The Information Sharing Model will not be discussed here in more detail.

## **2.3 The Open Geodata Model**

The Open Geodata Model is an extensible information model comprised out of geodata types which are coded abstractions of real-world space and time phenomena. The Geodata Model incorporates the following main objectives:

1. The model is independent regarding programming language, hard- and network representations,
2. Provide conceptual mechanisms for describing spatial reference systems and mapping,
3. Provide concepts for data storage structures and structure mapping,
4. Provide conceptual mechanisms for data interchange,
5. Provide concepts for meta information,
6. Incorporation of new techniques.

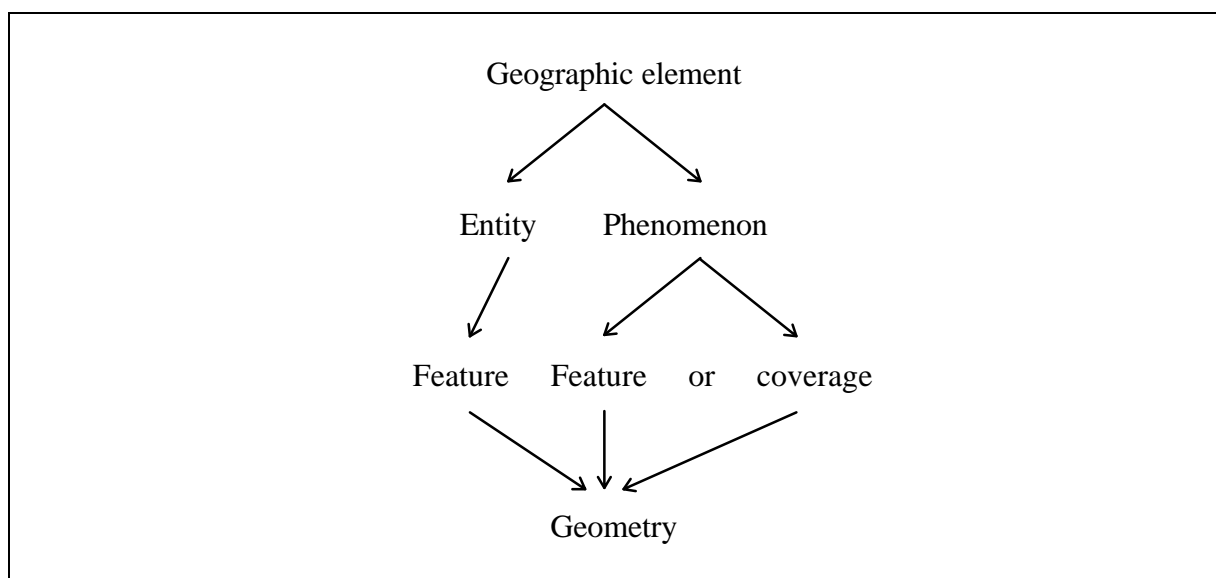
The Open Geodata Model is composed out of (NN,1996):

1. The Representation Model,
2. The Reference System,
3. The Storage Structure,
4. Data interchange Model,
5. Metadata Structure.

From the Open Geodata Model the Representation Model and Storage Structure are subject of this study. In specific it only deals with vector represented geographic elements. Other representations, like raster format, are not yet defined by OGC.

Geographic elements must be represented in some way that they can be stored in computer data structures. OGC has defined a model for the vector representation that leads to geometric elements that can be stored. Figure 4 envisions this model and is further discussed.

Figure 4: The layered model of geographic element representation.



Geographic elements can be categorised into two broad realms: entities and phenomena. Entities are recognisable discrete objects that have relatively well-defined boundaries or spatial extent. Phenomena vary more or less continuously over space and time and have no specific extent (Cook and Daniels, 1994). An Example of a phenomenon is temperature.

In addition to entities and phenomena another central aspect of geographic data modelling is geopositioning. Entities and phenomena are not meaningful in geoprocessing unless they are expressed in terms of a model which positions them in space and time. A *geometry* represents location of place and time. All entities and phenomena have location attached as part of their description in the Open Geodata Model, in addition to attributes that are not locational. Figure 5 envisions this concept (NN, 1996).

Figure 5: Relation between location and geometry.

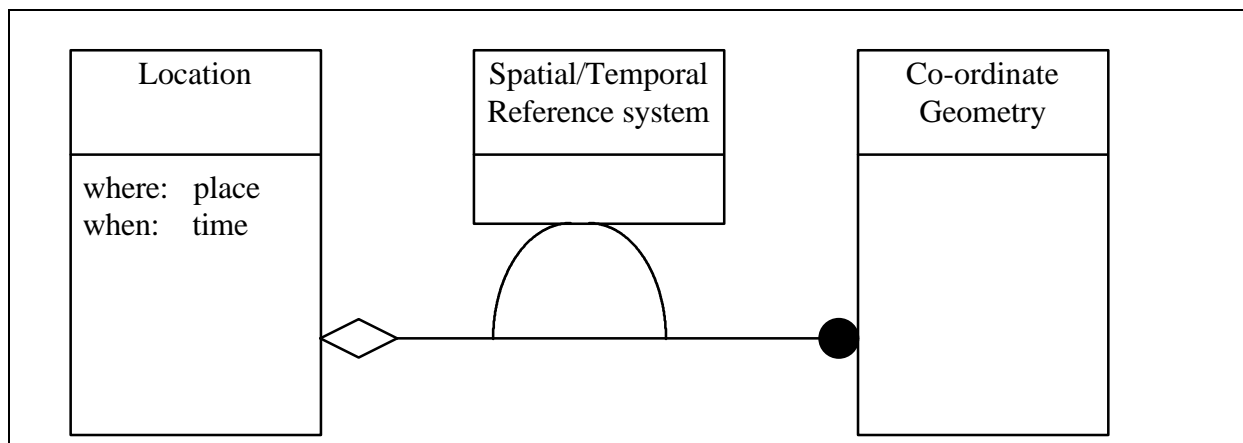


Figure 5 is an essential model object diagram ( a *type view* ) in the OGC concept that depicts a relationship between a location and the co-ordinate geometry used to represent the locationally important aspects of the entity or phenomenon. Each rectangle in a type view represents an object type and lines between rectangles represent associations between object types. The name of the object type appears at the top of the rectangle, separated from the rest of the contents by a horizontal line. The remainder of the rectangle holds value-typed properties ( integer, date, number, size, etc.) of the object type. The small black circle indicates a set, which means that a Location can have more than one Co-ordinate Geometry. The half-oval symbol indicates that the Spatial Temporal Reference system is an association property of the

association. The small diamond symbol is used to depict aggregation, which here describes the relationship between a location and a co-ordinate geometry. Aggregation as defined by Cook and Daniels (1994), means dependence of existence. In this type view, the diamond symbol expresses the fact that a semantically correct co-ordinate geometry cannot exist without a real world location that it references and the spatial/temporal reference system that is used to reference it.

In other words, a location has one or more co-ordinate geometry's that describe its spatial/temporal domain, the space and time occupied by the entity.

Two fundamental *geographic types* are *features* and *coverage's*. Features and coverage's can be used to map real world entities or phenomena to the Representation Model. A feature is a representation of a real world entity. It has a spatial domain, a temporal domain, or both as an attribute. A coverage is an association of points within a spatial/temporal domain to a value. That is, in a coverage each point has a particular simple or complex value.

Coverages have all of the characteristics of features, and therefore, features and feature collections are the central Representation Model elements. They also are the basic unit of access, management, manipulation, and interchange in the Services Model.

What is the difference between a feature and a coverage? A feature does not return a value for each point. At a given point it may contain another feature or coverage. In itself a feature does not return a value. A coverage, however, can be derived from a collection of features. At a point in the coverage there is a specific feature and an attribute value of that feature is returned.

Each feature typically has, not strictly required in the Representation Model, three components:

1. A description of the geometry in association with a reference system, including a statement of accuracy of the geometric model,
2. A description of the semantic properties of the feature; that is how it is defined,
3. Metadata that might be needed to position the feature in its context.

In the OGC concept there is also a model for the description of attribute data but this is not discussed here.

### 2.3.1 The SQL92 Representation Model

Geographic applications deal with a wide range of geometry's. The Open Geodata Model must provide the capability to deal with geometry's independently of the specific geometric representation chosen by the application. It must be all-inclusive, providing a base for representation and leave room for new definitions.

A feature consists out of geometry's which consist out of data types. The next geodata types are specified in the Representation Model:

1. **Point** - a zero dimensional topology. This type specifies a geometric location,
2. **Curve** - a one dimensional topology. This type specifies a family of geometric entities including line segments, line strings, arcs, b-spline curves, and so on,

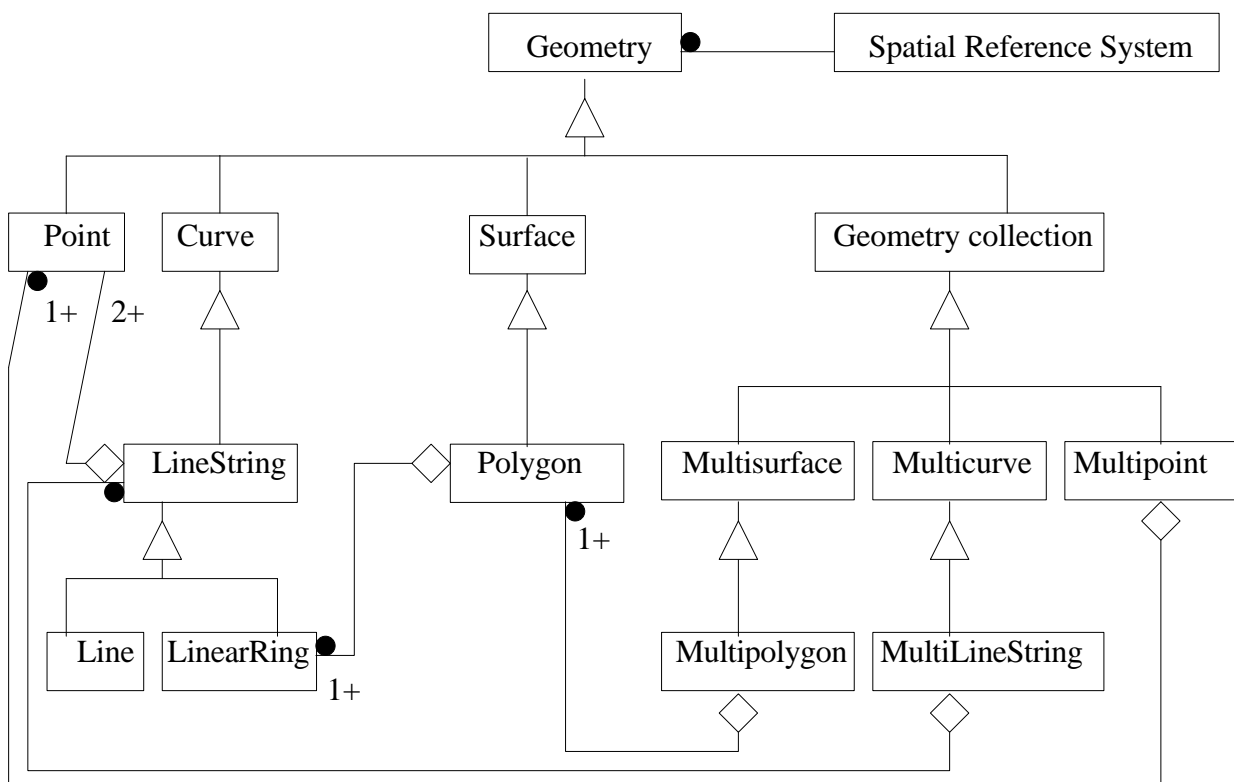


3. **Surface** - a two-dimensional topology. This type specifies a family of geometric entities including areas and surfaces that are defined in other ways,
4. **Solid** - a three-dimensional topology. This type specifies a family of geometric entities including volumes and solids that are defined in other ways.

Data type *Solid* refers to three dimensional space and is not further discussed here.

In the Open Geodata Model it is possible to combine elements of the same data type to a geometry collection. In Figure 6 the building blocks for the geometry specification and there hierarchy are shown.

Figure 6: Feature building block hierarchy.



OGC distinguishes three implementation schema's for features:

1. The implementation model for SQL provides specifications for storage and retrieval in RDBMS's where the language interface is SQL,
2. The implementation model for Object Linking and Embedding (OLE) provides specifications for storage and retrieval in diverse information sources where the interface is OLE/COM,
3. The implementation model for the Common Object Request Broker Architecture (CORBA) provides specifications for the object-oriented distributed systems where the language interface uses CORBA.

In relation to SDO and SDE the SQL implementation specification will be reviewed which is based on the SQL92 standard.

### **2.3.2 The SQL92 Implementation Model Data Architecture**

Simple geospatial geometry collections will conceptually be stored as tables with geometry valued columns in a RDBMS. Each geometry will be stored as a row in a table. The spatial attributes of geometry's will be mapped onto columns whose SQL data types are based on the underlying concept of additional geometric data types for SQL. The OGC model is based on the SQL92 standard (NN, 1996).

There are three categories distinguished:

- 1.a SQL92 with numeric data format representation types,
- 1.b SQL92 with binary data format representation,
2. SQL92 with Geometry Types.

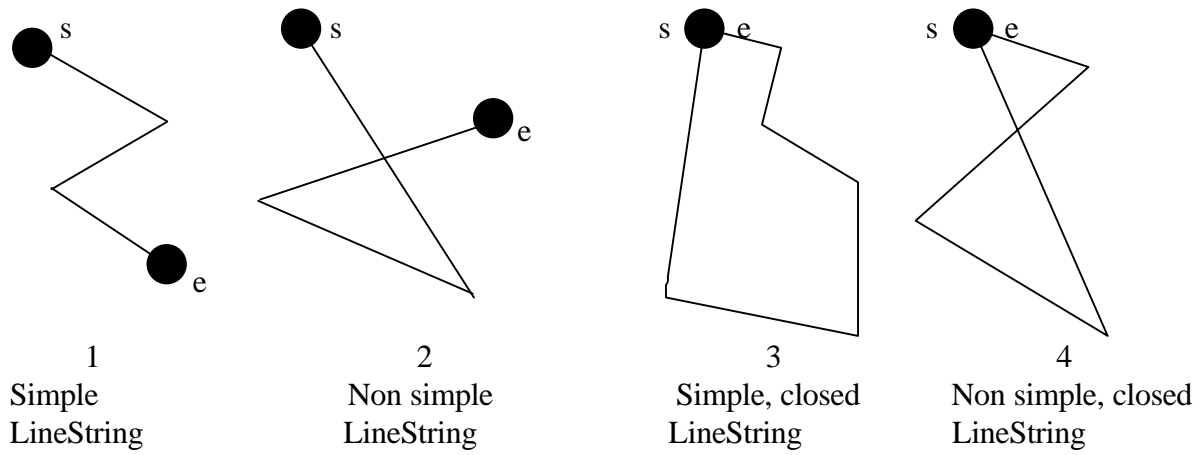
In the **SQL92** environment, a geometry value is stored using one or more rows in the geometry table. The geometry table may be implemented using either standard SQL numeric types or SQL binary types. There are various ways to store the same values in a relational database. For example, there are usually several ways to store numbers. In the OGC specification, the use of a storage alternative is not meant to be binding. Since the storage type of any column is available in the data dictionary, and casting operators between similar types are available, any particular implementation may use alternative storage formats as long as casting operations would not lead to difficulties. The **SQL92 with Geometry Types** is a specification that refers to the SQL92 environment that has been extended with a set of Geometry Types and operation functions on those types. This type of implementation can be found in systems from INFORMIX and IBM. In relation to SDO and SDE only the **SQL92** environment will be discussed here.

The geometry's as defined by OGC are more fully described in appendix A. In brief and with examples they are presented here.

A Point is a 0-dimensional geometry and represents a single location in co-ordinate space. A point has an x co-ordinate value and a y co-ordinate value.

