Integrating the World Wide Web and Database Technology

John K. Whetzel

The recent popularity of the World Wide Web (WWW, or the Web) has created a massive increase in both the supply and demand of Web-based technologies. However, the HyperText Markup Language (HTML) used to construct the Web has limitations that challenge information content providers who want to supply current, up-to-date information with minimal administrative overhead. A powerful, extensible solution to many of these challenges is the use of a database as a *back end*, or data source, for Web applications. Combining the Web with a database maximizes the strengths of its components. From the Web perspective, this combination offers user friendliness, cross-platform compatibility, and high-speed prototyping capabilities. From the database perspective, it offers relational data manipulation, high-speed search capabilities, and industrial-grade data input and retrieval. This paper describes experiences of application developers working at NCR, formerly AT&T Global Information Solutions. It also analyzes the strengths and weaknesses of the Web/database combination and seeks to prove that this combination is a viable alternative for providing database-oriented solutions.

Introduction

The recent advent of World Wide Web (WWW, or Web) technology has revolutionized the way companies provide information to their customers and employees. The ubiquitous nature of Web browsers and sites has caused information providers to scramble to the Internet in an attempt to catch up with the rising trend. Despite the advantages of this new medium, a brief exposure to working with the Web technologies will reveal some of its shortcomings. One way to overcome these is to link the information presentation capabilities of the Web with the power of a relational database. This paper examines four areas of combined strength for Web/database solutions:

- Basic concepts and definitions associated with Web/database connectivity and the links between them,
- Techniques used to integrate the Web with a relational database,
- The advantages and disadvantages of Web/database integration, and
- Future directions of Web/database technology and their possible impacts.

Concepts and Definitions

To understand Web/database interaction, it is important to first understand the key concepts that form the basis for disseminating information on the Web. Although these concepts are somewhat simplistic, they help set the stage for the discussion that follows.

The World Wide Web. The Web project, proposed by Tim Berniers-Lee at CERN, the European Laboratory for Particle Physics, was initiated to construct a distributed hypermedia

system. The Web model consists of interconnected machines serving information to clients. The project defined both the communication used between the clients and servers, called HyperText Transfer Protocol (HTTP), and the format of the text documents that were transferred, called HyperText Markup Language (HTML). Another key aspect of the Web definition was its ability to specify addresses, known as uniform resource locators (URLs), for particular objects within the system. When this concept for a distributed hypermedia system was combined with the world's largest wide area network, the Internet, the popular conception of the Web was born. The Web has expanded so rapidly that machines serving such information span the globe and their traffic regularly counts as some of the most voluminous on the Internet.

HyperText Transfer Protocol. HTTP is the protocol "spoken" by Web servers. Client programs that can speak HTTP, colloquially known as *browsers*, are used by people on the Internet to connect to HTTP servers and to download information. Dozens of browsers exist, and most hardware and software platforms are supported. HTTP is stateless, with each transaction requiring a new connection from client to server. As with other parts of the Web, there are several ongoing proposals for its improvement.

HyperText Markup Language. HTML is the formatting language used with the Web. As a subset, or a document-type definition (DTD), of the broader Standard Generalized Markup Language (SGML), it defines how authors must format the information they present on the Web. HTML is tag-based, using specific tags to define the format of words and phrases within a document. For example, if an author wants to designate the phrase "Object-Oriented Design" as the title of an HTML document, he or she must enclose it within title tags:

<TITLE>Object-Oriented Design</TITLE>.

Both HTML and SGML only specify how a document is structured, not how it is presented. The browser programs themselves have the final say on how particular tags are shown to the user. For example, in most, but not all, graphical browsers, the TITLE tag used earlier displays the given title in the top window decoration on the browser. This lack of consistency can lead to frustration for authors who prepare their documentation on one type of browser and expect it to be presented the same way on all browsers.

