18

DATABASE MANAGEMENT SYSTEMS

R G HEALEY

Database Management Systems (DBMS) are an integrated and crucial component of most successful GIS. DBMS are used to store, manipulate and retrieve data from a database. A key element in creating a spatial database is database design using a variety of data modelling techniques. Although the range of DBMS structures used in GIS includes inverted list, hierarchical, network and relational designs, it is the latter which has come to dominate the field. Two main approaches have been used in the design of GIS software systems: the hybrid and integrated models. Both have advantages and disadvantages for specific applications. Looking to the future, the main issues facing the spatial database world are the likely impact of the ideas of the object-oriented community and the need to develop distributed systems capable of handling temporal data in an efficient manner.

INTRODUCTION

Among the different threads that can be observed in the development of GIS methodology, one major one has been the progressive realization of the importance of database management systems (DBMS), initially for handling map attribute data, but increasingly for handling digital cartographic data also. While many of the operations required for data manipulation in GIS are now seen to be specific instances of more general classes of database problem, standard database tools have also been shown to have a number of limitations when applied to GIS processing.

This chapter examines the principles of database management, as they apply to GIS, and the strengths and weaknesses of alternative approaches to the use of database tools. The first section examines the fundamental characteristics of DBMS. This is followed by sections on data modelling, database design, database structures and alternative methods of utilizing a DBMS within GIS. The final section examines future developments and the ways in which they may contribute to the solution of outstanding technical and methodological problems in database management for GIS.

FUNDAMENTAL CHARACTERISTICS OF DBMS

To clarify definitions at the outset, the term 'database management system (DBMS)' will be used to refer to a software package for the storage, manipulation and retrieval of data from a database. A database is a collection of one or more data files or tables stored in a structured manner, such that interrelationships which exist between different items or sets of data can be utilized by the DBMS software for manipulation and retrieval purposes. The database will, in general, serve the data requirements of a variety of users rather than a single individual. If the restriction is enforced that the DBMS software provides the only means of access to the database, there are a number of important implications (Martin 1976):

- The method of data storage can be considered independently of the programs that access the database.
- A controlled and standardized approach to data input and update can be enforced, with appropriate validation checks to ensure data integrity and consistency between data files.
- Security restrictions on access to specific data subsets can be applied.
- A consistent approach can be adopted for managing simultaneous multi-user read and update operations on specific files or tables.

It is these factors, coupled with the elimination of unnecessary redundancy in data storage because multiple users can share data, that gives the combination of DBMS software and database its greatly enhanced efficiency and productivity in data management, as compared to individuals working with their own programs and file structures.

DBMS software components

The central component of a DBMS is the kernel software, usually written in C or FORTRAN, which controls the processing of queries, access paths to data, storage management, indexing and multi-user read/update operations. Linked to the kernel are a variety of interfaces to the user. These include query language interfaces, bulk data loaders, screen forms management systems, menu handlers, report writers and programming language interfaces. Query language interfaces allow the user to issue *ad hoc* queries against the database which result in data from one or several linked tables or files being retrieved. These queries are expressed in high level languages, which formerly were often system specific (Collins 1982; Martin 1983). Now there is convergence in the commercial marketplace on SQL, the ANSI standard for DBMS query languages. If the query language contains additional procedural functionality, which SQL itself does not, it can be called a 'fourth generation language'. Since these languages operate with query language commands, which may individually cause large amounts of program code in the database kernel to be executed, they are at once very powerful and potentially CPU intensive.

Programming language interfaces perform a different function. They enable users to embed query language statements within programs written in standard general purpose 'third generation' languages such as FORTRAN or C. Data can be retrieved from the database directly into variables or data structures accessible to the program, where they can be further processed. This approach is more flexible and efficient in the use of computing resources than employing a fourth generation language, but it is considerably more costly in terms of users' time! Interface programs such as screen handling systems, referred to above, are generally implemented as stand-alone utility programs which use these programming language interfaces to pass requests for data from the users through to the database kernel. A simplified diagram of the relationships between the host processor, the DBMS software and application programs is given in Fig. 18.1.

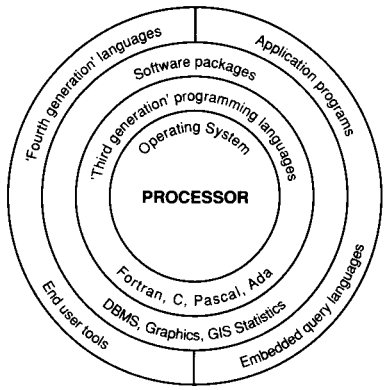

Fig. 18.1 The DBMS as part of a layered software model.

DATABASE DESIGN

Physical and logical database design

As a database is likely to be a shared resource which as it develops will come to represent a major investment by any given organization, it is most important that this investment is protected by careful database design. An initial distinction should be made between physical and logical

design. Physical design is concerned with the location of different parts of the database within the file system of the computer. This may include considerations such as spreading the database across multiple disk drives to balance input/output load or for security in the event of disk media failure. Physical design is the responsibility of the database administrator and should be quite distinct from logical design, which represents the user's view of the interrelationships between data sets stored in the database. If physical and logical design are kept separate, users can access their data sets without having to concern themselves with details about where and how those data sets are physically stored (Martin 1976).

Data analysis

The first stage in logical database design is the use of data analysis techniques to develop a clearly defined conceptual model of the relationships between different data sets. These relationships may be specific to a small number of data sets required for a single user, or they may extend to all the components of a large corporate database. Regardless of the size of system, if this conceptual model is not built correctly, the likely outcome will be an inefficient database structure with unnecessary redundancy in data storage and a poor match to users' requirements for data access and retrieval.

There are a variety of data analysis or data modelling techniques that can be used (Howe 1985; Worboys, Hearnshaw and Maguire 1990a), but the entity–relationship model approach (Chen 1976) has met with the widest acceptance. Chen's approach is based on a number of fundamental concepts including entity sets, attributes, domains, relationship sets and mappings.

Entity sets represent the generic structure of phenomena which are relevant to the specific database being designed. They might be towns, census districts, hotels or national parks, for example. Each entity belonging to a particular entity set will have a number of characteristics or attributes. In the case of census districts these might include an identification number, X,Y coordinates of a centroid and a list of census variables. Each attribute will have a range of possible values which constitutes its domain or value-set. For example, identification numbers may range between 0001 and 9999 or hotels may have tourist guide quality ratings between 1 and 5. Relationship sets are formally defined as subsets of the cross product of two or more entity sets (Tsichritzis and Lochovsky 1982). The specific relationship between individual members of the respective entity sets provides the subsetting mechanism, for example the fact that certain members of the hotel set are located in a particular town. Specific relationships or mappings between entity sets may take a variety of forms. One-to-one mappings refer to the situation where, for example, each town has one and only one set of municipal offices, while a one-to-many mapping would be where a town had a number of hotels. Many-to-many mappings deal with cases such as that of wholesalers distributing goods to different shopping centres. Each centre will be served by multiple wholesalers and each wholesaler will distribute goods to several centres. A number of refinements in the specification of relationships are possible (Ellis 1985) depending, for instance, on whether or not they are defined to be mandatory (every hotel must be located in one and only one town).

