



**Anthony Awtrey Consulting - Database Enabled Websites**

# Table of Contents

<a href="#"><u>Database Enabled Websites</u></a> .....	1
<a href="#"><u>Relational Databases</u></a> .....	2
<a href="#"><u>What is a database</u></a> .....	2
<a href="#"><u>How it works</u></a> .....	2
<a href="#"><u>Security</u></a> .....	3
<a href="#"><u>MySQL Tools</u></a> .....	3
<a href="#"><u>Creating Tables</u></a> .....	4
<a href="#"><u>Adding Data</u></a> .....	5
<a href="#"><u>Searching The Table</u></a> .....	5
<a href="#"><u>Updating Existing Data</u></a> .....	6
<a href="#"><u>Deleting Data</u></a> .....	6
<a href="#"><u>Saving A Database</u></a> .....	6
<a href="#"><u>PHP Programming</u></a> .....	8
<a href="#"><u>The Guestbook</u></a> .....	8
<a href="#"><u>Getting Started</u></a> .....	8

# Database Enabled Websites

This is a set of documents that outline the advantage and practical applications of database enabled websites. With the advent of low cost or free standards-based relational databases it has allowed many sites to explore their uses in web applications.

The sample application this tutorial is built around is a [guestbook](#) that I wrote for PHP. The application is available for download in the [File Area](#). Feel free to use, modify or redistribute it any way it pleases you since I have released the code under the [GPL](#).

I have developed this tutorial with some specific parameters in mind. I have tailored all the database information here to reflect the [MySQL](#) relational database because that is what is used by my provider [Pair Networks](#) and is the database I personally prefer. The programming instructions are based on the [PHP](#) server-side scripting language and the webserver is (of course) [Apache](#).

I will outline how to get Debian GNU/Linux to run these programs, but if you get it working with another distribution and will send [me](#) the instructions on how you did it I will add it here.

NOTE: On October 20th 1999 we had over 240 entries in the guestbook application so I modified it to only show the last 20 entries. I am in the process of updating the tutorial to reflect these changes.

Thanks to [othniel@netscape.net](mailto:othniel@netscape.net) and [dmcaruso@netscape.net](mailto:dmcaruso@netscape.net) who submitted the RedHat PHP outline.

## The Materials

- [The Guestbook Application](#)
- [The Guestbook Code](#)

## The Tutorial

- Chapter 1 - [The Database](#)
- Chapter 2 - [PHP Programming](#)
- Chapter 1 2 - [In PDF format](#)

## Distribution Specifics

- [Doing It With Debian](#)
- [Running PHP In RedHat](#)

# Relational Databases

One of the most powerful tools computers give us is the ability to store and search data. Early applications stored data for programs in files and used indexes to search the files for particular bits of data. These programs didn't fare very well on networks because there are problems with more than one computer trying to update data in a file simultaneously. This fact and the lack of standard database structure and command syntax encouraged the creation of the networked relational database.

When you write a program and want to be able to manipulate and search data you often have to construct the data files and the code to perform the manipulation and searches. This is a time consuming task and if you are developing many different applications you have to reinvent the data handling routines for every job. By removing the database functions such as file management, indexing, searches and simultaneous data access, you speed up development of applications and allow for these specialized database programs to become highly optimized and scalable.

## What is a database

A database is simply organized data. A database contains tables which are basically descriptions of types of data. Tables in turn contain records which is the actual data.

By using a common identifier between tables it is possible to "relate" one table to another. For instance, imagine you had a table that contained data about items sold in a store today. You also have tables that give you the details about the items and the vendors that sell them to you.

If you want to get a list of how much of each vendors items your sold today you will have to relate the sales table, the items table and the vendors table by a common field they all share. This aspect of relating data is what gives a relational database its power.

## How it works

A relational database can be seen as the data handling part of another application. The application instructs the database to perform searches, as well as add, delete and modify data via the Structured Query Language or SQL. The SQL standard is supported by all major database vendors, but the implementation of the full standard in all cases is not a guarantee. The common workhorse functions are the same in most cases.