Panel 1. Abbreviations, Acronyms, and Terms

CGI—common gateway interface
DSS—decision support system
GUI—graphical user interface
HTML—HyperText Markup Language
HTTP—HyperText Transfer Protocol
hypertext—words or phrases within a document that represent links to other information
ODBC—open database connectivity
SGML—Standard Generalized Markup Language
SQL—Structured Query Language
URL—uniform resource locator
WWW—World Wide Web

HTML has two key strengths. As its name indicates, HTML supports *hypertext*, words or phrases within a particular document that can be linked to another document. An example of this would be a hypertext document in which the word "semiconductor" was highlighted, underlined, or otherwise emphasized. When the user clicks on this word, the browser loads an entirely new document that, ideally, presents more information about semiconductors. When the wide-area distributed concept of the Internet is combined with this functionality, multiple documents spanning any number of geographically dispersed sites can be interlinked. HTML's other main strength is its ability to include hypermedia objects such as images, sounds, and video within a document. Users can view pictures that are placed *in-line*, or within the text, to illustrate important concepts, or they can click on a particular word to download a related video. This was an extremely important development in the text-intensive arena of the Internet. Seamless integration of documentation and hypermedia capabilities have helped to make HTML, and the Web in general, much easier for the average person to use.

Common Gateway Interface. Although standard HTML is a powerful way to present information, it has several limitations, a severe one of which is its static nature. Because HTML documents are text files, they need to be continually updated to incorporate new or changed information. Authors quickly realized that they

needed a way to keep dynamically changing information current. From this need grew the concept of a common gateway interface (CGI).

A CGI program redirects its output to an HTTP client; in other words, it dynamically generates HTML code. Such a program might be a script that lists the users logged on to a particular server. A CGI program enables visitors to an author's Web site to click on a particular hypertext link and see a list of all the users currently logged on to the system. Rather than designating a hypertext link to pull back an HTML file, the author can point the link at a script that checks the system and prints out a list of all on-line users. The HTTP server is designed to understand how to interpret this output and submit it to the client.

Such a powerful capability greatly expands the types of roles that a Web server can be designed to fulfill. These roles are further expanded by additions to HTML that allow for graphical fill-out forms whose information can be sent back to CGI programs for further interpretation. With these extensions, an HTML author can generate a graphical form with text input areas, check boxes, and buttons using only tags within the HTML language. The author can then craft a program that collects the information from this form, processes it, and sends interactive information back to the user. The techniques and concepts covered in this discussion leverage this capability.

Structured Query Language. The Structured Query Language (SQL) database is a common way to manipulate data within a database. It selects rows within the database tables that match given criteria. The language itself is fairly simple, yet extremely powerful, and is standardized in one form or another in almost all commercial relational database products.

Database Connectivity. Database connectivity allows access to a database from a remote client machine and enables an application not directly connected to the database to obtain the results of SQL-like queries. Although this technology is not standardized, most major relational databases have products that allow users to make a remote connection to their databases. Examples of such products are ORACLE's SQL*Net* and Microsoft's SQL Server.* On Microsoft platforms, open database connectivity (ODBC) specifies a standardized library of database calls that can be used from within applications to access databases. Database vendors supply versions of ODBC drivers that correspond to their particular database, making all connectivity issues transparent to users.

Techniques

This section describes the theory and the methods/tools used to create the programs that link a database to the Web.

Theory. There are several ways to link the Web to a relational database, but most of the methods rely on the CGI described earlier. The underlying key to database integration is creating a program that connects to a database, performs a specific operation, and outputs the results. Integrating this program with the CGI creates an interface that can send the output of a database query back to an HTTP client in five steps:

- *Step 1.* The client machine (running the Web browser) requests information from a URL on the Web server. (The URL then accesses a CGI program that retrieves data from the database.)
- *Step 2.* When the Web server receives the request from the client, it runs the requested CGI program, which connects with the database.
- *Step 3.* The database server runs the specified SQL code and outputs the results.
- *Step 4.* The CGI program retrieves the output sent back from the database. It then relays this information to the client, along with any additional HTML code that the browser requires to display the data.
- *Step 5.* Completing the loop, the Web server passes the HTML package that was dynamically generated by the CGI program back to the client's Web browser, where it is displayed.

