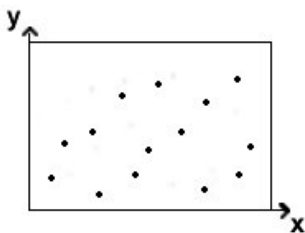
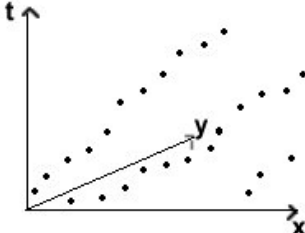
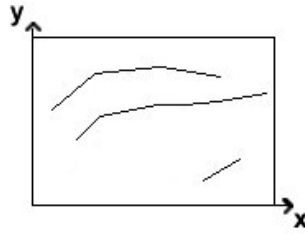
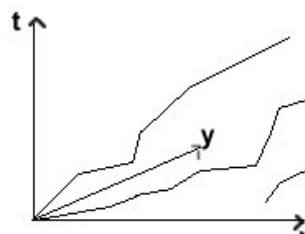


Moving objects in a geo-DBMS

Structuring, indexing, querying and visualizing moving point objects in a geo-DBMS context.

Marco Baars

	2D	3D
Points		
Lines		

MSc thesis
June 2004
Section GIS technology
OTB Research institute for housing and development studies
Delft University of Technology

Moving objects in a geo-DBMS
Structuring, indexing, querying and visualizing moving objects in a geo-DBMS.
Marco Baars
Delft, June 2004

MSc Thesis
Section GIS Technology
Faculty of Civil Engineering and Geosciences
OTB Research Institute for Housing and Development studies
Delft University of Technology



Preface

This thesis is the final result of my graduation assignment at the Delft University of Technology. This research has been done at the section GIS Technology of the OTB Research Centre for Housing and Development studies, which is a part of the Delft University of Technology.

For the readers of this report who are just interested in the final conclusions, I refer to chapter 7, where the conclusions and recommendations for further research are described. I would also kindly refer to the summary of this thesis. Also a summary in Dutch (Samenvatting) has been added. Some technical terms used in this thesis are briefly described in the glossary (appendix A).

Of course I would like to thank some people who assisted me in carrying out this research. First of all, I would like to thank my graduation professor Peter van Oosterom and my supervisors Edward Verbree and Ben Gorte for their useful input and comments. Also the Advise Office for Geo-Information and ICT (AGI), one of the parts of the Dutch Ministry of Traffic and Public Works, needs to be thanked for making the taxi data set available for this research. Because the complete section of GIS technology helped me with their technical knowledge, I also would like to thank them. At last I would like to mention the geodesy students, who supported me during my research and I especially would like to thank Thijs Brentjens, who helped me with many technical aspects and it was nice having company with him during the coffee and lunch breaks.

The subject of this thesis, moving object Database Management Systems, is a research topic that is very interesting and provides many challenges. Hopefully this thesis gives a useful input for research in the future on this topic.

Delft, June 2004.
Marco Baars

Summary

Spatiotemporal data is becoming more and more important. An example of a spatiotemporal data set is a moving point object data set. Cars, airplanes or pedestrians can be examples of such moving point objects.

A Database Management System (DBMS) has many advantages for storing large data sets in comparison to file-based approaches. The principle of the file-based approach is to make an application to operate on the data, which is stored in a data file. For every single application, a new program has to be developed. The main principle of the DBMS-approach is a data set in which the data is stored and a DBMS, which deals with the operations on the data. This DBMS takes care of the security of the data, keeps the data consistent and makes the data available for many different applications (interoperability). Spatial DBMSs like Oracle 9i Spatial have the extra advantage that they can handle spatial objects (for instance polygons or points) and can do spatial queries (find overlapping objects, calculate distances, etc.). In the case of moving point object data, it is worth to investigate whether a DBMS is useful or not.

The main question of this research is:

What is the potential and performance of a geo-DBMS to structure, index, query and visualize spatiotemporal point clouds of moving objects?

Some researchers have been developing spatiotemporal data structures. These structures have some disadvantages. For instance, the model made by Vazirgiannis and Wolfson is especially made for road networks and another data model (developed by Wolfson) is relatively complicated and can be used for objects that move freely in space like aircrafts. Some approaches have the disadvantage that they have a lot of redundant storage. To overcome these disadvantages, a new approach is introduced. This model could be used for every purpose that deals with moving point object data (this makes it generic) and it does not contain any redundant storage. An efficient indexing method makes querying of the data in a DBMS faster. For many query types, indexing methods are available. In moving point object cases, most of these indexing methods are based on the R-tree. Often it is not known in advance which queries are going to be done on a data set and which structure and which indexing methods are going to be chosen. So it needs to be investigated which indexing methods gives the fastest access to the data.

The main principle of this generic model is choosing a base table, from which, by using (materialized) views, three other data representations (based on different geometric data types) easily could be derived. In this way a set of four data representations is available for querying. These four data types are 2D points (x,y) , 3D points (x,y,t) , 2D lines $(x_i,y_i, x_{i+1},y_{i+1})$ and 3D lines $(x_i,y_i,t_i, x_{i+1},y_{i+1},t_{i+1})$. In the 2D representations time is regarded as an attribute. From this set, many queries could be formulated, like the object's speed, direction or acceleration. After the model is introduced, an efficient querying and indexing needs to be found. Because in Oracle 9i Spatial only the 2D and the 3D R-tree are implemented, the only way to manipulate the efficiency of accessing the data by the user is by formulating efficient queries.

To demonstrate that this generic model is fast and flexible, two case studies have been done. In the first case, the data has been collected in advance and being analyzed afterwards. So, the data set is static. Every 0.1 seconds, photos have been taken from a helicopter. The vehicles on the highway have been detected in multiple photos. So a data set with positions and times (photo numbers) is created and used for this case. The data has been successfully implemented in the

generic model and has been queried in multiple ways. Spatial querying in Oracle 9i Spatial is possible in two ways: with a spatial function or with a spatial operator. The main difference is that the spatial operator needs a spatial index and should be faster than the spatial function. Because this data set was very small, the query-optimizer decided not to use the spatial index, but doing a full table scan. So, for more complex queries, the access was not really fast, but for larger data sets, a faster access with the spatial index is expected.

The second case deals with real-time data. Because no true real-time data was available during the MSc-thesis project, a real-time simulation has been used to test the performance of the DBMS and the model. The data set contains GPS-tracking data from taxis driving in the surroundings of Rotterdam. Also for this real-time case, the model has been implemented successfully. Query times do not increase because of a continuously growing data set. The query times depend on an efficient way of formulating a query. Using the `SDO_FILTER` operator, which only uses the spatial index to check whether bounding boxes or rectangles are overlapping or not, is very fast and gives correct answers in 3D (2D space + time). In 2D, the query is posed with the `SDO_RELATE` operator (which compares two objects based on their geometries). This also gives fast response and correct answers.

The main conclusion is that the generic model for storing moving point data in a geo-DBMS is flexible and efficient. The data can be accessed in a fast way, depending on the type of query and the method used for indexing.

Samenvatting

Ruimtelijk-temporele gegevens worden steeds populairder. Een voorbeeld van deze ruimtelijk-temporele gegevensverzamelingen is een verzameling van bewegende punt objecten. Gegevens over posities van bewegende vliegtuigen, voetgangers en auto's zijn enkele voorbeelden hiervan.

Het opslaan van grote hoeveelheden gegevens in Database Management Systemen (DBMS) heeft vele voordelen ten opzichte van het opslaan in bestanden. Het principe van het opslaan van gegevens in bestanden is dat er applicaties nodig zijn om met de gegevens te kunnen werken. Voor elke toepassing dient een aparte applicatie te worden ontwikkeld. Het principe van de DBMS oplossing is een gegevens set waarin de gegevens staan opgeslagen en een DBMS om op de gegevens bewerkingen en bevestigingen uit te kunnen voeren. Dit DBMS zorgt ervoor dat de gegevens op een beveiligde en consistente manier worden opgeslagen en dat de gegevens door meerdere applicaties benaderd kunnen worden. Ruimtelijke DBMSs hebben het extra voordeel dat ook ruimtelijke gegevenstypen kunnen worden gedefinieerd zoals punten en vlakken en dat er ruimtelijke bevestigingen mogelijk zijn (bijvoorbeeld het zoeken van overlappende vlakken). In het geval van bewegende puntobjecten is het op zijn minste nuttig om uit te zoeken of de DBMS-oplossing gebruikt kan worden om de gegevens op te slaan, te bevestigen en te bewerken.

De hoofdvraag van dit onderzoek is:

Wat zijn de mogelijkheden en de performance van een geo-DBMS om ruimtelijk-temporele puntenwolken van bewegende objecten te structureren, te indexeren, te bevestigen en te visualiseren?

Er zijn onderzoekers geweest die opslagstructuren hebben ontwikkeld voor bewegende puntobjecten. De geformuleerde modellen hebben wel nadelen. Bijvoorbeeld het model van Vazirgiannis en Wolfson, dat ontworpen is voor het modelleren van bewegende voertuigen in stedelijke gebieden of een ander model (van Wolfson) waarin voertuigen waarbij de snelheid constant blijft zijn gemodelleerd. Er zijn ook modellen met een sterk redundante opslag. Een efficiënte indexering van de gegevens zorgt voor een snellere ontsluiting van de gegevens. Voor verschillende bevestigingstypen zijn verschillende indexeermethoden beschikbaar. In het geval van bewegende puntobjecten zijn deze indexeermethoden in vele gevallen gebaseerd op de R-tree. Vaak is het van tevoren niet duidelijk wat voor vragen er aan de gegevens gesteld zullen worden en welke structuur en indexeringen daarbij moet worden gekozen. Daarom is een generiek model ontwikkeld voor het opslaan van gegevens over bewegende punt objecten.

Het basisprincipe van het generieke model is het kiezen van een basistabel, waar drie andere representaties van de data (gebaseerd op verschillende geometrische data typen) vanaf kunnen worden geleid door gebruik te maken van (gematerializeerde) views. De 4 verschillende representaties zijn dan gebaseerd op een consistente set gegevens die 2D punten (x,y) , 3D punten (x,y,t) , 2D lijnen $(x_i,y_i, x_{i+1},y_{i+1})$ en 3D lijnen $(x_i,y_i,t_i, x_{i+1},y_{i+1},t_{i+1})$ bevat. Tijd wordt in de 2D representaties behandeld als een attribuut. Op deze gegevens kunnen dan vele bevestigingen worden gedaan, bijvoorbeeld om de snelheid en versnelling van objecten te bepalen of de bewegingsrichting.

Nu het model is geïntroduceerd, kan een efficiënte indexeer- en bevestigingsmethode worden gevonden. Aangezien Oracle 9i Spatial (het gebruikte DBMS voor dit onderzoek) slechts de 2D en de 3D R-tree heeft ingebouwd, kan optimalisatie van de efficiency voor bevestigingen door de gebruiker alleen worden bewerkstelligd door de bevestiging zelf.

Om te laten zien dat dit generieke model snel en flexibel is, is voor twee sets gegevens dit model geïmplementeerd in Oracle 9i Spatial. In het eerste geval zijn de gegevens van tevoren ingewonnen en achteraf in de DBMS geanalyseerd. Vanuit een helicopter is elke 0.1 seconden een foto gemaakt van een snelweg. De voertuigen op deze snelweg zijn in meerdere foto's gedetecteerd waardoor er een tabel ontstaat met voertuig_id's, tijdstippen (fotonummers) en posities. Deze gegevens zijn succesvol geïmplementeerd in het generieke model en een aantal bevestigingen is gedaan op deze gegevens. Omdat deze gegevensset vrij klein is, koos de Oracle query optimizer ervoor om een full-table scan te doen in plaats van gebruik te maken van de ruimtelijke indexering. Voor complexe bevestigingen, leidt dit tot een trage respons tijd. Het is te verwachten dat als de hoeveelheid gegevens toeneemt, dat de query-optimizer dan wel gebruik zal maken van de index.

De tweede case studie onderzoekt een real-time set gegevens. Er waren tijdens het afstudeeronderzoek geen real-time gegevens beschikbaar. Vandaar dat dit is gesimuleerd met track-logs van een aantal taxi's in Rotterdam en omgeving. Gedurende twee jaar zijn deze gegevens verzameld voor ongeveer 60 taxi's. Hieruit is een selectie gemaakt van zeven dagen waarvoor de simulatie is opgezet op een dusdanige manier dat er een continu groeiende tabel ontstaat. Het generieke model is ook in deze case succesvol geïmplementeerd. De tijden voor het genereren van het antwoord op een ruimtelijke vraag, groeit niet met een groeiende set gegevens, maar is afhankelijk van het aantal antwoorden, mits de bevestiging juist is opgesteld. Met behulp van de SDO_FILTER operator die kijkt op basis van de index of de minimum bounding boxes of rectangles interacteren, kunnen de gegevens op een snelle manier worden bevestigd in 3D (2D ruimte + tijd). In 2D is gebruik gemaakt van de SDO_RELATE operator. Ook deze geeft snelle en correcte resultaten.

De belangrijkste conclusie is dat het generieke model een snelle en flexibele methode is om gegevens van bewegende puntobjecten op te slaan in een DBMS. De gegevens kunnen op een snelle manier worden ontsloten afhankelijk van de bevestiging en de indexering.

Contents

Preface	iii
Summary	v
Samenvatting	vii
Contents	ix
1. Introduction	1
2. Space, time and DBMSs.....	3
2.1 CONCEPTS OF SPACE AND TIME	3
2.2 THE DATABASE MANAGEMENT SYSTEM	3
2.3 SPATIAL DATA SETS AND GEO-DBMSS	6
2.4 TEMPORAL DBMSS	7
2.5 CONCLUSIONS	9
3. Spatiotemporal modeling, indexing and querying methods.....	11
3.1 A FRAMEWORK FOR SPATIOTEMPORAL DATA MODELS.....	11
3.2 MODELING SPATIOTEMPORAL DATA	12
3.3 INDEXING METHODS	15
3.4 QUERYING SPATIOTEMPORAL DATA	19
3.5 CONCLUSIONS	20
4. A generic model for moving object DBMSs	23
4.1 THE PRINCIPLES OF THE MODEL	23
4.2 VIEWS AND MATERIALIZED VIEWS	25
4.3 FINAL REMARKS	27
5. Case I - Traffic seen from a helicopter (post processing)	29
5.1 IMPLEMENTATION OF THE GENERIC MODEL INTO ORACLE 9i SPATIAL	30
5.2 REFLECTION TO LANGRAN'S TECHNICAL REQUIREMENTS	30
5.3 QUERY: KEEP TWO SECONDS DISTANCE FROM YOUR PREDECESSOR.....	34
5.4 QUERY: CALCULATING TRAFFIC FLOW VARIABLES	41
5.4 CONCLUSIONS	43
6. Case II - Taxi cabs in Rotterdam (real-time)	45
6.1 REAL-TIME SIMULATION	45
6.2 REAL-TIME INDEXING	47
6.3 QUERYING THE REAL-TIME DATA IN 2D	49
6.4 QUERYING IN 3D	51
6.5 CONCLUSIONS	56
7. Conclusions and recommendations.....	57
7.1 CONCLUSIONS	57
7.2 RECOMMENDATIONS	60

Literature	63
Appendix A: Glossary	65
Appendix B: Scripts	67

1. Introduction

Temporality is an inherent aspect of geo-information. Nowadays applications of spatiotemporal GISs are becoming important. For instance with respect to Cadastral issues [15], national road databases [5], or the detection of traffic jams [14]. Traffic monitoring is an example of an application of modeling moving objects. Geo-DBMSs make it possible to manage large spatial data sets that can be accessed by multiple users at the same time. These spatial data sets usually contain 2D data, while more and more applications depend on 3D data [2]. In the case of moving objects, spatiotemporal data sets can also be seen as 3D data. (x-position, y-position, time). In the recent literature, the modeling of moving objects in a Geo-DBMS context is a subject [18].

Continuous movement of objects poses new challenges to database technology. In conventional DBMSs, data is assumed to remain constant unless it is explicitly modified [20]. This statement causes efficient structuring methods for the data when the data set is continuously growing. Also querying these data sets is an issue that has to be taken into account. Examples of spatiotemporal queries are: Which object has the highest speed, what is the mean distance between two objects, which objects are in a certain time interval in that polygon, etc. Answering these queries, requires an efficient storage and indexing method in the case the data set is large. Visualizing the moving objects and answers on the queries is an important issue while handling spatiotemporal applications. Important is the distinction between real-time monitoring of moving objects and modeling the data afterwards (post processing / data mining).

The main question of this research can be described as follows:

What is the potential and performance of a geo-DBMS to structure, index, query and visualize spatiotemporal point clouds of moving objects?

The word “potential” in this main question is focused in this research on efficiency and flexibility. To answer this main question, the following partial questions have been drawn:

1. Why could a geo-DBMS be an efficient and flexible way to store moving point data?
2. Which methods are available to structure and index moving point objects in a geo-DBMS context?
3. Does a generic model exist to implement moving point objects in a geo-DBMS like Oracle 9i Spatial?
4. Is this generic model sufficient for a static data set where the moving point object data is collected in advance?
5. Is this generic model sufficient for a dynamic data set, where the moving point objects are collected real-time?

These questions are answered by studying literature to find the available methods for structuring, indexing, querying and visualizing moving objects in a Geo-DBMS context. A generic model for moving object databases is developed. For two cases, this generic model has been implemented and tested, with Oracle 9i Spatial. This DBMS is chosen because it is one of the more advanced geo-DBMSs and also because of the knowledge available and experiences in the past in the section GIS technology at the TU Delft. The two cases (questions 4 and 5) are described in chapter 5 and 6 in this report.

After this introduction, an overview will be given of space and time and the role that DBMSs could have in this space and time context. An important question in this chapter is why DBMSs

could be used in the case of spatiotemporal data. Some examples of available data structures for organizing moving point object data in DBMSs and of the available methods for indexing this data, found in the literature are described in chapter 3. In chapter 4, a generic model for this data will be introduced. The main idea of this model is organizing the data in a base table with different views suitable for the actual use or purpose.

In chapters 5 and 6, two case studies are described where the generic model is implemented and tested. With these case studies, the advantages and disadvantages of the model should become clear. Chapter 5 deals with a case where all the data is collected in advance. In chapter 6, the implementation of a real-time data set in Oracle is described and analyzed. In the last chapter, some conclusions and recommendations for future research can be found.

2. Space, time and DBMSs

In this chapter, the concepts and definition of space and time will be discussed. In connection to Database Management Systems (DBMS), the advantages of the use of DBMSs will be mentioned to show why data with a spatial and a temporal aspect could be managed in such a DBMS. The final goal of this chapter is to make clear whether DBMSs or file-based systems could be useful to model and query moving point object data.

First, in section 2.1, the definitions of space and time will be introduced. After that, in 2.2, DBMSs will be introduced in contradiction to file-based systems. In section 2.3, describing geo-DBMSs will make the connection between space and DBMSs. Section 2.4 is dealing with temporal data and temporal DBMSs. In section 2.5, some conclusions will be drawn. It will become clear why it is useful to use DBMSs for spatiotemporal data.

2.1 Concepts of space and time

The ancient Greeks developed the first philosophies of space and time. The earliest work to develop an explicit conception of space and time may have been Hesiod's mythological and philosophical treatise 'Theogony' dating from the 7th century BC. In 'Theogony' the world emerges from a state of 'chaos' (a timeless and spaceless state) into 'chronos' (ordered time) when 'gaia' (the earth) appears [19].

Raper [19] states that before the 20th century, most writers dealt with space and time as separate and distinct domains. This view was common in mathematics, geometry and philosophy. Einstein's paper on the Special Theory of Relativity (published in 1905) revolutionized thinking on space and time making it clear that in some circumstances it was necessary to think of a unified space-time. Acceptance of space and time integration implies that the world can be regarded as consisting of four-dimensional 'geo-phenomena' and their inter-relations.

Langran mentions the space-time cube in [8]. Her goal is to identify a conception that treats the components of a spatiotemporal model most effectively and to develop that view of time as a conceptual model for a temporal GIS. The three-dimensional space-time cube represents one time and two space dimensions. Space-time cubes depict processes in two-dimensional space that are played out along a third temporal dimension. The trajectory of a two-dimensional object through time creates an upwards-moving worm-like pattern in this phase space.

There are different ways to look at space and time, in a separate way, or combined where the time is an extra dimension. The other way to look at space and time is to represent space and time as something continuous or as something discrete. These views on space and time will become important later in this thesis.

2.2 The Database Management System

This section deals with Database Management Systems. First the file-based approach is described. After that, some shortcomings of file-based systems are described and on the end, the DBMS approach is introduced. In this last part, the definitions of a DBMS can be found.

File-based approach

Worboys describes in [25] an example of a file-based system, where a vegetarian fast-food restaurant uses a file to store the menu and prices. Applications are developed to operate on the data. Records can be easily inserted and deleted. What characterizes a file-based system is a collection of applications that perform services like producing different reports for the end-user. Each application defines and processes its own data. They constitute what we call diversified systems, each branch of the company has its own data and its own applications for handling the data.

In the moving-point object case, a file-based approach is also easy to use and does not have a complex structure. If you would like to do a query on the data (for instance, which vehicles drove when faster than 120 km/h?), the data file needs to be structured and sorted in the most efficient way and a program in for instance Java or C++ is easily written to answer this query. Even if the amount of data is very large, by structuring the data in the most efficient way for this specific query, the answer could be derived in a fast way.

Worboys describes in his file-based example some problems that come with the growth and the use of the file-based system by more than one user:

Loss of integrity: Linkages between programs and files become complex. The programs made the relationships between the data in the files: if the relationships changed then the programs had to be changed. The development of software was becoming complex and costly and errors come in.

Loss of independence: A too close linkage between program and data causes high software maintenance costs. For example, a change in a secondary storage medium requires a partial rewriting of many of the programs.

Loss of security: People from outside can also work with the applications and the files. So they can easily make changes that could harm the data or even the company, which uses the system.

Some problems of file-based systems are described above. These problems result into a list of disadvantages of file-based systems. Some disadvantages from the list below are mentioned in [29] and [28].

Separation of the data: Information needed for a particular task may be in different files – or even different departments' files. This makes that the data are difficult to retrieve and combine.

Duplication of the data: Data could also be duplicated and this will lead to redundancy, that is: the same data can be stored in separate files. This can result in inconsistency – we update data in one file forgetting to update the same data stored in another file.

Data dependence: The application must handle the physical storage as well as the content of the file. This makes that the data and the application are depending on each other. There is no access and process control of data except what is implemented in the application.

Incompatibility of files: Files may vary due to, e.g., the used application programming language. This can make different files incompatible for other applications.

Ad hoc application programs: Application programs perform specific tasks. New tasks may require a new application to be developed. For each new query, a new application has to be developed.

Most of these disadvantages also count for moving point object data. If a query needs to be performed (for instance, which cars were on the highway A13 between 18th and 25th May 2004?), a copy of the original file needs to be made, structured in the most efficient way and a query needs to be programmed. Making copies of the original data can lead to inconsistency

when new data needs to be added. Another example is that someone else would like to do a query on the same file. The other customer uses Microstation .dgn files instead of the ESRI shapefiles customer 1 needs; a conversion is needed with perhaps some loss of information.

The DBMS approach

The database philosophy is an attempt to solve the problems described above. The main principle of the DMBS approach is that the data are stored in one logical centralized location. The Database Management System (DBMS) is a piece of software that manages the data by insulating the data from uncontrolled access, allowing definition of the data model, supporting manipulation of the data and providing appropriate two-way access channels between the exterior and the data. It allows the designer to define the structure of the data in the DBMS, providing levels of authorization that allow different groups of users access to appropriate data and managing transactions with the database that may be occurring concurrently. The DBMS also provides data independence, so that the data in the database are accessible without precise knowledge of implementation details [25].

