

Introduction to PHP5 with MySQL



Svein Nordbotten

Svein Nordbotten & Associates

Bergen 2009

Preface

This publication is an extract of text and illustrations from an online course offered at several institutions in 2006-2008. The course **curriculum** was *David Sklar: Learning PHP 5 O'Reilly*. (2004) This publication is referred to as **Sklar** in the course.

It is based on a set off applications which all are fully developed and explained.

Bergen 2009,

Svein Nordbotten

Contents

Preface	2
Session 1: Static web applications	6
Basics.....	6
Web applications	6
HTML - Hypertext Markup Language	7
HTML Format	8
Texts.....	8
Links	9
Images.....	10
Lists	11
Tables.....	12
Forms	12
Frames.....	14
CSS, JavaScript and XML.....	15
Session 2: Dynamic applications by PHP	16
Dynamic web sites	16
CGI and PERL	17
Applications Program Interfaces.....	18
PHP Language.....	18
Approach in this course	19
Simple example.....	19
Guessing example	21
Session 3: Dynamic application without database	24
Market research.....	24
System design	24
Market research.....	25
Market analysis	30
Session 4: Introducing the MySQL database.....	33

Dynamic applications and databases.....	33
Creation of a reference database to you personal library.....	33
Menu page	34
Listing the content of the database	37
Searching the database for a book reference.....	39
Deleting rows in the database	44
Removing database.....	45
Session 5: Polling with MySQL database.....	47
Opinion polls	47
Application design.....	47
Database	48
Application menu	48
Creating records and a list of panel members	49
Processing, statistics and rotation	53
Session 6: File processing.....	58
Maintaining files.....	58
Fetching files	59
Uploading files	62
Session 7: Functions in PHP	65
Functions.....	65
Authorization and authentication.....	66
Authorization	70
PIN code assignment.....	70
Authentication	71
Function library	72
Logging	72
Logging function.....	73
Example environment.....	73
Parsing.....	75
Session 8: Information retrieval.....	79
General model.....	79
Index module	80

Search module	84
Administrative module	87
Session 9: e-learning	91
Web courses.....	91
Course architecture.....	91
Authorization and authentication.....	92
Registration and authorization	93
List of content	95
Sessions.....	96
Instructor's tools.	100
Concluding remarks	103
Session 10: Web shop	104
e-shops.....	104
Business promotion	104
Buying products	106
Purchasing products.....	112
A final remark.....	115
A bibliography for further studies.....	116

Session 1: Static web applications

Basics

This session is a **short introduction** to **Hyper-Text Mark-up Language (HTML)** for those not acquainted with this language, and a **fast repetition** for those already experts in the language.

Web applications

The topic of this course is the design and implementation of **web applications**. In this context a web application is a **server-based system** which can interact with the user and respond with several interrelated pages for display at the user's computer.

We distinguish between **2** categories of applications, the **static** and the **dynamic** applications. An application is denoted as **static** if the pages returned have an **invariable** content. In a static application, i.e. the returned pages cannot be modified according to the individual characteristics or behavior of the user. The user makes a request to a host at which the web server processes the request and returns a web page to be displayed at the user's screen. Note that the web server can retrieve a file stored at the host, e.g. a .jpg file, and use it for composing the web page. The basic web server cannot, however, store or modify files submitted by the clients.

A **dynamic** application, on the other hand, can modify its responses by adding to the returned page the name of the user, the number of times this particular user has visited the application web site, her account data, course progress, etc. It requires a **special** program which are able to **additional** processing, for example to process and save data sent by the user or on demand return data stored in a data base to the user. The main objective of this course is to introduce you to the art of developing dynamic web applications.

The **Internet** was initiated in the **1970's** as a further development of the **ARPANET**. The **World Wide Web**, **WWW**, was developed and introduced in **1989** by **Tim Berners-Lee** and **Robert Cailliau** at the **European Particle Physics Laboratory (CERN)** as an **Internet** tool for collaborative knowledge-sharing. It became in short time very popular. **WWW** comprises today a large number of computers which make files available through Internet according to the **HyperText Transfer Protocol, HTTP**. Today, it is estimated that more than **300 M people worldwide** are using the web.

The visible content of a web file is called a **web document**. If a web document is prepared according to the **HTTP** protocol, it can be transferred from a host computer using appropriate software to a requesting client by **Internet**. Most documents are prepared by means of the tag-based language **HyperText Markup Language, HTML**, frequently supplemented with some additional tools. If the requesting client has the necessary **browser** software installed, the file received can be displayed and, if wanted, a new request can be generated, form exemplified by clicking a link in the displayed document.

A **web site** is usually a set of interrelated web-files hosted by a computer running a **web server**. Design and implementation of a web site have several aspects:

- the **topic** of the site
- the **layout** of the pages sent from the site
- the **functionality** of the site

The topics of a web site are varying and depending on the owner's interests and mission. We shall **not** in this course discuss which appropriate pages for web publication are, and which are not. Examples of both interesting and less interesting pages are easily found at the net.

The layout of pages is a fascinating subject. All kinds of **backgrounds colors** and **patterns, fonts** of different kinds and sizes, etc. are among the layout factors from which the designer can choose. Some pages have **animation** and/or **sound** embedded, others include programs transferred to and acting in the client computer. The layout of a page is an important subject because it probably has a significant impact on how the receivers will perceive the page. So far, the layout has to a large extent been determined by the latest hypes and layout rules. The heuristic design rules offered have usually been based on personal opinions and **limited empirical facts**. Large scale investigations of people's perception of alternative layouts are needed. However, layout is **neither** the main subject of this course.

The subject of this course is the **functionality** required to change the web arena from basically **static** to **dynamic** applications. The required functionality is the web site's ability to react on a visitor's behavior over a shorter or longer time period expressed by a series of requests and responses. It is called dynamic because the web pages returned to the client depend on the visitor's previous interaction.

Most web sites are still **static**, i.e. each web page is presented in the same way independent of client and time. **Dynamic** functionality means that the pages returned to the clients can be adjusted to previous input from the individual client and/or time. Development of dynamic web sites can be approached in many ways. In this course, we limit our discussion to the **functionality** based on the scripting language **PHP Language** and on the **PHP Application Engine**. However, before we embark on the dynamic aspects, we shall in this session briefly summarize the **HTML**.

HTML - Hypertext Markup Language

HTML is developed from **SGML Standard Generalized Markup Language** which was approved in **1986** as a standard for marking up documents so they can be stored and read by computers. **HTML** includes only a **smaller** fraction of the features covered by **SGML** and was aimed to be a convenient tool to express pages to be served to the users by **WWW**. The most recent version of **HTML** is **4.01**. An **XML** based version of **HTML 4.01** is **XHTML 1.0**. In this course we refer to the **HTML 4.01** version. To serve these the **HTML** pages, **web servers**, including the **Apache** servers, were developed. For the client side, a number of **browsers** were introduced of which **MS Internet Explorer** and **Netscape** have been the dominating.

The remaining of this session is a **short summary** of the most basic parts of **HTML** needed for this course. For more advanced use of **HTML**, readers are referred to more advanced literature.

HTML Format

To distinguish between the content of the computer **file** sent to the browser and the resulting page **displayed** on the users screen, we shall in this course refer to the former as a **HTML page** and the latter as a **web display**. The **HTML language** is governed by the use of a set of **tags**. A tag is a string surrounded by < and > (e.g. <center>) the following text. In many cases, the tag string is a single character (<p> : start of a new paragraph). Some tags are single such as the tag used for comments (<!-- **Comment** -->) Other tags require a corresponding end tag which is the tag string preceded by a / (</center> : end the centered text). These tags and the included text are called **tag blocks**. Some tags can be nested. There may for example be several paragraphs within a centered text. Many tags include attributes which can be required or optional ()

A complete **HTML** page consists of **several** parts. A typical basic structure may look like (line numbering is included in this and other pages for convenient reference, and should not be included in the page):

1. <!doctype html public "-//w3w//dtd html 4.0 transitional//en">
2. <html>
3. <head>
4. <title>
5. <!-- The title of the document may be typed here --></title>
6. </head>
7. <body>
8. <!-- The specific content of the page is typed in the body-block -->
9. </body>
10. </html>

Type this page and save it in your server with a filename, e.g. **blank.htm**. It can then be called from a client, but since it still has not any content, it will be displayed as a **blank** screen by the browser.

Note that this is the complete frame for an **HTML** page, it will also usually **function** with default specifications with only <html> </html> surrounding your text.

Texts

Let us give the page some **content**:

1. <!doctype html public "-//w3w//dtd html 4.0 transitional//en">
2. <html>
3. <head>
4. <title>text</title>
5. </head>
6. <body>
7. <center>

8. `<h2> About this session </h2></center>`
9. `<p> The purpose of this session is to introduce the course participants to the basic elements of HTML. It is hoped that the introduction will make it possible for the participants to read the HTML pages used in this course, and use the knowledge for preparing their own simple HTML pages in combination with the PHP scripts. </p>`
10. `<center>`
11. `<p> Good luck! </p>`
12. Greetings from `
`
13. the author
14. `</center>`
15. `</body>`
16. `</html>`

This page is named **text.htm** in the example. It illustrates how you can mark **headings** (standard tags are `<h1>`, `<h2>` and `<h3>`), color the text (16 different colors are **predefined**: red, blue, green, blue, etc. and many more are available by code representation), **paragraphs** (`<p>`), **line shift** (`
`) and **center** text (`<center>`).

Links

Hypertext is the trademark of **HTML**. We can easily develop a page which includes a **link** (using the `<a>` and `` tags) to another document, for example the page discussed in the section above. The `<a>` tag requires at least one **attribute**, **href**, the value of which is the name of the file enclosed in **double quotes** to which the link refers.

1. `<!doctype html public "-//w3w//dtd html 4.0 transitional//en">`
2. `<html>`
3. `<head>`
4. `<title>Link</title>`
5. `</head>`
6. `<body>`
7. `<center>`
8. `<h2> Link to the text page </h2></center>`
9. `<p> You may have links to several different destinations in one page. The one which is first clicked will be activated. Click the following link to get to the text page:</p>`
10. `<p>Link to the text page </p>`
11. `</center>`
12. `</body>`
13. `</html>`

Several links in sequence can be created to form a **menu** as in the menu to the **HTML** example of this session.

Images

In the age of **multi-media**, many **HTML** pages have illustrations ([Figure 1.1](#)). A possibility to include pictures in the pages is therefore required. We know from regular work with computers that pictures can be saved in a number of different file formats of which the **.gif** and the **.jpg** are used in connection with **HTML**.



Figure 1.1: A famous painting by Edvard Munch

We assume that we have an image of a well known painting by Edward Munch, the **Scream**, saved in a file named **munch.jpg** in the same folder as we use for our **HTML** pages. We can now write an **HTML** page which includes this image in the returned page for display.

1. `<!doctype html public "-//w3w//dtd html 4.0 transitional//en">`
2. `<html>`
3. `<head>`
4. `<title>image.htm</title>`
5. `</head>`
6. `<body>`
7. `<center>`

8. `<h2>A Munch picture displayed</h2>`
9. `< p>You requested a page displaying a picture by Edvard Munch. Here it is:</p>`
10. ``
11. `</center>`
12. `</body>`
13. `</html>`

The **tag** used is `` which can have several attributes of which **src** refers to the file in which the image is stored, is **required**. You can easily scale the picture by changing the attributes **width** and **height** in the image tag. The **metric** unit used is **pixels**. The **position** of the picture within the displayed page can be controlled by the attribute **align** with a number of possible **alternative values** (**left, middle, right, top, bottom**, a. o.). Note that the scaling and the positioning attributes are **optional**.

Lists

We have got used to the ability of modern word processor to prepare **numbered** and **unnumbered list**. The **HTML** has included this ability by the tag pairs ` ` and ` `.

The page in this example can serve as an illustration of this capability:

1. `<!doctype html public "-//w3w//dtd html 4.0 transitional//en">`
2. `<html>`
3. `<head>`
4. `<title>list.htm</title>`
5. `</head>`
6. `<body>`
7. `<center>`
8. `<h2>Menu for the example options</h2>`
9. `<p>This example illustrate the basic features of HTML which are:</p>`
10. ``
11. `Blank page`
12. `Text page`
13. `Page with link`
14. `Page with picture`
15. `Page with table`
16. `Form page`
17. `Frame page`
18. ``
19. `</center>`
20. `</body>`
21. `</html>`

The `` and `` tags **delimit** the individual elements, or lines, in the list. Note that in this page we use the **unnumbered** `` tag. By changing the start and end tag to `` and ``, the elements would be **numbered** consecutively from **1** and up.

Tables

The table tag, `<table>`, is very useful in several ways for presenting **one- (a list) and two-dimensional tables** with or without borders. When you consider the display of the menu in the previous example, it gives an unordered impression. Use of the table tag with associated tags can make it more **orderly**. Consider the following page which presents the list as a **one-dimensional table**:

```
1. <!doctype html public "-//w3w//dtd html 4.0 transitional//en">
2. <html>
3. <head>
4. <title>table.htm</title>
5. </head>
6. <body>
7. <center>
8. <h2><font color="#0000FF"><b>Menu for the example options<b></b></b></font></h2>
9. <p>This example illustrate the basic features of HTML which are:</p>
10. <table>
11. <tr><td><a href="blank.htm">1. Blank page</a></td></tr>
12. <tr><td><a href="text.htm">2. Text page</a></td></tr>
13. <tr><td><a href="link.htm">3. Page with link</a></td></tr>
14. <tr><td><a href="image.htm">4. Page with picture</a></td></tr>
15. <tr><td><a href="table.htm">5. Page with table</a></td></tr>
16. <tr><td><a href="form.htm">6. Form page</a> </td></tr>
17. <tr><td><a href="frame.htm">7. numbered Frame page</a></td></tr>
18. </table>
19. </center>
20. </body>
21. </html>
22. </table>
```

In addition to the `<table>` tag, we use the tags `<tr>` and `</tr>` to delimit a table **row**, and the tags `<td>` and `</td>` to mark an **element** in the row. In this example there is only one element per row, usually there are several. In **regular** tables there is always one element per column in each row. If the cell is empty it is marked by `<td></td>`.

In regular tables, there is usually also a **header** row with **column names**. The column names are marked with the tags `<th>` and `<(th>`. Each of the table tags can include one or **optional attributes** for defining **size**, **alignment**, **fonts**, **border**, etc. making the tags very flexible and useful.

Forms

One of the most **important properties** of **HTML** is the `<form>` tag which permits **sending** data to the server. This tag is the key to **combining HTML** and the **PHP** language to a tool for **creating dynamic** applications. The `<form>` tag makes it possible to create pages for the user with different **types of input** (radio buttons, check boxes, texts, files, etc) and send the input for further **processing by the server** according to a specified program, for example a **PHP** script. Note that

HTML itself has **no** facility for processing data on the server. (There are extensions of **HTML** which permit limited processing at the server).

We shall see a number of applications in the following sessions based on **interaction** between **HTML** and **PHP** scripts. For illustration of the **<form>** tag in this session, a form will be discussed and at the accepting server side a very simple **PHP** script will return a message confirming the submitted information.

The HTML form page looks like this:

```
1. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2. <html>
3. <head>
4. <title>form.htm</title>
5. </head>
6. <body>
7. <center>
8. <h2><font color="#0000FF">A form for sending a file for processing</font></h2>
9. <p>This form can be used for sending a file for alternative processing, Action A or B, and assumes a
   processing script at the server. In this example, the only action taken by the server is to return a
   message acknowledging the received file and message.</p>
10. <form action="acknowledge.php" method="post">
11. <table>
12. <tr><td>Message:</td><td><input name="message" type="text"></td></tr>
13. <tr><td>File:</td><td><input name="testfile" type="file" enctype="multipart/form-data"></td></tr>
14. <tr><td>Action A:</td><td><input name="processing" type="radio" value="A"></td></tr>
15. <tr><td>Action B:</td><td><input name="processing" type="radio" value="B"></td></tr>
16. <tr><td></td><td><input name="" type="submit" value="Submit file"></td></tr>
17. </table>
18. </form>
19. You can either use any <b>.htm</b> or <b>.doc</b> file you have on your client.
20. </center>
21. </body>
22. </html>
```

The **form** tag appears on **Line 10**. In this form, **2 attributes** are used, the **action**, which specify the **PHP** script for processing the submitted information, and the **method** determining which way the information should be transferred. Note that we must use the **post** method, why will be explain in a later session. We also postpone the discussion of the **PHP** script, **acknowledge.php**, to the next session.

The form **type of content** is determined by the **<input>** tags in **Lines 12 -16**. All input tags have **2 attributes** in common, the **name** and the **type** of input. As long as the **name** is not yet used, it can be chosen quite **freely** (avoid special characters and blanks). Available values of the **type** are **text**, **password**, **radio**, **checkbox**, **file**, **image**, and **submit**. For **type="file"** there is also a third attribute, **enctype**. For all types that are **optional** attributes which can determine the **size** of the fields for giving answers.

Input tags of **type="submit"** are special. They do not require any name specified, but you can text the submit button by means of the **value** attribute.

The form script can contain other tags than **<input>** as the **<select>** tag to create **menus**, **<textarea>** for creating an area into which the user can provide a **longer text**, and others.

Frames

The last feature of **HTML** we want to cover in this introduction is the **frames**. We have in the examples above developed a menu page from which we can select the special feature we want to be demonstrated. However, after the first demonstration, we have to use the Back button to find the menu again. We therefore need a way to **divide the screen** into **2** windows, one showing the menu **permanently** and the other displaying the **topic** selected for demonstration.

The frame feature of **HTML** permit us to divide the screen into **2** or more windows, all visible and active at the same time. This feature uses **2** tags, **<frameset>** and **<frame>**. The page below generates the effect we want.

1. `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"`
`"http://www.w3.org/TR/html4/frameset.dtd">`
2. `<html>`
3. `<head>`
4. `<title>frame.htm</title>`
5. `</head>`
6. `<frameset cols="20%,*" frameborder="yes" border="1" framespacing="0">`
7. `<frame src="table2.htm" name="leftFrame">`
8. `<frame src="blank.htm" name="mainFrame">`
9. `</frameset>`
10. `</html>`

Note the **difference** in the **<!doctype>** tag from those used in previous **HTML** pages and that the **<body>** tag is not included.

Lines 6 - 9 **specify** a frame set, because the **<frameset>** tag has **4** attributes, **cols**, with the value **"20%,*"** divides the width of the client's screen in **2** windows by a vertical border assigning **20%** of the screen to the left window and the rest to the right window, **frame border** and **border**, specifying a visible border of size **1**, and finally **frame spacing** which is set to **0**.

Inside the frame set block there are **2** **<frame>** tags, one for each window. They have both **3** attributes which specify the **src**, i.e. the file to provide content to and the **name** of the respective window. This page generates the **2** windows and their initial content (the right window is empty because it is generated by **blank.htm**. To understand how the further content of the windows is created, we need to look at a **modified** version of **table.htm** called **table2.htm** (only the part within the **<body>** block is reproduced):

1. <center>
2. <h2>Menu for the example options</h2>
3. <p>This example illustrate the basic features of HTML which are:</p>
4. <table>
5. <tr><td>1. Blank page</td></tr>
6. <tr><td>2. Text page</td></tr>
7. <tr><td>3. Page with link</td></tr>
8. <tr><td>4. Page with picture</td></tr>
9. <tr><td>5. Page with table</td></tr>
10. <tr><td>6. Form page</td></tr>
11. </table>
12. </center>

The only **difference** from the original table.htm is the inclusion of the argument **target** with value "**mainFrame**" in the <a> tags of **Lines 5 -10**. The target directs the browser to display the link in the window named **mainFrame**, i.e. the right hand window.

CSS, JavaScript and XML

The tool case for preparing web documents contains a number of useful objects. Close to **HTML** are **Cascading Style Sheets (CSS)**, **JavaScript** and **eXtensible Markup Language (XML)**.

CSS was developed for use with **HTML** and introduced in **1996**, and is implemented in most browsers.

Session 2: Dynamic applications by PHP

Dynamic web sites

The static model of the web interaction is based on a set of **pre-developed** static web pages stored on a host server, and in **3** basic steps: .

1. A client sends a **request** for a web page to the host.
2. The host sends a **copy** of the requested page to the client.
3. If desired, points 2 and 3 are **repeated** for new pages.

A node in the web which manages the host tasks is called a **web server**. In the static model, [Figure 2.1](#), the host has **no ability** to analyze the request and adjust the response accordingly. The

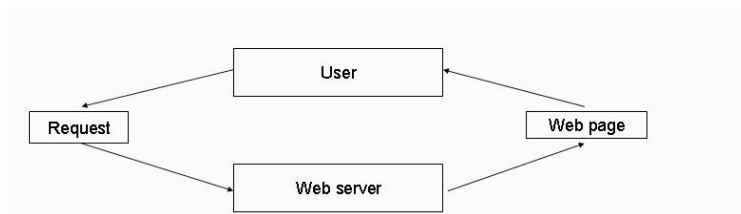


Figure 2.1: Static web server

response is a requested pre-designed web page. The request-response exchange is therefore called static. However, the exchange protocol used, **HTTP**, provides possibilities for some **additional** items of information to be sent to the host with the **request** without any instructions from the client. In the same way, the responding host can include additional information with the **response**, usually hidden for the receiver. The host has also capabilities to forward messages to other programs beyond the web server for additional processing. These possibilities for information processing **behind** the scene make it possible to create the additional functionality.

We shall use the term **dynamic web site** to emphasize that we are not concerned with a simple set of web pages with **HTML** tags, but with applications in which the pages returned to the client can be dynamically adjusted to fit the individual requests of the client. This course can serve as a first illustration of a dynamic web site. You have already experienced that when you submitted your **personal access code** entering this course, the system became accessible to you. If you had submitted an invalid access code, however, the host would have sent you a message adjusted to an unacceptable access code it received from you. The system must be able to compare your identity with a pre-loaded list of authorized identities. You will soon also see that if you try to go on to the **next session** before the time it is officially opened; you will receive a message regretting that the session is not yet open. When opening **date** is passed, and you have passed the

test at the end of the session, the system will respond by giving you access to the session. The system must be able to compare your request with its **clock time** and with your recorded test performance. If a student has not yet **completed** the required test, the host will return a message that the test must be done before the student can proceed. This means that the system must be able to **keep track** of your previous interactions.

Important characteristics of a dynamic web site are the ability to **authenticate** you, i.e. to verify your identity, to **record** your performance history, to react on the **time** for the request, to keep track of your **interactions** from you start a session and to its end, and sometimes even from session to session. The dynamic web site can be summarized by [Figure 2.2](#).

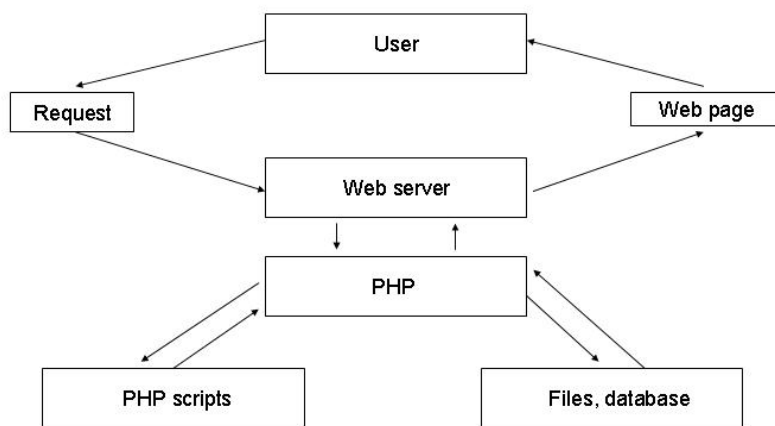


Figure 2.2: Diagram for a dynamic application

CGI and PERL

The **first** step toward dynamic web pages is the possibility for a remote client to request the execution of a process at the host. Use of the **FORM** tags of **HTML** requires, for example, that the server can perform a processing of the data submitted on the form. A program must exist for this purpose at the host site, and the web server must be able to communicate with this. We shall refer to such a program which supplements the **HTML pages** as a **script**.

The **Common Gateway Interface**, **CGI**, is a protocol specifying how certain scripts can communicate with web servers. One of the most frequently used tools for creating such scripts is the script language **PERL**. A **PERL** script stored in the host computer can be supplied with data from a request, for example sent by a **HTML FORM** page. The script can be designed to perform a variety of tasks such as save and retrieve data from a database, update a log, keep track of visitors, run a course, etc. It can also be designed to perform its task and then leave the result to the web server, which returns a web page generated by means of the script to the requesting client. Programming languages such as **C**, **C++**, **C#** and **JAVA** can also be used for creating

scripts. One reason for the popularity of **PERL** is that scripts programmed in **PERL** can be ported from one **operating** system to another with no or little modification.

Applications Program Interfaces

A **PERL-CGI** application is time-consuming because **PERL** scripts must be loaded, executed and unloaded **each time** they are used like interpretive programs, and do not offer the flexibility which may be required.

To improve this situation, **Application Servers** were developed. An application server is a **service** operating behind the web server. It processes script code, which the web server does not understand, and returns the results to the web server for returning to the requesting client. The applications server is a resource of permanently loaded executable programs, and is referred to as an **Applications Program Interface**, **API**. The advantages of using an **API** compared with the earlier interpretative programs are **increased** speed and flexibility because no loading and interpretation is needed. The **disadvantage** is that the **API** programs must be implemented and compiled **separately** for each type of operating system, and requires more **memory** space.

PHP Language

The well-known **API** tools include the **ASP** and **ASP.NET** from **Microsoft**, the open source system **PHP**, **iHTML** from **Inline Internet Systems**, and **ColdFusion MX** from **Macromedia**. In this course, we are leaving the comparisons between the tools to evaluators and sales people, and concentrate on **PHP** because it is an open source tool, easily available and supported by a large community of users. **PHP** was introduced in 1995 as Personal Home Pages. Since then, PHP has been developed to a very powerful tool for treating dynamic web sites.