The scenario described here, and pictured in Figure 1, is simplistic, but it shows the basic layer on which greater capability can be built. For example, the data within the CGI program could be massaged before it is returned to the client. The query sent to the database might be a listing of all the categories of a particular product that resides in inventory. The data returned from this query can be monstrously large. Rather than returning this data directly to the client, the CGI program can group the listings into more recognizable categories. Data from two separate queries (even, perhaps, from two

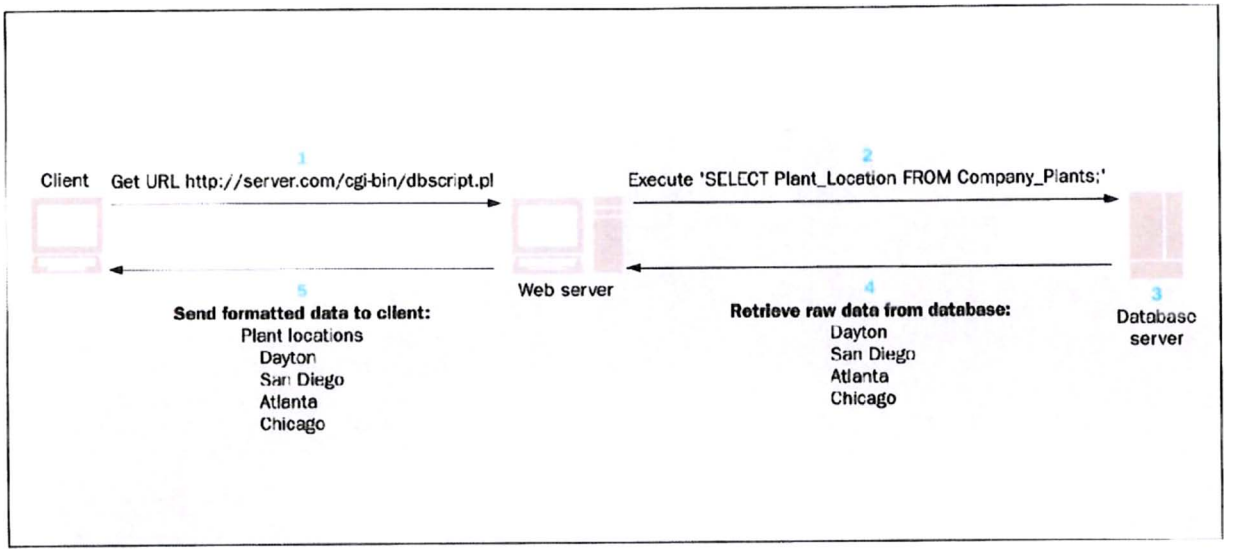

Figure 1. The circular process of Web/database connectivity. Information requests flow from the user through the Web server to the database, and back again.

separate databases) can also be combined into a single report. The final report could combine elements of text and character formatting as well as in-line images.

Another application for which HTML/database combinations are ideal is drill-down reports. The ability to "drill down" within a report to the underlying data that comprise particular data groupings is a primary characteristic of modern decision support systems (DSSs). An example of this type of system would be a report that lists the number of manufacturing defects for a given list of products. In an ideal report, a manager would be able to see these numbers and, by drilling down into the underlying information, learn exactly which types of defects were included in a given number. This is difficult to accomplish with a static report, but quite easy to do with HTML. Because HTML allows words within a text document to become hypertext links to other URLs, the CGI program can output hypertext links that call other CGI programs (or the same program with different options) to do further database queries. Figure 2 includes an example of this concept.

Methods/Tools. The theory itself is easy enough to understand, but creating the database connectivity programs can be challenging. One key to Web/database interaction is a gateway, such as a CGI program, which receives input from the user's browser and uses it to communicate with a database. Unfortunately, gateways must often be tailored to specific platforms or databases. The gateways described here are UNIX*/ SQL, ODBC, perl, and database-enhanced Web.

