Department of Civil Engineering

SURVEYING ENGINEERING

REQUIREMENTS FOR DATABASE SYSTEMS SUITABLE TO MANAGE LARGE SPATIAL DATABASES

Andrew U. Frank

Report No. 39

This paper was presented to the International Symposium on Spatial Data Handling, held in Zurich (Switzerland) August 20-24, 1984. It appeared in the proceedings, edited by -

University of Maine at Orono
103 Boardman Hall
Orono, Maine 04469

REQUIREMENTS FOR DATABASE SYSTEMS SUITABLE TO MANAGE LARGE SPATIAL DATABASES

Dr. Andrew U. Frank
University of Maine
103 Boardman Hall
Orono, Maine 04469
U. S. A.

1. Introduction

Information systems dealing with data related to location in real world space - here referred to as spatial data - are of increasing interest today. Many operations of government of all levels as well as planning and research exploit data about facts in relation to space. Such systems are found under different headings e.g. geographic information system, land information system, multi-purpose cadastre etc. This paper concentrates on general aspects of systems dealing with spatial data referred to as "spatial information systems", without consideration for differences between systems for differing purposes; an attempt for a taxonomy is presented elsewhere (Frank, 1980). This paper concentrates on systems which store data with an exact reference to location and describes geometry using points and vectors. We believe there are applications for such vector systems, especially if one considers the storage necessary for raster based systems. This is not to exclude systems of other types - there are obvious advantages in raster operations for certain tasks - but they seem to be substantially different and warrant a separate discussion.

Spatial information systems typically need large data collections, stored in a permanent manner on mass storage devices (disk) and accessing only small parts of these collections for execution of a simple operation. Early in the history of data processing users became aware of the similarity in their needs to store data and retrieve them later for processing. In lieu of writing procedures for these functions for each application anew, it was attempted to write a generally applicable program to provide these services. The idea of (generalized) database management system was born (CODASYL, 1962, 1971).

These efforts came from the commercial data processing world and were consequently oriented towards systems treating large commercial databases. In the past research efforts have been limited to commercial data collections and only in the newest literature discussions of "non-standard" database applications, like bibliographic databases, databases for Computer Aided Design, Geographic, and Land Information Systems, have emerged (Härder and Reuter, 1982) (Scheck and Lum, 1983). On the one hand, research in the area of spatial information systems has shown that functionality similar to standard database management systems is required. It seems that techniques used in commercial database management could be beneficial (Frank and Tamminen, 1982). On the other hand, researchers in database theory became interested in learning about requirements for non-standard applications. Non-standard applications raise other demands for the data base functions, and this paper will investigate into the specific re-

quirements for large spatial database.

It becomes evident that some of the decisions typically embodied in a database management system depend on the intended application area, and systems designed for standard commercial applications are thus not automatically suitable for non-standard applications. One conclusion from the discussion of two recent international panels on non-standard databases is the necessity to construct database management systems as toolboxes from which an application can select building blocks precisely suited for its needs. This not only to assemble the functions the application needs, but also to select suitable implementations of these functions, and finally to exclude unnecessary functions which only add to the complexity and the cost of the application.

As a framework for insuring compatibility between the building blocks, we use a hierarchically layered model, similar to the widely accepted model describing communication between computers (ISO/TC97/SC16).

This paper is the result of several years' work dedicated to applying the database concept to spatial data handling at the Swiss Federal Institute of Technology Zurich and now at the University of Maine at Orono. The ideas reported here are based on experience with the PANDA database management system we built (Frank, 1982a), and reflect either methods successfully implemented, or are a critic of methods which did not work and will be replaced in the future.

2. Reasons for using database management systems

The generally observed trend away from batch processing towards interactive, dialogue oriented applications changes the requirements for data storage: Many application programs access parts of the same data files in an unpredictable order. Users increasingly demand immediate access to large data collections.

Data collected in a database are valuable as much effort is necessary to collect and insert the data in the system and keep the data updated. These data must be available for a long period of time to justify the expenses, and obviously new, not foreseen changes in the application will occur during the lifetime of the data.

Under these circumstances the traditional simple file structure designed to facilitate a special application can not do anymore. Storage structures for multiple access, however, are complicated to program, and it seems inappropriate to repeat this effort over and over again.

A database management system should provide the following functionalities:

- Storage and retrieval of data; selection of data using different and varying key fields.
- Standardizing access to data and separating the data storage and retrieval functions from the programs using the

data. This makes database and application programs independent of each other, and changes in the one do not necessarily lead to changes in the other. This independence is important in order to accomodate the changes during the lifetime of a database.

- Interface between database and application programs should be based on a logical description of the data, and not make any detail of the physical storage structure visible to the applications.

- Making access functions in applications independent of the physical storage structure, so adaptations to expanding storage needs do not influence the application programs.

- Allowing for access to the data by several users at a time.

- Providing for the definition of consistency contraints for the data which will then be automatically enforced. Consistency constraints are rules which must hold for all data stored, and are an excellent technique to reduce the number of errors in a large data collection.

For a slightly different, more elaborate list see (Codd, 1982).

Access to data should be possible both from a high level language and from a userfriendly query language. The intergration of the database manipulation commands in the high level programming language is crucial to ease of use and can help to avoid hard-to-track bugs. A free-standing query language is helpful for casual users to retrieve data from the database to answer ad-hoc questions without any formal programming; this makes the database usable for one-of-a kind question.

All these basic requirements are valid for commercial data processing as well as for spatial data collections. However, the details of the requirements, especially for treatment of geometric objects and for production of graphical output are new to the database world.

3. Method

The framework for this study is a hierarchy of modules, each providing certain types of services or functions to the next upper layer. The lowest layer is directly related to the services provided by the operating system whereas the top layer provides services to the human user.

The bottom layers store data using the operating system for accessing the file system; these layers are mainly concerned with improving the performance of access to data by clustering techniques and buffer management. Services offered are 'store' and 'retrieve' operations for data elements (records) using internal keys.

The next layers will provide essentially the same operations, only making them secure. Transaction management defines at the

end of a successful transaction a point where consistency constraints are checked. Changes in the database are guaranteed against loss or interferences with other users.

