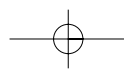


PART
V

DATABASES AND THE
INTERNET

DATABASE CONNECTIVITY AND WEB TECHNOLOGIES

14



CASIO UPGRADES CUSTOMER WEB EXPERIENCE

A global leader, Casio Computer Co., Ltd., has been developing consumer electronics since 1957. While the company creates high tech devices such as LCD TVs, digital cameras, handheld computers, and other communications devices, its Web site was behind the times. The site was written in HTML only, with little content and with an e-commerce page that was managed by a third party.

“It was an OK site, but not the kind of breakthrough experience we were hoping to offer our customers,” says Casio’s Internet services manager Michael McCormick.

The company wanted to upgrade the look and feel and add new functionality, including better merchandising capabilities and a more comprehensive shopping cart. The company chose Macromedia’s ColdFusion MX running on a Linux platform. (Macromedia recently merged with Adobe and the product is now Adobe ColdFusion.) A 15-person team including designers, programmers, and testers spent five months creating the new site. ColdFusion’s tag-based language, its reusability of code modules, and its debugging tools expedited this development. ColdFusion’s open architecture Web application server also facilitated integration with the company’s enterprise system. Now, users can browse through thousands of products, make purchases easily, and track their orders.

Casio also enjoys much greater administrative functionality, accessing inventory figures, sales reports, membership information, and order process and fulfillment data through the site. In addition, the company now can manage the site’s content itself, rather than turning to its Web development partner, Pipeline Interactive, each time it needs to update one of its 50,000 screens of content.

Yet, the most critical feature of the new site is its ability to cross-sell and upsell. “We can cross-sell a printer when someone buys a digital camera,” explains McCormick. “Or we can suggest additional ink cartridges when someone buys a printer. If a particular SKU isn’t in stock, [we] suggest substitute products that are similar—perhaps a different color.”

The result is that Casio’s e-commerce sales have doubled since the site was launched. The site boasts more than 700,000 registered users with more than one million page views per day.

Business
Vignette

14

DATABASE CONNECTIVITY AND WEB TECHNOLOGIES

FOURTEEN

In this chapter, you will learn:

- About the various database connectivity technologies
- How Web-to-database middleware is used to integrate databases with the Internet
- About Web browser plug-ins and extensions
- What services are provided by Web application servers
- What Extensible Markup Language (XML) is and why it is important for Web database development

As you know, a database is a central repository for critical business data. Such data can be generated through traditional business applications or via newer business channels such as the Web, a phone connection, a wireless PDA, or a smart phone. To be useful universally, the data must be available to all business users. Those users need access to the data via many avenues: a spreadsheet, a user-developed Visual Basic application, a Web front end, Microsoft Access forms and reports, and so on. In this chapter, you learn about the architectures used by applications to connect to databases.

The Internet has changed how organizations of all types operate. For example, buying goods and services via the Internet has become commonplace. In today's environment, interconnectivity occurs not only between an application and the database, but also between applications interchanging messages and data. Extensible Markup Language (XML) provides a standard way of exchanging unstructured and structured data between applications.

Given the growing relationship between the Web and databases, database professionals must know how to create, use, and manage Web interfaces to those databases. This chapter examines the basics of Web database technologies.



Preview

14.1 DATABASE CONNECTIVITY

The term *database connectivity* refers to the mechanisms through which application programs connect and communicate with data repositories. Database connectivity software is also known as **database middleware** because it interfaces between the application program and the database. The data repository, also known as the *data source*, represents the data management application (that is, an Oracle RDBMS, SQL Server DBMS, or IBM DBMS) that will be used to store the data generated by the application program. Ideally, a data source or data repository could be located anywhere and hold any type of data. For example, the data source could be a relational database, a hierarchical database, a spreadsheet, or a text data file.

The need for standard database connectivity interfaces cannot be overstated. Just as SQL has become the de facto data manipulation language, there is a need for a standard database connectivity interface that will enable applications to connect to data repositories. There are many different ways to achieve database connectivity. This section will cover only the following interfaces:

- Native SQL connectivity (vendor provided).
- Microsoft's Open Database Connectivity (ODBC).
- Data Access Objects (DAO) and Remote Data Objects (RDO).
- Microsoft's Object Linking and Embedding for Database (OLE-DB).
- Microsoft's ActiveX Data Objects (ADO.NET).
- Sun's Java Database Connectivity (JDBC).

You should not be surprised to learn that most interfaces you are likely to encounter are Microsoft offerings. After all, client applications connect to databases, and the majority of those applications run on computers that are powered by some version of Microsoft Windows. The data connectivity interfaces illustrated here are dominant players in the market, and more importantly, they enjoy the support of the majority of database vendors. In fact, ODBC, OLE-DB, and ADO.NET form the backbone of Microsoft's **Universal Data Access (UDA)** architecture, a collection of technologies used to access any type of data source and manage the data through a common interface. As you will see, Microsoft's database connectivity interfaces have evolved over time: each interface builds on top of the other, thus providing enhanced functionality, features, flexibility, and support.

14.1.1 NATIVE SQL CONNECTIVITY

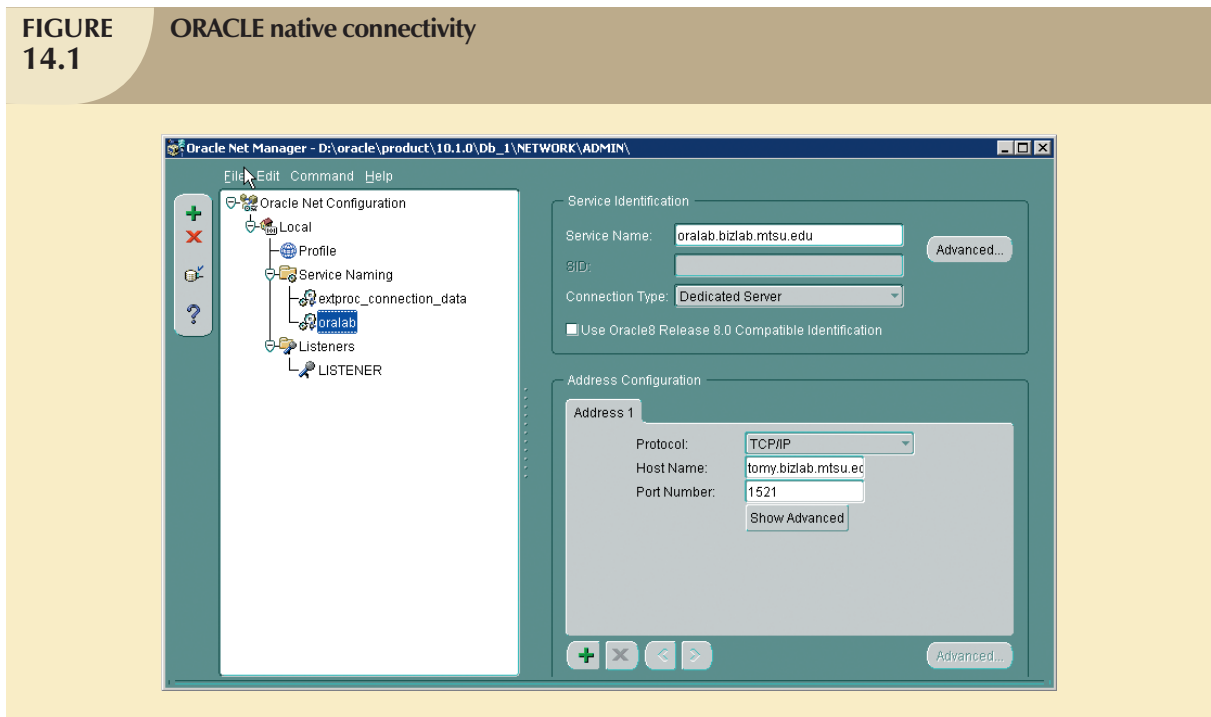
Most DBMS vendors provide their own methods for connecting to their databases. Native SQL connectivity refers to the connection interface that is provided by the database vendor and that is unique to that vendor. The best example of that type of native interface is the Oracle RDBMS. To connect a client application to an Oracle database, you must install and configure the Oracle's SQL*Net interface in the client computer. Figure 14.1 shows the configuration of Oracle SQL*Net interface on the client computer.

Native database connectivity interfaces are optimized for "their" DBMS, and those interfaces support access to most, if not all, of the database features. However, maintaining multiple native interfaces for different databases can become a burden for the programmer. Therefore, the need for "universal" database connectivity arises. Usually, the native database connectivity interface provided by the vendor is not the only way to connect to a database; most current DBMS products support other database connectivity standards, the most common being ODBC.

14.1.2 ODBC, DAO, AND RDO

Developed in early 1990s, **Open Database Connectivity (ODBC)** is Microsoft's implementation of a superset of the SQL Access Group **Call Level Interface (CLI)** standard for database access. ODBC is probably the most widely supported database connectivity interface. ODBC allows any Windows application to access relational data sources, using SQL via a standard **application programming interface (API)**. The Webopedia online dictionary

FIGURE 14.1 ORACLE native connectivity



(www.webopedia.com) defines an API as “a set of routines, protocols, and tools for building software applications.” A good API makes it easy to develop a program by providing all of the building blocks; the programmer puts the blocks together. Most operating environments, such as Microsoft Windows, provide an API so programmers can write applications consistent with the operating environment. Although APIs are designed for programmers, they are ultimately good for users because they guarantee that all programs using a common API will have similar interfaces. That makes it easy for users to learn new programs.

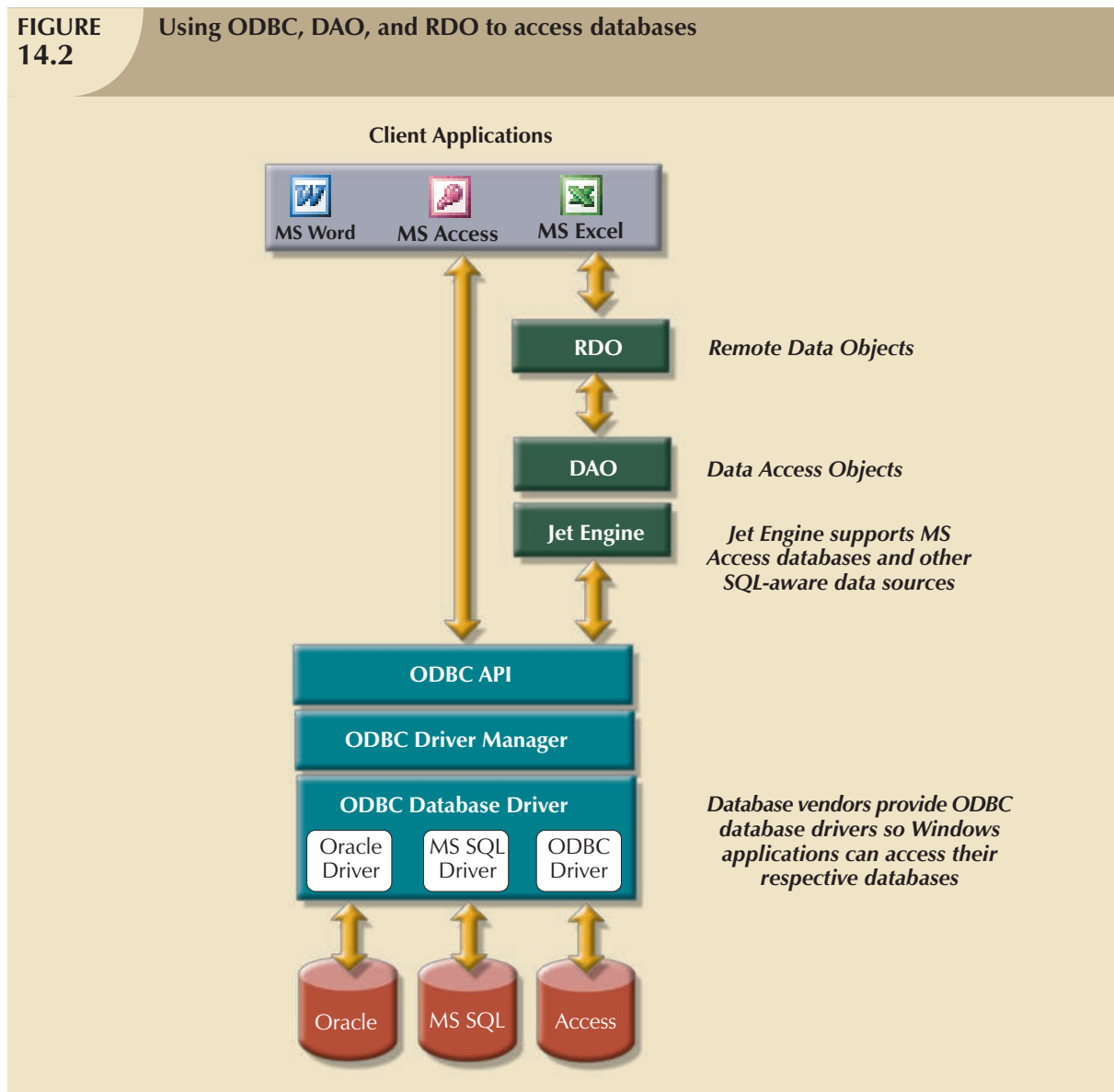
ODBC was the first widely adopted database middleware standard, and it enjoyed rapid adoption in Windows applications. As programming languages evolved, ODBC did not provide significant functionality beyond the ability to execute SQL to manipulate relational style data. Therefore, programmers needed a better way to access data. To answer that need, Microsoft developed two other data access interfaces:

- **Data Access Objects (DAO)** is an object-oriented API used to access MS Access, MS FoxPro, and dBase databases (using the Jet data engine) from Visual Basic programs. DAO provided an optimized interface that exposed to programmers the functionality of the Jet data engine (on which the MS Access database is based). The DAO interface can also be used to access other relational style data sources.
- **Remote Data Objects (RDO)** is a higher-level object-oriented application interface used to access remote database servers. RDO uses the lower-level DAO and ODBC for direct access to databases. RDO was optimized to deal with server-based databases, such as MS SQL Server, Oracle, and DB2.

