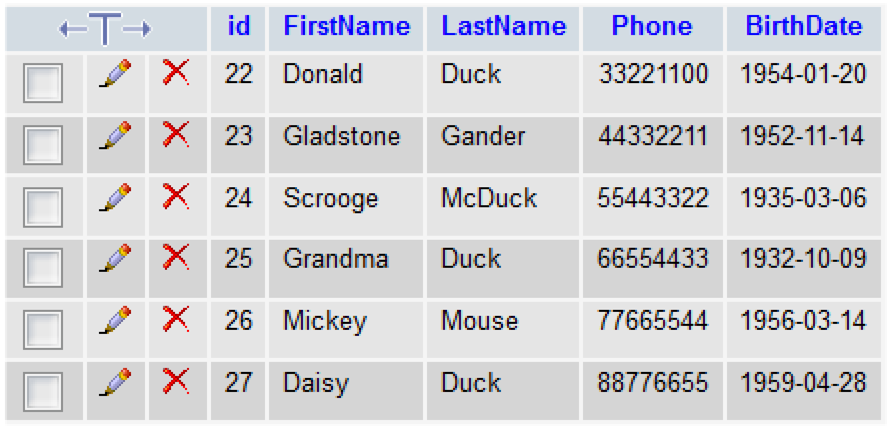
CHAPTER ONE

Databases

A database is a collection of information / data that is organized so that it can easily be retrieved, administrated and updated. Databases thereby enable the opportunity to create dynamic websites with large amounts of information.

A database usually consists of one or more tables. If you are used to working with spreadsheets, or maybe have used databases before, tables will look familiar to you with columns and rows:

