9 Creating and Maintaining Geographic Databases

LEARNING OBJECTIVES

After studying this chapter you will be able to:

- Understand the role of database management systems in GI systems.
- Recognize Structured (or Standard) Query Language (SQL) statements.
- Understand the key geographic database data types and functions.
- Be familiar with the stages of geographic database design.
- Understand the key techniques for structuring geographic information, specifically creating topology and indexing.
- Understand the issues associated with multiuser editing and versioning.

All large operational geographic information (GI) systems are built on the foundation of a geographic database. After the people who manage and operate GI systems, the database is arguably the most important part because of the costs of collection and maintenance and because the database forms the basis of all query, analysis, and decision-making activities. Today, virtually all large GI system implementations store data in a database management system (DBMS), a specialist piece of software designed to handle multiuser access to an integrated set of data. Extending standard DBMS to store geographic data raises several interesting challenges. Databases need to be designed with great care and should be structured and indexed to provide efficient query and transaction performance. A comprehensive security and transactional access model is necessary to ensure that multiple users can access the database at the same time. Ongoing maintenance is also an essential, but very resource-intensive, activity.

9.1 Introduction

A database can be thought of as an integrated set of data on a particular subject. Geographic databases are simply databases containing geographic data for a particular area and subject. It is quite common to encounter the term *spatial* in the database world. As discussed in Section 1.1.2, spatial refers to data about any type of space at both geographic and nongeographic scales. A geographic database is a critical part of an operational GI system both because of the cost of creation and maintenance and because of the impact of a geographic database on all analysis, modeling, and decision-making activities. Databases can be physically stored in files or by using specialist software programs called a DBMS. Today, most large organizations use a combination of files and DBMS for storing data assets.

A database is an integrated set of data on a particular subject.

The database approach to storing geographic data offers a number of advantages over traditional file-based datasets.

- Collecting all data at a single location reduces redundancy.
- Maintenance costs decrease because of better organization and reduced data duplication.
- Applications become data independent so that multiple applications can use the same data and can evolve separately over time.

- User knowledge can be transferred between applications more easily because the database remains constant.
- Data sharing is facilitated, and a corporate view of data can be provided to all managers and users.
- Security and standards for data and data access can be established and enforced.
- DBMSs are better able to manage large numbers of concurrent users working with vast amounts of data.

Using databases when compared to files also has some disadvantages.

- The cost of acquiring and maintaining DBMS software can be quite high.
- A DBMS adds complexity to the problem of managing data, especially in small projects.
- Single-user performance will often be better for files, especially for more complex data types and structures where specialist indexes and access algorithms can be implemented.

In recent years geographic databases have become increasingly large and complex (see Table 1.2). For example, a U.S. National Image Mosaic will be over 25 terabytes (TB) in size, a Landsat satellite global image mosaic at 15-m resolution is 6.5 TB, and the Ordnance Survey of Great Britain has approximately 450 million vector features in its MasterMap database covering all of Britain. This chapter describes how to create and maintain geographic databases and presents the concepts, tools, and techniques that are available to manage geographic data in databases. Several other chapters provide additional information that is relevant to this discussion. In particular, the nature of geographic data and how to represent them in GI systems were described in Chapters 2, 3, and 4, and data modeling and data collection were discussed in Chapters 7 and 8, respectively. Later chapters introduce the tools and techniques that are available to query, model, and analyze geographic databases (Chapters 13, 14, and 15). Finally, Chapters 16 through 18 discuss the important management issues associated with creating and maintaining geographic databases.

9.2 Database Management Systems

A DBMS is a software application designed to organize the efficient and effective storage and access of data.

Small, simple databases that are used by a small number of people can be stored on computer hard disks in standard files. However, large, more complex databases with many tens, hundreds, or thousands of users require specialist database management system (DBMS) software to ensure data integrity and longevity. A DBMS is a software application designed to organize efficient and effective data storage and access. To carry out this function, DBMSs provide a number of important capabilities. These are introduced briefly here and are discussed further in this and other chapters.

DBMSs are successful because they are able to provide the following:

- *A data model.* As discussed in Chapter 7, a data model is a mechanism used to represent real-world objects digitally in a computer system. All DBMSs include standard general-purpose core data models suitable for representing several object types (e.g., integer and floating-point numbers, dates, and text). Several DBMSs now also support geographic (spatial) object types.
- *A data load capability.* DBMSs provide tools to load data into databases. Simple tools are available to load standard supported data types (e.g., character, number, and date) in well-structured formats.
- *Indexes.* An index is a data structure that is used to speed up searching. All databases include tools to index standard database data types.
- *A query language.* One of the major advantages of DBMS is that they support a standard data query/manipulation language called SQL (Structured/Standard Query Language).
- *Security.* A key characteristic of DBMSs is that they provide controlled access to data. This includes the ability to restrict user access to just part of a database. For example, a casual GI system user might have read-only access to only part of a database, but a specialist user might have read and write (create, update, and delete) access to the entire database.
- *Controlled update.* Updates to databases are controlled through a transaction manager responsible for managing multiuser access and ensuring that updates affecting more than one part of the database are coordinated.
- *Backup and recovery.* It is important that the valuable data in a database are protected from system failure and incorrect (accidental or deliberate) update. Software utilities are provided to back up all or part of a database and to recover the database in the event of a problem.
- *Database administration tools.* The task of setting up the structure of a database (the schema), creating and maintaining indexes, tuning to improve

performance, backing up and recovering, and allocating user access rights is performed by a database administrator (DBA). A specialized collection of tools and a user interface are provided for this purpose.

- *Applications.* Modern DBMSs are equipped with standard, general-purpose tools for creating, using, and maintaining databases. These include applications for designing databases (CASE [computer-aided software engineering] tools) and for building user interfaces for data access and presentations (e.g., forms and reports).
- *Application programming interfaces (APIs).* Although most DBMSs have good general-purpose applications for standard use, most large, specialist applications will require further customization using a commercial off-the-shelf programming language and a DBMS programmable API.

This list of DBMS capabilities is very attractive to GI system users, and so, not surprisingly, virtually all large GI databases are managed using DBMS technology. Indeed, most GI system software vendors include DBMS software within their GI system software products or provide an interface that supports very close coupling to a DBMS. For further discussion of this subject, see Chapter 7.

Today, virtually all large GI systems use DBMS technology for data management.

9.2.1 Types of DBMSs

DBMSs can be classified according to the way they organize, store, and manipulate data. Three main types of DBMSs have been used in GI systems: relational (RDBMS), object (ODBMS), and object-relational (ORDBMS).

A relational database comprises a set of tables, each a two-dimensional list (or array) of records containing attributes about the objects under study. This apparently simple structure has proven to be remarkably flexible and useful in a wide range of application areas, such that historically over 95% of the data in DBMSs have been stored in RDBMSs. Most of the world's current DBMSs are built on a foundation of core relational concepts.

Object database management systems (ODBMSs) were initially designed to address several of the weaknesses of RDBMSs. These include the inability to store complete objects directly in the database (both object state and behavior; see Box 7.2 for an overview of objects and object technology). Because RDBMSs were focused primarily on business applications such as banking, human resource management, and stock control and inventory, they were never designed to deal with rich data types, such as geographic objects, sound, and video. A further difficulty is the poor performance of RDBMSs for many types of geographic query. These problems are compounded by the difficulty of extending RDBMSs to support geographic data types and processing functions, which obviously limits their adoption for geographic applications. ODBMSs can store objects persistently (semipermanently on disk or other media) and provide object-oriented query tools. A number of commercial ODBMSs have been developed, including GemStone/S Object Server from GemStone Systems, Inc., Objectivity/DB from Objectivity, Inc., and Versant Object Database from Versant Object Technology Corp.

Despite the technical elegance of ODBMSs, they have not proven to be as commercially successful as some people initially predicted. This is largely because of the massive installed base of RDBMSs and the fact that RDBMS vendors have now added many of the important ODBMS capabilities to their standard RDBMS software systems to create hybrid object-relational DBMSs (ORDBMSs). An ORDBMS can be thought of as an RDBMS engine with some additional capabilities for dealing with objects. They can handle both the data describing what an object is (object attributes such as color, size, and age) and the behavior that determines what an object does (object methods or functions such as drawing instructions, query interfaces, and interpolation algorithms), and these can be managed and stored together as an integrated whole. Examples of ORDBMS software include IBM's DB2 and Informix, Microsoft's SQL Server, and Oracle Corp.'s Oracle DBMS. Because ORDBMSs and the underlying relational model are so important in GI systems, these topics are discussed at length in Section 9.3.

A number of ORDBMSs have now been extended to support geographic object types and functions through the addition of seven key capabilities (these are introduced here and discussed further later in this chapter):

- Query parser—the engines used to interpret queries by splitting them up and decoding them have been extended to deal with geographic types and functions.
- Query optimizer—software query optimizers have been enhanced so that they are able to handle geographic queries efficiently. Consider a query to find all potential users of a new brand of premier wine to be marketed to wealthy households from a network of retail stores. The objective is to select all households within 3 km of a store that have an income greater than $110,000. This could be carried out in two ways:

1. Select all households with an income greater than $110,000; from this selected set, select all households within 3 km of a store.
2. Select all households within 3 km of a store; from this selected set select all households with an income greater than $110,000.

Selecting households with an income greater than $110,000 is an attribute query that can be performed very quickly. Selecting households within 3 km of a store is a geometric query that takes much longer. Executing the attribute query first (Option 1 above) will result in fewer geometry query tests for store proximity, and therefore the whole query will be completed much more quickly.

- Query language—query languages have been improved to handle geographic types (e.g., points and areas) and functions (e.g., select areas that touch each other).
- Indexing services—standard unidimensional DBMS data index services have been extended to support multidimensional (i.e., *x, y, z* coordinates) geographic data types.
- Storage management—the large volumes of geographic records with different sizes (especially geometric and topological relationships) have been accommodated through specialized storage structures.
- Transaction services—standard DBMSs are designed to handle short (subsecond) transactions, and these have been extended to deal with the long transactions common in many geographic applications.
- Replication—services for replicating databases have been extended to deal with geographic types and with problems of reconciling changes made by distributed users.

9.2.2 Geographic DBMS Extensions

A number of the major commercial DBMS vendors have released spatial database extensions to their standard ORDBMS products. IBM offers two solutions—DB2 Spatial Extender and Informix Spatial; Microsoft has released spatial capabilities in the core of SQLServer; and Oracle has spatial capability in the core of Oracle DBMS and a Spatial option that adds more advanced features (see Box 9.1). The open-source DBMS PostgreSQL has also been extended with spatial types and functions (PostGIS).

ORDBMSs provide core support for geographic data types and functions.

Although these systems differ in technology, scope, and features, they all provide basic capabilities to store, manage, and query geographic objects. This

Technical Box 9.1

Oracle Spatial

Oracle Spatial is an extension to the Oracle DBMS that provides the foundation for the management of spatial (including geographic) data inside an Oracle database. The standard types and functions in Oracle (CHAR, DATE, or INTEGER, etc.) are extended with geographic equivalents. Oracle Spatial supports three basic geometric forms:

- Points: Points can represent locations such as buildings, fire hydrants, utility poles, oil rigs, boxcars, or roaming vehicles.
- Polylines: Polylines can represent things like roads, rivers, utility lines, or fault lines.
- Polygons and complex polygons with holes: Polygons can represent things like outlines of cities, districts, floodplains, or oil and gas fields. A polygon with a hole might geographically represent a parcel of land surrounding a patch of wetland.

These simple feature types can be aggregated to richer types using topology and linear referencing capabilities (see Section 7.2.3.3). In addition, Oracle Spatial can store and manage georaster (image) data. Oracle Spatial extends the Oracle DBMS query engine to support geographic queries. There is a set of spatial operators to perform area-of-interest and spatial-join queries; length, area, and distance calculations; buffer and union queries; and administrative tasks. The Oracle Spatial SQL used to create a table and populate it with a single record is shown in the following script. (The characters after the dash on each line are comments that describe the operations. The discussion of SQL syntax in Section 9.4 will help decode this program.)

