NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

LIBRARY USER INTERFACE

GRADUATION PROJECT
COM - 400

STUDENT: ZIAD ABDELHAMIED
971002

SUPERVISOR: MR.OKAN DONANGIL

NICOISA-2000

Table of Contents:

Acknowledgment:

I would like to thank my parents, who showed me how to be human, taught me how to use the pen and was always there for me. Thanks for everything.

I would also like to thank my supervisor Mr.Okan Donangil. Without your support and understand this was going to be really difficult.

Ziad Abdelhamied

Abstract:

Library and Internet, two terms lead to the world of knowledge. Joining the two terms by designing a web library user interface with all the facilities of both library and Internet is defiantly making that world of knowledge unlimited. And this is what my project about.

Introduction:

Intranet Web sites are increasingly becoming the backbones of the information infrastructures within organizations. Prior to intranets, information in organizations was usually scattered across countless servers with obscure server names, user names, and passwords. To make matters more confusing, there was no uniform medium or format to access and view the information. One document could be stored in Microsoft Word format on one server, another document could be stored as a Postscript file on another server, and yet another document could be stored in WordPerfect format on another server. Not only did users have to remember all the obscure server names, user names, and passwords, they had to have countless helper applications (such as Postscript file viewer, Microsoft Word viewer, and so on) installed on their computers to browse information. Intranets solved this information-distribution problem by providing a cross-platform medium, Hypertext Transport Protocol (HTTP), and document format, Hypertext Markup Language (HTML).

Other reason that let me choose to design the library user interface as a web is the ability to provide the library services and information for students even when they are outside the university campus. That is simply by connecting to the university website through the Internet.

The information I'm providing in the library web need two types of web pages:

- Static for providing the fixed information like contacts, online books…etc. this type of pages is displayed as it is and contains the information that is rarely changes.
- Dynamic for retrieving the information on the library databases. Also called interactive because the information on it depends on the query entered by the user.

1.1 Web programming Language

1.1.1 HTML

The lingua franca of the Web is Hypertext Markup Language (HTML). The main function of HTML is to provide information that the browser can use to make formatting decisions for displaying the contents of a web document. This markup information includes basic text formatting, such as codes to bold or underline marked text. It also includes directions for inserting other files into the current file, particularly graphics. But the most distinctive feature of HTML is that it gives you the ability to mark areas of the document (both text and images) as hypertext.

Hypertext refers to a collection of documents that are cross-referenced, or linked. The advantage of hypertext is that connections between ideas can be easily conveyed by the author and followed by the reader. In traditional text, reading is linear: page 1, then page 2, until the end. Hypertext allows a nonlinear arrangement of connected ideas to be presented to readers, who can follow any thread of discussion they wish. More important, since all of these hypertext documents are located on the Web, documents can be linked to each other. This provides an enormous amount of flexibility in writing hypertext documents. For example, if this report were a hypertext document, I could link all of the computer terms that I discuss to their definitions in The Free Online Dictionary of Computing.

WEB LINK

The Free Online Dictionary of Computing is a huge compendium of computer and computing terminology. It's an excellent resource, located at www.instantweb.com/foldoc/index.html.

SGML: Parent of HTML

The publishing industry enthusiastically accepted computers as labor-saving devices almost as soon as they became commercially available. Unfortunately, however, the industry quickly realized that computers had problems figuring out what to do with text. The text itself wasn't the problem; it was more an issue of formatting text properly. The underlying need was for the computer to understand that a block of text was, for example, a headline, which may be treated differently by different publishers and in different contexts, but nevertheless should always be treated as a headline.

The answer to this problem was the Standardized General Markup Language (SGML), which was adopted by the International Standards Organization (ISO) in 1986. It provides a human-readable way to apply structure to the content of a text document such that it will be consistently formatted and interpreted. In essence, the SGML contains all of the information about the text that is not the text itself. (Note that SGML and WYSIWYG (what-you-see-is-what-you-get) are not the same! A WYSIWYG word processor may ensure that the title is in 18-point bold italic Times New Roman font, but the program does not know that it is a title. SGML on the other hand, recognizes the text as a title and thus can determine how to present information that is defined as a title.)

An SGML document consists of two essential components: text that has been "tagged" or "marked-up" with SGML information; and a document type definition (DTD) that explains and defines the tags for the document, describes the contents of each element, and delineates the relationship between elements. For example, a DTD for a play could contain a tag called <SCENE>, which contains a paragraph of information about the scene and a number of <STAGE DIRECTION>, <SPEAKER>, and <LINE> tags.

HTML is essentially a specific DTD for hypertext documents that more or less conforms to SGML standards. More recently, the more powerful eXtensible Markup Language (XML) DTD has been accepted by the World Wide Web Consortium (W3C) as a new standard for Web content. Microsoft has also lent its weight to the standard by promising to use XML for its Office suite of products and by using it to construct its Active Channel Format for push technology in Internet Explorer 4.0.

HTML is text-based language, which makes it much easier to create, maintain, and transfer web documents among different computer platforms and operating systems.

HTML formatting is applied to document components using tags. In most cases, a pair of tags mark the extent of the formatting. The start tag is always of the form <TAGNAME> and the end tag is always of the form </TAGNAME>. The tag is case-insensitive. The tag name that indicates text should be underlined is the letter U; for example, this statement:

<U>This text will be underlined</U> and this text will not.

would produce:

This text will be underlined and this text will not.

when it is interpreted by the browser.

One important thing to note is that the HTML documents are static. The formatting is embedded in the web document, stored on the server, and then accessed by the client. There is no provision for changing the content of web pages, especially with regard to input from the user. Essentially, the only thing a user can do is request a document, and all the server can do is provide it.

As the Web matured, HTML became a limiting factor for many projects that people wanted to execute. To address some of these limitations, the Internet Engineering Task Force (IETF) began to draft a new specification of HTML. But browser manufacturers, principally Netscape, responded even faster by implementing their own proprietary extensions to HTML that were only recognized by their own browser. Some of these extensions became part of the new HTML specification, while others remained Netscape-specific. This process is ongoing, with Microsoft also playing a major role.

NOTE

The HTML specification has undergone a number of revisions. For the latest information on HTML specifications and proposed changes, go straight to the source: the World Wide Web Consortium coordinates nearly all aspect of the Web's growth. You can find the W3C at www.w3c.org/.

One of the most important additions to the HTML 2.0 specification was the capability to create forms for user feedback. A basic set of components, such as text boxes, buttons, and list boxes, were defined to make web documents more interactive. These components changed the nature of web documents so that two-way communication between the browser and the server was more useful. This addition made rudimentary web databases a possibility.

1.2 Some Guidelines for Developing Effective Web User Interfaces

Web user interfaces are quite different compared with the user interfaces of traditional applications. They are especially different from Windows applications. The following are some guidelines for building effective user interfaces :

- Streamline forms to make data entry easier
- Optimize for limited bandwidth
- Use colors effectively
- Use fonts effectively
- Use the capabilities of the "intelligent client"

1.2.1 Streamlining Forms to Make Data Entry Easier

When building data entry forms, you should streamline them for ease of data entry. You can do this by trying to organize data entry fields into a logical order. Fields that are most likely to be filled in should be near the top of the form so they can be filled out without the need to skip a lot of fields.

Design your interface so that the reader does not have to scroll off the screen. Depending on the information, it might be better to offer a way to continue to a second screen instead of forcing the user to scroll down.

You should specify default values for HTML data entry controls. By supplying the most common default values, you help the user complete the form easily and quickly.

A final suggestion for entry forms is to clearly mark any fields that are required. By marking these fields, you show users exactly what they must enter.

TIP: It is good programming practice to include default data values to guide the user toward entering valid data.

1.2.2 Optimizing for Limited Bandwidth

Watch the size of your files when building data entry forms. No matter how well your Web forms are designed, they serve little purpose if they take too long to load and display on a Web browser. Web users are not always as patient as you would like them to be. If a page does not load within about 30 seconds, the user might move somewhere else (and fill in someone else's form!). Watch out for the following when designing Web forms to ensure the Web form loads quickly:

Avoid using large graphics in Web forms. In fact, most Web forms really do not require any graphics. Use graphics only if you have to.

Use Java applets and ActiveX controls only if you have to. Java applets and ActiveX controls can be used to add some flair to Web forms; however, they add to the form's file size and, in some cases, affect the time it takes to render the Web form.

Avoid using complex background graphics in Web forms. They take longer to load and add to the total file size of the Web form. Plain white backgrounds are fine for most Web forms. If you must do something fancy, select a two-color Web graphic with a left-hand border.

Avoid complex form controls. For example, avoid using a drop-down list with many selection items. Each item you add to a drop-down list must be downloaded with the form. If you have several drop-down lists with many items, then this can begin to add up in size.

TIP: Try not to add more than about 50 items to a selection list.

1.2.3 Use Colors Effectively

Colors can be used to make your Web forms more interesting and visually appealing. There is a flip side to using colors, however. Ineffective use of colors can make your Web forms unattractive and difficult to use.

When using colors, stick to a limited number of very dark and light colors. If you are using a light color for the text, use a dark color for the background to add contrast. Here are some guidelines for using colors more effectively:

Don't overuse colors. Keep the number of colors on your page to a minimum.

Don't assume the reader is using a monitor that supports a high number of colors. Browse your Web page with a browser running on a computer with only 256 colors. This is especially important if you are using images on your page.

Use colors only when necessary. A good use of colors is to display the labels of required data fields in red.

TIP: After building a Web form that uses colors, ask someone else to give you feedback on your color scheme.

1.2.4 Use Fonts Effectively

Fonts can be used to add a professional touch to your Web forms. When using fonts in Web pages (as well as Web forms), it's best to stick to fonts that are available in virtually all computers, such as

- Arial
- Courier
- Times Roman

Fonts, like colors, should not be overused. You should limit the number of different fonts used to as few as possible.

1.2.5 Using Capabilities of the "Intelligent Client"

When designing a Web site, you need to determine what browsers will be used to access your site. In addition to considering which browsers, you need to consider which versions. By knowing which browsers and versions are accessing your site, you can add more functionality to your pages by taking advantage of the browsers' built-in functionality.

For example, if you know that both Netscape Navigator and Microsoft Internet Explorer browsers are going to be used and if you know that you want to use scripting, then you should design your pages with JavaScript rather than VBScript. JavaScript is supported by both browsers, so your code will be portable. If you use VBScript, the Netscape browsers might not be able to run your scripts.

