Netherlands
organization for
applied scientific
research

FEL

TNO Physics and Electronics
Laboratory

report no.
FEL-89-B207

copy no.
8

A geographical extension to the relational datamodel

Netherlands
organization for
applied scientific
research

TNO-report

TNO Physics and Electronics
Laboratory

P.O. Box 96[illegible]
2509 JG The [illegible]
Oude Waalsdorperweg 63
The Hague, The Netherlands

Phone +31 70 26 42 21

report no. FEL-89-B207

copy no. 8

title

A geographical extension to the relational datamodel

author(s):

Ir. M.C. van Hekken
Ir. P.J.M. van Oosterom
Ir. M.R. Woestenburg

classification		
title	:	unclassified
abstract	:	unclassified
report	:	unclassified

no. of copies	:	27
no. of pages	:	30
appendices	:	0

date : August 14, 1989

All information which is classified according to Dutch regulations shall be treated by the recipient in the same way as classified information of corresponding value in his own country. No part of this information will be disclosed to any party

report no.	:	FEL-89-B207
title	:	A geographical extension to the relational datamodel
author(s)	:	Ir. M.C. van Hekken, Ir. P.J.M. van Oosterom, Ir. M.R. Woestenburg
institute	:	TNO Physics and Electronics Laboratory
date	:	August 14, 1989
NDRO no.	:	-
no. in pow '89	:	704.1

ABSTRACT

Modern Command and Control information systems contain both alphanumerical and geographical data (e.g. maps). Efficient storage and retrieval of geographical information from a relational database is not possible with current technology.
This report deals with this problem and presents a possible solution. The solution is based upon the extension of the relational model with some domain-types and database operators and uses special index-structures.

The contents of this report will be presented at the *'GEO '89'* symposium, to be held at Shape Technical Center (STC), october 1989.

rapport no.	:	FEL-89-B207
titel	:	A geographical extension to the relational datamodel
auteur(s)	:	Ir. M.C. van Hekken, Ir. P.J.M. van Oosterom, Ir. M.R. Woestenburg
instituut	:	Fysisch en Elektronisch Laboratorium TNO
datum	:	14 augustus 1989
hdo-opdr.no.	:	-
no. in iwp '89	:	704.1

SAMENVATTING

Moderne Command and Control systemen bevatten naast alphanumerieke gegevens ook vaak geografische gegevens (bijv. landkaarten). Het efficiënt opslaan in, en het opvragen van geografische gegevens uit een relationele database is niet goed mogelijk met de huidige technieken.
Dit rapport gaat in op deze problematiek en geeft een mogelijke oplossing aan. De oplossing is gebaseerd op uitbreiding van het aantal domein types en het aantal database operatoren en maakt gebruik van speciale index-strukturen.

De inhoud van dit rapport zal worden gepresenteerd op de *'GEO '89'* conferentie bij Shape Technical Center (STC), oktober 1989.

1 INTRODUCTION

In the past few years there has been a growing interest in Geographic Information Systems (GIS). Among the many applications that use GIS technology are the Command, Control, Communication and Intelligence (C^3I) systems. A major advantage of a GIS over the paper map is that the various users can interact with the system. Data are shared among different users and each user has his own view of the data. For this purpose, the GIS has to be based on an appropriate persistent data structure. However, most existing GIS lack these data structures.

Many of today's information systems, including C^3I, are build around a relational database. However, the relational data model is primarily developed for business data processing systems. In this document we will use the *Geographic Analysis Intelligence System* as a case to illustrate the required functionality of a C^3I system. We will emphasize the typical geographical properties of this system. The Geographic Analysis Intelligence System is a modified (but still very realistic) version of a C^3I system that is currently under development for the Royal NetherLands Air Force (RNLAF). To solve the problems that arise from the geographic nature of the system, we propose an extension to the relational data model that includes geographic capabilities. We have chosen the relational data model as our starting point, because it is a powerful and well-known model.

In section 2, a description of the functions and geographic requirements of the Geographic Analysis Intelligence System is given. This is followed by an explanation of the relational data model in section 3. Reasons for using a relational database in C^3I systems are given in section 4. Having discussed the Geographic Analysis Intelligence System and the relational data model, we will look into possible solutions to the geographic requirements of the system, using this relational data model, in section 5. Then, we will summarize the problems that arise when using current relational database management systems (DBMS) for this. Because we want to stay within the relational data model for a number of reasons, possible geographic extensions to this model are discussed in section 6. After this, we will revisit the geographic requirements of the

Geographic Analysis Intelligence System, using these extensions. Finally, in section 7, conclusions and recommendations are given.

2 THE GEOGRAPHIC ANALYSIS INTELLIGENCE SYSTEM

The example of a C3I system that is used in this document as a case, the Geographic Analysis Intelligence System, is similar to a C3I system that is currently being developed for the intelligence organizations on the RNLAF airbases. Two of the main tasks of the intelligence organization on an airbase are:

- To supply various kinds of assessed information to pilots for mission preparation,
- To supply the battlestaff of an airbase with assessed scenario information, e.g. current and previous activities of own and enemy troops.

It is evident that the provided information (target data, enemy weapon system data, airspace management data, battle flow data, etc.) must be accurate and up-to-date. Maintenance of these data becomes more and more time consuming, especially in times of tension and war, when these data change very frequently.

To support the tasks of the airbase intelligence organization, the Geographic Analysis Intelligence System must perform the following functions:

- Reliable and fast storage of intelligence data (target information, enemy weapon systems information, planning lines) and airspace management data (flying routes and operations zones). These data may either be entered or updated in various ways, e.g. by the user, by external systems or by magnetic tape readers.
- Retrieval, presentation and distribution of intelligence and airspace management information.