Using these fundamental concepts, it is possible to develop sophisticated models of data interrelationships. The availability of diagramming methods linked to the concepts allows graphical representations of models to be drawn. These are powerful aids to model articulation and as a means of communication. A simple example involving national parks, scenic trails and landscape features is given in Fig. 18.2.

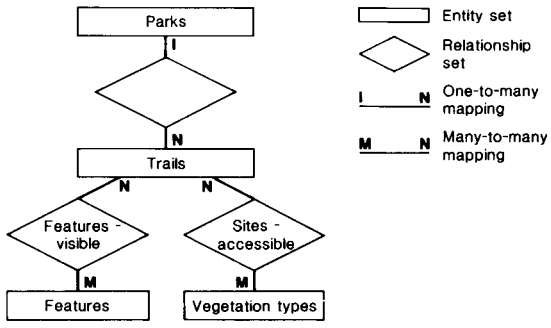

Fig. 18.2 An example entity–relationship model for a national parks database.

While the use of data analysis techniques is much less prevalent in the GIS literature than is desirable, important examples of the application of these methods to the structuring of relationships between types of digital cartographic data can be found in van Roessel and Fosnight (1984) Tuori and Moon (1984) and Hearnshaw, Maguire and Worboys (1989).

TYPES OF DBMS STRUCTURE

Once the data analysis is complete, the resulting data model must be implemented using suitable DBMS software. This might be developed in-house, but unless the organization is large and the programming resources considerable, it is unlikely it would have the required facilities. More likely, the software will be one of the large number of systems currently available commercially. These systems can be broadly categorized into four main types:

- inverted list systems;
- hierarchical systems;
- network systems;
- relational systems.

Important new approaches, which have not yet established a major presence in the commercial marketplace, are discussed in a later section. While some software packages may have characteristics drawn from more than one of the above types, in general the category into which any given package falls gives a clear indication of the way in which it structures data sets and their interrelationships at the level of logical database design. It should be noted also that the first two categories developed from refinement and improvement of commercial approaches to data management and are largely represented by older systems (Date 1986), while the last two are in turn both more recent in origin and more soundly based on theoretical, rather than pragmatic considerations. Choosing any one of these different types of system will have a major impact on the way in which the data model for a particular application problem maps onto the underlying database structure.

Inverted list systems

The basic storage mechanism for data in this DBMS structure is by means of tables/files containing rows (records) and columns (fields). The specific ordering of rows within tables has importance for the ways in which the data can be accessed. Further means of retrieving data are provided by search keys which index the occurrence of values for a specific field. If the field occurs in more than one table, this information will also be incorporated in the index (Date 1986). The index takes the form of an inverted list with a pointer from each row in the list to the actual disk location where the relevant data may be found.

The inverted list structure can be illustrated using the national parks example. Assume that there is a Trails entity set that has four attributes (Table 18.1(a)): an identification number (Trail#), Name, Category (easy E or difficult D for walking) and an identification number for the Park in which it is located (Park#). Similarly a Landscape Features entity set has attributes of Feature#, Type and Origin. The last of these describes the type of process responsible for its formation. The Features-visible relationship set has attributes Trail#, Feature# and two attributes for the latitude (Lat) and longitude (Long) in decimal degrees of the point on the trail from which the feature is visible. Directly translated into the format of data tables these might appear as indicated in Table 18.1(a).

A simple inverted list representation of the contents of the Features-visible table, if Feature# were defined to be a search key, can be seen in Table 18.1(b). More probably a number of search keys would be defined, including Trail# and Feature#, both of which appear in more than one table, and a combined key Trail#/Feature#. The overall sequence of index entries for the multiple search keys might appear as in Table 18.1(c).

In this way a single index structure can be created across all the tables in the database. Index entries for the same data value in the corresponding field in two different tables (cf. the first two lines of the index in Table 18.1(c)) are found adjacent to each other. This facilitates searches which require information to be retrieved on both the attributes of

Table 18.1(a) The structure of part of the example database.

1. The National Parks Table

Park#	Name	Size (ha)
1	Ben Wyvis	40 000
2	Braes of Tomintoul	28 000
3	Inveresk Hills	63 000

2. The Trails Table

Trail#	Name	Category	Park#
1	Loch Sissons	D	2
2	Linton Forest	E	3
3	Hutton Crags	E	1
4	Davis Valley	D	2
5	Gilbert Falls	D	3

3. The Features Table

Feature#	Type	Origin
1	Drumlins	Glacial
2	Braided Channel	Fluvial
3	Delta	Fluvial
4	Corrie	Glacial

4. The Features-Visible Table

Trail#	Feature#	Lat	Long
1	3	57.35	4.52
1	4	57.50	4.48
2	2	56.25	5.82
2	3	56.34	5.94
3	1	57.82	3.55
3	4	57.88	3.62
4	1	57.60	4.45
4	2	57.68	4.39
5	3	56.53	5.79
5	4	56.41	5.86

the trail and details of the features that are visible from it.

Proprietary systems such as ADR DATACOM/DB and ADABAS, which use the inverted list approach, are noted for high performance in large database environments. In the past they have not been heavily used in GIS applications. This is more a result of their main operating environments being IBM mainframes, when most GIS work is performed on minicomputers and workstations, than inherent problems of using this type of database architecture.

Table 18.1(b) The Features-Visible Table as an inverted list.

Feature#	Trail#
1	3
	4
2	2
	4
3	1
	2
4	1
	3
5	3
	4

Hierarchical systems

In common with inverted list systems, this type was not developed from a formal theoretical model of database design, but followed the approach of the IBM IMS (Information Management System) software, first released in 1968 and still one of the world's most widely used systems (Wiederhold 1983; Date 1986). The basic concept of the hierarchical system is a familiar one, with a specific entity set defined to be the root of the hierarchical tree and a set of parent–child pointers from each level down to the one below, to represent the relationships between linked types of entity sets. Each link at a given level must follow back through other levels to the root. A strict interpretation of these rules would produce an implementation of the example national park database similar to that shown in Fig. 18.3. It should be noted that the tree structure is multi-dimensional and only the branching structure for one Park (3) is shown. The attributes of each trail would be stored on disk together with pointers to the entities represented at the level below.

Even from this very simple example, one of the problems of translating the entity–relationship model into a hierarchical structure becomes

Table 18.1c Possible index entries for multiple search keys (partly following Date 1986).

Search key	Data value	Source table	Pointer
Trail#	1	Trails	To disk address of this row
Trail#	1	Features-visible	To disk address of first row with Trail# = 1
Trail#	1	Features-visible	To disk address of last row with Trail# = 1
Trail#	2	Trails	To disk address of this row
Trail#	2	Features-visible	To disk address of first row with Trail# = 2
.	.	.	.
.	.	.	.
Feature#	1	Features	To disk address of this row
Feature#	1	Features-visible	To disk address of first row with Feature# = 1
.	.	.	.
.	.	.	.
Trail#/ Feature#	1/3	Features-visible	To disk address of row with Trail# = 1 and Feature# = 3
.	.	.	.
.	.	.	.

