In R. Vinken, editor, Proc. Int. Coll. on Construction and Display of Geoscientific Maps Derived from Databases

Geol. Jb.	A 104	139–152	3 Fig.	Hannover 1988

A Database Language for Scientific Map Data

HANS-DIETER EHRICH, FRIEDRICH LOHMANN, KARL NEUMANN & ISA RAMM

Digital geoscientific maps, database, user interface, database language, data definition, data manipulation

Abstract: The paper presents the user interface of a DBMS for geoscientific applications. The user interface permits modelling geoscientific worlds in terms of (atomic and complex) objects and relationships; moreover, it offers pre-defined concepts for the graphic representation of stored information in maps. Beside the standard alphanumeric data types, geometric and temporal data types are offered. A database language is introduced that integrates the modelling concepts and the data types; it includes statements for data definition, for data manipulation and for the construction of maps.

[Eine Datenbanksprache für wissenschaftliche Karten]

Kurzfassung: In dieser Arbeit wird die Benutzerschnittstelle eines DBMS für geowissenschaftliche Anwendungen vorgestellt. Die Benutzerschnittstelle erlaubt die Modellierung geowissenschaftlicher Welten in Form von (atomaren und komplexen) Objekten und Beziehungen; darüber hinaus bietet sie vordefinierte Konzepte für die graphische Darstellung gespeicherter Informationen in Karten. Außer den üblichen alphanumerischen Datentypen werden geometrische und zeitliche Datentypen angeboten. Es wird eine Datenbanksprache vorgestellt, die die Modellierungskonzepte und Datentypen integriert; sie enthält Anweisungen zur Datendefinition, zur Datenmanipulation und für die Konstruktion von Karten.

Contents

Authors' address: Prof. Dr. H.-D. EHRICH, Dipl.-Inf. F. LOHMANN, Dipl.-Inf. K. NEUMANN, Dipl.-Inf. I. RAMM, Institut für Programmiersprachen und Informationssysteme, Technische Universität Braunschweig, Gaußstraße 12, D-3300 Braunschweig.

1. Introduction

Within the Deutsche Forschungsgemeinschaft priority project "Digital Geoscientific Maps" (VINKEN 1985), our project group in Braunschweig has been working on the subproject "Geo-DBMS Design".

As commercially available database systems are not suitable for so called non-standard applications - such as storing and retrieving geometric objects - we are aiming at designing a user interface that meets geoscientific requirements and developing an experimental prototype for this user interface.

The corresponding geo-DBMS software is to be implemented on top of a non-standard DBMS kernel, which is being developed by SCHEK's group in Darmstadt (DEPPISCH et al. 1985, SCHEK 1986, SCHEK & WATERFELD 1986). While the DBMS kernel will provide for efficiency in storing, retrieving and updating geoscientific data, our user interface offers convenient, high-level concepts for issuing complex and powerful database commands.

On top of the user interface, arbitrary application programs may be implemented; one of these will be a cartographic editor, which is being developed by BOLLMANN's group in Berlin. The context within the research project is shown in Fig. 1.