According to Peuquet [17], a data set is a collection of interrelated data specifically designed to be shared by multiple users. Data redundancy is controlled, and a uniform approach is used for accessing and modifying data within the data set. Database management systems (DBMS) incorporate data sets, as well as the computing software and hardware, the users, and the management staff to run the system. DBMSs, essentially computerized record-keeping systems, are used by virtually every enterprise today as a fundamental business tool for maintaining personnel, payroll, inventory and other information. DBMSs allow data to be kept secure, yet quickly accessed and updated by multiple users.

Except for the data and the DBMS, some other parts are important for a multi-user DBMS environment. In figure 2.1, these elements are drawn. At first there is the physically stored number of data sets, where the DBMS is taking care of for instance redundancy control or

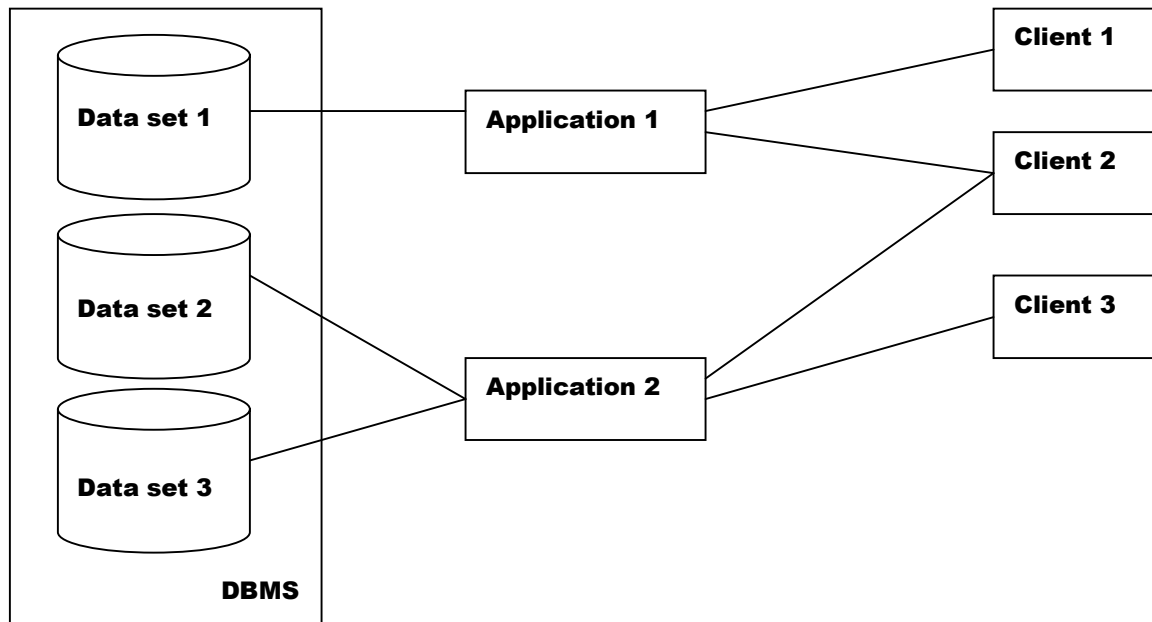


Figure 2.1 DBMS with multiple data sets are used through different applications by several clients.

accessing and modifying the data. Above this DBMS, there can be a number of applications, which operate on the data. An example of an application is a Web Server. Finally, there are several users, using the applications to operate on the data, stored in the DBMS, posing ad hoc queries and doing analysis on the data.

You can say that when subjects like multi-user, security, data integrity, consistency, redundancy and interoperability are relevant for the choice whether a file-based system or a DBMS approach should be chosen, the DBMS approach is preferred.

2.3 Spatial data sets and geo-DBMSs

In the previous section is shown that in many cases, the DBMS approach is preferred above the file-based approach. This section discusses spatial data sets and geo-Database Management Systems. Important is what the role of a geo-DBMS could be in spatiotemporal modeling.

A geo-DBMS is a DBMS for spatial data. The main difference between a “normal” and a “spatial” data set can be found in the data types and the data query language. In conventional data sets, data types like integers, floats, character strings and binary values exist. In the spatial data, extra data types are added in such a way that spatial objects could be defined. Examples of spatial objects are points, polylines and polygons. Coordinates, reference systems and dimensional information have to be added to the data.

In Oracle 9i Spatial, one of the most used spatial DBMSs, geometrical attributes are defined as the so-called MDSYS.SDO_GEOMETRY. This is a data type, in which many properties like the reference system or the geometry type could be managed as well as the coordinates of the data itself. In this MDSYS.SDO_GEOMETRY, the MDSYS indicates the schema in which the syntax, storage and semantics are described and SDO means Spatial Data Option.

An “SDO_GEOMETRY” always looks like:

```
Mdsys.sdo_geometry(  
    <sdo_gtype>,  
    <srid>,  
    <sdo_point>,  
    mdsys.sdo_elem_info_array(  
        <sdo_starting_offset>,  
        <sdo_etype>,  
        <sdo_interpretation>),  
    mdsys.sdo_ordinate_array(<coordinates>))
```

The meaning of the different elements of the sdo_geometry mean:

- sdo_gtype: This indicates the type of geometry (point, linestring, polygon, multipoint, multilinestring, multipolygon) and the dimension (0D, 1D, 2D, 3D) of its embedding space. Each geometry type has its own code, e.g. a 2D polygon has sdo_gtype = 2003. The first digit is the dimension and the last digit is the geometry type.
- sdo_srid: This is a reference to the spatial reference system used by the coordinates. In this research local (Cartesian-) coordinates are used, so no sdo_srid is specified (NULL).
- sdo_point: This element is used when only points are stored as single object or when a point is stored in addition to the other geometry. The SDO_POINT_TYPE has an x-, y- and z-element.
- sdo_elem_info: This specifies the elements of the geometry with references to the coordinates (starting offset in the ordinate array), information about the element itself (e-type) and an interpretation code (e.g. straight line, rectangle, circle) on how to interpret

the coordinates. This is stored in a variable array of numbers. A rectangular polygon specified by two coordinates is stored as `sdo_elem_info_array = (1,1003,3)`.

- `Sdo_ordinates`: This is a variable array of numbers and contains the coordinates.

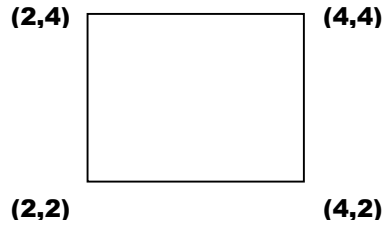


Figure 2.2 Polygon with its coordinates

For instance, a rectangular polygon like figure 2.2 can be described as:

```
INSERT INTO table (id, geometry) VALUES (1,  
Mdsys.sdo_geometry(2003,NULL,NULL,  
Mdsys.sdo_elem_info_array(1,1003,3),  
Mdsys.sdo_ordinate_array(2,2,4,4)));
```

This means that the elements of `sdo_geometry` data type are:

- `sdo_gtype` = 2003 (2D polygon)
- `sdo_srid` = NULL (no spatial reference system)
- `sdo_point` = NULL (no point type)
- `sdo_elem_info` = 1,1003,3 (coordinates start at position 1, outer polygon ring, rectangle)
- `sdo_ordinates` = 2,2, 4,4 (southwest and northeast coordinates)

The other main difference between a conventional and a spatial DBMS is the query language. In conventional DBMSs, often SQL (Structured Query Language) is used. A query in SQL always looks like:

```
SELECT <what/columns>  
FROM <table or view>  
WHERE <conditions>;
```

When there are spatial object types, also spatial querying needs to be possible. For instance, it has to be possible to find all overlapping polygons from two data sets. Oracle developed two methods for spatial querying. The first method is the spatial function, which does not use a spatial index and the second is the spatial operator, which requires a spatial indexing and needs to be posed into the where-clause. More about spatial querying will be discussed in the chapters 3, 5 and 6.

2.4 Temporal DBMSs

A reasonable goal for geographic information systems (GIS) is that they be capable of tracing and analyzing changes in spatial information. A non-temporal GIS describes only one data state. This means that historical states are essentially forgotten and the anticipated or forecast future cannot be treated. In contrast, a temporal GIS would trace the changing state of a study area, storing historic and anticipated geographic states.

The fundamental functions of a temporal GIS are inventory, analysis, updates, quality control, scheduling and display [8] (see figure 2.3).

Inventory	Store a complete description of the study area, and account for changes in both the physical world and computer storage.
Analysis	Explain, exploit, or forecast the components contained by and the processes at work in a region.
Updates	Supersede outdated information with current information.
Quality Control	Evaluate whether new data are logically consistent with previous versions and states.
Scheduling	Identify or anticipate threshold database states, which trigger predefined system responses.
Display	Generate a static or dynamic map, or a tabular summary, of temporal processes at work in a region.

Figure 2.3 Major temporal GIS functions [8].

Storing temporal information in a DBMS is not straightforward. The many different representations of time, makes modeling temporal aspects of information very complex. Peuquet [17] categorizes time and space as well into what can be termed both continuous or discrete and absolute or relative. In figure 2.4, a graphical representation of these four elements is drawn.

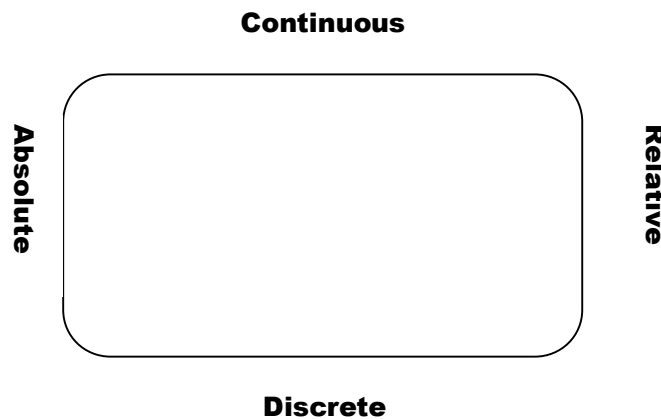


Figure 2.4 Varying views of reality [17]

Every representation of time can be placed in the schema above. For instance, a timestamp “June 16th, 2004 11:00 PM” is a discrete and absolute representation of time. While for instance a description like “Between last Thursday and yesterday” is a more continuous and relative one.

Not only the description of time in the above figure is complex, also the variety of representations of time in a DBMS deserves some special attention. Time can be defined in several data types:

- As a “date” data type: “June 16, 2004 11:03:24 PM”, “06-16-2004 23.03.24”, “11:35” or other variations.
- As an integer “26”. This is always needs some extra information (for instance 26 minutes before the current timestamp).
- As text “18th century”.

In your temporal GIS (including a temporal DBMS) it is necessary to make decisions about how to represent time, depending on your purpose. A temporal extension has been developed on the SQL-92 standard. The language is designated TSQL [21]. This document, describes some concepts, like the ontology. It states for instance “An instant is modeled by a timestamp coupled with an associated scale (e.g., day, year, month). A period is modeled by the composition of two

instant timestamps and the constraint that the instant timestamp that starts the period equals or precedes (in the given scale) the instant timestamp that terminates the period”.

A distinction needs to be made between valid time (also called user or real world time) and system time (transaction or DBMS time). If for instance object O is at position x,y at timestamp “June 16, 2004 11:03:24 PM” and is put into the DBMS three minutes later, then there is a difference between these two timestamps.

In the TSQL [21] extension on the SQL standard, definitions are given for timestamps, periods and instances. Except for these definitions, also some temporal functions are defined. Allen [1], defined in line with the 9-intersection model, a set of thirteen possible relationships between two temporal objects. Examples are “before”, “during” and “meets”. In figure 2.5, you can find a graphical representation of these temporal relationships, which can be used in temporal queries.

It can be concluded that time can be represented in many ways. The choice of how you would like to represent time in your DBMS, depends on the representation of time itself (continuous or discrete and absolute or relative), on how you would like to represent the data in your DBMS (as a “date” data type, as an integer or as text) and on which time you would like to manage the data in your DBMS (valid time or transaction time).

Relation	Pictorial example
X equal Y	XXX YYY
X before Y	XXX YYY
X after Y	XXX YYY
X meets Y	XXX YYY
X met-by Y	XXX YYY
X overlaps Y	XXX YYY
X overlapped-by Y	XXX YYY
X during Y	XXX YYYYYY
X contains Y	XXXXXX YYY
X starts Y	XXX YYYYYY
X started-by Y	XXXXXX YYY
X finishes Y	XXX YYYYYY
X finished-by Y	XXXXXX YYY

Figure 2.5 Relationships of time-intervals

2.5 Conclusions

This chapter presented space and time as two different dimensions, which are or are not integrated. There has also been spoken about file-based systems and a DBMS approach. Besides conventional DBMSs, also spatial DBMSs are available, where spatial data types and spatial querying are implemented. In temporal DBMSs, time can be represented in several ways, which makes the modeling of time in a DBMS complex.

As stated in section 2.1, time can be seen as an extra dimension of the space. So, if spatial DBMSs exist, if temporal DBMSs exist why shouldn't spatiotemporal DBMSs? Using file-based systems or a (spatial) DBMS needs to be considered for every special case. As stated before, if you deal with multi-user aspects, security, data integrity, consistency, redundancy or interoperability, the (spatial) DBMS approach seems to be more appropriate. The extra advantage of using a DBMS for spatial purposes is that tools for spatial querying on spatial data are already available and easy to use.

This thesis deals with moving object data, which is one special type of spatiotemporal data. The question is now, why should these data be stored and queried in a spatial DBMS? Because it is

possible that the purpose of the system that needs moving point data deals with multi-user aspects, security, integrity, consistency, redundancy and interoperability and because a complete suite of spatial queries and data types are already implemented in a geo-DBMS like Oracle 9i Spatial, it is at least worth to investigate whether it is useful to choose for a DBMS approach or not. Where file-based systems are often fast for special queries, DBMSs offer a perhaps less fast approach. But perhaps, this DBMS approach, with all its advantages, is still fast enough for many queries.

3. Spatiotemporal modeling, indexing and querying methods

In this chapter, different data structures and indexing methods found in the literature, will be described that deal with spatiotemporal data. In relation to the subject of this thesis, only the spatiotemporal structures and indexing methods will be described which deal with moving objects. This implies that changes in for instance the shape, color, name, etc., during a certain time period, will be out of the context. However, both the terms “moving point object data” and “spatiotemporal data” are used in this chapter. In this thesis, these terms are used as being equal.

The specific models described in this chapter will be extended in chapter 4 by a generic model for moving point objects in a DBMS context. However, this chapter is important to get an idea of the available methods and how these methods are related to each other. It gives an overview of the research that has been done before in this field.

Before going into the data structures and indexing methods, a framework is introduced in 3.1. This framework is important to put the methods for structuring and indexing in a context. The spatiotemporal data structures will be described in paragraph 3.2. Different approaches for different purposes will be described. In 3.3, the methods to index spatiotemporal data will be described. Indexing is important to increase the query-response time in large data sets. Section 3.4 deals with querying spatiotemporal data. In the last section, some conclusions will be derived from this chapter.

3.1 A framework for spatiotemporal data models

According to Langran [8], “five technical requirements will drive the development of a temporal GIS: a conceptual model of spatial change, treatment of aspatial attributes, data processing logistics, a spatiotemporal data access method and efficient algorithms to operate on the spatiotemporal data.” The first three requirements are part of the data structure; the temporal data access method is also related to indexing.

The first part of this technical framework, the conceptual model, can be described as “the configuration of information, as it will be represented to the computer. It defines the entities, attributes and relationships to be portrayed; it also defines the operations to be performed and the constraints to be enforced. [8]” In the next section, some conceptual models for moving point object data will be described.

Treatment of aspatial attributes is, except for the ‘attribute time’, outside the focus of this thesis. The color, shape, value, name, etc. are meant to stay constant. The objects, in this thesis can be represented as points, where other attributes than position or time, do not matter. However, it matters what you do with these ‘other aspatial attributes’. They do not change in time, so it might be useful to store these non-spatial, non-temporal attributes once and not on every timestamp in the case of moving point objects.

The third technical requirement, mentioned by Langran is the “data processing logistics”. This deals with the primary storage structure, error control, and updating the stored data. It depends on the purpose of your system. It depends on whether it deals with real-time or post-processing data, the use of a DBMS or not, the collection of only current data or also storing the past spatiotemporal data, etc. The data processing logistics can be described as the conditions for your system.

The spatiotemporal access methods, the fourth technical requirement, will be described in this chapter (3.3). A number of indexing methods that can handle moving point data is available in literature. Indexing is important to get a faster access of your data. It decreases query-response times. Choosing the most optimal indexing method for a specific query could minimize these query-response times.

The last technical requirement, the efficient algorithms, is also a very important one. You can imagine that it is efficient to store the results of queries that are used very often. With spatiotemporal data, you can think of the speed of moving objects. In many data structures this attribute is not available in the root table. For instance in Oracle 9i Spatial, it is possible to define “(materialized) views”. This is an efficient approach that stores the query results of pre-defined queries. Querying spatiotemporal data will be discussed in 3.4.

3.2 Modeling spatiotemporal data

A trajectory is the path described in space and time by a moving object. Such a trajectory can be represented in different ways. Some researchers described models to structure these trajectories into a DBMS. Ouri Wolfson et al described the MOST data model [24] and Marchand et al, defined the Spatiotemporal TOD [10]. Meng and Ding developed the DSTTMOD [11] and Vazirgiannis and Wolfson, also found a method to structure spatiotemporal data for moving objects [23]. In this section, these models will be described to get an overview of the previous work done in modeling moving point object data by making use of DBMSs.

3.2.1 Vazirgiannis and Wolfson

In [23], Michalis Vazirgiannis and Ouri Wolfson describe a model for moving objects on a road network. They show the need for a small and robust set of predicates with high expressive power, suitable for realistic implementation based on off the shelf DBMS technology. The underlying model they use consists of three parts.

The first part is a **map**, where each tuple in the relation represents a road segment, defined as the road section between 2 intersections, with the following attributes:

- *Polyline*: the block polyline given by a sequence of 2D x,y coordinates: (x1, y1),(x2,y2),...,(xn,yn). Usually the segment is a straight line, i.e. given by two (x,y) coordinates.
- *Fid*: the road segment number (id).
- *Attributes for geocoding such as left side from street name, left side to street name, right side from street name, right side to street name, postal code, speed limit, one-way or not, etc.*

The second part of the model is the **moving object**. The route of a moving object O is specified by giving the starting address or (x,y) coordinate (start_point), the starting time and the planned destination of the trip or (x,y) coordinate (end_point). The attributes that can be added in this table are the fixed part the moving object, for instance things like the driver, color, weight, length, etc.

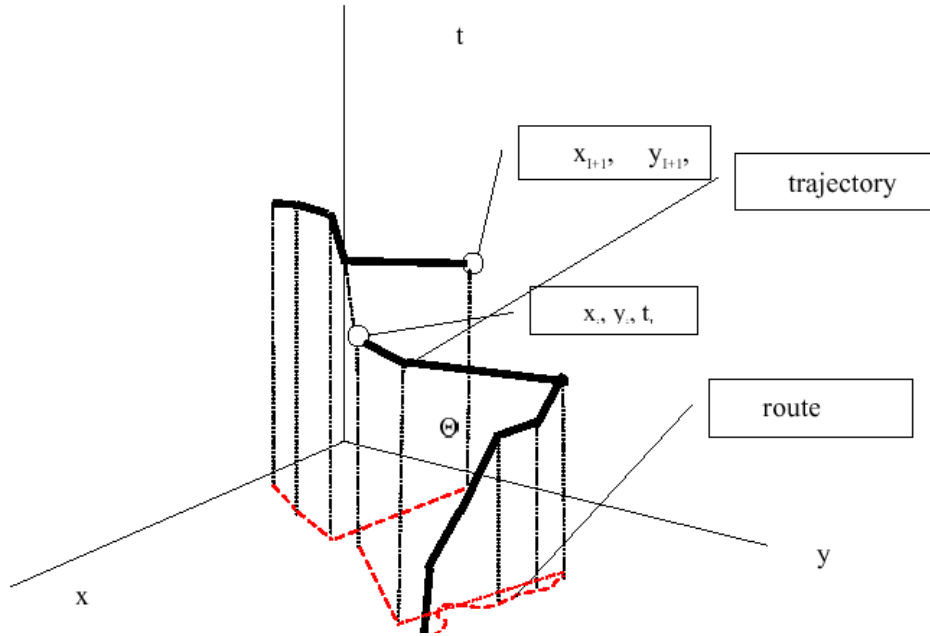


Figure 3.1. A moving point representation of two different objects [23]

Vazirgiannis and Wolfson defined the **trajectory** as the third part of their model. This part is the true spatiotemporal (dynamic) part of their approach. The trajectory of an object, denoted $T(O)$, is a relation with attributes *sequence#*, x , y , t , b . A tuple $[i, (x,y), t_i, b]$ in this relation indicates that (x,y) is the i 'th intermediate point on O 's route $L(O)$, and O will be there at time t_i . A trajectory is a piece-wise linear function in 3D. The attribute b is a Boolean, which is False, if the i 'th tuple is the beginning or the endpoint of a trajectory and True for if i is a point somewhere between the beginning and ending of the trajectory. If more than one moving objects are described in this model, an id-number needs to be added to the trajectory-part and the moving-object-part of the model. The connection between the road network and the trajectory is defined by a predicate "LOC(id,t)", which returns the location of the moving object (id) on time (t) on the road network. In figure 3.1 [23], an example of a moving point representation in trajectories is given.

3.2.2 The MOST data model

Ouri Wolfson et al describe in [24], the Moving Objects Spatio-Temporal (MOST) data model. In their approach, they also make a distinction between static attributes and dynamic attributes. Data in a DMBS are assumed to be constant and not continuously updated. The solution according to Wolfson et al is representing the location as a function of time. This function is defined as the speed between two sampled points. So, the location changes as time passes, even without an explicit update.

In more detail, a dynamic attribute A is represented by three sub-attributes, $A.updatevalue$, $A.updatetime$ and $A.function$, where $A.function$ is a function of a single variable t that has value 0 at $t=0$. The value of a dynamic attribute depends on the time, and it is defined as follows. At time $A.updatetime$ the value of A is $A.updatevalue$, and until the next update of A the value of A at time $A.updatetime+t_0$ (where t_0 is a positive number) is given by $A.updatevalue+A.function(t_0)$. An explicit update of a dynamic attribute may change its value sub attribute or its function sub-attribute, or both sub attributes. When for instance a moving object is at x-coordinate 100000

(meters) at the update time and is driving with a constant speed of 2 m/s eastwards ($x.function=2$), the x-coordinate at $t=60$ seconds after the last update-time is $100000+2*60=100120$.

This approach is straightforward for objects that move freely in space (e.g. aircraft). However, this would be inefficient (i.e. may generate many updates) for objects moving along a winding route, since each turn would constitute a change of $x.function$ and $y.function$ (where x and y are dynamic attributes of a moving object).

To address this problem, Wolfson et al extend the dynamic attribute concept to include the route as follows. The location attribute L ($L.x$ and $L.y$) is a dynamic attribute with five sub-attributes, namely $L.route$, $L.x.updatevalue$, $L.y.updatevalue$, $L.updatetime$, and $L.speed$. Among them, $L.route$ is (the pointer to) a line spatial object indicating the route on which an object is moving. $L.x.updatevalue$ and $L.y.updatevalue$ are the x and y coordinates of a point on $L.route$; it is the location of the moving object at time $L.updatetime$, i.e. the time of the last location-update. $L.speed$ is a linear function of the form $f(t)=b \cdot t$, assuming linear interpolation between two sample points. It is defined by the speed b of the moving object, and it gives the current distance from the starting location as a function of time t elapsed since $L.updatetime$. The location at time $L.updatetime+t$ is the point (x,y) which is at route distance $L.speed \cdot t$ along $L.route$ from the point with coordinates $(L.x.updatevalue, L.y.updatevalue)$.