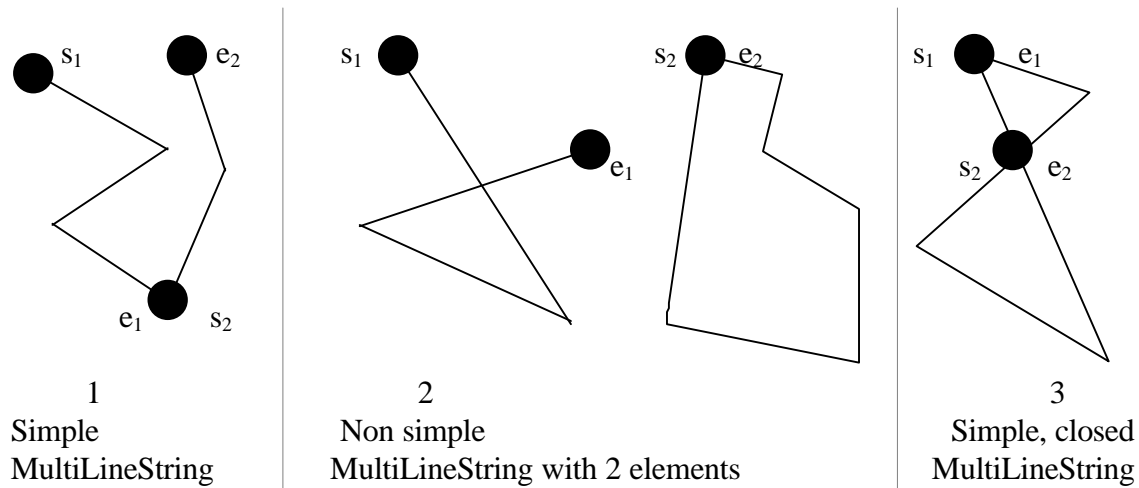
A curve is a one-dimensional geometric object usually stored as a sequence of points, with the subtype of curve specifying the form of the interpolation between points. The OGC specification defines only one subclass of curve, LineString, which uses linear interpolation between points. A curve is simple if it does not pass through the same point twice. A curve is closed if its start point is equal to its end point. A Curve that is simple and closed is a Ring. A Curve is defined as topological closed ( = no gaps). Figure 7 shows types of LineStrings.

Figure 7: Types of LineStrings.



A MultiLineString is a MultiCurve whose elements are LineStrings. Figure 8 shows this type.

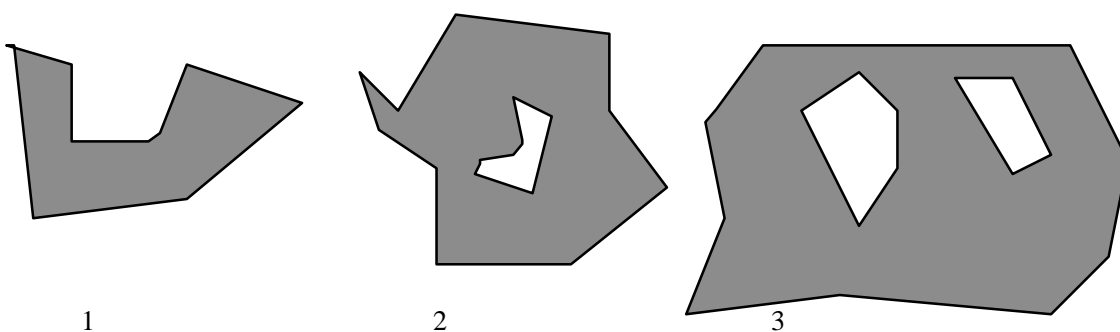
Figure 8: Multi building block geometry's.



In planar two dimensional space a Surface is a two-dimensional geometric object. The OpenGIS Abstract Specification defines a simple surface as consisting of a single ‘patch’ that is associated with one ‘exterior boundary’ and zero or more ‘interior’ boundaries. The only instantiable subclass of surface defined in this specification is the Polygon. A Polygon is a planar surface, defined by one exterior boundary and zero or more interior boundaries. Each interior boundary defines a hole in the polygon.

Figure 9 shows some examples of Polygons.

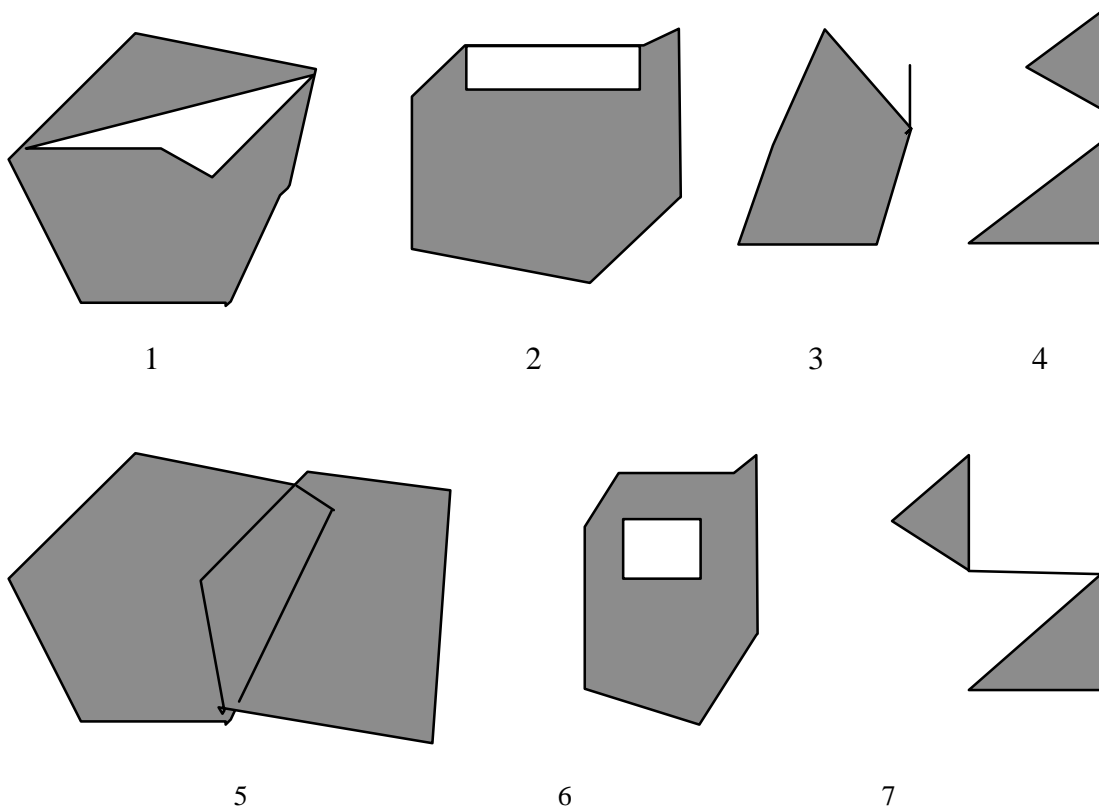
Figure 9: Examples of polygons.



Examples of polygons with 1, 2 and 3 rings respectively.

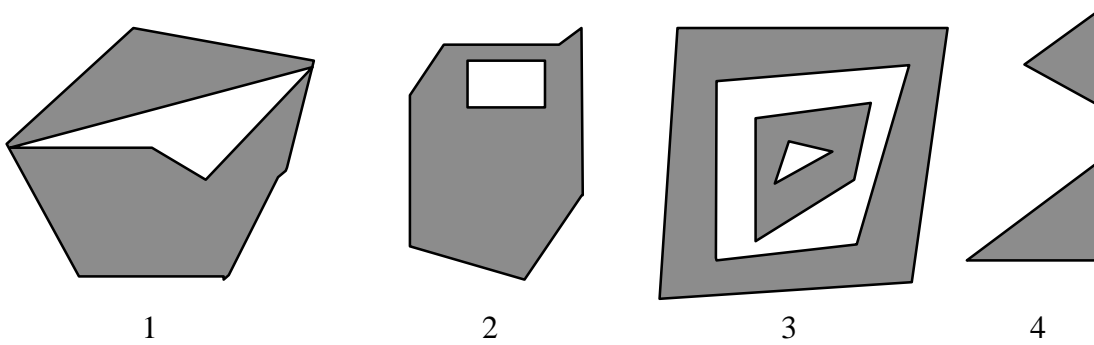
Figure 10 shows some examples of geometric objects that violate the assertions for Polygons and are not representable as single instances of Polygon.

Figure 10: Examples of objects that are not representable as a single instance of a polygon.



A MultiPolygon is a MultiSurface whose elements are Polygons. Figure 11 shows four examples of valid MultiPolygons all having two, one, two and two polygon elements respectively.

Figure 11: Examples of valid Multipolygons.

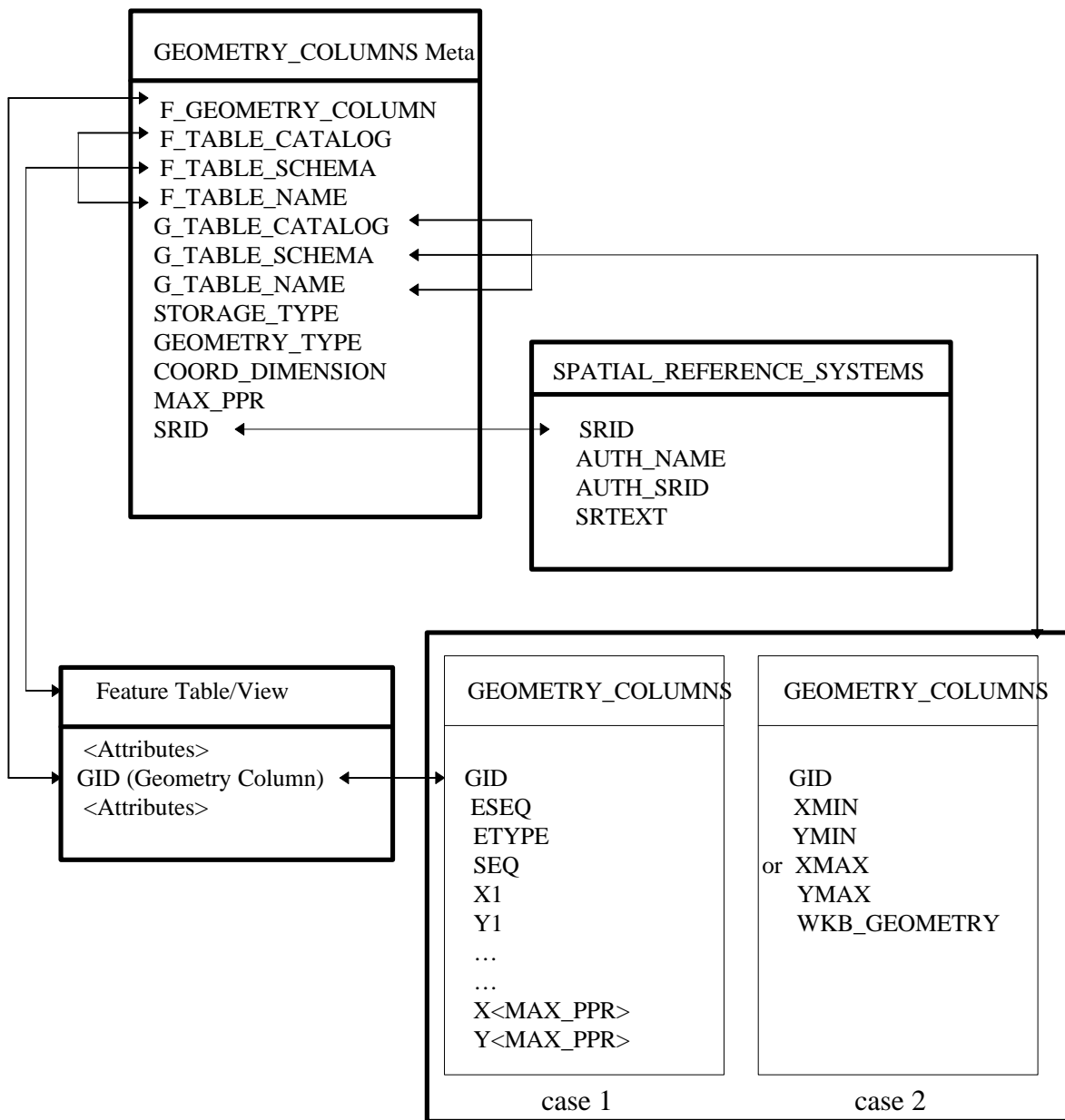


### 2.3.3 The SQL92 Implementation Model Storage Architecture

A **SQL92** implementation of OpenGIS simple geospatial feature collections defines a schema for storage of feature table, geometry and spatial reference system information. The OGC **SQL92** implementation does **not** define **SQL functions** for access, maintenance, or indexing of geometry's, as these functions cannot be uniformly implemented across database systems using the SQL92 standard ( NNc, 1997).

Figure 12 describes the database schema necessary to support the OpenGIS simple feature data model. A feature table or view corresponds to an OpenGIS feature class. Each feature view contains some number of features represented as rows in the view. Each feature contains some number of geometric attribute values represented as columns in the feature view. Each geometric column in a feature view is associated with a particular geometric view or table that contains geometry instances in a single spatial reference system. The correspondence between the feature instances and the geometry instances is accomplished through a foreign key that is stored in the geometry column of the feature table. This foreign key references the Geometry unique IDentification (GID) primary key of the geometry table.

Figure 12: Simple feature table storage schema.



Depending upon the type of storage specified by the geometry metadata, Geometry instances are stored in the **GEOMETRY\_COLUMNS** tables as either arrays of co-ordinate values in columns (case 1), or as binary values using an OpenGIS defined *Well-known Binary field* also known as the Binary Large Object (BLOB) (case 2). A binary geometry is called a Well-known Binary Geometry (WKBGeometry). A more detailed description of the geometry's is given in appendix B. It is sufficient to mention here that each geometry element has an unique



identifier. Storage of co-ordinate values in case 1 opens the possibility to address geometry's using the SQL interface. Storage in case 2 provides means to directly feed the co-ordinate values into the Geometry routine sets based on the OLE/COM or CORBA interface environment.

With the GEOMETRY\_COLUMNS meta view the geometric feature tables can be determined. Also a reference key is present to determine the reference system of the geometry's that are stored.

A Feature is an object with geometric attributes. Features are stored as rows in tables, each geometric attribute can be a foreign key reference to a geometry table or view. Relationships between Features are defined as *foreign key* references between feature tables.

### **2.3.4 The SQL92 Relational Operators**

The SQL92 Relational Operators in the OGC concept are used to test for the existence of a specified topological spatial relationship between two geometry's. They are based on the Boolean method. The basic approach is to compare two geometry's using a pair wise test of possible intersections of there morphology's. These operations are essential since they give the user the opportunity to select geometry's based upon a spatial neighbourhood relation. For example "select all geometry's that spatially fall within a query polygon" for functions as selecting and zooming.

All geometry's have an interior, boundary and exterior. The interior may have an empty set (Point).

Given a geometry  $a$ , let  $I(a)$ ,  $B(a)$  and  $E(a)$  represent the Interior, Boundary and Exterior of  $a$  respectively, the intersection of any two of  $I(a)$ ,  $B(a)$  and  $E(a)$  can result in a set of geometry's,  $x$ , of mixed dimension covering nine situations. The returned set of the comparison might be a mixture of the next unique results:

1. empty,
2. touch in one point,
3. touch on one line segment,
4. have a common part.

A dimensionally extended nine-intersection matrix (DE-9IM) then has the form:

	Interior	Boundary	Exterior
Interior	$\dim(I(a) \cap I(b))$	$\dim(I(a) \cap B(b))$	$\dim(I(a) \cap E(b))$
Boundary	$\dim(B(a) \cap I(b))$	$\dim(B(a) \cap B(b))$	$\dim(B(a) \cap E(b))$
Exterior	$\dim(E(a) \cap I(b))$	$\dim(E(a) \cap B(b))$	$\dim(E(a) \cap E(b))$

The set of interactions that may occur between geometry's consists of:

1. Polygon and Polygon (A/A),
2. Polygon and LineString (A/L),
3. Polygon and Point (A/P),
4. LineString and LineString (L/L),
5. LineString and Point (L/P),
6. Point and Point (P/P).

The  $dim(x)$  set as a result of a test in the DE-9IM matrix may have the following named types of interactions:

1. **Equal.** Geometry  $a$  and  $b$  are the same at the same location,
2. **Disjoint.** Geometry  $a$  and  $b$  are not spatially related,
3. **Touch.** Touch does not apply to the (P/P) interaction. The geometry's touch in one or more locations and in the case of (A/A) they even may touch along one or more line segments,
4. **Cross.** Cross only applies to (A/L), (A/P), (L/L), (L/P). When crossing the geometry's have one or more common locations but less common locations then the total number of co-ordinate pairs of the geometry with the fewest co-ordinates,
5. **In (within).** The entire geometry  $a$  falls totally within geometry  $b$ ,
6. **Overlap.** Overlap only applies to (A/A), (L/L), (P/P). Two Geometry's overlap if part of the geometry's location is common.

For user convenience also defined are:

**Contain.** Geometry  $a$  spatially completely falls within geometry  $b$ ,

**Intersect.** Geometry  $a$  and  $b$  are spatially related,

**Relate.** A function that tests a given interaction.

The Boolean buffer zone functions distinguished are:

1. **IntersectBufferZone** (Geometry having the bufferzone, interaction geometry, distance),
2. **ContainsBufferZone** (Geometry having the bufferzone, interaction geometry, distance),

3. **ContainedInBufferZone** (Geometry having the bufferzone, interaction geometry, distance).

### 2.3.5 The SQL92 Retrieval Model

The concept of retrieval of data is part of the OGC Services Model. The Services Model has not yet been published by OGC. Still, a preliminary outset of the model is given. As far as this outset relates to the SQL92 implementation it will be presented here.

