6

Data Models

6.1 Geospatial Data is Special

One of the reasons there is a steep learning curve for GIS is due to the diversity of the geospatial and attribute data formats involved with most GIS projects. Some of the commonly used geospatial formats include shapefiles, geodatabases, KMLs, GeoTIFFs, ERDAS imagine files (.img), AutoCAD Drawing Exchange Format files (.dxf), GPS Exchange Format files (.gpx), and GeoJSON files. GIS data can also be in tabular form (e.g., Excel, CSV, TXT, and dBase files). This diversity comes from the different fields (e.g., image processing, computer-aided design, and database management system) that have contributed to the genesis of the geospatial industry and due to the complexity of geospatial data itself. The complexity can be contrasted alongside attribute data in three respects. First, attribute data stores only nonspatial attributes, while geospatial data stores spatial aspects as well as nonspatial attributes. For example, mapping populations by county requires storing county boundaries (spatial objects) as well as populations (attributes) in GIS. Second, compared to attribute data, which is typically stored as a table, there is no unified method to encode the spatial aspects of the world. Some geographic phenomena are spatially continuous and amorphous (e.g., temperature, precipitation, carbon dioxide) whereas some geographic entities are spatially discrete and have well-defined shapes (e.g., land parcel, roads, buildings). Therefore, a single method cannot adequately represent geographic phenomena. Lastly, spatial data is multidimensional and representation of spatial entities is scale-dependent. For instance, the spatial aspects of a building can be represented in multiple dimensions (x, y, z) whereas nonspatial aspects (e.g., name, ownership, number of floors) are represented with one dimension for each attribute. Buildings can be represented as a zero-dimensional point in a city map but can be represented as a 2D polygon in a detailed land parcel map. A 3D representation of a building is required for virtual globe software such as Google Earth. All this complexity explains why unique methods have been developed to represent geography in GIS.

Most GIS software has functionality for loading (and editing) geospatial file formats or has conversion tools for different formats. For example, ArcGIS can directly display IMG and GeoTIFF files and convert KML, DXF, and GPX files to proprietary formats (shapefile and geodatabase). Open-source QGIS can directly load GeoJSON and OpenStreetMap (OSM) file formats. While proprietary GIS file formats are still common, recent trends are toward open standards that facilitate easy exchange of data and protocols among the general public (e.g., KML, GeoJSON) that can be exchanged across platforms and among multiple users on the web. In the following sections, we will look at how geospatial entities and phenomena are encoded in a GIS database. More specifically, we will review how spatial objects, events, and processes are structured in data (Section 6.2), how attribute data are stored (Section 6.3), and how spatiality and attributes are all encoded in geospatial data (Section 6.4). This chapter will help you make sense of the different data formats and competently work with both spatial and attribute data in GIS.

6.2 How Spatial Data Is Structured—Raster and Vector

Spatial aspects of the world are structured as data in GIS. Data model (or data structure) is used to encode aspects of entities that are relevant to an application domain. There are two fundamentally different data structures used to represent the spatial aspects of the world—raster and vector. Figure 6.1 illustrates the difference between raster and vector data structures in representing land cover such as forest and water. Raster data models divide space into pixels of uniform size and assigns attribute values (land cover) to each pixel. For example, the four pixels that cover the lower right are encoded as water for raster data in Figure 6.1. Vector data models delineate boundaries for different land covers and assigns other additional attribute values to each land cover area. A line forming boundaries for vector data is made of a set of points (vertices). The building block of raster data models is a pixel and that of vector data models is a point. Both building blocks are associated with (x, y) coordinates in reference to a spatial reference system as discussed in Chapter 2.

In Figure 6.1, raster data structure was used to encode a representative land cover value for each pixel since boundaries among different land covers are drawn along pixels. Therefore, pixelated lines between two different land covers (e.g., between water and beach) approximate the transition from one land cover to another land cover. The smaller the pixel size the more precise the land cover will be represented, but this precision is achieved at the expense of increased data size. Although the vector data structure might look more accurate in this example, it is not safe to say that vector representation is more accurate than raster as it is often difficult to delineate

boundaries among different land covers. However, it suffices to say that both vector and raster data structures have their strengths and weaknesses in representing geography. Since raster encodes attribute values given a controlled spatial unit (pixel), it is suitable for representing continuous and amorphous phenomenon such as temperature, where the pixel serves as the spatial sampling unit. By contrast, vector works well for representing spatial entities with well-defined shapes due to a relative emphasis on spatial forms.

