

# Creating academic database-backed web sites with HTML, PHP and MySQL

Revision: 18 February 2003

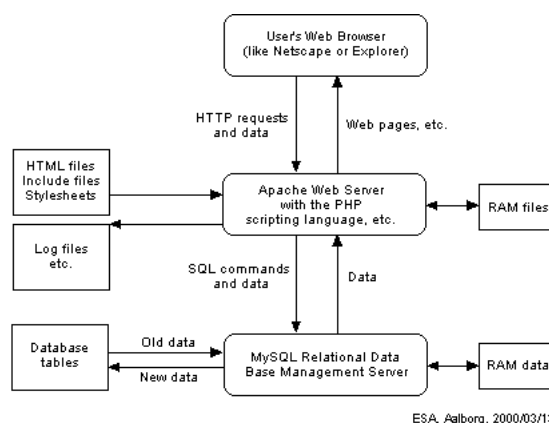
**Esben Sloth Andersen**

DRUID/IKE, Department of Business Studies, Aalborg University

## 1. Overview

In the early days of the World Wide Web, information was stored in simple files with only HTML markup. Today, the web server often processes scripts within web files that e.g. call other files, stylesheets, etc. Furthermore, the scripts may also acquire information from a relational database management system. Such a modern system gets data from database tables and includes them in the web page that is delivered to the user.

To make database-backed system of web pages like [Evolutionary Economics et al.](#), we need a system of tools. Thus we leave the user's viewpoint that there is just a web server that is capable of sending web pages. Instead we have to think in terms of a system consisting of a web server system (like Apache) as an entity that from the web files can interpret a programming language (like PHP or Perl) and thus can access a database management system (like MySQL). Such a system may look rather complicated - like it is sketched in the following figure:



Given this system or a similar software system, a single HTML file with scripts (e.g. in PHP) can generate a huge number of more or less different web pages that are sent to the user of the website. There are several ways of obtaining this multiformity:

1. Much of the contents of the web page displayed in the user's browser are taken from a database. The maintainer (or the secretary) create and modify data in the database rather than in the file with web-oriented code.
2. In the web page displayed to the user there may be links that call the same file that was originally used to produce the page. This call, however, may be done with different values of variables. These values may determine the behaviour of the scripts that influence the production of the page to the user.
3. In the web page for the user there may be input areas into which the user may enter a request. This request may set values of variables that influence the production of the web page for the user.
4. The user may use input forms to send information that might automatically be included in the database. This sent information may determine the next display of the web page. Since this solution raises practical problems as well as security problems, it is not discussed in the present report. Instead it is described how an input area can be used to invoke the sending of

an email to the maintainer of the system. If the maintainer accepts the suggestion, it is placed in the database and will influence future displays of the web page.

To allow the development and use of such a modern type of web pages, the web server needs the relevant software. The present report describes the widespread LAMP system = Linux + Apache + MySQL + PHP, and this system also exists in a WAMP version (where W = Windows). However, Windows-based servers often use other systems - like Microsoft's Active Server Pages (ASP). Furthermore, many web servers have none of these facilities. So before you consider developing a modern, database-backed web site, you should check these issues with your system administrator.

Given that an adequate software system is related to the web server, the web site developer has a radically increased set of possibilities. A first glimpse of these possibilities may be obtained by inspecting one of the files that are used to produce a web page on such a system. This file may be radically different from the HTML code that the receiver of the web page sees by choosing "Show page source" in the web browser (like Netscape or Explorer). The server-side file will probably have some ordinary text and HTML markup, but it will also have smaller or larger amounts of commands written in a programming ("scripting") language. In the simplest case the file (here: testevolinks.html) may look like:

```
<HTML>
<TITLE>Test page</TITLE>
<BODY>
<?php
include("header.inc");
include("testevolinks.inc");
include("footer.inc");
?>
</BODY>
</HTML>
```

This file has no independent contents. Instead the contents of the web page sent to the user will be fetched by three so-called include files (with extension: ".inc"). The first and the last of them defines the standardised header and footer of the page, while the file "testevolinks.inc" contains the information for the web page. To obtain the contents of these include files a PHP script is invoked. This script is surrounded by "<? ... ?>", and these strange parenthesis tells the (Apache) server program when to start and stop interpreting the file as PHP commands. All three PHP commands have the structure "include("filename");".