The third layer adds different types of access methods, e.g. access to data based on a key value or spatial location.

The fourth layer offers a logical structuring tool for the data and manipulations based on this logical schema. These services are then offered as extension to a high level programming language or an independent query language.

4. Assumptions

This paper is - like most papers - based on a series of assumptions. Some of them should be made explicite in order to prevent confusion of the reader.

4.1 Technology

The discussion in this paper is based on today's technology which may be more advanced than found in some installations. However, we exclude all experimental hardware.

Processor: We assume a VonNeumann hardware architecture and exclude future parallel multiprocessor hardware. The extent to which this can be beneficial to database operations seems to be in dispute (deWitt and Hawthorn, 1981).

Storage media: We assume use of main storage (formerly "core") directly accessible by the processor, and separated slower mass storage. Data on mass storage is only accessible after transfer into main storage. Access to data on main storage is much slower than to data stored in main memory (access times in the range of 10,000:1), and access is in larger chunks, with access time essentially constant and independent of the amount of data accessed.

Distribution of processing: Data stored on a central computer may be transferred for use in an 'intelligent' workstation having its own processor and program. We will exclude from the discussion the distribution of the storage of data on several cooperative computers. It seems that the basic problems of such arrangement have not yet found sufficiently general solution (Lampson et al, 1981).

Software technology: The considerations discussed in this paper are based on current programming techniques, using modern transportable, high level language.

Our implementation is all done using Pascal and a precompiler to facilitate modular programming and providing transportability of code to different hardware (IBM 370 VM/CMS, DECsystem-10, PERQ).

4.2 Application area

Size of the data collection: A spatial information system usually contains a large collection of data. Specifically, we assume that the size of the data collection is too large to fit in the main memory and must be kept on a mass storage device.

Number of different data types: It is assumed that the data consist only of a small number (<1000) of different data types, of which a great many of instances are stored.

Many applications: We further assume that the data stored will be used for a number of different applications.

These three assumptions are typical for the application of database software. In other cases simpler, cheaper or better adapted solutions are sometimes available.

Spatial data: A spatial information system typically stores data with known relations to space, and the treatment of data makes use of these spatial relations. The position in space is expressed as coordinate values in a given coordinate system.

Geometric data in vector form: Part of the data in spatial information systems describe geometric form and extension of objects in space, and the processing often explores geometric properties, and the rendering of results in form of maps is common. Geometric forms are expressed using vectors between known points.

Interactive retrieval for map presentation: In order to draw a map on an interactive terminal screen, a large number of data elements (records) must be retrieved. Estimates from counting point densities on produced outputs are over 2000 records, mainly point and line records, but also many different records describing the visible units (e.g. house, village and road descriptions). Retrieval of all data necessary for a map should be completed within 20 seconds, preferably faster to allow for truly interactive work.

5. Requirements for the Data Storage Layer

This layer interfaces with the actual storage operations provided by the operating system. Its main purpose is to improve speed of storage and retrieval. The requirements for this layer result from the maximum delay we allow for the drawing of a map on a screen. Some tests show that about 2000 data records are necessary for a drawing: in order to retrieve them in 20 seconds, average access time for a record must be less than 10 msec. Access times to physical storage on disk including overhead of operating systems are in the order of 50 to 200 msec. Therefore, this layer must reduce the number of physical accesses necessary to retrieve the records for the map. Two basic techniques are known:

- clustering of data
- buffer management

It is desirable that the interface to the operating system

be simple and only use standard functions found on many operation systems (read and write to random access files). This will make transporting the database management system - and the application using it - much simpler. However, it must be noted that some of the methods used in operating systems to improve performance of disk operations are detrimental to database access times, and it is advantageous if low level direct disk access operations can be used.

5.1 Physical Clustering

A physical access to the disk brings in a larger chunk of data, usually called a disk page. If we can arrange our records on disk pages in a way that each page contains several data records necessary for the map drawing, this scheme will reduce the number of physical accesses to disk pages by the average number of useful records per page.

This is feasible in all cases where a reasonable prediction of what data will be used together is possible. These data are then stored together and form a physical cluster (Salton and Wong, 1978). Fortunately for spatial retrieval for map drawings such predictions are easy, based on vicinity of objects. For a map we retrieve data from objects situated within a certain area. If we retrieve a data element of an object, chances are good that data from other objects in the vicinity are needed next. If such data are clustered together and retrieved with one physical access, we can achieve the goal of retrieving a map within a short time span, permitting interactive work.

A database system for spatial data must at least provide for physical clustering of data (typically a command like 'STORE NEAR x" where x is an already stored record). This requirement excludes many of the simpler relational database implementations, where the physical storage is directed by the primary key and can not be influenced by the user. It seems also advantageous, if data from different types (e.g. houses, roads and rivers) can be stored on the same page and do not necessarily require different physical accesses.

However, it is neither necessary nor desirable that physical clustering be visible on the level of the user interface; it should rather be used internally for fast spatial access and be of no concern to the user.

5.2 Buffering

Many programs show a locality of access, using the same data elements repeatedly over a short period of time. If these data elements can be kept in a buffer, the number of physical accesses to a slower mass storage device can be reduced: for each data element only one access to the slower device is necessary, and all following requests are satisfied using the buffered data. This strategy is generally employed in computers (virtual memory, cache etc.).

A database usually contains a single level of buffering for pages brought in from the disk. This is indispensable in order

to exploit a physical clustering of data on the mass storage device.

Programs dealing with spatial data typically show much locality of access to data but use a large working set (all the records for a map drawing). This is especially important for interactive programs. Users tend to work in an area for several interactions before they request another base map. Very often they ask for an overview map first, zoom then in on a detail of interest and start work on this.

In our experimental database management system PANDA we included a second level of buffering for single data elements. We currently set the size of this buffer to 3000 records, which allows us to keep a complete map drawing in buffer. Redrawing of a map does not require any additional physical accesses and is in consequence very fast.

We found that this concept has considerable influence on the programming style of an application: no attempts to keep database data stored in program variables are made since a second access to data is always very fast. For example, we do not use linear display list but produce graphics directly from the database records.

A database management should include utilities to determine optimal buffer sizes, according to a task.