You can use the browser's intelligence to help design more user-friendly applications. With client-side scripting, you can create a more effective Web user interface by giving the user instant feedback if a data entry field is not filled in properly. Client-side validation is not a substitute for server-side data validation, however. You still need to plan on validating data on the server by using an ASP subroutine. Client-side data validation complements server-side data validation. It is not a substitute.

NOTE: Microsoft FrontPage can automatically generate JavaScript code that validates the form data on the client side.

Chapter 2 Understanding Web Database Technology

There are three basic components to any web-based database application: web technology, database technology, and the technology that connects them. The software that connects the Web to the database is often called middleware since it sits between the application and the network. Figure 2.1 shows this generic layout and lists some of the technologies that will be covered in this report.

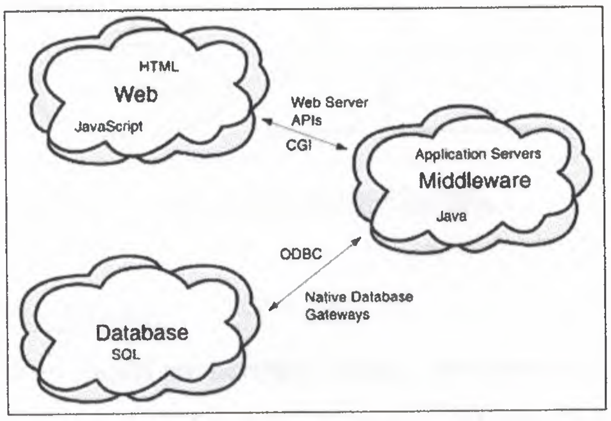


Figure 2.1 The three clouds of Web database technology.

2.1 The Web Side

Understanding how the Web works is essential to understanding how to put a database on the Web.

At its most basic level, the Web is a way to share information. The information is stored in a standard format that includes both the actual textual and/or numeric data and some degree of formatting control. This information is stored on a remote computer that is connected to a network, either the global Internet or some private network (a company's intranet for example). The real innovation is that the information is accessible to anyone connected to the network, not just people who have usage privileges on the computer. The basic outline of this process (as shown in Figure 2.2) is:

1. User requests a file from the remote computer using a web browser.
2. Web browser passes request through HTTP.
3. Web server on remote computer receives request and processes it.
4. If the file exists on the remote computer and is web-accessible, the remote computer delivers the file to the web server.
5. The web server forwards the file to the appropriate web browser client.

6. Web browser interprets the formatting embedded in the file and presents it to the user.

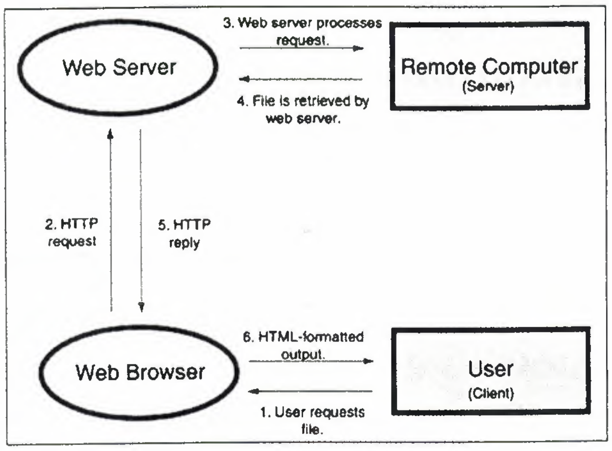


Figure 2.2 Web client/server diagram.

2.1.1 Web Clients

Web clients are normally called web browsers. The most commonly encountered browsers (Table 2.1), Netscape Navigator, Microsoft Internet Explorer, and the venerable Mosaic, are the user's gateway to the information available on the Web. These clients have three main functions:

1. Communicate with web servers on the Internet using the HTTP protocol.
2. Provide the tools to navigate between web documents and servers.
3. Offer a means of viewing the content of web documents.

The integration of these functions is what has made the Web so popular in comparison to older technologies such as anonymous FTP (file transfer protocol) and Gopher.

Table 2.1 Web Browser Popularity*

Server	Browserwatch	Georgia Tech	Zona Research
Netscape Navigator	58	60	62
Microsoft Internet Explorer	31	15	36
Others (including AOL)	11	25	2

*Based on percentage of users.

Sources: Browserwatch user survey (4/13/98), Georgia Tech 8th WWW Survey of self-reported webmasters (11/16/97), and Zona Reseach press release on corporate web browser choices (9/29/97).

The Client/Server Model

The relationship between the web browser and the remote computer is an example of the client/server model. In this scenario, the remote computer acts as a web server and the web browser acts as the client. The client sends a request for information or some action to the server. The server responds, typically by presenting a file to the client. The browser then takes the file, processes it, and presents it to the user.

The client/server model was developed to address a number of needs, primarily for large-scale installations such as government and industry. In the early days of mainframe computers, the clients were directly connected to the mainframe. Often called dumb terminals, the clients typically consisted solely of a monitor and a keyboard. One major advantage of this scheme is that a single powerful server can simultaneously respond to a number of different users. Because the clients are relatively cheap and the server is fairly expensive, this model controls costs since client terminals can be added as the number of users increase instead of buying one computer per person.

Another advantage of the client/server model is that files can be shared more easily since they are located on the same machine. Centralized storage makes it easier to collaborate on files.

The problem with the client/server model is that the demand for processor time and storage space is always increasing and those resources are finite. It's much more expensive to upgrade a large server than it is to replace a few desktop machines. Even more problematic is the "eggs in one basket" nature of the client/server model. When all users are working on one machine, a failure of the server results in a loss of computing power for everyone. Desktop machines can fail without affecting as many other people, and can usually be replaced with a spare?whereas there are rarely extra servers lying around as replacements!

It is interesting to note that we've come full circle in the past 30 years from the mainframe approach to desktop machines back to mainframes. The "network computer" or "thin client" that is being promoted by Sun and other manufacturers harkens back to the client/server model of the 1970s.

2.1.1.1 Client-side Processing

A more recent advance in web technology is client-side processing. Earlier in the discussion about the client/server model, I implied that the server did all the work while the client passively displayed it. This is true to a great extent for most web documents, but there are a growing number of options for incorporating more dynamic elements into web documents that can be interpreted by the browser itself.

The earliest arrival was Netscape's Livewire, which is now known as JavaScript. JavaScript and Microsoft's newer VBScript (roughly based on their Visual Basic programming environment) are both text-based languages directly embedded into the web document. Both are also event-driven languages that can manipulate the elements of a web document through the browser. The scripting capabilities consist primarily of triggered responses to browser events. These events range from direct user interaction, such as mouse clicks on elements of the page to more global browser events, such as the loading or unloading of a particular page.

NOTE

JavaScript has absolutely nothing to do with Java. It is not a different nor an easier version of the Java programming language. The name switch is essentially a marketing ploy by Netscape!

Scripting languages such as these are relatively easy to use, though not very powerful. Their main advantage is that they let the page react dynamically to the conditions or changes of the browser without needing to access the server. In essence, the scripts execute in the browser. This cuts down on the number of exchanges between the web server and the browser, which speeds up performance.

For example, JavaScript can be embedded in a page to change the layout, based on the type of browser that is being used to take advantage of browser-dependent HTML extensions. Another common use is to have a simple script that changes the state of a small, button like image on a web page from an on state to an off state when clicked. The most relevant use of scripts for web databases, however, is to validate data entries. If a text box is set up to contain a five-digit zip code, client-side scripting can check the length of the string in the text box and notify the user that an entry is incorrect before it is sent to the server. These types of scripted validation routines are very common in web database design.

Dynamic HTML (DHTML)

One of the goals of web developers is to be able to dynamically change the actual HTML code on a web page without having to interact with the server. This allows far more interactivity without having to squeeze interactivity programming back and forth through the narrow data bottleneck between the client browser and the remote server. This dynamic HTML (DHTML) is in active development and some features are already integrated into the 4.x version browsers from both Netscape and Microsoft.

Of course, Microsoft and Netscape both implement DHTML using differing technologies and specifications. Currently, there is virtually no way to script cross-browser DHTML. But the W3C is working on DHTML standards, and by the time going on, may have smoothed some of the differences.

2.1.2 Web Servers

Web servers are the workhorses of the World Wide Web. Their fundamental duty is to receive, interpret, and respond to the requests of web clients. Servers are responsible for providing the greatest percentage of web traffic across the network.

The phrase "web server" is often used in two distinct ways. At the most basic level, a web server is a piece of software that is actively "listening" for client HTTP requests. Some of the most popular web server packages (Table 2.2) include the original NCSA web server, Netscape Suitespot server and its other varieties, Microsoft Internet Information Server (IIS), O'Reilly Website Server, and the Apache web server.

Table 2.2 Web Server Popularity*

Server	Netcraft	Georgia Tech	Zona Research
Netscape Navigator	58	60	62
Netscape servers	12	22	42

Microsoft IIS	21	21	28
Apache server	48	30	N/A
Others (including Macs)	19	27	30 (inlcudes Apache)

*Based on percentage of users.

Sources: Netcraft automatic monthly survey (4/98) of most of all available sites (2,215,195 in this survey), Georgia Tech 8th WWW Survey of self-reported webmasters (11/16/97), and Zona Research survey reported in ZDNet 12/15/97.

The other sense of the phrase "web server" is the more traditional notion of a physical machine dedicated to a particular task. In many cases, especially in the business world, a particular computer (or cluster of computers) is dedicated solely to running a web server software package. In this case, there is no real difference between the software and the machine that is running that software. In other cases, the server software may be running on someone's desktop machine, sharing space with the normal day-to-day work that is done on the machine. Obviously, a dedicated machine is preferable for serving web documents.

2.1.2.1 <u>Server-side Processing</u>

The bulk of the web server's work consists of sending files to a web client. But as the Web has grown in popularity as a general computing idiom, facilities for interactivity have gradually made their way into server software. The ability to respond to client responses appeared in web server software to handle data that was entered into HTML forms. The information provided by the web client is processed by a program running on the server. The program on the server then either redirects the client to a URL based on the results of the program, or it dynamically generates a new web page in response to the input.

The Common Gateway Interface (CGI) is the most straightforward way to process responses from the Web. In a typical setup, an HTML form is submitted to the server, and that form data is then passed through CGI to a processing program. CGI also provides access to a standard set of general information (see Table 2.3) about the web client, such as the type of browser being used by the client.