Figure 14.2 illustrates how Windows applications can use ODBC, DAO, and RDO to access local and remote relational data sources.

As you can tell by examining Figure 14.2, client applications can use ODBC to access relational data sources. However, the DAO and RDO object interfaces provide more functionality. DAO and RDO make use of the underlying ODBC data services. ODBC, DAO, and RDO are implemented as shared code that is dynamically linked to the Windows operating environment through **dynamic-link libraries (DLLs)** which are stored as files with the .dll extension. Running as a DLL, the code speeds up load and run times.

FIGURE 14.2 Using ODBC, DAO, and RDO to access databases



The basic ODBC architecture has three main components:

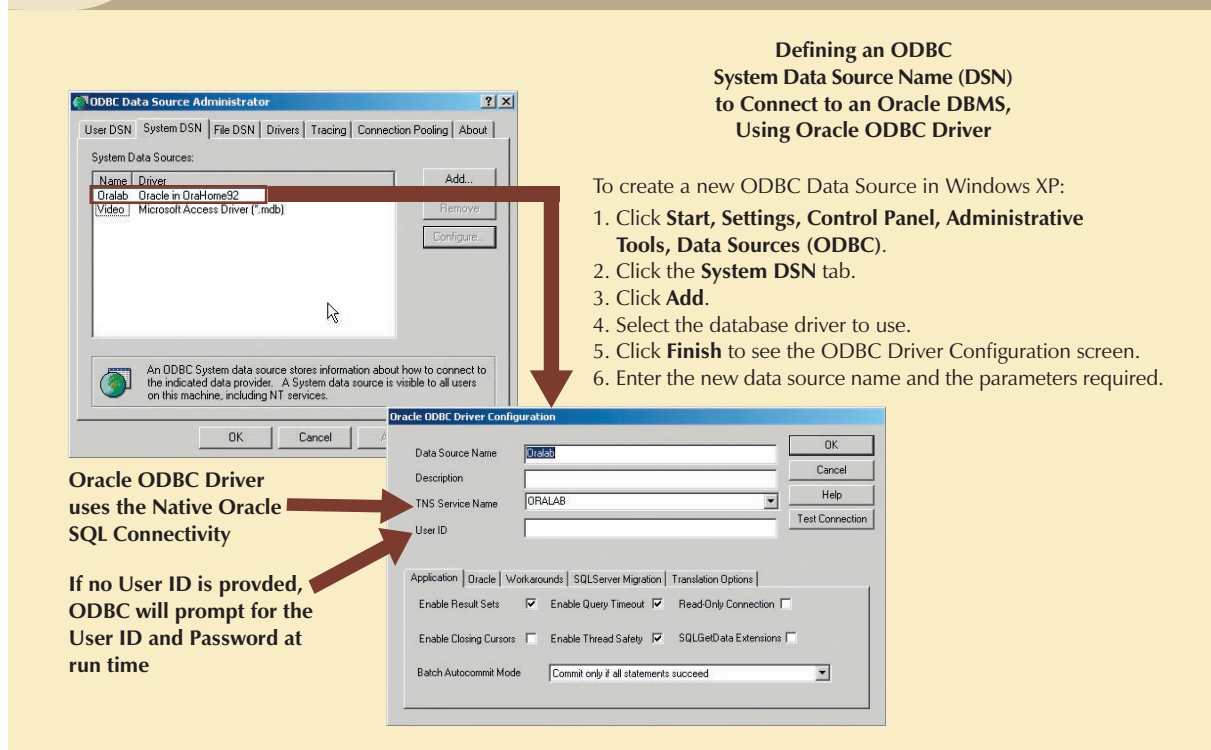
- A high-level *ODBC API* through which application programs access ODBC functionality.
- A *driver manager* that is in charge of managing all database connections.
- An *ODBC driver* that communicates directly to the DBMS.

Defining a data source is the first step in using ODBC. To define a data source, you must create a **data source name (DSN)** for the data source. To create a DSN you need to provide:

- An *ODBC driver*. You must identify the driver to use to connect to the data source. The ODBC driver is normally provided by the database vendor, although Microsoft provides several drivers that connect to most common databases. For example, if you are using an Oracle DBMS, you will select the Oracle ODBC driver provided by Oracle, or if desired, the Microsoft-provided ODBC driver for Oracle.

- A *DSN name*. This is a unique name by which the data source will be known to ODBC, and therefore, to applications. ODBC offers two types of data sources: user and system. *User data sources* are available only to the user. *System data sources* are available to all users, including operating system services.
- *ODBC driver parameters*. Most ODBC drivers require specific parameters in order to establish a connection to the database. For example, if you are using an MS Access database, you must point to the location of the MS Access (.mdb) file, and if necessary, provide a username and password. If you are using a DBMS server, you must provide the server name, the database name, the username, and the password needed to connect to the database. Figure 14.3 shows the ODBC screens required to create a System ODBC data source for an Oracle DBMS. Note that some ODBC drivers use the native driver provided by the DBMS vendor.

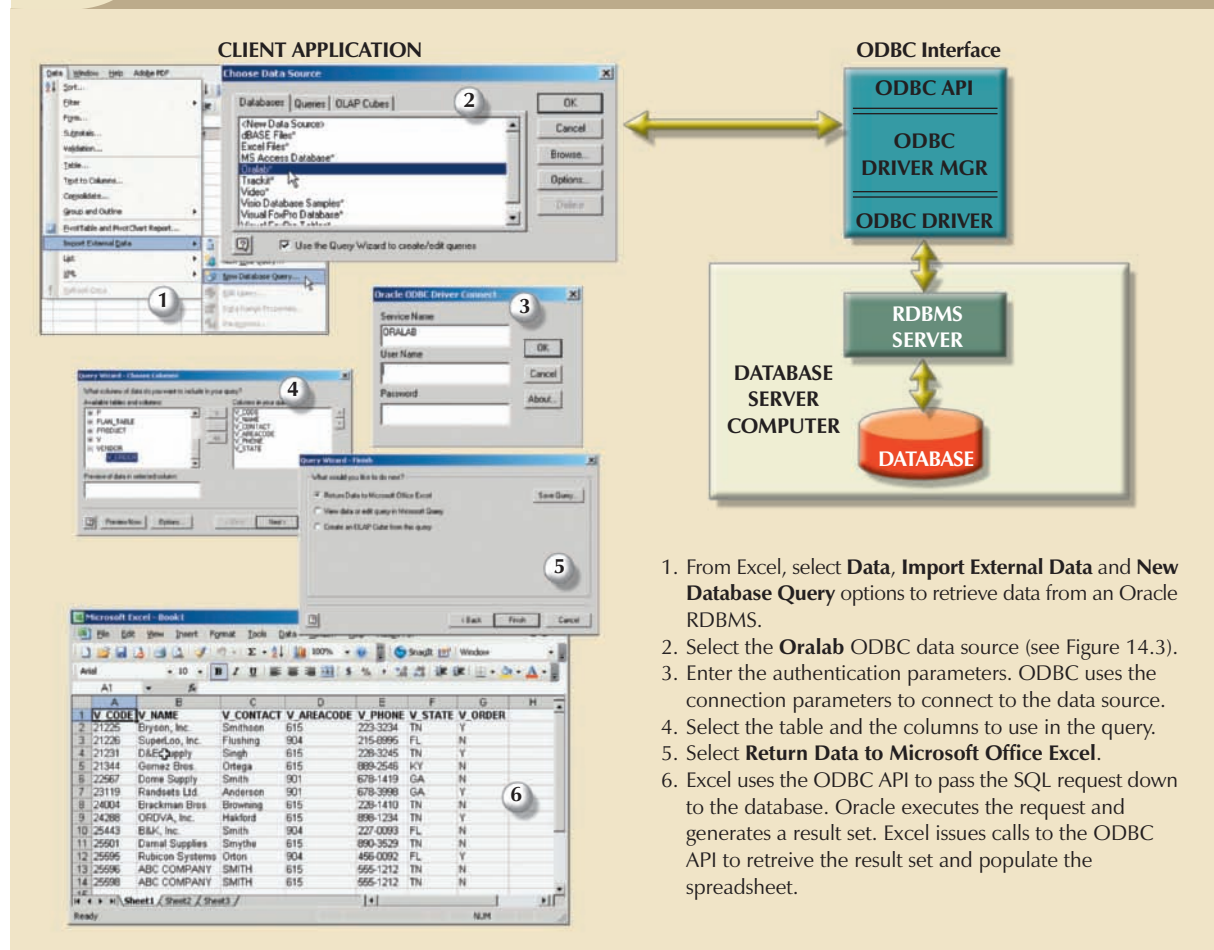
FIGURE 14.3 Configuring an Oracle ODBC data source



Once the ODBC data source is defined, application programmers can write to the ODBC API by issuing specific commands and providing the required parameters. The ODBC Driver Manager will properly route the calls to the appropriate data source. The ODBC API standard defines three levels of compliance: Core, Level-1, and Level-2, which provide increasing levels of functionality. For example, Level-1 might provide support for most SQL DDL and DML statements, including subqueries and aggregate functions, but no support for procedural SQL or cursors. The database vendors can choose which level to support. However, to interact with ODBC, the database vendor must implement all of the features indicated in that ODBC API support level.

Figure 14.4 shows how you could use MS Excel to retrieve data from an Oracle RDBMS, using ODBC. Because much of the functionality provided by these interfaces is oriented to accessing relational data sources, the use of the interfaces was limited when they were used with other data source types. With the advent of object-oriented programming languages, it has become more important to provide access to other nonrelational data sources.

FIGURE 14.4 MS EXCEL uses ODBC to connect to an Oracle database



1. From Excel, select **Data**, **Import External Data** and **New Database Query** options to retrieve data from an Oracle RDBMS.
2. Select the **Oracle** ODBC data source (see Figure 14.3).
3. Enter the authentication parameters. ODBC uses the connection parameters to connect to the data source.
4. Select the table and the columns to use in the query.
5. Select **Return Data to Microsoft Office Excel**.
6. Excel uses the ODBC API to pass the SQL request down to the database. Oracle executes the request and generates a result set. Excel issues calls to the ODBC API to retrieve the result set and populate the spreadsheet.

14.1.3 OLE-DB

Although ODBC, DAO, and RDO were widely used, they did not provide support for nonrelational data. To answer that need and to simplify data connectivity, Microsoft developed **Object Linking and Embedding for Database (OLE-DB)**. Based on Microsoft's Component Object Model (COM), OLE-DB is database middleware that adds object-oriented functionality for access to relational and nonrelational data. OLE-DB was the first part of Microsoft's strategy to provide a unified object-oriented framework for the development of next-generation applications.

OLE-DB is composed of a series of COM objects that provide low-level database connectivity for applications. Because OLE-DB is based on COM, the objects contain data and methods, also known as the interface. The OLE-DB model is better understood when you divide its functionality into two types of objects:

- *Consumers* are objects (applications or processes) that request and use data. The data consumers request data by invoking the methods exposed by the data provider objects (public interface) and passing the required parameters.
- *Providers* are objects that manage the connection with a data source and provide data to the consumers. Providers are divided into two categories: data providers and service providers.
 - *Data providers* provide data to other processes. Database vendors create data provider objects that expose the functionality of the underlying data source (relational, object-oriented, text, and so on).
 - *Service providers* provide additional functionality to consumers. The service provider is located between the data provider and the consumer. The service provider requests data from the data provider, transforms the data, and then provides the transformed data to the data consumer. In other words, the service provider acts like a data consumer of the data provider and as a data provider for the data consumer (end-user application). For example, a service provider could offer cursor management services, transaction management services, query processing services, and indexing services.