Raster and vector are also different in how (nonspatial) attributes are stored. For vector, attributes are typically stored as a table, where a row (record) is linked to a spatial object (land cover area). Multiple attributes such as Name, Public, and Owner are assigned to each land cover area in vector. For example, the second row in the table for vector in Figure 6.1 states that water is Public and Owner is state. For raster that record only one attribute (land cover) per pixel, it would be tedious to store an attribute value per pixel as a row in a table since there are too many pixels (rows) with the same value. Instead, if there is a table for raster it summarizes how many pixels exist for each unique pixel value. The fifth row in the table for raster in Figure 6.1 tells us that there are four pixels (see the column Count) for water. As vector and raster are distinctly different in how they represent both spatial and nonspatial aspects, the ways in which a query is performed and analysis is conducted are also different. For instance, subsets of vector data are selected using query tools (e.g., Select by Attribute in ArcGIS), but those of raster data are selected using a map algebra tool (e.g., Raster Calculator in ArcGIS).

Due to the relative focus of vector model on shape, vector data is associated with different geometry types (or dimensionality)—point, line, and polygon.

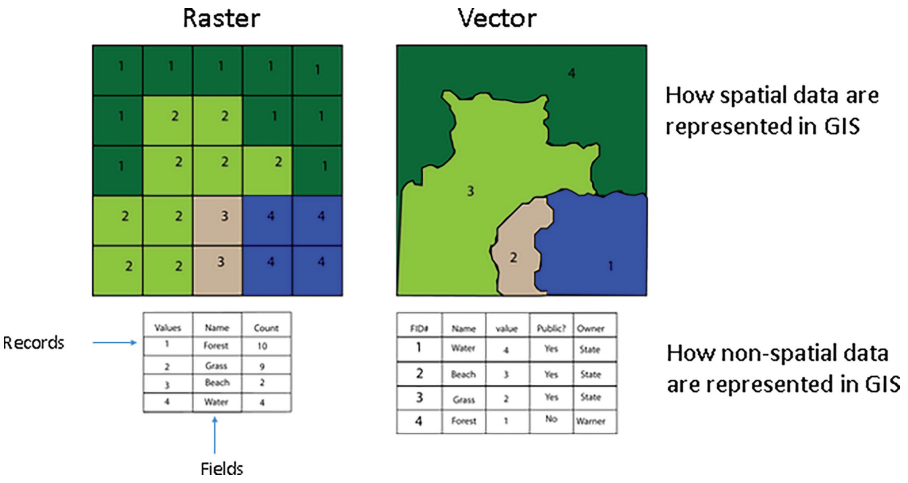


FIGURE 6.1
Data structure for spatial data: raster vs. vector.

Point (zero-dimensional) data is good for storing the location of any point of interest or event; line (1D) data can store length and direction such as roads. Polygon (2D) data is used to represent areas with size and shape. Figure 6.2 shows that vector data is comprised of points with (x, y) coordinates. Each line (or polyline) is made of a series of four vertices. Polygon data is made of a series of (x, y) coordinates where the first coordinate is the same as the last coordinate. Geometry type of vector data is largely dependent on intended use as well as geographic scale in which the data is displayed. Roads can be represented as polygons at local scale or for detailed cartographic display but can also be represented as connected linear features at regional scale or for routing applications.

To find the shortest path between two points along a transportation network, the transportation network should be represented as a network data model, which represents it as a set of nodes (junctions or intersections) and