The language by which we design our scripts is the **PHP Language**. Files in which scripts are saved are recognized by their extensions, **.php**. You are referred to the section [Software](#) to get detailed instructions for installing necessary software on your own **PC** to be able to develop and test your dynamic sites.

In the previous paragraph, the advantage of using a web **API** instead of an interpretive approach was emphasized. **PHP** was introduced on the market in **1995**. It started out as a **scripting** language based on **CGI**. Later, the **API** was developed. The current version is **PHP 5** which is a powerful system with an embedded **database** system, **SQLite**. Be certain that you have the **PHP 5** version installed.

PHP is widely used by individuals and enterprises among which there exist an active interchange of software and experience.

Approach in this course

Most courses and textbooks on programming and scripting languages start with the introduction of the language **syntax**. We shall take another approach, **learning by examples**, i.e. in each session we shall introduce a set of problems with their live solutions, and explain the syntax required by the examples. In parallel with studying the examples and the text, the student should read the relevant parts of the course **textbook** to make certain that (s)he will acquire the precise details of the language syntax.

Simple example

Imagine an application requiring **registration** of some personal data from visitors and which should be returned as confirmation of accepted data. This simple task cannot be done by use of **HTML** only because the response must be adjusted to the submitted data. [Figure 2.3](#) outlines the application in a diagram. The diagram indicates how the communications between the user and the host pass through the web server to the PHP scripts because the server cannot process the in data but is needed to return the web pages to the user for display. To summarize the task:

1. Design a **HTML** form for acquiring the required data
2. Develop a **PHP** script for returning a confirmation of received data

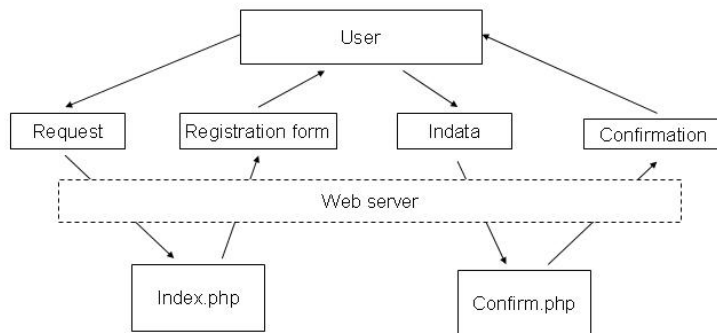


Figure 2.3: Diagram for the registration

The development of a **HTML** form, may result in a typical file as::

1. `<html>`
2. `<head>`
3. `<title>Registration</title>`
4. `<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">`
5. `</head>`
6. `<body>`
7. `<!-- index.htm -->`
8. `<center>`


```

9. <h2><font color="blue">Registration form</font></h2>
10. <form action="confirm.php" method="post">
11. <table>
12. <tr><td>First name:</td><td><input name="FirstName" type="text" size="30"></td></tr>
13. <tr><td>Last name:</td><td><input name="LastName" type="text" size="30"></td></tr>
14. <tr><td>Email adress:</td><td><input name="Email" type="text" size="28"></td></tr>
15. <tr><td></td><td><input name="" type="submit" value="Submit"></td></tr>
16. </table>
17. </form>
18. </center>
19. </body>
20. </html>

```

The page code specifies **3** text input fields and a submit button. There is nothing special with this code. Note that the statement in **Line 10** has an **ACTION** attribute with value **confirm.php** implying that the control is transferred to a **PHP** file. This tells us that an application can consist of a mixture of **.htm** and **.php** files. By convention in this course, the **first file** of any example is either named **index.htm** or **index.php**.

The purpose of the next file, **confirm.php**, is to instruct the server to return a confirmation for the received data. Except for a comment line including the name of the file, it contains a short **PHP** script. A **PHP** script is recognized by the start tag **<?php** and the end tag **?>**. The script is short, but introduces several basic **PHP Language** characteristics.

```

1. <!-- confirm.php -->
2. <?php
3. print("<center>");
4. print("<h3><font color=blue>The following data have been received:</font></h3>");
5. print("<table>");
6. print("<tr><td>First name:</td><td>$_POST[FirstName]</td></tr>");
7. print("<tr><td>Last name: </td><td>$_POST[LastName]</td></tr>");
8. print("<tr><td>Email address:</td> <td>$_POST[Email]</td></tr>");
9. print("</table>");
10. print("</center>");
11. ?>

```

These are:

- Each **PHP** statement line ends with **semicolon**.
- **print()** functions are used to return a message to the client.
- **\$_POST[]** elements are used to refer to values submitted in a form.

Let us first explain the **\$_POST[]**. All **variables** in **PHP** are recognized by **\$** as their first character of their name. A name followed by **[]** indicates an **array**, and the content of the square parentheses refers to the **key** for the element the value of which is contained in the expression. **\$_POST[]** is an **auto-global** array in which all variable values submitted in a **HTML FORM** tag with **METHOD="post"**. This means that the PHP automatically stores the variables in the array and that they are available to all parts of the application.

Note that in **PHP** the elements of the returned **HTML** page must be enclosed as **arguments** surrounded by **double quotes** in the print functions. If you replace the `$_POST[]` variables with the values they contain, remove the print, parentheses, quotes and semicolons, you get the HTML page returned to the client by the web server. See the Source code in your browser when running the example.

Guessing example

We shall advance our PHP demonstration by selecting a well known type of arithmetic **guessing** game. The task of the game is to guess the **sum** of all integers up to a **random number**. The server will, on your request, return a page containing the **upper limit number**, one box to fill with your name, another to fill with the **sum** you guess, and a button for submitting the guess to the server. When received, the server will compare your guess with the computed sum, and return an **personalized** result page to you. [Figure 2.4](#) displays the interaction between the server and the client. Even though this example is by topic quite different from the previous the general application structure is similar.

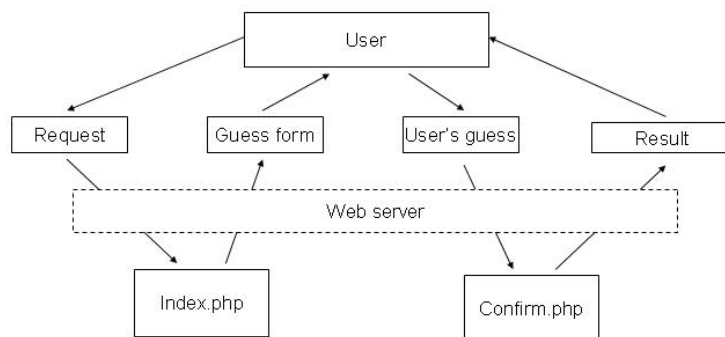


Figure 2.4 : Diagram for the guess application

The dynamic features of this scenario are interesting. On request, the client is **identified**, the upper limit number is **computed** and **memorized** by the server. When the response is accepted, the server **recognizes** the client, **compares** her/his answer with the stored computed sum, and **reports** back to the client. In other words, the server is able to combine **two (multiple)** requests from the same client, and **adjust** the response to the last contact to the content of previous requests.

Now, let us develop the solution to this problem. We obviously need:

1. A **PHP** script to select an upper limit number and the corresponding sum of integers from 1 up to the limit, and to compose a **HTML** form to return to the client adjusted for the upper limit number selected.
2. A second **PHP** script that can compare the content of the returned form with the previously computed sum, and compose a relevant response.

The first script is a mixture of **2 PHP** blocks and a **HTML** page. It can look as follows:

```

1. <!-- index.php -->
2. <?php
3. srand();
4. $_SESSION['randval']=rand(10,100);
5. ?>
6. <center>
7. <h2><font color="Red">Guess!</font></h2>
8. <form action="response. php" method="post">
9. <table>
10. <tr><td>My name is:</td> <td><input type="text" name="name"></td></tr>
11. <tr><td>I guess the sum of all integers from <b>1</b> to <?php print("<b> $_SESSION[randval] </b>")
    ?> is:<td><input type="text" name="guess"></td></tr>
12. <tr><td></td><td><input type="submit" value="Submit"></td></tr>
13. </table>
14. </form>
15. </center>

```

Since this file contains **PHP** code, it must be named **index.php**. The first **PHP** script block is on **Lines 2-5**. **Line 3** is a **PHP** function used to get a **random seed** for the function on the next line.

Functions in **PHP** always consist of a **name** followed by a **pair of parentheses**. The parentheses may be empty, as in this case, or contain one or more arguments. The **srand()** function requires no arguments. It is not necessary to assign the results of this function to any variable because we shall not need to refer explicitly to the seed.

Line 4 assigns the outcome of a second built in function, **rand(10,100)**. This function, which requires **2** arguments, generates a random integer between **10** and **100**, and assign the result to the variable, **\$_SESSION['randvalue']**. From the previous example we know that the **\$** means that it is a variable, and the **[]** indicates that the variable is stored in an array.

The **SESSION[]** array is used to **store** all variables which we want to access at different occasions during a session. Like the **\$_POST[]** array, the **\$_SESSION[]** array variables are **super global** variables (**It is assumed that you during configuration set the session_auto_start to ON in the php.ini during installation**). The session variables are kept for a default period up to **1440** second (**24** minutes) after which the session cookie expires. The server's **recognition** of a client is obtained by means of **cookies**, which is returned with the server's response to the first request from the client, and then connected to all requests from the client within the session.

Following the first **PHP** block, is a **HTML** form with an embedded **PHP** block, **<?php print(" \$_SESSION[randval] ") ?>** **embedded** in **Line 11**. The reason for including

this script in the middle of an **HTML** expression is that we want to include the **PHP** variable `$_SESSION[randvalue]` to be displayed for the client. Note that within double quotes, as in the print argument, single quotes are not used around the array keys, e.g. in `$_SESSION[randvalue]`.

The form calls upon the second script, **response.php**, which follows.

```
1. <!-- response.php -->
2. <?php
3. $sum="0";
4. for ($count=1;$count<=$_SESSION['randval'];$count++) {
5.     $sum=$sum + $count;
6. }
7. if ($sum == $_POST['guess']) {
8.     echo "$_POST[name], your guess was correct!";
9. }
10. elseif ($sum > $_POST['guess']) {
11.     echo "Sorry, $_POST[name], your guess <b>$_POST[guess]</b> is too low, the correct sum is
        <b>$sum</b>";
12. }
13. else {
14.     echo "Sorry, $_POST[name], your guess <b>$_POST[guess]</b> is too high, the correct sum is
        <b>$sum</b>";
15. }
16. ?>
```

The **Lines 3 - 5** compute the **correct** sum associated with the generated **upper limit integer**, `$_SESSION[randval]` by looping through a **for** loop with an index variable named `$count` which is increased by **1** using the incremental operator `++`, and for each loop the `$sum` is increased by the current index number.

Lines 7 – 16 contain a test of the guess submitted, and **return** an answer to the client. Three alternatives are possible: **Line 8** will be sent as an **HTML** page to the client if the sum guessed is **correct**, **elseif** the guess is **less** than the correct sum, **Line 11** will be executed, and, finally, if the guess is too **high**, **Line 1** is used for response to the client.

The last script illustrates how **PHP** can solve dynamic tasks by using `$_POST[]` and `$_SESSION[]` variables. Both these arrays contain global variables, i.e. variables which are **persistent** during the client's session, an important requirement for dynamic application development.

Session 3: Dynamic application without database

Market research

In this session, the introduction of **PHP** will be continued, and the scenario we shall use is online collection of data for **market research**. The marketing problem concerns **2** products, **A** and **B**, and we are interested in measuring the consumers' relative **preferences** for the two competing products. However, we have a suspicion that the respondents may have a tendency to vote for the product **listed first**. We want to randomize the sequence, i.e., **AB** and **BA**, to eliminate this effect. The **persistence of the preference** is another question we want to study. For this reason, we want the respondents to repeat their preference vote a certain time, e.g. a week, after the first vote. To attract consumers to provide their votes of preference, those who complete the **2** votes are eligible for participation in a **lottery**.

A file of responses must be built in which the **2** votes of the individual consumers can be connected by mean of a **unique identifier** for comparing responses as well as a file with name and addresses for those who are eligible for lottery participation.

Since this is a course focusing on design and development of dynamic web sites, the important questions about how to obtain **representative participants** and how many, are ignored. Also the questions about the evaluation of the reliability of the results are considered outside our scope in this course.

System design

[Figure 3.1](#) gives an **overall** idea about how we want to solve the task outlined above. There are **2**

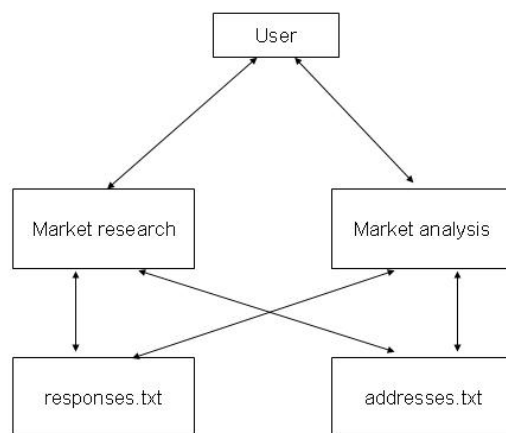


Figure 3.1: Market research

application parts which are both connected to the same 2 data files. The implementation will consist of sets of **.php**, **.htm** and **.txt** files demonstrating how it is possible to **mix different types of files** in the same application which will be presented and discussed in the following order:

Market_research:

- index.php
- prepare.php
- save.php
- form3.htm
- save3.php

Market_analysis:

- index.htm
- report.php
- report2.htm

Common text files:

- responses.txt
- addresses.txt

We use the convention introduced in the first session, and name the first file **index.php** which give us the advantage that we can open the application by calling the folder in which all the files reside. The files reflect the 3 sets of files, the **user module**, **market_research**, the **administration module**, **market_analysis**, and the **data files** as outlined in [Figure 3.1](#). In addition, some **global arrays of variables** (i.e., **\$_POST[]**, **\$_SESSION[]** and **\$_COOKIE[]**) exist for creating persistency in the application.

The 2 **.txt** files do not exist initially. but is generated when the first data are collected.

Market research

[Figure 3.2](#) gives a **simplified** picture of the Market Research part of the application. The **index**

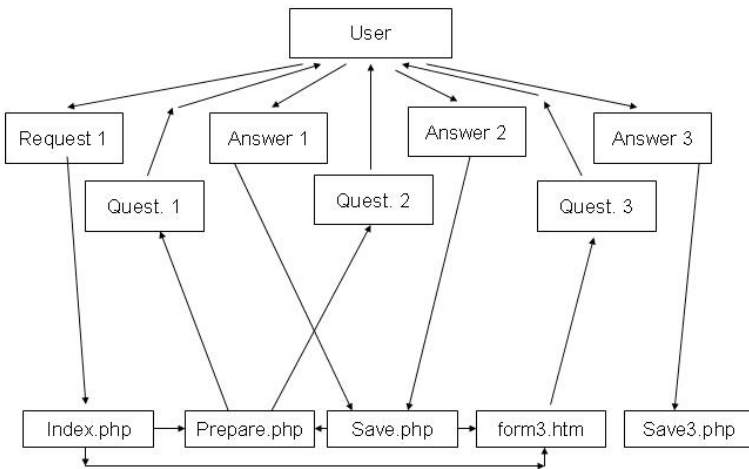


Figure 3.2: Market research diagram

page is used to send the users instructions about what to do. Since there are 3 questionnaires to be completed by the participants in the research survey, we define a variable, `$_SESSION['marker']`, in Line 4 to keep track of which questionnaire is the current. This variable is an element of a global array used for making variable values persistent for access in several scripts during a session. Line 3 test by means of a function `isset()` if `$_SESSION['marker']` has been defined, and if not define and set the variable to 1.

Next, observe that except for the 3 if statements, the remaining of this PHP script prepares 3 alternative HTML pages for display using the `print()` function. Line 6 prepares the first common part of the HTML page to be returned to the client, while the Lines 7, 10 and 13 test which questionnaire should be offered the user. Depending on the value of the marker variable, Line 8, 11 or 14 is sent with a tag linking to the appropriate questionnaire. If you study these print statements carefully, you may be surprised by noticing that there are no double quotes around the files to which the A tags refer. Expressions already enclosed in the double quotes of a PHP statements, should not contain any double quotes.

```

1. <!-- index.php -->
2. <?php
3. if(!isset($_SESSION['marker'])) {
4.     $_SESSION['marker']=1;
5. }
6. print("<center><h2><font color=Blue>Market research</font></h2></center>p>This is a market
   research to investigate the public's preferences for Product A and Product B. If you respond and
   complete two questionnaires, you will be eligible to participate in a lottery. The requirements
   are:</p>");
7. if($_SESSION['marker']==1) {
8.     print("<b>Request, complete and submit <a href=prepare.php>questionnaire 1</a></b>");
9. }
10. if($_SESSION['marker']==2) {
11.     print("<b>Request, complete and submit <a href=prepare.php>questionnaire 2</a></b>");
12. }
13. if($_SESSION['marker']==3) {

```



```

14. print("<b>Request, complete and submit <a href=form3.htm>questionnaire 3</a></b>");
15. }
16. print(" <p>The 2 first questionnaires require you make a single click only before you submit your
    response. The third questionnaire asks for you e-mail address for notification in case you become a
    winner in the lottery.</p><p><i>The market research sets a time-limited cookie in your
    browser.</i></p>");
17. ?>

```

index.php does not contain any other new **PHP** features, and we can proceed to the next script, which is the **prepare.php**. The purpose of this script is to **prepare** the **3** different questionnaires and keep track of which should be served.

Already the first line introduces an important new feature, i.e. the use of **cookies**. A cookie is a small message sent from the server to the client browser providing the receiving client with a **unique identifier**, a time- out specification and an identification of the server issuing the cookie. Cookies are kept in a special list in the computer, and deleted when timed out.

Before a browser sends a request to a server, its list of cookies is checked for any cookies from the server approached. If a relevant cookie is found, it is copied and **attached** to the request. The server receiving a request and scans the request for cookies. When a cookie is detected, the server has received a user identification.

In this way, it is possible to link items in a chain of interactions between the server and an individual client. Since the server is issuing and distributing cookies, the **anonymity** of the client can be maintained. In our application, we ask the client to answer our questions at **2** different occasions. We can link the answers by means of a cookie without inquiring about the name or other identification from the client assuming the client is using the same computer and is the only user of the computer.

In **Line 3 - 6**, we ask if the requesting client has a cookie called **user_id**, and, if not, prepare in Line 4 a cookie to be returned to the client with the response to its request. The name of the cookie to be sent is **user_id** and its unique value is the **exact time** obtained by the **PHP** function **time()** in **Line 4**, at the moment the cookie is set. In the function **setcookie()**, we specify the name of this cookie, in our application **user_id**, the value of the cookie, and the **lifetime** of the cookie. The value of the **user_id** must of course be **unique**. One way of obtaining such a value is to use the time when the cookie was set. This time value is available from the function **time()** in **Line 4**. This value is also used in the third argument in which a number of seconds are added to determine when the cookie should be deleted. In our particular application, the second preference form should be answered one week after the first at which the cookie is set and the timeout point should be **8** days later.

It is important that only one cookie is set for each visitor, and for that reason a test is made in **Line 3** for the existence of the particular application cookie, **\$_COOKIE['user_id']**. If it already is set, **Line 4** and **5** are not executed.


```

1. <!-- prepare.php -->
2. <?php
3. if(!isset($_COOKIE['user_id'])) {
4.     $time=time();
5.     setcookie('user_id', "$time", "$time" + 60);
6. }
7. rand();
8. $randval=rand(1,2);
9. $_SESSION['marker']++;
10. print("<center><h2><font color=Blue>Preference for products</font></h2></center><p>Thank you for
    visiting this page and expressing your opinion. Complete and submit this form. The second
    questionnaire should be completed one week after the first.</p> <FORM ACTION=save.php
    method=post>
11. <p>Please mark your preference by clicking a button. Comparing the 2 products A and B, I
    prefer:</p>");
12. if($randval == "1") {
13.     print("<p><INPUT TYPE=Radio NAME=preference VALUE=A> Product A</p>
14.     <p><INPUT TYPE=Radio NAME=preference VALUE=B> Product B</p><INPUT YPE=hidden
        NAME=form_type VALUE=1>");
15. }
16. else{
17.     print("<p><INPUT TYPE=Radio NAME=preference VALUE=B> Product B</p>
18.     <p><INPUT TYPE=Radio NAME=preference VALUE=A> Product A</p><INPUT TYPE=hidden
        NAME=form_type VALUE=2>");
19. }
20. print("<p><INPUT TYPE=submit NAME=SUBMIT VALUE=Submit></p>
21. </FORM>");
22. ?>

```

The last point to be mentioned is the **incremental** operator ++ used in **Line 9** well known from other languages. This line is equivalent to the longer statement `$_SESSION['marker']=$_SESSION['marker'] + 1;`. The questionnaires transformed to **HTML** form in **Line 10, 13** and **17** are served to the clients depending on the value of `$_SESSION['marker']` incremented in this way.

The returned responses from the clients are taken care of by the script **save.php**. The answers to the questionnaires **1** and **2** are saved in **response.txt**. If the file does not exist, it is established by the **PHP** function **touch()**. Before any file can be operated on, it must be opened by means of the **fopen()** function which requires **2** arguments, the file name and the action. There are **2** write action available, **write from the beginning** and **append to the end** of the file indicated by "**w**" and "**a**", respectively. In **Line 6** the **response.txt** is opened for append of data. The **fopen()** returns a **handle or reference**, in this script called **\$f**, and which is used in the file action function **fwrite()** in **Line 7**.

```

1. <!-- save.php -->
2. <?php
3. if(! file_exists("response.txt")) {
4.     touch("response.txt");
5. }
6. $f=fopen("response.txt", "a");

```


7. `fwrite($f,"User id: $_COOKIE[user_id], Form type: $_POST[form_type], Preference: $_POST[preference]\n");`
8. `?>`
9. `<center>`
10. `<p> Return to introduction.</p>`
11. `</center>`

fwrite() is the **PHP** function used for writing to a file. It requires **2** arguments, the file handle and a string of the items to be written to the file. In **Line 7**, the write function in this script, the first argument is the file handler **\$f** just established in the previous line, and a string of **3** name/value pairs for **User_id**, **Form type** and **Preference**, all with global variable values and delimited by commas. These are the **3** items in which we are interested in. Note the end of line symbol, **\n**, at the end of the string to get a line shift after each record.

In the example, you can return to complete questionnaire **2** immediately. In a **real application** in which we would like to observe the preference change during a week, we would design some kind of program which would remind the client about the second questionnaire in a week.

The registration of the participants of the lottery, implemented in **form3.htm**, is sent to the client when both data collection forms have been returned is a simple form calling **save2.php**. It **returns** the submitted name and email address by **METHOD="post"** in order to be easily available by **save2.php**.

1. `<!-- form3.htm -->`
2. `<center><h2>Your e-mail address</h2></center>`
3. `<p>If you are eligible for participating in the lottery, i.e. you have requested, completed and submitted the two questionnaires, we need your e-mail address to notify you in case you become a winner in the lottery.</p>`
4. `<form action="save2.php" method="post">`
5. `<table>`
6. `<tr><td>Your name:</td><td><input type="Text" name="myname"></td></tr>`
7. `<tr><td>e-mail address:</td><td><input type="Text" name="myemail"></td></tr>`
8. `<tr><td></td><td><input type="Submit" value="Submit"></td></tr>`
9. `</table>`
10. `</form>`

The **save2.php** script is very similar to the already discussed **save.php**. It resets the marker to the initial value 1 and thanks the client for his/her participation.

1. `<!-- save2.php -->`
2. `<?php`
3. `if(! file_exists("address.txt")) {`
4. `touch("address.txt");`
5. `}`
6. `$a=fopen("address.txt","a");`
7. `fwrite($a,"user id: $_COOKIE[user_id], Name: $_POST[myname], Email address: $_POST[myemail]\n ");`
8. `$_SESSION['marker']=1;`

9. ?>
10. <center>
11. <p> Thank you for your participation. You will be included in the lottery.</p>
12. </center>

Market analysis

The scripts so far have been aimed at collecting the preferences of the participants of the market research survey. However, we need also to have **tools for retrieving** the collected data for analysis. The second part of the application in this session is named **Market_analysis** (In fact, it is not an analysis, but a data retrieval): [Figure 3.3](#) gives an overview of this part of the

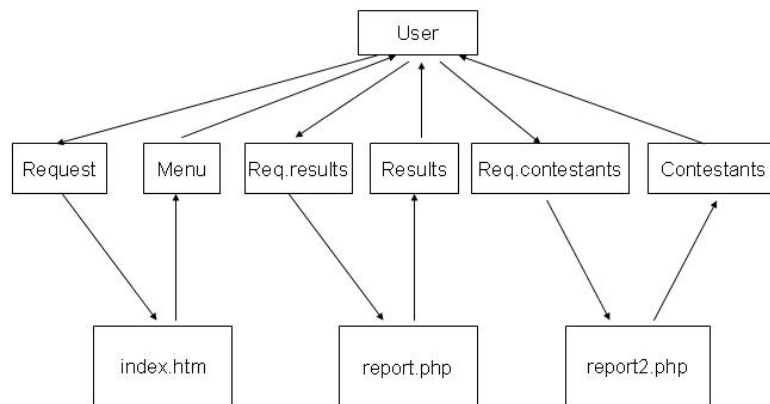


Figure 3.3: Market analysis

application. It starts with a simple **HTML** page, **index.htm**, which offers **2** options. The first activates the script **report.php** which prints the text file **response.txt**, the second starts **report2.php** which prints the text file **address.txt**.

The **index.htm** is an ordinary **HTML** page which links the **2** alternative scripts, **report.php** and **report2.php**.