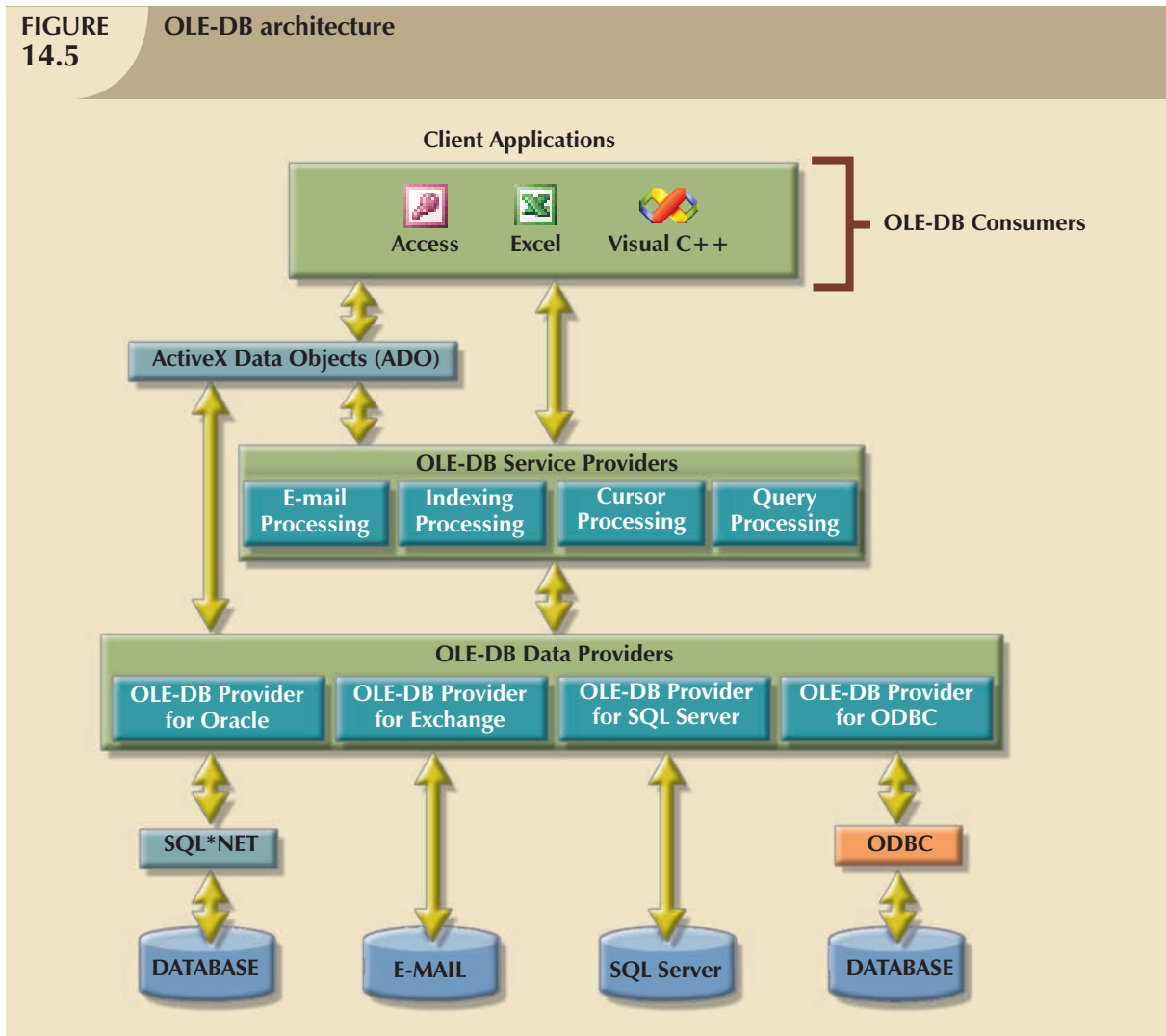
As a common practice, many vendors provide OLE-DB objects to augment their ODBC support, effectively creating a shared object layer on top of their existing database connectivity (ODBC or native) through which applications can interact. The OLE-DB objects expose functionality about the database; for example, there are objects that deal with relational data, hierarchical data, and flat-file text data. Additionally, the objects implement specific tasks, such as establishing a connection, executing a query, invoking a stored procedure, defining a transaction, or invoking an OLAP function. By using OLE-DB objects, the database vendor can choose what functionality to implement in a modular way, instead of being forced to include all of the functionality all of the time. Table 14.1 shows a sample of the object-oriented classes used by OLE-DB and some of the methods (interfaces) exposed by the objects.

TABLE 14.1 Sample OLE-DB Classes and Interfaces

OBJECT CLASS	USAGE	SAMPLE INTERFACES
Session	Used to create an OLE-DB session between a data consumer application and a data provider.	IGetDataSource ISessionProperties
Command	Used to process commands to manipulate a data provider's data. Generally, the command object will create RowSet objects to hold the data returned by a data provider.	ICommandPrepare ICommandProperties
RowSet	Used to hold the result set returned by a relational style database or a database that supports SQL. Represents a collection of rows in a tabular format.	IRowsetInfo IRowsetFind IRowsetScroll

OLE-DB provided additional capabilities for the applications accessing the data. However, it did not provide support for scripting languages, especially the ones used for Web development, such as Active Server Pages (ASP) and ActiveX. (A **script** is written in a programming language that is not compiled, but is interpreted and executed at run time.) To provide that support, Microsoft developed a new object framework called **ActiveX Data Objects (ADO)**, which provides a high-level application-oriented interface to interact with OLE-DB, DAO, and RDO. ADO provides a unified interface to access data from any programming language that uses the underlying OLE-DB objects. Figure 14.5 illustrates the ADO/OLE-DB architecture, showing how it interacts with ODBC and native connectivity options.

FIGURE 14.5 OLE-DB architecture



ADO introduced a simpler object model that was composed of only a few interacting objects to provide the data manipulation services required by the applications. Sample objects in ADO are shown in Table 14.2.

TABLE 14.2 Sample ADO Objects

OBJECT CLASS	USAGE
Connection	Used to set up and establish a connection with a data type. ADO will connect to any OLE-DB data source. The data source can be of any type.
Command	Used to execute commands against a specific connection (data source).
Recordset	Contains the data generated by the execution of a command. It will also contain any new data to be written to the data source. The Recordset can be disconnected from the data source.
Fields	Contains a collection of Field descriptions for each column in the Recordset.

Although the ADO model is a tremendous improvement over the OLE-DB model, Microsoft is actively encouraging programmers to use its new data access framework, ADO.NET.

14.1.4 ADO.NET

Based on ADO, **ADO.NET** is the data access component of Microsoft's .NET application development framework. The **Microsoft .NET framework** is a component-based platform for developing distributed, heterogeneous, interoperable applications aimed at manipulating any type of data over any network under any operating system and any programming language. Comprehensive coverage of the .NET framework is beyond the scope of this book. Therefore, this section will only introduce the basic data access component of the .NET architecture, ADO.NET.

It's important to understand that the .NET framework extends and enhances the functionality provided by the ADO/OLE-DB duo. ADO.NET introduced two new features critical for the development of distributed applications: DataSets and XML support.

To understand the importance of this new model, you should know that a **DataSet** is a disconnected memory-resident representation of the database. That is, the DataSet contains tables, columns, rows, relationships, and constraints. Once the data are read from a data provider, the data are placed on a memory-resident DataSet, and the DataSet is then disconnected from the data provider. The data consumer application interacts with the data in the DataSet object to make changes (inserts, updates, and deletes) in the DataSet. Once the processing is done, the DataSet data are synchronized with the data source and the changes are made permanent.

The DataSet is internally stored in XML format (you will learn about XML later in this chapter), and the data in the DataSet can be made persistent as XML documents. This is critical in today's distributed environments. In short, you can think of the DataSet as an XML-based, in-memory database that represents the persistent data stored in the data source. Figure 14.6 illustrates the main components of the ADO.NET object model.

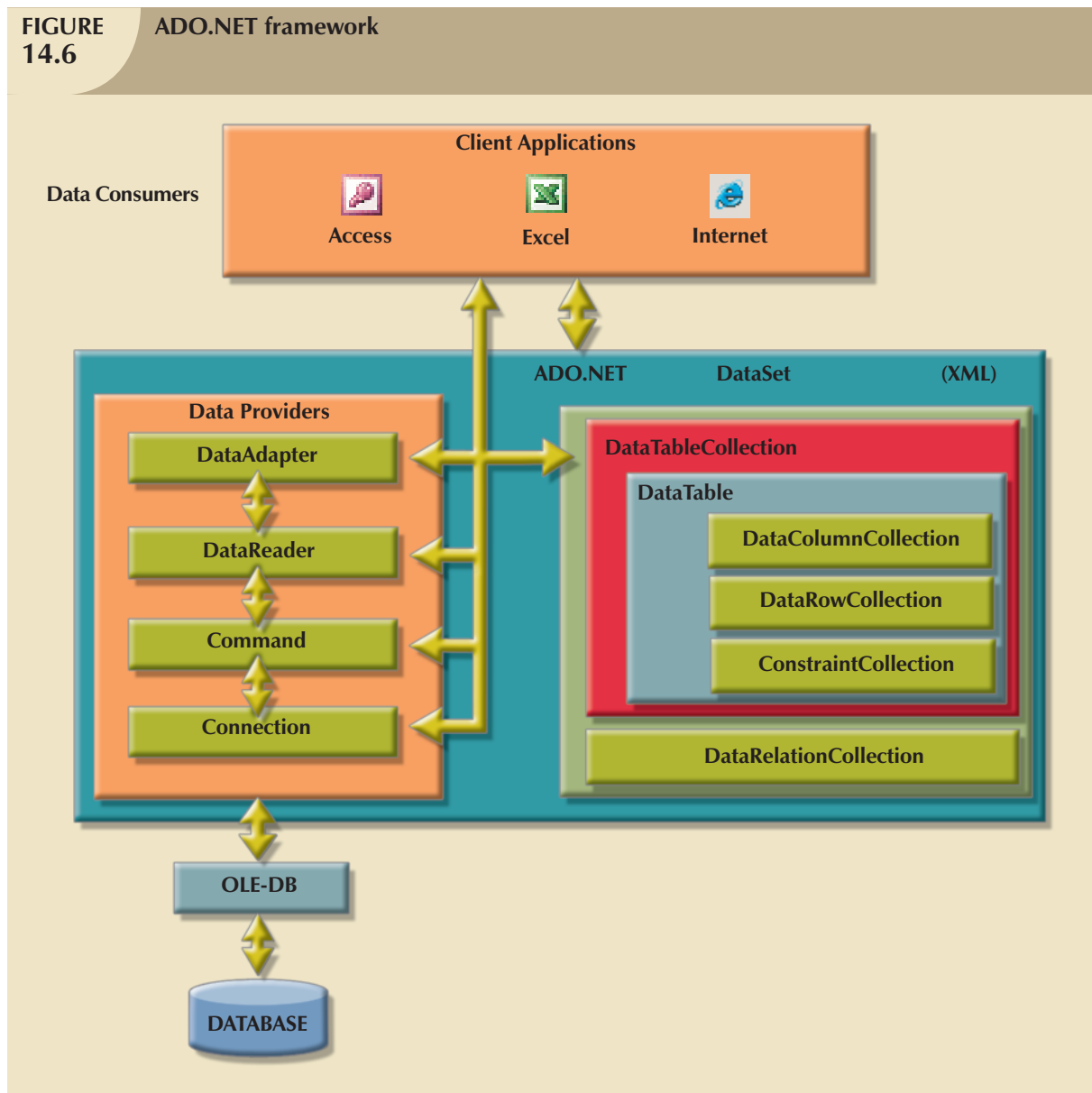
The ADO.NET framework consolidates all data access functionality under one integrated object model. In this object model, several objects interact with one another to perform specific data manipulation functions. Those objects can be grouped as data providers and consumers.

Data provider objects are provided by the database vendors. However, ADO.NET comes with two standard data providers: a data provider for OLE-DB data sources and a data provider for SQL Server. That way ADO.NET can work with any previously supported database, including an ODBC database with an OLE-DB data provider. At the same time, ADO.NET includes a highly optimized data provider for SQL Server.

Whatever the data provider is, it must support a set of specific objects in order to manipulate the data in the data source. Some of those objects are shown in Figure 14.6. A brief description of the objects follows.

- *Connection*. The Connection object defines the data source used, the name of the server, the database, and so on. This object enables the client application to open and close a connection to a database.
- *Command*. The Command object represents a database command to be executed within a specified database connection. This object contains the actual SQL code or a stored procedure call to be run by the database. When a SELECT statement is executed, the Command object returns a set of rows and columns.
- *DataReader*. The DataReader object is a specialized object that creates a read-only session with the database to retrieve data sequentially (forward only) in a very fast manner.
- *DataAdapter*. The DataAdapter object is in charge of managing a DataSet object. This is the most specialized object in the ADO.NET framework. The DataAdapter object contains the following objects that aid in managing the data in the DataSet: SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand. The DataAdapter object uses those objects to populate and synchronize the data in the DataSet with the permanent data source data.
- *DataSet*. The DataSet object is the in-memory representation of the data in the database. This object contains two main objects. The DataTableCollection object contains a collection of DataTable objects that make up the "in-memory" database, and the DataRelationCollection object contains a collection of objects describing the data relationships and ways to associate one row in a table to the related row in another table.

FIGURE 14.6 ADO.NET framework



- *DataTable*. The *DataTable* object represents the data in tabular format. This object has one very important property: *PrimaryKey*, which allows the enforcement of entity integrity. In turn, the *DataTable* object is composed of three main objects:
 - *DataColumnCollection* contains one or more column descriptions. Each column description has properties such as column name, data type, nulls allowed, maximum value, and minimum value.
 - *DataRowCollection* contains zero rows, one row, or more than one row with data as described in the *DataColumnCollection*.
 - *ConstraintCollection* contains the definition of the constraints for the table. Two types of constraints are supported: *ForeignKeyConstraint* and *UniqueConstraint*.

As you can see, a *DataSet* is, in fact, a simple database with tables, rows, and constraints. Even more important, the *DataSet* doesn't require a permanent connection to the data source. The *DataAdapter* uses the *SelectCommand* object to populate the *DataSet* from a data source. However, once the *DataSet* is populated, it is completely independent of the data source, which is why it's called "disconnected."

Additionally, `DataTable` objects in a `DataSet` can come from different data sources. This means that you could have an `EMPLOYEE` table in an Oracle database and a `SALES` table in a SQL Server database. You could then create a `DataSet` that relates both tables as though they were located in the same database. In short, the `DataSet` object paves the way for truly heterogeneous distributed database support within applications.

The ADO.NET framework is optimized to work in disconnected environments. In a disconnected environment, applications exchange messages in request/reply format. The most common example of a disconnected system is the Internet. Modern applications rely on the Internet as the network platform and on the Web browser as the graphical user interface. In the next section, you will learn details about how Internet databases work.

14.1.5 JAVA DATABASE CONNECTIVITY (JDBC)

Java is an object-oriented programming language developed by Sun Microsystems that runs on top of Web browser software. Java is one of the most common programming languages for Web development. Sun Microsystems created Java as a “write once, run anywhere” environment. That means that a programmer can write a Java application once and then without any modification, run the application in multiple environments (Microsoft Windows, Apple OS X, IBM AIX, etc.). The cross-platform capabilities of Java are based on its portable architecture. Java code is normally stored in pre-processed chunks known as applets that run on a virtual machine environment in the host operating system. This environment has well-defined boundaries and all interactivity with the host operating system is closely monitored. Sun provides Java runtime environments for most operating systems (from computers to hand-held devices to TV set-top boxes.) Another advantage of using Java is its “on-demand” architecture. When a Java application loads, it can dynamically download all its modules or required components via the Internet.