6. Requirements for data security

The services offered in commercial database management systems are generally sufficient. However, many of the sytems for use on smaller micro-and personal computers, as well as the systems specialized in geometric and geographic data, do very often not provide the functionality necessary to manage large data collections over long periods of use.

The layer in this paragraph will not add substantial new functionality but increase security of operations. Most of the functions in this layer will reduce performance - that is the price we have to pay for the security gained ("there ain't such a thing as a free lunch!").

6.1 Transactions

The contents of a database must fulfill some constraints, so called integrity constraints which help to guarantee long term usability of data by detecting clearly erroneous data before they are stored. To this end, a user introducing changes into a database starts a transaction and then changes the data records affected by these updates. All these changes are of a tentative nature until the user finally confirms them. Then the system checks that the new or changed data do not violate the integrity constraints. Only after these checks the changes are posted to the data collection as a permanent update. Each transaction leads the database from an initial (consistent) state to a new, again consistent, state; in this manner it is virtually impossible that the database becomes inconsistent. The transaction concept

is also very convenient for the design of interactive update procedures - the user can do all his intended changes and still have the option to abandon the changes completely.

6.2 Concurrent users

Large data collections are powerful but costly resources which must be put to their best use. In many cases more than one user must be able to access the database at a single moment in time. This is not very difficult as long as all users only retrieve data, but it becomes more dangerous if one or more users need to introduce changes in the database. If no special precautions are taken, a change from one user may wipe out a previously introduced change by another user or changes from different users may interfere and produce strange and unforeseeable results. It is insufficient to rely on manual procedures which are not integrated into the database management system as they are currently recommended for some graphical data management systems. Automatic control of concurrent use of a database is based on the transaction concept: the effects of a change are made visible to other users only at the end of the transaction. Concurrent users never see the effect of an incomplete transaction and interference between two concurrent users is resolved at the transaction level (Kung and Robinson, 1981).

6.3 Security against loss of data

Large collections of geographic or administrative data are valuable assets and must be guarded against loss. Even if the information is parallelly maintained in other, traditional forms (registers, maps, etc), the cost of transferring this information into machine readable form is considerable. In order to avoid such costs, it is necessary to protect the stored data from loss due to errors in operating or malfunctioning of hard-or software.

In order to determine the appropriate measures, we must access possible damages from loss of data (cost of reentering data, cost associated with interruption of operations etc.) and balance them against cost of operations to prevent such losses. It is generally assumed that most commercial operations place more importance on uninterrupted operations and, in consequence, very involved but secure schemes have been devised. In our area of application, lower levels of security may be acceptable, but as a minimum requirement a mechanism to prevent loss of data must be in place.

In all cases where more than just infrequent changes are made, the mechanism should include the updates of the database. A change the user has committed and the confirmation received from the database must not be lost any time later, independent of any interruption by hard-or software problems. The transaction serves here as the unit of recovery: uncommitted changes may get lost but committed and confirmed ones are permanent.

It is customary to distinguish two types of problems:

· Interruption of the database management program due to operating errors, or problems in the operating system or the hardware. Such interruptions occur on most installations quite frequently (one per day .. one per week). During such events all contents of main memory are lost and it is therefore necessary to write changed data to permanent mass storage before confirmation of a transaction.

· Loss of the storage media, again due to errors in operations or hardware defects (the so called "head crashes"). Such problems are usually very seldom (once a year) and slower procedures for recovering are acceptable. Traditional data processing used the "two generation" system, keeping the copies of data on magnetic tape for at least two previous defined states and for all updates in between. A similar method is applicable to database management: the database is copied to magnetic tapes in regular intervals and all changes between the copies are not only used to change the database but also written to an additional log file (in fact they should be written to the log file first).

6.4 Protection against unauthorized users

Some databases will contain confidential or secret information, and it is necessary to protect it against unauthorized users - more often against reading but also against changing the data. The requirements will vary greatly:

- keeping unauthorized users from accessing the database, a service the operating system should provide;
- making only certain parts of the database accessible to certain users - different methods are known and used, providing varying levels of selectivity in access and security.
- restricting the user to generalized information only and preventing direct access to individual data - most of the known systems have been proven insufficient, and the proposed methods are extremely involved (Denning and Schlorer, 1980).

7. Access methods

The layers discussed so far provide storage and retrieval of data using an internal identifier (database key) which is assigned to a data record when it is stored first. The record itself is considered by these methods as an uninterpreted fixed amount of bytes which are stored and later reproduced. For non-standard applications it is especially important that the storage system does not depend on interpretation of the data stored because this would limit the flexibility of arranging information in a record and produce undesirable dependencies between the different layers of the system. The database subsystem of a larger application must handle arbitrarily structured data records without dependency on their internal structure. The often provided records of one hierarchical layer composed of the basic data types, i.e. integers, reals, and strings, are not even sufficient for commercial applications, as examplified by numerous extensions offered (datatypes for money, date etc.). The access methods discussed in this section use parts of the data stored in a record; in order to achieve independence of the data description, the defining modules must export access functions to the relevant

data elements which are then used for the access methods.

It may be pointed out that we describe here logically defined methods for accessing data. An implementation must include algorithms and storage organizations in order to realize these methods and achieve fast responses.

7.1 Access by unique key

Users often need access to data records based on the value of one or several data fields (key fields). This requires in the simplest case that the stored values for these fields be unique, so that given values unequivocally identify the desired record. (This will be a consistency constraint of the database). It is generally necessary to be able to define more than one key for a record, so the record can be found using either value (e.g. number or name).

7.2 Access in hierarchies

Very often values identify an entity within one area, and the same value can be found in another area again. This typically happens in a hierarchy such as state, county and town. To fully identify an entity, all values must be given and only their combination must be unique.

7.3 Generalized access

We found that human users very often use names that are 'nearly' unique: names of towns or names of clients are almost always unique, but in special cases some additional information is necessary to identify the desired entity (e.g. the county of a town or the address of the client).

By the same token, we also allow for defining more than one key value for retrieving the same entity. In our PANDA database management system this access method was implemented and it seems very helpful to treat a common situation adequately.