1. <!-- index.htm -->
2. <html>
3. <head>
4. <title>index.htm</title>


```

5. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
6. </head>
7. <body>
8. <center>
9. <h3><font color="#0000FF">Data results</font></h3>
10. <p>There are 2 results available from market research online data collection:</p>
11. <table>
12. <tr><td>1. <a href="report.php">Results</a> from the research</td></tr>
13. <tr><td>2. <a href="report2.php">List</a> of qualified contestants</td></tr>
14. </table>
15. </center>
16. </body>
17. </html>

```

The **report.php** script **lists the data** as recorded by **save.php**. This time, we open the file for **reading** by the parameter **"r"**. By means of a while block, **Line 4-7**, using the function **feof(\$r)** as the core of the condition, the file is read line by line until the end of file is appearing. Each line is retrieved by the function **fgets()**, and sent for display.

```

1. <!-- report.php -->
2. <?php
3. $r=fopen( '../market_research/response.txt', 'r');
4. while(!feof($r)) {
5. $line=fgets($r);
6. print("$line<br>");
7. }
8. print("<center>
9. <p> <a href=admin.htm>Return</a> to menu.</p>
10. </center>");
11. ?>

```

The second script, **report2.php**, differs only in the specification of the file, **address.txt**, to be read. The lines read by this script have the content as written by **save2.php**.

```

1. <!-- report2.php -->
2. <?php
3. $r=fopen( '../market_research/address.txt', 'r');
4. while(!feof($r)) {
5. $line=fgets($r);
6. print("$line<br>");
7. }
8. print("<center>
9. <p> <a href=admin.htm>Return</a> to menu.</p>
10. </center>");
11. ?>

```

This application is characterized by **reading the records serially** to the server during the collection of data, and retrieving the results in the same order from the server after the collection has been completed. In many applications data already saved are updated in a **random order** as

well as requested in a non-serial order. For such application, use of a **database** will usually be a better solution and will be the topic of the next sessions.

Session 4: Introducing the MySQL database

Dynamic applications and databases

In previous sessions, we have studied examples of dynamic applications in which have made use of session variables to adjust the pages returned to the client dynamically to data provided , and use of files to store data permanently. The use of files can have serious drawbacks since reading or updating a record may require that the **whole file** must be searched.

Using a database instead of a file or a set of files makes it possible to **retrieve or update a single record**. A database has usually its own software, the **Data Base Management System**, which operates on the data. The most **popular** database used in connection with **PHP** is **MySQL**, which is another open source and free software. Commercial database software frequently used with **PHP** is **POSTGRESS**, **ORACLE** and **SyBASE**. All these database systems require **separate installation**.

MySQL must also be installed **separately** from **PHP** which has also to be **configured** to connect and operate with **MySQL**. Assuming that you have successfully installed **MySQL** as indicated in **Information -> Software**, the fundamentals of the **use** of the system will be introduced by means of a very simple example in the next sections.

Creation of a reference database to you personal library.

Most people buy and collect **books**. The collection can contain books belonging to different **categories** such as poetry, prose, fiction, science fiction, historic and contemporary documentaries, information systems methodology, web applications, databases management systems, theoretical and applied research, and many other genres. From time to time, we want to **return** to a book(s). When we recall the author's name or the title of the book the search may be easy. In some situations we may, however, only recall certain aspects discussed which make the search more difficult. A book reference system can then be of great assistance. A model for a book system is presented in [Figure 4.1](#).

As an introductory example to databases, we shall create a **MySQL** database, named **books**, which we can populate with the necessary data about each book in our personal library. We shall need the following files for our web application:

- **HTM** page with example menu
- **HTM** page recording data about a book
- **PHP** script for creating the database, and adding data recorded for a book
- **HTML** page for requesting a list of rows from the database
- **PHP** script for retrieving the rows and responding to the request
- **HTML** page for requesting data about a book(s)
- **PHP** script for retrieving the requested data from the database, and sending it to for display to the client

- **HTML** page to modify or delete a record(s) in the database
- **PHP** script for executing the modification/deletion

Adding records to database

This form is used to add a reference to a new book in the database. The database will be automatically established the first time this system is used.

You are free to develop your own categories, evaluation and location codes. The location code can be a combination of text and a number, for example Bookshelf_a_33.

Name(s) of author(s) :	<input type="text" value="Newman, Chris"/>
Title of book :	<input type="text" value="PHP in 10 minutes"/>
Publisher:	<input type="text" value="SAMS"/>
Year of publication :	<input type="text" value="2005"/>
Number of pages:	<input type="text" value="254"/>
Category:	<input type="text" value="Scripting"/>
Date read :	<input type="text" value="Feb 1 2006"/>
Evaluation:	<input type="text" value="Handy"/>
Book location :	<input type="text" value="Shelf 4"/>
Submit data:	<input type="button" value="Submit"/>

[Return](#) to menu.

Figure 4.1: Adding records

The application is fully functional, but you should consider making your personal modifications to the design before you start recording data for your personal library.

Menu page

The **index.htm** is a very simple **HTML** page displaying a **menu** with links to the different parts of the reference system::

```

1.  <!-- index.htm -->
2.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3.  "http://www.w3.org/TR/html4/loose.dtd">
4.  <html>
5.  <head>
6.  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
7.  <title>Untitled Document</title></head>
8.  <body>
9.  <center>
10. <h2><font color="blue">Menu for the book database</font></h2>
11. <p>The following menu lists the alternatives available for the database <b>books</b>. </p>
12. <table>
13. <tr><td><a href="add.htm">Insert </a> book data to a to database row.</td></tr>
14. <tr><td><a href="list.php">List</a> all rows in the book database.</td></tr>
15. <tr><td><a href="search.htm">Search</a> for a row of data in the database.</td></tr>
16. <tr><td><a href="update1.htm">Update</a> a row in the database.</td></tr>
17. <tr><td><a href="delete.htm">Delete</a> a row from the database.</td></tr>
18. <tr><td><a href="remove.htm">Remove</a> database content.</td></tr>
19. </table>
20. </center>
21. </body>
22. </html>

```


To obtain a nice visual impression, the links are embedded in a table. You can see the menu in [Figure 4.2](#). No further explanation of this form should be required at this stage.

List of book references									
id	Author	Title	Publisher	Year	Pages	Category	Date read	Evaluation	Location
3	Newman, Chris	PHP in 10 minutes	SAMS	2005	254	Scripting	Feb 1 2006	Handy	Shelf 4

[Return](#) to menu.

Figure 4.2: List of book references

Creating and populating a database

Let us start studying the **creation** of a bibliographic database. There are international standards and protocols for the content required for professional bibliographic databases. In this example, we shall ignore the standards and only specify elements needed in a database for **private** use. It must obviously contain such data as the name of the author(s), the book title, the publisher's name, when printed, and the number of pages.

We may also want to be able to make a rough **distinction** about the categories of books. You are free to establish your own codes for categories, evaluation and locations of the books. The **add.htm** is the **HTML** pages used for submitting the recorded data to the server.

1. `<!-- add.htm -->`
2. `<center><h3>Adding records to database</h3>`
3. `<p>This form is used to add a reference to a new book in the database. The database will be automatically established the first time this system is used. </p>`
4. `<p>You are free to develop your own categories, evaluation and location codes. The location code can be a combination of text and a number, for example Bookshelf_a_33. </p>`
5. `<form action="add.php" method="post">`
6. `<table>`
7. `<tr><td>Name(s) of author(s) : </td><td><input name="author" type="text"></td></tr>`
8. `<tr><td>Title of book : </td><td><input name="title" type="text"></td></tr>`
9. `<tr><td>Publisher: </td><td><input name="publisher" type="text"></td></tr>`
10. `<tr><td>Year of publication : </td><td><input name="year" type="text"></td></tr>`
11. `<tr><td>Number of pages: </td><td><input name="pages" type="text"></td></tr>`
12. `<tr><td>Category: </td><td><input name="category" type="text"></td></tr>`
13. `<tr><td>Date read : </td><td><input name="dateread" type="text"></td></tr>`
14. `<tr><td>Evaluation: </td><td><input name="evaluation" type="text"></td></tr>`
15. `<tr><td>Book location : </td><td><input name="location" type="text"></td></tr>`
16. `<tr><td>Submit reference t:</td><td><input name="" type="submit" value="Submit"></td></tr></table></form>`
17. `<p>Return to menu.</p>`
18. `</center>`

The 9 named values in the form block **Line 7- 15** are submitted to the server for processing by means of the **add.php** script. See [Figure 4.3](#).

Search for a book reference.

By specifying author, title, publisher, publishing year, category, and keywords the application can retrieve relevant book reference(s) if any.

Name(s) of author(s) :	<input type="text" value="Newman, Chris"/>
Title of book :	<input type="text" value="unspecified"/>
Publisher:	<input type="text" value="unspecified"/>
Year of publication :	<input type="text" value="unspecified"/>
Number of pages:	<input type="text" value="unspecified"/>
Category:	<input type="text" value="unspecified"/>
Date read :	<input type="text" value="unspecified"/>
Evaluation:	<input type="text" value="unspecified"/>
Book location :	<input type="text" value="unspecified"/>

Submit search criteria:

[Return](#) to menu.

Figure 4.3: Search for a book reference

The **add.php** is our first step into the world of **MySQL**. Let us first select a name, **books**, for our database, and **booktable** as the name of the single table in this simple database.

```

1. <!-- add.php -->
2. <?php
3. print("<center>");
4. $link=mysql("localhost","root","password");
5. if(!$link) die("<h3><font color=red>You must install MySQL</font></h3>");
6. $db_selected=mysql_select_db("books", $link);
7. if (!$db_selected) {
8. mysql_query("CREATE DATABASE books",$link);
9. mysql_select_db("books", $link );
10. mysql_query("CREATE TABLE booktable(id INT AUTO_INCREMENT, author VARCHAR(30), title
    VARCHAR(30), publisher VARCHAR(30), year VARCHAR(30), pages VARCHAR(30), catagory
    VARCHAR(30), dateread VARCHAR(30), evaluation VARCHAR(30), location VARCHAR(30))", $link);
11. }
12. mysql_query("INSERT INTO booktable(author,title, publisher, year, pages, category, dateread,
    evaluation, location)
    VALUES('$ _POST[author]','$ _POST[title]','$ _POST[publisher]','$ _POST[year]','$ _POST[pages]',
    '$ _POST[category]','$ _POST[dateread]','$ _POST[evaluation]','$ _POST[location]')", $link );
13. mysql_close($link);
14. print("<font color=blue>");
15. print ("<h3>The book record has been inserted . </h3>");
16. print("<p></p>");
17. print("<a href=index.htm>Return</a> to menu.");
18. print("</center>");
19. ?>

```

The first **mysql** statement in **Line 4** establishes the necessary connection between **PHP** and **MySQL** and returns a reference, **\$link**, which should be referred to in most **mysql** statements. The function requires 3 arguments: **localhost**, a user name **root**, and the **password** you submitted when you configured **MySQL**. *In the remaining of this course you must always substitute 'password in this function with your private MySQL password!*

If a connection **cannot** be established, we use the **PHP** function **die()** to inform the user and break the processing. The purpose of function **mysql_select_db()** in **Line 6** is to **select/open** the database (there may be several in your **MySQL**!) we want to work with. The success of the selection is tested in the next line and if the database in our context cannot be selected, we assume that it does not exist. By means of a **mysql_query()** function, the database is created by an **SQL** statement and a reference to the connection in **Line 8**. The **mysql_select_db()** must be repeated and now we can expect that it is successful. **Line 10** is another use of the **mysql()** function by which we create a table, **booktable**, in our database. Note that in a parenthesis of the specified table name, the name and type of each table column follow according to the **SQL** conventions.

The first column is for the variable **id**. This variable type is special, **INT AUTO_INCREMENT**, which means that each row inserted will be numbered consecutively. All the other **9** variables are of type **VARCHAR(30)**. Many other possibilities exist. The number of **pages** could for instance be specified as **INT**. **VARCHAR(30)** specifies character strings of varying length up to **30** characters. We are now ready to start inserting data into the database.

In the **mysql_query()** function in **Line 12**, the **9** variable values received from the client are **added** to the **booktable** by means of an **SQL INSERT** statement. Several aspects of this function should be noted. *First*, the variable **id** introduced in **Line 10**, should not to be specified because it is automatically inserted. *Second*, it is very important that the **SQL** syntax is correct. In particular, remember to enclose all values of type **VARCHAR** and other string types in **single quotes**, and do **NOT** use single quotes within the **\$_POST[]** because the **INSERT** statement itself is enclosed by double quotes. *Third*, be also certain that the elements in **booktable(..)** matches the values in **VALUES(..)**.

The last remark about this script is that as a general rule in an interactive application, if a connection between **PHP** and **MySQL** has been established, a **mysql_close()** function should be activated before entering a new page.

Listing the content of the database

The request for a **list** of all rows in the database does not require any additional data, and **list.php** can be called **directly** from the menu, **index.htm**.

```
1. <!-- list.php -->
2. <?php
3. print("<center>");
4. $link=mysql_connect("localhost","root","password");
5. if ($link) die("<h3><font color=red>You must install MySQL.</font></h3>");
6. $db_selected=mysql_select_db("book", $link);
7. if (!$db_selected) {
8.   print("<h3><font color=red>Database does not exist.</font></h3>")
9.   else {
10.    $r = mysql_query("SELECT * FROM booktable", $link);
```



```

11. if (!$r) {
12. print("<h3><font color=red>Booktable does not exist</font></h3>");
13. }
14. else {
15. print("<h2><font color=Blue>List of panel members</font></h2>
16. <table border>
17. <tr><b><Th>Id<Th>Author</Th><Th>Title</Th><Th>Publisher</Th><Th>Year</Th><Th>Pages</Th><Th>
    >Category</Th><Th>Date read<</Th><Th>Evaluation</Th><Th>Location</Th></b></tr>");
18. while ($row = mysql_fetch_array($r)) {
19. print("<tr><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td><td>$row[3]</td><td>$row[4]</td><t
    d>$row[5]</td><td> $row[6] </td><td>$row[7]</td><td>$row[8]</td><td>$row[9]</td>");
20. }
21. }
22. print("</table>");
23. }
24. mysql_close($link);
25. print("<p></p>");
26. print("<a href=index.htm>Return</a> to menu.");
27. print('</center>');
28. ?>

```

As in the previous script, the **MySQL** must be connected and the database books selected. Then in **Line 10** a **mysql_query()** function is called with an **embedded SQL SELECT** statement as argument. The * indicates that **all** table rows are wanted. The result of this selection is referred to by the reference/handle **\$r** which is tested for the existence of rows. If there are rows in the table, they are processed one by one in the **PHP** while statement in **Line 18**. As long as there are any rows left in **\$r**, **mysql_fetch_array()** will fetch on row separately and assign it to array **\$row[]** to be presented to the user in the table specified in **Lines 15-22**. Note that the first element of an array is 0. [Figure 4.4](#) illustrates a very short list.

Correct the required attributes.

Row Id:	<input type="text" value="3"/>
Name(s) of author(s) :	<input type="text" value="Newman"/>
Title of book :	<input type="text" value="PHP"/>
Publisher:	<input type="text" value="SAMS"/>
Year of publication :	<input type="text" value="2005"/>
Number of pages:	<input type="text" value="254"/>
Category:	<input type="text" value="Scripting"/>
Date read :	<input type="text" value="Feb 2005"/>
Evaluation:	<input type="text" value="Good"/>
Book location :	<input type="text" value="Shelf 4"/>
Submit data:	<input type="button" value="Submit"/>

[Return](#) to menu.

Figure 4.4: Correct required attributes

Searching the database for a book reference.

In a large collection of books, it can be difficult to decide which book is relevant for a particular situation and perhaps also find its physical location. We need a search function. The link to searching of the menu in **index.htm** points to **search.htm**:

```
1. <!-- search.htm -->
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
3. <<html>
4. <head>
5. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
6. <title>Untitled Document</title>
7. </head>
8. <body>
9. <center>
10. <h2><font color="blue">Search for a reference to a book.</font></h2>
11. <p>By specifying author, title, publisher, publishing year, category, and keywords the application can
    retrieve relevant book reference(s) if any.</p>
12. <form action="search.php" method="post">
13. <table>
14. <tr><td>Name(s) of author(s) : </td><td><input name="author" type="text"
    value="unspecified"></td></tr>
15. <tr><td>Title of book : </td><td><input name="title" type="text" value="unspecified"></td></tr>
16. <tr><td>Publisher: </td><td><input name="publisher" type="text" value="unspecified"></td></tr>
17. <tr><td>Year of publication : </td><td><input name="year" type="text"
    value="unspecified"></td></tr>
18. <tr><td>Number of pages: </td><td><input name="pages" type="text" value="unspecified"></td></tr>
19. <tr><td>Category: </td><td><input name="category" type="text" value="unspecified"></td></tr>
20. <tr><td>Date read : </td><td><input name="dateread" type="text" value="unspecified"></td></tr>
21. <tr><td>Evaluation: </td><td><input name="evaluation" type="text" value="unspecified"></td></tr>
22. <tr><td>Book location : </td><td><input name="location" type="text" value="unspecified"></td></tr>
23. <tr><td>Submit search criteria:</td><td><input name="" type="submit"
    value="Submit"></td></tr></table>
24. </form>
25. <p><a href="index.htm">Return</a> to menu.</p>
26. </center>
27. </body>
28. </html>
```

The **search.htm** is an ordinary **HTML** form page, but we specify the default string value **unspecified** for each variable. The default value(s) must be changed to the value associated with the book(s) searched. If for example **web applications** is used as category for books in this field, and we want to localize books published in 2005 in our library, **Year of publication** should be changed to **2005**, and **Category** to **web applications**. Or, if we search books of Sklar, David, we substitute **unspecified** with **Sklar, David** in the author box. Since the probability for any saved value should match this string it works in our application. See the search form in [Figure 4.5](#)

Search for a reference to a book.

By specifying author, title, publisher, publishing year, category, and keywords the application can retrieve relevant book reference(s) if any.

Name(s) of author(s) :	<input type="text" value="Sklar, David"/>
Title of book :	<input type="text" value="unspecified"/>
Publisher :	<input type="text" value="unspecified"/>
Year of publication :	<input type="text" value="unspecified"/>
Number of pages :	<input type="text" value="unspecified"/>
Category :	<input type="text" value="unspecified"/>
Date read :	<input type="text" value="unspecified"/>
Evaluation :	<input type="text" value="unspecified"/>
Book location :	<input type="text" value="unspecified"/>

Submit search criteria:

[Return to menu.](#)

Figure 4.5: Search for a reference to a book

When the request is submitted, it calls for **search.php** to process the request.

```

1. <!-- search.php -->
2. <?php
3. print("<center>");
4. $link=mysql_connect("localhost","root","password");
5. if (!$link) die("<h3><font color=red>You must install MySQL. </font></h3>");
6. if (!mysql_select_db("books")) die("Database books does not exist.");
7. print("<h3><font color=blue>List of requested rows.</font></h3>");
8. <table border>
9. <tr><b><Th>Id<Th>Author<Th>Title<Th>Publisher<Th>Year<Th>Pages<Th>Category<Th>Date
   read<Th>Evaluation<Th>Location</b></tr>");
10. $r=mysql_query("SELECT * FROM booktable WHERE ((author='$_POST[author]') | (title='$_POST[title]')
    | (publisher='$_POST[publisher]') | (year='$_POST[year]') | (pages='$_POST[pages]') |
    (category='$_POST[category]') | (dateread='$_POST[dateread]') | (evaluation='$_POST[evaluation]') |
    (location='$_POST[location]'))", $link);
11. while ($row=mysql_fetch_array($r)){
12. print("<tr><td> $row[0] <td> $row[1] <td> $row[2] <td> $row[3] <td> $row[4] <td> $row[5] <td>
    $row[6] <td> $row[7] <td> $row[8] <td> $row[9]");
13. }
14. print("</table>");
15. mysql_close($link);
16. print("<p></p>");
17. print("<a href=index.htm>Return</a> to menu.");
18. print("</center>");
19. ?>

```

The **mysql_query()** in **Line 10**, includes a **WHERE** clause with a composite OR condition (the symbol used for OR is |). Note that if we had used another logical operator as AND the trick using unspecified as default value in the form would not have worked. We use again the while() function to transfer the results referenced by \$r to an array \$row[] for presenting the results in a table for the user. The search result is illustrated by [Figure 4.6](#).

List of requested rows.									
Id	Author	Title	Publisher	Year	Pages	Category	Date read	Evaluation	Location
2	Sklar, David	Learning PHP 5	O'Reilly	2004	348	scripting	July 2005	Does not satisfy my criteria	Shelf B

[Return](#) to menu.

Figure 4.6: List of requested rows

Updating book references in the database

In our book reference library, there may be a need for changing or updating a row because of typos, incorrect data, revaluation of the referenced book, etc. Our solution to this task requires **1 HTM** page and **2 PHP** scripts.

The **update.htm** is an ordinary page which requires that you have the **id** of the row reference. See [Figure 4.7](#).

Update a row in the book database

Note the Id number of the row you want to to update and type it in the box:

[Return](#) to menu.

Figure 4.7: Update a row in the book database

1. `<!-- update.htm -->`
2. `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"`
`"http://www.w3.org/TR/html4/loose.dtd">`
3. `<html>`
4. `<head>`
5. `<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">`
6. `<title>Untitled Document</title>`
7. `</head>`
8. `<center>`
9. `<h2>Update a row in the book database</h2>`
10. `<p>Note the Id number of the row you want to to update and type it in the box:`
11. `<form action="update1.php" method="post">`
12. `<input name="id" type="text"><input name="" type="submit" value="Submit"> <p></p>`
13. `</form>`
14. `<p>Return to menu.</p>`
15. `</center>`
16. `<body>`
17. `</body>`

18. </html>

There is nothing new in this page, and we note that **Lines 12-13** call for processing of a row corresponding to the submitted **id** value by the script **update1.php**:

```
1. <!-- update1.php -->
2. <?php
3. print("<center>");
4. $link=mysql_connect("localhost","root","password");
5. if(!$link) die("<h3><font color=red> You must install MySQL. </font></h3>");
6. $db_selected=mysql_select_db("books");
7. if(!$db_selected) die("<h3><font color=red>Databas booksdoes not exist.</font></h3>");
8. $id=$_POST['id'];
9. $r=mysql_query("SELECT * FROM booktable WHERE id='$id'", $link);
10. while($row=mysql_fetch_array($r)) {
11. $author=$row[1];
12. $title=$row[2];
13. $publisher=$row[3];
14. $year=$row[4];
15. $pages=$row[5];
16. $category=$row[6];
17. $dateread=$row[7];
18. $evaluation=$row[8];
19. $location=$row[9];
20. }
21. print("<h2><font color=blue><p>Correct the required attributes.</font></h2></p>");
22. print("<form action=update2.php method=post>
23. <table>
24. <tr><td>Row Id: </td><td><input name=id type=text value=$id></td></tr>
25. <tr><td>Name(s) of author(s) : </td><td><input name=author type=text value=$author></td></tr>
26. <tr><td>Title of book : </td><td><input name=title type=text value=$title></td></tr>
27. <tr><td>Publisher: </td><td><input name=publisher type=text value=$publisher></td></tr>
28. <tr><td>Year of publication : </td><td><input name=year type=text value=$year></td></tr>
29. <tr><td>Number of pages: </td><td><input name=pages type=text value=pages></td></tr>
30. <tr><td>Category: </td><td><input name=category type=text value=$category></td></tr>
31. <tr><td>Date read : </td><td><input name=dateread type=text value=$dateread></td></tr>
32. <tr><td>Evaluation: </td><td><input name=evaluation type=text value=$evaluation></td></tr>
33. <tr><td>Book location : </td><td><input name=location type=text value=$location></td></tr>
34. <tr><td>Submit data:</td><td><input type=submit value=Submit></td></tr>
35. </table></form><p></p>");
36. print("<a href=index.htm>Return</a> to menu.");
37. print("</font></center>");
38. ?>
```

The aim of this first update script is to return a form to the user with all current data of row **\$id** to the client for inspection, correction and submittal to the server. **Lines 9-20** retrieve the data from the database and establish local variables to be used for creating the form. Because the form must be sent in **HTML** format to the client, the long **print()** function in **Lines 22- 35** is used to send the **HTML** tags within double quotes. **PHP** automatically equips the contents of these **HTML** tags with required quotes, and we must therefore remove all quotes in the tags.

According to the specifications, a form is displayed at the user's screen with all current variable values. The user can change or leave the values. The form is submitted for processing by **update2.php**:

```
1. <!-- update2.php -->
2. <?php
3. print("<center>");
4. $link=mysql_connect("localhost","root","password");
5. if (!$link) die("<h3><font color=red> You must install MySQL.</font></h3>");
6. $db_selected=mysql_select_db("books");
7. if (!$db_selected) die("<h3><font color=red>Database <b>books</b> does not exist.</font></h3>");
8. $n=mysql_query("SELECT * FROM booktable", $link);
9. if [mysql_nun_rows($n) == 0] die("<h3><font color=red>Table is empty.</font></h3>");
10. mysql_query("UPDATE booktable
11. SET
12. id='$_POST[id]',
13. author='$_POST[author]',
14. title='$_POST[title]',
15. publisher='$_POST[publisher]',
16. year='$_POST[year]',
17. pages='$_POST[pages]',
18. category='$_POST[category]',
19. dateread='$_POST[dateread]',
20. evaluation='$_POST[evaluation]',
21. location='$_POST[location]'
22. WHERE id='$_POST[id]'", $link );
23. mysql_close($link);
24. print ("<font color=blue><h3> Data for book with Id $_POST[id] has been updated.</h3></font>");
25. print("<a href=index.htm>Return</a> to menu.");
26. print("</center>");
27. ?>
```

This script is similar to **add.php** but uses a **mysql_query()** with the **SQL UPDATE** statement in **Lines 10-22** to change an existing row instead of the **INSERT** statement of **add.php** which adds a new row to the database. It uses **SET** with subsequent variable-value pairs for updating values in the row specified by the **WHERE** clause. As for the **INSERT** statement, it is very important that all values are enclosed by single quotes if they were defined as strings in the **CREATE TABLE** statement. [Figure 4.8](#) illustrates the page with data which can be updated before submitted.