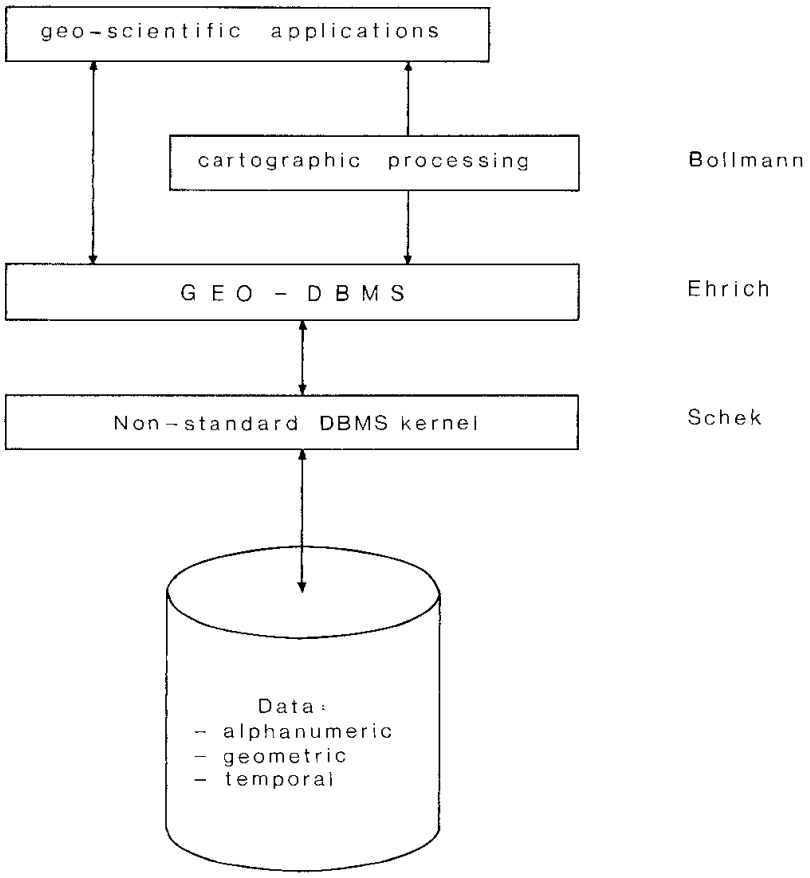

Fig. 1: Context within research project.

The user interface can be characterized by three basic features, which will be described in the following chapters:

- For the conceptual modelling of arbitrary geoscientific "worlds", there is a specific "geo-object model", which permits, among other things, the modelling of complex objects; we designed this model by extending the well-known ER-model and tailoring it to the specific needs of geoscientific applications (cf. chapter 2).
- Beside the standard alphanumeric data types, we offer geometric data types such as POINTS, LINES, and POLYGONS, and temporal data types for handling versions (cf. chapter 3).
- Based upon these features, we have developed a database language, which integrates the concepts of the geo-object model and the data types. The language includes a data definition part for conceptual modelling, a data manipulation part for creating, retrieving, updating and deleting geo-objects, and it provides commands for the construction of maps (cf. chapter 4).

The paper concludes with a survey of our current and future activities (chapter 5).

More details concerning all the aspects of the user interface can be found in RAMM et al. (1985) and LIPECK & NEUMANN (1986).

2. Geo-Object Model

2.1 Basic Ideas

In this chapter we present a brief introduction to the concepts of our geoobject model, which provides the basis for designing and structuring database schemata for arbitrary geoscientific applications. This model is based on the entity-relationship model (ER-model) proposed by CHEN (1976), which is well tried and widely accepted; we have enriched the ER-model by a few additional concepts and adapted it to the specific needs of geoscientific users.

A basic principle of our model is to keep the information about geoscientific objects separated from their graphic representation in maps. This principle is supported by distinguishing between geo-objects and mapobjects.

Using the geo-object model, geoscientific "worlds" are described in terms of geo-objects. These geo-objects represent the entities about which the user wishes to store information and which he wants to treat as a whole within his database, such as cities, rivers, states, vegetation areas, etc.; these entities may actually exist in the real world or they may just constitute an abstraction from reality. Most queries, evaluations and updates against the database affect these geo-objects, since they represent the geoscientific world the user is dealing with.

For graphic output of stored information, the model permits the construction of maps. In a map, geo-objects are represented by map objects which are derived from the information stored about geo-objects. Maps may be plotted, displayed on the screen or be subjected to further processing by a cartographic editor.

The distinction between geo-objects on the one hand and map objects on the other hand allows to store complete information about a geo-object, e.g. a city, including its geographic location and geometric shape, and choosing different representations (map

objects) for this city in different maps, e.g. its area or the border line of its area in a large scale map, or a small circle or single point in a small scale map.

2.2 Modelling Geoscientific Worlds

The geo-object model offers a set of universal concepts that enable the user to design appropriate database schemata for arbitrary geoscientific applications.

The model centers around the concept of geo-object. Any entity of a geoscientific "world", whether existing in reality or only in abstraction, that the user whishes to treat as a whole within his database, may be modeled as a geo-object, e.g. the river Rhine, the city of Cologne, the state North Rhine Westfalia, etc. Geo-objects of the same kind are treated as members of the same geo-object class, e.g. "River, "City" or "State". The choice of geo-object classes will typically depend on the respective branch of geosciences, like geology, soil sciences, etc.