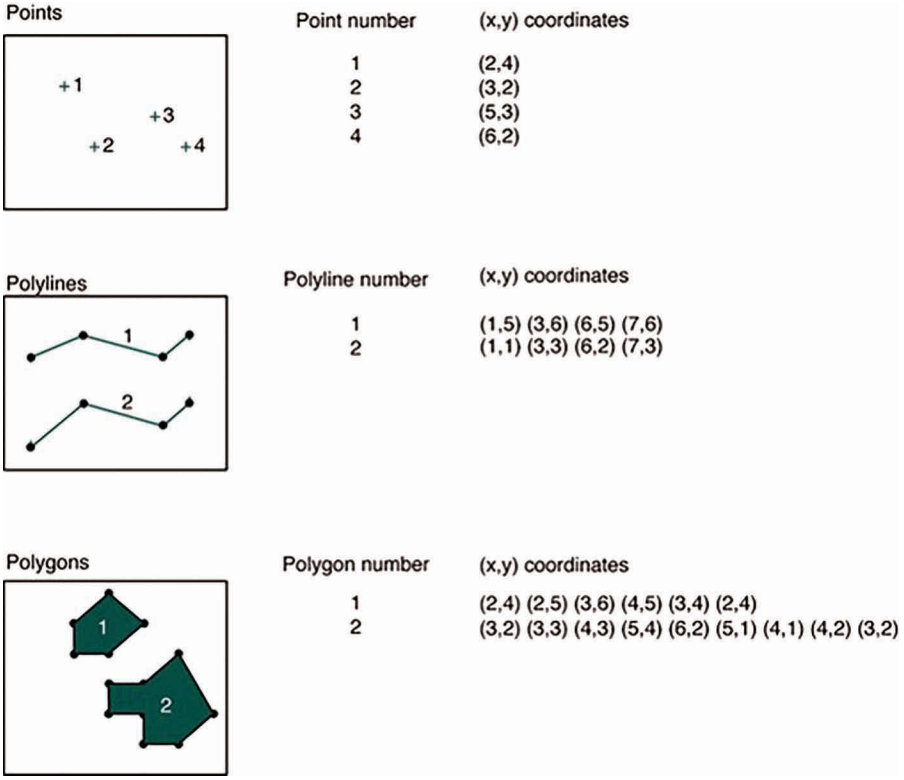
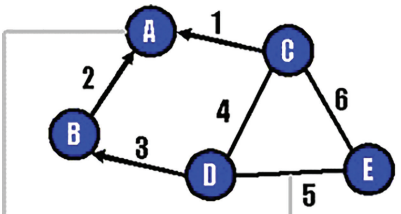


FIGURE 6.2
Representation of point, line, and polygon objects using the vector data model (Reprinted with permission from Paul Longley, Michael Goodchild, David Maguire, and David Rhind, *Geographic Information Systems and Science*, 2nd Edition (Wiley, 2005) p. 184).

links (edges or linear segments), and focuses on representing the connectivity among these elements along with the cost associated with traversing them. Figure 6.3 illustrates how connectivity is stored in a table. The first row for the Links table under a column header indicates that link 1 traverses from node C to node A in one way. Node C is linked to node E in both directions. Similarly, adjacency can be stored in a table as well (Figure 6.4). Moving in the direction of the arrow by Polyline# 6, the left-hand side polygon of the line is two and the right-hand side polygon of the line is five. Explicitly encoding qualitative spatial relationships among elements of the data model is a prerequisite to some spatial analyses including network analysis, spatial query, and overlay. Topological vector models use this type of vector data, compared to a simple vector model that does not store topology.

Finally, data structure is also dictated by the ways in which data is captured. For instance, the raster data structure of satellite imagery (e.g., Landsat TM product, DigitalGlobe's WorldView-2 product) is built into data capture methods just as an image is captured by an electronic scanner mounted on a satellite. Digitizing contours (comprised of vertices) from a topographic map placed on a digitizing table will result in vector data (a set of points with elevation values). Air-quality attributes such as ozone level can be measured using a GPS-equipped sensor at sample locations, and is thus stored as point (vector) data. In summary, geospatial data formats are either vector or raster data structures in which vector data is associated with geometry type (point, line, polygon; e.g., shapefile) and raster has a fixed spatial resolution (pixel size; e.g., GeoTIFF). In the next section, we will examine how attribute data is typically stored in GIS.



Nodes		
ID	Long	Lat
A	40.42345	-75.1245
B	40.31021	-75.2510
C	40.41882	-74.9124
D	40.25908	-75.0031
E	40.28990	-74.7893

Links			
ID	From	To	One_Way
1	C	A	Yes
2	B	A	Yes
3	D	B	Yes
4	D	C	No
5	D	E	No
6	C	E	No

FIGURE 6.3
Representation of connectivity among links and nodes for network data model.