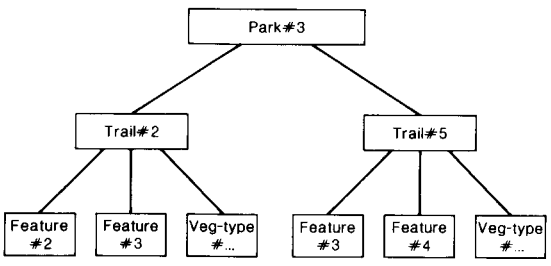

Fig. 18.3 Part of the national parks database in a hierarchical structure.

apparent. Although the one-to-many relationship between a park and the trails it contains is clearly represented, the many-to-many relationships between trails and the features visible from them are not handled in an equally satisfactory manner. Features that are visible from more than one trail will have their attributes stored multiple times, leading to undesirable redundancy in the data. One alternative would be 'horizontal' pointers from all the other occurrences of the feature in question to the one place in the tree where the attributes were stored, but this violates the parent–child requirement for linkages. The other alternative would be to classify all the details of features in separate tree structures and create pointers to those, but this violates the requirement for all linkages to link back to a single root. This latter approach tends to be that used for implementation purposes.

In summary, the hierarchical approach is very efficient for searching if all desired access paths follow the parent–child linkages. However, it requires a relatively inflexible structure to be placed on the problem at the outset, when the record type constituting the tree structure is set up. Many-to-many relationships do not fit naturally into the structure and extensive pointer systems are required to deal with them. The combination of inflexible structure and the overheads of maintaining or changing pointer systems makes extensive modification of the structure of hierarchical systems, to meet new requirements, a resource intensive operation. These reasons have contributed to the lack of adoption of this type of DBMS for flexible GIS requirements.

Network systems

Network systems are also referred to as CODASYL databases, because they are based on the proposals of the Database Task Group of the Conference on Data Systems Languages (CODASYL), the organization originally responsible for the definition of the COBOL programming language (Olle 1978; Date 1986). These proposals were subsequently developed into specific implementations of this kind of database structure, in products such as IDMS and MDBS III. The most important difference between the hierarchical and the network approach is that in the latter, a 'child' entity set can have more than

one parent and indeed any entity set can be linked to any other (Martin 1976).

Each entity set with its attributes is considered to be a node in the network. Relationship sets are represented as linkages in the form of pointers between individual entities in different entity sets. As a result, all the different forms of mapping – one-to-many, many-to-many, etc. – can be handled directly with large numbers of pointers. An example of how the parks database might be structured as a network database is given in Fig. 18.4, with the pointers represented as solid or dashed arrows.

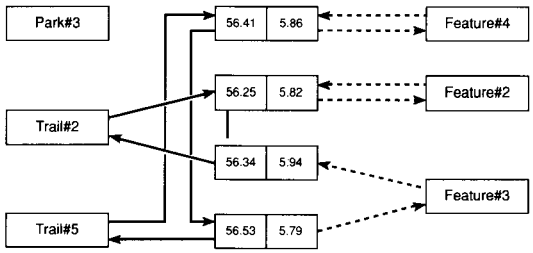

Fig. 18.4 Part of the national parks database in a network structure.

The network approach is powerful and flexible. For many applications it is also very fast and efficient in terms of CPU resources. From the implementation viewpoint, however, it may be comparatively difficult to set up the database correctly and, although the query language is comprehensive, it may also be complex and confusing for less expert users. Major restructuring of the database may be time consuming because of the extensive pointer structures that have to be rebuilt.

It might reasonably be thought that network databases would have found ready application in certain areas, such as work involving both GIS and locational analysis, where problems may fit quite naturally into the network structure. While important examples of this can be found (Armstrong, Densham and Rushton 1986; Densham and Armstrong 1987), it appears that the disadvantages of network systems from the user's viewpoint continue to militate against their widespread use in GIS.

Relational systems

The concepts of the relational approach were first set out by Codd (1970, 1979), as a means of describing data with their 'natural' structure only and ensuring independence of user-written application programs from the detailed storage formats of data within a database. In comparison to the previous approaches, relational systems are characterized by simplicity, in that all the data are represented in tables (relations) of rows and columns.

From the database design viewpoint, entity–relationship modelling fits very closely with relational systems. Each entity set is represented by a table, while each row or 'tuple' in the table represents the data for an individual entity. Each column holds data on one of the attributes of the entity set. Unlike other types of database, relationship sets describing many-to-many relationships between entity sets are also represented by a table of data values. The tables contain columns which reference the entity sets being related, together with further columns for any attributes of the relationship itself. Referring back to Table 18.1(a), the 'Features-visible' table is a good example of a relationship table, containing two columns, each of which references an entity set (the Trails and Features tables). A further pair of columns contain locational data which are attributes of the relationship set itself, as the coordinates are determined by the relative location of a particular feature with respect to viewpoints along a particular trail. Since relationships between entities are directly represented as tables, there is no requirement for pointers or linkages between data records to be set up, as was the case with hierarchical or network systems. The principal features of relational databases, the primary key, relational joins and normal forms, are now discussed in turn.

The primary key

The relational approach is firmly grounded in the mathematical theory of relational algebra (Ullman 1982). This has important implications for the design of database tables. Firstly, a set, as mathematically defined, cannot have duplicate values. Since each table or relation represents a set, it cannot, therefore, have any rows whose entire contents are duplicated. Secondly, as each row must be different to every other, it follows that a value in a single column, or a combination of values in multiple columns, can be used to define a primary key for the table, which allows each row to be

uniquely identified. Irrespective of whether the primary key is restricted to one column or spans several, no column that forms part of a key can be null, that is can contain a row location without a value, because this would have the potential for permitting duplicate rows to be stored. The uniqueness property allows the primary key to serve as the sole row level addressing mechanism in the relational database model (Date 1986).

Relational joins

The mechanism for linking data in different tables is called a relational join. Values in a column or columns in one table are matched to corresponding values in a column or columns in a second table. From the second table a further match to a third table can be made, and so on until the necessary data from the requisite number of tables have been retrieved. Matching is frequently based on a primary key in one table linked to a column in the second which is termed a foreign key. An example of the join mechanism is shown in Fig. 18.5. It should be noticed that in the relationship set table for 'Features-visible' the primary key spans the two columns containing id-numbers, because both are required to identify the row uniquely. Each of the two also acts as a foreign key to match to the id-numbers in the entity set tables.

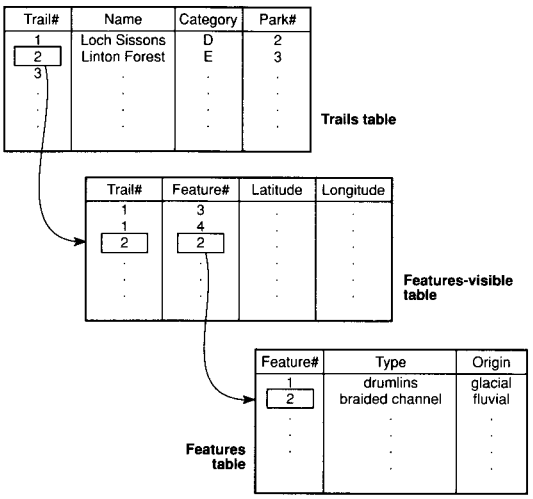

Fig. 18.5 Part of the national parks database in relational form as relational tables and with relational joins (see also Table 18.1).