PHP scripts may contain much more interesting commands. Here are a few that will be used in this report:

```
include("filename"); - Insert the contents of the file "filename" into the produced web
page
$variablename = "value"; - Assign a value (e.g. "person") to a variable (e.g. $choice)
echo $variablename; - Write out the value of the variable
echo "Text"; - Write out a piece of text
mysql_pconnect("localhost","username","password"); - Open the MySQL system
mysql_select_db("databasename"); - Specify the database to connect to
$result = mysql_query("SELECT * FROM esalinks"); - Link the variable $result to
all the the contents of the entries (rows) in a database table called esalinks
$myrow = mysql_fetch_array($result); - Select the next row in the table defined by
the link variable $result
($variablename == "testvalue") - Test whether the variable has the value "testvalue".
If so, the result is true (1). Otherwise, it is false (0)
if (testcondition) {trueaction;} else {falseaction;}; - This conditional expression first
tests whether the testcondition is true. If so, the trueaction is performed.
Otherwise, the falseaction is performed
do {doaction;} while (testcondition); - This loop continues to perform the doaction
until it is found that the testcondition has become false
```

There is a very large set of PHP commands, but the above listed examples will cover nearly all that is needed for the present report. However, before we start using the commands, we have to create a database.

## 2. Creating and maintaining a database

In the LAMP software system all dynamic information is stored in tables in a MySQL database. As an example we shall use the table "evolinks" that underlies the web site of links to [Evolutionary Economics et al.](#) You may want to [check the highly simplified test page](#) (see appendix) before considering the structure of the underlying database table ([the test page is also shown in the appendix](#)).

In the test page you see three sections:

1. Search facilities: select the information you want to display
2. Search results: a display of the search results; this section also includes messages to the user
3. Send a suggestion: you may enter suggestions that are sent to the maintainer of the database

Thus the whole test page is centred around the database or, rather, around a database table that is called "evolinks". The evolinks table is created in MySQL through a web-based tool called [phpMyAdmin](#). The evolinks table must, at least, have the following columns:

1. ID: A automatically incremented identification number of type TINYINT
2. Title: The title/main description of the web page of type TEXT
3. URL: The address of the web page of type TINYTEXT
4. Information: Further information on the web page of type TEXT
5. Type: The type of the web page may be chosen from the following SET of types: 'book', 'conference', 'journal', 'paper', 'person', 'organisation', 'simulation', 'site', 'teach', 'other'
6. Topic: The topic of the web page may be chosen from the following SET of topics: 'biological', 'complextheoretical', 'computational', 'cultural', 'epistemological', 'gametheoretical', 'general', 'industrial', 'institutional', 'macroevolutionary', 'organisational', 'phylogenetical', 'realhistorical', 'thoughthistorical', 'other'
7. Revision: The date of the last revision is of type DATE in the format '2003-02-10'

These elements of the table are easily created in phpMyAdmin. After this is done, phpMyAdmin can be used for the insertion of records (rows) into the table. Choose Insert. Then write the text or select from the sets that were defined above. An example is:

1. ID: [inserted automatically]
2. Title: Metcalfe, Stan
3. URL: [http://les.man.ac.uk/cric/J\\_Stan\\_Metcalfe/](http://les.man.ac.uk/cric/J_Stan_Metcalfe/)
4. Information: CRIC and University of Manchester, UK. Important contributor to both evolutionary economic theory and its applications
5. Type: person
6. Topic: general,industrial,macroevolutionary
7. Revision: 2003-02-10

After inserting this information, check that everything is OK, and press "Save". You now have a database table (called evolinks) with a single record. This table has the following structure:

ID	Title	URL	Information	Type	Topic	Revision
1	Metcalfe ...	<a href="http://...">http://...</a>	CRIC ...	person	general,...	2003-...
2						

If you want to modify a given record, you first have to find it. You can click "Browse", but there may be many records. So it is often better to click "Select". Then you get a search form. Write part of the

field "Title" and/or part of the field "Information". You have to use the wildcard character % before and after the searched word like %metcalfe%. When you have found the record, click "Edit", correct the relevant field, and click "Save".

The database administration system is not foolproof. Especially, you should notice that names and different abbreviations should be written precisely - since they are used for searching for the items. The system does not support correctly special characters. Get a table of special HTML characters (e.g. Å should be written &Aring;), etc.

### 3. Creating dynamic elements in web pages

After the database includes some records, you can access this information by commands in your web pages. You fetch information from the database table, you need a little knowledge about the SQL language (Structured Query Language) – the widespread standard for relational databases. The basic command is SELECT. This command has several options. For instance <SELECT \* FROM filename> means that we select all the columns from the table called "filename". If we add <WHERE Topic LIKE "%industrial%">, we only obtain the rows (records) in which there occurs the text "industrial" in the column called "Topic".