Both data entry / updating and data presentation can be alphanumerical and graphical. In the alphanumerical case, the data are displayed in tabular form on the screen, using alphanumeric characters. Input of new or changed data is done by an input device like a keyboard. In case of graphical data manipulation and presentation, the data are displayed on a background map, using both alphanumeric characters and graphical symbols. Data are then entered or updated using an input device like a mouse or a light pen in combination with the background map. Both ways of interaction with the data are needed to support the intelligence organization in its main tasks.

As, for the purpose of this document, we are mostly interested in the geographic requirements of the system, we will examine them in more detail. The background maps that the system uses must contain several map layers with information about:

- administrative boundaries, e.g. borderlines of countries,
- lines of communication, e.g. rivers, roads, railroads,
- land classification and covering, e.g. swamps, forests,
- terrain elevation,
- names of objects.

Maps of various scales must be available, with the possibility to display more information on a more detailed background map. It will be clear that the system must contain large quantities (hundreds of megabytes) of these background map data. The data on the background maps are static, i.e. they change only once in every few months. Data that change very often are the geographical intelligence objects that the system must contain. The amount of these data is somewhat smaller than the amount of background data. The intelligence objects can be divided into four data types:

- Single point location. Objects of this type are described by only one location, often called pin-point location. Examples are most target objects and threat objects.
- Circle. Circular objects are described by one central location and by a radius. A number of target objects and airspace management zones are circular.
- Polyline. Polylines are described by n locations, where the first location must be connected to the second location, the second location to the third location , ... and the n-1th location to the nth location. Planning lines and flying routes are polylines.
- Polygon. Polygons are described by n locations, where the first location must be connected to the second location, the second location to the third location , ..., the n-1th location to the nth location and the nth location to the first location. Some target areas and some airspace management zones are of this data type.

An example of the presentation of geographical objects on a background map is given in figure 1. Note that the information density is very high. It is better to present only the relevant map layers, which can be (de-)activated by the user.

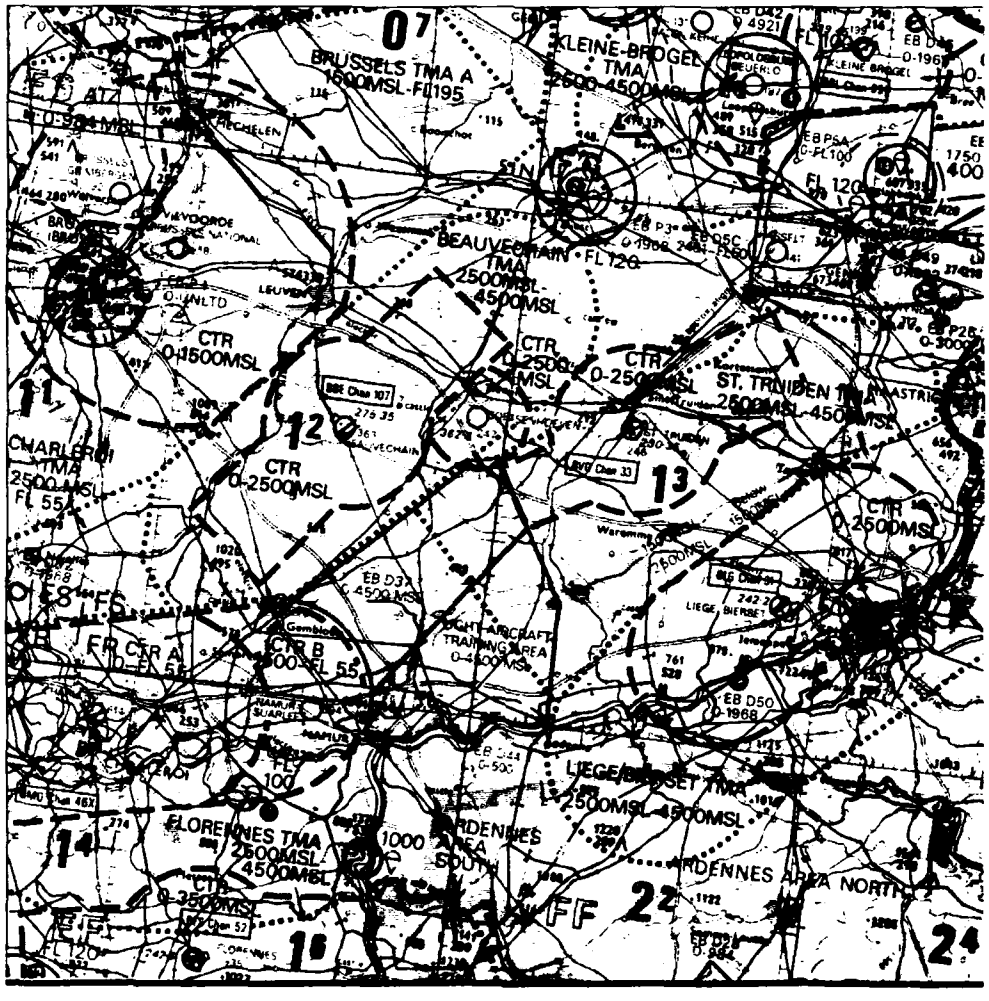

Figure 1: Background map with the geographic intelligence objects

Another geographical requirement of the Geographical Analysis Intelligence System is the need to store various levels of detail that can be distinguished in the system. Because background maps of varying scales are used, it is also required that the geographical

objects that are displayed on these maps can vary in detail. For instance, when a very schematic background map is used it is desirable to represent an airbase only as a pin-point location, using a simple symbol, whereas when a very detailed background map is used it may be desirable to display more details of the airbase, e.g. runways, shelters.

