Chapter 17. Web Database Connectivity

Table of contents

- Objectives
- Introduction
- Context
  - Basic concepts
  - Web-based client-server applications
  - Context summary
- Web database architectures
  - Components of a database application
    - Browser layer
    - Application logic layer
    - Database connection layer
    - Database layer
  - 2-tier client-server architecture
  - 3-tier client-server architecture
- Database gateways
  - Client-side solutions
  - Server-side solutions
- Client-side Web database programming
  - Browser extensions
    - JavaScript
    - Java
    - ActiveX
    - Plug-ins
  - External applications
- Server-side Web database programming
  - CGI (Common Gateway Interface)
    - Advantages and disadvantages of CGI
  - Extended CGI
  - HTTP server APIs and server modules
    - Server vendor modules
    - Advantages of server APIs and modules
  - Important issues
  - Comparison of CGI, server APIs and modules, and FastCGI
  - Proprietary HTTP servers
- Connecting to the database
  - Database API libraries
    - Native database APIs
    - Database-independent APIs: ODBC
    - Benefits of database APIs
    - Shortcomings of database APIs
  - Template-driven packages
    - The approach

        - Benefits of template-driven packages
        - Shortcomings of template-driven packages
    - GUI application builders
        - The approach
        - Benefits of visual tools
        - Shortcomings of visual tools
- Managing state and persistence in Web applications
    - Technical options
    - The URL approach
        - Benefits of the URL approach
        - Shortcomings of the URL approach
    - URL QUERY_STRING
        - Benefits of the hidden fields approach
        - Shortcomings of the hidden fields approach
    - HTTP cookies
        - Benefits of cookies
        - Shortcomings of cookies
    - Important considerations
        - Managing state on the client
        - Managing state on the server
- Security Issues in Web Database Applications
    - Proxy servers
    - Firewalls
    - Digital signatures
    - Digital certificates
    - Kerberos
    - Secure sockets layer (SSL) and secure HTTP (S-HTTP)
    - Java security
    - ActiveX security
- Performance issues in Web database applications
- Discussion topics

Objectives

At the end of this chapter you should be able to:

- Understand the requirements for connecting database systems to the Web.
- Critically compare a number of approaches that might be used to build the Web database connectivity.
- Make recommendations for a given company and specific scenario regarding which of the commonly used mechanisms is likely to be most appropriate, taking into consideration relative cost, security, likely transaction volumes and required performance.

Introduction

In parallel with this chapter, you should read Chapter 29 of Thomas Connolly and Carolyn Begg, “Database Systems A Practical Approach to Design, Implementation, and Management”, (5th edn.).

This chapter introduces you to the exciting topic of combining World Wide Web (WWW) technology with that of databases. It is about bridging a gap between new technologies and ‘old’ in order to achieve what has never been achievable before. The emergence of the WWW is arguably one of the most important technological advances in this century, and since its birth a decade ago, it has changed many people’s lives and had a profound impact on society.

The Web has been expanding at an incredible speed and even while you are reading this, hundreds and thousands of people are getting ‘online’ and hooked to the Web. Reactions to this technology are understandably mixed. People are excited, shocked, confused, puzzled or even angered by it. Whatever your reaction might be, you are being affected and benefiting from it. Without the Web technology, the creation of a global campus would not have been possible. It is fair to say that the WWW is playing and will continue to play an important role (perhaps the most important role) in shaping the future world of technology, business and industry.

The database technology has been around for a long time now, and for many business and government offices, databases systems have already become an essential and integral part of the organisation. Now the new technology has given the ‘old’ a shot in the arm, and the combination of the two creates many exciting opportunities for developing advanced database applications, which will in turn produce additional benefits for the traditional database applications. A multinational company, for example, can create a Web-based database application to enable the effective sharing of information among offices around the world.

As far as database applications are concerned, a key aspect of the WWW technology is that it offers a brand new platform to collect, deliver and disseminate information. Via the Web, a database application can be made available, interactively, to users and organisations anywhere in the world.

In this chapter, we are going to examine the impact that the WWW brings to the ‘traditional’ database technology. We will see how databases can be connected to the Web, and the most effective way of using the new technology to develop database applications. We will also study the most commonly used approaches for creating Web databases, and discuss related issues such as dynamic updating of Web pages inline with the changes in databases, performance, and security concerns.

As the Web contains many large, complex, multimedia documents, the materials covered in this chapter are relevant to the discussion of Object-oriented databases. The reason is that the Object-oriented model is considered the most suitable for the storage, organisation and retrieval of large sets of Web documents. Also, the need to process high volumes of queries and updates over the Web has an important impact on performance considerations. Traditional techniques need to be adapted or even changed to satisfy performance requirements of Web applications. Lastly, the discussion about client-server applications (Chapter 15) is also relevant to this chapter, because Web databases represent a new type of such applications.

Context

Basic concepts

Before we start to discuss Web database applications, we need to clarify a number of related terms and concepts.

Internet: It is a worldwide collection of interconnected computer networks, which belong to various organisations (e.g. educational, business and governments). It is not synonymous to the WWW. The services that are normally available on the Internet include email, real-time communication (e.g. conferencing and chat), news services, and facilities for accessing remote computers to send and receive documents.

The WWW or simply the Web: The WWW comprises software (e.g. Web servers and browsers) and data (e.g. Web sites). It simply represents a (huge) set of information resources and services that live on the Internet. Each Web site consists of a set of Web pages, which typically contain multimedia data (e.g. text, images, sound and video). In addition, a Web page can include hyperlinks to other Web pages which allow users (also called net surfers) to navigate through the Web of information pages.

Intranet: A Web site or group of sites which belongs to an organisation and can only be accessed by members of that organisation. Between the Internet and an intranet, there is an extra layer of software or hardware called a firewall. Its main function is to prevent unauthorised access to a private network (e.g. an intranet) from the Internet.

Extranet: An intranet which allows partial access by authorised users from outside the organisation via the Internet.

HTTP (HyperText Transfer Protocol): The standard protocol for transferring Web pages through the Internet. HTTP defines how clients (i.e. users) and servers (i.e. providers) should communicate.

HTML (HyperText Markup Language): A simple yet powerful language that is commonly used to format documents which are to be published on the Web.

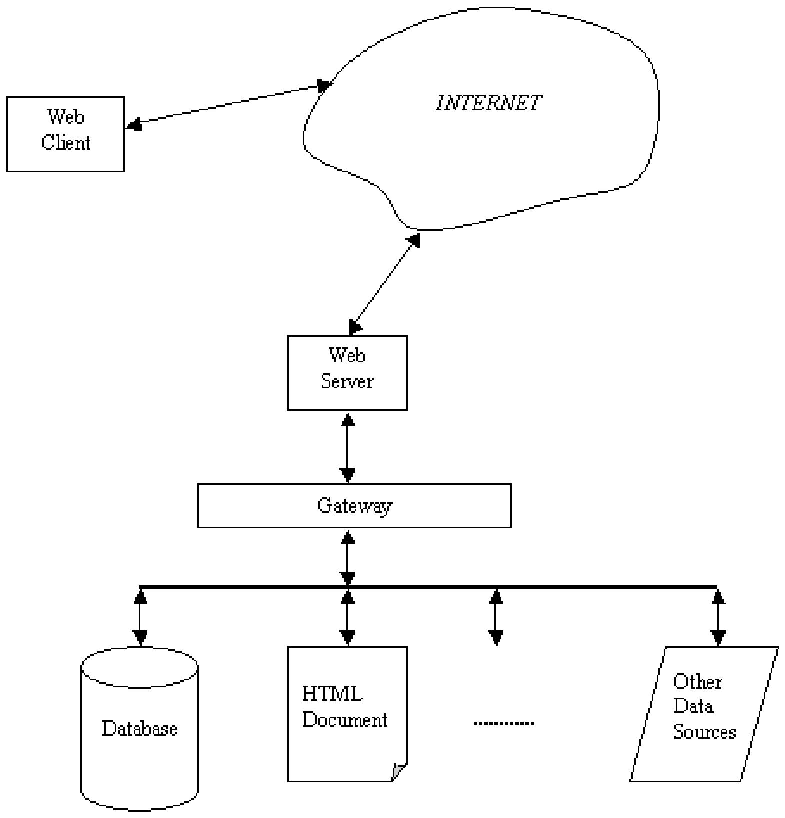
URL (Uniform Resource Locator): A string of alphanumeric characters that represents the location of a resource (e.g. a Web page) on the Internet and how that resource should be accessed.

There are two types of Web pages: static and dynamic.

1. **Static:** An HTML document stored in a file is a typical example of a static Web page. Its contents do not change unless the file itself is changed.
2. **Dynamic:** For a dynamic Web page, its contents are generated each time it is accessed. As a result, a dynamic Web page can respond to user input from the browser by, for example, returning data requested by the completion of a form or returning the result of a database query. A dynamic page can also be customised by and for each user. Once a user has specified some preferences when accessing a particular site or page, the information can be recorded and appropriate responses can be generated according to those preferences.