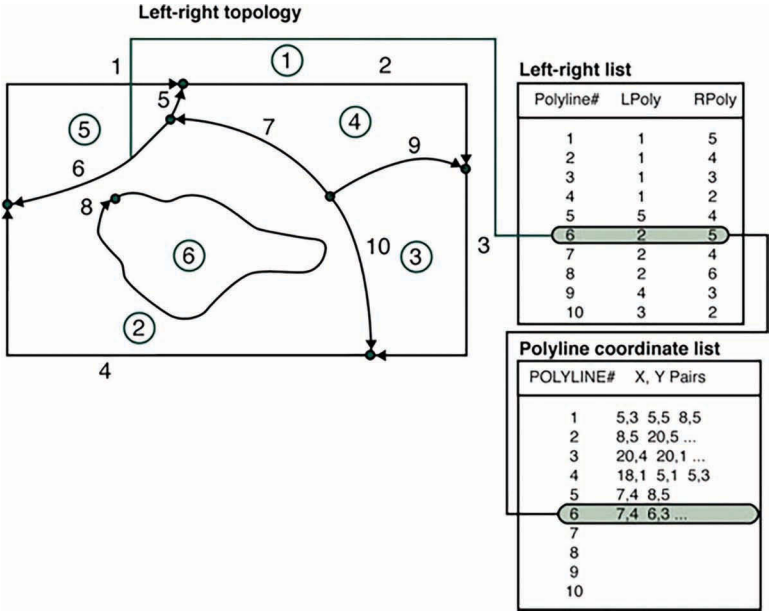


FIGURE 6.4
Representation of adjacency of a topologically structured polygon data layer. (Reprinted with permission from Paul Longley, Michael Goodchild, David Maguire, and David Rhind, *Geographic Information Systems and Science*, 2nd Edition (Wiley, 2005) p. 187).

6.3 How Attribute Data Is Structured—Relational Databases

6.3.1 Relational Databases

Probably the best-known data model for storing attribute data is the table view of the world, which became popular with the prevalence of relational databases. A relational database is based on a relational data model and organizes data into one or more tables of columns and rows, with relations defined among tables. As illustrated in Figure 6.5, a table represents an entity type or object class such as Cities or Students. In the table, rows represent instances of an object class such as Chicago and John Doe, and columns represent attributes of instances of an object class such as Name and Age as shown in the column header. Cells in the table contain attribute values of instances of an object class for a given domain or attribute, such as 1833 for the year Chicago was founded and 24 for John Doe's Age.

Tables can be related or joined based on common attribute values in relational databases. To establish the relation "lives in" in the statement like "a student lives in a city," two participating tables should have columns whose values exactly match. Figure 6.5 shows that a Cities table has a column Name

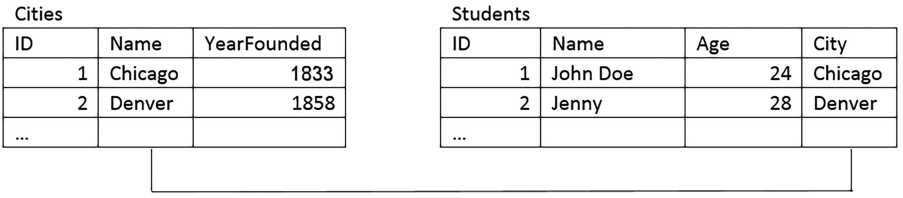


FIGURE 6.5
Relational database.

and a Students table has a column City, where values for two columns match; i.e., Cities.Name=Students.City given Table.Column notation. One student lives in one city if the relationship "lives in" is defined as the current primary residency. Therefore, it can be said that there is a one-to-one relationship between Students and Cities based on some predefined rules.

6.3.2 Database Normalization

It is important that data is consistent and structured to prevent any incorrect query or analysis results. The process of organizing data to ensure consistency and minimize redundancy is called database normalization. Normalization is especially crucial when you work with poorly structured spreadsheet files. The following checklist for database normalization follows Codd's rules for relational databases and will ensure that data are well structured, consistent, and have integrity:

- Is the first row a column header?
- Is the column about the same subject (domain)?
- Is a value atomic (single-valued)?
- Is a row unique (nonduplicating)?
- Is there a column that uniquely identifies an instance, and is the column populated?
- Is there a column that relates one table to another table (if a relationship exists), and is the column populated?
- Are other columns functionally dependent on a unique identifier?

Column header: the first row is always reserved for a column header in relational databases. Both tables (Cities and Students) in Figure 6.5 have a column header in the first row. It is good practice to use a descriptive but concise name without a special character (except for underscores) for the name of a column. There are usually limits to the number of characters that can be entered for the name of a column. Any column name longer than 10 characters will be truncated after data is loaded into a file geodatabase table; i.e., YearFounded will become YearFounde after data loading. "FirstName" or

"First_Name" is preferred to "First*Name" or "1stName" as "First*Name" contains a special character and "1stName" begins with a number.

Column domain: a value entered under a given column should pertain to the domain of a given column. For example, it is wrong to enter "Denver" under YearFounded because "Denver" is about the Name (not YearFounded) of Cities.

*Atomic value*s: you cannot have multiple values for a cell. That is, it is not good to enter '24, 25' for the age of John Doe even if he is 24.5 years old because having multiple values will result in inconsistent query results.

