26

Relational databases and beyond

M F WORBOYS

This chapter introduces the database perspective on geospatial information handling. It begins by summarising the major challenges for database technology. In particular, it notes the need for data models of sufficient complexity, appropriate and flexible human-database interfaces, and satisfactory response times. The most prevalent current database paradigm, the relational model, is introduced and its ability to handle spatial data is considered. The object-oriented approach is described, along with the fusion of relational and object-oriented ideas. The applications of object-oriented constructs to GIS are considered. The chapter concludes with two recent challenges for database technology in this field: uncertainty and spatio-temporal data handling.

1 INTRODUCTION TO DATABASE SYSTEMS

1.1 The database approach

Database systems provide the engines for GIS. In the database approach, the computer acts as a facilitator of data storage and sharing. It also allows the data to be modified and analysed while in the store. For a computer system to be an effective data store, it must have the confidence of its users. Data owners and depositors must have confidence that the data will not be used in unauthorised ways, and that the system has fail-safe mechanisms to cope with unforeseen events. Both data depositors and data users must be assured that, as far as possible, the data are correct. There should be sufficient flexibility to give different classes of users different types of access to the store. Most users will not be concerned with how the database works and should not be exposed to low-level database mechanisms. Data retrievers need flexible methods for establishing what is in the store and for retrieving data according to their requirements and skills. Users may have different conceptions of the organisation of the data in the store. The database interface should be sufficiently flexible to respond equally well to both single-time users with unpredictable and varied requirements, and to regular users with little variation in their requirements. Data should be retrieved as effectively as possible. It should be possible for users to link pieces of information together in the database to get the benefit of the added value from making the connections. Many users may wish to use the store, maybe even the same data, at the same time and this needs to be controlled. Data stores may need to be linked to other stores for access to pieces of information not in their local holdings.

1.2 Database history

Database management systems have grown out of file management systems that perform basic file handling operations such as sorting, merging, and report generation. During the 1950s, as files grew to have increasingly complex structures, an assortment of data definition products came into use. These became standardised by the Conference on Data Systems and Languages (CODASYL) in 1960 into the Common Business-Oriented Language, that is the COBOL programming language, which separates the definition on file structure from file manipulation. In 1969, the DataBase Task Group (DBTG) of CODASYL gave definitions for data description and definition languages, thus paving the way for hierarchal and network database management systems. The underlying model for these systems is navigational, that is connections between records are made by navigating explicit relationships between them. These relationships were 'hard-wired' into the database, thus limiting the degree to which such databases could be extended or distributed to other groups of users.

The acknowledged founder of relational database technology is Ted Codd, who in a pioneering paper (Codd 1970) set out the framework of the relational model. The 1970s saw the advent of relatively easy-to-use relational database languages such as the Structured Query Language, SQL, originally called the Structured English Query Language, SEQUEL (Chamberlin and Boyce 1974) and Query Language, QUEL (Held et al 1975), as well as prototype relational systems such as IBM's System R (Astrahan et al 1976) and University of California at Berkeley's Interactive Graphics and Retrieval Systems, INGRES (Stonebraker et al 1976).

From the latter part of the 1970s, shortcomings of the relational model began to become apparent for particular applications, including GIS. Codd himself provided extensions to incorporate more semantics (Codd 1979). Object-oriented notions were introduced from programming languages into databases, culminating in prototype object-oriented database systems, such as O_2 (Deux 1990) and ORION (Kim et al 1990). Today, object-oriented systems are well established in the marketplace, as are object-oriented extensions of relational systems, which may be where the future really is. Early developments in object-relational systems are described in Haas et al (1990) and Stonebraker (1986). SQL has developed into the international standard SQL-92, and SQL3 is being developed.

1.3 Data models

The data model provides a collection of constructs for describing and structuring applications in the database. Its purpose is to provide a common computationally meaningful medium for use by system developers and users. For developers, the data model provides a means to represent the application domain in terms that may be translated into a design and implementation of the system. For the users, it provides a description of the structure of the system, independent of specific items of data or details of the particular implementation.

A clear distinction should be made between data models upon which database systems are built, for example the relational model, and data models whose primary roles are to represent the meaning of the application domains as closely as possible (so-called *semantic data models*, of which entity-relationship modelling is an example: see also Martin, Chapter 6; Raper, Chapter 5). It might be that a semantic data model is used to develop applications for a database system designed around another model: the prototypical example of this is the use of the entity-relationship model to develop relational database applications. The three currently most important data modelling approaches are *record-based*, *object-based* and *object-relational*.

1.4 Human database interaction

Humans need to interact with database systems to perform the following broad types of task:

1 *Data definition*: description of the conceptual and logical organisation of the database, the database schema;
2 *Storage definition*: description of the physical structure of the database, for example file location and indexing methods;
3 *Database administration*: daily operation of the database;
4 *Data manipulation*: insertion, modification, retrieval, and deletion of data from the database.

The first three of these tasks are most likely to be performed by the database professional, while the fourth will be required by a variety of user types possessing a range of skills and experience as well as variable needs requirements in terms of frequency and flexibility of access.

User interfaces are designed to be flexible enough to handle this variety of usage. Standard methods for making interfaces more natural to users include menus, forms, and graphics (windows, icons, mice: see Egenhofer and Kuhn, Chapter 28; Martin 1996). Natural language would be an appropriate means of communication between human and database, but successful interfaces based on natural language have not yet been achieved. For spatial data, the graphical user interface (GUI) is of course highly appropriate. Specialised query languages for database interaction have been devised.