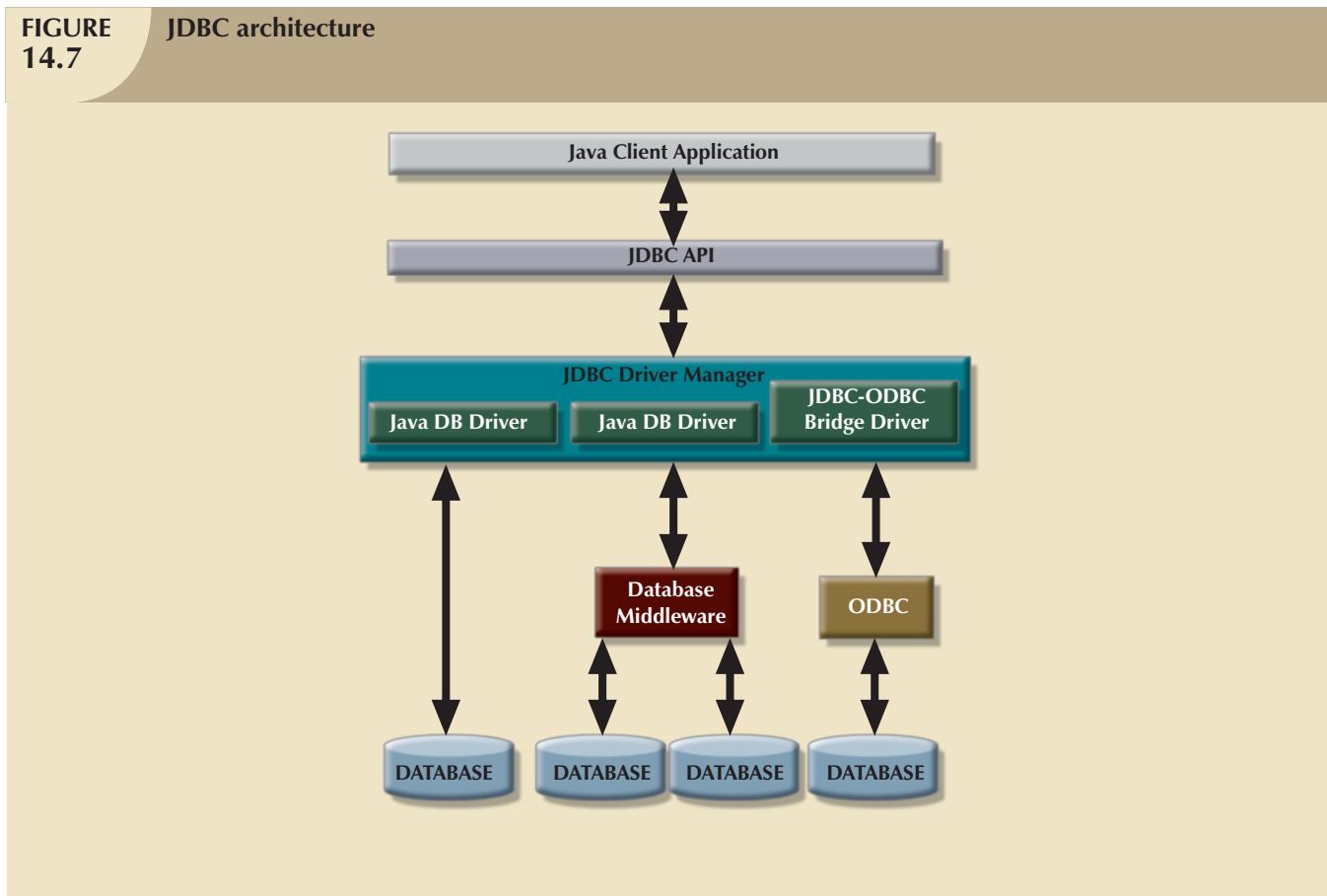
When Java applications want to access data outside the Java runtime environment, they use pre-defined application programming interfaces. **Java Database Connectivity (JDBC)** is an application programming interface that allows a Java program to interact with a wide range of data sources (relational databases, tabular data sources, spreadsheets, and text files). JDBC allows a Java program to establish a connection with a data source, prepare and send the SQL code to the database server, and process the result set.

One of the main advantages of JDBC is that it allows a company to leverage its existing investment in technology and personnel training. JDBC allows programmers to use their SQL skills to manipulate the data in the company's databases. As a matter of fact, JDBC allows direct access to a database server or access via database middleware. Furthermore, JDBC provides a way to connect to databases through an ODBC driver. Figure 14.7 illustrates the basic JDBC architecture and the various database access styles.

As you see in Figure 14.7, the database access architecture in JDBC is very similar to the ODBC/OLE/ADO.NET architecture. All database access middleware shares similar components and functionality. One advantage of JDBC over other middleware is that it requires no configuration on the client side. The JDBC driver is automatically downloaded and installed as part of the Java applet download. Because Java is a Web-based technology, applications can connect to a database directly using a simple URL. Once the URL is invoked, the Java architecture comes into place, the necessary applets are downloaded to the client (including the JDBC database driver and all configuration information), and then the applets are executed securely in the client's runtime environment.

Every day, more and more companies are investing resources in developing and expanding their Web presence and finding ways to do more business on the Internet. Such business will generate increasing amounts of data that will be stored in databases. Java and the .NET framework are part of the trend toward increasing reliance on the Internet as a critical business resource. In fact, it has been said that the Internet will become the development platform of the future. In the next section you will learn more about Internet databases and how they are used.

FIGURE 14.7 JDBC architecture



14.2 INTERNET DATABASES

Millions of people all over the world use computers and Web browser software to access the Internet, connecting to databases over the Web. Web database connectivity opens the door to new innovative services that:

- Permit rapid responses to competitive pressures by bringing new services and products to market quickly.
- Increase customer satisfaction through the creation of Web-based support services.
- Yield fast and effective information dissemination through universal access from across the street or across the globe.

Given those advantages, many organizations rely on their IS departments to create universal data access architectures based on Internet standards. Table 14.3 shows a sample of Internet technology characteristics and the benefits they provide.

TABLE 14.3 Characteristics and Benefits of Internet Technologies

INTERNET CHARACTERISTIC	BENEFIT
Hardware and software independence	Savings in equipment/software acquisition Ability to run on most existing equipment Platform independence and portability No need for multiple platform development
Common and simple user interface	Reduced training time and cost Reduced end-user support cost No need for multiple platform development
Location independence	Global access through Internet infrastructure Reduced requirements (and costs!) for dedicated connections
Rapid development at manageable costs	Availability of multiple development tools Plug-and-play development tools (open standards) More interactive development Reduced development times Relatively inexpensive tools Free client access tools (Web browsers) Low entry costs. Frequent availability of free Web servers Reduced costs of maintaining private networks Distributed processing and scalability, using multiple servers

In the current business and global information environment, it's easy to see why many database professionals consider the DBMS connection to the Internet to be a critical element in IS development. As you will learn in the following sections, database application development—and, in particular, the creation and management of user interfaces and database connectivity—are profoundly affected by the Web. However, having a Web-based database interface does not negate the database design and implementation issues that were addressed in the previous chapters. In the final analysis, whether you make a purchase by going online or by standing in line, the system-level transaction details are essentially the same, and they require the same basic database structures and relationships. If any immediate lesson is to be learned, it is this: *The effects of bad database design, implementation, and management are multiplied in an environment in which transactions might be measured in hundreds of thousands per day, rather than in hundreds per day.*

The Internet is rapidly changing the way information is generated, accessed, and distributed. At the core of this change is the Web's ability to access data in databases (local and remote), the simplicity of the interface, and cross-platform (heterogeneous) functionality. The Web has helped create a new information dissemination standard.

The following sections examine how Web-to-database middleware enables end users to interact with databases over the Web.

14.2.1 WEB-TO-DATABASE MIDDLEWARE: SERVER-SIDE EXTENSIONS

In general, the Web server is the main hub through which all Internet services are accessed. For example, when an end user uses a Web browser to dynamically query a database, the client browser requests a Web page. When the Web server receives the page request, it looks for the page on the hard disk; when it finds the page (for example, a stock quote, product catalog information, or an airfare listing), the server sends it back to the client.



ONLINE CONTENT

Client/server systems are covered in detail in **Appendix F, Client/Server Systems**, located in the Student Online Companion for this book.

Dynamic Web pages are at the heart of current generation Web sites. In this database-query scenario, the Web server generates the Web page contents before it sends the page to the client Web browser. The only problem with the preceding query scenario is that the Web server must include the database query result on the page *before* it sends that page back to the client. Unfortunately, neither the Web browser nor the Web server knows how to connect to and read data from the database. Therefore, to support this type of request (database query), the Web server's capability must be extended so it can understand and process database requests. This job is done through a server-side extension.

A **server-side extension** is a program that interacts directly with the Web server to handle specific types of requests. In the preceding database query example, the server-side extension program retrieves the data from databases and passes the retrieved data to the Web server, which, in turn, sends the data to the client's browser for display purposes. The server-side extension makes it possible to retrieve and present the query results, but what's more important is that *it provides its services to the Web server in a way that is totally transparent to the client browser*. In short, the server-side extension adds significant functionality to the Web server, and therefore, to the Internet.

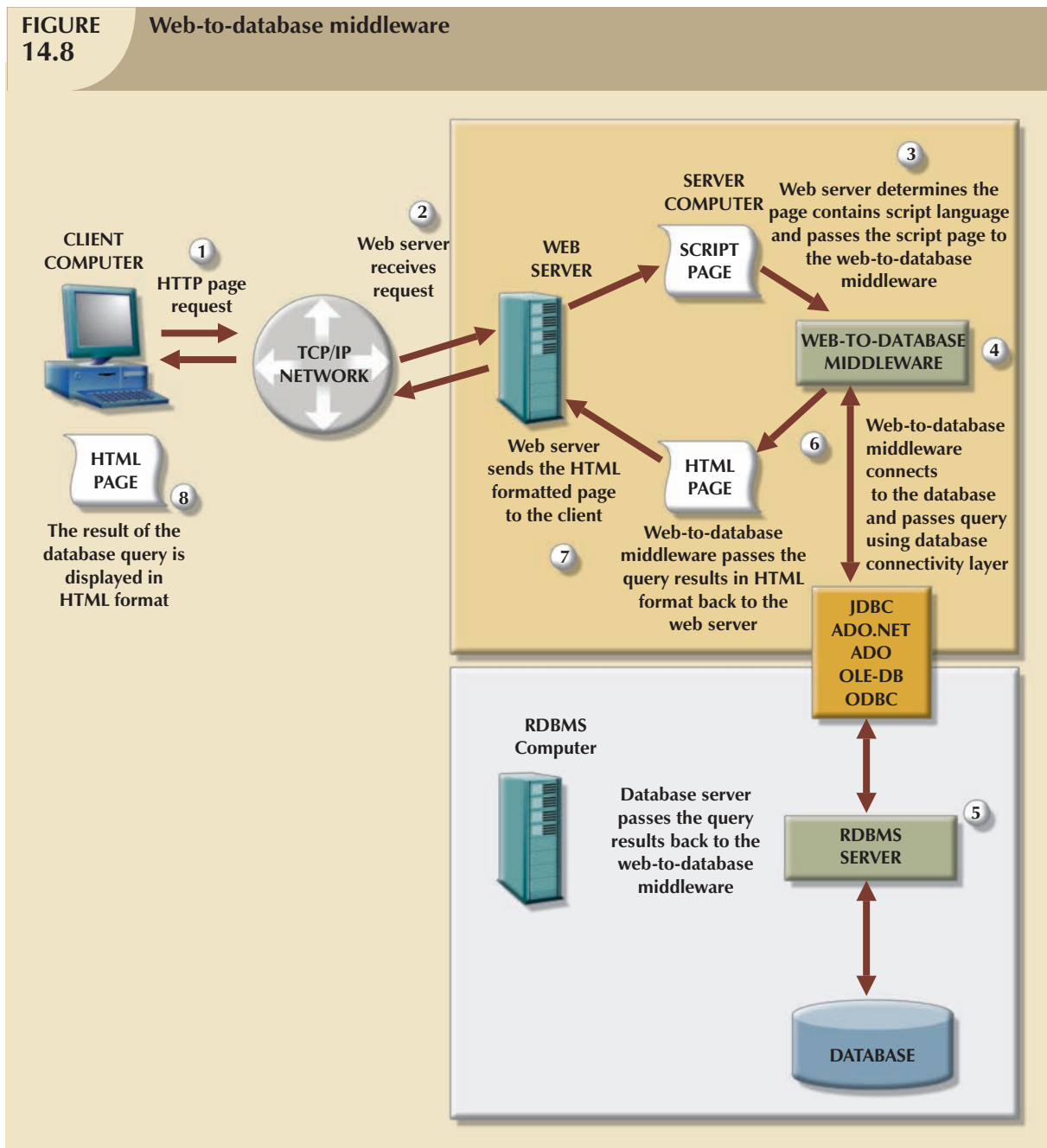
A database server-side extension program is also known as **Web-to-database middleware**. Figure 14.8 shows the interaction between the browser, the Web server, and the Web-to-database middleware.

Trace the Web-to-database middleware actions in Figure 14.8:

1. The client browser sends a page request to the Web server.
2. The Web server receives and validates the request. In this case, the server will pass the request to the Web-to-database middleware for processing. Generally, the requested page contains some type of scripting language to enable the database interaction.
3. The Web-to-database middleware reads, validates, and executes the script. In this case, it connects to the database and passes the query using the database connectivity layer.
4. The database server executes the query and passes the result back to the Web-to-database middleware.
5. The Web-to-database middleware compiles the result set, dynamically generates an HTML-formatted page that includes the data retrieved from the database, and sends it to the Web server.
6. The Web server returns the just-created HTML page, which now includes the query result, to the client browser.
7. The client browser displays the page on the local computer.