*A unique ro*w: no duplicates are allowed for identifying each instance. Suppose there are two rows representing Chicago with conflicting YearFounded values in the Cities table. Then which row should we go by?

*Identity integrit*y: this means that there should be a column that uniquely identifies an instance in the table, and the column should be populated with values. One or more columns that serve as a unique identifier is called a primary key (PK). For example, ID or Name can be a PK for the Cities table in Figure 6.3. If there are two students with the name Jenny in the table, then Name cannot be a PK for the Students table as it does not uniquely identify a student. PK values should not be missing because an instance cannot be accessed if the PK value is null.

*Referential integrit*y: this rule applies to at least two tables that are related (e.g., Cities and Students in Figure 6.5). The rule states that there should be a column that relates one table to another table, and the column should be populated. In Figure 6.5, the relationship is established because the values of Cities.Name match the values of Students.City. Here the column Students. City is the foreign key (FK) that relates the Students table to another table, Cities. The FK value should not be null because another table (Cities) cannot be referenced from one table (Students) if the FK value is null.

*Functional dependenc*y: non-PK columns should be functionally dependent on the PK in the table. For instance, suppose that you add another column PoliticalParty that represents the political affiliation of a city mayor in the Cities table. Then are those values (democratic, republican) a function of cities or mayors? Those values are a function of mayors, not cities. This means you should deal with those attributes (PoliticalParty) in a separate table called Mayor because political party is functionally dependent on a unique instance of mayor. Organizing data into tables that only contain relevant attributes serves to reduce redundancy in datasets.

Now we will apply the rules of database normalization to a CSV file you can download from the TRI (Toxic Release Inventory) Explorer at https://iaspub.epa.gov/triexplorer/tri_release.facility. This CSV file contains the toxic chemicals released at TRI facilities (Figure 6.6). Clearly, the file is not in the standard form. There are many problems that need to be fixed. First, rows 1, 3, 4, and 5 should be deleted while row 2 can be retained as a column header. A breakdown of the chemical compounds released from each facility is listed under the Facility column, which violates the second rule (column domain).

	A	B	C	D	E	F	G	H	I	J
1	TRI On-site and Off-site Reported Disposed of or Otherwise Released (in pounds), all facilities (of 354) for f									
2	Row #	Facility	TRIF ID	Latitude	Longitude	Total On-s	Total On- and			
3						Disposal o	Off-site			
4						Other Rele	Disposal or			
5							Other Releases			
6	1	/DELTA CC	60650LYM	41.817	-87.749	0	0			
7		ETHYLENE				0	0			
8		ZINC COM				0	0			
9	2	A FINKL &	60619VRSI	41.729	-87.592	509	335,874.00			
10		CHROMIU				121	48,159.00			
11		LEAD				5	7,222.00			
12		MANGANI				171	169,681.00			
13		MOLYBDE				33	2,649.00			
14		NICKEL				107	6,241.00			
15		ZINC (FUM				72	101,922.00			
16	3	AALLIED D	60131LLDI	41.934	-87.891	2,175.00	2,175.00			

FIGURE 6.6
Before database normalization.

You can delete the chemical compounds (in rows 7, 8, 10–15) if you are not interested in specific chemical compounds. Otherwise, you can transpose the amount of chemical compounds into new columns added to the last column as they are functionally dependent on a facility.

Figure 6.7 shows the data after normalization. The first row is a column header with a concise and descriptive name without special characters. The column is about the same subject, but each row is unique. TRIF_ID can be set to a PK as it uniquely and unambiguously identifies each facility. Two columns, OnSiteTx and TotalToxic, representing the amount of onsite and total toxic chemicals, have atomic values. Latitude, Longitude, OnSiteTx, and TotalToxic are functionally dependent on a PK.

Now suppose that you have data for lung cancer rate by zip code, and would like to examine how toxic chemicals relate to lung cancer. To link the TRI table to a cancer table, it is necessary to add and populate a new column named Zipcode in the TRI table (Figure 6.8). This column (TRI.Zipcode) is used as a FK that references the cancer table from the TRI table. Since there