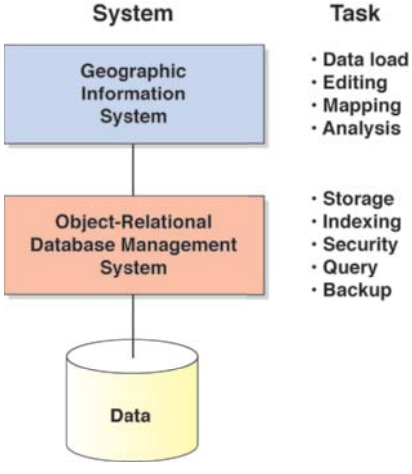
```
-- Create a table for routes (highways).
CREATE TABLE lrs_routes (
 route_id  NUMBER PRIMARY KEY,
 route_name  VARCHAR2(32),
 route_geometry  MDSYS.SDO_GEOMETRY);
-- Populate table with just one route
   for this example.
INSERT INTO lrs_routes VALUES(
 1,
 'Route1',
```

```
MDSYS.SDO_GEOMETRY(
 3002, -- line string, 3 dimensions:
    X,Y,M
 NULL,
 NULL,
 MDSYS.SDO_ELEM_INFO_ARRAY(1,2,1), -- one
    line string, straight segments
MDSYS.SDO_ORDINATE_ARRAY(
 2,2,0, -- Starting point - Exit1; 0
    is measure from start.
 2,4,2, -- Exit2; 2 is measure from
    start.
 8,4,8, -- Exit3; 8 is measure from
    start.
 12,4,12, -- Exit4; 12 is measure from
    start.
 12,10,NULL, -- Not an exit; measure
    will be automatically calculated &
    filled.
 8,10,22, -- Exit5; 22 is measure from
    start.
 5,14,27) -- Ending point (Exit6); 27
    is measure from start.
 )
);
```

Geographic data in Oracle Spatial can be indexed using R-tree and quadtree indexing methods (these terms are defined in Section 9.7.2). There are also capabilities for managing projections and coordinate systems, as well as long transactions (see discussion in Section 9.9.1). Finally, there are also some tools for elementary spatial data analysis (Chapters 13 and 14). Oracle Spatial can be used with all major GI software products, and developers can create specific-purpose applications that embed SQL commands for manipulating and querying data. Oracle has generated considerable interest among larger IT-focused organizations. IBM has approached this market in a similar way with its Spatial Extender for the DB2 DBMS and Spatial for Informix. Most recently Microsoft has added comparable spatial capabilities to its SQLServer DBMS. There are also open-source alternatives such as PostGIS.

is achieved by implementing the seven key database extensions described in the previous section. It is important to realize, however, that none of these is a complete GI system in itself. The focus of these extensions is data storage, retrieval, and management, and they lack the advanced capabilities for geographic editing, mapping, and analysis. Consequently, they must be used in conjunction with a conventional GI system except in the case of the simplest query-focused applications. Figure 9.1 shows how GI systems and DBMS software can work together in a generalized way and some of the tasks best carried out by each system.

Figure 9.1 The roles of GI systems and DBMS.



9.3 Storing Data in DBMS Tables

The lowest level of user interaction with a geographic database is usually the object class (also sometimes called a layer or feature class), which is an organized collection of data on a particular theme (e.g., all pipes in a water network, all soil polygons in a river basin, or all elevation values in a terrain surface). Object classes are stored in standard database tables. A table is a two-dimensional array of rows and columns. Each object class is stored as a single database table in a DBMS. Table rows contain objects (instances of object classes, e.g., data for a single pipe), and the columns contain object properties, or attributes as they are frequently called (Figure 9.2: see also Figure 9.10 as a conceptual example). The data stored at individual row, column intersections are usually referred to as values. Geographic database tables are distinguished from nongeographic tables by the presence of a geometry column (often called the shape column). To save space and improve performance, the actual coordinate values may be stored in a highly compressed binary form.

(A)

Attributes of STATES

FID	Shape*	AREA	STATE_NAME	STATE_FIPS
41	Polygon	51715.656	Alabama	01
49	Polygon	576556.687	Alaska	02
35	Polygon	113711.523	Arizona	04
45	Polygon	52912.797	Arkansas	05
23	Polygon	157774.187	California	06
30	Polygon	104099.109	Colorado	08
17	Polygon	4976.434	Connecticut	09
27	Polygon	2054.506	Delaware	10
26	Polygon	66.063	District of Columbia	11
47	Polygon	55815.051	Florida	12
43	Polygon	58629.195	Georgia	13
48	Polygon	6381.435	Hawaii	15
7	Polygon	83340.594	Idaho	16
25	Polygon	56297.953	Illinois	17
20	Polygon	36399.516	Indiana	18
12	Polygon	56257.219	Iowa	19
32	Polygon	82195.437	Kansas	20
31	Polygon	40318.777	Kentucky	21
46	Polygon	45835.898	Louisiana	22
2	Polygon	32161.664	Maine	23
29	Polygon	9739.753	Maryland	24
13	Polygon	8172.482	Massachusetts	25
50	Polygon	57898.367	Michigan	26
9	Polygon	84517.469	Minnesota	27
42	Polygon	47618.723	Mississippi	28
34	Polygon	69831.625	Missouri	29
1	Polygon	147236.031	Montana	30

Record: 0 Show: All Selected Records (0 out of 51 Selected.)

(B)

Attributes of POPULATION

STATE_FIPS	SUB_REGION	STATE_ABBR	POP1990	POP1996
53	Pacific	WA	4866692	5629613
30	Mtn	MT	799065	885762
23	N Eng	ME	1227928	1254465
38	W N Cen	ND	638800	633534
46	W N Cen	SD	696004	721374
56	Mtn	WY	453588	487142
55	E N Cen	WI	4891769	5144123
16	Mtn	ID	1006749	1201327
50	N Eng	VT	562758	587726
27	W N Cen	MN	4375099	4639933
41	Pacific	OR	2842321	3203820
33	N Eng	NH	1109252	1156932
19	W N Cen	IA	2776755	2831890
25	N Eng	MA	6016425	6066573
31	W N Cen	NE	1578385	1622272
36	Mid Atl	NY	17990455	18293435
42	Mid Atl	PA	11881643	12077607
09	N Eng	CT	3287116	3287604
44	N Eng	RI	1003464	993306
34	Mid Atl	NJ	7730188	7956917
18	E N Cen	IN	5544159	5801023
32	Mtn	NV	1201833	1532295
49	Mtn	UT	1722850	2000630
06	Pacific	CA	29760021	32218713
39	E N Cen	OH	10847115	11123416
17	E N Cen	IL	11430602	11731783
11	S Atl	DC	606900	550076

Record: 0 Show: All Selected Records (0 out of 51 Selec

(C)

Attributes of COMBINED STATES and POPULATION

FID	Shape*	AREA	STATE_NAME	STATE_FIPS	SUB_REGION	STATE_ABBR	POP1990	POP1996
0	Polygon	67286.875	Washington	53	Pacific	WA	4866692	5629613
1	Polygon	147236.031	Montana	30	Mtn	MT	799065	885762
2	Polygon	32161.664	Maine	23	N Eng	ME	1227928	1254465
3	Polygon	70810.156	North Dakota	38	W N Cen	ND	638800	633534
4	Polygon	77193.625	South Dakota	46	W N Cen	SD	696004	721374
5	Polygon	97799.492	Wyoming	56	Mtn	WY	453588	487142
6	Polygon	56088.066	Wisconsin	55	E N Cen	WI	4891769	5144123
7	Polygon	83340.594	Idaho	16	Mtn	ID	1006749	1201327
8	Polygon	9603.218	Vermont	50	N Eng	VT	562758	587726
9	Polygon	84517.469	Minnesota	27	W N Cen	MN	4375099	4639933
10	Polygon	97070.750	Oregon	41	Pacific	OR	2842321	3203820
11	Polygon	9259.514	New Hampshire	33	N Eng	NH	1109252	1156932
12	Polygon	56257.219	Iowa	19	W N Cen	IA	2776755	2831890
13	Polygon	8172.482	Massachusetts	25	N Eng	MA	6016425	6066573
14	Polygon	77328.336	Nebraska	31	W N Cen	NE	1578385	1622272
15	Polygon	48560.578	New York	36	Mid Atl	NY	17990455	18293435
16	Polygon	45359.238	Pennsylvania	42	Mid Atl	PA	11881643	12077607
17	Polygon	4976.434	Connecticut	09	N Eng	CT	3287116	3287604
18	Polygon	1044.850	Rhode Island	44	N Eng	RI	1003464	993306
19	Polygon	7507.302	New Jersey	34	Mid Atl	NJ	7730188	7956917
20	Polygon	36399.516	Indiana	18	E N Cen	IN	5544159	5801023
21	Polygon	110667.297	Nevada	32	Mtn	NV	1201833	1532295
22	Polygon	84870.187	Utah	49	Mtn	UT	1722850	2000630
23	Polygon	157774.187	California	06	Pacific	CA	29760021	32218713
24	Polygon	41192.863	Ohio	39	E N Cen	OH	10847115	11123416
25	Polygon	56297.953	Illinois	17	E N Cen	IL	11430602	11731783
26	Polygon	66.063	District of Columbia	11	S Atl	DC	606900	550076
27	Polygon	2054.506	Delaware	10	S Atl	DE	666168	724890

Record: 0 Show: All Selected Records (0 out of 51 Selected.) Options

Figure 9.2 Parts of GI systems database tables for U.S. states: (A) STATES table; (B) POPULATION table; (C) joined table—COMBINED STATES and POPULATION.

Relational databases are made up of tables. Each geographic class (layer) is stored as a table.

Tables are joined using common row/column values or keys, as they are known in the database world. Figure 9.2 shows parts of tables containing data about U.S. states. The STATES table (Figure 9.2A) contains the geometry (in the SHAPE field) and some basic attributes, an important one being a unique STATE FIPS (STATE_FIPS [Federal Information Processing Standard] code) identifier. The POPULATION table (Figure 9.2B) was created entirely independently, but also has a unique identifier

column called STATE_FIPS. Using standard database tools, the two tables can be joined based on the common STATE_FIPS identifier column (the key) to create a third table, COMBINED STATES and POPULATION (Figure 9.2C). Following the join, these can be treated as a single table for all GI systems operations such as query, display, and analysis. There is additional discussion of relational joins and their generalization to spatial joins in Section 13.2.2.

Database tables can be joined to create new views of the database.

In a groundbreaking description of the relational model that underlies the vast majority of the world's databases, in 1970 Ted Codd of IBM defined a series of rules for the efficient and effective design of database table structures. The heart of Codd's idea was that the best relational databases are made up of simple, stable tables that follow five principles:

1. There is only one value in each cell at the intersection of a row and column.
2. All values in a column are about the same subject.
3. Each row is unique.
4. There is no significance to the sequence of columns.
5. There is no significance to the sequence of rows.

Figure 9.3A shows a database table of land parcels for tax assessment that contradicts several of Codd's principles. Codd suggests a series of transformations called normal forms that successively improve the simplicity and stability and reduce the redundancy of database tables (thus reducing the risk of editing conflicts) by splitting them into subtables that are rejoined at query time. Unfortunately, joining large tables is computationally expensive and can result in complex database designs that are difficult to maintain. For this reason, nonnormalized table designs are often used in GI systems.