Table 2.3 List of CGI Environment Variables

Variable	Description
AUTH_TYPE	The authentication method used to validate a user.
CONTENT_LENGTH	The length of the data (in bytes or the number of characters) passed to the CGI program through the standard input.
CONTENT_TYPE	The MIME type of the query data, such as text/html.
DOCUMENT_ROOT	The directory from which web documents are served.
GATEWAY_INTERFACE	The revision of the Common Gateway Interface that the server uses.
HTTP_ACCEPT	A list of the MIME types that the client can accept.
HTTP_FROM	The email address of the user making the request. Most browsers do not support this variable.
HTTP_REFERER	The URL of the document that the client points to before accessing the CGI program.
HTTP_USER_AGENT	The browser the client is using to issue the request.
PATH_INFO	Extra path information passed to a CGI program.
PATH_TRANSLATED	The translated version of the path given by the variable PATH_INFO.
QUERY_STRING	The query information passed to the program. It is

	appended to the URL with a question mark (?).
REMOTE_ADDR	The remote IP address of the user making the request.
REMOTE_HOST	The remote hostname of the user making the request.
REMOTE_IDENT	The user making the request. This variable will be set only if the NCSA identityCheck flag is enabled and the client machine supports the RFC 931 identification scheme (ident daemon).
REMOTE_USER	The authenticated name of the users.
REQUEST_METHOD	The method with which the information request was issued.
SCRIPT_NAME	The virtual path (e.g., /cgi-bin/program) of the script being executed.
SERVER_NAME	The server's hostname or IP address.
SERVER_PORT	The port number of the host on which the server is running.
SERVER_PROTOCOL	The name and revision of the information protocol the request came in with.

Server-side processing is essential to any database-oriented web application. CGI provides one method for accessing programs that reside on the server, but it is fairly slow and requires a fair knowledge of the operating system (often Unix) and at least one programming language (usually Perl). Both Microsoft and Netscape have addressed the speed and complexity issues by incorporating application programming interfaces (APIs) into their web servers. The API allows software developers to access the server software directly, instead of through CGI, which drastically increases processing speed of forms and allows for a great deal of server customization.

CGI versus API: Which Is Better?

The basic argument between CGI and vendor-specific API interfaces for server-side processing boils down to compatibility. The Web was originally designed to be platform-independent. An HTML file from a particular server running a specific operating system, for example, can be interpreted by a browser running on any other platform. But, as early browser developers quickly noticed, there were many features that could be added to HTML to increase its usefulness. Unfortunately, these extensions to HTML could not be interpreted by other browsers since they were not part of the standard. This put the web page designer and the web surfer in the difficult position of making choices between compatibility and features.

The choice between using CGI or API-specific software to handle server-side processing is at once more complex and more straightforward. Since the web client is unaware of the method of server-side processing, all of the ramifications about implementing it rest on the web designer. This means that changes should be more or less transparent to the outside world. But since server-side processing requires a sizable investment of programming effort, changes are much more expensive.

The main advantage of CGI is that it is a universal format. All web servers implement some flavor of CGI capability. This means that if a program for processing a web form is written in Fortran-90 for an NCSA web server running on a Solaris 2.5 operating system, the same Fortran-90 program can be recompiled to run on a Windows NT 4.0 server running Netscape Enterprise Server. Both the HTML-based web forms and the processing program are fairly portable.

The downside of CGI is that it's slow. Each time a web client activates a program through CGI, a new instance of that processing program is started. So if 28 users submit forms to be processed, the server has to run 28 separate copies of the processing program! This will drastically affect the server if the processing is long or complex, or if there are many users.

Some server developers have tried to address this problem by allowing other pieces of software to directly interface with the server through an API. This greatly enhances the speed of processing since only one copy of the processing program is active. Each request for processing is either queued for sequential processing or handled in parallel through multitasking or multithreading.

In exchange for the speed, however, this type of approach locks you into a specific vendor. If you develop a custom program or buy a package that addresses the API of a particular web server platform, a new package must be developed or purchased to use any other server. A custom application written using Microsoft's ISAPI with Internet Information Server is practically useless on a Netscape server.

CGI is fine for simple scripts or institutions with deep programming expertise. It is also probably a good choice if you are outsourcing your web hosting since CGI can be used with virtually any Internet service provider. But I think the speed of API-based applications and the powerful tools that have been developed to support building them make it the clear winner for high-volume sites and applications that are hosted in-house.

2.2 The Database Side

Databases are a much more mature technology than the Web. Early systems were based on paper records, and later punch cards. Computer technology was applied to complex data processing tasks from their inception. Databases were one of the primary applications of early mainframes, and finally reached the desktop in the 1980s. During that period of development, a number of standards emerged to make it easier for different pieces of database software to be integrated. The large majority of this work was focused on relational databases since their use was so widespread.

2.2.1 Database Queries & SQL?

The most basic function of a database is to provide data based on user requests. In the early 1970s, IBM created a structured query language, better know as SQL to provide access to its relational database package.

SQL is based on the work of Dr. E. F. Codd, the father of the relational database. Since this was the first relational database software package, this language became the de facto standard for database development. It was officially adopted by both the American National Standards Institute (ANSI) and the International Standards Organization (ISO) in the late 1980s.

Fortunately, almost every commercial database understands SQL. This lingua franca of the database world makes it fairly easy to port data from one database package to the next. If a particular database operation is implemented using SQL, there should be no difference in how the query works on any other SQL-compliant database. This also means that it is definitely worth expending some time and energy to learn SQL!

NOTE

In reality, SQL is a language with as many dialects are there are vendors. Many have added their own extensions to the "standard" implementation of SQL. Many have also made different choices about how to quote strings or defaults for sort orders. In general though, very little tweaking should be required when moving typical SQL code among database vendors.

A Brief History of SQL

SQL first appeared in a prototype relational database system, System R, developed at the IBM San Jose Research Laboratory in 1974 by a team led by Donald Chamberlain. The database language they developed was called Structured English QUEry Language, or SEQUEL. As System R evolved into more sophisticated products, like IBM DB2, SEQUEL evolved into SQL.

SQL is firmly rooted in mathematical logic, specifically relational calculus. More formally, it is a mathematical formalization of relational algebra based on first-order predicate logic. Since that definition is probably as clear as mud to most of us, let's simply say that SQL is a nonprocedural and fairly English-like query language for databases.

The first standard implementation of SQL was dubbed SQL-86 by the International Standards Organization. There is also an SQL-89 and more recently SQL-92, also known as SQL2. To further complicate matters, both standards have multiple levels of compliance. And many vendors have yet to meet the full specification of specific compliance levels. So, despite the implication that there is one specific standard implementation of the SQL language, there are actually a number of dialects. Essentially, each database vendor has its own slightly different version of SQL, though the core functionality of each is nearly identical.

It incorporates a number of object-oriented features into the relational database design. A number of manufacturers are also writing database APIs based on SQL, such as Microsoft ODBC (discussed later in this report).

SQL is a very simple language, but its elements can be combined to quickly create powerful effects. It is also a language that reads very much like normal English (more so than HTML!), which certainly makes it easier to talk about. SQL can be used to do virtually anything to data in a relational database. It can perform maintenance tasks, such as deleting records or creating new tables, as well as provide a way to find data in the database. The basic commands are given in Table 2.4.

Table 2.4 Basic SQL Commands

Command	Description
DELETE	Remove data from a table.
INSERT	Add data to a table.
SELECT	Find data matching a specified set of criteria.
UPDATE	Change existing data in a table.

2.2.2 Database Servers

Database management systems (DBMS) have their roots in terminal-based mainframe applications. This means most modern databases are modeled on the client/server architecture. In other words, each DBMS consists of two distinct parts: client software that makes requests and a server that interprets the request and returns the appropriate data. In some cases, both pieces of software are integrated for use on the desktop computing platform (like Microsoft Access). In most cases however, there is a separate software package for each role.

History of DBMS Models

DBMS software was developed not long after computers became commercially available. While relational DBMS are in widespread use and OODB software is becoming more common, there are two other DBMS models that were extensively used in the 1960s and through the present day. All of these systems were designed to run on mainframe computer systems.

The hierarchical database model (HDM) was one of the earliest DBMS systems to gain popularity. The basic architecture, shown in Figure 2.3, looked like an inverted tree, starting with a root table with additional data tables as branches from the root. The only possible relationship in this model was parent-child. A single parent could be associated with many children, but each child could only be associated with one parent.

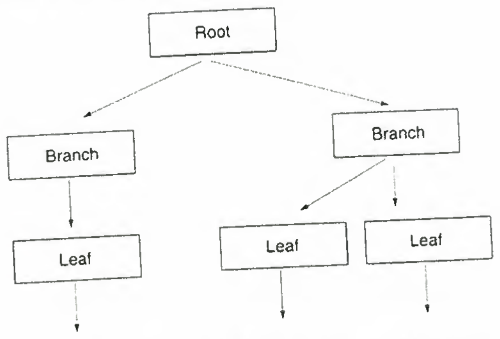


Figure 2.3 Hierarchical database model.

The data was linked either using a pointer or through the physical arrangement of tables. To access data, the DBMS had to start with the root and work through each branch leading to the desired record. Queries in such a system required extensive knowledge of the physical layout of the tables.

HDM had a number of advantages as a DBMS. It was an especially efficient architecture for accessing data stored on magnetic tape, which was the primary method of storing large data in the 1970s. It also had built-in referential integrity since the parent and child records were explicitly linked (adding a child required the existence of a parent, and deleting a parent led to the deletion of linked child records).

One drawback to HDM was the amount of redundant data that the design required in many situations. Another drawback was that complex relationships, especially many-to-many relationships, were exceedingly difficult to create. Both of these factors could lead to situations where certain queries were impossible without reconstructing the physical architecture of the database.

The network database model (NDM) was another DBMS architecture that was popular on mainframe systems. It fixed a number of the problems in the HDM. The architecture was essentially the same as the HDM with the primary difference that child branches could be shared by parent tables, as shown in Figure 2.4. The relationship was then a logical set where one table was the owner and the connected table(s) represented each member of the set. This arrangement allowed a one-to-many relationship between an owner and a member, and a one-to-one relationship between records in the member and the owner table. It also allowed records to exist in a member table without being related to an existing record in the owner table.

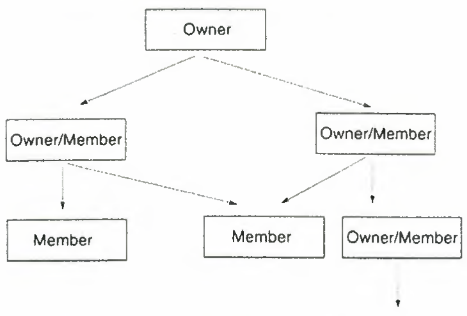


Figure 2.4 Network database model.