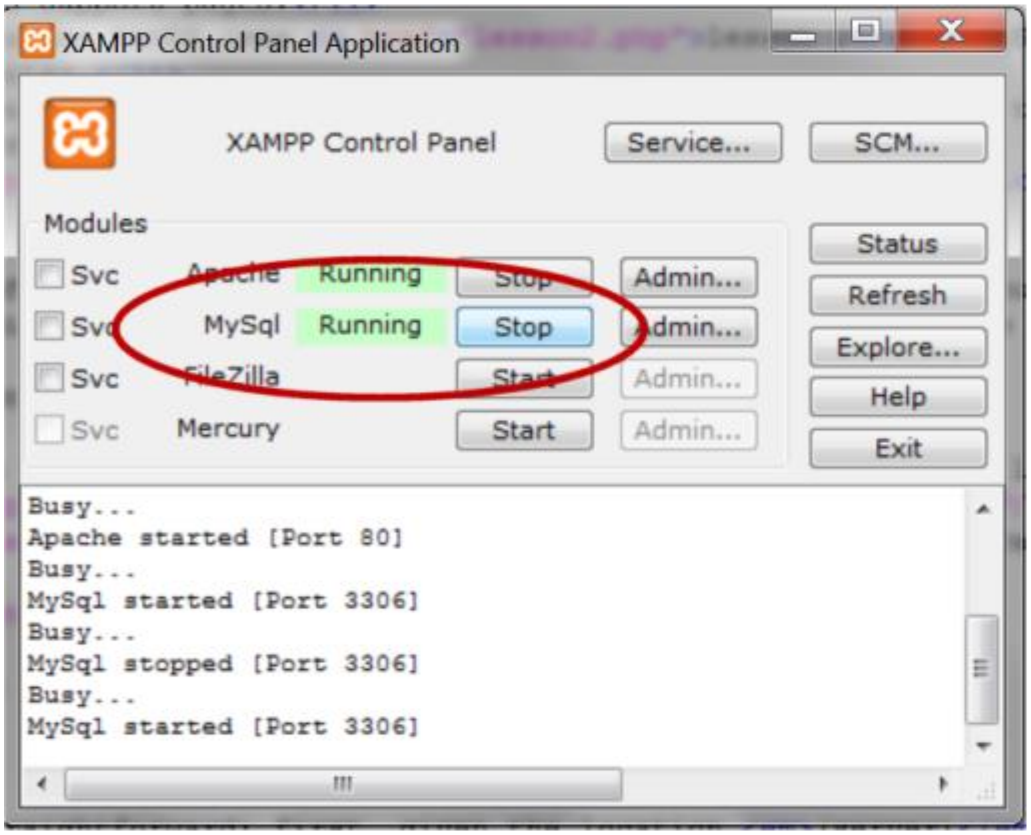


←T→			id	FirstName	LastName	Phone	BirthDate
☐	✎	✗	22	Donald	Duck	33221100	1954-01-20
☐	✎	✗	23	Gladstone	Gander	44332211	1952-11-14
☐	✎	✗	24	Scrooge	McDuck	55443322	1935-03-06
☐	✎	✗	25	Grandma	Duck	66554433	1932-10-09
☐	✎	✗	26	Mickey	Mouse	77665544	1956-03-14
☐	✎	✗	27	Daisy	Duck	88776655	1959-04-28

There are many different databases: MySQL, MS Access, MS SQL Server, Oracle SQL Server and many others. In this tutorial, we will use the MySQL database. MySQL is the natural place to start when you want to use databases in PHP.

You need to have access to MySQL in order to go through this lesson and the next lessons:

- If you have a hosted website with PHP, MySQL is probably already installed on the server. Read more at your web host's support pages.
- If you have installed PHP on your computer yourself and have the courage to install MySQL as well, it can be downloaded in a free version (MySQL Community Edition) at the MySQL's website.
- If you use XAMPP (see lesson 2), MySQL is already installed and ready to use on your computer. Just make sure MySQL is running in the Control Panel:



```
Busy...
Apache started [Port 80]
Busy...
MySql started [Port 3306]
Busy...
MySql stopped [Port 3306]
Busy...
MySql started [Port 3306]
```

In the rest of this lesson, we will look more closely at how you connect to your database server, before we learn to create databases and retrieve and update data in the following sessions.

Connection to database server

First, you need to have access to the server where your MySQL database is located. This is done with the function mysql_connect with the following syntax:

mysql_connect*(server, username, password)*

Pretty straightforward: First, you write the location of the database *(server)*, and then type in the *username* and *password*.

If you have your own website, you should read about location of your MySQL server on your host's support pages. Username and password will often be the same as those you use for FTP access. Otherwise contact your provider.

Example of a MySQL connection on a hosted website:

```
mysql_connect("mysql.myhost.com", "user001", "sesame") or die(mysql_error());
```

Example of a MySQL connection with XAMPP (default settings):

```
mysql_connect("localhost", "root", "") or die (mysql_error());
```

In the examples are added or die(mysql_error()) which, in brief, interrupts the script and writes the error if the connection fails.

Now we have made a connection to a MySQL server, and can now start creating databases and retrieve and insert data. This is exactly what we will look at in the next lessons.

By the way, keep in mind that it is good practice to close the database connection again when you're finished retrieving or updating data. This is done with the function mysql_close.

CHAPTER TWO

Create databases and tables

In the previous lesson we looked at how to create a connection to a database server. Next step is to create databases and tables.

We'll look at two ways to create databases and tables. First, how it is done in PHP, and then how it's made with the more user-friendly tool PhpMyAdmin, which is standard on most web hosts and in XAMPP.

If you have a hosted website with PHP and MySQL, a database has probably been created for you already and you can just skip this part of the lesson and start creating tables. Again, you should consult your host's support pages for more information.

Create a database and tables with PHP

The function mysql_query are used to send a query to a MySQL database. The queries are written in the language **S**tructured **Q**uery **L**anguage (SQL). SQL is the most widely used language for database queries - not only for MySQL databases - and is very logical and easy to learn. In this lesson and the next, you will learn the most important SQL queries.

When creating a database, the SQL query CREATE DATABASE is used with the following syntax:

```
CREATE DATABASE database name
```

Logical and easy, right!? Let's try to put it into a PHP script:

```
mysql_connect("mysql.myhost.com", "user", "sesame") or die(mysql_error());

mysql_query("CREATE DATABASE mydatabase") or die(mysql_error());

mysql_close();
```

First, we connect to the MySQL server. Next, we create a database named "mydatabase". And finally, we close the connection to the MySQL server again.