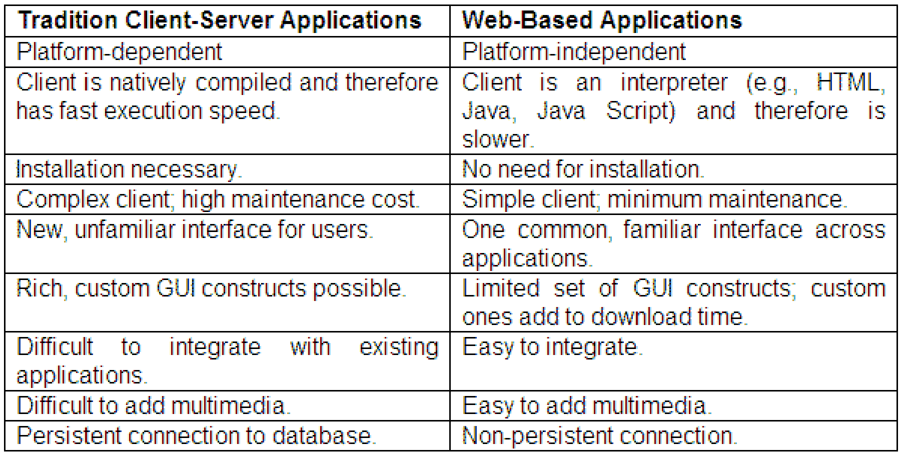
From the above, it can be seen that dynamic Web pages are much more powerful and versatile than static Web pages, and will be a focus for developing Web database applications. When the documents to be published are dynamic, such as those resulting from queries to databases, the appropriate hypertext needs to be generated by the servers. To achieve this, we must write scripts that perform conversions from different data formats into HTML 'on-the-fly'. These scripts also need to recognise and understand the queries performed by clients through HTML forms and the results generated by the DBMS.

In short, a Web database application normally interacts with an existing database, using the Web as a means of connection and having a Web browser or client program on the front end. Typically such applications use HTML forms for collecting user input (from the client); CGI (Common Gateway Interface, to be discussed later) to check and transfer the data from the server; and a script or program which is or calls a database client to submit or retrieve data from the database. The diagram below gives a graphical illustration of such a scenario. More will be discussed in later parts of this chapter.



Web-based client-server applications

As mentioned earlier in the Introduction, Web-based database applications are a new type of client-server application. Some of the traditional client-server database techniques may still be adapted. However, because of the incorporation of the Web technology, there are important differences as set out in the following table.



Tradition Client-Server Applications	Web-Based Applications
Platform-dependent	Platform-independent
Client is natively compiled and therefore has fast execution speed.	Client is an interpreter (e.g., HTML, Java, Java Script) and therefore is slower.
Installation necessary.	No need for installation.
Complex client; high maintenance cost.	Simple client; minimum maintenance.
New, unfamiliar interface for users.	One common, familiar interface across applications.
Rich, custom GUI constructs possible.	Limited set of GUI constructs; custom ones add to download time.
Difficult to integrate with existing applications.	Easy to integrate.
Difficult to add multimedia.	Easy to add multimedia.
Persistent connection to database.	Non-persistent connection.

Platform independence: Web clients are platform-independent and do not require modification to be run on different operating systems. Traditional database clients, on the other hand, require extensive porting efforts to support multiple platforms. This is arguably one of the most compelling reasons for building a Web-based client-server database application.

Interpreted applications: Web applications are written in interpreted languages (e.g. HTML and Java). This has an adverse effect on performance. In many applications, however, this is a price worth paying to gain the advantage of being platform independent. Time-critical applications may not be good candidates to be implemented on the Web.

No need for installation: Another benefit of Web database applications is that the need for installing special software is eliminated on the clients' side. It is pretty safe to assume that the clients have already had a Web browser installed, which is the only piece of software needed for the clients to run the applications.

Simple client: As a client needs just a browser to run a Web-based database application, the potential complications are minimised.

Common interface across applications: Again, because there is no need for specialised software, users have the benefit of using a browser for possibly different applications.

Limited GUI (Graphical User Interface): This is one area in which Web-based database applications may fall short. Highly customised application interfaces and highly interactive clients may not translate well as Web applications. This is because of the HTML limitations. At the moment, HTML forms do not offer an extensive feature set. Although JavaScript language can extend the functionality of HTML-based applications, it is too complex, adds to downloading time, and degrades performance.

Integrate with other applications: Because of the benefit of being platform independent, different applications that adhere to the HTML standard can be integrated without many difficulties.

Non-persistent connection to database: Persistent database connections are highly efficient data channels between a client and the DBMS, and therefore, are ideal for database applications. However, Web-based applications do not have this benefit. A Web-based client maintains its connection to a database only as long as is necessary to retrieve the required data, and then releases it. Thus, Web application developers must address the added overhead for creating new database connections each time a client requires database access.

Apart from the above differences, there are some other important concerns for Web-based applications:

- **Reliability of the Internet:** At the moment, there are reliability problems with the Internet. It may break down; data may be lost on the net; large amounts of data traffic may slow down or even overwhelm the network system.
- **Security:** Security on the Internet is of great concern for any organisation which has developed Web-based database applications. For example, the database may be broken into, or confidential data may be intercepted during transmission by unauthorised parties or even criminals.

At the present, a lot of research and development work is being carried out to address these concerns. There is no doubt that the potential problems can be overcome and over time, the Internet will be more reliable and more secure for connecting the world.

Context summary

In this section, we have drawn an overall picture of Web-based database applications. We have briefly mentioned how a DBMS can be integrated with the Web, and what their advantages and disadvantages are as compared to the traditional client-server applications. In the rest of this chapter, we will study the details of creating Web database applications and discuss the commonly used approaches of linking databases to the Web.

Review question 1

- Are the Internet and WWW (Web) the same concept? Why?
- What are intranets and extranets?
- What is a URL?
- Most Web sites have URLs starting with http://……. Why?

- What is a dynamic Web page? What are its characteristics?
- What are the major features of a Web-based client-server application?

Web database architectures

Components of a database application

Web database applications may be created using various approaches. However, there are a number of components that will form essential building blocks for such applications. In other words, a Web database application should comprise the following four layers (i.e. components):

- Browser layer
- Application logic layer
- Database connection layer
- Database layer

Browser layer

The browser is the client of a Web database application, and it has two major functions. First, it handles the layout and display of HTML documents. Second, it executes the client-side extension functionality such as Java, JavaScript, and ActiveX (a method to extend a browser's capabilities).

The three most popular browsers at the present are Mozilla Firefox (Firefox for short), Google Chrome and Microsoft Internet Explorer (IE).

All three browsers are graphical browsers. During the early days of the Web, a text-based browser, called Lynx, was popular. As loading graphics over the Internet can be a slow and time-consuming process, database performance may be affected. If an application requires a speedy client and does not need to display graphics, then the use of Lynx may be considered.

All browsers implement the HTML standard. The discussion of HTML is beyond this chapter, but you need to know that it is a language used to format data/documents to be displayed on the Web.

Browsers are also responsible for providing forms for the collection of user input, packaging the input, and sending it to the appropriate server for processing. For example, input can include registration for site access, guest books and requests for information. HTML, Java, JavaScript or ActiveX (for IE) may be used to implement forms.

Application logic layer

The application logic layer is the part of a Web database application with which a developer will spend the most time. It is responsible for:

- Collecting data for a query (e.g. a SQL statement).
- Preparing and sending the query to the database via the database connection layer.
- Retrieving the results from the connection layer.
- Formatting the data for display.

Most of the application's business rules and functionality will reside in this layer. Whereas the browser client displays data as well as forms for user input, the application logic component compiles the data to be displayed and processes user input as required. In other words, the application logic generates HTML that the browser renders. Also it receives, processes and stores user input that the browser sends.

Depending on the implementation methods used for the database application, the application logic layer may have different security responsibilities. If the application uses HTML for the front end, the browser and server can handle data encryption (i.e. a security measure to ensure that data will not be able to be intercepted by unauthorised parties). If the application is a Java applet and uses Java for the front end, then it itself must be responsible for adopting transmission encryption.

Database connection layer

This is the component which actually links a database to the Web server. Because manual Web database programming can be a daunting task, many current Web database building tools offer database connectivity solutions, and they are used to simplify the connection process.

The database connection layer provides a link between the application logic layer and the DBMS. Connection solutions come in many forms, such as DBMS net protocols, API (Application Programming Interface [see note below]) or class libraries, and programs that are themselves database clients. Some of these solutions resulted in tools being specifically designed for developing Web database applications. In Oracle, for example, there are native API libraries for connection and a number of tools, such as Web Publishing Assistant, for developing Oracle applications on the Web.

The connection layer within a Web database application must accomplish a number of goals. It has to provide access to the underlying database, and also needs to be easy to use, efficient, flexible, robust, reliable and secure. Different tools and methods fulfil these goals to different extents.

Note

An API consists of a set of interrelated subroutines that provide the functionality required to develop programs for a target operating environment. For example, Microsoft provides different APIs targeted at the construction of 16- and 32-bit Windows applications. An API would provide functions for all aspects of system activity, such as memory, file and process management. Specialised APIs are also supplied by software vendors to support the use of their products, such as database and network management systems.

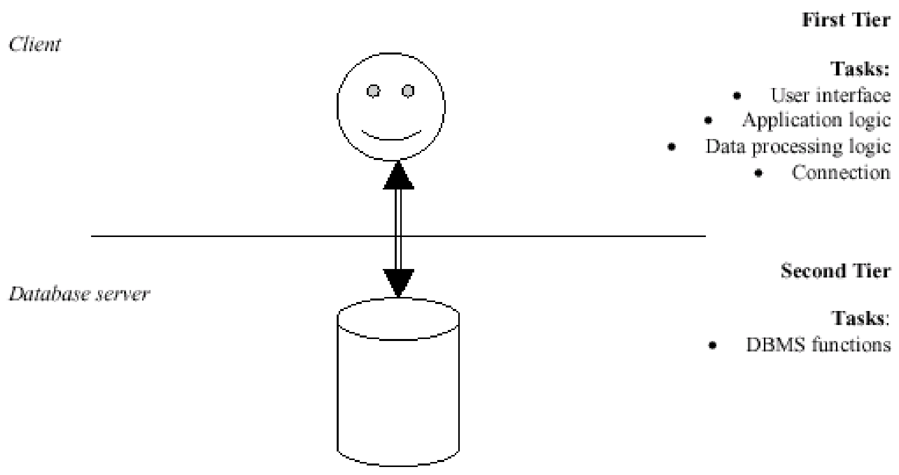
Database layer

This is the place where the underlying database resides within the Web database application. As we have already learned, the database is responsible for storing, retrieving and updating data based on user requirements, and the DBMS can provide efficiency and security measures.