7.4 Spatial access

A greal deal of data in a spatial information system are related to objects in space. For these objects location and extension are known. Many applications need access to data based on location (Burton, 1978). This is obvious for all retrievals in order to produce a map: all objects within the map frame must be retrieved and then graphically rendered. Similar accesses are necessary for internal operations to check incoming geometric data and for geometric manipulations of stored data. Spatial access to data is based on a query of the form:

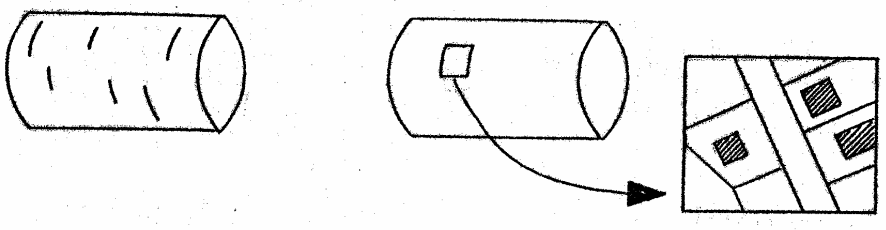
Retrieve all <object-type> within <area>,

where <area> is the description of the area of interest. In order to provide a general efficient function, it seems appropriate to reduce <area> to a rectangular box parallel to the coordinate system. More specific requests are then treated in two steps: in a first step all data within the rectangular envelope

of the area are retrieved, and the more expensive, exact selection process is done in a second step only on the data passing the first test.

The result of such a retrieval may be a large number of objects; a map drawing in the size of a CRT is, in our estimation, composed of 2000 .. 3000 objects. For an interactive application these data must be retrieved and displayed in a few seconds. This is only possible if a special data structure has been built before - sequential scan of the complete database takes much too long. Second, the data necessary must be physically clustered on the disk and accessible with a few disk access operations (if each object would be accessed individually, approximately 2000*50 msec = 100 sec would be required).

In simple words, it is necessary to find a mapping from real world space to the linear storage space, preserving vicinity.



regular distribution of data on disk
→ many accesses
→ slow response

clustered storage on disk
→ few accesses
→ fast response

figure 1

Only a few data structures are known to permit fast response to this type of two-dimensional range queries. They are all based on self-adjusting partition of space and clustering data of one partition in a disk page, so that access to one page brings in many entities useful for producing the requested map (Nievergelt et al., 1984) (Tamminen, 1982). The method described in (Nievergelt, 1984) is an adaptation of a more general hash based structure for storage of multidimensional objects. It turns out to be very similar to the method we have developed (Frank, 1981) and refined during the last few years (Frank, 1983).

This speeds up the retrieval of map output - all objects on a map are close to each other, and it benefits many other forms of geometric data processing. We can thus exploit the specific geometric locality of access observed in most algorithms of computational geometry (Dutton, 1978).

In the FIELD TREE method clustering is not only guided by location and extension of objects but includes also a component of level of importance of an object. This speeds up responses to "overview" requests (small area, all objects). Response time for queries is linearly dependent on the number of objects retrieved.

Other influences are the size of the area of the query and the number of objects stored for this area. The total number of objects stored in the system has no measureable influence on response time.

The traditional approach found in many of the systems on the market is to divide the world space into map sheets (sometimes called 'facettes'), and to store the data of these map sheets as individual files. The amount of data within such a file is then small enough to permit the use of linear search methods. This is in our opinion insufficient for the following reasons:

- the granularity of access is fixed and gives fast access only if the query area is comparable to the sheet size; for queries about much larger areas response is slow since many different files must be opened and read in;

- access to more than one sheet requires recombination of objects at the sheet borders. This is a complicated and time consuming operation and only possible with understanding of the internal structure of the representation of the objects. Thus the types of geometric objects treated are defined in the storage layer, and it is difficult for a user to add new geometric object definitions;

- most systems do not hide the sheet decision from the user, nor do they provide a query language working automatically across sheet boundaries. It seems acceptable to ask draftsmen to know which map sheet they update; it is however clearly inappropriate to require this knowledge from a casual user needing a thematic map.

8. Datamodels

A data model provides a method to describe the data necessary for an application domain. It must contain tools to describe the entities and the relationships between them in a structured, (semi-) formal way.

Often users will be forced to use two different data models:

- a very high level model, which contains tools to integrate many of the semantics of the data for the design of the database, and

- a lower level model, used with the database management system.

In this section we will briefly discuss requirements for the data model used for design. It is then important to check that the data model used with the selected database management system can render the situation expressed in the design data model adequately, and rules for translating constructs from the one to the other must be established.

As was stated before, a data model must provide tools to describe the situation as accurate as possible, following the cir-

cumstances we find in the real world. At least in the phase of the design of the database we should not consider aspects to facilitate data processing, but strive for a picture of the real world as correct as possible.

The task is similar to problems treated in Artificial Intelligence, there labeled 'knowledge organization'. Three fundamental conceptual tools for abstraction have been identified, namely:

- classification: forming a group of entities with the same properties
- aggregation: combining several different classes to a new one, and
- generalization: extracting from several different classes a more general one (Smith 1977) (Mylopoulos, 1980).

Most data models provide tools for classification and aggregation, but generalization is often lacking.

We use currently a system related to the entity-relationship model (Chen, 1976), stressing graphical, visual aids to make the data structure easy to see. Our method is based on entities, attributes, and relationships (sets) between the entities, comparable to many other data models proposed in the literature.

We extended this model with generalization to be able to express adequately the relationship between e.g. a 'straight line', an 'arc of circle' and the general concept of 'line'. This has proven to be a very important and often used concept and should be included in any method to design software (Borgida, 1984).