UNIX/SQL gateways. An SQL gateway is a program designed to interact with a particular database using SQL. The gateway itself may be a referenced CGI program or a set of libraries/programs that the CGI application uses for database connections. These gateways, primarily implemented on UNIX systems, are database specific. To build such a gateway, a programmer will need access to the database programming interfaces normally provided by the database vendor. These interfaces provide a set of library calls that can be used by programs to perform database operations such as connection and execution of database commands within a user program, often written in the C programming language. Examples of such interfaces are Pro*C* for ORACLE and ESQL/C for Informix. Often, these gateways are already available. Users can check the Web site of their database provider to see if a

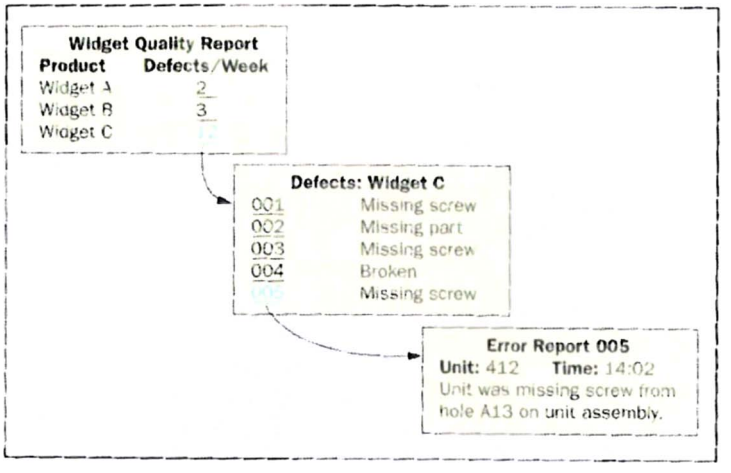

Figure 2. An example of the report drill down concept using hypertext.

particular gateway has already been created.

ODBC gateways. Microsoft's ODBC standard, a newly emerging trend for servers that run on Microsoft Windows,* Windows NT,* and Windows '95, connects databases from Windows-based programs such as those developed in Visual Basic* and Visual C++. Users who rely on ODBC have to find the appropriate ODBC driver for their particular database. This is a fairly easy task, with many of the major database drivers already available as a standard part of the operating system. The ODBC method offers users the same functionality as UNIX/SQL gateways, without the added burden of writing (or finding) the database-specific portions themselves. There is another major difference between ODBC gateways and UNIX/SQL-based gateways. Methods differ for passing CGI information between Microsoft Windows-based Web servers and most UNIX servers. Once the differences are understood, however, the methods of writing the two programs are relatively similar.

Perl gateways. Perl is an interpreted-script language that has become popular for Web server applications. This popularity is due, in part, to its speed, efficiency, and portability. Perl is available on both Windows-based and UNIX platforms, which is an advantage when writing for cross-platform compatibility. Because perl is used so extensively, several perl code libraries exist to help create CGI programs. To assist with database access, extensions to perl have also been designed, such as oraperl (ORACLE), sybperl (Sybase), and isqlperl (Informix) for UNIX platforms. For Windows-based platforms, later versions of perl can be combined with ODBC to provide a general database interface.

The primary advantage of using perl as a gateway is that the database connectivity portion of the application need only be written once. As soon as perl has been extended to communicate with a given type of database, such as oraperl for ORACLE, the programmer never has to worry about reprogramming the database communication portion. He or she can simply pass the SQL statement to a perl function call and retrieve the results. Although the same effect can be achieved using libraries in C, it is simpler and faster in perl.

Perl interfaces also allow the data being returned from the database to be analyzed piece by piece. Each row and column of the data being returned can be placed into separate variables for easy manipulation. The power of this method becomes obvious when it is compared with the methods used by other gateways, which return data in large chunks that need to be parsed to separate the individual elements. This method is especially powerful when used with HTML, because it allows much more latitude when trying to apply hypertext links or different formats to individual data items.