In many cases, when developing a Web database application, the underlying database has already been in existence. A major task, therefore, is to link the database to the Web (the connection layer) and to develop the application logic layer.

2-tier client-server architecture

Traditional client-server applications typically have a 2-tier architecture as illustrated in the figure below. The client (tier 1) is primarily responsible for the presentation of data to the user, and the server (tier 2) is responsible for supplying data services to the client. The client will handle user interfaces and main application logic, and the server will mainly provide access services to the underlying database.

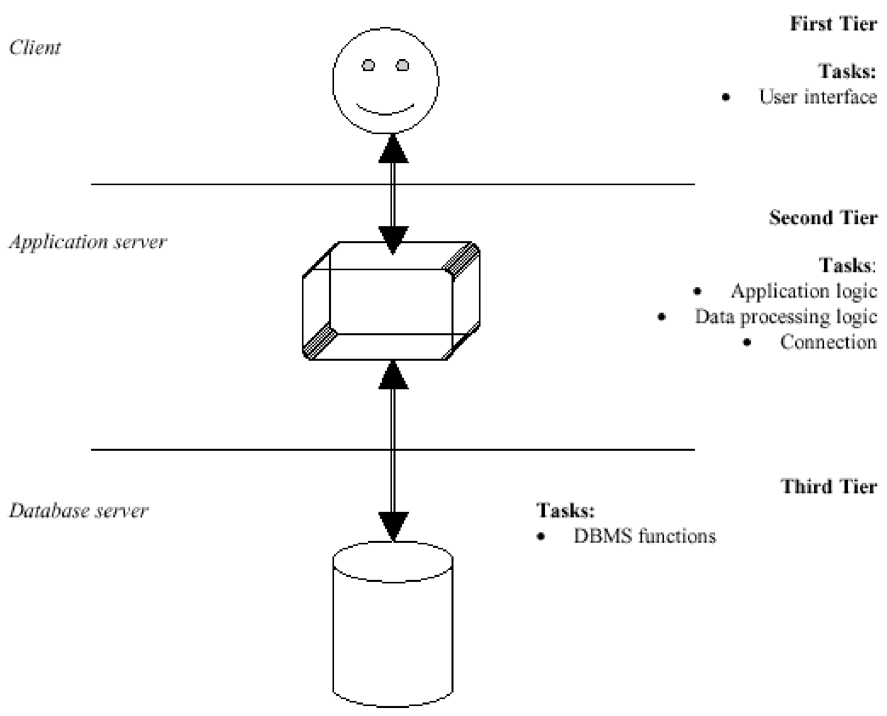


If such a 2-tier architecture is used to implement a Web database application, tier 1 will contain the browser layer, the application logic layer and the connection layer. Tier 2 accommodates the DBMS. This will inevitably result in a fat client.

3-tier client-server architecture

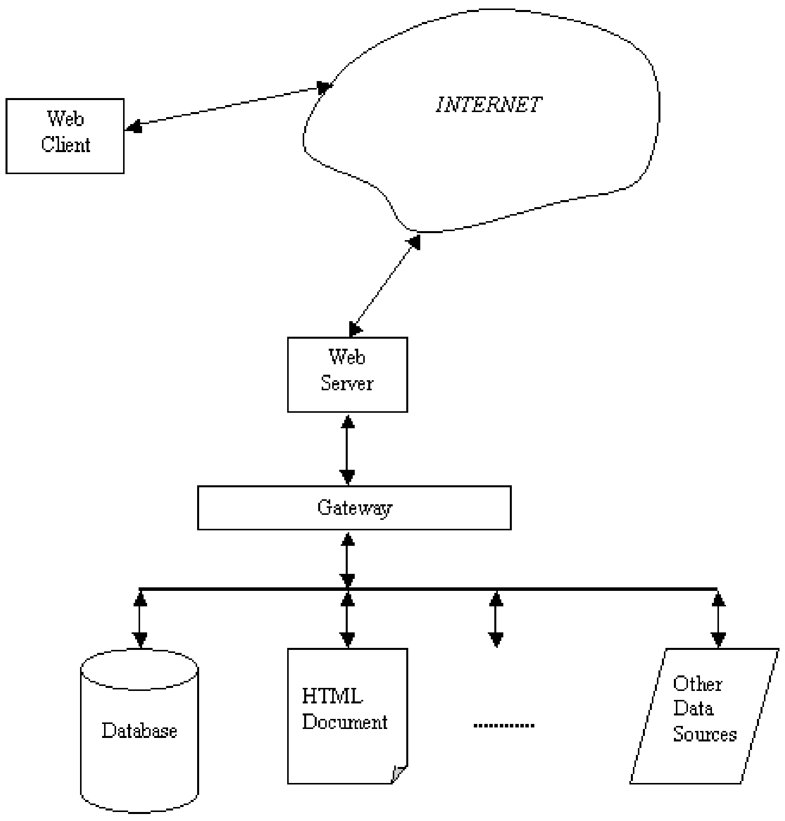
In order to satisfy requirements of increasingly complex distributed database applications, a 3-tier architecture was proposed to replace the 2-tier one. There are three tiers in this new architecture, each of which can potentially run on a different platform.

The first tier is the client, which contains user interfaces. The middle tier accommodates the application server, which provides application logic and data processing functions. The third tier contains the actual DBMS, which may run on a separate server called a database server.



The 3-tier architecture is more suitable for implementing a Web database application. The browser layer can reside in tier 1, together with a small part of the application logic layer. The middle tier implements the majority of the application logic as well as the connection layer. Tier 3 is for the DBMS.

Referring to the figure below, for example, it can be seen that the Web Client is in the first tier. The Web Server and Gateway are in the middle tier and they form the application server. The DBMS and possibly other data sources are in the third tier.



Having studied the Web database architectures, we should understand that the most important task in developing a Web database application is to build the database connection layer. In other words, we must know how to bridge the gap between the application logic layer and the database layer.

Review question 2

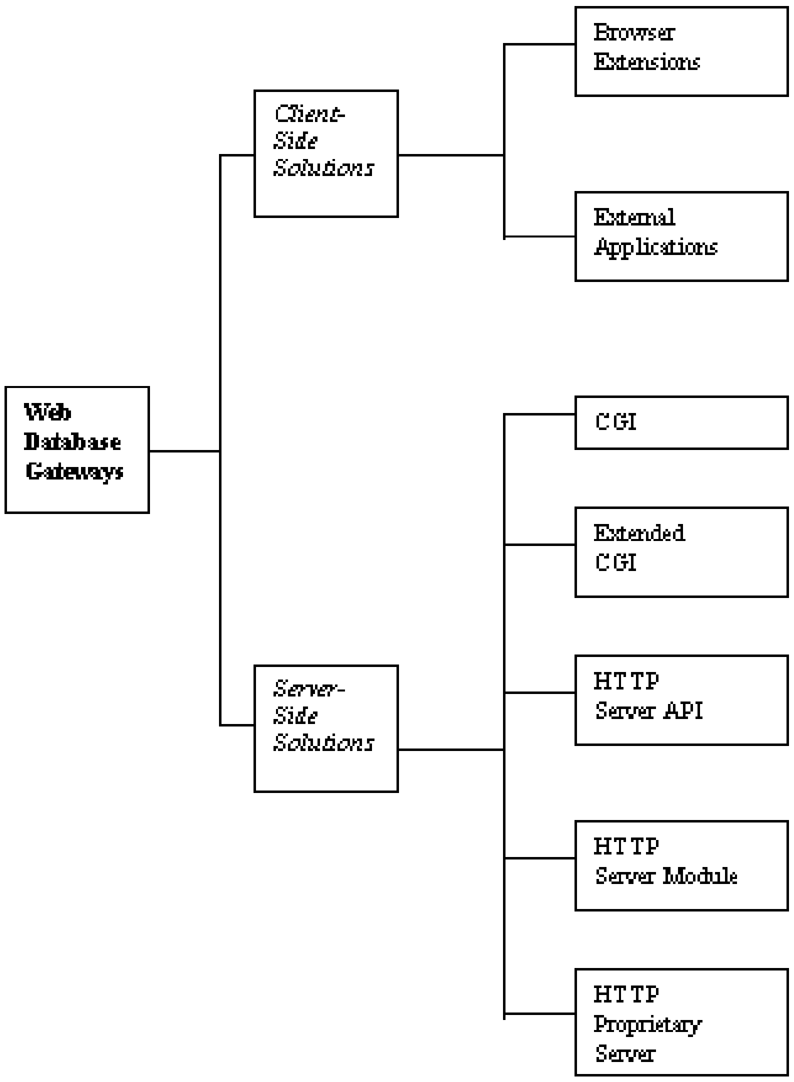
- What is the typical architecture of a Web database application?
- How can a 3-tier client-server architecture be used to implement a Web database application?

Database gateways

A Web database gateway is a bridge between the Web and a DBMS, and its objective is to provide a Web-based application the ability to manipulate data stored in the database. Web database gateways link stateful systems (i.e. databases) with a stateless, connectionless protocol (i.e. HTTP). HTTP is a stateless protocol in the sense that each connection is closed once the server provides a response. Thus, a Web server will not normally keep any record about previous requests. This results in an important difference between a Web-based client-server application and a traditional client-server application:

- In a Web-based application, only one transaction can occur on a connection. In other words, the connection is created for a specific request from the client. Once the request has been satisfied, the connection is closed. Thus, every request involving access to the database will have to incur the overhead of making the connection.
- In a traditional application, multiple transactions can occur on the same connection. The overhead of making the connection will only occur once at the beginning of each database session.