3.2.3 The Spatiotemporal Topological Operator Dimension

In their approach [10], Marchand et al describe a method to implement spatiotemporal topological operators in multidimensional databases (MDDBs) through a hierarchy of topological operators representing spatial and temporal relationships between instances of objects. This hierarchy covers the three possible domains of spatiotemporal topological constraints e.g. spatial, temporal and spatiotemporal. At the root of this hierarchy users can make use of simple operators such as “same place” or “same time, same place” in their multidimensional query.

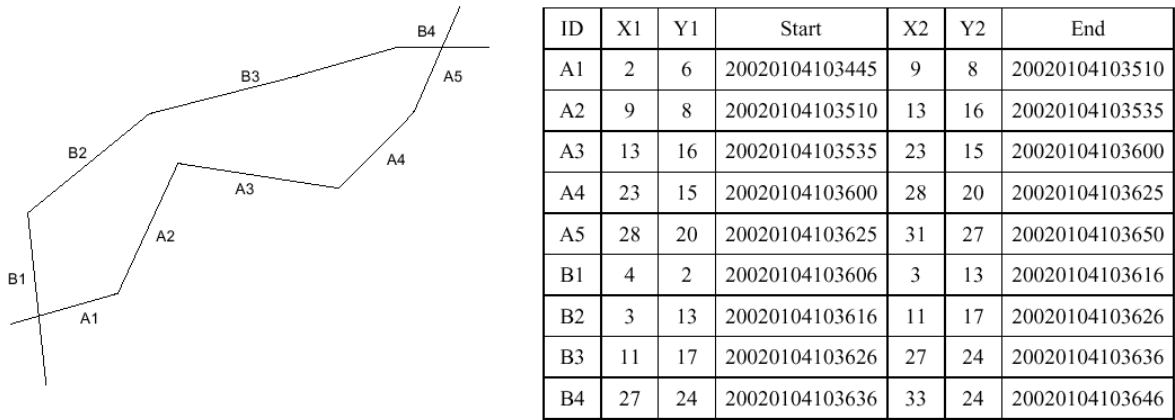


Figure 3.2 The data structure of spatial segmented trajectories

Something also mentioned in [10], is the data structure, used for these topological operators. Marchand et al did experiments, where they used a data structure like presented in figure 3.2. They used the trajectory (they call it a spatial segment) as a primitive. According to this article, this structure permits to topologically query, at the finest granularity, each individual spatial

entity as well as its temporal primitive. In the table, a lot of redundant storage is the result. Every x, y, t is denoted twice in the data set, but the advantage is the ease of use for the topological operators like BEFORE, DURING, ENDS AT, CROSSES, etc.

3.2.4 A Discrete Spatio-Temporal Trajectory Based Moving Object Database System

Meng and Ding describe in [11] their approach to model moving objects in a spatiotemporal DBMS. They call it the DSTTMOD, the Discrete Spatio-Temporal Trajectory Based Moving Object Database System. Their main goal was to support queries for location information not only in the past and present, but also in the future. In this model, trajectories are used to represent dynamic attributes of moving objects, including the past, current and future location information. Moving objects can submit moving plans of different length according to their moving patterns. Moreover, they can divide the whole moving plan into multiple sections and submit each section only when it is to be used. Different moving objects can set up different thresholds to trigger location updates. When a location update occurs to a moving object, not only its future trajectory is updated, but also the corresponding index records are adjusted.

The whole trajectory of a moving object is in this model represented by a set of line segments in the spatial-temporal space (X, Y, T) . Within each line segment, the movement of a moving object has the following properties:

- a) Spatially, the moving object moves along a straight line;
- b) The speed of the moving keeps constant.

The position of a moving object is in this way a linear interpolation between two sampled points (the begin and the end point of every trajectory). In order to get information about the future locations of the moving objects, these objects need to submit their moving plans to the system in advance. During the process of moving, when the deviation of the actual location from the anticipative location exceeds a certain threshold a location update is triggered. In this case, both the current and the subsequent segments of the trajectory need to be updated and the corresponding indexing structures must also be modified to reflect the up-to-date situation.

The structure of the table is the same as the structure of the method Marchand described ($MO_i, x_i, y_i, t_i, x_{i+1}, y_{i+1}, t_{i+1}$). So also in this approach, the trajectory segment is the main primitive with the implied redundancy.

3.3 Indexing methods

The previous section presented some methods to model moving object data in a DBMS. All these models are developed for one or more applications. In this section, some of the indexing methods to speed up access to the stored spatiotemporal data will be described.

Of course, indexing cannot be seen without knowing which queries are going to be done on the data. Section 3.4 deals with the querying of spatiotemporal data and that section discusses the choice for a sufficient indexing method.

This section starts with describing the principles of the R-tree, because the most indexing methods for spatiotemporal data are based on these principles. After that, some more advanced indexing methods will be described.

3.3.1 The 2D and 3D R-tree

An index is an ordered table where each record contains two fields: (1) the *index field*: This contains the value of the indexing field (in a sorted order) and (2) the *pointer field*: This contains the addresses of disk blocks that have the index value. When the index (which also requires disk space) requires only one disk block, a single-level index could be used. This works the same as the index in a book. When the index requires more than one disk block, the index also needs to be indexed (multi-level indexing) to cut down the search times [25].

On the origin of the most indexing methods for spatiotemporal data, is the R-tree, which is a multi-level indexing method. The R-tree indexing method is based on Minimum Bounding Rectangles (MBR) in the 2D-case or Minimum Bounding Boxes (MBB) in the 3D-case. An R-tree index stores this MBR or MBB that encloses each geometry in a spatial data set. This MBR or MBB is used to reduce the computational complexity in spatial queries and is defined along the axes. One of the properties of an R-tree is that all the leaf-nodes are on the same level (depth). Efficient insert and delete algorithms are defined to ensure that each node in the tree is always at least half full, so the overall tree structure remains balanced.

The advantage of using an R-tree index is that the irregular sized MBRs or MBBs can fit the objects in the real world, in contrary to the subdivision of space in the quadtree. The disadvantage is that the MBRs or MBBs can be much larger than the objects itself. It causes the R-tree index to select more candidate objects, because empty parts of the MBRs will fall within the query window. This increases the load in the exact computation (the second step in solving a query), because more objects need to be processed [2]. An example of 2D R-tree can be found in figure 3.3.

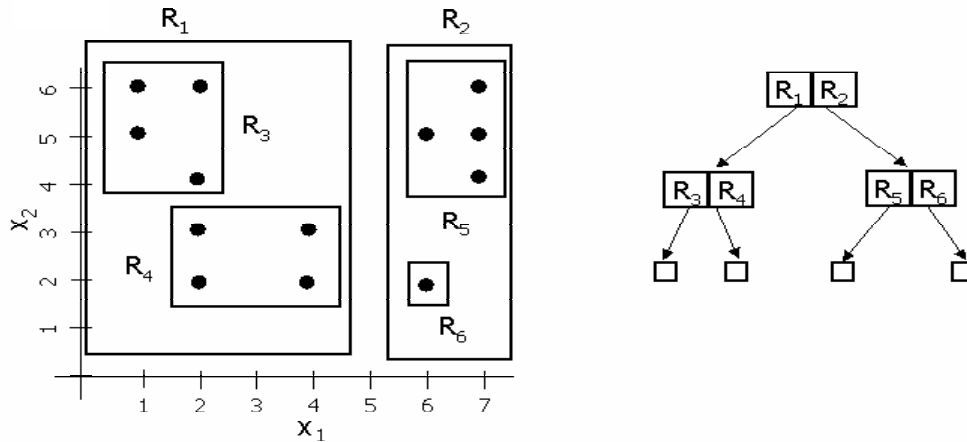


Figure 3.3 An example of a 2D R-tree

As said before, the R-tree is also developed in 3D. An example of how these MBBs are organized can be found in figure 3.4. In the case of spatiotemporal data, the index describes two spatial and one temporal dimension.

Numerous researches have been developing spatiotemporal access methods as an auxiliary structure to support spatiotemporal queries. In [12], an overview is given of many spatiotemporal indexing methods. The different indexing methods in this article are organized in three parts, the first part is about indexing the past, the second part is about indexing the current time and the last

part is about indexing methods that deal with spatiotemporal data from the future. In the next part of this section, some these methods will be described in a short way.

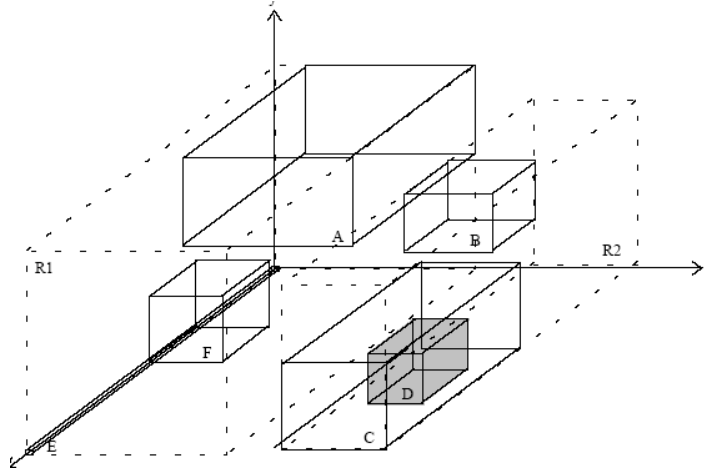


Figure 3.4 Example of minimum bounding boxes in the 3D R-tree [22]

3.3.2 Spatiotemporal indexing methods

In this paragraph only the methods that could be useful for moving objects will be mentioned and explained, selected from [12]. First, in short two methods will be described that index the past trajectories (RT-tree) or the future trajectories (TPR-tree). After that, two methods will be described in some more detail, the 2+3 R-tree that indexes the past and the current trajectories and a method developed by Pfoer and Jensen which makes use of network-constrained moving object data.

The **RT-tree** combines the R-tree and a temporal access method that has been developed to index past trajectories. In the RT-tree, a new entry is added to the regular R-tree that indicates start and end times of the current object. An RT-tree entry is of the form (id, MBR, t_s, t_e) , where id is the identifier, MBR is the minimum bounding rectangle of the trajectory, and t_s and t_e give the time interval in which this object is valid. The RT-tree supports spatial queries as efficient as the regular R-tree. However time slice queries and interval queries may span the whole tree. It differs from the 3D R-tree because no MBB is used, but a MBR. The third (temporal) dimension is treated in the index table as separate attributes.

One of the main methods to index future trajectories is the TPR-tree. The Time Parameterized R-tree (**TPR-tree**) employs the idea of parametric bounding rectangles in the R-tree. At the construction time, the TPR-tree builds the so-called conservative bounding rectangles that enclose a set of moving objects. The lower bound of the conservative bounding rectangle is set to move with the minimum speed of the enclosed points, while the upper bound is set to move with the maximum speed of the enclosed points. In this case, the conservative bounding rectangle never shrinks, and is guaranteed to always contain the enclosed moving objects. To avoid the case where the bounding rectangles grow to be very large, whenever the position of an object o is updated, all the bounding rectangles on the nodes along the path to the leaf at which o is stored are recomputed.

2+3 R-tree

An approach to index current and past trajectories is the **2+3 R-tree** [13]. The main principle of this method is to use two separate R-trees, one for two-dimensional points, and another one for three-dimensional lines (hence the name 2+3 R-tree). In the 2+3 R-tree whenever the end time of an object's position is unknown it is indexed under a two-dimensional R-tree, keeping the start time of its position along with its id. Note that the original R-tree (or any of its derivatives) keep only the object's id (or a pointer to the actual data record) and its MBR in the leaf nodes. The two-dimensional R-tree used in this approach is thus minimally modified. Once the end time of an "open" object's current state (i.e., position) is known, we are able to construct its three-dimensional line (the 3D trajectory), insert it into the three-dimensional R-tree and delete the existing entry from the two-dimensional R-tree.

It is important to note that now both trees may need to be searched, depending on the time point with respect to which the queries are posed. Similar to the case of the 3D R-tree any of the proposed R-tree derivatives could be used, provided that the leaf nodes of the two-dimensional one are minimally modified.

A final remark should be done. The 2+3 R-tree is the real-time version of the 3D R-tree. That is to say that the two-dimensional R-tree serves the single purpose of holding the current (i.e., open) intervals. Should one know all movements *a priori* the two-dimensional R-tree would not be used at all, hence the 2+3 R-tree would be reduced to the 3D R-tree presented earlier.

Pfoser and Jensen indexing method

In [18], Pfoser and Jensen describe the method they developed to access spatiotemporal data. Their method works for network-constrained moving objects. The main idea is that the 3-dimensional (x,y,t) space is mapped onto two 2-dimensional spaces. 2-dimensional indexing methods are fast and known very well.

A two-dimensional network can be reduced to a one-dimensional space, by taking the edges of the network and transforming them into intervals. The first edge becomes the first sub-interval, which starts where the 1D interval starts and extends a distance that corresponds to its distance in the network. The second edge then starts where the first ends, etc. The end of the sub-interval corresponding to the last edge is the end of the 1D interval. The second dimension is time. Two 2D-indexes, one for the network and one for the transformed trajectories can access the 3D (spatiotemporal) data. In figure 3.5, the main idea is drawn.

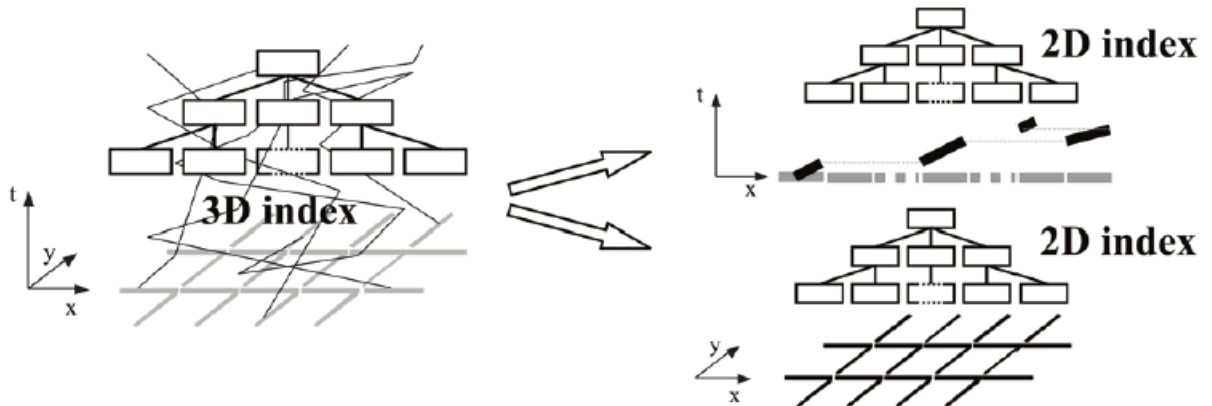


Figure 3.5. 3D-space is subdivided into two 2D-spaces with the Pfoser and Jensen access method. [18]

Pfoser and Jensen also did a performance test with this indexing method. They used five different types of networks, three synthetic and two real world networks. The experiments show that the lower the complexity of a network, the more likely the mapping approach proves to be beneficial over indexing the data in 3D space.

3.4 Querying spatiotemporal data

The last of the technical requirements mentioned by Langran, deals with efficient querying of the data. In the previous chapter is described that time is often seen as an extra dimension. For some cases, this is true and for other cases, time needs a special treatment. This special treatment regarding to querying a data set will be described in this section.

In advance, when the data is going to be structured in the DBMS, it is not known which queries are going to be done on the data. As a two dimensional example, there is a database of many spatial objects, let's say houses, roads, lakes, railways, etcetera. If you would like to find the longest road, your first selection could be to select all the roads from the table with all objects. A possibility is to make a new table with roads, choose an efficient indexing method to answer the question and do the query. For this specific query, you will get the fastest answer.

Another option is to answer the query on the table with all objects and select the longest road directly. So, this means not by making a copy of the original data, but just by adding a condition into your query where all the roads are going to be selected. The query itself will not be as fast as doing the query on a table with only roads and the specific indexing, but the main advantage is that you do not make a copy of the roads in a new table. Probably the same amount of work is done, but the variant with the 'copy' will be (in total) slower, because a copy has to be stored explicitly, which also takes time. Especially if you work in a multi-user environment, it is necessary that the data stays consistent and with copies of tables, the consistency is in danger.

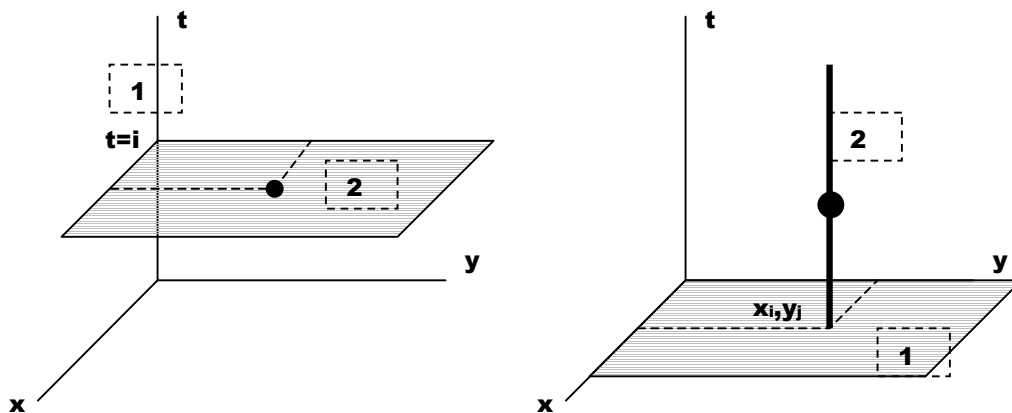


Figure 3.6. On the left, first do a query in the time dimension ($t=i$) and then in the spatial dimension., assuming that you are looking for a point in the 3D (x,y,t) space. On the right, first has been queried on the space $(x,y=x_i,y_i)$ and after that querying on the time dimension, assuming that you are looking for a point in the 3D (x,y,t) space.

Finding the longest road in your database with many kinds of objects will look like this in SQL:

```
SELECT max(length)
FROM table_with_all_objects
WHERE object_type="road";
```

Querying moving object data is possible in three ways:

- The first query type contains queries where only the time-dimension is involved. An example is “Which moving objects existed in the last hour?”
- In the second query type, only the spatial dimensions are involved, for example “Which pedestrians have ever been at the park?”
- The last query type contains both the spatial and the temporal dimension. An example is “Which pedestrians were in the park the last hour?”

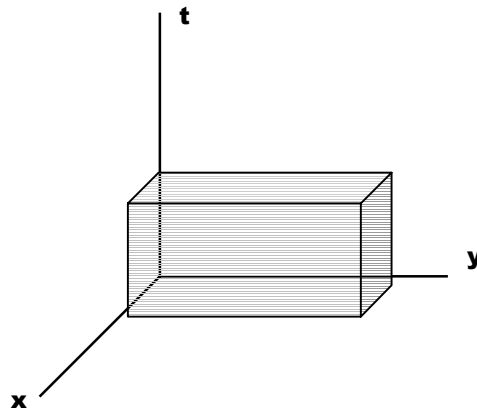


Figure 3.7. Querying in both the spatial and the time dimensions, gives a box in which the answer on the query can be found.

This third query type can be done in three ways, in figure 3.6 and 3.7 these queries are drawn:

- The first way is first querying in the 2D space dimension and after that in the 1D time-dimension.
- The second way is the opposite, first querying in the 1D time dimension before the 2D spatial query is going to be queried.
- The last possibility is querying the 3D-space (x,y,t) at once.

It is up to the query optimizer to estimate the most efficient query plan for a given query and available model (with index). With the first two querying methods, a 2D (spatial) index and a 1D (time) index seems logical, for the third method, a 3D index seems to be the most appropriate one.

For spatiotemporal data, the principle of not making copies of the table but querying the data at once, also counts. When using one table, which is in a real-time case continuously growing by updates, it is important that the data stays consistent and that no copies of the original data are going to be made in queries. So an efficient querying is very important.

3.5 Conclusions

This chapter started with a framework for a temporal GIS, which contains five technical requirements mentioned by Langran for a temporal GIS. The first requirement is a conceptual model. In 3.2, four different approaches to structure moving-point data are described. These

methods are mainly developed to structure moving point data in a DBMS context, designed for a special application.

In many cases, it is not known in advance what kind of queries are going to be done on a data set. So what is needed is a data structure that is sufficient for most of the queries, application independent. An approach for such a model is going to be presented in the next chapter.

Efficient querying (Langran's fifth technical requirement) is only possible if an efficient indexing method has been chosen that organizes the data in an optimal way for the query. If you would like to do a specific query, it is possible to make a copy of (a part of) the original data set, find the optimal indexing method and do the query. This method is in many cases fast, but making copies of tables can harm the consistency of your data. So, an intelligent, but perhaps less fast querying method is necessary on the original table, depending on how often the query is going to be posed and depending on the speed gain.

Once a data structure and indexing method has been chosen, the only way to optimize querying is by the query itself. Three ways of spatiotemporal querying have been described, first selecting on time and after that selecting on space or the opposite or selecting on space and time at the same time (3D querying). It is up to the query optimizer (which estimates the most efficient query plan), which method is going to be chosen. The influence of the user in choosing the most efficient way is by choosing a 2D structure for your data or a 3D structure and indexing method.

In section 3.3, some indexing methods are described. Because in Oracle 9i Spatial, the 2D and the 3D R-tree¹ indexing methods are the only implemented methods that are useful for spatiotemporal data, the choice of which indexing method is going to be chosen as the most efficient and the most flexible one is limited. The choice for 2D or 3D indexing depends on the queries that are going to be examined.

¹ Even when you make use of Linear Referencing with 4D data (X,Y,T,+M dimension of the linear referencing), only 3 dimensions can be indexed by the 3D R-tree.

4. A generic model for moving object DBMSs

In the previous chapter (section 3.2), four models to structure moving object data in a DBMS were presented. These models have some disadvantages. For instance, the model made by Vazirgiannis and Wolfson is especially made for road networks and the MOST-datamodel is mainly developed for objects that move freely in space with more or less constant speed like aircrafts, but is less sufficient for objects that do not move in straight lines. The other two approaches, described by Marchand et al and the one described by Meng and Ding have the disadvantage that they have a lot of redundant storage. To overcome these disadvantages, a new approach is introduced in this chapter. This model could be used for every purpose (this makes it generic) and it does not contain any redundant storage.

The previous chapter also concluded that if you do not know what queries are going to be done on the data set, a more generic model needs to be investigated that is perhaps less fast for a specific query, but is suitable for many queries. The question is now if this generic model is fast enough for many queries, making use of a suitable spatial index and an optimal query plan.

A moving object data model contains static parts (for instance information about the size or color of the moving object) and dynamic parts (the changing position during a time period). This generic model focuses on this dynamic part.

In this chapter, this generic model for moving point objects in a Geo-DBMS context will be introduced. In section 4.1, the components of this model will be introduced. Also some advantages and disadvantages of this model will be mentioned. Section 4.2 deals with views and materialized views in order to derive other ‘presentations’ of the same model. In the last section, 4.3 some conclusions will follow.

4.1 The principles of the model

It was shown in chapter 3, that a number of alternative models for moving point objects have been constructed. Two aspects can characterize these models. First, the time dimension is either separate or integrated with the spatial dimension (in such a case a 2D point and time as attribute becomes a 3D point in the spatiotemporal model). Secondly, for a single object the observations are either stored in separate records (with the sampled point in time) or in one record with a polyline attribute (kind of interpolation between the time samples). In the polyline case the time again can be separate or integrated with the spatial point data: 2D spatial polyline with separate attributes for timestamps or 3D spatiotemporal polyline) (see figure 4.1).