1.5 Database management

The software system driving a database is called the *database management system* (DBMS). Figure 1 shows schematically the place of some of these components in the processing of an interactive query, or an application program that contains within the host general-purpose programming language some database access commands. The DBMS has a query compiler that will parse and analyse a query and, if all is correct, generate execution code that is passed to the runtime database processor. Along the way, the compiler may call the query optimiser to optimise the code so that performance on the retrieval is improved. If the database language expression had been embedded in a general-purpose computer language such as C++, then an earlier precompiler stage would be needed. To retrieve the required data from the database, mappings must be made between the high-level objects in the query language statement and the physical location of the data on the storage device. These mappings are made using the system catalogue. Access to DBMS data is handled by the stored data manager, which calls the operating system for control of physical access to storage devices.

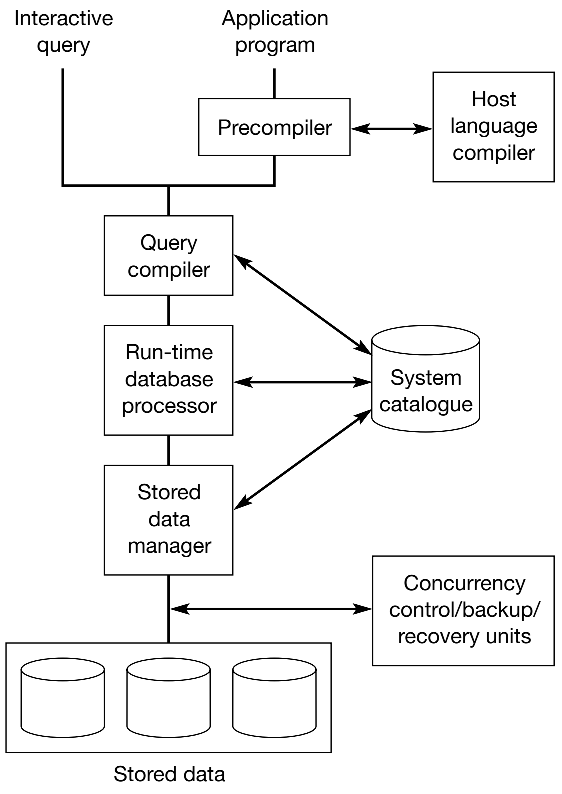

Fig 1. DBMS components used to process user queries.

The logical atom of interaction with a database is the *transaction*, broadly classified as *create*, *modify* (*update*), and *delete*. Transactions are either executed in their entirety (committed) or not at all (rollback to previous commit). The sequence of operations contained in transactions is maintained in a system log or journal, hence the ability of the DBMS to roll back. When a ‘commit’ is reached, all changes since the last commit point are then made permanent in the database. Thus, a transaction may be thought of as a unit of recovery. The DBMS seeks to maintain the so-called ACID properties of transactions: Atomicity (all-or-nothing), Consistency (of the database), Isolation (having no side-effects and unforeseen effects on other concurrent transactions), and Durability (ability to survive even after system crash).

2 RECORD-BASED DATA MODELS: RELATIONAL DATABASES

2.1 Introduction to the relational model

A record-based model structures the database as a collection of files of fixed-format records. The records in a file are all of the same record type, containing a fixed set of fields (attributes). The early network and hierarchical database systems, mentioned earlier, conform to the record-based data model. However, they proved to be too closely linked to physical implementation details, and they have been largely superseded by the relational model.

A relational database is a collection of tabular relations, each having a set of attributes. The data in a relation are structured as a set of rows. A row, or *tuple*, consists of a list of values, one for each attribute. An attribute has associated with it a domain, from which its values are drawn. Most current systems require that values are atomic – for example they cannot be decomposed as lists of further values – so a single cell in a relation cannot contain a set, list or array of values. This limits the possibilities of the pure relational model for GIS.

A distinction is made between a *relation schema*, which does not include the data but gives the structure of the relation (its attributes, their corresponding domains, and any constraints on the data) and a *relation*, which includes the data. The relation schema is usually declared when the database is set up and then remains relatively

Table 1 Tuples from the Country relation.

Name	*Population (millions)*	*Land area (thousand sq. miles)*	*Capital*
Austria	8	32	Vienna
Germany	81	138	Berlin
Italy	58	116	Rome
France	58	210	Paris
Switzerland	7	16	Bern

Table 2 Tuples from the City relation.

Name	*Country*	*Population (thousands)*
Vienna	Austria	1500
Berlin	Germany	3400
Hamburg	Germany	1600
Rome	Italy	2800
Milan	Italy	1400
Paris	France	2100
Zurich	Switzerland	300
Bern	Switzerland	100

unaltered during the lifespan of the system. A relation, however, will typically be changing frequently as data are inserted, modified and deleted. A *database schema* is a set of relation schemata and a *relational database* is a set of relations, possibly with some constraints. An example of a database schema, used throughout this chapter, comprises two relations Country and City, along with their attributes, as shown:

Country (Name, Population, Land Area, Capital)
City (Name, Country, Population).

Tables 1 and 2 show part of an example database according to this schema. Each row of a relation in a relational database is sometimes called a tuple.