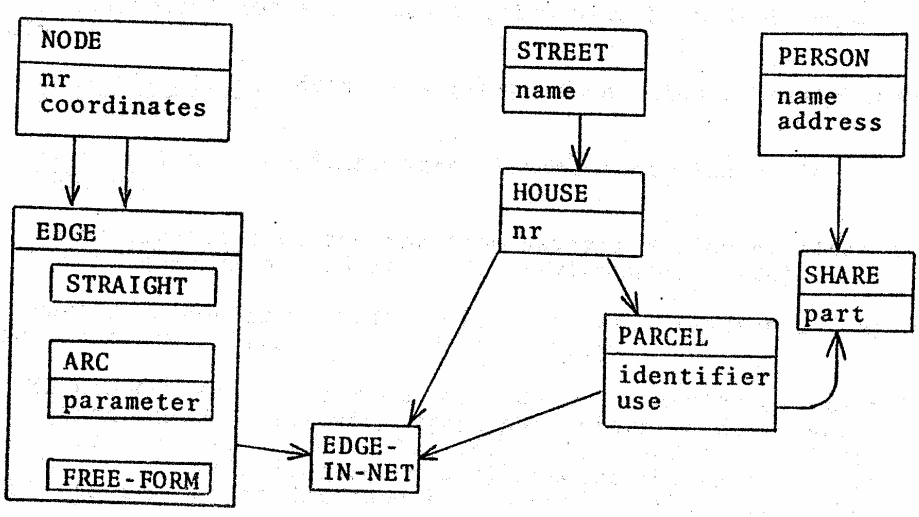


figure 2

In figure 2 we show a simple example of a schema using our schema design method (Frank, 1983a).

We use the same data model in the PANDA database, and only minor changes are needed from the design schema to the implementation schema.

It is possible to translate generalization to datamodels lacking this concept, but this results in more complicated schema and programs which do not naturally express their intents. It seems desirable to use high level programming languages which can support generalization, e.g. Pascal using records with variant parts, or languages that are based on generalization and inheritance (Simula, Smalltalk (Goldberg, 1983)).

9. Abstract Data Types

Some data models provide only one level of aggregation (record) and only the base data types found in most programming languages (i.e. integer, real, character string). This is clearly insufficient for non-standard applications like spatial information system. We encounter typically new datatypes like:

- values with associated error estimates
- points described by several coordinate values
- lines described by a sequence of points

and it is desirable that we can use the concept of 'Abstract Data Types' to design our software (Parnas, 1972) (Guttag, 1977). An Abstract Data Type is a package which contains a data structure and operations on this data structure. Such a package can be abstractly specified without discussing the implementation just by stating the results of applications of the operations. Application programs are only allowed to use the operations defined to manipulate these data and must never make use of special, not specified properties of the implementation, nor directly access or change these data.

This technique for software design and programming carries considerable advantages since small, self contained and easy-to-test programs may be written, each embodying one single idea. It minimizes the dependencies between different parts of a program, and therefore simplifies software maintenance and improvements.

Such an abstract data type can, for example, be defined and implemented for chain encoded lines and should support operations like:

- vector to chain encoding and reverse
- display routines for different output devices
- detection of intersections of such lines, etc.

If the database supports abstract data types, these encoded lines can be stored, retrieved and connected to other entities in

the database, without any special programming.

This example shows clearly that generalization is advantageous, as such chain-encoded lines are just a special form of line (others being e.g. straight vector, arc of circle), and it is easier if on a higher level of programming we can use the same operations for all these lines, completely disregarding the specific form of storage. A generalized Abstract Data Type 'line' will then offer the operations:

- display line
- detection of intersection of line

and internally select the correct algorithm for the representation at hand.

10. Special treatment of geometric data and their consistency constraints

An example how abstract data types help to treat geometric data was given in the previous section. Using abstract data types and generalization it is possible to build a complete package of routines to treat all sorts of geometric data.

In (Cox and Rhind, 1980) (Burton, 1979) examples of geometric data types , defined as abstract data types, are described. These proposals must be carefully revised to see if they can be applied to a certain problem domain. Special attention should be paid to treatment of the inevitable errors associated with measuring.

Using these basic building blocks more complex structures can be built, including polygon and network/partition structures (figure 3).

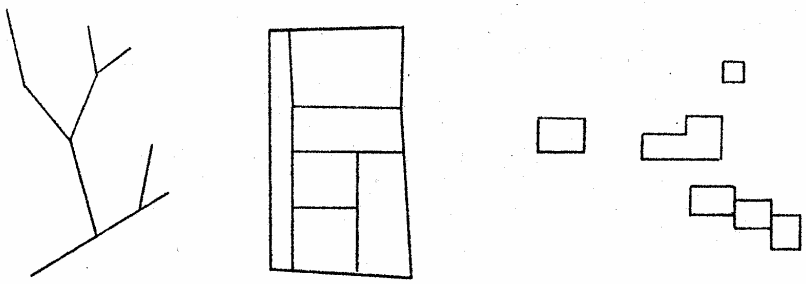


figure 3

These structures do not only contain metric information on the position of the points and the form of the lines, but also topological information on the connections between the points and on which areas are adjoining. Certain types of applications make explicite use of toplogical structures (e.g. shortest path detection), and new abstract data types must be formed to treat such objects.

Two aspects must be included in the definition of these abstract data types which make them somewhat different from other areas of application.

Graphic rendering: For most objects treated in a spatial database, operations for graphical rendering on a map must be included. First, this entails the connection of the database management system to the graphical output system. Our studies of existing graphics packages and the operations they offer, reveal two problems:

- they offer some, relatively primitive data structuring tools (e.g. segments (GKS 1984)), which are not sufficient to represent topological or other typical information structures, but are complicated and costly to use.

- they offer only few functions useful for cartographic applications, and most operations necessary for production of maps are either missing or do not provide the quality required (e.g. line styles).

It seems that a thorough investigation into the graphic operations necessary for cartography is inevitable. We concluded that we rather build our own library of graphics procedure and use directly the data in the database without additional storage. However, graphical output operations can not be included in the operations of an abstract data type of a stored object (e.g. a house, or a country) as the graphical rendering of the geometry of the object is subject to changes according to map style and scale. An additional mapping of geometric objects to graphical output is necessary.

Consistency constraints: For every large data collection intended for long term usage controls to check all incoming data must be installed. Without such checks in the long run small errors will invariable creep in and make the data collection worthless. Obviously no computerized system can check whether the data entered are correct (i.e. conforming to the exterior reality) or not, but we can at least make sure that the new data follow some pre-established rules and do not contradict data already stored. Internal contradictions in the data do not contribute to the confidence the users have in the answers they receive from the information system. All inconsistencies can cause problems when algorithms rely on certain conditions in the input data, and behave unexpectedly if the input data does not conform to these rules (e.g. exit of the program, loops, etc.). Algorithms may of course be written in order to guard against the most disturbing consequences of such inconsistencies, but in general it is not possible to produce the correct result (e.g. what is the area of the figure 4?).