All of these four models can be converted to each other (and could in that sense be considered equivalent) and most likely this can be realized with DBMS views (using spatial operators). In practice they may differ with respect to dynamic behavior (suitable or not for dynamically growing data in case of real-time monitoring) and ease of use during analysis and visualization. This depends on the specific platform used for the implementation (e.g. Informix has time-series data type support, which is then more efficient than a separate record for every moment in time). Anyhow, from the conceptual point of view, the models are quite similar and we will use the most elementary to illustrate spatiotemporal modelling. The base table, not assuming any explicit sequence, looks like (in a kind of pseudo SQL):

```
create table mov_obj(id, t, position);
```

```
-- primary key is pair (id,t), position is the geometry and t is the time (for
instance in seconds as integer)
```

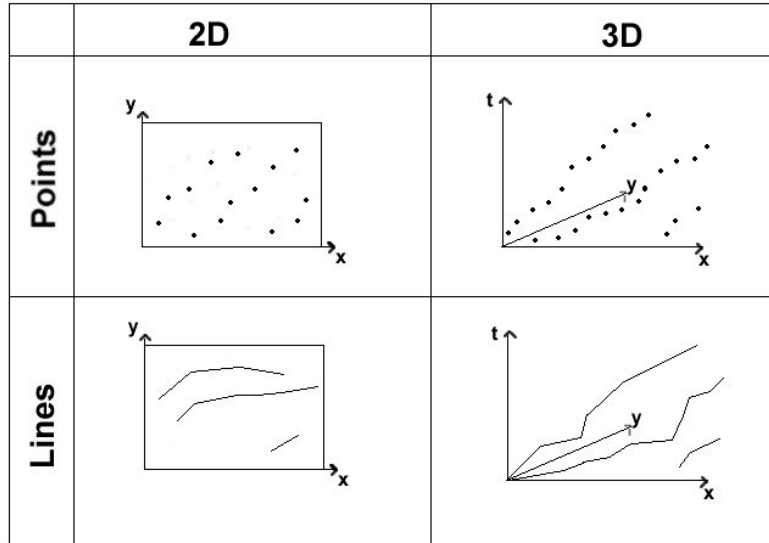


Figure 4.1 Top row with sampled time: 2D points (time separate) or 3D points (integrated time). Bottom row with interpolated time: 2D polylines (time separate) or 3D polylines (time integrated).

Based on this base table with 2D moving points and a separate time dimension, a number of views can be defined that could form the other 3 representations based on other geometries: the 3D points, the 2D lines and the 3D lines. This is examined below. More views are defined to have easy access to derived attributes such as speed and acceleration (see section 4.2).

```
create view mov_obj_3D_vw as --view with 3D points
select
s.id as id,
mdsys.sdo_geometry(3001, NULL,
mdsys.sdo_point_type(a.position.sdo_point.x, a.position.sdo_point.y, a.t),
NULL, NULL) as position
from mov_obj a;

create view trajectory_2D_vw as --view with 2D line segments
select
a.id as id,
mdsys.sdo_geometry(2002, NULL, NULL, mdsys.sdo_elem_info_array(1,2,1),
mdsys.sdo_ordinate_array(a.position.sdo_point.x, a.position.sdo_point.y,
b.position.sdo_point.x, b.position.sdo_point.y)) as position,
a.t as t_beg,
b.t as t_end
from mov_obj a, mov_obj b
where a.id=b.id and b.t=(select min(t) from mov_obj where t>a.t and id=a.id);

create view trajectory_3D_vw as --view with 3D line segments (with a more advanced PL/SQL
select --function, polylines could be returned)
a.id as id,
mdsys.sdo_geometry(3002, NULL, NULL, mdsys.sdo_elem_info_array(1,2,1),
mdsys.sdo_ordinate_array(a.position.sdo_point.x, a.position.sdo_point.y, a.t,
b.position.sdo_point.x, b.position.sdo_point.y, b.t)) as position
from mov_obj a, mov_obj b
where a.id=b.id and b.t=(select min(t) from mov_obj where t>a.t and id=a.id);
```

4.2 Views and materialized views

In this paragraph, the main principles of views and materialized views are described. First, in 4.2.1, the views are explained and relevant views in the generic model are given and 4.2.2 deals with a special type of view, the Oracle's materialized view.

4.2.1 Views

Views are customized presentations of data in one or more tables or other views. A view can also be considered a stored query. Views do not actually contain data. Rather, they derive their data from the tables on which they are based, referred to as the base tables of the views [16]. Like tables, views can be queried, updated, inserted into, and deleted from, with some restrictions. All operations performed on a view actually affect the base tables of the view. Views also provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table. They also hide data complexity and store complex queries. For users, views 'look' the same as tables.

In fact, the DBMS does not store the result of a query, but the view's definition in the data dictionary as the text of the query that defines the view. When you reference a view in a SQL statement, the DBMS executes the definition of the view and uses the answer for your SQL statement. The merged query will be optimized by the DBMS as if you issued the query without referencing the views. Therefore, the DBMS can use indexes on any referenced base table columns, whether the columns are referenced in the view definition or in the user query against the view.

Some examples of creating views, with respect to moving point data:

Find the next point of the same object in time:

```
create view move_obj_succ as
select t1.*, t2.t as next_t
from mov_obj t1, mov_obj t2
where t1.id=t2.id and t2.t=(select min(t) from move_obj where t>t1.t and id=t1.id);
```

Calculate the direction and the speed of the moving object (view on a view):

```
create view speed_obj_vw as
select t1.id, t1.t, t1.next_t, t1.position, dir=diff(t2.position-t1.position),
speed=distance(t2.position, t1.position)/(t2.t-t1.t)
from mov_obj_succ t1, mov_obj_succ t2
where t1.id=t2.id and t2.t=t1.next_t;
```

Calculate the acceleration of the moving object (view on view on view):

```
create view accel_obj_vw as
select t1.id, t1.t, t1.next_t, t1.position, t1.speed, accel=(t2.speed-t1.speed)/(t2.t-
t1.t)
from speed_obj_vw t1, speed_obj_vw t2
where t1.id=t2.id and t2.t=t1.next_t;
```

Using these views one can now derive statistic such as average speed. This can either be grouped by id (of vehicle), position and time. For grouping positions we assume a function (pos_group) to translate an xy(z)-coordinate in an encoding of a position group; e.g. parts of roads. Similar for time we assume a time group function (time_group) to get usable time units (e.g. a block of 15 minutes). One can also make combinations of grouping when computing averages (e.g. by position and time):

```
create view avg_speed_obj_vw as
select id, avg(speed) from speed_obj_vw group by id;

create view avg_speed_pos as
```

```
select position=pos_group(position), avg(speed) from speed_obj_vw group by
pos_group(position);

create view avg_speed_time as
select time=time_group(t, 15), avg(speed) from speed_obj_vw group by time_group(t,15);

create view avg_speed_pos_time as
select position=pos_group(position), time=time_group(t,15), avg(speed)
from speed_obj_vw group by pos_group(position), time=time_group(t,15);
```

Of course, one could compute all kinds of other statistics in a similar way; for example the minimum or maximum speed (for the same types of groupings) or compute average, minimum, maximum acceleration (and again grouped by the different options: id, time or position or combinations of these). Quite another type of view may be used to analyze how close the cars are together (as a possible indication of traffic jams). In this case the ordering of the base table data is a bit more difficult compared to ordering on the linear time scale (as the space is two-dimensional). However, we assume that the next car should be found ahead of the current driving direction (and also driving in the same direction).

```
create view dist_objs as
select p1.id1, p2.id, p1.t, p1.position, p1.speed, distance=length(diff(p1.position,
p2.position))
from speed_obj_vw p1, speed_obj_vw p2
where p1.t=p2.t and p1.dir = p2.dir and p2.id=
  (select closest(id, p1.position) from speed_obj_vw);
```

All these views may be nice from a functional point of view. However, without the proper storage and indexing the performance may be poor. Important aspects to consider are spatiotemporal clustering (so the physical ordering of the data) and spatiotemporal indexing (efficient selections of the record addresses based on spatiotemporal (range) queries). In Oracle the initial implementation would use an indexed organized table (on the key id, t) in order to obtain ordering of the data based on id and time. Further a 2D R-tree index (on position) or 3D R-tree functional index (on position and time) is used for initial spatiotemporal indexing.

4.2.2 Materialized views

Analysis may show that it is impossible to answer all (often used) queries based on a single physical ordering the base table and in the ultimate situation redundant data storage may be considered. Oracle offer ‘materialized views’ (not part of the SQL92 standard) to implement this in an effective manner. By default a materialized view is only refreshed on demand (by calling a specific Oracle refresh procedure). However, it is possible to specify that the materialized view must be refreshed automatically after the transaction on the base table is committed, for example:

```
create materialized view move_obj_succ_mv1
refresh fast on commit
as select t1.*, t2.t as next_t
  from mov_obj t1, mov_obj t2
 where t1.id=t2.id and t2.t=(select min(t) from move_obj where t>t1.t);
```

Especially in highly dynamic situations (rapid data growth) this is not without problems (as it may not be very efficient to update the materialized view after every transaction). In such a situation it may be more effective (depending on the application) to collect a number of transactions and to refresh the materialized view only periodically, for example (the first time it is refreshed after 45 minutes, then after every 30 minutes):

```
create materialized view move_obj_succ_mv2
refresh start with 'now+45 min' next '30 min'
```

```
as select t1.*, t2.t as next_t
   from mov_obj t1, mov_obj t2
  where t1.id=t2.id and t2.t=(select min(t) from move_obj where t>t1.t);
```

4.3 Final remarks

Tuning the generic model, by choosing the appropriate storage and index structures and (materialized) views, makes it efficient for a given application, that is, a set of typical queries for a given (static or dynamic) data set.

In the beginning of this chapter is stated that this model is generic, which means that it could be used for every purpose that deal with moving object data. So, also the model introduced by Vazirgiannis and Wolfson (see section 3.2) should fit in this model, or at least, all queries that can be posed on the Vazirgiannis and Wolfson model can also be posed on the introduced generic model. The Vazirgiannis and Wolfson model contains two static parts: the map and the moving object itself; and one dynamic part, the trajectory. The main characteristic of this dynamic part is that there is an object $T(O)$, which is the trajectory of object O , which consists of tuples [sequence_nr, id, (x,y), time, Boolean]. All this information is already available in the generic model. This model can be created from the generic model by sorting the base table on id and on time. Your created table will be nearly the same as described by Vazirgiannis and Wolfson, except that the sequence number and the Boolean are not available, but these can be added by using a PL/SQL function.

```
Create view mov_obj_VazWol as
Select * from mov_obj order by id, t;
```

It makes difference if your application deals with data that is already collected (post-processing) or with real-time data. In the first case, for instance indexing is much easier because the index (or indexes) only needs to be created once. In the real-time case, the data set is continuously growing, which needs special treatment of the indexing (which needs to be rebuild once in a while) and updating of the materialized views. How many times these materialized views have to be updated and how a fast index needs to be maintained, depends on your application, the amount of data and the users' intentions.

The described model is generic. To demonstrate this, the model is implemented for two applications, a post-processing data set and a real-time simulation. In the next chapters, 5 and 6, these cases are described and followed by conclusions.

5. Case I – Traffic seen from a helicopter (post processing)

In cooperation with groups for Photogrammetry and Remote Sensing from the Faculty of Aerospace Engineering and for Traffic Management from the Faculty of Civil Engineering and Geosciences within Delft University of Technology a DBMS has been populated with measurements of highway traffic during circumstances of congestion. The measurements are obtained by automatic analysis of image sequences that are taken with a digital high-resolution camera from a helicopter (see figure 5.1). A highway section of approximately 500m is monitored during an extended period of time (say 1 hr) with a recording frequency of 10 frames per second. On a crowded or congested highway this may lead to several millions of car observations.

The goal of this effort is to extract parameters for so-called microscopic traffic flow models from the observations by querying the database. Microscopic models are those that take individual car driver's behavior into account, (including, therefore, variability in behavior between different drivers), as opposed to macroscopic models, where cars are treated as uniform entities like molecules in a streaming fluid. Since microscopic models are not yet fully understood (they are subject of study in the Traffic Management group), the database system should offer the flexibility to extract a large and partly unforeseen variety of parameters. The parameters should reflect drivers behavior concerning, for example, acceleration and deceleration in reaction to maneuvers of the vehicle in front, lane changing behavior, gap acceptance, etc.

Except for doing research on traffic flows or getting experiences with automatic tracking of vehicles in photo series, this data file can also be the input for a case in this research, because positions (in pixels which are 0.1 meter on ground level) and timestamps (photo number) for each car are available after the vehicle detection step. This case could be used to test the generic model described in the previous chapter on a small data set where the data has been collected in advance. This means that the data set is not going to grow anymore and could be used to do analysis after the data is organized into the correct structure (post processing or data mining).

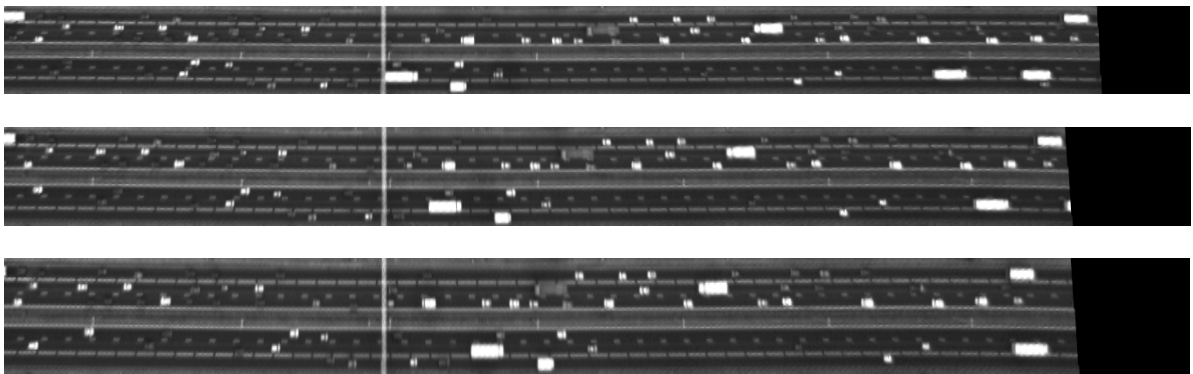


Figure 5.1 a series of three photos (approximately 1 second between each picture) with vehicles taken from a helicopter.

So, since the attributes (id, x, y, time) are available for every car, the generic model introduced in chapter 4 could be implemented for this data set in Oracle 9i Spatial. In section 5.1, issues that deal with the implementation of the generic model will be described. In 5.2, the implemented model will be reflected to the technical requirements, mentioned by Langran. In 5.3 and 5.4, two queries will be described and visualized. In 5.3, the query that calculates the vehicles that do not

keep two seconds distance from their predecessors and in 5.4 a query that calculates some traffic flow variables. On the end of this chapter, in 5.5, some conclusions will follow.

5.1 Implementation of the generic model into Oracle 9i Spatial

In chapter 4, the principles of the generic model are explained. The main idea is a base table and derived from this base table a set of views and materialized views. For this data set, a base table is created once, because all the data is collected in advance. The first choice that has to be made is whether using 2D points, 3D points, 2D polylines or 3D polylines as the base table.

In Oracle 9i Spatial, these four different geometry types are defined as following:

```
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(x, y, NULL), NULL, NULL)    --2D Point
SDO_GEOMETRY(3001, NULL, SDO_POINT_TYPE(x, y, t), NULL, NULL)        --3D Point
SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,2,1),
  SDO_ORDINATE_ARRAY(xi, yi, xi+1, yi+1))                          --2D Polyline
SDO_GEOMETRY(3002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,2,1),
  SDO_ORDINATE_ARRAY(xi, yi, ti, xi+1, yi+1, ti+1))              --3D Polyline
```

With an index on the geometry in the base table (e.g. a R-tree) and a limited amount of objects, these four base geometry types can be accessed in a fast way. On a view, it is also possible to create a spatial index via a functional index on the base table. On a materialized view on the other hand, spatial indexing is possible directly. So in this case, it might be useful to choose for a materialized view. These materialized views can be updated on demand. So the data model is flexible and can be accessed fast.

5.2 Reflection to Langran's technical requirements

According to Langran [8], a temporal GIS has five technical requirements: a conceptual model, treatment of aspatial attributes, data processing logistics, a spatiotemporal data access method and efficient algorithms to operate on the spatiotemporal data. Since the model is successfully implemented, the generic model could be reflected to Langran's technical requirements for a temporal GIS.

About the conceptual model has been spoken before. Vehicles are driving on a part of a highway. From a helicopter every 0.10 seconds, pictures of these vehicles are taken. So, there is a conceptual model of (car_id, x, y and time). The data is collected in advance, before it comes into the DBMS.

The third technical requirement of Langran's temporal GIS, deals with clustering, volume and error control. The first of these three, the clustering is more a matter of primary indexing (or storage structure of the table) and will be discussed in 5.2.2. The volume control is something that is important in this case, in spite of a small data set is used. The amount of data is now around 2300 records. But when during one hour, one kilometer with ten photos per second is going to be analyzed; the amount of data will grow to a very large amount of records. The third, error control is not within the scope of this research. Topics like lineage, completeness and logical consistency will not be discussed in this thesis. The other three requirements of Langrans temporal GIS will be discussed in this paragraph.

5.2.1 Treatment of aspatial attributes

A distinction can be made between the aspatial attribute "time" and the other aspatial attributes. First of all the "other aspatial attributes": In this case, the x_width and the y_width attributes are mentioned. These attributes deal with the size of the car. You can also imagine that you add for

instance attributes like color, kind of vehicle, driver information etcetera. If there are a lot of timestamps, objects and attributes, then it seems logical to store this information once and not on every timestamp. With a unique `car_id`, this data can then be made available, because they will stay constant during the time.

For the aspatial attribute time, there are four options:

- In the 3-dimensional case, time is used as a spatial attribute. This is, in many cases, not really correct. For instance, if you want to calculate distances, a distance between two timestamps is a 2-dimensional distance. Then, time should be treated as a normal attribute with special characteristics.
- You can store the attribute as an integer. It has the advantage that it is easy, because you can calculate with a decimal frame instead of hours, minutes and seconds, but it had the disadvantage that many applications (for instance ArcGIS 8.3) need time as a 'date' data type. This data type can also be calculated when using views.
- So, date is the third way of storing the attribute time. The advantage is that it is more easily to interpret hours, minutes etc than for instance 200 minutes from now. A disadvantage is that the "Date" data type can be interpreted in more than one ways. If you see for instance 09-07-2004, you do not know if this is July 9th or September 7th. In Oracle, the "Date" data type is internally stored as a number that you can access in the way you want (for instance "DD-MON-YY HH24-MI-SS" or "MM-DD-YYYY HH:MI").
- The last possibility is to store time as a sequence number like t_0, t_1, t_2 , etc. This can be used if the interval between t_i and t_{i+1} is the same for every interval and if the width of this interval is known. Implicitly this is the same as when you use integers.

These four options only deal with the implementation of the time-attribute. For a functional understanding of spatiotemporal DBMSs, these four options are the same, because they can all be transformed into another.

5.2.2 Indexing the moving object database

In paragraph 3.3, a lot of spatiotemporal indexing methods are discussed. Indexing methods (or spatiotemporal accessing methods) are one of Langran's five technical requirements for a spatiotemporal GIS. Most of these methods are based on the R-tree. The basis of the R-tree is that objects that are close to each other (spatially) are close to each other in the tree. In Oracle Spatial 9i, the R-tree is implemented. But it is implemented very deeply in the DBMS, so, that implementing other possibilities (variations of the R-tree) will be too much work for this thesis.

If you are using a spatiotemporal query, e.g. "Which objects were in polygon P during period $\langle t_i, t_j \rangle$?", you would like to have an indexing structure, that serves the needs of this query. So, a 3D R-tree seems the best, because objects that are close to each other in space and in time are in the same leaf node or at least in the same part of the R-tree.

For testing performance, the data set used in this case is too small. But one can describe how the indexing is implemented in Oracle Spatial. Deleting, inserting and updating the geometry of features will affect the query performance, because the pointers in the nodes of the R-tree (which automatically will be updated when objects are inserted, updated or deleted) that point to a place on the hard disc, do not work in the most optimal way anymore. If the increase in query time will be too high, the index needs to be rebuilt. Especially with real-time data sets, the data set is growing continuously. In that case, rebuilding the index is really necessary. More about this topic can be found in chapter 6, which deals with a real-time data set.

5.2.3 Operating on spatiotemporal data

Since the generic model is implemented, it is available to query. For often-used queries like speed and acceleration, views can be defined. Organizing the spatiotemporal data in a geo-DBMS has the advantage that you can use standard spatial queries like overlap and buffer. For instance, the question “which vehicles were on lane x at time t?” is a query where you can easily use the polygon overlap-function. In Oracle Spatial 9i, spatial queries can be formulated in two different ways:

- as a spatial operator (uses the spatial index), example:

```
select <attributes> from spatial_table a, spatial_table b
where sdo_relate (a.geometry, b.geometry,
'mask=anyinteract querytype=window')='TRUE';
```
- as a spatial function (does not use the spatial index), example:

```
select <attributes> from spatial_table a, spatial_table b
where sdo_geom.relate (b.shape, 'anyinteract', a.shape, 0.005)='TRUE';
```

Example:

For calculating the distance a vehicle has traveled between two successive measured points, the next statement has to be entered into SQL:

```
select a.id, b.t "t_beg", a.t "t_end",
sdo_geom.sdo_distance(a.position, b.position, 0.1) "distance"
from mov_obj a, mov_obj b
where a.id=b.id and b.t=(select min(t) from mov_obj where t>a.t and id=a.id);
```

Instead of entering “b.t=(select min(t) from mov_obj where t>a.t and id=a.id)”, a function like “NEXTTIME” could be implemented. NEXTTIME is one of the operators defined by Wolfson [24]. This could improve the querying of spatiotemporal data sets or could at least make the formulation of the query easier. In 4.2, an example has been given to calculate the next_t as an attribute in a view by making use of standard SQL (creation of the view mov_obj_succ).

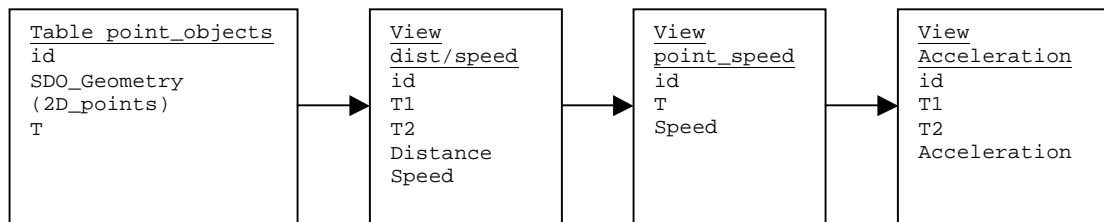


Figure 5.2 Root table (2D points) with three views, (1) the speed and distance between two points, (2) the speed in one point and (3) the acceleration between two points.