Table

TRI.csv

Row	Facility	TRIF_ID	Latitude	Longitude	OnSiteTx	TotalToxic
1	/DELTA COS GROUP DBA OLYMPIC OIL.5000 W 41ST ST, CICERO ILLINOIS 60804 (C	60650LYMPC5000	41.817	-87.749	0	0
2	A FINKL & SONS CO.1355 E 93RD ST, CHICAGO ILLINOIS 60619 (COOK)	60619VRSNC1355	41.729	-87.592	509	335874
3	AALLIED DIE CASTING CO OF IL.3021 CULLERTON DR, FRANKLIN PARK ILLINOIS 60	60131LLDDC3021	41.934	-87.891	2175	2175
4	ABLE DIE CASTING CORP.3907 WESLEY TERRACE, SCHILLER PARK ILLINOIS 6017	6017WBLDCS397	41.95	-87.863	153	153
5	ABLE ELECTROPOLISHING CO INC.2001 S KILBOURN AVE, CHICAGO ILLINOIS 6062	60623BLLCT2001S	41.854	-87.736	594.4	13066.4
6	ACCURATE ANODIZING.3130 S A US TIN AVE, CICERO ILLINOIS 60804 (COOK)	60650CCRTN3130	41.835	-87.774	45	45
7	ACCURATE DISPERSIONS.1111 MAPLE AVE, HOMEWOOD ILLINOIS 60430 (COOK)	60430CCRTD1111	41.567	-87.645	24	984.2
8	ACCURATE DISPERSIONS.192 W 155TH ST, SOUTH HOLLAND ILLINOIS 60473 (COO	60473MCWHR192	41.613	-87.622	5178	11400.25
9	ACCURATE THREADED FASTNERS.3550 W PRATT, LINCOLNWOOD ILLINOIS 60712	60645CCRTT3550	42.005	-87.718	0	0
10	ACE ANODIZING & IMPREGNATING INC.4161 BUTTERFIELD RD, HILLSIDE ILLINOIS 6	60162CNDZN4161	41.881	-87.885	2208	2316.01
11	ACID PRODUCTS CO INC.600 W 41ST ST, CHICAGO ILLINOIS 60609 (COOK)	60609CDPRD6W4	41.821	-87.642	0	0
12	ACME DIE CASTING LLC.3610 COMMERCIAL AVE, NORTHBROOK ILLINOIS 60062 (C	60062CMDCS3610	42.147	-87.869	241.2	242.55
13	ADHESIVE COATING TECHNOLOGIES INC.420 NORTHGATE PKWY, WHEELING ILLIN	60090CSTMT420N	42.147	-87.929	1629	1629

FIGURE 6.7
After database normalization.

TRI_with_zip

Row	Facility	TRIF_ID	Latitude	Longitude	OnSiteTx	TotalToxic	Zipcode
1	/DELTA COS GROUP	60650LYMPC5000	41.817	-87.749	0	0	60804
2	A FINKL & SONS CO.1	60619VRSNC1355	41.729	-87.592	509	335874	60619
3	AALLIED DIE CASTING	60131LLDDC3021	41.934	-87.891	2175	2175	60131
4	ABLE DIE CASTING C	6017WBLDCS397	41.95	-87.863	153	153	60176
5	ABLE ELECTROPOLIS	60623BLLCT2001S	41.854	-87.736	594.4	13066.4	60623
6	ACCURATE ANODIZIN	60650CCRTN3130	41.835	-87.774	45	45	60804
7	ACCURATE DISPERSI	60430CCRTD1111	41.567	-87.645	24	984.2	60430
8	ACCURATE DISPERSI	60473MCWHR192	41.613	-87.622	5178	11400.25	60473
9	ACCURATE THREADE	60645CCRTT3550	42.005	-87.718	0	0	60712
10	ACE ANODIZING & IMP	60162CNDZN4161	41.881	-87.885	2208	2316.01	60162
11	ACID PRODUCTS CO I	60609CDPRD6W4	41.821	-87.642	0	0	60609

zipcode	lngCncrRt
60001	0.5
60002	1
60003	1.5

FIGURE 6.8
Referential integrity.

are many TRI facilities in a zip code, the relationship between the TRI table and the cancer table is many to one.

6.3.3 Field Data Types

When adding a new field (column) to a table in GIS software, you will be prompted to choose a data type for the field. There are three main data types: text, date, and number. Text (or string) stores text data such as the name of a city (e.g., Chicago), and date stores date and time (e.g., October 21, 2018, 8:32 PM). Numeric data can be stored in one of four data types: short integer, long integer, float, or double. The name of these four numeric data types may vary depending on the computer system being used. Short integer and long integer data types store numerical values without fractional values. Float and double data types store numerical values with fractional values. More bits (units of computer storage) are assigned to a long integer (32 bits) than a short integer (16 bits). A short integer uses 15 bits to store a range of numbers and uses 1 bit to store sign (+,–). With 15 bits, you can store up to 2^{15} (2 to the 15th power = 32,768). A short integer can store an integer between –32,768 and 32,767. A long integer can store an integer between –2,147,483,648 to 2,147,483,647. Float has seven decimal digits of precision with maximum, and double has 15 decimal digits of precision. You can use double to store sufficiently large or small numerical values, such as annual government spending in dollars or rate of rare diseases in fractions.