The OGC Services Model incorporates the following main objectives:

1. The model is independent of operating system, hardware and human-technology interface,
2. Provide concepts on multi user data sharing,
3. Provide concepts upon “client” and “server” and Distributed Computer Environments (DCE),
4. Provide concepts upon interfacing language(-s),
5. Support the ability to (dynamically) extend the available services,
6. Harmonise geoprocessing environments.

Regarding query languages OGC states (NN, 1996) that the language must be able to deal with the four categories of properties each geometry incorporates. The four properties are:

1. Spatial extent,
2. Temporal extent,

3. Structured attributes defining the non spatial properties of a geometry,
4. Textual attributes containing (lengthily) descriptive data.

Until now OGC states that: “any operation on the properties on any type of geometry will be available from within the query language” ( NN, 1996).

This statement implies, for the spatial extent, the incorporation of the Boolean spatial operations as presented in paragraph 2.3.4 and buffer zone functions. All other properties, except the temporal extent, should already be covered by the functionality of the used interface language, since they are related to alphanumeric operations.

The temporal extent of a feature can be defined as a finite set of non-intersecting sequential time intervals. Query conditions on the temporal extent will typically need to use Boolean intersects for each time interval.

Query conditions on structured attributes will typically need to use the types of operations which are already supported by standard SQL.

Query conditions on textual attributes will typically involve text string reference and keyword functions. These type of operations lie at the core heart of full-text indexing/searching systems.

## **2.4 Discussion**

### **2.4.1 The OGC collaboration**

The ideal of a uniform approach of interoperable GIS decision support functionality is of great importance to the GIS community. The collaboration of users and developers gathers important players on the subject. The users view depicts the ideal of open systems where the efforts of combining “foreign” data and “foreign” programs are reduced to a minimum. Developers also strive for these goals since in that way GIS products can be marketed. But the developers are confronted with the investments they have done in their company current packages. In that respect it seems legitimate that the developers bring forward their own concepts to be accepted as OGC standards. These concepts can be very adequate and be acceptable to OGC rules and guidelines. However, it rises the question if OGC is unbounded and is free to develop concepts and directions based upon the ideal of OpenGIS without restrictions. The implementation of OpenGIS highly depends upon the concepts that are developed.

OGC states that they do not propagate a mandatory type of data storage architecture since it is more essential that retrieval and spatial operations are as much declarative as possible. However, OGC does describe storage architectures. Reasons for doing so are not made explicit.

### 2.4.2 The SQL92 representation Model

The SQL92 representation model is based on the existence of spatial data and operations in a RDBMS. Today's RDBMS systems require much knowledge on creating a storage and authorisation environment that comes along with the growing complexity of the RDBMS to support database environmental utilities and functions. This may cause the effect that GIS in RDBMS will only be implemented in long-term temporal systems. Short-term ad hoc GIS implementations may not want to be troubled with demanding RDBMS installations. Although the SQL92 representation model can be applied in any GIS instance OGC may want to consider how to deal with short-term ad hoc GIS databases.

### 2.4.3 The SQL92 Implementation Model Data Architecture

The SQL92 Implementation Model Data Architecture distinguishes the three basic feature types point, curve and surface. They are composed out of the basic geometry's Point, Linestring and Polygon. These elements can have attribute data attached to them.

In the vector representation there is an other kind of element that is of great importance. It can be named **Node** which has the spatial characteristic that it is an intermediate location on a curve or surface boundary. A Node can have attribute data. For Example a dam in a river, a crossing of roads, or the junction of three state borders.

The most significant difference with Point is that Node is directly coupled to a geometry.

It is recommended that OGC clarifies the observation Node.

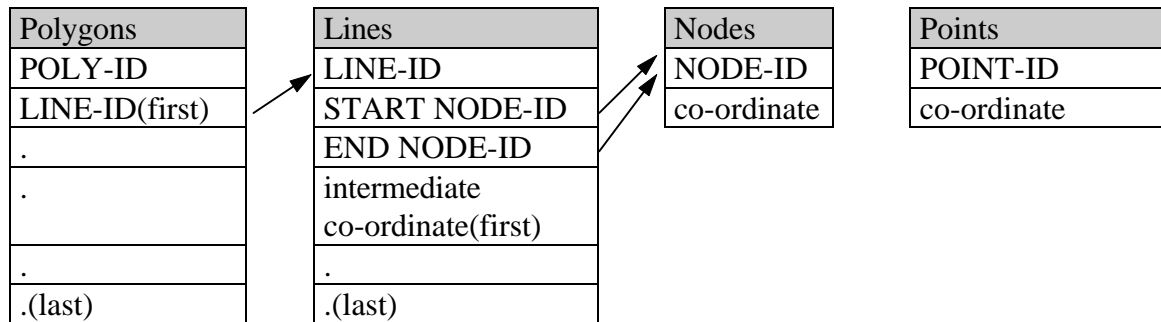
#### **2.4.4 The SQL92 Implementation Model Storage Architecture**

The SQL92 Implementation Model Storage Architecture presents different storage structures for data in numeric and data in binary format. OGC states that there are several kinds of internal representations to store numbers (co-ordinates) like integer, real and byte stored binaries. The way numbers are stored in elementary bit and byte representation is not related to the way these bytes are logically grouped into representational structures. A binary or numeric stored co-ordinate is part of a group of co-ordinates which are grouped into units that are structured in logical sets. In this respect it seems unnecessary to promote more than one storage architecture.

The Model Storage Architecture has an object based approach. This means that every geometry is self supporting, not dependent of any other entity. With this in mind the presented storage structure, in both representations, that holds the co-ordinate data lacks any kind of normalisation. Co-ordinate duplication occurs in any instance where geometry's interact. For example, when roads are presented as lines and a junction is considered, the co-ordinate of the junction is stored as many times as there are lines ending/starting at the junction. When polygon features are considered, two polygons that are neighbours have a common boundary. The co-ordinates of this boundary are stored separately for each polygon and are thereby present in the database in duplicate. Figure 13 shows a schema of a normalised table set representation in a topological correct environment which overcomes this lack of normalisation. Note that in this concept each geometry type ( Polygon, Line, Node, Point) can have its own attribute table(s) using the ID-values to relate to it.



Figure 13: A Schema on the structure of normalised tables holding vector data.



From Figure 13 it can be derived that every co-ordinate has to be stored only once to capture the entire spatial morphology of a representation of a part of the real world under consideration.

Conceptually the presented schema intrinsically bears topology. Neighbourhood queries can be posed on the table set using the existing SQL standard interface. For example a “Select all roads coming together at a crossing” would look like:

```

Select lines.line-id
from lines,
      nodes
where nodes.node-id = lines.start_node-id or
      nodes.node-id = lines.end_node-id and
      nodes.node-id = <value>;

```

A “Select the adjacent polygons of this polygon” would look like:

```

Select Polygons.Poly-id
from Polygons
where Polygon.line-id in ( Select POLYgons.line-id
                          from Polygons
                          where Polygon-id = <value> ) and
      Polygons.Poly-id != <value>;

```

As can be seen from these examples, using intrinsic topology relationship, spatial operators simply consist out of a set of SQL statements and there do not have to be additional spatial routines.

It is also clear that in this schema locating a geometry requires several “1 to Many” search actions on the tables. In practise this makes this concept slow and therefore will put heavy demands on the linear index methods of the tables and on the spatial index methods on the stored features for other not intrinsic spatial operations.

When response time is crucial the OGC binary storage architecture is very favourable since it can retrieve a geometry by a single “row hit”. The numeric storage architecture may require some more “hits”. In both storage structures topology related operations have to be computed every time they are induced.

It seems that while computer resource capacities change the storage of data and operations upon them change. In former days disk capacities were low and computation speed was moderate. It made sense to store the voluminous co-ordinate data sets efficiently in tables structured to support intrinsic topological relationships. Small disk capacities were used effectively and intensive spatial related computations were reduced. Currently disk space seems no problem nor the availability of spatial relations since they are computed at run-time and still there are swift responses. Because computation capacity has increased dramatically and disk searching is still a mechanical activity reducing disk activity, which is compensated by more computational activity, is more feasible. The OGC storage and retrieval architecture fits into this trend but moves away from an essential property of spatial data: **intrinsic topological relationship**.

Spatial editing in the OGC presented storage structures is simple. Co-ordinate transactions can be performed without any consideration regarding topological correctness. This can be an advantage or a threat as well.

In the presented schema in Figure 13 editing has to be performed in a program environment that guards topological correctness. But might also be simple. For example shifting a co-ordinate in a common boundary automatically updates all polygons involved.

The OGC storage structure with binary data types can only be addressed by an API. The storage structure with numeric data types has the advantage of being addressable by standard SQL and SQL incorporated interaction programs ( e.g. API).

The OGC storage structure with numeric data types defines a table with a finite number of columns to hold co-ordinates. An element of a geometry may have less, to same, or more co-ordinates than is defined in the table. Having less co-ordinates means loss of space, having more, means an additional row in the table. Future developments may have table implementations where per row the number of columns can vary. This saves space and reduces search activities in the table.

The storage structure with binary data types uses a binary large field. This field is finite. Future developments may have tables with binary large fields that are only bound by the resource capacity of the computer ( e.g. a row that spans all available disks (network) connected to the computer) and have variable length for each row.

### 2.4.5 The SQL92 Relational operators

Spatial operators can be categorised into 3 types:

1. Geometric calculation operators,
2. Spatial comparison operators,
3. Buffer operators.

“**Geometric calculation operators**” are functions that return morphology characteristics. OGC has specified: Length, Perimeter, Area, Centroid and Point\_on\_Surface. Other not addressed operators are: Distance, Start, End, and Closest.

“**Spatial comparison operators**” are functions that investigate the spatial relation between two morphology's. They can return Boolean values or the result geometry of a specified interaction. The spatial operations within the OGC concept are presented in par. 2.3.4. The relational operators as defined by OGC return Boolean values.

An important option is missing in this set. It is the “direction” argument, with which it is possible to determine what is “left or right” or “next related element”. The importance may become evident since there is no intrinsic topology present in the OGC concepts of storing spatial data. For example, with the direction argument it can be determined if one has to be on the left or right bank of a river, what is the next lot on a route, or what is the clock wise order of connections of roads at a crossing.

“**Buffer operators**”, are functions that can create a zone around a (set of) morphology('s). OGC has defined three types of buffer zones in relation to the interaction with the geometry('s)

from which the buffer originates. They do not calculate the bufferzone boundary itself but search for elements that are spatially related with the zone. In OGC they are only specified as Boolean operators.

## 2.4.6 The SQL92 Retrieval Model

Retrieval of geospatial data is based upon the concepts of SQL with the extension of specific functions to determine spatial interaction. OGC has not yet addressed the grammar and syntax of SQL where spatial interaction clauses are included.

Gadia (1993) and Samet (1994) discuss a SQL syntax where spatial operations and attribute operations are incorporated. The classic form of a select statement is:

“Select ... from ... where ...;”

They extend the syntax by a spatial clause:

“Select ... spatial interaction ... from ... where ...;”

Important to notice here is that in the same way as there can be nested select statements in the **where** clause it is possible to nest select statements in the **spatial interaction** clause.

Hettema (1995) describes a syntax where the **spatial condition** is incorporated into the “where”-clause of the SQL select statement. He also notes that there is a separation between the alphanumeric and spatial condition in a select query:

“Select ... from ... where ... <spatial condition> ... <alphanumeric condition>;”

SDO and SDE comply with the grammar defined by Hettema (1995).

## **2.5 Summary**

From this chapter three basic issues on the OGC concept emerge:

1. Geometry's are categorised into the classes simple and non simple morphology's.
2. The data storage architecture can be of numeric or binary format, each having a different storage structure.
3. The retrieval mechanism is still under OGC development and is therefore open for any kind of implementation until now.

These issue's leave room for interpretation and implementation into commercial packages. Regarding SDO and SDE they should be investigated. This should lead to insight to the extent OGC concepts are implemented and what the practical aspects of these implementation variants are to the user interaction. In the next chapter SDO and SDE will be investigated on data model, storage structure, spatial index in relation to retrieval, data manipulation and data retrieval in order to gain this insight.

### 3. The OGC concept in SDO and SDE

A characteristic of the information technology industry is the extremely rapid development of new versions of software packages. In this study ORACLE Corporation SDO version 7.3.3 is taken under consideration and ESRI SDE version 3.0.2. In this chapter parts of the functionality on these package versions are described and discussed.

In fact the user is not interested in the way data are organised in the database. She/he only wants to store and retrieve. However, any retrieval language needs specifications from the user in order to execute what is demanded. In the case of SQL the user has to know about the phenomena stored (entity types) storage structures in tables (**From** condition) and columns in these tables (**Insert/Select/Update/Delete** condition). Further the user should know about available operations and entity types (**Where** condition), both spatial and attribute based. And for convenience, spatial search (indexing) implementation in tables might be of help to speed up system response times.

In this context this chapter is divided into paragraphs on data model (entity types), storage structures (tables and columns), spatial indexing and data manipulation and retrieval for each package.

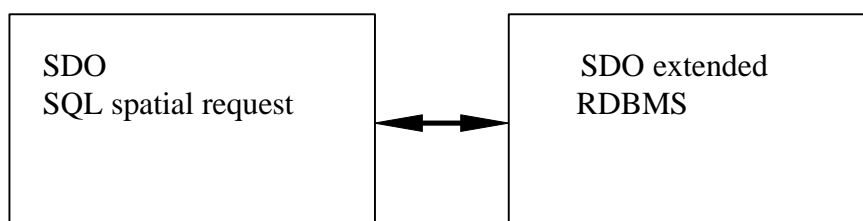
#### 3.1 SDO

ORACLE Corporation introduced Spatial Data Option (SDO) 7.3.3 as an integrated tool set of functions and procedures for there relational database ORACLE 7.3.3. It enables planar two

dimensional spatial vector data to be stored, accessed, and analysed within the RDBMS environment.

SDO is based on SQL with an extension for dealing with spatial entities and operations. The SDO architecture is shown in Figure 14.

Figure 14: SDO architecture



### 3.1.1 Data Model

In SDO geometric representations of world phenomena can be stored. In SDO geometric representations have the next set of characteristics:

1. are planar two dimensional,
2. can consist out of one or more elements,
3. can not have annotation,
4. can not have attribute values at any given point along the geometry.

The SDO data model is a hierarchical structure consisting out of elements, geometry's and layers. An element is the basic building block of a representation of a worlds entity or phenomenon.



SDO supports three geometric basic elements:

1. Point,
2. LineString,
3. Polygon.

Multi geometry's are supported in SDO. Based on the SDO\_ESEQ attribute more than one geometry can be part of a multi geometry entity or phenomenon. In the case of Polygon, a set of polygons can represent an entity but it also offers the possibility to deal with polygons having one or more (polygon type) holes in them.

A layer in SDO is defined as a heterogeneous collection of geometry's having the same attribute set.

The interpolation schema for line segments and polygon line segments is not made explicit. The spatial functions of SDO all imply a linear interpolation between co-ordinate pairs.

SDO uses a set of rules to validate geometry's. Self-crossing polygons are not supported, but self-crossing lines are. A self-crossing LineString does not have an implied interior. In appendix C the full set of validation rules is presented.

### **3.1.2 The Database Storage Structure**

A geometry in SDO is mapped against the relational constructs in the next set of aspects:

1. Each geometry is stored in a set of rows in one relational table or view,
2. Attribute data and geometry data are linked by a foreign key when they are stored in different tables or views,
3. The morphology of a geometry is stored as a series of values in table columns.

SDO uses four tables. One table to store geometry's and three tables to store spatial indexing and retrieval information. All the tables must have a typical SDO naming convention in order to let functions and operations locate the tables and act on them. The tables are:

1. <layer>\_SDOLAYER,
2. <layer>\_SDODIM,
3. <layer>\_SDOGEOM,
4. <layer>\_SDOINDEX.

In this representation “<layer>” is a user defined specification.

According to OGC SDO does not implicitly have a GEOMETRY\_COLUMNS metadata table nor a SPATIAL\_REFERENCE\_SYSTEM table. Location is stored as co-ordinate pairs using integer or real values. Processing takes place using these values. SDO has no algorithms to convert X- Y- co-ordinates into a spherical system.

Table <layer>\_SDOGEOM is conform the described data structure in paragraph 2.3.3. As stated in paragraph 1.1.2 co-ordinates have sequence to determine the morphology of a geometry. In the SDO environment this sequence is secured by an obligatory naming convention of columns in the table. To store co-ordinates columns must have the name

SDO\_X $n$  and SDO\_Y $n$ , where  $n$  is the N-th ordinate, starting with 1 and incremented by 1. In this way columns may be ordered randomly in the table. The number of co-ordinates that can be stored in a row is finite. Within SDO the total number of columns in a table is limited to 255. Thus, in a standard <layer>\_SDOGEOM table there can only be a maximum of 125 co-ordinates in one single row including all other required columns. More co-ordinates of the same geometry can be stored in an additional row with the restriction that SDO\_X1 and SDO\_Y1 of the additional row contains the values of SDO\_X<sub>last</sub> and SDO\_Y<sub>last</sub> of the previous row. The number of additional rows is not limited and they are identified by a row sequence identifier, SDO\_SEQ, starting with a value of 0 for the first geometry row and increasing by 1.