Normal forms

A certain amount of necessary data redundancy is implicit in the relational model because the join mechanism matches column values between tables. Without careful table design it is all too easy to introduce further unnecessary redundancy into the database. To prevent this, table design should follow Codd's (1970) theory of normal forms, which specifies what types of values columns may contain and how columns in a table are to be dependent on the primary key.

The first requirement of the theory is that all the tables must contain rows and columns as already noted, and column values must be atomic, that is they do not contain repeating groups of data, such as multiple values of a census variable for different years. The second requirement or normal form is that every column, which is not part of the primary key, must be fully dependent on the primary key. This can be understood most readily by considering the example in Fig. 18.6 of a table which is not in second normal form. In this case the feature name is dependent on Feature#, but not Trail#, because feature Type is not an attribute of the 'Features-visible' relationship set, but of the Features entity set. The effect of not meeting the requirement can be seen by the introduction of unnecessary redundancy into the table in rows 1 and 4. If the feature Type was changed in the first row and the corresponding change in the fourth row was overlooked, the database would be left in an inconsistent state.

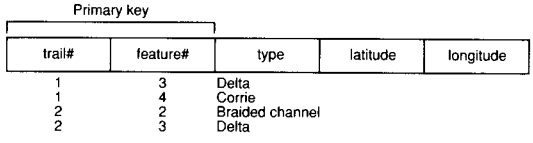

Fig. 18.6 An example of a national parks database table not in second normal form.

The third normal form requires that every non-primary key column must be non-transitively dependent on the primary key. Again, an example which is not in third normal form will be used for illustration (Fig. 18.7(a)). At first sight this may appear to be a convenient way of finding out which vegetation types can be seen in which parks, but the unnecessary redundancy in rows 1 and 3 highlights the error. The Park# is related to the Veg-type#

via the Trail#, that is two distinct relationship sets have been conflated. This transitive dependence is to be avoided (Fig. 18.7(b)). Instead, the relationship set tables should be kept distinct, thereby avoiding the transitive dependence problem (Figs. 18.7(c) and 18.7(d)). This eliminates the redundancy while making it very easy to add new Veg- type/Trail or Trail/Park combinations to the database. Further refinements of the basic normal forms to deal with a variety of design requirements, have led to the identification of Boyce-Codd, fourth and fifth normal forms (Fagin 1979). Nevertheless, the fundamental working rule for most circumstances has been summarized by Kent (1983) as ensuring that each attribute of a table represents a fact about the primary key, the whole primary key and nothing but the primary key. While this is entirely valid from the design viewpoint, it must also be said that practical implementation requirements may, on occasion, override theoretical considerations and lead to tables being merged and de-normalized, usually for performance reasons.

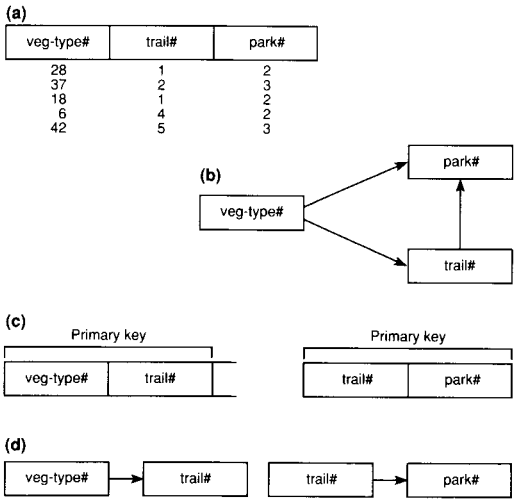

Fig. 18.7 The national parks database in relational form: (a) a table not in third normal form; (b) an illustration of transitive dependence; (c) normalizing the table structure; (d) avoidance of transitive dependence.

Advantages and disadvantages of relational systems

The advantages can be summarized as follows:

- Rigorous design methodology based on sound theoretical foundations.
- All the other database structures can be reduced to a set of relational tables, so they are the most general form of data representation.
- Ease of use and implementation compared to other types of system.
- Modifiability, which allows new tables and new rows of data within tables to be added without difficulty.
- Flexibility in *ad hoc* data retrieval because of the relational join mechanism and powerful query language facilities.

Disadvantages include:

- A greater requirement for processing resources with increasing numbers of users on a given system than with the other types of database.
- On heavily loaded systems, queries involving multiple relational joins may give slower response times than are desirable. This problem can largely be mitigated by effective use of indexing and other optimization strategies, together with the continued improvement in price performance in computing hardware from mainframes to PCs.

The important advantages of the relational approach and the availability of good proprietary software systems such as ORACLE, INGRES and DB2 have contributed greatly to the rapid adoption of this technology, both in the GIS field and automated data processing operations of all other kinds, since the beginning of the 1980s. Relational systems now dominate the market for DBMS in the GIS sector and this will continue for the foreseeable future.

Hybrid and Integrated Approaches to GIS Database Management

With continued developments in database design, storage methods and retrieval performance, it is now quite feasible to hold tens of gigabytes of digital cartographic data, map attribute data or both, using proprietary software and the more powerful hardware platforms available from a variety of vendors. Some examples of large spatial databases are shown in Table 18.2. Sheer data

volume, therefore, may no longer be a severe problem other than for the most demanding requirements of the military or national mapping agencies. Yet, if digital cartographic data, in particular, are to be retrieved from a commercial database for display mapping purposes at the user's screen, it may be necessary to retrieve thousands of rows of data in a small number of seconds, for effective graphical interaction to be possible (Frank 1984, 1988). This remains a problem even for fast workstations, because of the overheads of query language processing, index access and data unpacking that accompany the use of database software. Use of the computer file system directly, rather than through the intermediate step of the DBMS will generally yield faster response times. On the other hand, the DBMS provides a wide range of ready made data manipulation tools so programming effort can be concentrated on algorithms for spatial analysis and user interface requirements.

Table 18.2 Some large spatial databases.

Database	Object types	No. of co-ordinates ($\times 10^6$)	Nature of data
WDDES	polygons, lines	300	contours, rivers, boundaries
SOTER	polygons	150	soil polygons
CORINE	polygons, lines	50	natural resource and political
CGIS-CLI	polygons	90	land use potential
Alberta LRIS	polygons, lines	140	land tenure
Edmonton City	polygons, lines, points	4	urban infrastructure

WDDES = World Digital Data for the Environmental Sciences; SOTER = Soil and Terrain Database; CORINE = Coordinated Information on the European Environment; CGIS = Canada Geographic Information System; LRIS = Land Resources Information System.

The differing emphasis placed by GIS system designers on the advantages of the file system approach versus the database approach for storage of digital map coordinates, has led to the development of two different approaches to implementation, based on either a hybrid or an integrated data model (Bracken and Webster 1989). These will now be examined in greater detail.