The interaction between the Web server and the Web-to-database middleware is crucial to the development of a successful Internet database implementation. Therefore, the middleware must be well integrated with the other Internet services and the components that are involved in its use. For example, when installing Web-to-database middleware, the middleware must verify the type of Web server being used and install itself to match that Web server's requirements. In addition, how well the Web server and the Web-to-database service interact will depend on the Web server interfaces that are supported by the Web server.

FIGURE 14.8 Web-to-database middleware



14.2.2 WEB SERVER INTERFACES

Extending Web server functionality implies that the Web server and the Web-to-database middleware will properly communicate with each other. (Database professionals often use the word *interoperate* to indicate that each party can respond to the communications of the other. This book's use of *communicate* assumes interoperation.) If a Web server is to communicate successfully with an external program, both programs must use a standard way to exchange messages and to respond to requests. A Web server interface defines how a Web server communicates with external programs. Currently, there are two well-defined Web server interfaces:

- Common Gateway Interface (CGI).
- Application programming interface (API).

The **Common Gateway Interface (CGI)** uses script files that perform specific functions based on the client's parameters that are passed to the Web server. The script file is a small program containing commands written in a programming language—usually Perl, C++, or Visual Basic. The script file's contents can be used to connect to the database and to retrieve data from it, using the parameters passed by the Web server. Next, the script converts the retrieved data to HTML format and passes the data to the Web server, which sends the HTML-formatted page to the client.

The main disadvantage of using CGI scripts is that the script file is an external program that is individually executed for each user request. That scenario decreases system performance. For example, if you have 200 concurrent requests, the script is loaded 200 *different* times, which takes significant CPU and memory resources away from the Web server. The language and method used to create the script also can affect system performance. For example, performance is degraded by using an interpreted language or by writing the script inefficiently.

An application programming interface (API) is a newer Web server interface standard that is more efficient and faster than a CGI script. APIs are more efficient because they are implemented as shared code or as dynamic-link libraries (DLLs). That means the API is treated as part of the Web server program that is dynamically invoked when needed.

APIs are faster than CGI scripts because the code resides in memory, so there is no need to run an external program for each request. Instead, the same API serves all requests. Another advantage is that an API can use a shared connection to the database instead of creating a new one every time, as is the case with CGI scripts.

Although APIs are more efficient in handling requests, they have some disadvantages. Because the APIs share the same memory space as the Web server, an API error can bring down the server. The other disadvantage is that APIs are specific to the Web server and to the operating system.

At the time of this writing, there are four well-established Web server APIs:

- Netscape API (NSAPI) for Netscape servers.
- Internet Server API (ISAPI) for Microsoft Windows Web servers.
- WebSite API (WSAPI) for O'Reilly Web servers.
- JDBC to provide database connectivity for Java applications.

The various types of Web interfaces are illustrated in Figure 14.9.

Regardless of the type of Web server interface used, the Web-to-database middleware program must be able to connect with the database. That connection can be accomplished in one of two ways:

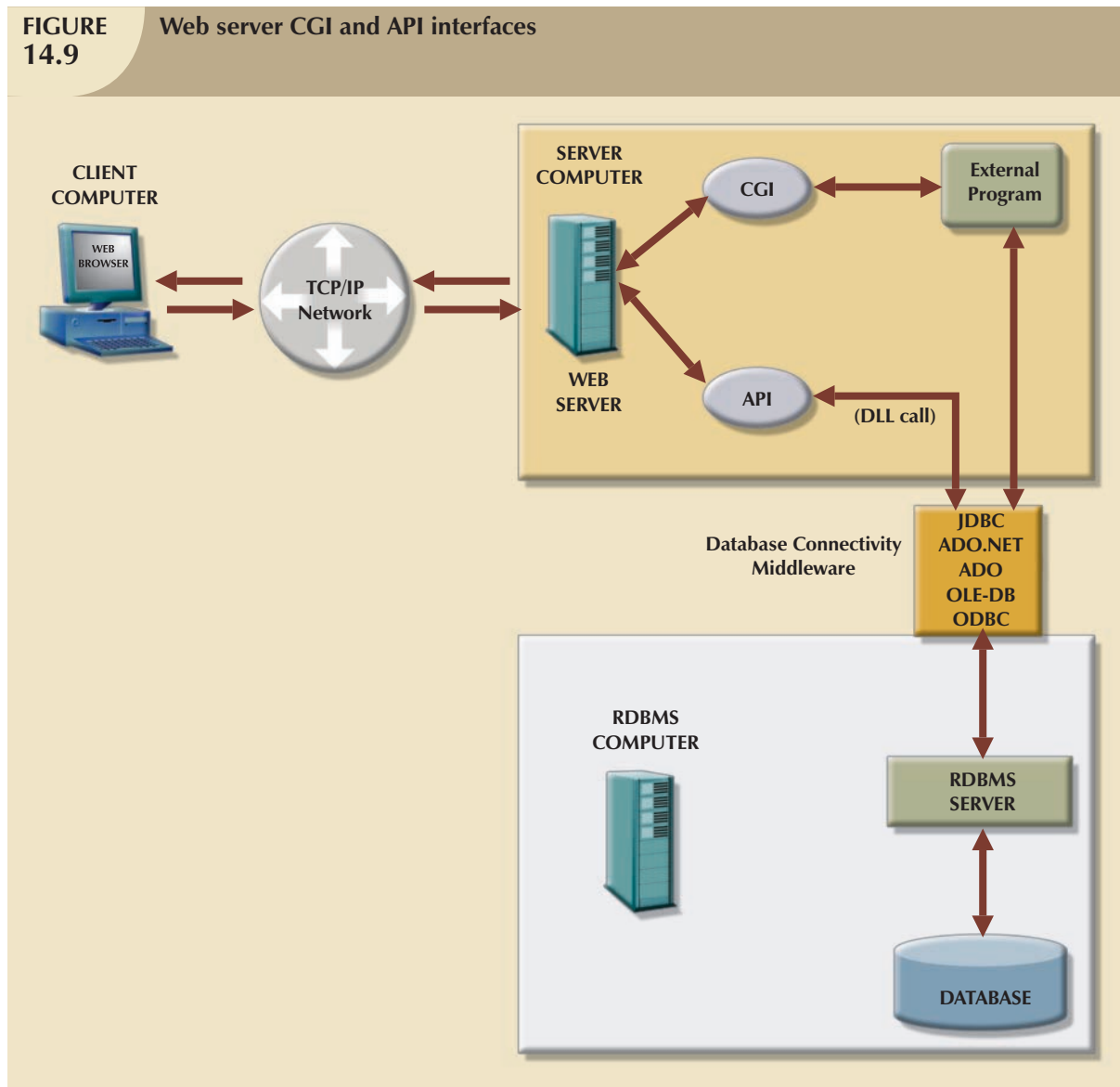
- Use the native SQL access middleware provided by the vendor. For example, you can use SQL*Net if you are using Oracle.
- Use the services of general database connectivity standards such as Open Database Connectivity (ODBC), Object Linking and Embedding for Database (OLE-DB), ActiveX Data Objects (ADO), the ActiveX Data Objects for .NET (ADO.NET) interface, or JDBC for Java connectivity.

14.2.3 THE WEB BROWSER

The Web browser is the application software in the client computer, such as Microsoft Internet Explorer, Apple Safari, or Mozilla Firefox, that lets end users navigate (browse) the Web. Each time the end user clicks a hyperlink, the browser generates an HTTP GET page request that is sent to the designated Web server, using the TCP/IP Internet protocol.

The Web browser's job is to *interpret* the HTML code that it receives from the Web server and to present the various page components in a standard formatted way. Unfortunately, the browser's interpretation and presentation capabilities are not sufficient to develop Web-based applications. That is because the Web is a **stateless system**—which means that at any given time, a Web server does not know the status of any of the clients communicating with it. That is, there is no open communication line between the server and each client accessing it, which, of course, is impractical in a *worldwide* Web! Instead, client and server computers interact in very short “conversations” that follow

FIGURE 14.9 Web server CGI and API interfaces



the request-reply model. For example, the browser is concerned only with the *current* page, so there is no way for the second page to know what was done in the first page. The only time the client and server computers communicate is when the client requests a page—when the user clicks a link—and the server sends the requested page to the client. Once the client receives the page and its components, the client/server communication is ended. Therefore, although you may be browsing a page and *think* that the communication is open, you are actually just browsing the HTML document stored in the local cache (temporary directory) of your browser. The server does not have any idea what the end user is doing with the document, what data is entered in a form, what option is selected, and so on. On the Web, if you want to act on a client's selection, you need to jump to a new page (go back to the Web server), therefore losing track of whatever was done before!

A Web browser's function is to display a page on the client computer. The browser—through its use of HTML—does not have computational abilities beyond formatting output text and accepting form field inputs. Even when the browser accepts form field data, there is no way to perform immediate data entry validation. Therefore, to perform such crucial processing in the client, the Web defers to other Web programming languages such as Java, JavaScript, and VBScript. The browser resembles a dumb terminal that displays only data and can perform only rudimentary processing such as

accepting form data inputs. To improve capabilities on the client side of the Web browser, you must use plug-ins and other client-side extensions. On the server side, Web application servers provide the necessary processing power.

14.2.4 CLIENT-SIDE EXTENSIONS

Client-side extensions add functionality to the Web browser. Although client-side extensions are available in various forms, the most commonly encountered extensions are:

- Plug-ins.
- Java and JavaScript.
- ActiveX and VBScript.

A **plug-in** is an external application that is automatically invoked by the browser when needed. Because it is an *external* application, the plug-in is operating-system specific. The plug-in is associated with a data object—generally using the file extension—to allow the Web server to properly handle data that are not originally supported. For example, if one of the page components is a PDF document, the Web server will receive the data, recognize it as a “portable document format” object, and launch Adobe Acrobat Reader to present the document on the client computer.

As noted earlier, Java runs on top of the Web browser software. Java applications are compiled and stored in the Web server. (In many respects, Java resembles C++.) Calls to Java routines are embedded inside the HTML page. When the browser finds this call, it downloads the Java classes (code) from the Web server and runs that code in the client computer. Java’s main advantage is that it enables application developers to develop their applications once and run them in many environments. (For developing Web applications, interoperability is a very important issue. Unfortunately, different client browsers are not 100 percent interoperable, thus limiting portability.)

JavaScript is a scripting language (one that enables the running of a series of commands or macros) that allows Web authors to design interactive sites. Because JavaScript is simpler to generate than Java, it is easier to learn. JavaScript code is embedded in the Web pages. It is downloaded with the Web page and is activated when a specific event takes place—such as a mouse click on an object or a page being loaded from the server into memory.

ActiveX is Microsoft’s alternative to Java. ActiveX is a specification for writing programs that will run inside the Microsoft client browser (Internet Explorer). Because ActiveX is oriented mainly to Windows applications, it has low portability. ActiveX extends the Web browser by adding “controls” to Web pages. (Examples of such controls are drop-down lists, a slider, a calendar, and a calculator.) Those controls, downloaded from the Web server when needed, let you manipulate data inside the browser. ActiveX controls can be created in several programming languages; C++ and Visual Basic are most commonly used. Microsoft’s .NET framework allows for wider interoperability of ActiveX-based applications (such as ADO.NET) across multiple operating environments.

VBScript is another Microsoft product that is used to extend browser functionality. VBScript is derived from Microsoft Visual Basic. Like JavaScript, VBScript code is embedded inside an HTML page and is activated by triggering events such as clicking a link.

From the developer’s point of view, using routines that permit data validation on the client side is an absolute necessity. For example, when data are entered on a Web form and no data validation is done on the client side, the entire data set must be sent to the Web server. That scenario requires the server to perform all data validation, thus wasting valuable CPU processing cycles. Therefore, client-side data input validation is one of the most basic requirements for Web applications. Most of the data validation routines are done in Java, JavaScript, ActiveX, or VBScript.

14.2.5 WEB APPLICATION SERVERS

A **Web application server** is a middleware application that expands the functionality of Web servers by linking them to a wide range of services, such as databases, directory systems, and search engines. The Web application server also provides a consistent run-time environment for Web applications.

Web application servers can be used to:

- Connect to and query a database from a Web page.
- Present database data in a Web page, using various formats.
- Create dynamic Web search pages.
- Create Web pages to insert, update, and delete database data.
- Enforce referential integrity in the application program logic.
- Use simple and nested queries and programming logic to represent business rules.

Web application servers provide features such as:

- An integrated development environment with session management and support for persistent application variables.
- Security and authentication of users through user IDs and passwords.
- Computational languages to represent and store business logic in the application server.
- Automatic generation of HTML pages integrated with Java, JavaScript, VBScript, ASP, and so on.
- Performance and fault-tolerant features.
- Database access with transaction management capabilities.
- Access to multiple services, such as file transfers (FTP), database connectivity, e-mail, and directory services.

As of this writing, popular Web application servers include ColdFusion by Adobe, Oracle Application Server by Oracle, WebLogic by BEA Systems, NetDynamics by Sun Microsystems, Fusion by NetObjects, Visual Studio.NET by Microsoft, and WebObjects by Apple. All Web application servers offer the ability to connect Web servers to multiple data sources and other services. They vary in terms of the range of available features, robustness, scalability, ease of use, compatibility with other Web and database tools, and extent of the development environment.



ONLINE CONTENT

To see and try a particular Web-to-database interface in action, consult **Appendix J, Web Database Development with ColdFusion**, in the Student Online Companion for this book. This appendix steps you through the process of creating and using a simple Web-to-database interface, and gives more detailed information on developing Web databases with Adobe ColdFusion middleware.