Figure 9.3B is a cleansed version of 9.3A that has been entered into a GI system DBMS: There is now only one value in each cell (Date and AssessedValue are now separate columns); missing values have been added; an OBJECTID (unique system identifier) column has been added; and the potential confusion between Dave Widseler and D Widseler has been resolved. Figure 9.3C shows the same data after some normalization to make it suitable for use in a tax assessment application. The database now consists of three tables that can be joined using common keys. Figure 9.3C `Attributes of Tab10_3b` can be joined to Figure 9.3C `Attributes of Tab10_3a` using the common ZoningCode column, and Figure 9.3C `Attributes of Tab10_3c` can be joined using OwnersName to create Figure 9.3D. It is now possible to execute SQL queries against these joined tables as discussed in the next section.

Figure 9.3 Tax assessment database: (A) raw data; (B) cleaned data in a GI systems DBMS (*continued*)

(A)

Attributes of Tab10_1

OBJECTID*	ParcelNumb	OwnerNam	OwnerAddress	PostalCode	ZoningCode	ZoningType	DateAssessed	AssessedValue
1	673-100	Jeff Peters	10 Railway Cuttings	114390	2	Residential	2002	220000
2	673-101	Joel Campbell	1115 Center Place	114390	2	Residential	2003	545500
3	674-100	Dave Widseler		114391	3	Commercial	99	249000
4	674-100		452 Diamond Plaza	114391	3	Commercial	2000	275500
5	674-100	D Widseler	452 Diamond Plaza	114391	3	Commercial	2001	290000
6	670-231	Sam Camarata	19 Big Bend Bld	114391	2	Residential	2004	450575
7	674-112	Chris Capelli	Hastings Barracks	114392	2	Residential	2004	350000
8	674-113	Sheila Sullivan	10034 Endin Mansions	114391	2	Residential	02	1005425

Record: 0 Show: All Selected Records (0 out of 8 Selected.) Options

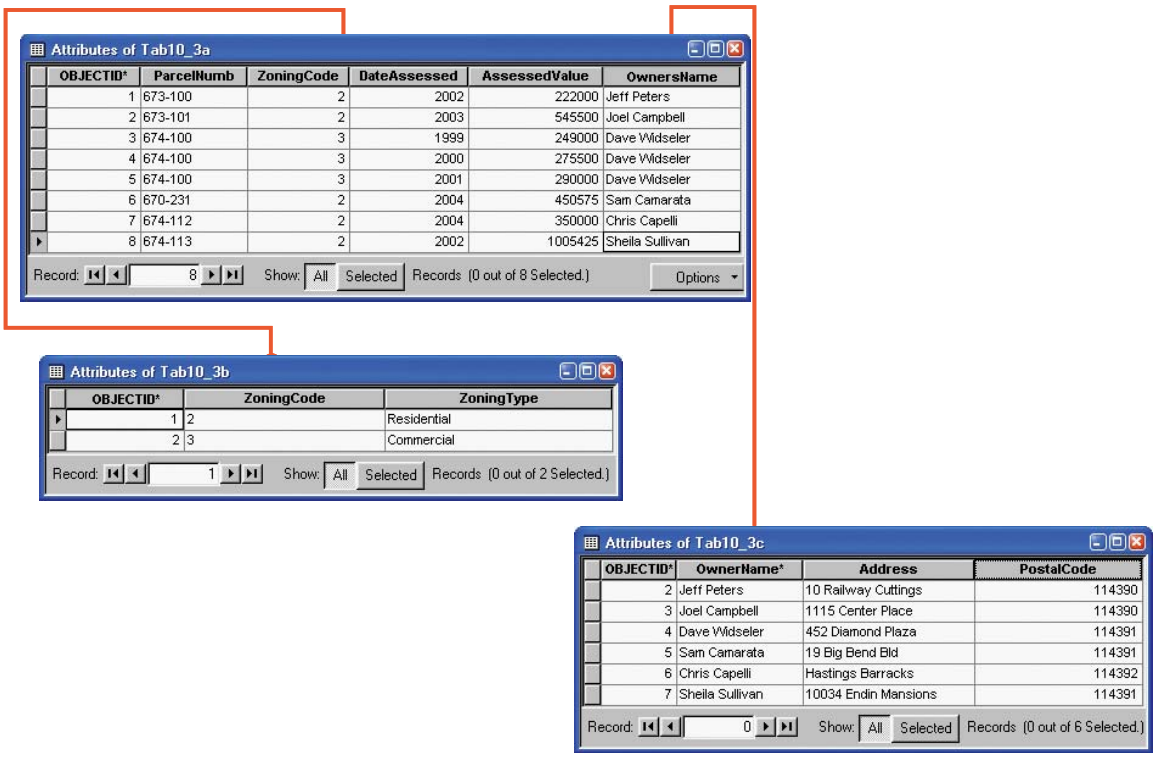
(B)

Attributes of Tab10_2

OBJECTID*	ParcelNumb	OwnerNam	OwnerAddress	PostalCode	ZoningCode	ZoningType	DateAssessed	AssessedValue
1	673-100	Jeff Peters	10 Railway Cuttings	114390	2	Residential	2002	220000
2	673-101	Joel Campbell	1115 Center Place	114390	2	Residential	2003	545500
3	674-100	Dave Widseler	452 Diamond Plaza	114391	3	Commercial	1999	249000
4	674-100	Dave Widseler	452 Diamond Plaza	114391	3	Commercial	2000	275500
5	674-100	Dave Widseler	452 Diamond Plaza	114391	3	Commercial	2001	290000
6	670-231	Sam Camarata	19 Big Bend Bld	114391	2	Residential	2004	450575
7	674-112	Chris Capelli	Hastings Barracks	114392	2	Residential	2004	350000
8	674-113	Sheila Sullivan	10034 Endin Mansions	114391	2	Residential	2002	1005425

Record: 0 Show: All Selected Records (0 out of 8 Selected.) Options

(C)



Attributes of Tab10_3a

OBJECTID*	ParcelNumb	ZoningCode	DateAssessed	AssessedValue	OwnersName
1	673-100	2	2002	222000	Jeff Peters
2	673-101	2	2003	545500	Joel Campbell
3	674-100	3	1999	249000	Dave Widseler
4	674-100	3	2000	275500	Dave Widseler
5	674-100	3	2001	290000	Dave Widseler
6	670-231	2	2004	450575	Sam Camarata
7	674-112	2	2004	350000	Chris Capelli
8	674-113	2	2002	1005425	Sheila Sullivan

Record: 8 Show: All Selected Records (0 out of 8 Selected.) Options

Attributes of Tab10_3b

OBJECTID*	ZoningCode	ZoningType
1	2	Residential
2	3	Commercial

Record: 1 Show: All Selected Records (0 out of 2 Selected.)

Attributes of Tab10_3c

OBJECTID*	OwnerName*	Address	PostalCode
2	Jeff Peters	10 Railway Cuttings	114390
3	Joel Campbell	1115 Center Place	114390
4	Dave Widseler	452 Diamond Plaza	114391
5	Sam Camarata	19 Big Bend Bld	114391
6	Chris Capelli	Hastings Barracks	114392
7	Sheila Sullivan	10034 Endin Mansions	114391

Record: 0 Show: All Selected Records (0 out of 6 Selected.)

(D)

Attributes of Tab10_3a_Tab10_3b_Tab10_3c

OBJECTID	ParcelNumb	DateAssessed	AssessedValue	ZoningType	ZoingCode	OwnerName	OwnerAddress	PostalCode
1	673-100	2002	222000	Residential	2	Jeff Peters	10 Railway Cuttings	114390
2	673-101	2003	545500	Residential	2	Joel Campbell	1115 Center Place	114390
3	674-100	1999	249000	Commercial	3	Dave Widseler	452 Diamond Plaza	114391
4	674-100	2000	275500	Commercial	3	Dave Widseler	452 Diamond Plaza	114391
5	674-102	2004	290000	Residential	2	Dave Widseler	452 Diamond Plaza	114391
6	670-231	2004	450575	Residential	2	Sam Camarata	19 Big Bend Bld	114391
7	674-112	2004	350000	Residential	2	Chris Capelli	Hastings Barracks	114392
8	674-113	2002	1005425	Residential	2	Sheila Sullivan	10034 Endin Mansions	114391

Record: 0 Show: All Selected Records (0 out of 8 Selected.) Options

Figure 9.3 (*continued*) (C) data partially normalized into three subtables; and (D) joined table.

9.4 SQL

The standard database query language adopted by virtually all mainstream databases is SQL (Structured or Standard Query Language: ISO Standard ISO/IEC 9075). There are many good background books and system implementation manuals on SQL, and so only brief details will be presented here. SQL may be used directly via an interactive command line interface; it may be compiled in a general-purpose programming language (e.g., C/C++/C#, Java, or Visual Basic); or it may be embedded in a graphical user interface (GUI). SQL is a set-based, rather than a procedural (e.g., Visual Basic) or object-oriented (e.g., Java or C#) programming language designed to retrieve sets (row and column combinations) of data from tables.

There are three key types of SQL statements: DDL (Data Definition Language), DML (Data Manipulation Language), and DCL (Data Control Language). A major revision of SQL in 2004 defined spatial types and functions as part of a multimedia extension called SQL/MM. A further revision in 2011 added temporal database queries.

The data in the database shown in Figure 9.3C may be queried to find parcels where the AssessedValue is greater than $300,000 and the ZoningType is Residential. This is an apparently simple query, but it requires three table joins to execute it. The SQL statements as implemented in the Microsoft Access DBMS are as follows:

```
SELECT Tab10_3a.ParcelNumb, Tab10_3c.Address,
   Tab10_3a.AssessedValue
```

```
FROM (Tab10_3b INNER JOIN Tab10_3a ON
    Tab10_3b.ZoningCode =
    Tab10_3a.ZoningCode) INNER JOIN Tab10_3c
    ON Tab10_3a.OwnersName =
    Tab10_3c.OwnerName
WHERE (((Tab10_3a.AssessedValue) >300000) AND
    ((Tab10_3b.ZoningType) ="Residential"));
```

The SELECT statement defines the columns to be displayed (the syntax is TableName.ColumnName). The FROM statement is used to identify and join the three tables (INNER JOIN is a type of join that signifies that only matching records in the two tables will be considered). The WHERE clause is used to select the rows from the columns using the constraints (((Tab10_3a.AssessedValue) >300000) AND ((Tab10_3b.ZoningType) ="Residential")). The result of this query is shown in Figure 9.4. This triplet of SELECT, FROM, WHERE is the staple of SQL queries.

SQL is the standard database query language. Today it has geographic capabilities.

In SQL, Data Definition Language statements are used to create, alter, and delete relational database structures. The CREATE TABLE command is used to define a table, the attributes it will contain, and the primary key (the column used to identify records uniquely). For example, the SQL statement to create a table to store data about Countries, with two columns (name and shape (geometry)), is as follows:

```
CREATE TABLE Countries (
name          VARCHAR(200) NOT NULL PRIMARY
    KEY,
shape         POLYGON NOT NULL
CONSTRAINT  spatial reference
CHECK         (SpatialReference(shape)  = 14)
)
```

This SQL statement defines several table parameters. The name column is of type VARCHAR (variable character) and can store values up to 200 characters. Name cannot be null (NOT NULL); that is, it must have a value, and it is defined as the PRIMARY KEY, which means that its entries must be unique. The shape column is of type POLYGON, and it is defined as NOT NULL. It has an additional spatial reference constraint (projection), meaning that a spatial reference is enforced for all shapes (Type 14—this will vary by system, but could be Universal Transverse Mercator (UTM)—see Section 4.8.2).