The primitive operations that can be supported by a relational database are the traditional set operations of union, intersection, and difference, along with the characteristically relational operations of project, restrict, join, and divide. The structure of these operations and the way that they can be combined is provided by relational algebra, essentially as defined by Codd (1970). The set operations union, intersection, and difference work on the relations as sets of tuples. The project operation applies to a single relation and returns a new relation that has a subset of attributes of the original. For example, Table 3 shows the Country relation projected onto its Name and Population attributes. The restrict operation acts on a relation to return only those tuples that satisfy a given condition. For example, Table 4 shows a restriction of the City relation, retrieving from the City relation those tuples containing cities with populations greater than two million. The join operation makes connections between relations, taking two relations as operands, and returns a single relation. The relation shown in Table 5 is a join of the Country and City relations, matching tuples when they have the same city names.

Table 3 Tuples from the Country relation after a project operation.

Name	*Population (millions)*
Austria	8
Germany	81
Italy	58
France	58
Switzerland	7

Table 4 Tuples from the City relation after a restrict operation.

Name	*Country*	*Population (thousands)*
Berlin	Germany	3400
Rome	Italy	2800
Paris	France	2100

Table 5 Tuples from the joined Country and City relations.

Name	*Country population (millions)*	*Land area (thousand sq. miles)*	*Capital*	*Country city*	*City population (thousands)*
Austria	8	32	Vienna	Austria	1500
Germany	81	138	Berlin	Germany	3400
Italy	58	116	Rome	Italy	2800
France	58	210	Paris	France	2100
Switzerland	7	16	Bern	Switzerland	100

2.2 Relational database interaction and SQL

From the outset, there has been a collection of specialised query languages for database interaction. For relational databases, the Structured or Standard Query Language (SQL) is a *de facto* and *de jure* standard. SQL may either be used on its own as a means of direct interaction with the database, or may be embedded in a general-purpose programming language. The most recent SQL standard is SQL-92 (also called SQL2: ISO 1992). There is a large effort to move forward to SQL3.

2.2.1 Schema definition using SQL

The data definition language component of SQL allows the creation, alteration, and deletion of relation schemata. It is usual that a relation schema is altered only rarely once the database is operational. A relation schema provides a set of attributes, each with its associated data domain. SQL allows the definition of a domain by means of a `CREATE DOMAIN` expression.

A relation schema is created by a `CREATE TABLE` command as a set of attributes, each associated with a domain, with additional properties relating to keys and integrity constraints. For example, the relation schema City may be created by the command:

```
CREATE TABLE    City
(Name           PlaceName,
Country         PlaceName,
Population      Population,
PRIMARY KEY     (Name)
```

This statement begins by naming the relation schema (called a table in SQL) as City. The attributes are then defined by giving each its name and associated domain (assuming that we have already created domains PlaceName and Population). The primary key, which serves to identify a tuple uniquely, is next given as the attribute Name. There are also SQL commands to alter a relation schema by changing attributes or integrity constraints and to delete a relation schema.

2.2.2 Data manipulation using SQL

Having defined the schemata and inserted data into the relations, the next step is to retrieve data. A simple example of SQL data retrieval resulting in the relation in Table 4 is:

```
SELECT *
FROM City
WHERE Population > 2000000
```

The `SELECT` clause indicates the attribute to be retrieved from the City relation (* indicates all attributes), while the `WHERE` clause provides the restrict condition. Relational joins are effected by allowing more than one relation (or even the same relation called twice with different names) in the `FROM` clause. For example, to find names of countries whose capitals have a population less than two million people, use the expression:

```
SELECT Country.Name
FROM Country, City
WHERE Country.Capital = City. Name
AND City. Population < 2000000
```

In this case, the first part of the `WHERE` clause provides the join condition by specifying that tuples from the two tables are to be combined only when the values of the attributes Capital in Country and Name in City are equal. Attributes are qualified by prefixing the relation name in case of any ambiguity.

Most of the features of SQL have been omitted from this very brief summary. The documentation on the SQL2 standard is about 600 pages in length. The reader is referred to Date (1995) for a good survey of the relational model and SQL2.

2.3 Relational technology for geographical information

There are essentially two ways of managing spatial data with relational technology: putting all the data (spatial and non-spatial) in the relational database (integrated approach), or separating the spatial from the non-spatial data (hybrid approach). The benefits of using an integrated architecture are considerable, allowing a uniform treatment of all data by the DBMS, and thus not consigning the spatial data to a less sheltered existence outside the database, where integrity, concurrency, and security may not be so rigorously enforced. In theory, the integrated approach is perfectly possible: for example, Roessel (1987) provides a relational model of configurations of nodes, arcs, and polygons. However, in practice the pure relational geospatial model has not up to now been widely adopted because of unacceptable performance (Healey 1991). Essentially, problems arise because of:

1 slow retrieval due to multiple joins required of spatial data in relations;
2 inappropriate indexes and access methods, which are provided primarily for 1-dimensional data types by general-purpose relational systems;

3 lack of expressive power of SQL for spatial queries.

The first problem arises because spatial data are fundamentally complex – polygons being sequences of chains, which are themselves sequences of points. The object-oriented and extended relational models are much better able to handle such data types. With regard to the second problem, extended relational models allow much more flexibility in declaring indexes for different types of data. For the third problem, the limitations of SQL have been apparent for some time in a number of fields (for example, CAD/CAM, GIS, multimedia databases, office information systems, and text databases). SQL3, currently being developed as a standard, promises much in this respect.