Current-generation systems involve more than just the development of Web-enabled database applications. They also require applications capable of intercommunicating with each other and with other systems not based on the Web. Clearly, systems must be able to exchange data in a standard-based format. That's the role of XML.

14.3 EXTENSIBLE MARKUP LANGUAGE (XML)

The Internet has brought about new technologies that facilitate the exchange of business data among business partners and consumers. Companies are using the Internet to create new types of systems that integrate their data to increase efficiency and reduce costs. Electronic commerce (*e-commerce*) enables all types of organizations to market and sell products and services to a global market of millions of users. E-commerce transactions—the sale of products or services—can take place between businesses (business-to-business, or B2B) or between a business and a consumer (business-to-consumer, or B2C).

Most e-commerce transactions take place between businesses. Because B2B e-commerce integrates business processes among companies, it requires the transfer of business information among different business entities. But the way in which businesses represent, identify, and use data tends to differ substantially from company to company. (Is a *product code* the same thing as an *item ID*?)

Until recently, the expectation was that a purchase order traveling over the Web would be in the form of an HTML document. The HTML Web page displayed on the Web browser would include formatting tags as well as the order details. HTML **tags** describe how something *looks* on the Web page, such as bold type or heading style, and often come in pairs to start and end formatting features. For example, the following HTML tags would put the words FOR SALE in bold in the Arial font:

```
<strong><font face=Arial>FOR SALE</font></strong>
```

If an application wants to get the order data from the Web page, there is no easy way to extract the order details (such as the order number, the date, the customer number, the item, the quantity, the price, or payment details) from an HTML document. The HTML document can only describe how to display the order in a Web browser; it does not permit the manipulation of the order's data elements, that is, date, shipping information, payment details, product information, and so on. To solve that problem, a new markup language, known as Extensible Markup Language, or XML, was developed.

Extensible Markup Language (XML) is a metalanguage used to represent and manipulate data elements. XML is designed to facilitate the exchange of structured documents, such as orders and invoices, over the Internet. The World Wide Web Consortium (W3C)¹ published the first XML 1.0 standard definition in 1998. That standard sets the stage for giving XML the real-world appeal of being a true vendor-independent platform. Therefore, it is not surprising that XML has rapidly become the data exchange standard for e-commerce applications.

The XML metalanguage allows the definition of new tags, such as <ProdPrice>, to describe the data elements used in an XML document. This ability to *extend* the language explains the *X* in XML; the language is said to be *extensible*. XML is derived from the Standard Generalized Markup Language (SGML), an international standard for the publication and distribution of highly complex technical documents. For example, documents used by the aviation industry and the military services are too complex and unwieldy for the Web. Just like HTML, which was also derived from SGML, an XML document is a text file. However, it has a few very important additional characteristics, as follows:

- XML allows the definition of new tags to describe data elements, such as <ProductId>.
- XML is case sensitive: <ProductID> is not the same as <Productid>.
 - XML tags must be well formed; that is, each opening tag has a corresponding closing tag. For example, the product identification would require the format <ProductId>2345-AA</ProductId>.
 - XML tags must be properly nested. For example, a properly nested XML tag might look like this:
<Product><ProductId>2345-AA</ProductId></Product>.
- You can use the <-- and --> symbols to enter comments in the XML document.
- The *XML* and *xml* prefixes are reserved for XML tags only.

XML is *not* a new version or replacement for HTML. XML is concerned with the description and representation of the data, rather than the way the data are displayed. XML provides the semantics that facilitate the sharing, exchange, and manipulation of structured documents over organizational boundaries. In short, XML and HTML perform complementary, rather than overlapping, functions. Extensible Hypertext Markup Language (XHTML) is the next generation of HTML based on the XML framework. The XHTML specification expands the HTML standard to include XML features. Although more powerful than HTML, XHTML requires very strict adherence to syntax requirements.

¹You can visit the W3C Web page, located at www.w3.org, to get additional information about the efforts that were made to develop the XML standard.

As an illustration of the use of XML for data exchange purposes, consider a B2B example in which Company A uses XML to exchange product data with Company B over the Internet. Figure 14.10 shows the contents of the ProductList.xml document.

FIGURE 14.10 Contents of the productlist.xml document

```

productlist.xml - Notepad
File Edit Format View Help
<?xml version="1.0"?>
<ProductList>
  <Product>
    <P_CODE>23109-HB</P_CODE>
    <P_DESCRIPT>Claw hammer</P_DESCRIPT>
    <P_INDATE>08/19/2007</P_INDATE>
    <P_QOH>23</P_QOH>
    <P_MIN>10</P_MIN>
    <P_PRICE>5.95</P_PRICE>
  </Product>
  <Product>
    <P_CODE>23114-AA</P_CODE>
    <P_DESCRIPT>Sledge Hammer, 12 lb.</P_DESCRIPT>
    <P_INDATE>09/01/2007</P_INDATE>
    <P_QOH>8</P_QOH>
    <P_MIN>5</P_MIN>
    <P_PRICE>14.40</P_PRICE>
  </Product>
</ProductList>

```

The XML example shown in Figure 14.10 illustrates several important XML features, as follows:

- The first line represents the XML document declaration, and it is mandatory.
- Every XML document has a *root element*. In the example, the second line declares the ProductList root element.
- The root element contains *child elements* or sub-elements. In the example, line 3 declares Product as a child element of ProductList.
- Each element can contain *sub-elements*. For example, each Product element is composed of several child elements, represented by P_CODE, P_DESCRIPT, P_INDATE, P_QOH, P_MIN, and P_PRICE.
- The XML document reflects a hierarchical tree structure where elements are related in a parent-child relationship; each parent element can have many children elements. For example, the root element is ProductList. Product is the child element of ProductList. Product has six child elements: P_CODE, P_DESCRIPT, P_INDATE, P_QOH, P_MIN, and P_PRICE.

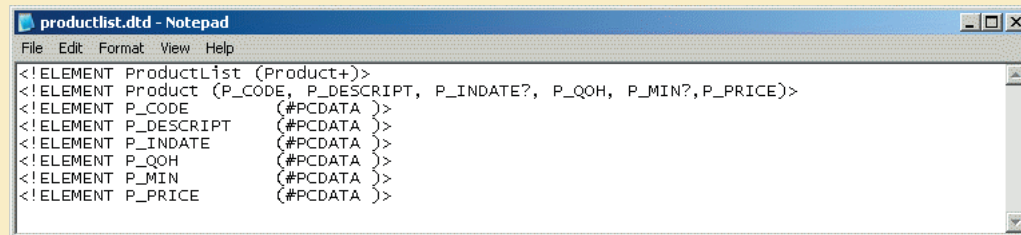
Once Company B receives the ProductList.xml document, it can process the document—assuming it understands the tags created by Company A. The meaning of the XML tags in the example shown in Figure 14.10 is fairly self-evident, but there is no easy way to validate the data or to check whether the data are complete. For example, you could encounter a P_INDATE value of “25/14/2007”—but is that value correct? And what happens if Company B expects a Vendor element as well? How can companies share data descriptions about their business data elements? The next section will show how document type definitions and XML schemas are used to address those concerns.

14.3.1 DOCUMENT TYPE DEFINITIONS (DTD) AND XML SCHEMAS

B2B solutions require a high degree of business integration between companies. Companies that use B2B transactions must have a way to understand and validate each other’s tags. One way to accomplish that task is through the use of Document Type Definitions. A **Document Type Definition (DTD)** is a file with a .dtd extension that describes XML elements—in effect, a DTD file provides the composition of the database’s logical model and defines the syntax rules

or valid tags for each type of XML document. (The DTD component is similar to having a public data dictionary for business data.) Companies that intend to engage in e-commerce business transactions must develop and share DTDs. Figure 14.11 shows the productlist.dtd document for the productlist.xml document shown earlier in Figure 14.10.

FIGURE 14.11 Contents of the productlist.dtd document



```

productlist.dtd - Notepad
File Edit Format View Help
<!ELEMENT ProductList (Product+)>
<!ELEMENT Product (P_CODE, P_DESCRIPT, P_INDATE?, P_QOH, P_MIN?,P_PRICE)>
<!ELEMENT P_CODE (#PCDATA )>
<!ELEMENT P_DESCRIPT (#PCDATA )>
<!ELEMENT P_INDATE (#PCDATA )>
<!ELEMENT P_QOH (#PCDATA )>
<!ELEMENT P_MIN (#PCDATA )>
<!ELEMENT P_PRICE (#PCDATA )>

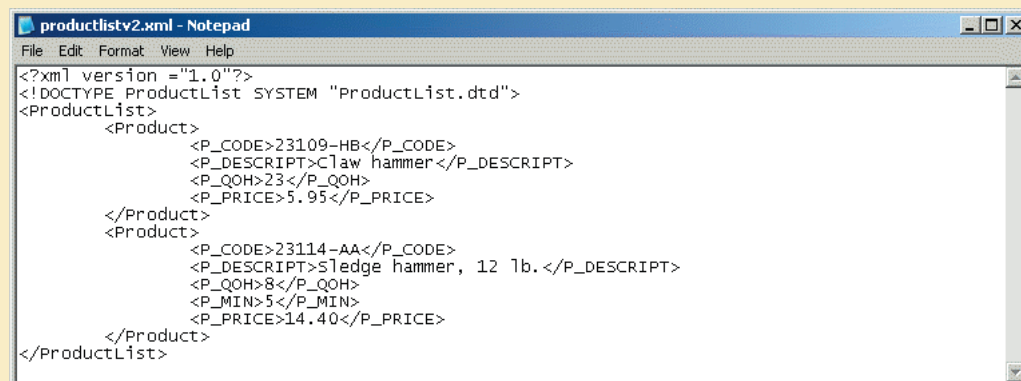
```

In Figure 14.11, note that the productlist.dtd file provides definitions of the elements in the productlist.xml document. In particular, note that:

- The first line declares the ProductList root element.
- The ProductList root element has one child, the Product element.
- The plus “+” symbol indicates that Product occurs one or more times within ProductList.
- An asterisk “*” would mean that the child element occurs zero or more times.
- A question mark “?” would mean that the child element is optional.
- The second line describes the Product element.
- The question mark “?” after the P_INDATE and P_MIN indicates that they are optional elements.
- The third through eighth lines show that the Product element has six child elements.
- The #PCDATA keyword represents the actual text data.

To be able to use a DTD file to define elements within an XML document, the DTD must be referenced from within that XML document. Figure 14.12 shows the productlistv2.xml document that includes the reference to the productlist.dtd in the second line.

FIGURE 14.12 Contents of the productlistv2.xml document



```

productlistv2.xml - Notepad
File Edit Format View Help
<?xml version="1.0"?>
<!DOCTYPE ProductList SYSTEM "ProductList.dtd">
<ProductList>
  <Product>
    <P_CODE>23109-HB</P_CODE>
    <P_DESCRIPT>Claw hammer</P_DESCRIPT>
    <P_QOH>23</P_QOH>
    <P_PRICE>5.95</P_PRICE>
  </Product>
  <Product>
    <P_CODE>23114-AA</P_CODE>
    <P_DESCRIPT>Sledge hammer, 12 lb.</P_DESCRIPT>
    <P_QOH>8</P_QOH>
    <P_MIN>5</P_MIN>
    <P_PRICE>14.40</P_PRICE>
  </Product>
</ProductList>

```


In Figure 14.12, note that the P_INDATE and P_MIN do not appear in all Product definitions because they were declared to be optional elements. The DTD can be referenced by many XML documents of the same type. For example, if Company A routinely exchanges product data with Company B, it will need to create the DTD only once. All subsequent XML documents will refer to the DTD, and Company B will be able to verify the data being received.

To further demonstrate the use of XML and DTD for e-commerce business data exchanges, assume the case of two companies exchanging order data. Figure 14.13 shows the DTD and XML documents for that scenario.

