WEB-ENABLED DATABASE CONNECTIVITY: A COMPARISON OF PROGRAMMING, SCRIPTING, AND APPLICATION-BASED ACCESS

SHAE JANSONS *has worked with information systems for 14 years, and earned her M.S. in management information systems from Governors State University in University Park, Illinois. Her research interests include Internet and E-commerce applications and programming.*
GARY J. COOK *earned his Ph.D. in management information systems at Arizona State University, Tempe, and his M.B.A. at California State University, Fresno. His research interests include E-commerce, decision support systems, and human–computer interactions.*

Shae Jansons and Gary J. Cook

Legacy databases represent a valuable repository of stored knowledge that could prove useful if made available to a broader audience via Internet applications. This article provides a framework that will assist managers in reviewing and selecting appropriate techniques for integrating their legacy database systems with Web technology, based on their particular needs, resources, and constraints. In general, for smaller organizations with lower levels of expertise, relatively few data requests, and many resource constraints, scripting can provide a method of accessing a legacy database and publishing it via a Web-enabled application in the most cost-efficient method. For larger organizations with more resources, or for applications where a number of data requests are expected, an implementation using a Web application server such as Allaire ColdFusion is recommended. For organizations with detailed, highly specialized requirements or mission-critical information, a programming approach using Java/JDBC (perhaps in conjunction with XML) should be considered.

Legacy databases represent a valuable repository of stored knowledge that could prove useful if made available to a broader audience via Internet applications. The audience might consist of customers, suppliers, employees, researchers, or governmental agencies, for example, accessing dynamic Web pages containing catalogs, inventories, research studies, or reports produced as needed from current database sources. The Internet and World Wide Web allow organizations to provide access to this legacy data while using new presentation techniques.

A Web-enabled application using client/server architecture can optimize network protocols and memory requirements.

The advantages of a Web-enabled database are many. An accessible, searchable database can provide opportunities for news, teaching, learning, research; enable online booking and catalog services; facilitate E-commerce; provide online support; improve productivity; reduce training needs; reduce costs; increase revenue; and enable information sharing among a company's employees or partners. Web-enabled database applications can be run on intranets (private internal networks), extranets (private networks that allow links to other trusted private networks, usually business partners), or the Internet, all of which use client/server architecture.

There are several factors that encourage publishing legacy database files to Web-enabled applications.

1. Web technology is widely distributed (and becoming more so every day).
2. Web technology uses an open architecture and is platform independent. Many people — some with Macs, some with Windows platforms, and others with UNIX/Linux machines — can all access the same information in a Hypertext Markup Language (HTML) page via a Web browser without difficulty.
3. Web-enabled applications are easy to use, having familiar Windows look-and-feel graphical user interfaces (GUIs).
4. From an organizational perspective, a Web-enabled application using client/server architecture can optimize network protocols and memory requirements. In client/server architecture, functionality (and therefore, workload) is divided between the Web server computer, which stores the data and Web pages, and the client computer, which requests and displays Web pages for the user.

This article provides a framework that will assist managers in reviewing and selecting appropriate techniques for integrating their legacy database systems with Web technology, based on their particular needs, resources, and constraints.

LEGACY DATABASE SYSTEMS

Legacy database systems are not constructed with new application tools and information for the express purpose of publishing information on the World Wide Web. Legacy database systems are preexisting and vary widely in form and content, including structured database files and text files (also known as "flat" files). This data, representing an organization's intellectual property, might be stored on word processors, spreadsheets, and databases in a variety of software- and hardware-dependent platforms. Some data is stored in formats that are obsolete, or stored in a proprietary format no longer available to the organization. For example, a shift in hardware capabilities can render data inaccessible; a database built on a UNIX system may not be easily converted to a Windows Web environment, or vice versa.

The dilemma of accessing legacy systems in a Web-enabled environment is a widespread phenomenon. Legacy database systems are held by myriad organizations, varying from small mom-and-pop retailers and not-for-profit groups to large corporations. Each type of organization has different information needs and varying expenditure levels for information management. Transforming legacy files into data sources for Web-enabled applications has quite often been difficult. Organizations must consider several criteria when deciding if and when they should integrate their legacy database system with Web-enabled technology.