Correct the required attributes.

Row Id:	<input type="text" value="1"/>
Name(s) of author(s) :	<input type="text" value="Trachtenberg"/>
Title of book :	<input type="text" value="Upgrading"/>
Publisher:	<input type="text" value="OReilly"/>
Year of publication :	<input type="text" value="2004"/>
Number of pages:	<input type="text" value="pages"/>
Category:	<input type="text" value="Scripting"/>
Date read :	<input type="text" value="December"/>
Evaluation:	<input type="text" value="Very good"/>
Book location :	<input type="text" value="Self"/>
Submit data:	<input type="button" value="Submit"/>

[Return](#) to menu.

Figure 4.8: Correct the required attributes

Deleting rows in the database

Just as we need operations for adding and updating a row, an operation for deleting a row is wanted. Before we can start the operation, the **\$id** of the row to be deleted must be found. It can be obtained by the list option. **delete.htm** is an **HTML** page for specifying the **\$id** of a row wanted to be deleted.

1. `<!-- delete.htm -->`
2. `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"`
`"http://www.w3.org/TR/html4/loose.dtd">`
3. `<html>`
4. `<head>`
5. `<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">`
6. `<title>Untitled Document</title>`
7. `</head>`
8. `<center>`
9. `<h2>Delete a row in the book database</h2>`
10. `<p>Note the Id number of the row you want to delete and type it in the box:`
11. `<form action="delete.php" method="post">`
12. `<input name="id" type="text"><input name="" type="submit" value="Submit"> <p></p>`
13. `</form>`
14. `<p>Return to menu.</p>`
15. `></center>`
16. `<body>`

17. `</body>`
18. `</html>`

Delete a row in the book database

Note the Id number of the row you want to delete and type it in the box:

[Return](#) to menu.

Figure 4.9: Delete a row in the book database

The page is simple (See [Figure 4.9](#)) and sends an **\$id** with a request for processing by the **delete.php** script:

```

1. <!-- delete.php -->
2. <?php
3. print("<center>");
4. $link=mysql_connect("localhost","root","password");
5. if (!$link) die("<h3><font color=red>You must install MySQL.</font></h3>");
6. $db_selected=mysql_select_db("books", $link);
7. if (!$db_selected) die("<h3><font color=red>Database <b>books</b> does not exist.</font></h3>");
8. $r=mysql_query("DELETE FROM booktable WHERE id=='$_POST[id]'", $link);
9. if (!r) {
10. print("<h3><font color=red>The row does not exist.</font></h3>");
11. }
12. else {
13. print("<h2><font color=red>Row $_POST[id] has been removed from database books.</font></h2>");
14. }
15. print("<p></p><a href=index.htm>Return</a> to menu.");
16. print("</center>");
17. mysql_close($link);
18. ?>

```

In an **mysql_query()** of **Line 8**, the **SQL DELETE** statement controls the deletion of the row specified in its **WHERE** clause.

Removing database

It is also possible to instruct the server to remove the database **books**. The **HTML** page **remove.htm** used is simple:

```

1. <!-- remove.htm -->
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

```



```

3. "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
7. <title>Untitled Document</title>
8. </head>
9. <center>
10. <h2><font color="blue">Remove the database books </font></h2>
11. <p><font color="red"><b>Note that all book references in your database will be lost by executing this
    operation.</b></font></p>
12. <form action="remove.php" method="post">
13. Remove all content:<input name="" type="submit" value="Submit"> <p></p>
14. </form>
15. <p><a href="index.htm">Return</a> to menu.</p>
16. </center>
17. <body>
18. </body>
19. </html>

```

The page is shown in [Figure 4.10](#), and requires no explanations. It calls upon an even shorter **remove.php** script.



Figure 4.10: Remove database content

```

1. <!-- remove.php -->
2. <?php
3. print("<center>");
4. $link=mysql_connect("localhost","root","password") die("<h3><font color=red>You are not
    connected to MySQL. </font></h3>");
5. $db_selected=mysql_select_db($link) die ("<h3><font color=red>Database <b>books</b> does not
    exist. </font></h3>");
6. $r=mysql_query("DROP DATABASE books"; $link);
7. print("<h2><font color=red>Database content has been removed.</font></h2>");
8. print("<p></p><a href=index.htm>Return</a> to menu.</center>");
9. mysql_close($link);
10. ?>

```

This script introduces another new **SQL** statement, **DROP TABLE**, which only requires the name of the table.

Session 5: Polling with MySQL database

Opinion polls

In session 2, we studied a market research example which did not require any data base backup. In this session we shall consider a similar scenario: a **polling organization** the aim of which is to collect the public opinion about the preferences for 5 political parties (or product brands) **A, B, C, D, and E**.

The organization uses a **panel** with a fixed number of members as basis for its services. A file with separate records for each panel members is kept in a database. Each **Monday** a list of panel members with their contact addresses is **retrieved** and **used** by interviewers who in telephone interviews asking panel members which party the member would have voted for if there had been a public vote, or in case of products which product they would have purchased, that Monday. The answers are subsequently saved in the database for retrieval, computation and publication of statistics to subscribing clients each **Wednesday**. The panel members can be stratified by **age** and **area** in which they live.

Use of a panel gives usually more precise estimates of the political time fluctuations than a random sample would do. However, to avoid that the panel becomes obsolete or the members worn out, the panel is made **rotating**, i.e. it is slowly renewed. Each **Friday**, the **n** oldest members of the panel are **removed** while **n** new members are **inserted** in the panel. It is assumed that the organization at any point of time must have the possibility to update the information about panel members who have moved, changed telephone numbers, etc.

It is desired that the management of the panel members and their answers can be implemented as a web application because the staff of the organization works from different locations.

Application design

The overall composition of the application design is outlined in [Figure 5.1](#). We can distinguish

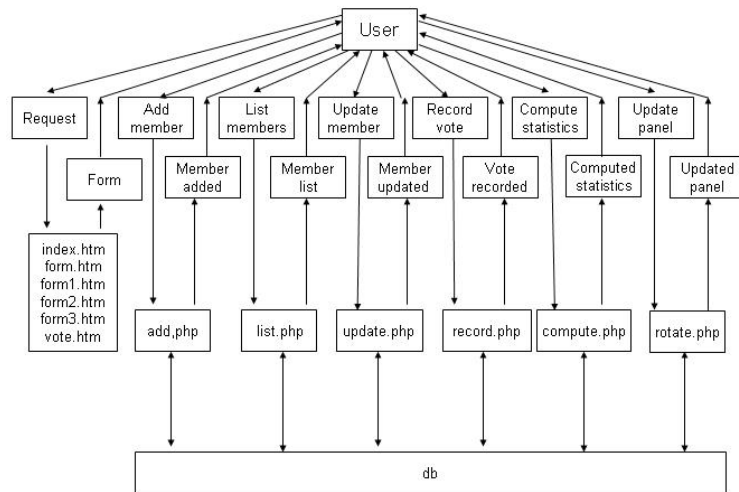


Figure 5.1: Opinion polling application diagram

between a **database** and a **set of processes** working with the database which is typical for many dynamic web applications.

The application is a mixture of 4 **HTML** and 6 **PHP** scripts **interacting** with a database.

Database

We could apply the database manager developed in the previous session to set up a database for the present application. Creation of a database as preparation for this application is, however, not necessary because it will automatically be created the first time the application is run.

Application menu

The first and obvious task is to create an opening page with a **menu** by which the user can select the action wanted. A simple **HTML** page will provide the service needed. The following **index.htm** file is the implementation used in our example:

1. `<!-- index.htm -->`
2. `<center>`
3. `<h2>Opinion polls</h2>`
4. `<p>The Opinion polls system is initialized with a sample of panel members. Each Monday a List of panel members to be interviewed is generated. The data for panel members can be updated if necessary. After the interviews, the votes are recorded recorded. The table is the basis for computing statistics for the week. At the end of the week, the first panel member on the list is deleted, and a new member added at the end of the list.</p>`


```

5. <table>
6. <tr><td><a href="form.htm">Initialize</a> table of panel members</td></tr>
7. <tr><td><a href="list.php">List</a> panel members for interviews</td></tr>
8. <tr><td><a href="form3.htm">Update</a> data for panel member</td></tr>
9. <tr><td><a href="vote.htm">Record</a> interview votes</td></tr>
10. <tr><td><a href="Compute.php">Compute</a> statistics for the week</td></tr>
11. <tr><td><a href="form2.htm">Delete</a> first and add new panel member</td></tr>
12. </table>
13. </center>

```

The page has a simple and ordinary structure using a **href** tags for providing the links to the 5 different services included in the system. A **table** tag with associate **tr** and **td** tags are used to give the page an orderly appearance.

Creating records and a list of panel members

The polling of opinions requires a **sample** of persons to interview. If this had been a course in sample surveys, we would have spent considerable time on the problem how to get a **representative** sample of the voting population. In this course, we assume that the statisticians have completed their job, and that a list of names, etc. exists. A facility for **recording** these data in the database is now needed. It is implemented by an **HTML** and a **PHP** files.

The first file is named **form.htm**. It includes in **Line 4** a **FORM** tag with **METHOD="post"** and referring to file **add.php**. The **METHOD="post"** is important because we use it to create global variables. The **FORM** block includes input boxes for **Family Name**, **FirstName**, **Telephone**, **Age**, and **Area**.

```

1. <!-- form.htm --->
2. <center>
3. <h2><font color="Blue">Form to be used for adding new members to the interview panel</font></h2>
4. <form action="add.php" method="post">
5. <table>
6. <tr><td>Family name:</td><td><input type="text" name="FamilyName"></td></tr>
7. <tr><td>First name:</td><td><input type="text" name="FirstName"></td></tr>
8. <tr><td>Telephone no:</td><td><input type="text" name="Telephone"></td></tr>
9. <tr><td>Age(18-100):</td><td><input type="text" name="Age"></td></tr>
10. <tr><td>Area (1-10):</td><td><input type="text" name="Area"></td></tr>
11. <tr><td></td><td><input type="submit" name="NewMember" value="Submit new panel member"></td></tr>
12. </table>
13. </form>
14. </center>

```

[Figure 5.2](#) displays the form for including a new member into the panel.

Form to be used for adding new members to the interview panel

Figure 5.2: Establishing a panel of interview members

When this form has been completed and submitted to the server, the **add.php** file is called. The first action is defining and specifying the handle **\$d** referring to the database **named "db"**. You are of course free to select any convenient name for your database. In **Line 4**, the variable is used to open the database and generate the database resource handler **\$db**. Later in this script, the reference to the open database is always **\$db**. The first time this script is called **Line 4** will create the database

As you recall, we created the database without specifying any tables. The first time this script is run, the required **table for storing** the panel member data must be created. We use the same approach to test for the existence of **MySQL**, the database **poll** and the table **Voters** as we did in Session 4. Note that we use **VARCHAR()** as column type in this example.

```

1. <?php
2. print("<center>");
3. $link=mysql_connect("localhost","root","password");
4. if (!$link) die("<h3><font color=red>You must install MySQL. </font></h3>");
5. $db_selected= mysql_select_db("poll", $link);
6. if (!$db_selected) {
7.     mysql_query("CREATE DATABASE poll", $link);
8.     mysql_select_db("poll", $link);
9.     mysql_query("CREATE TABLE Voters(id INT NOT NULL AUTO_INCREMENT, PRIMARY KEY(id),
        FamilyName VARCHAR(30),FirstName VARCHAR(30), Telephone VARCHAR(8), Age VARCHAR(3), Area
        VARCHAR(2), Vote VARCHAR(2))", $link) or die("Ikke noen tabell");
10. }
11. mysql_query("INSERT INTO Voters(FamilyName,FirstName, Telephone, Age, Area, Vote)
        VALUES('$ _POST[FamilyName]','$ _POST[FirstName]','$ _POST[Telephone]','$ _POST[Age]','$ _POST[Area
        ]','')", $link);
12. print("<center><font color=blue>");
13. print ("<h3>$ _POST[FirstName] $ _POST[FamilyName] has been added to list of voters.</h3>");
14. print("<p></p>");
15. print("<a href=index.htm>Return</a> to menu.");
16. print("</font></center>");

```


17. ?>

Each **Monday**, the votes of each panel member are collected by telephone interviews. As a basis for the interviews, a list of all **current panel members** is needed. The list is generated from the file **list.php** called from the menu. As you can see, the content of this file is a **mixture** of **HTML** and **PHP**. The title and the heading of the list is specified by **HTML Lines 2 - 5** while the remaining of the file is a **PHP** script. This mixture design is quite usual and convenient even though the **same result** could have obtained with **PHP** only using the **PHP** print function to specify the list title, table tag and table heading.

```
1. <!-- list.cfm -->
2. <center>
3. <h2><font color=Blue>List of panel members</font></h2>
4. <table>
5. <TR><Th><b>ID</b></Th><Th><b>Family Name</b></Th><Th><b>First
   name</b></th><Th><b>Telephone</b></Th><Th><b>Age</b></Th><Th><b>Area</b></Th></tr>
6. <?php
7. $link=mysql_connect("localhost","root","password");
8. mysql_select_db("poll", $link);
9. $r=mysql_query("SELECT * FROM Voters ORDER BY Id ASC", $link);
10. while ($row = mysql_fetch_array($r)) {
11. print("<tr><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td><td>$row[3]</td><td>$row[4]</td><t
    d>$row[5]</td>");
12. }
13. mysql_close($link);
14. print("</TABLE></font>");
15. print("<p></p>");
16. print("<a href=index.htm>Return</a> to menu. </center>");>
17. ?>
```

The database **name** is specified in **Line 8**. In **Line 9**, an **mysql_query[]** is called with a **"SELECT * FROM Voters ORDERED BY Id ASC"** string.(ASC is an abbreviation for **ascending**). Recall that the variable **Id** is an automatically generated and assigned to the inserted members as integers with increasing values. In other words, the **first** record in the **ASC** ordering has the **lowest Id** value and the last record is the **oldest** member of the panel.

The **while**-block in **Lines 10 - 12** fetches **1** row represented as an array from the query result referred to by **\$r** each time the loop is traversed. The array is referred to as **\$row**, and the individual elements of the row are **\$row[0]**, **\$row[1]**, **\$row[2]**, **\$row[3]**, **\$row[4]**, and **\$row[5]** in the **print()** of **Line 11**. The last column, **Vote**, in the table is excluded to avoid that the interview is influenced by previous votes.

There may be **changes** in name, telephone number, age, and area since last interview. To permit changes the list can be **updated**. The file **form3.htm** is an **HTML** update form which calls on the **PHP update.php** file for processing the update data submitted. Do not forget that **METHOD="post"** is required in the **form** tag. [Figure 5.3](#) shows the form for updating the individual data for a panel member.

Updating data for panel member

Click the list option to find the Id for the panel member, and complete the form with the updated data (all fields must be completed) :

Member Id:	<input type="text"/>
Family name:	<input type="text"/>
First name:	<input type="text"/>
Telephone no:	<input type="text"/>
Age(18-100):	<input type="text"/>
Area (1-10):	<input type="text"/>
<input type="submit" value="Submit updated data"/>	

Figure 5.3: Updating individual data for panel members

1. `<!-- form3.htm -->`
2. `<center>`
3. `<h2>Updating data for panel member</h2>`
4. `<p>Click the list option to find the Id for the panel member, and complete the form with the updated data (all fields must be completed) :</p>`
5. `<table>`
6. `<form action="update.php" method="post">`
7. `<tr><td>Member Id:</td><td><input type="text" name="Id"></td></tr>`
8. `<tr><td>Family name:</td><td><input type="text" name="FamilyName"></td></tr>`
9. `<tr><td>First name:</td><td><input type="text" name="FirstName"></td></tr>`
10. `<tr><td>Telephone no:</td><td><input type="text" name="Telephone"></td></tr>`
11. `<tr><td>Age(18-100):</td><td><input type="text" name="Age"></td></tr>`
12. `<tr><td>Area (1-10):</td><td><input type="text" name="Area"></td></tr>`
13. `<tr><td></td><td><input type="submit" value="Submit updated data"></td></tr>`
14. `</form>`
15. `</table>`
16. `</center>`

The script of **update.php** demonstrates how to **update** (change) records of the database table. It assumes that the **\$Id** is found by for example the list option, and uses the variable **\$_POST[Id]** submitted by the **form3.htm** in a **WHERE** clause to **locate** the row to be updated. All other variables **except** **Vote** can be changed. The focal lines are **Lines 5-12** with the **SQL** string surrounded by double quotes. You can forget double quotes **within** double quotes, but do not forget to enclose all strings to be sent to the database by **single** quotes.

1. `<!-- update.php -->`
2. `<?php`
3. `$link=mysql_connect("localhost","root","password");`
4. `mysql_select_db("poll",$link);`
5. `mysql_query("UPDATE Voters`
6. `SET`
7. `FamilyName='$_POST[FamilyName]',`
8. `FirstName='$_POST[FirstName]',`


```

9. Telephone='$_POST[Telephone]',
10. Age='$_POST[Age]',
11. Area='$_POST[Area]'
12. WHERE Id='$_POST[Id]' ", $link);
13. print("<center><font color=blue>");
14. print("<center>");
15. print("<font color=blue>");
16. print ("<h3> Data for voter with ID $_POST[Id] has been updated.</h3>");
17. print("<a href=index.htm>Return</a> to menu.");
18. print("</font></center>");
19. ?>

```

Lines 13-18 generates an **HTML** page to be **returned** confirming that the voter with **ID=** `$_POST[Id]` has been updated.

Processing, statistics and rotation

After the panel members have been interviewed, their **votes** must be recorded in the system. A simple **HTML** page of the following type can be used:

```

1. <!-- vote.htm -->
2. <center>
3. <h2><font color="#0000FF">Vote recording</font></h2>
4. <p>Print out the list of panel members. Use the Id number from the list when recording the vote of the individual interviewed person. </p>
5. <form action=record.php method=post>
6. <table>
7. <tr><td>Id number of panel member</td><td><input type="text" name="id" size=4></td></tr>
8. <tr><td>Vote</td><td><input type= text name="vote"></td></tr>
9. <tr><td></td><td><input type=submit value=Record></td></tr>
10. </table>
11. </form>
12. <p><a href="index.htm">Return</a> to menu.</p>
13. </center>

```

[Figure 5.4](#) shows the form for rotating the panel.

To **preserve privacy** as much as possible, only **Id** number and **Vote** are recorded. The form tag specifies the **record.php** as the script for processing the submitted data.

Vote recording

Print out the list of panel members and use the Id number when recording the vote of th individual interviewed person.

Id number of panel member:

Vote:

[Return to menu.](#)

Figure 5.4: Recording a vote

This script demonstrates an **mysql_query()** function with an **UPDATE** string by which the record is updated with the vote given in **Line 6**.

1. `<!-- record.php --><?php$link=mysql_connect("localhost","root","password");mysql_select_db("poll", $link);$id=$_POST[id]";mysql_query("UPDATE Voters SET Vote='$_POST[vote]' WHERE id='$id'", $link); print("<center><h3>The vote of panel member with Id=$id has been recorded.</h3>");print("<p>Return to vote recording.</p></center>");`
2. `?>`

The rest of the script should by now be trivial.

One of the options in the menu is to **compute the statistics** for the week. This option calls the **compute.php** script. The statistics computed is quite simple and meant **only** to be an example. It starts by defining **5** alternative votes, **\$A**, **\$B**, **\$C**, **\$D** and **\$E** and setting these initially equal **"0"** in **Line 3 - 7**.

1. `<!-- compute.php -->`
2. `<?php`
3. `$A="0";`
4. `$B="0";`
5. `$C="0";`
6. `$D="0";`
7. `$E="0";`
8. `$link=mysql_connect("localhost","root","password");`
9. `mysql_select_db("poll",$link);`
10. `$r= mysql_query("SELECT Vote FROM Voters", $link);`
11. `while($row=mysql_fetch_array($r)) {`
12. `$s=$row[0];`
13. `switch($s) {`
14. `case "A": $A++;`
15. `break;`
16. `case "B": $B++;`


```

17. break;
18. case "C": $C++;
19. break;
20. case "D": $D++;
21. break;
22. case "E": $E++;
23. }
24. }
25. mysql_close($link);
26. print("<center><font color=blue><h2>Vote frequencies this week</h2></font>");
27. print("<table>");
28. print("<tr><th>Alternative:</th><th>Frequency:</th></tr>");
29. print("<tr><td>A</td><td>$A</td></tr>");
30. print("<tr><td>B</td><td>$B</td></tr>");
31. print("<tr><td>C</td><td>$C</td></tr>");
32. print("<tr><td>D</td><td>$D</td></tr>");
33. print("<tr><td>E</td><td>$E</td></tr>");
34. print("</table>");
35. print("<p></p>");
36. print("<a href=index.htm>Return</a> to menu. </center>");
37. ?>

```

In **Line 10** all values in column **Vote** of table **Voters** are referenced, and the script continue with a **counting** loop in **Line 11 - 24**. For each new row the vote (in **\$row(0)**) is assigned to the variable **\$s**. The **multi-branch switch(\$s)** statement is used in the head of a loop in **Line 13 - 23**., and depending on the **case** of the current row, the corresponding cumulating variable is **incremented** by **1** (obtained by the **operator ++**). When all rows have been traversed, the cumulated values of the variables are printed out.

More **sophisticated** and **useful** statistics can be produced if we define the database table to take care of both **current** and **previous** week's votes. Such statistics can tell us about the voters' migration from one party category to another.

At the end of each week the panel must be **rotated**, i.e. the **n** first (longest serving) members should be deleted and replaced by **n** new members. Without loss of much generality, we have simplified the rotating to **delete the first member** and **add one new member**. The **form2.htm** page, shown in [Figure 5.5](#), takes care of the recording of the new members' personal data, and calls the **delete.php** script at the server to complete the task.

```

1. <!-- form2.htm --->
2. <center>
3. <h2><font color="Blue">Delete first and add new member to the panel</font></h2>
4. <h3>New member data:</h3>
5. <table>
6. <form action="rotate.php" method="post">
7. <tr><td>Family name:</td><td><input type="text" name="FamilyName"></td></tr>
8. <tr><td>First name:</td><td><input type="text" name="FirstName"></td></tr>
9. <tr><td>Telephone no:</td><td><input type="text" name="Telephone"></td></tr>
10. <tr><td>Age(18-100):</td><td><input type="text" name="Age"></td></tr>
11. <tr><td>Area (1-10):</td><td><input type="text" name="Area"></td></tr>

```


12. `<tr><td></td><td><input type="submit" name="NewMember" value="Submit new member"></td></tr>`
13. `</form>`
14. `</table>`
15. `<p>Return to menu.</p></center>`

Delete first and add new member to the panel

New member data:

Family name:	<input type="text"/>
First name:	<input type="text"/>
Telephone no:	<input type="text"/>
Age(18-100):	<input type="text"/>
Area (1-10):	<input type="text"/>
<input type="submit" value="Submit new member"/>	

[Return](#) to menu.

Figure 5.5: Rotating the panel of interview members

rotate.php retrieves all records from table **Voters** and order them by ascending value of the attribute **Id**. Since **Id** is a **PRIMARY KEY** automatically assigned, a newer record will always have a higher Id value than an older. The first array obtained by **sql_fetch_array()** in **Line 6** will therefore be **the member with the longest service** at the panel and the one to be deleted. **Line 7** contains the **SQL** statement **DELETE** used for **removing records** from a populated table. In **Line 8** the new member is inserted in the panel.

1. `<!-- rotate.php -->`
2. `<?php`
3. `$link=mysql_connect("localhost","root","password");`
4. `mysql_select_db("poll",$link);`
5. `$r=mysql_query("SELECT * FROM Voters ORDER BY Id ASC", $link);`
6. `$row = mysql_fetch_array($r);`
7. `mysql_query("DELETE FROM Voters WHERE Id='$row[0]'", $link);`
8. `mysql_query("INSERT INTO Voters(FamilyName,FirstName, Telephone, Age, Area, Vote)
VALUES('$$_POST[FamilyName]','$$_POST[FirstName]','$$_POST[Telephone]','$$_POST[Age]','$$_POST[Area]
' ,'-')");`
9. `mysql_close($link);`
10. `print("<center>");`
11. `print("<h3>First member has been deleted and new member added.</h3>");`
12. `print("Return to menu.");`
13. `print("</center>");`
14. `?>`

Improvements to this application have been suggested at several places in the text. You can for example try to introduce **gender**, i.e. separate categories for male and female voters, and/or expand the database table to take care of **2** weeks' votes for each panel member. In the computation script, this will permit **comparison** of special tables for male and female voters. The computation script can also be extended to generate tables displaying the migration of voters among parties from last to current week.

Session 6: File processing

So far, we have considered **databases** as the main storage for sets of data. However, it is frequently needed and efficient to work with **serial** data stored as files if random access to elements of the set is not predominant. Typical examples are the logging example of the last session, and text files where required access to the individual elements of the file usually is sequential.

In this session, we shall discuss **3** typical file applications, i.e. **maintaining** files on the server, **fetching remote files** from another server to our server, and **uploading** files from a client computer to our server. (Downloading/copying files available at the server are no problem with software usually available at the client computer).

Maintaining files

As a first simple example, consider an application for which you want to keep a **log of the visits** to the application. In early web days, a number of sites offered to keep records of the visits to your site by placing a link to their logging application on your home page. Now, we can take care of the logging ourselves.

The logging feature can be demonstrated by the following page and script. The log records are saved **sequentially** in a **.txt** file named **log.txt**. The **index.htm** is a **substitute** for the application you want to keep a log for. A real application would use the users **PIN** code **directly** in a code similar to **log_response.php**.