So far so good... but things become a little bit more complicated when we want create tables in PHP. When creating tables, we use the SQL query CREATE TABLE with the following syntax:

```
CREATE TABLE *table name*
(
*column1_name DATA_TYPE,*
*column2_name DATA_TYPE,*
*column3_name DATA_TYPE,*
...
)
```

table_name and *column_name* are of course the name of the table and the columns, respectively. *DATA_TYPE* are used to specify the data type to be inserted into the column. The most commonly used data types are:

INT	For numbers without decimals
DECIMAL	For numbers with decimals
CHAR	Short text up to 255 characters
TEXT	For plain text up to 65,535 characters
LONGTEXT	For long passages of text up to 4,294,967,295 characters
Date	For dates in the format YYYY-MM-DD
Time	For time in the format HH:MM:SS
DATETIME	For date and time in the format YYYY-MM-DD HH:MM:SS

All in all, logical and relatively easy. Let's try to put it into an example:

```
mysql_connect("mysql.myhost.com", "user", "sesame") or die(mysql_error());
mysql_select_db("people") or die(mysql_error());
```

```
mysql_query("CREATE TABLE MyTable (
 id INT AUTO_INCREMENT,
 FirstName CHAR,
 LastName CHAR,
 Phone INT,
 BirthDate DATE
 PRIMARY KEY(id)
)") Or die(mysql_error());
mysql_close ();
```

In the example, we start by connecting to the MySQL server. Next we use the function mysql_select_db to select the database "people". Then we create the table "persons" with 5 columns.

Note that at the "id" column, we first use INT to specify that the column contains numbers and then add AUTO_INCREMENT to automatically increase the number and ensure a unique ID for each row.

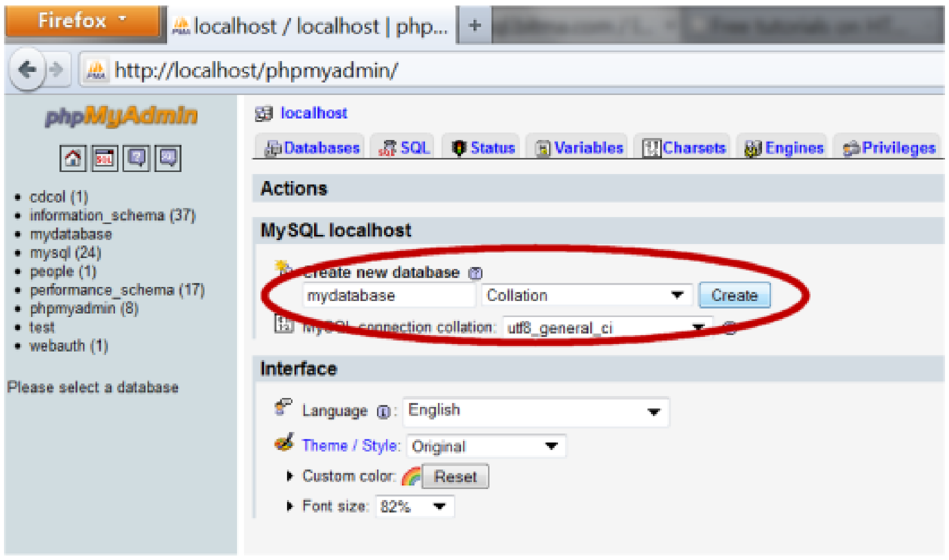
At the end, we use PRIMARY KEY to set the "id" column as the primary key. The primary key uniquely identifies each record (/row) in the table, which becomes very useful later when we update the database.

Create database and tables with phpMyAdmin

It can be useful to be able to create databases and tables directly in PHP. But often, it will be easier to use phpMyAdmin (or any other MySQL administration tool), which is standard on most web hosts and XAMPP. The screendumps below shows how to create a database and tables in phpMyAdmin.