An easy way to understand how this works is to imagine you need to instruct someone to list the vendors your company uses. You have to write the instructions to perform this action. Before relational databases you would have to write the instructions to cover walking down to accounting, looking for the right file cabinet, opening it, finding the correct files, opening each of them and copying all the names down. If someone else was using a file you needed, you simply had to wait until they finished. Since relational databases came into existence those steps have been reduced to a single phone call and the uttered phrase, "SELECT \* FROM VENDORS;"

SQL is a very human readable language. It does have syntax rules, but it is not as hard as learning a programming language. The most basic types of queries are SELECT, INSERT and UPDATE. Select searches for data, INSERT adds data and UPDATE changes existing data. DELETE is also fairly common,

but I end up using it to clean up bad records by hand rather than in a program.

## Security

Relational databases also have excellent security. In most database programs there is a special database that contains access permissions for users and databases. This allows a database administrator the ability to tune permissions to needs. The basic set of permissions in MySQL include the following:

- Select Priviledge - Ability to search data in tables
- Insert Priviledge - Ability to add data in tables
- Update Priviledge - Ability to modify data in tables
- Delete Priviledge - Ability to delete data in tables
- Index Priviledge - Ability to index tables
- Alter Priviledge - Ability to alter tables
- Create Priviledge - Ability to create databases
- Drop Priviledge - Ability to delete databases
- Grant Priviledge - Ability to delete databases
- Reload Priviledge - Ability to reload database priviledges
- Shutdown Priviledge - Ability to shutdown the database program
- Process Priviledge - Ability to change the individual threads running in the database program
- File Priviledge - Ability to import / export data from / to files

The administrator has a special account that has all the priviledges to start with and can be used to create custom accounts. If you intend to run a database yourself you will have to create databases and assign permissions yourself. While not especially complicated, it is tricky and is detailed step by step for MySQL on Debian at [my website](#).

## MySQL Tools

MySQL is a freely available yet full featured relational database. It has a number of tools to manage the databases it operates, but there are only three I use regularly and only one of those is necessary if you aren't managing the database yourself. The command is "mysql" and it is a shell that allows an operator to enter SQL commands directly to the database. This command requires a few command line arguments to get it connected properly. A typical command line I use is as follows:

```
mysql -uuser -ppassword -hhost.domain.com database
```

- *-u* specifies the database user account to use
- *-p* specifies the database user password to use
- *-h* specifies the database server to connect to
- *database* specifies the database to act on

Once you are connected you will see this:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 1 to server version: 3.21.33b
```

```
Reading history-file /home/aawtrej/.mysql_history  
Type 'help' for help.
```

mysql>

That `mysql>` prompt will allow you to enter any valid SQL statement and it will be executed by the database. I use this to test queries and manage the tables of data in the database.

SQL statements can be entered all on one line or broken into smaller pieces since the "mysql" program ignores extra white space which includes tabs, spaces, or carriage returns. All SQL statements must contain the correct syntax and be terminated with a ";" (semicolon).

## Creating Tables

I will now take you through setting up a simple table. We want to store messages our site visitors leave. We want to track who they are, their email address, the time they visited and their message. Additionally, we need to define a primary key to uniquely identify the record. Since we may get more than one "Tom" we can't use any of the data fields for the key. Here is the SQL statement to create a table to store the data in.

```
CREATE TABLE guests (  
  guest_id int(4)  
    unsigned  
    zerofill  
    DEFAULT '0000'  
    NOT NULL  
    auto_increment,  
  guest_name varchar(50),  
  guest_email varchar(50),  
  guest_time timestamp(14),  
  guest_message text,  
  PRIMARY KEY (guest_id)  
);
```

We have defined 5 types of data to track for our guestbook. The first field is named "guest\_id". The "int(4)" means it is a 4 digit number. The "unsigned" means that it can only be a positive number. The "zerofill" is something I add to force the number to take up all the digits by displaying leading zeros. Using that parameter will make the numbers "0001", "0002", etc. Issuing the DEFAULT statement makes the counting start at "0000".

"NOT NULL" requires a little explanation. When a field has never had any data posted into it, it is considered NULL. This is not the same as containing "0" since "0" is a character. A field that is created as "NOT NULL" must have data in it. Lastly, "auto\_increment" means that the number in subsequent records will be generated by adding "1" to the previous record.

The next 2 fields "guest\_name" and "guest\_email" are defined as "varchar(50)". This type of field is a variable length character string. This means the name can be anywhere from 0 to 50 characters long.

The "guest\_time" data type is "timestamp(14)". The timestamp datatype is automatically updated by the database when the record is updated. The number of characters defines the format of the data. 14 characters includes a 4 digit year and a 2 digit month, day, hour, minute and second.

The "guest\_message" type is "text". This is a non-indexed data type that allows for 65,535 characters to be stored in it. By non-indexed it means that the data is not readily searchable by the database.

The last line defines the "guest\_id" as the primary key. Every table should have a "PRIMARY KEY" defined.

This is the unique identifier for each record in the table.

The command to look at all the tables in a database is:

```
SHOW TABLES;
```

You can look at the properties of the fields in a table by issuing this statement:

```
DESCRIBE TABLES;
```

There is also ways to alter table and field properties using the "ALTER TABLE" function, but I leave that to you for your homework. Feel free to pursue the online documentation at the [MySQL website](#).

Deleting a table is as simple as issuing the following command.

```
DROP TABLE guests;
```

## Adding Data

Now that we have the table created we will have to add data to it. I suggest putting a record in directly to test the sql statement you will use in the program. Here is the SQL query to add a record to the database:

```
INSERT INTO guests (  
  guest_id, guest_name, guest_email, guest_time, guest_message  
) values(  
  0000, 'Tony', 'tony@awtrey.com', NULL, 'This is how it works!'  
);
```

The statement starts by telling the database we want to insert data into the "guests" table. Next we list the fields we are going to update and finally, the data we want inserted in each field. You have to match the order of the listed fields and the data.

The "guest\_id" field is autoincrementing, so it doesn't really matter what you insert into it. I put 0000 just because I wanted to. Strings like the ones for "guest\_name", "guest\_email" and "guest\_message" must be quoted. To put a quote character in the field you must put a "\" (backslash) in front of it. The "guest\_time" field is set automatically, but I insert the special NULL character, which is the manual trigger, because that's the kind of guy I am.

If that is added to the database correctly you will see:

```
Query OK, 1 row affected (0.00 sec)
```

## Searching The Table

Getting the data back from the database is even easier. This statement will return all the data in the "guests" table:

```
SELECT * FROM guests;
```

In the "mysql" program the returned data will appear in a text form that resembles this:

GUEST_ID	GUEST_NAME	GUEST_EMAIL	GUEST_TIME	GUEST_MESSAGE
0001	Tony	tony@awtrey.com	19990811105100	This is how it works!

Sometimes you only want certain records. The statement to limit a search to particular criteria is like this:

```
SELECT * FROM guests WHERE guest_name='Tony';
```

This will only return the records where the "guest\_name" is "Tony". You can also limit the fields that are returned in this way:

```
SELECT guest_name,guest_email FROM guests;
```

This should return something that looks like this:

GUEST_NAME	GUEST_EMAIL
Tony	tony@awtrey.com

## Updating Existing Data

Changing data is a little trickier. To change the email address of the previously entered data use this statement:

```
UPDATE guests  
SET guest_email='info@awtrey.com'  
WHERE guest_email='tony@awtrey.com';
```

The successful execution of the query will result in:

```
Query OK, 1 row affected (0.00 sec)
```

The resulting data when selected should look like this:

GUEST_ID	GUEST_NAME	GUEST_EMAIL	GUEST_TIME	GUEST_MESSAGE
0001	Tony	info@awtrey.com	19990811105100	This is how it works!

## Deleting Data

To delete all the data from a table only needs the following statement:

```
DELETE FROM guests;
```

To delete specific records the query would look like this:

```
DELETE FROM guests WHERE guest_name='Tony';
```

The successful execution of the query will result in:

```
Query OK, 1 row affected (0.00 sec)
```

## Saving A Database

One other valuable feature most databases possess is the ability to output SQL statements that define the structure and content of the tables in a database. MySQL is no exception and includes the "mysqldump" utility for this purpose. The command is issued from the command line, not within the "mysql" program.

```
mysqldump -uuser -ppassword -hhost.domain.com database > filename.sql
```

- *-u* specifies the database user account to use
- *-p* specifies the database user password to use
- *-h* specifies the database server to connect to
- *database* specifies the database to act on
- *> filename.sql* directs the output to a file. Normally output is directed to the screen.

The output of a mysqldump can be directed back into a blank database to recreate a complete set of tables and data. This is especially handy if you develop a structure on a staging system and need to quickly and easily move the whole contents of a database to a new system. Here is the command to do that using the "mysql" utility.