The advantage of NDM was that data could be accessed starting from any table and worked through the appropriate sets in the correct order. This greatly increased the complexity of questions that could be answered using the database. Queries also accessed data very quickly in this model.

Unfortunately, it was still necessary to know the structure of the database to work through the appropriate sets to find data. It was also exceedingly difficult to change the structure of the database since the logical sets for a particular query were related to the logical layout of the database.

As each model was implemented, the database users came up with questions to ask of the data that required more complex models. The relational database model (RDM) was developed in the late 1960s to address the inadequacies of the HDM and NDM approaches to database design. This is by far the most common model for modern databases. But in keeping with the patterns of progress, the object-oriented database model (OODM) is growing in popularity in situations where the RDM is not effective (such as media management and where inheritance is important or where data items are linked in complex ways to other data items). The power that each successive model gives the user simply results in users wanting even more power!

While this client/server architecture has much in common with the Internet's client/server model, there is one significant difference: Each server DBMS can only be accessed by a specific client. In other words, a Brand X RDBMS server requires a Brand X RDBMS client. This makes merging data from different sources or porting data to a new database platform a nontrivial task. This problem was alleviated in some ways by the introduction of SQL, which provides a common syntax for database functions. But each vendor has created their own dialect of SQL that accesses the full capabilities of their database server and thus requires their own specific client to generate the proprietary SQL code. In other words, the basic commands are the same, but any significant database application is going to require a significant amount of tweaking to port to a new DBMS.

2.3 Open Database Connectivity (ODBC)

In 1988, a number of database vendors, including Microsoft, Sybase, DEC, and Lotus, were all individually working on a way to solve this problem by providing common access to databases from a variety of vendors. The goal was to allow any program to transparently access data stored in the native data format of any database application. They pooled their efforts and jointly developed the Open Database Connectivity (ODBC) standard, which they released in 1992. Late that same year, ODBC was adopted by the ANSI SQL committee, which officially made it the standard interface for database access.

ODBC provides an abstraction layer between the application interface and the database, which effectively hides the differences and peculiarities of each specific database. The ODBC standard model is shown in Figure 2.5. This model provides a vendor-independent development environment for database applications. In effect, a client application can be written once using calls to ODBC, and can then access data stored in any ODBC-compliant DBMS. The only required component is an ODBC software driver for a particular DBMS that can translate the database query into the specific syntax of its particular database format.

ODBC has made the developer's job much easier. It provides a common way to access databases, while SQL provides the common syntax to perform database manipulations. Combined, these technologies make database application development much more efficient. For example, an application can be written using an inexpensive ODBC-compliant database on a desktop machine, which can be ported to a large server running an industrial-strength DBMS package simply by changing the ODBC driver for that application.

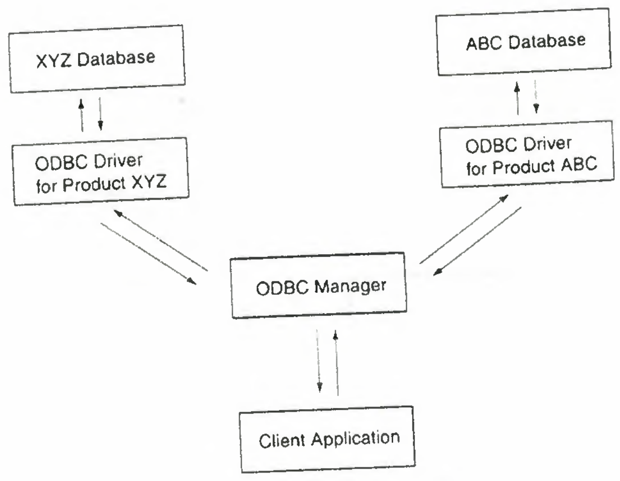


Figure 2.5 ODBC standard models.

NOTE

The differences among individual database packages are handled by configuring the driver options for each database. For example, Microsoft SQL Server implements a security requirement that forces each database user to log in with a name and password before manipulating a database. The ODBC driver for MS-SQL Server allows these parameters to be set. Microsoft Access, on the other hand, has no such requirement, so its ODBC driver does not have that set of parameters.

In practice, ODBC is much more relevant to the Windows platform than Unix or Macintosh. Virtually every Windows database is ODBC-compliant. Many of the larger database vendors have ODBC drivers that allow Windows-based applications to access databases running on a Unix database server, but some do not. Development tools for accessing ODBC-compliant databases are also few and far between for platforms other than Windows. This is changing as Microsoft and various Unix vendors merge their standards technologies; but currently, ODBC is mainly a Windows standard.

Figure 2.6 shows a schematic diagram of the role that these products play and the technologies that can be used to make the various connections among the components.

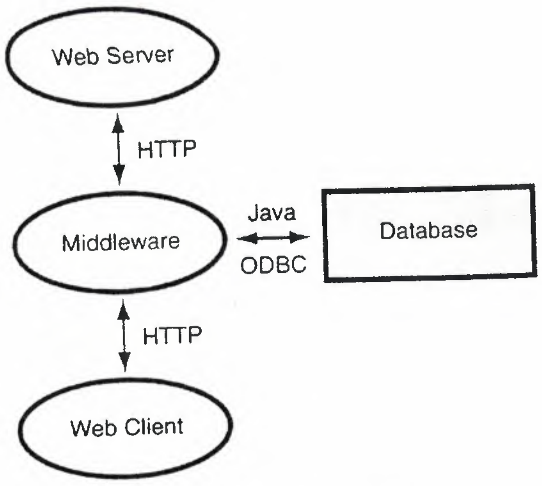


Figure 2.6 Schematic of database middleware solutions.

A more detailed explanation of the steps involved in retrieving data from the database is:

1. The web client makes a request using some sort of form or hypertext link (like a button).
2. The request is sent to the web server through HTTP.
3. The web server receives the request and passes it to the middleware through either CGI or the server's native API.
4. The middleware processes the request, formulates the appropriate SQL commands, and passes it to the database server using ODBC or to its own database server as appropriate.
5. The database server receives the SQL request through ODBC and translates it into native instructions.
6. The database server receives the requested data from the database and sends it back to the middleware.
7. The middleware formats or processes the returned data into some format appropriate to the Web and sends it to the web server.
8. The web server returns the data from the database to the web client.

This is a much more complex process than the typical request for a web page from the server, which takes less than half that many steps!

The advantage of this approach is that any ODBC-compliant can be plugged into the web application. In fact, one application could access data stored in several different database systems and combine it. The user has no way of knowing how many different databases are being accessed nor what kind of databases they are. The main problem with this method is that a number of these steps can become speed and bandwidth bottlenecks. It is also impossible to further tweak the speed of the individual components in any significant way.

In some cases, vendors have addressed the need for web applications by creating entirely new web server software from scratch to address shortcomings in the existing servers. The desire to access databases has been one of the primary forces driving these types of developments. Figure 2.7 shows the architecture of a server that integrates database and web server functionality.

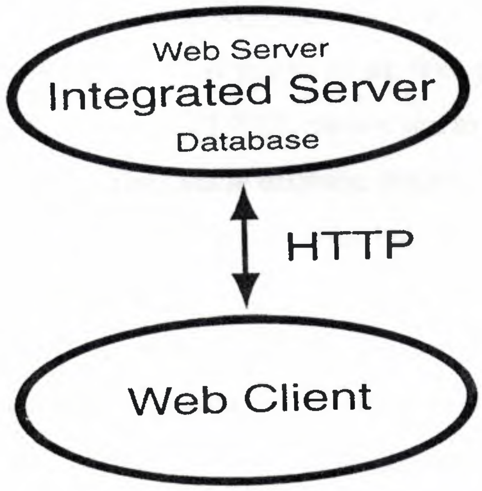


Figure 2.7 Schematic of an integrated web database server.

This scheme makes the server processing more straightforward. The general sequence of events is:

1. The web client makes a request using some sort of form or hypertext link (like a button).
2. The request is sent to the integrated server through HTTP.
3. The integrated server receives the request and translates data requests into native instructions for the database (or queries an internal database).
4. The integrated server receives the requested data from the database and sends it back to the web client as HTML.

This scenario is essentially identical to the normal web page request scheme. It allows the vendor to make any number of speed and throughput tweaks, because there are no standards bodies to satisfy. But creating an integrated server from scratch is a time- and labor-intensive process that leads to much higher costs for this sort of tool.

It's also less likely that any ODBC database can be plugged in since the server is optimized for specific databases or even a proprietary integrated database.

2.4 Summery of the more common DB & DB tools

2.4.1 Databases Available

This section lists some of the more common database solutions that are widely used on the Web.

Oracle

Oracle is the largest database developer in the world, providing databases for Windows NT and various UNIX flavors. Oracle has created its own set of tools (mainly PL/SQL, in conjunction with the Oracle Web Agent). These tools, coupled with the Oracle Web server, allow you to create Web pages with little effort using information stored in the database. PL/SQL allows you to form stored procedures that help speed the database query. The Oracle database engine is a good choice for large businesses that handle large amounts of information but, of course, you're going to pay for that.

Sybase

Sybase System 11 is a SQL database product that has tools for a dynamic Web page production. A product by Powersoft, the NetImpact Studio, integrates with Sybase to provide a rich set of tools to help anyone create dynamic HTML documents. The NetImpact Studio consists of an HTML browser/editor accompanied by a personal web server. They allow you to create pages using a WYSIWYG interface. The Studio also comes with a Web database, support for JavaScript, and support for connecting to application servers.

NetImpact can be used in conjunction with PowerBuilder, an application used to create plug-ins and ActiveX components. It also can be used to complement Optima++, which is used to create plug-ins and supports Java applet creation.

Sybase can also be used with web.sql to create CGI and NSAPI (Netscape Server Application Programming Interface) applications that access the Sybase database server using Perl. Sybase is available for Windows NT, and UNIX.

mSQL

mSQL is a middle-sized SQL database server for UNIX that is much more affordable than the commercial SQL servers available on the market. Written by David Hughes, it was created to allow users to experiment with SQL and SQL databases.

Version 1.0.16 is free for non-commercial use, but you have to pay for individual and commercial use.

Illustra

Illustra, owned by Informix, is the commercial version of the Berkeley's Postgres. Available for both Windows NT and UNIX, Illustra uses an ORDBMS, or Object-Relational Database Management System. By using ORDBMS, queries are performed at very quick speeds. Illustra also uses DataBlade modules that help perform and speed queries. The Web Datablade module version 2.2 allows the incorporation of your data on the Web with reduced effort.