FIGURE 14.13 DTD and XML documents for order data

OrderData.dtd

```

<!ELEMENT orderData (ORD_ID,ORD_DATE,CUS_NAME,ORD_SHIPTO,ORD_PRODS*,ORD_TOT)>
<!ELEMENT ORD_ID (#PCDATA )>
<!ELEMENT ORD_DATE (#PCDATA )>
<!ELEMENT CUS_NAME (#PCDATA )>
<!ELEMENT ORD_SHIPTO (#PCDATA )>
<!ELEMENT ORD_PRODS (P_CODE, P_DESCRIPT, P_QOH, P_PRICE)+>
<!ELEMENT P_CODE (#PCDATA )>
<!ELEMENT P_DESCRIPT (#PCDATA )>
<!ELEMENT P_QOH (#PCDATA )>
<!ELEMENT P_PRICE (#PCDATA )>
<!ELEMENT ORD_TOT (#PCDATA )>

```

OrderData.xml

```

<?xml version = "1.0"?>
<!DOCTYPE OrderData SYSTEM "orderData.dtd">
<orderData>
  <ORD_ID>34583</ORD_ID>
  <ORD_DATE>12/08/2007</ORD_DATE>
  <CUS_NAME>Jill Atkins</CUS_NAME>
  <ORD_SHIPTO>1234 Crown Rd, Chicago, IL34564</ORD_SHIPTO>
  <ORD_PRODS>
    <P_CODE>2309-HB</P_CODE>
    <P_DESCRIPT>Claw Hammer</P_DESCRIPT>
    <P_QOH>2</P_QOH>
    <P_PRICE>5.95</P_PRICE>
  </ORD_PRODS>
  <ORD_PRODS>
    <P_CODE>23114-AA</P_CODE>
    <P_DESCRIPT>Sledge Hammer, 12 lb.</P_DESCRIPT>
    <P_QOH>1</P_QOH>
    <P_PRICE>14.40</P_PRICE>
  </ORD_PRODS>
  <ORD_TOT>26.30</ORD_TOT>
</orderData>

```

“+” sign indicates one or more ORD_PRODS elements

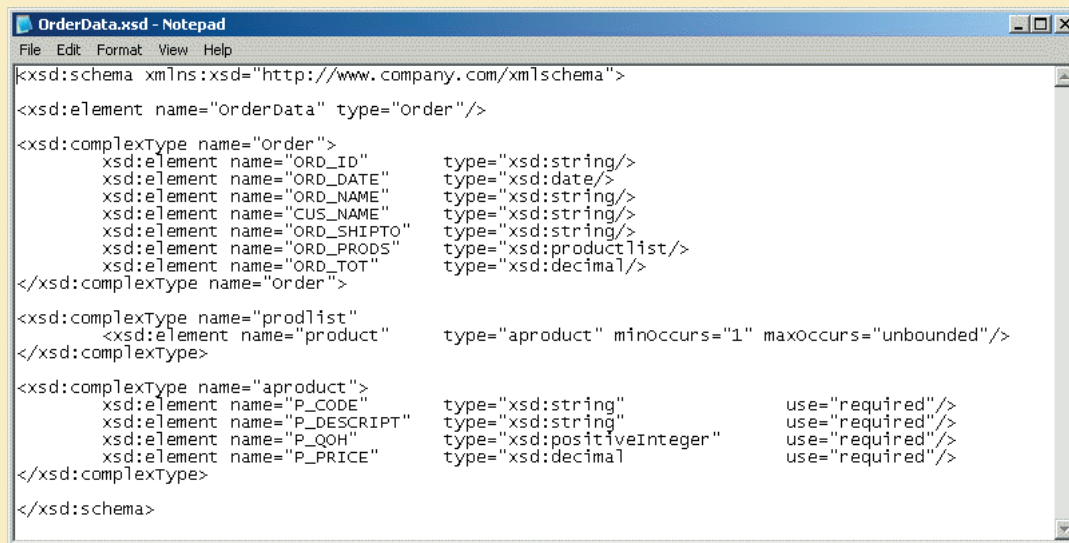
Two ORD_PRODS elements in XML document

Although the use of DTDs is a great improvement for data sharing over the Web, a DTD provides only descriptive information for understanding how the elements—root, parent, child, mandatory, or optional—relate to one another. A DTD provides limited additional semantic value, such as data type support or data validation rules. That information is very important for database administrators who are in charge of large e-commerce databases. To solve the DTD problem, the W3C published an XML Schema standard in May 2001 to provide a better way to describe XML data.

The **XML schema** is an advanced data definition language that is used to describe the structure (elements, data types, relationship types, ranges, and default values) of XML data documents. One of the main advantages of an XML schema is that it more closely maps to database terminology and features. For example, an XML schema will be able to define common database types such as date, integer or decimal, minimum and maximum values, list of valid values, and required elements. Using the XML schema, a company would be able to validate the data for values that may be out of range, incorrect dates, valid values, and so on. For example, a university application must be able to specify that a GPA value be between zero and 4.0, and it must be able to detect an invalid birth date such as "14/13/1987." (There is no 14th month.) Many vendors are adopting this new standard and are supplying tools to translate DTD documents into XML Schema Definition (XSD) documents. It is widely expected that XML schemas will replace DTD as the method to describe XML data.

Unlike a DTD document, which uses a unique syntax, an **XML schema definition (XSD)** file uses a syntax that resembles an XML document. Figure 14.14 shows the XSD document for the OrderData XML document.

FIGURE 14.14 The XML schema document for the order data



```

OrderData.xsd - Notepad
File Edit Format View Help
<?xml:stylesheet href="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" type="text/css" media="all" data-bbox="205 406 781 640"/>
<xsd:schema xmlns:xsd="http://www.company.com/xmlschema">
  <xsd:element name="orderData" type="order"/>
  <xsd:complexType name="order">
    xsd:element name="ORD_ID" type="xsd:string/>
    xsd:element name="ORD_DATE" type="xsd:date/>
    xsd:element name="ORD_NAME" type="xsd:string/>
    xsd:element name="CUS_NAME" type="xsd:string/>
    xsd:element name="ORD_SHIPTO" type="xsd:string/>
    xsd:element name="ORD_PRODS" type="xsd:productlist/>
    xsd:element name="ORD_TOT" type="xsd:decimal/>
  </xsd:complexType name="order">
  <xsd:complexType name="prodlist">
    <xsd:element name="product" type="aproduct" minOccurs="1" maxoccurs="unbounded"/>
  </xsd:complexType>
  <xsd:complexType name="aproduct">
    xsd:element name="P_CODE" type="xsd:string" use="required"/>
    xsd:element name="P_DESCRIPT" type="xsd:string" use="required"/>
    xsd:element name="P_QOH" type="xsd:positiveInteger" use="required"/>
    xsd:element name="P_PRICE" type="xsd:decimal" use="required"/>
  </xsd:complexType>
</xsd:schema>

```

The code shown in Figure 14.14 is a simplified version of the XML schema document. As you can see, the XML schema syntax is similar to the XML document syntax. In addition, the XML schema introduces additional semantic information for the OrderData XML document, such as string, date, and decimal data types; required elements; and minimum and maximum cardinalities for the data elements.

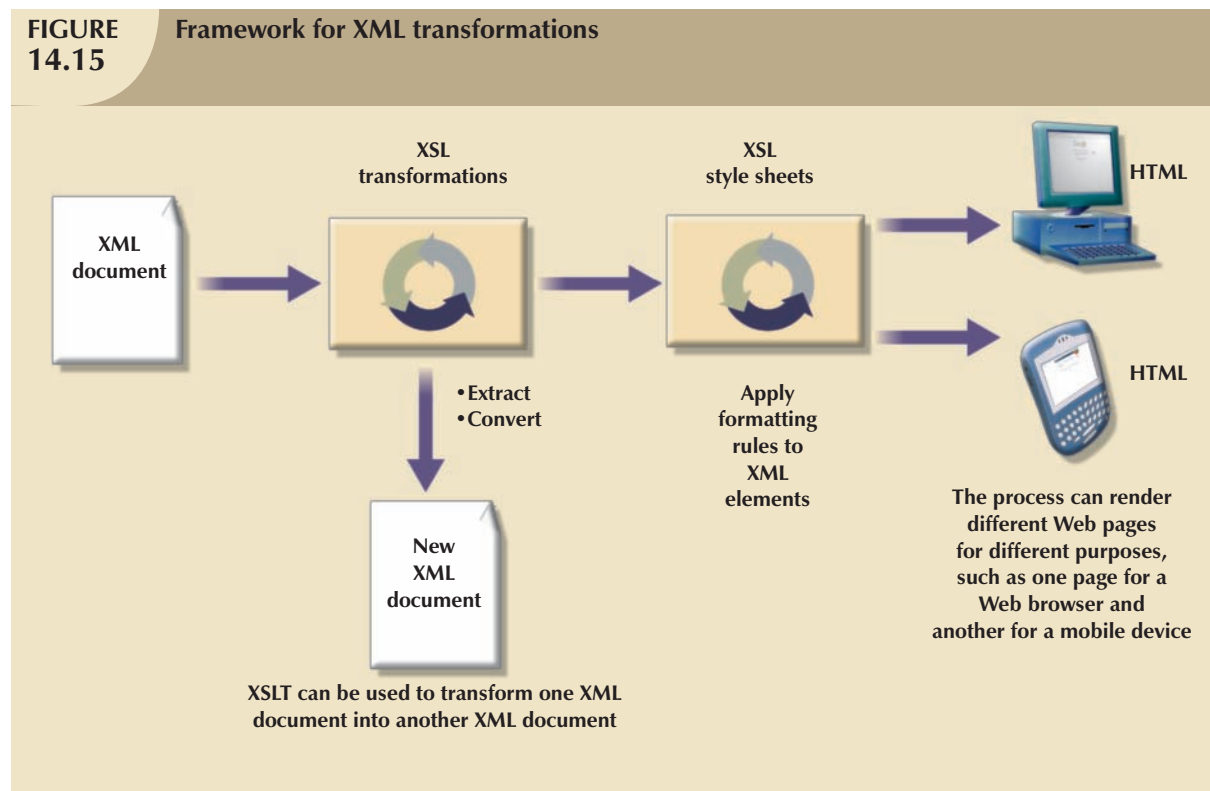
14.3.2 XML PRESENTATION

One of the main benefits of XML is that it separates data structure from its presentation and processing. By separating data and presentation, you are able to present the same data in different ways—which is similar to having views in SQL. But what mechanisms are used to present data?

The Extensible Style Language (XSL) specification provides the mechanism to display XML data. XSL is used to define the rules by which XML data are formatted and displayed. The XSL specification is divided in two parts: Extensible Style Language Transformations (XSLT) and XSL style sheets.

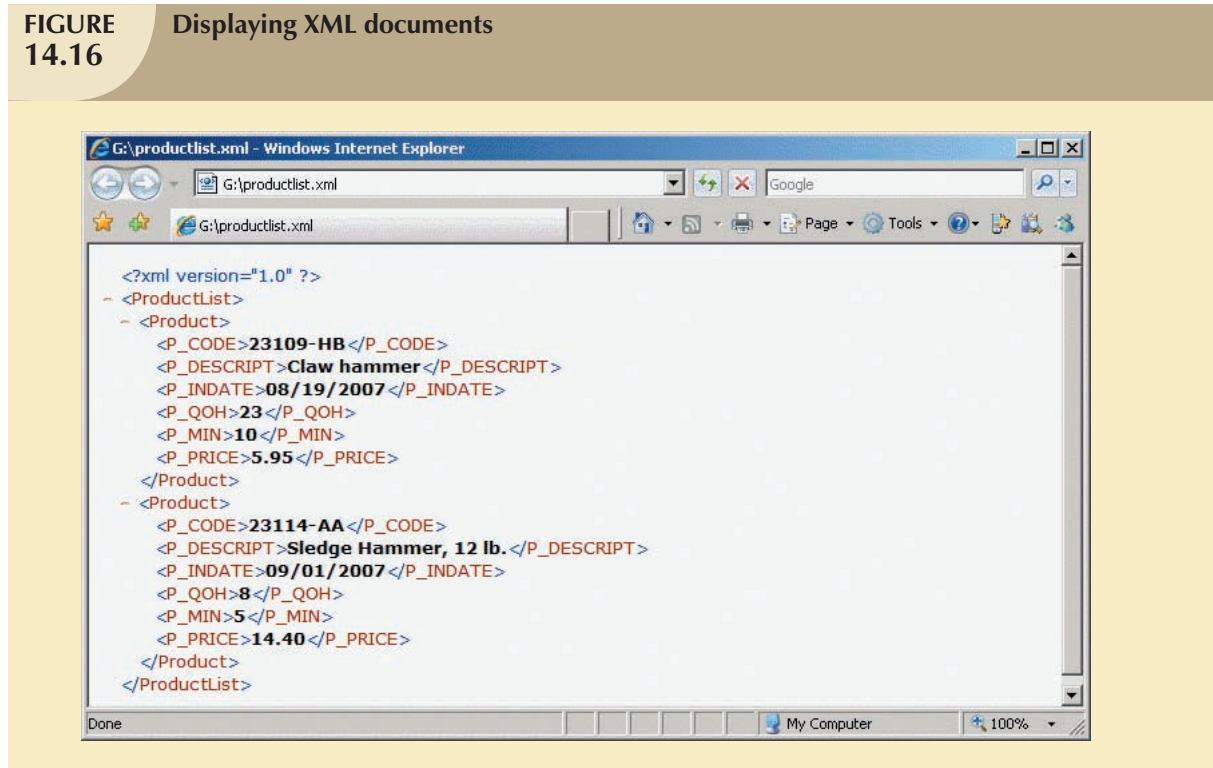
- *Extensible Style Language Transformations (XSLT)* describe the general mechanism that is used to extract and process data from one XML document and enable its transformation within another document. Using XSLT, you can extract data from an XML document and convert it into a text file, an HTML Web page, or a Web page that is formatted for a mobile device. What the user sees in those cases is actually a view (or HTML representation) of the actual XML data. XSLT can also be used to extract certain elements from an XML document, such as the product codes and product prices, to create a product catalog. XSLT can even be used to transform one XML document into another XML document.
- *XSL style sheets* define the presentation rules applied to XML elements—something like presentation templates. The XSL style sheet describes the formatting options to apply to XML elements when they are displayed on a browser, cellular phone display, PDA screen, and so on.

Figure 14.15 illustrates the framework used by the various components to translate XML documents into viewable Web pages, an XML document, or some other document.