The hybrid data model

The starting point for this approach is that data storage mechanisms which are optimal for locational information are not optimal for attribute/thematic information (Morehouse 1985; Aronson 1985). On this basis, digital cartographic data are stored in a set of direct access operating system files for speed of input/output, while attribute data are usually stored in a standard commercial relational type DBMS such as INFO, ORACLE, INGRES or INFORMIX (Fig. 18.8). The GIS software manages linkages between the cartographic files and the DBMS during different map processing operations such as overlay. While a number of different approaches to the storage of the cartographic data are used, the linking mechanism to the database is essentially the same, based on unique identifiers stored in a database table of attributes that allow them to be tied to individual map elements.

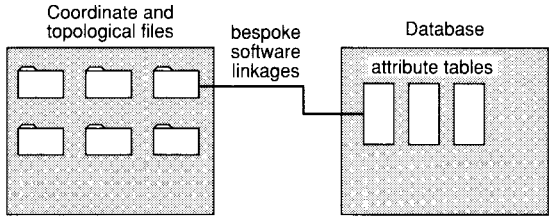

Fig. 18.8 The hybrid GIS model.

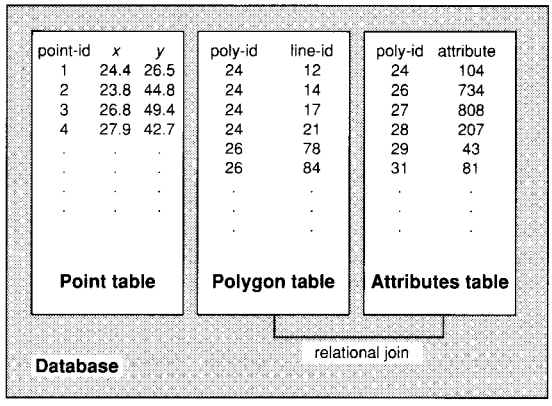

Fig. 18.9 The integrated GIS model with a normalized structure.

Since most hybrid systems use relational databases, any classification of hybrid types must rely largely on differences in the cartographic data storage mechanism. Several types can be identified, including CAD-based, vector–topological and quadtree-based systems. In CAD-based systems, map features are held as graphics elements, but without any topological information, so it is legitimate to enquire as to whether they really fall into the category of GIS rather than digital mapping systems, because spatial analysis cannot readily be performed. Examples of this type would include INTERGRAPH IGDS/DMRS and MICROSTATION-32. The former, it should be noted, uses a network rather than a relational DBMS. Smaller systems that provide links between graphics software such as AUTOCAD and PC DBMS would also fit into this category (cf. Cowen *et al.* 1986). Vector–topological systems may hold the topological map information in a set of linked files very similar to the structure that might be expected if the data were inside, rather than outside, the relational DBMS (Morehouse 1989). Alternatively, a more compact file structure may be used. The ESRI ARC/INFO, GEOVISION and INTERGRAPH MICROSTATION GIS systems are examples of this approach. The last of these provides software to convert graphics-based design files into topologically structured files (Intergraph Corporation 1989a). Quadtree-based systems do not have the same level of representation in the commercial marketplace, at the time of writing, as the previous types, but research using such systems shows considerable promise (Gahegan and Hogg 1986; Gahegan and Roberts 1988). It is also expected that commercial quadtree-based systems such as SPANS will provide linkages to relational databases in the future.

From an initial situation where each GIS supported only one DBMS, a clear trend is now being established to provide access to multiple DBMS systems (McLaren 1990), with vendors such as ESRI, INTERGRAPH and others developing generic relational DBMS interfaces (ESRI 1989; Intergraph Corporation 1989b). Since these DBMS systems are also actively developing a distributed capability, to allow data to be stored transparently on different computer nodes, but accessed and updated as though they were held locally (Date 1985), the potential number of issues raised in relation to system configuration, performance and management is very large indeed. Some of these have been examined by Webster (1988) and Seaborn (1988).

The integrated data model

The integrated data model approach is also described as the spatial database management system approach, with the GIS serving as the query processor sitting on top of the database itself (Guptill 1987; Morehouse 1989). Most implementations to date are of the vector–topological kind, with relational tables holding map coordinate data for points/nodes and line segments, together with other tables containing topological information, in a manner partly similar to that described by van Roessel (1987). Attributes may be stored in the same tables as the map feature database or in separate tables accessible via relational joins (Fig. 18.9).

The integrated data model has a number of implications in terms of the special characteristics of spatial data. From the database viewpoint, it is perfectly possible to store both the coordinates and the topological information required to characterize digital cartographic elements using a design based on Codd's Normal Forms, as van Roessel's detailed analysis has shown. Using this approach, X,Y coordinate pairs for individual vertices along line segments are stored as different rows in a database table. However, from the GIS viewpoint, these data often need only to be accessed as 'bundles', each one comprising all the pairs of coordinates for a given line, or all the line identifiers for the lines forming the bounding edges of polygons. This is the case when the data are being retrieved for display purposes and little or no analysis is being performed on individual coordinate values.

Under these circumstances, storage of individual coordinate pairs in different rows of a database table creates substantial performance overheads, if large amounts of data have to be retrieved quickly for graphics purposes. To achieve satisfactory retrieval performance it has been found necessary to store coordinate strings in long or 'bulk data' columns in tables. This means, however, that the table is strictly no longer in first Normal Form, because each column value is not atomic (Fig. 18.10). This point has been emphasized by independent developments on several different integrated data model systems (Lorie and Meier 1984; Bundock 1987; Charlwood, Moon and Tulip

1987; Waugh and Healey 1987). Empirical evidence of the performance differential is given in Lewis (1987). It has also been shown that retrieving batches of rows containing coordinates using an 'array fetch' mechanism (Dimmick 1985), can reduce the performance overhead while avoiding the use of non-normalized tables (Sinha and Waugh 1988).

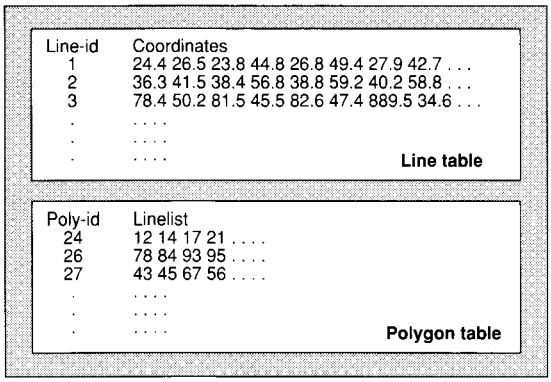

Fig. 18.10 The integrated GIS model using long data types.

The performance issue must also be addressed in respect of very large digital cartographic data banks, since both the US Geological Survey (Guptill 1986; Starr and Anderson 1991 in this volume) and the Ordnance Survey (Smith 1987; Sowton 1991 in this volume) have developed integrated data models for national mapping applications, suitable for implementation in a relational database framework. The Ordnance Survey solution has been to implement its system using a Britton-Lee IDM dedicated database computer. Machines of this kind have hardware support to accelerate standard operations such as relational joins and can support a much larger number of concurrent data accesses by multiple users than a software only database system (Su 1988).