In the following, we will often use the term "object" synonymously to "geoobject" whenever no confusion between geo-objects and map objects can occur.

Every single object is described by its attribute values; the city of Cologne, for instance, might be described by its name, its geometry and its number of inhabitants. Accordingly, every object class is characterized by its attributes and their respective data types from which the attribute values may be taken; the object class "City", for instance, might be characterized by the attributes "Name" (data type: STRING), "Geometry" (data type: POLYGONS) and "Number of inhabitants" (data type: INTEGER).

The object classes are typically characterized by the attributes NAME, GEOMETRY, VERSION and a variety of thematic attributes peculiar to the specific class. The data types (alphanumeric, geometric and temporal) provided for the attribute values are presented in chapter 3.

Beyond the description by attributes, geo-objects may also be composed of the other objects; such geo-objects are called complex objects. The geo-object "Federal Republic of Germany", for example, may be described as being the set of the states North Rhine Westfalia, Lower Saxony, Bavaria, etc.

The model provides three ways of composing objects of others: A complex object may be a set of objects of identical type (no ordering), it may be a list of objects of identical type (ordered), or it may be a tuple (or aggregation) of objects of different classes.

Moreover, the model allows classes of geo-objects to be structured into hierarchies by defining generalizations (in the computer science sense) and partitions. Thus, the classes "River", Canal" and "Lake", for instance, might be united to "Inland Waters", which in its turn might be a sub-class of "Waters".

In addition to the concepts of composing complex objects and defining generalizations and partitions, there is another way of expressing a linkage between objects of different classes, namely by defining a relationship class for arbitrary geo-object classes; accordingly, relationships can be established between objects of these classes. As an example, a relationship class "lies on" might be defined for the object classes "City" and "River", and an instance of this relationship might then be inserted to express the fact that the city of Cologne lies on the river Rhine.

2.3 Graphic Representation

For graphic representation of stored information about geo-objects, our model offers predefined concepts for the construction of maps.

Maps are treated as objects of their own within the geo-object model. As such, they can be created, deleted, updated, retrieved, etc.; furthermore, they can be output, i.e. plotted, displayed on the screen or be written onto a file to enable further processing by a cartographic editor.

Every map consists of a map face, which carries the actual information to be shown, and a map margin, which contains the legend with explanation of signs, scale, etc.

Both map face and map margin are composed of map objects. Map objects are the graphic elements which represent geo-objects in maps. They are divided into several predefined map object classes, some geometric and some non-geometric.

The shape and location of a geo-object, given by the value of its geometric attribute (data types: POINTS, LINES, POLYGONS), is represented by a corresponding geometric map object. The geometry of the map object may, however, be chosen to be different from the geometry of the original geo-object; thus, it is not only possible to represent, for instance, a geo-object of the class "City" (geometry: POLYGONS) by a map object of the geometric class POLYGONS (e.g. a red coloured polygon), but as well by its border line (geometric class LINES) or its center of gravity (geometric class POINTS).

In addition to the attribute GEOMETRY, which describes the shape and location of a map object within the map sheet, there are other attributes which specify its graphic representation, i.e. its colour, pattern, etc.

The non-geometric class TEXT is provided to represent any alphanumeric information, like, for example, the names of geo-objects. Other non-geometric classes are PIE_CHART and BAR_CHART, which permit the suggestive representation of thematic information. The non-geometric classes also have various attributes concerning their location, their graphic representation, etc.

Map objects cannot exist on their own, but only as components of a certain map; this association between a map object and the corresponding map should be unique and fixed. Since updates of map objects often depend on their local environment within a certain map (consider, for example, the placement of letterings), undesirable side effects in other maps can be avoided by this rule.

3. Data Types

Within the database, both geo-objects and map objects are represented by data, namely by their attribute values. In the following, we present a survey of the underlying set of permissable data types and their respective operations.

To begin with, there are, of course, the standard alphanumeric data types - INTEGER, REAL, STRING, BOOL, etc. - with their usual operations.