1. `<!-- index.htm -->`
2. `<center>`
3. `<h2>Logging</h2>`
4. This page demonstrates how you can log a visitor passing a certain location of your application site. The example assumes that you give a PIN code (any string will be ok) while in a real application with assigned PIN codes, the code will usually be present as a global session variable.
5. `<table><form action="log_response.php" method="post">`
6. `<tr><td>Please type you PIN code:</td><td><input name="pin" type="password"></td></tr>`
7. `<tr><td></td><td><input name="" type="submit" value="Submit"></td></tr>`
8. `</form>`
9. `</table>`
10. `</center>`

The **log_response.php** script starts by obtaining the values of **3** variables, the **time \$now** by means of the built-in function **strftime()**, **\$pin** from the application, and the **name of the script** to which the logging code is attached.

```
1. <!-- log_response -->
2. <?php
3. $now=strftime('%c');
4. $pin=$_POST['pin'];
5. $location=$_SERVER['SCRIPT_FILENAME'];
6. if(!file_exists('log.txt')) {
7. $e=fopen('log.txt','a');
8. fwrite($e,"LOG FILE");
9. fwrite($e,"\n");
10. fclose($e);
11. }

12. $f=fopen('log.txt','a');
13. fwrite($f,"$now $pin $location");
14. fwrite($f,"\n");
15. fclose($f);

16. print("<center>");
17. print("<h3> Your visit has been logged </h3>");
18. print("Click <a href=log.txt> here </a> to see the log");
19. print("</center>");
20. ?>
```

In **Lines 6 - 11**, the **log.txt** is **created** with a heading, if it does not already exist, by the **fopen()** and **fwrite()** and then **closed** by **fclose()** to be ready to process the log records. These are each **saved** in **Lines 12 -15**.

The only purpose of the final set of **Lines 16 -19** is to provide a **return** from this script, and they would not appear in a real logging script. In this example, the content of the log can be **displayed** by clicking the link in **Line 18**.

Fetching files

In some applications, it is required that the server **downloads/uploads** files from/to other servers in order to provide the intended services to its clients. As an example, consider a server which is maintaining a copy of a continuously updated news source based on a regular scanning of source.

The application example uses a **continuously** running **Internet agent** to maintain a server file with a recent copy from a news server. The demonstration of this example requires **3** programs, an **index.htm** page for displaying **2** example options, an **agent.php** script which is **downloading and saving** the news page every **1800** second, and a **serve.php** script to **serve** the client with the most recent news copy.

The **index.htm** is by now trivial and its function is only to provide the user with the option of 2 alternative requests:

```
1. <!-- index.htm -->
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
3. <html>
4. <head>
5. <title>Untitled Document</title>
6. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
7. </head>
8. <body>
9. <center>
10. <h1><font color="#0000FF"> Agent menu</font></h1>
11. <p>In this example you can:</p>
12. <table>
13. <tr><td>1. <a href="agent.php">Start</a> the agent. It will run as long as you keep the window open.
    </td></tr>
14. <tr><td>2. <a href="serve.php">Request</a> copy of most recent news if the agent is
    running.</td></tr>
15. </table>
16. </center>
17. </body>
18. </html>
```

The **option 1** calls the **agent.php** to start and **run** the agent. **Line 5**, containing a **HTML meta** tag, instructs the server to **repeat** the request every **1800** seconds as long as the message in **Line 13** is **visible**. You can see the selection displayed in [Figure 6.1](#). Note that this agent is running only when the window is kept open.



Figure 6.1: Agent menu

The content of the news source (**edition.cnn.com/index.html** in the current example) is obtained by means of the **stream function**, **file_get_contents()**, which can as well be used for obtaining contents from remote files (as in the example) as well as from local files as in the previous session. **Line 11** fetches the content of the specified remote file and assign it to the variable **\$content**. The next line includes **another** stream function, **file_put_contents()**, which stores the

content of a variable in a specified file. The syntax for specifying a local stream includes 2 components, **file://** is called the **scheme**, while **/PHPRoot/news.htm** is called the **target**. Note the 3 slashes, **///**. In our context, **PHPRoot** is the name of the top document directory of our web domain. The scheme **http://** indicates that the file operation concerns a **remote file** in contrast to a remote file. The scheme **file://** syntax indicates a **local file** operation.

```
1. <!-- agent.php --->
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
3. <html>
4. <head>
5. <meta http-equiv="refresh" content="1800">
6. <title>Agent</title>
7. </head>
8. <body>
9. <center>
10. <?php
11. $content=file_get_contents('http://edition.cnn.com/index.html');
12. file_put_contents("file:///PHPRoot/news.htm", $content);
13. print("The agent is running");
14. ?>
15. </center>
16. </body>
17. </html>
```

As long as the agent is **running**, the file **news.htm** is updated regularly every **1800** second. To be able to keep the agent running and at the **same time** continue to investigate this example, you must therefore open **another session** for the remaining part of the example. In a real situation, the start and running of the agent will be the privilege of the web-master and the clients will be limited to requesting the latest copy of **news.htm**

Each half hour (**1800** seconds), the content of the file **news.htm** is **refreshed**. The users can request a copy by option 2 in **index.htm** which calls **serve.php**:

```
1. <!-- serve.php -->
2. <?php
3. $news= file_get_contents('file:///PHPRoot/news.htm');
4. print("$news");
5. ?>
```

The variable **\$news** is used for **sending** a reply to the requesting client by means of the **print()** in **Line 4**. Note that in this script, the same stream function **file_get_contents()** as in **agent.php** (where it was used to fetch a remote file) is here used for a local target with the scheme **file//**.

Uploading files

So far, we have been studying file processing on the server level and downloading files to the clients. In some applications, it is required that files can be **uploaded** from the clients to the server. The security risk connected to this feature should not be underestimated.

As an example, consider this online course. The **reports** to assignments given in the course should be **uploaded** to a specified directory on the server. The course instructor and the students should at any time be able to see the list of uploaded reports, and if wanted, read any of the uploaded files.

1. `<!-- index.htm --`
2. `<center>`
3. `<h1>Uploading and retrieving files</h1>`
4. `< p>Do you want to:</p>`
5. `<table>`
6. `<tr><td>See the list of submitted files</td></tr>`
7. `<tr><td>Upload a file from your own computer</td></tr>`
8. `</center>`

index.htm presents 2 options, **see** the list of uploaded files or **upload** a report file, as displayed in [Figure 6.2](#).



Figure 6.2: Uploading and retrieving files

The first option of the **index.htm** file is to get a **listing** of the uploaded files from the directory (**./file** relative to the script) in which the saved files are stored:.

1. `<!-- list.php -->`
2. `<?php`
3. `print("<center>");`
4. `print("<h2>List of uploaded files :</h2>");`
5. `$dir="./file";`
6. `if ($handle= opendir($dir)){`
7. `print("Files:
");`
8. `print("<table>");`
9. `while (($file = readdir($handle)) !=false) {`
10. `if ($file !=".." && $file !=".") {`
11. `print("<tr><td> $file </td></tr>");`
12. `}`


```

13. }
14. print("</table>");
15. closedir($handle);
16. }
17. else print("No files uploaded.");
18. print("</center>");
19. ?>

```

If the directory **exists**, it is opened with a returned handle, **\$handle**, and a list of the files stored in the directory is displayed with an **HTML** **a** tag for each file which permits the content of the individual files displayed. An example of the resulting list is shown in [Figure 6.3](#).



Figure 6.3: List of images

The second option results in a second **HTML** page, **upload.htm**. A few special aspects of this form page should be noted. **First**, the form tag must as usual have **POST** specified as method. **Second**, the tag must include the attribute **enctype="multipart/form-data"**, and, **third**, an input tag with attribute **type="file"** must be present.

```

1. <!-- upload.htm -->
2. <center>
3. <h2><font color="Blue">Uploading</font></h2>
4. <form action="upload2.php" method="POST" enctype="multipart/form-data">
5. <table>
6. <tr><td>Name to be assigned to file at server:</td> <td><input name="name" type="text"></td></tr>
7. <input name="MAX_FILE_SIZE" type="hidden" value="30000">
8. <tr><td>Identify the file at your computer to be uploaded:</td>
9. <td><input type="file" name="upload" > </td></tr>
10. <tr><td></td><td><input type="submit" Value="Upload file"></td></tr>
11. </table>
12. </form>
13. </center>

```

The form is displayed in [Figure 6.4](#)

Uploading

Name to be assigned to file at server:

Identify the file at your computer to be uploaded:

Figure 6.4: Uploading

The form page **upload.htm** calls the **PHP** script **upload2.php** which **confirms** the uploading of the file by means of the statements in **Line 3-8**, and stores the file by means of **Line 9 -10**. The **move_uploaded_file()** is another **stream function**.

```

1. <!-- upload2.php -->
2. <?php
3. $destination_file="./file/$_POST[name]";
4. move_uploaded_file($_FILES['upload']['tmp_name'], $destination_file);
5. print("A file with the following attributes has been uploaded: <br>");
6. print(" Remote name:".$_FILES['upload']['name']."<br>");
7. print(" File type:".$_FILES['upload']['type']."<br>");
8. print(" Size in bytes:".$_FILES['upload']['size']."<br>");
9. print(" Temporary name:".$_FILES['upload']['tmp_name']."<br>");
10. print(" Error code:".$_FILES['upload']['error']."<br>");
11. ?>

```

Note that the **concatenation** operator **'.'** is used in the **print()** functions in **Lines 6-9** to join the texts surrounded by double quotes with **PHP** array variables. You can see a confirming message in [Figure 6.5](#).

```

A file with the following attributes has been uploaded:
Remote name:MyFile.doc
File type:application/msword
Size in bytes:19456
Temporary name:C:\WINNT\TEMP\php69.tmp
Error code:0

```

Figure 6.5: Confirmative message

Session 7: Functions in PHP

Functions

One of the most frequently ways for re-using code is creating a **function**. A function can be considered as a **complex operator**, and is either a **built-in** function or a **user-defined** function. A function is recognized in the script by a name followed by a pair of **parentheses** which embraces none, one or several arguments. While the names of variables are case-sensitive in **PHP**, the function names are **not case-sensitive**.

We have already met a number of **built-in** functions such as **print()**, **mysql_query()**, etc. The internal, built-in functions come with the language processor and are available for calls when needed.

The **user-defined** functions must, as the name indicates, be defined by the developer and be made available before they can be called from a script. The syntax for defining a function is:

1. **function my_function (\$a, \$b, \$c) {**
2. **function code**
3. **return \$d**
4. **}**

The specified **arguments** **\$a**, **\$b** and **\$c** (your definition determines the number and names of the arguments) are called the **formal arguments** and are the names you use in the function definition. If the function returns any values, they are represented by **\$d** which can be a single variable or an array. A defined function can be called from a script if it is **accessible** for the script. It can be made available either by **copying** the complete function definition into the script, or by **inserting** an **include** ("**my_functions.php**") in the beginning of the calling script file. The advantages of the latter option are **saved** space, and that the file **my_functions.php** can contain **several** user-defined functions needed to be called from several scripts.

A **call** from a script to a user-defined function has the syntax **\$D=myfunction(\$A, \$B, \$C)**. The argument names **\$A**, **\$B**, and **\$C**, called the **actual arguments**, are the variable names used in the calling script for the variable values you want to pass to the function. It is important to note that **these values** are given to the formal variables **\$a**, **\$b** and **\$c** within the function while the variables **\$A**, **\$B** and **\$C** outside the function maintain their values independent of the computations within the function. The function value **\$D** is the output **return** from the function available to the calling script from the return variable **\$d** in the definition. In some cases, a function may have **no output** and this variable and the assignment operator are not needed. The function can also return several values in which case **\$D** is an array. By means of the **return** value(s) can the external values **\$A**, **\$B** and **\$C** be changed if required.

Note that the symbols used here for formal and actual variables are examples. You are free to use the names of your own choice in your functions and applications.

Authorization and authentication

By **authorization** we mean the assignment of access identities to a web site visitor.

Authentication is the checking of the validity of identities provided by a visitor to obtain access to the site. When designing a web application, authorization and authentication is frequently required functionalities for varying reasons. The site owner may for example want to know **who** are visiting the site, to be able to provide **personalized service** to customers, to keep track of the **performance** of student visitors, etc.

These functionalities are used with minor variation in different applications. As an example of more user-defined functions, we will design a **login** module which can perform authorization and authentication in web applications. [Figure 7.1](#) indicates the overall structure of the login

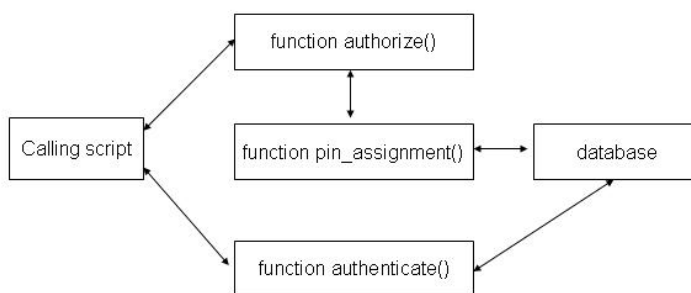


Figure 6.1: Overall view of the A&A functions

functionality. There are alternative designs which could have been used. If for example it is important to preserve the anonymity of the users, we could have asked the user to select a **PIN** code herself and let the server **hash** the selected code and check that it is **free**. The **PIN** external code would then be **unavailable** for the host while the anonymous internal code could be used for analysis of the site visits. However, a hash code approach would not have satisfied applications in which it is required that the host can **recognize** the users' external identities, e.g. in e-shop billing and e-courses with grading.

We start the example application by developing **2** ordinary **HTML** pages. The first, **index.htm**, generates the **login form** returned to the visitor when calling the application.

1. `<!-- index.htm -->`
2. `<center>`
3. `<h1> Login</h1>`
4. `</center>`

5. Thank you for your interest in this site. To get access to the application, you have to be authorized.<i>If you already are registered</i>, please go directly to the login.

6. <i>If you are new and want access to the application</i>, we need some information from you, and you will need a personal identity number (PIN). Please continue with the registration.
7. <p></p>
8. <center>
9. <table>
10. <tr><td>Login with your</td></tr>
11. <FORM ACTION="functioncalls.php" method="post">
12. <tr><td>Your username:</td> <td><input name="username" type="text" size="20"></td> </tr>
13. <tr><td>Your PIN code:</td><td> <INPUT TYPE="password" name ="submitted_pin" SIZE="20"></td></tr>
14. <input name="login" type="hidden" value="1">
15. <tr><td>Click the button:</td> <td><INPUT TYPE="SUBMIT" NAME="response" VALUE="Submit"></td></tr>
16. </FORM>
17. </table>
18. </center>

This form provides the server with the variables **username** and **PIN** used for authentication. The only new aspect in this page is the use of the attribute **HIDDEN** in the **INPUT** tag in **Line 14**. The **hidden variable** named **login** with value="1" is **invisible** for the client. By means of this variable, the server will be able to distinguish the variables sent by this page from those sent from the next form. See [Figure 7.2](#). The form calls the script **functioncalls.php** which represent an application for which we require controlled login.

Login

Access to this site is restricted to registered visitors only. If you already are registered, please go directly to the login.
If you are new and want to become a registered user, we need some information from you, and you will need a personal identity number (PIN).
Please continue with the [registration](#).

Login with your

Your username:

Your PIN code:

Click the button:

Figure 7.2: Login

The form also includes a link to **registration.htm** for visitors who are **not yet registered**. The form in this second **HTML** page collects the information required for **authorization**, in this example **first** and **last** name and a **username** chosen by the visitor. Other personal information such as gender, age and home region, can of course be included if required.

1. <!-- registration.htm -->
2. <html>
3. <head>
4. <title>applications</title>
5. </head>
6. <center>
7. <h1>Registration</h1>

8. `<p>In order to recognize and serve the different requirements of our visitors, each visitor needs her/his own username and PIN code.
 Please, complete and submit the form, and your username and PIN code will be returned to you.</p>`
9. `<FORM ACTION="functioncalls.php" method="post">`
10. `<table>`
11. `<tr><td>Your first name:</td> <td><input name="firstname" type="text" SIZE="20"></td> </tr>`
12. `<tr><td>Your last name:</td> <td><input name="lastname" type="text" SIZE="20"></td> </tr>`
13. `<tr><td>Your user name:</td><td> <INPUT TYPE="text" name ="username" SIZE="20"></td></tr>`
14. `<input name="registration" type="hidden" value="1">`
15. `<tr><td>Click the button:</td> <td><INPUT TYPE="SUBMIT" NAME="response" VALUE="Submit"></td></tr>`
16. `</table>`
17. `</FORM>`
18. `</center>`
19. `</body>`
20. `</html>`

Note that also this page has a **hidden input variable**, named **registration** with value="1", for the same reason as the first form. This form is shown in [Figure 7.3](#). This form specifies the same **PHP** script, **functioncalls.php**, for server processing as did the first form.



Figure 7.3: Registration

The **PHP** script **functioncalls.php** can be considered as the **application** script of the example. It contains several new features. **Line 3** **includes** another file, in this case **functions.php**. The file **functions.php** contains **4** functions we shall discuss in detail below.

The **Lines 2-11** check the existence of **MySQL**, the database **db** and the table **Users**. **Line 12** has an include call to a file called **my_functions.php** in which the definition of our functions reside. We shall return to the structure of this file below after we have discussed the definitions of the functions.

1. `<!-- functioncalls.php -->`
2. `<?php`
3. `//Open connection/database/table`
4. `$link=mysql_connect("localhost","root",'password');`
5. `);`
6. `$db="db";`


```

7. $db_selected=mysql_select_db($db, $link);
8. if(!$db_selected) {
9.     mysql_query("CREATE DATABASE $db", $link);
10.    mysql_select_db($db, $link);
11.    mysql_query( "CREATE TABLE Users (firstname VARCHAR(20), lastname VARCHAR(20), email
        VARCHAR(20), PIN VARCHAR(10))", $link);
12. }
13. include "my_functions.php";
14. //Function calls
15. if (isset($_POST['login'])) {
16.     $approved=authentication($db, $_POST['username'], $_POST['submitted_pin']);
17.     if ($approved[0]=="yes")
18.         print("<h2><center><font color=blue>$approved[1], you are logged in</font></center></h2>");
19.     else
20.         print("<p><center><font color=red>Your PIN code was invalid</font></center></p>");
21. }
22. if(isset($_POST['registration'])) {
23.     $reg=authorization($db,$_POST['firstname'],$_POST['lastname'], $_POST['username']);
24.     print("<center><font color=blue>You have been successfully authorized to access the site<br>
        Your username is: $reg[0], and your PIN is: $reg[1].<p></p>");
25.     print("<a href=index.htm>Return </a>to Login.</center> ");
26. }
27. }
28. mysql_close($link);
29. ?>

```

In **Line 15** the existence of the **hidden variable login** is tested by means of the built-in function **isset()**. If it has been assigned a value, the server will know that it is a **login form** it has received, and a **function call** to the function **authentication()** is made in the following line. The authentication function requires **3** arguments, the database handle **\$db**, as well as the **\$username** and the **\$PIN** submitted by the user on the login form. A return value for the variable **approved** is expected from the function. If the returned value is **"yes"**, a message is sent back to the client in **Line 18** **confirming** that the user is logged in. If not, the login **failed**, and a message about the failure is sent by the next couple of lines.

If a value for the hidden variable **registration** is received from the client instead of **login**, the function **authorization()** is called in **Line 23..** This function is expected to return the values of **username** and **PIN**. The function **output** must therefore include **2** variables, in this case the **username** and the **PIN** in an **array** with **2** elements defined within the function. See [Figure 7.4](#) for illustration of a successful registration

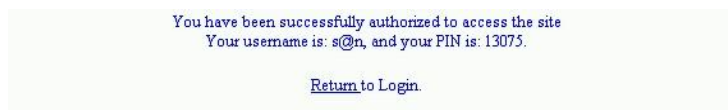


Figure 7.4: Successful registration

Authorization

We proceed to the definition of the functions, and start with the definition of **authorization()**. This function should be able to generate a **new, unused PIN**, append the row with the person's first name, last name, user name and generated **PIN** in the created table of the database.

The task of **generating** a new **PIN** code is left to another function, **pin_assignment()** which only requires the database handle, **\$db**, as argument. This function, which is the next to be discussed, delivers a new and unused **\$PIN**. Note that one function can call on another function.

The data for the new visitor is **inserted** into the database table **Users** by a **mysql_query()** function. We want the function to return 2 variable values, **\$username** and **\$PIN**, in an array which is obtained by the return statement in **Line 5**.

```
1. //Authorization.php
2. function authorization($db,$firstname, $lastname, $email){
3.   $PIN=pin_assignment($db);
4.   mysql_query("INSERT INTO Users(firstname, lastname, email, PIN)
      VALUES('$firstname','$lastname','$email', '$PIN')");
5.   return array($email,$PIN);
6. }
```

PIN code assignment

We saw in the previous function definition that **one function can call another**. As a matter of fact, a function can also call itself which results in a **recursion**. In the **pin_assignment()** function, the first step is to retrieve all records (arrays) from database table **Users** in **Line 3**, and set initially the test variable **\$used="yes"** before entering the random number generation and testing **while** loop in **Line 5**.

In **Line 6** a random **seed** is planted followed by generation of a random integer, **\$PIN2**, in the range **1000** to **99999** (these limits can be set according to the particular needs). The random seed is planted to **avoid** that the same sequence of number is generated each time the application is run.

The variable **\$used** is now re-set to **"no"**. A **test** to see if the generated number has been used before, runs from **Line 9** to **13**. Here **\$result** is the array corresponding to the current row **\$r** of the database table and **\$result['PIN']** is the value of the **PIN** column of that row. If this **\$result['PIN']** is **identical** (**=**) to the generated **\$PIN2**, the generated integer is **used**, and variable **\$used** is assigned the value **"yes"**. There is no reason to continue the test, the loop is therefore **broken** by the instruction **break** statement in **Line 12**, and the processing is directed back to **Line 5**.

```
1. //Pin assignment
2. function pin_assignment($db) {
3.   $r= mysql_query("Select * FROM Users");
```



```

4. $used="yes";
5. while ($used=="yes"){
6.   srand();
7.   $PIN2=rand(1000,99999);
8.   $used="no";
9.   while($result=mysql_fetch_array($r)) {
10.    if ($result['PIN']==$PIN2) {
11.     $used="yes";
12.     break;
13.    }
14.   }
15. }
16. return $PIN2;
17. }

```

Each time the control is returned to **Line 5**, a new integer is generated and the test **repeated**, until an integer not used in the table **Users** is found, i.e. the test loop is exited with **\$used="no"**. Then the unused integer **\$PIN2** is **returned** to the calling script.

Authentication

The third function we need is the **authentication()**. The call passes **\$db**, submitted **\$username** and **\$PIN** code to this function. The task for this function is to check if the passed values are **valid** according to the **User** table. To avoid problems with **disturbing white space**, etc. in connection with the string values, the **mysql_escape_string()** function is used in **Line 3** and **4**. **2** arrays are declared in the next lines. **\$result** will be used for storing the content of a retrieved row (usually **0** or **1** matching row) from table **Users**, while the array **\$approved** will be used to return **2 values** to the calling script.

```

1. //Authentication
2. function authentication($db, $username, $PIN) {
3.   $username2=sqlite_escape_string($username);
4.   $PIN2=mysql_escape_string($PIN);
5.   $result=array();
6.   $approved=array();
7.   $r= mysql_query("Select * FROM Users WHERE PIN='$PIN2'");
8.   while ($result=mysql_fetch_array($r)) {
9.     if ($result['email'].$result['PIN']==$username2.$PIN2) {
10.      $approved[0]="yes";
11.      $approved[1]=$result['firstname'];
12.     }
13.   else $approved[0]="no";
14.   return $approved;
15. }
16. }

```

After possible retrieval of a row where the **PIN** column has the value **\$PIN2**, the content is inserted into the array **\$result** in **Line 8**. In the next row, we use the **concatenation operator** (".")

and 2 concatenated strings, `$result['username'].$result['PIN']` from the table **Users** and `$username2.$PIN2`) from the client, are compared in **Line 9**. By concatenating **username** and **PIN** code, we have the extra security that the combination of **username** and **PIN** code are validated.

If the submitted data are validated, the first element `$approved[0]` of the array **\$approved** is assigned the value **"yes"**, and the second element `$approved[1]` is assigned the value `$result['firstname']` from the matching row of the database. If not validated, the first element is set to **"no"**. The last action is to return the array **\$approved** to the application script.

Function library

The 4 functions created for this application can either be copied into the **PHP** files in which they are called, or, as we have done in this example, simply collect all functions in a library file called **my_functions.php**. The structure of this file is:

```
1. //my_functions.php
2. <?php
3. //Name: My_functions.php
4. //Function:mysql_table_exists is copied here
5. //Function: authorization is copied here
6. //Function: pin_assignment is copied here
7. //Function: authentication is copied here
8. ?>
```

The library file is stored in the same directory as the other files of the example. The functions are made available to the application with the **include()** statement as demonstrated in the script **functioncalls.php**.

More function can be added into the file which in fact becomes a **library file** and can be used in different applications which need one or more of the functions. The advantages are that all functions are kept in one location making maintenance more effective.

Logging

To demonstrate the last statement, and to introduce a few more aspect, we introduce a second example, the **logging function**. By logging we mean recording when authorized users are passing specified observation points in our application system. Logging is particularly important for the application usage analysis when studying how a system is **actually used** and for preparing **improvements** based on this experience.

Logging function

Below is a function definition, **logging.php**, which requires a single argument, a **PIN identification**. The value of the current user's PIN is frequently available as `$_SESSION['PIN']`, and can be used as argument in calling this function.

```
1. //Logging.php
2. function logging($PIN) {
3.     if(!file_exists('log.htm')) {
4.         $f=fopen('log.htm','wb');
5.         fwrite($f, "LOG FILE<br>");
6.         fclose($f);
7.     }
8.     $time=strftime('%c');
9.     $record="$time, $PIN, $_SERVER[SCRIPT_FILENAME]<br>";
10.    $stream='log.htm';
11.    $f=fopen($stream,'aw');
12.    fwrite($f, $record);
13.    fclose($f);
14. }
```

The function specification, **logging.php**, is stored as the fifth function in the **my_functions.php** file.