A further aspect of handling large spatial databases is the need to convert 2-D coordinate information into 1-D spatial keys that can be stored as database table columns. These can then be indexed in the normal way and used for fast retrieval of map elements contained within or overlapping a specified geographical search area (Abel and Smith 1986; Waugh and Healey 1987; Abel 1988).

Comparison of hybrid and integrated data model approaches

From the commercial viewpoint, the success of hybrid systems cannot be denied, while integrated systems such as Prime's SYSTEM9, although developed later, have yet to make a comparable impact on the market. In the past, the problem for integrated systems may have been a function of the powerful hardware required to achieve good performance and the correspondingly high cost of entry. With the rapid growth of the graphics workstation market and technological improvements this factor is rapidly declining in importance. Much more significant for the future, when comparing hybrid and integrated approaches, will be the rapid expansion of networked and distributed computing. This increases the problems of multi-user data access, data security, data integrity and overall database management (Bundock 1987) for which the DBMS vendors, as opposed to GIS vendors, are well advanced in attempting to provide solutions. In hybrid systems, special purpose programming is currently used to link the digital cartographic and attribute data, instead of the standard relational join mechanism used in an integrated system. Extending the argument, there is a danger that a great deal of further specialist programming will have to be done, for a distributed hybrid GIS to attain the functionality that can already be provided by the leading DBMS packages. The performance advantage of hybrid systems remains, but it can be expected that integrated systems will progressively close the gap by the use of memory caching techniques, bulk data types and array fetching mechanisms, to minimize database input/output. Further improvements in determining optimal search paths for database queries, particularly when distributed processing is involved, can also be anticipated.

Both hybrid and integrated approaches can, therefore, be expected to coexist for the foreseeable future, but with a growing appreciation of the advantages of the integrated model, as the price–performance of hardware continues to improve.

SPATIAL QUERY LANGUAGES

Implicit in the spatial database approach is a requirement to query and interrelate different data

sets in a manner that is meaningful in the GIS context. Unfortunately, standard query languages like the Structured Query Language, SQL, are restricted to supporting queries based on relational joins, sub-queries, grouping functions and query combination operators. Some types of GIS function, such as retrieving map elements that fall within or overlap a rectangular window, can be performed using standard, albeit complex SQL queries. Others, such as data layer intersection (overlay), cannot, because they require operations to be applied to the data which are beyond the scope of the available query language functions. A summary of the range of operators required for spatial query languages is given by Guptill (1986). There are now several systems where the query language has been extended to incorporate such operators (Charlwood, Moon and Tulip 1987; Ingram and Phillips 1987; Herring, Larsen and Shivakumar 1988). With relational DBMS systems there are different ways in which such facilities can be implemented. One approach is to pre-process non-standard SQL and convert, if possible, the spatial operators into more complex but standard SQL, which is then passed to the database kernel. A second approach, at the pre-processing stage, is to convert the query into a component that can be used to retrieve data from the DBMS. These data are then further processed by calls to user-supplied code which perform the GIS operations. A third approach is to link the user-supplied code directly to the database kernel, which will render the developer unpopular with the vendor, but will enable the spatial operators to become fully part of the query language!

OBJECT-ORIENTED DBMS: THE FUTURE FOR GIS DATABASE MANAGEMENT?

Extensions of the integrated model to incorporate spatial query language functions are a tacit recognition that, in the final analysis, it is not sufficient merely to hold data on map elements in the database. For GIS purposes it must also be possible to access the operations to be performed on these elements. This brings the conceptualization of the problem very close to that provided by the object-oriented approach (Aronson 1987).

As the most recent of the models for data handling and processing discussed here, care needs to be taken with the term 'object oriented' (Maguire, Worboys and Hearnshaw 1990). Even some of the main proponents are reluctant to use it too widely, since 'few people agree on exactly what it means' (Rowe and Stonebraker 1987). Following Somerville (1989) and Rowe (1986) an object can be defined as an entity that has a state represented by the values of local variables (instance variables) and a set of operations or methods (instance methods) that operate on the object. Individual objects belong to a class that defines the type of object. Classes may have variables that describe characteristics of the class as a whole. Each class has a superclass from which it can inherit both instance variables and methods. For example, an object class called polygon may be defined which is also the superclass for another class called land parcel. The object definitions might, therefore, appear as indicated in Table 18.3(a) and 18.3(b). By convention, the superclass (object) is at the top of the object hierarchy, which might be structured as in Fig. 18.11. All the instance methods and variables of the polygon superclass are inherited by the land parcel class, unless they are re-defined at the land parcel level.

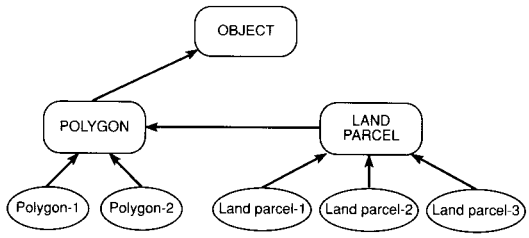

Fig. 18.11 A hierarchy of object classes.

The designers of POSTGRES, an object-oriented DBMS designed to be the successor to INGRES, the relational DBMS, have developed a variety of techniques to facilitate object management within a database environment. These include storage of the structure of object hierarchies in relational tables, inheritance of instance variables and methods, storage of query language or programming language procedures representing instance methods, as special fields in relational tables, and support for abstract/user defined data types and the operations supported on them. It should be noted that the POSTGRES designers

Table 18.3(a) Polygon object definition.

Superclasses (object)
Class variables
Number_of_polygons
Instance variables
List_of_nodes
List_of_arcs
Area
Instance methods
Calculate_centroid
Draw
Overlay

Table 18.3(b) Land parcel object definition.

Superclasses (polygon)
Class variables
Instance variables
Value
Owner
Instance methods
Transfer_ownership
Re_zone

regard an evolutionary approach as the best way forward, that is adding object-oriented facilities to an existing relational database framework. This is in contrast to some of the proponents of object-oriented programming, who wish to achieve the same kind of functionality, but without the perceived constraints of the relational model (Cox 1986; Wells 1988; Stroustrup 1988). Other authorities expect to see convergence of the programming and database viewpoints in future (Tsichritzis and Nierstrasz 1988).

The major example of the object-oriented approach in the GIS field is the INTERGRAPH TIGRIS system (Herring 1987) which utilizes object-oriented programming, rather than object-oriented DBMS techniques or an object-oriented interface. Further work in this area has been reported, in terms of user interface aspects (Egenhofer and Frank 1988), data modelling (Kemp 1990; Orenstein 1990; Worboys, Hearnshaw and Maguire 1990a) and query modelling (Worboys, Hearnshaw and Maguire 1990b). It is clear, however, that a fuller exploration of the potential of object-oriented DBMS systems, as the organizing framework for spatial databases, may have to wait until they move further from the 'proof-of-concept' stage (Thatte 1988) towards widespread availability.