A major consideration is the type of systems (hardware and software) they already possess. One recommended planning process for implementing database–Web integration follows six steps:

1. Document the current computing environment, including hardware and software, Internet connectivity, and in-house expertise.
2. Develop a gap analysis (existing versus desired future environment) and quantify costs/benefits.
3. Refine and compare alternatives.
4. Consolidate security issues.
5. Evaluate Web application tools.
6. Design, implement, test, and evaluate a pilot system.

This article focuses on the fifth step, that of evaluating different methods of developing the database–Web application, comparing techniques based on several criteria:

❑ Relative costs
❑ Ease of implementation
❑ Portability
❑ Technology support and longevity
❑ Security
❑ Maintenance/administration issues

ODBC is somewhat limited in its flexibility, possessing only a subset of the functionality available in native database applications.

OPEN DATABASE CONNECTIVITY (ODBC)

To successfully integrate a legacy database and Web-enabled application, a method of reliably accessing and communicating with the database is necessary. The most common technique used to access database systems is Open Database Connectivity (ODBC), which is the current industry standard. ODBC provides connectivity between two different computer applications, for example, database and Web server software. ODBC acts as a translator to process requests into vendor-specific code and responses into user-friendly data. Developed by Microsoft in 1992, ODBC is a platform- and database-neutral standard and offers developers multi-database application programming interfaces (APIs). APIs make it possible to access multiple databases using the same code, minimizing time and effort and ensuring compatibility.

Because of its many advantages, ODBC has proven quite popular; the vast majority of database vendors include ODBC drivers with their products. ODBC's database neutrality, multiple platform support, and adoption as a standard make it appealing. Prior to ODBC, developers had to use "native" database methods, which are proprietary to a specific vendor and always differ from methods used by competing vendors; these native APIs are often written in C/C++ and require programming skills to extend. ODBC allows flexibility and makes it possible for users to replace one database product with another without extensive coding. ODBC support has become an important key to providing database–Web integration in database software.

ODBC must be installed on a computer prior to its use. PC users running any Windows environment will already have ODBC, which is standard in the Windows operating system. ODBC includes two items: an ODBC Administrator and a list of database drivers, each specific to a different database program. When a DBMS (database management system) is installed, an included ODBC driver will install automatically and will appear in the driver list of the ODBC Administrator. IBM, Oracle, Sybase, Informix, Microsoft, Information Builders, Cross Access, Intersolv, and Visigenic are among the major database software vendors that ship ODBC drivers with their products.

ODBC enables database-application connectivity by building a data source (called a Data Source Name, or DSN), which is an abstraction of the database that includes the database server, name, schema, network library, and any other critical information. The DSN also defines the interface the application must use to send requests and receive information through the connection; these definitions are common to all vendors. Before a database using ODBC can be accessed, the DSN must be created using the ODBC Administrator. Once a database connection is established, requests can be passed to the database via the ODBC connection using Structured Query Language (SQL), which is an accepted standard for querying databases. After processing the request, the database will return a record set of requested information, which can then be processed for presentation on an HTML page.

Despite its many advantages, ODBC has some drawbacks. Using ODBC to access a database can increase processing overhead up to 15 to 50 percent from using the database's native methods; and ODBC is somewhat limited in its flexibility, possessing only a subset of the functionality available in native database applications. Because of the need to meet many software requirements, ODBC represents a lowest-common-denominator approach to database connectivity. While ODBC is easily configured for a few users, setting up a large number of client computers can be tedious for the network administrator. Because of these drawbacks, built-in support for programming languages such as Perl, Python, and PHP, as well as C/C++ libraries, is becoming increasingly common in the major database products.

IMPLEMENTING A DATABASE CONNECTION

Accessing data is a critical part of publishing a legacy database, but implementing the connection and providing a user interface and method of data manipulation and presentation are also essential. There are numerous techniques to accomplish this, and these vary widely in cost, level of expertise required, and ease of implementation and maintenance. In general, most methods fall into one of three broad (and sometimes overlapping) categories:

- ❑ Programming language-based access
- ❑ Script language-based access
- ❑ Application-based access

Programming language-based methods of database access are not new. Programming languages such as C, C++, Pascal, COBOL, and many other languages can perform database operations via native or ODBC connections. Most programming languages, however, do not integrate well with a Web environment. One newer programming language — Java — is especially well suited to the Web because of its cross-platform capabilities, open architecture, and ability to make applets that display in a Web browser.