Data can be inserted into this table using the SQL INSERT command:

```
INSERT INTO Countries
(Name, Shape) VALUES ('Kenya', Polygon
   (('(x y, x y, x y, x y) ,2))
```

Actual coordinates would need to be substituted for the *x*, *y* values. Several additions of this type would result in a table like the following:

Name	Shape
Kenya	Polygon geometry
South Africa	Polygon geometry
Egypt	Polygon geometry

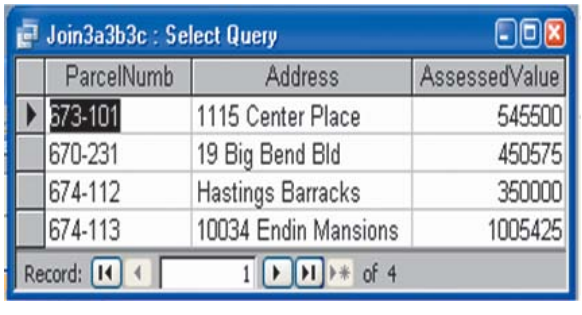
Data Manipulation Language statements are used to retrieve and manipulate data. Objects with a size greater than 11,000 can be retrieved from the countries table using a SELECT statement:

```
SELECT Countries.Name,
FROM Countries
WHERE Area(Countries.Shape) > 11000
```

In this system, the area is computed automatically from the shape field using a DBMS function and does not need to be stored.

Data Control Language statements handle authorization access. The two main DCL keywords are GRANT and REVOKE, which authorize and rescind access privileges, respectively.

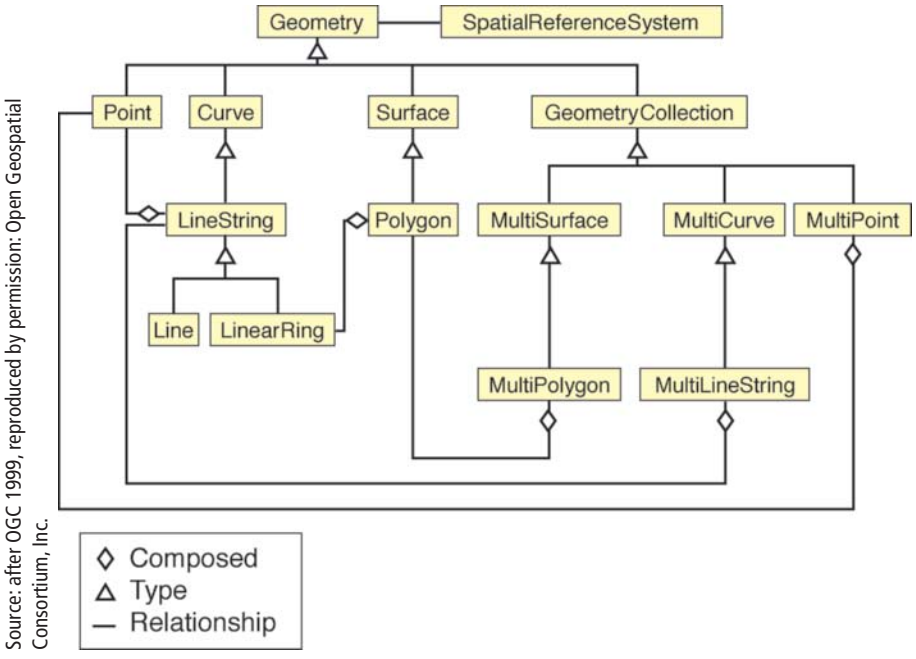
Figure 9.4 Results of a SQL query against the tables in Figure 9.3C (see text for query and further explanation).



9.5 Geographic Database Types and Functions

Several attempts have been made to define a superset of geographic data types that can represent and process geographic data in databases. Unfortunately, space does not permit a review of them all here. This discussion will instead focus on the practical aspects of this problem and will be based on the widely accepted ISO (International Organization for Standards) and Open Geospatial Consortium (OGC) standards. The GI systems community working under the auspices of ISO and OGC has defined the core geographic types and functions to be used in a DBMS and accessed using the SQL query language (see Section 9.4 for a

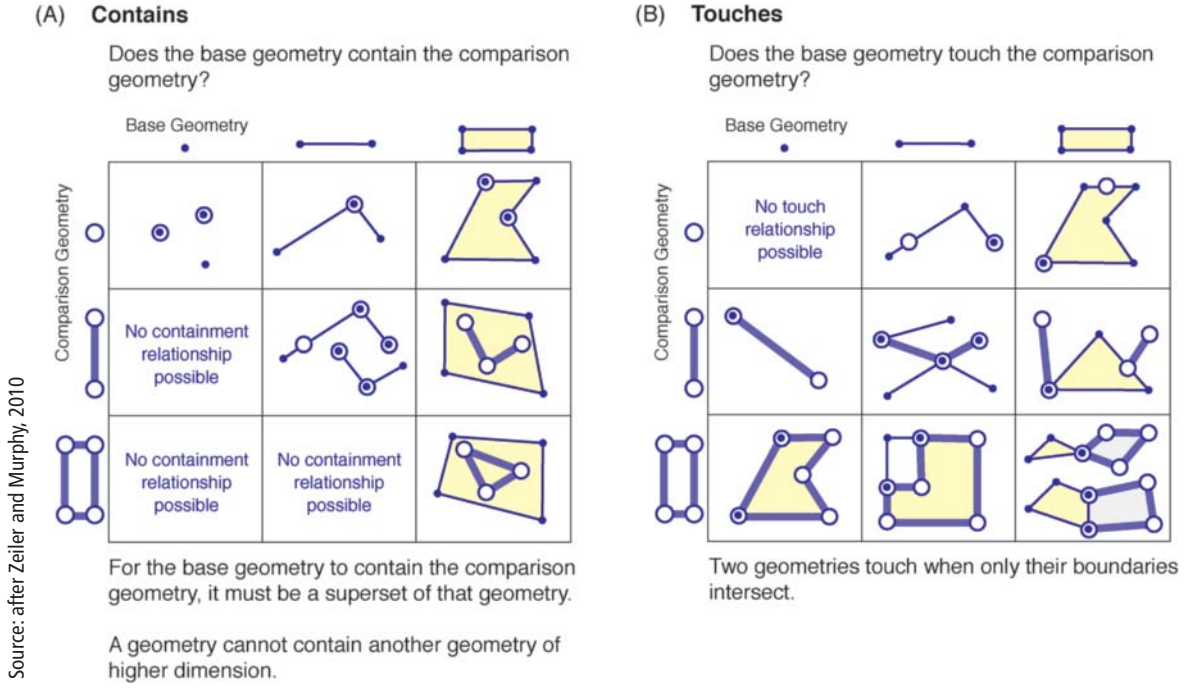
Figure 9.5 Geometry class hierarchy.



discussion of SQL). The geometry types are shown in Figure 9.5. In this hierarchy, the Geometry class is the root class. It has an associated spatial reference (coordinate system and projection, for example, Lambert Azimuthal Equal Area). The Point, Curve, Surface, and GeometryCollection classes are all subtypes of Geometry. The other classes (boxes) and relationships (lines) show how geometries of one type are aggregated from others (e.g., a LineString is a collection of Points). For further explanation of how to interpret this object model diagram, see the discussion in Section 7.3.

According to this ISO/OGC standard, there are nine methods for testing spatial relationships between these geometric objects. Each method takes as input two geometries (collections of one or more geometric objects) and evaluates whether or not the relationship is true. Two examples of possible relations for all point, line, and area combinations are shown in Figure 9.6.

Figure 9.6 Examples of possible relations for two geographic database operators: (A) Contains; and (B) Touches operators.



In the case of the point–point Contain combination (upper left square in Figure 9.6A), two comparison geometry points (big circles) are contained in the set of base geometry points (small circles). In other words, the base geometry is a superset of the comparison geometry. In the case of the line–area Touches combination (middle right square in Figure 9.6B), the two lines touch the area because they intersect the area boundary. The full set of Boolean operators to test the spatial relationships between geometries is

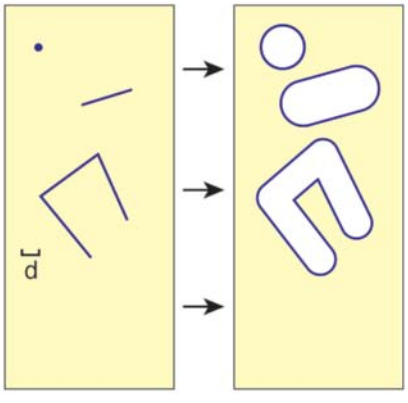
- Equals—are the geometries the same?
- Disjoint—do the geometries share a common point?
- Intersects—do the geometries intersect?
- Touches—do the geometries intersect at their boundaries?
- Crosses—do the geometries overlap (can be geometries of different dimensions, for example, lines and polygons)?
- Within—is one geometry within another?
- Contains—does one geometry completely contain another?
- Overlaps—do the geometries overlap (must be geometries of the same dimension)?
- Relate—are there intersections between the interior, boundary, or exterior of the geometries?

Seven methods support spatial analysis on these geometries. Four examples of these methods are shown in Figure 9.7.

- Distance—determines the shortest distance between any two points in two geometries (Section 13.3.1).
- Buffer—returns a geometry that represents all the points whose distance from the geometry is less than or equal to a user-defined distance (Section 13.3.2).
- ConvexHull—returns a geometry representing the convex hull of a geometry (a convex hull is the smallest polygon that can enclose another geometry without any concave areas: think of stretching an elastic band around the polygon).
- Intersection—returns a geometry that contains just the points common to both input geometries.
- Union—returns a geometry that contains all the points in both input geometries.
- Difference—returns a geometry containing the points that are different between the two geometries.
- SymDifference—returns a geometry containing the points that are in either of the input geometries, but not both.

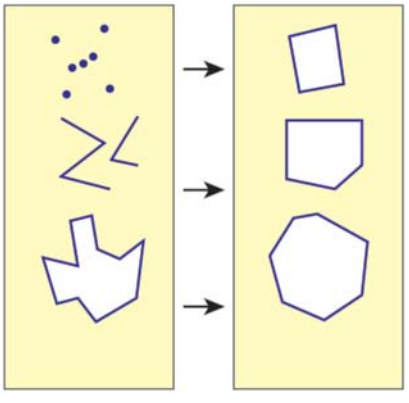
Figure 9.7 Examples of spatial analysis methods on geometries: (A) Buffer; (B) Convex Hull (*continued*)

(A) **Buffer**



Given a geometry and a buffer distance, the buffer operator returns a polygon that covers all points whose distance from the geometry is less than or equal to the buffer distance.

(B) **Convex Hull**



Given an input geometry, the convex hull operator returns a geometry that represents all points that are within all lines between all points in the input geometry.

A convex hull is the smallest polygon that wraps another geometry without any concave areas.

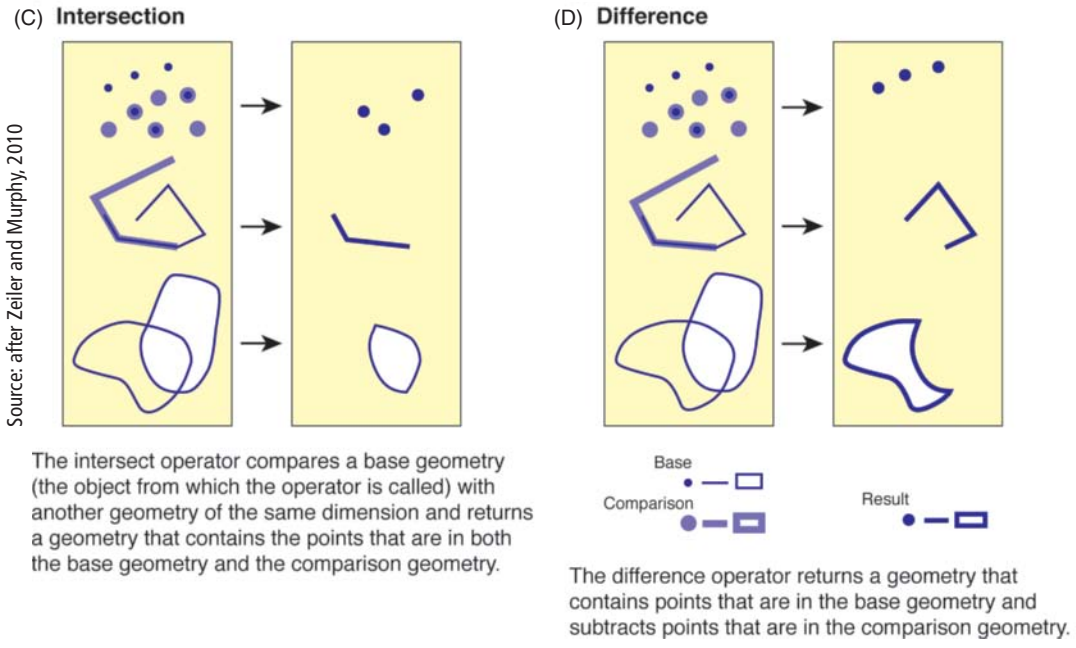


Figure 9.7 (*continued*) (C) Intersection; (D) Difference.