Example environment

To be able to **demonstrate** the use of this function, we shall need a few more surrounding files: a menu **index.htm**, **start.php**, **view.php**.

The **index.htm** looks like this:

```
1. <!-- index.htm -->
2. <html>
3. <head>
4. <title>Index.htm</title>
5. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
6. </head>
7. <body>
8. <center>
9. <h2><font color="blue">Logging menu</font></h2>
10. Do you want to:
11. <table>
12. <tr><td>Make a log entrance:</td><td><a href="start.php">Yes</a></td></tr>
13. <tr><td>See the log:</td> <td><a href="view.php">Yes</a> </td></tr>
14. </table>
15. </center>
16. </body>
```


17. </html>

The only justification of this page is to demonstrate either **initiating** a log record or **seeing** the log file. [Figure 7.5](#) demonstrates the displayed page.



Figure 7.5: Login menu

The purpose of the **start.php** script is to **simulate** the entering into an application. The first statement, **Line 2**, of this script include our function library **my_functions.php** which makes the function **logging()** available. In a real application a user will have logged in with a **PIN** code. In our example, we **assume** that the in PIN code is "**1234**". When our activation of the **start.php** has been logged, we will get a message in return. In a real application there are usually no need for returning messages each time a logging has been carried out.

```
1. <?php
2. include('my_functions.php');
3. $PIN='12345';
4. logging($PIN);
5. print("<center><font color=blue>Your visit to this test page Index.php has been
   logged</font><center>");
6. ?>
```

To complete out exemplification of the logging function, we also need a short script we can call (the second option of the **index.htm**) to **view** the log file. **view.htm** serves this purpose. In this script, we make use of **\$stream** and the powerful **file_get_contents()** function. This function **opens, reads the content of and closes** the specified file. The contents of the file is read into a string variable (in the example **called \$contents**).

```
1. <?php
2. //view.php
3. $stream='log.htm';
4. $contents=file_get_contents($stream);
5. print("$contents");
6. ?>
```

An example of a recently started log can be seen in [Figure 7.6](#). Try to identify scripts in your own application work which you think can be used in other applications and start building you private library of user-defined functions. Note that the file **log.htm** with the recorded events is located in the same directory as the **logging.php**. In real applications the record file should be kept in some other directory.


```
LOG FILE
10/17/06 22:58:57, 12345, C:/PHPRoot/oca/Courses/31122849115/php_mysql/sessions/session7/examples/Logging/start.php
```

Figure 7.6: Log start

Parsing

Frequently, an application requires that each word of a text is identified and recorded. The process of traversing a text, locating and recording each word is called **parsing**. Parsing is a basic requirement for building search engines and text retrieval systems.

We shall see how a parsing function can be designed and developed. We start with a simple form page, **parser.htm**, by which the name of the file to be parsed is specified. We assume that the text to be parsed is located at the client computer while the 'application' and the parser function reside at the server, and prepare a simple **.htm** form page:

1. `<!-- index.htm --->`
2. `<center>`
3. `<h2>Text file indexing</h2>`
4. `<p>Select a .txt file from your pc:</p>`
5. `<form action="application.php" method="POST" enctype="multipart/form-data">`
6. `<input type="hidden" name="MAX_FILE_SIZE" value="50000">`
7. `<table>`
8. `<tr><td>Name of your local file:</td><td><input type="file" name="client_file"></td></tr>`
9. `<tr><td></td><td><input type="submit" value="Submit"></td></tr>`
10. `</table>`
11. `</form>`
12. `</center>`

We note that the **form** tag in **Line 5** includes the **enctype** attribute with value **multipart/form-data**, the hidden **input** with attribute **MAX_FILE_SIZE** and the **input** tag of type **file** which all are **necessary** for file uploading. When submitted to the server, the script **application.php** is activated. See [Figure 7.7](#).

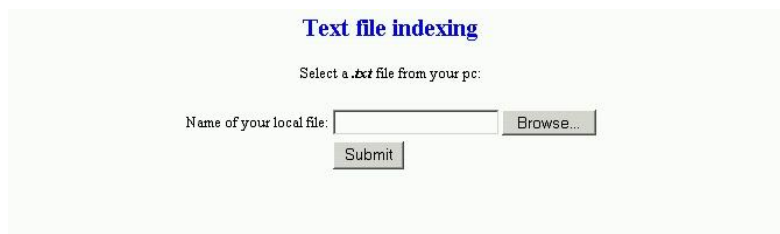


Figure 7.7: Text file indexing

The purpose of the **application.php** is to represent any application which requires a specified text file to be parsed. The parsing is carried out by a function called **parser.php** which we have saved in our library of functions, **my_functions.php**. We return to the content of this function below. Here we start by noting that our library of functions is included in **Line 3**.


```

1. <!-- application.php -->
2. <?php
3. include('my_functions.php');
4. $destination_file='./file/server_file';
5. move_uploaded_file($_FILES['client_file']['tmp_name'], $destination_file );
6. $c=file_get_contents($destination_file);
7. print("<h2><font color=blue>Text:</font></h2>$c");
8. $document_size=strlen($c);
9. print("<p>Document size:<b> $document_size</b> characters</p>");
10. $frequency_list=array();
11. $rx="/[a-zA-Z]/";
12. $frequency_list=parser($c,$rx);
13. arsort($frequency_list);
14. $rows=0;
15. $sum=0;
16. print("<center><h2><font color=blue>Word frequencies.</font></h2>");
17. print("<table>");
18. foreach($frequency_list as $key => $value) {
19. print("<tr><td>$key</td><td>$value</td></tr>");
20. $sum=$sum+$value;
21. }
22. print("</table><br>");
23. $rows=count($frequency_list);
24. print("Number of rows: <b>$rows</b><br>");
25. print("Sum of words: <b>$sum</b></center>");
26. ?>

```

In **Lines 5-6**, the test file uploaded is stored and retrieved as a string, **\$c**. **Line 8** demonstrates how we can obtain the **size** in characters of the uploaded file by means of the **built-in function strlen()**. **Lines 10-11** are preparations for the call to the our **user-built function, parser(\$rx, \$c)**, the declaration of the array **\$frequency_list** is to prepare for the **output of the function**, and the definition of **\$rx** specified the **conditions** for locating the words in the text. The concept **word** can be defined in several ways. In this example, a word is a **consecutive** chain of characters in the ranges **a-z** and **A-Z**. The content of the variable **\$rx**, **"/a-zA-Z/"**, is called a regular expression written in the **Perl Compatible Regular Expression** syntax.

The **core line** in this application is **Line 12** which calls the **parser()** function. The function has **2** arguments, **\$c** containing the **document string**, and **\$rx** determining **how** the string shall be parsed. The value of the function, **\$frequency_list** is an array containing the different words appearing in the document as **indices** and the frequency of appearance as **values**. Since there is no reason for distinguishing between actual and formal arguments, we use the same notation for both.

The remaining part of the script specifies how the results should be **visualized**. **Line 13** says that the frequency list should be presented in **descending** order, the variables **\$sum** keeps track of the **number of words** in the text and the variable **\$rows** tells us the **number of different words** identified.

The function **parser(\$file)** looks like this;

```
1. <!-- parser.php -->
2. <?php
3. function parser($c,$rx) {
4.   $c=strtolower($c);
5.   $document_size=strlen($c);
6.   $frequency_list=array();
7.   $characters_processed=0;
8.   $word="";
9.   while($characters_processed < $document_size) {
10.    $char=substr($c, $characters_processed,1);
11.    if (preg_match($rx, $char)) {
12.     $word=$word.$char;
13.     $characters_processed++;
14.    }
15.    else {
16.     $word=trim($word);
17.     if (!preg_match($rx, $word)) {
18.     }
19.     else {
20.      if(!array_key_exists($word, $frequency_list)) {
21.       $frequency_list[$word]=1;
22.      }
23.      else {
24.       $frequency_list[$word]++;
25.      }
26.     }
27.     $word="";
28.     $characters_processed++;
29.    }
30.   }
31.   return $frequency_list;
32. }
33. ?>
```

Note that within the function the function arguments can be recognized as variables. The first task is to standardize all words to lower case. The built-in function in **Line 4** takes care of this. . The built-in function, **strlen()**, in **Line 5** can therefore recognize the text in **\$c** and derive the **number of characters** in the text. In a loop, which span from **Line 9** to **Line 30**, each character of the string, **1** at a time, is extracted by the function **substr(\$c, \$characters_processed, 1)** in **Line 10** .

Line 11 contains an **if()** with a function **preg_mach(\$rx, \$char)** which is **true** if the character value of **\$char** belongs to the expression **\$rx**. In this case, the character will be added to **\$word** in **Line 12**. If the function yields **false**, it means that the word has come to an **end**, and it should be recorded. However, it is necessary to test that the 'word' is not a **non-alphabetic character** which is done by means of the second regular expression in **Line 17**. If the **\$word** contains a **real word**, **Line 20** test if it is a new word, and if so, adds a new row with value **1** to the

\$frequency_list, else it increments the value of the row in which **\$word** is recorded by the ++ operator and **Line 27** makes the variable **\$word** ready for starting a new word.

When the text is processed, the compiled frequency list is returned to the calling **application.php** as an array and displayed for the client.

This function is added to the file **my_functions.php**.

Session 8: Information retrieval

General model

As an introduction to use of **MySQL** in **Session 4**, we studied an application to keep track of our private library. It had **2** deficiencies:

- the application required that a classification for each book was provided for recording, and
- the book itself was not electronically available for reading.

Search engines are becoming tools for obtaining information as well on the internet as on local nets. In this session we shall investigate some of the basic problems and solutions connected with systems developed for searching and retrieval of data. There exist a great variety of systems and we have to limit ourselves to a simple system which stores text files and permits users to search for a wanted file(s) based on the content of the file. In [Figure 8.1](#), the general layout for our

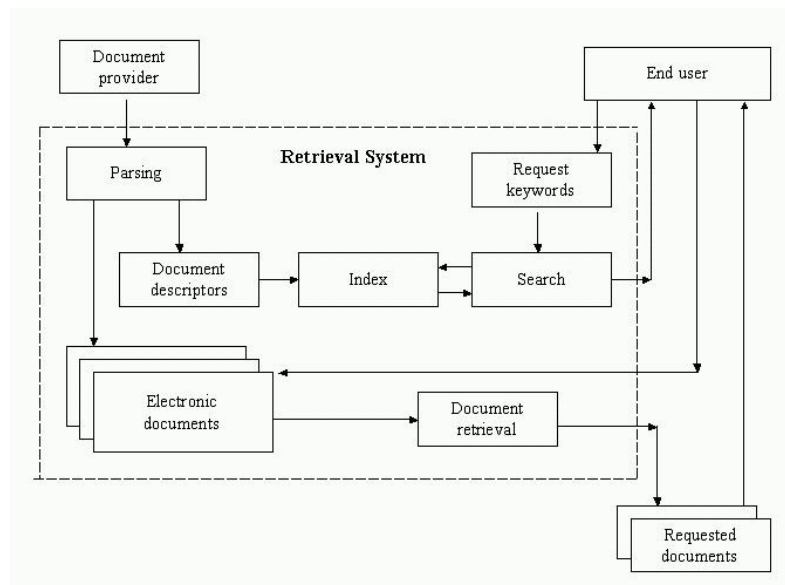


Figure 8.1: Retrieval system

model is shown. As common for all applications discussed in this course, we assume that the system is hosted by a remote server, and that all communication with the system on the net. In principle, there are 3 interfaces to the system:

1. Feeding the system with text files to be indexed and stored,
2. Searching for references to and retrieval of text files,
3. Administrating the system

Each interface is served by a separate module which we shall discuss in the following sections.

Index module

In the most primitive version, we need a possibility to submit text files to the system, and our first step will be to create an **.htm** form page for uploading a named text file. We have already studied several uploading examples.

1. `<!-- index.htm -->`
2. `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"`
3. `"http://www.w3.org/TR/html4/loose.dtd">`
4. `<html>`
5. `<head>`
6. `<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">`
7. `<title>Untitled Document</title>`
8. `</head>`
9. `<body>`
10. `<center>`
11. `<h2>Uploading a text file</h2>`
12. `<p>This form is for uploading text files to the search and retrieval system. You need to specify 2 names associated with the file, the local name of the file at your computer, and the name to be carried by the file in the system. The latter should be as descriptive as possible.</p>`
13. `<form action="descriptor.php" method="post" enctype="multipart/form-data">`
14. `<input type="hidden" name="MAX_FILE_SIZE" value="50000">`
15. `<table>`
16. `<tr><td>Local file name:</td><td><input name="local_file" type="file"></td></tr>`
17. `<tr><td>Name in the system:</td><td><input name="system_file" type="text"></td></tr>`
18. `<tr><td></td><td><input type="submit" value="Submit"></td></tr>`
19. `</table>`
20. `</form>`
21. `</center>`
22. `</body>`
23. `</html>`

The form permits to specify a document file (**.txt**, **.php**, **.doc**, etc.) on the client computer and request it uploaded to the server naming it with another name if wanted. The file server name should be a name describing the file and have the extension **.htm** if the files normally will be read from the screen. We are now well acquainted with uploading of files, and there are no special tricks hidden in this page. Note that the page is calling **descriptor.php** and that you must remember to include the important hidden input in Line 14. The displayed form is shown in [Figure 8.2](#).

Figure 8.2: Uploading a text file

The form calls the **descriptor.php** script for processing of the uploaded file. The purpose of this script is twofold:

1. Analyze the content of the file, derive descriptors and save the result in an appropriate way.
2. Store the complete file for future retrieval

descriptor.php starts by including the library file **my_functions.php** because we shall make use of the function **parser.php** developed in the last session. **Lines 4-11** connects to **MySQL**, select the database **ir** if it exists or creates the database **ir** and table **descriptors** if they do not exist. Note the use of the function **die()** with argument **mysql_error()** demonstrated in **Line 10**. If, for some reason, the table descriptors cannot be created, this function can be informative during the creation and debugging of the script.

```

1. <!-- descriptor.php -->
2. <?php
3. include('../..../my_functions.php');

4. $link=mysql_connect('localhost','root','password');
5. if (!$link) die("<center><h3><font color=red>Install MySQL. </font></h3></center>");
6. $db_selected= mysql_select_db("ir", $link);
7. if (!$db_selected) {
8. mysql_query("CREATE DATABASE ir",$link) or die("<center><h3><font color=red>Cannot create
database ir. </font></h3></center>");
9. mysql_select_db("ir", $link);
10. mysql_query("CREATE TABLE descriptors(id INT NOT NULL AUTO_INCREMENT, PRIMARY
KEY(id), word VARCHAR(20), document_words VARCHAR(6), descriptor_frequency VARCHAR(6),
document_ref VARCHAR(40) )", $link)
or die("<center><h3><font color=red>Cannot create table Descriptors.</font></h3></center>
".mysql_error($link));
11. }

12. $document_ref="../documents/$_POST[system_file]";
13. move_uploaded_file($_FILES['local_file']['tmp_name'], $document_ref );
14. $c=file_get_contents($document_ref);

15. print("<h2><font color=blue>Indexing summary:</font></h2>");
16. $size=strlen($c);
17. print("Document size:<b> $size</b> characters<br>");

18. $frequency_list=array();

```



```

19. $rx="/[a-zA-Z]/";
20. $frequency_list=parser($c,$rx);

21. arsort($frequency_list);

22. $rows=0;
23. $document_words=0;

24. foreach($frequency_list as $key => $value) {
25. $document_words=$document_words+$value;
26. }

27. $rows=count($frequency_list);

28. print("Number of words: <b>$document_words</b><br>");
29. print("Number of unique words: <b>$rows</b><br>");

30. $stop_list=array();
31. $stop_list=unserialize(file_get_contents('../AdminModule/stop_list.txt'));

32. $descriptor_list=array();
33. $number=1;
34. foreach($frequency_list as $key => $value) {
35. if (strlen($key) >2) {
36. $stop_marker=0;
37. foreach($stop_list as $key_stop => $value_stop) {
38. if ($key == $value_stop) {
39. $stop_marker=1;
40. }
41. }

42. $percent=100*$value/$document_words;
43. if (($stop_marker == 0) && ($percent >= 0.1)) {
44. $descriptor_list[$key]=$percent;
45. $number++;
46. }

47. }
48. }

49. print("<h3>List of selected descriptors </h3>");
50. print("<table>");
51. print("<tr><th><strong>Descriptor</strong></th><th><strong> Pct. of words</strong></th></tr>");
52. foreach($descriptor_list as $key => $value) {
53. print("<tr><td>$key</td><td>$value</td></tr>");
54. }
55. print("</table>");

56. foreach($descriptor_list as $key => $value) {
57. $descriptor_frequency= $value*$document_words/100;
58. mysql_query("INSERT INTO descriptors(word, document_words, descriptor_frequency, document_ref)
VALUES('$key', '$document_words', '$descriptor_frequency', '$document_ref')",$link) or die("Nothing
INSERTED.<br>".mysql_error($link));
59. }

60. ?>

```


Lines 11-13 **upload** the specified file from the client to the server and save it in the directory documents, and create a string copy, **\$c**, for further analysis. In **Line 20**, the function **parser.php** is used for decomposing the text string **\$c** into an array, **\$frequency_list**, applying the regular expression **/[a-zA-Z]/**.

The **frequency_list** contains all unique words appearing in the document with this frequency. You can see an example in [Figure 8.3](#). A number of the most frequent words, such as pronouns,

Indexing summary:

Document size: **1458** characters
Number of words: **223**
Number of unique words: **115**

List of selected descriptors

Descriptor	Pct. of words
color	4.0358744394619
php	2.2421524663677
web	1.3452914798206
previously	0.89686098654709
logging	0.89686098654709
file	0.89686098654709
creating	0.89686098654709
interface	0.89686098654709
session	0.89686098654709
earlier	0.89686098654709
content	0.89686098654709
include	0.89686098654709
information	0.89686098654709
log	0.89686098654709
applications	0.89686098654709
cfm	0.89686098654709
course	0.89686098654709

Figure 8.3: Indexing summary

prepositions, conjunctions, etc., are not useful descriptors for a document. For that reason it is efficient to create a so-called **stop word list** containing words we want to remove from the frequency list. **Line 31** assumes that such a list, **stop_list.txt**, has been created and stored in the **AdminModule** directory. This file is a **serialized** form of an array of all stop words. The file has been serialized by the built-in function **serialize()** to a form which is convenient for storing in a file. The file must therefore be retrieved and converted back to an array, **stop_list**, by a function **unserialize()**. In **Lines 32-41**, an array of **descriptors** is created from the **frequency_list** by removing all words with **1** or **2** characters, and using the **stop_word** array.

At the other end of the word frequency distribution, we have the words with a rare occurrence. If a word counts for, say, less than **0.1 %** of all words of a paper, the probability is usually **low** that the word is **significant** for the content of the document. A second removal is therefore carried out in **Lines 42-46** in which only words occurring with relative frequency $\geq 0.1 \%$ are kept in the **descriptor_list** array. Finally, in the block of **Lines 49-59**, the selected descriptors are displayed and **inserted** in the database for use in future searches. Each descriptor within each document is recorded in a **separate** row with data on the document's total number of words, the descriptor's frequency and a link to the document.

Search module

Our purpose with the system is to identify those documents in the collection containing a content in which we are interested. We specify our interest in a request created by a set of keywords which we hope will match the descriptors of wanted documents. In the form HTML page search.htm the user can specify her/his keyword(s). Separate multiple keywords with the symbol ' '.

```

1. <!-- search.htm -->
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
3. <html>
4. <head>
5. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
6. <title></title>
7. </head>
8. <center>
9. <h2><font color="blue">Search for documents</font></h2></center>

10. <p>This form let you search for documents in your collection based on document descriptors - frequently
    appearing appearing terms in the documents. The words have been extracted from the documents by full-text
    parsing and selected according to the frequencies of appearance. Non-informative words such as pronouns,
    conjunctions prepositions as well as very general words for describing time such as day, night, month, etc. are
    not suitable descriptors.</p>
11. <p>In requests for professional documents, you should therefore use words from the professional
    vocabulary such as file, database, system, program, web, etc. You can use a single search word, for examples
    '<b>internet</b>', or you can form a search string like <b>word, word, word3</b>. </p>
12. <center>
13. <form action="search.php" method="post">
14. <table>
15. <tr><td>Search string:</td><td><input name="search_string" type="text " size="50"></td></tr>
    <tr><td></td><td><input type="submit" value="Submit"></td></tr>

16. </table>
17. </form>
18. </center>

```



```

19. <body>
20. </body>
21. </html>

```

The form page ([Figure 8.4](#)) calls the script **search.php** for retrieving links to documents with

Search for documents

This form let you search for documents in your collection based on document descriptors - frequently appearing terms in the documents. The words have been extracted from the documents by full-text parsing and selected according to the frequencies of appearance. Non-informative words such as pronouns, conjunctions prepositions as well as very general used words for describing time such as day, night, month, etc. are not suitable descriptors.

In requests for professional documents, you should therefore use words from the professional vocabulary such as file, database, system, program, web, etc. You can use a single search word, for examples **Internet**, or you can form a search string like **word1 OR word2 OR word3**.

Search string:

Figure 8.4: Search for documents

matching descriptors. Note that the existence of tables in an existing database can be tested by an SQL statement as shown in Line 7. After connecting to and opening the database **ir**, the script converts the search string read from the form to an array, search array, by means of a new **PHP** function in **Line 10**. The statement **\$a=explode(\$b,\$c)** divides the string **\$c** into sub strings based on the separator **\$b** and stores the sub strings in array **\$a**. In **Lines 13-23**, the array is read and a search condition string of the form **(('word1') OR ('\$word2') OR ('\$word3'))** is formed. This procedure is required for use in the following **SQL SELECT** statement for retrieving matching records in the database.

```

1. <?php
2. $link=mysql_connect('localhost','root','password');
3. if (!$link)
4. die("<center><h3><font color=red>Install MySQL. </font></h3></center>");

5. $db_selected= mysql_select_db("ir", $link); 6. if (!$db_selected) {die("<center><h3><font
color=red>Database ir does not exist.</font></h3></center>");}

7. if (!$t=mysql_query("SHOW TABLES FROM ir", $link)) {die("<center><h3><font color=red>Table does
not exist.
8. </font></h3></center>");}

9. $search_array=array();
10. $search_array=explode(",",$_POST['search_string']);

11. $search_condition="";
12. $keyword="";

13. foreach($search_array as $keyword) {
14. if($search_condition == "") {
15. $keyword=trim($keyword);
16. $search_condition="((word ='$keyword')";
17. }

```



```

18. else {
19. $keyword=trim($keyword);
20. $search_condition=$search_condition." OR (word='".$keyword."')";
21. }

22 }

23 $search_condition=$search_condition.'";

24. print("<p><center><b><font color=red>Your query condition:</font> $search_condition</b></p>");

25. $r=mysql_query("Select * FROM descriptors WHERE $search_condition", $link);
27. $row = mysql_fetch_array($r);

28. if (!isset($row[0])) {
29. die("<h3><font color=red><b>No match with search condition in database.</b></font></h3>");
}

30. print("<table BORDER>");
31. print("<caption><font color=blue><h3>Documents retrieved</h3></font></caption>");
32. print("<tr><th><b>Document link</b></th><th><b>Document score</b></th></tr>");

33. $previous=array();
34. $first_row=1;

35. while ($row = mysql_fetch_array($r)) {

36. if ($first_row == 1) {

37. $previous[0]=$row[0];
38. $previous[1]=$row[1];
39. $previous[2]=$row[2];
40. $previous[3]=$row[3];
41. $previous[4]=$row[4];
42. $first_row=0;
43. }
44. else {

45. if ($row[4] == $previous[4] ) {

46. $row[3]=$previous[3] +$row[3];

47. $previous[0]=$row[0];
48. $previous[1]=$row[1];
49. $previous[2]=$row[2];
50. $previous[3]=$row[3];
51. $previous[4]=$row[4];
52. }
53. else {

54. print("<tr><td><a href=$previous[4]>$previous[4]</a></td><td align=center>$previous[3]</td></tr>");

55. $previous[0]=$row[0];
56. $previous[1]=$row[1];
57. $previous[2]=$row[2];

```



```

58. $previous[3]=$row[3];
59. $previous[4]=$row[4];
60. }
61. }
62. }
63. print("<tr><td><a href=$previous[4]>$previous[4]</a></td><td align=center>$previous[3]</td><tr>");
64. print("<p></p>");
65. print("</table>");

66. print("<p><b>Document score is the sum of the frequencies of all search words in each
document</b></p>");
67. print("</center>");
68. mysql_close($link);
69. ?>

```

From **Line 35** the script is devoted to the task of computing and associating scores with the individual relevant documents. The scores are here computed as the sum of the frequencies of all keywords in the respective documents. There are many other possibilities for computing scores, and before attaching to much weight to the scores the concept used should be fully understood.

A search result page is displayed in [Figure 8.5](#).

Your query condition: ((word = 'PHP'))

Documents retrieved

Document link	Document score
./documents/session5.htm	34
./documents/session6.htm	76
./documents/about.htm	10

Document score is the sum of the frequencies of all search words in each document

Figure 8.5: Search results

Administrative module

The most **common error** beginners commit when constructing a web application is to forget that the application needs to be **managed**, **maintained** and **updated**. The content of the database ir of the current application will from time to time for example need to be in **inspected**. The **list of stop word** will certainly need to be adjusted by adding new insignificant words, and removing meaningful words. Sometimes, a need for orderly **removing** the database with tables can also appear. An **administrative module** should be able to take care of these and other tasks needed for keeping the application running.