In this case, on a root table with 2D point objects, three views are created (figure 5.2):

- First, the one described above, this is the speed and the distance between two measured points. The geometry (position) of the objects could also be added to the view; but because this dist_view contains information between two points, the geometry should be a line-geometry.

```
create view dist_view as
select
a.id,
b.t as t1,
a.t as t2,
sdo_geom.sdo_distance(a.position, b.position, 0.1) as distance,
sdo_geom.sdo_distance(a.position, b.position, 0.1)/(a.t - b.t)*3.6 as speed
from mov_obj a, mov_obj b
where a.id=b.id and b.t=(select min(t) from mov_obj where t>a.t and id=a.id);
```

- Second, a view on the first view, where the speed in a measured point is calculated by taking the average of the speed before and the speed after the measured point (taken from the `dist_view`).

```
create view point_speed_view as
select a.id, b.t1 as t, (a.speed+b.speed)/2 as point_speed
from dist_view a, dist_view b
where a.id=b.id and a.t2=b.t1;
```

- Third, a view on a view on a view; the acceleration of the vehicle between two measured points. Taking the sum of the speed in two measured points and dividing it by the time between these points calculate this.

```
create view accel_view as
select a.id, a.t as t1, b.t as t2,
((b.point_speed-a.point_speed))/(b.t-a.t) as accel
from point_speed_view a, point_speed_view b
where a.id=b.id and b.t=(select min(t) from point_speed_view where t>a.t and id=a.id);
```

In the database, this syntax is stored and not the results. If you would like to use the results of this query, you can treat this like a table. An example, which vehicles were driving faster than 120 km/h and at which time:

```
select id, t1, t2, speed
from dist_view where speed>120 order by t1;
```

ID	T1	T2	SPEED
108	0	1	122.4
102	1	2	122.4
108	1	2	122.4
108	2	3	122.4
108	3	4	122.4
108	4	5	122.4
108	5	6	122.4
108	6	7	126
108	7	8	122.4
108	8	9	126
108	9	10	126
108	10	11	122.4

12 rows selected.

As shown above, with the generic model, a base table with a set of views and materialized views, operating on the spatiotemporal data is easy. In the next two paragraphs, two slightly more complex queries are described on the spatiotemporal data.

Other examples of spatial queries that can be done easily on a data set are for instance:

On which time, car A (with id=108) starts to accelerate:

```
SELECT t1
FROM accel_view
WHERE id=108 and t1=(select min(t1) from accel_view where accel>0);
```

On which place, the speed is the lowest?

```
SELECT a.speed, b.position
FROM dist_view a, mov_obj b
WHERE a.id=b.id and a.t=b.t
and a.speed=(select min(speed) from dist_view);
```

In the same way, numerous other queries can be done. Some of the more interesting queries are going to be discussed in more detail in the next sections.

5.3 Query: Keep two seconds distance from your predecessor

The Dutch government recently ran a campaign to keep two seconds distance from your predecessor on a highway. This was, of course, to avoid car crashes, but can also be a spatiotemporal query on this helicopter data set: Which cars do not keep two seconds distance from their predecessor? Or which cars are within two seconds distance in front of another car?

In section 3.4 is stated that queries could be solved in different ways. In this section, these three ways of answering the query above will be described. In the first part of this section, the way of querying looks like the left figure of figure 3.6 (select with fixed time) and in the second part, the query will be answered by using the right figure of figure 3.6 (select with fixed position). In the third part, the 3D querying will be described. This method looks the most like figure 3.7. In 5.3.4, visualization of this query will be the topic.

5.3.1 A spatial query, method 1

To answer the question described above, a spatiotemporal query can be used that is doing a selection in the time dimension before doing the spatial query. The first step is to define a rectangle in front of each car, where no cars are allowed. The width of this rectangle is the same as the width of the road (e.g. 2 meters) and the length of each rectangle is the same as the speed (in meters/second) of the vehicle multiplied by two seconds (see figure 5.3).

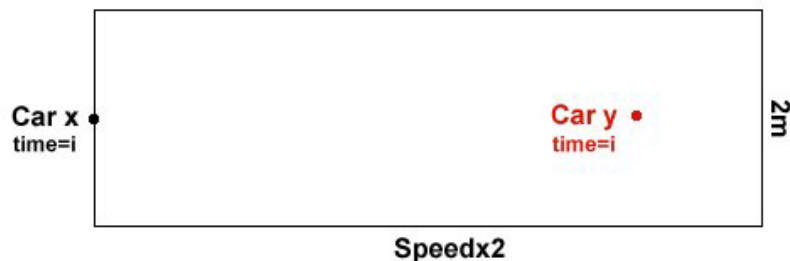


Figure 5.3 Calculating a rectangle gives the area where no other cars are “allowed” at the same time. Car x does not keep enough distance to car y at time=i.

In the SQL statement described below, can be seen that this rectangle has been build for a straight road and the reference system of the coordinates are x-axis (road length), y-axis (road width). For more complicated situations, the calculation of the coordinates of these rectangles is a little more complicated, because also the driving direction is necessary. The width of this rectangle (speed_box) is then perpendicular to the driving direction. You have to calculate such a rectangle on every timestamp and for every car.

```
create materialized view speed_box --build materialized view with rectangles
build immediate as
select a.id, a.t, b.point_speed,
mdsys.sdo_geometry(2003, NULL,NULL,
mdsys.sdo_elem_info_array(1,1003,3),
mdsys.sdo_ordinate_array(a.position.sdo_point.x, (a.position.sdo_point.y-10),
(a.position.sdo_point.x+b.point_speed/0.18), (a.position.sdo_point.y+10))) as position
from mov_obj a, point_speed_view b
where a.id=b.id and a.t=b.t;
```

The second step is to use the SDO_GEOM.RELATE function or the SDO_RELATE operator to check whether another vehicle (point object) is in that area at that time:

```

select a.id, b.id, b.t
from speed_box a, mov_obj b
where a.t=b.t and a.id<>b.id
and sdo_geom.relate (b.position, 'inside', a.position, 0.005)='TRUE'; -- function

select a.id, b.id, a.t
from speed_box a, mov_obj b
where a.t=b.t and a.id<>b.id
and sdo_relate (a.position, b.position, 'mask=inside querytype=window')='TRUE';
-- operator

```

The main difference is that the spatial operator needs a spatial index and the spatial function does not. Unfortunately, the querying is not very fast. For 2300 objects, the query-time is about 3 minutes (for both the spatial function and the spatial operator). The performance should be much better. So looking into the query-plan, to see what happens might be useful:

PLAN_TABLE_OUTPUT

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		359	62107	14
* 1	HASH JOIN		359	62107	14
2	TABLE ACCESS FULL	MOV_OBJ	2300	103K	4
3	TABLE ACCESS FULL	SPEED_BOXES	2300	285K	8

Predicate Information (identified by operation id):

```

1 - access("B"."T"="A"."T")
    filter("B"."ID"<>"A"."ID" AND
           "MDSYS"."SDO_RTREE_RELATE" (B."POSITION",A."POSITION", 'mask=ANYINTERACT
           querytype=window')='TRUE')

```

Note: cpu costing is off

As you can see in the query-plan described above, for both tables, a “full-table-scan” is used, which means that the query-optimizer does not use the spatial index. Even when an index on the id and time attributes is used, the spatial index is ignored. So, for this small amount of records, the Oracle-query-optimizer thinks that it is more efficient to ignore the index. By making use of the index, the query-response times should be much lower and probably with a larger amount of data, the query-optimizer will make use of the index, because then a full-table-scan is more expensive in terms of performance.

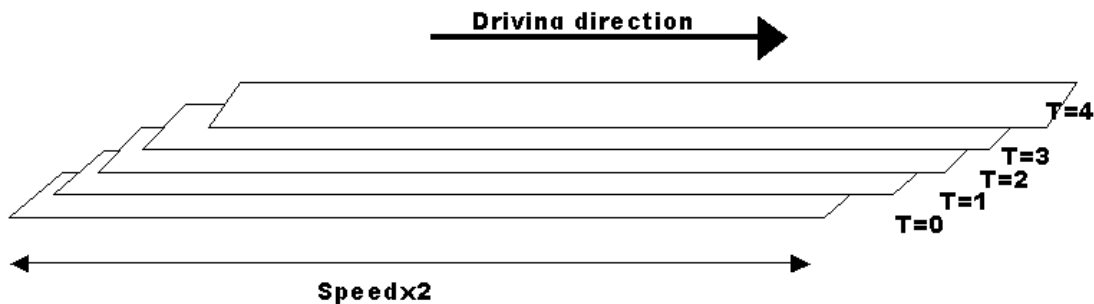


Figure 5.4 the areas in front of the cars overlap, so an effective 2D R-tree indexing is not possible.

In figure 5.4 you can see an example of the geometry of all rectangles (like figure 5.3) in front of one car. As you can see, these rectangles are overlapping each other. This makes the indexing very difficult and does not make the querying much faster. This is probably another reason why the query-optimizer does not use the spatial index. So, for a very large data set, it is recommended to find another method or to split the data set into smaller parts.

As described here, the query was in 2D. Doing the same query in 3D is much more difficult because the rectangles (see figure 5.3) will become boxes. This is not a standard Oracle 9i Spatial primitive (except for the boxes in the 3D R-tree) and needs to be implemented to do this query. Another shortcoming of Oracle 9i Spatial is that the spatial functions and operators do not work in 3D. The only operator that works in 3D is the SDO_FILTER, which uses the spatial index to identify either the set of spatial objects that are likely to interact spatially with a given object (such as an area of inters), or pairs of spatial objects that are likely to interact spatially. Objects interact spatially if they are not disjoint. In chapter 6, a method is described to solve this problem.

5.3.2 Spatial querying, method 2

In the previous section, first selecting all the objects that were on the same time on the road as the studying object and then querying the 2D space have solved the query finding the objects that are within two seconds distance from each other. In this section, the results for the alternative querying method are going to be discussed; first selecting in the spatial dimension and secondly in the time dimension. The query can be rewritten as “Which objects are on position x,y between 0 and 2 seconds from now. Because a vehicle does not come exactly on the same position as its predecessor, an interpolation is necessary in the driving direction (by making use of the 2D line segments (trajectories), which is one of the materialized views described in chapter 4.1). The position x,y needs to be considered as a line segment perpendicular to the driving direction (see figure 5.5). So, the query will become an intersection between two sets of line segments.

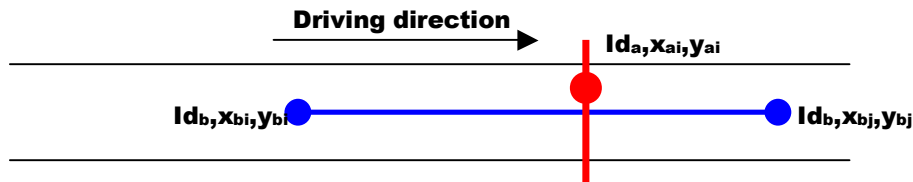


Figure 5.5 Finding the objects that were on position x_{ai}, y_{ai} at time i results in an intersection of two sets of line segments.

The following statement will create the materialized view with line segments perpendicular to the driving direction (in this case along the x-axis):

```
create materialized view mov_obj_ppd_lin
build immediate as
select t.id as id, t.t as t,
mdsys.sdo_geometry(2002, NULL, NULL,
mdsys.sdo_elem_info_array(1,2,1),
mdsys.sdo_ordinate_array(t.position.sdo_point.x, (t.position.sdo_point.y-10),
t.position.sdo_point.x, (t.position.sdo_point.y+10))) as position
from mov_obj t;
```

In the query, the trajectories need to be found that have a starting timestamp, which is between 0 and 2 seconds after $t=i$, where i is the timestamp of the line perpendicular on the driving direction. In SQL, the query will look like:

```

select s.id, t.id, t.t
from trajectory_2d_mvwm s, mov_obj_ppd_lin t
where
s.id<>t.id
and s.t>t.t and s.t<(t.t+20)
and sdo_relate(s.position, t.position, 'mask=anyinteract querytype=window')='TRUE';

```

The answer will be retrieved in around 1 minute and the results are correct. The answers were a little bit different than the query described in 5.3.1, but that is because the query was formulated in a different way. This querying method is faster than the one in 5.3.1. If you take a look at the query plan Oracle uses, you can see that the results are still not satisfactory, because only one of the two spatial indexes is used:

PLAN_TABLE_OUTPUT

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	CREATE TABLE STATEMENT		301	2330K	11823 (0)
1	LOAD AS SELECT				
2	NESTED LOOPS		301	2330K	11823 (0)
3	TABLE ACCESS FULL	MOV_OBJ_PPD_LIN	3594	13M	8 (0)
* 4	TABLE ACCESS BY INDEX ROWID	TRAJECTORY_2D_MVW	1	3956	11823 (1)
5	DOMAIN INDEX	ITRAJECTORY_2D_MVW			

Predicate Information (identified by operation id):

```

4 - filter("S"."ID"<>"T"."ID" AND "S"."T_BEG">"T"."T" AND
           "S"."T_BEG"<"T"."T"+20)

```

A conclusion that can be drawn is that the query optimizer in some cases chooses not to use the spatial index. Why it does not choose the spatial index is not clear, so this could be a recommendation for further research. It is expected that the spatial index will be used when the data set is much bigger.

5.3.3 Spatial querying, method 3

The third method for spatiotemporal querying is querying in the spatial and the temporal dimensions at the same time. The rectangles in front of each car (see figure 5.3) will now become boxes with the time as height. The main problem is that boxes do not exist in Oracle 9i Spatial. Arens [2] has found a solution to implement a 3D primitive into a geo-DBMS. In his thesis, he proposes an extension to Oracle Spatial that describes a 3D polyhedron as a set of 3D faces. The implementation should be:

```

INSERT INTO table (id, geometry) VALUES (2,
mdsys.sdo_geometry(3008, NULL, NULL,
mdsys.sdo_elem_info_array(
25,1006,1, 29,1006,1, 33,1006,1, 41,1006,1, 45,1006,1),
--25 is the first face, the first 24 are used by the coordinates
mdsys.sdo_ordinate_array(
1,1,0, 1,3,0, 3,3,0, 3,1,0, 1,1,2, 1,3,2, 3,1,2, 3,3,2, -- the coordinates
1,2,3,4, --bottom face starts at index 25
8,7,6,5, --top face starts at index 29
1,4,8,5, --front face starts at index 33
2,6,7,3, --back face starts at index 37
1,5,6,2, --left face starts at index 41
4,3,7,8 --right face starts at index 45
)));

```


This means that elements of `sdo_geometry` are:

- `sdo_gtype` = 3008 (3D polyhedron)
- `sdo_srid` = NULL (no spatial reference system)
- `sdo_point` = NULL (no point type)
- `sdo_elem_info` = 6 times x,1006,1 (exterior polyhedron boundary, x is where the face starts)
- `sdo_ordinates` = 8 coordinate triplets and 6 face descriptions

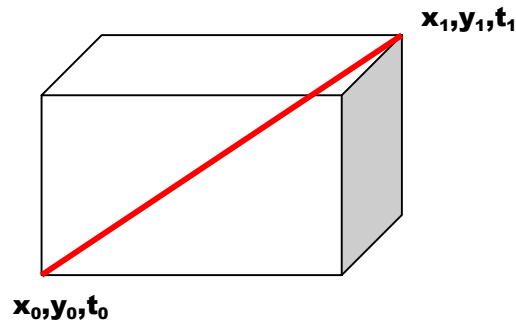


Figure 5.6 A query with a box is implemented as a SDO_FILTER operator with a line, which bounding box is the same as the box needed for the query

The only problem is now that Oracle does not recognize 3008 as `sdo_gtype`. For this case it is not important if a real box is implemented, because the only operator that works in 3D is the SDO_FILTER operator. So, another object, that has the same bounding box, as the box described above should also work. Therefore, implementing a 3D line is also a possibility. This line should look like figure 5.6.

The 3D lines that define the box in front of each car are created as:

```
create materialized view speed_box_lin
build immediate as
select s.id, s.t, mdsys.sdo_geometry(3002,NULL,NULL,
mdsys.sdo_elem_info_array(1,2,1),
mdsys.sdo_ordinate_array
(s.position.sdo_point.x, (s.position.sdo_point.y-10), s.t,
(s.position.sdo_point.x+t.point_speed/0.18), (s.position.sdo_point.y+10), s.t)) as
position
from mov_obj s, point_speed t where s.id=t.id and s.t=t.t;
```

And the 3D spatiotemporal query (which makes use of the SDO_FILTER operator) looks like:

```
select a.id, b.id, a.t
from speed_box_lin a, mov_obj_3d_vw b
where
b.t=a.t and b.id<>a.id and
sdo_filter (b.position, a.position, 'querytype=window')='TRUE';
```

This statement implies that in the 'mov_obj_3d_vw' time is also an attribute. Otherwise instead of 'b.t', 'b.position.sdo_point.z' must be entered. Both materialized views are indexed spatially with a 3D spatial R-tree.

The results of this query are (of course) the same as the query described in 5.3.1, but the query-response times are not much faster (about 2 minutes). If you look into the query-plan, you can see that the query-optimizer also chooses to use a full-table scan for this query.

PLAN_TABLE_OUTPUT

Id	Operation	Name	Rows	Bytes	TempSpc	Cost
0	SELECT STATEMENT		34	262K		267
* 1	HASH JOIN		34	262K	3168K	267
2	TABLE ACCESS FULL	MOV_OBJ_3D_VW	817	3156K		3
3	TABLE ACCESS FULL	SPEED_BOX_LIN	3431	12M		8

Predicate Information (identified by operation id):

```

1 - access("B"."SYS_NC00008$"="A"."T")
    filter("B"."ID"<>"A"."ID" AND
           "MDSYS"."SDO_RTREE_FILTER"("B"."POSITION",A"."POSITION",'querytype=window')
           ='TRUE')

```

Note: cpu costing is off

It can be concluded that querying in 3D (2D space + time) is in this case not faster than querying in 2D. A reason for this is that the query-optimizer does not use the spatial index.

5.3.4 Visualization

In ‘Cartography, visualization of spatial data’ [7], Kraak and Ormeling discuss the use of dynamic variables opposed to traditional graphic variables, which are used to represent spatial data within individual frames. According to them an appropriate dynamic graphic can be used if the spatial data it represents is dynamic by nature. But as Bertin has stated, the user should be aware of: ‘movement only introduces one additional variable, it will be dominant, it will distract all attention from the other (graphical) variables’ [3]. MacEachren [9] has defined the dynamic variables: duration, order, rate of change, frequency, display time and synchronization. These dynamic variables can be seen as additional tools to design an animation.

Knowing this theory, ESRI has provided an extension on ArcGIS to handle spatiotemporal data: the Tracking Analyst [26]. Its functionality is comparable to its predecessor, available as an extension to ArcView 3.x, although the possibilities to serve real-time data to the application are now part of ArcIMS. It is possible to display point and track data (real-time and fixed time) by temporal symbology renderers (shape and size), symbolize time by color (show the aging of data), actions (based on attribute or spatial queries) and highlighting. Besides, the user is in control by the interactive playback manager to start, pause, stop and rewind the animation (for an example, see figure 5.6). Note that the colors of the objects are based on their id (which does not add much information), therefore another attribute may be more interesting to use as color attribute basis; e.g. acceleration (red: slow down, green: speed up, and yellow: about equal speed).

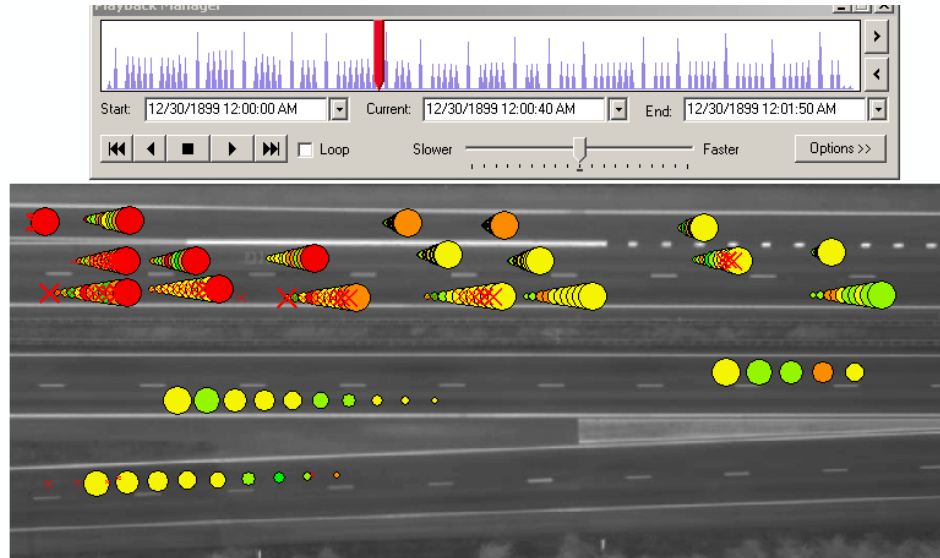


Figure 5.6 the last 10 timestamps of cars with crosses if they do not keep enough distance. The green dots are cars that accelerate and red if they decelerate; yellow for equal speed.

As with most cartographical visualizations the dynamic map could be used for exploratory data analysis to reveal unknown or difficult to recover information from the data alone. One could state database queries to expose this kind of information, but some more qualitative and descriptive facts like trends, are perhaps to be appreciated by carefully inspection of the animation. For this, the dynamic variables should be appropriate and careful used, with the notification of Bertin in mind.

In an animation (see also figure 5.6), vehicles are displayed as circles that are moving on a road. The last 10 timestamps are visible (the last timestamp is the biggest and 10 timestamps before the smallest). There's a cross on the circle when a driver did not keep enough distance to his predecessor (remember that 10 timestamps is one second).

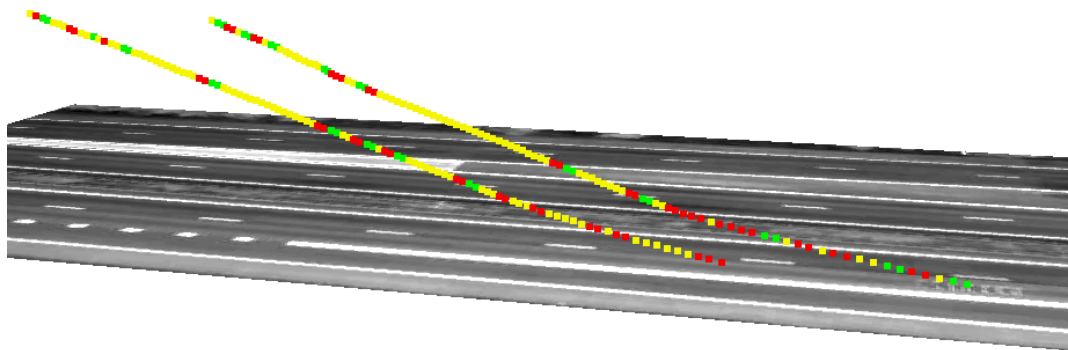


Figure 5.7 Visualization method: Dots (two vehicles) plotted in 3D (time is third dimension). With red dots, the vehicle is decelerating, green accelerating and yellow if the speed is more or less equal. Another option is to make the distance to the predecessor as third dimension.

Another option is to visualize in 3D, where parts of the trajectory become red when the vehicle is decelerating and is green when the vehicle is accelerating; yellow if the speed is more or less constant (see figure 5.7). This can be done for instance with ArcScene, also a part of ESRI's ArcGIS.

5.4 Query: Calculating traffic flow variables

The data set used in this case was originally meant for research on traffic flows. These variables are used to analyze and test existing traffic models. The most common used traffic flow variables are [6]:

- Mean Time Headway: The mean period between the passing moment of the preceding vehicle and the vehicle considered on a certain location.

$$\bar{h} = \frac{1}{n} \sum_{i=1}^n h_i = \frac{T}{n}$$

- h_i is the time between two passing cars on a road segment
- T is the length of the period
- n is the number of cars on the road segment.
- Mean Distance Headway: The mean distance between the rear bumper of the preceding vehicle and the rear bumper of the considered vehicle at a specific moment in time on a road segment.

$$\bar{s} = \frac{1}{m} \sum_{j=1}^m s_j = \frac{X}{m}$$

- s_j is the distance headway of the j th vehicle
- X is the length of the road segment
- m is the number of vehicles present on the road segment.
- Intensity: number of vehicles in a given time interval on a specific location.

$$q = \frac{n}{T} = \frac{1}{h}$$

- n is the number of vehicles
- T is the length of the period of time
- Density: number of vehicles per unit of length on a specific road segment.

$$k = \frac{m}{X} = \frac{1}{s}$$

- m is the number of vehicles
- X is the length of the road section
- Mean speed (instantaneous): mean speed on a road section at a given moment.

$$u_M = \frac{1}{n} \sum_{i=1}^n v_i$$

- v_i is the speed of vehicle i at a given moment
- n is the number of vehicles on a road section at a given moment.