Microsoft SQL

Microsoft released its own SQL database server as a part of its Windows NT Back Office Suite. Microsoft is trying to compete with Oracle and Sybase. It has released the server , but you must also buy the SQL Server Internet Connector. These two products allow you to provide unlimited access to the server from the Web.

Postgres95

Postgres95 is a SQL database server developed by the University of California at Berkeley for use on UNIX systems. Older versions of Postgres are also available but no longer supported.

Ingres

Ingres (Interactive Graphics Retrieval System) comes in both a commercial and public domain version. Berkeley originally developed Ingres to work with graphics in a database environment, but the school no longer supports the public domain version. You can still find it on the university's Web site.

Ingres uses the QUEL query language as well as SQL. QUEL is a superset of the original SQL, making Ingres even more powerful. The public domain version is available for UNIX systems.

Computer Associates owns the commercial version of Ingres, called OpenIngres. This version is quite robust and capable of managing virtually any database application. The commercial version is available for UNIX, VMS, and Windows NT.

FoxPro

Microsoft's Visual FoxPro has been a favorite for Web programmers, mostly because of its long-time standing in the database community, as well as its third-party support. FoxPro is an Xbase database system that is widely used for smaller business and personal database applications. FoxPro is also available for most Windows platforms.

Microsoft Access

Microsoft Access is a relational database management system that is part of the Microsoft Office suite. Microsoft Access can be used to create HTML documents based on the information stored in the Access database with the help of Microsoft's Internet Assistant or with the use of Microsoft's Active Server Pages (ASP). Microsoft's Internet Assistant is an add-on available free of charge for Access users. Using Microsoft's ASP technology requires the use of MS Information Server with ASP installed. Microsoft Access can also support ActiveX controls, which make Access even more powerful when used with the Microsoft Internet Explorer.

Side-by-Side Comparison

Choosing a database to suit an organization's needs is difficult, and should be carefully planned. It's quite difficult to tell which database would best suit the needs without spending a bit of time with a company and seeing how that company operates. Even so, Table 2.5 might help narrow down choices.

Table 2.5 A Comparison of Some of the Most Widely Used Databases on the Web

Database	Platforms	Suggested Use
Oracle	UNIX, NT	Large business
Sybase	UNIX, NT	*Large business*
MSQL	UNIX	Personal, small business
Illustra	UNIX, NT	Medium to large business
MS SQL	NT	Medium to large business
Postgres95	UNIX	Personal and small to medium business
Ingres	UNIX, NT	Small to large business
Foxpro	Windows Macintosh	Small to medium business
MS Access	Windows	Personal and small to medium business

2.4.2 Database Tools

Just as there are multiple databases available, there are also multiple methods of integrating the database with the World Wide Web. What tools should be used depends heavily on what platform the database resides, the knowledge of programming, and the programming language skills. In the following section, a few of the most common tools that make accessing databases easy for Web developers.

PHP/FI

PHP/FI was developed by Rasmus Lerdorf, who needed to create a script that enable him to log visitors on to his page. The script replaced a few smaller ones that were creating a load on Lerdorf's system. This script became PHP, which is an initialization for Rasmus' Personal Home Page tools. Lerdorf later wrote a script that enabled him to embed commands within an HTML document to access a SQL database. This script acted as a forms interpreter (hence the name FI), which made it easier to create forms using a database. These two scripts have since been combined into one complete package called PHP/FI.

PHP/FI grew into a small language that enables developers to add commands within their HTML pages instead of running multiple smaller scripts to do the same thing. PHP/FI is actually a CGI program written in C that can be compiled to work on any UNIX system. The embedded commands are parsed by the PHP/FI script, which then prints the results through another HTML document. Unlike using JavaScript to access a database, PHP/FI is browser-independent because the script is processed through the PHP/FI executable on the server.

PHP/FI can be used to integrate mSQL, along with Postgres95, to create dynamic HTML documents. It's fairly easy to use and quite versatile.

Cold Fusion

Allaire created Cold Fusion as a system that enables you to write scripts within an HTML. Cold Fusion, a database interface, processes the scripts and then returns information within the HTML text in the script. Allaire wrote Cold Fusion to work with just about every Web server available for *Windows NT and integrates with just* about every SQL engine--including those database servers available on UNIX machines (if a 32-bit ODBC driver exists). A version for Sun Solaris has also been recently released.

Cold Fusion works by processing a form, created by you, that sends a request to the Web server. The server starts Cold Fusion and sends the information the visitor entered to Cold Fusion engine, which is used to call a template file. After reading the information the visitor entered, Cold Fusion processes that information according to the template's instructions. Next, it returns an automatically generated HTML document to the visitor.

w3-mSQL

w3-mSQL was created by David Hughes, the creator of mSQL, to simplify accessing an mSQL database from within your Web pages. w3-mSQL works as a CGI script which is used to parse your Web pages. The script reads your HTML document, performs any queries required, and sends the result to the server and then on to your site's visitor. w3-mSQL is much like a smaller-scale PHP/FI; it makes it easy for you to create Web documents that contain information based on what is in your database.

MsqlPerl

MsqlPerl is a Perl interface to the mSQL database server. Written by Andreas Koenig, it utilizes the mSQL API and allows you to create CGI scripts in Perl, complete with all the SQL commands available to mSQL.

MsqlJava

MsqlJava is an API that allows you to create applets that can access an mSQL database server. The package has been compiled with the Java Developer's Kit version 1.0 and tested using Netscape 3.0.

WDB

WDB is a suite of Perl scripts that helps you create applications that allow you to integrate SQL databases with the World Wide Web. WDB provides support for Sybase, Informix, and mSQL databases, but has been used with other database products as well.

WDB uses what Bo Frese Rasmussen calls "form definition files," which describes how the information retrieved from the database should be displayed on the visitor's web browser. WDL automatically creates forms on-the-fly that allow the visitor to query the database. This saves you a lot of the work preparing a script to query a database. The user submits the query and WDB performs a set of conversions, or links, so the visitor can perform additional queries by clicking one of the links.

Web/Genera

Web/Genera is a software toolset used to integrate Sybase databases with HTML documents. Web/Genera can be used to retrofit a Web front end to an existing Sybase database, or it can be used to create a new one. When using Web/Genera, you are required to write a schema for the Sybase database indicating what fields are to be displayed, what type of data they contain, what column they are stored in, and how you want the output of a query formatted. Next, Web/Genera processes the specifications, queries the database, and formats an HTML document. Web/Genera also supports form-based queries and whole-database formatting that turns into text and HTML.
Web/Genera's main component is a program called symfmt, which extracts objects from Sybase databases based on your schema. After the schema is written, compile the schema by using a program, called sch2sql, which creates the SQL procedures that extract the objects from the database.
After you have compiled the schema, you can retrieve information from the database using URLs. When you click a link, the object requested is dynamically loaded from the Sybase database, formatted as HTML, and then displayed to the visitor.

Web/Genera was written by Stanley Letovsky and others for UNIX.

MORE

MORE is an acronym for Multimedia Oriented Repository Environment and was developed by the Repository Based Software Engineering Program (RBSE).
MORE is a set of application programs that operate in conjunction with a Web server to provide access to a relational database. It was designed to allow a visitor access to the database using a set of CGI scripts written in C. It was also designed so that a consistent user interface can be used to work with a large number of servers, allowing a query to check information on multiple machines. This expands the query and gathers a large amount of information.

DBI

DBI's founder, Tim Bunce, wanted to provide a consistent programming interface to a wide variety of databases using Perl. Since the beginning, others have joined in to help build DBI so that it can support a wide variety of databases through the use of a Database Driver, or DBD. The DBD is simply the driver that works as a translator between the database server and DBI. A programmer only has to deal with one specification, and the drivers use the appropriate method to access any given database.

DBGateway

DBGateway is a 32-bit Visual Basic WinCGI application that runs on a Windows NT machine as a service that provides World Wide Web access to Microsoft Access and FoxPro databases. It is being developed as part of the Flexible Computer Integrated Manufacturing (FCIM) project. DBGateway is a gateway between your CGI applications and the database servers. Because your CGI scripts only communicate with the Database Gateway, you only need to be concerned with programming for the gateway instead of each individual database server. This provides two advantages--programming a query is much easier because the gateway handles the communication with the database, and scripts can be easily ported to different database systems.

The gateway allows a visitor to the site to submit a form that is sent to the server. The server hands the request to the gateway, which decodes the information and builds a query forming the result based on a template, or it can send the query's result raw.

Microsoft's Visual InterDev

Visual InterDev is a visual interface in which you can create web applications that easily integrate with various databases. Visual InterDev is a graphical environment that allows you to create Active Server Pages (ASP). It comes with a full set of tools to add a whole range of HTML tags and attributes while allowing you to do so with VBScript or Jscript.

Chapter 3 Active server page (ASP)

Active Server Pages (ASPs) is Microsoft's answer to CGI scripting. Using Microsoft's ASP technology, you can create scripts using Visual Basic Script, Java, Jscript, or even PerlScript without a need to compile your code beforehand.

Active Server Pages is a very powerful and yet easy-to-learn server-side scripting environment. Active Server Pages comes with Internet Information Server (IIS) for Windows NT Server, and with Personal Web Server for Windows NT Workstation and Windows 98. This environment enables you to create a Web site that is dynamic, fast, and interactive without requiring you to worry about the capabilities of your clients' browsers, which you must do if you rely on client-side scripting like client-side JavaScript or client-side Visual Basic Script (VBScript).

Using Microsoft's Active Server Pages

When a Web browser contacts your server, the browser uses the HyperText Transfer Protocol (HTTP) to request some resource by name. A typical request is

GET /index.html HTTP/1.0

This request tells the server that the browser wants the Web page stored in the server's document root (/) whose name is index.html. The message also notifies the server that the client is using version 1.0 of HTTP.

In most cases the server's job is simple--it locates the requested page and sends it back to the client. If the filename ends in a special suffix, however, the server may have to perform additional processing. For example, the Webmaster may turn on server-side includes; if the filename ends in SHTML the server examines the file for special commands and then executes each command. The output of each command is inserted into the file; the finished file is then sent back to the client.

On the Microsoft Internet Information Server (IIS) and its smaller cousin, the Personal Web Server, the server checks to see if the filename ends in ASP. If it does, the server examines the file looking for scripts and then runs each script. The server inserts the output of each script into the file and sends the finished file back to the client.