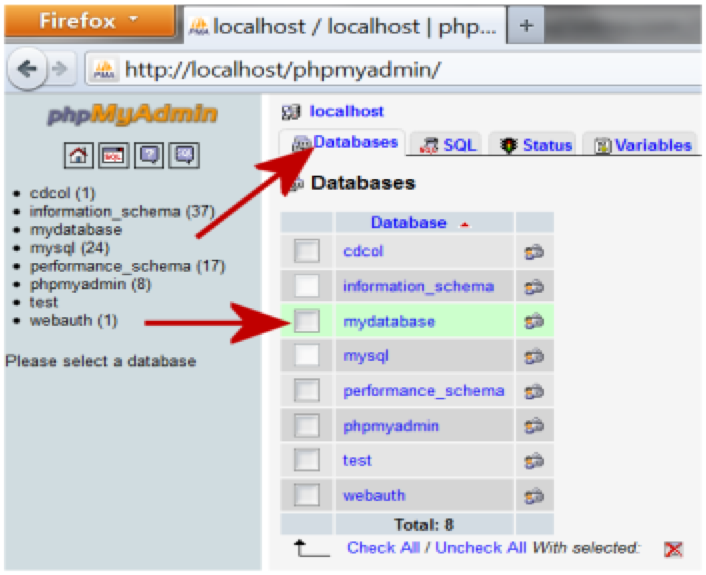
Start by logging onto phpMyAdmin. Often, the address will be the same as your MySQL server (eg. "http://mysql.myhost.com") and with the same username and password. In XAMPP, the address is http://localhost/phpmyadmin/.

When you are logged on, simply type a name for the database and press the button "Create":

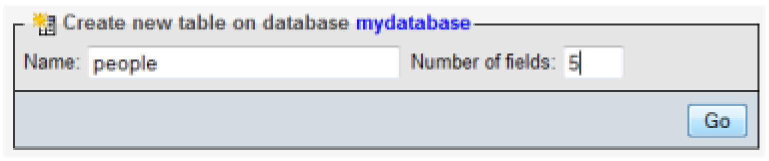


At some hosts, it's possible the have already created a database, and you may not have the rights to create more. If that is the case, you obviously just use the assigned database.

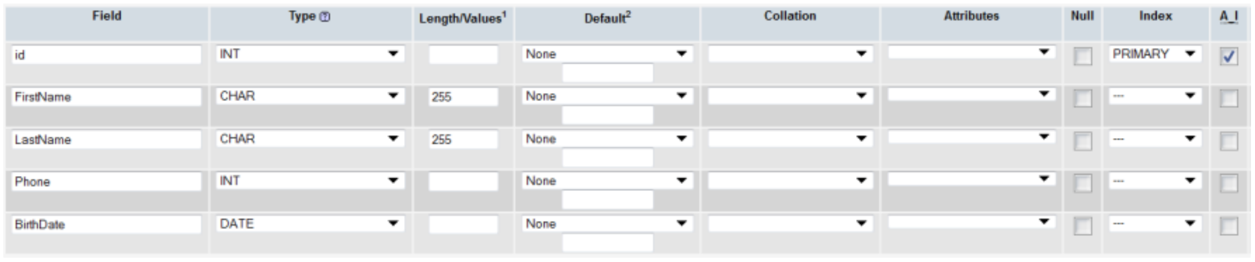
To create a table, click on the tab "Databases" and choose a database by clicking on it:



Then there will be a box titled "Create new table in database", where you type the name of the table and the number of columns and press the button "Go":



Then you can name the columns and set the data type, etc., as in the SQL example above.



Field	Type	Length/Values[1]	Default[2]	Collation	Attributes	Null	Index	A_I
id	INT		None				PRIMARY	✓
FirstName	CHAR	255	None				---	
LastName	CHAR	255	None				---	
Phone	INT		None				---	
BirthDate	DATE		None				---	

Notice, that here we also set "id" as PRIMARY KEY and uses AUTO_INCREMENT (A_I).

Now you have created your own database and table. In the next lessons, we look at how to insert, retrieve and delete data in a database

CHAPTER THREE

Insert data into a database

In this lesson, we look at how you can insert data into the database directly from your PHP scripts.

Insert data using SQL

You use SQL to insert data in a database in the same way that you can use SQL to create databases and tables. The syntax of the SQL query is:

```
INSERT INTO TableName(column1, column2, ...) VALUES(value1, value2, ...)
```

As you can see, you can update multiple columns in the SQL statement by specifying them in a comma-separated list. But of course, it is also possible to specify just one column and one value. The columns that are not mentioned in the SQL statement will just be empty.