```
mysql -uuser -ppassword -hhost.domain.com database < filename.sql
```

- *-u* specifies the database user account to use
- *-p* specifies the database user password to use
- *-h* specifies the database server to connect to
- *database* specifies the database to act on
- *< filename.sql* directs the mysql program to read instructions from a file instead of the command line.

---

© 1999 by [Anthony Awtrey Consulting](#)

# PHP Programming

**Due to the length of some of the code lines, this page may not display properly on all browsers or all resolutions.**

NOTE: On October 20th 1999 we had over 240 entries in the questbook application so I modified it to only show the last 20 entries. I am in the process of updating the tutorial to reflect these changes.

PHP is a server-side scripting language that can be used on a host of web servers and platforms. I prefer to use it with Apache on either FreeBSD or Linux, but it can even run on Win32 platforms.

What server-side scripting language means is that the script is put into the HTML files that make up a site, but the server processes the script BEFORE it is sent to the client browser. PHP code is not visible if you view the source of a page because the server processes the code and returns only the output. This is easier to code and debug than writing CGI scripts in Perl or C since the HTML form and related code are all in one page and PHP puts any errors on the browser.

Another advantage that PHP offers is the ability to directly connect to relational databases using full featured internal functions. It supports a whole fleet of databases including Oracle, DB2, mSQL and MySQL. I have written this tutorial with an eye to the MySQL database server since that is what this site uses. Most of the information here should be applicable regardless of the backend database.

## The Guestbook

To highlight how easy it is to do database backed websites I wrote a simple guestbook using PHP. The code will be examined here to explain how the PHP code works and why I did certain things. One of my goals in writing database backed websites is to keep related pages and functions as simple as possible, but where possible to use a single file to make troubleshooting easier. This might make things more confusing at first, but it will work better in the long term.

The application is very simple. When you go to the page it loads the "Add A Message" form and then connects to the database to get all the guestbook entries. Once it has selected the data it builds the html to display the output and delivers it to the browser.

When someone fills out the form and hits the submit button, the form action reloads the page, adds the data to the database, then selects all the data again including the new entry and builds the html to display the output and delivers it to the browser.

The code I have commented on this page is available in my [File Area](#). In my comments I have used teal to represent comments, red to indicate PHP code and blue to indicate HTML.

## Getting Started

The first item of note about PHP is how the server knows whether a bit of text in the file is PHP or HTML. The server is alerted to begin processing PHP code when it encounters a `<?>`. It continues to process the text as PHP code until it encounters a `?>`. The wonderful part about PHP is that you can break into and out of PHP

processing mode for little snippets of code.

Lets start by looking at the guestbook program as an example. The very first line of code `<?>` tells the server to enter PHP mode. The next line `/*` should look familiar to any Perl, C or Java programmers out there as the beginning of a comment. That kind of comment will run until it hits a `*/`

```
<?
/*
PHP Guestbook
Written by Tony Awtrey
Anthony Awtrey Consulting
See http://www.awtrey.com/support/dbeweb/ for more information
```

1.1 - Oct. 20, 1999 - changed the SQL statement that reads data back out of the database to reverse the order putting the newest entries at the top and limiting the total displayed by default to 20. Added the ability to get the complete list by appending the URL with `'?complete=1'`. Added the code and additional query to count and list the total number of entries and included a link to the complete list.

1.0 - Initial release

This is the SQL statement to create the database required for this application.

```
CREATE TABLE guests (
  guest_id
    int(4)
    unsigned
    zerofill
    DEFAULT '0000'
    NOT NULL
    auto_increment,
  guest_name varchar(50),
  guest_email varchar(50),
  guest_time timestamp(14),
  guest_message text,
  PRIMARY KEY (guest_id)
);

*/
```

This next section of code demonstrates a second kind of comment that PHP shares with C++, Perl and Javascript. Any line that starts with `//` is taken as a comment. The next line of actual PHP code tests to see if this page is being loaded as a HTTP POST query. When HTML forms are processed, they can either put data to the server using the GET method or the POST method. GET is the one that puts a `?` after the page followed by a string of key=value pairs. POST methods hide the data being input by listing it is a special environment variable called `HTTP_POST_VARS`. If we are posting, this is the beginning of how we put data into the database.