There are a number of different ways to create Web database gateways. Generally, they can be grouped into two categories: client-side solutions and server-side solutions, as illustrated below:



Client-side solutions

The client-side solutions include two types of approaches for connections: browser extensions and external applications.

Browser extensions are add-ons to the core Web browser that enhance and augment the browser's original functionality. Specific methods include plug-ins for Firefox, Chrome and IE, and ActiveX controls for IE. Also, all the three types of browsers (Firefox, Chrome and IE) support Java and JavaScript languages (i.e. Java applets and JavaScript can be used to extend browsers' capabilities).

External applications are helper applications or viewers. They are typically existing database clients that reside on the client machine and are launched by the Web browser in a particular Web application. Using external applications is a quick and easy way to bring legacy database applications online, but the resulting system is neither open nor portable. Legacy database clients do not take advantages of the platform independence and language independence available through many Web solutions. Legacy clients are resistant to change, meaning that any modification to the client program must be propagated via costly manual installations throughout the user base.

Server-side solutions

Server-side solutions are more widely adopted than the client-side solutions. A main reason for this is that the Web database architecture requires the client to be as thin as possible. The Web server should not only host all the documents, but should also be responsible for dealing with all the requests from the client.

In general, the Web server should be responsible for the following:

- Listening for HTTP requests.
- Checking the validity of the request.
- Finding the requested resource.
- Requesting authentication if necessary.
- Delivering requested resource.
- Spawning programs if required.
- Passing variables to programs.
- Delivering output of programs to the requester.
- Displaying error message if necessary.

The client (browser) should be responsible for some of the following:

- Rendering HTML documents.
- Allowing users to navigate HTML links.
- Displaying image.
- Sending HTML form data to a URL.

- Interpreting Java applets.
- Executing plug-ins.
- Executing external helper applications.
- Interpreting JavaScript and other scripting language programs.
- Executing ActiveX controls in the case of IE.

In the following sections, we are going to discuss both client-side and server-side solutions in some detail.

Review question 3

- What is a gateway in a Web database application and why is it needed?
- Where can we implement a gateway for a Web database application?

Client-side Web database programming

Major tasks of client-side Web database application programming include the creation of browser extensions and the incorporation of external applications. These types of gateways take advantage of the resources of the client machine, to aid server-side database access. Remember, however, it is advantageous to have a thin client. Thus, the scope of such programming on the client-side should be limited. A very large part of the database application should be on the server side.

Browser extensions

Browser extensions can be created by incorporating script language interpreters to support script languages (e.g. JavaScript), bytecode interpreters to support Java, and dynamic object linkers to support various plug-ins.

JavaScript

JavaScript is a scripting language that allows programmers to create and customise applications on the Internet and intranets. On the client side, it can be used to perform simple data manipulation such as mathematical calculations and form validation. JavaScript code is normally sent as a part of an HTML document and is executed by the browser upon receipt (the browser must have the script language interpreter).

Note that JavaScript has little to do with Java language. JavaScript was originally called LiveScript, but it was changed to benefit from the excitement surrounding Java. The only relationship between JavaScript and Java is a gateway between the former and Java applets (Web applications written in Java).

JavaScript provides developers with a simple way to access certain properties and methods of Java applets on the same page, without having to understand or modify the Java source code of the applet.

Connection to databases

As a database gateway, JavaScript on the client side does not offer much without the aid of a complementary approach such as Java, plug-ins and CGI (Common Gateway Interface, to be discussed later). For example:

- If a Java applet on a page of HTML has access to a database, a programmer can write JavaScript code using LiveConnect to manipulate the applet.
- If there is a form on the HTML document and if an action parameter for that form refers to a CGI program that has access to a database, a programmer can write JavaScript code to manipulate the data elements within the form and then submit it (i.e. submit a kind of request to a DBMS).

Performance

JavaScript can improve the performance of a Web database application if it is used for client-side state management. It can eliminate the need to transfer state data repeatedly between the browser and the Web server. Instead of sending an HTTP request each time it updates an application state, it sends the state only once as the final action. However, there are some side effects resulting from this gain in performance. For example, it may result in the application becoming less robust if state management is completely on the client side. If the client accidentally or deliberately exits, the session state is lost.

Java

As mentioned earlier, Java applets can be manipulated by JavaScript functions to access databases. In general, Java applets can be downloaded into a browser and executed on the client side (the browser should have the bytecode interpreter). The connection to the database is made through appropriate APIs (Application Programming Interface, such as JDBC and ODBC). We will discuss the details in the next section: Server-Side Web Database Programming.

ActiveX

ActiveX is a way to extend Microsoft IE's (Internet Explorer) capabilities. An ActiveX control is a component on the browser that adds functionality which cannot be obtained in HTML, such as access to a file on the client side, other applications, complex user interfaces, and additional hardware devices. ActiveX is similar to Microsoft OLE (Object Linking and Embedding), and ActiveX controls can be developed by any organisation and individual. At the present, more than one thousand ActiveX controls, including controls for database access, are available for developers to incorporate into Web applications.

Connection to databases

A number of commercial ActiveX controls offer database connectivity. Because ActiveX has abilities similar to OLE, it supports most or all the functionality available to any Windows program.

Performance

Like JavaScript, ActiveX can aid in minimising network traffic. In many cases, this technique results in improved performance. ActiveX can also offer rich GUIs. The more flexible interface, executed entirely on the client side, makes operations more efficient for users.

Plug-ins

Plug-ins are Dynamic Link Libraries (DLL) that give browsers additional functionality. Plug-ins can be installed to run seamlessly inside the browser window, transparent to the user. They have full access to the client's resources, because they are simply programs that run in an intimate symbiosis with the Web browser.

To create a plug-in, the developer writes an application using the plug-in API and native calls. The code is then compiled as a DLL. Installing a plug-in is just a matter of copying the DLL into the directory where the browser looks for plug-ins. The next time that the browser is run, the MIME type(s) that the new plug-in supports will be opened with the plug-in. One plug-in may support multiple MIME types.

There are a number of important issues concerning plug-ins:

- Plug-ins incur installation requirements. Because they are native code, not packaged with the browser itself, plug-ins must be installed on the client machine.
- Plug-ins are platform dependent. Whenever a change is made, it must be made on all supported platforms.

Connection to databases

Plug-ins can operate like any stand-alone applications on the client side. They can be used to create direct socket connections to databases via the DBMS net protocols (such as SQL *Net for Oracle). Plug-ins can also use JDBC, ODBC, OLE and any other methods to connect to databases.

Performance

Plug-ins are loaded on demand. When a user starts up a browser, the installed plug-ins are registered with the browser along with their supported MIME types, but the plug-ins themselves are not loaded. When a plug-in for a particular MIME type is requested, the code is then loaded into memory. Because plug-ins use native code, their executions are fast.

External applications

External helper applications can be new or legacy database clients, or a terminal emulator. If there are existing traditional client-server database applications which reside on the same machine as the browser, then they can be launched by the browser and execute as usual.

This approach may be an appropriate interim solution for migrating from an existing client-server application to a purely Web-based one. It is straightforward to configure the browser to launch existing applications. It just involves the registration of a new MIME type and the associated application name. For organisations that cannot yet afford the time and funds needed to transfer existing database applications to the Web, launching legacy applications from the browser provides a first step that requires little work.

Maintenance issues

Using the external applications approach, the existing database applications need not be changed. However, it means that all the maintenance burdens associated with traditional client-server applications will remain. Any change to the external application will require a very costly reinstallation on all client machines. Because this is not a pure Web-based solution, many advantages offered by Web-based applications cannot be realised.

Performance

Traditional client-server database applications usually offer good performance. They do not incur the overhead of requiring repeated connections to the database. External database clients can make one connection to the remote database and use that connection for as many transactions as necessary for the session, closing it only when finished.

Review question 4

What are the major tasks involved in client-side Web database programming?

Server-side Web database programming

CGI (Common Gateway Interface)

CGI is a protocol for allowing Web browsers to communicate with Web servers, such as sending data to the servers. Upon receiving the data, the Web server can then pass them to a specified external program (residing on the server host machine) via environment variables or standard input stream (STDIN). The external program is called a CGI program or CGI script. Because CGI is a protocol, not a library of functions written specifically for any particular Web server, CGI programs/scripts are language independent. As long as the program/script conforms to the specification of the CGI protocol, it can be written in any language such as C, C++ or Java. In short, CGI is the protocol governing communications among browsers, servers and CGI programs.

In general, a Web server is only able to send documents and to tell a browser what kinds of documents it is sending. By using CGI, the server can also launch external programs (i.e. CGI programs). When the server recognises that a URL points to a file, it returns the contents of that file. When the URL points to a CGI program, the server will execute it and then send back the output of the program's execution to the browser as if it were a file.

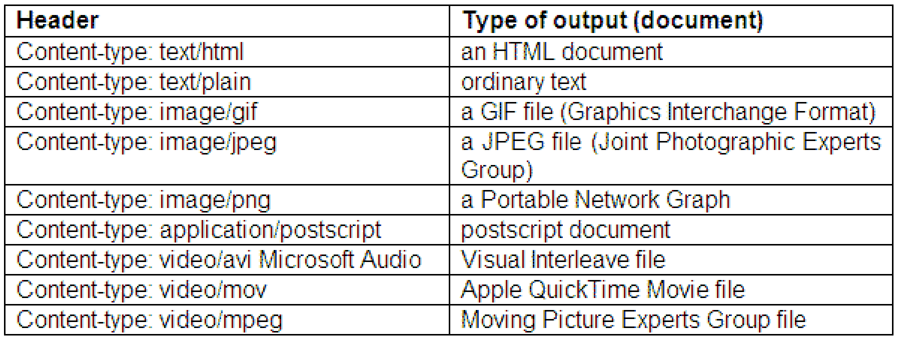
Before the server launches a CGI program, it prepares a number of environment variables representing the current state of the server which is requesting the action. The program collects this information and reads STDIN. It then carries out the necessary processing and writes its output to STDOUT (the standard output stream). In particular, the program must send the MIME header information prior to the main body of the output. This header information specifies the type of the output.