Database-enhanced Web servers. Database connectivity features have been built into Web servers, which are just beginning to emerge and gain popularity. In this

category, servers are largely database dependent, though several Microsoft-based servers support ODBC-compliant databases. From a technical perspective, CGI programs on these Web servers need only use specialized calls to the server to communicate with their database, removing the burden of database connectivity from the CGI programmer.

Leveraging Web Opportunities

Integrating databases with the Web provides many possibilities for how information stored in a database can be distributed to customers and/or employees. However, the tradeoffs associated with using this technology versus its potential advantages are a very important issue for application developers. This section examines how the uniqueness of Web technology offers beneficial, powerful advantages and several distinct disadvantages.

Advantages. Four key advantages can be accrued using Web/database technology:

- Ease of administration,
- Deployment,
- Development speed, and
- Flexible information presentation.

Ease of administration is a result of how the database connectivity is maintained. As the previous section described, the only connection to the database is from the Web server itself. The Web server acts as a "middle man" that handles all the communications and simply passes the data back to the client. The power of this approach can be seen by contrasting it with the more traditional, non-Web methods for accessing a database. Client applications that rely on these methods have to be able to communicate directly with the database. Each client machine and application need all the requisite drivers, programs, and networking necessary to communicate with the database server. In many cases, this is a monumental task that radically increases the application's possible points of failure. If the database connectivity is not correctly maintained on any individual machine, that application will not work.

The Web approach needs only one connection—from the Web server to the database. All other clients connect using standard Web browsers that are designed to handle the networking to the Web server. As a result, the application developer does not have to worry about this aspect of the application's design; most of the graphical user interface and networking drivers have already been provided. The developer only needs to worry about managing the information that the server sends back to the client. Figure 3 shows a representative configuration for a database-linked Web server. Although the users of a Web/database application may be geographically dispersed, the primary development arena for the application can be confined to a fairly small set of local area machines.

A second major advantage of using Web techniques to access a database is deployment. Using Web technology to disseminate information enhances the value of the browsers that already exist. This is powerful for two reasons. First, these browsers are already available across almost all platforms, which relieves the developer of trying to develop graphical user interfaces (GUIs) across multiple customer machines and operating systems. Second, Web browsers are so widespread and inexpensive (some are even free) that developers can assume that customers will be able to use the tool as soon as the Web server is available, avoiding deployment issues such as installation and synchronized roll-outs. By leveraging the wide-area aspects of Web technology, as well as its ubiquity, developers can almost simultaneously deploy an application across multiple platforms and multiple countries.

This leads to the third major advantage for leveraging Web and database integration—speed of development. For the reasons cited earlier, large portions of the normal development cycle, such as deployment and client design, do not apply to Web-based projects. In addition, the text-based tags of HTML allow for rapid modification, which makes it easy to use feedback to continually improve the look and feel of the tool. With many of the traditional application development environments, such as the X Window System,* modifying a GUI is no small task. Even with modern development tools, the changes necessary to modify even the fields in an input form are daunting. More importantly, once the tool is deployed, making changes becomes an ever-increasing expense, because all older versions of the tool have to be replaced.

This is not an issue when using the Web, however, where changing a form is as simple as modifying a tag in a text file. Also, even when the tool is in general

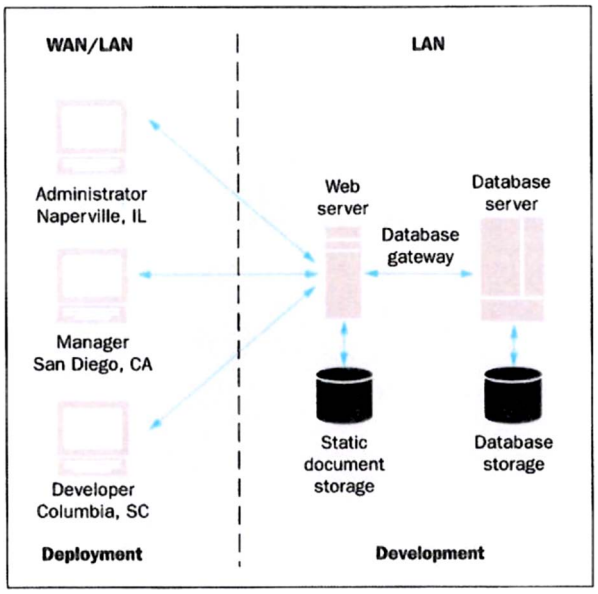

Figure 3. A sample Web/database configuration showing how development can be accomplished locally, while information is distributed globally.