```
////////////////////////////////////
// This checks to see if we need to add another guestbook entry.
////////////////////////////////////
if (($REQUEST_METHOD=='POST')) {
```

This next section of code removes the dangerous characters from the data from the form. Any input recieved from an unverified source like the Internet should be treated as if it were an attack or crack attempt. To prevent this I remove the redirection characters (`<>`) and the "pipe" character (`|`) and replace them with spaces. PHP provides a complete set of string handling functions, including the `strtr` or "string translate"

function.

Another nice thing it does is automatically adds the backslashes to quote characters for you to keep from causing problems sending the string data to the database.

The last item of note is the use of the `$$key` to indirectly refer to a variable. This code basically walks through each key=value pair and temporarily assigns the value to a variable called `$this`. Once the data has been untainted it is assigned back into the original value statement

```

////////////////////////////////////
// This loop removed "dangerous" characters from the posted data
// and puts backslashes in front of characters that might cause
// problems in the database.
////////////////////////////////////
for(reset($HTTP_POST_VARS);
    $key=key($HTTP_POST_VARS);
    next($HTTP_POST_VARS)) {
    $this = addslashes($HTTP_POST_VARS[$key]);
    $this = strtr($this, ">", " ");
    $this = strtr($this, "

```

Now we need to check that all the fields were filled in completely. If they were filled out we are ready to put the data into the table.

```

////////////////////////////////////
// This will catch if someone is trying to submit a blank
// or incomplete form.
////////////////////////////////////
if ($name $email $message ) {

```

First we define the INSERT query we are going to send. PHP automatically puts the data on the form into variables named the same as the form INPUT tags. `$name`, `$email` and `$message` are all created by PHP when the form is posted. These expand out in the definition of the `$query` variable. Note the `.=` symbol. This concatenates the subsequent `$query` definitions with the first instead of redefining `$query`.

The three lines that begin with "mysql" are all it takes to put data into a database. The first line connects to the database host using the supplied username and password, the second line selects the database to have the data updated to and the third line issues the query. the "or die" statements are a primitive error handler that will tell you at least what failed if you did something wrong.

```

////////////////////////////////////
// This is the meat of the query that updates the guests table
////////////////////////////////////
$query = "INSERT INTO guests ";
$query .= "(guest_id, guest_name, ";
$query .= "guest_email, guest_time, guest_message) ";
$query .= " values(0000,'$name','$email',NULL,'$message)";
mysql_pconnect("db2.pair.com","tator_w","password")
    or die("Unable to connect to SQL server");
mysql_select_db("tator_awtrey") or die("Unable to select database");
mysql_query($query) or die("Insert Failed!");

```

If the form was NOT completely filled out a variable called `$notall` is created and set to "1". If the form was NOT called via the POST method all the preceding code was skipped down to the last bracket. Then the `?>` indicates we are back in HTML land.

```

} else {

////////////////////////////////////
// If they didn't include all the required fields set a variable

```

```
// and keep going.
////////////////////////////////////
$notall = 1;

}
}
?>
```

This next section is the beginning of the standard HTML part of the page.

```
<!-- Start Page -->
<HTML>
<HEAD>
<TITLE>Add a Message</TITLE>
</HEAD>

<BODY BGCOLOR="white">

<H1>Add A Message</H1>
```

This next section is a good example of how to mix and match PHP and HTML. The first line is a standard HTML comment. The second line switches to PHP and sets an IF statement to detect if the `$notall` variable was set to indicate that the form was incomplete. If it was set, then output the next line *which is HTML, not even PHP code*, and continue. If it was not set the HTML is not returned to the browser. How about that?!

```
<!-- Let them know that they have to fill in all the blanks -->
<? if ($notall == 1) { ?>
<P><FONT COLOR="red">Please answer all fields</FONT></P>
<? } ?>
```

This is the section that creates the form on the browser. The things to note are the little snippets of PHP to fill in the fields on the form if the person submitted an incomplete form. This keeps them from having to input **ALL** the data in again.

```
<!-- The bits of PHP in the form allow the data that was already input
to be placed back in the form if it is filled out incompletely -->
```

```
<FORM METHOD="post" ACTION="guest.php3">
<PRE>
Your Name:      <INPUT
                  TYPE="text"
                  NAME="name"
                  SIZE="20"
                  MAXSIZE="50"
                  VALUE="<? echo $name; ?>">
Your Email:     <INPUT
                  TYPE="text"
                  NAME="email"
                  SIZE="20"
                  MAXSIZE="50"
                  VALUE="<? echo $email; ?>">

Enter Message:
<TEXTAREA NAME="message" COLS="40" ROWS="8" WRAP="Virtual">
<? echo $message; ?>
</TEXTAREA>

<INPUT TYPE="submit" VALUE="Add">

</PRE>
</FORM>
```

<HR>

Here we have the same two lines that connect to the host and select the database as before. The query is defined and called.