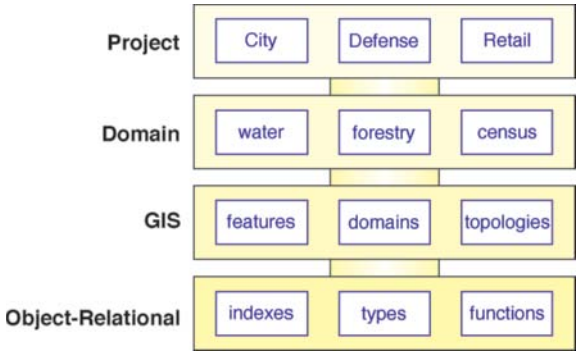
9.6 Geographic Database Design

This section is concerned with the technical aspects of logical and physical geographic database design. Chapter 7 provides an overview of these subjects, and Chapters 16 to 18 discuss the organizational, strategic, and business issues associated with designing and maintaining a geographic database.

9.6.1 The Database Design Process

All GI system and DBMS software packages have their own core data model that defines the object types and relationships that can be used in an application (Figure 9.8). A DBMS package will define and implement a model for data types and access functions, such as those implemented in SQL and discussed in Section 9.4. DBMSs are capable of dealing with simple geographic features and types (e.g., points, lines, areas, and sometimes also rasters) and relationships. A GI system can build on top of these simple feature types to create more advanced types and relationships (e.g., TINs, topologies, and feature-linked annotation geographic relationships; see Chapter 7 for a definition of these terms). The GI system types can be combined with domain data models that define specific object classes and relationships for specialist domains (e.g., water utilities, city parcel maps, and census geographies). Last, individual projects create specific physical instances of data models that are populated with data (objects for the specified object classes). For example, a city planning department may build a database of sewer lines that uses a water/wastewater (sewer) domain data model template that is built on top of core GI systems and DBMS models. Figure 7.2 and Section 7.1.2 show three increasingly abstract stages in data modeling: conceptual, logical, and physical. The result of a data-modeling exercise is a physical database design. This design will include specification of all data types and relationships, as well as the actual database configuration required to store them.

Figure 9.8 Four levels of data model available for use in GI systems projects.



Database design involves the creation of conceptual, logical, and physical models in the six practical steps that are shown in Figure 9.9.

Database design involves three key stages: conceptual, logical, and physical.

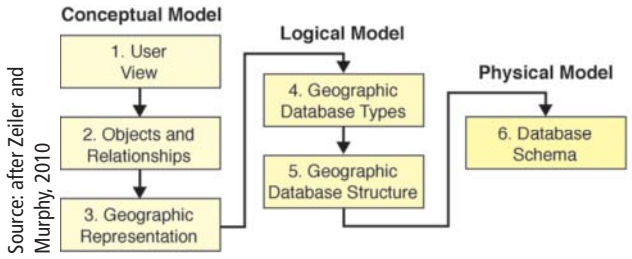


Figure 9.9 Stages in database design.

9.6.1.1 Conceptual Model

Model the User's View This involves tasks such as identifying organizational functions (e.g., controlling forestry resources, finding vacant land for new buildings, and maintaining highways), determining the data required to support these functions, and organizing the data into groups to facilitate data management. This information can be represented in many ways; for example, a report with accompanying tables is often used.

Define Objects and Their Relationships Once the functions of an organization have been defined, the object types (classes) and functions can be specified. The relationships between object types must also be described. This process usually benefits from the rigor of using object models and diagrams to describe a set of object classes and the relationships between them.

Select Geographic Representation Choosing the types of geographic representation (discrete object or continuous field; see Section 3.5) will have profound implications for the way a database is used, and so it is a critical database design task. It is, of course, possible to change between representation types, but this is computationally expensive and results in loss of information.

9.6.1.2 Logical Model

Match to Geographic Database Types This stage involves matching the object types to be studied to specific data types (point, line, area, georaster, etc.) supported by the GI systems that will be used to create and maintain the database. Because the data model of the GI system is usually independent of the actual storage mechanism (i.e., it could be implemented in Oracle, PostGIS, Microsoft Access, or a proprietary file system), this activity is defined as a logical modeling task.

Organize Geographic Database Structure This stage includes tasks such as defining topological associations, specifying rules and relationships, and assigning coordinate systems.

9.6.1.3 Physical Model

Define Database Schema The final stage is definition of the actual physical database schema that will hold the database data values. This is usually created using the DBMS software's Data Definition Language. The most popular of these is SQL with geographic extensions (see Section 9.4), although some nonstandard variants also exist in older DBMSs.

9.7 Structuring Geographic Information

Once data have been captured in a geographic database according to a schema defined in a geographic data model, it is often desirable to perform some structuring and organization in order to support efficient query, analysis, and mapping. There are two main structuring techniques relevant to geographic databases: topology creation and indexing.

9.7.1 Topology Creation

The subject of topology was covered in Section 7.2.3.2 from a conceptual data modeling perspective and is revisited here in the context of databases where the discussion focuses on the two main approaches to structuring and storing topology in a GI system DBMS.

Topology can be created for vector datasets using either batch or interactive techniques. Batch topology builders are required to handle CAD, survey, simple feature, and other unstructured vector data imported from nontopological systems. Creating topology is usually an iterative process because it is seldom possible to resolve all data problems during the first pass, and manual editing is required to make corrections. Some typical problems that may arise are shown in Figures 8.11, 8.12, and 8.13 and are discussed in Section 8.3.2.2. Interactive topology creation is performed dynamically at the time objects are added to a database using GI system editing software. For example, when adding water pipes using interactive vectorization tools (see Section 8.3.2.1), before each object is committed to the database topological connectivity can be checked to see if the object is valid (that is, it conforms to some preestablished rules for database object connectivity).

Two database-oriented approaches have emerged in recent years for storing and managing topology: normalized and physical. The normalized model focuses on the storage of an arc–node data structure. It is said to be normalized because each object is decomposed into individual topological primitives for storage in a database and then subsequent reassembly when a query is posed. For example, polygon

objects are assembled at query time by joining tables containing the line segment geometries and topological primitives that define topological relationships (see Section 7.2.3.2 for a conceptual description of this process). In the physical model, topological primitives are not stored in the database, and the entire geometry is stored together for each object. Topological relationships are then computed on-the-fly whenever they are required by client applications.

Figure 9.10 is a simple example of a set of database tables that store a dataset according to the normalized topology model. The dataset (sketch in top left corner) comprises three feature classes (Parcels, Buildings, and Walls) and is implemented in three tables. In this example the three feature class tables have only one column (ID) and one row (a single instance of each feature in a feature class). The Nodes, Edges, and Faces tables store the points, polylines, and polygons for the dataset and some of the topology (for each Edge the From–To connectivity and the Faces on the Left–Right in the direction of digitizing). Three other tables (Parcel X Face, Wall X Edge, and Building X Face) store the cross-references for assembling Parcels, Buildings, and Walls from the topological primitives.

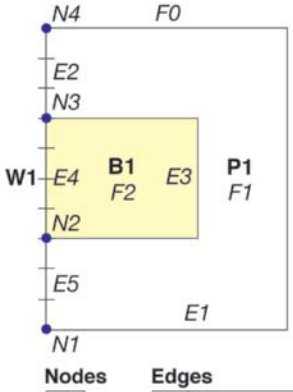
The normalized approach offers a number of advantages to GI systems users. Many people find it comforting to see topological primitives actually stored in the database. This model has many similarities to the arc–node conceptual topology model (see Section 7.2.3.2), and so it is familiar to many users and easy to understand. The geometry is only stored once, thus minimizing database size and avoiding "double digitizing" slivers (Section 8.3.2.2). Finally, the normalized approach easily lends itself to access via a SQL API. Unfortunately, there are three main disadvantages associated with the normalized approach to database topology: query performance, integrity checking, and update performance/complexity.

Query performance suffers because queries to retrieve features from the database (the most common type of query) must combine data from multiple tables. For example, to fetch the geometry of Parcel P1, a query must combine data from four tables (Parcels, Parcel X Face, Faces, and Edges) using complex geometry/topology logic. The more tables that have to be visited, and especially the more that have to be joined, the longer it will take to process a query.

The standard referential integrity rules in DBMS are very simple and have no provision for complex topological relationships of the type defined here. There are many pitfalls associated with implementing topological structuring using DBMS techniques such as stored procedures (program code stored in a database), and in practice systems have resorted to implementing the business logic to manage things like topology external to the DBMS in a middle-tier application server or a set or server-side program code.

Updates are similarly problematic because changes to a single feature will have cascading effects on many tables. This raises attendant performance (especially scalability, that is, large numbers of concurrent queries) and integrity issues. Moreover, it is uncertain how multiuser updates will be handled that entail long transactions with design alternatives (see Sections 9.9 and 9.9.1 for coverage of these two important topics). For comparative purposes the same dataset used in the normalized model (Figure 9.10) is implemented using the physical model in Figure 9.11. In the physical model the three feature classes (Parcels, Buildings, and Walls) contain the same IDs, but differ significantly

Figure 9.10 Normalized database topology model.



Parcels

ID
P1

Buildings

ID
B1

Walls

ID
W1

Parcel x Face

Parcel	Face
P1	F1
P1	F2

Wall x Edge

Wall	Edge	Order	Orientation
W1	E2	1	+
W1	E4	2	-
W1	E5	3	+

Building x Face

Parcel	Face
B1	F2

Nodes

ID
N1
N2
N3
N4

Edges

ID	Vertices	From	To	Left	Right
E1	(0,10),(8,10),(8,0),(0,0)	N4	N1	F0	F1
E2	(0,10),(0,7)	N4	N3	F1	F0
E3	(0,7),(5,7),(5,3),(0,3)	N3	N2	F1	F2
E4	(0,3),(0,7)	N2	N3	F0	F2
E5	(0,3),(0,0)	N2	N1	F1	F0

Faces

ID
F0
F1
F2

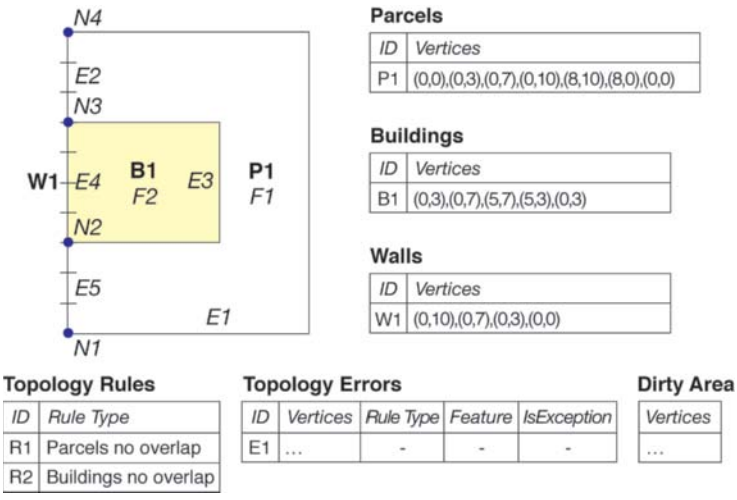


Figure 9.11 Physical database topology model.

in that they also contain the geometry for each feature. The only other things required to be stored in the database are the specific set of topology rules that have been applied to the dataset (e.g., parcels should not overlap each other, and buildings should not overlap with each other), together with information about known errors (sometimes users defer topology cleanup and commit data with known errors to a database) and areas that have been edited but not yet validated (had their topology (re)built).