In the second place, our model provides the temporal data types DATE and INTERVAL. These data types permit storing different versions of a geo-object or of a

map. If you consider a certain district, for instance, whose boundaries have been changed on April 1, 1963, and on January 1, 1984, you might store three versions of this geo-object, the respective version attributes being the intervals [-, 63/03/31], [63/04/01, 83/12/31] and [84/01/01, -].

There are operations for
- creating an interval of two DATE values,
- determining the beginning and the end of an interval,
- computing the length of an interval,
- comparing intervals (IN, BEFORE, AFTER, AT, EQUAL),
- intersecting intervals, and
- determining the first or the last interval from a given set of intervals.

In the third place, our model provides geometric data types, such as POINTS, LINES, POLYGONS, RASTER, SPACE_POINTS, WINDOW, etc. A single value of type POINTS, LINES, or POLYGONS represents a set of points, lines, or polygons, respectively. If you consider, for instance, the Federal Republic of Germany as a single geo-object, its geometry - including West Berlin and the Frisian Islands - can be treated as a single value of type POLYGONS.

The geometric data types are equiped with a variety of operations:

- Firstly, there are operations for relationship tests; they take geometries as their parameters and deliver a boolean result.

  Within this group, the operations CUT, TOUCH, IN and EQUAL are provided.

  Most of these operations are polymorphic, which means that various combinations of data types are admissible for their parameters; the operation CUT, for example, which checks if two geometries have a non-void intersection, may be used to compare two LINES values, to com pare two POLYGONS values or to compare a LINES value with a POLYGONS value.
- Secondly, there are operations for geometric calculation; they take geometries as their parameters and deliver geometric results.
  Within this group, there are operations for generating geometries, like creating a point, creating a line from a list of points, etc., and there are powerful operations such as UNION and INTERSECTION, BORDER, SURROUNDING, CENTRE_OF_GRAVITY, etc.
  Most of these operations are polymorphic, too; INTERSECTION, for instance, may be used to compute the intersection of a pair of POINTS values, of a pair of LINES values, of a pair of POLYGONS values, of a POINTS value and a LINES value, or of a LINES value and a POLYGONS value; BORDER may be used to compute the end points of lines or the border lines of polygons; SURROUNDING may be used to compute the smallest surrounding rectangle of a POINTS, LINES, POLYGONS or RASTER value; CENTRE_OF_GRAVITY may be used to compute the centre of gravity of a LINES value or of a POLYGONS value, etc.
- Thirdly, there are operations for numeric calculation; they take geometries as their parameters and deliver numeric results. Within this group, there are operations for computing the LENGTH of a LINES value, the AREA of a POLYGONS value, the DISTANCE between points or lines, the CARDINALITY of POINTS, LINES

and POLYGONS values (i.e. the number of single points in the POINTS value, ...) etc.

4. Database Language

4.1 Survey

The database language integrates the concepts of the geo-object model (described in chapter 2) and the alphanumeric, temporal and geometric data types (described in chapter 3).

Thus, it enables the user

1. to model his view of a geoscientific world by defining and dropping geo-object classes and relationship classes (data definition, cf. section 4.2),
2. to handle the masses of data belonging to his geoscientific world by issuing powerful, set-oriented CREATE, UPDATE, RETRIEVE, DELETE and DISPLAY commands (data manipulation, cf. section 4.3), and
3. to arrange stored information in maps for graphic representation by creating, updating and deleting map objects and maps (map construction, cf. section 4.4).

Database languages which integrate a variety of different concepts tend to become rather tortuous; we avoided this, however, by designing a language that combines descriptive elements with procedural ones, such that complicated queries are broken up into intelligible portions. In this, our language resembles the relational database language QUEL (STONEBRAKER et al. 1976).

Basic queries can be formulated using predicates for qualification; arbitrary qualification formulas can be expressed in a single command unless explicit quantifiers are needed. More complicated sets of desired objects are constructed iteratively by alternating descriptive queries and set operations such as union or difference; the language allows to store the intermediate results in temporary object classes and referring to them by object variables.

Being based on an object calculus, our database language is well founded semantically. Like QUEL, it is complete (DATE 1981, ULLMAN 1982), if the notion of relational completeness is adapted to ER based models in the sense of PARENT & SPACCAPIETRA (1984).