CONCLUSIONS

Evolving trends in the design and implementation of general purpose database management systems have been examined, with regard to their significance for the handling of both digital cartographic and attribute data. The leading role of relational methods has been identified. The fact that their widespread adoption in the data processing world at large coincided with the rapid upsurge of interest in GIS in the 1980s, has undoubtedly contributed to their success. Extensions to the relational model and query languages to handle both spatial operators and object orientation are already well advanced. Further developments in the handling of temporal attributes in spatial databases will assume greater importance in future, as data volumes grow and database update in mature systems takes over from database creation (Snodgrass 1987; Rowe and Stonebraker 1987; Langran 1988, 1989; Price 1989).

Finally, across the entire spectrum of database applications there is the possibility of using methods from knowledge-based systems and natural language processing; but these are discussed by Smith and Ye Jiang (1991 in this volume). In an analogous manner to object-oriented methods, however, it remains uncertain as to whether an extended relational or other type of DBMS should act as the repository for the knowledge base (Tsichritzis and Nierstrasz 1988). Past history suggests that while in the research phase a variety of approaches will vie for position, subsequently market pressures will enforce a concentration on standardized solutions, characterized by maintainability and simplicity of structure.

REFERENCES

Abel D J (1988) Relational data management facilities for spatial information systems. *Proceedings of the 3rd*

International Symposium on Spatial Data Handling. International Geographical Union, Columbus Ohio, pp. 9–18
Abel D J, Smith J L (1986) A relational GIS database accommodating independent partitionings of the region. *Proceedings of the 2nd International Symposium on Spatial Data Handling*. International Geographical Union, Columbus Ohio, pp. 213–24
Armstrong M P, Densham P J, Rushton G (1986) Architecture for a microcomputer based spatial decision support system. *Proceedings of the 2nd International Symposium on Spatial Data Handling*. International Geographical Union, Columbus Ohio, pp. 120–31
Aronson P (1985) Applying software engineering to a general purpose geographic information system. *Proceedings of AUTOCARTO 7*. ASPRS, Falls Church Virginia, pp. 23–31
Aronson P (1987) Attribute handling for geographic information systems. *Proceedings of AUTOCARTO 8*. ASRPS, Falls Church Virginia, pp. 346–55

Bracken I, Webster C (1989) Towards a typology of geographical information systems. *International Journal of Geographical Information Systems* **3**: 137–52
Bundock M (1987) An integrated DBMS approach for geographic information systems. *Proceedings of AUTOCARTO 8*. ASRPS, Falls Church Virginia, pp. 292–301

Charlwood G, Moon G, Tulip J (1987) Developing a DBMS for geographic information: a review. *Proceedings of AUTOCARTO 8*. ASPRS, Falls Church Virginia, pp. 302–15
Chen P (1976) The Entity–Relationship Model – towards a unified view of data. *Association for Computing Machinery Transactions on Database Systems* **1** (1): 9–36
Clark D M, Hastings D A, Kineman J J (1991) Global databases and their implications for GIS. In: Maguire D J, Goodchild M F, Rhind D W (eds.) *Geographical Information Systems: principles and applications*. Longman, London, pp. 217–31, Vol 2
Codd E F (1970) A relational model of data for large shared data banks. *Communications of the Association for Computing Machinery* **13** (6): 377–87
Codd E F (1979) Extending the database relational model to capture more meaning. *Association for Computing Machinery Transactions on Database Systems* **4** (4): 397–434
Collins J (1982) *Review of Competitive Database Software*. Savant, Carnforth
Cowen D J, Hodgson M, Santure L, White T (1986) Adding topological structure to PC-based CAD databases. *Proceedings of the 2nd International Symposium on Spatial Data Handling*. International Geographical Union, Columbus Ohio, pp. 132–41
Cox B J (1986) *Object-Oriented Programming: An evolutionary approach*. Addison-Wesley, Reading. Massachusetts. 274pp.
Crain I K (1990) Extremely large spatial information systems: a quantitative perspective. *Proceedings of the 4th International Symposium on Spatial Data Handling*, Volume 2. International Geographical Union, Columbus Ohio, pp. 632–41

Date C J (1985) *An Introduction to Database Systems*. Volume II. Addison-Wesley, Reading Massachusetts
Date C J (1986) *An Introduction to Database Systems*. 2nd edn. Addison-Wesley, Reading Massachusetts
Densham P J, Armstrong M (1987) A spatial decision support system for locational planning: design, implementation and operation. *Proceedings of AUTOCARTO 8*. ASPRS, Falls Church Virginia, pp. 112–21
Dimmick S (1985) *Pro-Fortran User Guide*. Oracle Corporation, Menlo Park California

ESRI (1989) *ARC/INFO V5.0 Users Guide*, Volumes I and II. ESRI Inc., Redlands California
Egenhofer M J, Frank A U (1988) Designing object-oriented query languages for GIS: human interface aspects. *Proceedings of the 3rd International Symposium on Spatial Data Handling*. International Geographical Union, Columbus Ohio, pp. 79–96
Ellis H (1985) Twenty years of data analysis. In: Holloway S (ed.) *Data Analysis in Practice*. Database Specialist Group, The British Computer Society, London, pp. 99–120

Fagin R (1979) Normal forms and relational database operations. *Proceedings of the ACM SIG-MOD International Conference on Management of Data*, pp. 153–60
Frank A U (1984) Requirements for database systems suitable to manage large spatial databases. *Proceedings of the 1st International Symposium on Spatial Data Handling*, Volume 1. International Geographical Union, Columbus Ohio, pp. 38–60
Frank A U (1988) Requirements for a database management system for a GIS. *Photogrammetric Engineering and Remote Sensing* **54** (11): 1557–64

Gahegan M N, Hogg J (1986) A pilot geographical information system based on linear quadtrees and a relational database for regional analysis. In: Diaz B M, Bell S B M (eds.) *Spatial Data Processing Using Tesseral Methods*. Natural Environment Research Council, Swindon, p. 213–32
Gahegan M N, Roberts S A (1988) An intelligent, object-oriented geographical information system. *International Journal of Geographical Information Systems* **2**: 101–10
Guptill S C (1986) A new design for the US Geological Survey's National Digital Cartographic Database. In: Blakemore M J (ed.) *Proceedings of AUTOCARTO LONDON*, Volume 2. Royal Institution of Chartered Surveyors, London, pp. 10–18
Guptill S C (1987) Desirable characteristics of a spatial database management system. *Proceedings of AUTOCARTO 8*. ASPRS, Falls Church Virginia, pp. 278–281

Hearnshaw H M, Maguire D J, Worboys M F (1989) An introduction to area-based spatial units: a case study of Leicestershire. *Midlands Regional Research Laboratory Research Report 1*. MRRL, Leicester
Herring J R (1987) TIGRIS: topologically integrated geographic information system. *Proceedings of AUTOCARTO 8*. ASPRS, Falls Church Virginia, pp. 282–91
Herring J R, Larsen R C, Shivakumar J (1988) Extensions to the SQL query language to support spatial analysis in a topological database. *Proceedings of GIS/LIS '88*, Volume 2, ASPRS/ACSM, Falls Church Virginia, pp. 741–50
Howe D R (1985) *Data Analysis for Data Base Design*. Edward Arnold, London