Such commands are included in the PHP scripts. For instance, you may want to write out all records that have the type 'person'. To do so, you put into your dynamic web page a few PHP commands. Basically, you have to ask the web server system (Apache) to open the MySQL database and store the requested records in a variable. This is done in the following way:

```
<?php
mysql_pconnect("localhost","username","password");
mysql_select_db("databasename");
$result =
mysql_query("SELECT * FROM evolinks WHERE Type LIKE \"%person%\"");
?>
```

In the last command we see that the quotation marks in the SQL command occur within the quotation marks in the PHP command. Therefore, we have to use the backslash (\) before each of the inner quotation marks. When we have run the PHP command, the selected records can be accessed through the variable called \$result. To write out nicely the information, you may write a PHP loop that runs through the selected records and prints out the information mixed with HTML code:

```
<?php
if ($myrow = mysql_fetch_array($result))
{
echo "<OL>\n";
do
{
echo "<LI>";
echo "<A HREF=\"", $myrow["URL"], "\">", $myrow["Title"];
echo "</A> - ", $myrow["Information"], "<BR>";
} while ($myrow = mysql_fetch_array($result));
echo "</OL>";
};
?>
```

This script starts by fetching a row from the database table. If this row is empty, nothing happens. Otherwise a do-loop is entered. Here the rows are processed one by one. The elements of each row is printed out with added HTML markup. The result of running the loop will be a web page in ordinary HTML format like:

```
<OL>
<LI><A HREF="http://les.man.ac.uk/cric/J_Stan_Metcalfe/">
Metcalfe, Stan</A>
- CRIC and University of Manchester, UK. Important contributor
to both evolutionary theory and its applications
</OL>
```

When this HTML code is delivered to the user's browser, the output will look like:

1. [Metcalf, Stan](#) - CRIC and University of Manchester, UK. Important contributor to both evolutionary economic theory and its applications

## 4. Creating menus

A database can be used in a great many ways, and these options should be offered to the users of database-backed web sites. Consider the following partial URLs for the web page:

```
http://.../testevolinks.html
http://.../testevolinks.html?choice=person
```

The first URL is quite normal. The second has added a question mark followed by the specification of the value of a variable ("choice").

The specific values of variables in the URLs may look ugly to the user, but this method of supplying values has one great advantage: The user may copy the full URL and send it to other people. When they check the URL, they will see exactly the same information as the sender of the URL.

Let us consider the variable "choice", whose value is "person". This value is stored under the symbolic name \$choice. So we can write a PHP script that takes any value of the variable by just writing \$choice. This feature can be included in a call to MySQL. However, to do so we need double quotes within double quotes, and here we use the escape character "\". Thus the formulation is "\"\$choice\"", and the call is:

```
$result =
mysql_query("SELECT * FROM evolinks WHERE Type LIKE \"%$choice%\"");
```

The only difference between this method of access to the database and the one described above is that we call with a variable that is set by the user. Thus a given web page can have many specific forms. If the choice were not "person" but "paper", the page would display the list of papers that are available in the database. To obtain such a menu, we may write:

```
Choose an option:
<A HREF="testevolinks.html?choice=paper">Find all papers</A>,
<A HREF="testevolinks.html?choice=person">Find all persons</A>
```

This HTML code displays as:

Choose an option: [Find all papers](#), [Find all persons](#).

What happens when the user clicks one of the options? The result is simply that the web server is called for a new version of the same page as is already displayed. The only difference is that the value of the choice variable is changed. Thus the PHP call to the MySQL server becomes different, and thus the display of the page to the user will also be different.

This type of menu is often sufficient for academic sites, but it can easily be made more fancy. In the present report we shall only consider the standard facilities that are built into HTML. The basic tool is an HTML-defined "form", which allows for several ways for users to define and submit information. Here we shall just consider a pull-down menu. It may be defined in the following way:

```
<FORM METHOD="get" ACTION="testevolinks.html">
Select which links to display:
<SELECT name=choice>
<OPTION selected value=all>All links
<OPTION value=papers>Papers
<OPTION value=persons>Persons
</SELECT>
<INPUT TYPE="submit" VALUE="Search!">
```

</FORM>

This HTML code gives the following display:

Select which links to display:

Before you use pull-down menus extensively, you should consider whether they really help the user. The author of the present report often considers them a burden rather than helpful. So he normally applies simple text menus - like described above.