Scripting languages provide a second means to access database files, traditionally through common gateway interface (CGI) scripts. This method is well documented and widely used. A variety of languages can be used for creating scripts, including VBScript, JavaScript (also known as JScript or EMCAScript-262), Perl, PHP, and Python.

The third method of database access is accomplished through software applications. These programs, sometimes referred to as "middleware," are available from a number of software vendors. Each application has advantages and disadvantages, and range widely in cost, ease of use, and portability. For this study, Allaire ColdFusion was chosen. ColdFusion has become popular for its rapid application development (RAD) capabilities. Its strength — and a major reason for its popularity — lies in its ability to access and manipulate data in a database for quick and easy presentation via the Web. It produces dynamic HTML pages based on database information using a combination of ODBC connections and vendor-specific coding.

JAVA AND JAVA DATABASE CONNECTIVITY

Java is a relatively new programming language, developed by Sun Microsystems. Java has become popular because it is an open standard, and can, with installation of the Java Virtual Machine (JVM) or Java Runtime Environment (JRE), be run on practically any computer operating system. JVM and JRE are both available free from Sun Microsystems. In addition, most operating system developers are building support for Java into their systems.

Java can be used to develop servlets, which function much like CGI scripts, and applets, which are programs that run within the confines of a client's Web browser. Servlets can be written in Java using the HTTPServlet class, which supports the "PUT" and "GET" functions and other hypertext transfer protocol (HTTP) standards. Because the Internet currently operates using HTTP standards, Java is well suited to be used via Internet connections. In combination with the eXtensible Markup Language (XML), servlets can be a powerful tool to deliver structured data to clients. Applets usually reside on a server and are downloaded to a client upon demand; they are only able to connect to the server from which they were downloaded, thus increasing server security. Applets also have an advantage in that no software needs to be pre-installed on client machines. Additionally, applets can execute on any hardware supporting a Web browser and, therefore, are platform independent. A disadvantage of applets is their built-in security feature. Applets cannot call ODBC or native APIs directly; they must use a Java API to access any database.

The original release of the Java language did not support database connectivity, but incorporated this feature by 1996 with the inclusion of the Java Database Connectivity (JDBC) API. The JDBC standard was formulated with the input of major database software vendors, including Sybase, Oracle, Informix, Symantec, and Intersolv. JDBC operation and functionality are based on ODBC structure and performance.

As of late 2000, there were nearly 150 available JDBC drivers, produced by numerous vendors, including IBM, Sybase, Oracle, Compaq, Netscape, Lotus Development, Borland, Open Link, and Visigenic. JDBC drivers are available from Sun Microsystems, database vendors, third-party software developers, and middleware developers.

JDBC, similar to ODBC, provides connectivity between two different computer applications: a Java program (program or applet) and a database. In essence, JDBC is to Java what ODBC is to other programming and scripting languages. Like ODBC, JDBC uses a DSN to represent a connection to a database. JDBC is also "Web aware," using a URL to express the type of connection, driver, host, port number, and DSN.

There are actually four different types of JDBC drivers. They differ in how much Java code each contains, the type of protocol each uses to communicate with the database, and the type of computing environment in which they can be implemented. JDBC drivers are either two-tier or three-tier design. A two-tier design is one in which the client and database communicate directly; a three-tier design includes an intermediate programming layer that acts as liaison between client and database. Two-tier drivers tend to be faster and perform better in high-volume interactions.

Type 1 drivers are available for free download with the Java Development Kit (JDK) from Sun Microsystems, as well as from a few other vendors. They take advantage of the widespread use and availability of ODBC and can easily access databases from several vendors. Type 1 drivers provide a three-tier approach by enabling a Java program to connect to a database through an ODBC driver. This approach, however, results in relatively poor performance. In addition, drivers must be pre-installed on clients, and just-in-time download of applets is not supported. Because of these limitations, Type 1 drivers are not widely used.