3 OBJECT-BASED DATA MODELS

3.1 Introduction and the entity–relationship–attribute approach

The primary components of an object-based model are its *objects* or *entities*. The entity–relationship–attribute (ERA) model and the object-oriented (OO) models are the two main object-based modelling approaches. The ERA approach is attributed to Chen (1976) and has been a major modelling tool for relational database systems for about 20 years. In the ERA approach, an entity is a semantic data modelling construct and is something (such as a country) that has an independent and uniquely identifiable existence in the application domain. Entities are describable by means of their attributes (for example, the name, boundary, and population of a country). Entities have explicit relationships with other entities. Entities are grouped into entity types, where entities of the same type have the same attribute and relationship structure. The structure of data in a database may be represented visually using an ERA diagram. Figure 2 shows an ERA diagram representing the structure of these data in the example database schema in Tables 1 and 2. Entity types are represented by rectangles with offshoot attributes and connecting edges showing relationships. The ERA approach is fully discussed by Bédard (Chapter 29), and so is not considered further here.

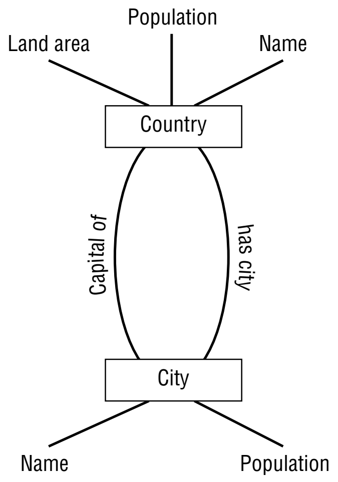

Fig 2. Example of an ERA diagram.

3.2 The object-oriented approach

3.2.1 Objects, classes, encapsulation, and identity

For many application domains, including GIS, ERA modelling has proved too limited and is being superseded by the OO approach. The OO approach is in use both as a method of semantic data modelling and as a model of data handled by object-oriented programming and database management systems. From the database systems viewpoint, the OO model adapts some of the constructs of object-oriented programming languages to database systems. The fundamental idea is that of encapsulation which places a wrapper around an identifiable collection of data and the code that operates upon it to produce an object. The state of an object at any time is determined by the value of the data items within its wrapper. These data items are referred to as *instance variables*, and the values held within them are themselves objects. This is an important distinction between objects (in the OO sense) and entities (in the ERA sense) which have a two-tier structure of entity and attribute.

The American National Standards Institute (ANSI 1991) Object-Oriented Database Task Group Final Technical Report describes an object as something 'which plays a role with respect to a request for an operation. The request invokes the operation that defines some service to be performed.' The code associated with a collection of data in an object provides a set of methods that can be performed upon it. As well as executing methods on its own data, an object may as part of one of its methods send a message to another object, causing that object to execute a method in response. This highly active environment is another feature that distinguishes between OO and ERA, which is essentially a collection of passive data. An object has both state, being the values of the instance variables within it, and behaviour, being the potential for acting upon objects (including itself). Objects with the same types of instance variables and methods are said to be in the same object class. Figure 3 shows some instance variables and methods associated with classes Country and Polygon and the manner in which the class Polygon is referenced as an instance variable by the class Country. Figure 4 shows in schematic form an object encapsulating state and methods, receiving a message from another object

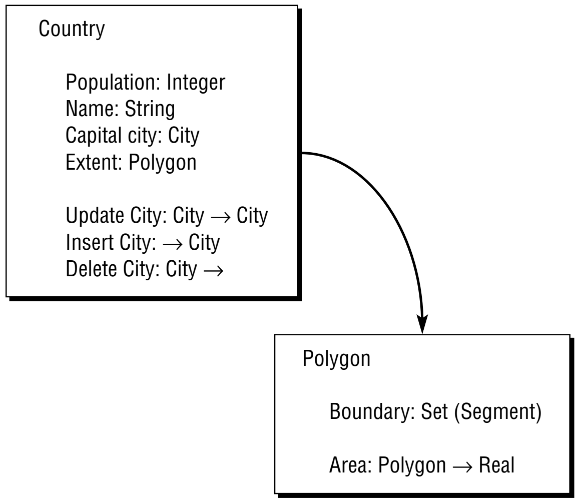

Fig 3. Part of the class descriptions for *Country* and *Polygon*.

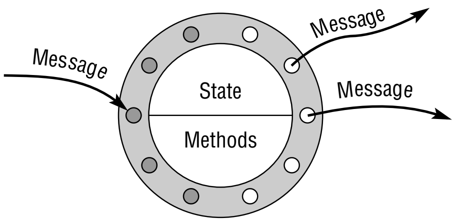

Fig 4. State, methods, and messages of an object.