To display and manipulate geographical data (both on an alphanumerical and on a graphical display), it is very important how fast these data can be retrieved from and stored into the system. A typical retrieval operation is the selection of geographical objects of one of the types mentioned above within a certain area of interest. For instance, for the purpose of mission preparation, it is important to select only the relevant objects for display, because a very large number of geographical objects are available in the system and displaying them all would confuse the pilot. Relevant objects for the pilot would be the target of the mission, the threat objects that are near enough to the mission flying route to cause some threat and relevant airspace management data. The same applies to other functions of the Geographic Analysis Intelligence System.

Because of the data requirements of the Geographic Analysis Intelligence System it seems appropriate to use a relational database for storage and retrieval of the data. In the next section a description of the relational model is given.

3 THE RELATIONAL DATA MODEL

In this section we will give an overview of the basic concepts of the relation data model, that is the basis of Relational DataBase Management Systems (RDBMS). Next, a description is given how relational databases can be defined and manipulated, using a relational query language.

3.1 Mathematical description

As their name implies, relations play a major part in relational databases. First, a formal definition of a relation [Codd70]:

> Given a collection of sets D_1, D_2,... D_n, R is a relation on these sets if it is a set of ordered n-tuples $<d_1, d_2,... d_n>$ with $d_i \in D_i$. The sets D_i (for $1<i<n$) are called the domains of R and n is the degree of R. The elements d_i are the attributes of the relation.

In the relational database the data types of attribute domains are limited to: fixed length string, integer and real. A classic example is the following relation:

employee(**empl#**, name, salary, street, town)

In this example the employee number (empl#) and his salary are integers and the other attributes are strings of length 30. A relation consists of a number of tuples or records (attribute domains with actual values), which can be described by a two-dimensional table.

Each row of this table contains a tuple. The columns each describe a certain attribute. Within a relation at least one attribute (or combination of attributes) must have a unique value for each tuple. This attribute is called the primary key, i.e. it identifies the tuple and other attributes that are dependent on it. The primary key may not have an empty value. In our example empl# is the primary key. If an attribute (or a combination of attributes) other than those of the primary key are suitable for identification of the tuple, this attribute is called a candidate key. In the relation "employee" the attribute name is a candidate key. Candidate keys that are not primary key are called alternate keys. A

foreign key is an attribute (or combination of attributes) in a relation, whose value must match those of the primary key in another relation. The other attributes are referred to as secondary attributes. The table in figure 2 shows a fictive contents of the relation.

empl#	name	salary	street	town
775	van Hekken	20.000	Main street 3	Delft
816	van Oosterom	20.000	Boulevard 12	Woudenberg
940	Woestenburg	25.000	Station road 6	Delft

Figure 2: Fictive contents of a relation

The normal forms of relations are important during the design of a relational database application. They help to remove redundancy, so that it is easier to keep the database consistent if it changes. In the process of normalisation, the concept of (complete) functional dependency and multivalued dependency plays an important part. Functional dependency is defined as follows:

> Attribute *b* of relation R is functionally dependent on attribute *a* of R, if and only if every value of *a* determines exactly one value of *b*.

In this definition the attributes *a* and *b* may be a combination of "atomic" attributes.
Complete functional dependency is defined as follows:

> Attribute *b* of relation R is complete functionally dependent on attribute *a* of R, if attribute *b* is functionally dependent on attribute *a* and not on any subset of attribute *a*.

Multivalued dependency is defined as follows:

> There is a multivalued dependency from attribute *a* of relation R to attribute *b* of R, if a value of *a* determines a set of values of *b*. Both *a* and *b* will be part of the (candidate) key of R.

Using these definitions, the five normal forms are defined as follows:

- NF1: A relation is in first normal form if and only if all attribute domains contain "atomic" values.

- NF2: A relation is in second normal form if and only if it is in first normal form and every of its (secondary) attributes are complete functionally dependent on the candidate key of the relation.
- NF3: A relation is in third normal form if and only if it is in second normal form and every of its secondary attributes are not transitively dependent on the candidate key of the relation.
- NF4: A relation is in fourth normal form if and only if it is in third normal form and does not contain more than one multivalued dependency.
- NF5: A relation is in fifth normal form if and only if it is in fourth normal form and it cannot be decomposed into three or more smaller tables without loss of information.

Note that the relation "employee" defined above is in fifth normal form.

3.2 The relational query language

To manage relational databases, current RDBMS contain query languages like SQL (Structured Query Language [Date81]), that will be used as an example in the rest of this document. The main parts of SQL are the Data Definition Language (DDL) and the Data Manipulation Language (DML). The most important property of the DDL is to define relations in the database (called tables). For example, in figure 3.a the relation "employee" is defined using the DDL part of SQL.

```
CREATE TABLE EMPLOYEE
            (     EMPL_NR          NUMBER(3)   NOT NULL,
                  NAME             CHAR(30),
                  SALARY           NUMBER(5),
                  STREET           CHAR(30),
                  TOWN             CHAR(30)   );
```

Figure 3.a: SQL definition of relation "employee"

Using the DML, existing tuples can be deleted from the relation (table) and new ones can be added, using the operations DELETE resp. INSERT. It is also possible to update the

attribute values of existing tuples, using the operation UPDATE. This is not allowed for the primary key. Data can be retrieved from the data base by searching in a table, using the operation SELECT. The available operators can be categorized into the following classes [ChFu80]:

- comparison operators (=, ≠, <, ≤, > and ≥);
- logical operators (AND, OR and NEGATE);
- statistical operators (MIN, MAX, COUNT, TOTAL and AVERAGE);
- set operators (Cartesian product, union, intersection and difference);
- special database operators (restriction, projection and join).