In the next section we present an outline of the database language by means of elementary examples which illustrate most of its features; this description essentially follows the introduction to the language given by LIPECK & NEUMANN (1986).

4.2 Data Definition

Geo-object classes, relationship classes and partitions are defined by means of a DEFINE statement; they can be removed from the schema using a DROP command.

A geo-object class "City", for example, might be defined as follows:

```
DEFINE_GEOOBJ City
  (NAME          STRING (20),
  GEOMETRY       POINTS,
  population     INTEGER);
```

Note that system-defined key words are written in capital letters in contrast to other identifiers chosen by the user.

To define a class of complex objects, the type of composition has to be specified in addition to the attributes and their data types. This is done by the key words SET_OF, LIST_OF or TUPLE for sets, lists, or aggregations.

If we assume that an object class "Single_State" has already been defined, we might define a complex object class "State_Union" as follows:

```
DEFINE_COMPOBJ State_Union
  (NAME          STRING (25),
  GEOMETRY       POLYGONS,
  VERSION        INTERVAL,
  population     INTEGER,
  regime         STRING (30),
  federative     BOOL,
  members        LIST_OF Single_State);
```

To establish generalizations of some object classes to more abstract classes, the DEFINE_PART (define partition) command is used, for example:

```
DEFINE_PART State (State_Union, Single_State)
  (NAME,
  GEOMETRY,
  VERSION,
  population,
  regime);
```

Note that the "superclass" which is defined in the DEFINE_PART statement may only have those attributes which are common to all subclasses.

Relationships between object classes are defined by the DEFINE_REL statement, for example:

```
DEFINE_REL Capital (State, City);
```

4.3 Data Manipulation

After a database schema has been defined, the user may insert objects using the CREATE command (or by bulk loading). In the following example only string and version constants are filled in; the dots denote further missing constants such as geographic co-ordinates:

```
CREATE_COMPOBJ State_Union
  ('Germany', (...), [49/05/24, -], ...);
```

To illustrate the usage of the UPDATE command, we will now present some statements related to the foundation of the member state Baden-Württemberg, which was created by uniting the former member states Württemberg-Baden, Württemberg-Hohenzollern and Baden.

To begin with, a respective geo-object has to be created:

```
CREATE_GEOOBJ Single_State
  ('Baden-Württemberg', , [52/04/25, -], ...);
```

After that, the UPDATE command is used to fill in those attribute values which have not yet been specified.

Note that in RETRIEVE, UPDATE and DELETE commands the objects of all classes are referred to by typified object variables. These variables are declared with a RANGE_OF statement, which establishes a connection between a variable and an object class.

```
RANGE_OF s IS Single_State; /* Baden-Württemberg */
RANGE_OF sl IS Single_State; /* Württemberg-Baden */
RANGE_OF s2 IS Single_State; /* Württemberg-Hohenzollern */
RANGE_OF s3 IS Single_State; /* Baden */

UPDATE_GEOOBJ s
    SET s.GEOMETRY = UNION (s1. GEOMETRY,
                            UNION (s2.GEOMETRY, s3.GEOMETRY)),
        s.population = sl.population + s2.population + s3.population
    WHERE s.NAME = 'Baden-Württemberg'
        AND sl.NAME = 'Württemberg-Baden'
        AND s2.NAME = 'Württemberg-Hohenzollern'
        AND s3.NAME = 'Baden';
```

Subsequently, the single state Baden-Württemberg is added to the state union Germany using another form of the UPDATE command; it is assumed that the list of member states of Germany is sorted by areas, and that Baden-Württemberg is to be added at its appropriate position:

```
RANGE_OF u IS State_Union; /* Germany */
RANGE_OF i IS INDEX;       /* index variable */

UPDATE_COMPOBJ u
    INSERT members.i s
    WHERE u.NAME = 'Germany'
      AND s.NAME = 'Baden-Württemberg'
      AND AREA (u.members.i-1.GEOMETRY) <= AREA (s.GEOMETRY)
      AND AREA (u.members.i.GEOMETRY) > AREA (s.GEOMETRY);
```