Type 2 drivers employ a two-tier approach. Many database vendors, among them IBM and Oracle, ship Type 2 drivers with their products. By connecting directly to the database's native API, the Type 2 driver is able to take advantage of the richer, more flexible native methods of querying and manipulating data. Type 2 drivers provide better performance than Type 1 drivers and are especially suited to intranets where there is high administrative control over client computers. A disadvantage is that the drivers/APIs must be installed on each client.

Type 3 drivers are best in networks that connect multiple Java-based clients to many different databases (i.e., the Internet). Type 3 drivers reside on the server, allowing Java-based clients to read/write to databases located anywhere on the network. They use a three-tier approach, in which the driver translates JDBC calls into a database-independent network protocol that is then translated into a database protocol by the server. Because Type 3 drivers allow for automatic downloading of applets, they are ideal for Internet/intranet-based, multi-user, data-intensive applications.

Type 4 drivers function much like Type 2 drivers, but allow automatic applet downloading. These drivers are especially practical for Web-enabled applications because of this capability. They exhibit high performance standards, with quick connection times and information retrieval; they also, however, tend to be much larger files. Because they depend on the database vendors' proprietary database access protocols, Type 4 drivers are usually only available from the vendors and are typically more costly than other JDBC driver types. Because proprietary database APIs must be loaded on the client to use Type 4 drivers, this driver type is best suited for an intranet environment where there is administrative control of client machines.

Because Java is platform independent, its implementation does not rely on a specific type of hardware. This can be beneficial to organizations that already have a variety of in-house computer hardware. However, lower hardware costs may be offset by higher programming costs. Java/JDBC requires the skill of Java programmers. An organization desiring to implement a Java-based Web-enabled application must have the resources to acquire programmers and appropriate JDBC drivers.

SCRIPTING LANGUAGES AND DATABASE ACCESS

CGI scripts are the traditional method used to access and deliver data to Web sites. Scripts can be written in a variety of languages, both programming and scripting. CGI scripts reside on the server and enable building Web pages that contain forms, which include variables (fields, radio buttons, check boxes, etc.) whose values can be manipulated by the client. When the user submits the form, the CGI program on the server receives the values of the variables, processes them, and returns a new HTML page to the client with new data.

This type of delivery entails high overhead because the server must process each new request independently. The CGI script must perform several functions (steps) to process one database query, including:

1. Read the query parameters from the form submitted by the client.
2. Connect to the database, construct a SQL query, and send it to the database.
3. Receive the results of the query from the database and format them into HTML.
4. Send the HTML page back to the client for display on the browser.

Standard Web protocols are used for Steps 1 and 4; Steps 2 and 3 must be coded within the CGI script, or must employ another program for processing.

The most widely used scripting language to accomplish Steps 2 and 3 is Perl, although Python is gaining rapidly in popularity. Perl is a powerful scripting language that reliably works on a variety of platforms. It is a free, public-use language. Although it is possible to compile Perl programs (running them locally with Java-like "virtual machines"), Perl is most often used as interpreted language. This requires that the Perl engine be installed on the server in order for scripts to execute. The Perl engine works on nearly every operating system, including all common versions of UNIX, 16- and 32-bit Windows, MacOS, and IBM AS400. There are no special configurations needed to run Perl, and scripts can be created in a simple text editor such as Notepad. Perl technical support can be purchased, but there are also numerous autonomous online Perl documentation sites, independently written books, and user groups.

Perl has many ardent supporters, but detractors maintain that CGI and scripting languages such as Perl are too slow, inflexible, and insecure for today's Internet applications.

Perl was designed as a multi-purpose language and originally did not include database access APIs. However, Perl developers have written a Perl Database Interface (DBI) module to access and query databases. The Perl DBI does not communicate directly with a database; instead, it locates and loads the appropriate native database driver on the local machine. The driver communicates with the database, and the DBI communicates with the driver on behalf of the Perl program.

Prior to DBI, Perl was predominantly used to access data located in flat files; this remains a common implementation. This technique presents limitations; complex queries involving JOIN statements (which join information in two separate files) or complex SELECT statements are difficult to implement.