These operators are the basic parts of the query language. The queries can be formulated by an application program (through a call to the RDBMS) or directly by the operator. In this way it is possible to ask both simple and more complex questions (called queries) to the RDBMS and specify the format of the required answer. For example, the question "Select the names of the employees who live in Delft", is formulated in figure 3.b, using SQL. The result of this query is given in figure 3.c.

```
SELECT NAME
FROM EMPLOYEE
WHERE TOWN = "Delft"
```

Figure 3.b: Example of SQL query

```
name


van Hekken
Woestenburg
```

Figure 3.c: Result of query of 3.b

A very important operation is the joining of relations. The attribute of one relation is matched with a corresponding attribute in another table. All combinations make up the tuples of the new relation. For instance, if we define the relation project (**proj#, empl#**) in

which both attributes are primary key and create a corresponding table with SQL, that contains the data given in figure 4.a, we can ask the question "Select the names of the employees that are working on project with number 20357". In SQL, this question is stated as in figure 4.b. The two tables are joined by the "empl#". The result of the query is given in figure 4.c.

proj#	empl#
20461	775
20357	775
20357	816
20461	940

Figure 4.a: Fictive contents of the relation "project".

```
SELECT NAME
FROM PROJECT, EMPLOYEE                          this is the join operation
WHERE PROJECT.EMPL_NR = EMPLOYEE.EMPL_NR        this specifies the join
AND PROJECT.PROJ_NR = 20357
```

Figure 4.b: SQL query to select all employee names working on project 20357

name
van Hekken
van Oosterom

Figure 4.c: Result of query of 4.b

In case of selection of tuples from a relation, all tuples have to be visited and checked for the right attribute value. This will take a lot of time in a realistic large table and, if required often, is very annoying. In many commercial implementations, including ORACLE, INGRES, etc. this problem can be solved by putting an index on often searched attributes of the relation. This is an additional structure (sorted on the

attribute(s)) that contains references to the original tuples. The index is implemented as a B-tree [BaMc73], which allows searching in O(log n) time.

In the next section we will give more reasons to use RDBMS in C3I systems. In the subsequent section examples of an implementation of the Geographic Analysis Intelligence System, using current RDBMS, will be presented.

4 THE USE OF A RELATIONAL DATABASE

Like many of today's C3I systems, the Geographic Analysis Intelligence System will be built around a RDBMS to store the data. Background maps, although they do not change very often, should, if possible, be stored in the same database. There are a number of reasons for using a RDBMS:

- By using a relational database, data redundancy and data inconsistency are avoided, because every single item is stored only once.
- A relational database ensures a flexible growth path in case of changes in operational concepts and changes in the structure of the data that have to be managed.
- A relational database ensures enough flexibility to support simple and efficient interfacing with other C3I systems.

After the concept of the relational data model, current RDBMS are primarily developed for business data processing. In these systems the data that have to be managed are not geographically oriented and there is generally no need for graphical presentation of information. However, as will be clear from the description in a previous section, the Geographic Analysis Intelligence System does contain these geographical data.

Based on the graphical functionality of the system required, we will describe the problems that arise when we try to develop a pure relational solution for it.

5 A PURE RELATIONAL SOLUTION OF THE PROBLEM

In this section we will first develop a pure relational solution of some geographical aspects of the case, both in the area of data definition and in the area of data manipulation. This or a similar approach is used in several systems described in the literature. For example, GEO QUEL [BeSt77], a geographic information retrieval and display system build on top of INGRES. As we will see, there are some problems with these solutions, which are summarized in the third subsection. In the last subsection we will provide reasons not to leave the relational data model, but to extend current RDBMS with geographical capabilities.

5.1 Definition of data

This subsection shows that it is possible to store geographical data of the Geographic Analysis Intelligence System in a RDBMS without any modifications or extensions. The first kind of geographical data that we want to store are the (static) background map data (borderlines of countries, rivers, trafficways, etc). The solution presented here is a slightly simplified version of the data structure presented by Van Roessel [Roes87], who developed this data structure mainly for the interchange of geographical data. He applies a technique (described by Smith [Smit85]) for developing a set of fully normalized relations to the topological data model. This is a well-known data model [PeCh75] that deals with nodes, chains and polygons (see figure 5), and hence is very suitable for describing the kind of background data that we want to use. The following relations are defined:

1 point(**pntid**, x, y) **bold items** denote a primary key
2 node(**nodeid**, *pntid*) *italic items* denote a foreign key
3 chainpoints(**chainid, pntid, pntseq**) multiple points in same chain
4 chaintopol(**chainid**, *strnode*, *endnode*, *lftpol*, *rgtpol*)
5 polygon(**polid, chainid**, chnseq)

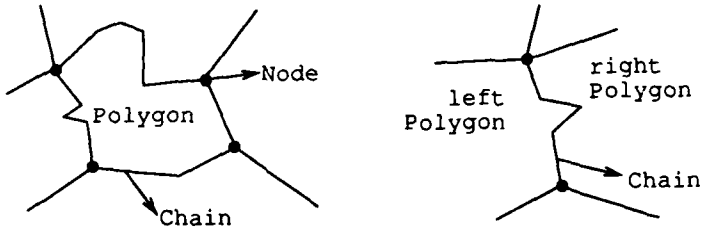

Figure 5: Chains and nodes

The chain concept of the topological data model is captured in two relations: chainpoints and chaintopol. The first one describes the shape of the chain. As the order of points within a chain are important and tuples within a relation are not ordered, the chainpoints relation must also have an attribute describing this order (the attribute "pntseq"). The second relation (chaintopol) captures the topological aspects of a chain.

It is surprising that these intuitively clear (and good) relations are derived by Smith's techniques and therefore fully normalized. The fact that polygons may have (nested) holes is removed from Van Roessel's example because of clarity. For the same reason, only a single map layer (one geographic theme) is modelled. However, the concept of spatial domain (a limited geographic region of interest) is removed from the model of Van Roessel. The domain should, in principle, be infinitely large and not limited, as the paper map sheets, to a certain region.