```
<?
////////////////////////////////////
// This is where we connect to the database for reading.
////////////////////////////////////
mysql_pconnect("db2.pair.com","tator_r","password")
    or die("Unable to connect to SQL server");
mysql_select_db("tator_awtrey") or die("Unable to select database");
```

This section performs a select query that counts the number of records in the database.

```
////////////////////////////////////
// This is where we count the number of entries.
////////////////////////////////////
$query = "SELECT COUNT(*) FROM guests";
$numguests = mysql_query($query) or die("Select Failed!");
$numguest = mysql_fetch_array($numguests);
```

This query is set depending on whether the URL is set to retrieve all the records or not.

```
////////////////////////////////////
// This is where we decide to get all the entries or just the last 20.
// This variable is set by just adding a '?complete=1' after the URL.
////////////////////////////////////
if ($complete == 1) {
    $query = "SELECT * FROM guests ORDER BY guest_time DESC";
} else {
    $query = "SELECT * FROM guests ORDER BY guest_time DESC LIMIT 20";
}
$guests = mysql_query($query) or die("Select Failed!");
```

?>

```
<!-- This is where we report the total messages. -->
<P>
<A HREF="guest.php3?complete=1"><? echo $numguest[0]; ?> people</A> have
left me a message.
</P>
```

>?

This next section sends a query to the database to count the total number of records in the table that have data in them. This is an easy way to count records.

```
////////////////////////////////////
// This is where we count the number of entries.
////////////////////////////////////
$query = "SELECT COUNT(*) FROM guests";
$numguests = mysql_query($query) or die("Select Failed!");
$numguest = mysql_fetch_array($numguests);
```

?>

```
<!-- This is where we report the total messages. -->
<P>
<A HREF="guest.php3?complete=1"><? echo $numguest[0]; ?> people</A> have
left me a message.
</P>
```

```
<?
```

This next section is a loop that runs as long as there are records. The **while** statement returns each record to an array called **\$guest**. It contains all the data selected from the table in array variables named for the field name. Therefore if you want the "guest\_name" value, you can get it by referring to **\$guest['guest\_name']** This is an incredibly powerful feature since it allows you to change the structure of a database without breaking the application.

```

////////////////////////////////////
// This will loop as long as there are records waiting to be processed.
// Notice the plain HTML inside the while loop structure. PHP is flexible
// enough to allow you to break into and out of the "code" at any point.
////////////////////////////////////
while ($guest = mysql_fetch_array($guests)) {

?>

```

This is the actual beginning of the loop. Most of it is standard HTML with PHP inserted to supply the database fields.

```

<TABLE BORDER="1" WIDTH="500">
<TR><TD>
Name: <? echo $guest['guest_name']; ?>
</TD><TD>
Email: <A HREF="mailto:<? echo $guest['guest_email']; ?>">
      <? echo $guest['guest_email']; ?></A>
</TD><TD>

```

Using the MySQL timestamp datatype in PHP requires a little coding. This next bit takes apart the MySQL data and generates a formatted date. The **substr** function just grabs a substring from a given string. The **mktime** function generates a timestamp that the **date** function uses to generate the date and time in the format specified.

```

<
////////////////////////////////////
// The database has a timestamp record type that we can use to show the
// date the guestbook was filled out.
////////////////////////////////////
$datefromdb = $guest['guest_time'];
$year = substr($datefromdb,0,4);
$mon  = substr($datefromdb,4,2);
$day  = substr($datefromdb,6,2);
$hour = substr($datefromdb,8,2);
$min  = substr($datefromdb,10,2);
$sec  = substr($datefromdb,12,2);
$orgdate = date("l F dS, Y h:i A",mktime($hour,$min,$sec,$mon,$day,$year));
echo "Date: $orgdate\n";
?>

```

The last bit just uses PHP to drop in the variables where needed. The closing **}** ends the while loop and once all the data is output, the page finishes up with standard HTML.

```

</TD></TR>
<TR><TD COLSPAN="3">
<? echo $guest['guest_message']; ?>
</TD></TR>
</TABLE>
<BR>

<? } ?>

```

```
</BODY>  
</HTML>
```