use, changes are still relatively easy to make. The interface that the client sees is built from information sent by the server. Changing the information transmitted by the server makes those changes available simultaneously to all users of the tool. Web technology eliminates the traditional mindset that making changes to an application is extremely expensive after the application is deployed. With the client/server concept used by the Web, changes can be made continuously with no additional deployment costs or delays in schedules.

The fourth advantage of using Web technology to interact with databases is flexibility of information presentation. The Web/database combination reflects a synergy between two technologies that emphasize a decentralized, distributed focus. Within the Web, documents need no longer be linear. Large documents can be broken up into smaller, more manageable, chunks whose order of presentation is largely decided by the reader. Overviews can lead to more detail, which, in turn, can lead the reader to a level of detail difficult to obtain with more traditional linear methods of information presentation. This philosophy aligns itself comfortably with relational database methodologies. In essence, the database is used as a source of raw data that can be manipulated and organized to present information in a variety of ways.

As a front end, Web browsers allow the user to control these views and to retrieve more underlying data directly from the database whenever it is needed. An example of this concept is a Web interface to a database that holds *bill of material* information—a listing of all the subcomponents that comprise a particular product—for numerous products. By itself, this database is simply many rows of raw data. The relationships established within the database, however, allow for several different views of the same information. For instance, the user might click on a hypertext link that retrieves a list of the company's products from the database. A second link might break a particular product into its subcomponents, and a third link might list all the products that share a particular subcomponent. All these views are relatively easy to create within a database environment that was designed for these types of data manipulation. To try to present these same combinations of views from static files, however, would be a tedious task indeed. Combining the data manipulation power of a database with the information presentation power of the Web creates an application that is both flexible and powerful.

Disadvantages. Although the Web/database linkage has several important strengths, it also has a few weaknesses. When the graphical portion of the HTML interface, though powerful, is compared with the functionalities normally available to an industrial-strength GUI application, it leaves a lot to be desired, especially in its ability to generate graphical forms with buttons, text fields, and selection menus. These features are useful, but they cannot activate (or deactivate) portions of the form based on inputs from other sections of the form. For example, a developer might want to create an application that allows the user to query a database of automobile parts for trucks and cars. Once the user has indicated that he or she is looking for a specific truck part, subsequent menus would show only those parts that pertain to trucks. Although this functionality is frequently needed in application programming, it is absent in basic HTML. Techniques exist to circumvent this problem (such as using one form to ask the "truck vs. car" question and

reloading a new form with only truck information), but if they are presented improperly, they can make the application bulky and difficult to use.

Another disadvantage of working with HTML is that it generates, almost by definition, a stateless application. Each page or form that the user loads initiates a new connection with the Web server. In the case of a normal Web server, no information is kept between connections to track the position, or state, of the user. Often, such information as where the user has already been within the Web site and the nature of his or her previous input can be useful in an application.

An example of this is a program that prompts the user for his or her name at the very first screen. In a traditional program, it is easy to make this information available at any point in its execution. The user might run three reports and issue several queries, yet the program should still be able to recall the name that was typed at the beginning. With HTML, however, because every query is a new connection with no maintained state, this type of manipulation becomes much more tricky, making it necessary to pass along any information that might be needed later in the program. This information is "handed along" from connection to connection until it is needed. Modifications being built into Web servers today address this need, yet the basic premise of Web applications as stateless is still a barrier to many types of programming. As with the previous example, working around this difficulty is possible, but tedious.