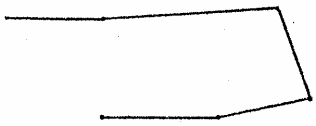
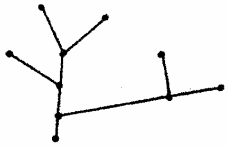


figure 4

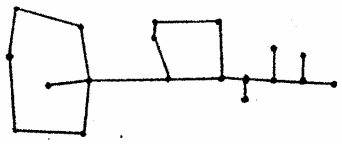
The rules that specify conditions the data must fulfill are called integrity or consistency constraints. They may come in different forms, e.g. specifying a range for a value like "the age of a person is between 0 and 120 years", or more complex, "the boundary polygon of a county is closed".

Integrity constraints for geometric descriptions are usually more involved and not easy to formulate. On the other hand, their violation - especially that of topological constraints - can not be forgiven by most exploiting programs. The U. S. Bureau of Census, which holds probably the largest collection of topologically structured spatial data, has spent considerable effort to install checking procedures for DIME files (Corbett and Farnsworth, 1972) (Cranford, 1978) (White, 1984).

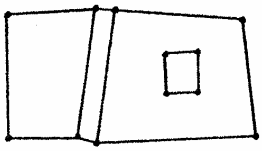
We generalized this topological approach to several different geometrical structures (some examples given in figure 5) which can be characterized by few topological conditions (Frank, 1983).



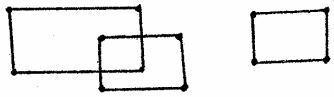
a) connected, planar, no cycles



b) connected, planar, some cycles



c) not connected, planar, all cycles



d) not connected, not planar, all cycles

figure 5

Figure 5a, forms a 'tree' and there are no cycles, but all edges are connected. In contrast, in figure 5c all edges are part of cycles, but the graph is not connected.

Tentatively we use the three conditions

- connected - not connected
- all cycles - some cycles - no cycles

- planar graph - nonplanar graph (i.e. edges may/must not cross).

This seems to cover at least a large class of applications.

In an interactive environment it is necessary to check these conditions after each change that affects the topological structure, at least for the entities affected. At the moment we work on improving the efficiency of these checks.

11. Interactive Query Language

The discussion so far has described design criteria which are primarily important for the programmer of application programs, and determine the effort and cost involved in designing, programming, documenting and maintaining applications dealing with spatial data.

Often users need answers to relatively simple problems, but they do not have the time or the knowledge to write special programs in a high level programming language. In such cases an interactive query language can provide a method to express a simple information need in a language easy to learn and to get an immediate answer. Interactive query languages are the casual user's key to the wealth of data stored in an information system!

Interactive query languages are available for most commercial database management systems. They are widely used, and several studies were made to determine their relative user-friendliness (Reisner et al., 1975).

They are usually based on the relational data model, and provide a method to select the data the user is interested in using some form of predicate calculus. In figure 6 a typical formulation in SQL (Chamberlin et al., 1976) is given for illustration.

```
SELECT  StreetName, HouseNr
FROM    House
WHERE   HouseNr = 20
AND     StreetName =
        SELECT StreetName
        FROM   Street
        WHERE  StreetName = 'Main Street'
```

figure 6

Second, they provide some mechanism to describe the form the answer is desired in; this part of the query specification is usually not given by the user and default values are used. The output is in all systems a list, some more advanced programs also offering simple forms of graphics.

Commercially oriented systems do not provide any methods to retrieve data, and have them presented as a map. They may be marginally useful for retrieval of spatial data in alphanumeric form, but can not fulfill the requirements of a user of a spatial information system.

Essentially the same functionality is necessary for an interactive query language in a spatial information system, but input and output must be possible in form of map drawings. In (Frank 1982b) we present the design of MAPQUERY, a query language for retrieval of spatial data. In the meantime it has been implemented (Eggenhofer, 1984).

The following special problems, added to those addressed in alphanumeric query languages, had to be solved:

- selection of data: it is not sufficient in most cases just to retrieve the data element the user asks for, but sufficient other data must be retrieved and rendered to build a context within which the user's data may be interpreted. The context can be either explicitely selected by the user or it may be a named standard context.

- selections of a map window: it is necessary to determine the area which must be shown (and for which the context must be retrieved). The area can either be explicitly specified by the user using coordinate input, preferably interactively indicating two corners on a previously drawn map, or selected by the system in order to accommodate the object the user has selected (e.g. a county, a house, or a street).

- defining the map symbols to be used for representing the data. Usually the user will be content with some standard lists of conventional signs, but in certain cases the graphical differentiation based on a selected attribute and value will be necessary (e.g. show all houses of a town with different crosshatching depending on age).

- input of selection criteria using a pointing device and the map resulting from the previous query.

The implemented prototype proved feasibility of the idea. Inspection of the code - approximatively 5000 lines of Pascal - reveal some points warranting closer scrutiny before a new implementation is started:

- the information on the data structure in the database must be accessible to the query language program (it is advantageous to make it generally available).

- more descriptive information on the data structure, including geometric properties etc, should be present in the database.

- advanced forms of mappings from the conceptual schema used for the database design to the partial and simplified schema (user view) the user needs for formulating a query are necessary; such mappings must include simplifications of the geometric structures internally used to a single attribute "geometry" in the users view.

12. Conclusions

Database management systems offer many advantageous properties over traditional file structures:

- increased modelling power to better render the data structure of the application area;
- use of data by several users at the same time, and for many different applications;
- easier maintenance of programs and faster adaptation to changing requirements.

To apply the database concept to spatial data collections seems the only way to realize more complex multi-purpose spatial databases. However, many of the database management systems available were designed for commercial applications, and are not well suited for the management of spatial data.