Finally, the former member states are deleted, and the VERSION attribute of Germany is updated:

```
DELETE_GEOOBJ s
  WHERE s.NAME = 'Württemberg-Baden'
       OR s.NAME = 'Württemberg-Hohenzollern'
       OR s.NAME = 'Baden';

UPDATE_COMPOBJ u
   SET u.VERSION = s.VERSION
  WHERE u.NAME = 'Germany'
     AND s.NAME = 'Baden-Württemberg';
```

To illustrate the usage of the RETRIEVE coammand, we will now present the statements necessary to retrieve all cities in the Federal Republic of Germany except those which are capitals of member states. This task is done in three steps.

First, all cities in the Federal Republic of Germany. i.e. all city names, are retrieved:

```
RANGE_OF u IS State Union LAST;
RANGE_OF c IS City IN_WINDOW (06, 47, 14, 55); /* Germany */
RANGE_OF i IS INDEX;

RETRIEVE_OBJ_INTO result (city_name = c.name)
  WHERE WITHIN (c.GEOMETRY, u.GEOMETRY)
  AND u.NAME = 'Germany';
```

The RETRIEVE command generates a temporary object class. The objects to be retrieved are qualified by a predicate in the WHERE clause, their attribute names and associated values are specified in the so-called target list.

Note that the specification LAST in the RANGE_OF statement indicates that the variable u ranges over the last versions of state unions only; the specification IN WINDOW indicates that the varibale c ranges over the specified geometric window only. Both restrictions of range may simplify the query and may accelerate storage accesses.

In the second step, all capitals of member states are retrieved:

```
RETRIEVE_OBJ_INTO cap_of_state
  (city_name = c.NAME,
  GEOMETRY = c.GEOMETRY)
WHERE capital (u.members.i, c)
AND u.NAME = 'Germany';
```

In a last step, the desired set of objects is constructed, which is the set difference between result and cap_of_state; the resulting set of cities is to be displayed on the terminal:

```
RANGE_OF res IS result;
RANGE_OF cos IS cap_of_state;

DELETE GEOOBJ res
  WHERE res.city_name = cos.city_name;
DISPLAY res ON TERMINAL;
```

Note that the set difference is determined by the specified DELETE statement.

4.4 Map Construction

Above, we have introduced statements for handling user defined geoobject classes like City, State, etc. In the following, we present commands for arranging stored information in maps; these commands are associated with the system defined classes of maps and mapobjects.

Suppose the user wants to construct a map which contains all member states of the Federal Republic of Germany (displayed as green bordered polygons) together with their capitals (represented by smal red circles); both member states and capitals are to be lettered with their names. To this end, an (empty) map has to be created first:

```
CREATE_MAP ('Germany_States', 3000000, ...);
```

The name of the map is 'Germany_States', 3000000 means the scale. All other attributes are left out here. Now the version of the new map is set to the version of the latest State_Union object:

```
RANGE_OF u IS State_Union LAST;
RANGE_OF m IS MAP LAST;

UPDATE MAP m
   SET m.VERSION = u.VERSION
 WHERE m.NAME = 'Germany States'
     AND u.NAME = 'Germany';
```

Subsequently, all the desired mapobjects are generated: the green bordered polygons (mapobject class: POLYGONS), the small red circles (mapobject class: POINTS), and the letterings (mapobject class: TEXT). The creation of mapobjects and their insertion into the face of the specified map are performed simultaneously. Note that a transformation from world co-ordinates into sheet co-ordinates is performed implicitly whenever mapobjects of a geometric class are generated.

```
RANGE_OF s IS Single_State LAST;

CREATE_MAPOBJ_FROM s
  INSERT_INTO m.FACE.LAST
  (POLYGONS: TYPE = HOLLOW;
        COLOUR = 'green')
  WHERE s IN u.members
    AND u.NAME = 'Germany'
    AND m.NAME = 'Germany_States';

CREATE_MAPOBJ_FROM s
  INSERT_INTO m.FACE.LAST
  (TEXT: CONTENTS = s.NAME,
        COLOUR = 'black',
        POSITION = CENTRE (s.GEOMETRY))
  WHERE s IN u.members
    AND u.NAME = 'Germany'
    AND m.NAME = 'Germany_States';

RANGE_OF cos IS cap_of_state;

CREATE_MAPOBJ_FROM cos
  INSERT_INTO m.FACE.LAST
  (POINTS: MARKER = CIRCLE,
        COLOUR = 'red')
  WHERE m.NAME = 'Germany_States';

CREATE_MAPOBJ_FROM cos
  INSERT_INTO m.FACE.LAST
  (TEXT: CONTENTS = cos.city_name,
        COLOUR = 'black',
        POSITION = cos.GEOMETRY)
  WHERE m.NAME = 'Germany_States';
```