The physical model requires that an external client or middle-tier application server is responsible for validating the topological integrity of datasets. Topologically correct features are then stored in the database using a structure that is much simpler than the normalized model. When compared to the normalized model, the physical model offers two main advantages of simplicity and performance. Because all the geometry for each feature is stored in the same table column/row value, and there is no need to store topological primitives and cross-references, this is a very simple model. The biggest advantage and the reason for the appeal of this approach is query performance. Most DBMS queries do not need access to the topology primitives of features, and so the overhead of visiting and joining multiple tables is unnecessary. Even when topology is required, it is faster to retrieve feature geometries and recompute topology outside the database than to retrieve geometry and topology from the database.

In summary, the normalized and physical topology models have both advantages and disadvantages. The normalized model is implemented in Oracle Spatial and can be accessed via a SQL API, making it easily available to a wide variety of users. The physical model is implemented in ArcGIS and offers fast update and query performance for high-end GI system applications. Esri has also implemented a long transaction and versioning model based on the physical database topology model.

9.7.2 Indexing

Geographic databases tend to be very large and geographic queries computationally expensive. As a result, geographic queries, such as finding all the customers (points) within a store trade area (polygon), can take a very long time (perhaps 10 to 100 seconds or more for a 50 million customer database). The point has already been made in Section 7.2.3.2 that topological structuring can help speed up certain types of queries such as adjacency and network connectivity. A second way to speed up queries is to index a database and use the index to find data records (database table rows) more quickly. A database index is logically similar to a book index; both are special organizations that speed up searching by allowing random instead of sequential access. A database index is, conceptually speaking, an ordered list derived from the data in a table. Using an index to find data reduces the number of computational tests that have to be performed to locate a given set of records. In DBMS jargon, indexes avoid expensive full-table scans (reading every row in a table) by creating an index and storing it as a table column.

> **A database index is a special representation of information about objects that improves searching.**

Figure 9.12 presents a simple example of the standard DBMS one-dimensional B-tree (balanced tree) index that is found in most major commercial DBMSs. Without an index, a search of the original data in this table to guarantee finding any given value would involve 16 tests/read operations (one for each data point). The B-tree index orders the data

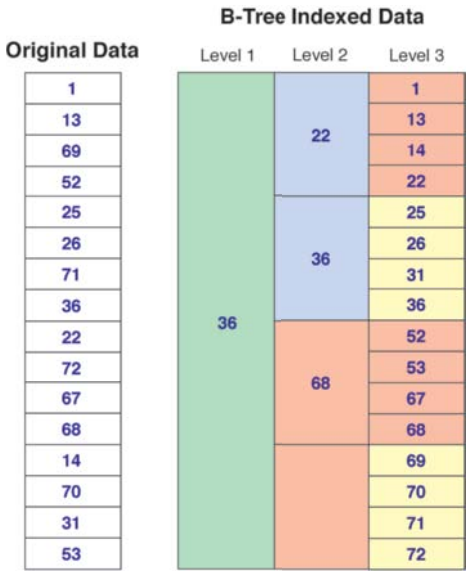


Figure 9.12 An example of a B-tree index.

and splits the ordered list into buckets of a given size (in this example it is four and then two), and the upper value for the bucket is stored (it is not necessary to store the uppermost value). To find a specific value, such as 72, using the index involves a maximum of six tests: one at Level 1 (less than or greater than 36), one at Level 2 (less than or greater than 68), and a sequential read of four records at Level 3. The number of levels and buckets for each level can be optimized for each type of data. Typically, the larger the dataset, the more effective indexes are in retrieval performance.

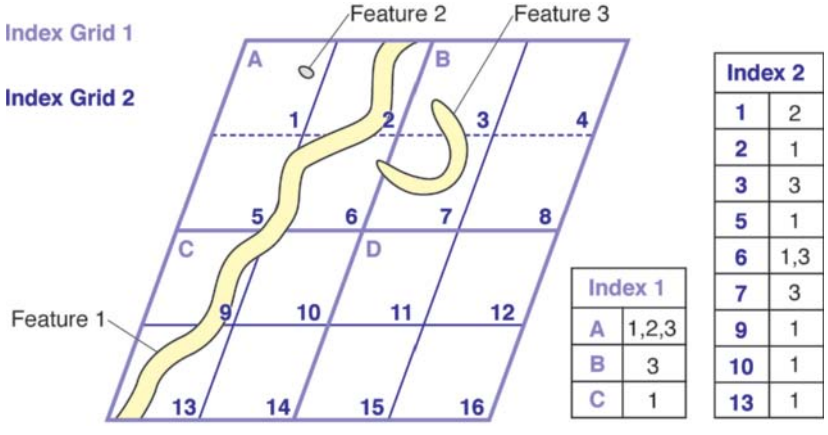
Unfortunately, creating and maintaining indexes can be quite time consuming; this is especially an issue when the data are very frequently updated. Because indexes can occupy considerable amounts of disk space storage, requirements can be very demanding for large datasets. As a consequence, many different types of index have been developed to try to alleviate these problems for both geographic and nongeographic data. Some indexes exploit specific characteristics of data to deliver optimum query performance, others are fast to update, and still others are robust across widely different data types.

The standard indexes in DBMSs, such as the B-tree, are one dimensional and are very poor at indexing geographic objects. Many types of geographic indexing techniques have been developed, some of which are experimental and have been highly optimized for specific types of geographic data. Research shows that even a basic spatial index will yield very significant improvements in spatial data access and that further refinements often yield only marginal improvements at the costs of simplicity, as well as speed of generation and update. Three main methods of general practical importance have emerged in GI systems: grid indexes, quadtrees, and R-trees.

9.7.2.1 Grid Index

A grid index can be thought of as a regular mesh placed over a layer of geographic objects. Figure 9.13 shows a layer that has three features indexed using two grid levels. The highest (coarsest) grid (Index 1) splits the layer into four equal-sized cells. Cell A includes parts of Features 1, 2, and 3; Cell B includes a part of Feature 3; and Cell C has part of Feature 1. There are no features on Cell D. The same process is repeated for the second-level index (Index 2). A query to locate an object searches the indexed list first to find the object and then retrieves the object geometry or attributes for further analysis (e.g., tests for overlap, adjacency, or containment with other objects on the same or another layer). These two tests are often referred to as primary and secondary filters. Secondary filtering, which involves geometric processing, is much more computationally expensive. The performance of an index is clearly related to the relationship between grid and

Figure 9.13 A multilevel grid geographic database index.



object sizes, and on object density. If the grid size is too large relative to the size of object, too many objects will be retrieved by the primary filter, and therefore a lot of expensive secondary processing will be needed. If the grid size is too small, any large objects will be spread across many grid cells, which is inefficient for draw queries (queries made to the database for the purpose of displaying objects on a screen). For data layers that have a highly variable object density (for example, administrative areas tend to be smaller and more numerous in urban areas in order to equalize population counts), multiple levels can be used to optimize performance. Experience suggests that three grid levels are normally sufficient for good all-round performance. Grid indexes are one of the simplest and most robust indexing methods. They are fast to create and update and can handle a wide range of types and densities of data. For this reason they have been quite widely used in commercial GI systems.

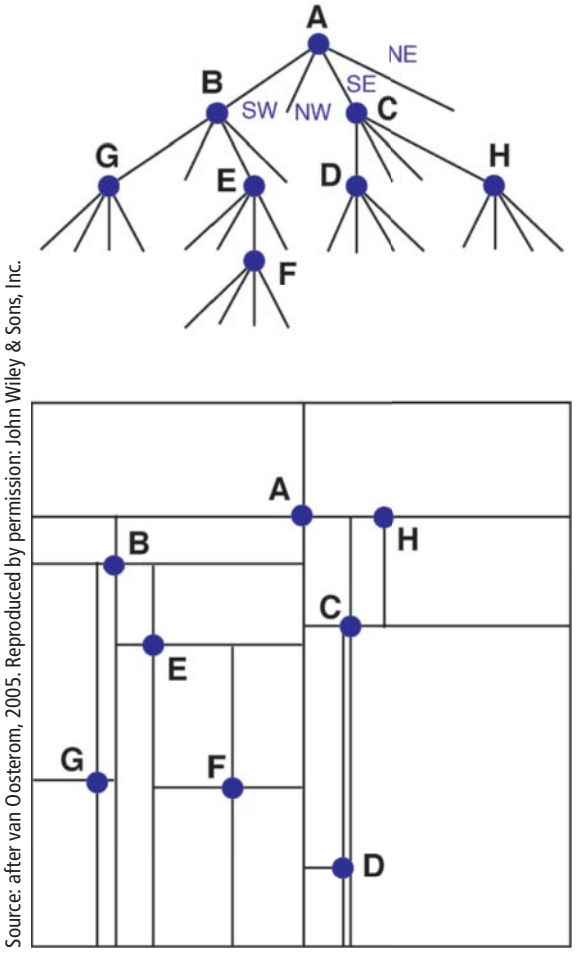
Grid indexes are easy to create, can deal with a wide range of object types, and offer good performance.

9.7.2.2 Quadtree Indexes

Quadtree is a generic name for several kinds of indexes that are built by recursive division of space into quadrants. In many respects, quadtrees are a special type of grid index. The difference here is that in quadtrees space is always recursively split into four quadrants based on data density. Quadtrees are data structures used for both indexing and compressing geographic database layers, although the discussion here will relate only to indexing. The many types of quadtrees can be classified according to the types of data that are indexed (points, lines, areas, surfaces, or rasters), the algorithm that is used to decompose (divide) the layer being indexed, and whether fixed or variable resolution decomposition is used.

In a point quadtree, space is divided successively into four rectangles based on the location of the points (Figure 9.14). The root of the tree corresponds to the region as a whole. The rectangular region is divided into four usually irregular parts based on the (*x*,*y*) coordinates of the first point. Successive points subdivide each new subregion into quadrants until all the points are indexed.

Region quadtrees are commonly used to index lines, areas, and rasters. The quadtree index is created by successively dividing a layer into quadrants. If a quadrant cell is not completely filled by an object, then it is subdivided again. Figure 9.15 is a quadtree of a woodland (red) and water (white) layer. Once a layer has been decomposed in this way, a linear index can be created using the search order shown in Figure 9.16. By reducing two-dimensional



Source: after van Oosterom, 2005. Reproduced by permission: John Wiley & Sons, Inc.

Figure 9.14 The point quadtree geographic database index.

geographic data to a single linear dimension, a standard B-tree can be used to find data quickly.

Quadtrees have found favor in GI systems because of their applicability to many types of data (both raster and vector), their ease of implementation, and their relatively good improvements in search performance.