What data type you should use depends on the type of data, the presence of fractional values, and the possible range of numerical values. For example, a short integer is inadequate to store populations of large cities since the

largest possible value with a short integer is 32,767. In contrast, using double to represent only three possible values (1, 2, 3) is a waste of storage space. Another consideration when choosing data type are operations. It makes sense to use a numerical data type if you want to do algebraic operations such as calculating population changes. In US census data, the FIPS (Federal Information Processing Standard) code is used to uniquely identify administrative units such as county and tract. It is convenient to store FIPS codes as text (string) data to utilize string manipulation functions for parsing or concatenation. For example, the FIPS code for the census tract 102.01 in Cook County, Illinois is 17031010201, where the first two characters (17) represents a state, the next three characters (031) represents a county, and the next six characters (010201) represent a tract. A string function Left([fips],2) can be used to extract the first two characters (state) from the field [fips] in a field calculator tool.

Since attribute data is stored as a table following a relational data model in GIS, you can use SQL queries. SQL, pronounced as "sequel," is a standard (or structured) query language for relational databases that allows you to interact with the data. With the SQL SELECT statement, you can select features or records that meet a certain condition. For instance, the SQL SELECT statement SELECT * FROM tl_2013_17_tract WHERE "COUNTYFP"='031' selects records for all columns from the table tl_2013_17_tract where a field COUNTYFP is equal to '031'. The general form for the query expression within WHERE is <Field_name> <Operator> <Value or String>. If a field data type is a string (text), the value should be single quoted. Otherwise, the value does not have to be quoted. You can combine expressions with the AND and OR operators. For example, "AREA" > 1,500 AND "GARAGE" > 3 selects houses that have more than 1,500 square feet and a garage for three or more cars.

6.3.4 Table Join

As illustrated above, it is often necessary to join or relate tables in GIS. Table join is a fairly common task. For example, you may want to map unemployment rate by county from a CSV file or tab-delimited file containing unemployment rates. To make this map, an attribute table of unemployment rate by county should be joined to spatial data of county boundaries. Table join appends attributes of a source table (unemployment rate) to the end of a target table (county boundary). Answering the following questions will help you use table join properly:

- Do the two tables share the same attribute values?
- Are those common attributes of the same data types?
- What is the relationship between the two tables?
- In what direction should you perform table join?

Each question is answered using the TRI table and Cancer table example shown in Figure 6.9. The TRI table contains the ID, the amount of toxic chemical, and the zipcode of the TRI facility, and the Cancer table contains the cancer rate by zipcode.

*Presence of common attribut*e: two tables should share the attribute values on which a table join will be based. The values of the last column zipcode in the TRI table are the same as those of the first column zipcode in the Cancer table. Hence, two tables can be joined based on the condition TRI. zipcode=Cancer.zipcode.

Data type of a common key: if the data types of two tables to be joined do not belong to the same category (number, text, date), table join cannot be used. That is, if TRI.zipcode is the text data type, then Cancer.zipcode should also be the text data type for table join to work. If TRI.zipcode is a numerical data type (short, long, float, double), then Cancer.zipcode should also be the numerical data type.

Relationship between tables: relationship refers to how many rows in one table are associated with rows in another table. One zipcode contains many TRI facilities. Therefore, there is one-to-many relationship between the Cancer by zipcode and the TRI table.

Direction of table join: Table join can be used in two directions. That is, you can join the TRI table to the Cancer table, or vice versa. Since a source table is appended to a target table, it is always a good idea to join a one-side table to a

TRI

TRI_ID	Toxic	zipcode
1	25354	60001
2	3422	60001
3	253	60002
4	345	60002

Cancer

zipcode	cancerrt
60001	3.4
60002	1.4

TRI+Cancer if you join Cancer (one-side) to TRI (many-side)

TRI_ID	Toxic	zipcode	cancerrt
1	25354	60001	3.4
2	3422	60001	3.4
3	253	60002	1.4
4	345	60002	1.4

Cancer+TRI if you join TRI (many-side) to Cancer (one-side)

zipcode	cancerrt	TRI_ID	Toxic
60001	3.4	1? 2?	25354? 3422?
60002	1.4	3? 4?	253? 345?