Finally, the map may be stored on a file (named, for instance, 'Germany_Map') in order to be sent to a graphics display or editor afterwards:

```
RETRIEVE_MAP_INTO m_out (m.*)
  WHERE m.NAME = 'Germany_States';
DRAW m_out ON 'Germany_Map';
```

After adding the title 'Bundesrepublik Deutschland' and making some minor changes in the placement of letterings, the resulting map is the one shown in Fig. 2 (unfortunately without colours for reasons of reproduction).

Fig. 2: Map of Germany.

5. Current and Future Activities

This paper has introduced the database language and the underlying modelling concepts and data types for the user interface of a geo-DBMS.

Meanwhile, a first partial prototype has already been implemented (ERNESTI 1986, STAHS 1986, WARNEBEOLD 1986), which bases on the relational database machine IDM 500. Programming was done in C on a VAX 750 running with ULTRIX. Graphic output is based on the Graphic Kernel_System (GKS) (ENDERLE, KANSY & PFAFF 1984). The internal structure of the prototype is shown in Fig. 3.

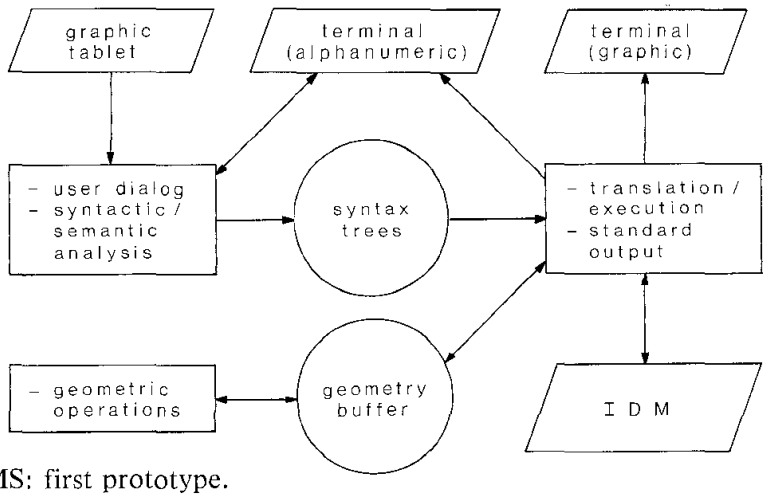

Fig. 3: Geo-DBMS: first prototype.

As the geo-DBMS is to be re-implemented on top of the geokernel (cf. chapter 1) in a later phase of the project, this first prototype is meant to be provisional; hence, only a subset of the projected language has been implemented. Nevertheless, the first prototype already permits modelling simple geoscientific worlds, handling 2-dimensional geometric data and displaying the stored information in maps. The map of Germany shown in Fig. 2 has been constructed solely by using this first prototype.

As our geo-DBMS only provides basic concepts for the construction of maps, we are currently working on an interface to cartographic editors (cf. Fig. 1); this means that a map may be written onto a file by a DRAW_MAP command, be subjected to arbitrary cartographic processing and subsequently be rewritten into the database by a RESTORE_MAP command.

In the following, we give a brief survey of the activities we are planning for the future:

- Development of a few prototype applications, like the one sketched by RAMM & NEUMANN (1986),
- embedding the database language into a host language to enable application programs to communicate directly with the database,
- extending the database language by concepts for defining and handling arbitrary complex objects,
- extending the database language by concepts for defining additional data types (e.g. 3-dimensional geometries) and their respective operations by the skilled user.