9.7.2.3 R-tree Indexes

R-trees group objects using a rectangular approximation of their location called a minimum bounding rectangle (MBR) or minimum enclosing rectangle (see Box 9.2). Groups of point, line, or area objects are indexed based on their MBR. Objects are added to the index by choosing the MBR that would require the least expansion to accommodate each new object. If the object causes the MBR to be expanded beyond some preset parameter, then the MBR is split into two new MBRs. This may also cause the parent MBR to

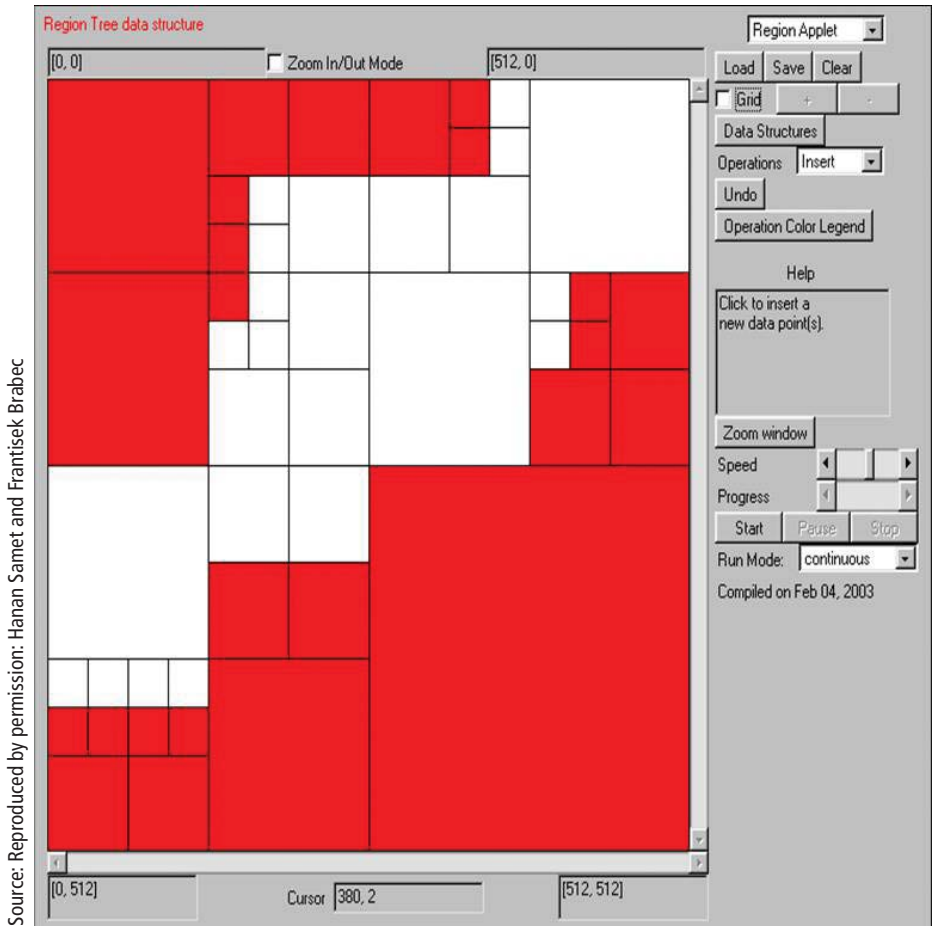


Figure 9.15 The region quadtree geographic database index.

become too large, resulting in this also being split. The R-tree shown in Figure 9.17 has two levels. The lowest level contains three "leaf nodes"; the highest has one node with pointers to the MBR of the leaf nodes. The MBR is used to reduce the number of objects that need to be examined in order to satisfy a query.

R-trees are popular methods of indexing geographic data because of their flexibility and excellent performance.

R-trees are suitable for a range of geographic object types and can be used for multidimensional data. They offer good query performance, but the speed of update is not as fast as for grids and quadtrees. The spatial extensions to the IBM Informix DBMS and Oracle Spatial both use R-tree indexes.

Figure 9.16 Linear quadtree search order.

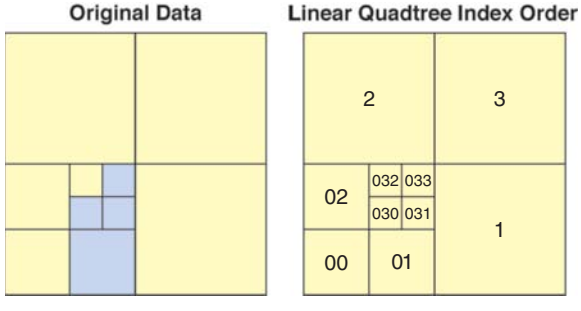
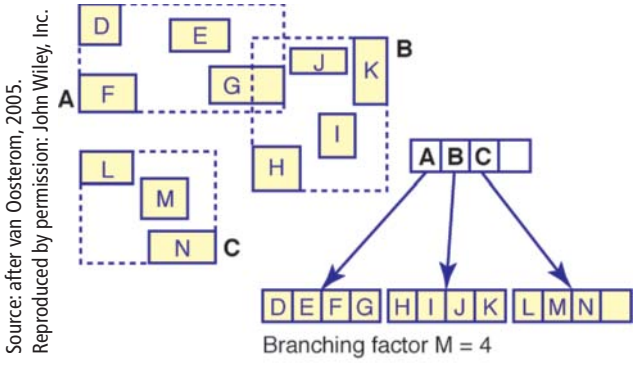


Figure 9.17 The R-tree geographic database index.



Technical Box 9.2

Minimum Bounding Rectangle

Minimum bounding rectangles (MBRs) are very useful structures that are widely implemented in GI systems. An MBR essentially defines the smallest box whose sides are parallel to the axes of the coordinate system that encloses a set of one or more geographic objects. It is defined by the two coordinates at the bottom left (minimum *x*, minimum *y*) and the top right (maximum *x*, maximum *y*), as is shown in Figure 9.18.

MBRs can be used to generalize a set of data by replacing the geometry of each of the objects in the box with two pairs of coordinates defining the box. A second use is for fast searching. For example, all the area objects in a database layer that are within a given study area can be found by performing an area-on-area contains test (see Figure 9.18) for each object and the study area boundary. If the area objects have complex boundaries (as is normally the case in GI systems), this can be a very time-consuming task. A quicker approach is to split the task into two parts. First, screen out all the objects that are definitely in and definitely out by comparing their MBRs. Because very few coordinate comparisons are required, this is very fast. Then use the full geometry outline of the remaining area objects to determine containment. This is computationally expensive for areas with complex geometries.

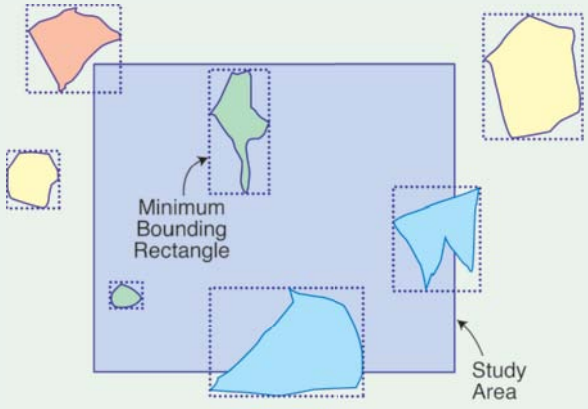


Figure 9.18 Area in area test using MBR. An MBR can be used to determine objects definitely in the study area (green) because of no overlap, definitely out (yellow), or possibly in (blue). Objects possibly in can then be analyzed further using their exact geometries. Note the red object that is actually completely outside, although the MBR suggests it may be partially within the study area.

9.8 Editing and Data Maintenance

Editing is the process of making changes to a geographic database by adding new objects or changing existing objects as part of data load or database update and maintenance operations. A database update is any change to the geometry and/or attribute of one or more objects, a change to an object relationship, or any change to the database schema. A general-purpose geographic database will require many tools for tasks such as geometry and attribute editing, database maintenance (e.g., system administration and tuning), creating and updating indexes and topology, importing and exporting data, and georeferencing objects.

Contemporary GI systems come equipped with an extensive array of tools for creating and editing geographic object geometries and attributes. These tools form workflow tasks that are exposed within the framework of a WYSIWYG (what you see is what you get) editing environment. The objects are displayed using map symbology, usually in a projected coordinate system space and frequently on top of "background" layers such as aerial photographs or satellite images, or street centerline files. Object coordinates can be digitized into a geographic database using many methods, including freehand digitizing on a digitizing table, on-screen heads-up vector digitizing by copying existing raster or vector sources, on-screen semiautomatic line following, automated feature recognition, and reading survey measurements from an instrument (e.g., GPS or Total Station) file (see Sections 4.9 and 8.2.2.1). The end result is always a layer of (*x*,*y*) coordinates with optional *z* and *m* (height and attribute) values. Similar tools also exist for loading and editing raster data.

Data entered into the editor must be stored persistently in a file system or database, and access to the database must be carefully managed to ensure continued security and quality. The mechanism for managing edits to a file or database is called a transaction. There are many challenging issues associated with implementing multiuser access to geographic data stored in a DBMS, as discussed in the next section.

9.9 Multiuser Editing of Continuous Databases

For many years, one of the most challenging problems in GI data management was how to allow multiple users to edit the same continuous geographic database at the same time. In GI applications the objects of interest (geographic features) do not exist in isolation and are usually closely related to surrounding objects. For example, a tax parcel will share a common boundary with an adjacent parcel, and changes in one will directly affect the other; similarly, connected road segments in a highway network need to be edited together to ensure continued connectivity. It is straightforward to provide multiple users with concurrent read and query access to a continuous shared database, but more difficult to deal with conflicts and avoid potential database corruption when multiple users want write (update) access. However, solutions to both of these problems have been implemented in mainstream GI systems and DBMS. These solutions extend standard DBMS transaction models and provide a multiuser framework called versioning.

9.9.1 Transactions

A group of edits to a database, such as the addition of three new land parcels and changes to the attributes of a sewer line, is referred to as a "transaction." In order to protect the integrity of databases, transactions are atomic; that is, transactions are either completely committed to the database or they are rolled back (not committed at all). Many of the world's GI and non-GI databases are multiuser and transactional; that is, they have multiple users performing edit/update operations at the same time. For most types of databases, transactions take a very short (subsecond) time. For example, in the case of a banking system, a transfer from a savings account to a checking account takes perhaps 0.001 second. It is important that the transaction is coordinated between the accounts and that it is atomic; otherwise one account might be debited and the other not credited. Multiuser access to banking and similar systems is handled simply by locking (preventing access to) affected database records (table rows) during the course of the transaction. Any attempt to write to the same record is simply postponed until the record lock is removed after the transaction is completed. Because banking transactions, like many other transactions, take only a very short amount of time, users never even notice whether a transaction is deferred.

A transaction is a group of changes that are made to a database as a coherent group. All the changes that form part of a transaction are either committed, or the database is rolled back to its initial state.

Although some geographic transactions have a short duration (short transactions), many extend to hours, weeks, and months and are called long transactions. Consider, for example, the amount of time necessary to capture all the land parcels in a city subdivision (housing estate). This might take a few hours for an efficient operator working on a small subdivision, but an inexperienced operator working on a large subdivision might take days or weeks. This may cause three multiuser update problems. First, locking the whole or even part of a database to other updates for this length of time during a long transaction is unacceptable in many types of applications, especially those involving frequent maintenance changes (e.g., utilities and land administration). Second, if a system failure occurs during the editing, work may be lost unless there is a procedure for storing updates in the database. Also, unless the data are stored in the database, they are not easily accessible to others who would like to use them.

9.9.2 Versioning

Short transactions use what is called a pessimistic locking concurrency strategy. That is, it is assumed that conflicts will occur in a multiuser database with concurrent users and that the only way to avoid database corruption is to lock out all but one user during an update operation. The term *pessimistic* is used because this is a very conservative strategy, assuming that update conflicts will occur and that they must be avoided at all costs. An alternative to pessimistic locking is optimistic versioning, which allows multiple users to update a database at the same time. Optimistic versioning is based on the assumption that conflicts are very unlikely to occur, but if they do occur then software can be used to resolve them.