## 5. Allowing for flexible user input

The menu-driven user interface may be augmented by features allowing users to specify requests and information. For instance, the user may be interested in finding the database entries about a particular person or a particular concept. In this case it is impractical to list all persons or all search words. Instead the user supplies this information. This feature can be implemented in the following way.

We start by creating an HTML form that sends the result to the database web page. This form may look like:

```
<FORM METHOD="get" ACTION="testevolinks.html">
<INPUT TYPE=hidden NAME=choice VALUE=request>
Find a word in the database (like: Metcalfe)
<INPUT TYPE="text" SIZE="20" NAME="search" VALUE="">
<INPUT TYPE="submit" VALUE="Search!">
</FORM>
```

The search form will look like:

Find a word in the database (like: Metcalfe)

The result of entering a query into the form and submitting it is that the database web page is recalled with the added information that the variable \$search has the query value (e.g. Metcalfe). To obtain the correct answer from the database web page, we need it to include code like:

```
if ($choice == "request")
{$result = mysql_query("SELECT * FROM evolinks
WHERE Title LIKE \"%$search%\"");}
```

In this code we use the variable \$search in the query to the evolinks database, like we saw earlier.

HTML forms may also be used for sending user comments and suggestions to the maintainer of the database. The easiest solution is to let the HTML form send an email. This may be done by the following code:

```
<FORM METHOD="post"
ACTION="testevolinks.html?choice=reply">
<B>Send a suggestion</B><P>
Your name: <INPUT TYPE="text" NAME="author" SIZE="30"><BR>
Your suggestion:
<TEXTAREA NAME="suggestion" COLS="60"
WRAP="VIRTUAL" ROWS="2"></TEXTAREA><BR>
<INPUT TYPE="submit" NAME="submit" VALUE="Send!">
</FORM>
```

This HTML form will look like:

**Send a suggestion**

Your name:  Email:

Your suggestion:

If you press submit, no email is sent from this test site. You only see the answer (in section 2). In actual life, the result of submitting the real form depends on the related PHP code. To send an email to the database maintainer, we may simply write something like:

```
<?php
if ($submit)
{mail("maintaineremail", "Suggestion by $author",
"$author\n$email\n$suggestion");};
?>
```

Given that the mail function is working on your web server, the result of invoking this PHP script by pressing "Send!" will be an email to the database maintainer with the following elements:

```
To: maintaineremail
Title: Suggestion by $author
Contents:
$author
$email
$suggestion
```

We finish by thanking the contributor. This happens by resending the web page with the variable \$choice = reply. Given this value, the only output (except menus and the suggestions form) is determined by the following PHP script:

```
<?php
if ($choice == "reply")
{echo "<B>Thanks for your suggestion</B><P>";};
?>
```

The resulting output is:

**Thanks for your suggestion**

## 6. Creating an integrated web page

This report has presented several features in relation to database-backed web sites, so it may be relevant to consider how they are put together into an integrated file. To see how this is done, you may once more want to [check out the simplified test page \(the test page is also shown in the appendix\)](#).

The test page "testevolinks.html" is delivered to the user's browser by the Apache web server. But the web server sees the file that is underlying this page in quite a different way than the user (even if he or she checks out the source code of the web page). Here is, more or less, what is seen by the Apache server:

```
<HTML><HEAD><TITLE>Test page</TITLE></HEAD>
<BODY>

<H2>Test the methods for accessing the
database table "evolinks"</H2>

<H3>1. Search facilities</H3>
<A HREF="testevolinks.html?choice=all">Find all links</A><P>
<A HREF="testevolinks.html?choice=paper">Find all papers</A><P>
<A HREF="testevolinks.html?choice=person">Find all persons</A><P>
<FORM METHOD="get" ACTION="testevolinks.html">
<INPUT TYPE=hidden NAME=choice VALUE=request>
Find a word in the database (like: Metcalfe)
<INPUT TYPE="text" SIZE="20" NAME="search" VALUE="">
<INPUT TYPE="submit" VALUE="Search!">
</FORM>

<H3>2. Search results</H3>
<?php
```

```

mysql_pconnect("localhost","username","password");
mysql_select_db("databasename");
if ($choice == NULL)
{echo "<B>Please choose one of the above options!</B>";};
if ($choice == reply)
{echo "<B>Thanks for your suggestion</B>";};
if ($choice == all)
{echo "<B>This is a full listing of the link database</B>";
$result = mysql_query("SELECT * FROM esalinks");};
if ($choice == request)
{echo "<B>You have requested links with the text ",$search,"</B>";
$result =
mysql_query("SELECT * FROM esalinks WHERE Title LIKE \"%$search%\");};
if ($choice == person || $choice == paper)
{echo "<B>This is a listing of ",$choice,"s</B>";
$result =
mysql_query("SELECT * FROM esalinks WHERE Type LIKE \"%$choice%\");};

if ($myrow = mysql_fetch_array($result))
{
echo "<OL>";
do
{
echo "<LI>";
echo "<A HREF=\"", $myrow["URL"], "\">", $myrow["Title"];
echo "</A> - ", $myrow["Information"], "<BR>";
} while ($myrow = mysql_fetch_array($result));
echo "</OL>";
};
?>

<H3>3. Send a suggestion</H3>
<FORM METHOD="post"
ACTION="testevolinks.html?choice=reply">
Your name: <INPUT TYPE="text" NAME="author" SIZE="30">
Email: <INPUT TYPE="text" NAME="email" SIZE="29">
<BR>
Your suggestion:
<TEXTAREA NAME="suggestion" COLS="60"
WRAP="VIRTUAL" ROWS="3"></TEXTAREA>
<BR>
<INPUT TYPE="submit" NAME="submit" VALUE="Send!">
</FORM>
<?php
if ($submit)
{ };
?>

</BODY>
</HTML>