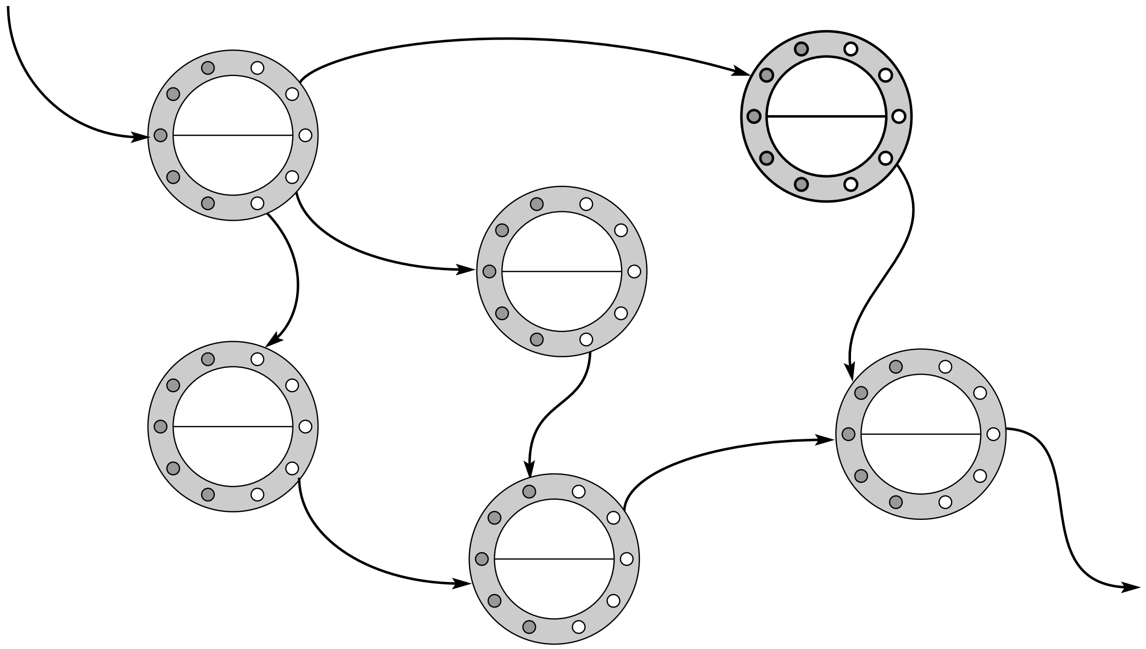

Fig 5. Messages between objects.

and executing methods which result in two messages output. Figure 5 shows the interaction of several objects in response to a message to one of them.

With encapsulation, the internal workings of an object are transparent to users and other objects, which can communicate with it only through a set of predefined message types that the object can understand and handle. To take an example from the real world, I usually do not care about the state of my car under the bonnet (internal state of object class Car) provided that when I put my foot on the accelerator (send message) the car's speed increases (change in the internal state leading to a change in the observable properties of the object). From the viewpoint external to the object, it is only its observable properties that are usually of interest.

3.2.2 Inheritance and composition of objects

Inheritance is an important system and semantic modelling construct, and involves the creation of a new object class by modifying an existing class. Inheritance in an object-oriented setting allows inheritance of methods. Thus, Triangle and Rectangle are subclasses of Polygon. The subclasses inherit all the instance variables and methods from the superclass as well as adding their own. In this example, Triangle and Rectangle may have specialised methods, for example the algorithm implementing the operation Area may be different for Rectangle and Triangle, and each will be different from an Area algorithm for Polygon. This phenomenon, where an operator with the same name has different implementations in different classes, is called *operator polymorphism*. An example of an inheritance hierarchy of spatial object classes is given below (see Figure 8).

Object *composition* allows the modelling of objects with complex internal structures. There are several ways in which a collection of objects might be composed into a new object. *Aggregation* composes a collection of object classes into an aggregate class. For example, an object class Property might be an aggregate of object classes Land Parcel and Dwelling. To quote Rumbaugh et al (1991), 'an aggregate object is semantically an extended object that is treated as a unit in many operations, although physically is made up of several lesser objects'. *Association* groups objects all from the same class into an associated class. For example, an object class Districts might be an association of individual district object classes.

As an illustration of some of these constructs, Figure 6 shows the object class Country (as an abstract object class, represented as a triangle) with three of its instance variables Name, Population, and Area. Variables Name and Population reference printable object class Character String (represented as an oval) and Area references abstract class Polygon. The class Polygon has instance variable Boundary referencing an association of class Segment (the association class shown in the figure as a star and circle). Each segment has a Begin and End Point, and each Point has a Position which is an aggregation (shown as a cross and circle) of printable classes X-coordinate and Y-coordinate.

3.3 Object-oriented database management systems

3.3.1 Making OO persistent

Object-Oriented Programming Languages (OOPLs) such as C++ and Smalltalk provide the capabilities to support the OO approach described above, including the creation, maintenance, and deletion of objects, object classes, and inheritance hierarchies. Object-Oriented Database Management Systems (OODBMS) supplement these capabilities with database functionality, including the ability to support:

1. persistent objects, object classes, and inheritance hierarchies;
2. non-procedural query languages for object class definition, object manipulation, and retrieval;
3. efficient query handling, including query optimisation and access methods;
4. appropriate transaction processing (ACID properties), concurrency support, recovery, integrity, and security.