Example: Insert a new person in the table

In this example we use the database from lesson 18. Let's say we want to insert a person into the database. It could be the person *Gus Goose* with the phone number *99887766* and *1964-04-20* as the date of birth.

The SQL statement would then look like this:

```
$strSQL = "INSERT INTO people(FirstName,LastName,Phone,BirthDate)
VALUES('Gus','Goose','99887766 ','1964-04-20')";

mysql_query($strSQL) or die(mysql_error());
```

As you can see, SQL statements can get quite long, and you can easily lose track. Therefore, it can be an advantage to write the SQL statement in a slightly different way:

```
strSQL = "INSERT INTO people(";

strSQL = strSQL . "FirstName, ";
strSQL = strSQL . "LastName, "
strSQL = strSQL . "Phone, ";
strSQL = strSQL . "birth) ";

strSQL = strSQL . "VALUES (";

strSQL = strSQL . "'Gus', ";
strSQL = strSQL . "'Goose', ";
strSQL = strSQL . "'99887766', ";

strSQL = strSQL . "'1964-04-20')";

mysql_query($strSQL) or die(mysql_error());
```

This way, the SQL statement is built up by splitting the sentence into small parts and then putting those parts together in the variable **$strSQL**.

In practice, it makes no difference which method you choose, but once you start working with larger tables, it's crucial that you always keep track, so choose the method you find most convenient.

Try running the following code to insert Gus Goose into the database:

```
<html>
<head>
<title>Insert data into database</title>
</head>
<body>
<?php

// Connect to database server
mysql_connect("mysql.myhost.com", "user", "sesame") or die (mysql_error ());

// Select database
mysql_select_db("mydatabase") or die(mysql_error());

// The SQL statement is built
```

```
$strSQL = "INSERT INTO people(";

$strSQL = $strSQL . "FirstName, ";
$strSQL = $strSQL . "LastName, ";

$strSQL = $strSQL . "Phone, ";
$strSQL = $strSQL . "BirthDate) ";

$strSQL = $strSQL . "VALUES(";

$strSQL = $strSQL . "'Gus', ";

$strSQL = $strSQL . "'Goose', ";
$strSQL = $strSQL . "'99887766', ";

$strSQL = $strSQL . "'1964-04-20')";

// The SQL statement is executed
mysql_query($strSQL) or die (mysql_error());

// Close the database connection
mysql_close();
?>

<h1>The database is updated!</h1>
</body>
</html>
```

Save user input into a database

Often you want to save user input in a database.

As you've probably already figured out, this can be done by creating a form as described in lesson 11 - where the values from the form fields can be inserted in the SQL statement. Suppose you have a simple form like this:

```
<form action="insert.php" method="post">
```

```
<input type="text" name="FirstName" />
<input type="submit" value="Save" />

</form>
```

The form submits to the file **insert.php** where you, as shown in lesson 11, can get the user's input by requesting the form content. In this particular example, an SQL statement could look like this:

```
strSQL = "INSERT INTO people(FirstName) values('" . $_POST["FirstName"] . "')"
```

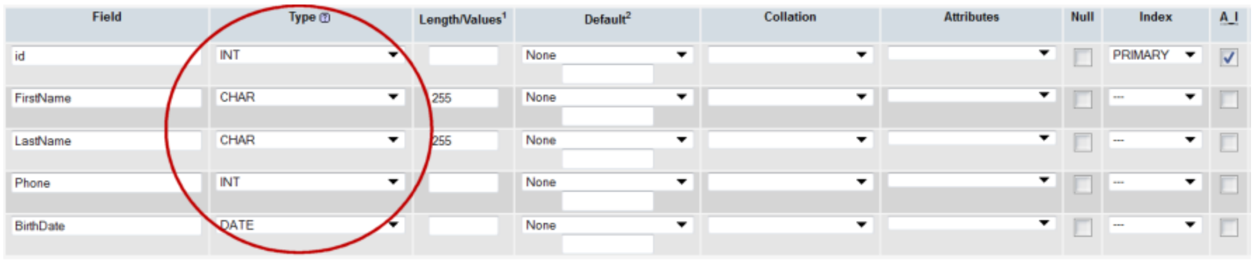
In the same way, it is possible to retrieve data from cookies, sessions, query strings, etc.