Some possible options are outlined in **menu.htm**:


```

1. <!-- menu.htm -->
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3. "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
7. <title></title>
8. </head>
9. <center>
10. <h3><font color="blue">Administrative menu for search engine.</font></h3>
11. <table>
12. <tr align="left" ><td><a href="inspect.php">Inspect database content</a></td></tr>
13. <tr align="left"><td><a href="stop_words.php">Display stopwords</a></td></tr>
14. <tr align="left"><td><a href="add.htm">Add stop word</a></td></tr>
15. <tr align="left"><td><a href="remove.htm">Remove stop word</a></td></tr>
16. <tr align="left"><td><a href="remove.php">Remove database and tables</a></td></tr>
17. </table>
18. </center>
19. <body>
20. </body>
21. </html>

```

The form itself is trivia ([Figure 8.6](#)). In this example, we shall limit ourselves to consider a single script: After initializing the database, it can be useful to study its content. The script **inspect.php** displays the content of the database by descriptors in alphabetical order.



Figure 8.6: Administrative menu for search engine

```

1. <!-- inspect.php -->
2. <?php
3. $link=mysql_connect("localhost",'root','maximus');
4. if (!$link) die("<center><h3><font color=red>Install MySQL. </font></h3></center>");
5. $db_selected= mysql_select_db("ir", $link);

6. if (!$db_selected) die("<center><h3><font color=red>Database ir does not exist.</font></h3></center>");

```



```

7. $t=mysql_query("SHOW TABLES FROM ir", $link);

8. if (!$t) die("<center><h3><font color=red>Table Descriptors does not exist.</font></h3></center>");

9. print("<center>");
10. print("<h3><font color=blue>Database content</font></h3>"); //
11. print("<table Border>"); //
12. print("<tr><th><b>Word</b></th><th><b>Document words</b></th><th><b>Descriptor
frequency</b></th> <th><b>Document reference</b></th></tr>");
13. $r=mysql_query("SELECT * FROM descriptors ORDER by word asc", $link) or die("Nothing SELECTED");

14. while ($row = mysql_fetch_array($r)) {
15. $word=$row[1];
16. $document_words=$row[2];
17. $descriptor_frequency=$row[3];
18. $document_ref=$row[4];
19. print("<tr><td>$word</td><td>$document_words</td>
<td>$descriptor_frequency</td><td>$document_ref</td> </tr>");
20. }
21. print("</table>");
22. print("</center>");
23. mysql_close($link);
24. ?>

```

In **Line 9** all document descriptors in alphabetic order and referred to by the handle **\$r**. Once again, we use **mysql_fetch_array()** to get access to the content of the columns in each individual row. Refer to the **SQL CREATE TABLE** in descriptors.php for the numbering of the columns with the first, **row[0]**, containing the **id**.

You will find illustrations of the all recorded descriptor list in [Figure 8.7](#), and the beginning of

Database content			
Word	Document words	Descriptor frequency	Document reference
about	223	2	./documents/about.htm
about	223	2	./documents/about.htm
action	5374	6	./documents/session5.htm
add	6874	10	./documents/session4.htm
add	5374	6	./documents/session5.htm
addition	223	1	./documents/about.htm
addition	223	1	./documents/about.htm
admin	223	1	./documents/about.htm
admin	223	1	./documents/about.htm
age	5374	17	./documents/session5.htm
agent	2633	16	./documents/session6.htm
alternatives	223	1	./documents/about.htm
alternatives	223	1	./documents/about.htm
application	5374	9	./documents/session5.htm
application	6678	26	./documents/session6.htm
applications	223	2	./documents/about.htm
applications	223	2	./documents/about.htm
applications	2633	3	./documents/session6.htm
applications	6678	7	./documents/session6.htm
approved	6678	14	./documents/session6.htm

Figure 8.7: Database content

A stop word list is illustrated in [Figure 8.8](#).

List of stop words	
0	all
1	already
2	also
3	and
4	another
5	are
6	available
7	blue
8	can
9	care
10	different
11	discuss
12	div
13	does
14	each
15	font
16	for
17	from
18	future

Figure 8.8: List of stop words

Session 9: e-learning

Web courses

In this session, implementation of web **courses** using **PHP** is discussed. As a student of this course, you should be particularly well armed with good ideas from your personal experience. It is impossible to go through a complete course in detail. The course you are attending contains for example more about **1000** files of different types organized in a structure with about **180** folders. In this session, we concentrate on discussing a few essential problems common for most Web courses.

As an application example and with reference to Session 8, a hypothetical web course on **Information Retrieval** is used. We assume that the following list can be used as a guide for our discussion:

- Course architecture
- Authorization and authentication
- Texts
- Illustrations
- Literature
- Evaluation

You find a link to the implementation of the example at the end of the session. You can either register yourself getting your own **PIN** code, or you can behave as already registered with e-mail "*dummy@dummy*" and PIN code "*0*".

Course architecture

Development of a web course, like any **IT** system, is an **art**. There are no absolute, proven rules for what is the right or the best approach. The more complex the objectives are, the more **elaborated** course structure will be required. In this example application, a folder with a flat organization of all needed files will be considered acceptable. All the files for the example are in a single folder (with the exception of a database located outside the directly accessible area and referred to as **irCourse**).

Security considerations are important only in connection with course design. We use the course application as a case for for using the functions **authorization** and **authentication** of users already discussed in Session7. Along the road, we shall also make comments to other forms of security. In [Figure 9.1](#), the overall organization for the example course is depicted. The figure indicates

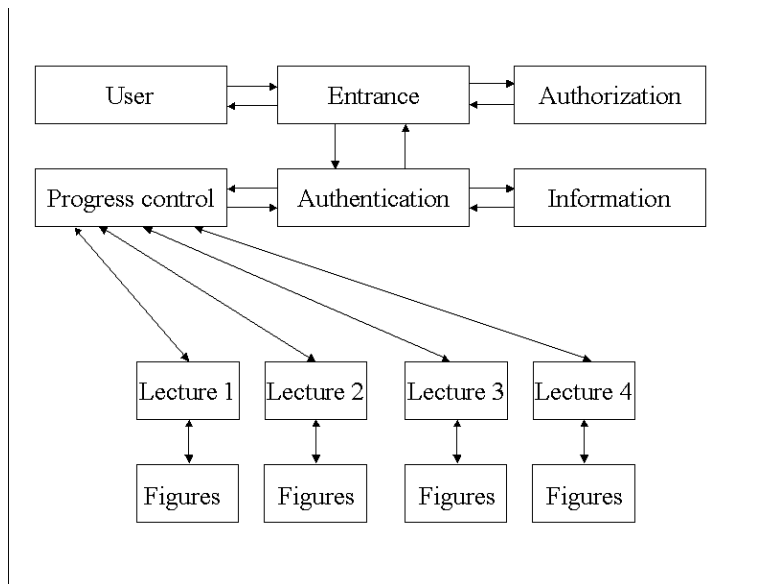


Figure 9.1: Course organization

that topics we are particularly interested in discussing are authorization, authentication and progress control.

Authorization and authentication

In this example we use the functions developed in **Session 7** and accessible in the library **mysql_functions.php**.

The **first** page we need is the **index.htm** which opens our example course scenario. It is a variation of the file we have used previously and starts by a **introductory text** to already admitted, and to new, applying students. If the caller is **new**, he/she is in **Line 4** asked to go on for **registration**, while already **registered** students can proceed to the **login** as specified in the form specified in **Lines 8-13**.

Consider the **login** alternative first. The login process requires that the student types his/her **username/e-mail address** and personal **PIN** code which she/he received when registered. The login alternative is recognized by sending the **hidden** variable with name **login** and value **"1"**.

The **index.htm** page is quite ordinary and looks like this:

```

1 <!-- index.cfm --->
2. <center>

```


3. <h1> Login</h1>

4. <p>

Thank you for your interest in this course. Access to the course is restricted to registered visitors only.<i>If you already are registered</i>, please go directly to the login.</p>

5.<p><i>If you are new and want to become a registered user</i>, we need some information from you, and you will need a personal identity number (PIN). Please continue with the registration</p>

6. <table><

7. <tr><td>Login with your</td></tr>

8. <FORM ACTION="validation.php" method="post">

9. <tr><td>Your username:</td> <td><input name="username" type="text" size="20"></td> </tr>

10. <tr><td>Your PIN code:</td><td> <INPUT TYPE="password" name ="submitted_pin" SIZE="20"></td></tr>

11. <input name="login" type="hidden" value="1">

12. <tr><td>Click the button:</td> <td><INPUT TYPE="SUBMIT" NAME="response" VALUE="Submit"></td></tr>

13. </FORM>

14. </table>

</center>

Note that this example applies a more **strict** authentication policy than that followed by **PHP with MySQL!**

Registration and authorization

If the student replies that he wants to **register**, the **registration.htm** script is called:

1. <!-- registration.cfm -->

2. <html>

3. <head>

4. <title>applications</title>

5. </head>

6. <center>

7. <h1>Registration</h1>

8. <p>In order to recognize and serve the different requirements of our visitors, each visitor needs her/his own username and PIN code.
 Please, complete and submit the form Your username and PIN code will be returned to you.</p>

9. <FORM ACTION="validation.php" method="post">

10. <table>

11. <tr><td>Your first name:</td> <td><input name="firstname" type="text" SIZE="20"></td> </tr>

12. <tr><td>Your last name:</td> <td><input name="lastname" type="text" SIZE="20"></td> </tr>

13. <tr><td>Your user name:</td><td> <INPUT TYPE="text" name ="username" SIZE="20"></td></tr>

14. <input name="registration" type="hidden" value="1">


```

15. <tr><td>Click the button:</td> <td><INPUT TYPE="SUBMIT" NAME="response"
VALUE="Submit"></td>></tr>
17. </table>
18. </FORM>
19. </center>
20. </body>
21. </html>

```

The **PIN** could either be **self-composed**, i.e. the person who request registration provides his/her own password, or it **system assigned**. Self-composed **PIN**s have the advantages that they may be easier for the owners to **remember**, and they can by special techniques (hashing) be kept secret also for the system staff. Compared with the system assigned **PIN**'s, the disadvantages of self-composed **PIN**s are they may be **easy to guess**, and they cannot easily serve as internal identifiers. In this example, the registration form does not offer self-composed **PIN**s indicating that we have chosen to use **system assigned** identifiers. The registration alternative is recognized by the attached **hidden** variable with name **registration** and value **"1"**.

Both **index.htm** and **registration.htm** call the script **validation.php** for processing. This script is a variation of the script **functioncalls.php** used for introducing the functions **authentication()** and **authorization()** in **Session 7**. The version we use in this example is shown below.

```

1. <!-- validation.php -->
2. <?php
3. //Open connection/database/table
4. $link=mysql_connect("localhost", "root","maximus");
5. $db="irCourse";
6. $db_selected=mysql_select_db($db, $link);
7. if(!$db_selected) {
8. mysql_query("CREATE DATABASE $db", $link);
9. mysql_select_db($db, $link);
10. mysql_query( "CREATE TABLE Users(firstname VARCHAR(20), lastname VARCHAR(20), email VARCHAR(20),
PIN VARCHAR(10))",
$link);
11. }
12. //Function calls
13. include "mysql_functions.php";
14. if (isset($_POST['login'])) {
15. $approved=authentication($db, $_POST['username'], $_POST['submitted_pin']);
16. if ($approved[0]=="yes") {
17. $_SESSION['PIN']=$_POST['submitted_pin'];
18. print("<h3><center><font color=blue>$approved[1], you are logged in.<br>
19. Please <a href=content.htm>continue</a></font></center></h3>");
20. }
21. else {
22. print("<p><center><font color=red>Your PIN code was invalid</font></center></p>");
23. }

```



```

24. }
25. if(isset($_POST['registration'])) {
26. $reg=authorization($db,$_POST['firstname'],$_POST['lastname'], $_POST['username']);
27. print("<center><font color=blue>You have been successfully authorized to access the site.<br>
28. Your username is: $reg[0], and your PIN is: $reg[1].<p></p>");
29. print("<a href=index.htm>Return </a>to Login.</center> ");
30. }
31. mysql_close($link);
32. ?>

```

The script starts by connection to **MySQL**, selecting database **irCourse** and checking the existence of the database in **Lines 4-11**. Our **library of functions** is included in **Line 13**. Remember to insert the **relative path** if the library is residing in another directory!

In **Line 14** and **Line 25**, the data received are identified either as **login** data or **registration** data, and the respective function is called. If the case is login and the authentication is successful, a message is returned to the student with a link to **content.htm** which can be activated. If the case is registration, a message with the username and the PIN code are returned including also a link to the login page, **index.htm**. Note that we define a new internal variable **\$_SESSION['PIN']** in **Line 17** for future use.

List of content

After a positive authentication page is processed, **content.php** is called. The first question is why this is a **php** script and not an **htm** page. The answer is that we want to record all visitors to this page by making use of the function **logging.php** in the library **mysql_functions.php**. Lines 2-6 take care of this task.

```

1. <!-- content.php -->
2. <?php
3. include("mysql_functions.php");
4. logging($_SESSION['PIN']);
5. ?>
6. <h2><font color="Blue">Information Retrieval Course</font></h2>
7. <h1><font color="Blue">Content:</font></h1>
8. <ol>
9. <li>session: <a href="text.htm">Introduction</a></li>
10. <li>session: <a href="text2.htm">Description and query language</a></li>
11. <li>session: <a href="text3.htm">Document indexing</a></li>
12. <li>session: File organization</li>
13. <li>session: Search operation</li>
14. <li>session: <a href="evaluation.htm">Evaluation</a></li>
15. <p></p>
16. <li><a href="literature.htm">References</a></li>
17. <li><a href="figures.htm">Figures</a></li>
18. </ol>

```


The remaining part of this page is rather trivial and requires no further comments. From the list of contents, there are links to the different parts of the course. .([Figure 9.2](#)).

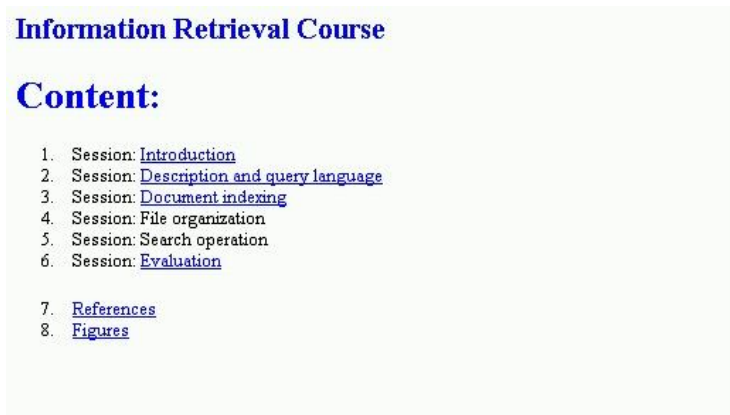


Figure 9.2: Content of an Information Retrieval Course

Sessions

As illustration, only a few components are implemented in this example and listed below. We use again the same trick as above and declare each completed session file as a **php** script, and continue to log visits to the sessions. In addition, we have a requirement that it should not be permitted to bypass the authentication for entry to the course.

To be able to satisfy the last requirement, we first define a new function:

```
1. function logged_in($PIN) {
2. if(!isset($PIN)) {
3. exit("<center><h1><font color=red>You can access this application only after logging in.</font></h1></center>
>");
4. }
5. }
```

We call this function with **logged_in(\$_SESSION['PIN'])**. Obviously, if the client has not went through the authorization procedure, the variable **\$_SESSION['PIN']** cannot be set and the system will provide a warning message and exit the application.

Session 1 can look like this:

```
1. <!--session1.php -->
2. <?php
3. include("mysql_functions.php");
4. logged_in($_SESSION['PIN']);
```


5. logging(\$_SESSION['PIN']);

6. ?>

7. <h1>A COURSE IN INFORMATION RETRIEVAL</h1>

8. <h2>Session 1:Introduction</h2>

9. <p>The topic information retrieval concerns the structure, analysis, implementation, search and dissemination of documents representing information. The purpose of an information retrieval system is to satisfy needs for information in a best possible way. </p>

10. <p>A typical modern information retrieval system is implemented in a host computer which can be accessed on internet from client computers. It is implemented with 2 sets of software, the client software and the server software.</p>

11. <p>The required client server is the basic software for working with the internet, while the server requires the general software to provide services on internet as well as specialized software for the information retrieval application. </p>

12. <p>The information retrieval application is build with a collection of documents as in an ordinary library or files as with a provider of electronic document representations as the core. To help the user to identify the documents in which he/she is interested, a set of files with meta data for the documents are developed and frequently organized in a database. In some applications, but far from all, even the electronic documents themselves can be included in the database.</p>

13. <p>To interact with the system, the user must use a query language which has been adjusted to the type of meta data in the database. The user must be able to describe the general properties of the unknown documents he/she wants to identify. On the other side, the retrieval system must be able to interpret the requests, communicate with the user for more details if necessary, and search in the system for the documents wanted. Figure 1 gives an overview of a retrieval system.</p>

14. <p>Depending on the users knowledge about the system, the components of the query language, the meta data for the documents included in the collection, and the composition of documents, the retrieval process may be more or less successful. To be able to compare one retrieval system application with a second, measures of performance are needed. For information retrieval, 2 measures, recall and precision, have been widely used.</p>

15. <p>If A is the subset of the documents which are relevant for a certain task expressed the query by Q, and B is the retrieved documents, the ratio $(A \text{ AND } B)/A$ is called the recall of the retrieval system for the query Q. The precision of the expressed Q for the same task is the ratio $(A \text{ AND } B)/B$. Since the evaluation of the recall in principle assumes that the set of relevant documents in the collection is known (if it is known, no retrieval problem exists), the set A has to be estimated. Precision, on the other hand, requires no knowledge outside the retrieved set B.</p>

16. <h3>Literature</h3>

17. <p>Return to the Content.</p>

Note that the links to other sessions, literature, figures, etc. are included as in a usual **HTML** tags.. [Figure 9.3](#) shows a part of the session.

A COURSE IN INFORMATION RETRIEVAL

Session 1: Introduction

The topic information retrieval concerns the structure, analysis, implementation, search and dissemination of documents representing information. The purpose of an information retrieval system is to satisfy needs for information in a best possible way.

A typical modern information retrieval system is implemented in a host computer which can be accessed on internet from client computers. It is implemented with 2 sets of software, the client software and the server software.

The required client server is the basic software for working with the internet, while the server requires the general software to provide services on internet as well as specialized software for the information retrieval application.

The information retrieval application is build with a collection of documents as in an ordinary library or files as with a provider of electronic document representations as the core. To help the user to identify the documents in which he/she is interested, a set of files with meta data for the documents are developed and frequently organized in a database. In some applications, but far from all, even the electronic documents themselves can be included in the database.

Figure 9.3: The start of Session 1

There are several opinions about how to control the **progress** of students through a course depending on the author's experience and beliefs. One hypothesis is that students should **not** progress too fast through the sessions. Times, at which each session is opened, are implemented for **Sessions 2-6** in this example. We shall return to how the variables `$_SESSION['opening_time_2']`, `$_SESSION['opening_time_3']` and `$_SESSION['opening_time_6']` are set at the end of this session.

Alternative hypotheses are that the learning from the current session should be **tested** before a student is permitted to advance to the next, or that deadlines and closing dates for the sessions constitute a positive learning pressure. A number of interesting hypotheses can be tested in connection with course progress regulation.

The time access control is expressed in **Lines 4-9** in the following **3 php** scripts.

```
1. <!-- session2.php -->
2. <?php
3. if (isset($_SESSION['opening_time_02'])) {
4. if ($time() < $_SESSION['opening_time_2']) {
5. print("<center><h3><font color=red>Session not yet open</font></h3><p>");
6. print("<u>Return to Contents.</p></center>");
7. exit();
8. }
9. }
10. include("mysql_functions.php");
11. logged_in($_SESSION['PIN']);
12. logging($_SESSION['PIN']);
13. ?>

14. <h1><font color="Blue">A COURSE IN INFORMATION RETRIEVAL</font></h1>
15. <h2><font color="Blue">Session 2: Description and query language</font></h2>
16. <p> No text uploaded. </p>
```


Line 3 test if an opening time has been set (for setting opening times, see below), and in **Lines 4-7**, a test is done whether the opening time has occurred or not.

The next 2 scripts follow the same procedure:

```
1. <!-- session3.php -->
2. <?php
3. if (isset($_SESSION['opening_time_02'])) {
4. if ($time() < $_SESSION['opening_time_3']) {
5. print("<center><h3><font color=red>Session not yet open</font></h3><p>");
6. print("<u>Return</u> to Contents</p> </center>");
7. exit();
8. }
9. }
10. include("mysql_functions.php");
11. logged_in($_SESSION['PIN']);
12. logging($_SESSION['PIN']);
13. ?>

14. <h1><font color="Blue">A COURSE IN INFORMATION RETRIEVAL</font></h1>
15. <h2><font color="Blue">Session 3: Document indexing</font></h2>
16.<p> No text uploaded. </p>
```

The next session text example is **Session 6**:

```
1. <!-- evaluation.php -->
2. <?php
3. if (isset($_SESSION['opening_time_02'])) {
4. if ($time() < $_SESSION['opening_time_6']) {
5. print("<center><h3><font color=red>Session not yet open</font></h3><p>");
6. print("<u>Return</u> to Contents.</p></center>");
7. exit();
8. }
9. }
10. include("mysql_functions.php");
11. logged_in($_SESSION['PIN']);
12. logging($_SESSION['PIN']);
13. ?>

14. <h1><font color="Blue">A COURSE IN INFORMATION RETRIEVAL</font></h1>

15. <h2><font color="Blue">Session 6:Evaluation</font></h2>

16. <p>A retrieval system can be evaluated bases on a number of criteria including its effectiveness to provide
a satisfactory output, operational, maintenance and capital costs. In this session, we focus on the effectiveness
of the system with particular reference to the 2 central concepts: recall and retrieval.</p>

17. <p><a href="figure2.jpg">Figure 2</a> presents the relations among the different document sets and the
```


2 measures. Evaluating a retrieval system with these measures requires an experiment which can be outlined by the following steps:

18.
19. Delimit the collection for the experiment
20. Define a set of retrieval queries representative for the use of the collection
21. Draw a random sample of the collection documents
22. Let experts decide how many documents in the sample are relevant for the different queries
23. Estimate the total number of items in the collection relevant for the different queries
24. Run the queries and let experts decide how many relevant items are found in each query
25. Compute recall and precision measures based on the estimated totals and the relevant documents from the queries
26.

27. <h3>Literature</h3>

28. Return to the content.

Instructor's tools

The course implementation is now as complete as planned with one exception. Still, the question about how to set the opening times has to be discussed. As pointed out in several occasions, in general an application is not complete without a tool box for the administrator. This should of course not be accessible for the users.

In create another directory, **tools**, in the examples directory in which we can enter a **menu.htm**, (Figure 9.4) with 2 links to **list_students.php** and **set_openings.htm**. The menu is so simple

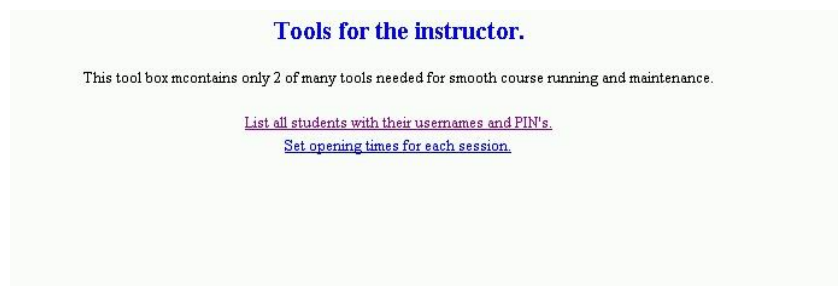


Figure 9.4: Tools for the instructor

that we don't have to waste time on the form. Instead we take a short time to look at the **list.php**

```
1.<?php
2.$link=mysql_connect("localhost", "root","maximus");
3.$db="irCourse";
4.$db_selected=mysql_select_db($db, $link);

5.if(!$db_selected) {
6. mysql_query("CREATE DATABASE $db", $link);
7. mysql_select_db($db, $link);
```



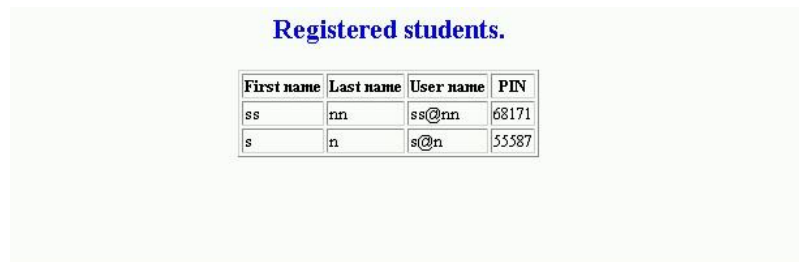
```

8. mysql_query( "CREATE TABLE Users(firstname VARCHAR(20), lastname VARCHAR(20), email
VARCHAR(20), PIN VARCHAR(10))", $link);
9. }

10. print("<center><h2><font color=blue>Registered students. </font></h2>");
11. $r= mysql_query("Select * FROM Users", $link);
12. print("<table border>");
13. print("<tr><th>First name</th><th>Last name</th><th>User name</th><th>PIN</th></tr>");
14. while ($row=mysql_fetch_array($r)) {
print("<tr><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td><td>$row[3]</td></tr>");
15. }
16. ?>

```

There is nothing new in this script, but note that the handle **\$r** provided by the **mysql_query()** in **Line 11** is used in the following **while** loop to extract each row as an array of column values for printing. (See [Figure 9.5](#)).