CAUTION: Don't just change the extension of every file on your server from HTM to ASP. Every time the server has to check a file for scripts, it takes a bit of processing time. If you only use ASP when a file contains a script, the load on the server is small. If you force the server to check every page, the load adds up and can reduce the performance of your server.

Because ASP runs on the server, this technology does not depend on the end user's browser or platform. As long as the ASP scripts generate valid HTML, the resulting page should work as well on a Macintosh running Navigator 4.0 as it does on a Windows 95 machine running Microsoft Internet Explorer 3.02.

Getting Input from HTML Forms

Virtually all Web browsers support HTML forms, so ASP applications can use HTML forms to communicate with almost anyone browsing your Web site. HTML forms are used to transfer data entered by users to an ASP application.

There are primarily two ways in which HTML forms send data to the Web server:

- Using the GET method.

(This is specified in the <FORM METHOD="GET" ... HTML tag of the data entry HTML form.)

- Using the POST method.

(This is specified in the <FORM METHOD="POST" ... HTML tag of the data entry form.)

NOTE

Many Web designers prefer to use POST rather than GET to send form data back to the server. The advantage of POST is that there is no limit to the number of characters that can be sent back, while some combinations of software truncate GET strings

Accessing a Database

One of the most useful things you can do with an ASP script is give the user access to a database.

Along with ASP technology, Microsoft provides a component called the ActiveX Database Object (ADO), which is based on Microsoft's Open Database Connectivity (ODBC) standard. The ADO provides you with access to any OLE/DB or ODBC-compatible data source for use within your scripts.

ADO and FrontPage 2000

FrontPage 2000 now manages database connections through ADO (Microsoft® ActiveX® Data Objects). There is no need to manage connections through the ODBC control panel, it is all done through FrontPage 2000. The implementation of a Global.asa file allows access to the database by setting up the server object and accessing the data source all with in the web, using ASP. FrontPage 2000 is not reliant on system DSNs.

ADO is designed as an easy-to-use application level interface to Microsoft's newest and most powerful data access paradigm, OLE DB. OLE DB provides high-performance access to any data source, including relational and non-relational databases, e-mail and file systems, text and graphics, custom business objects, and more. ADO is implemented with a small footprint, minimal network traffic in key Internet scenarios, and a minimal number of layers between the front-end and data source-all to provide a lightweight, high-performance interface. ADO is easy to use because it is called using a familiar metaphor-the OLE Automation interface, available from just about any tool and language on the market today. And since ADO was designed to combine the best features of and eventually replace RDO and DAO, it uses similar conventions with simplified semantics to make it easy to learn for today's developers.

What are the ActiveX Data Objects (ADO)?

ActiveX Data Objects are a language-neutral object model, that exposes data raised by an underlying OLE DB provider. The most commonly used OLE DB provider is the OLE DB provider for ODBC drivers, which exposes ODBC data sources to ADO. FrontPage 2000 uses these data objects to access local and remote data sources.

The ADO objects provide you with the fastest, easiest, and most productive means for accessing all kinds of data sources. The ADO model strives to expose everything that the underlying data provider can do, while still adding value by giving you shortcuts for common

---

NOTE

Microsoft recommends that users migrate to ADOs, but it will continue to support other methods into the near future.

Chapter 4 Some Views From The Project

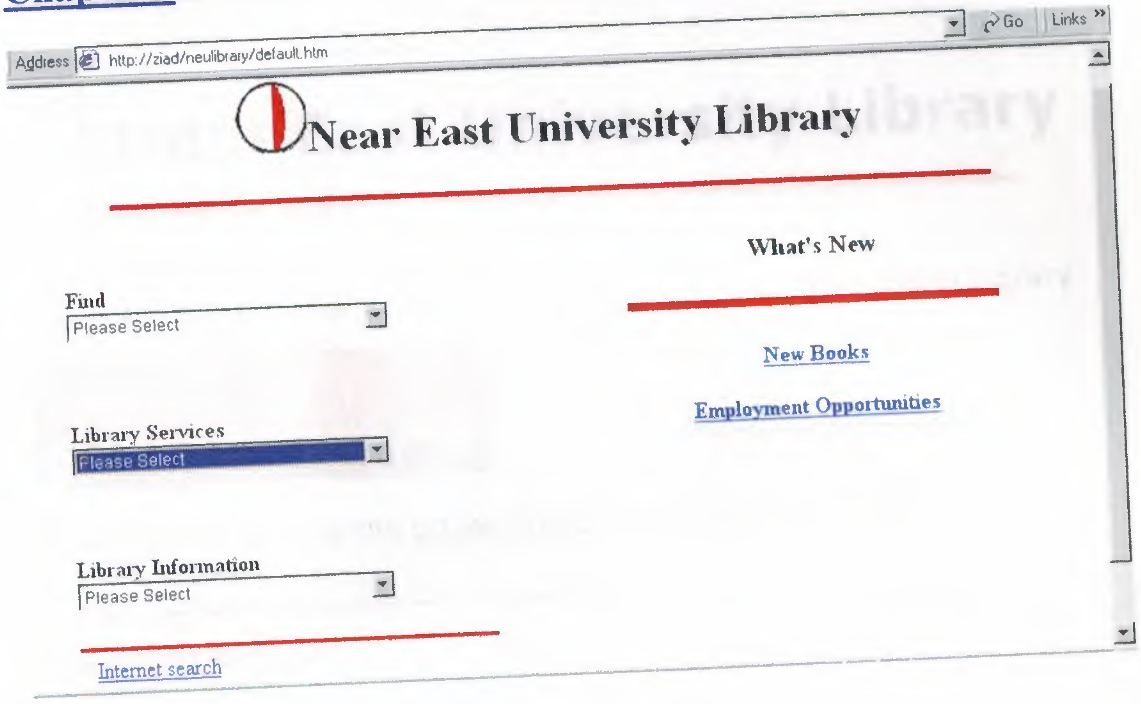


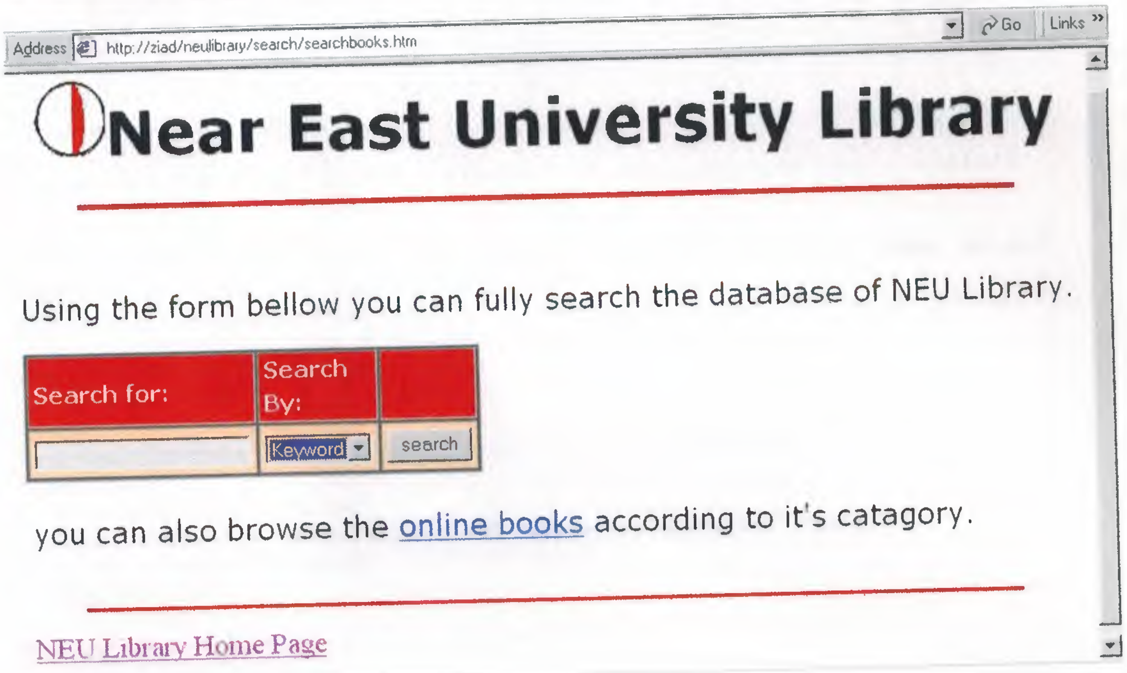
Figure 4.1 The NEULibrary Hompage

As you can see in Figure 4.1 I have designed the main page (Homepage) to contain three main sections represented by three drop-down menus. Each contains a number of links to its classified information. The drop-down menus are classified as follows:

- Find
  - Search the Library Catalog for a Book.
  - Browse the Online Books By Category.
- Library Services
  - Reserve a Book.
- Library Information
  - 10 things a new student should know.
  - Library Guides.
  - General Information.
  - Frequently-Asked Questions.
  - Borrowing at NEU Library.

Beside the three menus I have placed a links to (what's new) and (internet search) for a specific reasons. When a user browses a website once a day in average he/she usually doesn't want to spend time looking for what' new so psychologically putting that links directly in front of him/her you will 1st save his/her time, 2nd insure that he/she will check it. The (internet search) is for breaking the discipline rule and it is also assumed to be frequently accessed page by the students. Why, this discussed later in this chapter.

The Find drop-down menu


Address http://ziad/neulibrary/search/searchbooks.htm

Near East University Library

Using the form bellow you can fully search the database of NEU Library.

Search for:	Search By:	
	Keyword	search

you can also browse the online books according to it's catagory.

NEU Library Home Page

Figure 4.2 the Library catalog search page

Using the page shown in Figure 4.2 the user can search the database of the library. Although it is not an ASP page but the search form is containing a script that will pass the entered text as a parameter to the specified ASP page according to option selected form the (search By) drop-down menu. The script of the search form is shown in Figure 4.3.

```
<script language="JavaScript">
<!--

//
// Script by Ziad Abdelhamied for the graduation project
//

function startSearch(){
searchString = document.searchForm.searchText.value;
if(searchString != ""){
searchEngine = document.searchForm.whichEngine.selectedIndex + 1;
finalSearchString = "";

if(searchEngine == 1){
finalSearchString = "keyresult.asp?keyword=" + searchString;
}
if(searchEngine == 2){
finalSearchString = "titleresult.asp?title=" + searchString ;
}
if(searchEngine == 3){
finalSearchString = "isbnresult.asp?isbnnumber=" + searchString;
}
location.href = finalSearchString;
}
}

// -->
</script>
<form name="searchForm">

<table width=320 border="1" cellpadding=3 cellspacing=2 bgcolor=444444>

<tr>
<td bgcolor=#990000><font color="#FFFFFF" size="1" face="Verdana, Arial,
sans-serif"><b>Search for:</b></font>
<td bgcolor=#990000><font size=1 face="Verdana, Arial, sans-serif"
color="#FFFFFF"><b>Search By:</b></font>
<td bgcolor=#990000>&nbsp;

<tr>
<td bgcolor=navajowhite><input style="background: dddddd" name="searchText"
type="text">
<td bgcolor=navajowhite>
<select style="background: dddddd" name="whichEngine" size="1">
<option selected>Keyword
<option>Title
<option>ISBN</select>
<td bgcolor=navajowhite><input type="button" value="search"
onClick="startSearch()">

</table>
</form>
```

Figure 4.3 the code of the Library catalog search page

While the above script pass the query data to an ASP page Figure 4.4 show part of the script in (keyresult.asp) one of the pages that access the database and process the query request.