Most common beginner mistakes

In the beginning, you will probably get a lot of error messages when you try to update your databases. There is no room for the slightest inaccuracy when you work databases. A misplaced comma can mean the database is not being updated, and you get an error message instead. Below, we describe the most common beginner mistakes.

Wrong data types

It is important that there is consistency between the type of data and column. Each column can be set to a data type. The screenshot below shows the data types for the table "people" in our example.



Field	Type	Length/Values[1]	Default[2]	Collation	Attributes	Null	Index	A_I
id	INT		None			☐	PRIMARY	✓
FirstName	CHAR	255	None			☐	---	☐
LastName	CHAR	255	None			☐	---	☐
Phone	INT		None			☐	---	☐
BirthDate	DATE		None			☐	---	☐

An error occurs if you, for example, attempt to insert text or numbers in a date field. Therefore, try to set the data types as precisely as possible.

Below is the most common data types listed:

Setting	Data Type	Size
CHAR	Text or combinations of text and numbers. Can also be used for numbers that are not used in calculations (e.g., phone numbers).	Up to 255 characters - or the length defined in the "Length"
TEXT	Longer pieces of text, or combinations of text and numbers.	Up to 65,535 characters.
INT	Numerical data for mathematical calculations.	4 bytes.
DATE	Dates in the format YYYY-MM-DD	3 bytes.
TIME	Time in the format hh:mm:ss	3 bytes.
DATETIME	Date and time in the format YYYY-MM-DD hh:mm:ss	8 bytes.

SQL statements with quotes or backslash

If you try to insert text that contains the characters single quote ('), double quote (") or backslash (\), the record may not be inserted into the database. The solution is to add backslashes before characters that need to be quoted in database queries.

This can be done with the function addslashes this way:

```
<?php

$strText = "Is your name O'Reilly?";
$strText = addslashes($strText);

?>
```

All single quotes ('), double quotes (") and backslashs (\) will then get an extra backslash before the character. This would only be to get the data into the database, the extra \ will not be inserted. Please note that PHP runs addslashes on all $_GET, $_POST, and $_COOKIE data by default. Therefore do not use addslashes on strings that have already been escaped.

In the next lesson you will learn to retrieve data from your database. But first, try to insert some more people in your database (as shown in the example above with Gus Goose).

CHAPTER FOUR

Get data from database

Now it's time to retrieve data from our database to our PHP pages.

This is really one of the most important lessons in this tutorial. Once you have read and understood this lesson, you will realize why database-driven web solutions are so powerful, and your views on web development will be expanded dramatically.

SQL queries

To retrieve data from a database, you use queries. An example of a query could be: "get all data from the table 'people' sorted alphabetically" or "get names from the table 'people'".

Again, the language **S**tructured **Q**uery **L**anguage (SQL) is used to communicate with the database. Try looking at this simple example:

> Get all data from the table 'people'
>
> Will be written like this in SQL:
>
> SELECT * FROM people

The syntax is pretty self-explanatory. Just read on and see how SQL statements are used in the examples below.

Example 1: Retrieve data from a table

This example uses the database and table from lesson 19 and lesson 18. Therefore, it is important that you read these lessons first.

The example shows how data in the table "people" is retrieved with an SQL query.

The SQL query returns a result in the form of a series of **records**. These records are stored in a so-called **recordset**. A recordset can be described as a kind of table in the server's memory, containing rows of data (records), and each record is subdivided into individual fields (or columns).

A recordset can be compared to a table where each record could be compared to a row in the table. In PHP, we can run through a recordset with a loop and the function mysql_fetch_array, which returns each row as an array.

The code below shows how to use mysql_fetch_array to loop through a recordset:

```
<html>
<head>
<title>Retrieve data from database </title>
</head>
<body>

<?php
// Connect to database server
mysql_connect("mysql.myhost.com", "user", "sesame") or die (mysql_error ());

// Select database
mysql_select_db("mydatabase") or die(mysql_error());

// SQL query
$strSQL = "SELECT * FROM people";

// Execute the query (the recordset $rs contains the result)
$rs = mysql_query($strSQL);

// Loop the recordset $rs
// Each row will be made into an array ($row) using mysql_fetch_array
while($row = mysql_fetch_array($rs)) {

   // Write the value of the column FirstName (which is now in the array $row)
  echo $row['FirstName'] . "<br />";

  }

// Close the database connection
mysql_close();
?>
```

```
        </body>
        </html>
```

Notice that for every record how we get the content of the column "FirstName" by typing $row['FirstName']. Similarly, we can get the content of the column "Phone" by writing $row['Phone'], for example.