There are also specialized versions of Perl, such as Sybperl and Oraperl, which are basically Perl implementations of the Sybase and Oracle API libraries, respectively. Perl's major shortcoming is that the client has no easy way of interacting with data; once the query is made, data returned and displayed, there is no way to further query or manipulate data. To perform any other data handling, another CGI script must be executed on the server.

CGI scripts using Perl are relatively easy to implement and can provide cost-effective solutions for database–Web connectivity projects. However, they also have clear disadvantages. A critical issue is server performance, which degrades quickly if many requests are made on the system. Security is also a major concern. CGI scripts operate on the server at the request of a remote client; scripts may contain bugs that can compromise server security in one of four ways:

1. Allowing private or confidential information stored on the server to be accessed
2. Allowing private or confidential information being sent to the server to be intercepted
3. Allowing information about the host server to be accessed, possibly allowing unauthorized users into system files
4. Allowing unauthorized users to execute commands on the server that modify or damage system files

Due to degradation of performance and security issues, CGI scripts should rarely be used without other tools. Perl has many ardent supporters, but detractors maintain that CGI and scripting languages such as Perl are too slow, inflexible, and insecure for today's Internet applications.

Newer approaches that attempt to address these issues include FastCGI, mod_perl, and PHP. FastCGI runs one or more instances of Perl continuously, thus avoiding the biggest drawback of CGI: creating a new process for each script request. mod_perl addresses this issue by embedding the Perl interpreter inside the Web server. PHP takes a similar approach by embedding the interpreter inside the Web server. PHP goes a step further by using code that is embedded in HTML files.

Microsoft released Active Server Pages (ASP) in 1996 as another alternative to CGI/Perl. ASP can improve server performance while still allowing developers to use scripting for database connectivity. ASP offers the advantage of being tightly integrated with the Web server software, increasing performance over traditional Perl/CGI scripting. It has worked well with Rapid Application Development (RAD) tools, such as Visual Basic, supports user-generated database queries, and can execute business logic on the server side.

ASP was available only on Windows NT operating systems, however, until third-party vendor ChiliSoft developed a cross-platform ASP application in 1997. ChiliSoft ASP works with many operating systems, including most varieties of UNIX, and works with major Web server software packages, including Netscape Enterprise and FastTrack, Apache, and O'Reilly Web Server.

ChiliSoft ASP can be used with a variety of development applications, including Microsoft's InterDev and FrontPage 2000, Macromedia's Drumbeat and Dreamweaver, NetObject's Fusion and ScriptBuilder, and Allaire's HomeSite. It uses ODBC connections and ships with a variety of drivers, including Oracle, Sybase, Informix, DB2, and dBase.

ChiliSoft ASP allows a choice of scripting languages (including VBScript, JScript, or Perl, although VBScript is the most popular for this application) or programming languages (including Visual Basic, Pascal via Delphi, C, C++, Java, and COBOL). However, the manufacturer suggests that for maximum portability, developers must use C++ or Java. The need for skilled programmers, as well as the price tag, make ChiliSoft ASP a costly alternative to Perl/CGI scripting for small to mid-sized businesses.

MIDDLEWARE APPLICATIONS: COLDFUSION

A middleware program is one that allows two other applications to transparently collaborate across processes and networks. In Web-enabled applications, a middleware program operates between the server and client. This specialized middleware is called Web application server software. There are numerous Web application server products currently on the market; Allaire's ColdFusion is one such product.

There are some important performance considerations that should be addressed when choosing middleware. One critical consideration is expansion; a database access middleware must address scalability. Middleware must also be assessed in terms of processing overhead. Two measures for assessing middleware performance are *message latency* (processing time delay in milliseconds) and *data throughput* (number of bytes a user application can transmit or receive in a unit of time using a reliable protocol). These performance criteria become more critical as the requests on the system grow and should be considered from the early planning stages. One of the reasons ColdFusion has become so popular is its above-average performance in these critical areas. In addition, it permits very high security and integrates Java and XML.