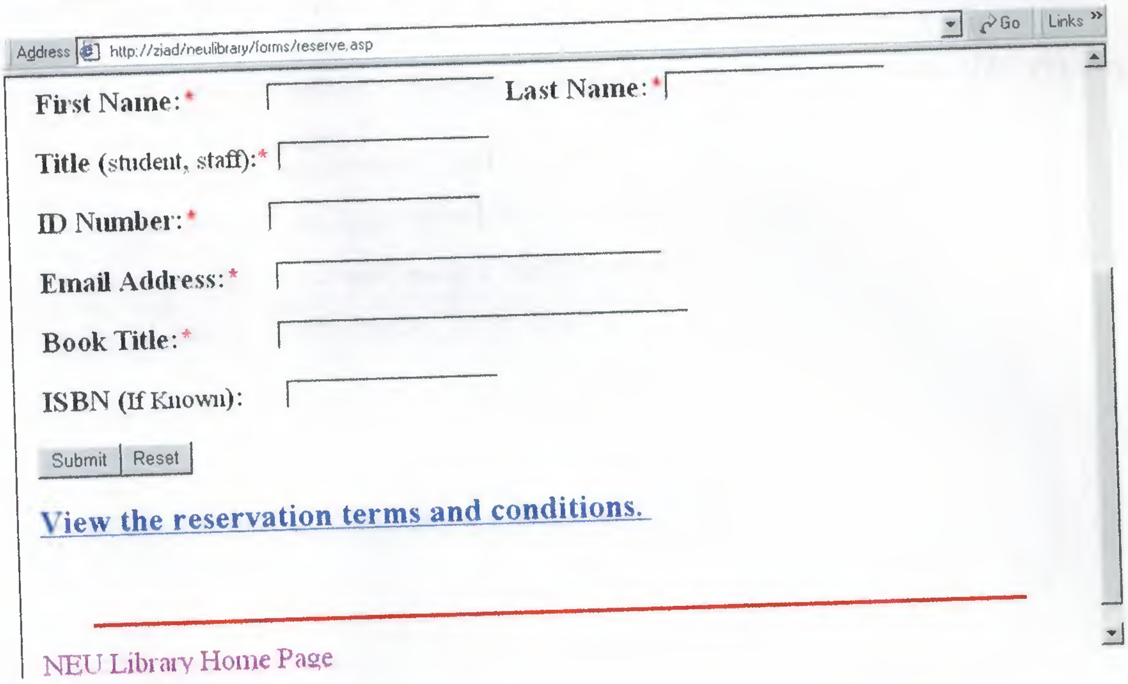
```
<%
fp_sQry="SELECT * FROM Books WHERE (keyword LIKE '%::keyword::%')"
fp_sDefault="keyword="
fp_sNoRecords="No book is found with this criteria please go back and change
your query."
fp_sDataConn="onlinebook"
fp_iMaxRecords=256
fp_iCommandType=1
fp_iPageSize=10
fp_fTableFormat=False
fp_fMenuFormat=False
fp_sMenuChoice=""
fp_sMenuValue=""
fp_iDisplayCols=11
fp_fCustomQuery=False
BOTID=0
fp_iRegion=BOTID
%>
```

Figure 4.4 part of keyresult.asp

Library Services

Reserve a Book

If the user searched for a book and that book is not available online the user will have an option link to the reservation form. After he click on the link the reservation form will be displayed with the (Book Name) and the (ISBN) boxes already fielded. That is simple done by passing the data to the reservation form using the post method and placing a request for that data in the default value of the boxes. Figure 4.5 shows the reservation form while Figure 4.6 shows how the initial value request is assigned to the (Book Name) field.



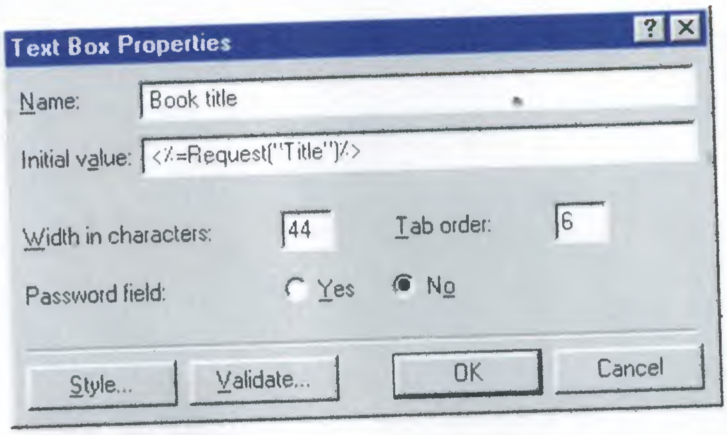
Address http://ziad/neulibrary/forms/reserve.asp

First Name:*	
Last Name:*	
Title (student, staff):*	
ID Number:*	
Email Address:*	
Book Title:*	
ISBN (If Known):	

View the reservation terms and conditions.

NEU Library Home Page

Figure 4.5 the reservation page



Text Box Properties

Name:	Book title
Initial value:	<%=Request("Title")%>
Width in characters:	44
Tab order:	6
Password field:	Yes [✓ No]