FIGURE 6.9
Table join.

many-side table to avoid losing any information. If you join Cancer (one-side) to TRI (many side), then all the data from the source table (Cancer) will be preserved after the table join as illustrated in the second panel of Figure 6.7. If you join TRI (many-side) to Cancer (one-side), some data from the source table will be lost. That is, either 1 or 2 (not all) for the first row with zipcode 60001 will be saved as shown in the third panel of Figure 6.9.

6.4 How Spatial and Attribute Data are Put Together

So far we have reviewed spatial and attribute data structures separately. As mentioned earlier, GIS store both spatial and attribute data, but the question is how are these two data types combined? To answer this question, consider how you would store both spatial and attribute data in a table to explore the demographic characteristics of some communities (Figure 6.10). Storing attributes in a table is relatively straightforward. A community named Lincoln Park has a population of 1,000 with 500 households as shown in Figure 6.8. That's easy enough. However, storing spatial aspects in a table is not that straightforward. You could enter a series of coordinates into a new column named Boundaries (see the last column in Figure 6.10), but the value for the last column would be multivalued unlike other attribute values. This violates Codd's rule of database normalization, which states that data values should be single-valued. In addition, it is hard to tell how many coordinate values will be entered in this column. This is not desirable as the storage unit for a column is limited. In other words, encoded spatial data does not neatly fit into a table representation.

The IT industry has long been dominated by relational databases. In fact, the market share of database management systems (DBMS), which are based on relational data models, is 86%. Thus the GIS industry has had to learn to work with the data dissemination practices of the database industry. So how does the GIS industry work with the issue of spatial data not being neatly encoded into the relational model? One way is to store spatial data and attribute tables separately and to link spatial data to attribute tables through common keys (attributes), otherwise known as a georelational model. In this model a widely used data format called a shapefile separates spatial data (stored in.shp and. shx files) from attribute data (.dbf files). Another option is to use predefined spatial data types or geometry types (such as Point, LineString, Polygon) that

ID	Community	Population	Households	White	Black	Asian	PopAgeUnder15	PopAgeOver65	Boundaries
1	Lincoln Park	1000	500	400	50	50	200	200	(2,2), (3,3), (4,4)...
2	Rogers Park								
3	West Town								
	...								

FIGURE 6.10
Representing spatial data in a table.

can be recognized by different software systems. More recently invented geospatial data formats such as KML and GeoJSON rely on Open Geospatial Consortium (OGC) spatial data types. By building your geodatabase (native data format in ArcGIS) on a relational model, you can organize all types of geospatial data (vector, raster, and table) in one place, and utilize advanced capabilities to ensure data integrity and enable high-level data modeling.

In summary, a table cannot adequately represent spatial data because spatial data is diverse in nature, scale-dependent, and multidimensional. To address this complexity, new spatial data models such as geodatabases incorporate the concept of object-orientation that represents constructs as objects while retaining the advantages that relational databases offer. An object-based model views the world as a hierarchy of object classes and allows users to define data by its behavior ('do') in addition to its status ('be'). In addition to dealing with the complexity of geospatial data, the geospatial industry is also faced with the needs for simple geospatial data that works across different platforms and software systems. Today open standards are increasingly adopted, and spatial data models have evolved to meet the needs of the geospatial industry.

ACTIVITY 6.1 GET TO KNOW GEOSPATIAL DATA (ARCGIS 10.5)

- Get the data for this lab activity. The Lab06 folder contains the following public-domain geospatial data:

Name	Data Format	Data Downloaded and Processed From
NED	ESRI GRID	National Map Viewer
CensusBlockGroup	Shapefile	US Census Bureau
NLCD_clipped	ERDAS IMAGINE	National Map Viewer
Streets	Shapefiles	Census TIGER/Line

1. Explore Properties of Geospatial Data in ArcCatalog

- Open ArcCatalog in ArcGIS. ArcCatalog allows you to explore, manage, and organize geospatial data.
- Connect to the working folder (Figures 6.11 and 6.12).

  The folder connection will be mapped in the Catalog Tree on the left (Figure 6.13). Items in the selected folder connection are shown in the Catalog Display on the right. There are four datasets (metadata in XML excluded). Among them, two data layers are shapefiles and two are raster datasets as shown under the Type panel in the Catalog Display.

FIGURE 6.11
ArcCatalog—Connect to folder 1; from Copyright © 2018 Esri, ArcGIS, ArcMap, and the GIS User Community. All rights reserved. With Permission.