There are essentially two choices for the developer of an OODBM system: extend a relational system to handle OO, or build a database system around an OO programming language. Both choices have been tried, and section 3.4 on object-extensions to relational technology explores the former. With regard to the latter, object-oriented features will already be supported by the OOPL, so there is the need to add persistency, query handling, and transaction processing. An approach to persistency is to add a new class Persistent Object and allow all database classes to inherit from this class. The class Persistent Object will include methods to:

1. create a new persistent object;
2. delete a persistent object;

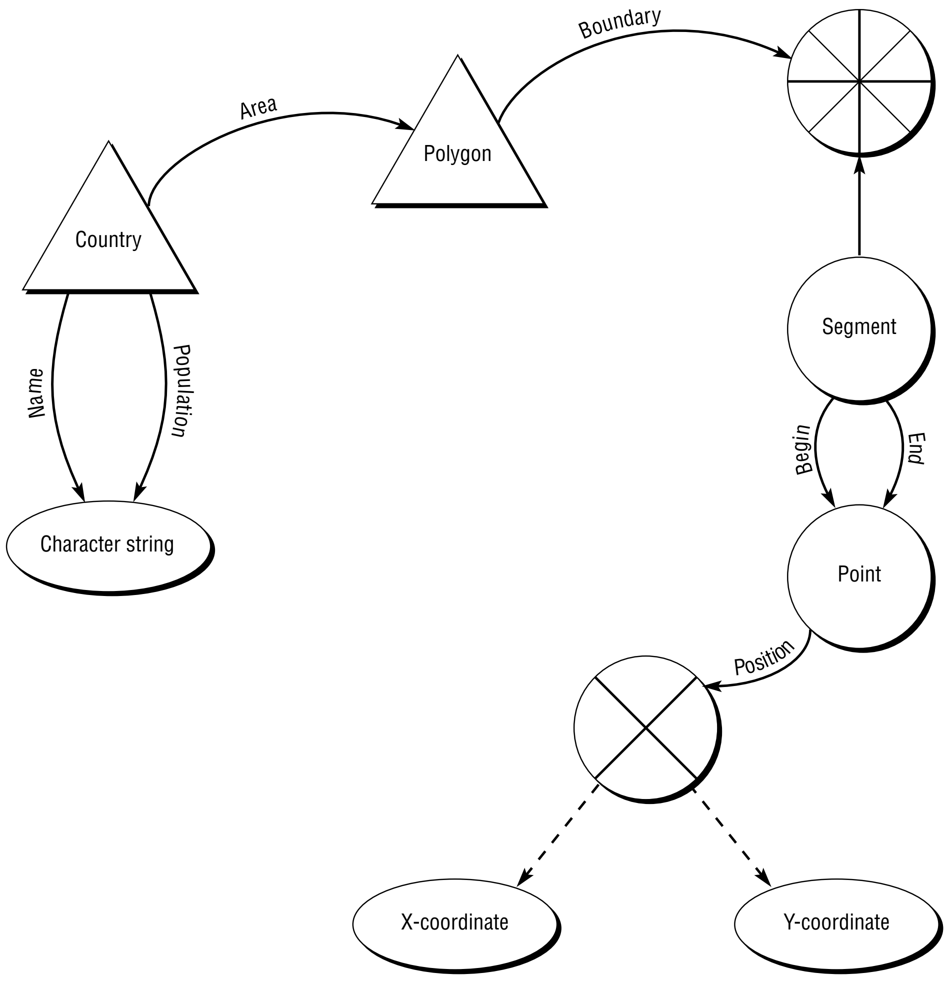

Fig 6. Complex objects.

3 retrieve the state of a persistent object;
4 provide concurrency control;
5 modify a persistent object.

A key benefit of an OODBMS is the support it provides for a unified programming and database environment. However, most current OODBMS treat persistent and non-persistent data differently. A fundamental distinction between RDBs and OODBs is that between call-by-value and call-by-reference. In an RDB, relationships are established by value matching. In our example, to retrieve the population of the capital of Germany, a join between Country and City is made using the value of the Capital field, Berlin. In an OODB (see Figure 7), the connection is made by navigation using the object identifiers (OIDs). The Capital instance variable of Country points to the appropriate City object.

3.3.2 Standardisation of OO systems

The OO approach is more complex than the relational model and has not yet crystallised into a set of universally agreed constructs; even basic constructs like inheritance have been given several different interpretations. Nevertheless, there has been considerable work to arrive at some common definitions.

The Object Management Group (OMG) is a consortium of hardware and software vendors, founded in 1990 with the aim of fostering standards for interoperability of applications within the OO approach. To this end, it has defined the OMG Object Model (see, for example, Kim 1995) many of the concepts of which have been discussed above. An important component of the OMG work is the

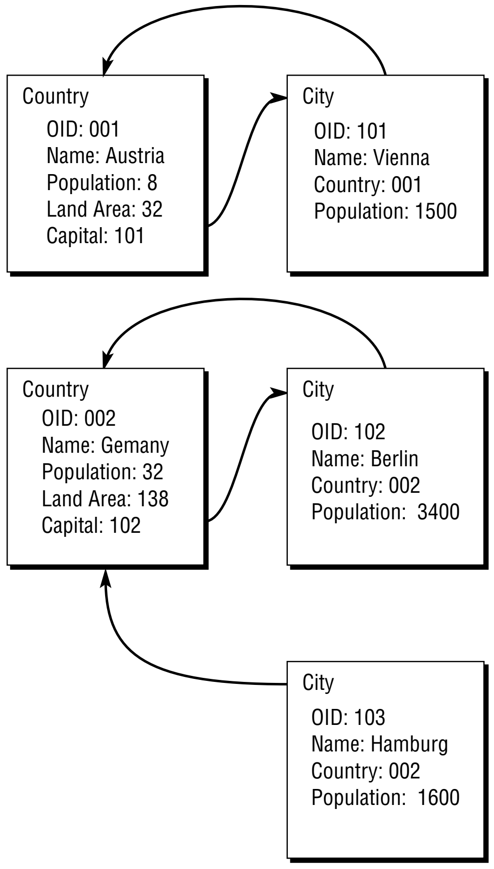

Fig 7. Navigation using object identifiers.