Ingram K, Phillips W (1987) Geographic information processing using a SQL-based query language. *Proceedings of AUTOCARTO 8*. ASPRS, Falls Church Virginia, pp. 326–35
Intergraph Corporation (1989a) *Microstation Analyst (MGA) Reference Manual*. Intergraph Corporation, Huntsville Alabama
Intergraph Corporation (1989b) *Relational Interface (RIS) User Reference Manual*. Intergraph Corporation, Huntsville Alabama

Kemp Z (1990) An object-oriented data model for spatial data. *Proceedings of the 4th International Symposium on Spatial Data Handling*, Volume 2. International Geographical Union, Columbus Ohio, pp. 659–68
Kent W (1983) A simple guide to five normal forms in relational database theory. *Communications of the Association for Computing Machinery* **26** (2): 120–25

Langran G (1988) Temporal GIS design tradeoffs. *Proceedings of GIS/LIS '88*, Volume 2. ASPRS/ACSM, Falls Church Virginia, pp. 890–99
Langran G (1989) A review of temporal database research and its use in GIS applications. *International Journal of Geographical Information Systems* **3**: 215–32
Lewis P (1987) Spatial data handling using relational databases. Unpublished MSc thesis, Department of Geography, University of Edinburgh, Scotland.
Lorie R A, Meier A (1984) Using a relational DBMS for geographical databases. *Geo-Processing* **2**: 243

Maguire D J, Worboys M F, Hearnshaw H M (1990) An introduction to object-oriented geographical information systems. *Mapping Awareness* **4** (2): 36–9
Martin J (1976) *Principles of Database Management*. Prentice-Hall, Englewood Cliffs New Jersey
Martin J (1983) *4th Generation Languages*, Volume 1. Savant, Carnforth Lancashire
McLaren R A (1990) Establishing a corporate GIS from component data sets – the database issues. *Mapping Awareness* **4** (2): 52–8
Morehouse S (1985) ARC/INFO: a geo-relational model for spatial information. *Proceedings of AUTOCARTO 8*. ASPRS, Falls Church Virginia, pp. 388–97
Morehouse S (1989) The architecture of ARC/INFO. *Proceedings of AUTOCARTO 9*. ASPRS, Falls Church Virginia, pp. 266–77

Olle T W (1978) *The Codasyl Approach to Database Management*. Wiley, Chichester. 287pp.
Orenstein J A (1990) An object-oriented approach to spatial data processing. *Proceedings of the 4th International Symposium on Spatial Data Handling*, Volume 2. International Geographical Union, Columbus Ohio, pp. 669–78

Price S (1989) Modelling the temporal element in land information systems. *International Journal of Geographical Information Systems* **3**: 233–44

Rowe L A (1986) A shared object hierarchy. In: Stonebraker M R, Rowe L A (eds.) *The POSTGRES Papers. Memorandum No. UCB/ERL M86/85*. College of Engineering, University of California, Berkeley
Rowe L A, Stonebraker M R (1987) The progres data model. *Proceedings of the 13th Conference on very large databases*, Brighton, England, pp. 83–96

Seaborn D W (1988) Distributed processing and distributed databases in GIS – separating hype from reality. *Proceedings of GIS/LIS '88*, Volume 1. ASPRS/ACSM, Falls Church Virginia, pp. 141–4
Sinha A K, Waugh T C (1988) Aspects of the implementation of the GEOVIEW design. *International Journal of Geographical Information Systems* **2**: 91–100
Smith N S (1987) Data models and data structures for Ordnance Survey. *Proceedings of the Ordnance Survey/SORSA Symposium, Durham, May 1987*.
Smith T R, Ye Jiang (1991) Knowledge-based approaches in GIS. In: Maguire D J, Goodchild M F, Rhind D W (eds.) *Geographical Information Systems: principles and applications*. Longman, London, pp. 413–25, Vol 1
Snodgrass R (1987) The temporal query language TQUEL. *Association for Computing Machinery Transactions on Database Systems* **12**: 247
Somerville I (1989) *Software Engineering*, 3rd edn. Addison-Wesley, Reading Massachusetts, 653pp.
Sowton M (1991) Development of GIS-related activities at the Ordnance Survey. In: Maguire D J, Goodchild M F, Rhind D W (eds.) *Geographical Information Systems: principles and applications*. Longman, London, pp. 23–38, Vol 2
Starr L E, Anderson K E (1991) A USGS perspective on GIS. In: Maguire D J, Goodchild M F, Rhind D W (eds.) *Geographical Information Systems: principles and applications*. Longman, London, pp. 11–22, Vol 2
Stroustrup B (1988) What is Object-Oriented Programming? *IEEE Software* **5** (3): 10–20
Su S (1988) *Database Computers: principles, architectures and techniques*. McGraw-Hill, New York, 497pp.

Thatte S (1988) Report on the object-oriented database workshop: implementation aspects. In: Power L, Weiss Z (eds.) *OOPSLA '87 Addendum to the Proceedings. Special Issue of SIGPLAN Notices* **23** (5): 87

Tsichritzis D C, Lochovsky L C (1982) *Data Models*. Prentice-Hall, Englewood Cliffs New Jersey
Tsichritzis D C, Nierstrasz O M (1988) Fitting round objects into square databases. In: Gjessing S, Nygaard K (eds.) *Proceedings of ECOOP '88, the European Conference on Object-Oriented Programming*. Springer-Verlag, Berlin, pp. 283–99
Tuori M, Moon G C (1984) A topographic map conceptual data model. *Proceedings of the 1st International Symposium on Spatial Data Handling*, Volume 1, International Geographical Union, Columbus Ohio, pp. 28–37

Ullman J D (1982) *Principles of Database Systems*. Computer Science Press, Rockville Maryland

van Roessel J W (1987) Design of a spatial data structure using the relational normal forms. *International Journal of Geographical Information Systems* **1**: 33–50
van Roessel J W, Fosnight E A (1984) A relational approach to vector data structure conversion. *Proceedings of the 1st International Symposium on Spatial Data Handling*, Volume 1. International Geographical Union, Columbus Ohio, pp. 78–95

Waugh T C, Healey R G (1987) The GEOVIEW design. A relational database approach to geographical data handling. *International Journal of Geographical Information Systems* **1**: 101–18
Webster C (1988) Disaggregated GIS architecture. Lessons from recent developments in multi-site database management systems. *International Journal of Geographical Information Systems* **2**: 67–80
Wells D (1988) How object-oriented databases are different from relational databases. In: Power L, Weiss Z (eds.) *OOPSLA '87 Addendum to the Proceedings. Special Issue of SIGPLAN Notices* **23** (5): 81
Wiederhold G (1983) *Database Design*. 2nd edn. McGraw-Hill, London, 751pp.
Worboys M F, Hearnshaw H, Maguire D J (1990a) Object-oriented data modelling for spatial databases. *International Journal of Geographical Information Systems*. 369–83
Worboys M F, Hearnshaw H, Maguire D J (1990b) Object-oriented data and query modelling for geographical information systems. *Proceedings of the 4th International Symposium on Spatial Data Handling*. International Geographical Union, Columbus Ohio, pp. 679–88