The user can specify how many co-ordinate pairs there are in a row in the <layer>\_SDOGEOM table. But most geometry's have more or less co-ordinate pairs that will fill out a complete (set of) row(s). Since the table has a fixed number of ordinate columns there will be empty cells in the table.

It is remarked here that SDO has functions that can determine the optimal number of co-ordinates per row in the <-layer>\_SDOGEOM table (NNb, 1997). But these functions imply that the geometry's to be stored are ( by approximation) already known and loaded into the table. Frequent changes in geometry numbers and number of co-ordinates per geometry require re-optimisation of the table.

The other tables regarding spatial indexing are beside the scope of this study but will be discussed in brief in the next paragraph since they influence the way geometry's are retrieved using the SDO spatial extended SQL implementation.

### 3.1.3 Spatial index

The three tables to store indexing and retrieval information are:

1. <layer>\_SDOLAYER,
2. <layer>\_SDODIM,
3. <layer>\_SDOINDEX.

In <layer>\_SDOLAYER the number of ordinates in <layer>\_SDOGEOM and the number of levels of the quad tree indexing system is stored.

In <layer>\_SDODIM the boundaries of the space to be considered is stored. Geometry's may not be outside these boundaries, not even in part. This means that the entire space of interest must be known by forehand. If not, the dimensions have to be adjusted and the indexing information recompiled. In this table also the comparison tolerance is stored. This value is necessary since decimal values in SDO may be stored in real format and real format is approximated in the computer operating system as a power with base 2. Within the tolerance distance all co-ordinates are considered to depict the same location.

In <layer>\_SDOINDEX indexing information is stored. SDO uses the quad tree spatial indexing schema. In short, the index is denoted using unique hierarchical numeric identifiers corresponding to the ordered sequence of rectangles in the quad subdivision of space (Samet, 1990). Since these numbers are one dimensional they can be used to populate the spatial index table. Table <layer>\_SDOINDEX is maintained for this purpose.

Appendix D gives an example of physical storage of geometry's in SDO.

### 3.1.4 Data manipulation

Data in SDO can be inserted using three methodologies:

1. Bulk loading using the ORACLE LOADER utility,
2. Transactional SQL INSERT statement inserting values per column,
3. Transactional SQL INSERT statement with SDO functions inserting geometry's.

Bulk loading of spatial data with ORACLE LOADER is conform loading any ORACLE table. Data must be in correct format and as values per column where each row is in accordance with SDO requirements. The LOADER input data files in fact reflect the correct table contents.

Using the transactional SQL INSERT statement inserting values per column acts upon one row of data per call. Geometry's covering more than one row require multiple calls of the INSERT statement in which case values for the required columns have to be repeated and/or adjusted in the statement. The user must self supervise the correctness of the representation of the geometry in the rows it occupies.

The transactional SQL INSERT statement with SDO functions inserting geometry's requires two statements to insert a geometry. The benefit of these functions is that row-wrapping is performed automatically. First An initialisation function is to be called and then an insert function (SDO\_GEOM.INIT\_ELEMENT). Then SDO\_GEOM.ADD\_NODES can be called to insert the geometry. These functions are provided to relieve the user from row wrapping when multiple row inserts are required. Validation of inserted geometry's can be performed by the SDO\_GEOM.VALIDATE\_GEOMETRY function.

After all geometry's are inserted the spatial indexes and table indexes have to be populated.

This a user action, which is supported by functions.

Geometry morphology manipulation can be performed using standard SQL statements like UPDATE. Discarding geometry's can be performed by the SQL statement DELETE. The user has the responsibility to discard the data in all four spatial related tables.

### **3.1.5 Data retrieval**

#### **3.1.5.1 Spatial operators**

For the purpose of SDO spatial operations in the SQL interface language, SQL is extended with a set of functions. There are four categories of functions:

1. Geometry Functions,
2. Window Functions,
3. Administrative Procedures,
4. Partitioned Point Data Functions.

The **Partitioned Point Data** Functions are related to legacy point data tables in former releases of SDO and are no further discussed here.

The **Administrative** Procedures are provided to create and maintain the spatial index structure in the database. They will also not be discussed here.

The **Window** Functions is a set of functions to create and delete an “overlay select polygon”, like a zoom window, in a flexible and fast way. In fact these functions create an instance of the SDO system in the form of the tables and indexes with there exact equal spatial indexing parameters that can hold all types of geometry’s. Because the spatial indexing is the same as the data set under consideration fast primary and secondary filter operations are performed.

The Window functions are not further discussed here.

The **Geometry** Functions is a set of functions that deals with geometry insertion and validation, and geometry “**spatial comparison operators**”. Some are already brought forward, but next the full list is given. SDO Geometry Functions are:

- |                               |  |
|-------------------------------|--|
| 1. SDO_GEOM.INIT_ELEMENT      | Initialises space for a geometry to be stored, |
| 2. SDO_GOEM.ADD_NODES         | Stores geometry’s,                             |
| 3. SDO_GEOM.VALIDATE_GEOMETRY | Determines if a geometry is valid,             |
| 4. SDO_GEOM.INTERACT          | Determines if two geometry’s are disjoint,     |
| 5. SDO_GEOM.RELATE            | Determines how two geometry’s interact.        |

Functions SDO\_GEOM.INIT\_ELEMENT, SDO\_GEOM.ADD\_NODES and SDO\_GEOM.VALIDATE\_GEOMETRY are already described. SDO\_GEOM.INTERACT is equal to the OGC **Intersect** function. SDO\_GEOM.RELATE is an implementation of OGC **Interact**.

SDO\_GEOM.RELATE can indicate the next spatial interactions:

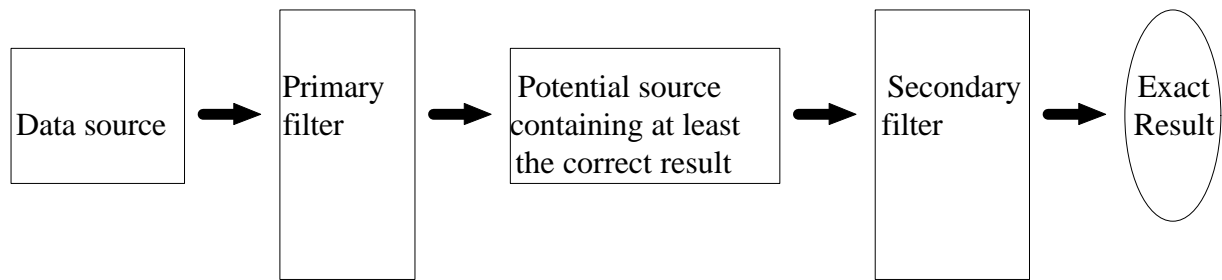
1. **Anyinteract**. Geometry *a* and *b* are spatially related,
2. **Contains**. Geometry *b* is entirely within geometry *a* and the boundaries do not touch,
3. **Coverdby**. Geometry *a* is entirely within geometry *b* and the boundaries touch,
4. **Covers**. Geometry *b* is entirely within geometry *a* and the boundaries touch,
5. **Disjoint**. Geometry *a* and *b* are not spatially related,
6. **Equal**. Geometry *a* and *b* at every boundary location the same,
7. **Inside**. Geometry *a* is entirely within geometry *b* and the boundaries do not touch,
8. **Overlapbdydisjoint**. Geometry *a* and *b* overlap but their boundaries do not interact,
9. **Overlapbdyintersect**. Geometry *a* and *b* overlap and their boundaries intersect,
10. **Touch**. Geometry *a* and *b* share a common boundary point, but no interior points.

### 3.1.5.2 The Query Model concept

In SDO the feature retrieval and spatial operations mechanism is based upon SQL. Spatial query's are performed in a two step operation and the mechanism is called the "two-tier" query model. The two operations are referred to as **primary** and **secondary** filter operations. Figure 15 depicts the model.



Figure 15: The SDO two tier retrieval model.



The primary filter permits fast selection of a limited number of candidate rows. The primary filter uses approximations in order to reduce computational complexity. The secondary filter applies exact geometry relation computation upon the result set of the primary filter.

SDO does not require the use of both the primary and secondary filter. In some cases, just using the primary filter is sufficient. The superset the primary filter produces is sufficient for applications like zooming in on an area. The features in the superset are clipped by the visualisation software to display the features in the zoom window.

The SDO query model has a strong relationship with the implementation of the indexing mechanism in SDO. As discussed in paragraph 4.2.3 table <layer>\_SDOINDEX contains the quadtree index reference code for each geometry. In an overlay operation the quadtree index reference code of the overlaying geometry('s) is compared with the code in <layer>\_SDOINDEX. This is a one dimensional search and will quickly provide the candidate set as the primary filter result. This method also reveals why it is important that all geometry's must fall within the quadtree segmented space under consideration. Logically the primary filter may be defined as: "Select all geometry-id's that have the same quadtree index codes as those given by this geometry (or set of geometry's)".

The secondary filter is an exact calculation of the geometry's relationship using there morphology. A function called SDO\_GEOM.RELATE() actually computes spatial relationships between two given geometry's. The OGC function relationships are clarified in paragraph 2.3.4.

### 3.1.5.3 The Query Model implementation

The query implementation model takes the same form as defined by Hettema (1995). That is:

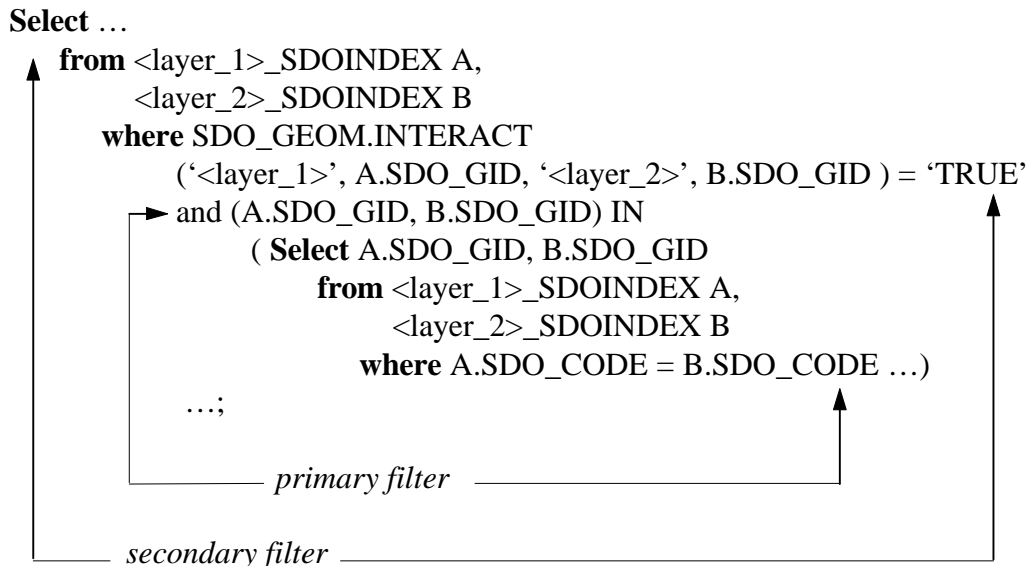
“Select ... from ... where ... <spatial condition>...<alphanumeric condition>;”

A query using the primary filter only takes the form:

```
Select ...  
  from <layer_1>_SDOINDEX A,  
        <layer_2>_SDOINDEX B  
  where A.SDO_CODE = B.SDO_CODE ...;
```

Note that the “spatial operation” is executed using an alphanumeric statement on the SDO\_CODE column which holds the quadtree index. Any valid alphanumeric statement regarding the SDO\_CODE is correct here. The **where** clause can be extended with other alphanumeric operators.

A query using the primary filter and secondary filter where the primary filter output is the input to the secondary filter, that uses the INTERACT operator, takes the form:

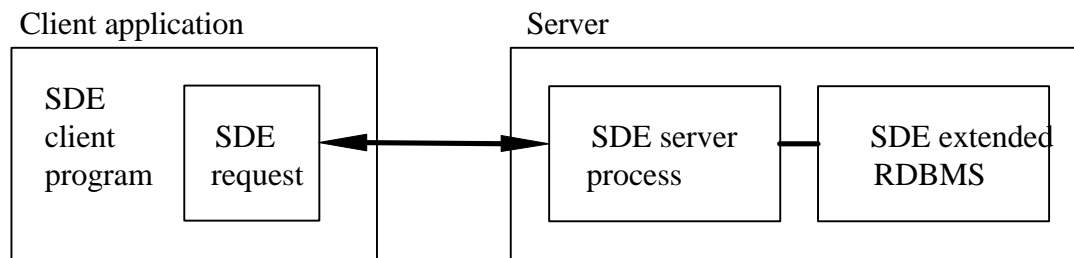


## 3.2 SDE

ESRI introduced Spatial Data Engine (SDE) 3.0.2 as an integrated tool set of functions and procedures for the relational database ORACLE 7.3.3. It enables three dimensional and planar two dimensional spatial vector data to be stored, accessed, and analysed within the RDBMS environment.

SDE is based on an imbedded SQL application program interface (API) with a set of non SQL based spatial routines. It is structured upon the client-server architecture. The client program directs SDE requests to the SDE server process. The SDE server returns results to the client program. For clarity reasons the intermediate SDE request manager is left out of this schema. The SDE architecture is shown in Figure 16.

Figure 16: SDE architecture



The server process actually handles the data operations. It passes back only those data that meet the client request. In this way the amount of data load across the network is kept to a minimum. SDE uses co-operative processing, meaning that both the server and client execute requests. Requests on data already at the client side (like zooming in) are processed on the client side as well as CPU intensive tasks ( like clipping or overlaying) where it is not required to search large volumes of data.

### 3.2.1 Data Model

In SDE geometric representations of world phenomena can be stored. In SDE they have the next set of characteristics:

1. can be planar two or three dimensional,
2. can consist out of one or more elements,
3. can have annotation,
4. can have attribute values at any given point along the geometry,
5. have verification rules to ensure accuracy.

The SDE data model is a hierarchical structure consisting out of elements, geometry's, called shapes, and layers.

SDE supports five geometric basic elements:

1. Nil,
2. Point,
3. Line,
4. Simple Line or LineString,
5. Polygon.

A Nil element is an element with no co-ordinates and merely acts as a placeholder. For example the result of the interaction between two disjoint elements is a Nil element.

A layer in SDE is defined as a heterogeneous collection of geometry's having the same attribute set.

The interpolation schema for line segments and polygon line segments is not made explicit. The spatial functions of SDE all imply a linear interpolation between co-ordinate pairs.

A Line element is a line type that may cross itself. A self-crossing Line does not have an implied interior. A MultiPoint and MultiLineString geometry's are supported in SDE. SDE also does support MultiPolygon geometry's. Within SDE a multi element polygonal geometry consists out of sub elements who are completely contained within the outermost element. This

gives SDE the possibility to deal with polygons that have 1 or more holes in them. Appendix D presents the full set of geometry validation rules in SDE.

### **3.2.2 The Database Storage Structure**

A geometry in SDE is mapped against the relational constructs in the next set of aspects:

1. Each geometry is stored in a row in a relational table or view,
2. Attribute data and geometric data are linked by a foreign key when they are stored in different tables or views,
3. The morphology of a geometry is regarded as a value in a table/view field.

SDE uses four tables. one table to store the geometry's, two tables to store indexing and retrieval information, and one table, defined by the user, which holds the attribute data but receives an extra column to refer to the spatial data. The three actual SDE tables must have a typical SDE naming convention in order to let functions and operations locate the tables and act on them. The tables are:

1. user table (with the SDE reference column),
2. LAYERS,
3. F<layer\_id>,
4. S<layer\_id>.

In this representation <layer\_id> is a user defined specification that is stored in LAYERS.

According to OGC SDE does not implicitly have a GEOMETRY\_COLUMNS metadata table nor a SPATIAL\_REFERENCE\_SYSTEM table. The data in these tables is in SDE contained in the table LAYERS.

The F<layer\_id> stores the geometry's and is in accordance with the OGC data structure described in paragraph 2.3.3 with the extension of three dimensional data columns. F<layer\_id> has an extra set of attributes regarding geometry type, Node data and "calculated operators" (NNd, 1997). Location is stored as (a series of) co-ordinate pairs using positive integer values stored in binary format into one column in F<layer\_id>. To convert ( to and from) floating point values a scale factor (resolution) and x-y-offset are calculated for each coverage and stored in LAYERS. It is beyond the scope of this study to discuss in detail how the co-ordinates are physically stored in SDE. Basically SDE uses Binary Large Object fields (BLOB's) to store the entire morphology of a geometry into one column.