Common Object Request Broker Architecture (CORBA) standard, which specifies an Interface Definition Language for distributed access to objects (see Sondheim et al, Chapter 24). The Object Database Management Group (ODMG) is a consortium of OODBMS vendors, founded in 1990 with the aim of arriving at a commonly agreed OO database interface. The ODMG has defined object definition, manipulation, and query languages, corresponding to data definition and manipulation languages in relational systems.

3.4 Object extensions to relational technology

Object-relational models combine features of object-based and record-based models. They enhance the standard relational model with some object-oriented features, as opposed to OODBMS, that build database functionality around an OO programming language. Enhancements include complex, possibly user-defined data types, inheritance, aggregation, and object identity. Early work at the University of California at Berkeley on the inclusion of new data types in relational database systems (Stonebraker 1986) led to the POSTGRES DBMS (Stonebraker and Rowe 1986). Parallel developments at the IBM Research Laboratories at San Jose, California resulted in the STARBURST project (reported in Haas et al 1990). These developments have led to contemporary proprietary object-relational systems as well as to the addition of object features to new releases of widely used proprietary relational systems. With regard to query languages that support OO extensions to the relational model, SQL3 is currently under development as an international standard. SQL3 is upwardly compatible with SQL-92, and adds support for objects, including multiple inheritance and operators. The goal for object-relational systems is to provide the wide range of object-oriented functionality that has proved so useful for semantic data modelling and programming systems, while at the same time giving the efficient performance associated with the relational model.

Objects are structured by the relational model as tuples of atomic values such as integers, floats, Booleans, or character strings. This provides only a limited means to define complex data types. Object-relational systems allow non-atomic types. A common extension of the relational model to provide for complex data types is to allow *nested relations*. In a nested relation, values of attributes need not be atomic but may themselves be relations.

Relational database systems provide hashing and B-tree indexes for access to standard, system-provided data types. A major extension that an object-relational system allows is the provision of more appropriate indexes for user-defined types. Object-relational systems provide for the definition of a range of indexes appropriate to a heterogeneous collection of object classes.

3.5 OOGIS

A basic requirement for any OO approach to GIS is a collection of spatial object classes. Figure 8 uses the notation of Rumbaugh et al (1991) to represent an inheritance hierarchy of some basic classes. Class Spatial is the most general class, which is specialised into classes Point and Extent (sets of points). Spatial extents may be classified according to dimension, and examples of classes in one dimension (Polyline) and two dimensions (Polygon) are given. Class Polyline is further specialised into classes Open Polyline and Closed Polyline, the former having two distinct end-points while the latter is joined and has no end-points. Of course, this is just an example of some basic spatial object classes. Table 6 shows some sample methods that will act upon these classes. The name of the method is given, along with the classes upon which it acts and the class to which the result belongs.

The OO approach to geospatial data management is now well established. For some time there have been innovatory proprietary GIS that provide OO programming language support, including spatial object classes, overlaying flat-file, or relational databases. There now exist proprietary GIS that incorporate a full OODBMS. Papers that survey the application of OO to GIS include Egenhofer and Frank (1992); Worboys (1994); Worboys et al (1990).

Table 6 Inheritance hierarchy of spatial object classes.

Method	*Operand*	*Operand*	*Result*
Equals?	Spatial	Spatial	Boolean
Belongs?	Point	Extent	Boolean
Subset?	Extent	Extent	Boolean
Intersection	Extent	Extent	Extent
Union	Extent	Extent	Extent
Difference	Extent	Extent	Extent
Boundary	Polygon		ClosedPolyline
Connected?	Extent		Boolean
Extremes	OpenPolyline		Set(Point)
Within?	Point	ClosedPolyline	Boolean
Distance	Point	Point	Real
Bearing	Point	Point	Real
Length	Polyline		Real
Area	Polygon		Real
Centroid	Polygon		Point

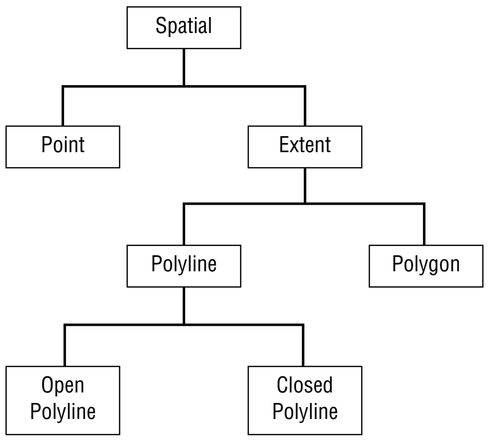

Fig 8. Spatial object class inheritance hierarchy.

4 CONCLUSIONS AND CHALLENGES

The purpose of a database is to serve a user community as a data store for a particular range of applications. Regarding geospatial applications, relational databases have fallen short of effectively achieving that purpose for two main reasons:

1. the relational model has not provided a sufficiently rich set of semantic constructs to allow users to model naturally geospatial application domains;
2. relational technology has not delivered the necessary performance levels for geospatial data management.