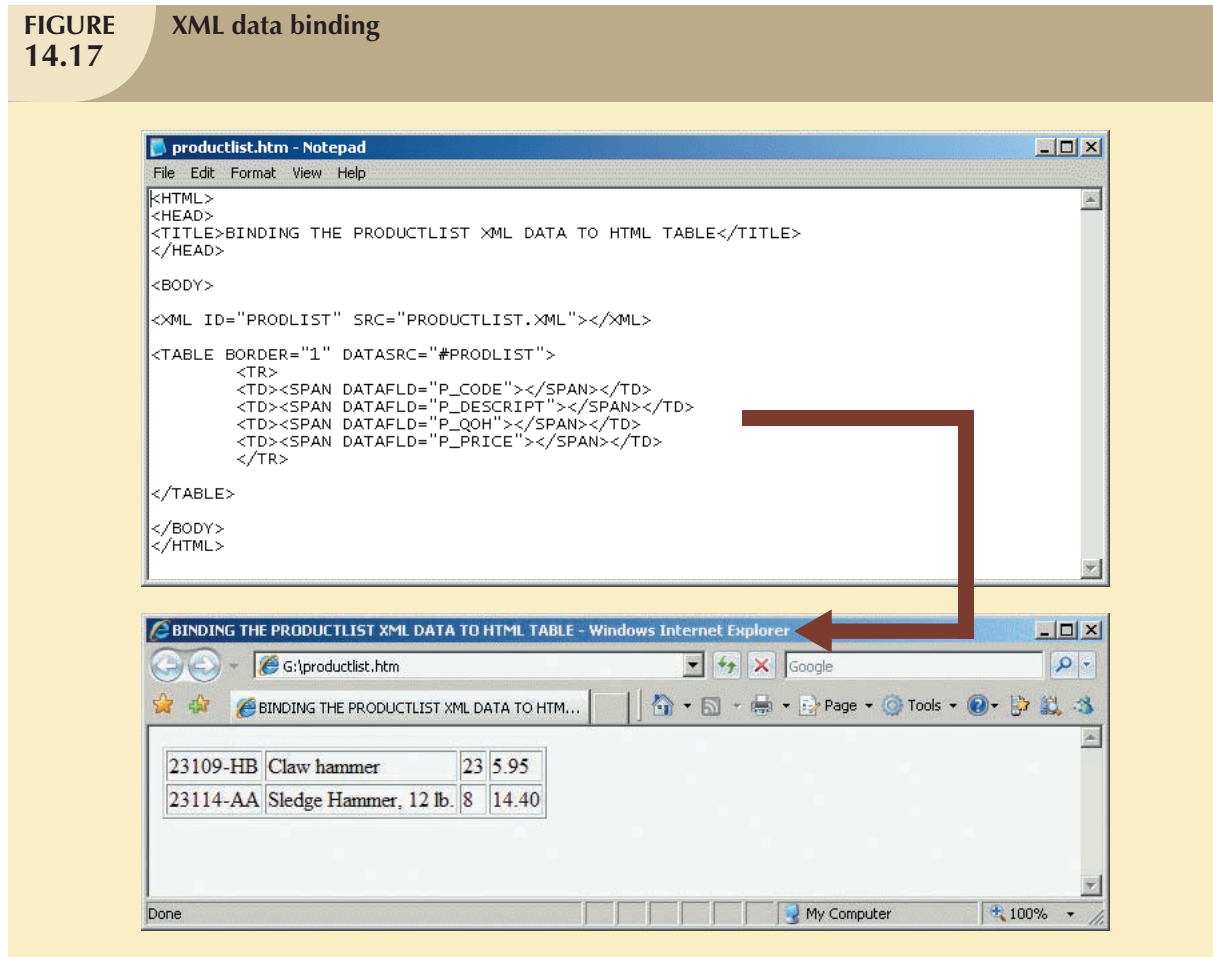
To display the XML document with Microsoft Internet Explorer (MSIE) 5.0 or later, enter the URL of the XML document in the browser's address bar. Figure 14.16 is based on the productlist.xml document created earlier. As you examine Figure 14.16, note that MSIE shows the XML data in a color-coded, collapsible, treelike structure. (Actually, this is the MSIE default style sheet that is used to render XML documents.)

FIGURE 14.16 Displaying XML documents



Internet Explorer also provides *data binding* of XML data to HTML documents. Figure 14.17 shows the HTML code that is used to bind an XML document to an HTML table. The example uses the `<xml>` tag to include the XML data in the HTML document to later bind it to the HTML table. This example works in MSIE 5.0 or later.

FIGURE 14.17 XML data binding



14.3.3 XML APPLICATIONS

Now that you have some idea what XML is, the next question is, how can you use it? What kinds of applications lend themselves particularly well to XML? This section will list some of the uses of XML. Keep in mind that the future use of XML is limited only by the imagination and creativity of the developers, designers, and programmers.

- *B2B exchanges.* As noted earlier, XML enables the exchange of B2B data, providing the standard for all organizations that need to exchange data with partners, competitors, the government, or customers. In particular, XML is positioned to replace EDI as the standard for the automation of the supply chain because it is less expensive and more flexible.
- *Legacy systems integration.* XML provides the “glue” to integrate legacy system data with modern e-commerce Web systems. Web and XML technologies could be used to inject some new life in “old but trusted” legacy applications. Another example is the use of XML to import transaction data from multiple operational databases to a data warehouse database.

- *Web page development.* XML provides several features that make it a good fit for certain Web development scenarios. For example, Web portals with large amounts of personalized data can use XML to pull data from multiple external sources (such as news, weather, and stocks) and apply different presentation rules to format pages on desktop computers as well as mobile devices.
- *Database support.* Databases are at the heart of e-commerce applications. A DBMS that supports XML exchanges will be able to integrate with external systems (Web, mobile data, legacy systems, and so on) and thus enable the creation of new types of systems. These databases can import or export data in XML format or generate XML documents from SQL queries while still storing the data, using their native data model format. Alternatively, a DBMS can also support an XML data type to store XML data in its native format. The implications of these capabilities are far-reaching—you would even be able to store a hierarchical-like tree structure inside a relational structure. Of course, such activities would also require that the query language be extended to support queries on XML data.
- *Database meta-dictionaries.* XML can also be used to create meta-dictionaries, or vocabularies, for databases. These meta-dictionaries can be used by applications that need to access other external data sources. (Until now, each time an application wanted to exchange data with another application, a new interface had to be built for that purpose.) DBMS vendors can publish meta-dictionaries to facilitate data exchanges and the creation of data views from multiple applications—hierarchical, relational, object-oriented, object-relational, or extended-relational. The meta-dictionaries would all use a common language regardless of the DBMS type. The development of industry-specific meta-dictionaries is expected. These meta-dictionaries would enable the development of complex B2B interactions, such as those likely to be found in the aviation, automotive, and pharmaceutical industries. Also likely are application-specific initiatives that would create XML meta-dictionaries for data warehousing, system management, and complex statistical applications. Even the United Nations and a not-for-profit standards-promoting organization named Oasis are working on a new specification called *ebXML* that will create a standard XML vocabulary for e-business. Other examples of meta-dictionaries are HR-XML for the human resources industry; the metadata encoding and transmission standard (METS) from the Library of Congress; the clinical accounting information (CLAIM) data exchange standard for patient data exchange in electronic medical record systems; and the extensible business reporting language (XBRL) standard for exchanging business and financial information.
- *XML databases.*² Given the huge number of expected XML-based data exchanges, businesses are already looking for ways to better manage and utilize the data. Currently, many different products are on the market to address this problem. The approaches range from simple middleware XML software, to object databases with XML interfaces, to full XML database engines and servers. The current generation of relational databases is tuned for the storage of normalized rows—that is, manipulating one row of data at a time. Because business data do not always conform to such a requirement, XML databases provide for the storage of data in complex relationships. For example, an XML database would be well suited to store the contents of a book. (The book's structure would dictate its database structure: a book typically consists of chapters, sections, paragraphs, figures, charts, footnotes, endnotes, and so on.) Examples of XML databases are Oracle, IBM DB2, MS SQL Server, Ipedo XML Database (www.ipedo.com), Tamino from Software AG (www.softwareag.com), and the open source dbXML from <http://sourceforge.net/projects/dbxml-core>.
- *XML services.* Many companies are already working on the development of a new breed of services based on XML and Web technologies. These services promise to break down the interoperability barriers among systems and companies alike. XML provides the infrastructure that facilitates heterogeneous systems to work together across the desk, the street, and the world. Services would use XML and other Internet technologies to publish their interfaces. Other services, wanting to interact with existing services, would locate them and learn their vocabulary (service request and replies) to establish a “conversation.”

² For a comprehensive analysis of XML database products, see “XML Database Products” by Ronald Bourret at www.rpbouret.com.

S U M M A R Y

- Database connectivity refers to the mechanisms through which application programs connect and communicate with data repositories. Database connectivity software is also known as *database middleware*. The data repository is also known as the *data source* because it represents the data management application (that is, an Oracle RDBMS, SQL Server DBMS, or IBM DBMS) that will be used to store the data generated by the application program.
- Microsoft database connectivity interfaces are dominant players in the market and enjoy the support of most database vendors. In fact, ODBC, OLE-DB, and ADO.NET form the backbone of Microsoft's Universal Data Access (UDA) architecture. UDA is a collection of technologies used to access any type of data source and manage any type of data, using a common interface.
- Native database connectivity refers to the connection interface that is provided by the database vendor and is unique to that vendor. Open Database Connectivity (ODBC) is Microsoft's implementation of a superset of the SQL Access Group Call Level Interface (CLI) standard for database access. ODBC is probably the most widely supported database connectivity interface. ODBC allows any Windows application to access relational data sources, using standard SQL. Data Access Objects (DAO) is an object-oriented API used to access MS Access, MS FoxPro, and dBase databases (using the Jet data engine) from Visual Basic programs. Remote Data Objects (RDO) is a higher-level object-oriented application interface used to access remote database servers. RDO uses the lower-level DAO and ODBC for direct access to databases. RDO was optimized to deal with server-based databases, such as MS SQL Server and Oracle.
- Based on Microsoft's Component Object Model (COM), Object Linking and Embedding for Database (OLE-DB) is a database middleware developed with the goal of adding object-oriented functionality for access to relational and nonrelational data. ActiveX Data Objects (ADO) provides a high-level application-oriented interface to interact with OLE-DB, DAO, and RDO. Based on ADO, ADO.NET is the data access component of Microsoft's .NET application development framework, a component-based platform for developing distributed, heterogeneous, interoperable applications aimed at manipulating any type of data over any network under any operating system and any programming language. Java Database Connectivity (JDBC) is the standard way to interface Java applications with data sources (relational, tabular, and text files).
- Database access through the Web is achieved through middleware. To improve capabilities on the client side of the Web browser, you must use plug-ins and other client-side extensions such as Java and Javascript, or ActiveX and VBScript. On the server side, Web application servers are middleware that expands the functionality of Web servers by linking them to a wide range of services, such as databases, directory systems, and search engines.
- Extensible Markup Language (XML) facilitates the exchange of B2B and other data over the Internet. XML provides the semantics that facilitates the exchange, sharing, and manipulation of structured documents across organizational boundaries. XML produces the description and the representation of data, thus setting the stage for data manipulation in ways that were not possible before XML. XML documents can be validated through the use of Document Type Definition (DTD) documents and XML Schema Definition (XSD) documents. The use of DTD, XML schemas, and XML documents permits a greater level of integration among diverse systems than was possible before this technology was made available.

KEY TERMS

ActiveX, 589	Document Type Definition (DTD), 592	plug-in, 589
ActiveX Data Objects (ADO), 578	dynamic-link libraries (DLLs), 574	Remote Data Objects (RDO), 574
ADO.NET, 580	Extensible Markup Language (XML), 591	script, 578
application programming interface (API), 573	Java, 582	server-side extension, 585
Call Level Interface (CLI), 573	JavaScript, 589	stateless system, 587
client-side extensions, 589	Java Database Connectivity (JDBC), 582	tag, 591
Common Gateway Interface (CGI), 587	Microsoft .NET framework, 580	Universal Data Access (UDA), 573
Data Access Objects (DAO), 574	Object Linking and Embedding for Database (OLE-DB), 577	VBScript, 589
database middleware, 573	Open Database Connectivity (ODBC), 573	XML schema, 595
DataSet, 580		XML schema definition (XSD), 595
data source name (DSN), 575		Web application server, 589
		Web-to-database middleware, 585



ONLINE CONTENT

Answers to selected Review Questions and Problems for this chapter are contained in the Student Online Companion for this book.

REVIEW QUESTIONS

1. Give some examples of database connectivity options and what they are used for.
2. What are ODBC, DAO, and RDO? How are they related?
3. What is the difference between DAO and RDO?
4. What are the three basic components of the ODBC architecture?
5. What steps are required to create an ODBC data source name?
6. What is OLE-DB used for, and how does it differ from ODBC?
7. Explain the OLE-DB model based on its two types of objects.
8. How does ADO complement OLE-DB?
9. What is ADO.NET, and what two new features make it important for application development?
10. What is a DataSet, and why is it considered to be disconnected?

11. What are Web server interfaces used for? Give some examples.
12. Search the Internet for Web application servers. Choose one and prepare a short presentation for your class.
13. What does this statement mean: "The Web is a stateless system." What implications does a stateless system have for database application developers?
14. What is a Web application server, and how does it work from a database perspective?
15. What are scripts, and what is their function? (Think in terms of database application development.)
16. What is XML, and why is it important?
17. What are Document Type Definition (DTD) documents, and what do they do?
18. What are XML Schema Definition (XSD) documents, and what do they do?
19. What is JDBC, and what is it used for?



ONLINE CONTENT

The databases used in the Problems for this chapter can be found in the Student Online Companion for this book.

PROBLEMS

In the following exercises, you set up database connectivity using MS Excel.

1. Use MS Excel to connect to the Ch02_InsureCo MS Access database using ODBC, and retrieve all of the AGENTS.
2. Use MS Excel to connect to the Ch02_InsureCo MS Access database using ODBC, and retrieve all of the CUSTOMERS.
3. Use MS Excel to connect to the Ch02_InsureCo MS Access database using ODBC, and retrieve the customers whose AGENT_CODE is equal to 503.
4. Create an ODBC System Data Source Name Ch02_SaleCo using the Control Panel, Administrative Tools, Data Sources (ODBC) option.
5. Use MS Excel to list all of the invoice lines for Invoice 103 using the Ch02_SaleCo System DSN.
6. Create an ODBC System Data Source Name Ch02_Tinycollege using the Control Panel, Administrative Tools, Data Sources (ODBC) option.
7. Use MS Excel to list all classes taught in room KLR200 using the Ch02_TinyCollege System DSN.

8. Create a sample XML document and DTD for the exchange of customer data.
9. Create a sample XML document and DTD for the exchange of product and pricing data.
10. Create a sample XML document and DTD for the exchange of order data.
11. Create a sample XML document and DTD for the exchange of student transcript data. Use your college transcript as a sample.

(Hint: To answer Problems 8–11, use Section 14.3.1 as your guide.)