Table S<layer\_id> holds the spatial indexing system. Although beside the scope of this study it will be discussed in brief since it is possible to make use of this table in retrieval operations using SQL.

Appendix D gives an example of physical storage of geometry's in SDE.

### **3.2.3 Spatial index**

The spatial index is a two dimensional grid system. One or three grids, each having a specific cell size, are created with a layer. The first (lowest) grid level has the smallest cell size. Higher grids have a cell size at least three times larger than the prior level. Table S<layer\_id> is used

to store the index information. Each row consists out of a geometry ID and the grid cell number it occupies.

SDE manages the grid index system itself. The user can only fine tune the number of grid layers and the cell size. All other functions are performed by SDE. For example, expanding the grid due to the insertion of geometry's outside the grid area is managed by SDE. However, performance is highly dependent from the users influence upon grid parameters and if there are many geometric mutations the user has to tune the index again.

The spatial index table can be used with standard SQL for fast location of geometry's in an area of interest. For computing actual geometry's meeting the user spatial search criteria a "two-tier" search algorithm is used. First a potential superset of geometry's is retrieved by denoting all geometry's which fall within the grid cell's according to the area of interest. Secondly actual geometry's are determined by spatial relation computation. This in it self is also a two step operation. First the bounding envelope box of each geometry is compared with the bounding envelope of the compare geometry before actual co-ordinates are retrieved and interactions are calculated.

In SDE the user does not have to make data query's in which spatial indexing and retrieval boost are incorporated. SDE resolves this.



### 3.2.4 Data manipulation

Since the spatial component is stored in binary fields that cannot be processed by SQL, the only possibility to insert a geometry is the SDE insert function set.

SDE also provides an extensive set of functions to change the morphology of a geometry.

SDE provides functions to discard (part of) a geometry from the SDE related tables.

### 3.2.5 Data retrieval

#### 3.2.5.1 Spatial operators

The relational operators in SDE consists out of a set of “**spatial comparison operators**”. The relational operators return Boolean values as well as geometric results. The next set is incorporated:

1. **Distance**. Returns the closest distance between geometry *a* and *b*,
2. **Difference**. Returns the logical AND NOT geometry of geometry *a* and *b*,
3. **Intersect**. Returns the logical AND geometry of geometry *a* and *b*,
4. **Clip**. Returns the logical AND geometry of geometry *a* with a supplied envelop rectangle,
5. **Containing**. Returns TRUE if geometry *b* is entirely within geometry *a* and the boundaries do not touch,
6. **Crossing**. Returns TRUE if geometry *a* and *b* overlap and their boundaries intersect,

7. **Disjoint.** Returns TRUE if geometry *a* and *b* are not spatially related,
8. **Equal.** Returns TRUE if geometry *a* and *b* at every boundary location are the same,
9. **Overlapping.** Returns TRUE if geometry *a* and *b* overlap but their boundaries do not interact,
10. **Touching.** Returns TRUE if geometry *a* and *b* share a common boundary point or segment, but no interior points or segments,
11. **Within.** Returns TRUE if geometry *a* is entirely within geometry *b* and the boundaries do not touch,
12. **Overlay.** Overlays two geometry's,
13. **Symmetrical difference.** Returns the logical XOR geometry of geometry *a* and *b*,
14. **Union.** Returns the logical OR geometry of geometry *a* and *b*.

The function SE\_shape\_find\_relation returns a mask defining all spatial relationships between the supplied geometry's.

It is remarked here that there are no “direction arguments”. Topological relations like “Next link” and “Left side and Right side” are absent.

“**Geometric calculation operators**” in SDE cover a wide range including Boolean characteristics requests, as well as morphology characteristics. The most important in SDE are: area, length, extent, co-ordinate system, number of parts, number of points, all points, all co-ordinates.

A “**buffer operator**” is present to create a buffer around a (set of) geometry('s).

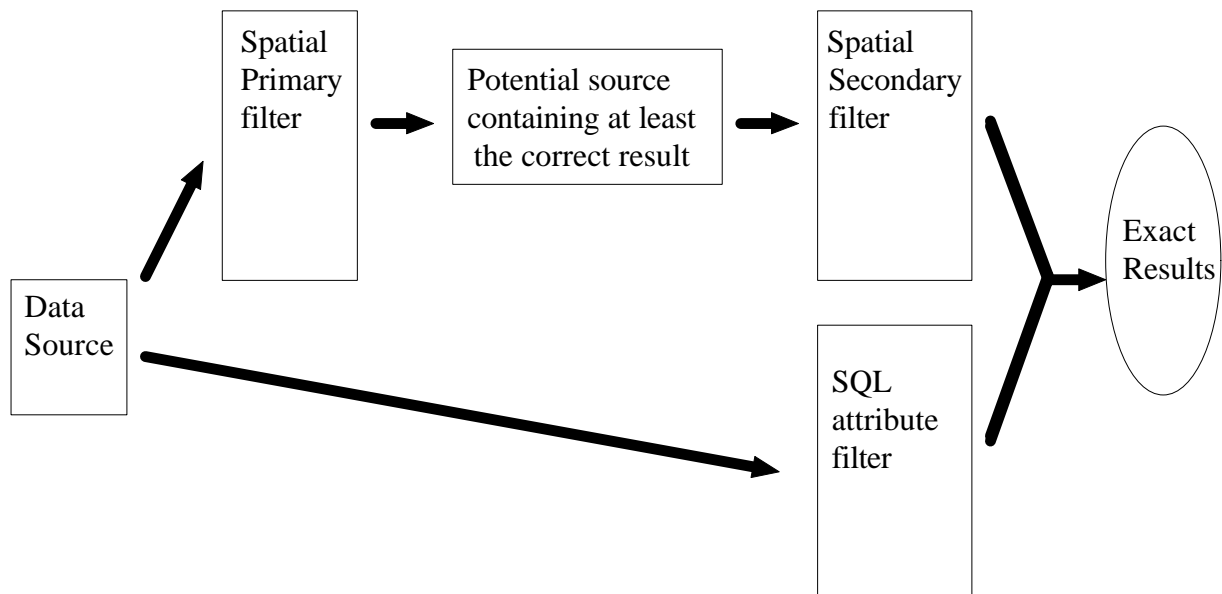
### 3.2.5.2 The Query Model concept

In SDE the feature retrieval and spatial operations mechanism is based on a dual input component execution. There is one input component dealing with attribute based selection and one component dealing with the spatial operations and selections.

The two components are referred to as the **Spatial stream filter** and the **Attribute stream filter**.

When the **spatial stream filter** and **attribute stream filter** are declared and set a “commit” is issued and the system starts to execute the stream job. To the user it is unknown how and in which order all parts of the filters are executed. However, the user can direct SDE to let the attribute constraint be executed first and the spatial constraint in second place or visa versa. In that case the user must have knowledge about performance induced by order of reduction of the potential candidates to the selected set. Figure 17 depicts the retrieval model.

Figure 17: The SDE two tier/two request retrieval model.



The primary filter permits fast selection of a limited number of candidate rows. The primary filter uses approximations in order to reduce computational complexity. The secondary filter applies exact geometry relation computation upon the result set of the primary filter.

### 3.2.5.3 The Query Model implementation

Executing a SDE stream filter results in a selected set of records in the table. In SDE this is called a “cursor”. A “cursor” provides a mechanism to iterate sequentially through the set of selected records.

In order to retrieve a set of selected records, derived from a combination of spatial and attribute selection, a request environment has to be established. This environment is also called a “stream”. A request can be made up out of a spatial retrieval constraint and/or an attribute constraint.

The attribute constraint is invoked by an SQL SELECT query. In the SDE API it takes the following form:

```
declare:
attrib[n]                = <output_attribute_column>          (or list)
Sqlc->tables[n]          = <the_user_spatial_enabled_table>   (or list)
Sqlc->where               = <where_clause>
```

set constraint:

```
perform: SE_stream_query(<stream_id>,<num_output_colmuns>,attrib,Sqlc)
        SE_stream_execute(<stream_id>)
```

The spatial constraint is invoked by allocating a set of geometry's that have to be related to an other set of geometry's using a specific method. For example relate all geometry's against a zoom window using the "pass through" method and select those which do (truth). In the SDE API it takes the following form:

```
declare:
Filter.table              = <the_user_spatial_enabled_table>
Filter.column             = <the_spatial-ID_column_in_the_table>
Filter.filter.shape       = <a_shape>
Filter.method              = <method_keyword>
Filter.truth              = <boolean_to_set_"must"_or_"must not"_satisfy_constraint>
Filter.filter_type        = SE_SHAPE_FILTER
```

set constraint:

```
perform: SE_stream_set_spatial_contraints(<stream_id>,<search_order_keyword>, ..., Filter)
        SE_stream_execute(<stream_id>)
```

In the above schema, in the Filter.filter.shape argument, one geometry is related to many others. It is also possible to make up a "many to many" relation by specifying a selected set (or table) in this argument.

## 3.3 Discussion

### 3.3.1 OpenGIS and the SDO implementation

Regarding data model, storage structure and data retrieval the next remarks can be made.

SDO has, according to OGC, a Multi element geometry support.

The SDO system has no implicit GEOMETRY\_COLUMNS meta table, nor a SPATIAL\_REFERENCE\_SYSTEM table. However, these tables can be implemented and populated by the user. Attention must be given here to the <layer>\_SDODIM table. This table holds the spatial reference method but is restricted to linear interpolation reference methods like “longitude-latitude” and “x-y”. At most the table holds the rectangle quadrant unit utilised to indicate co-ordinates and the dimension of the overall space in consideration.

OGC has not yet developed standards or rules in order to insert or accept data into the storage system. Partly this is understandable since each implementation requires its own insert parameters and rules. On the other hand it could be useful to have guidelines to the extent of which insert operations within the SQL environment are valid. OGC only specifies correct geometry elements but does not describe which instances of geometry's are still acceptable. In the case of SDO it is obvious that any kind of constellation of co-ordinates can be inserted. As a service SDO provides a function to determine whether a geometry is valid. Still, validation of geometry's is not an implicit integrity rule in the table. Future development of SDO may incorporate such facility.

With operations in the SDO environment arguments in the spatial functions can be passed. These arguments are not yet in compliance with the OGC textual representation of geometry's.

Typically, SDO has its own implementation of the spatial index mechanism as well as the one dimensional index mechanism for addressing table rows. OGC has no concepts developed yet, if they ever should be necessary. However, the specific implementation of SDO has great influence on the way data can be retrieved and how statements are compiled to make use of the indexing mechanisms. SDO has clearly developed its own exterior SQL query interface here.

As OGC has defined a set of spatial related operations SDO has a set of spatial relational operators. In Table 1 they are compared. SDO operators are a superset of the OGC definition.

Table 1: Comparison of spatial operators between the OGC concept and the SDO implementation.

OGC operation	SDO equivalent	Remark
Equal	Equal	
Disjoint	Disjoint	
Touch	Touch	In SDO restricted to 1 point at the boundary.
Cross	Overlapbdyintersect	In SDO also polygons may be examined (see OGC Overlap).
In	Contains Coverby Covers Inside Overlapbdydisjoint	In SDO several situations of “In” are defined.
Overlap	Overlapbdyintersect	Also see OGC Cross.
Contain	Contain	In SDO with the addition that boundaries do not touch.
Interact	Anyinteract	
Relate	Relate	

No “direction argument” is present to determine “next link” or “left and right side”.

In SDO there are functions to create a geometry. The two main reasons for their presence are the complex standard insert operations on consecutive table rows occupied by the geometry and the execution of an immediate spatial indexing update operation.

SDO does not incorporate geometry “morphology manipulation” functions nor “delete” functions. They are recommended since not only actions on the geometry table are involved but also actions to maintain a consistent spatial indexing mechanism. Executing standard SQL DELETE and UPDATE require insight of the user in the way data are stored in SDO. This is not in accordance with the initial nature of SQL. SQL is based on the principle of being a declarative language. SQL offers the user the concept of *what* the system should do not *how*



to do it. ORACLE stored procedures facility could be enhanced with functions to overcome this situation.

As an extension to the above the query model requires knowledge about the indexing mechanism from the user and how to utilise its capabilities in retrieving data. This is not in accordance with the initial nature of SQL. Although it is simple to activate the primary filter some knowledge on the indexing mechanism is required.

The implementation of SDO reveals that the package is entirely build on the bases of normalised tables and SQL as query interface. This is a major aspect of SDO.

In general it can be stated that, with remarks, SDO is an implementation of the OGC concepts.

### **3.3.2 OpenGIS and the SDE implementation**

Regarding geometry types SDE only supports Nil, Point, Multipoint, LineString, MultiLineString, Polygon and MultiPolygon. OGC has not defined a “Nil” geometry, a geometry that is not specified nor loaded. On all other mentioned geometry types all requirements posed on these elements by OGC are implemented. “Multi element identical type geometry’s” are supported by SDE. Multi geometry type geometry collections are not supported by SDE.

The SDE system has no implicit GEOMETRY\_COLUMNS meta table, nor a SPATIAL\_REFERENCE\_SYSTEM table. Data in these columns are stored in the SDE table LAYERS.

OGC has not yet developed standards or rules in order to insert or accept data into the storage system. In SDE an extensive set of insert, manipulate and delete functions are implemented. SDE incorporates a function that validates a geometry as being acceptable when an attempt is made to insert the geometry into the SDE system.

Typically, SDE has its own implementation of the grid oriented spatial index mechanism as well as the one dimensional index mechanism for addressing table rows. OGC has no concepts developed on spatial indexing.

OGC has not (yet) addressed the concepts of an API interface. ESRI has implemented its own concepts here.

As OGC has defined a set of spatial related operators SDE has a set of spatial relational operators. Moreover SDE has a set of spatial “addition and subtraction” functions. In Table 2 they are compared. SDE operators are a superset of the OGC definition.

Table 2: Comparison of spatial operators between the OGC concept and the SDE implementation.

OGC operation	SDE equivalent	Remark
Equal	Equal	
Disjoint	Disjoint	
Touch	Touch	In SDE interiors may not touch. In the Line/Point case the point may only be at an end of the line.
Cross	Crossing	In SDE not for point geometry's. 2 lines with a common end do not cross.
In	Within	In SDE inside touching is allowed.
Overlap	Overlapping	In SDE the resultant geometry's may not be Nil and must be of different morphology than both input geometry's. Only for Polygon/Polygon, Line/Line and Multipoint/Multipoint
Contain	Containing	In SDE the complement of "Within" where touching is allowed.
Interact	Overlay	In SDE a geometry set is returned.
Relate	Overlay	In SDE a geometry set is returned.
	Distance Difference Intersect Clip Overlay Symmetrical-difference Union	Spatial "addition" and "subtraction". In OGC not specified.

No "direction argument" is present to determine "next link" or "left and right side".

SDE has no SQL based spatial query interface. Attribute selection is based on SQL query, but the spatial selection is a typical SDE concept and implementation. Knowledge by the user on the spatial index mechanism can help in defining SQL query's for approximation selection.

The implementation of SDE reveals that the package is build on the concept of normalised tables with an obligatory use of an application program routine set. With SDE ESRI has

implemented its own standard interaction between data storage and retrieval. User programs have to be created (bought) according to this specification to interact with the SDE system.

In general it can be stated that OGC concepts are integrated in SDE. There are significant developments which OGC has not yet addressed, like third dimension and “Measure” (z-value or attributes linked to a co-ordinate).

### 3.3.3 SDO and SDE compared

Using OGC as a reference SDO and SDE can be compared. Within the setting of this study the aspects of the Data Model, Storage Structure, Spatial indexing and Data Retrieval are presented in the next set of tables.

Regarding the Data Models there is a set of common and different implementation aspects.

Table 3: Data Model

Aspect	OGC	SDO	SDE
Full range of Multi-element geometry's	Yes	No	No
Node features	No	No	Indirectly using “Measure”
Unique Id reference between spatial and attribute data	Yes	Yes	Yes
Collections of geometry's are stored as layers	Yes	Yes	Yes
Intrinsic topological relationship	No	No	No
number of dimensions	2	2	3
Annotation	No	to be user implemented	Yes
Multi dimensional indexing technique	No	Yes	No

Regarding the storage structure there are major differences between SDE and SDO.

Table 4: Storage structure

Aspect	OGC	SDO	SDE
“set theory” normalised tables	Yes	Yes	Yes
Geometry’s stored in one table	Yes	Yes	Yes
Spatial index in one table	not defined	Yes	Yes
Layer related data and user extendible	Yes	Yes	Yes
Ordinates as column values	Yes	Yes	No
Ordinates as binary strings in a column	Yes	No	Yes
Geometry storage in one record	Yes	No	Yes
Geometry storage in more than one record	Yes	Yes	No
Bounding rectangle for each geometry	Yes	No	Yes
Ordinate physical storage		real values and a comparison tolerance	positive integer values with an offset and scale

The indexing technique of both packages has a conceptual extreme different approach. In the context of this study only a few aspects, that are relevant to data retrieval, are brought forward. It is remarked here that the performance of the spatial index highly depends on knowledge by forehand of the user of the entire (potential) spatial data set to be indexed. OGC conceptually has not made any efforts regarding spatial indexing.