ColdFusion is powerful and stable software that is both user- and developer-friendly. ColdFusion performs multiple functions, including security, user authentication, language support (including Java, JavaScript, VBScript, and C++), server fault tolerance, load balancing, query caching, connection pooling, file transfers, and mail and directory services, in addition to database access. ColdFusion supports ODBC; and although it supports Java programming, it does not implement JDBC. ColdFusion uses a Web-based module, called the ColdFusion Administrator, to configure, manage, and maintain the ColdFusion Server program that processes data requests. ColdFusion must be installed on the same computer as the Web server software and must be configured to operate with the server software.

ColdFusion functionality is implemented through the use of ColdFusion Markup Language (CFML) tags that are embedded in an HTML page. CFML is a proprietary language that looks much like HTML. Like HTML, CFML can be produced with a simple text editor, or one can use ColdFusion Studio (an integrated development environment (IDE)) to build ColdFusion Web sites. ColdFusion Studio comes bundled with the single-user ColdFusion Professional program; because it is tightly integrated with the ColdFusion Server, ColdFusion Studio enables rapid development of sites, having built-in support for all ColdFusion functions.

ColdFusion Server interprets CFML tags, opens the database, sends an appropriate request based on client input, receives and processes information from the database, and sends the processed HTML page to the Web server for presentation to the client. ColdFusion uses "templates" written in HTML and CFML; data is added to the templates dynamically, based on parameters input by the client.

A major advantage of ColdFusion is that it supports drill-down capabilities; that is, the ability to click on data to reveal more detailed information without sending information back to the server for reprocessing. Most Web applications (such as Perl scripts) do not support this functionality; each request for information must be sent to the server and processed individually. Drill-down capability is provided in ColdFusion through the use of multiple template pages and query caching. ColdFusion also implements data storage techniques, much as programming or scripting languages do. Advanced data types supported by ColdFusion include lists (similar items separated by a common delimiter), arrays (related information stored in a one-, two-, or three-dimensional grid), and structures (key-value pairs in a grouped set). ColdFusion also offers form validation, debugging and troubleshooting assistance, report writing capabilities, client session management, and automatic e-mail response.

The major disadvantage of implementing ColdFusion is the relatively high cost of software. A limited version of ColdFusion Server, — ColdFusion Express — is available for free download, but it has reduced functionality and is only appropriate for small, personal applications.

For the greatest flexibility, security, and scalability, some organizations should consider a combination of Web server application software and Java.

The learning curve for ColdFusion is moderately low for simple applications, but more advanced uses require skilled technical staff to implement and maintain it, resulting in higher personnel and operating costs. In addition, ColdFusion operates only with major Web server software packages (e.g., Apache, Netscape's FastTrack and Enterprise servers, Microsoft IIS and Personal Web Server, O'Reilly Web Site Pro). Other Web servers can be used, but these are supported through a CGI script, not a native API, making performance slower.

CONCLUSIONS AND RECOMMENDATIONS

Implementing Web–database connectivity with legacy database systems can provide a rich source of information for broad audiences. It can also, however, be a complicated, expensive proposition. There are many software products and programming techniques that can be employed to access legacy systems and process the data for Web publication. The exact implementation an organization chooses will depend on many factors; existing hardware/software, level of expertise, expected use, and resource constraints are some primary considerations in any implementation.

For smaller organizations with low levels of expertise, relatively few data requests, and many resource constraints, scripting can provide a method of accessing a legacy database and publishing it via a Web-enabled application in the most cost-efficient method. While a "pure" scripting environment is probably not the best implementation for all but the smallest, most secure intranet, it can provide connectivity for an organization that does not have resources to implement other, more expensive techniques.

For larger organizations with more resources, or for applications that expect a number of data requests, an implementation using a Web application server such as Allaire's ColdFusion is highly recommended. ColdFusion can provide a highly scalable application that can grow as the organization grows. Although it entails an initial investment in software, in the long-term, this solution will provide the most cost-effective, reliable, and secure method of data access and presentation.