The variables mean speed, density and mean distance headway are considered on a road section (for instance one lane between kilometers a and b) at a specific moment in time. The variables intensity and mean time headway are considered on a cross section of a road or lane for a certain period of time.

It would be very useful for traffic flow researchers to have these variables in the geo-DBMS if you also keep the data in the geo-DBMS. So, with a query (in a view), these variables can be calculated.

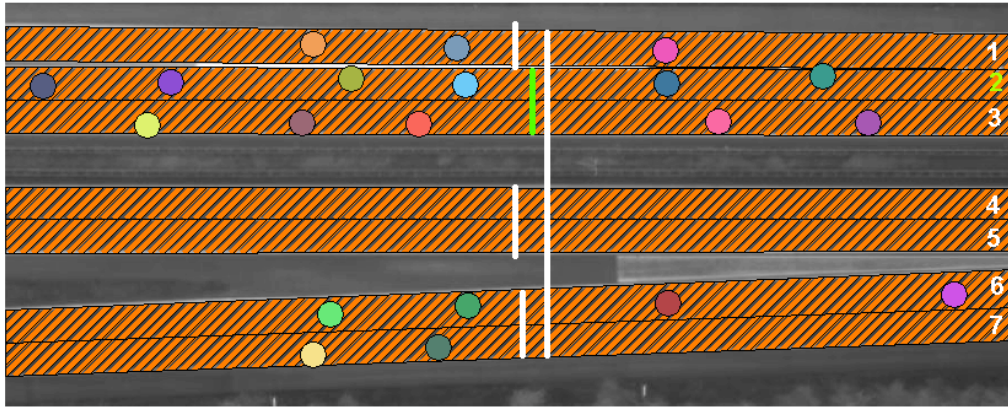


Figure 5.8 The variables are calculated on the lines and the lanes

For the variables mean speed, density and mean distance headway, polygons are defined that are the same as the lanes. So for every timestamp on every lane, these variables were calculated. An example for lane number 2 and after 12×0.1 seconds (see figure 5.10):

```
select
avg(sdo_geom.sdo_area(b.position,0.1))/((count(a.id))*100) "MDH (m)",
count(a.id) / (avg(sdo_geom.sdo_area(b.position,0.1))*0.01) "Density (veh/m)",
sum(c.point_speed)/count(c.id) "Mean_speed (km/h)"
from mov_obj a, lanes b, point_speed c
where b.lane_nr=2
and a.t=12
and a.id=c.id
and c.t=12
and sdo_relate (a.position, b.position, 'mask=anyinteract querytype=window') = 'TRUE';
```

The results:

MDH (m)	DENSITY (veh/m)	MEAN_SPEED (km/h)
42.0466667	.023783098	16.2

Also the other two variables, mean time headway and intensity can be calculated, with such a query (where the time period T is 10 (=1 second)). The mean time headway can only be calculated when the number of vehicles is not equal to zero:

```
select count(a.id)/1 as intensity (veh/s), 1/count(a.id) as MTH (s)
from trajectory_2d_vw a
where a.t1>0 and a.t1<10 and
sdo_relate(a.position, mdsys.sdo_geometry(2002,NULL,NULL,
mdsys.sdo_elem_info_array(1,2,1),
mdsys.sdo_ordinate_array(450,240,450,302)), 'mask=anyinteract querytype=window')='TRUE';
```

These variables are calculated on lines (like detection lines on a highway). The number of vehicles passing such a line is calculated. So for every line, the two variables are calculated. The results for the highlighted line (figure 5.8) are:

INTENSITY	MTH
(veh/s)	(s)

.818181818	1.22222222

5.4 Conclusions

In this chapter, the results of the first case study are discussed. The generic model, as described in chapter 4, is successfully implemented in Oracle 9i Spatial and loaded with test data from this test case. With a normal 2D R-tree, the data is spatially indexed and with a small set of views, the data is queried. Simple queries like “maximum speed” or “time of acceleration” can easily be done. Two more complex queries are discussed, calculating the vehicles that are within two seconds distance from their predecessor and calculating variables that are used in traffic flow modeling.

Querying is possible in three different ways:

- first querying in the spatial dimensions and after that in the time dimension,
- the second method is first querying in the time dimension and after that in the spatial dimensions
- and the third method is querying in 3D, where the spatial and the temporal dimensions are queried at once (see chapter 3).

These methods are compared in this case study. The query “Which objects keep less than 3 seconds distance from their predecessors?” is posed in three different ways, “Which objects are in two seconds distance from vehicle A?”, “Are there any objects on position x_a, y_a between now and two seconds?” and the third way was the first query in 3D (2D space + time). The answers for the first and second method were not exactly the same, but that is because the queries were not exactly the same.

But what differ are the query response times. For the first and the third method, the query optimizer does not use the spatial index and for the second method, only one spatial index is used and because of that much faster. It is expected that when the number of objects grow, the query optimizer will make use of the spatial index, because a full table scan will be more expensive in that case in terms of performance. When the query optimizer makes use of the spatial index and when it does not, could be a recommendation for further research.

A choice has to be made, which of the four representations, based on four different geometries is to be used as the base table. The 2D points, the 2D lines, the 3D points or the 3D lines. If one is chosen (for instance 2D points), the other primitives can be derived in materialized views.

The choice also depends on whether you would like to use 2D or a 3D spatial index (which could also be functional index on a 2D location and time). A 3D index should be better, because then the objects that are close to each other in space and time are close to each other in the R-tree. This is not tested.

The third consideration is the spatiotemporal queries, which you would like to use. In Oracle 9i Spatial, the number of 3D queries is limited (only the SDO_FILTER is available in 3D) and the number of 3D primitives is limited (for instance, a box is no primitive in Oracle 9i Spatial; 3D boxes are only possible in indexes, not as a real primitive).

With the ArcGIS 8.3 Tracking Analyst, the moving objects can be animated in an effective way. By careful inspection of the animation, a lot of information can be derived from the information, like the vehicles that accelerate or the vehicles that are driving too close to their predecessors.

6. Case II - Taxi cabs in Rotterdam (real-time)

The Advise Office for Geo-Information and ICT (AGI), one of the parts of the Dutch Ministry of Traffic and Public Works, has a data set available for research, where GPS tracking data is stored of a number of taxis operating near Rotterdam. For a period of two years, data has been captured and stored in ASCII files. For this case, a set of one week tracking data is used. The tracking data has already been preprocessed, which means that the records, which for instance only contain zeros as a result of too few satellites, are deleted from these files. So, the used data set is a raw data file, where all “not possible measurements” are deleted.

Every two or three seconds, a GPS point is stored on a batch in the taxi, which has been read out into an ASCII file. So, each batch number stands for a taxi number. All these files are inserted into one table in Oracle 9i Spatial, which results in a table with more than 250.000 records (about 60 taxis during one week). Each record contains the following attributes:

```
SQL> describe taxidata;
Name                                         Null?      Type
-----
POSITION                                     MDSYS.SDO_GEOMETRY
T                                              DATE
TAXI_ID                                     NUMBER(8)
```

The original data set contained much more information about the taxi itself, but that is of less importance in this case. With this table, the first part of the generic model is a fact. Internally in the DBMS, the DATE data type is treated as a number that corresponds with time in (metric) days.

The goal of this case is to investigate how the generic model works for a real-time case and how indexing works for 2D and 3D (2D location + time) with a continuously growing data set. Also querying spatiotemporal data in 2D and 3D will be an issue in this case.

Because the goal of this chapter deals with real-time data and because all the data is collected in advance, the data has to be simulated as real-time. Section 6.1 will describe how the real-time situation is simulated. Section 6.2 will deal with indexing issues. Indexing is important, because it gives faster access of your data. Especially in real-time, indexing is difficult and has to be handled with great care. Querying the data set will be discussed in 6.3 and 6.4. Because the data set is big, the performance of the geo-DBMS with the spatial index could be tested. Section 6.3 deals with 2D querying and 6.4 with querying in 3D, which needs a special treatment because of the limitations of Oracle 9i Spatial. In the last section 6.5, some conclusions are derived from this test.

6.1 Real-time simulation

The data used for this case is collected in 1998. To make this case “real-time”, the data has to be simulated as being real-time. ESRI will soon release its “ArcIMS Tracking Server” [27]. The ArcIMS Tracking Server provides a solution for collecting and sending real-time data from many data sources and formats to Web and desktop clients. The Tracking Server contains two functional elements: the Real-Time Message Server and the Real-Time Web Mapping component. These elements work together to collect and distribute real-time data to the people who use it on the Web and desktop clients [4].

The main goal of using ArcIMS Tracking Server in our application is simulating a real-time situation in such a way that a table in Oracle 9i Spatial will be filled for as long as the simulation exists. In this way, the index and the generic model could be tested and bottlenecks in spatiotemporal modeling in a real-time situation could be found. Unfortunately, the ESRI ArcIMS Tracking Server is not released at the moment of writing this thesis. Therefore, another solution has to be found.

The solution is a PL/SQL built application, which selects every 10 seconds objects from the original table and inserts this into another table. The selected objects are objects within a time interval of 10 seconds. This time interval shifts while time passes. In this way, there is a continuously growing table with spatiotemporal data.

```
insert into taxi_temp2                                -- the new growing table
(id, position, t)
select
  id, position, t
from
  taxidata                                           -- the original data set
where
  t < (current_date + 10 / (24 * 60 * 60) - 2082.6) and
  t >= (current_date - 2082.6);
-- 2082.6 is an initiation value, which is the difference between
-- now and the first data point in days. The value 10 / (24 * 60 * 60) means
-- 10 seconds. So data is selected in the interval [now-init_value,
-- now-init_value+10sec>
```

This SQL statement is repeated every 10 seconds by using a “shell-script” that sleeps for 10 seconds (see appendix B) and then posing the above described SQL-statement. So the new table “taxi_temp2” will on the end of the simulation be a copy from the original table, at least in the 2D case. The chosen base table contains 2D data. So for testing the 2D index, this real-time simulation suffices. For the 3D case, the script above looks a little bit different (see appendix B),

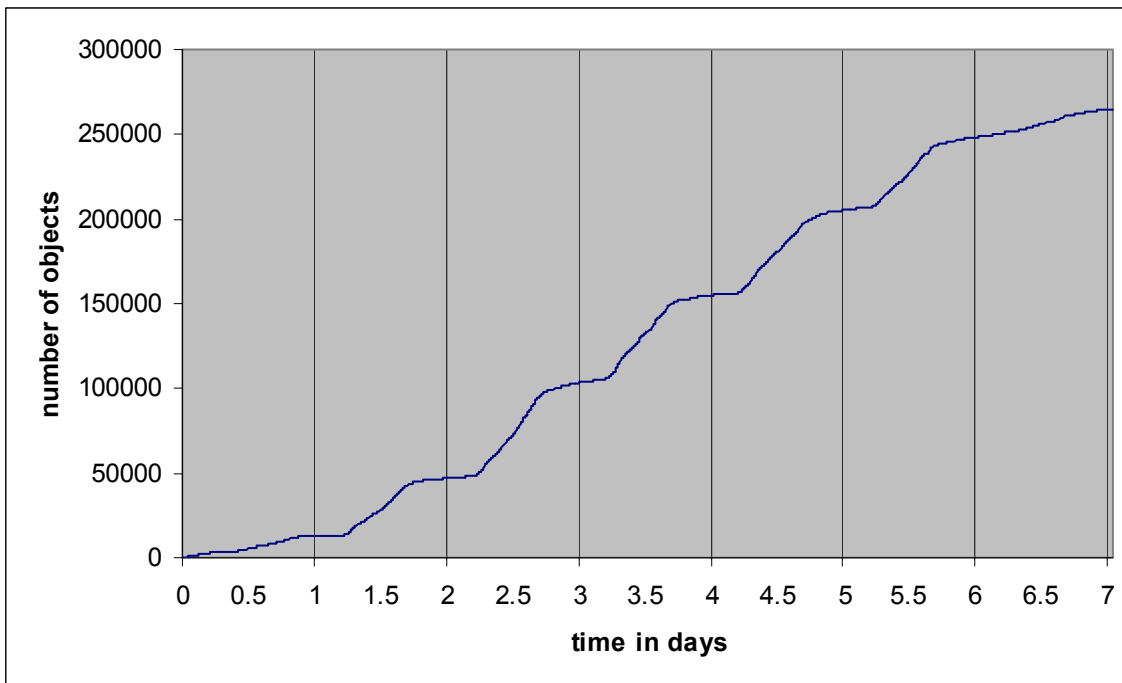


Figure 6.1 The growing of the table in relation to the time. The number 1 on the x-axis means at 12:00 midnight after one day.

because the “position” has to be a 3D coordinate, where the third coordinate is “time”, not as a “date”-data type, but as a number. An alternative is to use a functional index, where the x,y-position of the vehicles and the time dimension are treated separately in the model, but are combined in the index.

When this real-time simulation is used, the data set that is the result after seven days of growing does not contain the same amount of records as the original table. This is because the “shell-script” sleeps for ten seconds, and then it is inserting new records in the table (which costs time) and sleeps again for 10 seconds. So 10 seconds of data growth is put into the growing table in a time that is a little bit longer than 10 seconds. So there are some gaps in the growing data set. But this has no influence on the rest of the results, because there is still a continuously growing data set which is nearly as large as the original data set.

In figure 6.1, the growing of the table in relation to the time is given. You can see the difference between day (line is more vertical) and night (line is more horizontal). This pattern is shown every day.

6.2 Real-time indexing

Now there is a continuously growing data set. Before spatial querying, indexing the continuously growing data set is necessary, because it improves query times. The question is now, how often is rebuilding the spatial index necessary.

Not every time a record is inserted into the growing table, the index has to be rebuilt. It depends on the parameters Oracle uses for the number of leaf nodes and the number of entities allowed in each leaf node. Every time, a leaf node has to be split or if the depth of the R-tree has to grow, the pointers in the index that refer to the clustering of the data on the hard disc will be involved. Oracle uses a number that gives an indication of the R-tree quality. This number increases while inserting records into a table. Oracle compares this quality number after inserting a record with the quality number after updating the R-tree. If the quality number increases with 50% (this is the default), Oracle ‘advises’ to rebuild the index.

The quality parameter can increase enormously if a new object is inserted into the table that is outside the bounding box of one of the nodes in the R-tree or if the number of objects in a leaf node is too high, so that the index needs to go one level deeper.

```
select
sdo_tune.rtree_quality('user','index_name'),
from dual;
```

With the SQL statement above, the R-tree quality is given. Oracle just ‘advises’ to rebuild the index, but does not do this automatically. With the script, given in appendix B, the index quality number will be checked and the R-tree will be rebuilt if necessary. Every minute this script is executed, for getting an overview of the index during the test.

Results for the 2D and 3D R-tree

It is expected that on the first part of the test, the index have to be rebuild often. Because the tree is not very deep, the numbers of objects that will fit into the leaf nodes are limited. When the R-tree is going to be deeper (more levels of non-leaf nodes), the number of objects that will fit in the leaf nodes will grow and the tree will be more stable.

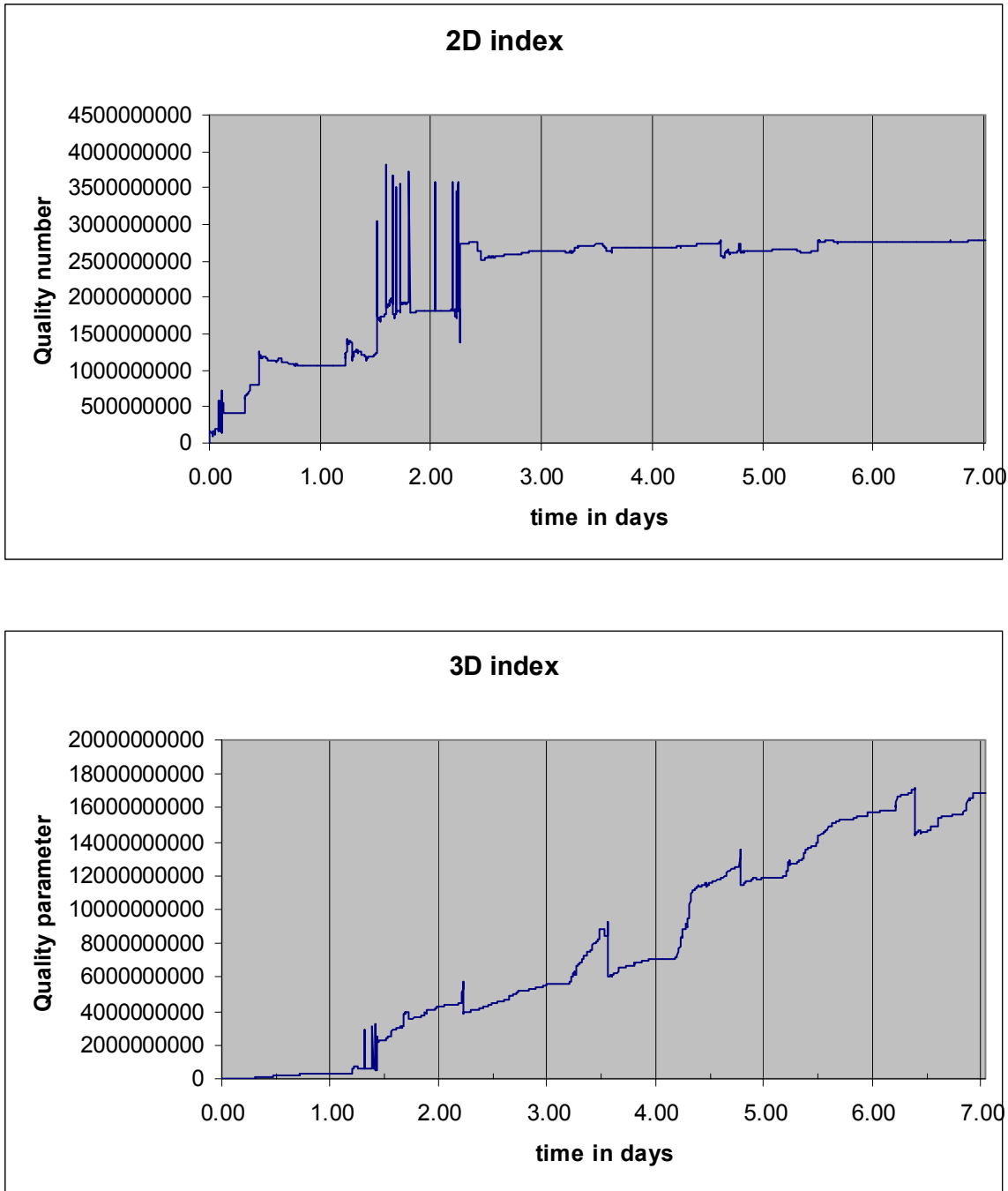


Figure 6.2. Above the Oracle quality parameter during the 2D test. The index is rebuilt only in the beginning of the test, later it stays more or less constant. Under, the behaviors of the 3D index, where the quality parameter grows during the test, because the time dimension grows.

The expectations of the test with the 3D index differ from the 2D case. Because the time dimension grows continuously, the result of the 3D test should not be the same as the result from the 2D test. It depends on the scale of the time parameter in the 3D index, how many times the index needs to be rebuilt. If the time parameter has a large scale (for instance you use minutes as unit), the 3D space will grow very fast in the time-dimension. If you use for instance days as a

unit (which is actually used), the 3D space will not grow so fast, so it is not necessary to rebuild the index that often.

In figure 6.2, you can see how the index parameter behaves during the test and demonstrates that this is according to the expectations. In figure 6.3, you can see the data grow in 2D. In orange, the points objects are drawn that were inserted into the table on the first day and in green the point objects that were inserted on the second day. You can see that the area, in which the point objects exist, has grown. So, the bounding rectangles from the R-tree needed to be enlarged and the tree becomes unstable and had to be rebuilt. After this second day, the area in which the objects exist does not grow anymore and you can see that the index quality parameter stays constant.

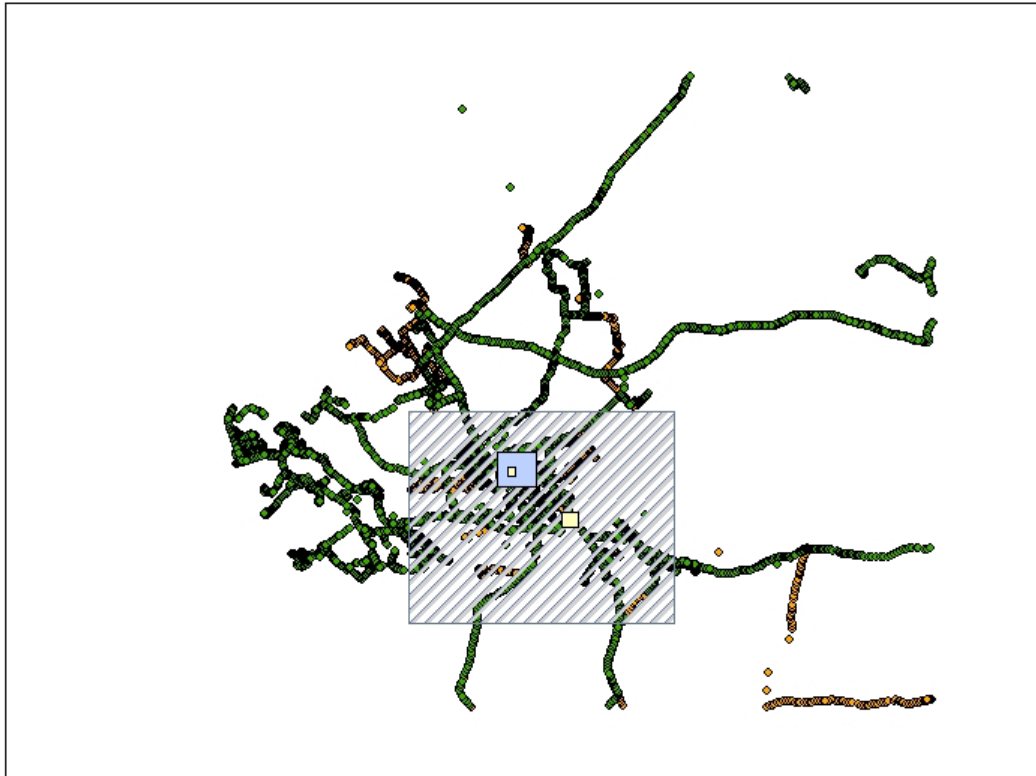


Figure 6.3. In orange the point objects that were added into the table on the first day and in green the objects that were added on the second day. Also the querying areas are drawn.

Another thing that can be said about these results is that the index quality parameter will increase when the number of objects increases. In the beginning the parameter grows faster than later in the test. What the effect is on the query times will be discussed in 6.3.

6.3 Querying the real-time data in 2D

During each test (the 2D test and the 3D test), the continuously growing data set has been queried. To test the generic model for this real-time case, a query has been used that has been repeated every minute. For five regions, the query was “How many objects were in polygon P the last hour”. In two dimensions, this query is relatively easy. But in 3D, this query will be “How many objects were in this box (because time is the third dimension).” This query will be discussed in paragraph 6.4.

For the 2D query, a spatial operator (SDO_RELATE) has been used. As said before, for the spatial operator, the data set must be spatially indexed. The two data sets that have been compared in the query are taxi_temp2 (the continuously growing table with point data) and the study_area (the five polygons which have been queried).