Table 5: Spatial indexing

Aspect	OGC	SDO	SDE
Method		Quadtree	Grid
Decomposition of space by uniform rectangles		Yes	Yes
Rectangle identification values transformed for one-dimensional tables indexing		Yes	Yes
Indexing technique suitable for a “two-tier” retrieval schema *		Yes	Yes
Each tier can be used for retrieval		Yes	Yes
User knowledge on the indexing schema required		Yes	Yes
Automatic adjustment of spatial index		No	Partly

- The first “tier” supplies a superset of possible elements containing at least the correct answers. The second “tier” calculates exact results.

Data manipulation in SDO can completely be executed by using SQL. Due to the storage of every ordinate in a column, standard SQL is sufficient to modify a geometry.

In SDE manipulation of a geometry can only be performed using the routines available from the API. However, the number of functions available is extensive and covers a wide range of spatial manipulation actions.

Spatial data retrieval is composed out of spatial operation functions and spatial query grammar.

The spatial operators in this study are categorised into operators regarding calculations, comparison and buffers.

Table 6: Data Retrieval

Aspect	OGC	SDO	SDE
Method		SQL	API
Calculation operators	Yes	No	Yes
Comparison operators with Boolean result	Yes	Yes	Yes
Comparison operators with geometry result	No	No	Yes
“Addition” and “subtraction” of geometry’s	No	No	Yes
Boundary direction	No	No	No
Buffer operators	Yes	No	Yes
full SQL interface*		Yes	No

\* The spatial query grammar in SDO is entirely imbedded in SQL. It takes the form:

“Select ... from ... where ... <spatial condition>...<alphanumeric condition>;”

The spatial condition is based on Boolean syntax.

The spatial query grammar in SDE consists out of specifying the desired operation and its parameters in a set of variables and then passing then on to be executed. The syntax and grammar are a specific SDE implementation on the bases of a proprietary ESRI specified API.

### 3.4 Summary

In this chapter the OpenGIS concepts according to OGC are used as a reference to examine the SDO and SDE packages. This examination has the following set of results:

1. Simple and non simple geometry's in SDO and SDE can be incorporated. In this respect SDO and SDE are a superset of OGC definitions. However, not all OGC defined non simple geometry's are package valid.
2. The data storage architecture and implementation of SDO and SDE are, with the exception of some details, according to OGC rules and standards.
3. Retrieval of data in SDO and SDE is highly bound to the concepts of the developers. In SDO this is a full SQL integrated environment. In SDE it is a defined API interface.
4. Comparison operations in both packages return Boolean values as defined by OGC. SDE is capable of returning resulting geometry's from a comparison operation.

Retrieval is an aspect OGC has to address. It should cover the SQL and API interface. The first step in retrieval is SQL3 in which spatial data types are defined. The next step would be to resolve SQL and API grammar and syntax.

In it self storage is not a user interest unless it requires knowledge of the user in order to formulate a retrieval statement.

Maintenance of correct spatial indexing is still in the hands of the user. In both packages the user is obligated to address spatial indexing for swift spatial search. Developers are asked to automate the spatial indexing mechanisms as much as possible.

SDO and SDE differ greatly in their set of geometry validation rules. This means that both systems will react differently when data are inserted. Inserting the same set of data in both packages might result in different reactions of the system. It can be concluded here that this is unacceptable to the user. It requires a more in depth analysis of both package behaviours.

At the outset of this study there is the wish to incorporate the Dutch National Topographic Map “TOP10vector” into a RDBMS. In the next chapter special occurrences of geometry’s of this set are inserted and tested in SDO and SDE.



## **4. Implementing geodata in SDO and SDE: a case study**

### **4.1 Geometry morphology's from the Dutch National Topographic Map**

In order to determine the morphology and characteristics of geometry's, as they emerge in every day practise, a representative geodata set of large volume is visited. In this case a data set which is intensively used by the Dutch GIS community is the Dutch National Topographic Map (TOP10vector) scale 1 : 10,000. The data set occupies 7 Gbyte of disk space and contains over 15 million polygons, 75 million separate lines and 1.5 million points.

The Dienst Landelijk Gebied (Government Service for Land and Water Management) converted TOP10vector from the Intergraph computer platform at the Dutch Topographic Survey Institute to ARC/INFO format ( Cattenstart, 1992). The data were stored in correct topological relationship ( all datasets were ARC/INFO "CLEANed"). This action resulted in a set of geometry's according to the ARC/INFO concept of valid geometry's.

The TOP10vector originates from the Dutch Topographic Survey and is compiled within the Intergraph system. The data are physically and conceptually converted from Intergraph to ARC/INFO format. Before ARC/INFO topology was created the data were visited in order to detect geometry morphology's as the map was compiled at the Dutch Topographic Survey. It should be mentioned here that at that point the data are in compliance with the rules for valid geometry morphology's within the Intergraph system. However, as the data were visited it became clear that Intergraph validation rules are extremely open and leave room for many occurrences of geometry morphology's.

A program filter, in conjunction with visual inspection, was used to find specific occurrences of geometry morphology's. Apart from OGC valid simple morphology's, as described in section 2.3, a set of morphology's were derived as shown in Table 7. Based on the method of "strong inference" (Platt, 1964) these morphology's were inserted and tested in SDO and SDE whether they are valid or not.

In Table 7 the morphology's of geometry's that are tested are presented. In the SDO and SDE column the package reaction is denoted:

1. **Accepted** The morphology is accepted as presented,
2. **Modified** The morphology receives a co-ordinate modification. This can be discarding, addition or movement,
3. **Converted** The morphology is split, joined or reclassified into an other (set of) geometry type(s),
4. **Rejected** The morphology was not accepted,
5. **Unknown** Due to imperfection of the checking routines the acceptance is unknown.



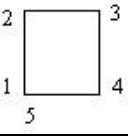
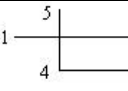
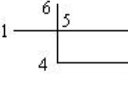
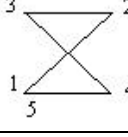
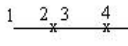
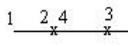
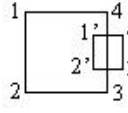
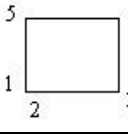
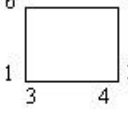

It is remarked here that in the case of modification not only the morphology's shape is changed but also related information might be lost. In SDE this might be a "Measure", in SDO a foreign key to other data which is coupled to the co-ordinate column.

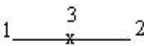
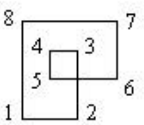
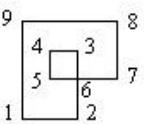
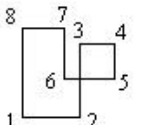
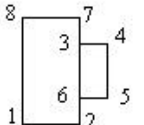
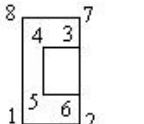
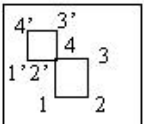
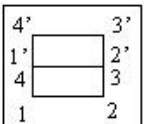
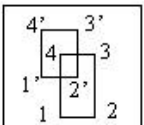
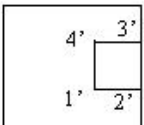
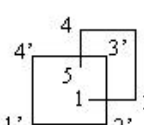
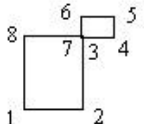
In Table 7 in the validation columns a code is provided referring to the next set of remarks:

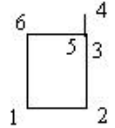
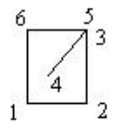
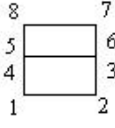
- |   |  |
|---|--|
| 1 | 0 = Zero length segment,                                       |
| 2 | 1 = LineString or Poly boundary is self-intersecting,          |
| 3 | 2 = Polygon does not close properly,                           |
| 4 | 3 = The number of points is less than required for geometry,   |
| 5 | 4 = Polygon patch has no area,                                 |
| 6 | 5 = Co-ordinate discarded or added without loss of morphology. |

Due to imperfections in the SDO\_VALIDATE\_GEOMETRY routine the morphology's marked with a asterisk ( \* ) are assessed using a mix of contra checking morphology's and the RELATE function. These results should be validated as "best estimations" of geometry validity in SDO.

Table 7: Special geometry morphology's and there validation in SDO and SDE.

#	Geometry type	Morphology	Origin	OGC type	SDO valid	SDE valid
1	Point		2 geometry's at the same location. Overlapping geometry's.	Multi point	Accepted	Accepted
2	Line		Zero length line. Digitising error? Resolution error?	Simple	Accepted	Rejected code: 0
3	Line		Closed line.	Simple	Accepted	Accepted
4	Line		Topological cross. No co-ordinate at the crossing	Non simple	Accepted	Accepted
5	Line		Topological cross with co-ordinate at the crossing	Non simple	Accepted	Accepted
6	Line		Crossing line with interior.	Non simple	Accepted	Accepted
7	Line		Zero length segment Digitising error ?	Simple	Accepted	Modified code: 0
8	Line		Reverse direction Resolution error?	Non simple	Accepted	Accepted
9	Polygon		Topological Overlap	Multi polygon	Rejected <sup>*</sup> code: 1	Rejected code: 1
10	Polygon		Polygon with zero length segment.	Simple	Accepted <sup>*</sup>	Modified code: 0
11	Polygon		Overlapping segments	Non simple	Rejected <sup>*</sup> code: 1	Rejected code: 2
12	Polygon		Zero perimeter. Resolution error?	Simple	Unknown	Rejected code: 3

13	Polygon		Zero area. Resolution error?	Simple	Unknown	Rejected code: 4
14	Polygon		Boundary inward loop with no co-ordinate at the crossing	Non simple	Rejected <sup>*</sup> code: 1	Rejected code: 1
15	Polygon		Touching inside island.	Non simple	Rejected <sup>*</sup> code: 1	Rejected code: 1
16	Polygon		Boundary outward loop with no co-ordinate at the crossing	Non simple	Rejected <sup>*</sup> code: 1	Rejected code: 1
17	Polygon		Overlapping outward segment.	Non simple	Rejected <sup>*</sup> code: 1	Rejected code: 1
18	Polygon		Inward overlapping segment.	Non simple	Rejected <sup>*</sup> code: 1	Rejected code: 1
19	Polygon		Touching islands	Simple	Accepted <sup>*</sup>	Converted Modified code: 5
20	Polygon		Islands sharing a segment boundary.	Simple	Accepted <sup>*</sup>	Converted Modified code: 5
21	Polygon		Islands overlapping.	Non simple	Rejected <sup>*</sup> code: 1	Rejected code: 1
22	Polygon		Island sharing boundary segment.	Non simple	Unknown	Rejected code: 1
23	Polygon		Unclosed overlap. Digitising error?	Multi polygon	Rejected <sup>*</sup> code: 2	Rejected code: 2
24	Polygon		Crossing outer boundary with co-ordinates at the crossing.	Multi polygon	Accepted <sup>*</sup>	Modified code: 5

25	Polygon		Outer boundary with overlapping segments outside	Non simple	Rejected* code: 1	Rejected code: 2
26	Polygon		Outer boundary with overlapping segments inside	Non simple	Rejected* code: 1	Rejected code: 2
27	Polygon		Outer boundary segments split polygon.	Multi polygon	Rejected* code: 1	Rejected code: 2

## 4.2 Discussion

The presented geometry morphology's in Table 7 indicate that their origins are related to:

1. The actual observations "in the field",
2. Transformation errors from observation to computer held data,
3. Computer intrinsic storage errors, like resolution.

In all cases morphology's emerge that have to be stored properly. As a critic it is stated here that it is not to the system to discard geometry morphology's simply because the system is not capable of coping with these "anomaly's". It is also stated here that in every case where morphology's are automatically modified, decomposed, converted or rejected the system should send out a warning indicating that the system is not capable dealing with the specific geometry morphology. It should be the user, with knowledge of the systems limitations, to decide what to do in such cases (and may instruct the system to deal with it giving the systems rules, capability's and limitations).

Regarding Point geometry's there are no differences between OGC, SDO and SDE.

Regarding Line type geometry's the next set of observations are made and presented in Table 8.

Table 8: Acceptance of Line type geometry's

Aspect	OGC	SDO	SDE
Zero length line	Simple	Yes	No
Zero length line segment	Simple	Yes	No
Self crossing	Non Simple	Yes	No

Regarding Polygon type geometry's the next set of observations are made and presented in Table 9.

Table 9: Acceptance of Polygon type geometry's

Aspect	OGC	SDO	SDE
Island overlap	Multi polygon	No	No
Zero length boundary segment	Simple	Yes	No
Zero perimeter	Simple	?	No
Zero area	Simple	?	No
Boundary cross	Non simple	No	No
Boundary overlap	Non simple	No	No
Touching islands	Multi polygon	Yes	No

In SDE the 'internals' of islands disappear, since in the concept of SDE the internals of 'holes' are of no concern to the geometry under consideration (see Table 7, occurrence 20). A (set of) hole(s) with internals should be treated as a new geometry. Further, SDE does not allow for simulated dangle line segments in a Polygon geometry (see Table 7, occurrence 25 & 26). Zero length line segments result in deletion of a co-ordinate implying loss of a potential measure (see Table 7, occurrence 10).



Table 7 is evidence to the fact that SDE and SDO are a superset of the OGC simple geometry implementation concept. The question now arises why OGC distinguishes a difference between simple geometry morphology's and others? It is obvious that the commercial packages anticipate every day practical occurrences and implementations of spatial phenomena. A more in depth OpenGIS research is recommended here.

On the other hand, to operate properly SDO and SDE do not accept all types of non simple geometry's. They are either not accepted, are modified, or decomposed to simple geometry's. To the user it is important to understand this modification and decomposition necessity since a phenomenon or entity in the users view is "torn" into several parts. The user must be aware that when she/he retrieves the phenomenon or entity a set of (modified) geometry's is returned. And this set of geometry's can be any combination of Point, LineString and Polygon geometry's in any quantity, which have to be retrieved in a correct way. On improper retrieval the user may think that his stored entities are corrupted, or even worse, that they do not longer exist in the database.

The most important issue is that both SDO and SDE have different sets of validation rules. This results in different implementations of the same dataset into the packages. This will lead to problems when two datasets from different packages are integrated. To the user this is an most unwanted situation. It is recommended to investigate the issue of converting non simple to simple geometry's and define in depth rules for valid geometry's.

### 4.3 Summary

From a common dataset OGC specified simple and non simple geometry's were derived and tested in SDE and SDO. From this test it can be concluded that SDE and SDO are a superset of OGC specifications, dealing with a wider range of geometry morphology's than OGC has marked "simple".

Due to different sets of validation rules SDO and SDE deal differently with special occurrences of non simple geometry's. This leads to different implementations of the same dataset into both packages. From a users point of view this is a most unwanted situation. It is recommended that OGC investigates the conversion of non simple to simple geometry's and define in more depth the rules for valid simple geometry's.

In some cases geometry's are modified, meaning that co-ordinates are deleted, moved or added. This could lead to loss of information when attribute data are related to the specific co-ordinate. In other cases geometry's are decomposed to a set of other simple geometry's. The user should be aware of this decomposition since her/his conception of a world phenomena is stored differently in computer held data.

## **5. Conclusions**

### **5.1 OGC**

The Open GIS Consortium has managed to bring users and developers together to exchange concepts upon the development of OpenGIS. OGC has produced concepts, guidelines and rules on GIS inter-operability regarding data exchange and program interaction. In this respect computer held representations of world phenomena are addressed and standards on spatial data retrieval are put forward. OGC has made a major contribution to “Open interoperability” in the GIS world.

At the outset of this study three major aspects of OpenGIS were taken into consideration:

1. The feasibility of OGC defined morphology's of spatial geometry's,
2. A uniform data storage environment,
3. A uniform language interface.

The first aspect addresses the Geodata Model, the second the Storage Architecture and the third the Spatial Operators and Retrieval Facility. Conclusions on these aspects are given in the next set of subparagraphs.

### 5.1.1 Geodata Model

In the vector representation world phenomena are captured in geometry's. A geometry is a representation that can be made up as a Point, Line, Polygon or combinations of them. OGC has categorised geometry's in "simple" and "non simple" morphology's (par. 2.3.1). From this study it emerges that simple and non simple geometry's arise in every day practise (par. 4.1). It is therefore not clear why OGC has made this distinction. However, the research on storage of morphology's shows that to operate properly, systems tend to convert non simple morphology's into simple ones, or do not accept the non simple morphology (par. 4.1). In the first case the user is confronted with the fact that a logical unit in her/his view of the world is broken up into more than one unit. In the second case the user is left behind with entity's that are rejected and seem not operable in a computer environment. This could lead to a more object based approach of world phenomena in computer held data.