First name	Last name	User name	PIN
ss	nn	ss@nn	68171
s	n	s@n	55587

Figure 9.5: Registered students

The second tool we want for the tool case is a capability to set opening times for each session maybe spaced a week apart. We need a form page of the type:

```

1. <!-- set_time.htm -->
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3. "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <title>Untitled Document</title>
7. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
8. </head>

9. <center>
10. <h2><font color="blue">Opening time for course sessions.</font></h2>
11. <p>The opening of time for each session can be set by the following form. The time specified is the server
time. </p>
12. <table>
13. <form action="set_time.php" method="post">
14. <tr><td>Session:</td> <td><input name="session" type="text" value="00" size="4"

```



```

maxlength="2"></td></tr>
15. <tr><td>Hour:</td><td><input name="hour" type="text" value="00" size="4"
maxlength="2"></td></tr>
16. <tr><td>Minute:</td><td><input name="minute" type="text" value="00" size="4"
maxlength="2"></td></tr>
17. <tr><td>Second:</td><td><input name="second" type="text" value="00" size="4"
maxlength="2"></td></tr>
18. <tr><td>Month:</td><td><input name="month" type="text" value="00" size="4"
maxlength="2"></td></tr>
19. <tr><td>Day of month:</td><td><input name="day" type="text" value="00" size="4"
maxlength="2"></td></tr><tr>
20. </tr><td>Year:</td><td><input name="year" type="text" value="2006" size="4"
maxlength="4"></td></tr>
21. <tr><td>Submit:</td><td><input type="submit" value="Submit"></td></tr>
22. </form>
23. </table>
24. </center>

25. <body>
26. </body>
27. </html>

```

The form has pre-set width and values for each time component. If you want the system to make the session accessible at the beginning of each 24 hour day, the hour, minute and second can be left with default values. Type always "0" in a vacant space to the left of a single digit. ([Figure 9.6](#)).

Opening time for course sessions.

The opening of time for each session can be set by the following form. The time specified is the server time.

Session:	<input type="text" value="00"/>
Hour:	<input type="text" value="00"/>
Minute:	<input type="text" value="00"/>
Second:	<input type="text" value="00"/>
Month:	<input type="text" value="00"/>
Day of month:	<input type="text" value="00"/>
Year:	<input type="text" value="2006"/>
Submit:	<input type="submit" value="Submit"/>

Figure 9.6: Table for specifying when a session should open

The form calls `set_time.php` which introduces a few interesting time related function.

```

1. <!-- set_time.php -->
2. <?php
3. $ts=mktime($_POST['hour'],$_POST['minute'],$_POST['second'],$_POST['month'],
$_POST['day'],$_POST['year']);
4. $ct=time();
5. $s=$_POST['session'];

```



```

6. $_SESSION['opening_time_'].$s]=$ts;
7. print("<center>");
8. print("<h3>You have set the timestamp $ts for Session $s.</br>
The timestamp for when set was $ct. </h3>");
9. print("</center>");
10. ?>

```

The **mktime()** function in **Line 3** converts the 6 time-related form variables to a **timestamp**, i.e. a unique value corresponding to the number of seconds since the beginning of **1970**, to the specified point of time. Another function, **time()**, fetches the data from the clock of the server, and gives the timestamp for the point of time it was executed. These 2 function make it possible to set up a condition which is satisfied at the time specified in **mktime()** relative to the server time.

The opening times set in this example will only last for your session. Other students will not experience the opening times you have set, neither will you next time you enter the example. To make the opening times permanent, they must be saved in a file or a database, and retrieved at the beginning of the application.

Concluding remarks

We have in this session studied some challenges connected with implementation of a web course. The **dynamics** required are mainly associated with the authentication of students and the use of the course. There is several obvious **research tasks** associated with web courses. Web courses can be an **excellent choice** for implementing some topics and a **bad** for other. Which and why? Who are the students benefiting from a web course? Which are **efficient** authentication variables for a course? Is authentication really **necessary** and why?

Most courses have a number of structural attributes in common. It is possible to develop course generators, which permit the author to select his/her preferred structure and of course content. The **PHP 5 with MySQL** course was developed by means of a generator.

Session 10: Web shop

e-shops

One of the most popular and talked about web applications is **e-shops**, **e-business** or **e-commerce**. Complete commercial systems are available to buy from the shelf, new web shops have emerged and many have disappeared. Great expectations obviously exist for the future of web shops. These applications also demonstrate a number of web application aspects.

In this session we discuss and demonstrate some of the basic principles for a **web shop**. The example is a web shop, which are selling the **web scripts** we have introduced in this course. As all the other examples, our web shop application is not complete, and can be improved in many ways.

The **essential** scripts of the application are discussed below. Some trivial pages used in this example as **conditions.htm**, **shipping.htm**, **support.htm** and **about.htm** are illustrated in **figures**, but are not discussed below. It is **recommended** that you make yourself acquainted with the example before you start studying the scripts in detail.

Business promotion

Operating a web shop requires product **promotion**, i.e. dissemination of information about the products offered, prices, sales conditions, shipping, information about the company and its addresses. In addition to distribution of information by **huge lists** of e-mail addresses and advertisements, a web shop must have a **home page** with links to required information and provide the possibility to order/buy products online. In our example, **Software Shop** has a home page generated by the page **index.htm**. This homepage, [Figure 10.1](#), will serve as an introduction to this application.

Welcome to the Software Shop

The Software Shop has an exclusive suite of software for small companies. We have well satisfied customers and would be glad to see you among them. Please study out list of Product and if you fin any item of interest, click for more details and price. You can buy the product safely on the net and the merchandise will be shipped to you according to the alternative you prefer.

- [Products](#)
- [Sales conditions](#)
- [Shipping](#)
- [Support](#)
- [About Software Shop](#)

Figure 10.1: Main page for Software Shop

In a fancy commercial application, the home page script should probably contain **icons**, **flash** or **applet driven animation**, etc. The main focus in our example is, however, the **dynamics** aspects from a **PHP** point of view. On the introductory page, a menu with links provides a list of the services and information offered to the customers. The example concentrates on **products**, **orders** and **sales**. On the **Introductory page**, the links to **products.htm** and information scripts are on **Lines 5 - 9**:

```

1. <!-- index.htm --->
2. <h1><font color="Blue">Welcome to the Software Shop</font></h1>
3. <p>The Software Shop has an exclusive suite of software for small companies. We have well satisfied
   customers and would be glad to see you among them. Please study out list of Product and if you fin any item of
   interest, click for more details and price. You can buy the product safely on the net and the merchandise will be
   shipped to you according to the alternative you prefer.</p>
4. <ul>
5. <li><a href="products.htm">Products</a></li>
6. <li><a href="conditions.htm">Sales conditions</a></li>
7. <li><a href="shipping.htm">Shipping</a></li>
8. <li><a href="support.htm">Support</a></li>
9. <li><a href="about.htm">About Software Shop</a></li>
10. </ul>

```

This page is of minimal interest in the context of new dynamic web features.

The **product.htm** page generates a table with a row for each product. This page is also a straight forward demonstration of the **HTML** table tag features.

```

1. <!-- products.htm --->
2. <center>
3. <h1><font color="Blue">Products</font></h1>
4. <p>Our products cover dynamic web applications. They all require a web server and a ColdFusion
   application server installed on your host computer:</p>
5. <h3>Product list</h3>
6. <table border="2" cellpadding="10" cellspacing="10">
7. <tr><td><b>Product name</b></td><td><b>Demonstration</b></td><td><b>Price</b></td></tr>
8. <tr><td>Market research</td><td><a
   href="..../session3/examples/market_research">Demo</a></td><td>$ 150,00</td></tr>
9. <tr><td>Opinion polls</td><td><a href="..../session4/examples/opinion_polls">Demo</a></td><td>$
   150,00</td></tr>
10. <tr><td>Search engine</td><td><a href="..../session6/examples/search_engine">Demo</a></td><td>$
   200,00</td></tr>
11. <tr><td>Net course </td><td><a href="..../session7/examples/e-learning">Demo</a></td><td>$
   350,00</td></tr>
12. <tr><td>e-Shop</td><td><font color="Blue">The current application</font></td><td>$
   250,00</td></tr>
13. </table>

```

Each product is listed with the possibility to get an **online demo** of the product. The table also informs the customers about the **prices** and has a link to **ordering**. ([Figure 10.2](#)).

Products

Our products cover dynamic web applications. They all require a web server and a ColdFusion application server installed on your host computer.

Product list

Product name:	Demonstration:	Price:
Market research	Demo	\$ 150,00
Opinion polls	Demo	\$ 150,00
Search engine	Demo	\$ 200,00
Net course	Demo	\$ 350,00
e-Shop	The current application	\$ 250,00

Do you want to order? [Yes](#)/[No](#)

Figure 10.2: Products and prices

Buying products

Buying products is taken care of by the **form.htm** script which generates the form ([Figure 10.3](#))

Customer form

If you are a new customer, please give the information necessary to send the products you buy to you. Please answer all questions
 If you are a previous customer, please go to the last part of the form and submit your email address:

First name:

Last name:

Street address:

City:

State:

Country:

Zip no.:

e-mail:

OR

Have you previously bought products from us, please type your

e-mail address:

Figure 10.3: Customer form

by which necessary data about the customer are collected:

1. <!-- form.htm --->
2. <center>
3. <h1>Customer form</h1>
4. <p>If you are a new customer, please give the information necessary to send the products you buy to you.

Please answer all questions
 If you are a previous customer, please go to the last part of the form and submit your email address:</p>

```
5. <table >
6. <form action="customers.php" method="post">
7. <tr><td>First name: </td><td><input type="text" name="firstname"></td></tr>
8. <tr><td>Last name: </td><td><input type="text" name="lastname" ></td></tr>
9. <tr><td>Street address: </td><td><input type="text" name="street"></td></tr>
10. <tr><td>City: </td><td><input type="text" name="city" ></td></tr>
11. <tr><td>State: </td><td><input type="text" name="state" value=""></td></tr>
12. <tr><td>Country: </td><td><input type="text" name="country"></td></tr>
13. <tr><td>Zip no.: </td><td><input type="text" name="zip"></td></tr>
14. <tr><td>e-mail:</td><td><input type="text" name="submitted_email"></td></tr>
15. <input type="hidden" name="mark" value="0">
16. <tr><td><td><input type="submit" value="Submit"></td></tr>
17. </form>
18. </table>

19. <p><b>OR</b></p>

20. <p>Have you previously bought products from us, please type your</p>
21. <table cellpadding="10">
22. <form action="customers.php" method="post" >
23. <tr><td>e-mail address: </td><td><input type="text" name="submitted_email"></td></tr>
24. <input type="hidden" name="mark" value="1">
25. <tr><td></td><td><input type="submit" value="Submit"></td></tr>
26. </form>
27. </table>
28. </center>
```

The **FORM** tags for 'new' customers extend on **Lines 6-17** and collect **name** and **address**. A second form block on **Lines 22 - 26** for 'old' customers collects only the **e-mail** address. The application must be able to 'remember' the data provided, and to **check** the e-mail address and retrieve the information when a customer responds as an 'old' customer. To distinguish which of the two blocks is submitted, a 'hidden' variable **mark** is used with value **"0"** set in **Line 15** if the customer is a 'new', and set to **"1"** set in **Line 24** if he/she is an 'old'. Both blocks call on the script **customers.php** when submitted. ([Figure 10.4](#)).

We have the following data about you:

Customer id:	1
First name:	s
Last name:	n
Street address:	b
City:	p
State:	
Country:	n
Zip code:	5
E-mail:	s@n

Are the above data correct? [Yes/No](#)

Figure 10.4: Verifying customer information

Before we proceed to the **customers.php** script, we need a **database**, **shop**, and a **table**, **customers**. The table must have the columns/types:

- **firstname VARCHAR(30)**
- **lastname VARCHAR(20)**
- **street VARCHAR(20)**
- **city VARCHAR(20)**
- **state VARCHAR(20)**
- **country VARCHAR(20)**
- **zip VARCHAR(10)**
- **email VARCHAR(30)**

All variables are defined as text variables.

A second table, **sales**, will also be created. This table will be discussed in connection with the script **sale.php** below.

The **customers.php** script has **3** tasks.

- **update** the database table **customers** with the data about the customer if he is 'new' (**Line 15**),
- **retrieve** customer data from the database for display if he is an 'old' customer (**Line 35**)
- **display** recorded customer data for the client.

```

1. <!-- customer.php -->
2. <?php
3. $link=mysql_connect("localhost", "root","password");
4. $db="shop";
5. $db_selected=mysql_select_db($db, $link);
6. if(!$db_selected) {
7. mysql_query("CREATE DATABASE $db", $link);
8. mysql_select_db($db, $link);
9. mysql_query( "CREATE TABLE customers(id INT NOT NULL AUTO_INCREMENT, PRIMARY KEY(id),firstname
VARCHAR(30), lastname VARCHAR(20), street VARCHAR(20),city VARCHAR(20), state VARCHAR(20), country
VARCHAR(20),zip VARCHAR(10), email VARCHAR(30))", $link);

```



```

10. mysql_query("CREATE TABLE sales(customer_id VARCHAR(10),total VARCHAR(10),ddate VARCHAR(10)",
$link);
11. }

12. if ($_POST['mark'] == "0") {
13. mysql_query("INSERT INTO customers(firstname,lastname,street,city,state,country, zip, email) VALUES
('$ _POST[firstname]','$ _POST[lastname]', '$ _POST[street]',
'$ _POST[city]','$ _POST[state]','$ _POST[country]','$ _POST[zip]', '$ _POST[submitted_email]'", $link);br> 14.
print("<center><h3><font color=blue>You are recorded with the following data:</font></h3>");
15. print("<table>");
16. $r=mysql_query("SELECT * FROM customers WHERE email='$ _POST[submitted_email]'", $link) or die("Your
record cannot be found.");

17. while ($row = mysql_fetch_array($r)) {
18. $ _SESSION['id']=$row[0];
19. print("<tr><td>Customer id:</td><td>$ _SESSION[id]</td></tr>");
20. print("<tr><td>First name:</td> <td>$ _POST[firstname]</td></tr>");
21. print("<tr><td>Last name:</td> <td>$ _POST[lastname]</td></tr>");
22. print("<tr><td>Street address:</td> <td>$ _POST[street]</td></tr>");
23. print("<tr><td>City:</td> <td>$ _POST[city]</td></tr>");
24. print("<tr><td>State:</td> <td>$ _POST[state]</td></tr>");
25. print("<tr><td>Country:</td> <td>$ _POST[country]</td></tr>");
26. print("<tr><td>Zip code:</td> <td>$ _POST[zip]</td></tr>");
27. print("<tr><td>E-mail:</td> <td>$ _POST[submitted_email]</td></tr>");
28. print("</table>");
29. }
30. print("<p>Are the above data correct? <a href=order.htm>Yes</a></a>
href=form.htm>No</a></p>");
31. print("</center>");
32. }
33. else {
34. print("<center>");
35. $r=mysql_query("SELECT * FROM customers WHERE email='$ _POST[submitted_email]'", $link) or die("Your
email cannot be found.");
36. }
37. print("<center><h3><font color=blue>We have the following data about you:
38. </font></h3>");
39. print("<table>");
40. while ($row = mysql_fetch_array($r)) {
41. $ _SESSION['id']=$row[0];
42. print("<tr><td>Customer id:</td> <td>$ _SESSION[id]</td></tr>");
43. print("<tr><td>First name:</td> <td>$row[1]</td></tr>");
44. print("<tr><td>Last name:</td> <td>$row[2]</td></tr>");
45. print("<tr><td>Street address:</td> <td>$row[3]</td></tr>");
46. print("<tr><td>City:</td> <td>$row[4]</td></tr>");

```



```

47. print("<tr><td>State:</td> <td>$row[5]</td></tr>");
48. print("<tr><td>Country:</td> <td>$row[6]</td></tr>");
49. print("<tr><td>Zip code:</td> <td>$row[7]</td></tr>");
50. print("<tr><td>E-mail:</td> <td>$row[8]</td></tr>");
51. }
52. print("</table>");

53. print("<p>Are the above data correct? <a href=order.htm>Yes</a>/<a
href=form.htm>No</a></p>");
54. print("<center>");
55. }
56. mysql_close($link);
57. ?>

```

Line 12 checks if the data received concern a **new** customer. If so, a `mysql_query()` with an **SQL INSERT INTO table customers**, inserts the submitted data. The 'old' customers are taken care of by **Lines 33 - 53**. Usually only one row is retrieved, but if the customer has submitted data as 'new' customer twice or more, there can be more than one record. ([Figure 10.4](#)). Finally, if the customer **accepts** the displayed data, the control is transferred to script **order.htm**. Note the **Lines 16-18** and **Line 41** for forming the variable `$_SESSION['id']`.

Order

Please, mark the items you want to buy:

- ☒ Market research - \$150.00
- ☐ Opinion poll - \$150.00
- ☒ Search engine - \$200.00
- ☐ Net course - \$350.00
- ☒ e-Shop - \$250

Figure 10.5: Order form

The next step is **order.htm**.. The page contains a set of multiple inputs ([Figure 10.5](#)). The values of the checked items are saved in a shopping array named **order**:

1. `<!-- order.htm -->`
2. `<center>`
3. `<h1>Order</h1>`
4. `<p>Please, mark the items you want to buy:</p>`


```

5. <form action="sum.php" method="post" >
6. <table>
7. <tr><td><input type="checkbox" name="order[Market research]" value="150">Market research -
$150.00</td></tr>
8. <tr><td><input type="checkbox" name="order[Opinion poll]" value="150">Opinion poll - $150.00</td></tr>
9. <tr><td><input type="checkbox" name="order[Search engine]" value="200">Search engine -
$200.00</td></tr>
10. <tr><td><input type="checkbox" name="order[Net course]" value="350">Net course - $350.00</td></tr>
11. <tr><td><input type="checkbox" name="order[e-Shop]" value="250">e-Shop - $250</td></tr>
12. <p></p>
13. <tr><td><input type="submit" value="Order"></td></tr>
14. </table>
15. </form>
16. </center>

```

ORDER RECEIVED

Date: 18.10.2006

Thank you for ordering the following items:

Market research	150
Search engine	200
e-Shop	250
Total	600

Please return your creditcard information:

Card type:

☐ VISA
☐ MASTERCARD
☐ AMERICAN EXPRESS

Card number:

Expire date:

Figure 10.6: Payment form

When the above form is submitted with **1** or more **selected** products, the control is left to the **sum.php** script. Here are several **new PHP** features introduced. In **Line 4** the built-in function **getdate()** is used. It returns an **associative** array with the indices **seconds**, **minutes**, **hours**, **mday**, **mon**, **year**, **weekday**, and **month**. In **sum.php** we make use of a few of these elements. Note that we here have an example of a **mixed PHP** and **HTML** script.

```

1. <!-- sum.php -->
2. <?php
3. $total=0;
4. $day=getdate();
5. print("<center><h2><font color=blue>ORDER RECEIVED</h2>");
6. print("<p>Date: $day[mday].$day[mon].$day[year] </p>");
7. print("<p>Thank you for ordering the following items:</p> ");

```



```

8. print("<table BORDER='>");
9. foreach($_POST['order'] as $key => $value) {
10. print("<tr><td>$key</td> <td>$value</td>");
11. $total=$total + $value;
12. }
13. $_SESSION['total']=$total;
14. print("<tr><td> Total</td> <td><b>$total</b></td></tr>");
15. print("</table>");
16. ?>

17. <p>Please return your creditcard information:</br>
18. <form action="sale.php" method="post">
19. Card type:
20. <table>
21. <tr><td><input type="radio" name="card_type" value="VISA">VISA</td></tr>
22. <tr><td><input type="radio" name="card_type" value="Mastercard">MASTERCARD</td></tr>
23. <tr><td><input type="radio" name="card_type" value="Ammerican Express">AMERICAN
EXPRESS</td></tr>
24. </table>
25. <p></p>Card number:<input type="text" validate="creditcard" name="card_no"><br>
Expire date:&nbsp;&nbsp; <input type="text" name="expire_date">
26. <p><input type="submit" value="Submit order"></p>

27. </form>
28. </center>

```

Purchasing products

- id
- total
- ddate

Submitting the shopping list and the credit card information establish a **purchase**. The task of the script **sales.php** is to process the purchase. The **date** of the purchase is determined in **Line 12** and the transaction including customer identification, total value of the transaction and the date is booked are stored in table sales by means of the **sql INSERT INTO** in **Line 15**.


```

1. <!-- sale.php -->
2. <?php
3. $link=mysql_connect("localhost", "root","password");
4. $db="shop";
5. $db_selected=mysql_select_db($db, $link);
6. if(!$db_selected) {
7. mysql_query("CREATE DATABASE $db", $link);
8. mysql_select_db($db, $link);
9. mysql_query( "CREATE TABLE customers(id INT NOT NULL AUTO_INCREMENT, PRIMARY KEY(id),firstname
VARCHAR(30), lastname VARCHAR(20), street VARCHAR(20),city VARCHAR(20), state VARCHAR(20), country
VARCHAR(20),zip VARCHAR(10), email VARCHAR(30))", $link);
10. mysql_query("CREATE TABLE sales(id VARCHAR(10),total VARCHAR(10),ddate VARCHAR(10)", $link);
11. }

12. $day=getdate();
13. print("<center><h2><font color=blue>ORDER RECEIVED</h2>");
14. print("<p>Date: $day[mday].$day[mon].$day[year] </p>");
15. mysql_query("INSERT INTO sales(id,total,ddate) VALUES('$_SESSION[id]','$_SESSION[total]', '$day')",$link);
16. print("<center><h3>Sales for customer no. $_SESSION[id] has been processed.</h3>");
17. print("<p>Print out a <a href=receipt.php>receipt</a>.</p></center>");
18. mysql_close($link);
19. ?>

```

sales.php returns a message that the transaction has been processed, and offers a printout of a receipt ([Figure 10.7](#)):

```

1. <!-- receipt.php -->
2. <?php
3. $link=mysql_connect("localhost", "root","password");
4. $db="shop";
5. $db_selected=mysql_select_db($db, $link);
6. if(!$db_selected) {
7. print("<center><h2><font color=red>The database does not exist.</font></h2>
8. </center>");
9. exit();
10. }

11. print("<h2>RECEIPT</h2>");br> 12. $r=mysql_query("SELECT firstname,lastname, street, city,state,country,
zip FROM customers WHERE id = '$_SESSION[id]'", $link);
13. while ($row = mysql_fetch_array($r)) {
14. print("<b>$row[0] $row[1]<br>");
15. print("$row[2]<br>");
16. print("$row[3] $row[6]<br>");
17. print("$row[4]<br>");
18. print("$row[5]<br></b>");
19. }

```



```

20. $day=getdate();
21. $r=mysql_query("SELECT total FROM sales WHERE id = '$_SESSION[id]'", $link);
22. mysql_query("INSERT INTO sales(id,total,ddate) VALUES('$_SESSION[id]','$_SESSION[total]', '$day')",$link);
23. print("<p></p>");
24. print("We have charged your account with <b>$ $_SESSION[total]</b> for products sent to you.<br>");
25. print(" Thank you for ordering our products.<br>");
26. print("<p></p>");
27. print("<i>Software Shop</i>");
28. print("<p><a href=index.htm>Return</a> to product page.</p>");
29. mysql_close($link);
30. ?>

```



Figure 10.7: Confirmed order

The output **receipt** is illustrated in [Figure 10.8](#).

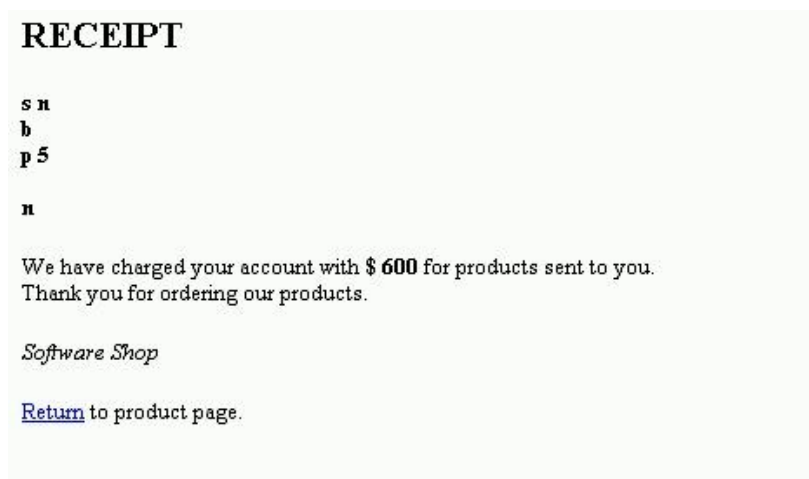


Figure 10.8: Receipt

A final remark

You have arrived to the end of this course. We hope you have enjoyed our discussions and examples, and become enthusiastic about the possibilities of dynamic web applications.

This course assumes the use of the traditional approach to **PHP** and **MySQL**. **PHP 5** offers also the use of object oriented programming and **MySQL** has an object oriented interface, **mysqli**, available.

If you are planning to improve your skills with **PHP** and **MySQL**, we strongly recommend visits to the **PHP** and **MySQL** home pages.

A bibliography for further studies

Bandyopadhyay, N (2001): e-Commerce: Context, Concepts and Consequences 1/e. McGrawHill. NY.

Brook-Bilson, R. (2003): Programming ColdFusion. Second Edition. O'Reilly. Ca.

Castro, E. (1999): PERL and CGI. Peachpit Press. CA.

Comer, D.E. (2005): The Internet Book. Prentice Hall. NJ.

Flanagan, D. (2001): JavaScript: The Definitive Guide, 4th Edition. O'Riley, Ca.

Hatfield, B. (1999): Active Server Pages for Dummies. IDG. CA.

McFarland, D. S. (2004): Dreamweaver MX 2004: The Missing >Manual. O'Riley, Ca.

Mohammed, R., Fisher, R., Javorski, B., and Cahill, A. (2001): Internet Marketing: Building Advantage in the networked Economy 1/e. McGrawHill. NY.

Moock, C. (2002): ActionScript for Flash MXC. The Definitive Guide. 2nd Edition. O'Reilly. Ca.

Reding, E. (2001): Building an E-Business: From the Ground Up 1/e. McGrawHill. NY.

Sebesta, R.W. (2005): Programming the World Wide Web. Pearson-Addison Wesley. NJ.

Wall, L., Christiansen, T. and Schwartz, R. (1996): Programming Perl. O'Reilly. Ca.