The used polygons differed in size. One covered the whole area, one the whole urban area in and around Rotterdam, one covered the centre of Rotterdam and two small areas, the Rotterdam Train Station and a highway cross Ridderkerk.

The used query looks like this (both taxi_temp2 and study_area make use of 2D spatial index:

```
select
  count(a.id),
  b.name
from
  taxi_temp2 a,
  study_area b
where
  sdo_relate(a.position, b.position, 'mask=inside querytype=window')='TRUE'
  and a.t>(current_date-2082.6-1/24) --2082.6 is the initialization value
  and t<= (current_date-2082.6)
group by b.name;
```

In a “Log-file”, all the answers and query times are collected (for a script see appendix B). A graph of all the query times looks like figure 6.4.

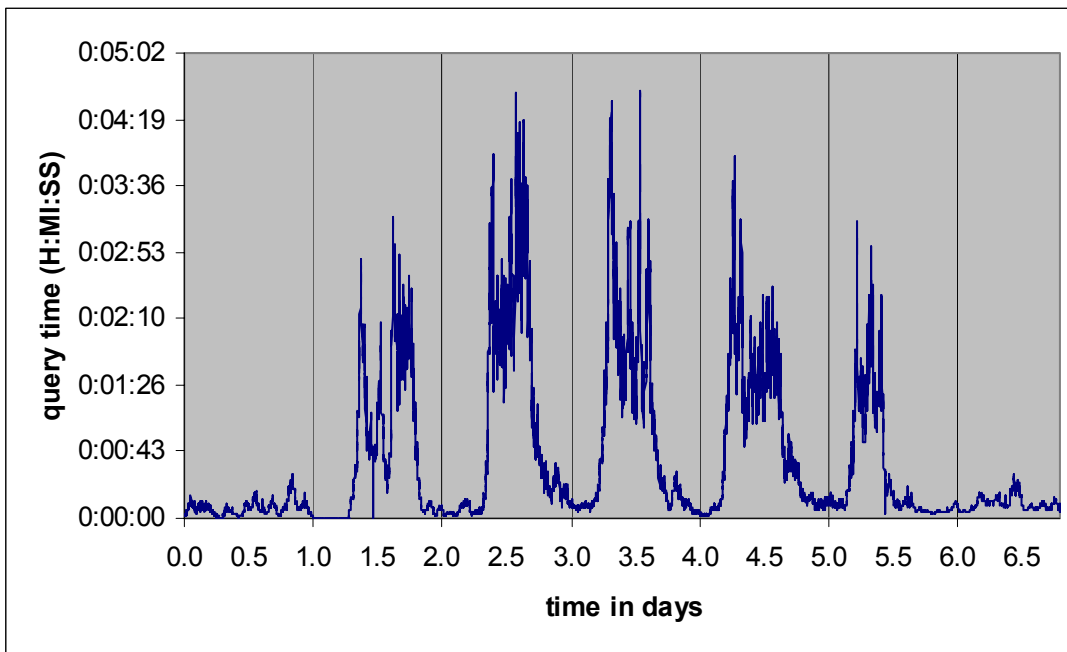


Figure 6.4 The query times of a spatiotemporal query: “How many objects were in these polygons the last hour?” on the growing spatiotemporal data set. 0.0 is at night, 0.5 is at 12:00 PM and so on.

If you compare this result to the graph of the growing data set (figure 6.1), you can see that there is a dependency. When many objects are added, the query time increases (that is a result of the fact that there are more answers). The average query time is 39 seconds. The answers can be

found in figure 6.6. In contradiction to the prediction that the query time was dependant on the index, the query time is mainly dependant on the number of selected objects within the time frame.

In the query described above, the `sdo_relate` operator has been used. This is an operator that checks if two geometries interact, also for complex geometries. Another option is to use the `sdo_filter` operator. This is an operator that checks if the bounding rectangles (or boxes) of two geometries interact. For a rectangle and for a point, the bounding rectangle is equal to the object itself. Because this `sdo_filter` operator is less complex, query times should be faster. This prediction is not true because the query optimizer does not use the spatial index.

Query plan with an index on the 2D geometries, when making use of the `sdo_filter` operator (which lets the query optimizer choose for a full-table scan):

PLAN_TABLE_OUTPUT

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	7862	608
1	SORT AGGREGATE		1	7862	
* 2	FILTER				
3	NESTED LOOPS		181	1389K	608
* 4	TABLE ACCESS FULL	TAXIDATA_TEMP	221	850K	166
* 5	TABLE ACCESS FULL	STUDY_AREA	1	3923	2

The resulting query response times are therefore the same as shown in figure 6.4.

6.4 Querying in 3D

The same test as in 2D has been carried out in 3D with respect to the real-time simulator and rebuilding the index. Some things are more difficult in 3D, for instance the 3D querying. Now, a point is not a x,y pair anymore and a polygon a series of x,y pairs, but the point is now an x,y,t

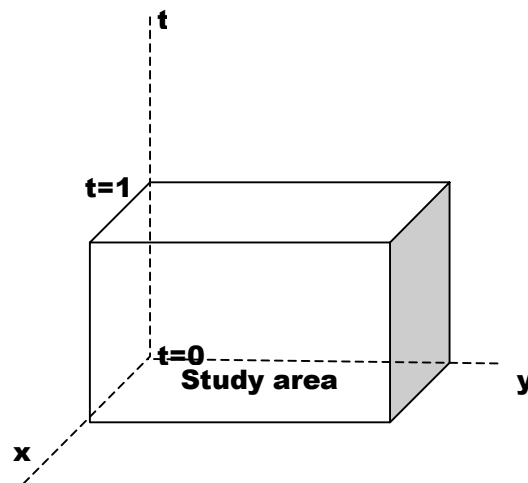


Figure 6.5 The query “how many polygons were in polygon P (is a box) the last hour” becomes a 3D query, where the third dimension is t.

triple and the polygon is a series of x,y,t triples. So your query is no longer “How many objects were in polygon P the last hour?” but it will be “How many objects were in box B”, where $B_x=P_x$, and $B_y=P_y$, and $T \in [t_{\text{now}}, t_{\text{now}} - 1 \text{ hour}]$. This query box looks like figure 6.5.

3D points

First of all, it is necessary to know how the 3D points are implemented in Oracle 9i Spatial. Normally, a 3D point is implemented in Oracle like this:

```
INSERT INTO table (id, position) VALUES (1,
Mdsys.sdo_geometry(3001, NULL,
mdsys.sdo_point_type(1,1,0), --the point coordinates (1,1,0)
NULL, NULL));
```

In this `mdsys.sdo_point_type`, the three values must be of the NUMBER type. In the original table, the time is a DATE type. So, this DATE type needs to be converted into a NUMBER type (for instance the number of seconds). There is chosen for a solution where `time=current_date-min(time)`. This returns a NUMBER type that will become bigger every second.

Secondly, the study box needs to be implemented. The same method has been used that has been described in section 5.3, which makes use of the 3D bounding box of a 3D line.

SDO_FILTER

In Oracle 9i Spatial, only one operator works in 3D. This is the `SDO_FILTER`. This operator uses the spatial index to identify either the set of spatial objects that are likely to interact spatially with a given object (such as an area of interest), or pairs of spatial objects that are likely to interact spatially, based on a box overlap. Objects interact spatially if they are not disjoint.

This operator performs only a primary filter operation based on overlapping boxes. The secondary filtering operation, performed by the `SDO_RELATE` operator, can be used to determine with certainty if objects interact spatially, but this is only relevant for other objects than boxes or rectangles, because the box or rectangles are equal to the Minimum Bounding Boxes or Rectangles in the index.

To see if two groups of polygons are likely to interact spatially (based on the index), the `SDO_FILTER` operator is implemented like:

```
SELECT A.gid
FROM Polygons A, query_polys B
WHERE SDO_FILTER(A.Geometry, B.Geometry, 'querytype=WINDOW')='TRUE';
```

In this example, the table Polygons A must be spatially indexed. It is not said that all objects from Polygons A and query_polys B interact spatially, but their bounding rectangles interact.

The 3D query

Now, there is a 3D line, with a bounding box. When we go back to the query “How many point objects were in polygon P the last hour?” in 3D, the 3D line is not the same for every query. The idea is to repeat the query every minute, so the line will move into the time direction every minute. Because the 3D box changes every time you pose it (because time passes), a new record with geometry needs to be inserted into the table with “study boxes” and the index needs to be rebuild. After that, the query can be posed. All of this looks like:

```

delete study_boxes;

insert into study_boxes (position) values
(mdsys.sdo_geometry(3002,NULL,NULL,
mdsys.sdo_elem_info_array(1,2,1),
mdsys.sdo_ordinate_array(35680,403400,
((select current_date-min(a.t) from taxidata a)-(1/24)),
151440, 489500,
(select current_date-min(a.t) from taxidata a)))); --insert the box into table

alter index Istudy_boxes rebuild parameters ('sdo_fanout=32 sdo_indx_dims=3
layer_gtype=line'); --rebuild the index of the box

set timing on;
insert into stquery_results4
select
count(a.id), current_timestamp
from taxidata_temp4 a, study_boxes b
where sdo_filter (a.position, b.position, 'querytype=window')='TRUE'; --the actual query
set timing off;

```

The “current_date-min(a.t)” is used, because this is the same as the implemented t-value in the x,y,t triple in the taxidata_temp4 table (the growing table with 3D point data). This query has been carried out for two of the same “polygons” as in the 2D case. So, the same query (with different polygons) has been carried out two times every minute, independent from each other. In figure 6.6 you can see the results of both queries.

The query-plan for the 3D sdo_filter operation looks like:

PLAN_TABLE_OUTPUT

Id	Operation	Name	Rows	Bytes	Cost(%CPU)
0	SELECT STATEMENT		1	7866	13929 (0)
1	SORT AGGREGATE		1	7866	
2	NESTED LOOPS		172K	1290M	13929 (0)
3	TABLE ACCESS FULL	STUDY_BOXES	82	314K	2 (0)
4	TABLE ACCESS BY INDEX ROWID	TAXIDATA_TEMP4	2098	8078K	13929 (1)
5	DOMAIN INDEX	ITAXIDATA_TEMP4			

So, you can see (in figure 6.6) that the queries give realistic and proposed correct results. But something more important is the time that is necessary for answering the question. In figure 6.7, you can see the query-times elapsed (from the log-file) for both queries as a function of the time. It can be concluded that querying the complete data set with the SDO_FILTER operator is very quick (hardly never more than one second response time). It can also be concluded that the query response times are not too much dependent on the quality of the index, because the query response times do not increase when the number of objects in the queried data set increase. From the query-plan it can be concluded that in this query, the spatial index is used and works very well.

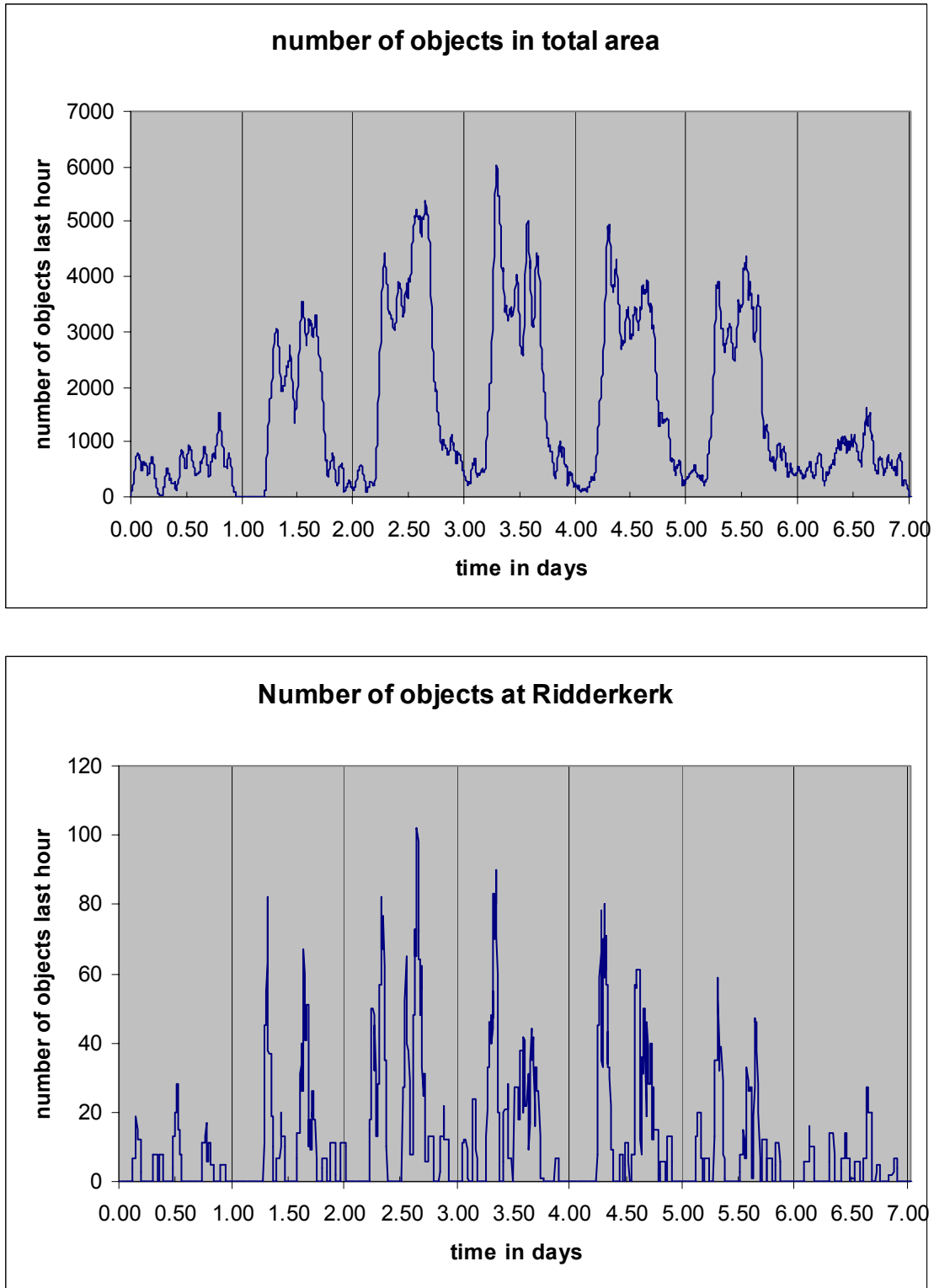


Figure 6.6 Results of 3D query, where in the upper graph, you see the number of objects in the total area for the last hour and in the lower graph, you see the results for a small part of the area.

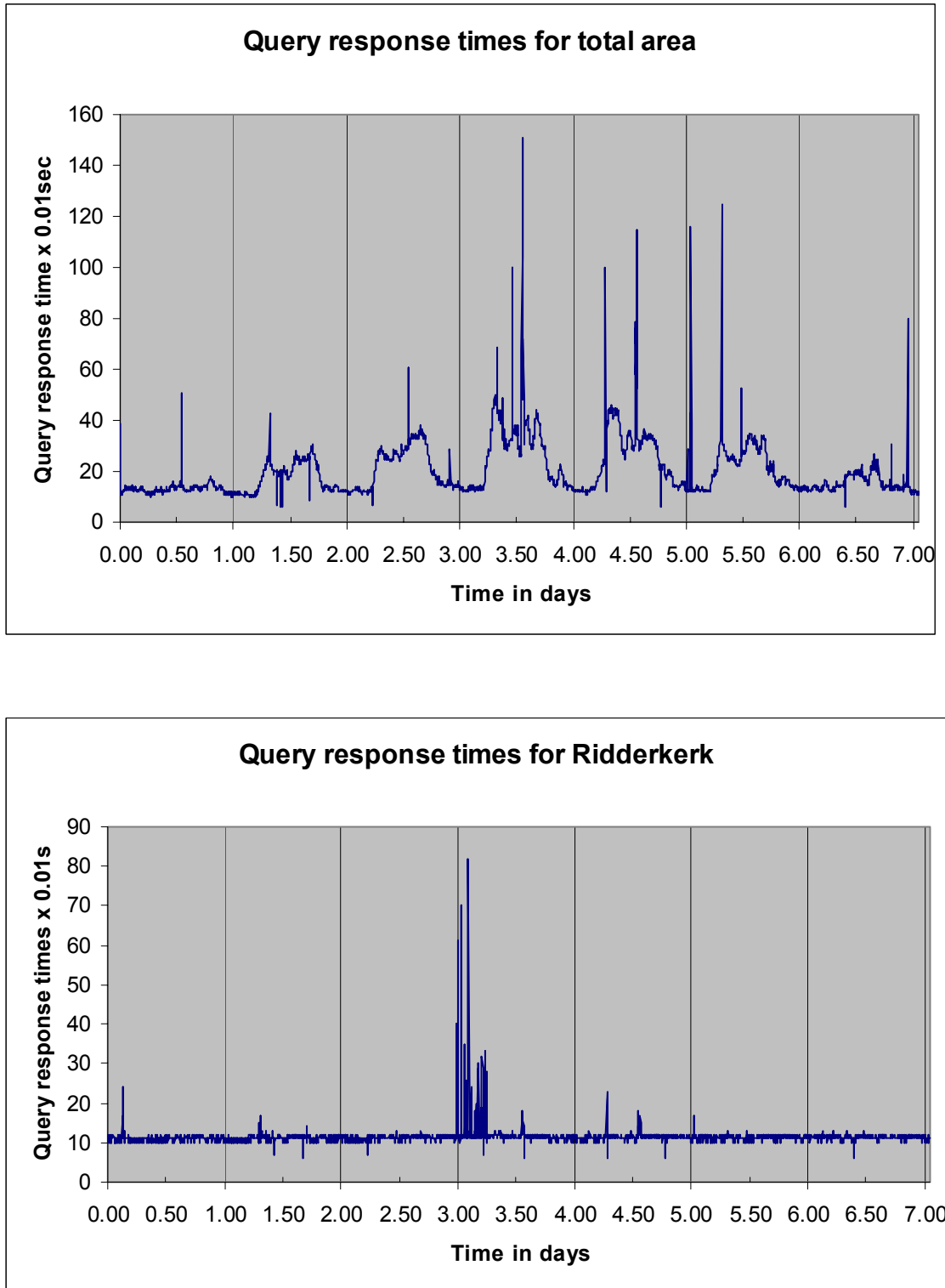


Figure 6.7 Query response times for the 3D query done with the SDO_Filter operator on the continuously growing dataset. In the upper graph, you can see the response times for the query done on the total area and in the lower graph, you can see the query posed on a smaller area.

6.5 Conclusions

A real-time simulator has been built to test if the generic model is flexible and fast. Because at the moment of writing this thesis, the ESRI ArcIMS Tracking Server was not available, a PL/SQL application has been build, where a table is growing every 10 seconds with more and more data. So there is a continuously growing table with point data. To see what happens to the 2D and 3D index in the case of a real-time data set, some tests have been carried out. Also 2D and 3D queries have been done on the continuously growing data set. In this section, some conclusions that can be derived from these tests are discussed.

The first conclusion is that the generic model also suffices while using a real-time data set. A continuously growing data set has been successfully implemented into the generic model with views. The model is fast and flexible.

It is possible to implement an index that can be rebuilt automatically. Oracle 9i Spatial uses a quality parameter. The quality parameter can increase relatively fast if a new object is inserted into the table that is outside the bounding box of one of the nodes in the R-tree or if the number of objects in a leaf node is too high, so that the index needs to go one level deeper.

For the 2D R-tree, the quality of the index does not increase when the number of objects will become bigger. Oracle 9i Spatial uses a number that gives an idea of the quality of the R-tree. This quality parameter increases when new objects are inserted into the 2D R-tree. The more objects are entered into the table and the more these objects cover a larger part of the area, the 2D R-tree will become more stable and does not need to be rebuilt very often.

The quality of the 3D R-tree will increase when time passes in the real-time situation. This is because the objects that are inserted in the table are not close to the existing objects in the 3D space (two spatial and one temporal dimension). That is because the data set is growing in the temporal dimension, while the spatial dimensions were covered after a while. Depending on the units used for the spatial dimensions (days or minutes), the 3D R-tree needs to be rebuilt often.

The query time for the 2D query “Which objects where in polygon P during the last hour?” is not dependant on the number of objects in the continuously growing table. It depends on the number of objects that has to be selected. A reason for this could be that the continuously growing table is sorted on time, because when new records are inserted, they will be inserted on the end of the table. So, all objects that were inserted in the last hour are sorted close to each other on the end of the table.

Doing the same query in 3D is much more difficult, because for the query “Which 3D points were in box B?”, boxes need to be defined. In this case, the height of the query box is “one hour”. The width and breadth are the same as the polygons in the 2D query. As a solution, the SDO_FILTER operator has been used. This is the only operator that works in 3D and selects objects that are likely to interact based on the index.

It can be concluded that querying the complete data set or a part of it with the SDO_FILTER operator is very fast (hardly never more than one second response time). It can also be concluded that the query response times are not dependent on the quality of the index, because the query response times do not increase when the number of objects in the queried data set increase.

7. Conclusions and recommendations

Spatiotemporal data is becoming more and more important for many different applications. An example of spatiotemporal data is the moving point object data. Geo Database Management Systems have the advantage that large amounts of geo-data can be stored and accessed in an efficient way. Especially when aspects like multiple users, data consistency, redundancy, and integrity are important, DBMSs have many advantages. In Oracle 9i Spatial, there are spatial functions and operators available to query the geographical data.

This leads to the following question, which is the main question of this research:

What is the potential and performance of a geo-DBMS to structure, index, query and visualize spatiotemporal point clouds of moving objects?

Some models are described that are able to structure moving point data in a geo-DBMS environment. All these models have some disadvantages. A model is introduced in this thesis that is generic, which means that it should be suitable for all applications that deal with moving point object data.

The main principle of this generic model is choosing a base table, from which, by using (materialized) views, three other data representations (based on different geometric data types) easily could be derived. In this way a set of four data representations is available for querying. These four data types are 2D points (x,y) , 3D points (x,y,t) , 2D lines $(x_i,y_i, x_{i+1},y_{i+1})$ and 3D lines $(x_i,y_i,t_i, x_{i+1},y_{i+1},t_{i+1})$ where in the 2D representations time is regarded as an attribute.

This model is implemented for two cases; the first case deals with static data (where the data has been collected in advance and analyzed in the DBMS afterwards). The second case deals with real-time data, where the collection and analysis are done at (nearly) the same time.

Conclusions from this research are described in this chapter. These conclusions will answer the main question and the sub-questions defined in the introduction (chapter 1). In section 7.1, these conclusions are described, followed by some recommendations for future research in 7.2.

7.1 Conclusions

The conclusions described in this section can also be found in the chapters 2 to 6 in this thesis. These conclusions will be discussed using the sub-questions described in chapter 1.

Why could a geo-DBMS be an efficient and flexible way to store moving point data?

This thesis deals with moving object data, which is one special type of spatiotemporal data. The question is now, why should these data be stored and queried in a spatial DBMS? Because a complete suite of spatial queries and data types are already implemented in a geo-DBMS (Oracle 9i Spatial) and because a DBMS takes care of the data, it is at least worth to investigate whether it is useful to choose for a database approach or not. Another reason is that it is possible to define ad-hoc queries in an easy way (flexibility) and because DBMSs have a good performance (efficiency).

Which methods are available to structure and index moving point objects in a geo-DBMS context?

Langran described a framework for a temporal GIS by giving technical requirements. One of these technical requirements is a conceptual model, from which the modeling of dynamic point clouds is an example. Four conceptual models are described. These models have some disadvantages. For instance, the model made by Vazirgiannis and Wolfson is especially developed for road networks and the MOST-datamodel (developed by Wolfson) is relatively complicated and suitable for objects that move freely in space like aircrafts. The other two approaches, described by Marchand et al and the one described by Meng and Ding have the disadvantage that they have a lot of redundant storage. To overcome these disadvantages, a new approach is introduced in this thesis. This model could be used for every purpose (this makes it generic) and it does not contain any redundant storage.