Refer to the figure under the Basic Concepts section. The CGI approach enables access to databases from the browser. The Web client can invoke a CGI program/script via a browser, and then the program performs the required action and accesses the database via the gateway. The outcome of accessing the database is then returned to the client via the Web server. Invoking and executing CGI programs from a Web browser is mostly transparent to the user. The following steps need to be taken in order for a CGI program to execute successfully:

- The user (Web client) calls the CGI program by clicking on a link or by pressing a button. The program can also be invoked when the browser loads an HTML document (hence being able to create a dynamic Web page).
- The browser contacts the Web server, asking for permission to run the CGI program.
- The server checks the configuration and access files to ensure that the program exists and the client has access authorisation to the program.
- The server prepares the environment variables and launches the program.
- The program executes and reads the environment variables and STDIN.
- The program sends the appropriate MIME headers to STDOUT, followed by the remainder of the output, and terminates.

- The server sends the data in STDOUT (i.e. the output from the program's execution) to the browser and closes the connection.
- The browser displays the information received from the server.

As mentioned earlier, when preparing data for the browser to display, the CGI program has to include a header as the first line of output. It specifies how the browser should display the output. This header may be one of the following types:



Header	Type of output (document)
Content-type: text/html	an HTML document
Content-type: text/plain	ordinary text
Content-type: image/gif	a GIF file (Graphics Interchange Format)
Content-type: image/jpeg	a JPEG file (Joint Photographic Experts Group)
Content-type: image/png	a Portable Network Graph
Content-type: application/postscript	postscript document
Content-type: video/avi Microsoft Audio	Visual Interleave file
Content-type: video/mov	Apple QuickTime Movie file
Content-type: video/mpeg	Moving Picture Experts Group file

Primarily, there are four methods available for passing information from the browser to a CGI program. In this way, clients' input (representing users' specific requirements) can be transmitted to the program for actions.

1. Passing parameters on the command line.
2. Passing environment variables to CGI programs.
3. Passing data to CGI programs via STDIN.
4. Using extra path information.

Detailed discussions on these methods are beyond the scope of this chapter. Please refer to any book dealing specifically with the CGI topic.

Advantages and disadvantages of CGI

Advantages

CGI is the de facto standard for interfacing Web clients and servers with external applications, and is arguably the most commonly adopted approach for interfacing Web applications to data sources (such as databases). The main advantages of CGI are its simplicity, language independence, Web server independence and its wide acceptance.

Disadvantages

The first notable drawback of CGI is that the communication between a client (browser) and the database server must always go through the Web server in the middle, which may cause a bottleneck if there is a large number of users accessing the Web server simultaneously. For every request submitted by a Web client or every response delivered by the database server, the Web server has to convert data from or to an HTML document. This incurs a significant overhead to query processing.

The second disadvantage of CGI is the lack of efficiency and transaction support in a CGI program. For every query submitted through CGI, the database server has to perform the same logon and logout procedure, even for subsequent queries submitted by the same user. The CGI program could handle queries in batch mode, but then support for online database transactions that contain multiple interactive queries would be difficult.

The third major shortcoming of CGI is due to the fact that the server has to generate a new process or thread for each CGI program. For a popular site (like Yahoo), there can easily be hundreds or even thousands of processes competing for memory, disk and processor time. This situation can incur significant overhead.

Last but not least, extra measures have to be taken to ensure server security. CGI itself does not provide any security measures, and therefore developers of CGI programs must be security conscious. Any request for unauthorised action must be spotted and stopped.

Extended CGI

As discussed in the previous section, one of the major concerns with CGI is its performance. With CGI, a process is spawned on the server each time a request is made for a CGI program. There is no method for keeping a spawned process alive between successive requests, even if they are made by the same user. Furthermore, CGI does not inherently support distributed processing, nor does it provide any mechanism for sharing commonly used data or functionality among active and future CGI requests. Any data that exists in one instance of a CGI program cannot be accessed by another instance of the same program.

In order to overcome these problems, an improved version of CGI, called FastCGI, has been developed with the following features:

- Language independence: As with CGI, FastCGI is a protocol and not dependent on any specific language.
- Open standard: Like CGI, FastCGI is positioned as an open standard. It can be implemented by anyone. The specifications, documentation and source code (in different languages) can be obtained at the Web site https://soramimi.jp/fastcgi/fastcgispec.html.

- Independence from the Web server architecture: A FastCGI application need not be modified when an existing Web server architecture changes. As long as the new architecture supports the FastCGI protocol, the application will continue to work.
- Distributed computing: FastCGI allows the Web application to be run on a different machine from the Web server. In this way, the hardware can be tuned optimally for the software.
- Multiple, extensible roles: In addition to the functionality offered by CGI (i.e. receiving data and returning responses), FastCGI can fill multiple roles such as a filter role and an authoriser role. A FastCGI application can filter a requested file before sending it to the client; the authoriser program can make an access control decision for a request, such as looking up a username and password pair in a database. If more roles are needed, more definitions and FastCGI programs can be written to fulfil them.
- Memory sharing: In some cases, a Web application might need to refer to a file on disk. Under CGI, the file would have to be read into the memory space of that particular instance of the CGI program; if the CGI program were accessed by multiple users simultaneously, the file would be loaded and duplicated into different memory locations. With FastCGI, different instances of the same application can access the same file from the same section of memory without duplication. This approach improves performance.
- Allocating processes: FastCGI applications do not require the Web server to start a new process for each application instance. Instead, a certain number of processes are allotted to the FastCGI application. The number of processes dedicated for an application is user-definable. These processes can be initiated when the Web server is started or on demand.

FastCGI seems to be a complete solution for Web database programming, as it includes the best features of CGI and server APIs. In the following sections, a number of CGI-alternative approaches are discussed.

HTTP server APIs and server modules

HTTP server (Web server) APIs and modules are the server equivalent of browser extensions. The central theme of Web database sites created with HTTP server APIs or modules is that the database access programs coexist with the server. They share the address space and run-time process of the server. This approach is in direct contrast to the architecture of CGI, in which CGI programs run as separate processes and in separate memory spaces from the HTTP server.

Instead of creating a separate process for each CGI program, the API offers a way to create an interface between the server and the external programs using dynamic linking or shared objects. Programs are loaded as part of the server, giving them full access to all the I/O functions of the server. In addition, only one copy of the program is loaded and shared among multiple requests to the server.

Server vendor modules

Server modules are just prefabricated applications written in some server APIs. Developers can often purchase commercial modules to aid or replace the development of an application feature. Sometimes, the functionality required in a Web database application can be found as an existing server module.

Vendors of Web servers usually provide proprietary server modules to support their products. There are a very large number of server modules that are commercially available, and the number is still rising. For example, Oracle provides the Oracle PL/SQL module, which contains procedures to drive database-backed Web sites. The Oracle module supports both NSAPI and ISAPI.

Advantages of server APIs and modules

Having database access programs coexist with the HTTP server improves Web database access due to improved speed, resource sharing, and the range of functionality.

- **Server speed**

  API programs run as dynamically loaded libraries or modules. A server API program is usually loaded the first time the resource is requested, and therefore, only the first user who requests that program will incur the overhead of loading the dynamic libraries. Alternatively, the server can force this first instantiation so that no user will incur the loading overhead. This technique is called preloading. Either way, the API approach is more efficient than CGI.

- **Resource sharing**

  Unlike a CGI program, a server API program shares address space with other instances of itself and with the HTTP server. This means that any common data required by the different threads and instances need exist only in one place. This common storage area can be accessed by concurrent and separate instances of the server API program.

  The same principle applies to common functions and code. The same set of functions and code are loaded just once and can be shared by multiple server API programs. The above techniques save space and improve performance.

- **Range of functionality**

A CGI program has access to a Web transaction only at certain limited points. It has no control over the HTTP authentication scheme. It has no contact with the inner workings of the HTTP server, because a CGI program is considered external to the server.

In contrast, server API programs are closely linked to the server; they exist in conjunction with or as part of the server. They can customise the authentication method as well as transmission encryption methods. Server API programs can also customise the way access logging is performed, providing more detailed transaction logs than are available by default.

Overall, server APIs provide a very flexible and powerful solution to extending the capabilities of Web servers. However, this approach is much more complex than CGI, requiring specialised programmers with a deep understanding of the Web server and sophisticated programming skills.

Important issues

- **Server architecture dependence**

  Server APIs are closely tied to the server they work with. The only way to provide efficient cross-server support is for vendors to adhere to the same API standard. If a common API standard is used, programs written for one server will work just as well with another server. However, setting up standards involves compromises among competitors. In many cases, they are hard to come by.