This paper has drawn a framework for discussion of the implementation of a database management system, and enumerated features important for spatial data bases:

- In order to achieve fast response time, attention must be paid to the physical clustering of data on the storage device;
- the buffering schema must allow to hold all data necessary for a map drawing to avoid repeated reads from the storage device.
- consistency constraints are rules describing properties of the data which must hold always, and which are enforced automatically. Consistency constraints must include descriptions of the geometric properties of the data. This reduces errors introduced in the data collection and makes data collection usable for a long time.

A database management system especially suitable for spatial data handling has been built to show viability of the solutions proposed. Building a database of sewer line data on a personal computer (PERQ workstation), including graphical interaction, was then easy. Presently we work on improving the modeling of geometric data structures and operations for cartographic rendering, independently of application area.

Most database management systems offer an interactive query language to retrieve data from the database without formal programming. A similar feature for spatial databases, including cartographic output, has been shown feasible. Such query languages are invaluable tools for casual users to exploit the wealth of information collected in a database.

BIBLIOGRAPHY

Burton, W., 1978. "Efficient Retrieval of Geographical Information on the Basis of Location". In (Dutton, 1978).

Burton, W., 1979. "Logical and Physical Data Types in Geographical Information Systems". Geo-Processing. Vol. 1 p. 167.

Borgida, A., T. Mylopoulos and H. Wong., 1984. "Generalization as a bases for software specification". In M. Brodie et al. (Eds). Perspectives on Conceptual Modeling. Berlin Springer.

Chamberlin, D.D., and et al., 1976. "Sequel 2: a unified approach to data definitions, manipulations and control". IBM Journal Research and Development Vol. 20 p. 560.

Chen P., 1976. "The entity-relationship model: toward a unified view of data". ACM Transactions on Database Systems. Vol. 1 p. 9.

CODASYL, 1962. "An Information Algebra, Phase I report". Commun ACM Vol. 5, p. 190-204.

CODASYL, 1971. Data Base Task Group (DBTG) Report, 1971.

Codd, E.F., 1982. "Relational Database: A Practical Foundation for Productivity". Commun. ACM. Vol. 25 p. 109.

Corbett, T. and G. Fransworth, 1972. "Theoretical Bases of Dual Independent Map Encoding (DIME)". International DIME Colloquium Conference Proceedings. Census Bureau, Washington, D.C.

Cox, N.J., B. K. Aldred and D.W. Rhind, 1980. "A relational data base system and a proposal for a geographical data type". Geo-Processing Vol. 1 p. 217.

Cranford, G.F., 1978. "Editing and updating Geographic Base Files - A Discussion of Practical Processing Alternatives", in (Dutton, 1978).

Denning, D.E., U. Schlörer, 1980. "A Fast Procedure for Finding a Tracker in a Statistical Database". ACM Transactions on Database Systems. Vol. 5.

DeWitt, D.J. and P.B. Hawthorn, 1981. "A Performance Evaluation of Database Machine Architectures". In: C. Zaniolo and C. Delobel (Eds.). Proceedings VII International Conference on Very Large Database. Cannes (France) p. 199.

Dutton, G. "Navigating ODYSSEY" In: (Dutton, 1978).

Dutton, G. (Ed), 1978. First international advanced study symposium on toplogical data structures for geographic information systems". Harvard Papers on Geographic Information Systems. Harvard University, Cambridge, Massachusetts.

Eggenhofer, M., 1984. "Implementation of MAPQUERY" (in german). Institute of Geodesy and Photogrammetry, Swiss Federal Institute of Technology. Zurich, Switzerland.

Frank, Andre, 1980. "Land Information System - an attempt for definition" (in german): Nachrichten aus dem Karten - und Vermessungswesen, Reihe 1, Heft 81. Frankfurt (FRG), 1980.

Frank, A., 1981. "Applications of DBMS to Land Information Systems". In C. Zaniolo and C. Delobel (Eds.) Proceedings VII International Conference on Very Large Data Bases. Cannes (France). p. 448.

Frank, A., 1982a. PANDA: Pascal Network Database Management System (in german). Report 62. Institute for Geodesy and Photogrammetry, Swiss Federal Institute of Technology, Zurich, Switzerland.

Frank, A. 1982b. "MAPQUERY: Data Base Query Language for Retrieval of Geometric Data and their Graphical Representation". SIGGRAPH '82 Conference Proceedings, Computer Graphics. Vol. 16. p. 199.

Frank, A. and M. Tamminen, 1982. "Management of spatially reference data". In Leick, A. (Ed.) Proceedings International Symposium Land Information at the Local Level. University of Maine at Orono. P. 330.

Frank, A., 1983a. Data Structures for Land Information Systems-Semantic, Topological and Spatial Relations in Data of Geo-Sciences (in german). Ph.D. thesis. Swiss Federal Institute of Technology. Zurich, Switzerland.

Frank, A., 1983b. "Storage Methods for Space Related Data: The FIELD TREE". In: M. Barr (Ed.) Spatial Algorithms for Processing Land Data with a Microcomputer. Boston, Lincoln Institute of Land Policy.

GKS. Graphical Kernel System American National Standard Draft Proposal. X 3. 124-198x. Computer Graphics Vol. 18.

Goldberg, A. and D. Robson, 1983. Smalltalk - 80: The Language and its Implementation. Addison Wesley.

Guttag, J., 1977. "Abstract Data Types and the Development of Data Structures". Commun. ACM, Vol. 20 p. 396.

Härder, Th. and A. Reuter. Database Systems for Non-Standard Applications. Report 54/82, Fachbereich Information, Universität Kaiserslautern. (FRG).

ISO/TC97/SC16. "ISO Reference Model of Open System Interconnections". Version 4 as of June 1979.

Kung, H.T. and J.T. Robinson, 1981. "On Optimistic Methods for Concurrency Control". ACM Transactions on Database Systems. Vol. 6 p. 213.

Lampson, B.W. et al., 1981. Distributed Systems - Architecture and Implementation. Lecture Notes in Computer Science 105, Berlin, Springer.

Mylopoulos, T., 1980. "An Overview of Knowledge Representation". Proceedings of the Workshop on Data Abstraction, Databases, and Conceptual Modeling. Pingree Park, Colorado. SIGMOD Record. Vol. 11 p. 5.