Efficient querying is only possible if an efficient storage structure and indexing method has been chosen that organizes the data in an optimal way for the query. If you would like to do a query, it is possible to make a copy of (a part of) the original data set, find the most efficient indexing method and do the query. This method is in many cases fast (except that making a copy of the data can be slow), but making copies of tables can harm the consistency of your data. So, an intelligent, but perhaps less fast querying method is necessary on the original table.

In Oracle 9i Spatial, only the 2D and the 3D R-tree indexing methods are implemented and suitable for spatiotemporal data. The choice of which indexing method is going to be chosen as the most efficient and the most flexible one is limited. The choice for 2D or 3D spatial indexing depends on the queries that are going to be examined.

Does a generic model exist to implement moving point objects in a geo-DBMS like Oracle 9i Spatial?

A generic model is introduced for modeling moving object data in a geo DBMS. Adapting the generic model, by choosing the appropriate storage and index structures and (materialized) views, makes it efficient for a given application, that is, a set of typical queries for a given (static or dynamic data set). With a normal 2D R-tree index, the 2D data (and a 3D R-tree for 3D data) is accessed and with a small set of views and materialized views, the data is queried.

It makes difference if your application deals with data that is already collected (post-processing) or with real-time data. In the first case, for instance indexing is much easier because all the data is already in the DBMS. In the real-time case, the data set is continuously growing, which needs special treatment of the index and the updating of the views and materialized views. How many times these views have to be updated and how often the index needs to be rebuild, depends on your application, the amount of data and the users' intentions.

A choice has to be made, which of the four primitives is to be used as the base table. The 2D points, the 2D lines, the 3D points or the 3D lines. If one is chosen (for instance 2D points), the other primitives can be derived in (materialized) views. The choice for a base table depends on the spatiotemporal queries, which you would like to use. In Oracle 9i Spatial, the number of 3D queries is limited (only the SDO_FILTER is available in 3D) and the number of 3D primitives is limited (for instance, a box is no primitive in Oracle 9i Spatial). There only exist boxes (3D) in the 3D R-tree index (Minimum Bounding Box). These can be used for the querying with the SDO_FILTER operator.

Is this generic model sufficient for a static data set where the moving point object data is collected in advance?

The generic model is successfully implemented in Oracle 9i Spatial and loaded with test data from a test case with moving object data that were collected in advance. Every 0.1 seconds, photos of a piece of highway were taken from a helicopter. The vehicles in these photos were detected and tracked in multiple photos. This resulted in a data set (2300 records) with positions and times for every vehicle that was driving on that part of the highway in that period.

With an R-tree index, the data is accessed and with a small set of views, the data is queried. Simple queries like “maximum speed” or “time of acceleration” can easily be done. Two more complex questions are discussed, calculating the vehicles that are within two seconds distance from their predecessor and calculating variables that are used in traffic flow modeling.

Querying is possible in three different ways. The first method queries the spatial dimension before querying the time dimension. The second method is the opposite, first querying the temporal dimension before querying the spatial dimensions. And the third method queries the spatial and temporal dimensions at once with a 3D query. These methods are compared in this static case study.

The query “Which objects keep less than 2 seconds distance from their predecessors?” is posed in three different ways, “Which objects are in two seconds distance from vehicle A?” and “Are there any objects on position x_a, y_a between now and two seconds?”. The results were a little bit different because both queries are not exactly the same. Querying in three dimensions resulted in the same answers as for the first method.

Something more important is the query times. For the first and the third querying method, the query optimizer did not use the spatial index, but chose for a full table scan. For the second method, only one spatial index is used and is because of that much faster. It is expected that when the number of objects increase, the query optimizer will make use of the spatial index, because a full table scan will be more expensive in that case in terms of performance.

With the ArcGIS 8.3 Tracking Analyst, the moving objects can be animated in an effective way. By careful inspection of the animation, a lot of information can be derived from the information, like the vehicles that accelerate or the vehicles that are driving too close to their predecessors.

Is this generic model sufficient for a dynamic data set, where the moving point objects are collected real-time?

For the second case study, a data set has been used in which about 60 taxis have been driven near Rotterdam for two years. One week of this data set has been used (250000 records) to test the generic model for real-time storage and analysis. Because no real-time data was available, a real-time simulator has been built on this taxi data set to test if the generic model is flexible and fast in real-time situations.

It can be concluded that the generic model also suffices when using a real-time data set. A continuously growing data set has been successfully implemented into the generic model with views. The model is fast and flexible. It is possible to implement an index that can be rebuilt automatically.

For the 2D R-tree, the quality of the index does not increase when the number of objects will become bigger. Oracle 9i Spatial uses a number that gives an idea of the quality of the R-tree. This quality parameter increases when new objects are inserted into the 2D R-tree. The more

objects are entered into the table and the more these objects cover a larger part of the area, the 2D R-tree will become more stable and does not need to be rebuilt very often.

The quality of the 3D R-tree will increase when time passes in the real-time situation. This is because the objects that are inserted in the table are not close to the existing objects in the 3D space (two spatial and one temporal dimension). That is because the data set is growing in the temporal dimension, while the spatial dimensions were covered after a while. Depending on the units used for the spatial dimensions (days or minutes), the 3D R-tree needs to be rebuilt often.

The query time for a 2D query “Which objects were in polygon P during the last hour?” is not dependant on the number of objects in the continuously growing table. It depends on the number of objects that are selected. A reason for this is that the objects that are inserted on the end of the table. So the continuously growing table is sorted on time. Because of that, he used 2D query selects objects that are close to each other in the sorted table.

For querying in 3D, the SDO_FILTER operator has been used. This is the only operator that works in 3D and selects objects that are likely to interact based on the index. It can be concluded that querying the complete data set or a part of it with the SDO_FILTER operator is very fast (hardly never more than one second response time). The 3D R-tree gives for this query fast access to the data.

7.2 Recommendations

Some recommendations will follow from the conclusions described in 7.1.

Tests have been carried out to demonstrate if the generic model described chapter 3 is fast and flexible. This has been done for 2 and 3 dimensions. It is recommended to prove this also for four dimensions (x,y,z and time). Then, this model could also be applied to for instance airplanes.

Another recommendation is to test the ESRI ArcIMS Tracking Server. It is assumed that this makes spatiotemporal modeling in real-time situations a lot easier and the real-time data can be visualized directly with the Tracking Analyst. It provides a solution for collecting and sending real-time tracking data from many data sources to web or desktop clients. This server could replace the real-time simulator used in the real-time case study.

In this research, the standard Oracle 9i Spatial R-tree has been used. But according to several researchers more efficient algorithms are available. Testing such indexing methods could improve accessing the moving point data in a geo DBMS.

A spatial index uses the primary key and the spatial (or spatiotemporal) information of the object. When you are dealing with moving point objects, the minimum bounding box (or rectangle) of a point object is the point object itself. So, all the information that is in the base table (id, position and time) is also available in the index. In theory it should be possible to not even store the base table and just have an index, which contains all the information. It is recommended to test if this method works with the current DBMS-technology.

A 3D index in spatiotemporal modeling is built on the spatial and the temporal dimension. Scaling the time dimension can influence the spatial index. For instance, using minutes as a unit instead of days enlarges the time dimension. It is recommended to investigate on which scale the index works in the most efficient way.

Querying spatiotemporal data in Oracle 9i Spatial is possible with functions and with operators. The main difference is that a spatial function does not need a spatial index while a spatial operator does. The spatial operator does not always use the spatial index. Sometimes, the query optimizer determines that a full table scan is more efficient than using the index. This is especially the case when small data sets are used. It needs to be investigated when the spatial operator does and when it does not use the spatial index.

At might be interesting to test the generic model also for other spatiotemporal applications like for instance a cadastral database where spatial changes occur with polyhedrons, polygons or polylines. It is recommended to implement this generic model in some applications e.g. for traffic monitoring.

Literature

- [1] Allen, J.F., 1983, *Maintaining Knowledge about Temporal Intervals*, in Communications of ACM Volume 26, Issue 11, p 832-843.
- [2] Arens, C.A. 2003. *Maintaining Reality – Modelling 3D spatial objects in a Geo-DBMS using a 3D primitive*. Delft. Masterthesis, section GIS technology, TU Delft.
- [3] Bertin, J., *Semiology of graphics*. Madison, Wisc.: University of Wisconsin Press, 1983.
- [4] ESRI, White paper "What is the ArcIMS Tracking Server?" May 2003.
- [5] Heres, L. 2000. *Hodochronologies: History and time in the National Road Database*. In: Time in GIS: Issues in spatio-temporal modelling. Delft, Nederlandse Commissie voor Geodesie. P46-56.
- [6] Hoogendoorn, S.P., Botma, H., Minderhoud, M.M., *Traffic flow theory and simulation*. TU Delft, Transportation and Traffic Engineering Section, Faculty of Civil Engineering, 2004.
- [7] Kraak, M.-J. and Ormeling, F. J., 1996, *Cartography, visualization of spatial data*, (London: Addison Wesley Longman).
- [8] Langran, G. 1992, *Time in Geographic Information Systems*. London, Taylor & Francis.
- [9] MacEachren, A.M., 1994, *Visualization in modern cartography: Setting the Agenda*. In Visualization in Modern Cartography (A. M. MacEachren and D. R. F. Taylor, Oxford, UK: Pergamon), pp. 1-12.
- [10] Marchand P., Brisebois A., Bédard Y., Edwards G., 2003, *Implementation and evaluation of a hypercube-based method for spatio-temporal exploration and analysis*. Journal of the International Society of Photogrammetry and Remote Sensing (ISPRS) theme issue "Advanced techniques for analysis of geo-spatial data" dans la catégorie "multi-scale hierarchies of spatial operators".
- [11] Meng X., Ding Z., *DSTTMOD: A Discrete Spatio-Temporal Trajectory Based Moving Object Database System*. DEXA2003, LNCS 2736(Springer Verlag), September,2003, Prague.444-453
- [12] M. F. Mokbel, T. M. Ghanem, and W. G. Aref, "Spatio-temporal Access Methods", IEEE Data Engineering Bulletin, 26(2), 40-49, Jun., 2003.
- [13] Nascimento, M.A., Silva, J.R.O. and Theodoridis, Y., Access Structures for Moving Points. A Timecenter technical report. Aalborg 1998.
- [14] NRC Handelsblad, 9-9-2003, article *Mobiele filemeldingen* (in Dutch).
- [15] Oosterom, P. van, 2000, *Time in Cadastral Maps*. In: Time in GIS: Issues in spatio-temporal modelling. Delft, Nederlandse Commissie voor Geodesie. P36-45
- [16] Oracle 9i Database Concepts release 2
- [17] Peuquet, D.J., 2002, *Representations of Space and Time*. New York, Guilford Press.
- [18] Pfoser, D and Jensen, C.S., 2003, *Indexing of Network-Constrained Moving Objects*. A TIMECENTER Technical report.
- [19] Raper, J. 2000. *Multidimensional Geographic Information Science*. London, Taylor & Francis.
- [20] Saltenis, S., Jensen, C.S., Leutenegger, S.T. Lopez, M.A. 1999. *Indexing the positions of continuously moving objects*. Aalborg. A TIMECENTER Technical report.
- [21] Snodgrass, R.T. *SQL2 Language Specification*. University of Arizona, Tucson 1994.
- [22] Theodoris Y, Vazirgiannis, M. and Selles, T. *Spatio-Temporal Indexing for Large Multimedia Applications*. In Proceedings of the IEEE Conference on Multimedia Computing and Systems, IMCS, June 1996.
- [23] Vazirgiannis, M. Wolfson, O., *A Spatiotemporal Model and Language for Moving Objects on Road Networks*. 2001. In Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases, pages 20-35, 2001. 8

- [24] Wolfson O., Xu B., Chamberlain S., Jiang L., *Moving Object Databases: Issues and Solutions*. Proceedings of the 10th International Conference on Science and Statistical Database Management, Capri, Italy, 1998: 111-122.
- [25] Worboys, M.F., 1995, *GIS – A computing perspective*. London, Taylor & Francis.

Internet

- [26] ESRI, <http://www.esri.com/software/arcgis/arcgisxtensions/trackinganalyst/index.html>, 2004
- [27] ESRI, http://www.esri.com/software/arcims/tracking_server.html (February 2004)
- [28] Hansen, K.T. Introduction to Databases, <http://www.aitel.hist.no/fag/dbs-e/lek01/lesson01.pdf>
- [29] Department of Computer Science, University College Cork.
<http://www.cs.ucc.ie/~abf/CS507-8/L2.pdf>

Appendix A: Glossary

2D – 2-Dimensional; 2D objects are flat, e.g. a polygon; objects in 2D space are spanned with 2 coordinates (usually length and width).

3D – 3-Dimensional; 3D objects have a volume, e.g. a polyhedron or a 3D point; objects in 3D space are spanned with 3 coordinates (usually length, width and height or length, width and time).

Consistency – When all copies of data are the same (changes in the data are updated in all copies), the data is consistent.

Database – Term, which can be used for DBMS and for data set.

Data set – A collection of related data;

DBMS – Database management system; collection of programs to maintain the data in the data sets.

DBMS approach – Approach to organize data by making use of a DBMS instead of organizing data in files.

Density – Number of vehicles per unit of length.

File-based approach – Approach to organize data by making use of programs; The data is organized in files.

Full-table scan – Procedure to search for records by scanning the complete table and not by making use of an index.

Geo-DBMS – DBMS that supports the management of geographical data.

GIS – Geographical Information System; decision supporting system for storing, maintaining, querying, analyzing and visualizing geographical data.

Index – An ordered table where each record contains an index-field (value) and a pointer field (address on hard disk).

Integrity – When the linkages (relationships) between data sets are well maintained, the integrity of the data set is good.

Intensity – Number of vehicles per number of time.

Interoperability – The ability for a system or components of a system to provide information portability and inter application, cooperative process control;

(Materialized) views – View on the data that can be treated as a table in a DBMS, but only the syntax is stored and not the data itself; A view is materialized when the data is also stored in a table that can be updated on demand.

MBB – Minimum Bounding Box; box around a 3D object that is parallel with the coordinate system axis and encloses the object.

MBR – Minimum Bounding Rectangle; rectangle around a 2D object that is parallel with the coordinate system axis and encloses the object.

MDSYS.SDO_GEOMETRY – A geographical data type, used in Oracle 9i Spatial.

Mean Distance Headway – The mean distance between the rear bumper of the preceding vehicle and the rear bumper of the considered vehicle.

Mean Time Headway – The mean period between the passing moment of the preceding vehicle and the vehicle considered.

Moving point objects – Objects that can be represented as points and move in space and time.

Oracle 9i Spatial – One of the most often used DBMSs with a spatial extent.

PL/SQL – Procedural Language/Structured Query Language; an Oracle extension to allow procedures in SQL.

Point cloud – Cloud of points; in this thesis a set of moving point objects.

Post-processing data – data that has been collected in advance; the processing or analysis of the data comes afterwards.

Query – A question or request to select a number of objects.

Query-plan - Plan used by the Query-optimizer in which order parts of the query are going to be carried out.

Query-optimizer – An internal process that chooses the most optimal query plan for answering a query.

Query-response time – The time between posing a query and getting the response.

Real-time data – Data that is going to be processed or analyzed at the same moment, or nearly the same moment as the data is collected.

Redundancy – When data (for instance a coordinate) is stored more than one time, the data set contains redundancy.

R-tree – 2D or 3D spatial index that tiles up objects.

SDO_ELEM_INFO – This specifies the elements of the geometry with references to the coordinates, information about the element itself and an interpretation code in the MDSYS.SDO_GEOMETRY data type.

SDO_FILTER – Spatial operator that can be used when two or more dimensions are involved; this operator compares bounding boxes or rectangles to find candidates that interact spatially.

SDO_GTYPE – This indicates the type of geometry in the MDSYS.SDO_GEOMETRY data type.

SDO_ORDINATES – This is a variable array of numbers and contains the coordinates in the MDSYS.SDO_GEOMETRY data type.

SDO_POINT – This element is used when only points are stored as single object or when a point is stored in addition to the other geometry in the MDSYS.SDO_GEOMETRY data type.

SDO_RELATE – Spatial operator that can be used when two dimensions are involved; this operator compares coordinates to find the objects that interact spatially.

SDO_SRID – This is a reference to the spatial reference system used by the coordinates in the MDSYS.SDO_GEOMETRY data type.

Space-time cube – Three-dimensional representation that contains two spatial dimensions and one temporal dimension;

Spatial function – Function that works on spatial data; A spatial function does not need a spatial index to operate.

Spatial operator – Function that works on spatial data; A spatial operator needs a spatial index to operate.

SQL – Structured Query Language; language to query data in a DBMS.

Trajectory – The path of a moving object described in space in time.

Appendix B: Scripts

This appendix contains the scripts used for the second case study (Taxi cabs in Rotterdam).

2D Real-time simulator. This script is run every 10 seconds. The continuously growing table, is `taxidata_temp`.

```
insert into taxidata_temp
(id, position, t_time)
select id, position, t_time from taxidata where
t_time<(current_date+10/(24*60*60)-2074.696) and
t_time>=(current_date-2074.696);
exit;
```

Repeated every 10 seconds by making use of a shell script:

```
#!/bin/sh
while (sleep 10);
do
sqlplus baars/marco@gisbase @minute_taxidata_update.sql &
done
```

3D Real-time simulator. This script is run every 10 seconds. A table (`taxidata_temp4`) is growing every 10 seconds with records. The `current_date-min(s.t_time)` clause is necessary to make a “NUMBER” data type from a “DATE” data type that is ascending. This is necessary because in the `sdo_geometry`, only “NUMBER” datatypes are allowed. So finally the time dimension in the `sdo_geometry` is a number, which is in fact the number of days between now, and the first data point in days. So, this number should not be interpreted as a “real” timestamp, but for the test this makes no sense.

```
lock table taxidata_temp4 in exclusive mode;
insert into taxidata_temp4 (id, position, t_time)
select s.id,
mdsys.sdo_geometry(3001, NULL,
mdsys.SDO_POINT_TYPE(s.position.sdo_point.x,
s.position.sdo_point.y,
(select current_date-min(s.t_time) from taxidata s)),
NULL, NULL),
s.t_time
from taxidata s
where
t_time<(current_date+30/(24*60*60)-2151.64) and --value 2151.64 is an initiation value
t_time>=(current_date-2151.64);
```

Also makes use of a shell script for repeating every 30 seconds:

```
#!/bin/sh
while (sleep 30);
do
sqlplus baars/marco@gisbase @minute_taxidata_update4.sql &
done
```

Script, which checks the quality parameter of the index and **rebuilds the 2D R-tree** if necessary (see also figure 6.2).

```

set echo on;
set serveroutput on;

DECLARE

min_qual number;
qual number;
degr number;
nrob number;
statement varchar2(400);
qual2 number;

BEGIN

select quality into min_qual from upd_qual_val4 where
time=(select max(time) from upd_qual_val4);

select
sdo_tune.rtree_quality('baars','Itaxidata_temp4'),
sdo_tune.quality_degradation('baars','Itaxidata_temp4'),
count(b.taxi_id) into qual, degr, nrob
from dual, taxidata_temp4 b;

IF
(qual/min_qual-1 > 0.5)
THEN

insert into idx_qual_taxidata_temp4
(time, degradation, quality, nr_obj, upd_bool)
values (current_timestamp, degr, qual, nrob, 1);
lock table taxidata_temp4 in exclusive mode;
statement := 'alter index Itaxidata_temp4 rebuild';
execute immediate statement;
select
sdo_tune.rtree_quality('baars','Itaxidata_temp4') into qual2
from dual;
insert into upd_qual_val4 (time, quality) values (current_timestamp, qual2);

ELSE
insert into idx_qual_taxidata_temp4
(time, degradation, quality, nr_obj, upd_bool)
values (current_timestamp, degr, qual, nrob, 0);
END IF;
END;
/
exit;

```

With a shell script to repeat this every minute:

```

#!/bin/sh
while (sleep 60);
do
sqlplus baars/marco@gisbase @idx_update.sql &
done

```

A script that checks the quality of the **3D R-tree** and rebuilds if necessary (see also figure 6.2).

```

set echo on;
set serveroutput on;

DECLARE

min_qual number;
qual number;

```

```

degr number;
nrob number;
statement varchar2(800);
qual2 number;

BEGIN

select quality into min_qual from upd_qual_val4 where
time=(select max(time) from upd_qual_val4);

select
sdo_tune.rtree_quality('baars','Itaxidata_temp4'),
sdo_tune.quality_degradation('baars','Itaxidata_temp4'),
count(b.id) into qual, degr, nrob
from dual, taxidata_temp4 b;

IF
(qual/min_qual-1 > 0.5)
THEN

insert into idx_qual_taxidata_temp4
(time, degradation, quality, nr_obj, upd_bool)
values (current_timestamp, degr, qual, nrob, 1);
lock table taxidata_temp4 in exclusive mode;
statement := ' alter index Itaxidata_temp4 rebuild parameters (''sdo_fanout=32
sdo_indx_dims=3 layer_gtype=point'')';
execute immediate statement;
select
sdo_tune.rtree_quality('baars','Itaxidata_temp4') into qual2
from dual;
insert into upd_qual_val4 (time, quality) values (current_timestamp, qual2);

ELSE
insert into idx_qual_taxidata_temp4
(time, degradation, quality, nr_obj, upd_bool)
values (current_timestamp, degr, qual, nrob, 0);
END IF;
END;
/
exit;

```

And a shell script which repeats this every 3 minutes:

```

#!/bin/sh
while (sleep 180);
do
sqlplus baars/marco@gisbase @idx_upd_qual4.sql &
done

```

A 2D spatiotemporal query, which has been done every minute. Resulting query times are in figure 6.4.

```

select
count(a.id),
b.name,
to_char(max(a.t_time), 'DD-MON-YY HH24:MI:SS') "max time"
from
taxi_temp2 a,
study_area b
where
sdo_relate(a.position, b.position, 'mask=inside querytype=window')='TRUE'
and a.t_time>(current_date-2082.6-1/24)
and t_time<= (current_date-2082.6)
group by b.name
order by count(a.id);

```


And also here the shell script that repeats this every minute:

```
#!/bin/sh
while (sleep 60);
do
sqlplus baars/marco@gisbase @stquery 2>&1 | tee -a stquery_output.txt &
done
```

A **3D spatiotemporal query**, using the SDO_FILTER operator. Results can be found in figures 6.6 and 6.7. This has been done for two different geometries, one that covers the complete area (with the coordinates mentioned in this script below) and one that covers a small area. That script is the same except the coordinates.

```
set serveroutput on;
set echo on;

delete study_boxes;

insert into study_boxes (position) values
(mdsys.sdo_geometry(3002,NULL,NULL,
mdsys.sdo_elem_info_array(1,2,1),
mdsys.sdo_ordinate_array(35680,403400,
((select current_date-min(a.t_time) from taxidata a)-(1/24)),
151440, 489500,
(select current_date-min(a.t_time) from taxidata a))));

alter index Istudy_boxes rebuild parameters ('sdo_fanout=32 sdo_indx_dims=3
layer_gtype=line');

set timing on;
insert into stquery_results4
select
count(a.id), current_timestamp
from taxidata_temp4 a, study_boxes b
where sdo_filter (a.position, b.position, 'querytype=window')='TRUE';
set timing off;
exit;
```

The shell-script for repetition:

```
#!/bin/sh
while (sleep 180);
do
sqlplus baars/marco@gisbase @stquery4 2>&1 | tee -a stquery_output4.txt &
done
```