- **Platform dependence**

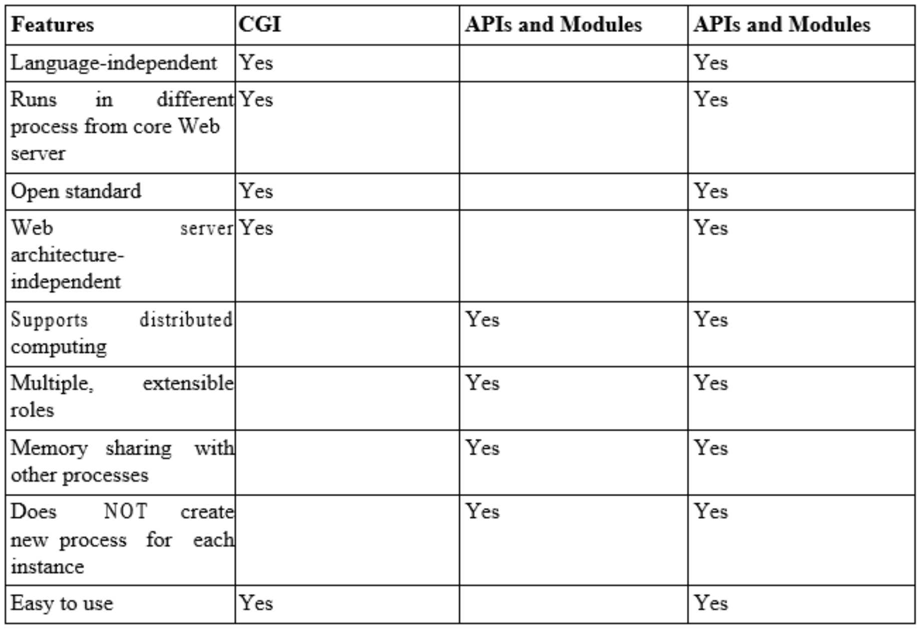
  Server APIs and modules are also dependent on computing platforms. Some servers are supported on multiple platforms. Nevertheless, each supporting version is dependent on that platform. Similarly, the Microsoft server is only available for various versions of Windows.

- **Programming language**

  Most Web servers can be extended using a variety of programming languages and facilities. In addition, Microsoft provides an application environment called Active Server Pages. Active Server Pages is an open, compile-free application environment in which developers can combine HTML, scripts and reusable ActiveX server components to create dynamic and powerful Web-based business solutions.

Comparison of CGI, server APIs and modules, and FastCGI

The following table provides a straightforward comparison among approaches of CGI, server APIs and modules, and FastCGI:



Features	CGI	APIs and Modules	APIs and Modules
Language-independent	Yes		Yes
Runs in different process from core Web server	Yes		Yes
Open standard	Yes		Yes
Web server architecture-independent	Yes		Yes
Supports distributed computing		Yes	Yes
Multiple, extensible roles		Yes	Yes
Memory sharing with other processes		Yes	Yes
Does NOT create new process for each instance		Yes	Yes
Easy to use	Yes		Yes

Proprietary HTTP servers

A proprietary HTTP server is defined as a server application that handles HTTP requests and provides additional functionality that is not standard or common among available HTTP servers. The functionality includes access to a particular database or data source, and translation from a legacy application environment to the Web.

Examples of proprietary servers include IBM Domino, Oracle Application Express Listener and Hyper-G. These products were created for specific needs. For Domino, the need is tight integration with legacy Lotus Notes applications, allowing them to be served over the Web. Oracle Application Express Listener was designed to provide highly efficient and integrated access to back-end Oracle databases. For Hyper-G, the need is to have easily maintainable Web sites with automatic link update capabilities.

The main objectives of creating proprietary servers are to meet specialised and customised needs, and to optimise performance. However, the benefits of proprietary servers must be carefully weighed against their exclusive ties to a Web database product (which may bring many shortcomings). It requires a thorough understanding of the business requirements in order to determine whether or not a proprietary Web server is appropriate in a project.

Review question 5

- What is CGI?
- What are the typical steps in the procedure by a Web client of invoking a CGI program?
- What are Web server APIs and server modules?
- Compare the features of CGI, FastCGI, and server APIs and modules.

Connecting to the database

In previous sections, we have studied various approaches that enable browsers (Web clients) to communicate with Web servers, and in turn allow Web clients to have access to databases. For example, CGI, FastCGI or API programs can be invoked by the Web client to access the underlying database. In this section, we are going to discuss how database connections can actually be made via those CGI/FastCGI/API programs. We will learn what specific techniques, tools and languages are available for making the connections. In short, we will see how the database connection layer is built for the underlying database.

In general, database connectivity solutions include the use of:

- Native database APIs
- Database-independent APIs
- Template-driven database access packages
- Third-party class libraries

Do not be confused with the concepts of Web server APIs and database APIs. Web server APIs are used to write server applications, in which database APIs are used specifically for connecting to and accessing the database. Also, database APIs can be used to write a CGI program which allows developers to create a Web application with a database back end. Similarly, a template-driven database access package, along with a program written in a Web server's API (e.g. NSAPI, ISAPI), is another way to link a Web front end to a database back end.

Database API libraries

Before we look at specific API database connectivity solutions, let's give background to database API libraries.

Database API libraries are at the core of every Web database application and gateway. Regardless how a Web database application is built (whether by manually coding CGI programs or by using a visual application builder), database API libraries are the foundation of database access.

The approach

Database API libraries are collections of functions or object classes that provide source code access to databases. They offer a method of connecting to the database engine (under a username and password if user authentication is supported by the DBMS), sending queries across the connection, and retrieving the results and/or error messages in a desired format.

Traditional client-server database applications have already employed database connectivity libraries supplied by vendors and third-party software companies (i.e., third party class libraries). Because of this fact of wider user base, database APIs have the advantage over other gateway solutions for Web database connectivity.

The Web database applications that require developers to use database API libraries are mainly CGI, FastCGI or server API programs. Web database application building tools, including template-driven database access packages and visual GUI builders, use database APIs as well as the supporting gateways (such as CGI and server API), but all these interactivities are hidden from the developers.

Native database APIs

Native database APIs are platform-dependent as well as programming language dependent. However, most popular databases (such as Oracle) support multiple platforms in the first place, and therefore, the porting between different platforms should not require excessive effort.

In general, programs that use native database APIs are faster than those using other methods, because the libraries provide direct and low-level access. Other database access methods tend to be slower, because they add another layer of programming to provide the developer a different, easier, or more customised programming interface. These additional layers slow the overall transaction down.

Native database API programming is not inherently dependent on a Web server. For example, a CGI program using native API calls to Oracle that works with the Netscape server should also work with other types of servers. However, if the CGI program also incorporates Web server-specific functions or modules, it will be dependent on that Web server.

Database-independent APIs: ODBC

The most popular standard database-independent API was pioneered by Microsoft. It is called ODBC (Open Database Connectivity) and is supported by all of the most popular databases such as Microsoft Access and Oracle.

ODBC requires a database-specific driver or client to be loaded on the database client machine. In a Java application that accesses Oracle, for example, the server that hosts the Java application would need to have an Oracle ODBC client installed. This client would allow the Java application to connect to the ODBC data source (the actual database) without knowing anything about the Oracle database.

In addition to the database-specific ODBC driver being installed on the client machine, Java requires that a JDBC-ODBC bridge (i.e. another driver. JDBC stands for Java Database Connectivity) be present on the client machine. This JDBC-ODBC driver translates JDBC to ODBC and vice versa, so that Java programs can access ODBC-compliant data sources but still use their own JDBC class library structure.

Having the database-specific ODBC driver on the client machine dictates that Web database Java applications or applets using ODBC be 3-tiered. The database client of the Web application must reside on a server: either the same server as the Web server or a remote server. Otherwise, the database-specific ODBC driver would have to exist on every user's computer, which is a very undesirable situation. The diagram below provides a graphical illustration of such an architecture.