As we have seen, the Geographic Analysis Intelligence System contains a number of dynamic geographical objects, some of them at levels of various detail. Target objects (airbases, buildings, bridges, etc.) and threat objects are described by a single point location. Target areas, planning lines and airspace management data are described by polygons or polylines. To store these objects, that are (:n contradiction to, for instance, the background polygons) independent of each other, somewhat simpler relations than the

ones given above can be used. For instance, targets and threats that can be described by one point location are defined as:

target(**target_id**, *pntid*, target_atr_1, ..., target_atr_n)
threat(**threat_id**, *pntid*, threat_atr_1, ..., threat_atr_n)

Operations zones that can be described as an area are defined as:

ops_zone(**ops_zone_name**, ops_zone_atr_1, ..., ops_zone_atr_n)
ops_zone_area(**ops_zone_name, ops_zone_loc_seq**, *pntid*)

This relation contains the locations that form lines that enclose the operations zone (area). If the area is described by n locations, location 1 must be connected (by a line part) to location 2, location 2 to location 3, ... and location n to location 1. For this purpose, a location ordering attribute is used ("ops_zone_loc_seq"). Note that *pntid* is used instead of *chainid* in the relation ops_zone_area. This is because we are not interested in the topology.

When using objects with various detail levels, the attribute *pntid* (and possibly some other geographic attributes) will normally be included (as a foreign key) in all the relations that describe those detail levels.

5.2 Spatial Queries

Beside the possibility to define data in current RDBMS (using the DDL part), it is also possible to query and manipulate those data (using the DML part). All RDBMS use some query language to retrieve and manipulate data of the database. For this purpose, the query language contains relational operators. Because, as we have seen, the main functions of the Geographic Analysis Intelligence System are entry, updating, storage, retrieval and presentation of (partly geographic) intelligence and airspace management data, it would be desirable to have the possibility to ask questions like "Select all threat objects within an operations zone" or "Select the target objects that are closest to a fixed point location" just as easy as questions like "Select the names of the employees who live in Delft" or "Select the employee number of the employee with the highest salary".

However, in current RDBMS, spatial (geographical) queries become very complex and difficult. For instance, the question "Select all threat objects within an operations zone" can be stated relatively easy in SQL when the operations zone is a rectangle denoted by the opposite corners x_1, y_1 and x_2, y_2 (see figure 6.a). However, if the operations zone is a polygon of an arbitrary number of points, the SQL query will become very complex (if not impossible). Furthermore, it is not possible to ask the question "Select the target objects that are closest to a fixed point location" in an easy and natural way in SQL, as proposed in figure 6.b. In this example query the fixed point location is denoted by "x_f" and "y_f".

```
SELECT THREAT_ID
FROM THREAT, POINT
WHERE TARGET.PNTID = POINT.PNTID
AND   x1 < POINT.X < x2
AND   y1 < POINT.Y < y2
```

Figure 6.a: Example of selection of threats within rectangle

```
SELECT MIN(DISTANCE(POINT.X, POINT.Y, xf, yf)
FROM TARGET, POINT
WHERE TARGET.PNTID = POINT.PNTID
```

Figure 6.b: Example of minimum distance determination

In these query examples a join of two relations is necessary, which would not be the case if data types like a single point location were available. It would be even more difficult (if not impossible) to write the distance calculation formula using the available relational operators of SQL. The reason for the complexity c.q. impossibility of the geographical queries is obvious. The RDBMS has no knowledge of the way in which the geographical data types (points, polylines and polygons) are implemented and, as a consequence, has no straightforward relational operators, like DISTANCE, available for these data types, as it has for numbers and character strings.

The same problems apply to questions like "Select all threat objects within a certain operations zone". In SQL no operation is available to test directly whether a point location is located within an area that is described in a manner like the relation "ops_zone_area".

Some of the problems that we have seen from the examples in this subsection will be summarized in the next subsection.

5.3 Problems of the pure relational solution

The major spatial requirements for a database can be stated as follows [RoFS88]: "The database must support domains which consist of non-atomic non-zero space objects". Beside storing these objects this also means that the database must support direct spatial search and computation.

As will be clear by now, the spatial properties required by the presented case are very hard or even impossible to meet with current RDBMS. Reasons for this are:

- Current RDBMS lack geographic data types. Point locations must be implemented using character strings or numbers. Polylines and polygons are represented by multiple rows in a table with a repetition of the id of the object on each row. Furthermore, a sequence number must be added to each row (denoting a point location), because these would otherwise be unordered. This is all redundant data storage which can be avoided by other representations.
- As a second and closely related problem, the manipulation of objects is complex: a difficult query, possibly followed by some processing. This is due to the fact that because current RDBMS lack geographical data types, there are no geographical operators available and a very complex combination of the available relational operators for character strings and numbers must be chosen (if possible). Sometimes it is even impossible to perform this processing within the database and special application functions must be provided, i.e. the knowledge of the geographical data structure must be built into the applications. For example, the calculation of the area of a polygon.

- Finally, efficient search for spatial objects is impossible. Only one-dimensional search can be performed efficiently, that is in O(log n) time.

Despite these problems, in the next subsection we will argue that it is not convenient and not necessary to completely abandon the relational data model.

5.4 Sticking to the relational data model

Although the relational DBMSs are designed primarily for business data processing, the developed database concepts, as we have seen earlier, such as data independence, data integration, controlled redundancy, security and privacy are equally important for databases with geographic capabilities. This is one reason for not leaving the relational model. Another reason is that the relational model is conceptually simple and well-known with large groups of users. In every spatial database there also has to be a way to store the normal (alphanumerical) data, so why not use the relational model for this purpose. This approach is chosen by several implementators of commercially available GIS. Because of the problems with the spatial data, their database is often split into two parts (not necessarily noticeable by the end-user): a relational part and a special part for the geometric data. These two parts are linked by corresponding object-id's of the attribute and geometric parts. It will be clear that it is more elegant to combine the two database parts in one system and thus avoiding the problem of linking these together.