The order of the recordset is exactly the same as in the table in the database. But in the next example, it will be shown how to sort recordset.

Example 2: Sort the data alphabetically, chronologically or numerically

Often it can be helpful if a list or table of data is presented alphabetically, chronologically or numerically. Such sorting is very easy to do with SQL, where the syntax **Order By ColumnName** is used to sort according to the column contents.

Look at the SQL statement from the example above:

```
strSQL = "SELECT * FROM people"
```

The records can, for example, be sorted alphabetically by the first name of the people this way:

```
        strSQL = "SELECT * FROM people ORDER BY FirstName"
```

Or chronologically by date of birth like this:

```
        strSQL = "SELECT * FROM people ORDER BY BirthDate"
```

The sorting can be charged from **ascending** to **descending** by adding **DESC**:

```
        strSQL = "SELECT * FROM people ORDER BY BirthDate DESC"
```

In the following example, the people are sorted by age:

```
        <html>
        <head>

        <title>Retrieve data from database </title>

        </head>
```

```
<body>

<?php
// Connect to database server
mysql_connect("mysql.myhost.com", "user", "sesame") or die (mysql_error ());

// Select database
mysql_select_db("mydatabase") or die(mysql_error());

// SQL query
$strSQL = "SELECT * FROM people ORDER BY BirthDate DESC";

// Execute the query (the recordset $rs contains the result)
$rs = mysql_query($strSQL);

// Loop the recordset $rs
while($row = mysql_fetch_array($rs)) {

   // Write the value of the column FirstName and BirthDate
  echo $row['FirstName'] . " " . $row['BirthDate'] . "<br />";

  }

// Close the database connection
mysql_close();
?>

</body>
</html>
```

Try to change the SQL statement yourself and sort the records by first name, last name or phone number.

Retrieve selected data

Until now, our SQL statement retrieves **all** rows from the table. But often you need to set criteria in the SQL query for the data to be retrieved, for instance, if we only want the rows for those who have a particular phone number or a certain last name.

Say, we only want to retrieve people from the database who have the phone number "66554433". That could be done like this:

```
strSQL = "SELECT * FROM people WHERE Phone = '66554433 '"
```

There are six relational operators in SQL:

= Equals

< Less than

> Greater Than

<= Less than or equal to

>= Greater than or equal to

!= Not equal to

In addition, there are some logical operators:

AND
OR
NOT

In the next example, we use conditions to set up an address book.

Example 3: Address book

In this example, we will try to combine many of the things you have just learned. We will make a list of the names from the database where each name is a link to further details about the person.

For this, we need two files - **list.php** and **person.php** - with the following code:

The code of list.php

```
<html>
<head>
```

```
<title>Retrieve data from the database</title>
</head>
<body>

<ul>

<?php
// Connect to database server
mysql_connect("mysql.myhost.com", "user", "sesame") or die (mysql_error ());

// Select database
mysql_select_db("mydatabase") or die(mysql_error());

// SQL query
$strSQL = "SELECT * FROM people ORDER BY FirstName DESC";

// Execute the query (the recordset $rs contains the result)
$rs = mysql_query($strSQL);

// Loop the recordset $rs
while($row = mysql_fetch_array($rs)) {

   // Name of the person
  $strName = $row['FirstName'] . " " . $row['LastName'];

   // Create a link to person.php with the id-value in the URL
   $strLink = "<a href = 'person.php?id = " . $row['id'] . "'>" . $strNavn . "</a>";

    // List link
   echo "<li>" . $strLink . "</li>";

  }

// Close the database connection
mysql_close();
?>

</ul>
```

```
</body>
</html>
```