```

In this code we easily recognise that basic structure of the test page, which has already been discussed:

1. Search facilities: here we see calls like "testevolinks.html?choice=paper" as well as an input form into which free-format requests can be made
2. Search results: here come the tricky PHP code with choices, requests to MySQL, and the production of output in HTML format
3. Send a suggestion: here the user is made able to send an email to the maintainer of the database; however, the PHP call to the email service has been deleted from this test page

No attempts will be made to explain the details of the code (this explanation is to some extent found in the above sections). Two comments are, however, relevant:

- Although the code gives the basics of the production of the file "testevolinks.html?choice=paper", a few features have been omitted for reasons of clarity and simplification

- The choices in section 2 are made simpler than in real-life PHP scripts. In the present simplified version, the variable choice only has a few values (reply, all, paper, person, request, reply). Therefore, it can be tested in a very simple way. In actual life more complex decision structures are needed. They may be constructed with if-else expressions.

If you want to explore PHPs potential for creating more complex web pages and web sites, visit the [extended test page](#) and the [download site](#).

## Literature and documentation

- [DuBois, Paul \(2003\), MySQL \(2nd edn\) \[amazon.com\]](#)
- [Greenspun, Philip \(1999\), Philip and Alex's Guide to Web Publishing \[amazon.com\]](#)
- [HTML documentation](#)
- [Merrall, Graeme: PHP/MySQL Tutorial](#)
- [MySQL documentation](#)
- [PHP documentation](#)
- [phpMyAdmin documentation](#)
- [Thomas, Deepak et al. \(2002\), Professional PHP4 Programming \[amazon.com\]](#)

## Caveat

To create database-backed web sites with the described system is not easy. It requires that the developer has a combination of different capabilities. Of special importance is some experience in the incremental method of software development. Beginners often want to make big jumps in the development of their code, but even a tiny error the PHP script means that no output is displayed. Furthermore, there are very poor debugging tools. So the development must take place in tiny steps, and after each change it must be checked whether the script is functioning. Another prerequisite is a good relationship to a competent computer system administrator. The reason for the this requirement is that all the mentioned systems have to be installed and maintained, and this may be hard work.

## Acknowledgements

The author's work with the design principles and implementation of database-backed web sites has been supported by [Lars Andersen, Department of Business Studies](#), Aalborg University and Michael F. S. Christensen, formerly [Department of Computer Science](#), Aalborg University.

## Appendix: The test page

### Test the methods for accessing the database table "ev

This is a test page related to Andersen's report [Creating database-backed web](#)

#### 1. Search facilities

[Find all links](#)

[Find all papers](#)

[Find all persons](#)

Find a word in the database (like: Metcalfe)

#### 2. Search results

**You have requested links with the text metcalfe**

1. [Foster, John and Metcalfe, Stan \(eds\) \(2002\), Frontiers of Evolutionary Economics: Competition, Self-Organization, Innovation Policy \[amazon.com\]](#) - Contributions to the first meeting of the "Brisbane Club"
2. [Metcalfe, Stan](#) - CRIC and University of Manchester, UK. Important contributor to both evolutionary economic theory its applications
3. [Metcalfe, Stan \(1998\), Evolutionary Economics and Creative Destruction \[amazon.com\]](#)

#### 3. Send a suggestion

Your name:  Email:

Your suggestion:

[No email will actually be sent, but you will see the response!]