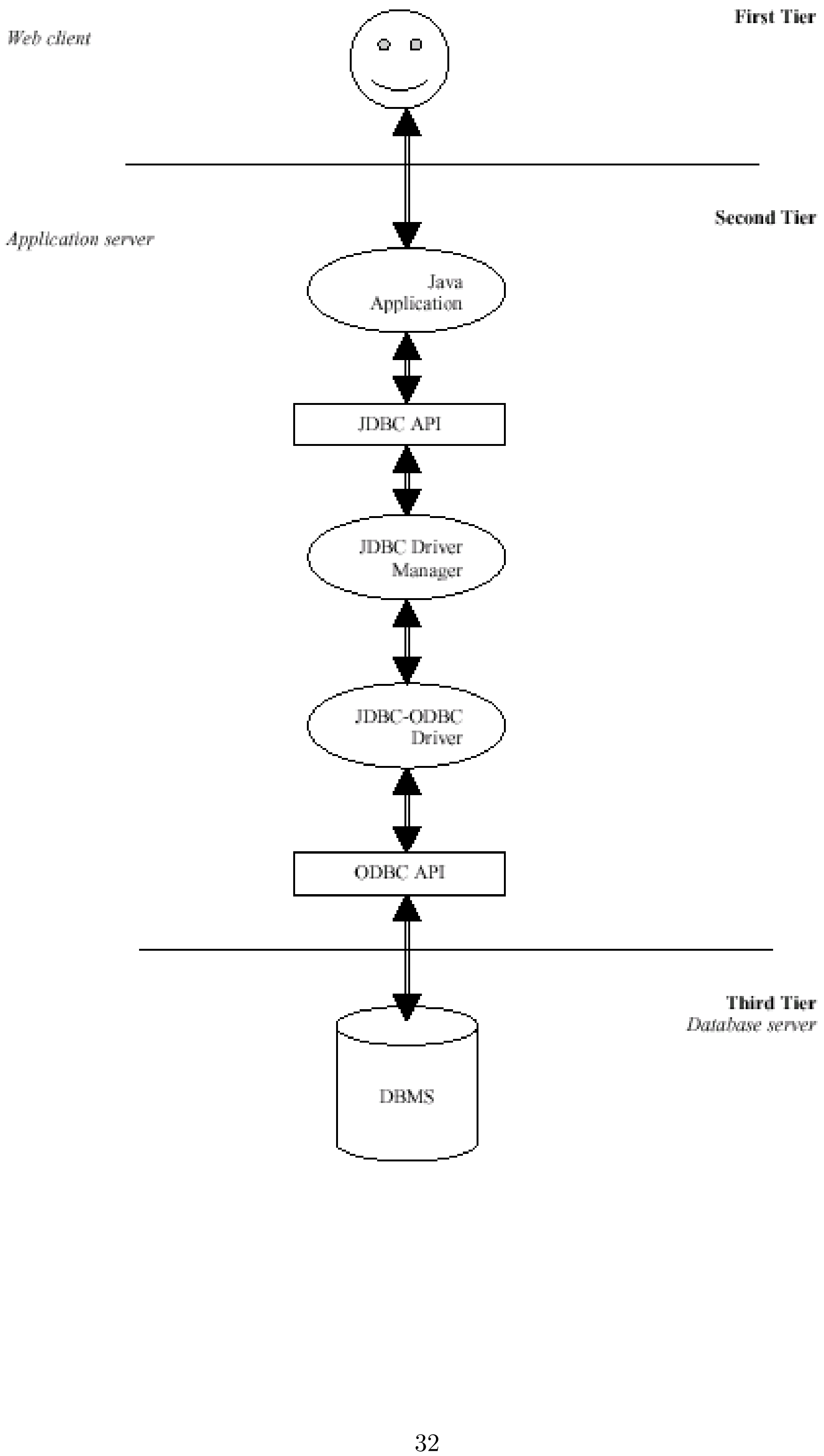
Benefits of database APIs

Database APIs (native or independent) arguably offer the most flexible way in which Web database applications are created. Applications created with native database APIs are more efficient than those with database-independent APIs. This database connectivity solution is the fastest way to access database functionality and has been tested rigorously in the database software industry. It is worth noting that database APIs have been used successfully for years even before the invention of the Web.

Shortcomings of database APIs

The most notable disadvantage of programming in database API is complexity. For rapid application development and prototyping, it is better to use a high-level tool, such as template-driven database access software or visual application builders.

Another disadvantage is with ODBC. Because ODBC standardises access to databases from multiple vendors, applications using ODBC do not have access to native SQL database calls that are not supported by the ODBC standard. In some cases, this can be inconvenient and may even affect application performance.


First Tier
Web client
Second Tier
Application server
Java Application
JDBC API
JDBC Driver Manager
JDBC-ODBC Driver
ODBC API
Third Tier
Database server
DBMS

Template-driven packages

The approach

Template-driven database connectivity packages are offered by database vendors and third-party developers to simplify Web database application programming. Such a package usually consists of the following components:

- Template consisting of HTML and non-standard tags or directives
- Template parser
- Database client daemons

Template-driven packages are very product dependent. Different DBMSs require database access templates in different formats. An application developed for one product will be strongly tied to it. Migrating from one product to another is very difficult and requires a rewrite of all the database access, flow control and output-formatting commands.

An example of a template-driven package is PHP.

Benefits of template-driven packages

The most important benefit from using a template-driven package is speed of development. Assuming an available package has been installed and configured properly, it takes as little time as a few hours to create a Web site that displays information directly from the database.

Shortcomings of template-driven packages

The structures of templates are normally predetermined by vendors or third-party developers. As a result, they only offer a limited range of flexibility and customisability. Package vendors provide what they feel is important functionality, but, as with most off-the-shelf tools, such software packages may not let you create applications requiring complex operations.

Although templates offer a rapid path to prototyping and developing simple Web database applications, the ease of development is obtained for the cost of speed and efficiency. Because the templates must be processed on demand and require heavy string manipulation (templates are of a large text type or string type; they must be parsed by the parser), using them is slow compared with using direct access such as native database APIs.

The actual performance of an application should be tested and evaluated before the usefulness of such a package is ruled out. The overhead of parsing templates may be negligible if using high-performance machines. Other factors, such as development time or development expertise, may be more important than a higher operational speed.

GUI application builders

Visual Web database building tools offer an interesting development environment for creating Web database applications. For developers accustomed to point-and-click application programming, these tools help speed the development process. For instance, Visual Basic and/or Microsoft Access developers should find such a tool intuitive and easy to use.

The approach

The architectures of visual building tools vary. In general, they include a user-friendly GUI (Graphical User Interface), allowing developers to build a Web database application with a series of mouse clicks and some textual input. These tools also offer application management so that a developer no longer needs to juggle multiple HTML documents and CGI, NSAPI or ISAPI programs manually.

At the end of a building session, the tool package can generate applications using various techniques. Some applications are coded using ODBC; some use native database APIs for the databases they support; and others may use database net protocols.

Some of these tools create their own API, which can be used by other developers. Some generate code that works but can still be modified and customised by developers using various traditional IDEs, compilers and debuggers.

A building tool may generate a CGI program or a Web server API program (such as NSAPI and ISAPI). Some sophisticated tools even offer all the options. The developer can choose what he/she wants.

Unlike native database APIs or template-driven database connectivity packages, visual Web database development tools tend to be as open as possible. Many offer development support for the popular databases.

Benefits of visual tools

Visual development tools can be of great assistance to developers who are familiar and comfortable with visual application development techniques. They offer rapid application development and prototyping, and an organised way to manage the application components. Visual tools also shield the developer from low-level details of Web database application development. As a result, a developer can create a useful Web application without the need to know what is happening in the code levels.

Shortcomings of visual tools

Depending on the sophistication of the package used, the resulting programs may be slower to execute than similar programs coded by an experienced programmer. Visual application building tools, particularly Object-oriented ones, tend to generate fat programs with a lot of unnecessary sub-classing.

Another potential drawback is cost. A good visual tool may be too expensive for a small one-off development budget.

Review question 6

- What are database APIs? Who uses them and why?
- Why are template-driven packages useful for building database connections? What are the shortcomings?
- How can we benefit from using visual development tools to build database connections?

Managing state and persistence in Web applications

State is an abstract concept of being, which can be explained by a set of rules, facts or truisms. A state in a database application includes a set of variables and/or other means to record who the user/client is, what tasks he/she has been doing, at what position he/she is at a particular instance in time, and many other useful pieces of information about a database session. Persistence is the capability of remembering a state and tracking state changes across different applications or different periods of time within an instance of an application or multiple instances.

The requirement of state maintenance in Web database applications results in the increased complexity. As mentioned before in the Context section, HTTP is connectionless, which means that once an HTTP request is sent and a response is received, the connection to the server is closed. If a connection were to be kept open between client and server, the server could at any time query the client for state information and vice versa. The server would be able to know the identity of the user throughout the session once the user logged in. However, the reality is that there is no constant connection throughout the session. Thus, the server cannot have memory of the user's identity even after user login. In this situation, programmers must find a way to make session state persist.

Technical options

There are several options available to programmers to maintain state. They range from open systems options defined in HTTP and CGI standards, to proprietary mechanisms written from scratch.

The most important task in maintaining persistence is to keep track of the identity of the user. If the identity can persist, any other data/information can usually be made to persist in exactly the same manner.

The URL approach

It works as follows:

- A registration or login page is delivered to the user.
- The user types in a username and password, and then submits the page.
- The username and password pair are sent to a server-side CGI program, which extracts the values from the QUERY_STRING environment variable.
- The values are checked by the server to determine whether or not the user is authenticated.
- If he/she is authenticated, the authenticated state is reflected in a randomly generated session ID (SID), which is stored in a database along with other necessary data to describe the state of the user session.
- The SID can then be stored in all URLs within HTML documents returned by the server to the client, therefore tracking the identity of the user throughout the session.

Benefits of the URL approach

The URL approach is easy to use to maintain state. To retrieve a state, the receiving CGI program need only collect the data from environment variables in the GET method and act on it as necessary. To pass on, set or change the state, the program simply creates new URLs with the appropriate data.

Shortcomings of the URL approach

If the state information has to be kept in the URL, the URL becomes very long and can be very messy. Also, such a URL displays part of the application code and low-level details. This causes security concerns, and may be used by hackers.

If an application manages state on the client side using the URL method, the state will be lost when the user quits the browser session unless the user bookmarks the URL. A bookmark saves the URL in the browser for future retrieval. If state is maintained solely in the URL without any server-side state data management, bookmarking is sufficient to recreate the state in a new browser session. However, having the user perform this maintenance task is obviously undesirable.

URL QUERY_STRING

This is another popular method of maintaining state. A registered user in a site has a hidden form appended to each page visited within the site. This form contains the username and the name of the current page. When the user moves from one page to another, the hidden form moves as well and is appended to the end of the succeeding HTML page.

Benefits of the hidden fields approach

Like the URL approach, it is easy to use to maintain state. In addition, because the fields are hidden, the user has a seamless experience and sees a clean URL.

Another advantage of using this approach is that, unlike using URLs, there is no limit on the size of data that can be stored.

Shortcomings of the hidden fields approach

As with the URL approach, users can fake states by editing their own version of the HTML hidden fields. They can bring up the document source in an editor, change the data stored, and then submit the tampered form to the server. This raises serious security concerns.

Data is also lost between sessions. If the entire session state is stored in hidden fields, that state will not be accessible after the user exits the browser unless the user specifically saves the HTML document to disk or with a bookmark. Again, it is undesirable to involve users in this kind of maintenance task.

HTTP cookies

An HTTP cookie is a technique that helps maintain state in Web applications. A cookie is in fact a small text file containing:

- Name of the cookie
- Domains for which the cookie is valid
- Expiration time in GMT
- Application-specific data such as user information

Cookies are sent by the server to the browser, and saved to the client's disk. Whenever necessary, the server can request a desired cookie from the client. The client browser will check whether it has it. If it does, the browser will send the information stored in the cookie to the server.

Benefits of cookies

Cookies can be completely transparent. As long as a user does not choose the browser option to be alerted before accepting cookies, his/her browser will handle incoming cookies and place them on the client disk without user intervention.