The code for person.php

```
<html>
<head>
<title>Retrieve data from database</title>
</head>
<body>

<dl>

<?php
// Connect to database server
mysql_connect("mysql.myhost.com", "user", "sesame") or die (mysql_error ());

// Select database
mysql_select_db("mydatabase") or die(mysql_error());

// Get data from the database depending on the value of the id in the URL
$strSQL = "SELECT * FROM people WHERE id=" . $_GET["id"];
$rs = mysql_query($strSQL);

// Loop the recordset $rs
while($row = mysql_fetch_array($rs)) {

        // Write the data of the person
        echo "<dt>Name:</dt><dd>" . $row["FirstName"] . " " . $row["LastName"] .
"</dd>";
        echo "<dt>Phone:</dt><dd>" . $row["Phone"] . "</dd>";
        echo "<dt>Birthdate:</dt><dd>" . $row["BirthDate"] . "</dd>";

}

// Close the database connection
mysql_close();
?>
```

```
</dl>
<p><a href="list.php">Return to the list</a></p>

</body>

</html>
```

CHAPTER FIVE

Delete data from database

In the two previous lessons, you have learned to insert and retrieve data from a database. In this lesson, we'll look at how to delete records in the database, which is considerably easier than inserting data.

Delete data using SQL

The syntax for an SQL statement that deletes records is:

```
DELETE FROM TableName WHERE condition
```

Example: Delete a record

When deleting a record, you can use the unique AutoNumber field in the database. In our database, it is the column named **id.** Using this unique identifier ensures that you only delete one record. In the next example, we delete the record where id has the value 24:

```
<html>
<head>
<title>Delete data in the database</title>
</head>

<body>

<?php
// Connect to database server
mysql_connect("mysql.myhost.com", "user", "sesame") or die (mysql_error ());

// Select database
mysql_select_db("mydatabase") or die(mysql_error());

// The SQL statement that deletes the record
$strSQL = "DELETE FROM people WHERE id = 24";
```

```
mysql_query($strSQL);

// Close the database connection
mysql_close();
?>

<h1>Record is deleted!</h1>

</body>
</html>
```

Remember that there is no "Recycle Bin" when working with databases and PHP. Once you have deleted a record, it is gone and cannot be restored.

CHAPTER SIX

Update data in a database

In previous lessons, you have learned to insert, retrieve and delete data from a database. In this lesson, we will look at how to update a database, i.e., edit the values of existing fields in the table.

Update data with SQL

The syntax for an SQL statement that updates the fields in a table is:

```
UPDATE TableName SET TableColumn='value' WHERE condition
```

It is also possible to update multiple cells at once using the same SQL statement:

```
UPDATE TableName SET TableColumn1='value1', TableColumn2='value2' WHERE condition
```

With the knowledge you now have from the lessons 19, 20 and 21, it should be quite easy to understand how the above syntax is used in practice. But we will of course look at an example.

Example: Update cells in the table "people"

The code below updates Donald Duck's first name to **D.** and changes the phone number to **44444444**. The other information (last name and birthdate) are not changed. You can try to change the other people's data by writing your own SQL statements.

```
<html>
<head>
<title>Update data in database</title>

</head>
<body>
```

```
<?php
// Connect to database server
mysql_connect("mysql.myhost.com", "user", "sesame") or die (mysql_error ());

// Select database
mysql_select_db("mydatabase") or die(mysql_error());

// The SQL statement is built
$strSQL = "Update people set ";
$strSQL = $strSQL . "FirstName= 'D.', ";
$strSQL = $strSQL . "Phone= '44444444' ";

$strSQL = $strSQL . "Where id = 22";

// The SQL statement is executed
mysql_query($strSQL);

// Close the database connection
mysql_close();
?>
<h1>The database is updated!</h1>
</body>
</html>
```

This example completes the lessons on databases. You have learned to insert, retrieve, delete and update a database with PHP. Thus, you are actually now able to make very advanced and dynamic web solutions, where the users can maintain and update a database using forms.

If you want to see a sophisticated example of what can be made with PHP and databases, try to join our community. It's free and takes approximately one minute to sign up. You can, among other things, maintain your own profile using the form fields. Maybe you will get ideas for your own site.

This also ends the tutorial. PHP gives you many possibilities for adding interactivity to your web site. The only limit is your imagination - have fun!