For larger organizations, or those with detailed, highly specialized requirements or mission-critical information, Java/JDBC (perhaps in conjunction with XML) should be considered. This is especially true for businesses involved in extranets where data exchange is a critical business function. For the greatest flexibility, security, and scalability, some organizations should consider a combination of Web server application software and Java. ColdFusion, which supports Java and XML, could be an ideal, cost-effective solution for long-term database–Web connectivity implementation. ▲

Bibliography

1. The Allaire Corporation Web Site (www.allaire.com).
2. Andreessen, M. Database Connectivity, *Netscape Communications Corporation*, 1996. Retrieved Nov. 6, 2000, from home.netscape.com/columns/techvision/databases.htm.
3. Bidgoli, H. "An Integrated Model for Introducing Intranets." *Information Systems Management*, Summer 1999, 78-87.
4. Chen, J.Q. and Heath, R.D. "Building Web Applications," *Information Systems Management*, Winter 2001, 68-79.
5. "ChiliSoft Adds AIX, Apache," *eWeek*, April 26, 1999. Retrieved March 5, 2001, from www.zdnet.com/eweek.
6. ChiliSoft ASP. AdRite Business Support. Retrieved March 5, 2001, from www.adrite.com/script7a.htm.
7. "Chilisoft ASP Development Tools." *ChiliSoft White Articles*. Retrieved March 5, 2001, from www.chilisoft.com/whitepeppers/default.asp.
8. The ChiliSoft Web Site (www.chilisoft.com).
9. Cook, R. "Web Building to Transaction Processing Crescendo," *NetScape World*, Sept. 1, 1996. Retrieved Oct. 14, 2000, from www.netscapeworld.com/netscapeworld/nw-09-1996/nw-09-dbms.html.
10. Cooke, K. "An Introduction to ASP," *WebMonkey*. Retrieved March 5, 2001, from www.hotwired.lycos.com/Webmonkey/98/39/index2a.html?tw=programming.
11. Cox, T. "Using Perl with Databases," *Byte*, 15(5), 57-58, 1998.
12. DeJesus, E.X. "The Middleware Riddle," *Byte*, 65-70, 1996.
13. Dominus, M.J. *Short Guide to DBI (The Perl Database Module)*, O'Reilly & Associates, 1999. Retrieved from www.perl.com/pub/1999/10/DBI.html.
14. Forta, B. *The Allaire ColdFusion Web Application Construction Kit*, 3rd ed., Indianapolis, IN: Que Books, 1998.
15. Friesen, N. *Publishing Your Database on the Web,* University of Alberta, Canada, Academic Technologies for Learning. Retrieved Nov. 4, 2000, from www.atl.ualberta.ca/articles/Web/database.cfm.
16. Guan, Hi., Ip, H., and Zhang, Y. "Java-based Approaches for Accessing Databases on the Internet and a JDBC-ODBC Implementation," *Computing & Control Engineering Journal*, 9(2), 71-78, 1998.
17. Guelich, S., Gundavaram, S., and Birznieks, *CGI Programming with Perl*, Cambridge, MA: O'Reilly & Associates, 2000