Not addressed by OGC is the observation of the feature Node (par. 2.4.3). A Node is an intermediate location on a boundary ( of a line or polygon ) that can have its own set of attributes.

Also not addressed by OGC is dealing with non simple classified geometry's. In the SDO and SDE package this is a proprietary solution alike the rules to validate or convert geometry's. This leads to different implementations of data sets ( par. 4.1). Exchange of data between packages will be subject to discrepancy. An unwanted user situation. It could be stated that standards on data exchange might be of higher importance than standards on storage architectures.

### 5.1.2 OGC storage Architecture

Geometry's have to be physically stored into the computer. In this study the storage in the SQL92 environment is investigated. Regarding Relational databases with no geometric feature types OGC supports two storage structures; numeric and binary (par. 2.3.2). From the users point of view this is confusing. The user only wants the data to be in computer held format and retrieve them upon request. The developers point of view is to preserve commercial interests on investments in storage structures and retrieval mechanisms already made in GIS packages that are brought onto the market. The political influence of the developers on OGC is evident here. It reveals that OGC is not entirely free in developing concepts and standards in an unbound fashion.

OGC states that they do not propagate a mandatory type of data storage architecture (par. 2.4.4) since it is more essential that retrieval and spatial operations should be uniform and as much declarative as possible. Still two types of storage structures are approved. The basic difference between the two types is that in the one every data element can be addressed by SQL and in the other only by a typical set of program routines.

The OGC concept of storing geometry's is object based. Therefore, both storage architectures lack intrinsic topology, meaning that every co-ordinate is stored only ones in the storage system. Relational referencing is accomplished in the case of allocating a specific (co-ordinate) datum to one or more geometry's. In the OGC concept geometry's are stored as independent features having all spatial property's stored within the geometry declaration in the database. This leads to duplication of data and unrelated duplicate data in the case of common/shared geometric parts.

### 5.1.3 OGC spatial operations and retrieval

Spatial operators can be categorised into “Geometric calculation”, “Spatial comparison” and “Buffer” operators. They are addressed by OGC. Regarding “Spatial comparison” only Boolean results are returned. In practice the resulting geometry is of equal importance. “Direction arguments”, in order to determine what is to “the left or right”, or the “next link”, in the “comparison operations” are not (yet) part of the OGC concept (par. 2.3.5).

The OGC storage concept bears no “intrinsic topology”. “Topological” operators now require computational activities each time spatial relations are required.

Standards on retrieval of spatial data is not yet addressed by OGC (par. 2.3.5). Until now (first half 1998) OGC has only specified operators upon spatial data. There is lack of a uniform declarative language grammar based on RDBMS's. Syntax, semantics and logic of a user interface are until now subject to developers concepts and implementations. The development of a full spatial integrated language, where SQL3 might be a first step, is of great importance to the user community.

Important issues to be addressed in a declarative language are “data manipulation functions”, “selected set output” and “spatial operators”.

Regarding the OGC defined storage structure in Binary Large Objects (BLOB) (par. 2.3.3) it is impossible to retrieve data from within the BLOB based upon a declarative language. The interface has to have knowledge on the storage structure within the binary field.

Spatial entity's have a temporal dimension. OGC has not (yet) addressed the issue of the temporal dimension in the approved storage structures.

## 5.2 OpenGIS in practise

From the three aspects of OpenGIS, under consideration in this study, the next set of actual questions were derived.

1. What are the differences between the OGC concepts on the geodata implementation in SDE and SDO:
  - A. Which OGC features are defined and (not) accepted in SDE and SDO,
  - B. Which OGC storage architecture is implemented in SDE and SDO,
  - C. Which OGC Spatial operations are defined and implemented in SDE and SDO,
  - D. What OGC data retrieval interface is defined and implemented in SDE and SDO,
2. What are the practical implications of spatial data in SDE and SDO.

Answers to these questions are presented in the next two subparagraphs on SDO and SDE.

### 5.2.1 SDO

In general SDO is an implementation of OGC concepts.

OGC geometry types are supported (par. 3.1.1). Non simple geometry's have to be decomposed to simple geometry's. Simple geometry's and multi part geometry's are supported. Geometry type Node is not incorporated in SDO.

The SDO storage structure is object based and is in accordance with OGC numeric storage architecture (par. 3.1.2). SDO uses, a users defined, fixed number of co-ordinate columns in the spatial table. This causes geometry's to be stored in multiple records and waste of space

due to the fact that not all geometry's have the same number of co-ordinates, leaving columns unused.

SDO spatial data manipulation functions (par. 3.1.4) are limited to insert support and (window) overlay select functions. SDO has no extensive set of manipulation functions. Data manipulation is to the user and is standard SQL oriented.

Spatial comparison operators are implemented as Boolean functions and cover the entire definition of OGC. SDO has no buffer functions.

SDO uses an ORACLE Corporation extended SQL for spatial operations in the user interface (par. 3.1.5).

Spatial indexing accelerates spatial search, but the user has to have knowledge on the index mechanism implementation to make effective use of it in SQL statements.

In it self storage is not a user interest unless it requires knowledge of the user in order to formulate a retrieval statement.

Maintenance of correct spatial indexing is still in the hands of the user. Every geometry spatial manipulation requires a user initiated update of the spatial indexing mechanism. For the Window functions this is performed automatically.

SDO has a limited set of geometry validation rules. Programs and systems interacting with SDO should be defined wide enough to deal with all SDO valid geometry's.

At the outset of this study there is the wish to incorporate the Dutch National Topographic Map "TOP10vector" into a RDBMS. Over 95 percent of the geometry's in this dataset are of



the simple geometry class and acceptable to SDO. Some geometry morphology's are not accepted by SDO and they need special attention. Dealing with this deflections with care, it can be stated that TOP10vector can be stored and retrieved using the SDO system.

### **5.2.2 SDE**

In general SDE is an implementation of OGC concepts.

OGC geometry types are supported (par. 3.2.1). Non simple geometry's have to be decomposed to simple geometry's. Simple geometry's and multi part geometry's are supported. Geometry type Node is incorporated as "Measure". Not only value's can be stored as a "measure" but reference identification codes to other attribute data are possible.

The SDE storage structure is object based and is in accordance with OGC binary storage architecture (par. 3.2.2). Each geometry is stored as one record in the spatial table. The geometry string of co-ordinates is loaded into a Binary Large Object column. To retrieve co-ordinates from the BLOB the internal "bookkeeping" of the byte string must be known. It is therefore impossible for SQL to retrieve single ordinate value's.

SDE has an extensive spatial data manipulation set. This is due to the fact that co-ordinates are stored in BLOB's and not directly SQL addressable by the user.

Spatial comparison operators are implemented as Boolean and as "resulting geometry" functions and cover the entire definition of OGC. It is a superset since SDE has additional functions or functions that can return more detailed answers than as defined by OGC.

SDE uses an ESRI defined API to programs interacting with SDE (par. 3.2.5).

Spatial indexing accelerates spatial search, but the user has to have knowledge on the index mechanism implementation to make effective use of it in SQL retrieval statements.

In it self storage is not a user interest unless it requires knowledge of the user in order to formulate a retrieval statement.

Maintenance of correct spatial indexing requires user initiation, but is further highly automated.

The user must have knowledge on the way the selected set is retrieved from the potential set since the user can determine whether the attribute reduction must be performed first or the spatial reduction. When much geometry manipulation has taken place the user must initiate the spatial indexing optimisation again.

SDE has an extensive set of geometry validation rules. Non simple geometry's are decomposed to simple geometry's. The user should be aware of this effect, since what is a logical unit in her/his view is now a composition of a set of geometry's.

With this set of validation rules it is possible to incorporate the Dutch National Topographic Map "TOP10vector" into SDE.

## 5.3 Future developments

### 5.3.1 OpenGIS

In the case of SQL92 with binary and numeric storage architectures there are several aspects for future research.

A more detailed study is recommended, which could provide knowledge on dealing with the Geodata Model, like feature Node, in order to be more complete.

The fact that non simple geometry's are decomposed to simple geometry's requires universal rules. In this way non simple geometry's are decomposed and resolved to computer held data in a uniform way and are identical in every implementation within commercial packages.

A major aspect recommended to be considered are standards and rules for geodata retrieval, both SQL and API based. Further, it is recommended that connections like JAVA and Active-X to geospatial extended RDBMS's become subject to closer research.

The object based approach of OGC might become the future object oriented approach. The first step in standards on retrieval could be SQL3 in which spatial data types are defined. The next step would be to resolve SQL and API grammar and syntax and other interface types.

With the set of spatial comparison operators the first steps on a library with spatial calculation algorithms is made. One step higher than this primitives is the development of integration

functions like dissolving, intersecting and unification. This development will be of major importance to the user because these functions enable the user to integrate all kinds of data sets to new meaningful data sets.

The issue of the temporal dimension of spatial data is subject to further development.

The development of a more uniform approach of spatial vector data in DBMS systems is clearly started. But there are also other spatial representations like raster data. The development of OpenGIS on this type of representation is yet to be started where OGC can play an important role. This task is demanding since the diversity of spatial data formats is substantial.

### **5.3.2 SDO**

The development of SDO is transformed to what is named “Spatial Cartridge” (NNa,1997), based on ORACLE version 8. Spatial Cartridge is build upon the concepts of SDO. The next set of developments are initiated by ORACLE Corporation:

1. The geometry types will be extended with mathematical functions (like spline, circle) in order to support analytical spatial algorithms like in stereometry,
2. co-ordinates to be stored in a new field called “vararray”. Vararray only occupies as much space as necessary to store data. The length can be from 0 to the theoretical maximum of the systems resources,
3. There will be an automatic administration on spatial indexing for SQL INSERT, UPDATE, and DELETE statements regarding spatial entity’s,

4. In the field of spatial analytical functions a “shortest distance between objects” function will be incorporated,
5. Internally there will be a query optimiser for faster system response,
6. There will be enhancements to support the “callout” of spatial data to foreign programs (like geohydrolic models) and receiving back results,
7. A simple spatial viewer to visualise spatial data will be available.

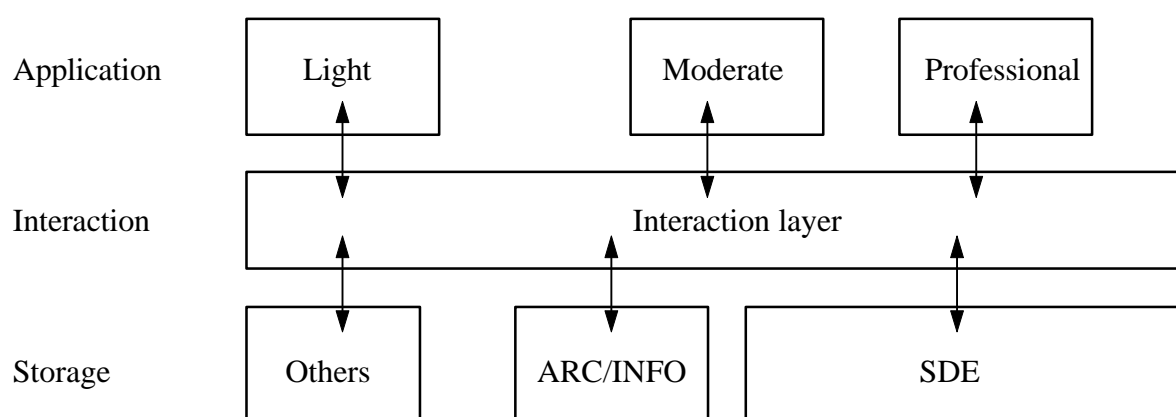
From this study two more issue's can be added to the list. It is recommended to extend the spatial operations facility with buffer functions and to extend the set of geometry validation rules in order to help develop standards on this topic.

All developments by ORACLE Corporation are based on the concept of who is best in a particular field of interest. ORACLE Corporation leaves spatial analyses and visualisation to developers in the field of GIS and CAD, where ORACLE Corporation makes efforts to store and retrieve spatial data in ORACLE RDBMS.

### 5.3.3 SDE

In the development concept of ESRI SDE will become a major basic structure on spatial data storage. Upon this storage structure sets of spatial functions can be “plugged in”. Figure 18 envisions this concept.

Figure 18: Possible development of the ESRI product line.



ESRI already has an extensive set of geospatial manipulation and integration functions. They are incorporated in several products, like ARC/INFO, Map object, ArcView and SDE. Based on a solid data storage structure an interaction layer, providing services to storage systems and services to many types of request systems, the user will no longer be concerned about many kinds of storage aspects. Retrieval will be of more importance and customised to the users needs and concepts.

ESRI calls SDE “open”, since it is possible for the user to write her/his own applications based on the SDE API. But this is still a proprietary environment. Research by ESRI could help supporting OGC in defining truly open systems.

Full automated spatial indexing is a field of ongoing development.

An SQL interface on top of SDE would boost the integration of SDE into RDBMS applications. In that respect it is mentioned here that SDE 3.0.2 supports the SDO storage structure. The integration of SDO and SDE is very meaningful to the user community, having an SQL interface and a full set of spatial operations.

## **5.4 Summary**

Apart from details, SDO and SDE can be approved as implementations of OGC OpenGIS standards and rules. Major differences between SDO and SDE are the numeric and binary storage structure, different sets of geometry validation rules and the SQL and API interface. Both SDO and SDE can be utilised to store geospatial data, like TOP10vector, into and RDBMS.

Still there are developments recommended, like standards on the SQL and API retrieval interface. In SDO and SDE they are proprietary implemented due to lack of standards which are not yet defined by OGC.

Differences in the geometry validation rules leads to different occurrences of the same dataset in both packages. This is a user unwanted situation.

When reviewing the conclusions and what is ahead in the world of GIS it can be stated that OGC has broken new ground in the development of OpenGIS and has made a major contribution. Developers understand the importance of open systems and make efforts to implement these ideas and concepts. Thus, the boundary's in GIS have been opened to new

exciting fields. But on the other hand it can be stated that not all work is done yet and that in every day use there are still boundaries in OpenGIS.



## Epilogue

We mostly use Geographic Information Systems for economic reasons. But it was Chief Seattle (Seattle, 1855) who pointed out that we do not own The Earth, nor physical nor spiritual.

*The only true harmony all creatures can experience is affection  
returned by Mother Nature with the gift of love and understanding.*

...every snowflake, every heartbeat...

## References

1. Cattenstart, G.C., 1992, Van IGDS naar ARC/INFO. Projectgroep Digitaal Topografisch Basisbestand. Ministry van Landbouw en Visserij, Nederland.
2. Cook, S. and Daniels, J., 1994, Designing Object Systems: Object-Oriented Modelling with Syntropy. Prentice Hall, London.
3. Egenhofer, M.J., 1992, Why not SQL? Int. J. GIS Vol. 6 No. 2.
4. Gadia, S.K., 1993, Parametric databases: seamless integration of spatial, temporal, belief, and ordinary data. Sigmod record, 22, 1 (Mar.).
5. Hettema, H.A., 1995, Naar een ruimtelijk SQL. NGT Geodesia 1995-II.
6. Land, P.D., 1997, Drawing the line between SQL and GIS. MSc thesis at the Manchester Metropolitan University.
7. NN, 1996, OpenGIS. Revision 0 of the Open GIS Consortium, Inc. Internet website [www.OpenGIS.org](http://www.OpenGIS.org)
8. NNa, 1997, Oracle Spatial Cartridge. An Oracle Technical White Paper. Oracle Corporation.
9. NNb, 1997, Oracle Spatial Data Option User's Guide and Reference release 7.3.3. Part No. A53264-02.
10. NNc, 1997, OpenGIS Simple Features Specification For SQL. Revision 0 of the OpenGIS Consortium, Inc. Internet website [www.OpenGIS.org](http://www.OpenGIS.org)
11. NNd, 1997, SDE version 3.0 Administrators Guide. ESRI Redlands.
12. Oosterom, P. van, & Vijlbrief, T., 1991, Building a GIS on top of the open DBMS "Postgres". Proceedings EGIS 1991.
13. Platt, J.R., 1964, Strong Inference, Science, 146, pp 347-352.
14. Samet, H., 1990, The Design and Analysis of Spatial Data Structures. ISBN 0-201-50255-0.
15. Samet, H., 1994, Spatial Data Models and Query Processing. Modern Database Systems, 1994.
16. Seattle, Chief of the Dwamish tribe, 1855, Address to the Government of the United States of America.  
Published in Dutch, 1993: "Hoe kun je de lucht bezitten?". ISBN 90 6224 198 0
17. Vijlbrief, T., & Oosterom, P. van, 1992, The GEO<sup>++</sup> System: an extensible GIS. Proceedings of the 5<sup>th</sup> International Symposium on Spatial Data Handling. International Geographical Union IGU.
18. Wildschut, A.N., 1998, Database Management Systemen en Geografische informatie. GIS Nieuws 1, 1998.