Nievergelt, J. et al., 1984. "The Grid File: An Adaptable Symmetric Multikey File Structure". ACM Transaction on Database Systems. Vol. 9.

Parnas, D.L., 1972. "On the Criteria to be Used in Decomposing Systems into Modules". Commun. ACM. Vol. 15 p. 1053.

Reisner, P., R.F. Boyce, D.D. Chamberlin, 1975. "Human Factors Evaluation of Two Data Base Query Languages - Square and Sequel", AFIPS Conference Proceedings, 1975 National Computer Conference, Vol. 44, AFIPS Press, 1975.

Salton, G. and A. Wong, 1978. "Generation and Search of Clustered Files". ACM Transaction on Database Systems. Vol. 3.

Scheck, H.J. and V. Lum, 1983. "Complex Data Objects: Text, Voice, Images: Can DBMS Manage them?" In: M. Schkolnick, C. Thanos (Eds). Proceedings 9th International Conference on Very Large Databases. Florence (Italy).

Smith, J.M. and D.C.P. Smith, 1977. "Database abstraction: Aggregation and Generalization". ACM Transactions on Database Systems. Vol. 2 p. 105.

Tamminen M., 1982. "Efficient Spatial Access to a Database". Proceedings ACM SIGMOD Conference.

White, M.S., 1984. "Technical Requirements and Standards for a Multipurpose Data System". The American Cartographer. Vol. 11 p. 15.

REPORT: Surveying Engineering Publications and Reprints
The following reports were published and are available upon request

1. Defining the Celestial Pole, A. Leick, Manuscripta Geodetica, Vol. 4, No. 2.

2. A New Generation of Surveying Instrumentation, A. Leick, The Maine Land Surveyor, Vol. 79, No. 3.

3. The Teaching of Adjustment Computations at UMO, A. Leick, The Maine Land Surveyor, Vol. 79, No. 3.

4. Spaceborne Ranging Systems - A useful tool for network densification, A. Leick, The Maine Land Surveyor, Vol. 80, No. 1.

5. Potentiality of Lunar Laser Range - Differencing for Measuring the Earth's Orientation, A. Leick, Bulletin Geodesique.

6. Crustal Subsidence in Eastern Maine, D. Tyler, J. Ladd and H. Borns; NUREG/CR-0887, Maine Geological Survey, June 1979.

7. Land Information Systems for the Twenty-First Century, E. Epstein and W. Chatterton, Real Property, Probate and Trust Journal, American Bar Association, Vol. 15, No. 4, 890-900 (1980).

8. Analysis of Land Data Resources and Requirements for the City of Boston, Epstein, E.F., L.T. Fisher, A. Leick and D.A. Tyler, Technical Report, Office of Property Equalization, City of Boston, December 1980.

9. Legal Studies for Students of Surveying Engineering, E. Epstein and J. McLaughlin, Proceedings, 41st Annual Meeting, American Congress on Surveying and Mapping, Feb. 22-27, 1981, Washington, D.C.

10. Record of Boundary: A Surveying Analog to the Record of Title, E. Epstein, ACSM Fall Technical Meeting, San Francisco, Sept. 9, 1981.

11. The Geodetic Component of Surveying Engineering at UMO, A. Leick, Proceedings of 41st Annual Meeting of ACSM, Feb. 22-24, 1981.

12. Use of Microcomputers in Network Adjustments, A. Leick, ACSM Fall Technical Meeting, San Francisco. Sept. 9, 1981. (co-author: Waynn Welton, Senior in Surveying Engineering).

13. Vertical Crustal Movement in Maine, Tyler, D.A. and J. Ladd, Maine Geological Survey, Augusta, Maine, January 1981.

14. Minimal Constraints in Two-Dimensional Networks, A. Leick, Journal of the Surveying and Mapping Division (renamed to Journal of Surveying Engineering), American Society of Civil Engineers, Vol. 108, No. SU2, August 1982.

15. "Storage Methods for Space Related Data: The FIELD TREE", A. Frank in: MacDonald Barr (Ed.) Spatial Algorithms for Processing Land Data with a Minicomputer. Lincoln Institute of Land Policy 1983.

16. "Structure des données pour les systèmes d'information du territoire", (Date Structures for Land Information Systems), A. Frank in: Proceedings 'Gestion du territoire assistee par ordinateur's, November 1983, Montreal.

17. "Semantische, topologische und räumliche Datenstrukturen in Landinformationssystemen (Semantic, topological and spatial data structures in Land Information Systems) A. Frank and B. Studenman, FIG XVII Congress Sofia, June 1983. Paper 301.1.

18. Adjustment Computations, A. Leick, 250 pages.

19. Geometric Geodesy, 3D-Geodesy, Conformal Mapping, A. Leick.

20. Text for the First Winter Institute in Surveying Engineering, A. Leick, D. Tyler, 340 pages.

21. Adjustment Computations for the Surveying Practitioner, A. Leick, (co-author: D. Humphrey, Senior in Surveying Engineering).

22. Advanced Survey Computations, A. Leick, 320 pages.

23. Surveying Engineering Annual Report, 1983-84.

24. Macrometer Satellite Surveying, A. Leick, ASCE Journal of Surveying Engineering, August, 1984.

25. Geodetic Program Library at UMO, A. Leick, Proceedings, ACSM Fall Convention, San Antonio, October, 1984.

26. GPS Surveying and Data Management, A. Leick, URISA Proceedings, Seatle, August, 1984.

27. Adjustments with Examples, A. Leick, 450 pages.

28. Geodetic Programs Library, A. Leick.

29. Data Analysis of Montgomery County (Penn) GPS Satellite Survey, A. Leick, Technical Report, August, 1984.

30. Macintosh: Rethinking Computer Education for Engineering Students, A. Frank, August, 1984.

31. Surveying Engineering at the University of Maine (Towards a Center of Excellence), D. Tyler and E. Epstein, Proceedings, MOLDS Session, ACSM Annual Meeting, Washington, March, 1984.

32. Innovations in Land Data Systems, D. Tyler, Proceedings, Association of State Flood Plain Managers, Annual Meeting, Portland, Maine, June 1984.