18. Hauf, B. "An Evaluation of Three Approaches to WWW Application Development," *EarthWeb*, 1999. Retrieved Sept. 24, 2000, from www.earthWeb.com/earthWeb.
19. Hightower, L. "Publishing Dynamic Data on the Internet," *Dr. Dobb's Journa*l, 22, 70-72, 1997.
20. Hines, J.R. "Fresh from Your Database to the Web," *IEEE Spectrum*, 34, 20-21, 1997.
21. "Hosting Database Driven ASP Web Sites Using ChiliSoft ASP," *ChiliSoft White Articles*. Retrieved March 5, 2001, from www.chilisoft.com/whitepeppers/default.asp.
22. Houstis, E.N., Rice, J.R., Gallopoulos, E., and Bramley, R. (Eds.). *Enabling Technologies for Computational Science: Frameworks, Middleware and Environments*, Norwell, MA: Kluwer Academic Publishers, 2000.
23. Joch, A. "JDBC's Growing Pains," *Byte*, 15(5), 112M-112N, 1999.
24. Kroenke, D.M. *Database Processing: Fundamental, Design and Implementation,* 7th ed., Upper Saddle River, NJ: Prentice Hall, 1999.
25. Lazar, P.Z. and Holfelder, P. "Web Database Connectivity with Scripting Languages," *Web Journal*, Vol. 2(2). Retrieved Nov. 14, 2000, from www.ora.com/catalog/wj6/excerpt/index.html.
26. Linthicum, D.S. "Hooking Your Site up to a Database," *Computer Shopper Magazine*. Retrieved Oct. 31, 2000, from www.zdnet.com/.
27. MacNeil, B. "Revamp Your Web Site to Improve Its Return on Investment," WWWiz Corporation. Retrieved Oct. 31, 2000, from wwwiz.com/issueRabinovitch/html/article4.html.
28. Matthews, M.S. and Poulsen, E.B. *FrontPage 2000: The Complete Reference*, Berkeley, CA: Osborne/McGraw-Hill, Publishers, 1999.
29. Microsoft Active Server Pages (ASP), Ace Computer. Retrieved March 5, 2001, from www.acecomputer.com/asppages/mainasp.asp.
30. Mitchell, I. "Opening the World's Intellectual Capital," *Chicago Tribune*, August 6, 1999. Retrieved Oct. 24, 2000, from chicagotribune.com/tech/specialreport/article/0,2669,2-32592,FF.html.
31. Nance, B. "Database Access via ODBC and JDBC," *Network Computing*, 2000. Retrieved Sept. 21, 2000, from www.networkcomputing.com/netdesign/odbc1.html, odbc2.html, odbc3.html, odbc4.html, odbc5.html, odbc6.html.
32. North, K. "Understanding Multidatabase APIs and ODBC," *DMS Online*, 1994. Retrieved Oct. 26, 2000, from www.dbmsmag.com/.
33. Ptak, R.L. "Designing a Business Justified Intranet Project," *Information Systems Management*, Spring 1998, 13-19.
34. The Perl Website, O'Reilly & Associates (www.perl.com).
35. Rabinovitch, E. "Presentation Methods: Web-to-Host Integration," *IEEE Communications Magazine*, 35, 90,1997,.
36. Railsback, K. "Linux: It Isn't Just for Web Servers Anymore," *Linux Magazine*, June 2001, 40-48.
37. Rob, P. and Coronel, C. *Database Systems: Design, Implementation and Management*, 4th ed., Cambridge, MA: Thompson Learning, 1999.
38. Rosenblatt, B. "Tools to Meld the Web & Relational Databases, Parts 1 & 2." Retrieved Nov. 6, 2000, from www.sunworld.com/swol-01-1996/swol-01-cs.html, swol-2-cs.html.
39. Ruh, W., Herron, T., and Klinker, P. *IIOP Complete: Understanding CORBA and Middleware Interoperability,* Reading, MA: Addison-Wesley Longman, Inc., 2000.
40. Rymer, J. "The Muddle in the Middle," *Byte*, April 1996, 67-70.
41. Sood, M. "Examining JDBC Drivers," *Dr. Dobb's Journal*, 15(2), 82,1998.
42. Sood, M. "JDBC Drivers and Web Security," *Dr. Dobb's Journal*, 15(7), 90,1999.
43. Staunton-Lambert, K.J. "An Investigation into the Interconnectivity of Internet and Database Technologies," 1998. Retrieved Nov. 6, 2000, from www.hipstream.force9.co.uk/dissertation/week5/ODBC.html.
44. Stien, L.D. "The World Wide Web Security FAQ," 1997. Retrieved Dec. 2, 2000, from www.perl.com/pub/doc/FAQs/cig/wwwsf1.html#Q1.
45. Thomas, W. "No Brainer Database Publishing, Parts 1-6," *Web Developer*, 1997. Retrieved Oct. 14, 2000, from www.Webdeveloper.com/database/ db_no_brainer_db_pub_1.html, db_no_brainer_db_pub_2.html, bdb_no_brainer_db_pub_6.html.
46. Tidwell, D. "Servlets and XML: Made for Each Other," *DeveloperWorks*, 2000. Retrieved Nov. 16, 2000 from www-4.ibm.com/software/developer/library/servelets-and-xml/index.html.
47. Vaughan-Nichols, S.J. "The DBMS/Web Connection," Sm@rtPartner. Retrieved Oct. 31, 2000, from www.zdnet.com/sp/stories/issue/0,4537,2157028,00.html.
48. "Why Active Server Pages," *ChiliSoft White Articles*. Retrieved March 5, 2001, from www.chilisoft.com/whitepeppers/default.asp.