In the following section we will propose some extensions to the relational model to make it more suitable for geographic operations. These extensions will occur in the DDL part of the query language, e.g. like additional data types and more than one level of detail, as well as in the DML part of the query language. As we will see, because of these extensions, new index structures must be provided too.

6 THE GEOGRAPHIC EXTENSION

The actual geographic extension of the relational data model consists of three parts. First, new geographic data types (domains) are introduced. Second, operators are defined for these new data types. Third and finally, new indexing methods are required when dealing with geographic information. Note that in this document we concentrate on the two dimensional extension. However, three or higher dimensional extensions are quite similar.

We will describe these extensions at a conceptual level, that is, the exact syntax (of for example the query language) will be omitted. Further, the alphanumerical interaction technique is not sufficient when dealing with geographical data. Some kind of combination with graphical interaction is needed, but this is also outside the scope of the research described in this document.

6.1 Data types (or domains)

Nearly all geographic data processing [NaWa79, Oost88] is performed with the vector, the raster or a combination of these geometric data format. The vector format has three subtypes: *point, polyline* and *polygon.* The emphasis in this document is on the vector representation, because it allows more flexible manipulations, though the raster representation has also advantages. Though their representations might be complex, these new data types must be regarded as atomic values in the data model. So it is possible to define the following relations:

target(**target_id**, target_point, target_atr_1, ..., target_atr_n)
threat(**threat_id**, threat_point, threat_atr_1, ..., threat_atr_n)
ops_zone(**ops_zone_id**, ops_zone_polygon, ops_zone_atr_1, ..., ops_zone_atr_n)

Note that the relation "ops_zone_area" is no longer necessary. The relations are at least in the first normal form, because they only contain atomic attributes. The data types of target_point, threat_point and ops_zone_polygon are POINT(2) and POLYGON(2), respectively. The "2" behind POINT and POLYGON indicates that this is a two

dimensional attribute. POLYGON(2) does not mean that this is a polygon with 2 points! In the same manner it is possible to use these data types in a three-dimensional variant.

The circle is not included as a basic data type, though it is one of the intelligence objects, see section 2. The rationale behind this is that the proposed extension must be compact. The introduction of a circle would also require operators for the circle. If a circle is introduced, why not also introduce an ellipse, a spline, and so on. The circle can be approximated by a polygon.

A more advanced extension might also include the data types: *procedure* and reference to another relation (*query*). Both these data types are useful for the implementation of the detail levels. Associated with a polyline or polygon is a line generalization algorithm to reduce the number of points used, when working with small scale maps. Another method of dealing with detail levels is to allow references to other tables describing the refinement of objects at a larger scale map [ChKu81].

6.2 Spatial Operators

We will only describe the basic operators. More complex operators are not part of the database system, but belong in a specific application. For example: network calculations (shortest path, travelling salesman, location of service centre), advanced visualization techniques (PRISM maps, Digital Elevation Models) or simulation (evacuation of a region by road, flow of liquids through pipes). The polygon-overlay takes two sets of polygons and calculates all intersections, which results in a third set of polygons. Though it is a complex operation, it is used in many GIS applications. So, perhaps it should also be in the set of standard operators.

Many spatial operators have been described by various authors [ChFu80, JoCa88, Güti88, RoFS88]. We do not claim that our lists of operators are complete, but they should give a good impression of the basic spatial operators. We distinguish three fundamental classes of spatial operators in addition to the five classes of section 3.2. These are: spatial comparison operators, spatial calculations operators and topological operators. Also, there must be an "output operator" that directly displays the result of a geographical query on a graphics display.

- *Spatial calculation operators* return a real/integer value or geometric value. Some of the most important operators are: DISTANCE (operands: two points, polylines or polygons), LENGTH (operand: one polyline), PERIMETER (operand: one polygon), AREA (operand: one polygon), CLOSEST (operand: one point, polygon or polyline (and a set of candidate objects)), INTERSECTION (operands: two polylines or polygons), UNION (operands: two polylines or polygons).
- *Topological operators* return a geometric value. Some examples: neighbours, next link (in a polyline network), left and right polygons of a polyline, begin and end nodes of polylines. We are not sure that we need a special class for the topological operators, because the topological model can be captured in very natural manner in the standard relational model, see section 5.1.
- *Spatial comparison operators* return a Boolean (TRUE or FALSE). Though the actual calculations are often (partly) the same as in the previous classes of operators, they form a separate group. Some examples: INTERSECTS, INSIDE, OUTSIDE, NORTH_OF, NEIGHBOUR_OF, LARGER_THAN. Note that it is often possible to simulate these operators by combining the normal comparison operators and the other geographic operators.

There are always other spatial operators, that might be useful for some application. But it is impossible to include them all. An "open" database is the solution for this problem. If a operator is not within the set of basic spatial operators, it is implemented (by the user) and after it is certified it is added to the database system. In this way other users also benefit from the new capabilities. Note that organizational actions have to be taken, e.g. someone must have the responsibility for clear and unique names of operators. Some examples of non-basic spatial operators: calculation of the voronoi diagram, the convex hull, the smallest enclosing circle, and so on. These are pure geometric problems, but less common than the ones in the set of basic operators. The research in the new area of science called "Computational Geometry" [ShPr85, BlGü88] offers very efficient techniques for implementing the (basic and non-basic) spatial operators. Regardless of the implementation it is possible to formulate queries with the basic operators (in SQL-like syntax) as in figure 7.

```
SELECT MIN(DISTANCE(TARGET_POINT, point))
FROM TARGET                          Query of section 5.2 is indeed possible.

SELECT CLOSEST(TARGET_POINT, point)
FROM TARGET                          Same query, but formulated more efficient.

SELECT THREAT_ID
FROM   THREAT
WHERE  INSIDE((SELECT OPS_ZONE_POLYGON
               FROM OPS_ZONE
               WHERE OPS_ZONE_ID = 12), THREAT_POINT)
```

Figure 7: Examples of queries using "extended" operator set

6.3 Indexing techniques

The B-tree, an indexing technique used in many database implementations, combines several desirable properties. It is a dynamic height balanced structure, that is, inserts, deletes and updates of entries may be interchanged by searches. Because of the balanced nature searches are efficient (in O(log n) time). Also, the B-tree has a high memory occupation rate [BaMc73].