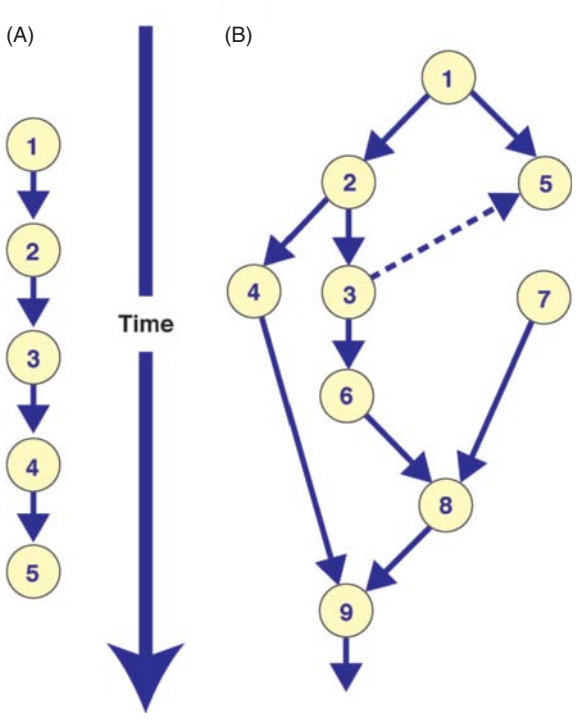
The two strategies for providing multiuser access to geographic databases are pessimistic locking and optimistic versioning.

Versioning sets out to solve the long transaction and pessimistic locking concurrency problem that was described earlier in this chapter. It also addresses a second key requirement peculiar to geographic databases—the need to support alternative representations of the same objects in the database. In some important geographic applications, it is a requirement to allow designers to create and maintain multiple object designs. For example, when designing a new housing subdivision, the water department manager may ask two designers to lay out alternative designs for a water system. The two designers would work concurrently to add objects to the same database layers, snapping to the same objects. At some point, they

may wish to compare designs and perhaps create a third design based on parts of their two designs. While this design process is taking place, operational maintenance editors could be changing the same objects with which they are working. For example, maintenance updates resulting from new service connections or repairs to broken pipes will change the database and may affect the objects used in the new designs.

Figure 9.19 compares linear short transactions and branching long transactions as implemented in a versioned database. Within a versioned database, the different database versions are logical copies of their parents (base tables); that is, only the modifications (additions and deletions) are stored in the database (in version change tables). A query against a database version combines data from the base table with data in the version change tables. The process of creating two versions based on the same parent version is called branching. In Figure 9.19B, Version 4 is a branch from Version 2. Conversely, the process of combining two versions into one version is called merging. Figure 9.19B also illustrates the merging of Versions 6 and 7 into Version 8. A version can be updated at any time with any changes made in another version. Version reconciliation can be seen between Versions 3 and 5. Because the edits contained within Version 5 were reconciled with 3, only the edits in Versions 6 and 7 are considered when merging to create Version 8.

Figure 9.19 Database transactions: (A) linear short transactions; (B) branching version tree.



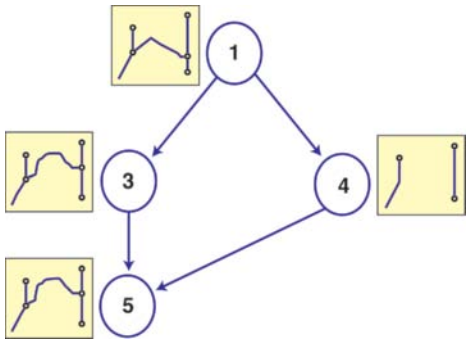


Figure 9.20 Version reconciliation. For Version 5 the user chooses via the GUI the geometry edit made in Version 3 instead of 1 or 4.

There are no database restrictions or locks placed on the operations performed on each version in the database. The versioning database schema isolates changes made during the edit process. With optimistic versioning, it is possible for conflicting edits to be made within two separate versions, although normal working practice will ensure that the vast majority of edits made will not result in any conflicts (Figure 9.20). In the event that conflicts are detected, the data management software will handle them either automatically or interactively. If interactive conflict resolution is chosen, the user is directed to each feature that is in conflict and must decide how to reconcile the conflict. The GUI will provide information about the conflict and display the objects in their various states. For example, if the geometry of an object has been edited in the two versions, the user can display the geometry as it was in any of its previous states and then select the geometry to be added to the new database state.

9.10 Conclusion

Database management systems are now a vital part of large modern operational GI systems as well as related interdisciplinary fields such as computer science and engineering (see Box 9.3). They bring with them standardized approaches for storing and, more important, accessing and manipulating geographic data using the SQL query language. GI systems provide the necessary tools to load, edit, query, analyze, and display geographic data. DBMSs require a database administrator (DBA) to control database structure and security and to tune the database to achieve maximum performance. Innovative work in the GI system field has extended standard DBMS to store and manage geographic data and has led to the development of long transactions and versioning that have application across several fields.

Biographical Box 9.3

Oliver Günther, Computer Scientist

Oliver Günther (Figure 9.21) received a master's degree in industrial engineering from Karlsruhe Institute of Technology and M.S. and Ph.D. degrees in computer science from the University of California at Berkeley. After some time as junior faculty at UC Santa Barbara and its National Center for Geographic Information and Analysis (NCGIA), he set up a research group on environmental information systems at FAW Ulm (Germany). From 1993 until 2011 he held the chair of information systems at Humboldt-Universität zu Berlin. From 2005–2011 he also served as Dean of Humboldt's School of Business and Economics. In January 2012 he was appointed President of the University of Potsdam. Oliver's Ph.D. thesis, "Efficient Structures in Geometric Data Management" (Springer-Verlag, 1988), provided some theoretical foundations for spatial data management with a particular focus on index structures. His ACM Computing Surveys article on multidimensional access methods (1998, with Volker Gäde) remains one of the most-read references on the subject (Figure 9.22). His textbook, *Environmental Information Systems* appeared at about the same time (Springer-Verlag, 1998). In the late 1990s he contributed to the design of the first German Environmental Data Catalogue (UDK). His more recent work focuses on privacy and security issues. Oliver Günther has

Source: Oliver Günther

Figure 9.21 Oliver Günther, Computer Scientist.

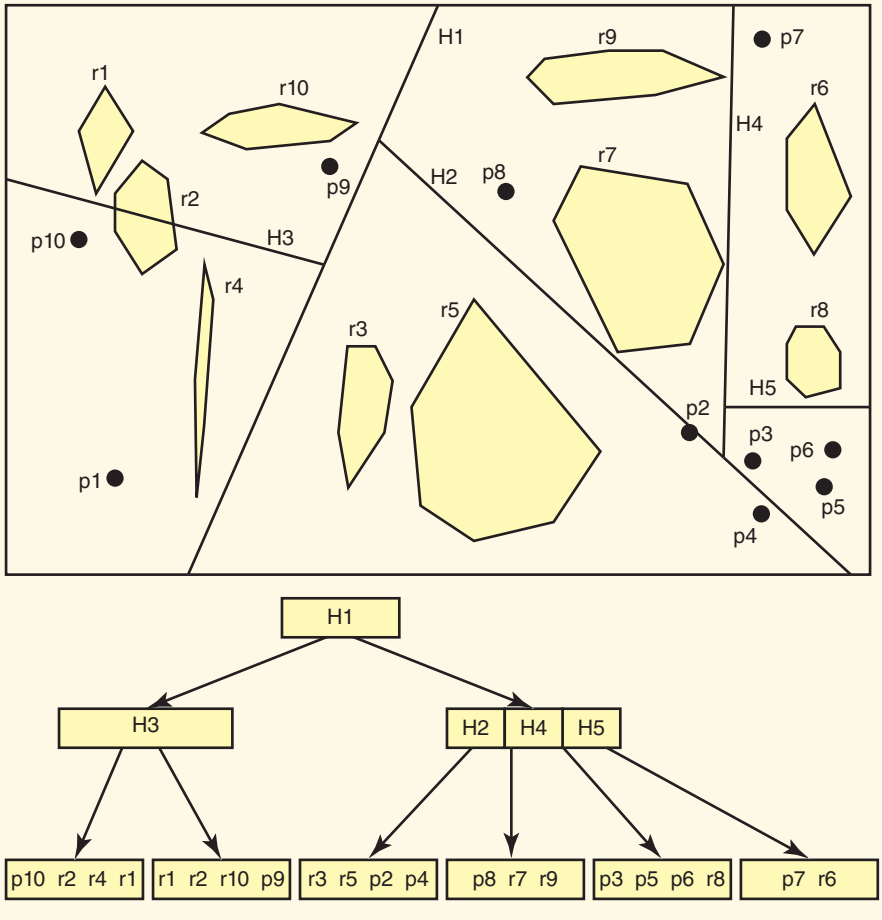


Figure 9.22 Cell tree data indexing structure. In the cell tree, each geometric object O is represented by a set of possibly overlapping convex cells whose union equals O. Cell tree nodes are split if they do not fit on one disk page anymore. The algorithm then looks for a splitting hyperplane that balances the entries approximately evenly among the two resulting subspaces and intersects a minimum number of cells.

held visiting faculty positions at the European School of Management and Technology in Berlin, the École Nationale Supérieure des Télécommunications in Paris, at UC Berkeley, and the University of Cape Town. He has served as a consultant and board member to various government agencies and high-tech companies, including Poptel AG, Germany's first voice-over-IP company. In 2011–2012 he served as President of the German Informatics Society. As a high school student, Oliver was a two-time winner of Germany's National Mathematics Competition.

Looking back at the evolution of geographic and environmental information systems, Oliver remarks:

> When I first thought about GI systems and environmental applications, I differentiated between four distinct phases: data capture, data aggregation, data storage, and data analysis. Many of the problems we faced in the 1980s have now been solved. Data capture has changed fundamentally due to the ubiquitous availability of all kinds of sensors. Data aggregation and also data analysis have profited from the excellent work of our data mining colleagues. Algorithms for routing and map generalization have also improved greatly, which led to the high-performance navigation systems which we got used to very quickly. Data storage has become more efficient, partly due to the smart data structures we developed back then. Now the ability to load databases into main memory will change the big picture one more time. Metadata management, finally, has become a crucial component of any major information system, taking care of search and access control. However, these technical advances must be balanced with rising concerns about privacy and security. Overall, GI systems and environmental applications have turned from a niche application to an essential part of our information infrastructure. It was nice to be able to contribute somewhat to these developments.

Questions for Further Study

1. Identify a geographic database with multiple layers and draw a diagram showing the tables and the relationships between them. Which are the primary keys, and which keys are used to join tables? Does the database have a good relational design?
2. What are the advantages and disadvantages of storing geographic data in a DBMS?
3. Is SQL a good language for querying geographic databases?
4. Why are there multiple methods of indexing geographic databases?

Further Reading

Date, C. J. 2005. *Database in Depth: Relational Theory for Practitioners*. Sebastopol, CA: O'Reilly Media, Inc.

Hoel, E., Menon, S., and Morehouse, S. 2003. Building a robust relational implementation of topology. In Hadzilacos, T., Manolopoulos, Y., Roddick, J. F., and Theodoridis, Y. (eds.). *Advances in Spatial and Temporal Databases. Proceedings of 8th International Symposium, SSTD 2003 Lecture Notes in Computer Science,* Vol. 2750.

OGC. 1999. *OpenGIS Simple Features Specification for SQL, Revision 1.1.* Available at www.opengis.org.

Samet, H. 2006. *The Foundations of Multidimensional and Metric Data Structures*. San Francisco: Morgan-Kaufmann.

Shekar, S. and Chawla, S. 2003. *Spatial Databases: A Tour.* Upper Saddle River, NJ: Prentice Hall.

van Oosterom, P. 2005. Spatial access methods. In Longley, P. A., Goodchild, M. F., Maguire, D. J., and Rhind, D. W. (eds.) *Geographic Information Systems: Principles, Techniques, Applications, and Management* (abridged ed.). Hoboken, NJ: Wiley. 385–400.

Yeung, A. K. W. and Hall, G. B. 2007. *Spatial Database Systems: Design, Implementation and Project Management* (GeoJournal Library). Dordrecht: Springer.

Zeiler, M. and Murphy, J. 2010. *Modeling Our World: The ESRI Guide to Geodatabase Concepts* (2nd ed.). Redlands, CA: ESRI Press.