Figure 4.6 the initial value for the book title filed

The Internet search page

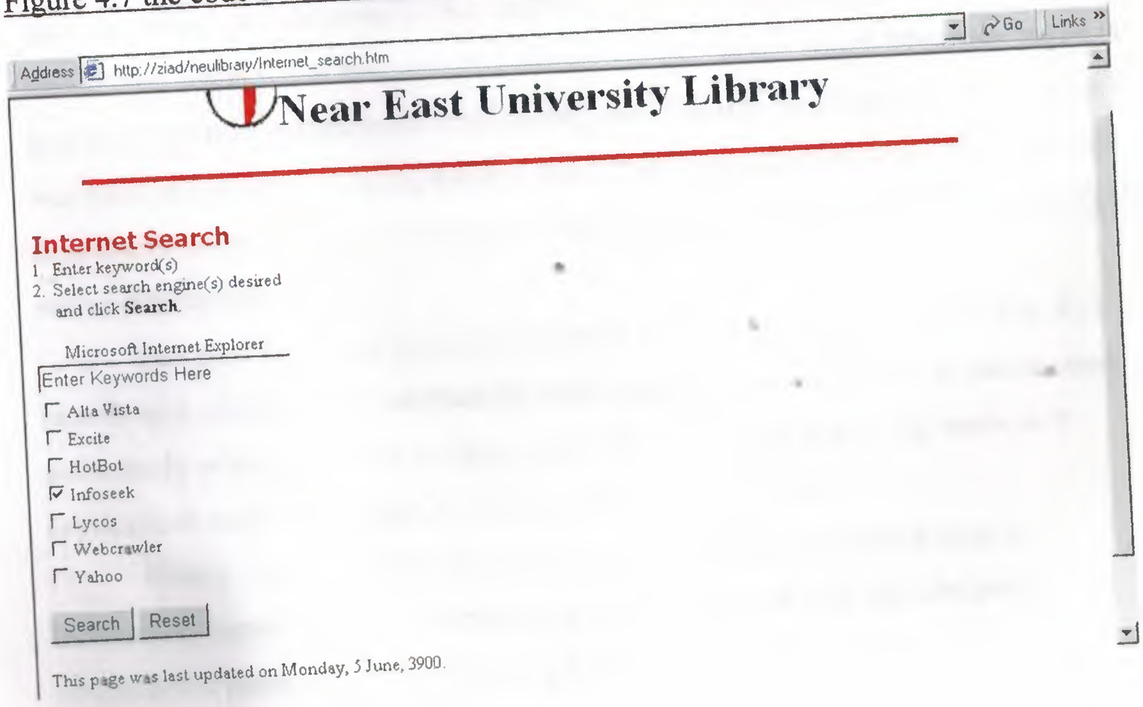
This page is designed as a service to reduce the time spent jumping from one search engine to another. It allow the user to search using more than one engine by only one click. Figure 4.7 & 4.8 show the page code and its layout.

```
<HTML><HEAD><TITLE>Internet search</TITLE>
<META content="text/html; charset=iso-8859-1" http-equiv=Content-Type>
<META content="Microsoft FrontPage 4.0" name=GENERATOR>
<META content="Chip Smith, Red Rose Int'l" name=creator>
<META content="Javascript Multiple Engine Internet Search Page"
name=description>
<META content="Internet Search, Multiple Search Engines, Javascript Search"
name=subject>
<META
content="Search, Multiple Engines, Multiple Search Engines, Javascript Search,
Internet Search"
name=keywords><NOSCRIPT>
<meta name="Microsoft Border" content="b">
</HEAD>
<BODY aLink=#800000 bgColor=#FFFFFF link=#800000 vLink=#800000>
<H2>&nbsp;</H2>
<H2><FONT color=red>Please enable JavaScript in your browser preferences and
then Reload this page!!! </FONT></H2></NOSCRIPT>
<SCRIPT language=javascript>
<!--
//
// Script by Ziad Abdelhamied for the graduation project
//

function netsearch(formname)

{
   var a1
   var a2
   var b1
   var b2
   var c1
   var c2
   var d1
   var d2
   var e1
   var e2
   var f1
   var f2
   var g1
   var g2
   var plus
   var TEXT
   var noENGINE
   var haveTEXT


    TEXT=formname.TEXT.value;

    noEngine=true;
    haveTEXT=true;
    plus=""

    if (TEXT=="")
    {
    alert("Please type in some text!")
```

```
  haveTEXT=false
  }
     else
     {
     for (var i=0; i < TEXT.length; i++)
         {
          if (TEXT.charAt(i)==" ")
             {
             plus+="%20"
             }
         else
             {
             plus += TEXT.charAt(i)
             }
          }
      }

TEXT=plus


   //ALTAVISTA
   a1=formname.altavista.checked;

   a2="http://www.altavista.digital.com/cgi-
bin/query?pg=q&what=web&fmt=.&q="+TEXT;
      if (a1)
         {
         noEngine=false
         if (haveTEXT)
           {
           newWindow=window.open(a2,
"av","toolbar,location,directories,status,menubar,scrollbars,resizable=1")
           }
        }

    //EXCITE
    b1=formname.excite.checked;
    b2="http://www.excite.com/search.gw?trace=a&search="+TEXT;
      if (b1)
         {
         noEngine=false
         if (haveTEXT)
           {
           newWindow=window.open(b2,
"e","toolbar,location,directories,status,menubar,scrollbars,resizable=1")
           }
        }

    //HOTBOT
    c1=formname.hotbot.checked;


c2="http://www.search.hotbot.com/hResult.html?SM=MC&MT="+TEXT+"&DV=7&RG=.com&DC=10
&DE=2&OPs=MDRTP&_v=2&DU=days&SW=web&search.x=23&search.y=8";
      if (c1)
         {
         noEngine=false
         if (haveTEXT)
           {
           newWindow=window.open(c2,
"h","toolbar,location,directories,status,menubar,scrollbars,resizable=1")
           }
        }

    //INFOSEEK
    d1=formname.infoseek.checked;
  d2="http://www.infoseek.com/Titles?qt="+TEXT+"&col=WW&sv=IS&lk=noframes&nh=10";
    if (d1)
      {
      noEngine=false
      if (haveTEXT)
        {
        newWindow=window.open(d2, "i",
"toolbar,location,directories,status,menubar,scrollbars,resizable=1")
        }
      }

   //LYCOS
   e1=formname.lycos.checked;

   e2="http://www.lycos.com/cgi-
bin/pursuit?query="+TEXT+"&matchmode=and&cat=lycos&x=33&y=10";
    if (e1)
      {
      noEngine=false
      if (haveTEXT)
        {
        newWindow=window.open(e2,
"l","toolbar,location,directories,status,menubar,scrollbars,resizable=1")
        }
      }

   //WEBCRAWLER
   f1=formname.webcrawler.checked;
   f2="http://www.webcrawler.com/cgi-bin/WebQuery?searchText="+TEXT;
    if (f1)
      {
      noEngine=false
      if (haveTEXT)
        {
        newWindow=window.open(f2,
"w","toolbar,location,directories,status,menubar,scrollbars,resizable=1")
        }
      }

   //YAHOO
   g1=formname.yahoo.checked;
   g2="http://search.yahoo.com/bin/search?p="+TEXT;
    if (g1)
      {
      noEngine=false
      if (haveTEXT)
        {
        newWindow=window.open(g2,
"y","toolbar,location,directories,status,menubar,scrollbars,resizable=1")
        }
      }

   //noENGINE
    if (noEngine)
      {
      alert("Please select a search engine!")
      }

 // END Netsearch

 }

 function getPath(url) {

         lastSlash = url.lastIndexOf("/")

         return url.substring(0, lastSlash + 1)
```

```
}

// -->

</SCRIPT>
<!--
<H2><font color="red">
This search page requires JavaScript to run. Please hit your
<strong><i>BACK</i></strong> button and follow the link at the bottom of the page
to get the latest version of Internet Explorer.<br> Thank you.
</font></H2>
-->
  <table border="0" cellpadding="0" cellspacing="4" width="94%">
    <tr>
      <td valign="bottom">
      <p align="center"><font size="6"><b><img border="0"
src="http://ziad/neulibrary/images/neu.gif" align="baseline" width="51"
height="51">Near
      East University Library</b></font></p>
      <hr width="90%" size="4" color="#800000">
      </td>
    </tr>
  </table>
<FORM name=engines onsubmit=netsearch(engines)>
<P><FONT color=#800000 face=Tahoma size=4><STRONG>Internet
Search</STRONG></FONT> <FONT size=2><BR>1.&nbsp; Enter keyword(s) <BR>2.&nbsp;
Select search engine(s) desired <BR>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;and click
<STRONG>Search</STRONG>. <BR><BR><!-- //    <input type="text" size="25"
maxlength="200" name="TEXT" value="Enter Keywords Here"
onClick="form.TEXT.select()">
// -->
<SCRIPT language=javascript>
if (navigator.appName=="Netscape") {
   document.write("<font size='3'>")

document.write("&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
&nbsp;&nbsp;&nbsp;&nbsp;" + navigator.appName)
   document.write("<br>");
   document.write("<input type='text' size='25' maxlength='200' name='TEXT'
value='Enter Keywords Here' onFocus='form.TEXT.select()'>")
   document.write("</font>")
   }
else {
   document.write("&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;" +
navigator.appName);
   document.write("<br>");
   document.write("<input type='text' size='25' maxlength='200' name='TEXT'
value='Enter Keywords Here' onClick='form.TEXT.select()'> ");
    }
</SCRIPT>
<BR><INPUT name=altavista type=checkbox>Alta Vista<BR><INPUT name=excite
type=checkbox>Excite<BR><INPUT name=hotbot type=checkbox>HotBot<BR><INPUT
CHECKED name=infoseek type=checkbox>Infoseek<BR><INPUT name=lycos
type=checkbox>Lycos<BR><INPUT name=webcrawler type=checkbox>Webcrawler<BR><INPUT
name=yahoo type=checkbox>Yahoo<BR><BR><INPUT type=submit value=Search> <INPUT
type=reset value=Reset> <BR></FONT></P></FORM><FONT size=2>
<SCRIPT language=JavaScript>
<!--


   function initArray() {
      this.length = initArray.arguments.length

      for (var i = 0; i < this.length; i++)
    this[i+1] = initArray.arguments[i]

 }

 var DOWArray = new initArray("Sunday","Monday","Tuesday","Wednesday",
                              "Thursday","Friday","Saturday");
 var MOYArray = new initArray("January","February","March","April",
                              "May","June","July","August","September",
                              "October","November","December");

  var LastModDate = new Date(document.lastModified);
  document.write("This page was last updated on ");
  document.write(DOWArray[(LastModDate.getDay()+1)],", ");

  document.write(LastModDate.getDate()," ");
  document.write(MOYArray[(LastModDate.getMonth()+1)],",
",(LastModDate.getYear()+1900));

  document.write(".");

// -->
</SCRIPT>
</FONT></BODY></HTML>
```

Figure 4.7 the code of the internet search page



Address
http://ziad/neulibrary/Internet_search.htm
Go
Links
Near East University Library
Internet Search
1. Enter keyword(s)
2. Select search engine(s) desired and click Search.
Microsoft Internet Explorer
Enter Keywords Here
Alta Vista
Excite
HotBot
[✓ Infoseek]
Lycos
Webcrawler
Yahoo
Search
Reset

This page was last updated on Monday, 5 June, 3900.

Figure 4.8 the internet search page

Conclusion:

Doing this project I have learned a lot about the technologies that underlie the Web and databases, as well as the technologies that begin to put them together. The web-related technologies that I have studied ranged from HTML to web servers. I had also looked at the structure and format of HTML documents with a quick reference to its parent (SGML) and its future (XML). And also the processing capabilities of the Web on both the client and server sides. The larger part of this project was the technologies underlying databases. The majority of applications in the real world currently use relational databases, so their architecture was covered with more concern.

Another essential database technology is Structured Query Language (SQL), the more-or-less standard grammar for creating, manipulating, and populating relational databases. I have learned the basic SELECT, UPDATE, INSERT, and DELETE commands, as well as the highlights of the rest of the language. It's also extremely important to remember that each database vendor has added proprietary (that is, nonstandard) extensions to SQL that only work on that implementation. Nevertheless, it's useful to have at least a pidgin version of a common database language.

The ODBC standard is another important database technology, especially on Windows-based platforms. This is an abstraction layer that allows database clients and database servers to talk to each other regardless of the intervening platform or vendor issues. The vendor (or a third party) writes an ODBC driver, which translates requests into SQL and then passes them to the server and performs the reverse process on the way back to the client. The net result is that a client can access any ODBC-compliant data source without regard to the native database format (and without even knowing the native database format).

The most important section for me was the web application model. The Web is becoming a common user interface for accessing applications as well as information, particularly where databases are concerned. The technologies that can make web applications work, including CGI, Java, JavaScript and ASP.

Finally although I still can't put a proper definitions for terms such as Information superhighway, Cyberspace and The Virtual World but I defiantly understand it more better after finishing this project.

References:

Books:

- **SE USING HTML 4, 4TH EDITION**
  By: Jerry Honeycutt
- **Sams Teach Yourself Active Server Pages in 24 Hours**
  By: Christoph Wille
- **Running a Perfect Intranet**
  By: Rich Casselberry
- **Sams Teach Yourself Active Server Pages 2.0 in 21 Days**
  By: Sanjaya Hettihewa
- **Choosing a Database for Your Web Site**
  By: John Paul Ashenfelter

Internet:

- **A comprehensive resource including papers, articles, books, partial books, FAQs, tutorials and more for HTML, ASP, DHTML, web design, database technologies and more**
  - http://msdn.microsoft.com
- **more Java scripts & DHTML resources**
  - http://dynamicdrive.com
  - http://wsabstract.com
  - http://www.javascripts.com
- **CGI Resource Index**
  - http://cgi.resourceindex.com
- **FrontPage 2000 Start Page**
  - http://www.microsoft.com/insider/frontpage2000/default.htm

- **Information about database and database tools**
  - **Oracle** http://www.oracle.com/products/tools/WDS/
  - **Sybase** http://www.sybase.com/
  - **mSQL** http://www.Hughes.com.au/
  - **Illustra** http://www.informix.com/
  - **Microsoft SQL** http://www.microsoft.com/sql/.
  - **Postgres95** http://s2k-ftp.CS.Berkeley.EDU:8000/postgres/
  - **Ingres** http://www.naiua.org/
  - **Cold Fusion** http://www.allaire.com/
  - **w3-mSQL** http://hughes.com.au/software/w3-msql.htm
  - **MsqlPerl** ftp://Bond.edu.au/pub/Minerva/msql/Contrib/
  - **WDB** http://arch-http.hq.eso.org/wdb/html/wdb.html
  - **DBI** http://www.hermetica.com/technologia/DBI/
  - **DBGateway** http://fcim1.csdc.com/
  - **Microsoft's Visual InterDev** http://www.microsoft.com/vinterdev/