However, the B-tree is only suited for one-dimensional (number/string) attributes. Multiple indexes on more than one attribute of a relation is possible, but (with current implementations of RDBMS) only one can be used for solving a query like: "Select all employees with empl# > 700 and 15.000 < salary < 22.500" or "Select all threats in the region 5 < x < 10 and 12 < y < 20". These point queries can be solved efficiently by the *KDB-tree* [Robi81]. The KDB-tree is a KD-tree adapted for secondary storage and can handle point data from any dimension. The KD-tree can not handle the other geometric data types: polyline and polygon. In the literature there are several solutions for this problem, such as: R-tree, R^+-tree [FaRS87], Field-tree [Fran83], Cell-tree [Günt88]. We will not discuss the KDB-tree and the other tree structures, except for the R-tree. The R-tree is described in a little more detail as an exemplary spatial indexing method.

The *R-tree* was defined by Guttman [Gutt84] in 1984. The leaf nodes of the R-tree contain entries of the form: *(I,object-identifier)*, where *object-identifier* is a pointer to a data object and *I* is a bounding box (or Minimal Bounding Rectangle, MBR). An advantage of the R-tree is that (pointers to) objects (e.g. polygons) are stored in the leaf nodes, instead of lower level primitives which must be assembled to the original object. The internal nodes contain entries of the form: *(I,child-pointer)*, where *child-pointer* is a pointer to a child and *I* is the MBR of that child. The maximum number of entries in each node is called the *branching factor M* and is chosen to suit paging and disk I/O buffering. The INSERT and DELETE algorithms of Guttman assure that the number of entries in each node lies between m and M, where $m \leq M/2$ is the minimum number of entries per node.

Figure 8 shows a R-tree with two levels and $M=4$. The lowest level contains tree leaf nodes and the highest level contains one node with pointers and MBRs of the leaf nodes. *Coverage* is defined as the total area of all the MBRs of the leaf nodes. *Overlap* is the total area contained within two or more leaf MBRs. In figure 8 the coverage is $A \cup B \cup C$ and the overlap is $A \cap B$. It is clear that efficient searching demands both low coverage and overlap; a *packed* R-tree. Roussopoulos et al. [RoLe85] describe the PACK algorithm which creates an initial R-tree that is more efficient than the R-tree created by the INSERT algorithm. The *R+-tree*, a modification of the R-tree, avoids overlap at the expense of more nodes and multiple references to some objects.

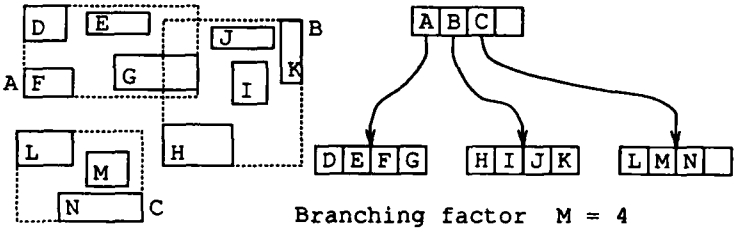

Figure 8: R-tree

7 CONCLUSION

It is impossible to cover all topics related to geographic databases in this short document. For example, issues concerning: the use of raster data, the incorporation of knowledge based techniques and more pictorial oriented user interfaces, were hardly discussed. The case *Geographic Analysis Intelligence System* is an information system with typical geographic requirements. The proposed extension to the relational data model can be used as the basis for all kinds of Geographic Information Systems. In fact the model is also useful for other pictorial applications, for example CAD/CAM systems.

We tried to keep the extension realistic and within the concepts of the original relational data model. With realistic we mean that the implementation is based on known, recently developed techniques. So, it should be possible to implement the model within the next two or three years. At the moment we are starting the development on Sun workstations of a prototype of the system described in this document. We have the intention to use POSTGRES [StRo86, Post88], the successor of INGRES, for this purpose. POSTGRES is an experimental RDBMS of the University of California (Berkeley), that supports complex objects (instead of only strings, integers and reals) and provides user extensibility for data types, operators and access methods.

REFERENCES

[BaMc73] R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1:173-189, 1973.

[BeSt77] Richard R. Berman and Michael Stonebraker. GEO-QUEL a system for the manipulation and display of geographic data. *ACM Computer Graphics*, 11(2):186-191, 1977.

[BlGü88] G. Blankenagel and R.H. Güting. Internal and external algorithms for the points-in-regions problem - The inside join of geo-relational algebra. In *CG'88, International Workshop on Computational Geometry*, pages 85-89, March 1988.

[ChFu80] N.S. Chang and K.S. Fu. *A Relational Database System for Images*, volume 80 of *Lecture Notes in Computer Science*, pages 288-321, Springer-Verlag, 1980.

[ChKu81] Shi-Kuo Chang and Tosiyasu L. Kunii. Pictorial data-base systems. *Computer (U.S.A.)*, 14(11):13-21, November 1981.

[Codd70] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377-387, 1970.

[Date81] C.J. Date. *An Introduction to Database Systems*. Addison-Wesley Publishing Company, Massachusetts, 1981.