Furthermore, the geo-DBMS will have to be re-implemented on top of the geokernel once the implementation ot the kernel has been finished. This second prototype will provide the full scope of the database language, including user-definable data types. The response times - which are not yet satisfactory in the first prototype - will improve con-

siderably as the kernel will provide for efficiency in the storing, retrieving and updating of information.

A subject of further investigation is to make the construction of thematic maps more comfortable; this can be done by the definition of rules which represent a subset of cartographic knowledge. A system for translating such rules into the database language has been sketched by DRAWIN, NEUMANN & EHRICH (1987).

Acknowledgement: We thank our colleague Udo W. LIPECK for his important contributions to the project.

6. References

CHEN, P.P. (1976): The Entity-Relationship Model - Toward a Unified View of Data. - ACM ToDS **1:** 9-36; New York.

DATE, C.J. (1981): An Introduction to Database Systems. - 3rd Ed. Reading (Mass) (Addison-Wesley).

DRAWIN, M., NEUMANN, K. & EHRICH, H.-D. (1987): Regelorientierte Erzeugung von Karten-Entwürfen auf geowissenschaftlichen Datenbanken. - GI-Fachtagung „Datenbanksysteme für Büro, Technik und Wissenschaft"; Darmstadt.

DEPPISCH, U., OBERMEIT, V., PAUL, H.-B., SCHEK, H.-J., SCHOLL, M. & WEIKUM, G. (1985): The Storage Subsystem of a Database Kernel System. - Techn. Rep. TH Darmstadt, DSVI-1985-Tl.; Darmstadt.

ENDERLE, G., KANSY, K. & PFAFF, G. (1984): Computer Graphics Programming, GKS - The Graphics Standard. - Berlin (Springer).

ERNESTI, B. (1986): Realisierung eines Geo-Datenbank-Prototyps: Der Geometrieprozessor mit einfacher graphischer Ausgabe. - Dipl.-Arb. TU Braunschweig; Braunschweig.

LIPECK, U.W. & NEUMANN, K. (1986): Modelling and Manipulating Objects in Geoscientific Databases. - Proc. 5th int. Conf. on the Entity-Relationship Approach; Dijon.

PARENT, C. & SPACCAPIETRA, S. (1984): An Entity-Relationship Algebra. - Proc. IEEE Conf. on Data Engineering: 500-509; New York.

RAMM, I. & NEUMANN, K. (1986): Anwendungen auf Geo-Datenbanken: Ein Beispiel. - Int. Ber. TU Braunschweig; Braunschweig.

–, LIPECK, U.W. & EHRICH, H.-D. (1985): Eine Benutzerschnittstelle für geowissenschaftliche Datenbanken. - Informatik-Ber. TU Braunschweig, **85-06;** Braunschweig.

SCHEK, H.-J. (1986): Datenbanksysteme für die Verwaltung geometrischer Objekte. - In: HOMMEL, G. & SCHINDLER, S. (Hrsg.): Proc. 16. Jahrestag. GI **I:** 483-479; Berlin.

- & WATERFELD, W. (1986): A Database Kernel System for Geoscientific Applications. - Proc.int.Symp.Spat.Data Handling, Seattle: 273-288; Williamsville.

STAHS, T. (1986): Realisierung eines Geo-Datenbank-Prototyps: Der Übersetzer. - Dipl.-Arb. TU Braunschweig; Braunschweig.

STONEBRAKER, M., WONG, E., KREPS, P. & HELD, G. (1976): The Design and Implementation of INGRES. - ACM ToDS, **1:** 198-222; New York.

ULLMANN, J.D. (1982): Principles of Database Systems. - 2nd Ed., Rockville (Md.) (Computer Sci. Press).

VINKEN, R. (1985): Digitale Geowissenschaftliche Kartenwerke - ein neues Schwerpunktprogramm der Deutschen Forschungsgemeinschaft. - Nachr. aus dem Karten- und Vermessungswesen, **I, 95:** 163-173; Frankfurt a.M.

WARNEBOLD, P. (1986): Realisierung eines Geo-Datenbank-Prototyps: Der Benutzerdialog mit syntaktischer und semantischer Analyse. - Dipl.-Arb. TU Braunschweig, Braunschweig.