No list of Web disadvantages would be complete without at least some mention of its security concerns. Because the protocols used with the Web depend largely on wide-area networking, such as the Internet, the Web is subject to most of the security concerns that pervade usage of the Internet. Data sent to and from Web clients is sent "in the clear." In other words, this information, at least in the case of the Internet, travels across public wires outside most corporations' span of control and in a form that is fairly easy to intercept and read. In addition, the same ubiquitous browsers that are an advantage to the development of Web-based applications also imply that potentially anyone with a browser can use, or abuse, the application. Although Web administrators can limit those who have access to particular applications in several different ways, each method has its problems. Corporations who are providing Web applications behind secure "firewalls" have less to worry about in this regard, but security can still be an issue if items such as payroll or performance evaluations are involved. A number of approaches can be taken to reduce these security problems.

Several emerging standards require encryption to be used to secure the information provided via Web servers. Until these standards are more pervasive throughout the Web community—and are supported by the majority of available browsers—security will continue to be an issue for application developers.

Future Directions

Although the techniques used to link databases to Web servers are still in the early stages of deployment, several new technologies appear promising. One of these is a new breed of Web server with the ability to link to databases built directly into it. This allows Web authors to add SQL links within their HTML documents and scripts. These servers use the same concepts mentioned in this paper, but their operation is hidden from the user, making the authoring process much easier. Even though a number of these servers are being offered by various companies, not enough momentum exists to drive this functionality into many of the mainstream Web server packages.

Other entrants in the Web/database business also include the database companies themselves. These companies now realize that many customers want the ability to link the information within databases to Web content. ORACLE, for instance, provides a tool kit consisting of several different programs that facilitate Web access to their SQL*Net product. Other companies have taken the concept one step further by including entire Web sites within the database itself. These products offer features such as version control and change history tracking, both of which can be useful tools in a large Web site.

In the arena of tools, several recent arrivals on the Web product scene can help developers retrieve information from databases. For instance, a number of software libraries have been developed to allow common CGI access for servers that use Microsoft Windows and Windows NT. These packages take advantage of the

ODBC interface provided by Microsoft to help facilitate the data retrieval. Another exciting product being developed is a database module for the newest version of perl, perl5. This version has significant advantages compared to earlier versions, one of the most important of which is its ability to write third-party modules that can "plug in" to perl. One proposed module—DBI (Database Interface, formerly DBperl)—is a type of replacement for current perl 4.X database extensions such as oraperl and sybperl. DBI is a common set of application program interface calls designed to be applicable to any database. To make this module operate, a developer only needs the proper drivers for his or her particular database type. Allowing common portability not only across versions of perl on different platforms, but also with different databases, will make database-type linkages much more prevalent and easy to use.

Conclusion

Combining HTTP servers with the strengths of a database is a natural progression of Web technology. Although this technology is still in its infancy, continual advances are making such a combination easier to achieve. One primary consideration for this technology is its impact on traditional application development, especially in the client/server arena. The advantages inherent in the Web, and the direction in which it is heading, make it an obvious choice for tool development in the near future. The Web/database combination has enormous potential for creating DSSs. The ease with which HTML can implement a data "drill-down" concept, coupled with the data storage and retrieval capabilities of a database, make it a prime candidate for DSS development. More importantly, the rapid prototyping capabilities inherent in HTML support quick modifications and improvements to applications built on this technology.

There are extremely powerful facets of Web technology that have yet to be fully explored. Using database integration, developers can increase the types of information that can be disseminated and widen the range of applications that can be generated. Linking the Web to relational database technology represents an efficient, flexible means of storing, retrieving, and presenting information.

***Trademarks**

(Manuscript approved February 1996)

John K. Whetzel *is a software engineer in the Quality and Productivity Services, Server Systems Department at NCR Corporation, formerly AT&T Global Information Solutions, in Columbia, South Carolina. He works on World Wide Web administration and deployment, database integration, and software tool development. Mr. Whetzel joined the company in 1992, after receiving B.S. degrees in both computer engineering and electrical engineering from the University of Missouri in Columbia, and an M.B.A. from the University of South Carolina in Columbia.*