[FaSR87] Christos Faloutsos, Timos Sellis, and Nick Roussopoulos. Analysis of object oriented spatial access methods. *ACM SIGMOD*, 16(3):426-439, December 1987.

[Fran83] André Frank. *Storage methods for space related data: The FIELD TREE*. Eidgenössische Technische Hochschule Zürich, Institut für Geodäsie und Photogrammetrie, Bericht nr. 71, 1983.

[Günt88] Oliver Günther. *Efficient Structures for Geometric Data Management*. Number 337 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1988.

[Güti88] R.H. Güting. Geo-relational algebra: A model and query language for geometric database systems. *Advances in Database Technology - EDBT'88*, pages 506-527, March 1988.

[Gutt84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *ACM SIGMOD*, 13:47-57, 1984.

[JoCa88] Thomas Joseph and Alfonso F. Cardenas. PICQUERY: A high level query language for pictorial database management. *IEEE Transactions on Software Engineering*, 14(5):630-638, May 1988.

[NaWa79] George Nagy and Sharad Wagle. Geographic data processing. *Computer Surveys*, 11(2):139-181, June 1979.

[Oost88] Peter van Oosterom. Spatial data structures in Geographic Information Systems. In *NCGA's Mapping and Geographic Information Systems* (Orlando, Florida), pages 104-118, September 1988.

[PeCh75] Thomas K. Peucker and Nicholas Chrisman. Cartographic data structures. *The American Cartographer*, 2(1):55-69, 1975.

[Post88] POSTGRES Reference Manual, 1988.

[PrSh85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry*. Springer-Verlag, 1985.

[Robi81] J.T. Robinson. The KDB-tree: A search structure for large multidimensional dynamic indexes. *ACM SIGMOD*, 10:10-18, 1981.

[Roes87] J.W. van Roessel.Design of a spatial data structure using the relational normal forms. *International Journal of Geographical Information Systems*, 1(1):33-50, 1987.

[RoFS88] Nick Roussopoulos, Christos Faloutsos, and Timos Sellis. An efficient pictorial database system for PSQL. *IEEE Transactions on Software Engineering*, 14(5):639-650, May 1988.

[RoLe85] Nick Roussopoulos and Daniel Leifker. Direct spatial search on pictorial databases using packed R-trees. *ACM SIGMOD*, 14(4):17-31, December 1985.

[Smit85] H. C. Smith. Database design: composing fully normalized tables from a rigourous dependency diagram. *Communications of the ACM*, 28: 826, 1985.

[StRo86] Michael Stonebraker and Lawrence A. Rowe. The design of POSTGRES. *ACM SIGMOD '86*, 15(2):340-355, 1986.

UNCLASSIFIED

REPORT DOCUMENTATION PAGE (MOD-NL)

1. DEFENSE REPORT NUMBER (MOD-NL)	2. RECIPIENT'S ACCESSION NUMBER	3. PERFORMING ORGANIZATION REPORT NUMBER
TD 891083		FEL-89-B207
4. PROJECT/TASK/WORK UNIT NO.	**5. CONTRACT NUMBER**	**6. REPORT DATE**
20357	-	AUGUST 14, 1989
7. NUMBER OF PAGES	**8. NUMBER OF REFERENCES**	**9. TYPE OF REPORT AND DATES COVERED**
30	24	FINAL REPORT

10. TITLE AND SUBTITLE
A GEOGRAPHICAL EXTENSION TO THE RELATIONAL DATAMODEL

11. AUTHOR(S)
M.C. VAN HEKKEN, P.J.M. VAN OOSTEROM, M.R. WOESTENBURG

12. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
PHYSICS AND ELECTRONICS LABORATORY TNO, P.O. BOX 96864, 2509 JG THE HAGUE
OUDE WAALSDORPERWEG 63, THE HAGUE, THE NETHERLANDS

13. SPONSORING/MONITORING AGENCY NAME(S)
TNO DIVISION OF NATIONAL DEFENSE RESEARCH, THE NETHERLANDS

14. SUPPLEMENTARY NOTES
THE PHYSICS AND ELECTRONICS LABORATORY IS PART OF THE NETHERLANDS ORGANIZATION FOR APPLIED SCIENTIFIC RESEARCH

15. ABSTRACT (MAXIMUM 200 WORDS, 1044 POSITIONS)
MODERN COMMAND AND CONTROL INFORMATION SYSTEMS CONTAIN BOTH ALPHANUMERICAL AND GEOGRAPHICAL DATA (E.G. MAPS). EFFICIENT STORAGE AND RETRIEVAL OF GEOGRAPHICAL INFORMATION FROM A RELATIONAL DATABASE IS NOT POSSIBLE WITH CURRENT TECHNOLOGY.
THIS REPORT DEALS WITH THIS PROBLEM AND PRESENTS A POSSIBLE SOLUTION. THE SOLUTION IS BASED UPON THE EXTENSION OF THE RELATIONAL MODEL WITH SOME DOMAIN-TYPES AND DATABASE OPERATORS AND USES SPECIAL INDEX-STRUCTURES.

16. DESCRIPTORS	IDENTIFIERS
CHARTS COMMAND CONTROL COMMUNICATION & INTELLIGENCE DATABASES INFORMATION STORAGE AND RETRIEVAL	GEOGRAPHICAL INFORMATION SYSTEMS (GIS)

17a. SECURITY CLASSIFICATION (OF REPORT)	17b. SECURITY CLASSIFICATION (OF PAGE)	17c. SECURITY CLASSIFICATION (OF ABSTRACT)
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED

18. DISTRIBUTION/AVAILABILITY STATEMENT	17d. SECURITY CLASSIFICATION (OF TITLES)
UNLIMITED AVAILABLE	UNCLASSIFIED

UNCLASSIFIED