### The Point data type

A Point is a zero dimensional geometry and represents a single location in co-ordinate space.

The assertions for points (the rules that define valid points) are:

1. A point has for each dimension an ordinate value.

Special occurrences:

None.

### The MultiPoint data type

A MultiPoint is a zero dimensional geometric collection.

The assertions for Multipoint are:

1. The elements of a MultiPoint are restricted to Points,
2. The Points are not connected or ordered.

A MultiPoint is *simple* if no two Points in the MultiPoint are equal (have identical co-ordinate values).

Special occurrences:

None.

### The Curve data type

A curve is a one-dimensional geometric object stored as a sequence of points, with the subtype of curve specifying the form of the interpolation between points. The OGC specification defines only one subclass of curve, LineString, which uses linear interpolation between points. Topologically a curve is a one dimensional geometric object that is the homeomorphic image of a real, closed, interval:

$D = [a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$  under a mapping  $f: [a, b] \rightarrow \mathbb{R}^2$ .

The assertions for Curve are:

1. Each Curve has a start point and an end point,
2. The boundary of a Curve consists of its start point and end points,
3. The sequence of points determine the global morphology of the Curve,
4. A Curve is defined as topologically closed (= no gaps),
5. Each consecutive pair of points defines a line segment,
6. For each Curve there is only one algorithm determining the form of all line segments.

A Curve is *simple* if it does not pass through the same point twice:

$\forall c \in \text{Curve}, [a, b] = c.\text{Domain},$

$c.\text{IsSimple} \Leftrightarrow (\forall x_1, x_2 \in (a, b) \ x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)) \wedge (\forall x_1, x_2 \in [a, b] \ x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2))$

Special occurrences:

1. A Curve is closed if its start point is equal to its end point,
2. A Curve that is *simple* and closed is a Ring,
3. A LineString is a Curve with linear interpolation between points,
4. A LinearRing is a LineString that is both closed and simple,
5. A Line is a LineString with exactly two points.

## The MultiCurve data type

A MultiCurve is a one-dimensional Geometry Collection whose elements are Curves. MultiCurve is a non-instantiable class in the OGC specification, it defines a set of methods for its subclasses and is included for reasons of extensibility.

The assertions for MultiCurve are:

1. All elements of the MultiCurve are Curves,
2. A MultiCurve is defined as topologically closed.

A MultiCurve is *simple* if and only if all of its elements are *simple* and the only intersections between any two elements occur at points that are on the boundaries of both elements.

Special occurrences:

1. A MultiCurve is closed if all of its elements are closed,
2. A MultiLineString is a MultiCurve whose elements are LineStrings.

## The Surface data type

In planar two dimensional space a Surface is a planar two dimensional geometric object.

The assertions for Surface are:

1. A Surface is bounded by a set of Curves that enclose the Surface, the exterior boundary,
2. A Surface may have zero or more interior boundaries.

A Surface is *simple* when it consists out of a single ‘patch’ that is associated with one ‘exterior boundary’ and zero or more ‘interior’ boundaries.

Polyhedral surfaces are formed by ‘stitching’ together simple surfaces along their boundaries. The boundary of a simple surface is the set of closed curves corresponding to its ‘exterior’ and ‘interior’ boundaries.

The only instantiable subclass of Surface defined in the OGC specification is the Polygon. A Polygon is a planar surface, defined by one exterior boundary and zero or more interior boundaries. Each interior boundary defines a hole in the polygon.

The assertions for Polygon are:

1. Polygons are topologically closed,
2. The boundary of a polygon consists of a set of LinearRings that make up its exterior and interior boundaries,
3. No two rings in the boundary cross, the rings in the boundary of a polygon may intersect at a point but only as a tangent:

$$\forall P \in \text{Polygon}, \forall c1, c2 \in P.\text{Boundary}(), c1 \neq c2, \forall p, q \in \text{Point}, p, q \in c1, p \neq q, [p \in c2 \Rightarrow q \notin c2]$$

4. A Polygon may not have cut lines, spikes or punctures:

$$\forall P \in \text{Polygon}, P = \text{Closure}(\text{Interior}(P))$$

5. The Interior of every Polygon is a connected Point set,
6. The Exterior of a Polygon with one or more holes is not connected. Each hole defines a connected component of the Exterior.

The combination of 1 and 3 make a Polygon a Regular Closed point set. Polygons are simple geometry’s.

## The MultiSurface data type

A MultiSurface whose elements are Surfaces. They are made up from Polygons. A MultiPolygon is a MultiSurface whose elements are Polygons.

The assertions for Multipolygons are:

1. The interiors of two Polygons that are elements of a MultiPolygon may not intersect,

$$\forall M \in \text{MultiPolygon}, \forall Pi, Pj \in M.\text{Geometries}(), i \neq j, \text{Interior}(Pi) \cap \text{Interior}(Pj) = \emptyset$$

2. The Boundaries of any 2 Polygons that are elements of a MultiPolygon may not 'cross' and may touch at only a finite number of points. (Note that crossing is prevented by assertion 1 above),

$$\forall M \in \text{MultiPolygon}, \forall Pi, Pj \in M.\text{Geometries}(), \forall ci \in Pi.\text{Boundaries}(), cj \in Pj.\text{Boundaries}() \\ ci \cap cj = \{p1, \dots, pk \mid pi \in \text{Point}, 1 \leq i \leq k\}$$

3. A MultiPolygon is defined as topologically closed,
4. A MultiPolygon may not have cut lines, spikes or punctures, a MultiPolygon is a Regular, Closed point set:

$$\forall M \in \text{MultiPolygon}, M = \text{Closure}(\text{Interior}(M))$$

5. The interior of a MultiPolygon with more than one Polygon is not connected, the number of connected components of the interior of a MultiPolygon is equal to the number of Polygons in the MultiPolygon.

The boundary of a MultiPolygon is a set of closed curves corresponding to the boundaries of its element Polygons. Each curve in the boundary of the MultiPolygon is in the boundary of exactly one element Polygon, and every curve in the boundary of an element Polygon is in the boundary of the MultiPolygon.

## Appendix B      Storage in the SQL implementation Data Architecture

---

According to OGC geometry's can be stored either as numerals in ordinate columns (case 1) or as strings of ordinates in a binary byte stream (case 2).

### The numeric representation

The normalised geometry implementation defines fixed width SQL92 tables. Each primitive element in the geometry is distributed over some number of adjacent rows in the table ordered by a sequence number (SEQ), and identified by a primitive type reference (ETYPE). Each geometry identified by a key (GID), consists of a collection of elements numbered by an element sequence (ESEQ).

The columns in the Geometry table as in case 1 are defined as follows:

GID	NUMBER	NOT NULL
ESEQ	INTEGER	NOT NULL
ETYPE	INTEGER	NOT NULL
SEQ	INTEGER	NOT NULL
X1	<ordinate type>	
Y1	<ordinate type>	
...	<repeated for each ordinate, repeated for each point>	
X<max_ppr>	<ordinate type>	
Y<max_ppr>	<ordinate type>	
...		
<attribute name>	<attribute type>	
Constraint: Primary key = (GID,ESEQ,SEQ)		

MAX\_PPR is the maximum number of co-ordinates per row in the view.

The rules for geometric entity representation in the normalised SQL92 schema are defined as follows:

1. ETYPE designates the geometry type,
2. Geometry's may have multiple elements. The ESEQ value identifies the individual elements.
3. An element may be built up from multiple parts (rows). The rows and their proper sequence are identified by the SEQ value,
4. Polygons may contain holes, as described in the geometry object model,
5. Polygon rings must close when assembled from an ordered list of parts. The SEQ value designates the part order,
6. Co-ordinate pairs that are not used must be set to Nil in complete sets (both X and Y). This is the only way to identify the end of the list of co-ordinates,
7. For geometry's that continue onto an additional row (as defined by a constant element sequence number or ESEQ) the last point of one row is equal to the first point of the next,
8. There is no limit on the number of elements in the geometry, nor to the number of rows an element occupies.

### The binary representation

The binary geometry implementation uses the same GID as a key, but stores the geometry using the Well-known Binary Representation for Geometry (WKBGeometry). The geometry table includes the minimum bounding rectangle for the geometry as well as the WKBGeometry for the geometry. This permits construction of spatial indexes without accessing the actual geometry structure, if desired.

The columns in the Geometry table as in the case of WKBGEOMETRY are defined as follows:

GID	NUMBER	NOT NULL, PRIMARY KEY
XMIN	<ordinate type>	
YMIN	<ordinate type>	
XMAX	<ordinate type>	
YMAX	<ordinate type>	
WKB_GEOMETRY	VARBINARY	
<attribute name>	<attribute type>	

The WKB\_GEOMETRY is a variable length byte stream with a specific organisation. The organisation consists out of computer representation units and the basic storage blocks which can logically be joined to hold features and feature type identifications.

The computer representations are:

1. Byte. 1 byte holding 8 bits,
2. Uint32. 32 bit unsigned integer of 4 bytes,
3. Double. 64 bit double precision number of 8 bytes.

The order in which bits are processed is of great importance to read and write data correctly. However, for this study it would lead too far to consider read/write order of bits in a Byte. (Big and Little Endian). It is sufficient to note that the computer's implementation of read/write order should be in accordance with the way data are stored. It is simple to let "foreign" computer read and write Big and Little Endian stored data.

Important here is that integer values are stored as Unsigned integers, meaning that only nonnegative integer values in the range 0 to 4294967295 ( $2^{32}$ ) can be stored.

The basic building blocks are:

1. Point:
  - Double, x-ordinate
  - Double y-ordinate
2. LinearRing:
  - Uint32, numPoints (number of Points)
  - Point, numPoints
3. wkbGeometryType:
  - values: 1 = wkbPoint
  - 2 = wkbLineString
  - 3 = wkbPolygon
  - 4 = wkbMultiPoint
  - 5 = wkbMultiLineString
  - 6 = wkbMultiPolygon
  - 7 = wkbGeometryCollection
4. wkbByteOrder:
  - values: 0 = wkbXDR (Big Endian)
  - 1 = wkbNDR (Little Endian)

5. WKBPoint:

- Byte, ByteOrder
- UInt32, wkbType
- Point
- 6. WKBLineString:
  - Byte, ByteOrder
  - UInt32, wkbType
  - UInt32, numPoints
  - Point, points(num\_Points)
- 7. WKBPolygon
  - Byte, ByteOrder
  - UInt32, wkbType
  - UInt32, num\_Rings
  - LinearRing, rings(num\_Rings)
- 8. WKBMultiPoint
  - Byte, ByteOrder
  - UInt32, wkbType
  - UInt32, num\_wkbPoints
  - WKBPoint, WKBPoints(num\_wkbPoints)
- 9. WKBMultiLineString
  - Byte, ByteOrder
  - UInt32, wkbType
  - UInt32, num\_wkbLineStrings
  - WKBLineString, WKBPoints(num\_wkbLineStrings)
- 10. WKBMultiPolygon
  - Byte, ByteOrder
  - UInt32, wkbType
  - UInt32, num\_wkbPolygons
  - WKBPolygon, WKBPolygons(num\_wkbPolygons)
- 11. WKBGeometry:
  - Union { WKBPoint, WKBLineString, WKBPolygon, WKBGeometryCollection, WKBMultiPoint, WKBMultiLineString, WKBMultiPolygon }
- 12. WKBGeometryCollection:
  - Byte, ByteOrder
  - UInt32, wkbType
  - UInt32, num\_wkbGeometry's
  - WKBGeometry, wkbGeoemtry's(num\_wkbGeometry's)

The assertions for the Well-known Binary Representation for Geometry's are the same as described in paragraph 2.3.2. Additional assertions are:

1. Rings are simple and closed,
2. No two Rings in the boundary of a Polygon may cross each other. The linear Rings in the boundary of a polygon may intersect at most at a single point but only as a tangent,
3. The interiors of two polygons that are elements of a Multipolygon may not intersect,
4. The boundary's of any two polygons that are elements of a Multipolygon may touch at only a finite number of points.



### SDO

The SDO geometry validation rules are:

For points:

1. Must have a co-ordinate.

For LineStrings:

1. A LineString must have at least two co-ordinates.

For Polygons:

1. Must have at least three co-ordinates and must be closed.
2. No crossing of segments or duplicate co-ordinates may occur in the polygon boundaries.

### SDE

The SDE geometry validation rules are:

For points:

1. The area and length of a point are set to 0.0,
2. A single point's envelope is equal to the points ordinate values,
3. The envelope of a MultiPoint is set to the minimum bounding box.

For LineStrings:

1. Sequential duplicate co-ordinates are removed,
2. Each element must have at least two co-ordinates,
3. Each element may not cross itself. The start and end co-ordinate may be the same,
4. Elements may touch each other at there boundary (endpoints),
5. The length is the sum of all elements.

For Lines ( spaghetti elements):

1. Each element must have at least two distinct co-ordinates,
2. Sequential duplicate co-ordinates are removed,
3. The length is the sum of all elements.

For Polygons:

1. Sequential duplicate co-ordinates are removed,
2. The line segments should make up a closed ring,
3. There must be a counter clockwise direction in the co-ordinate sequence of the exterior boundary, a clockwise direction on interior boundary's,
4. Interior boundary's must be completely inside the outer boundary and may not touch the outer boundary,
5. Multiple interior boundary's that touch are combined into one interior boundary,
6. An interior boundary that touches the outer boundary is converted into an inversion of the outer boundary,
7. Interior boundary's crossing the outer boundary are eliminated but the inner part is converted to the outer boundary,

8. Elements in a Multipolygon may not overlap, but touching is allowed,
9. Elements with a common boundary are merged,
10. The perimeter is calculated over all boundary's,
11. The area is calculated over all boundary's ( the holes are not part of the calculation),
12. The envelope is set as the minimum bounding box of the entire geometry.

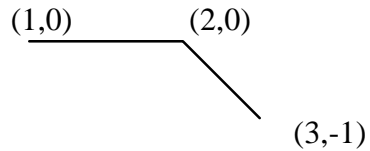
## Appendix D      Example of data storage in SDO and SDE

---

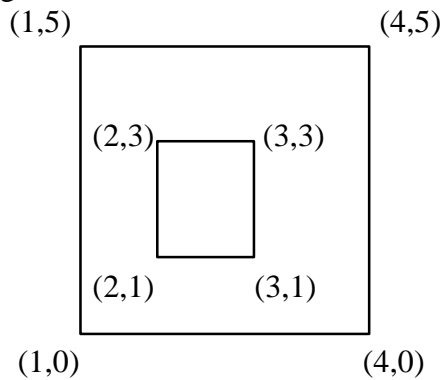
Only the essential storage characteristics are shown in this appendix.

Point      (2,1)    +

Line



Polygon



SDO geometry storage in SDO\_GEOM:

SDO_GID	SDO_ESEQ	SDO_ETYPE	SDO_SEQ	SDO_X1	SDO_Y1	SDO_X2	SDO_Y2
100	0	1	0	2	1		
101	0	2	0	1	0	2	0
101	0	2	1	2	0	3	-1
102	0	3	0	1	0	4	0
102	0	3	1	4	0	4	5
102	0	3	2	4	5	1	5
102	0	3	3	1	5	1	0
102	1	3	0	2	1	3	1
102	1	3	1	3	1	3	3
102	1	3	2	3	3	2	3
102	1	3	3	2	3	2	1

SDE geometry storage in F (only geometry and query important columns are shown):

Fid	entity	eminx	eminy	emaxx	emaxy	area	len	points (BLOB)
100	1	2	1	2	1	0	0	(2,1)
101	2	1	-1	3	0	0	2.4	(1,0),(2,0),(3,-1)
102	3	1	0	4	5	13	22	(1,0),(4,0),(4,5),(1,5)  (2,1),(2,3),(3,3),(3,1)

## Glossary

---

API	Application Program Interface
BLOB	Binary Large Object
CAD	Computer Aided Design
CORBA	Common Object Request Broker
Coverage	Set of data having the same characteristics
DBMS	Database Management System
DCE	Distributed Computer Environment
Element	Basic building block of a geometry or geometry collection
ESRI	Environmental Systems Research Institute
Feature	Specific occurrence of a geometry type
Geometry	A representation denoted as a point, line or polygon.
GIS	Geographical Information System
Line	Geometry represented as a set of connected co-ordinates
Node	Co-ordinate at a geometry boundary having attributes
ODBC	Open DataBase Connection
OGC	Open GIS Consortium
OLE	Object Linking and Embedding
OpenGIS	Interoperability of spatial related data and software
ORACLE	ORACLE Corporation
ORACLE	Relational Database Management System developed by ORACLE Corp.
Point	Geometry represented as a single co-ordinate
Polygon	Geometry represented as an area bounded by a closed line
RDBMS	Relational Database Management System
SDE	Spatial Data Engine (from ESRI)
SDO	Spatial Data Option (from ORACLE)
SQL	Structured Query Language
TOP10vector	TOPographic digital vector map scale 1:10.000 of the Netherlands
WKB	Well Known Binary