Cookies are stored in a separate file, whose location is handled by the browser and difficult for the user to find. Also, cookies are difficult to tamper with. This increases security.

Because cookies are stored on the client disk, the cookie data is accessible even in a new browser session. It does not require theuser to do anything.

If a programmer chooses to set an expiration date or time for a cookie, the browser will invalidate the cookie at the appropriate time.

Shortcomings of cookies

The amount of data that can be stored with a cookie is usually limited to 4 kilobytes. If an application has very large state data, other techniques must be considered.

Because cookies are physically stored on the client disk, they cannot move with the user. This side effect is important for applications whose users often change machines.

Although cookies are difficult to tamper with, it is still possible for someone to break into them. Remember a cookie is just a text file. If a user can find it and edit it, it can still cause security problems.

Important considerations

Managing state on the client

An application can maintain all of its state on the client-side with any of the methods discussed in the previous section.

- **Benefits of the client-side maintenance**

  On attraction of maintaining state on the client is simplicity. It is easier to keep all the data in one place, and by doing it on the client, it eliminates the need for server database programming and maintenance.

  If an application uses client-side extensions to maintain state, it can also provide a faster response to the user because the need to network access is eliminated.

- **Shortcomings of the client-side maintenance**

If all the state data is on the client-side, there is a danger that users can somehow forge state information by editing URLs, hidden fields, and cookies. This leads to security risks in server programs.

With the exception of the cookie approach to maintaining state, there is no guarantee that the necessary data will be saved when the client exits unexpectedly. Thus, the robustness of the application is compromised.

Managing state on the server

This approach for maintaining state actually involves using both the client and the server. Usually a small piece of information, either a user ID or a session key is stored on the client-side. The server program uses this ID or key to look up the state data in a database.

- **Benefits of the server-side maintenance**

  Maintaining state on the server is more reliable and robust than the client-side maintenance. As long as the client can provide an ID or a key, the user's session state can be restored, even between different browsing sessions.

  Server-side maintenance can result in thin clients. The less dependent a Web database application is on the client, the less code needs to exist on or be transmitted to the client.

  Server-side maintenance also leads to better network efficiency, because only small amounts of data need to be transmitted between the client and the server.

- **Shortcomings of the server-side maintenance**

  The main reason an application would not be developed using server-side state maintenance is its complexity, because it requires the developer to write extensive code. However, the benefits of implementing server-side state management outweigh the additional work required.

Review question 7

- What is state and persistence management in Web database applications?
- What are the technical options available for managing state and persistence?

Security Issues in Web Database Applications

Security risks exist in many areas of a Web database application. This is because the very foundations of the Internet and Web – TCP/IP and HTTP – are very weak with respect to securities. Without special software, all Internet traffic travels in the open and anyone with a little bit skill can intercept data transmission on the Internet. If no measures are taken, there will be many security loopholes that can be explored by malicious users on the Internet.

In general, security issues in Web database applications include the following:

- Data transmission (communication) between the client and the server is not accessible to anyone else except the sender and intended receiver (privacy).
- Data cannot be changed during transmission (integrity).
- The receiver can be sure that the data is from the authenticated sender (authenticity).
- The sender can be sure the receiver is the genuinely intended one (non-fabrication).
- The sender cannot deny he/she sent it (non-repudiation).
- The request from the client should not ask the server to perform illegal or unauthorised actions.
- The data transmitted to the client machine from the server must not be allowed to contain executables that will perform malicious actions.

At the present, there are a number of measures that can be taken to address some of the above issues. These measures are not perfect in the sense that they cannot cover every eventuality, but they should help get rid of some of the loopholes. It must be stressed that security is the most important but least understood aspect of Web database programming. More work still needs to be done to enhance security.

Proxy servers

A proxy server is a system that resides between a Web browser and a Web server. It intercepts all requests to the Web server to determine if it can fulfil the requests itself. If not, it forwards the requests to the Web server.

Due to the fact that the proxy server is between browsers and the Web server, it can be utilised to be a defence for the server.

Firewalls

Because a Web server is open for access by anyone on the Internet, it is normally advised that the server should not be connected to the intranet (i.e., an organisation's internal network). This way, no one can have access to the intranet via the Web server.

However, if a Web application has to use a database on the intranet, then the firewall approach can be used to prevent unauthorised access.

A firewall is a system designed to prevent unauthorised access to or from a private network (intranet). It can be implemented in either hardware, software, or both. All data entering or leaving the intranet (connected to the Internet) must pass through the firewall. They are checked by the firewall system and anything that does not meet the specified security criteria is blocked.

A proxy server can act as a firewall because it intercepts all data in and out, and can also hide the address of the server and intranet.

Digital signatures

A digital signature consists of two pieces of information: a string of bits that is computed from the data (message) that is being signed along with the private key of the requester for the signature.

The signature can be used to verify that the data is from a particular individual or organisation. It has the following properties:

- Its authenticity is verifiable using a computation based on a corresponding public key.
- If the private key is kept secret, the signature cannot be forged.
- It is unique for the data signed. The computation will not produce the same result for two different messages.
- The signed data cannot be changed, otherwise the signature will no longer verify the data as being authentic.

The digital signature technique is very useful for verifying authenticity and maintaining integrity.

Digital certificates

A digital certificate is an attachment to a message used for verifying the sender's authenticity. Such a certificate is obtained from a Certificate Authority (CA), which must be a trust-worthy organisation.

When a user wants to send a message, he/she can apply for a digital certificate from the CA. The CA issues an encrypted certificate containing the applicant's public key and other identification information. The CA makes its own key publicly available.

When the message is received, the recipient uses the CA's public key to decode the digital certificate attached to the message, verifies it as issued by the CA, and then obtains the sender's public key and identification information held within the certificate. With this information, the recipient can send an encrypted reply.

Kerberos

Kerberos is a server of secured usernames and passwords. It provides one centralised security server for all data and resources on the network. Database access, login, authorisation control, and other security measures are centralised on trusted Kerberos servers. The main function is to identify and validate a user.

Secure sockets layer (SSL) and secure HTTP (S-HTTP)

SSL is an encryption protocol developed by Netscape for transmitting private documents over the Internet. It works by using a private key to encrypt data that is to be transferred over the SSL connection. Netscape, Firefox, Chrome and Microsoft IE support SSL.

Another protocol for transmitting data securely over the Internet is called Secure HTTP, a modified version of the standard HTTP protocol. Whereas SSL creates a secure connection between a client and a server, over which any amount of data can be sent securely, S-HTTP is designed to transmit individual messages securely.

In general, the SSL and S-HTTP protocols allow a browser and a server to establish a secure link to transmit information. However, the authenticity of the client (the browser) and the server must be verified. Thus, a key component in the establishment of secure Web sessions using SSL or S-HTTP protocols is the digital certificate. Without authentic and trustworthy certificates, the protocols offer no security at all.

Java security

If Java is used to write the Web database application, then many security measures can be implemented within Java. Three Java components can be utilised for security purposes:

- The class loader: It not only loads each required class and checks it is in the correct format, but also ensures that the application/applet does not violate system security by allocating a namespace. This technique can effectively define security levels for each class and ensure that a class with a lower security clearance can never be in place of a class with a higher clearance.
- The bytecode verifier: Before the Java Virtual Machine (JVM) will allow an application/applet to execute, its code must be verified to ensure: compiled code is correctly formatted; internal stacks will not overflow or underflow; no illegal data conversions will occur; bytecode instructions are correctly typed; and all class member accesses are valid.

- The security manager: An application-specific security manager can be defined within a browser, and any applets downloaded by this browser are subject to its (security manager's) security policies. This can prevent a client from being attacked by dangerous methods.

ActiveX security

For Java, security for the client machine is one of the most important design factors. Java applet programming provides as many features as possible without compromising the security of the client. In contrast, ActiveX's security model places the responsibility for the computer's safety on the user (client). Before a browser downloads an ActiveX control that has not been digitally signed or has been certified by an unknown CA, it displays a dialog box warning the user that this action may not be safe. It is up to the user to decide whether to abort the downloading, or continue and accept a potential damaging consequence.

Review question 8

- What are the major security concerns in Web database applications?
- What are the measures that can be taken to address the security concerns?

Performance issues in Web database applications

Web database applications are very complex, more so than stand-alone or traditional client-server applications. They are a hybrid of technology, vendors, programming languages, and development techniques.

Many factors work together in a Web database application and any one of them can hamper the application's performance. It is crucial to understand the potential bottlenecks in the application as well as to know effective, well-tested solutions to address the problems.

The following is a list of issues concerning performance:

- **Network consistency:** The availability and speed of network connections can significantly affect performance.
- **Client and server resources:** This is the same consideration as in the traditional client-server applications. Memory and CPU are the scarce resources.
- **Database performance:** It is concerned with the overhead for establishing connections, database tuning, and SQL query optimisation.
- **Content delivery:** This is concerned with the content's download time and load time. The size of any content should be minimised to reduce download time; and appropriate format should be chosen for a certain

document (mainly images and graphics) so that load time can be minimised.

- **State maintenance:** It should always minimise the amount of data transferred between client and server and minimise the amount of processing necessary to rebuild the application state.
- **Client-side processing:** If some processing can be carried out on the client-side, it should be done so. Transmitting data to the server for processing that can be done on the client-side will degrade performance.
- **Programming language:** A thorough understanding of the tasks at hand and techniques available can help choose the most suitable language for implementing the application.

Discussion topics

In this chapter, we have studied various approaches for constructing the Web database connectivity, including GUI-based development tools. Both Oracle and Microsoft offer visual development tools. Discuss:

1. What are the pros and cons of using GUI-based tools?
2. Do you prefer programming using APIs or visual tools? Why?