This chapter has argued that the OO approach provides part of the solution to these difficulties. It might be that the rapprochement between OO and relational technologies offers the best possible way forward.

There are still significant challenges for the database community in this area, and the chapter concludes by mentioning two of them. First, handling uncertain information has always played a major part in GIS, because many phenomena in the geographical world cannot be represented with total precision and accuracy (Fisher, Chapter 13). Reasoning with uncertain information and managing the associated data in a database remains an important research topic. Deductive databases incorporate logical formalisms, usually subsets of first-order logic, into databases, thereby increasing the expressive power of the query languages and allowing richer semantics for the data models (see, for example, Ceri et al 1990). There has also been work on the fusion of deductive and object technologies for GIS (Paton et al 1996).

Second, the world is in a continual state of change. Classical database technology provides only the capability to manage a single, static snapshot of the application domain. There are two ways in which this can be extended: *temporal* databases manage multiple snapshots (history) of the application domain as it evolves; and *dynamic* databases where a single snapshot changes in step with a rapidly and continuously changing application domain. There are many geospatial applications for both types of extension. Temporal GIS are required to handle such diverse applications as spatio-temporal patterns of land ownership and use, navigation, and global environmental modelling (see Peuquet, Chapter 8). Dynamic systems are required to model such rapidly changing contexts as transportation networks. Problems with development of such systems include the enormous volumes of data required for temporal databases and real-time transaction processing requirements in dynamic systems. These matters are covered in more detail by Bédard (Chapter 29).

References

ANSI 1991 *Object-Oriented Database Task Group final report*. X3/SPARC/DBSSG OODBTG. American National Standards Institute

Astrahan M, Blasgen M, Chamberlin D, Eswaran K, Gray J, Griffiths P, King W, Lorie R, McJones P, Mehl J, Putzolu G, Traiger I, Wade B, Watson V 1976 System R: a relational approach to database management. *ACM Transactions on Database Systems* 1: 97–137

Ceri S, Gottlob G, Tanca L 1990 *Logic programming and databases*. Berlin, Springer

Chamberlin D, Boyce R 1974 SEQUEL: a structured English query language. *Proceedings ACM SIGFIDET Workshop Conference, New York*. ACM Press: 249–64

Chen P P-S 1976 The entity–relationship model – toward a unified view of data. *ACM Transactions on Database Systems* 1: 9–36

Codd E 1970 A relational model for large shared data banks. *Communications of the ACM* 13: 377–87

Codd E 1979 Extending the relational database model to capture more meaning. *ACM Transactions on Database Systems* 4: 397–434

Date C J 1995 *An introduction to database systems*, 6th edition. Reading (USA), Addison-Wesley

Deux O 1990 The story of O_2. *Institute of Electrical and Electronics Engineers Transactions on Knowledge and Data Engineering* 2: 91–108

Egenhofer M J, Frank A 1992 Object-oriented modeling for GIS. *Journal of the Urban and Regional Information Systems Association* 4: 3–19

Haas L M, Chang W, Lohman G M, McPherson J, Wilms P F, Lapis G, Lindsay B, Piransesh H, Carey M J, Shekita E 1990 Starburst mid-flight: as the dust clears. *IEEE Transactions on Knowledge and Data Engineering* 2: 143–60

Healey R G 1991 Database management systems. In Maguire D J, Goodchild M F, Rhind D W (eds) *Geographical information systems: principles and applications*. Harlow, Longman/New York, John Wiley & Sons Inc. Vol. 1: 251–67

Held G D, Stonebraker M R, Wong E 1975 INGRES: a relational database system. *Proceedings AFIPS 44, Montvale.* AFIPS Press: 409–16

ISO 1992 *Database language SQL*. Document ISO/IEC 9075. International Organisation for Standardisation

Kim W (ed) 1995 *Modern database systems: the object model, interoperability, and beyond*. New York, ACM Press

Kim W, Garza J, Ballou N, Woelk D 1990 Architecture of the ORION next-generation database system. *IEEE Transactions on Knowledge and Data Engineering* 2: 109–25

Martin D J 1996 *Geographic information systems: socioeconomic applications*, 2nd edition. London, Routledge

Paton N, Abdelmoty A, Williams M 1996 Programming spatial databases: A deductive object-oriented approach. In Parker D (ed.) *Innovations in GIS 3*. London, Taylor and Francis: 69–78

Roessel J W van 1987 Design of a spatial data structure using the relational normal forms. *International Journal of Geographic Information Systems* 1: 33–50

Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorensen W 1991 *Object-oriented modeling and design*. Englewood Cliffs, Prentice-Hall

Stonebraker M 1986 Inclusion of abstract data types and abstract indexes in a database system. *Proceedings 1986 IEEE Data Engineering Conference, Los Alamitos*. IEEE Computer Society: 262–9

Stonebraker M, Rowe L 1986 The design of POSTGRES. *ACM SIGMOD International Conference on Management of Data, New York*. ACM Press: 340–55

Stonebraker M, Wong E, Kreps P 1976 The design and implementation of INGRES. *ACM Transactions on Database Systems* 1: 189–222

Worboys M F 1994b Object-oriented approaches to georeferenced information. *International Journal of*